



**HAL**  
open science

# Architectures and Training for Visual Understanding

Hugo Touvron

► **To cite this version:**

Hugo Touvron. Architectures and Training for Visual Understanding. Computer Science [cs]. Sorbone Université, 2022. English. NNT: . tel-03945782

**HAL Id: tel-03945782**

**<https://hal.science/tel-03945782>**

Submitted on 18 Jan 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

**THÈSE DE DOCTORAT DE SORBONNE UNIVERSITÉ**  
Spécialité **Informatique**  
École Doctorale Informatique, Télécommunications et Électronique (Paris)

**Architectures and Training for Visual Understanding**  
**Architectures et Apprentissage pour l'Interprétation d'Image**

Présentée par  
**Hugo Touvron**

Dirigée par  
**Pr. Matthieu CORD**

Pour obtenir le grade de  
**DOCTEUR de SORBONNE UNIVERSITÉ**

Présentée et soutenue publiquement le 29 Septembre 2022

Devant le jury composé de :

<b>Pr. Graham TAYLOR</b> <i>Professor, University of Guelph</i>	Rapporteur
<b>Pr. François FLEURET</b> <i>Professor, University of Geneva</i>	Rapporteur
<b>Pr. Andrew ZISSERMAN</b> <i>Professor, University of Oxford &amp; DeepMind</i>	Examineur
<b>Dr. Cordelia SCHMID</b> <i>Research director, INRIA &amp; Google</i>	Examinatrice
<b>Pr. Nicolas THOME</b> <i>Professor, Sorbonne Université</i>	Examineur
<b>Pr. Matthieu CORD</b> <i>Professor, Sorbonne Université &amp; Valeo.ai</i>	Directeur de thèse
<b>Dr. Hervé JÉGOU</b> <i>Director, Research Scientist, Meta (FAIR)</i>	Encadrant de thèse



# CONTENTS

CONTENTS	iii
RÉSUMÉ	vii
ABSTRACT	ix
ACKNOWLEDGEMENTS	xi
1 INTRODUCTION	1
1.1 Deep learning for image classification . . . . .	1
1.2 Outline and Contributions . . . . .	6
2 LEARNING FINE-GRAINED IMAGE REPRESENTATIONS WITH COARSE LABELS	11
2.1 Related work . . . . .	12
2.2 About granularity . . . . .	13
2.3 Graft: Fitting a finer granularity . . . . .	15
2.4 Experiments . . . . .	18
2.5 Additional experiments . . . . .	24
2.6 Visualization . . . . .	28
2.7 Conclusion . . . . .	32
3 VISION TRANSFORMERS	37
3.1 Related work . . . . .	37
3.2 Vision transformer: overview . . . . .	38
3.3 Distillation through attention . . . . .	39
3.4 DeiT experiments . . . . .	41
3.5 Going deeper with vision transformers . . . . .	50
3.6 Specializing layers for class attention . . . . .	56
3.7 Our CaiT models . . . . .	60
3.8 Conclusion . . . . .	69
4 MULTI-LAYER PERCEPTRON FOR COMPUTER VISION	71
4.1 Related work . . . . .	72
4.2 ResMLP . . . . .	72
4.3 Experiments . . . . .	74
4.4 Conclusion . . . . .	81
5 REVISITING CONVNET ARCHITECTURE	83
5.1 Related work . . . . .	83
5.2 Attention-based pooling with PatchConvNet . . . . .	84
5.3 Main experimental results . . . . .	88
5.4 Conclusion . . . . .	96
6 ARCHITECTURE AND TRAINING INTERACTION REVISITED	97
6.1 Related work . . . . .	98
6.2 Revisit training & pre-training for Vision Transformers . . . . .	98
6.3 Experiments . . . . .	103
6.4 Additional Ablations . . . . .	115
6.5 Conclusion . . . . .	117
7 CONCLUSION	119







## RÉSUMÉ

L'apprentissage machine et plus particulièrement l'apprentissage profond (*deep learning*) ont un impact de plus en plus important dans notre société. En effet, ces approches sont devenues prépondérantes dans de nombreux domaines tels que le traitement du langage naturel avec la détection de contenu haineux, la synthèse de documents ou encore la vision par ordinateur avec le diagnostic médical ou le développement des voitures autonomes. Le succès de l'apprentissage profond est souvent associé à des architectures désormais emblématiques, comme AlexNet [123], ResNet [94] ou GPT [155]. Ces succès ont également été alimentés par des procédures d'optimisation bien conçues, qui néanmoins ne sont généralement pas au centre des préoccupations. Pour la classification d'image, le challenge ImageNet [168] a été un accélérateur pour le développement de nouvelles architectures mais aussi de nouvelles stratégies d'optimisation.

Dans cette thèse, nous discutons des interactions qu'il existe entre les architectures et les procédures d'entraînement. Nous étudions plus spécifiquement l'architecture des Transformers [204], appliqués à la vision par ordinateur, pour lesquels les procédures d'entraînement sont encore peu explorées. Elles sont pourtant essentielles pour compenser l'absence de *prior* architectural spécifique au traitement d'image. Nous proposons dans ce travail des procédures d'entraînement capables d'obtenir des performances état de l'art pour des Transformers ou même pour des architectures plus simples se rapprochant de perceptrons multi-couches. Plus précisément, nous commençons par étudier la possibilité d'apprendre avec des étiquettes grossières par une modification de la procédure d'entraînement. Nous étudions ensuite différents types d'architectures pour la vision par ordinateur. Nous analysons en particulier leurs caractéristiques, leurs avantages, leurs inconvénients et la manière de les entraîner. Enfin, nous étudions l'impact des interactions entre les architectures et les procédures d'entraînement. L'ensemble de nos approches sont évaluées en classification d'image sur ImageNet et en transfert. Nous nous évaluons également sur des tâches annexes comme par exemple la segmentation sémantique.





## ABSTRACT

Nowadays, machine learning and more particularly deep learning have an increasing impact in our society. This field has become prevalent, for instance in natural language processing where it has led to concrete applications to hate speech detection and document summarization. Similarly for computer vision, it enables better image interpretation, medical diagnosis, and major steps towards autonomous driving. Deep learning success is often associated with emblematic architectures, like AlexNet [123], ResNet [94] or GPT [155]. These successes were also powered by well-designed optimisation procedures, which are usually not the main focus of discussions. In image classification, the ImageNet [168] challenge was an accelerator for the development of new architectures but also for new optimisation recipes.

In this thesis, we discuss the interactions between architectures and training procedures. We study more specifically Transformers [204] architecture applied for visual understanding. Currently, transformers training procedures are less mature than those employed with convolutional networks (convnets). However, training is key to overcoming the limited architectural *priors* of transformers. For this reason, we focus on training procedures capable of obtaining interesting performance for transformers or even simpler architectures close to the multi-layer perceptron. We start by studying the possibility of learning with coarse labels through a modification of the training procedure. We then study different kinds of architectures for computer vision. We study their features, their advantages, their drawbacks and how to train them. Finally, we study the impact of the interaction between architecture and training process. All our approaches are evaluated in image classification on ImageNet and in transfer learning. We also evaluate our methods on additional tasks such as semantic segmentation.



## ACKNOWLEDGEMENTS

This is the part of the manuscript that I find the most difficult to write. I think this part is especially challenging when you had the chance to work with so many and such great collaborators. Properly thanking them would require a full chapter. Here I just present the abstract version of it.

First of all, I would like to thank my PhD supervisors Hervé and Matthieu who accompanied me during these three years. I learned a lot from them. Working with them has been a very enriching and intellectually stimulating experience. I hope to have the opportunity to collaborate with them again in the future. I thank the members of my committee: Pr. Graham Taylor, Pr. François Fleuret, Pr. Andrew Zisserman, Dr. Cordelia Schmid and Pr. Nicolas Thome for their interest in my work and taking the time to review and assess it. I am honored that they took the time, in their busy agenda, to take over this thankless task, which is particularly time-consuming for reviewers. Special thanks to Pr. Andrew Zisserman for the amazing titles of his papers, which have inspired a lot our own titles.

I thank all my amazing collaborators with whom I had the chance to work on great projects: Matthijs Douze, Mathilde Caron, Jakob Verbeek, Stephane d'Ascoli, Giulio Biroli, Levent Sagun, Pierre Stock, Francisco Massa, Natalia Neverova, Alex Sablayrolles, Ivan Laptev, Matthew Leavitt, Julien Mairal, Ross Wightman, Edouard Grave, Andrea Vedaldi, Ishan Misra, Alaa El-Nouby, Ari Morcos, Gautier Izacard, Piotr Bojanowsky, Armand Joulin, Gabriel Synnaeve and Ben Graham. Without them most of this work would not have been possible. I also thank my other colleagues at Meta and La Sorbonne for the fruitful exchanges that we got during my tenure as a PhD student.

Finally, I thank my family for their support during these 3 years, even during the intense times like the deadline nights, which were probably too frequent.



## CONTENTS



## INTRODUCTION

Computer vision is about understanding how to obtain a high-level representation of images and videos. High-level representations are obtained by projecting the image into a vector space with a certain structure that makes it easier to extract the information needed to interpret the image. This allows complex tasks such as recognising concepts in an image or performing action recognition in video. While it is easy for a human to recognise a given concept, it is hard to design an algorithm that would do the same. Indeed, when we see a cat, we know quite easily that it is a cat, but it is almost impossible to describe in an algorithmic way all the steps that lead us to this conclusion. Early approaches were based on handcrafted image representations, *i.e.* manually designed and relying on expert knowledge. The most emblematic strategy is certainly the Bag-of-Words (BoW) method, which encodes and pools local features on visual dictionaries. BoW was the state-of-the-art approach for image classification in the 2000s. Inspired by information retrieval [170], the pioneering work [138] introduced a BoW scheme for image representation using a color dictionary, extended to Gabor feature dictionary by [74], and finally popularized using SIFT features (Scale-Invariant Feature Transform [137]) for visual recognition [43, 179]. In the 2010s, we then witnessed the emergence of deep learning methods, which gradually overtook all traditional computer vision approaches.

The computer vision field includes many tasks such as image classification, detection or segmentation (see Figure 1.1 for an illustration). Today, the gold standard approaches to solve these tasks are based on deep learning. This thesis falls within this context. In the following, we detail the basics of deep learning for vision and position our contributions.

### 1.1 Deep learning for image classification

#### 1.1.1 Image classification

Application of deep learning for computer vision is quite old. For instance, the backpropagation algorithm commonly used to train deep learning models was introduced by Hinton, Rumelhart and Williams [167] in 1986. Lecun et al. [126] used Convolutional neural network trained with a backpropagation algorithm to perform digit recognition. However, this task re-

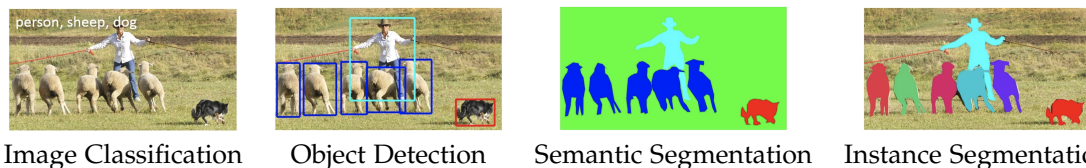


Figure 1.1 – Illustration of different tasks usually studied in computer vision. Credit Lin et al. *Microsoft COCO: Common Objects in Context*.



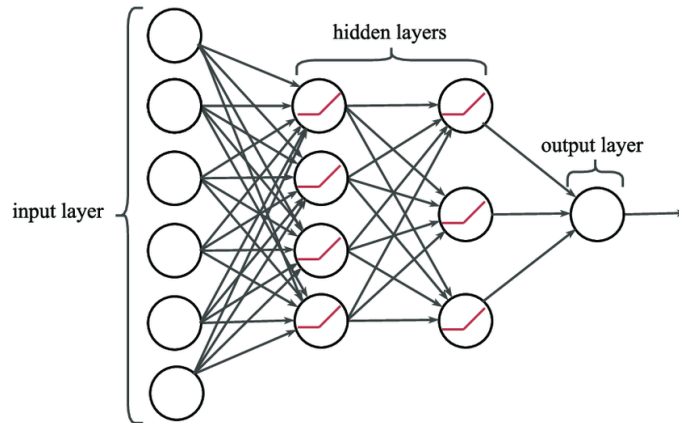


Figure 1.2 – Illustration of a multi-layers perceptrons (MLP). Credit: Martin Cenek, Researchgate.

mains a quite simple classification task with small black and white images and only 10 classes. We had to wait until 2012 to see the first successes of deep learning on more complex image classification tasks, such as the ImageNet challenge [168]. This challenge with 1000 classes and over 1.28M images is quite complex. ImageNet is a problem that is representative of more real-world use cases than digit recognition. Before 2012 the ImageNet challenge was won year after years by approaches relying on handcrafted features. In 2012, Krizhevsky et al. [123] won the challenge by a large margin over competing approaches by using the AlexNet convolutional neural network (see Figure 1.3 for an illustration of the architecture) trained with backpropagation and data augmentation. Following the success of AlexNet year after year, many new deep learning approaches have in succession won this challenge, reaching today human level performances for this task.

Although the ImageNet challenge is not organized anymore, the ImageNet dataset is so core to computer vision that it is still used as a benchmark to measure progress in image understanding. The evolution of the state of the art on the ImageNet dataset [168] reflects the progress with convolutional neural network architectures and learning [123, 177, 187, 199, 200, 228]. Any progress usually translates to improvements in other related tasks such as detection or segmentation.

The ImageNet example shows us the importance of large datasets for deep learning approaches. Indeed, we need large amounts of data to reach good performance with deep architectures. Generally, we also need annotations which can be very expensive to obtain. For instance, for the image classification task, it is sometimes necessary to have expert knowledge to be able to correctly label some concepts. However, some approaches can partially overcome this problem as we will detail later in this manuscript.

In addition to a large amount of data, the success of deep learning approaches relies on two essential components. The architecture which defines the capacity of the model, (i.e. the set of functions that can be approximated layer by layer) and the learning procedure, which corresponds to all the methods used to learn the weights of the model. The learning process allows a convergence of the model's weights towards a certain function to solve a given task. There are strong interactions between the architecture and the learning procedure as the design has a direct influence on the gradient calculation (Which is used in the back propagation algorithm) but also on the stability of the learning procedure. In this thesis, we study these two aspects and their interaction.

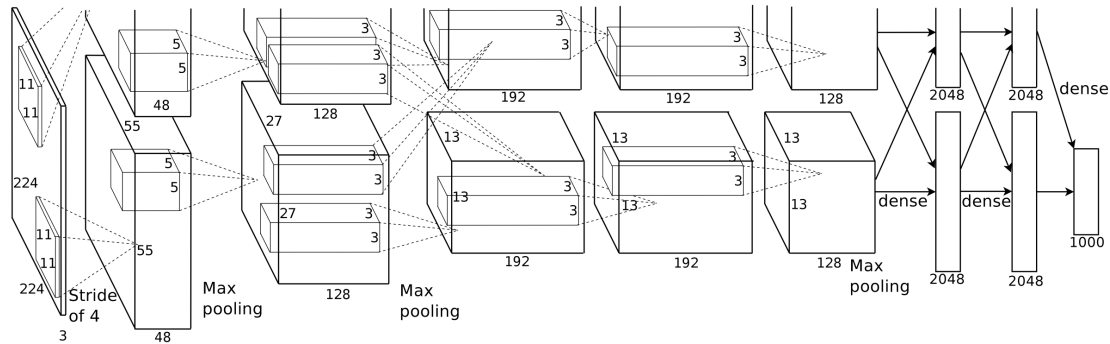


Figure 1.3 – Illustration of the convolutional neural network AlexNet. Credit: Krizhevsky et al. *ImageNet Classification with Deep Convolutional Neural Networks*.

### 1.1.2 Deep Architectures Design

**Multi-layer Perceptron (MLP)**, also called feedforward neural networks or deep feedforward networks, is illustrated in Figure 1.2. The Perceptron was introduced by Rosenblatt [165] in 1958. Cybenko [48] show that MLP are universal function approximators which explains their interest in the Machine learning community. Some studies have shown that MLP are competitive with convnets for the tasks of digit recognition [42, 176], keyword spotting [31] and handwriting recognition [20]. Several works [130, 140, 201] have questioned if MLP are also competitive on natural image datasets, such as CIFAR-10 [122]. More recently, d’Ascoli *et al.* [51] have shown that a MLP initialized with the weights of a pretrained convnet achieves performance that are superior than the original convnet. Neyshabur [143] further extends this line of work by achieving competitive performance when training an MLP from scratch but with a regularizer that constrains the models to be close to a convnet. Nevertheless, these studies have been conducted on small scale datasets with the purpose of studying the impact of architectures on generalization in terms of sample complexity [62] and energy landscape [116].

**Convolutional neural networks (Convnet)** were introduced by Fukushima et al. [76] and trained with backpropagation by Lecun et al. [126]. Based on specific layer transforms that correspond to convolution filters, convnet have been the main design paradigm for image understanding tasks, as initially demonstrated on image classification tasks. Indeed, since 2012’s AlexNet [123], convnets have dominated this benchmark and have become the *de facto* standard. The evolution of the state of the art on the ImageNet dataset [168] reflects the progress with convolutional neural network architectures and learning [123, 177, 187, 199, 200, 228]. For instance, the residual connections allows one to train a deeper architecture as evidenced by ResNet [94]. This allows for a more stable learning process and better performance. Many convnet architectures, but also transformers have adopted this design with residual connections. Today in computer vision, convnets hold many state-of-the-art results. However, transformers have become a more and more viable alternative.

**Transformers** are a deep architecture introduced in 2017 by Vaswani et al. [204]. By leveraging a global attention process between all features at every layer, they have become the default architecture in many domains like Natural Language Processing (NLP) where it still holds many state-of-the-art results. Motivated by the success of attention-based models in Natural Language Processing [56, 155, 205, 208], there has been increasing interest in architectures leveraging attention mechanisms within convnets [107, 128, 241]. More recently several researchers have proposed hybrid architecture transplanting transformer ingredients to convnets to solve vision tasks [28, 174].

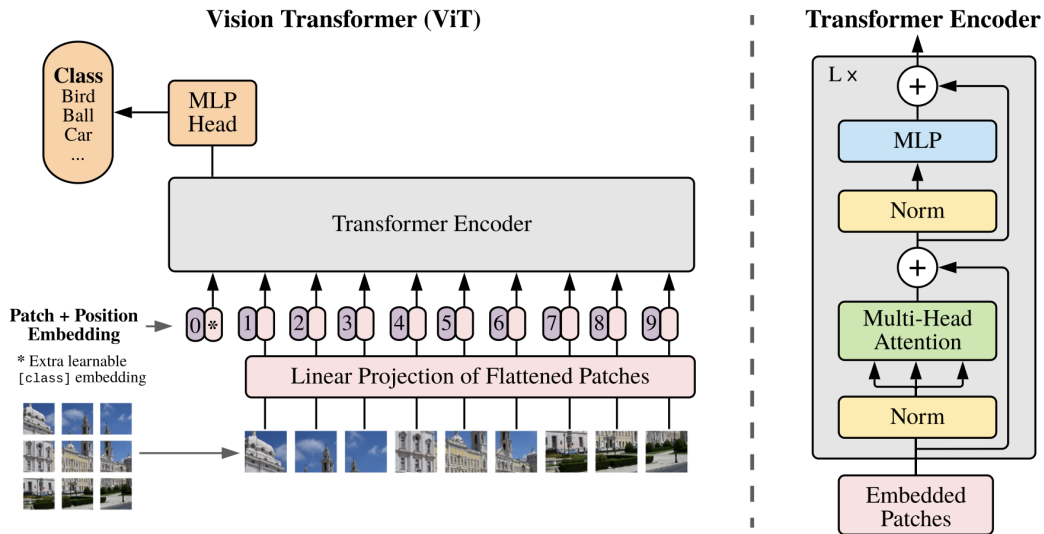


Figure 1.4 – Illustration of the vision transformers architecture (ViT). credit: Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*.

Dosovitskiy et al. [61] introduced the Vision transformers (ViT) architecture (See Figure 1.4 for illustrations). ViT is an architecture directly inherited from Natural Language Processing [205], but applied to image classification with raw image patches as input. Transformers are now used successfully in computer vision and gives excellent results in many tasks such as image classification, segmentation or detection. This architecture also has a strong potential in multimodal approaches given its good performances in image processing but also in NLP.

**MLP revisited.** Following the success of transformers in computer vision MLPs have been used recently in computer vision on more complex problem like large scale image classification but also in NLP. These revisited MLPs adopt a residual architecture similar to that of the transformers and a patch splitting of the image in order to decrease the complexity. The blocks of this architecture correspond to classical MLPs as detailed in the first paragraph. Although their results are slightly worse than transformers, their performances are very competitive given the relative simplicity of these architectures.

### 1.1.3 Learning process

The learning process, although less emphasized than the architecture design, plays a central role in the success of deep learning. Without a well-designed training regime the performance of neural network approaches is generally poor. The first successes of deep learning in image classification are related to new architectures but also to new training procedures. This is generally the case for supervised learning tasks (i.e. when using the ground truth label in the learning process) and also for self-supervised approaches (i.e. when the model learns by itself by comparing different images or different augmentation of the same image). However, the interaction between architecture and training procedure is still not studied extensively.

Data is an important component of deep learning approach. Indeed, it is necessary to have enough data to be able to train deep architectures. However, having data is usually not enough. Indeed, the available annotations will have an impact. Nevertheless, to be able to exploit correctly

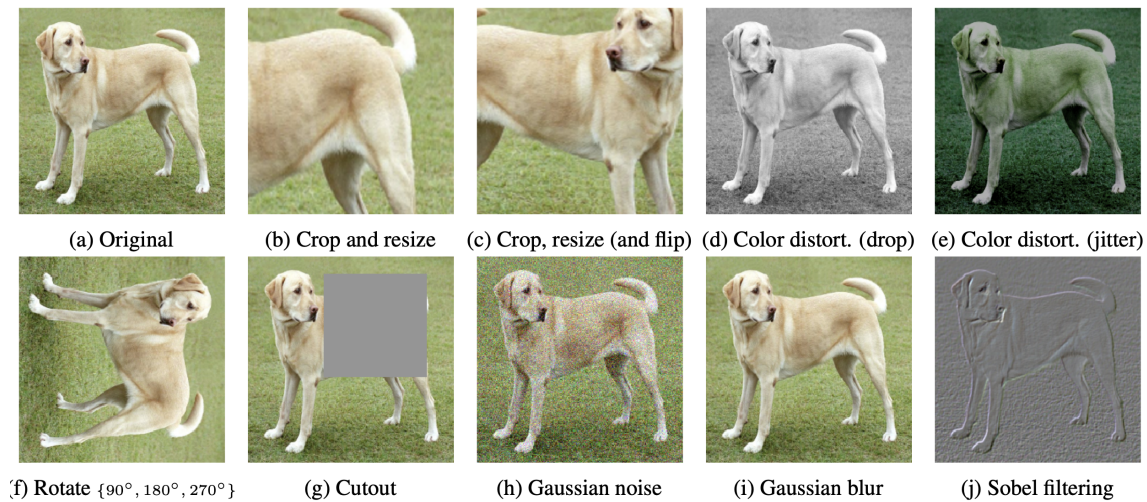


Figure 1.5 – Illustration of different data augmentation on an image with the label dog. The different data augmentation don't change the concept inside the image. Credit: Chen et al. *A Simple Framework for Contrastive Learning of Visual Representations*.

the information at our disposal we have to adapt the learning process correctly. For image classification the elements involved in the learning process are the data augmentation and the elements related to optimisation procedure. We detail these concepts in the following paragraphs but for a more detailed description we refer to Goodfellow et al. [84], Bishop et al. [19] and Hastie et al [91].

**Data augmentation.** For image classification, data augmentation amounts to applying transformations to the images [44, 57, 184, 237, 242], which do not change the class of the images (see Figure 1.5 for illustrations). For example, if we want to classify images of dogs, we can apply rotations because this does not change the content and being rotation invariant is a good property here. On the other hand, if we want to do digit recognition, we don't want to be rotation invariant because we wouldn't be able to distinguish between the 6's and the 9's. In this case, we use other kinds of data augmentation such as colour variations. The purpose of the data augmentation is to help to learn the right property for a given task. It also allows us to limit the overfitting by artificially generating more images. Overfitting appears when the model adapts too much to the training data and generalises poorly with new data. Underfitting is the opposite behavior when the model is not able to give good results on training data.

**Optimization.** Optimization procedures are generally based on gradient descent approaches. Gradient descent amounts to updating the weights of the model following the direction of the gradient in order to minimize the cost function. The elements related to the optimization procedure are diverse. For instance, the choice of the optimizer (i.e the algorithms that are used to update the weights), the choice of the learning rate schedule (i.e the size of the steps in the gradient direction) and all the regularization approaches. This has an impact on the way the model weights are learned and the performance we can reach for a given task. The regularization refers here to all the approaches adding constraints on the learning process of the model weights in order to limit the risk of overfitting. There are many different optimization approaches, each with its own advantages and disadvantages. These approaches are used according to the architecture and the task we want to solve. Nevertheless, there is no well-established theory for the choice of the training components. They are usually designed according to the intuition and experience of the experimenters. Indeed, it is an experimental and mostly empirical process. However, the

training design is important to find the right trade-off between overfitting and underfitting, each regularization method and each optimizer potentially having different properties.

**Training: trends.** In the same way that there has been an evolution of architecture for image classification tasks, there is an evolution of the training procedures. The training procedure used in the AlexNet paper [123] was one of the first standard procedure for training deep learning architecture for image classification. It uses 90 epochs with batch size 128 and step decay for the learning rate by dividing the learning rate by 10 when the validation loss does not decrease anymore. They use SGD optimizer with L2 regularization and dropout. For the data augmentation they use: a random crop of size  $224 \times 224$  extracted in images of size  $256 \times 256$ , random horizontal flip of images and perform PCA for altering the intensities of the image's channels. GoogleNet paper [184] subsequently introduces new components: label smoothing and random resized crop instead of random crop. This paper also uses smoother learning rates (reduction of 4% every 8 epochs). The ResNet [94] paper re-used this component with a bigger batch size 256 allowed by more powerful GPUs. Afterwards, the procedure often adopted by default is the procedure proposed in the TorchVision library of PyTorch, which is similar to the one proposed by Goyal et al. [85]. This training procedure is significantly inspired by the previous one. They use for the data augmentation: Random resized crop, horizontal flip, colour jitter and a normalization of the RGB channel. For the optimization they use SGD optimizer with batch size 512 and 90 epochs. The learning rate is divided by 10 every 30 epochs. Nowadays, the default training procedure has changed a lot and uses more data augmentation and regularization, which usually improves the model performance. These new training procedures benefit to both the old and the new architectures. However, some architectures benefit more from it.

These training improvements on ImageNet have translated into progress on other datasets through transfer learning. Transfer learning in a two-stage procedure in which the first step it to first pre-train a model on another dataset (generally ImageNet), and then re-trains the model slightly on the new dataset. This generally gives better performance than approaches that do not start from a pre-trained model, since it leverages more training data.

After this brief introduction of deep architectures and training procedures for computer vision, we now detail the contributions of this PhD thesis.

## 1.2 Outline and Contributions

### 1.2.1 Learning fine-grained image representations with coarse labels

Being able to classify fine grained concepts is quite challenging. Indeed, it is generally necessary to have a large amount of annotated data to be able to recognise a concept. With fine grained concepts, the annotation is more difficult to obtain because you need expert knowledge to be able to annotate precisely the images. However, it is for this kind of application that automatic recognition approaches are even more useful. In Chapter 2, we tackle the problem of learning a finer representation than the one provided by training labels. This enables fine-grained category retrieval of images in a collection annotated with coarse labels only. Our network is learned with a nearest-neighbour classifier objective, and an instance loss inspired by self-supervised learning. By jointly leveraging the coarse labels and the underlying fine-grained latent space, it significantly improves the accuracy of category-level retrieval methods. At the time of publication, this strategy was outperforming all competing methods for retrieving or classifying images at a finer granularity than that available at train time. It also improves the accuracy for transfer learning tasks to fine-grained datasets.

**Outline.** Chapter 2 introduces Grafit, a method for learning fine-grained image representation with coarse labels. First, we present the Grafit method and the intuition behind the design. Then we introduce different tasks designed for coarse-to-fine tasks in order to evaluate the method performance. Lastly, we conduct extensive experiments on datasets with different levels of granularity in order to validate our approach.

**Publication.** Chapter 2 is based on the paper “*Grafit: Learning fine-grained image representations with coarse labels*”, Hugo Touvron, Alexandre Sablayrolles, Matthijs Douze, Matthieu Cord, Hervé Jégou, *ICCV 2021* (see Grafit paper [198]).

## 1.2.2 Transformers for computer vision

Transformers architecture, introduced by Vaswani et al. [205] for machine translation are currently the reference model for all natural language processing (NLP) tasks. Many improvements of convnets for image classification are inspired by transformers. For example, Squeeze and Excitation [107], Selective Kernel [128] and Split-Attention Networks [241] exploit mechanisms akin to transformers self-attention (SA) mechanism. The Dosovitsky et al [61] vanilla transformer architecture was shown to address image understanding tasks such as image classification. These high-performing vision transformers are pre-trained on private dataset with hundreds of millions of images not available and uses a large infrastructure, thereby limiting their adoption.

In Chapter 3, we produce competitive convolution-free transformers by training on ImageNet only. We train them on a single computer in less than 3 days. Our reference vision transformer (86M parameters) achieves top-1 accuracy of 83.1% (single-crop) on ImageNet with no external data. We also introduce a teacher-student strategy specific to transformers. It relies on a distillation token ensuring that the student learns from the teacher through attention. We show the interest of this token-based distillation, especially when using a convnet as a teacher. This leads us to report results competitive with convnets for both ImageNet (where we obtain up to 85.2% accuracy) and when transferring to other tasks.

Moreover, we build and optimize deeper transformer networks for image classification as the optimization of image transformers has been little studied so far. In particular, we investigate the interplay of architecture and optimization of such dedicated transformers. We propose two modifications that significantly improve the convergence and accuracy of deep transformers. This leads us to produce models whose performance does not saturate early with more depth. For instance we obtain 86.2% top-1 accuracy on ImageNet when training with no external data.

**Outline.** Chapter 3 shows for the first time that it is possible to have state of the art performance with transformers on ImageNet without using hundreds of millions of images for pre-training. First, we introduce a new training procedure for Vision transformers and then we introduce Layer Scale and Class Attention, two methods for training deeper transformers.

**Publication.** Chapter 3 is based on the papers “*Training data-efficient image transformers & distillation through attention*”, Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, Hervé Jégou, *ICML 2021* (see DeiT paper [193]) and “*Going deeper with Image Transformers*”, Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, Hervé Jégou, *ICCV 2021* (see CaiT paper [197]). The code is available at <https://github.com/facebookresearch/deit>.

### 1.2.3 Multi-layers perceptron for computer vision

After the success of transformers in computer vision we can question whether this success is related to the self-attention process or whether it is more related to the training procedure and patching of the input. Indeed, the training procedures and input splitting adopted by transformers are quite different from those usually used for convnet. On the other hand, the attention process allows long range interaction that is adapted to the input, whereas convnet allows more local and input independent operations.

We further discuss these phenomena in Chapter 4. We introduce ResMLP, which is an architecture entirely built upon multi-layer perceptron. It is a simple residual network that alternates (i) a linear layer in which image patches interact, independently and identically across channels, and (ii) a two-layer MLP network in which channels interact independently per patch. When trained with our modern training strategy using heavy data augmentation and optionally distillation, it attains surprisingly good accuracy/complexity trade-offs on ImageNet. We also train ResMLP models in a self-supervised setup, to further limit priors from employing a labelled dataset.

**Outline.** Chapter 4 shows that it is possible to have good performance on computer vision and NLP tasks with an MLP-like architecture. First, we introduce ResMLP, a pure residual MLP architecture working on tokens without normalization process relying on data statistics. Then, we validate our architecture on large scale experiments and provide extensive ablation in order to understand the impact of each component.

**Publication.** Chapter 4 is based on the paper “ResMLP: Feedforward networks for image classification with data-efficient training”, Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, Hervé Jégou, *arXiv 2021, submitted to TPAMI (minor revision)* (see ResMLP paper [192]). The code associated is publicly available at <https://github.com/facebookresearch/deit>.

### 1.2.4 Revisiting convnet architecture

After studying transformers and MLP for computer vision, we adopt concepts from these architectures to revisit the design of convnets. The aim is to have simpler models with the possibility to obtain visualisations like those of transformers. In Chapter 5, we show how to augment any convolutional network with an attention-based global map to achieve non-local reasoning. We replace the final average pooling by an attention-based aggregation layer akin to a single transformer block, that ponders how the patches are involved in the classification decision. We plug this learned aggregation layer with a simplistic patch-based convolutional network parametrized by two parameters (width and depth). In contrast with a pyramidal design, this architecture family maintains the input patch resolution across all the layers. It yields competitive trade-offs between accuracy and complexity, in terms of memory consumption, as shown by our experiments on various computer vision tasks: object classification, image segmentation and detection.

**Outline.** Chapter 5 revisits the design of convnets by taking inspiration from the works on vision transformers. We first introduce the learn aggregation layer and analyse its interest for class activation map visualization. Then we present the PatchConvnet architecture. Finally, we evaluate it on different tasks in order to show that a simple architecture can be efficient on many tasks.

**Publication.** Chapter 5 is based on “Augmenting Convolutional networks with attention-based aggregation”, Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Piotr Bojanowski, Armand Joulin,

Gabriel Synnaeve, Hervé Jégou, *arXiv 2021, under review NeurIPS 2022* (see PatchConvnet paper [195]). The code is publicly available at <https://github.com/facebookresearch/deit>.

### 1.2.5 Architecture and training interaction

Architecture design is quite well studied for vision transformers. Many variants have been introduced to reduce the cost of attention by introducing for example more efficient attention [66, 71, 134] or pooling layers [99, 134, 210]. Some papers re-introduce spatial biases specific to convolutions within hybrid architectures [86, 218, 224]. However, training recipes seem to be the key elements with general architectures like transformers. Indeed, with general architecture the risk of overfitting is more important due to the flexibility of the architecture. If we find proper training recipes we should be able to learn similar or better operation than the more specific architectures. Recent works show that ViTs benefit from self-supervised pre-training, in particular Bert-like pre-training like BeiT [11].

In Chapter 6, we revisit the supervised training of ViTs. Our procedure builds upon and simplifies a recipe introduced for training ResNet-50. It includes a new simple data augmentation procedure with only 3 augmentations, closer to the practice in self-supervised learning. Our evaluations on Image classification (ImageNet-1k with and without pre-training on ImageNet-21k), transfer learning and semantic segmentation show that our procedure outperforms by a large margin previous fully supervised training recipes for ViT.

**Outline.** Chapter 6 revisits the training procedures for vision transformers and highlights the interactions between training and architectures. We first introduce a new data augmentation for vision transformers and then propose a new training procedure for the ViT architecture. We evaluate our models on different tasks and compare the performance of our training recipes with Bert like approach and discuss the interactions between training and architectures.

**Publication.** Chapter 6 is based on the papers “*ResNet strikes back: An improved training procedure in timm*”, Ross Wightman, Hugo Touvron, Hervé Jégou, *NeurIPS workshop 2021* (see RSB paper [215]) and “*DeiT III: Revenge of the ViT*”, Hugo Touvron, Matthieu Cord, Hervé Jégou, *ECCV 2022* (see DeiT III paper [194]). The corresponding code is available at <https://github.com/facebookresearch/deit> and <https://github.com/rwightman/pytorch-image-models>.





## LEARNING FINE-GRAINED IMAGE REPRESENTATIONS WITH COARSE LABELS

Image classification now achieves a performance that meets many application needs [94, 123, 200]. However, in practice, datasets and labels available at training time do not necessarily correspond to those needed in subsequent applications [50]. The granularity of the training-time concepts may not suffice for fine-grained downstream tasks. This has encouraged the development of specialized classifiers offering a more precise representation. Fine-grained classification datasets [102] have been developed for specific domains, for instance to distinguish different plants [41] or bird species [209]. Gathering a sufficiently large collection with fine-grained labels is difficult by itself, as it requires to find enough images of rare classes, and annotating them precisely requires domain specialists with in-domain expertise. This is evidenced by the Open Images construction annotation protocol [125] that states that: “*Manually labeling a large number of images with the presence or absence of 19,794 different classes is not feasible*”. For this reason they resorted to computer-assisted annotation, at the risk of introducing biases.

To circumvent this issue, we propose in this chapter, a new strategy *Grafit*, to get strong classification and image retrieval performance on fine concepts using only coarse labels at training. Our work leverages two intuitions. First, in order to improve the granularity beyond the one provided by image labels, we need to exploit another signal than just the labels. For this purpose, we build upon the works [15, 227] that exploit *two* losses to address both image classification and instance recognition, leveraging the “free” annotations provided by multiple data augmentations of a same instance, in the spirit of self-supervised learning [21, 29, 34, 87]. The second intuition is that it is better to explicitly infer coarse labels even when classifying at a finer granularity. For this purpose, we propose a simple method that exploits both a coarse classifier and image embeddings to improve fine-grained category-level retrieval. This strategy outperforms existing works that exploit coarse labels at training time but do not explicitly rely on them when retrieving finer-grained concepts [220].

By these ways our *Grafit* method is able to liberate the data collection process from the quirks of a rigid fine-grained taxonomy, as previously discussed. To validate our strategy, we investigate two challenging applications :

**On-the-fly classification.** For this task, the fine-grained labels are available at test time only, and we use a non-parametric kNN classifier [220] for on-the-fly classification, *i.e.* without training on the fine-grained labels.

**Category-level Retrieval.** Given a collection of images annotated with coarse labels, like a product catalog, we aim at ranking these images according to their fine-grained semantic similarity to a new query image outside the collection, as illustrated by Figure 2.3. We believe that this new task better is more realistic than the on-the-fly classification setting.

This chapter is organized as follows. After reviewing related works in Section 2.1, we present our *Grafit* method in Section 2.3. Section 2.4 compares our approach against baselines on various datasets, and presents an extensive ablation.

**Publication.** Chapter 2 is based on the paper “*Grafit: Learning fine-grained image representations with coarse labels*”, Hugo Touvron, Alexandre Sablayrolles, Matthijs Douze, Matthieu Cord, Hervé Jégou, *ICCV 2021* [198].

## 2.1 Related work

**Label granularity in image classification.** In computer vision, the concept of granularity underlies several tasks, such as fine-grained [41, 102] or hierarchical image classification [55, 217, 231]. Some authors consider a formal definition of granularity, see for instance Cui et al. [46]. In this chapter, we only consider levels of granularity relative to each other, where each coarse class is partitioned into a set of finer-grained classes.

In some works on hierarchical image classification [68, 88, 163, 186], a coarse annotation is available for all training images, but only a subset of the training images are labelled at a fine granularity. In this chapter, we look at the case where no fine labels are available at training time.

**Train-Test granularity discrepancy.** A few works consider the case where the test-time labels are finer than those available at training time and where each fine label belongs to one coarse label. Approaches to this task are based on clustering [220] or transfer learning [110]. Huh et al. [110] address the question: “is the feature embedding induced by the coarse classification task capable of separating finer labels (which it never saw at training)?” To evaluate this, they consider the 1000 ImageNet classes as fine, and group them into 127 coarse classes with the WordNet [72] hierarchy. Wu et al. [220] evaluate on the 20 coarse classes of CIFAR-100 [124] and on the same subdivision of ImageNet into 127 classes. They evaluate their method, Scalable Neighborhood Component Analysis (SNCA), with a kNN classifier applied on features extracted from a network trained with coarse labels. Note that this work departs from the popular framework of object/category discovery [37, 77, 106, 206, 207], which is completely unsupervised.

In our work we mainly compare to the few works that consider coarse labels at train time, therefore SNCA [220] is one of our baselines. We adopt their coarse labels definition and evaluation procedure for on-the-fly classification.

**Unified embeddings for classes and instances.** Similar to Wu et al. [220], several Distance Metric Learning (DML) approaches like the Magnet loss [161] or ProxyNCA [141, 189] jointly take into account intra- and inter-class variability. This improves transfer learning performance and favors in some cases the emergence of finer hierarchical concepts. Berman et al. proposed Multigrain [15], which simply adds to the classification objective a triplet loss that pulls together different data-augmentations of a same image. Recent works on semi-supervised learning [16, 17, 180, 227, 232, 240] rely on both supervised and self-supervised losses to get information from unlabelled data. For instance the approach of Xie et al. [227] is similar to Multigrain, except that the Kullback-Leibler divergence replaces the triplet loss. Matching embeddings of the same images under different data-augmentations is the main signal in current works on self-supervised learning, which we discuss now.

**Unsupervised and Self-Supervised Learning.** In unsupervised and self-supervised approaches [29, 34, 80, 87, 115, 203] the model is trained on unlabeled data. Each image instance is considered as a distinct class and the methods aim at making the embeddings of different data-augmentations of a same instance more similar than those of other images. To deal with finer semantic levels than those provided by the labels, we use an approach similar to BYOL [87]. BYOL only requires pairs of positive elements (no negatives), more specifically different augmentations of the same image. A desirable consequence is that this limits contradictory signals on the classification objective.

**Transfer Learning.** Transfer learning datasets [23, 121, 144] are often fine grained and rely on a feature extractor pre-trained on another set of classes. However, the fine labels are not a subset of the pre-training labels, so we consider transfer learning as a generalization of our coarse-to-fine task. It is preferable to pre-train on a domain similar to the target [47], *e.g.*, pre-training on iNaturalist [102] is preferable to pre-training on ImageNet if the final objective is to discriminate between species of birds. The impact of pre-training granularity is discussed in prior works [46, 233]. In Section 2.4.6 we investigate how Grafit pre-training performs on fine-grained datasets with transfer learning task.

## 2.2 About granularity

Is it possible to create representations that discriminate between classes finer than the available coarse labels? Considering that we have seen only coarse labels at training time, how can we exploit the coarse classifier for fine-grained classification, if useful at all? In this section we discuss these two questions and construct an experiment to analyze the role of the losses and of the coarse classifier. We then provide empirical observations.

**Practical setup.** In the following two experiments, we consider the CIFAR-100 benchmark that has two granularity levels with 20 and 100 classes (see Section 2.4.1).

We denote by  $f$  the Resnet-18 trunk mapping from the image space to an embedding space. We train the neural network trunk  $f$  with three possible losses:

- Baseline: regular cross-entropy classification training  $\mathcal{L}_{\text{CE}}$  with coarse or fine classes;
- Triplet loss: training a triplet loss  $\mathcal{L}_{\text{Triplet}}$  to differentiate between image instances (does not use the labels);
- $\mathcal{L}_{\text{CE}} + \mathcal{L}_{\text{Triplet}}$ : sum of the two losses. This is intended to be a simple proxy of Grafit.

### 2.2.1 Experiment: separating arbitrary fine labels

This experiment is inspired both by the Rademacher complexity [24] and by the self-supervised learning (SSL) literature [16]. In SSL, the standard way to evaluate the quality of a feature extractor  $f$  is to measure the accuracy of the network after learning a linear classifier  $l$  for the target classes on top of  $f$ . The Rademacher complexity measures how a class of functions (*i.e.*  $l \circ f$ , with  $f$  fixed and  $l$  learned) is able to classify a set of images with random binary labels.

For this experiment we train the trunk  $f$  jointly with a (coarse class) classifier with  $\mathcal{L}_{\text{CE}}$  using coarse labels. We hope to improve the granularity of  $f$ , *i.e.* improve the network trunk such that a (finer-grained) classifier  $l$  trained on top of  $f$  performs better at discriminating between instances that have the same coarse label.

**Random labels.** We generate synthetic fine labels by the following process: for each coarse label, we randomly and evenly split the training images into two subcategories, yielding 40 classes in total. Inspired by the empirical Rademacher estimation, we sample 10 distinct splits of random labels. For each split, we learn a linear classifier  $l$  on top of  $f_i$ . We then compute the mean accuracy (top-1, %) of  $l \circ f_i$  on the training examples for the three losses. By evaluating to what extent one can fit a linear classifier  $l$  on top of  $f$ , this experiment measures how well the data are spread in the representation spaces.

Table 2.1 – Separability experiment on CIFAR-100. The trunk is trained with coarse labels only. Images with the same coarse label are randomly grouped into two distinct fine-grain labels (40 distinct labels in total). Then we fine-tune a linear classifier on the synthetic labels and measure the top-1 accuracy on fine-labels. When conditioning, the estimator exploits the hierarchy: we first predict the coarse class and condition on it to make the final prediction. We report results with three training losses.

Training loss	Top-1 (%)	
	no cond.	coarse cond.
$\mathcal{L}_{\text{CE}}$	53.7 $\pm 0.3$	54.5 $\pm 0.3$
$\mathcal{L}_{\text{Triplet}}$	26.4 $\pm 0.3$	—
$\mathcal{L}_{\text{CE}} + \mathcal{L}_{\text{Triplet}}$	57.1 $\pm 0.2$	<b>58.5</b> $\pm 0.3$
Random network	8.7 $\pm 0.3$	—

Table 2.2 – Top-1 accuracy of a ResNet-18 on CIFAR-100 for different training schemes. We report the results after finetuning of the linear classifier on the fine labels (see Section 2.2). The Triplet training is unsupervised, therefore the results for the two columns are identical.

Method	Train coarse	Train fine
$\mathcal{L}_{\text{CE}}$	80.4 $\pm 0.2$	80.6 $\pm 0.2$
$\mathcal{L}_{\text{Triplet}}$	76.5 $\pm 0.2$	76.5 $\pm 0.2$
$\mathcal{L}_{\text{CE}} + \mathcal{L}_{\text{Triplet}}$	<b>80.9</b> $\pm 0.2$	<b>81.3</b> $\pm 0.2$

**Impact of the loss terms.** We report the results in Table 2.1. We can see that, to distinguish between our synthetic fine labels, training with the triplet loss  $\mathcal{L}_{\text{Triplet}}$  in combination with the classification loss  $\mathcal{L}_{\text{CE}}$  is essential: the sum of losses outperforms each individual loss.

**Conditioning.** We also measure the impact of *conditioning on coarse classes*: we first predict the coarse label with the coarse classifier, and leverage its softmax output to classify the fine class. This clearly improves the accuracy, which motivates our fusion strategy inspired by this conditioning in Section 2.3.2.

## 2.2.2 Experiment: varying the training granularity

In this section we make empirical observations related to the training granularity in the embedding space.

We train  $f$  with one of the three losses and either coarse or fine labels as supervision. In a second stage, we train a linear classifier  $l$  on the Resnet-18 trunk with fine class supervision, and evaluate its accuracy on the test set.

**Accuracies.** We first quantify the quality of the representation space. The accuracies are reported in Table 2.2. We observe that the coarse labels are almost as good as the fine labels for pre-training. The unsupervised  $\mathcal{L}_{\text{Triplet}}$  loss performs significantly worse, which concurs with our previous separability experiment. Combining this loss with the  $\mathcal{L}_{\text{CE}}$  loss improves it, both with coarse and fine supervisions.

**Size of the representation space.** We quantify the information content of embedding vectors by computing their principal components analysis (PCA). This is a reasonable proxy for information content given that the features are separated by linear classifiers afterwards. We observe the cumulative energy of the PCA components ordered by decreasing energy. We assume that a more

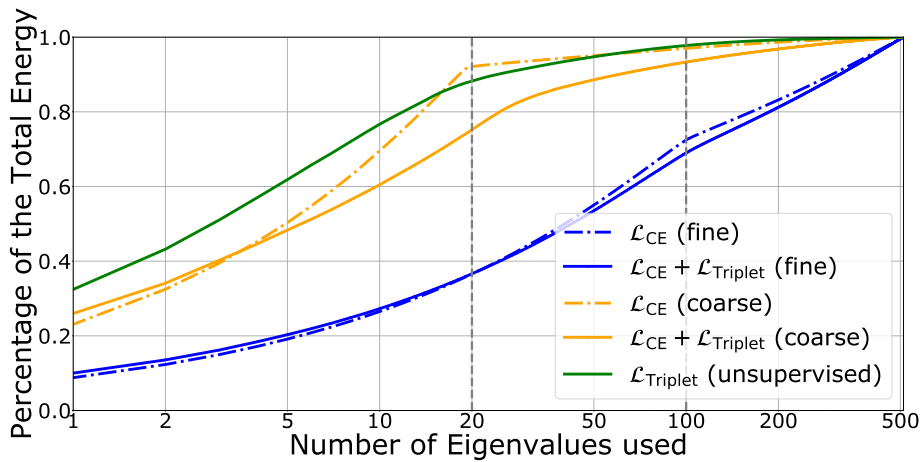


Figure 2.1 – Cumulative energy of the PCA decomposition of CIFAR-100 image embeddings, depending on the granularity of the training labels (20 or 100 classes).

uniform energy distribution (and thus lower curves) means that the representation is richer, since a few vector components cannot summarize it.

Figure 2.1 shows the results. When training with  $\mathcal{L}_{\text{CE}}$  loss, the most uniform distribution for the principal components is obtained for the fine supervision. This is expected since it is a finer-grain separation of entities and that can not be summarized with a subspace as small as the one associated with a relatively small number of categories. Notice that the training granularity (20 or 100 classes) can be read as an inflection point on the PCA decomposition curves. The loss  $\mathcal{L}_{\text{Triplet}}$  is not very informative on its own but does improve the cross-entropy representation when combined with it.

**Discussion.** This simple preliminary experiment shows that the label granularity has a strong impact on very basic statistics of the embedding distribution. It is the basic intuition behind Graft: a rich representation can be obtained using just coarse labels, if we combine them with a self-supervised loss.

## 2.3 Graft: Fitting a finer granularity

Figure 2.2 depicts our approach at training time. In this section, we discuss the different components and training losses. Then, we detail how we produce the category-level ranking, and how we perform on-the-fly classification.

### 2.3.1 Training procedure: Graft and Graft FC

We first introduce an instance loss inspired by BYOL [87] that favors fine-grained recognition. The Graft model includes a trunk network  $f_\theta$ , to which we add two multi-layer perceptrons (MLP): a “projector”  $P_\theta$  and a predictor  $q_\theta$ . In the Graft FC variant,  $P_\theta$  is linear for a more direct fair comparison with Wu et al. [220]’s projector. The learnable parameters are represented by the vector  $\theta$ . As in BYOL we define a “target network”  $f_\xi$  as an exponential moving average of the main network  $f_\theta$ : the parameters  $\xi$  are not learned, but computed as  $\xi \leftarrow \tau\xi + (1 - \tau)\theta$ , with a target decay rate  $\tau \in [0, 1]$ .

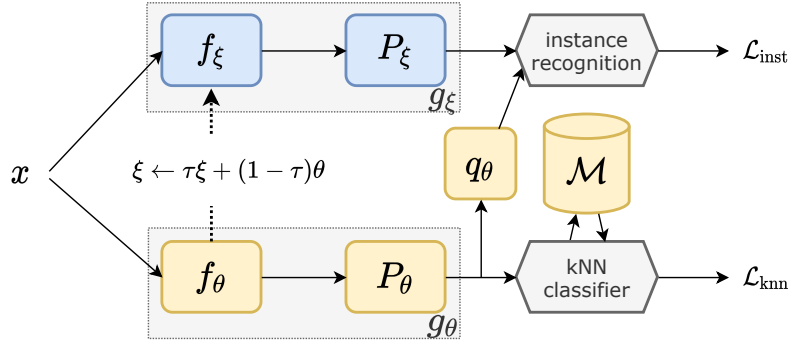


Figure 2.2 – Illustration of our method at train time. The convnet trunk that receives gradient is  $f_\theta$  and is used to update the target network  $f_\xi$  as a moving average. The database of neighbors is updated by averaging embedding in each mini-batch with corresponding embeddings in the database.

**Instance loss.** Each image  $x$  is transformed by  $T$  data augmentations  $(t_1, \dots, t_T)$ . Denoting  $\cos$  the cosine similarity and  $g_\theta(x) = P_\theta(f_\theta(x))$ , the instance loss is:

$$\mathcal{L}_{\text{inst}}(x) = - \sum_{1 \leq i \neq j \leq T} \frac{\cos(q_\theta \circ g_\theta(t_i(x)).g_\xi(t_j(x)))}{T(T-1)}, \quad (2.1)$$

The instance loss allows the network to discriminate at the instance level, which is a finer granularity than the class level. We give more insights about this loss in Section 2.2.

**kNN loss.** A parametric classifier with softmax yields a representation that does not generalize naturally to new classes [220] and is not adapted for kNN classification. Therefore, inspired by the neighborhood component analysis [83, 139, 169], Wu et al. [220] propose a loss function optimized directly for kNN evaluation, that we adopt and denote by  $\mathcal{L}_{\text{knn}}$ . Let  $x_i$  be a training image with coarse label  $y_i$  and  $\sigma$  a temperature hyper-parameter. For each image  $x_i$  we select  $x_j (j \neq i)$  as its neighbor with probability  $p_{i,j}$ , computed as

$$p_{i,j} \propto \exp(\cos(g_\theta(x_i), g_\theta(x_j))/\sigma), \quad (2.2)$$

where the  $p_{i,j}$  are normalized so that  $\sum_{j \neq i} p_{i,j} = 1$ . The loss is then defined as:

$$\mathcal{L}_{\text{knn}}(x_i, y_i) = - \log \sum_{j, y_j = y_i, j \neq i} p_{i,j}. \quad (2.3)$$

We  $\ell_2$ -normalize after the  $P_\theta$  projection. The  $\mathcal{L}_{\text{knn}}$  scores all classes with Equation 2.3.

**Memory of embeddings.** One of the limitations of the kNN approach is that it requires to use all the features of the training set. To avoid recomputing all the embeddings of the training set, we use a memory  $\mathcal{M} = \{m_1, \dots, m_i, \dots\}$ . It is updated as follows: when the image  $x_i$  in the training set is in the current mini-batch, we update its embedding  $m_i$  as follows:  $m_i \leftarrow \frac{1}{2}(m_i + g_\theta(x_i))$ . In order to limit the memory space needed, we apply the  $\mathcal{L}_{\text{knn}}$  loss on the space of the projected features, which allows us to store smaller embedding and hence requires less memory. For instance for ImageNet we have to store 1.2M training images. Without the projection with ResNet-50 architecture for  $f_\theta$ , the memory size is  $2048 \times 1.2M$  but with a projection on a space of size 256 the memory size is  $256 \times 1.2M$  what is  $\times 8$  smaller.

**Combined loss.** Our method is summarized in Figure 2.2. The total loss at training time for an image  $x$  with label  $y$  is:

$$\mathcal{L}_{\text{tot}}(x) = \mathcal{L}_{\text{knn}}(g_{\theta}(x), y) + \mathcal{L}_{\text{inst}}(x). \quad (2.4)$$

Section 2.5 empirically shows that weighting differently the losses does not bring much difference.

**Adapting the architecture at test-time.** The training parameters include the model weights  $(f_{\theta}, P_{\theta})$  and the parameters related to  $\mathcal{L}_{\text{inst}}$  ( $f_{\xi}, P_{\xi}$  and  $g_{\theta}$ ) as described previously. At test time we remove the  $\mathcal{L}_{\text{inst}}$  branch, keeping only  $f_{\theta}$  and  $P_{\theta}$ . In order to have consistent representations of all the training images with the final weights, we re-compute  $m_i = g_{\theta}(x_i)$  for each training image  $x_i$  and store it in  $\mathcal{M}$ .

### 2.3.2 Category-level retrieval

For a given test image  $x'$  the task is to order by semantic relevance all images from the training collection. In our coarse-to-fine case, a search result is deemed correct if it has the same fine label as the query.

**Cosine-based ranking.** The standard strategy to order the images is to compute  $g_{\theta}(x')$ , and to order all images  $x_i$  in the collection by their cosine similarity score  $\cos(g_{\theta}(x_i), g_{\theta}(x'))$  to the query (the  $g_{\theta}(x_i)$  are pre-computed in  $\mathcal{M}$ ). The experiments in Section 2.4 show that the way Graft embeddings are trained already improves the ranking with that method.

**Ranking conditioned by coarse prediction.** Let  $x'$  be a test image and  $x$  a training image with coarse class  $y$ . Let  $p_c(x, y)$  be the probability that the image  $x$  has coarse label  $y$  according to our classifier. Our conditional score  $\psi_{\text{cond}}$  is a compromise between the embedding similarity and the coarse classification, in spirit of the loss in Equation 2.4:

$$\psi_{\text{cond}}(x', x) = \cos(g_{\theta}(x'), g_{\theta}(x)) + \log\left(\frac{p_c(x', y)}{1 - p_c(x', y)}\right). \quad (2.5)$$

Note that, in that case, we rely on the fact that the collection in which we search is the training set, so that the coarse labels associated with the collection are known. In Section 2.4 we show that  $\psi_{\text{cond}}$  improves the category-level retrieval performance in the coarse-to-fine context.

**Conditional ranking: Oracle.** If we assume that the coarse label of the query test image is known (given by an oracle), then we can set  $p_c(x', y) = 1_{y=y'}$  with  $y'$  the coarse class of the test image  $x'$ . This boils down to systematically putting images with the same coarse class as the test image first in the ranking. Experimentally, this shows the impact of test label prediction on the score, and provides an upper bound on the performance of the conditional ranking strategy. It is also relevant in practice in a scenario where the user provides this coarse labeling, for instance by selecting it from an interface.

### 2.3.3 On-the-fly classification

In on-the-fly classification, a kNN classifier “knows” about the fine classes of the training images only at test time [220]. Such a non-parametric classification does not require any training or fine-tuning. Note this flexible classifier can handle settings with evolving datasets, including dynamic add-ons of new classes, although such setups are outside the scope of this chapter.



For a test image  $x$  we compute the embedding  $g_\theta(x)$  and compare it to the training image embeddings stored in  $\mathcal{M}$ . We select the  $k$  embeddings maximizing the cosine similarity to the query,  $(x_1, \dots, x_k)$ , with labels  $(y_1, \dots, y_k)$ . For a direct comparison with Wu et al. [220] and consistently with Equation 2.3, we apply an exponentially decreasing neighbour weighting that computes the probability that  $x$  belongs to class  $y$  as

$$p_{\text{kNN}}(x, y) \propto \sum_{j=1, y_j=y}^k \exp(\cos(g_\theta(x), g_\theta(x_j))/\sigma). \quad (2.6)$$

We normalize the probabilities so that  $\sum_y p_{\text{kNN}}(x, y) = 1$ .

## 2.4 Experiments

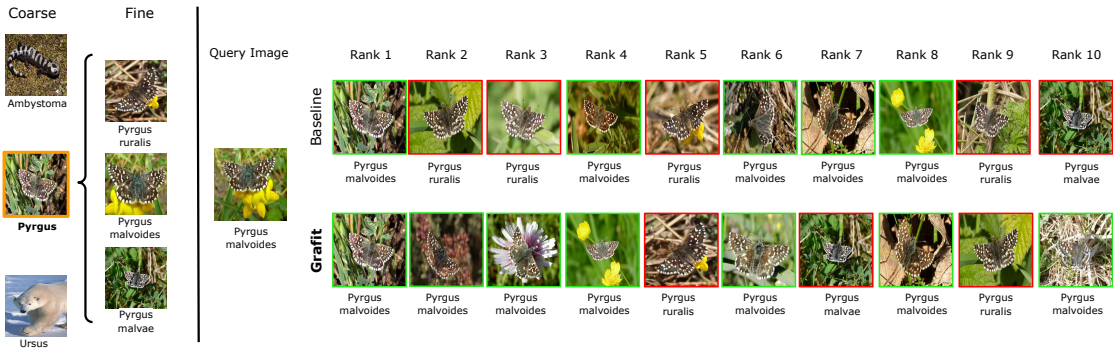


Figure 2.3 – Category-level retrieval orders images based on their semantic similarity to a query. Our Graft method, although it has used only coarse labels (like ‘pyrgus’) at training time, produces a ranking consistent with fine-grained labels. Unsupervised learning is a particular case of this task, in which the set of coarse labels is reduced to a singleton. Image credit: [2].

We consider evaluation scenarios where it is beneficial to learn at a finer granularity than that provided by the training labels. The first two tasks are coarse-to-fine tasks (category-level retrieval and on-the-fly classification), where we measure the capacity of the network to discriminate fine labels without having seen them at training time. The third protocol is vanilla transfer learning, where we transfer from Imagenet to a fine-grained dataset.

### 2.4.1 Datasets and evaluation metrics

We carry out our evaluations on public benchmarks, which statistics are detailed in Table 2.3.

**CIFAR-100** [124] has 100 classes grouped into 20 coarse concepts of 5 fine classes each. For instance the coarse class *large carnivore* includes fine classes *bear*, *leopard*, *lion*, *tiger* and *wolf*. In all experiments, we use the coarse concepts to train our models and evaluate the trained model using the fine-grained labels.

**ImageNet** [168] classes follow the WordNet [72] hierarchy. We use the 127 coarse labels defined in Huh et al. [110] in order to allow for a direct comparison with their method.

Table 2.3 – Datasets used for our different tasks. The four top datasets offer two or more levels of granularity, we use them for all coarse-to-fine tasks. The bottom three are fine-grained datasets employed to evaluate transfer learning performance.

Dataset	Train size	Test size	#classes
CIFAR-100 [124]	50,000	10,000	20/100
ImageNet [168]	1,281,167	50,000	127/1000
iNaturalist 2018 [104]	437,513	24,426	6/.../8,142
iNaturalist 2019 [103]	265,240	3,003	6/.../1,010
Flowers-102 [144]	2,040	6,149	102
Stanford Cars [121]	8,144	8,041	196
Food101 [23]	75,750	25,250	101

**iNaturalist-2018** offers 7 granularity levels from the most general to the most specific, that follow the biological taxonomy: Kingdom (6 classes), Phylum (25 classes), Class (57 classes), Order (272 classes), Family (1,118 classes), Genus (4,401 classes) and Species (8,142 classes). We consider pairs of (coarse,fine) granularity levels in our experiments. **iNaturalist-2019** is similar to iNaturalist-2018 with fewer classes and images, and yields similar conclusions.

**Flowers-102, Stanford Cars and Food101** are fine-grained benchmarks with no provided coarse labelling. Therefore we can use them for the transfer learning task.

**Evaluation metrics.** For category-level retrieval we report the mean average precision (mAP), as commonly done for retrieval tasks [8, 151]. For on-the-fly classification we report the top-1 accuracy in order to be directly comparable with prior work [220].

## 2.4.2 Baselines

We use existing baselines and introduce stronger ones:

**Wu’s baselines [220]** use activations of a network learned with cross-entropy loss, but evaluated with a kNN classifier. Huh et al. [110] evaluate how a network trained on the 127 ImageNet coarse classes transfers on the 1000 fine labels<sup>1</sup>.

**Our main baseline:** we learn a network with cross-entropy loss, and perform retrieval or kNN-classification with the  $\ell_2$ -normalized embedding produced by the model trunk. We point out that, thanks to our strong optimization strategy borrowed from recent works [96, 187], this baseline by itself outperforms all published results in several settings, for instance our ResNet-50 baseline without extra training data outperforms on ImageNet a ResNet-50 pretrained on the large dataset YFCC100M [232] (see Table 2.12).

**SNCA.** Wu et al. [220] proposed a SNCA, a model optimized with a loss suitable for knn classification. In our implementation, we add a linear operator  $P_\theta$  to the network trunk  $f_\theta$  when training the supervised loss  $\mathcal{L}_{\text{knn}}$ .

**SNCA+.** We improve SNCA with our stronger optimization procedure. The retrieval or kNN evaluation uses features from a MLP instead of a simple linear projector, which means that its number of parameters is on par with Grafit (and larger than Grafit FC).

1. They fine-tune a linear classifier with fine labels. We do not consider this task in the body of this chapter, but refer to section 2.5.2: our approach provides a significant improvement in this case as well.

Table 2.4 – **Coarse-to-fine: comparison with the state of the art** for category-level retrieval (mAP, %) and kNN classification (top-1, %), with the ResNet50 architecture. We compare Grafit with the state of the art [220] and our stronger baselines. We highlight methods that use more parameters (32.9M vs  $\sim$ 23.5M), we refer the reader to Table 2.7 for more details.

Method	CIFAR-100		ImageNet-1k	
	kNN	mAP	kNN	mAP
Baseline, Wu et al. [220]	54.2	–	48.1	–
SNCA, Wu et al. [220]	62.3	–	52.8	–
Baseline (ours)	71.8	42.5	54.7	22.7
ClusterFit+	72.5	23.0	59.5	12.7
SNCA+	72.2	35.9	55.4	31.8
Grafit FC	75.6	55.0	<b>69.1</b>	<b>44.4</b>
Grafit	<b>77.7</b>	<b>55.7</b>	<b>69.1</b>	42.9

**ClusterFit+.** Same as for SNCA, we improve ClusterFit [233] with our training procedure, and cross-validate the number of clusters (15000 for Imagenet and 1500 for CIFAR-100). As a result we improve its performance and have a fair comparison, everything being equal otherwise.

### 2.4.3 Experimental details

**Architectures.** Most experiments are carried out using the ResNet-50 architecture [94] except for Section 2.4.6 where we also use RegNet [156] and ResNeXt [230].

**Training settings.** Our training procedure borrows from the bag of tricks [96]: we use SGD with Nesterov momentum and cosine learning rates decay. We follow Goyal et al.’s [85] recommendation for the learning rate magnitude:  $lr = \frac{0.1}{256} \times \text{batchsize}$ . The data augmentation consists of random resized crop, RandAugment [44] and Erasing [245]. We train for 600 epochs with batches of 1024 images at resolution  $224 \times 224$  pixels (except for CIFAR-100:  $32 \times 32$ ). We set the temperature  $\sigma$  to 0.05 in all our experiments following Wu et al. [220]. We refer the reader to Section 2.5.1 for more training details.

For the on-the-fly classification task, the unique hyper-parameter cross-validated is  $k$ , with experiment with  $k \in \{10, 15, 20, 25, 30\}$ .

### 2.4.4 Coarse-to-fine experiments

**CIFAR and ImageNet experiments.** Table 2.4 compares Grafit results for coarse to fine tasks with the baselines from Section 2.4.2. On CIFAR-100, Grafit outperforms other methods by +5.5% top-1 accuracy. On ImageNet the gain over other methods is +13.7%.

Grafit also outperforms other methods on category-level retrieval, by 13.2% on CIFAR and 11.1% on ImageNet. In the Table 2.4 we shows that Grafit not only provides a better on-the-fly classification (as evaluated by the kNN metric), but that the ranked list is more relevant to the query (results for mAP).

**Coarse-to-Fine with different taxonomic ranks.** We showcase Grafit on various levels of coarse granularity by training one model on each annotation level of iNaturalist-2018 and evaluating on all levels with kNN classification (Table 2.5) and retrieval (Table 2.6).

Table 2.5 – kNN evaluation on iNaturalist-2018 with different semantic levels. The symbol  $\emptyset$  refers to the unsupervised case (a unique class). We compare with the best competing method according to Table 2.4.

		Train →	$\emptyset$	Kingdom	Phylum	class	Order	Family	Genus	Species
		↓Test / #classes →	1	6	25	57	272	1,118	4,401	8,142
ClusterFit+	Kingdom		70.9	94.7	95.0	95.3	95.6	96.2	96.3	96.1
	Phylum		48.8	87.4	90.3	90.7	91.1	92.6	92.6	92.2
	Class		40.4	80.2	83.8	85.7	86.7	88.8	88.8	88.2
	Order		17.1	54.5	59.0	61.4	70.8	73.9	74.3	72.3
	Family		5.6	38.3	42.1	44.4	54.3	63.0	64.2	61.9
	Genus		0.9	26.7	29.5	31.5	40.1	49.4	53.9	51.7
	Species		0.3	21.8	23.7	25.2	32.7	40.3	44.7	43.4
	$\emptyset$									
Graftit	Kingdom		95.5	98.1	98.2	98.2	98.2	98.2	98.4	98.3
	Phylum		90.0	94.1	96.6	96.7	96.8	96.7	96.9	96.7
	Class		82.2	87.5	90.9	94.5	94.9	94.9	95.0	95.0
	Order		54.0	61.7	66.9	72.7	87.1	87.5	87.6	87.3
	Family		33.7	42.1	48.7	55.1	70.9	81.8	82.4	82.1
	Genus		20.5	27.0	33.5	39.5	54.2	64.6	75.6	75.5
	Species		15.9	20.4	25.5	30.8	42.7	51.2	61.9	67.7
	$\emptyset$									

Table 2.6 – Category-retrieval evaluation (mAP, %) on iNaturalist-2018 with different semantics levels. We compare with the best baseline according to Table 2.4.

		Train →	Kingdom	Phylum	class	Order	Family	Genus	Species
		↓Test / #classes →	6	25	57	272	1,118	4,401	8,142
SNCA+	Kingdom		97.6	83.3	75.9	59.2	56.0	54.9	55.0
	Phylum		59.8	91.7	79.4	49.1	35.0	32.3	32.2
	Class		41.3	73.1	89.9	49.2	28.1	23.6	23.0
	Order		9.09	24.9	35.7	77.9	35.3	18.0	15.0
	Family		2.24	6.43	11.2	35.7	68.4	29.1	21.7
	Genus		0.39	2.47	5.03	18.1	36.6	60.5	46.0
	Species		0.19	1.86	3.80	12.8	26.4	46.0	54.9
	$\emptyset$								
Graftit	Kingdom		98.6	88.3	79.7	60.8	58.0	55.9	55.5
	Phylum		67.8	97.2	82.1	50.9	38.9	34.2	33.0
	Class		50.1	74.9	95.4	51.2	32.3	25.9	24.1
	Order		17.7	30.7	42.7	88.3	42.3	21.1	16.2
	Family		8.70	13.2	18.0	43.9	83.1	34.8	24.2
	Genus		6.78	9.72	13.5	29.0	46.9	77.2	53.9
	Species		6.45	9.02	12.1	23.6	35.6	55.4	70.0
	$\emptyset$								

Figure 2.4 presents results with retrieval and kNN classification for two of the most interesting cases: when the train and test granularities are the same (left), and on the finest test level (Species) with varying granularities at training time (right). On the left, the accuracy of all methods decreases as the granularity increases: this is expected as the task moves from coarse classification to fine, as it is more difficult to discriminate amongst a larger number of classes.

We observe that the performance drop of Graftit for category-level retrieval is reduced in comparison with other methods. On the right figures, the accuracy of all methods increases as the level of annotation increases (keeping evaluation at Species). Graftit also stands out in this context, outperforming other methods.

We report comprehensive results with Graftit and the baselines from Section 2.4.2 on iNaturalist-2019 & 2018 in the section 2.5.3.

**Visualizations.** Figure 2.3 shows visual results for the category-level retrieval task with Graftit. All the results for the baseline and Graftit have the correct coarse label, but our method is better at a finer granularity. In Section 2.6 we show that the improvement is even more evident when the granularity level at training time is coarser.

Figure 2.5 presents t-SNE visualizations [202] of the latent spaces associated with the baseline and Graftit for images associated with a sub-hierarchy of iNaturalist-2018.

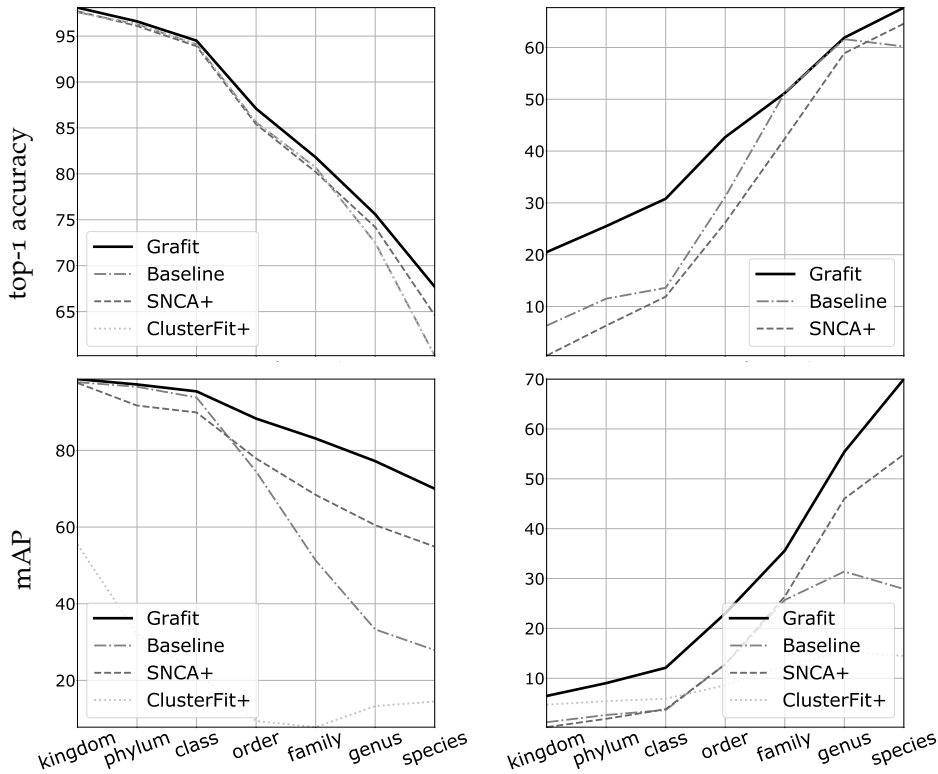


Figure 2.4 – Evaluation on iNaturalist-2018 [104] with and *left*: train=test granularity *right*: test at finest granularity (species). We compare our method Grafit, SNCA+, ClusterFit+ and Baseline. *Top*: on-the-fly kNN classification (top-1 accuracy); *bottom*: category-level retrieval (mAP).

### 2.4.5 Ablation studies

**Losses, architectural choice and conditioning.** Table 2.7 presents a study on CIFAR-100 and ImageNet-1k, where we ablate several components of our method. A large improvement stems from the instance loss when it supplements the supervised kNN loss. It is key for discriminating at a finer grain. The category-level retrieval significantly benefits from our approach, rising from 22.7% to 44.4% in the best case. Coarse conditioning also has a consistent measurable impact on performance, yielding around 3 mAP points across the various settings.

**Sanity check: training with coarse vs fine labels.** Table 2.8 compares the performance gap of several methods when training with coarse labels vs fine labels. The performance improvement of Grafit over competing methods on Imagenet is quite sizable: with fine-tuning, Grafit with coarse labels is almost on par with the baseline on fine labels. For on-the-fly classification, Grafit with coarse labels reaches 69.1% performance on Imagenet, significantly decreasing the gap with fine-grained labels settings. The kNN classification performance is 79.3%. This concurs with our prior observations in Section 2.4.4 on iNaturalist-2018.

Overall, in this setting Grafit provides some slight yet systematic improvement over the baseline. With a ResNet-50 architecture at image resolution  $224 \times 224$  pixels, Grafit reaches **79.6%** top-1 accuracy with a kNN classifier on ImageNet, which is competitive with classical cross-entropy results published for this architecture. See Section 2.5 for a comparison (Table 2.12) and more results on ImageNet-1k.

Table 2.7 – Ablation study on CIFAR-100 and ImageNet with ResNet50 architecture. We report results both for on-the-fly classification (kNN classifier, top-1 accuracy, %) and category-level retrieval (mAP, %). The rows corresponding to the main baselines and methods discussed through this chapter are highlighted: our baseline and improved SNCA+ in grey and red, and our two variants GrafIt-FC and GrafIt in blue. The last row is the result that GrafIt would obtain with a perfect coarse classification.

Loss			knn head proj.	coarse cond.	CIFAR-100		ImageNet-1k		
$\mathcal{L}_{CE}$	$\mathcal{L}_{knn}$	$\mathcal{L}_{inst}$	$P_\theta$		kNN	mAP	kNN	mAP	#Params
✓	–	–	–	–	71.8	42.5	54.7	22.7	23.5M
✓	–	–	–	✓	71.8	43.1	54.7	24.4	23.5M
–	–	✓	–	–	54.3	14.3	41.7	3.47	23.5M
✓	–	✓	–	–	76.9	51.0	65.0	26.0	23.5M
–	✓	–	FC	–	70.0	39.7	57.8	30.7	23.8M
–	✓	✓	FC	–	75.6	53.6	<b>69.1</b>	41.7	23.8M
–	✓	✓	FC	✓	75.6	55.0	<b>69.1</b>	<b>44.4</b>	23.8M
–	✓	–	MLP	–	72.2	35.9	55.4	31.8	32.9M
–	✓	–	MLP	✓	72.2	41.4	55.4	32.9	32.9M
–	✓	✓	MLP	–	<b>77.7</b>	52.9	<b>69.1</b>	39.4	32.9M
–	✓	✓	MLP	✓	<b>77.7</b>	<b>55.7</b>	<b>69.1</b>	42.9	32.9M
–	✓	✓	MLP	oracle	77.7	59.3	69.1	47.2	32.9M

Table 2.8 – We compare coarse-to-fine and fine-to-fine context with mAP (%), kNN (top-1, %) and fine-tuning (FT) with fine labels (top-1, %) on ImageNet.

Method (with ResNet50)	Train Coarse			Train Fine		
	mAP	kNN	FT	mAP	kNN	FT
Baseline	22.7	54.7	78.1	51.5	78.0	<b>79.3</b>
SNCA+	31.8	55.4	77.9	72.0	79.1	77.4
GrafIt FC	<b>44.4</b>	<b>69.1</b>	<b>78.3</b>	<b>72.4</b>	79.2	78.5
GrafIt	42.9	<b>69.1</b>	77.9	71.2	<b>79.6</b>	78.0

## 2.4.6 Transfer Learning to fine-grained datasets

We now evaluate GrafIt for transfer learning task on fine-grained datasets.

**Settings.** We initialize the network trunk with ImageNet pre-trained weights and fine-tune the model. For our method, the network trunk  $f_\theta$  remains fixed and the projector  $P_\theta$  is discarded. For all methods we fine-tune during 240 epochs with a cosine learning rate schedule starting at 0.01 and batches of 512 images (details in Section 2.5.4).

**Classifier.** We experiment with two types of classifiers: a standard linear classifier (FC) and a multi-layer perceptron (MLP) composed of two linear layers separated by a batch-normalization and a ReLU activation. We introduce this MLP because, during training, both GrafIt and SNCA+ employ an MLP projector, so their feature space is not learned to be linearly separable. In contrast, the baseline is trained with a cross-entropy loss associated with a linear classifier.

**Tasks.** We evaluate on five classical transfer learning datasets: Oxford Flowers-102 [144], Stanford Cars [121], Food101 [23], iNaturalist 2018 [104] & 2019 [103]. Table 2.3 summarizes some statistics associated with each dataset.

**Results.** Table 2.9 compares a ResNet-50 pretrained on ImageNet with GrafIt, SNCA+, ClusterFit [233] and our baseline on five transfer learning benchmarks. Our method outperforms all methods. The table also shows the relatively strong performance of SNCA+.

Table 2.9 – Comparison of transfer learning performance for different pre-training methods. All methods use a ResNet-50 pre-trained on Imagenet. The training procedures are the same (except the result reported for ClusterFit [233]). We report the top-1 accuracy (%) with a single center crop evaluation at resolution  $224 \times 224$ . See Table 2.15 of Appendix 2.5.4 for additional results with other architectures.

Dataset	Baseline	ClusterFit [233]	ClusterFit+	SNCA+	Grafit	Grafit FC
Flowers-102	96.2	–	96.2	<b>98.2</b>	<b>98.2</b>	97.6
Stanford Cars	90.0	–	89.4	92.5	92.5	<b>92.7</b>
Food101	88.9	–	88.9	88.8	<b>89.5</b>	88.7
iNaturalist 2018	68.4	49.7	67.5	69.2	<b>69.8</b>	68.5
iNaturalist 2019	73.7	–	73.8	74.5	<b>75.9</b>	74.6

Table 2.10 – State of the art for transfer learning with pretrained ImageNet-1k models. We report top-1 accuracy (%) with a single center crop. For Grafit we use the RegNetY-8.oGF [156] (39M params) with resolution  $384 \times 384$  pixels that is  $4\times$  faster than EfficientNet-B7 at inference. “Res” refer to the inference resolution in pixels.

Dataset	Best reported results (%)				Grafit Top-1
	State of the art	# Params	Res	Top-1	
Flowers-102	EfficientNet-B7 [187]	64M	600	98.8	<b>99.1</b>
Stanford Cars	EfficientNet-B7 [187]	64M	600	<b>94.7</b>	<b>94.7</b>
Food101	EfficientNet-B7 [187]	64M	600	93.0	<b>93.7</b>
iNaturalist 2018	ResNet-152 [40]	60M	224	69.1	<b>81.2</b>
iNaturalist 2019	–	–	–	–	<b>84.1</b>

Table 2.10 compares Grafit with the RegNetY-8.oGF [156] architecture against the state of the art, on the same benchmarks. Note that this architecture is significantly faster than the EfficientNet-B7 and ResNet-152 employed in other papers, and that we use a lower resolution in most settings.

In Table 2.10 we consider models pre-trained on ImageNet with and fine-tuned on the fine-grained target dataset. In each case we report results with Grafit (with a MLP for the projector  $P_\theta$ ) and Grafit FC. See more detailed results in Section 2.5 Table 2.16.

In summary, Grafit establishes the new state of the art. We point out that we have used a consistent training scheme across all datasets, and a single architecture that is more efficient than in competing methods.

## 2.5 Additional experiments

This section details the training procedure and provides more experimental results.

### 2.5.1 Training settings

As described in the main part, our training procedure is inspired by Tong et al. [96]: we use SGD with Nesterov momentum and cosine learning rates decay. We follow Goyal et al.’s [85] recommendation for the learning rate magnitude:  $\text{lr} = \frac{0.1}{256} \times \text{batchsize}$ . The augmentations include random resized crop, RandAugment [44] and Erasing [245]. We train for 600 epochs with batches of 1024 images of resolution  $224 \times 224$  pixels (except for CIFAR-100 where the resolution is  $32 \times 32$ ). For Grafit with  $\mathcal{L}_{\text{inst}}$  we use  $T = 4$  different data-augmentations on ImageNet and  $T = 8$

Table 2.11 – Category-level (mAP, %) and one-the-fly kNN classification (top-1, %) performance in a coarse-to-fine setting on CIFAR-100 with different loss weighting. Our total loss is defined as  $\mathcal{L}_{\text{tot}}(x) = \mathcal{L}_{\text{knn}}(g_{\theta}(x), y) + \lambda \mathcal{L}_{\text{inst}}(x)$  with  $\lambda$  being a real-valued coefficient.

$\lambda$	0.0	0.2	0.4	0.6	0.8	1.0	1.2	1.4
mAP	35.9	46.3	49.6	51.4	52.4	52.9	52.8	52.4
kNN	72.2	70.0	73.2	74.8	75.8	77.7	77.4	77.7

Table 2.12 – Performance comparison (top-1 accuracy) with our ResNet-50 baseline and state of the art ResNet-50 on ImageNet. All results are with single center crop evaluation with image resolution  $224 \times 224$ .

Method	Extra-data	Top-1 (%)
ResNet-50 [94] PyTorch	–	76.2
RandAugment [44]	–	77.6
CutMix [237]	–	78.6
Noisy-Student [228]	JFT-300M [228]	78.9
Billion Scale [232]	YFCC100M [190]	79.1
Our Baseline	–	<b>79.3</b>

on CIFAR-100. For the supervised loss we use one data-augmentation in order to have the same training procedure as our supervised baseline.

**Weighting of the losses.** We investigate the impact of weighting the losses  $\mathcal{L}_{\text{knn}}$  and  $\mathcal{L}_{\text{inst}}$ . For example, on CIFAR-100 classification, Table 2.11 shows that an equal weighting gives the best or near-best results. Therefore, to avoid adding a hyper-parameter and in order to simplify the method, we chose to not use weighting, *i.e.* we just sum up the two losses.

**A strong Baseline.** Our training procedure improves the ResNet-50 performance and thus is a strong baseline against which we can compare Grafit. Therefore, Table 2.12 compares our baseline on ImageNet with other ResNet-50 training procedures. We observe that our training procedure gives better results than many other approaches. This makes it possible to isolate the contribution of our improved training practices and that of the Grafit loss.

## 2.5.2 {coarse,fine}-to-{coarse,fine}: evaluation

We compare our main baselines and Grafit’s performance in the 4 following scenarios: coarse-to-coarse, coarse-to-fine, fine-to-fine and fine-to-coarse. The evaluations are performed with two classifiers: a kNN classifier (kNN) on top of the embeddings and a linear classifier with the fine-tuned network (FT) with a cross-entropy loss.

The results in Table 2.13 show that Grafit training improves the accuracy in almost all settings, including the fine-to-fine setting, which is just the regular image classification setting with the usual ImageNet labels.

## 2.5.3 Coarse-to-Fine with different taxonomic rank

**Datasets.** We carry out evaluations on iNaturalist-2018 (iNat-18), and with iNaturalist-2019 [103](iNat-19), which is a subset of iNaturalist-2018 [104] where classes with too few images have been removed. iNaturalist 2019 dataset is thus composed of 268,243 images divided into 1,010 classes



Table 2.13 – Performance comparison (top-1 accuracy) when learning and testing at different granularities (ResNet50). For CIFAR-100, there are 100 fine and 20 coarse concepts. ImageNet covers 1000 fine and 127 coarse concepts. We report the results of both the kNN classifier and of a linear classifier fine-tuned with the target granularity (FT).

	Method	↓ Test	Train Coarse		Train Fine	
			kNN	FT	kNN	FT
CIFAR-100	Baseline	Coarse	89.3 ±0.1	89.4 ±0.2	90.3 ±0.1	90.5 ±0.2
	SNCA+		88.4 ±0.3	88.9 ±0.3	88.8 ±0.1	90.2 ±0.1
	Grafit		<b>90.6</b> ±0.1	<b>90.6</b> ±0.1	<b>90.6</b> ±0.3	<b>90.9</b> ±0.2
	Baseline	Fine	71.8 ±0.3	82.3 ±0.2	82.7 ±0.2	82.7 ±0.2
	SNCA+		72.2 ±0.3	82.0 ±0.4	81.7 ±0.1	82.9 ±0.1
	Grafit		<b>77.7</b> ±0.2	<b>83.7</b> ±0.2	<b>83.2</b> ±0.3	<b>83.7</b> ±0.2
ImageNet-1k	Baseline	Coarse	87.0 ±0.1	<b>87.6</b> ±0.1	87.4 ±0.1	87.9 ±0.1
	SNCA+		87.7 ±0.1	87.5 ±0.1	88.9 ±0.1	87.2 ±0.1
	Grafit		<b>88.4</b> ±0.1	87.3 ±0.1	<b>89.2</b> ±0.1	<b>87.7</b> ±0.1
	Baseline	Fine	54.7 ±0.2	<b>78.1</b> ±0.1	78.0 ±0.1	<b>79.3</b> ±0.1
	SNCA+		55.4 ±0.2	77.9 ±0.1	79.1 ±0.1	77.4 ±0.1
	Grafit		<b>69.1</b> ±0.2	77.9 ±0.1	<b>79.6</b> ±0.1	78.0 ±0.1

at the finest level. From the coarse to the finest level, we have 3 classes for Kingdom, 4 classes for Phylum, 9 classes for Class, 34 classes for Order, 57 classes for Family, 72 classes for Genus and 1,010 classes for Species.

**Results.** We report exhaustive results with our two coarse-to-fine evaluation protocols with all our baselines on iNaturalist-2018 [104] and iNaturalist-2019 [103] in Table 2.14.

We comment more specifically the kNN classification accuracy (left) because for retrieval, Grafit outperforms all the baselines by a large margin. The table on the right is divided in 10 matrices each containing results for one combination of a method and a dataset (iNat-18 or 19).

The diagonal values in the matrices correspond to a traditional setting where the training and the test granularity are the same. Even in this case, the Grafit descriptors outperforms the baseline methods most often. On iNaturalist 2018, for Species, the finest and most challenging level, the additional Grafit loss improves the top-1 accuracy by 7% absolute. The gain is more marginal for iNaturalist 2019 (+0.9%), which shows that Grafit is especially useful for unbalanced class distributions where some classes are in a low-shot training regime.

The lower triangle of each matrix reports the coarse-to-fine results, which is the setting in which we focus in this chapter. Grafit obtains the best results for most combinations, with accuracy gains of around 10 points with respect to the baseline and by a few points for ClusterFit+. It is interesting to look at the  $\emptyset$  column, which is the unsupervised case. In that case, the baseline training reduces to a random network, but Grafit is able to extract signal from the kNN loss.

The upper triangle is the fine-to-coarse setting, where finer labels are available for the training images than what is used at test time. This is obviously not the setting of this chapter but it is worth discussing these results. A natural baseline for fine-to-coarse is to discard the fine labels and train only with the coarse labels induced by the fine annotation. This would yield the same accuracy as on the corresponding entry of the diagonal of the matrix. Irrespective of the method, the fine-to-coarse training does not necessarily outperform this simple strategy.

Table 2.14 – Evaluation on iNaturalist-2018/2019 with all combinations of training / testing semantic levels. *Left*: on-the-fly k-NN classification accuracy (top-1, %) *Right*: category-level retrieval (mAP, %). We highlight the **best** and **second-best** result across methods for each train-test granularity combination. The diagonals (test = train granularity) are in **bold**. Lower triangles are coarse-to-fine combinations, handled in this chapter. Upper triangles (fine-to-coarse) are reported for reference but not formally addressed by our approach: better strategies would exploit the hierarchy of concepts more explicitly.

Test \ Train		∅	King.	Phyl.	Class	Order	Fam.	Gen.	Spec.
iNaturalist-2018									
# classes:		1	6	25	57	272	1118	4401	8142
Baseline	Kingdom	70.9	<b>97.6</b>	98.0	98.1	98.2	98.2	97.9	97.5
	Phylum	48.8	88.0	<b>96.3</b>	96.4	96.6	96.7	96.2	95.2
	Class	40.4	77.1	86.7	<b>94.1</b>	94.7	94.8	94.1	92.9
	Order	17.1	43.6	55.0	61.0	<b>85.6</b>	86.6	85.5	82.6
	Family	5.6	23.0	32.8	36.7	62.0	<b>80.7</b>	79.7	76.1
	Genus	0.9	10.0	17.3	20.1	41.7	63.0	<b>72.5</b>	68.3
	Species	0.3	6.3	11.5	13.6	31.2	51.3	<b>61.6</b>	<b>60.2</b>
SNCA+	Kingdom	71.2	<b>97.7</b>	97.9	98.1	97.9	98.0	98.2	98.3
	Phylum	48.0	68.7	<b>96.1</b>	96.4	96.4	96.5	96.7	96.7
	Class	39.4	56.7	84.8	<b>93.9</b>	94.3	94.6	94.7	94.7
	Order	16.2	23.3	47.4	59.0	<b>85.4</b>	86.2	86.7	86.7
	Family	5.2	7.8	23.2	33.2	57.8	<b>80.2</b>	81.1	81.3
	Genus	0.9	1.3	10.4	17.6	36.9	56.8	<b>74.2</b>	74.1
	Species	0.3	0.5	6.3	11.9	26.2	42.4	58.9	<b>64.6</b>
ClusterFit+	Kingdom	70.9	<b>94.7</b>	95.0	95.3	95.6	96.2	96.3	96.1
	Phylum	48.8	87.4	<b>90.3</b>	90.7	91.1	92.6	92.6	92.2
	Class	40.4	80.2	83.8	<b>85.7</b>	86.7	88.8	88.8	88.2
	Order	17.1	54.5	59.0	61.4	<b>70.8</b>	73.9	74.3	72.3
	Family	5.6	38.3	42.1	44.4	54.3	<b>63.0</b>	64.2	61.9
	Genus	0.9	26.7	29.5	31.5	40.1	49.4	<b>53.9</b>	51.7
	Species	0.3	<b>21.8</b>	23.7	25.2	32.7	40.3	44.7	<b>43.4</b>
Grafit FC	Kingdom	91.1	<b>97.8</b>	98.1	98.4	98.3	98.4	98.5	98.4
	Phylum	81.7	93.0	<b>96.4</b>	96.9	97.0	96.9	97.1	96.8
	Class	71.9	86.0	90.7	<b>94.8</b>	95.0	95.1	95.3	95.0
	Order	41.8	58.5	66.8	72.2	<b>86.8</b>	87.1	87.3	87.2
	Family	22.4	38.8	48.4	54.4	70.4	<b>81.1</b>	81.6	81.7
	Genus	11.4	24.6	33.1	38.6	53.0	63.9	<b>73.8</b>	74.2
	Species	8.13	18.8	<b>25.6</b>	29.9	41.5	50.9	60.9	<b>65.9</b>
Grafit	Kingdom	95.5	<b>98.1</b>	98.2	98.2	98.2	98.2	98.4	98.3
	Phylum	90.0	94.1	<b>96.6</b>	96.7	96.8	96.7	96.9	96.7
	Class	82.2	87.5	90.9	<b>94.5</b>	94.9	94.9	95.0	95.0
	Order	54.0	61.7	66.9	72.7	<b>87.1</b>	87.5	87.6	87.3
	Family	33.7	42.1	48.7	55.1	70.9	<b>81.8</b>	82.4	82.1
	Genus	20.5	27.0	33.5	39.5	54.2	64.6	<b>75.6</b>	75.5
	Species	15.9	20.4	<b>25.5</b>	30.8	42.7	51.2	61.9	<b>67.7</b>
iNaturalist-2019									
# classes:		1	3	4	9	34	57	72	1010
Baseline	Kingdom	77.0	<b>98.9</b>	98.9	99.0	99.3	99.4	99.3	98.9
	Phylum	73.8	97.1	<b>98.7</b>	98.9	99.2	99.2	99.2	98.7
	Class	63.3	87.6	90.3	<b>98.0</b>	98.5	98.6	98.6	98.0
	Order	17.9	49.6	56.4	70.8	<b>95.6</b>	95.5	96.0	95.2
	Family	12.4	42.1	50.4	65.0	89.4	<b>94.8</b>	95.1	94.4
	Genus	9.6	39.2	46.5	62.1	86.1	91.5	<b>94.8</b>	93.9
	Species	1.5	9.8	13.5	20.6	34.5	39.9	42.4	<b>75.0</b>
SNCA+	Kingdom	76.9	<b>98.6</b>	98.9	99.2	99.2	99.3	99.1	99.0
	Phylum	73.3	87.1	<b>98.8</b>	99.1	99.1	99.1	98.9	99.0
	Class	62.3	74.9	84.1	<b>98.2</b>	98.6	98.3	98.1	97.8
	Order	17.6	19.7	30.2	55.4	<b>95.3</b>	95.2	95.2	94.2
	Family	12.2	12.7	20.7	45.5	88.2	<b>94.5</b>	94.6	93.5
	Genus	9.3	9.2	17.1	41.6	85.0	91.2	<b>94.0</b>	93.1
	Species	1.3	1.0	1.8	10.4	36.0	40.8	42.3	<b>74.7</b>
ClusterFit+	Kingdom	77.0	<b>96.4</b>	96.1	95.8	95.7	95.7	95.4	97.0
	Phylum	73.8	94.2	<b>95.0</b>	94.6	94.3	94.4	93.8	95.5
	Class	63.3	88.7	90.1	<b>91.3</b>	90.1	90.9	90.6	93.5
	Order	17.9	65.5	67.9	70.9	<b>76.8</b>	79.0	78.1	83.2
	Family	12.4	59.5	62.0	65.4	71.7	<b>75.6</b>	75.3	80.4
	Genus	9.6	56.9	59.3	62.7	68.7	72.6	<b>73.9</b>	78.6
	Species	1.5	24.5	25.6	27.3	31.1	33.6	33.9	<b>49.6</b>
Grafit FC	Kingdom	93.1	<b>98.9</b>	99.0	99.2	99.2	99.4	99.4	99.2
	Phylum	90.9	98.2	<b>98.9</b>	99.1	99.1	99.3	99.2	99.0
	Class	82.6	94.9	96.4	<b>98.2</b>	98.3	98.7	98.7	98.3
	Order	52.5	80.0	83.5	89.5	<b>95.8</b>	96.0	95.9	95.3
	Family	45.3	74.6	78.9	86.0	93.4	<b>95.2</b>	95.4	94.7
	Genus	41.7	71.7	76.3	84.0	91.7	93.4	<b>95.0</b>	94.3
	Species	12.0	29.5	32.6	40.4	51.8	53.2	53.9	<b>75.9</b>
Grafit	Kingdom	96.9	<b>99.2</b>	99.1	99.2	99.2	99.0	99.0	99.0
	Phylum	96.4	98.8	<b>98.9</b>	99.0	99.0	98.9	98.9	98.7
	Class	93.0	97.0	97.1	<b>98.2</b>	98.4	98.3	98.1	97.8
	Order	81.3	89.0	89.3	91.2	<b>95.9</b>	95.3	95.3	94.5
	Family	76.5	85.2	85.2	87.8	93.1	<b>94.5</b>	94.5	93.8
	Genus	73.8	82.7	83.1	85.8	91.3	92.6	<b>94.2</b>	93.4
	Species	31.0	41.6	41.4	46.0	51.8	53.5	55.3	<b>75.3</b>
iNaturalist-2019									
# classes:		3	4	9	34	57	72	1010	
Baseline	Kingdom	99.0	98.2	88.9	73.6	65.8	67.4	58.6	
	Phylum	87.1	<b>98.9</b>	90.8	71.7	59.8	61.6	51.7	
	Class	67.2	77.6	<b>98.2</b>	68.8	55.1	56.3	42.8	
	Order	15.1	21.1	33.7	<b>94.8</b>	68.6	57.6	26.2	
	Family	9.72	13.8	24.2	70.7	<b>94.2</b>	80.6	31.5	
	Genus	7.77	11.0	21.3	59.6	<b>81.4</b>	93.9	34.8	
	Species	1.09	1.55	3.60	10.8	14.8	<b>16.9</b>	<b>57.0</b>	
SNCA+	Kingdom	98.4	90.1	82.0	63.5	60.9	60.3	55.0	
	Phylum	84.1	<b>97.7</b>	87.7	62.6	55.9	55.3	49.3	
	Class	63.2	75.6	<b>95.5</b>	59.0	50.0	49.1	38.5	
	Order	11.5	17.2	32.4	<b>83.0</b>	64.3	54.4	15.7	
	Family	6.53	10.0	20.1	75.2	<b>90.9</b>	78.8	19.5	
	Genus	5.08	7.61	18.1	71.5	84.6	<b>92.8</b>	22.0	
	Species	0.40	0.65	2.11	15.4	17.1	18.6	<b>72.3</b>	
ClusterFit+	Kingdom	55.1	55.0	54.7	54.4	54.5	54.6	55.5	
	Phylum	49.0	<b>49.1</b>	48.8	48.2	48.3	48.4	49.3	
	Class	36.8	36.9	<b>37.1</b>	36.3	36.4	36.5	37.6	
	Order	7.5	7.7	8.2	<b>8.4</b>	8.5	8.4	9.7	
	Family	5.6	5.9	6.4	6.8	<b>7.4</b>	7.3	8.9	
	Genus	4.9	5.2	5.8	6.1	6.8	<b>6.9</b>	8.6	
	Species	2.4	2.5	2.9	3.5	4.1	4.2	<b>10.1</b>	
Grafit FC	Kingdom	99.2	93.2	86.5	63.8	62.3	62.2	56.1	
	Phylum	88.6	<b>99.2</b>	90.7	63.0	58.9	57.9	50.5	
	Class	70.4	80.9	<b>98.5</b>	61.6	53.9	52.5	39.9	
	Order	25.1	32.6	45.4	66.3	<b>96.3</b>	70.3	58.6	18.4
	Family	20.8	26.7	38.2	84.5	<b>95.8</b>	82.2	23.0	
	Genus	18.9	24.7	34.0	78.3	88.4	<b>95.7</b>	25.7	
	Species	6.83	9.63	14.7	28.4	29.7	31.5	<b>78.4</b>	
Grafit	Kingdom	99.4	93.1	85.9	62.7	61.5	60.9	56.6	
	Phylum	88.8	<b>99.2</b>	90.2	62.6	58.4	57.2	51.0	
	Class	71.8	81.9	<b>98.6</b>	61.3	53.2	52.0	40.4	
	Order	30.7	36.6	48.1	<b>96.4</b>	69.3	58.3	19.1	
	Family	28.2	30.8	41.8	82.5	<b>95.1</b>	81.5	23.7	
	Genus	28.0	29.8	40.5	76.2	87.5	<b>94.8</b>	26.3	
	Species	18.7	18.9	21.8	32.5	33.3	34.7	<b>77.5</b>	

## 2.5.4 Transfer Learning Tasks

This section details the experimental settings for the transfer learning and reports more results and comparisons.

**Fine-tuning settings** As described in Section 2.4.6 We initialize the network trunk with ImageNet pre-trained weights and fine-tune the model.

For our method, we keep the pre-trained network trunk  $f_\theta$ . But the projector  $P_\theta$  is discarded. For all methods we fine-tune during 240 epochs with a cosine learning rate schedule starting at 0.01, and batches of 512 images.

For fine-tuning results in Table 2.10 we additionally use Cutmix [237] and FixRes [199] during fine-tuning and we fine-tune with more epochs (1000 for Flowers [144] and Cars [121], 300 for Food-101 [23] and iNat [104, 103]). These choices improve the performance for all the methods.

**Results.** Table 2.15 compares the performance obtained with Grafit for different architectures. We report results with Grafit topped with either a multi-layer perceptron (MLP) or a linear classifier (FC). The accuracy increases for larger models. This shows that, although ResNet-50 serves as a running example architecture for Grafit, the method applies without modifications to other architectures.

Table 2.16 compares the performances obtained with Grafit and baselines with MLP and FC classifier. For all settings, the flexibility of the MLP is useful to outperform the linear classifier (FC). The transfer learning results are better or as good for Grafit variants. The gap with the baseline methods is higher for the iNaturalist variants. This is because the datasets are more challenging, as evidenced by the relatively low accuracies reported.

## 2.6 Visualization

**CIFAR.** Figure 2.6 shows for a given test image in CIFAR-100 the 10 nearest neighbours in the training according the cosine similarity in the embedding space. In Figure 2.6 models are trained on the 20 CIFAR-100 coarse classes. The correct classes are indicated in green. For example, in the first row, Grafit correctly identifies a butterfly query in 9 out of 10 results, while the baseline method succeeds only 5 times. The second row is a relative failure case, because Grafit confuses a van with a pickup truck. However, it correctly matches the colors of the vehicles.

**iNaturalist.** Figure 2.7, 2.8 and Figure 2.9 present similar results for three examples for iNaturalist-2018, but with several levels of granularity for the training set, which allow one to visualize the importance of the training granularity as well. Each granularity level is identified with a color. The frame color around the image indicates which at which granularity the match is correct: for example, light orange means it is correct at the order level and green means that the result is correct at the finest granularity (Species).

We can observe from the colors and the image content that the level at which Grafit is correct is almost systematically better than the baseline<sup>2</sup>. For example, the baseline model trained at the genus granularity in Figure 2.7 matches the deer query with a moose (rank 3).

In Figure 2.8, the butterfly is matched relatively easily with other butterflies by both classifiers, even when they are trained with coarse granularity. This is because butterflies have quite distinctive textures. However, Grafit slightly outperforms the baseline for finer granularity levels.

<sup>2</sup>. These examples are representative of typical comparisons, as we have not cherry-picked to show cases where our method is better.

Table 2.15 – Transfer learning task with various architectures pretrained on ImageNet with Grafit. We report the Top-1 accuracy (%) for the evaluation with a single center crop at resolution  $224 \times 224$ .

# Params	ResNeXt150- 32x4 [230]		ResNeXt150D- 32x4 [96]		ResNeXt150D- 32x8 [96]		RegNety- 4GF [156]		RegNety- 8GF [156]	
	25M		25M		48M		21M		39M	
Dataset	FC	MLP	FC	MLP	FC	MLP	FC	MLP	FC	MLP
Flowers-102 [144]	95.5	<b>98.3</b>	95.9	<b>98.6</b>	96.3	<b>98.7</b>	98.1	<b>98.6</b>	<b>99.0</b>	98.8
Stanford Cars [121]	91.6	<b>92.9</b>	88.7	<b>93.3</b>	90.9	<b>93.8</b>	<b>93.3</b>	92.7	<b>94.0</b>	93.4
Food101 [23]	89.6	<b>89.9</b>	90.2	<b>90.3</b>	90.9	<b>91.1</b>	91.2	91.3	92.1	<b>92.4</b>
iNaturalist 2018 [104]	67.7	<b>71.2</b>	68.9	<b>72.4</b>	71.4	<b>74.4</b>	73.8	<b>74.2</b>	76.4	<b>76.8</b>
iNaturalist 2019 [103]	75.3	<b>76.3</b>	75.8	<b>77.6</b>	77.8	<b>78.7</b>	<b>78.1</b>	77.9	79.8	<b>80.0</b>

Table 2.16 – Transfer learning with ResNet-50 pretrained on ImageNet. Comparison between different pre-training methods and two different classifiers trained on the target domain (a linear FC or an MLP). We report the top-1 accuracy (%) with a single center crop evaluation at resolution  $224 \times 224$ .

Dataset	Baseline		SNCA+		ClusterFit+		Grafit		Grafit FC	
	FC	MLP	FC	MLP	FC	MLP	FC	MLP	FC	MLP
Flowers-102 [144]	96.2	95.7	94.3	<b>98.2</b>	96.2	96.1	97.6	<b>98.2</b>	94.3	97.6
Stanford Cars [121]	90.0	89.8	91.6	<b>92.5</b>	89.4	89.3	91.4	<b>92.5</b>	91.4	<b>92.7</b>
Food101 [23]	88.2	88.9	88.7	88.8	88.5	88.9	88.9	<b>89.5</b>	88.5	88.7
iNaturalist 2018 [104]	65.0	68.4	64.7	69.2	64.2	67.5	65.6	<b>69.8</b>	65.2	68.5
iNaturalist 2019 [103]	72.8	73.7	73.1	74.5	71.8	73.8	74.1	<b>75.9</b>	73.9	74.6

Figure 2.9 shows an orca query, which is quite distinctive with its black-and-white skin. The baseline method is unable to distinguish it from other marine mammals, even when trained at the finest granularity. Grafit is able to distinguish these textures more accurately, so it gets perfect retrieval results even when trained at the genus granularity.

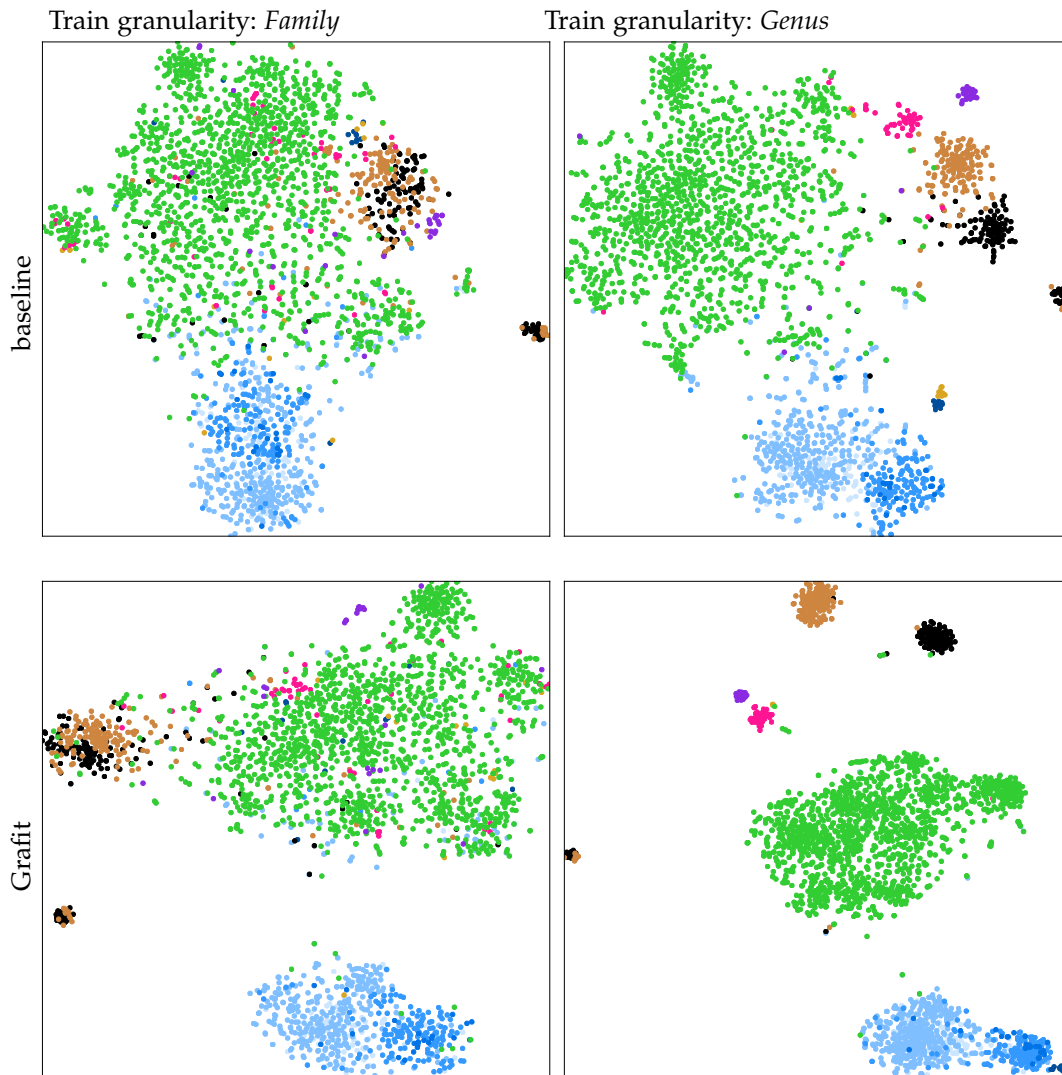


Figure 2.5 – t-SNE representations of features from images of the family *paridae*, focusing on the genus *baeolophus* (in blue). When trained with granularity *Family*, all depicted points have the same coarse label, while granularity *Genus* means that the network has seen 7 distinct labels. Visually, Graftit offers a better separation of the images than the baseline w.r.t. the two finest level ‘*Genus*’ and ‘*Species*’.

#### Family Paridae:

- Baeolophus
  - Baeolophus atricristatus
  - Baeolophus bicolor
  - Baeolophus inornatus
  - Baeolophus ridgwayi
  - Baeolophus wollwerberi
- Cyanistes
- Lophophanes
- Parus
- Periparus
- Poecile
- Sittiparus

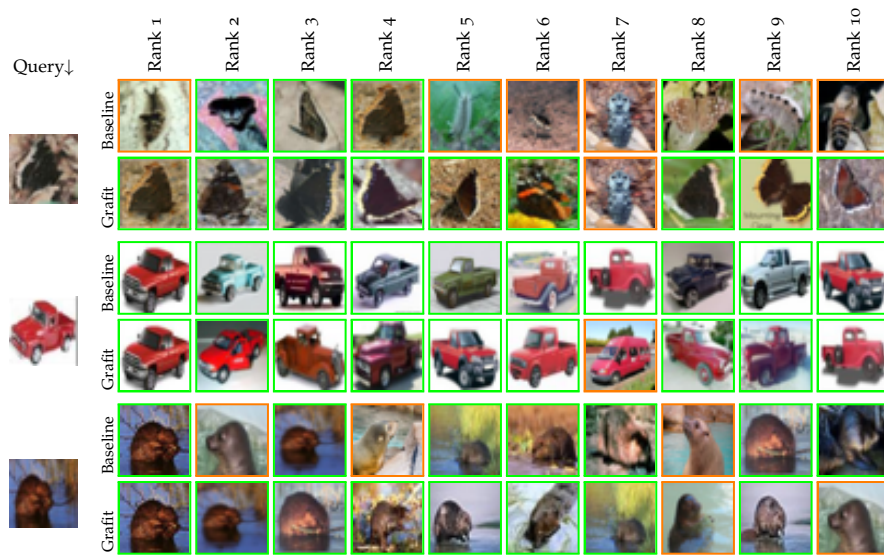


Figure 2.6 – CIFAR-100: For given test images (*top*), we present the ranked list of train images most similar with embeddings obtained with a baseline method (*top*) and our method (*bottom*) train with coarse labels. Images in **green** indicate that the image belongs to the correct fine class; **orange** indicates the correct coarse class but incorrect fine class. In this example, all results are correct w.r.t. coarse granularity.

## 2.7 Conclusion

In this chapter, we have introduced Grafit, a method to learn image representations at a finer granularity than the one offered by the annotation at training time. Inspired by recent self-supervised learning approach, we carefully design a joint learning scheme integrating instance and coarse-label based classification losses. For the latter one, we exploit a knn strategy but with a dedicated process to manage the memory both at train-time and for inference at test-time.

We propose two original use-cases to deeply evaluate coarse-trained fine-grained testing evaluation, for which Grafit exhibits outstanding performance.

It improves the performance for fine-grained category retrieval within a coarsely annotated collection. For on-the-fly kNN classification, Grafit significantly reduces the gap with a network trained with fine labels. For instance, we improve by **+16.3%** the top-1 accuracy for on-the-fly classification on ImageNet. This improvement is still +9.5% w.r.t. our own stronger baseline, everything being equal otherwise. Grafit also improves transfer learning: our experiments show that our representation discriminates better at a finer granularity. It also translates into better transfer learning performance to fine-grained datasets, outperforming the current state of the art with a more efficient network.


























































































































































Train levels	Method	Query Image	Neighbours in train									
			Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8	Rank 9	Rank 10
kingdom	Baseline											
	Grafit											
phylum	Baseline											
	Grafit											
class	Baseline											
	Grafit											
order	Baseline											
	Grafit											
family	Baseline											
	Grafit											
genus	Baseline											
	Grafit											
species	Baseline											
	Grafit											

Figure 2.7 – We compare Grafit and Baseline for different training granularity. We rank the 10 closest images in the iNaturalist-2018 train set for a query image in the test set. The ranking is obtained with a cosine similarity on the features space of each of the two approaches. See Table 7.1 for image copyrights.



Train levels	Method	Query Image	Neighbours in train									
			Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8	Rank 9	Rank 10
kingdom	Baseline											
	Grafit											
phylum	Baseline											
	Grafit											
class	Baseline											
	Grafit											
order	Baseline											
	Grafit											
family	Baseline											
	Grafit											
genus	Baseline											
	Grafit											
species	Baseline											
	Grafit											

Figure 2.8 – We compare Grafit and Baseline for different training granularity. We rank the 10 closest images in the iNaturalist-2018 train set for a query image in the test set. The ranking is obtained with a cosine similarity on the features space of each of the two approaches. See Table 7.2 for image copyrights.

Train levels	Method	Query Image	Neighbours in train									
			Rank 1	Rank 2	Rank 3	Rank 4	Rank 5	Rank 6	Rank 7	Rank 8	Rank 9	Rank 10
kingdom	Baseline											
	Grafit											
phylum	Baseline											
	Grafit											
class	Baseline											
	Grafit											
order	Baseline											
	Grafit											
family	Baseline											
	Grafit											
genus	Baseline											
	Grafit											
species	Baseline											
	Grafit											

Figure 2.9 – We compare Grafit and Baseline for different training granularity. We rank the 10 closest images in the iNaturalist-2018 train set for a query image in the test set. The ranking is obtained with a cosine similarity on the features space of each of the two approaches. See Table 7.3 for image copyrights.



## VISION TRANSFORMERS

Designing new deep architectures is an active research field in computer vision since the 2012's AlexNet [123]. Especially, attention-based models such as transformers have recently been introduced for image classification. In this chapter, we propose two works related to this architecture: DeiT for Data efficient image Transformers and CaiT for Class attention in image Transformers.

The chapter is organized as follows: we review related works in Section 3.1, and focus on transformers for image classification in Section 3.2. We introduce our distillation strategy for transformers in Section 3.3. The experimental section 3.4 provides analysis and comparisons against both convnets and recent transformers, as well as a comparative evaluation of our transformer-specific distillation. Section 3.4.5 details our training scheme. It includes an extensive ablation of our data-efficient training choices, which gives some insight on the key ingredients involved in DeiT. Section 3.6 introduces our second contribution, namely **class-attention layers**, that we present in Figure 3.7. It is akin to an encoder/decoder architecture, in which we explicitly separate the transformer layers involving self-attention between patches, from class-attention layers that are devoted to extract the content of the processed patches into a single vector so that it can be fed to a linear classifier. This explicit separation avoids the contradictory objective of guiding the attention process while processing the class embedding. We refer to this new architecture as CaiT (Class-Attention in Image Transformers). In the experimental Section 3.7, we empirically show the effectiveness and complementary of our approaches. We provide visualizations of the attention mechanisms in Section 3.7.2.5.

**Publication.** Chapter 3 is based on the papers “*Training data-efficient image transformers & distillation through attention*”, Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, Hervé Jégou, *ICML 2021* (see DeiT paper [193]) and “*Going deeper with Image Transformers*”, Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, Hervé Jégou, *ICCV 2021 (Oral)* (see CaiT paper [197]). The code is available at <https://github.com/facebookresearch/deit>.

### 3.1 Related work

As detailed in chapter 1, Transformers have been introduced by Vaswani et al. [205]. Many improvements of convnets for image classification are inspired by transformers. For example, Squeeze and Excitation [107], Selective Kernel [128] and Split-Attention Networks [241] exploit mechanisms akin to transformer's self-attention (SA).

Vision transformers (ViT) [61] close the gap with the state of the art on ImageNet, without using any convolution. This performance is remarkable since convnet methods for image classification have benefited from years of tuning and optimization [97, 214]. Nevertheless, according to this study [61], a pre-training phase on a large volume of curated data is required for the learned transformer to be effective.

**Knowledge Distillation** (KD), introduced by Hinton et al. [100], refers to the training paradigm in which a *student* model leverages “soft” labels coming from a strong *teacher* network. This is the output vector of the teacher’s softmax function rather than just the maximum of scores, which gives a “hard” label. Such training improves the performance of the student model (alternatively, it can be regarded as a form of compression of the teacher model into a smaller one – the student). On the one hand the teacher’s soft labels will have a similar effect to label smoothing [236]. On the other hand as shown by Wei et al. [213] the teacher’s supervision takes into account the effects of the data augmentation, which sometimes causes a misalignment between the real label and the image. For example, let us consider an image with a “cat” label that represents a large landscape and a small cat in a corner. If the cat is no longer on the crop of the data augmentation it implicitly changes the label of the image. KD can transfer inductive biases [4] in a soft way in a student model using a teacher model where they would be incorporated in a hard way. For example, it may be useful to induce biases due to convolutions in a transformer model by using a convolutional model as teacher. In this chapter, we study the distillation of a transformer student by either a convnet or a transformer teacher. We introduce a new distillation procedure specific to transformers and show its superiority.

**Deeper architectures** usually lead to better performance [94, 177, 184], however this complicates their training process [181, 182]. One must adapt the architecture and the optimization procedure to train them correctly. Some approaches focus on the initialization schemes [81, 94, 221], others on multiple stages of training [164, 177], multiple losses at different depth [184], adding components in the architecture [9, 243] or regularization [108].

## 3.2 Vision transformer: overview

In this section, we briefly recall preliminaries associated with the vision transformer [61, 205], and further discuss positional encoding and resolution.

**Multi-head Self Attention layers (MSA).** The attention mechanism is based on a trainable associative memory with (key, value) vector pairs. A *query* vector  $q \in \mathbb{R}^d$  is matched against a set of  $k$  *key* vectors (packed together into a matrix  $K \in \mathbb{R}^{k \times d}$ ) using inner products. These inner products are then scaled and normalized with a softmax function to obtain  $k$  weights. The output of the attention is the weighted sum of a set of  $k$  *value* vectors (packed into  $V \in \mathbb{R}^{k \times d}$ ). For a sequence of  $N$  query vectors (packed into  $Q \in \mathbb{R}^{N \times d}$ ), it produces an output matrix (of size  $N \times d$ ):

$$\text{Attention}(Q, K, V) = \text{Softmax}(QK^\top / \sqrt{d})V, \quad (3.1)$$

where the Softmax function is applied over each row of the input matrix and the  $\sqrt{d}$  term provides appropriate normalization.

In [205], a Self-attention layer is proposed. Query, key and values matrices are themselves computed from a sequence of  $N$  input vectors (packed into  $X \in \mathbb{R}^{N \times D}$ ):  $Q = XW_Q$ ,  $K = XW_K$ ,  $V = XW_V$ , using linear transformations  $W_Q, W_K, W_V$  with the constraint  $k = N$ , meaning that the attention is in between all the input vectors.

Finally, Multi-head self-attention layer (MSA) is defined by considering  $h$  attention “heads”, i.e.  $h$  self-attention functions applied to the input. Each head provides a sequence of size  $N \times d$ . These  $h$  sequences are rearranged into a  $N \times dh$  sequence that is reprojected by a linear layer into  $N \times D$ .

**Transformer block for images.** To get a full transformer block as in [205], we add a Feed-Forward Network (FFN) on top of the MSA layer. This FFN is composed of two linear layers separated by a GeLU activation [98]. The first linear layer expands the dimension from  $D$  to  $4D$ , and the second

layer reduces the dimension from  $4D$  back to  $D$ . Both MSA and FFN are operating as residual operators thanks to skip-connections, and with a layer normalization [7].

In order to get a transformer to process images, our work builds upon the ViT model [61]. It is a simple and elegant architecture that processes input images as if they were a sequence of input tokens. The fixed-size input RGB image is decomposed into a batch of  $N$  patches of a fixed size of  $16 \times 16$  pixels ( $N = 14 \times 14$ ). Each patch is projected with a linear layer that conserves its overall dimension  $3 \times 16 \times 16 = 768$ .

The transformer block described above is invariant to the order of the patch embeddings, and thus does not consider their relative position. The positional information is incorporated as fixed [205] or trainable [78] positional embeddings. They are added before the first transformer block to the patch tokens, which are then fed to the stack of transformer blocks.

**The class token** is a trainable vector, appended to the patch tokens before the first layer, that goes through the transformer layers, and is then projected with a linear layer to predict the class. This class token is inherited from NLP [56], and departs from the typical pooling layers used in computer vision to predict the class. The transformer thus process batches of  $(N + 1)$  tokens of dimension  $D$ , of which only the class vector is used to predict the output. This architecture forces the self-attention to spread information between the patch tokens and the class token: at training time the supervision signal comes only from the class embedding, while the patch tokens are the model’s only variable input.

**Fixing the positional encoding across resolutions.** Touvron et al. [199] show that it is desirable to use a lower training resolution and fine-tune the network at the larger resolution. This speeds up the full training and improves the accuracy under prevailing data augmentation schemes. When increasing the resolution of an input image, we keep the patch size the same, therefore the number  $N$  of input patches does change. Due to the architecture of transformer blocks and the class token, the model and classifier do not need to be modified to process more tokens. In contrast, one needs to adapt the positional embeddings, because there are  $N$  of them, one for each patch. Dosovitskiy et al. [61] interpolate the positional encoding when changing the resolution and demonstrate that this method works with the subsequent fine-tuning stage.

### 3.3 Distillation through attention

In this section, we assume we have access to a strong image classifier as a teacher model. It could be a convnet, or a mixture of classifiers. We address the question of how to learn a transformer by exploiting this teacher. As we will see in Section 3.4 by comparing the trade-off between accuracy and image throughput, it can be beneficial to replace a convolutional neural network by a transformer. This section covers two axes of distillation: hard distillation versus soft distillation, and classical distillation versus the distillation token.

**Soft distillation** [100, 213] minimizes the Kullback-Leibler divergence between the softmax of the teacher and the softmax of the student model.

Let  $Z_t$  be the logits of the teacher model,  $Z_s$  the logits of the student model. We denote by  $\tau$  the temperature for the distillation,  $\lambda$  the coefficient balancing the Kullback–Leibler divergence loss (KL) and the cross-entropy ( $\mathcal{L}_{\text{CE}}$ ) on ground truth labels  $y$ , and  $\psi$  the softmax function. The distillation objective is

$$\mathcal{L}_{\text{global}} = (1 - \lambda)\mathcal{L}_{\text{CE}}(\psi(Z_s), y) + \lambda\tau^2\text{KL}(\psi(Z_s/\tau), \psi(Z_t/\tau)). \quad (3.2)$$

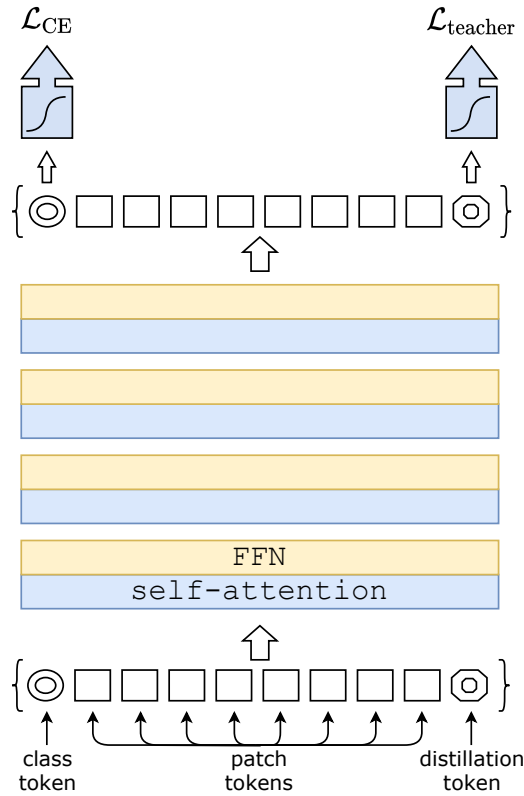


Figure 3.1 – Our distillation procedure: we simply include a new *distillation token*. It interacts with the class and patch tokens through the self-attention layers. This distillation token is employed in a similar fashion as the class token, except that on output of the network its objective is to reproduce the (hard) label predicted by the teacher, instead of true label. Both the class and distillation tokens input to the transformers are learned by back-propagation.

**Hard-label distillation.** We introduce a variant of distillation where we take the hard decision of the teacher as a true label. Let  $y_t = \operatorname{argmax}_c Z_t(c)$  be the hard decision of the teacher, the objective associated with this hard-label distillation is:

$$\mathcal{L}_{\text{global}}^{\text{hardDistill}} = \frac{1}{2} \mathcal{L}_{\text{CE}}(\psi(Z_s), y) + \frac{1}{2} \mathcal{L}_{\text{CE}}(\psi(Z_s), y_t). \quad (3.3)$$

For a given image, the hard label associated with the teacher may change depending on the specific data augmentation. We will see that this choice is better than the traditional one, while being parameter-free and conceptually simpler: The teacher prediction  $y_t$  plays the same role as the true label  $y$ .

Note also that the hard labels can also be converted into soft labels with label smoothing [185], where the true label is considered to have a probability of  $1 - \varepsilon$ , and the remaining  $\varepsilon$  is shared across the remaining classes. We choose  $\varepsilon = 0.1$  in our all experiments that use true labels.

**Distillation token.** We now focus on our proposal, which is illustrated in Figure 3.1. We add a new token, the distillation token, to the initial embeddings (patches and class token). Our distillation token is used similarly as the class token: it interacts with other embeddings through self-attention, and is output by the network after the last layer. Its target objective is given by the distillation component of the loss. The distillation embedding allows our model to learn from the

output of the teacher, as in a regular distillation, while remaining complementary and compatible with the class embedding.

Interestingly, we observe that the learned class and distillation tokens converge towards different vectors: the average cosine similarity between these tokens equal to 0.06. As the class and distillation embeddings are computed at each layer, they gradually become more similar through the network, all the way through the last layer at which their similarity is high ( $\cos=0.93$ ), but still lower than 1. This is expected since as they aim at producing targets that are similar but not completely identical.

We verified that our distillation token adds something to the model, compared to simply adding an additional class token associated with the same target label: instead of a teacher pseudo-label, we experimented with a transformer with two class tokens. Even if we initialize them randomly and independently, during training they converge towards the same vector ( $\cos=0.999$ ), and the output embedding are also quasi-identical. This additional class token does not bring anything to the classification performance. In contrast, our distillation strategy provides a significant improvement over a vanilla distillation baseline, as validated by our different experiments in the Section 3.4.2.

**Fine-tuning with distillation.** We use both the true label and teacher prediction during the fine-tuning stage at higher resolution. We use a teacher with the same target resolution, typically obtained from the lower-resolution teacher by the method of Touvron et al [199]. We have also tested with true labels only but this reduces the benefit of the teacher and leads to a lower performance in the settings we considered.

**Classification with our approach: joint classifiers.** At test time, both the class or the distillation embeddings produced by the transformer are associated with linear classifiers and able to infer the image label. Yet our referent method is the late fusion of these two separate heads, for which we add the softmax output by the two classifiers to make the prediction. We evaluate these three options in Section 3.4.

## 3.4 DeiT experiments

This section presents a few analytical experiments and results. We first discuss our distillation strategy. Then we comparatively analyze the efficiency and accuracy of convnets and vision transformers for image classification task.

### 3.4.1 Transformer models

As mentioned earlier, our architecture design is identical to the one proposed by Dosovitskiy et al. [61] with no convolutions. Our only differences are the training strategies, and the distillation token. Also we do not use a MLP head for the pre-training but only a linear classifier. To avoid any confusion, we refer to the results obtained in the prior work by ViT, and prefix ours by DeiT. If not specified, DeiT refers to our referent model DeiT-B, which has the same architecture as ViT-B. When we fine-tune DeiT at a larger resolution, we append the resulting operating resolution at the end, e.g, DeiT-B $\uparrow$ 384. Last, when using our distillation procedure, we identify it with an alembic sign as DeiT $\alembic$ .

The parameters of ViT-B (and therefore of DeiT-B) are fixed as  $D = 768$ ,  $h = 12$  and  $d = D/h = 64$ . We introduce two smaller models, namely DeiT-S and DeiT-Ti, for which we change



Table 3.1 – Variants of our DeiT architecture. The larger model, DeiT-B, has the same architecture as the ViT-B [61]. The only parameters that vary across models are the embedding dimension and the number of heads, and we keep the dimension per head constant (equal to 64). Smaller models have a lower parameter count, and a faster throughput. The throughput is measured for images at resolution  $224 \times 224$ .

Model	ViT model	embedding dimension	#heads	#layers	#params	training resolution	throughput (im/sec)
DeiT-Ti	N/A	192	3	12	5M	224	2536
DeiT-S	N/A	384	6	12	22M	224	940
DeiT-B	ViT-B	768	12	12	86M	224	292

Table 3.2 – We compare on ImageNet [168] the performance (top-1 acc., %) of the student as a function of the teacher model used for distillation.

Teacher Models	acc.	Student: DeiT-B $\uparrow$ <sub>384</sub>	
		pretrain	
DeiT-B	81.8	81.9	83.1
RegNetY-4GF	80.0	82.7	83.6
RegNetY-8GF	81.7	82.7	83.8
RegNetY-12GF	82.4	83.1	84.1
RegNetY-16GF	82.9	83.1	84.2

the number of heads, keeping  $d$  fixed. Table 3.1 summarizes the models that we consider in the first part of this chapter.

### 3.4.2 Distillation

Our distillation method produces a vision transformer that becomes on par with the best convnets in terms of the trade-off between accuracy and throughput, see Table 3.5. Interestingly, the distilled model outperforms its teacher in terms of the trade-off between accuracy and throughput. Our best model on ImageNet-1k is 85.2% top-1 accuracy outperforms the best ViT-B model pre-trained on JFT-300M at resolution 384 (84.15%). For reference, the current state of the art of 88.55% achieved with extra training data was obtained by the ViT-H model (600M parameters) trained on JFT-300M at resolution 512. Hereafter we provide several analysis and observations.

**Convnets teachers.** We have observed that using a convnet teacher gives better performance than using a transformer. Table 3.2 compares distillation results with different teacher architectures. The fact that the convnet is a better teacher is probably due to the inductive bias inherited by the transformers through distillation, as explained in Abnar et al. [4]. In all of our subsequent distillation experiments the default teacher is a RegNetY-16GF [156] (84M parameters) that we trained with the same data and same data-augmentation as DeiT. This teacher reaches 82.9% top-1 accuracy on ImageNet.

**Comparison of distillation methods.** We compare the performance of different distillation strategies in Table 3.3. Hard distillation significantly outperforms soft distillation for transformers, even when using only a class token: hard distillation reaches 83.0% at resolution  $224 \times 224$ , compared to the soft distillation accuracy of 81.8%. Our distillation strategy from Section 3.3 further improves the performance, showing that the two tokens provide complementary information useful for classification: the classifier on the two tokens is significantly better than the independent class and distillation classifiers, which by themselves already outperform the distillation baseline.

Table 3.3 – Distillation experiments on ImageNet with DeiT, 300 epochs of pre-training. We report the results for our new distillation method in the last three rows. We separately report the performance when classifying with only one of the class or distillation embeddings, and then with a classifier taking both of them as input. In the last row (class+distillation), the result correspond to the late fusion of the class and distillation classifiers. This late fusion corresponds to the average of classifiers outputs.

method ↓	Supervision		ImageNet top-1 (%)			
	label	teacher	Ti 224	S 224	B 224	B↑384
DeiT– no distillation	✓	✗	72.2	79.8	81.8	83.1
DeiT– usual distillation	✗	soft	72.2	79.8	81.8	83.2
DeiT– hard distillation	✗	hard	74.3	80.9	83.0	84.0
DeiT <sub>m</sub> : class embedding	✓	hard	73.9	80.9	83.0	84.2
DeiT <sub>m</sub> : distil. embedding	✓	hard	74.6	81.1	83.1	84.4
DeiT <sub>m</sub> : class+distillation	✓	hard	74.5	81.2	83.4	84.5

Table 3.4 – Disagreement analysis between convnet, image transformers and distilled transformers: We report the fraction of sample classified differently for all classifier pairs, i.e., the rate of different decisions. We include two models without distillation (a RegNetY and DeiT-B), so that we can compare how our distilled models and classification heads are correlated to these teachers.

	groundtruth	no distillation		DeiT <sub>m</sub> student (of the convnet)		
		convnet	DeiT	class	distillation	DeiT <sub>m</sub>
groundtruth	0.000	0.171	0.182	0.170	0.169	0.166
convnet (RegNetY)	0.171	0.000	0.133	0.112	0.100	0.102
DeiT	0.182	0.133	0.000	0.109	0.110	0.107
DeiT <sub>m</sub> – class only	0.170	0.112	0.109	0.000	0.050	0.033
DeiT <sub>m</sub> – distil. only	0.169	0.100	0.110	0.050	0.000	0.019
DeiT <sub>m</sub> – class+distil.	0.166	0.102	0.107	0.033	0.019	0.000

The distillation token gives slightly better results than the class token. It is also more correlated to the convnets prediction. This difference in performance is probably due to the fact that it benefits more from the inductive bias of convnets. We give more details and an analysis in the next paragraph. The distillation token has an undeniable advantage for the initial training.

**Agreement with the teacher & inductive bias?** As discussed above, the architecture of the teacher has an important impact. Does it inherit existing inductive bias that would facilitate the training? While we believe it difficult to formally answer this question, we analyze in Table 3.4 the decision agreement between the convnet teacher, our image transformer DeiT learned from labels only, and our transformer DeiT<sub>m</sub>.

Our distilled model is more correlated to the convnet than with a transformer learned from scratch. As to be expected, the classifier associated with the distillation embedding is closer to the convnet than the one associated with the class embedding, and conversely the one associated with the class embedding is more similar to DeiT learned without distillation. Unsurprisingly, the joint class+distil classifier offers a middle ground.

**Number of epochs.** Increasing the number of epochs significantly improves the performance of training with distillation, see Figure 3.2. With 300 epochs, our distilled network DeiT-B<sub>m</sub> is already better than DeiT-B. But while for the latter the performance saturates with longer schedules, our distilled network clearly benefits from a longer training time.

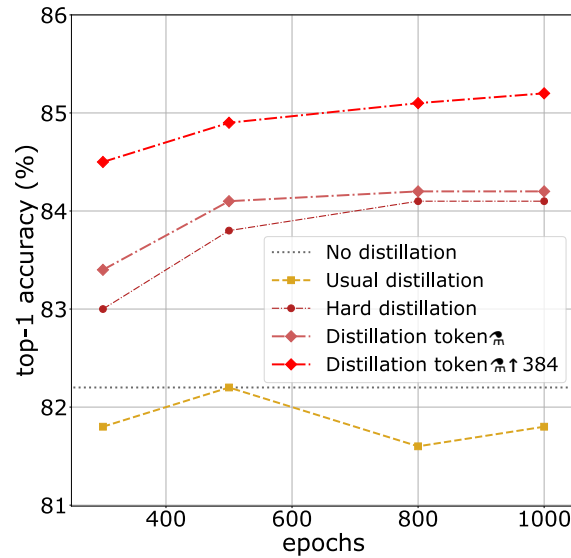


Figure 3.2 – Distillation on ImageNet [168] with DeiT-B: performance as a function of the number of training epochs. We also provide the performance without distillation but it saturates after 400 epochs.

### 3.4.3 Efficiency vs accuracy: a comparative study with convnets

In the literature, the image classification methods are often compared as a compromise between accuracy and another criterion, such as FLOPs, number of parameters, size of the network, etc.

We focus in Figure 3.3 on the tradeoff between the throughput (images processed per second) and the top-1 classification accuracy on ImageNet. We focus on the popular state-of-the-art EfficientNet convnet, which has benefited from years of research on convnets and was optimized by architecture search on the ImageNet validation set.

Our method DeiT is slightly below EfficientNet, which shows that we have almost closed the gap between vision transformers and convnets when training with ImageNet only. These results are a major improvement (+6.3% top-1 in a comparable setting) over previous ViT models trained on ImageNet-1k only [61]. Furthermore, when DeiT benefits from the distillation from a relatively weaker RegNetY to produce DeiT<sub>768</sub>, it outperforms EfficientNet. It also outperforms by 1% (top-1 acc.) the ViT-B model pre-trained on JFT300M at resolution 384 (85.2% vs 84.15%), while being significantly faster to train.

Table 3.5 reports the numerical results in more detail and additional evaluations on ImageNet V2 and ImageNet Real, that have a test set distinct from the ImageNet validation, which reduces overfitting on the validation set. Our results show that DeiT-B<sub>768</sub> and DeiT-B<sub>384</sub> outperform, by some margin, the state of the art on the trade-off between accuracy and inference time on GPU.

### 3.4.4 Transfer learning: Performance on downstream tasks

Although DeiT perform very well on ImageNet it is important to evaluate them on other datasets with transfer learning in order to measure the power of generalization of DeiT. We evaluated this on transfer learning tasks by fine-tuning on the datasets in Table 3.6. Table 3.7 compares DeiT transfer learning results to those of ViT [61] and state of the art convolutional architectures [187]. DeiT is on par with competitive convnet models, which is in line with our previous conclusion on ImageNet.

Network	#param.	image size	throughput (image/s)	ImNet top-1	Real top-1	V2 top-1
Convnets						
ResNet-18 [94]	12M	224 <sup>2</sup>	4458.4	69.8	77.3	57.1
ResNet-50 [94]	25M	224 <sup>2</sup>	1226.1	76.2	82.5	63.3
ResNet-101 [94]	45M	224 <sup>2</sup>	753.6	77.4	83.7	65.7
ResNet-152 [94]	60M	224 <sup>2</sup>	526.4	78.3	84.1	67.0
RegNetY-4GF [156]*	21M	224 <sup>2</sup>	1156.7	80.0	86.4	69.4
RegNetY-8GF [156]*	39M	224 <sup>2</sup>	591.6	81.7	87.4	70.8
RegNetY-16GF [156]*	84M	224 <sup>2</sup>	334.7	82.9	88.1	72.4
EfficientNet-B0 [187]	5M	224 <sup>2</sup>	2694.3	77.1	83.5	64.3
EfficientNet-B1 [187]	8M	240 <sup>2</sup>	1662.5	79.1	84.9	66.9
EfficientNet-B2 [187]	9M	260 <sup>2</sup>	1255.7	80.1	85.9	68.8
EfficientNet-B3 [187]	12M	300 <sup>2</sup>	732.1	81.6	86.8	70.6
EfficientNet-B4 [187]	19M	380 <sup>2</sup>	349.4	82.9	88.0	72.3
EfficientNet-B5 [187]	30M	456 <sup>2</sup>	169.1	83.6	88.3	73.6
EfficientNet-B6 [187]	43M	528 <sup>2</sup>	96.9	84.0	88.8	73.9
EfficientNet-B7 [187]	66M	600 <sup>2</sup>	55.1	84.3	–	–
EfficientNet-B5 RA [44]	30M	456 <sup>2</sup>	96.9	83.7	–	–
EfficientNet-B7 RA [44]	66M	600 <sup>2</sup>	55.1	84.7	–	–
KDforAA-B8	87M	800 <sup>2</sup>	25.2	85.8	–	–
Transformers						
ViT-B/16 [61]	86M	384 <sup>2</sup>	85.9	77.9	83.6	–
ViT-L/16 [61]	307M	384 <sup>2</sup>	27.3	76.5	82.2	–
DeiT-Ti	5M	224 <sup>2</sup>	2536.5	72.2	80.1	60.4
DeiT-S	22M	224 <sup>2</sup>	940.4	79.8	85.7	68.5
DeiT-B	86M	224 <sup>2</sup>	292.3	81.8	86.7	71.5
DeiT-B $\uparrow$ 384	86M	384 <sup>2</sup>	85.9	83.1	87.7	72.4
DeiT-Ti $\uparrow$ 384	6M	224 <sup>2</sup>	2529.5	74.5	82.1	62.9
DeiT-S $\uparrow$ 384	22M	224 <sup>2</sup>	936.2	81.2	86.8	70.0
DeiT-B $\uparrow$ 384	87M	224 <sup>2</sup>	290.9	83.4	88.3	73.2
DeiT-Ti $\uparrow$ 384 / 1000 epochs	6M	224 <sup>2</sup>	2529.5	76.6	83.9	65.4
DeiT-S $\uparrow$ 384 / 1000 epochs	22M	224 <sup>2</sup>	936.2	82.6	87.8	71.7
DeiT-B $\uparrow$ 384 / 1000 epochs	87M	224 <sup>2</sup>	290.9	84.2	88.7	73.9
DeiT-B $\uparrow$ 384	87M	384 <sup>2</sup>	85.8	84.5	89.0	74.8
DeiT-B $\uparrow$ 384 / 1000 epochs	87M	384 <sup>2</sup>	85.8	85.2	89.3	75.2

Table 3.5 – Throughput on and accuracy on ImageNet [168], ImageNet Real [18] and ImageNet V2 matched frequency [159] of DeiT and of several state-of-the-art convnets, for models trained with no external data. The throughput is measured as the number of images that we can process per second on one 16GB V100 GPU. For each model we take the largest possible batch size for the usual resolution of the model and calculate the average time over 30 runs to process that batch. With that we calculate the number of images processed per second. Throughput can vary according to the implementation: for a direct comparison and in order to be as fair as possible, we use for each model the definition in the same GitHub [214] repository.

\* : Regnet optimized with a similar optimization procedure as ours, which boosts the results. These networks serve as teachers when we use our distillation strategy.

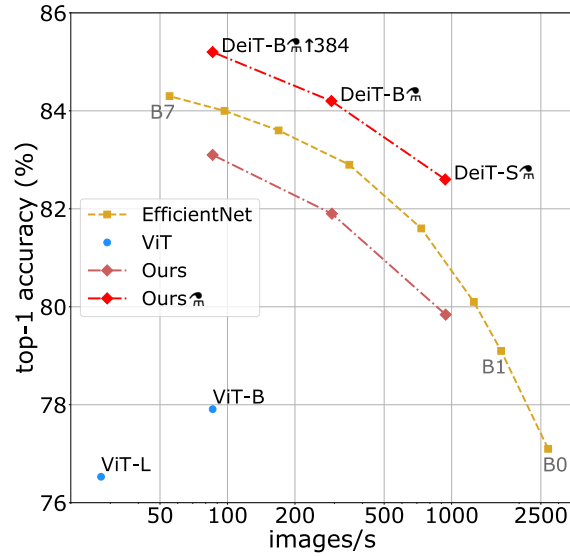


Figure 3.3 – Throughput and accuracy on ImageNet of our methods compared to EfficientNets, trained on ImageNet-1k only. The throughput is measured as the number of images processed per second on a V100 GPU. DeiT-B is identical to ViT-B, but the training is more adapted to a data-starving regime. It is learned in a few days on one machine. The symbol  $\overset{m}{\bullet}$  refers to models trained with our transformer-specific distillation. See Table 3.5 for details and more models.

Table 3.6 – Datasets used for our different tasks.

Dataset	Train size	Test size	#classes
ImageNet [168]	1,281,167	50,000	1000
iNaturalist 2018 [104]	437,513	24,426	8,142
iNaturalist 2019 [103]	265,240	3,003	1,010
Flowers-102 [144]	2,040	6,149	102
Stanford Cars [121]	8,144	8,041	196
CIFAR-100 [124]	50,000	10,000	100
CIFAR-10 [124]	50,000	10,000	10

**Comparison vs training from scratch.** We investigate the performance when training from scratch on a small dataset, without ImageNet-1k pre-training. We get the following results on the small CIFAR-10, which is small both w.r.t. the number of images and labels:

Method	RegNetY-16GF	DeiT-B	DeiT-B $\overset{m}{\bullet}$
Top-1	98.0	97.5	98.5

For this experiment, we tried we get as close as possible to the ImageNet pre-training counterpart, meaning that (1) we consider longer training schedules (up to 7200 epochs, which corresponds to 300 ImageNet epochs) so that the network has been fed a comparable number of images in total; (2) we re-scale images to  $224 \times 224$  to ensure that we have the same augmentation. The results are not as good as with ImageNet pre-training (98.5% vs 99.1%), which is expected since the network has seen a much lower diversity. However they show that it is possible to learn a reasonable transformer on CIFAR-10 only.

Table 3.7 – We compare Transformers based models on different transfer learning task with ImageNet pre-training. We also report results with convolutional architectures for reference.

Model	ImageNet	CIFAR-10	CIFAR-100	Flowers	Cars	iNat-18	iNat-19	im/sec
Grafit ResNet-50 [198]	79.6	–	–	98.2	92.5	69.8	75.9	1226.1
Grafit RegNetY-8GF [198]	–	–	–	99.0	94.0	76.8	80.0	591.6
ResNet-152 [40]	–	–	–	–	–	69.1	–	526.3
EfficientNet-B7 [187]	84.3	98.9	91.7	98.8	94.7	–	–	55.1
ViT-B/32 [61]	73.4	97.8	86.3	85.4	–	–	–	394.5
ViT-B/16 [61]	77.9	98.1	87.1	89.5	–	–	–	85.9
ViT-L/32 [61]	71.2	97.9	87.1	86.4	–	–	–	124.1
ViT-L/16 [61]	76.5	97.9	86.4	89.7	–	–	–	27.3
DeiT-B	81.8	99.1	90.8	98.4	92.1	73.2	77.7	292.3
DeiT-B $\uparrow$ 384	83.1	99.1	90.8	98.5	93.3	79.5	81.4	85.9
DeiT-B $\uparrow$ 768	83.4	99.1	91.3	98.8	92.9	73.7	78.4	290.9
DeiT-B $\uparrow$ 768	84.4	99.2	91.4	98.9	93.9	80.1	83.0	85.9

### 3.4.5 Training details & ablation

In this section we discuss the DeiT training strategy to learn vision transformers in a data-efficient manner. We build upon PyTorch [150] and the timm library [214]<sup>1</sup>. We provide hyper-parameters as well as an ablation study in which we analyze the impact of each choice.

**Initialization and hyper-parameters.** Transformers are relatively sensitive to initialization. After testing several options in preliminary experiments, some of them not converging, we follow the recommendation of Hanin et al. [90] to initialize the weights with a truncated normal distribution.

Table 3.9 indicates the hyper-parameters that we use by default at training time for all our experiments, unless stated otherwise. For distillation we follow the recommendations from Cho et al. [36] to select the parameters  $\tau$  and  $\lambda$ . We take the typical values  $\tau = 3.0$  and  $\lambda = 0.1$  for the usual (soft) distillation.

**Data-Augmentation.** Compared to models that integrate more priors (such as convolutions), transformers require a larger amount of data. Thus, in order to train with datasets of the same size, we rely on extensive data augmentation. We evaluate different types of strong data augmentation, with the objective to reach a data-efficient training regime.

Auto-Augment [45], Rand-Augment [44], and random erasing [245] improve the results. For the two latter we use the timm [214] customizations, and after ablation we choose Rand-Augment instead of AutoAugment. Overall our experiments confirm that transformers require a strong data augmentation: almost all the data-augmentation methods that we evaluate prove to be useful. One exception is dropout, which we exclude from our training procedure.

**Regularization & Optimizers.** We have considered different optimizers and cross-validated different learning rates and weight decays. Transformers are sensitive to the setting of optimization hyper-parameters. Therefore, during cross-validation, we tried 3 different learning rates ( $5 \cdot 10^{-4}$ ,  $3 \cdot 10^{-4}$ ,  $5 \cdot 10^{-5}$ ) and 3 weight decay (0.03, 0.04, 0.05). We scale the learning rate according to the batch size with the formula:  $lr_{scaled} = \frac{lr}{512} \times batchsize$ , similarly to Goyal et al. [85] except that we use 512 instead of 256 as the base value.

1. The timm implementation already included a training procedure that improved the accuracy of ViT-B from 77.91% to 79.35% top-1, and trained on Imagenet-1k with a 8xV100 GPU machine.

Ablation on ↓	Pre-training	Fine-tuning	Rand-Augment	AutoAug	Mixup	CutMix	Erasing	Stoch. Depth	Repeated Aug.	Dropout	Exp. Moving Avg.	top-1 accuracy	
												pre-trained 224 <sup>2</sup>	fine-tuned 384 <sup>2</sup>
none: DeiT-B	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8 ±0.2	83.1 ±0.1
optimizer	SGD	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✗	74.5	77.3
	adamw	SGD	✓	✗	✓	✓	✓	✓	✓	✗	✗	81.8	83.1
data augmentation	adamw	adamw	✗	✗	✓	✓	✓	✓	✓	✗	✗	79.6	80.4
	adamw	adamw	✗	✓	✓	✓	✓	✓	✓	✗	✗	81.2	81.9
	adamw	adamw	✓	✗	✗	✓	✓	✓	✓	✗	✗	78.7	79.8
	adamw	adamw	✓	✗	✓	✗	✓	✓	✓	✗	✗	80.0	80.6
	adamw	adamw	✓	✗	✗	✗	✓	✓	✓	✗	✗	75.8	76.7
regularization	adamw	adamw	✓	✗	✓	✓	✗	✓	✓	✗	✗	4.3*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✗	✓	✗	✗	3.4*	0.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✗	✗	✗	76.5	77.4
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✓	✗	81.3	83.1
	adamw	adamw	✓	✗	✓	✓	✓	✓	✓	✗	✓	81.9	83.1

Table 3.8 – Ablation study on training methods on ImageNet [168]. The top row ("none") corresponds to our default configuration employed for DeiT. The symbols ✓ and ✗ indicates that we use and do not use the corresponding method, respectively. We report the accuracy scores (%) after the initial training at resolution 224×224, and after fine-tuning at resolution 384×384. The hyper-parameters are fixed according to Table 3.9, and may be suboptimal.

\* indicates that the model did not train well, possibly because hyper-parameters are not adapted.

The best results use the AdamW optimizer with the same learning rates as ViT [61] but with a much smaller weight decay, as the weight decay reported in the ViT paper hurts the convergence in our setting.

We have employed stochastic depth [108], which facilitates the convergence of transformers, especially deep ones [69, 70]. For vision transformers, they were first adopted in the training procedure by Wightman [214]. Regularization like Mixup [242] and Cutmix [237] improve performance. We also use repeated augmentation [15, 101], which provides a significant boost in performance and is one of the key ingredients of our proposed training procedure.

**Exponential Moving Average (EMA).** We evaluate the EMA of our network obtained after training. There are small gains, which vanish after fine-tuning: the EMA model has an edge of is 0.1 accuracy points, but when fine-tuned the two models reach the same (improved) performance.

**Fine-tuning at different resolution.** We adopt the fine-tuning procedure from Touvron et al. [200]: our schedule, regularization and optimization procedure are identical to that of Fix-EfficientNet but we keep the training-time data augmentation (contrary to the dampened data augmentation of Touvron et al. [200]). We also interpolate the positional embeddings: In principle any classical image scaling technique, like bilinear interpolation, could be used. However, a bilinear interpolation of a vector from its neighbors reduces its  $\ell_2$ -norm compared to its neighbors. These low-norm vectors are not adapted to the pre-trained transformers and we observe a significant drop in accuracy if we employ use directly without any form of fine-tuning. Therefore we adopt a bicubic interpolation that approximately preserves the norm of the vectors, before fine-tuning the network with either AdamW [136] or SGD. These optimizers have a similar performance for the fine-tuning stage, see Table 3.8.

Methods	ViT-B [61]	DeiT-B
Epochs	300	300
Batch size	4096	1024
Optimizer	AdamW	AdamW
learning rate	0.003	$0.0005 \times \frac{\text{batchsize}}{512}$
Learning rate decay	cosine	cosine
Weight decay	0.3	0.05
Warmup epochs	3-4	5
Label smoothing $\varepsilon$	$\times$	0.1
Dropout	0.1	$\times$
Stoch. Depth	$\times$	0.1
Repeated Aug	$\times$	$\checkmark$
Gradient Clip.	$\checkmark$	$\times$
Rand Augment	$\times$	9/0.5
Mixup alpha.	$\times$	0.8
Cutmix alpha.	$\times$	1.0
Erasing prob.	$\times$	0.25

Table 3.9 – Ingredients and hyper-parameters for our method and ViT-B.

image throughput size (image/s)	ImageNet-1k [168] acc. top-1	Real [18] acc. top-1	V2 [159] acc. top-1
160 <sup>2</sup> 609.31	79.9	84.8	67.6
224 <sup>2</sup> 291.05	81.8	86.7	71.5
320 <sup>2</sup> 134.13	82.7	87.2	71.9
384 <sup>2</sup> 85.87	83.1	87.7	72.4

Table 3.10 – Performance of DeiT trained at size 224<sup>2</sup> for varying finetuning sizes on ImageNet-1k, ImageNet-Real and ImageNet-v2 matched frequency.

By default and similar to ViT [61] we train DeiT models with at resolution 224 and we fine-tune at resolution 384. We detail how to do this interpolation in Section 3.2. However, in order to measure the influence of the resolution we have finetuned DeiT at different resolutions. We report these results in Table 3.10.

**Training time.** A typical training of 300 epochs takes 37 hours with 2 nodes or 53 hours on a single node for the DeiT-B. As a comparison point, a similar training with a RegNetY-16GF [156] (84M parameters) is 20% slower. DeiT-S and DeiT-Ti are trained in less than 3 days on 4 GPU. Then, optionally we fine-tune the model at a larger resolution. This takes 20 hours on a single node (8 GPU) to produce a FixDeiT-B model at resolution 384×384, which corresponds to 25 epochs. Not having to rely on batch-norm allows one to reduce the batch size without impacting performance, which makes it easier to train larger models. Note that, since we use repeated augmentation [15, 101] with 3 different augmentation as in Touvron et al. [199], we only see one third of the images during a single epoch<sup>2</sup>.

2. Formally it means that we have 100 epochs, but each is 3x longer because of the repeated augmentations. We prefer to refer to this as 300 epochs in order to have a direct comparison on the effective training time with and without repeated augmentation.



### 3.5 Going deeper with vision transformers

So far in this chapter, we have introduced DeiT, which are image transformers that do not require very large amount of data to be trained, thanks to improved training and in particular a novel distillation procedure.

However, in DeiT works, there is no evidence that depth can bring any benefit when training on Imagenet only: the deeper ViT architectures have a low performance, while with DeiT we only considered transformers with 12 blocks of layers. In our early experiments, we observe that Vision Transformers become increasingly more difficult to train when we scale architectures. Depth is one of the main source of instability. For instance with our DeiT procedure we fail to properly converge above 18 layers without adjusting hyper-parameters. Large ViT [61] models with 24 and 32 layers were trained with large training datasets, but when trained on Imagenet only the larger models are not competitive. In the rest of this chapter, we study more specifically how to train deeper image transformers.

Residual architectures are prominent in computer vision since the advent of ResNet [94]. They are defined as a sequence of functions of the form

$$x_{l+1} = g_l(x_l) + R_l(x_l), \quad (3.4)$$

where the function  $g_l$  and  $R_l$  define how the network updates the input  $x_l$  at layer  $l$ . The function  $g_l$  is typically the identity, while  $R_l$  is the main building block of the network: many variants in the literature essentially differ on how this residual branch  $R_l$  is constructed or parametrized [156, 187, 230]. Residual architectures highlight the strong interplay between optimization and architecture design. As pointed out by He *et al.* [94], residual networks do not offer better representational power. They achieve better performance because they are easier to train: shortly after their seminal work, He *et al.* discussed [95] the importance of having a clear path both forward and backward, and advocate setting  $g_l$  to the identity function.

The vision transformers [61] instantiate a particular form of residual architecture: after casting the input image into a set  $x_0$  of vectors, the network alternates self-attention layers (SA) with feed-forward networks (FFN), as

$$\begin{aligned} x'_l &= x_l + \text{SA}(\eta(x_l)) \\ x_{l+1} &= x'_l + \text{FFN}(\eta(x'_l)) \end{aligned} \quad (3.5)$$

where  $\eta$  is the LayerNorm operator [7]. This definition follows the original architecture of Vaswani *et al.* [204], except the LayerNorm is applied before the block (*pre-norm*) in the residual branch, as advocated by He *et al.* [95]. Child *et al.* [35] adopt this choice with LayerNorm for training deeper transformers for various media, including for image generation where they train transformers with 48 layers.

How to normalize, weigh, or initialize the residual blocks of a residual architecture has received significant attention both for convolutional neural networks [26, 25, 95, 243] and for transformers applied to NLP or speech tasks [9, 109, 243]. In Section 3.5.1, we revisit this topic for transformer architectures solving image classification problems. Examples of approaches closely related to ours include Fixup [243], T-Fixup [109], ReZero [9] and SkipInit [53].

Following our analysis of the interplay between different initialization, optimization and architectural design, we propose an approach that is effective to improve the training of deeper architecture compared to current methods for image transformers. Formally, we add a learnable diagonal matrix on output of each residual block, initialized close to (but not at) 0. Adding this

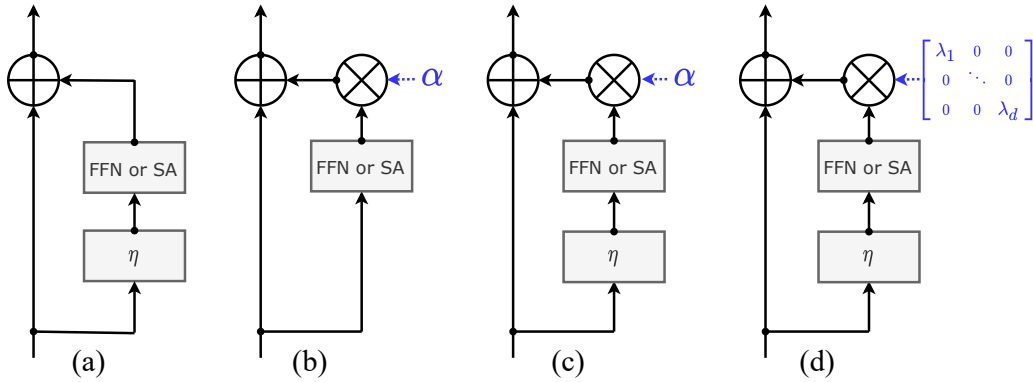


Figure 3.4 – Normalization strategies for transformer blocks. (a) The ViT image classifier adopts pre-normalization like Child *et al.* [35]. (b) ReZero/Skipinit and Fixup remove the  $\eta$  normalization and the warmup (i.e., a reduced learning rate in the early training stage) and add a learnable scalar initialized to  $\alpha = 0$  and  $\alpha = 1$ , respectively. Fixup additionally introduces biases and modifies the initialization of the linear layers. Since these methods do not converge with deep vision transformers, (c) we adapt them by re-introducing the pre-norm  $\eta$  and the warmup. Our main proposal (d) introduces a per-channel weighting (i.e., multiplication with a diagonal matrix  $\text{diag}(\lambda_1, \dots, \lambda_d)$ , where we initialize each weight with a small value as  $\lambda_i = \varepsilon$ ).

simple layer after each residual block improves the training dynamic, allowing us to train deeper high-capacity image transformers that benefit from depth. We refer to this approach as **LayerScale**.

### 3.5.1 Deeper image transformers with LayerScale

Our goal is to increase the stability of the optimization when training transformers for image classification especially when we increase their depth.

Figure 3.4 depicts the main variants that we compare for helping the optimization. They cover recent choices from the literature: as discussed in the introduction, the architecture (a) of ViT and DeiT is a pre-norm architecture [61, 193], in which the layer-normalisation  $\eta$  occurs at the beginning of the residual branch. Note that the original architecture of Vaswani *et al.* [204] applies the normalization after the block, but in our experiments the DeiT training does not converge with post-normalization.

Fixup [243], ReZero [9] and SkipInit [53] introduce learnable scalar weighting  $\alpha_l$  on the output of residual blocks, while removing the pre-normalization and the warmup, see Figure 3.4(b). This amounts to modifying Eqn. 3.5 as

$$\begin{aligned} x'_l &= x_l + \alpha_l \text{SA}(x_l) \\ x_{l+1} &= x'_l + \alpha'_l \text{FFN}(x'_l). \end{aligned} \quad (3.6)$$

ReZero simply initializes this parameter to  $\alpha = 0$ . Fixup initializes this parameter to  $\alpha = 1$  and makes other modifications: it adopts different policies for the initialization of the block weights, and adds several weights to the parametrization. In our experiments, these approaches do not converge even with some adjustment of the hyper-parameters.

Our empirical observation is that removing the warmup and the layer-normalization is what makes training unstable in Fixup and T-Fixup. Therefore we re-introduce these two ingredients so that Fixup and T-Fixup converge with DeiT models, see Figure 3.4(c). As we see in the experimental section, these amended variants of Fixup and T-Fixup are effective, mainly due to the learnable

parameter  $\alpha_l$ . When initialized at a small value, this choice does help the convergence when we increase the depth.

**Our proposal LayerScale** is a per-channel multiplication of the vector produced by each residual block, as opposed to a single scalar, see Figure 3.4(d). Our objective is to group the updates of the weights associated with the same output channel. Formally, LayerScale is a multiplication by a diagonal matrix on output of each residual block. In other terms, we modify Eqn. 3.5 as

$$\begin{aligned} x'_l &= x_l + \text{diag}(\lambda_{l,1}, \dots, \lambda_{l,d}) \times \text{SA}(\eta(x_l)) \\ x_{l+1} &= x'_l + \text{diag}(\lambda'_{l,1}, \dots, \lambda'_{l,d}) \times \text{FFN}(\eta(x'_l)), \end{aligned} \quad (3.7)$$

where the parameters  $\lambda_{l,i}$  and  $\lambda'_{l,i}$  are learnable weights. The diagonal values are all initialized to a fixed small value  $\varepsilon$ : we set it to  $\varepsilon = 0.1$  until depth 18,  $\varepsilon = 10^{-5}$  for depth 24 and  $\varepsilon = 10^{-6}$  for deeper networks. This formula is akin to other normalization strategies ActNorm [118] or LayerNorm but executed on output of the residual block. Yet we seek a different effect: ActNorm is a data-dependent initialization that calibrates activations so that they have zero-mean and unit variance, like batchnorm [113]. In contrast, we initialize the diagonal with small values so that the initial contribution of the residual branches to the function implemented by the transformer is small. In that respect our motivation is therefore closer to that of ReZero [9], SkipInit [53], Fixup [243] and T-Fixup [109]: to train closer to the identity function and let the network integrate the additional parameters progressively during the training. LayerScale offers more diversity in the optimization than just adjusting the whole layer by a single learnable scalar as in ReZero/SkipInit, Fixup and T-Fixup. As we will show empirically, offering the degrees of freedom to do so per channel is a decisive advantage of LayerScale over existing approaches. In Section 3.5.3, we present other variants or intermediate choices that support our proposal, and a control experiment that aims at disentangling the specific weighting of the branches of LayerScale from its impact on optimization procedure.

Formally, adding these weights does not change the expressive power of the architecture since they can be integrated into the previous matrix of the SA and FFN layers without changing the function implemented by the network.

### 3.5.2 LayerScale main experiments

**Experimental setting.** Our implementation is based on the timm library [214]. Unless specified otherwise, for this analysis we make minimal changes to hyper-parameters compared to our initial DeiT training scheme. In order to speed up training and optimize memory consumption we have used a sharded training provided by the Fairscale library<sup>3</sup> with fp16 precision.

In the following, we analyse various ways to stabilize the training with different architectures. At this stage we consider a Deit-Small model<sup>4</sup> during 300 epochs to allow a direct comparison with our preliminary results with DeiT training. We measure the performance on the ImageNet-1k [54, 168] classification dataset as a function of the depth.

**Adjusting the drop-rate of stochastic depth.** The first step to improve convergence is to adapt the hyper-parameters that interact the most with depth, in particular Stochastic depth [108]. This method is already popular in NLP [69, 70] to train deeper architectures. The per-layer drop-rate depends linearly on the layer depth, but in our experiments this choice does not provide an advantage compared to the simpler choice of a uniform drop-rate  $d_r$ . In Table 3.11 we show that

3. <https://pypi.org/project/fairscale/>

4. <https://github.com/facebookresearch/deit>

Table 3.11 – **Improving convergence at depth** on ImageNet-1k. The baseline is DeiT-S with uniform drop rate of  $d = 0.05$  (same expected drop rate and performance as progressive stochastic depth of 0.1). Several methods include a fix scalar learnable weight  $\alpha$  per layer as in Figure 3.4(c). We have adapted Rezero, Fixup, T-Fixup, since the original methods do not converge: we have re-introduced the Layer-normalization  $\eta$  and warmup. We have adapted the drop rate  $d_r$  for all the methods, including the baseline. The column  $\alpha = \varepsilon$  reports the performance when initializing the scalar with the same value as for LayerScale. †: *failed before the end of the training*.

depth	baseline		scalar $\alpha$ weighting				LayerScale
	$d_r = 0.05$	adjust [ $d_r$ ]	Rezero	T-Fixup	Fixup	$\alpha = \varepsilon$	
12	79.9	79.9 [0.05]	78.3	79.4	80.7	80.4	80.5
18	80.1	80.7 [0.10]	80.1	81.7	82.0	81.6	81.7
24	78.9†	81.0 [0.20]	80.8	81.5	82.3	81.1	82.4
36	78.9†	81.9 [0.25]	81.6	82.1	82.4	81.6	82.9

the default stochastic depth of DeiT allows us to train up to 18 blocks of SA+FFN. After that the training becomes unstable. By increasing the drop-rate hyper-parameter  $d_r$ , the performance increases until 24 layers. It saturates at 36 layers (we measured that it drops to 80.7% at 48 layers).

**Comparison of normalization strategies.** We carry out an empirical study of the normalization methods discussed in Section 3.5.1. As previously indicated, Rezero, Fixup and T-Fixup do not converge when training DeiT off-the-shelf. However, if we re-introduce LayerNorm<sup>5</sup> and warmup, Fixup and T-Fixup achieve convergence and even improve training compared to the baseline DeiT. We report the results for these “adaptations” of Fixup and T-Fixup in Table 3.11.

The modified methods are able to converge with more layers without saturating too early. ReZero converges, we show (column  $\alpha = \varepsilon$ ) that it is better to initialize  $\alpha$  to a small value instead of 0, as in LayerScale. All the methods have a beneficial effect on convergence and they tend to reduce the need for stochastic depth, therefore we adjust these drop rate accordingly per method. Figure 3.5 provides the performance as the function of the drop rate  $d_r$  for LayerScale. We empirically use the following formula to set up the drop-rate for the CaiT-S models derived from DeiT-S:  $d_r = \min(0.1 \times \frac{\text{depth}}{12} - 1, 0)$ . This formulaic choice avoids cross-validating this parameter and overfitting, yet it does not generalize to models with different  $d$ : We further increase (resp. decrease) it by a constant for larger (resp. smaller) working dimensionality  $d$ .

Fixup and T-Fixup are competitive with LayerScale in the regime of a relatively low number of blocks (12–18). However, they are more complex than LayerScale: they employ different initialization rules depending of the type of layers, and they require more changes to the transformer architecture. Therefore we only use LayerScale in subsequent experiments. It is much simpler and parametrized by a single hyper-parameter  $\varepsilon$ , and it offers a better performance for the deepest models that we consider, which are also the more accurate.

### 3.5.3 Layerscale: Analysis and variations.

**Statistics of branch weighting.** We evaluate the impact of Layerscale for a 36-blocks transformer by measuring the ratio between the norm of the residual activations and the norm of the activations of the main branch  $\|g_l(x)\|_2 / \|x\|_2$ . The results are shown in Figure 3.6. We can see that training a model with Layerscale makes this ratio more uniform across layers, and seems to prevent some layers from having a disproportionate impact on the activations. Similar to prior works [9, 243] we

5. Bachlechner *et al.* report that batchnorm is complementary to ReZero, while removing LayerNorm in the case of transformers.

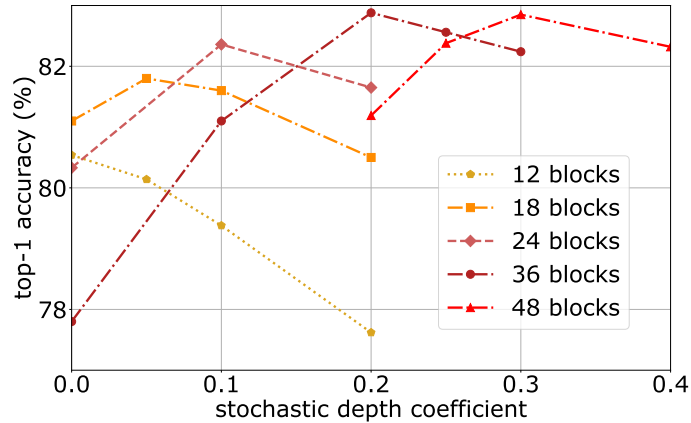


Figure 3.5 – We measure the impact of stochastic depth on ImageNet with a DeiT-S with LayerScale for different depths. The drop rate of stochastic depth needs to be adapted to the network depth.

Table 3.12 – Performance when increasing the depth. We compare different strategies and report the top-1 accuracy (%) on ImageNet-1k for the DeiT training (Baseline) with and without adapting the stochastic depth rate  $d_r$  (uniform drop-rate), and a modified version of Rezero with LayerNorm and warmup. We compare different initialisation of the diagonal matrix for LayerScale. We also report results with 0 initialization, Uniform initialisation and small constant initialisation. Except for the baseline  $d_r = 0.1$ , we have adapted the stochastic depth rate  $d_r$ .

depth	baseline	baseline	ReZero	LayerScale [ $\varepsilon$ ]		
	$d_r = 0.1$	$[d_r]$	$\alpha = 0$	$\lambda_i = 0$	$\lambda_i = \mathcal{U}[0, 2\varepsilon]$	$\lambda_i = \varepsilon$
12	79.9	79.9 [0.05]	78.3	79.7	80.2 [0.1]	80.5 [0.1]
18	80.1	80.7 [0.10]	80.1	81.5	80.8 [0.1]	81.7 [0.1]
24	78.9 $\uparrow$	81.0 [0.20]	80.8	82.1	82.1 [ $10^{-5}$ ]	82.4 [ $10^{-5}$ ]
36	78.9 $\uparrow$	81.9 [0.25]	81.6	82.7	82.6 [ $10^{-6}$ ]	82.9 [ $10^{-6}$ ]

hypothesize that the benefit is mostly the impact on optimization. This hypothesis is supported by the control experiment that we detail in section 3.5.3.

**Variations on LayerScale init.** For the sake of simplicity and to avoid overfitting per model, we have chosen to do a constant initialization with small values depending on the model depth. In order to give additional insight on the importance of this initialization we compare in Table 3.12 other possible choices.

**LayerScale with 0 init.** We initialize all coefficients of LayerScale to 0. This resembles Rezero, but in this case we have distinct learnable parameters for each channel. We make two observations. First, this choice, which also starts with residual branches that output 0 the beginning of the training, gives a clear boost compared to the block-wise scaling done by our adapted ReZero. This confirms the advantage of introducing a learnable parameter per channel and not only per residual layer. Second, LayerScale is better: it is best to initialize to a small  $\varepsilon$  different from zero.

**Random init.** We have tested a version in which we try a different initial weight per channel, but with the same average contribution of each residual block as in LayerScale. For this purpose we initialize the channel-scaling values with the Uniform law ( $\mathcal{U}[0, 2\varepsilon]$ ). This simple choice ensures that the expectation of the scaling factor is equal to the value of the classical initialization of LayerScale. This choice is overall comparable to the initialization to 0 of the diagonal, and inferior to LayerScale.

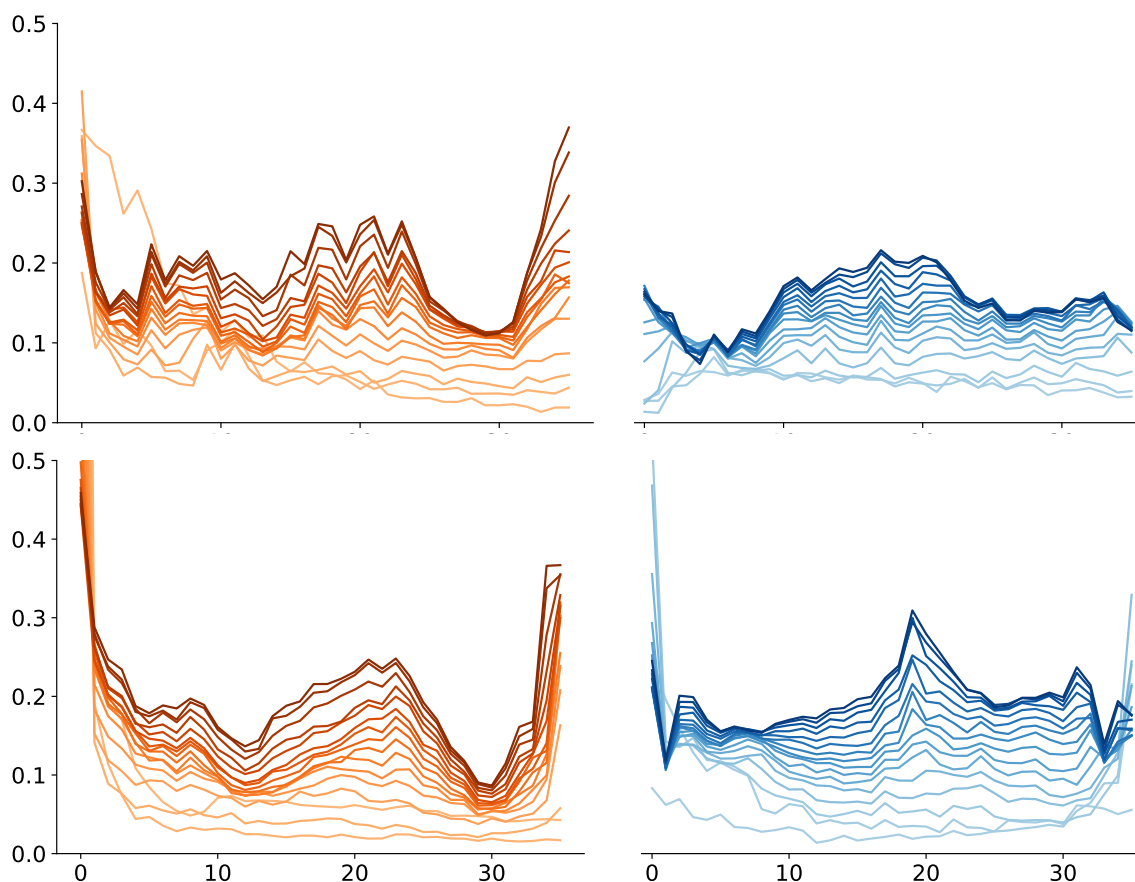


Figure 3.6 – Analysis of the contribution of the residual branches (*Top*: Self-attention ; *Bottom*: FFN) for a network comprising 36 layers, *without* (red) or *with* (blue) Layerscale. The ratio between the norm of the residual and the norm of the main branch is shown for each layer of the transformer and for various epochs (darker shades correspond to the last epochs). For the model trained with layerscale, the norm of the residual branch is on average 20% of the norm of the main branch. We observe that the contribution of the residual blocks fluctuates more for the model trained without layerscale and in particular is lower for some of the deeper layers.

**Re-training.** LayerScale makes it possible to get increased performance by training deeper models. At the end of training we obtain a specific set of scaling factors for each layer. Inspired by the lottery ticket hypothesis [75], one question that arises is whether what matters is to have the right scaling factors, or to include these learnable weights in the optimization procedure. In other terms, what happens if we re-train the model with the scaling factors obtained by a previous training?

In this experiment below, we try to empirically answer that question. We compare the performance (top-1 validation accuracy, %) on ImageNet-1k with DeiT-S architectures of different depths. Everything being identical otherwise, in the first experiment we use LayerScale, i.e. we have learnable weights initialized at a small value  $\varepsilon$ . In the control experiment we use fixed scaling factors initialised at values obtained by the LayerScale training.

	Depth $\rightarrow$	12	18	24	36
LayerScale		80.5	81.7	82.4	82.9
Re-trained with fixed weights		80.6	81.5	81.2	81.6

We can see that the control training with fixed weights also converges, but it is only slightly better than the baseline with adjusted stochastic depth drop-rate  $d_r$ . Nevertheless, the results are lower than those obtained with the learnable weighting factors. This suggests that the evolution of the parameters during training has a beneficial effect on the deepest models.

## 3.6 Specializing layers for class attention

In this section, we introduce the CaiT architecture, depicted in Figure 3.7 (right). This design aims at circumventing one of the problems of the ViT architecture: the learned weights are asked to optimize two contradictory objectives: (1) guiding the self-attention between patches while (2) summarizing the information useful to the linear classifier. Our proposal is to explicitly separate the two stages, in the spirit of an encoder-decoder architecture, see Section 3.1.

**Later class token.** As an intermediate step towards our proposal, we insert the so-called class token, denoted by CLS, later in the transformer. This choice eliminates the discrepancy on the first layers of the transformer, which are therefore fully employed for performing self-attention between patches only. As a baseline that does not suffer from the contradictory objectives, we also consider **average pooling** of all the patches on output of the transformers, as typically employed in convolutional architectures.

### 3.6.1 CaiT: A simple encoder-decoder for image classification

Our CaiT network consists of two distinct processing stages visible in Figure 3.7:

1. The *self-attention* stage is identical to the one used in Vision transformer, but with no class embedding (CLS).
2. The *class-attention* stage is a set of layers that compiles the set of patch embeddings into a class embedding CLS that is subsequently fed to a linear classifier.

This class-attention alternates in turn a layer that we refer to as a multi-head class-attention (CA), and a FFN layer. In this stage, only the class embedding is updated. Similar to the one fed in ViT and DeiT on input of the transformer, it is a learnable vector. The main difference is that, in our architecture, we do not copy information from the class embedding to the patch embeddings during the forward pass. Only the class embedding is updated by residual in the CA and FFN processing of the class-attention stage.

**Multi-head class attention.** The role of the CA layer is to extract the information from the set of processed patches. It is identical to a SA layer, except that it relies on the attention between (i) the class embedding  $x_{\text{class}}$  (initialized at CLS in the first CA) and (ii) itself plus the set of frozen patch embeddings  $x_{\text{patches}}$ . We discuss why we include  $x_{\text{class}}$  in the keys in Section 3.6.3.

Considering a network with  $h$  heads and  $p$  patches, and denoting by  $d$  the embedding size, we parametrize the multi-head class-attention with several projection matrices,  $W_q, W_k, W_v, W_o \in \mathbf{R}^{d \times d}$ , and the corresponding biases  $b_q, b_k, b_v, b_o \in \mathbf{R}^d$ . With this notation, the computation of the CA residual block proceeds as follows. We first augment the patch embeddings (in matrix

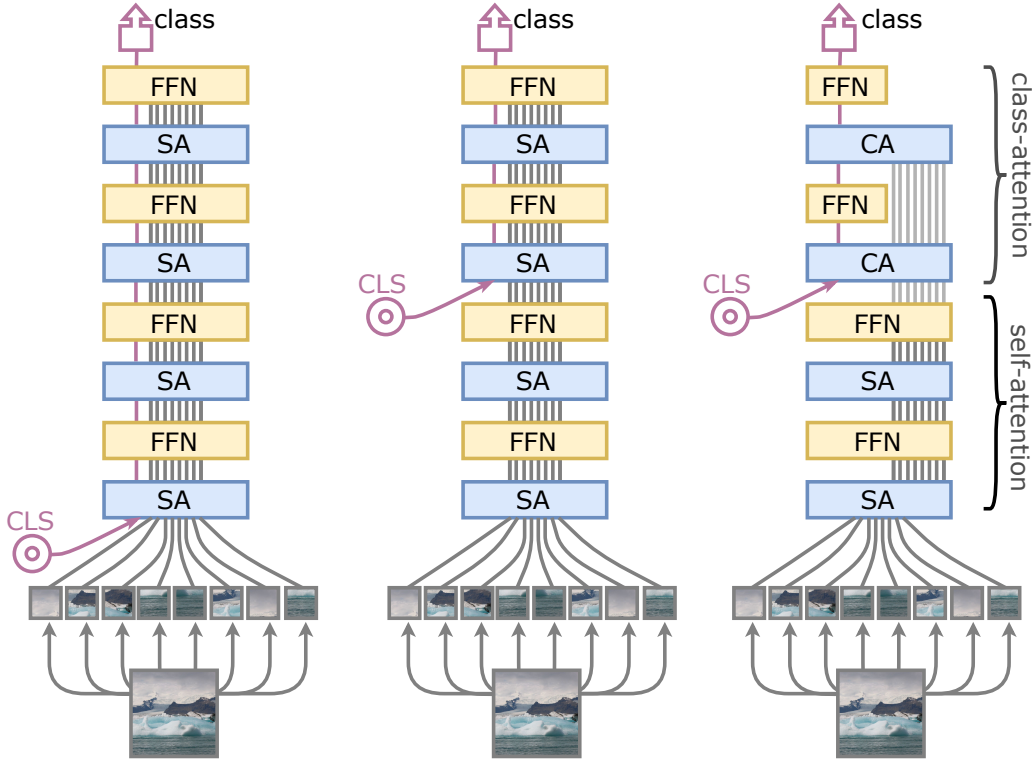


Figure 3.7 – In the ViT transformer (*left*), the class embedding (CLS) is inserted along with the patch embeddings. This choice is detrimental, as the same weights are used for two different purposes: helping the attention process, and preparing the vector to be fed to the classifier. We put this problem in evidence by showing that inserting CLS later improves performance (*middle*). In the CaiT architecture (*right*), we further propose to freeze the patch embeddings when inserting CLS to save compute, so that the last part of the network (typically 2 layers) is fully devoted to summarizing the information to be fed to the linear classifier.

form) as  $z = [x_{\text{class}}, x_{\text{patches}}]$  (see Section 3.6.3 for results when  $z = x_{\text{patches}}$ ). We then perform the projections as follows:

$$Q = W_q x_{\text{class}} + b_q, \quad (3.8)$$

$$K = W_k z + b_k, \quad (3.9)$$

$$V = W_v z + b_v. \quad (3.10)$$

The class-attention weights are given by

$$\text{Attention}(Q, K) = \text{Softmax}(Q.K^T / \sqrt{d/h}) \quad (3.11)$$

where  $Q.K^T \in \mathbf{R}^{h \times 1 \times p}$ . This attention is involved in the weighted sum  $\text{Attention}(Q, K) \times V$  to produce the residual output vector

$$\text{out}_{\text{CA}} = W_o \text{Attention}(Q, K) V + b_o, \quad (3.12)$$

which is in turn added to  $x_{\text{class}}$  for subsequent processing.

The CA layers extract the useful information from the patches embedding to the class embedding. In preliminary experiments, we empirically observed that the first CA and FFN give the main boost, and a set of 2 blocks of layers (2 CA and 2 FFN) is sufficient to cap the performance.



Table 3.13 – Variations on CLS with DeiT-Small (no LayerScale): we change the layer at which the class embedding is inserted. In ViT and DeiT, it is inserted at layer 0 jointly with the projected patches. We evaluate a late insertion of the CLS, as well as our design choice to introduce specific class-attention layers.

depth: SA+CA	insertion layer	top-1 acc.	#params	FLOPs
Baselines: DeiT-S and average pooling				
12: 12 + 0	0	79.9	22M	4.6B
12: 12 + 0	n/a	80.3	22M	4.6B
Late insertion of class embedding				
12: 12 + 0	2	80.0	22M	4.6B
12: 12 + 0	4	80.0	22M	4.6B
12: 12 + 0	8	80.0	22M	4.6B
12: 12 + 0	10	80.5	22M	4.6B
12: 12 + 0	11	80.3	22M	4.6B
DeiT-S with class-attention stage (SA+FFN)				
12: 9 + 3	9	79.6	22M	3.6B
12: 10 + 2	10	80.3	22M	4.0B
12: 11 + 1	11	80.6	22M	4.3B
13: 12 + 1	12	80.8	24M	4.7B
14: 12 + 2	12	80.8	26M	4.7B
15: 12 + 3	12	80.6	27M	4.8B

In the experimental section, we denote by 12+2 a transformer when it consists of 12 blocks of SA+FFN layers and 2 blocks of CA+FFN layers.

**Complexity.** The layers contain the same number of parameters in the class-attention and self-attention stages: CA is identical to SA in that respect, and we use the same parametrization for the FFNs. However the processing of these layers is much faster: the FFN only layers processes matrix-vector multiplications.

The CA function is also less expensive than SA in term of memory and computation because it computes the attention between the class vector and the set of patch embeddings:  $Q \in \mathbf{R}^d$  means that  $Q.K^T \in \mathbf{R}^{h \times 1 \times p}$ . In contrast, in the “regular self-attention” layers SA, we have  $Q \in \mathbf{R}^{p \times d}$  and therefore  $Q.K^T \in \mathbf{R}^{h \times p \times p}$ . In other words, the initially quadratic complexity in the number of patches becomes linear in our extra CaiT layers.

### 3.6.2 Preliminary analysis with late insertion and class-attention layers

In Table 3.13 we study the impact on performance of the design choices related to class embedding. We depict some of them in Figure 3.7. As a baseline, average pooling of patches embeddings with a vanilla DeiT-Small achieves a better performance than using a class token. This choice, which does not employ any class embedding, is typical in convolutional networks, but possibly weaker with transformers when transferring to other tasks [65].

**Late insertion.** The performance increases when we insert the class embedding later in the transformer. It is maximized two layers before the output. Our interpretation is that the attention process is less perturbed in the 10 first layers, yet it is best to keep 2 layers for compiling the patches embedding into the class embedding via class-attention, otherwise the processing gets closer to a weighted average.

Table 3.14 – CaiT models with and without distillation token. All these models are trained with the same setting during 400 epochs.

Model	Distillation token	
	✗	✓
XXS-24 $\Upsilon$	78.4	78.5
M-24 $\Upsilon$	84.8	84.7

**Our class-attention layers** are designed on the assumption that there is no benefit in copying information from the class embedding back to the patch embeddings in the forward pass. Table 3.13 supports that hypothesis: if we compare the performance for a total number of layers fixed to 12, the performance of CaiT with 10 SA and 2 CA layers is identical to average pooling and better than the DeiT-Small baseline with a lower number of FLOPs. If we set 12 layers in the self-attention stage, which dominates the complexity, we increase the performance significantly by adding two blocks of CA+FFN.

### 3.6.3 Ablation: Design of the class-attention stage

In this subsection we report some results obtained when considering alternative choices for the class-attention stage.

**Not including class embedding in keys of class-attention.** In our approach we chose to insert the class embedding in the class-attention: By defining

$$z = [x_{\text{class}}, x_{\text{patches}}], \quad (3.13)$$

we include  $x_{\text{class}}$  in the keys and therefore the class-attention includes attention on the class embedding itself in Eqn. 3.9 and Eqn. 3.10. This is not a requirement as we could simply use a pure cross-attention between the class embedding and the set of frozen patches.

If we do not include the class token in the keys of the class-attention layers, i.e., if we define  $z = x_{\text{patches}}$ , we reach 83.31% (top-1 acc. on ImageNet1k-val) with CaiT-S-36, versus 83.44% for the choice adopted by default in the chapter. This difference of +0.13% is likely not significant, therefore either choice is reasonable. In order to be more consistent with the self-attention layer SA, in the sense that each query has its key counterpart, we have kept the class embedding in the keys of the CA layers as stated in this chapter.

**Remove LayerScale in Class-Attention.** If we remove LayerScale in the Class-Attention blocks in the CaiT-S-36 model, we obtain a top-1 accuracy of 83.36% on ImageNet1k-val, versus 83.44% with LayerScale. The difference of +0.08% is not significant enough to conclude on a clear advantage. For the sake of consistency we have used LayerScale after all residual blocks of the network.

**Distillation with class-attention.** We report results with hard distillation, which in essence replaces the label by the average of the label and the prediction of the teacher output. This is the choice we adopted in our main experiments, since it provides better performance than traditional distillation as shown earlier in this chapter.

In Table 3.14 we report the results obtained when inserting a distillation token at the same layer as the class token, i.e., on input of the class-attention stage. In this case we do not observe an advantage of this choice over hard distillation when using class-attention layers. Therefore we have only considered the hard distillation.

Table 3.15 – CaiT models: The design parameters are depth and  $d$ . The mem columns correspond to the memory usage. All models are initially trained at resolution 224 during 400 epochs. We also fine-tune these models at resolution 384 (identified by  $\uparrow 384$ ) or train them with distillation ( $\Upsilon$ ). The FLOPs are reported for each resolution.

CAiT model	depth (SA+CA)	$d$	#params ( $\times 10^6$ )	FLOPs ( $\times 10^9$ )		Top-1 acc. (%): Imagenet1k-val			
				@224	@384	@224	$\uparrow 384$	@224 $\Upsilon$	$\uparrow 384\Upsilon$
XXS-24	24+2	192	12.0	2.5	9.5	77.6	80.4	78.4	80.9
XXS-36	36+2	192	17.3	3.8	14.2	79.1	81.8	79.7	82.2
XS-24	24+2	288	26.6	5.4	19.3	81.8	83.8	82.0	84.1
XS-36	36+2	288	38.6	8.1	28.8	82.6	84.3	82.9	84.8
S-24	24+2	384	46.9	9.4	32.2	82.7	84.3	83.5	85.1
S-36	36+2	384	68.2	13.9	48.0	83.3	85.0	84.0	85.4
S-48	48+2	384	89.5	18.6	63.8	83.5	85.1	83.9	85.3
M-24	24+2	768	185.9	36.0	116.1	83.4	84.5	84.7	85.8
M-36	36+2	768	270.9	53.7	173.3	83.8	84.9	85.1	86.1

Table 3.16 – Hyper-parameters for training CaiT models: The only parameters that we adjust per model are the drop rate  $d_r$  of stochastic depth and the LayerScale initialization  $\varepsilon$ .

CAiT model	XXS-24	XXS-36	XS-24	XS-36	S-24	S-36	S-48	M-24	M-36	M-48
hparams $d_r$	0.05	0.1	0.05	0.1	0.1	0.2	0.3	0.2	0.3	0.4
$\varepsilon$	$10^{-5}$	$10^{-6}$	$10^{-5}$	$10^{-6}$	$10^{-5}$	$10^{-6}$	$10^{-6}$	$10^{-5}$	$10^{-6}$	$10^{-6}$

### 3.7 Our CaiT models

In this section, we report our experimental results related to CaiT with our DeiT training. We present our models in Subsection 3.7. Section 3.7.1 details our results on ImageNet and Transfer learning. We provide an ablation of hyper-parameter and ingredients in Section 3.7.2.

Our CaiT models are built upon ViT: the only difference is that we incorporate LayerScale in each residual block (see Section 3.5.1) and the two-stages architecture with class-attention layers described in Section 3.6. Table 3.15 describes our different models. The design parameters governing the capacity are the depth and the working dimensionality  $d$ . In our case  $d$  is related to the number of heads  $h$  as  $d = 48 \times h$ , since we fix the **number of components per head** to 48. This choice is a bit smaller than the value used in DeiT. We also adopt the **crop-ratio** of 1.0 optimized for DeiT by Wightman [214]. Table 3.21 and 3.22 in the ablation section 3.7.2 support these choices.

We incorporate talking-heads attention [173] into our model. It increases the performance on ImageNet of DeiT-Small from 79.9% to 80.3%.

**The hyper-parameters** are identical to those provided in DeiT [193], except mentioned otherwise. We use a batch size of 1024 samples and train during 400 epochs with repeated augmentation [15, 101]. The learning rate of the AdamW optimizer [136] is set to 0.001 and associated with a cosine training schedule, 5 epochs of warmup and a weight decay of 0.05. We report in Table 3.16 the two hyper-parameters that we modify depending on the model complexity, namely the drop rate  $d_r$  associated with uniform stochastic depth, and the initialization value  $\varepsilon$  of LayerScale.

**Fine-tuning at higher resolution ( $\uparrow$ ) and distillation ( $\Upsilon$ ).** We train all our models at resolution 224, and optionally fine-tune them at a higher resolution to trade performance against accuracy [61, 193, 199]: we denote the model by  $\uparrow 384$  models fine-tuned at resolution  $384 \times 384$ . We also train

models with distillation ( $\Upsilon$ ) as suggested by Touvron *et al.* [193]. We use a RegNet-16GF [156] as teacher and adopt the “hard distillation” [193] for its simplicity.

### 3.7.1 Results

#### 3.7.1.1 Performance/complexity of CaiT models

Table 3.15 provides different complexity measures for our models. As a general observation, we observe a subtle interplay between the width and the depth, both contribute to the performance as reported by Dosovitskiy *et al.* [61] with longer training schedules. But if one parameter is too small the gain brought by increasing the other is not worth the additional complexity.

Fine-tuning to size 384 ( $\uparrow$ ) systematically offers a large boost in performance without changing the number of parameters. It also comes with a higher computational cost. In contrast, leveraging a pre-trained convnet teacher with hard distillation as suggested by Touvron *et al.* [193] provides a boost in accuracy without affecting the number of parameters nor the speed.

#### 3.7.1.2 Comparison with the state of the art on ImageNet

Our main classification experiments are carried out on ImageNet [168], and also evaluated on two variations of this dataset: ImageNet-Real [18] that corrects and give a more detailed annotation, and ImageNet-V2 [159] (matched frequency) that provides a separate test set. In Table 3.17 we compare some of our models with the state of the art on ImageNet classification when training without external data. We focus on the models CaiT-S36 and CaiT-M36, at different resolutions and with or without distillation.

On ImageNet-1k val, CaiT-M48 $\uparrow$ 448 $\Upsilon$  achieves 86.5% of top-1 accuracy, which is a significant improvement over DeiT (85.2%). It is the state of the art, on par with a recent concurrent work [25] that has a significantly higher number of FLOPs. Our approach outperforms the state of the art on ImageNet with reassessed labels, and on ImageNet-V2, which has a distinct validation set which makes it harder to overfit.

#### 3.7.1.3 Transfer learning

We evaluated our method on transfer learning tasks by fine-tuning on the datasets in Table 3.18.

**Fine-tuning procedure.** For fine-tuning we use the same hyperparameters as for training. We only decrease the learning rates by a factor 10 (for CARS, Flowers, iNaturalist), 100 (for CIFAR-100, CIFAR-10) and adapt the number of epochs (1000 for CIFAR-100, CIFAR-10, Flowers-102 and Cars-196, 360 for iNaturalist 2018 and 2019). We have not used distillation for this finetuning.

**Results.** Table 3.19 compares CaiT transfer learning results to those of EfficientNet [187], ViT [61] and DeiT [193]. These results show the excellent generalization of the transformers-based models in general. Our CaiT models achieve excellent results, as shown by the overall better performance than EfficientNet-B7 across datasets.

### 3.7.2 Ablation and visualization

In this section we provide different sets of ablation, in the form of a transition from DeiT to CaiT. Then we provide experiments that have guided our hyper-parameter optimization. As

Table 3.17 – **Complexity vs accuracy** on ImageNet [168], ImageNet Real [18] and ImageNet V2 matched frequency [159] for models trained without external data. We compare CaiT with DeiT [193], ViT-B [61], TNT [89], T2T [235] and to several state-of-the-art convnets: Regnet [156] improved by Touvron et al. [193], EfficientNet [44, 187, 226], Fix-EfficientNet [200] and NFNet [25]. Most reported results are from corresponding papers, and therefore the training procedure differs for the different models. For ImageNet V2 matched frequency and ImageNet Real we report the results provided by the authors. When not available (like NFNet), we report the results measured by Wightman [214] with converted models, which may be suboptimal. The RegNetY-16GF is the teacher model that we trained for distillation. We report the best result in **bold** and the second best result(s) underlined.

Network	nb of param.	nb of FLOPs	image size		ImNet top-1	Real top-1	V2 top-1
			train	test			
RegNetY-16GF	84M	16.0B	224	224	82.9	88.1	72.4
EfficientNet-B5	30M	9.9B	456	456	83.6	88.3	73.6
EfficientNet-B7	66M	37.0B	600	600	84.3	–	–
EfficientNet-B5 RA	30M	9.9B	456	456	83.7	–	–
EfficientNet-B7 RA	66M	37.0B	600	600	84.7	–	–
EfficientNet-B7 AdvProp	66M	37.0B	600	600	85.2	89.4	76.0
Fix-EfficientNet-B8	87M	89.5B	672	800	85.7	<u>90.0</u>	75.9
NFNet-F0	72M	12.4B	192	256	83.6	88.1	72.6
NFNet-F1	133M	35.5B	224	320	84.7	88.9	74.4
NFNet-F2	194M	62.6B	256	352	85.1	88.9	74.3
NFNet-F3	255M	114.8B	320	416	85.7	89.4	75.2
NFNet-F4	316M	215.3B	384	512	85.9	89.4	75.2
NFNet-F5	377M	289.8B	416	544	86.0	89.2	74.6
NFNet-F6+SAM	438M	377.3B	448	576	<b>86.5</b>	89.9	75.8
Transformers							
ViT-B/16	86M	55.4B	224	384	77.9	83.6	–
ViT-L/16	307M	190.7B	224	384	76.5	82.2	–
T2T-ViT t-14	21M	5.2B	224	224	80.7	–	–
TNT-S	24M	5.2B	224	224	81.3	–	–
TNT-S + SE	25M	5.2B	224	224	81.6	–	–
TNT-B	66M	14.1B	224	224	82.8	–	–
DeiT-S	22M	4.6B	224	224	79.8	85.7	68.5
DeiT-B	86M	17.5B	224	224	81.8	86.7	71.5
DeiT-B $\uparrow$ 384	86M	55.4B	224	384	83.1	87.7	72.4
DeiT-B $\uparrow$ 384 $\Upsilon$ 1000 epochs	87M	55.5B	224	384	85.2	89.3	75.2
Our deep transformers							
CaiT-S36	68M	13.9B	224	224	83.3	88.0	72.5
CaiT-S36 $\uparrow$ 384	68M	48.0B	224	384	85.0	89.2	75.0
CaiT-S48 $\uparrow$ 384	89M	63.8B	224	384	85.1	89.5	75.5
CaiT-S36 $\Upsilon$	68M	13.9B	224	224	84.0	88.9	74.1
CaiT-S36 $\uparrow$ 384 $\Upsilon$	68M	48.0B	224	384	85.4	89.8	76.2
CaiT-M36 $\uparrow$ 384 $\Upsilon$	271M	173.3B	224	384	86.1	<u>90.0</u>	76.3
CaiT-M36 $\uparrow$ 448 $\Upsilon$	271M	247.8B	224	448	<u>86.3</u>	<b>90.2</b>	<u>76.7</u>
CaiT-M48 $\uparrow$ 448 $\Upsilon$	356M	329.6B	224	448	<b>86.5</b>	<b>90.2</b>	<b>76.9</b>

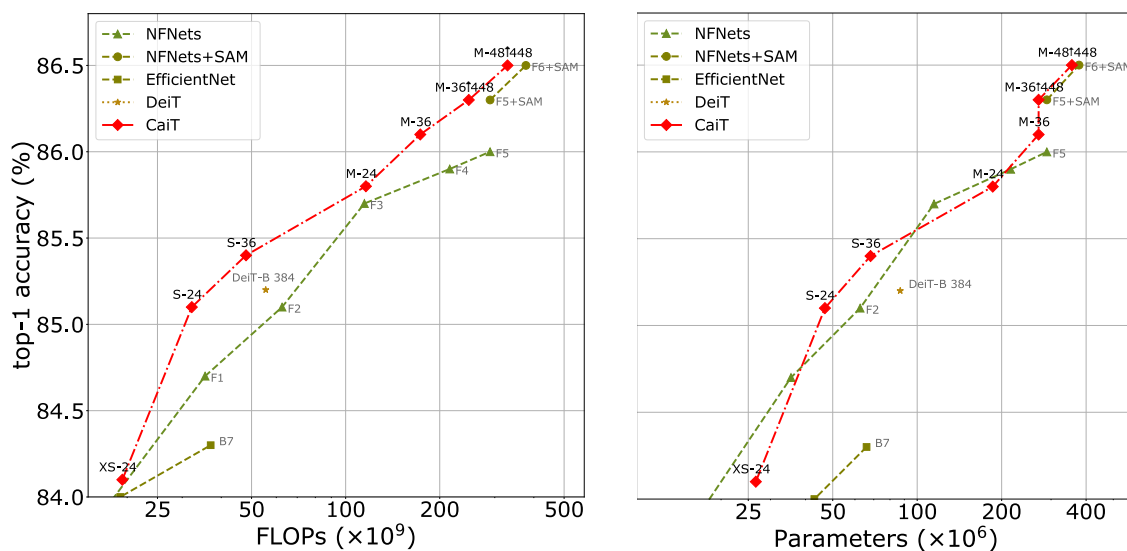


Figure 3.8 – We represent FLOPs and parameters for our best CaiT  $\uparrow 384$  and  $\uparrow 448$  models trained with distillation. They are competitive on ImageNet-1k-val with the sota in the high accuracy regime, from XS-24 to M-48. Convolution-based neural networks like NFNets and EfficientNet are better in low-FLOPs and low-parameters regimes.

Table 3.18 – Datasets used for our different tasks.

Dataset	Train size	Test size	#classes
ImageNet [168]	1,281,167	50,000	1000
iNaturalist 2018 [104]	437,513	24,426	8,142
iNaturalist 2019 [103]	265,240	3,003	1,010
Flowers-102 [144]	2,040	6,149	102
Stanford Cars [121]	8,144	8,041	196
CIFAR-100 [124]	50,000	10,000	100
CIFAR-10 [124]	50,000	10,000	10

mentioned, we use the same hyperparameters as in DeiT [193] everywhere except stated otherwise. We have only changed the number of attention for a given working dimension (see Section 3.7.2.2), and changed the crop-ratio (see Section 3.7.2.3).

### 3.7.2.1 Step by step from DeiT-Small to CaiT-S36

In Table 3.20 we present how to gradually transform the DeiT-S [193] architecture into CaiT-36, and measure at each step the performance/complexity changes. One can see that CaiT is complementary with LayerScale and offers an improvement without significantly increasing the FLOPs. As already reported in the literature, the resolution is another important step for improving the performance and fine-tuning instead of training the model from scratch saves a lot of computation at training time. Last but not least, our models benefit from longer training schedules.

### 3.7.2.2 Optimization of the number of heads

In Table 3.21 we study the impact of the number of heads for a fixed working dimensionality. This architectural parameter has an impact on both the accuracy, and the efficiency: while the number of FLOPs remain roughly the same, the compute is more fragmented when increasing this

Table 3.19 – Results in transfer learning. All models are trained and evaluated at resolution 224 and with a crop-ratio of 0.875 in this comparison (see Table 3.22 for the comparison of crop-ratio on ImageNet).

Model	ImageNet	CIFAR-10	CIFAR-100	Flowers	Cars	iNat-18	iNat-19	FLOPs
EfficientNet-B7	84.3	98.9	91.7	98.8	<b>94.7</b>	–	–	37.0B
ViT-B/16	77.9	98.1	87.1	89.5	–	–	–	55.5B
ViT-L/16	76.5	97.9	86.4	89.7	–	–	–	190.7B
DeiT-B 224	81.8	99.1	90.8	98.4	92.1	73.2	77.7	17.5B
CaiT-S-36 224	83.4	99.2	92.2	98.8	93.5	77.1	80.6	13.9B
CaiT-M-36 224	83.7	99.3	93.3	99.0	93.5	76.9	81.7	53.7B
CaiT-S-36 $\Upsilon$ 224	83.7	99.2	92.2	99.0	94.1	77.0	81.4	13.9B
CaiT-M-36 $\Upsilon$ 224	<b>84.8</b>	<b>99.4</b>	<b>93.1</b>	<b>99.1</b>	94.2	<b>78.0</b>	<b>81.8</b>	53.7B

Table 3.20 – Ablation: we present the ablation path from DeiT-S to our CaiT models. We highlight the complementarity of our approaches and optimized hyper-parameters. Note, Fine-tuning at higher resolution supersedes the inference at higher resolution. See Table 3.11 for adapting stochastic depth before adding LayerScale. †: training failed.

Improvement	top-1 acc.	#params	FLOPs
DeiT-S [ $d=384,300$ epochs]	79.9	22M	4.6B
+ More heads [8]	80.0	22M	4.6B
+ Talking-heads	80.5	22M	4.6B
+ Depth [36 blocks]	69.9†	64M	13.8B
+ <b>Layer-scale</b> [ $init \ \varepsilon = 10^{-6}$ ]	80.5	64M	13.8B
+ Stch depth. adaptation [ $d_r=0.2$ ]	83.0	64M	13.8B
+ <b>CaiT architecture</b> [ <i>specialized class-attention layers</i> ]	83.2	68M	13.9B
+ Longer training [400 epochs]	83.4	68M	13.9B
+ Inference at higher resolution [256]	83.8	68M	18.6B
+ Fine-tuning at higher resolution [384]	84.8	68M	48.0B
+ Hard distillation [ <i>teacher: RegNetY-16GF</i> ]	85.2	68M	48.0B
+ Adjust crop ratio [ $0.875 \rightarrow 1.0$ ]	85.4	68M	48.0B

Table 3.21 – Deit-Small: for a fixed 384 working dimensionality and number of parameters, impact of the number of heads on the accuracy and throughput (images processed per second at inference time on a single V100 GPU).

# heads	dim/head	throughput (im/s)	GFLOPs	top-1 acc.
1	384	1079	4.6	76.80
2	192	1056	4.6	78.06
3	128	1043	4.6	79.35
6	64	989	4.6	79.90
8	48	971	4.6	80.02
12	32	927	4.6	80.08
16	24	860	4.6	80.04
24	16	763	4.6	79.60

Table 3.22 – We compare performance with the default crop-ratio of 0.875 usually used with convnets, and the crop-ratio of 1.0 [214] that we adopt for CaiT.

Network	Crop Ratio		ImNet	Real	V2
	0.875	1.0	top-1	top-1	top-1
S36	✓	–	83.4	88.1	73.0
	–	✓	83.3	88.0	72.5
S36↑384	✓	–	84.8	88.9	74.7
	–	✓	85.0	89.2	75.0
S36Υ	✓	–	83.7	88.9	74.1
	–	✓	84.0	88.9	74.1
M36Υ	✓	–	84.8	89.2	74.9
	–	✓	84.9	89.2	75.0
S36↑384Υ	✓	–	85.2	89.7	75.7
	–	✓	85.4	89.8	76.2
M36↑384Υ	✓	–	85.9	89.9	76.1
	–	✓	86.1	90.0	76.3

number of heads and on typical hardware this leads to a lower effective throughput. Choosing 8 heads in the self-attention offers a good compromise between accuracy and speed. In Deit-Small, this parameter was set to 6.

### 3.7.2.3 Adaptation of the crop-ratio

In the typical (“center-crop”) evaluation setting, most convolutional neural networks crop a subimage with a given ratio, typically extracting a  $224 \times 224$  center crop from a  $256 \times 256$  resized image, leading to the typical ratio of 0.875. Wightman *et al.* [214] notice that setting this crop ratio to 1.0 for transformer models has a positive impact: the distilled DeiT-B↑ 384 reaches a top1-accuracy on ImageNet-1k val of 85.42% in this setting, which is a gain of +0.2% compared to the accuracy of 85.2% reported with DeiT in Table 3.5.

Our measurements concur with this observation: We observe a gain for almost all our models and most of the evaluation benchmarks. For instance our model M36↑384Υ increases to 86.1% top-1 accuracy on ImageNet-1k val.



### 3.7.2.4 Longer training schedules

As shown in Table 3.20, increasing the number of training epochs from 300 to 400 improves the performance of CaiT-S-36. However, increasing the number of training epochs from 400 to 500 does not change performance significantly (83.44 with 400 epochs 83.42 with 500 epochs). This is consistent with the observation of DeiT, which notes a saturation of performance from 400 epochs for the models trained without distillation.

### 3.7.2.5 Visualizations

**Attention map** In Figure 3.9 we show the attention maps associated with the individual 4 heads of a XXS CaiT model, and for the two layers of class-attention. In CaiT and in contrast to ViT, the class-attention stage is the only one where there is some interaction between the class token and the patches, therefore it conveniently concentrates all the spatial-class relationship. We make two observations:

- The first class-attention layer clearly focuses on the object of interest, corresponding to the main part of the image on which the classification decision is performed (either correct or incorrect). In this layer, the different heads focus either on the same or on complementary parts of the objects. This is especially visible for the waterfall image;
- The second class-attention layer seems to focus more on the context, or at least the image more globally.

**Illustration of saliency in class-attention** In figure 3.10 we provide more visualisations for a XXS model. They are just illustration of the saliency that one may extract from the first class-attention layer. As discussed previously this layer is the one that, empirically, is the most related to the object of interest. To produce these visual representations we simply average the attention maps from the different heads (depicted in Figure 3.9), and upsample the resulting map to the image size. We then modulate the gray-level image with the strength of the attention after normalizing it with a simple rule of the form  $(x - x_{\min}) / (x_{\max} - x_{\min})$ . We display the resulting image with `cividis` colormap.

For each image we show this saliency map and provides all the class for which the model assigns a probability higher than 10%. These visualizations illustrate how the model can focus on two distinct regions (like racket and tennis ball on the top row / center). We can also observe some failure cases, like the top of the church classified as a flagpole.

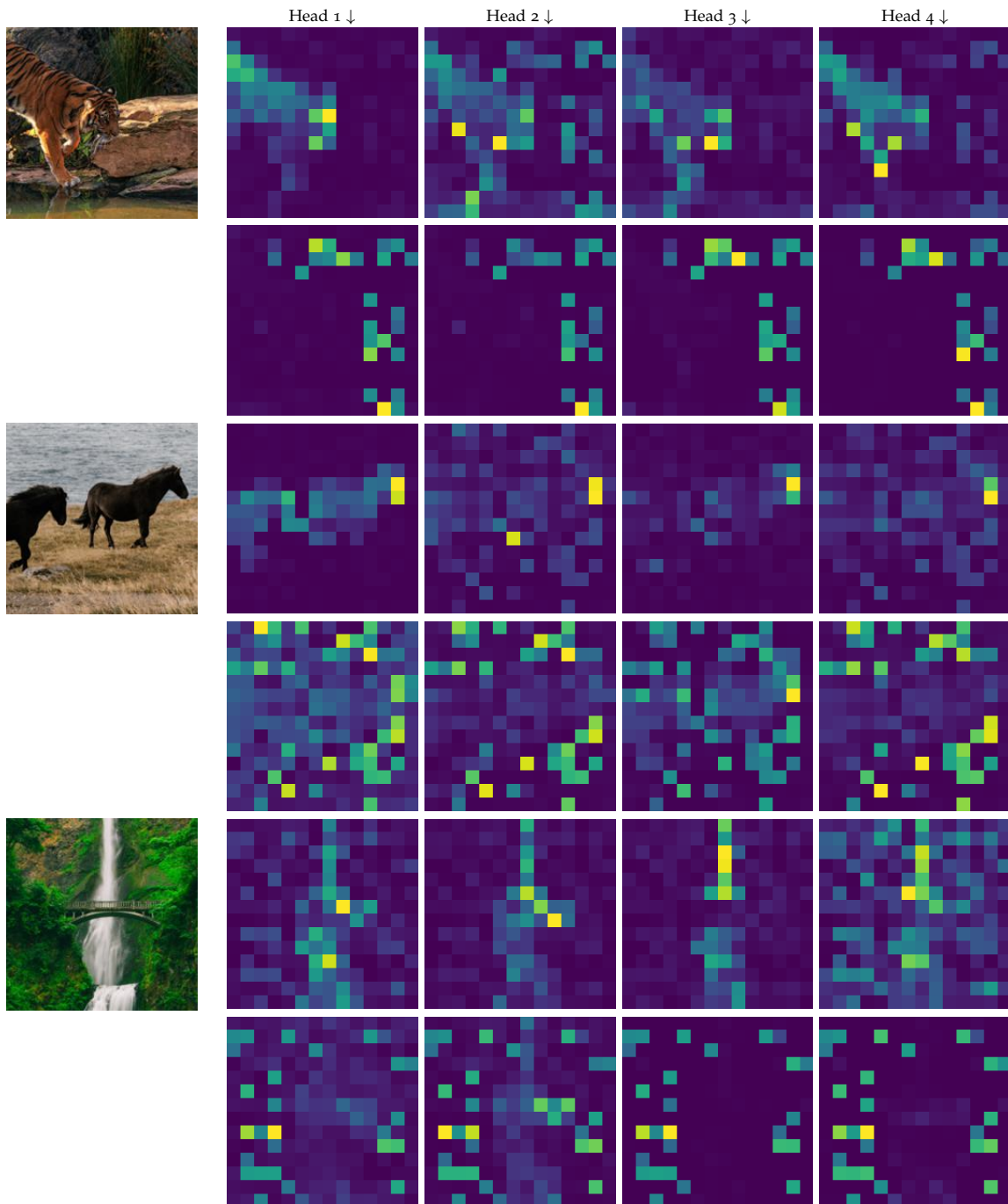


Figure 3.9 – Visualization of the attention maps in the class-attention stage, obtained with a XXS model. For each image we present two rows: the top row correspond to the four heads of the attention maps associated with the first CA layer. The bottom row correspond to the four heads of the second CA layer.

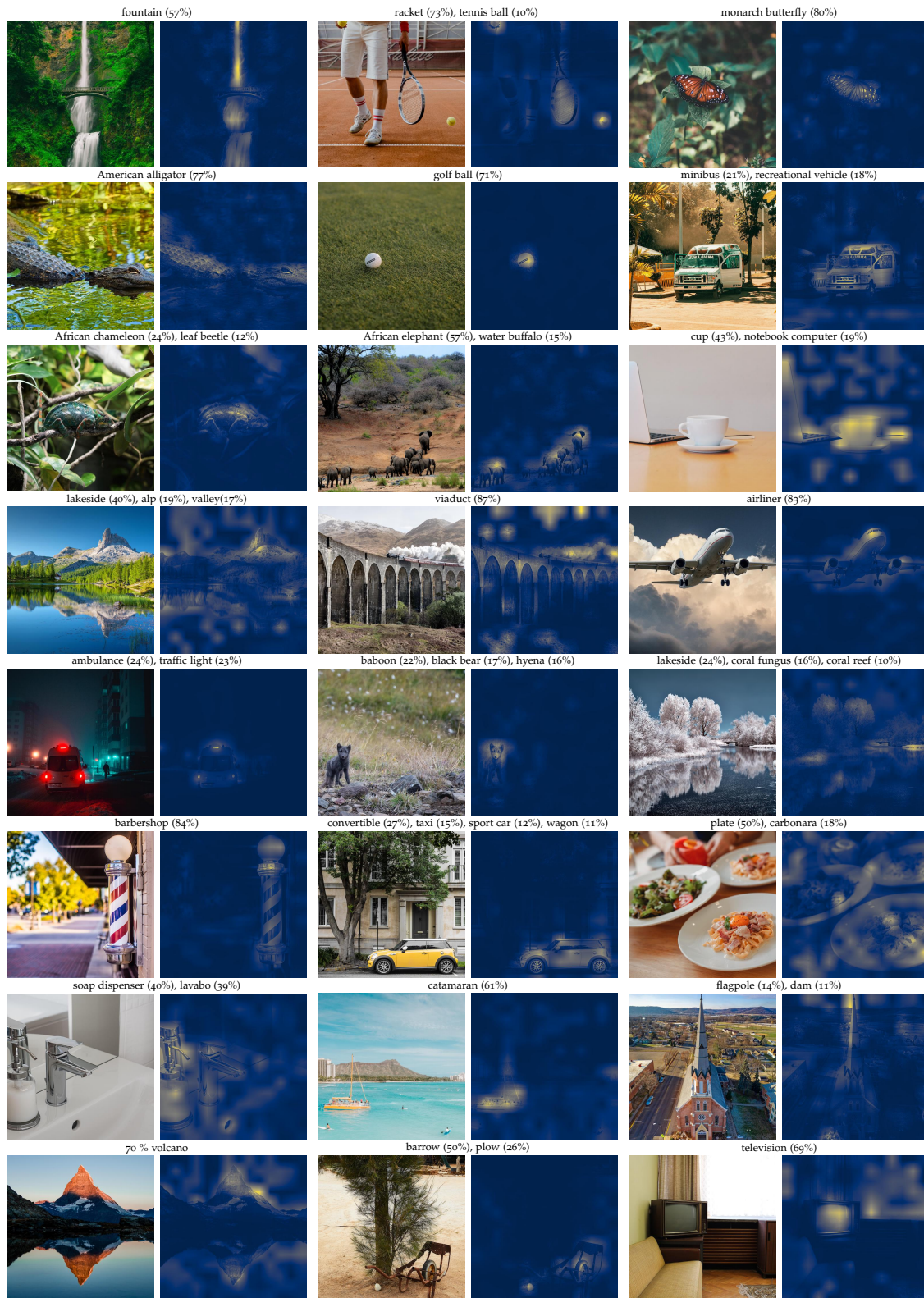


Figure 3.10 – Illustration of the regions of focus of a CaiT-XXS model, according to the response of the first class-attention layer.

## 3.8 Conclusion

In this chapter, we have shown how to train transformer-based image classification neural networks with Imagenet-1k data only. With our Data-efficient image Transformers (DeiT), we report large improvements over previous results, see Figure 3.3. Our ablation study details the hyper-parameters and key ingredients for a successful training, such as repeated augmentation. We show that our neural networks that contain no convolutional layer can achieve competitive results against the state of the art on ImageNet with no external data. They are learned on a single node with 4 GPUs in three days (we can accelerate the learning of the larger model DeiT-B by training it on 8 GPUs in two days). Our two new models DeiT-S and DeiT-Ti have fewer parameters and can be seen as the counterpart of ResNet-50 and ResNet-18.

We address another question: how to distill these models? We introduce a token-based strategy, specific to transformers and denoted by DeiT $\Upsilon$ , and show that it advantageously replaces the usual distillation procedure. Interestingly, with our distillation, image transformers learn more from a convnet than from another transformer with comparable performance.

We propose LayerScale that significantly facilitates the convergence and improves the accuracy of image transformers at larger depths. It adds a few thousands of parameters to the network at training time (negligible w.r.t. the total number of weights). Our specific class-attention design offers a more effective processing of the class embedding. In addition, this simplifies the visualisation of attention maps.

After studying the transformers architecture for computer vision we will try to understand the importance of attention in the next chapter.



## MULTI-LAYER PERCEPTRON FOR COMPUTER VISION

As shown in the chapter 3, the transformer architecture [204], adapted from its original use in natural language processing with only minor changes, has achieved performance competitive with the state of the art on ImageNet-1k [168] when pre-trained with a sufficiently large amount of data [61]. Retrospectively, this achievement is another step towards learning visual features with less priors: Convolutional Neural Networks (CNN) had replaced the hand-designed choices from hard-wired features with flexible and trainable architectures. Vision transformers further remove several hard decisions encoded in the convolutional architectures, namely the translation invariance and local connectivity.

This evolution toward less hard-coded priors in the architecture has been fueled by better training schemes [61, 193]. In this chapter, we push this trend further forward by showing that a purely multi-layer perceptron (MLP) architecture, called Residual Multi-Layer Perceptrons (ResMLP), is competitive on image classification. ResMLP is designed to be simple and encoding little prior knowledge about images: it takes image patches as input, projects them with a linear layer, and sequentially updates their representations with two residual operations: (i) a *cross-patch* linear layer applied to all channels independently; and (ii) a *cross-channel* single-layer MLP applied independently to all patches. At the end of the network, the patch representations are average pooled, and fed to a linear classifier.

The ResMLP architecture is strongly inspired by the vision transformers (ViT) [61], yet it is much simpler in several ways: we replace the self-attention sublayer by a linear layer, resulting in an architecture with only linear layers and GELU non-linearity [98]. We observe that the training of ResMLP is more stable than ViTs when using the same training scheme as in Chapter 3, removing the need for batch-specific or cross-channel normalizations such as BatchNorm, GroupNorm or LayerNorm. We speculate that this stability comes from replacing self-attention with linear layers. Finally, another advantage of using a linear layer is that we can still visualize the interactions between patch embeddings, revealing filters that are similar to convolutions on the lower layers, and longer range in the last layers.

We further investigate if our purely MLP based architecture could benefit to other domains beyond images, and particularly, with more complex output spaces. In particular, we adapt our MLP based architecture to take inputs with variable length, and show its potential on the problem of Machine Translation. To do so, we develop a sequence-to-sequence (seq2seq) version of ResMLP, where both encoder and decoders are based on ResMLP with cross-attention between the encoder and decoder [10]. This model is similar to the original seq2seq Transformer with ResMLP layers instead of Transformer layers [204]. Despite not being originally designed for this task, we observe that ResMLP is competitive with Transformers on the challenging WMT benchmarks.

The chapter is organised as follows: We first detail the related work, We outline our ResMLP architecture in Figure 4.1 and detail it further in Section 4.2. In section 4.3 we conduct image classification, semantic segmentation and machine translation experiments with our ResMLP architecture. Section 4.4 concludes the chapter.

**Publication.** Chapter 4 is based on the paper “ResMLP: Feedforward networks for image classification with data-efficient training”, Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, Hervé Jégou, *arXiv 2021* (see ResMLP paper [192]). The code associated is available at <https://github.com/facebookresearch/deit>.

## 4.1 Related work

We review the research on applying Fully Connected Network (FCN) for computer vision.

**Fully-connected network for images.** Many studies have shown that FCNs are competitive with convnets for the tasks of digit recognition [42, 176], keyword spotting [31] and handwriting recognition [20]. Several works [130, 140, 201] have questioned if FCNs are also competitive on natural image datasets, such as CIFAR-10 [122]. More recently, d’Ascoli *et al.* [51] have shown that a FCN initialized with the weights of a pretrained convnet achieves performance that are superior than the original convnet. Neyshabur [143] further extend this line of work by achieving competitive performance by training an FCN from scratch but with a regularizer that constrains the models to be close to a convnet. These studies have been conducted on small scale datasets with the purpose of studying the impact of architectures on generalization in terms of sample complexity [62] and energy landscape [116]. In our work, we show that, in the larger scale setting of ImageNet, FCNs can attain surprising accuracy without any constraint or initialization inspired by convnets.

Finally, the application of FCN networks in computer vision have also emerged in the study of the properties of networks with infinite width [145], or for inverse scattering problems [117]. More interestingly, the Tensorizing Network [146] is an approximation of very large FCN intending to remove prior by approximating even more general tensor operations, *i.e.*, not arbitrarily marginalized along some pre-defined sharing dimensions. However, their method is designed to compress the MLP layers of a standard convnets.

**Other architectures with similar components.** A fully connected layer is equivalent to a convolution layer with a  $1 \times 1$  receptive field, and several work have explored convnet architectures with small receptive fields. For instance, the VGG model [177] uses  $3 \times 3$  convolutions, and later, other architectures such as the ResNext [229] or the Xception [38] mix  $1 \times 1$  and  $3 \times 3$  convolutions. In contrast to convnets, interactions between patches may be obtained via a linear layer that is shared across channels, and rely on absolute rather than relative positions.

## 4.2 ResMLP

In this section, we describe our architecture, ResMLP, as depicted in Figure 4.1. ResMLP is inspired by ViT and this section focuses on the changes made to ViT that lead to a purely MLP based model. We refer to Dosovitskiy *et al.* [61] for more details about ViT.

**The overall ResMLP architecture.** Our model, denoted by ResMLP, takes a grid of  $N \times N$  non-overlapping patches as input, where the patch size is typically equal to  $16 \times 16$ . The patches are then independently passed through a linear layer to form a set of  $N^2$   $d$ -dimensional embeddings.

The resulting set of  $N^2$  embeddings are fed to a sequence of *Residual Multi-Layer Perceptron* layers to produce a set of  $N^2$   $d$ -dimensional output embeddings. These output embeddings are

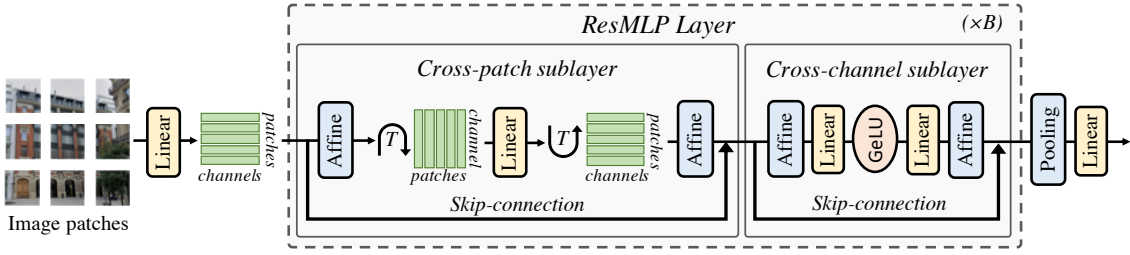


Figure 4.1 – **The ResMLP architecture.** After linearly projecting the image patches into high dimensional embeddings, ResMLP sequentially processes them with (1) a cross-patch linear sublayer; (2) a cross-channel two-layer MLP. The MLP is the same as the FCN sublayer of a Transformer. Each sublayer has a residual connection and two Affine element-wise transformations.

then averaged (“average-pooling”) as a  $d$ -dimension vector to represent the image, which is fed to a linear classifier to predict the label associated with the image. During the training we use the cross-entropy loss.

**The Residual Multi-Perceptron Layer.** Our network is a sequence of layers that all have the same structure: a linear sublayer applied across patches followed by a feedforward sublayer applied across channels. Similar to the Transformer layer, each sublayer is paralleled with a skip-connection [94]. The absence of self-attention layers makes the training more stable, allowing us to replace the Layer Normalization [7] by a simpler Affine transformation:

$$\text{Aff}_{\lambda, \beta}(\mathbf{x}) = \text{Diag}(\boldsymbol{\lambda})\mathbf{x} + \boldsymbol{\beta}, \quad (4.1)$$

where  $\boldsymbol{\lambda}$  and  $\boldsymbol{\beta}$  are learnable weight vectors. This operation only rescales and shifts the input element-wise. This operation has several advantages over other normalization operations: first, as opposed to Layer Normalization, it has no cost at inference time, since it can be absorbed in the adjacent linear layer. Second, as opposed to BatchNorm [112] and Layer Normalization, the  $\text{Aff}$  operator does not depend on batch statistics. The closer operator to  $\text{Aff}$  is the LayerScale introduced by Touvron *et al.* [197], with an additional bias term. For convenience, we denote by  $\text{Aff}(\mathbf{X})$  the Affine operation applied independently to each column of the matrix  $\mathbf{X}$ .

We apply the  $\text{Aff}$  operator at the beginning (“pre-normalization”) and end (“post-normalization”) of each residual block. As a pre-normalization,  $\text{Aff}$  replaces LayerNorm without using channel-wise statistics. Here, we initialize  $\boldsymbol{\lambda} = \mathbf{1}$ , and  $\boldsymbol{\beta} = \mathbf{0}$ . As a post-normalization,  $\text{Aff}$  is similar to LayerScale and we initialize  $\boldsymbol{\lambda}$  with the same small value as in [197].

Overall, our Multi-layer perceptron takes a set of  $N^2$   $d$ -dimensional input features stacked in a  $d \times N^2$  matrix  $\mathbf{X}$ , and outputs a set of  $N^2$   $d$ -dimension output features, stacked in a matrix  $\mathbf{Y}$  with the following set of transformations:

$$\mathbf{D} = \mathbf{X} + \text{Aff} \left( (\mathbf{A} \cdot \text{Aff}(\mathbf{X})^\top)^\top \right), \quad (4.2)$$

$$\mathbf{Y} = \mathbf{D} + \text{Aff}(\mathbf{C} \cdot \text{GELU}(\mathbf{B} \cdot \text{Aff}(\mathbf{D}))), \quad (4.3)$$

where  $\mathbf{A}$ ,  $\mathbf{B}$  and  $\mathbf{C}$  are the main learnable weight matrices of the layer. Note that Eq (4.3) is the same as the feedforward sublayer of a Transformer with the  $\text{ReLU}$  non-linearity replaced by a  $\text{GELU}$  function [98]. The dimensions of the parameter matrix  $\mathbf{A}$  are  $N^2 \times N^2$ , *i.e.*, this “cross-patch” sublayer exchanges information between patches, while the “cross-channel” feedforward sublayer works per location. Similar to a Transformer, the intermediate activation matrix  $\mathbf{D}$  has the same



dimensions as the input and output matrices,  $\mathbf{X}$  and  $\mathbf{Y}$ . Finally, the weight matrices  $\mathbf{B}$  and  $\mathbf{C}$  have the same dimensions as in a Transformer layer, which are  $4d \times d$  and  $d \times 4d$ , respectively.

**Differences with the Vision Transformer architecture.** Our architecture is closely related to the ViT model [61]. However, ResMLP departs from ViT with several simplifications:

- *no self-attention blocks*: it is replaced by a linear layer with no non-linearity,
- *no positional embedding*: the linear layer implicitly encodes information about patch positions,
- *no extra “class” token*: we simply use average pooling on the patch embeddings,
- *no normalization based on batch statistics*: we use a learnable affine operator.

**Class-MLP as an alternative to average pooling.** We propose an adaptation of the class-attention token introduced in CaiT [197]. In CaiT, this consists of two layers that have the same structure as the transformer, but in which only the class token is updated based on the frozen patch embeddings. We translate this method to our architecture, except that, after aggregating the patches with a linear layer, we replace the attention-based interaction between the class and patch embeddings by simple linear layers, still keeping the patch embeddings frozen. This increases the performance, at the expense of adding some parameters and computational cost. We refer to this pooling variant as “class-MLP”, since the purpose of these few layers is to replace the average pooling operation.

**Sequence-to-sequence ResMLP.** Similar to Transformer, the ResMLP architecture can be applied to sequence-to-sequence tasks. First, we follow the general encoder-decoder architecture from Vaswani *et al.* [204], where we replace the self-attention sublayers by the residual multi-perceptron layer. In the decoder, we keep the cross-attention sublayers, which attend to the output of the encoder. In the decoder, we adapt the linear sublayers to the task of language modeling by constraining the matrix  $\mathbf{A}$  to be triangular, in order to prevent a given token representation to access tokens from the future. Finally, the main technical difficulty from using linear sublayers in a sequence-to-sequence model is to deal with variable sequence lengths. However, we observe that simply padding with zeros and extracting the submatrix  $\mathbf{A}$  corresponding to the longest sequence in a batch, works well in practice.

## 4.3 Experiments

In this section, we present experimental results for the ResMLP architecture on image classification and machine translation. We also study the impact of the different components of ResMLP in ablation studies. We consider three training paradigms for images:

- *Supervised learning*: We train ResMLP from labeled images with a softmax classifier and cross-entropy loss. This paradigm is the main focus of our work.
- *Self-supervised learning*: We train our ResMLP architecture with the DINO method of Caron *et al.* [30] that trains a network without labels by distilling knowledge from previous instances of the same network.
- *Knowledge distillation*: We employ the knowledge distillation procedure proposed in chapter 3 to guide the supervised training of ResMLP with a convnet.

Table 4.1 – **Comparison between architectures on ImageNet classification.** We compare different architectures based on convolutional networks, Transformers and feedforward networks with comparable FLOPs and number of parameters. We report Top-1 accuracy on the validation set of ImageNet-1k with different measure of complexity: throughput, FLOPs, number of parameters and peak memory usage. All the models use  $224 \times 224$  images as input. By default the Transformers and feedforward networks uses  $14 \times 14$  patches of size  $16 \times 16$ , see Table 4.3 for the detailed specification of our main models. The throughput is measured on a single V100-32GB GPU with batch size fixed to 32. For reference, we include the state of the art at the time of publication for models trained with ImageNet training only.

	Arch.	#params ( $\times 10^6$ )	throughput (im/s)	FLOPS ( $\times 10^9$ )	Peak Mem (MB)	Top-1 Acc.
<i>State of the art</i>	CaiT-M48 $\uparrow$ 448T [197]	356	5.4	329.6	5477.8	<b>86.5</b>
	NfNet-F6 SAM [25]	438	16.0	377.3	5519.3	<b>86.5</b>
<i>Convolutional networks</i>	EfficientNet-B3 [187]	12	661.8	1.8	1174.0	81.1
	EfficientNet-B4 [187]	19	349.4	4.2	1898.9	82.6
	EfficientNet-B5 [187]	30	169.1	9.9	2734.9	83.3
	RegNetY-4GF [153]	21	861.0	4.0	568.4	80.0
	RegNetY-8GF [153]	39	534.4	8.0	841.6	81.7
	RegNetY-16GF [153]	84	334.7	16.0	1329.6	82.9
<i>Transformer networks</i>	DeiT-S [193]	22	940.4	4.6	217.2	79.8
	DeiT-B [193]	86	292.3	17.5	573.7	81.8
	CaiT-XS24 [197]	27	447.6	5.4	245.5	81.8
<i>Feedforward networks</i>	ResMLP-S12	15	1415.1	3.0	179.5	76.6
	ResMLP-S24	30	715.4	6.0	235.3	79.4
	ResMLP-B24	116	231.3	23.0	663.0	81.0

### 4.3.1 Experimental setting

**Datasets.** We train our models on the ImageNet-1k dataset [168], that contains 1.2M images evenly spread over 1,000 object categories. In the absence of an available test set for this benchmark, we follow the standard practice in the community by reporting performance on the validation set. This is not ideal since the validation set was originally designed to select hyper-parameters. Comparing methods on this set may not be conclusive enough because an improvement in performance may not be caused by better modeling, but by a better selection of hyper-parameters. To mitigate this risk, we report additional results in transfer learning and on two alternative versions of ImageNet that have been built to have distinct validation and test sets, namely the ImageNet-real [18] and ImageNet-v2 [159] datasets. We also report a few data-points when training on ImageNet-21k. Our hyper-parameters are mostly adopted from chapter 3.

**Hyper-parameter settings.** In the case of supervised learning, we train our network with the Lamb optimizer [234] with a learning rate of  $5 \times 10^{-3}$  and weight decay 0.2. We initialize the LayerScale parameters as a function of the depth by following CaiT [197]. The rest of the hyper-parameters follow the default setting used in Chapter 3. For the knowledge distillation paradigm, we use the same RegNety-16GF [156] as in Chapter 3 with the same training schedule. The majority of our models take two days to train on eight V100-32GB GPUs.

### 4.3.2 Main Results

In this section, we compare ResMLP with architectures based on convolutions or self-attentions with comparable size and throughput on ImageNet.

Table 4.2 – **Self-supervised learning** with DINO [30]. Classification accuracy on ImageNet-1k val. ResMLPs evaluated with linear and  $k$ -NN evaluation are on par with convnets but inferior to ViTs.

Models	Params. ( $\times 10^6$ )	FLOPS ( $\times 10^9$ )	ImNet-val top-1 acc.	
			Linear	$k$ -NN
ResNet-50	25	4.1	75.3	67.5
ViT-S/16	22	4.6	77.0	74.5
ViT-S/8	22	22.4	<b>79.7</b>	<b>78.3</b>
ViT-B/16	87	17.5	78.2	76.1
ResMLP-S12	15	3.0	67.5	62.6
ResMLP-S24	30	6.0	72.8	69.4

**Supervised setting.** In Table 4.1, we compare ResMLP with different convolutional and Transformer architectures. For completeness, we also report the best-published numbers obtained with a model trained on ImageNet alone. While the trade-off between accuracy, FLOPs, and throughput for ResMLP is not as good as convolutional networks or Transformers, their strong accuracy still suggests that the structural constraints imposed by the layer design do not have a drastic influence on performance, especially when training with enough data and recent training schemes.

**Self-supervised setting.** We pre-train ResMLP-S12 using the self-supervised method called DINO [30] during 300 epochs. We report our results in Table 4.2. The trend is similar to the supervised setting: the accuracy obtained with ResMLP is lower than ViT. Nevertheless, the performance is surprisingly high for a pure MLP architecture and competitive with Convnet in  $k$ -NN evaluation. Additionally, we also fine-tune a model pre-trained with self-supervision on ImageNet using the ground-truth labels. Pre-training substantially improves performance compared to a ResMLP-S24 solely trained with labels, achieving 79.9% top-1 accuracy on ImageNet-val (+0.5%).

**Knowledge distillation setting.** We study our model when training with the knowledge distillation approach of Touvron *et al.* [193]. In their work, the authors show the impact of training a ViT model by distilling it from a RegNet. In this experiment, we explore if ResMLP also benefits from this procedure and summarize our results in Table 4.3 (Blocks “Baseline models” and “Training”). We observe that similar to DeiT models, ResMLP greatly benefits from distilling from a convnet. This result concurs with the observations made by d’Ascoli *et al.* [51], who used convnets to initialize feedforward networks. Even though our setting differs from theirs in scale, the problem of overfitting for feedforward networks is still present on ImageNet. The additional regularization obtained from the distillation is a possible explanation for this improvement.

### 4.3.3 Visualization & analysis of the linear interaction between patches

In Figure 4.2, we show in the form of squared images, the rows of the weight matrix from cross-patch sublayers at different depths of a ResMLP-S24 model. The early layers show convolution-like patterns: the weights resemble shifted versions of each other and have local support. Interestingly, in many layers, the support also extends along both axes; see layer 7. The last 7 layers of the network are different: they consist of a spike for the patch itself and a diffuse response across other patches with different magnitude; see layer 20.

**Measuring sparsity of the weights.** The visualizations described above suggest that the linear communication layers are sparse. We analyze this quantitatively in more detail in Figure 4.3.

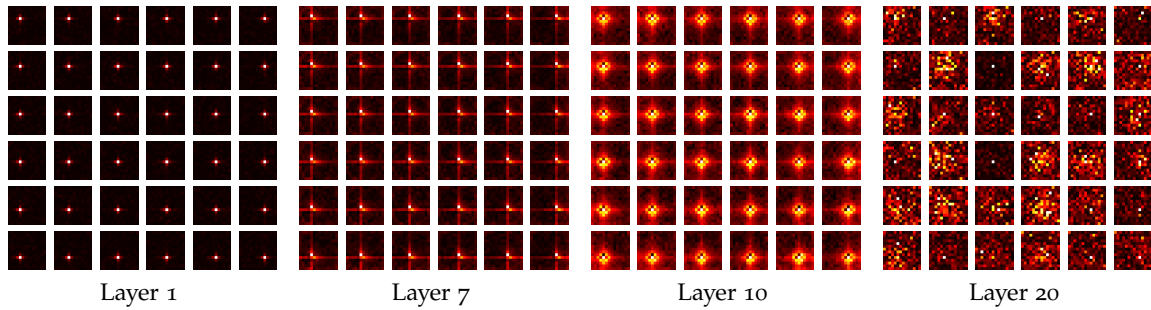


Figure 4.2 – **Visualisation of the linear layers in ResMLP-S24.** For each layer we visualise the rows of the matrix  $\mathbf{A}$  as a set of  $14 \times 14$  pixel images, for sake of space we only show the rows corresponding to the  $6 \times 6$  central patches. We observe patterns in the linear layers that share similarities with convolutions.

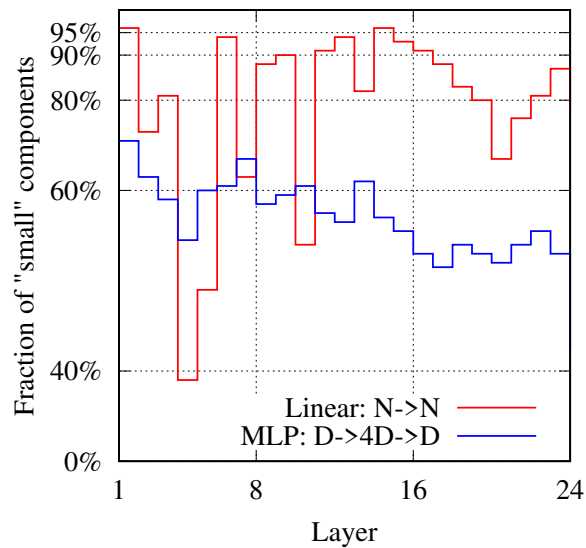


Figure 4.3 – **Sparsity of linear interaction layers.** For each layer (linear and MLP), we show the rate of components whose absolute value is lower than 5% of the maximum. Linear interaction layers are sparser than the matrices involved in the per-patch MLP.

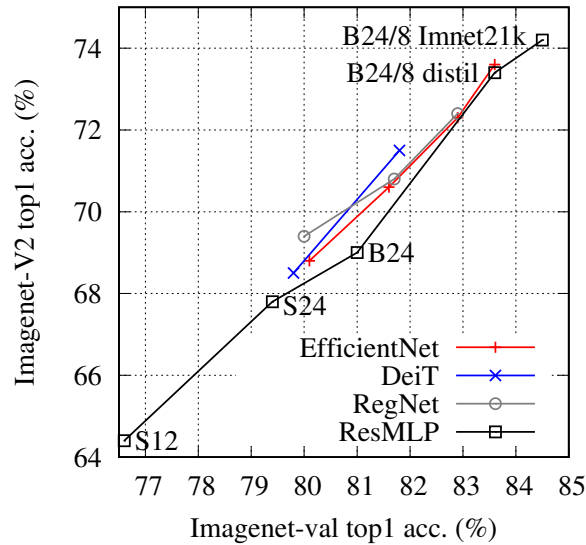


Figure 4.4 – **Top-1 accuracy on ImageNet-V2 vs. ImageNet-val.** ResMLPs tend to overfit slightly more under identical training method. This is partially alleviated with by introducing more regularization (more data or distillation, see e.g., ResMLP-B24/8-distil).

We measure the sparsity of the matrix  $\mathbf{A}$ , and compare it to the sparsity of  $\mathbf{B}$  and  $\mathbf{C}$  from the per-patch MLP. Since there are no exact zeros, we measure the rate of components whose absolute value is lower than 5% of the maximum value. Note, discarding the small values is analogous to the case where we normalize the matrix by its maximum and use a finite-precision representation of weights. For instance, with a 4-bits representation of weight, one would typically round to zero all weights whose absolute value is below 6.25% of the maximum value.

The measurements in Figure 4.3 show that all three matrices are sparse, with the layers implementing the patch communication being significantly more so. This suggests that they may be compatible with parameter pruning, or better, with modern quantization techniques that induce sparsity at training time, such as Quant-Noise [70] and DiffQ [64]. The sparsity structure, in particular in earlier layers, see Figure. 4.2, hints that we could implement the patch interaction linear layer with a convolution. We provide some results for convolutional variants in our ablation study. Further research on network compression is beyond the scope of this chapter, yet we believe it worth investigating in the future.

**Communication across patches.** If we remove the linear interaction layer (linear  $\rightarrow$  none), we obtain substantially lower accuracy (-20% top-1 acc.) for a “bag-of-patches” approach. We have tried several alternatives for the cross-patch sublayer, which are presented in Table 4.3 (block “patch communication”). Amongst them, using the same MLP structure as for patch processing (linear  $\rightarrow$  MLP). The simpler choice of a single linear square layer led to a better accuracy/performance trade-off – considering that the MLP variant requires compute halfway between ResMLP-S12 and ResMLP-S24 – and requires fewer parameters than a residual MLP block.

The visualization in Figure 4.2 indicates that many linear interaction layers look like convolutions, the kernels being increasingly larger closer to the output layer. Hence in our ablation, we replaced the linear layer with different types of  $3 \times 3$  convolutions. The depth-wise convolution does not implement interaction across channels – as our linear patch communication layer – and yields similar performance with a comparable number of parameters and FLOPs. While full  $3 \times 3$  convolutions yield best results, they require roughly double the number of parameters and FLOPs. Interestingly, the depth-separable convolutions combine accuracy close to that of full

Table 4.3 – **Ablation.** Our default configurations are presented in the three first rows. By default we train during 400 epochs. The “old-fashioned” is similar to what was employed for ResNet [94]: SGD, 90-epochs waterfall schedule, same augmentations up to variations due to library used.

Ablation	Model	Patch size	Params $\times 10^6$	FLOPs $\times 10^9$	Variant	top-1 acc. on ImageNet		
						val	real [18]	v2 [159]
Baseline models	ResMLP-S12	16	15.4	3.0	12 layers, working dimension 384	76.6	83.3	64.4
	ResMLP-S24	16	30.0	6.0	24 layers, working dimension 384	79.4	85.3	67.9
	ResMLP-B24	16	115.7	23.0	24 layers, working dimension 768	81.0	86.1	69.0
Normalization	ResMLP-S12	16	15.4	3.0	Aff $\rightarrow$ Layernorm	77.7	84.1	65.7
Pooling	ResMLP-S12	16	17.7	3.0	average pooling $\rightarrow$ Class-MLP	77.5	84.0	66.1
Patch communication	ResMLP-S12	16	14.9	2.8	linear $\rightarrow$ none	56.5	63.4	43.1
	ResMLP-S12	16	18.6	4.3	linear $\rightarrow$ MLP	77.3	84.0	65.7
	ResMLP-S12	16	30.8	6.0	linear $\rightarrow$ conv $3\times 3$	77.3	84.4	65.7
	ResMLP-S12	16	14.9	2.8	linear $\rightarrow$ conv $3\times 3$ depth-wise	76.3	83.4	64.6
	ResMLP-S12	16	16.7	3.2	linear $\rightarrow$ conv $3\times 3$ depth-separable	77.0	84.0	65.5
Patch size	ResMLP-S12/14	14	15.6	4.0	patch size $16\times 16\rightarrow 14\times 14$	76.9	83.7	65.0
	ResMLP-S12/8	8	22.1	14.0	patch size $16\times 16\rightarrow 8\times 8$	79.1	85.2	67.2
	ResMLP-B24/8	8	129.1	100.2	patch size $16\times 16\rightarrow 8\times 8$	81.0	85.7	68.6
Training	ResMLP-S12	16	15.4	3.0	old-fashioned (90 epochs)	69.2	76.0	56.1
	ResMLP-S12	16	15.4	3.0	pre-trained SSL (DINO)	76.5	83.6	64.5
	ResMLP-S12	16	15.4	3.0	distillation	77.8	84.6	66.0
	ResMLP-S24	16	30.0	6.0	pre-trained SSL (DINO)	79.9	85.9	68.6
	ResMLP-S24	16	30.0	6.0	distillation	80.8	86.6	69.8
	ResMLP-B24/8	8	129.1	100.2	distillation	83.6	88.4	73.4
	ResMLP-B24/8	8	129.1	100.2	pre-trained ImageNet-21k (60 epochs)	<b>84.4</b>	<b>88.9</b>	<b>74.2</b>

$3\times 3$  convolutions with a number of parameters and FLOPs comparable to our linear layer. This suggests that convolutions on low-resolution feature maps at all layers is an interesting alternative to the common pyramidal design of convnets, where early layers operate at higher resolution and smaller feature dimension.

#### 4.3.4 Ablation studies

Table 4.3 reports the ablation study of our base network and a summary of our preliminary exploratory studies.

**Control of overfitting.** Since MLPs are subject to overfitting, we show in Fig. 4.4 a control experiment to probe for problems with generalization. We explicitly analyze the differential of performance between the ImageNet-val and the distinct ImageNet-V2 test set. The relative offsets between curves reflect to which extent models are overfitted to ImageNet-val w.r.t. hyperparameter selection. The degree of overfitting of our MLP-based model is overall neutral or slightly higher to that of other transformer-based architectures or convnets with same training procedure.

**Normalization & activation.** Our network configuration does not contain any batch normalizations. Instead, we use the affine per-channel transform Aff. This is akin to Layer Normalization [7], typically used in transformers, except that we avoid to collect any sort of statistics, since we do not need it for convergence. In preliminary experiments with pre-norm and post-norm [95], we observed that both choices converged. Pre-normalization in conjunction with Batch Normalization could provide an accuracy gain in some cases.

We choose to use a GELU [98] function. We also analyze the activation function: ReLU [82] also gives a good performance, but it was a bit more unstable in some settings. We did not manage to get good results with SiLU [98] and HardSwish [105].

Table 4.4 – **Evaluation on transfer learning.** Classification accuracy (top-1) of models trained on ImageNet-1k for transfer to datasets covering different domains. The ResMLP architecture takes  $224 \times 224$  images during training and transfer, while ViTs and EfficientNet-B7 work with higher resolutions, see “Res.” column.

Architecture	FLOPs	Res.	CIFAR <sub>10</sub>	CIFAR <sub>100</sub>	Flowers <sub>102</sub>	Cars	iNat <sub>18</sub>	iNat <sub>19</sub>
EfficientNet-B7 [187]	37.0B	600	98.9	<b>91.7</b>	<b>98.8</b>	<b>94.7</b>	–	–
ViT-B/16 [61]	55.5B	384	98.1	87.1	89.5	–	–	–
ViT-L/16 [61]	190.7B	384	97.9	86.4	89.7	–	–	–
DeiT-B/16 [193]	17.5B	224	<b>99.1</b>	90.8	98.4	92.1	<b>73.2</b>	<b>77.7</b>
ResNet50 [198]	4.1B	224	–	–	96.2	90.0	68.4	73.7
Grafit/ResNet50 [198]	4.1B	224	–	–	97.6	92.7	68.5	74.6
ResMLP-S12	3.0B	224	98.1	87.0	97.4	84.6	60.2	71.0
ResMLP-S24	6.0B	224	98.7	89.5	97.9	89.5	64.3	72.5

**Pooling.** Replacing average pooling with Class-MLP, see Section 4.2, brings a significant gain for a negligible computational cost. We do not include it by default to keep our models more simple.

**Patch size.** Smaller patches significantly increase the performance, but also increase the number of flops (see Block “Patch size” in the ablation Table 4.3). Smaller patches benefit more to larger models, but only with an improved optimization scheme involving more regularization (distillation) or more data.

**Training.** Consider the Block “Training” in Table 4.3. ResMLP significantly benefits from modern training procedures such as those used in Chapter 3. For instance, the Chapter 3 training procedure improves the performance of ResMLP-S12 by 7.4% compared to the training employed for ResNet [94]<sup>1</sup>. This is in line with recent work pointing out the importance of the training strategy over the model choice [13, 156]. Pre-training on more data and distillation also improve the performance of ResMLP architecture, especially for the bigger models, *e.g.*, distillation improves the top-1 accuracy of ResMLP-B24/8 by 2.6%.

**Other analysis.** In our early exploration, we evaluated several alternative design choices. As in transformers, we could use positional embeddings mixed with the input patches. In our experiments we did not see any benefit from using these features. This observation suggests that our cross-patch sublayer provides sufficient spatial communication, and referencing absolute positions obviates the need for any form of positional encoding.

**Transfer learning** We evaluate the quality of features obtained from a ResMLP architecture when transferring them to other domains. The goal is to assess if the features generated from a feedforward network are more prone to overfitting on the training data distribution. We adopt the typical setting where we pre-train a model on ImageNet-1k and fine-tune it on the training set associated with a specific domain. We report the performance with different architectures on various image benchmarks in Table 4.4, namely CIFAR-10 and CIFAR-100 [122], Flowers-102 [144], Stanford Cars [121] and iNaturalist [103]. We refer the reader to the corresponding references for a more detailed description of the datasets.

We observe that the performance of our ResMLP is competitive with the existing architectures, showing that pretraining feedforward models with enough data and regularization via data augmentation greatly reduces their tendency to overfit on the original distribution. Interestingly, this regularization also prevents them from overfitting on the training set of smaller dataset during the fine-tuning stage.

1. Interestingly, if trained with this “old-fashion” setting, ResMLP-S12 outperforms AlexNet [123] by a margin.

### 4.3.5 Machine translation

We also evaluate the ResMLP transpose-mechanism to replace the self-attention in the encoder and decoder of a neural machine translation system. We train models on the WMT 2014 English-German and English-French tasks, following the setup from Ott *et al.* [149]. We consider models of dimension 512, with a hidden MLP size of 2,048, and with 6 or 12 layers. Note that the current state of the art employs much larger models: our 6-layer model is more comparable to the base transformer model from Vaswani *et al.* [204], which serves as a baseline, along with pre-transformer architectures such as recurrent and convolutional neural networks. We use Adagrad with learning rate 0.2, 32k steps of linear warmup, label smoothing 0.1, dropout rate 0.15 for En-De and 0.1 for En-Fr. We initialize the LayerScale parameter to 0.2. We generate translations with the beam search algorithm, with a beam of size 4. As shown in Table 4.5, the results are at least on par with the compared architectures.

Table 4.5 – **Machine translation** on WMT 2014 translation tasks. We report tokenized BLEU on *newstest2014*.

Models	GNMT [219]	ConvS2S [79]	Transf. (base) [204]	ResMLP-6	ResMLP-12
EN-DE	24.6	25.2	<b>27.3</b>	26.4	26.8
EN-FR	39.9	40.5	38.1	40.3	<b>40.6</b>

### 4.3.6 Semantic Segmentation

We perform semantic segmentation experiments on the ADE20k [248] datasets with ResMLP models pre-trained on ImageNet. We adopt the classical UperNet [222] setting with the  $\times 3$  schedule [12, 67]. We finetune for the semantic segmentation task at resolution  $224 \times 224$  and evaluate the network with the usual resolution by adopting a sliding window. We report results in Table 4.6 and compare ResMLP with DeiT with the same setting. In that case ResMLP is used as backbone in order to extract features before the UperNet part. At training time we resize images at resolution  $896 \times 224$  and apply RandomResizeCrop in order to have an image of size  $224 \times 224$ . ResMLP obtains interesting results, but note that vision transformers remain better. One of the main limitations is the fixed spatial linear layer which makes it more difficult to adapt the ResMLP architecture at different resolutions. Indeed, the size of the linear layer applied to the spatial dimension is fixed and can only be used with a certain image size. It is possible to interpolate the weights to adapt to other resolutions but this does not allow a very good adaptability.

Table 4.6 – **Semantic segmentation** results on ADE20K dataset with UperNet and  $\times 3$  settings. All architectures are trained with crop of size  $224 \times 224$

Model	DeiT-S	DeiT-B	ResMLP-12	ResMLP-24	ResMLP-36	ResMLP-B24
mIoU (%)	40.5	42.3	35.9	39.1	39.1	<b>42.5</b>

## 4.4 Conclusion

In this chapter, we have shown that a simple residual architecture, whose residual blocks consist of a one-hidden layer feed-forward network and a linear patch interaction layer, achieves high performance on ImageNet classification benchmarks, provided that we adopt a modern training strategy such as those described in chapter 3 for transformer-based architectures. Thanks



to their simple structure, with linear layers as the main mean of communication between patches, we can visualize the filters learned by this simple MLP. While some of the layers are similar to convolutional filters, we also observe sparse long-range interactions as early as the second layer of the network.

In summary, the main contributions are the following:

- We propose a drastically simplified design for neural network, namely ResMLP, whose main difference compared to a MLP is the residual design borrowed from earlier convolutional neural network architectures;
- Albeit simple, ResMLP reaches surprisingly decent accuracy/complexity trade-offs when trained on ImageNet-1k only;
- These models benefit significantly from distillation methods introduced in chapter 3 and are easily combined with modern self-supervised learning methods based on data augmentation, such as DINO [30];
- A seq2seq ResMLP achieves competitive performances compared to a seq2seq Transformers on the WMT benchmark for Machine Translation.

Our model with less of spatial prior gives interesting perspectives to design new networks with less inductive bias than most of convolutional neural networks.

In the next chapter, we study how to revisit convnet architectures with elements inherited from transformers and MLP architectures like patch splitting.

## REVISITING CONVNET ARCHITECTURE

In transformers, the so-called “class token” correlates with the patches most related to the classification decision. Therefore, the softmax in the self-attention blocks, especially in the last layers, can be used to produce attention maps showing the interaction between the class token and all the patches. Such maps have been employed for visualization purposes [30, 61]. It gives some hints on which regions of a given image are employed by a model to make its decision. However, the interpretability remains limited: producing these maps involves some fusion of multiple softmax in different layers and heads.

In this chapter, we investigate similar visualization properties for convnets: we augment convnets with an attention map. More precisely, we replace the usual average pooling layer by an attention-based layer. Indeed, nothing in the convnets design precludes replacing their pooling by attention [111, 14]. We design our attention-based pooling layer such that it explicitly provides the weights of the different patches. Compared to ViT, for which the aggregation is performed across multiple layers and heads, our proposal offers a single weight per patch, and therefore a simple way to interpret the attention map: it is the respective contribution of each patch in the weighted sum summarizing the images.

We introduce a simple patch-based convolutional architecture that keeps the input resolution constant throughout the network. This design departs from the historical pyramidal architectures of LeNet [126], AlexNet [123] or ResNet [94, 95], to name only a few. Their pyramidal design was motivated by the importance of reducing the resolution while increasing the working dimensionality. That allowed one to maintain a moderate complexity while progressively increasing the working dimensionality, making the space large enough to be separable by a linear classifier. In our case, we simplify the trunk after a small pre-processing stage that produces the patches. We adopt the same dimensionality throughout all the trunk, fixing it equal to that of the final layer, e.g. our aggregation layer. We refer to it as PatchConvNet.

This chapter is organised as follows: we first detail the related work. In section 5.2, we describe our architecture PatchConvNet and present the training recipes. In section 5.3, we conduct image classification and semantic segmentation experiments.

**Publication.** Chapter 5 is based on the submission “*Augmenting Convolutional networks with attention-based aggregation*”, Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Piotr Bojanowski, Armand Joulin, Gabriel Synnaeve, Hervé Jégou, *arXiv 2021, under review NeurIPS 2022* (see PatchConvnet paper [195]). The code is available at <https://github.com/facebookresearch/deit>.

### 5.1 Related work

**Attention-based architectures for vision.** Early works have introduced attention into convnets [14, 158, 175, 212, 216], but it is only recently that a fully attention-based architecture, the vision transformer [61] (ViT), has become competitive with convnets on ImageNet [61, 193]. The particularity of this model is that it processes images as a set of non-overlapping patches, without any convolu-

tional or downsampling layers. Nevertheless, several works have recently proposed to re-introduce convolutions and downsampling into this architecture. For example, some architectures [86, 225] leverage convolutional layers in the first layers of the vision transformer architecture, while others, such as Swin [134], LeViT [86], or PiT [99] exploit a pyramid structure to gradually reduce the spatial resolution of the features.

The pyramid-based methods are more compatible with prior detection frameworks, and aim at improving the computational efficiency (FLOPs). As a downside, the pyramidal approaches dramatically reduce the resolution of the last layers, and hence the quality of their attention maps, making their predictions harder to interpret. Another shortcoming is their relatively high memory usage [171]. In chapter 3, we adopt a few layers in one decoder using class-attention instead of self attention. Other works [111, 114, 127] propose this type of attention to aggregate features at the end of the network or at intermediate levels. Our learned attention-based pooling is more simple and offers a better interpretability than these approaches.

**MLP and other patch-based approaches.** Architectures based on patches [132] have been proposed beyond transformers, in particular, based on Multi-Layer Perceptron (MLP) layers such as MLP-Mixer [191] and the work chapter 4. Most related to our work, the ablation study of chapter 4 shows the potential of patch-wise convolution over MLPs in terms of performance. In line with the ConViT model [52], CoatNet [49] is a patch-based architecture with blocks yielding local-interactions followed by transformer blocks. Concurrently, replacing self-attention layers with convolution layers has been explored in the ConvMixer paper [6].

**Explainability of the classification decision.** There are many strategies to explain the classification decision of a network [160, 238], and most notably by highlighting the most influential regions that led to a decision [178, 246, 73]. Grad-cam methods [172, 32] are certainly the most used methods to give explanation about network decision. Inspired by the CAM [246] principle, they exploit the gradients from the network decision to identify specific object locations that can be backprojected onto the image. These methods act as general external probes that project the network activity back into the image space, even though [148, 63] have shown evidence that convnet features contain rough information about the localization of objects. Unlike these external approaches, the self-attention layers of vision transformers offer a direct access to the location of the information used to make classification decisions [61, 193, 197, 30]. Our built-in class attention mechanism shares the same spirit of *interpretable by design* computer vision models [166]. However, unlike our mechanism, self-attention layers do not distinguish between classes on the same image without additional steps [33].

## 5.2 Attention-based pooling with PatchConvNet

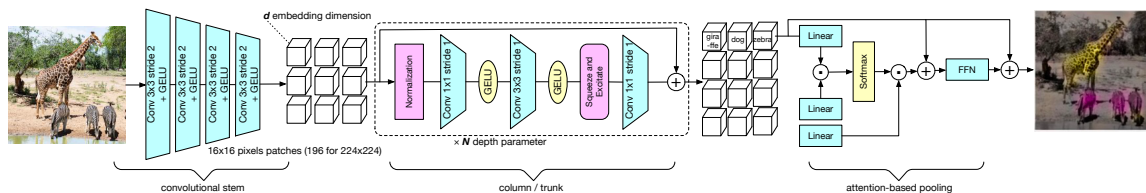


Figure 5.1 – Detail of the full model, with the convolutional stem on the left, the convolutional main block in the middle, and here topped with multi-class attention-based pooling on the right.

The learned aggregation layer is best associated with a high-resolution feature map. Therefore, while it can be combined with any convolutional architecture like a regular ResNet-50, our suggestion is to combine it with an architecture that maintains the resolution all across the layers. Some works exist, however they offer underwhelming trade-offs [191, 192]. To remedy that problem, we introduce PatchConvNet. This design, which is illustrated in Figure 5.1, is intended to concentrate most of the compute and parameters in the columnar trunk. The architecture family is parametrized by the embedding dimension  $d$ , and the number of repeated blocks in the trunk  $N$ . Below, we describe the architecture and its training in more details.

### 5.2.1 Architecture design

**The convolutional stem** is a light-weight pre-processing of the image pixels whose role is to segment and map an image into a set of vectors. In ViT, this exactly corresponds to the patch extraction step [61]. Therefore, we refer to the vectors resulting from this pre-processing as *patches*. Recent papers [86, 66] have shown that it is best to adopt a convolutional pre-processing, in particular for stability reasons [224]. In our case, we borrow the convolutional stem from LeViT [86]: a small ConvNet that is applied to the image of size  $W \times H \times 3$  and produces a vector map of  $W/16 \times H/16 \times d$ . It can be viewed as a set of  $k$  non-overlapping  $d$ -dimensional patches. In our experimental results, except if mentioned otherwise, we use a convolutional stem consisting of four  $3 \times 3$  convolutions with a stride of  $2 \times 2$ , followed by a GELU non-linearity [98]. We illustrate the convolutional stem in Figure 5.1.

**The column,** or trunk, is the part of the model which accounts for most of the layers, parameters, and compute. It consists of  $N$  stacked residual convolutional blocks as depicted in Figure 5.1. The block starts with a normalization, followed by a  $1 \times 1$  convolution, then a  $3 \times 3$  convolution for spatial processing, a squeeze-and-excitation layer [107] for mixing channel-wise features, and finally a  $1 \times 1$  convolution right before the residual connection. Note that we can interpret the  $1 \times 1$  convolutions as linear layers. A GELU non-linearity follows the first two convolutions. The output of this block has the same shape as its input: the same number of tokens of the same dimension  $d$ .

Using BatchNorm [113] often yields better results than LayerNorm [7], provided the batches are large enough. As shown in Section 5.3, we also observe this for our model family. However, BatchNorm is less practical when training large models or when using large image resolutions because of its dependency on batch size. In that setup, using BatchNorm requires an additional synchronization step across multiple machines. This synchronization increases the amount of node-to-node communication required per step, and in turn, training time. In other situations, like for detection and segmentation, the images are large, limiting the batch size and possibly impacting performance. Because of all those reasons, unless stated otherwise, we adopt LayerNorm.

**Attention-based pooling.** At the output of the trunk, the pre-processed vectors are aggregated using a cross-attention layer inspired by transformers. We illustrate this aggregation mechanism in Figure 5.1. A *query* class token attends to the projected patches and aggregates them as a weighted summation. The weights depend on the similarity of projected patches with a trainable vector (CLS) akin to a class token. The resulting  $d$ -dimensional vector is subsequently added to the CLS vector and processed by a feed-forward network (FFN). As opposed to the class-attention decoder by Touvron *et al.* [197] we use a single block and a single head. This drastic simplification has the benefit of avoiding the dilution of attention across multiple channels. Consequently, the communication between the class token and the pre-processed patches occurs in a single softmax, directly reflecting how the pooling operator weights each patch.

We can easily specialize the attention maps per class by replacing the CLS vector with a  $k \times d$  matrix, where each of the  $k$  columns is associated with one of the classes. This specialization allows us to visualize an attention map for each class, as shown in Figure 5.6. The impact of the additional parameters and resulting FLOPs is minimal for larger models in the family. However, this design increases peak memory usage and makes the optimization of the network more complicated. We typically do that in a fine-tuning stage with a lower learning rate and smaller batch size to circumvent these issues. By default, we use the more convenient single class token.

## 5.2.2 Discussion: analysis & properties

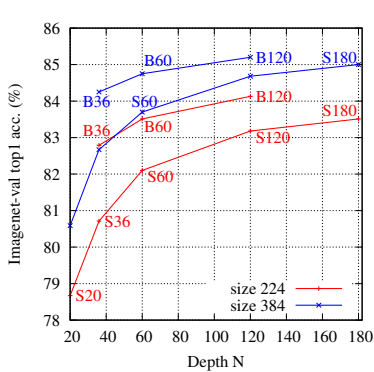


Figure 5.2 – Analysis of the accuracy as a function of width (S:  $d=384$ , B:  $d=768$ ) and depth  $N$ . Depending on the performance criterion (importance of latency, resolution, FLOPs), one could prefer either deeper models or wider models. See Bello *et al.* [13] for a study on the relationship between model size, resolution and compute.

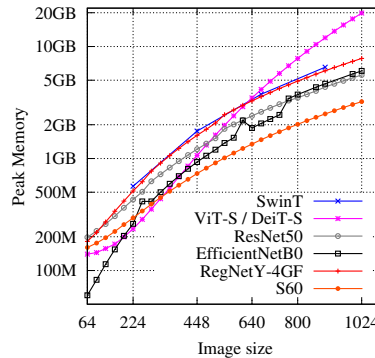


Figure 5.3 – Peak memory for varying resolution and different models. Some models like Swin require a full training at the target resolution. Our model scales linearly as a function of the image surface, like other ConvNets. This is in contrast to most attention-based models, which abide by a quadratic complexity for images large enough.

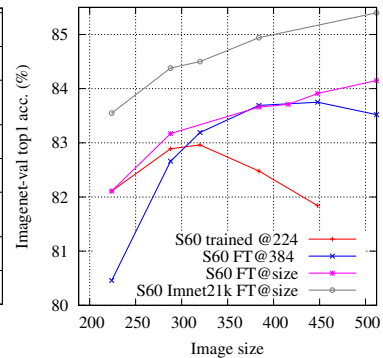


Figure 5.4 – Accuracy at different resolutions for the S60 model. We analyze models trained at size 224 or fine-tuned (FT) @384, and compare them to models fine-tuned at the target inference size to show the tolerance to test-time resolution changes. The best model are pre-trained on ImageNet21k at resolution 224 or 320 and fine-tuned at test-time resolution.

Below we discuss several properties of our convolutional trunk augmented with the proposed attention-based aggregation stage.

1. *Simple parametrization.* Our main models are fully defined by width and depth. See Figure 5.2 for results obtained with these models at two different resolutions (224 and 384). Following the same convention as in previous work on vision transformers and vision MLPs [61, 193, 192], we refer by S the models with an vector size of  $d=384$  per patch, by B when  $d=768$ , and by L for  $d=1024$ . We use the S60 model for most of our ablations and comparisons since it has a similar number of parameters and FLOPs as a ResNet-50.
2. *Visualization.* Our model allows to easily visualize the network activity. Saliency maps are directly extracted from our network without any post-processing.
3. *Constant resolution across the trunk.* The patch-based processing leads to a single processing resolution in the trunk. Therefore the activation size is constant across the whole network. The memory usage is (almost) constant at inference time, up to the pre- and post-processing stage, which are comparatively less demanding. Compared to traditional ConvNets, the network has a coarser processing in the early stages, but a finer resolution towards the output of the trunk.

4. *Linear scaling with image size.* This is a key difference with Vision Transformers. Pyramidal transformers such as LeViT [86], SwinTransformer [134] or MViT [71] partly solve the problem by breaking the quadratic component by rapidly down-scaling the image. However, they don't avoid the memory peaks happening with very large images. As a consequence of that constant memory usage and linear scaling, our model smoothly scales to larger resolutions, as shown in Figure 5.3 where we report the Peak Memory usage at inference time as a function of the image size.
5. *Easy change of resolution.* We do not require any positional encoding, as the relative patch positions are handled by the convolutions. In that respect our approach is more flexible than most approaches that needs to be fine-tuned or trained from scratch for each possible target resolution. In Figure 5.4 we show that the properties of our models are quite stable under relatively significant resolution changes.
6. *No pooling.* There is no pooling or other non-reversible operator in our architecture. Formally the function implemented by the trunk is bijective until the aggregation stage. We do not exploit this property, but it may be useful in contexts like image generation [58, 119].

### 5.2.3 Training recipes

Like many other works (see Liu et al. [133], Table I), our training algorithm inherits from the Chapter 3 procedure for training transformers. We adopt the Lamb optimizer [234] (a variant of AdamW [136]) with a half-cosine learning schedule and label smoothing [185]. For data augmentation, we include the RandAugment [44] variant by Wightman *et al.* [215], Mixup [242] ( $\alpha = 0.8$ ) and CutMix [237] ( $\alpha = 1.0$ ). Notably, we include Stochastic Depth [108] that is very effective for deep transformers [197], and for which we observe the same effect with our deep PatchConvNet. We adopt a uniform drop rate for all layers, and we cross-validate this parameter on ImageNet1k for each model. We also adopt LayerScale [197] introduced in Chapter 3. For the deepest models, the drop-rate hyper-parameter (often called “drop-path”) can be set as high as 0.5, meaning that we can potentially drop half of the trunk. A desirable byproduct of this augmentation is that it accelerates the training. Note that we do not use gradient clipping, Polyak averaging, or erasing to keep our procedure simple enough.

We now detail some context-dependent adjustments, based on datasets (ImageNet1k or ImageNet21k), and training (from scratch or fine-tuned). Note that, apart our sensitivity study, we use the same Seed  $o$  for all our experiments [215] to prevent picking a “lucky seed” [152] that would not be representative of the model performance.

**Training on ImageNet1k.** We train during 400 epochs with a batch size of 2048 and a learning rate fixed at  $3.10^{-3}$  for *all models*. Based on early experiments, we fixed the weight decay to 0.01 for S models and 0.05 for wider models, but practically we observed that the stochastic depth parameter had a preponderant influence and the most important to adjust, similar to prior observations with ViT *et al.* [197]. We use repeated augmentation [15] only when training with this dataset.

**Fine-tuning at higher resolutions.** We fine-tune our models at higher resolutions in order to correct the train-test resolution discrepancy [199], and to analyze the behavior of our models at higher resolutions. This can save a significant amount of resources because models operating at larger resolutions are very demanding to train. For fine-tuning, we use a smaller batch size of 1024 in order to compensate for the larger memory requirements. We fix the learning rate to  $10^{-5}$ , the weight decay to 0.01, and fine-tune during 10 epochs for all our models.

**Training on ImageNet21k.** We train during 90 epochs as in prior works [61, 134]. We trained with a batch size of 2048 with a learning rate of  $3 \cdot 10^{-3}$  and weight decay of 0.01, or when possible with a batch size of 4096 in order to accelerate the training. In that case we adjust the learning rate to  $4 \cdot 10^{-3}$ .

**Fine-tuning from ImageNet21k to ImageNet1k** is a more involved modification of the network than just fine-tuning across resolutions because one needs to re-learn the classifiers. In that case, we adopt a longer fine-tuning schedule of 100 epochs along with a batch size of 1024 and an initial learning rate of  $5 \cdot 10^{-4}$  with a half-cosine schedule.

## 5.3 Main experimental results

This section presents our main experimental results in Image classification, detection and segmentation. We also include an ablation study. Our code depends on the PyTorch [3] and timm libraries [214]. Model weights and implementation of our main models are open-source.

### 5.3.1 Class activation

In Figure 5.5, we show the attention maps extracted from ViT by using a visualization as in [30]. It involves some post-processing as there are multiple layers and heads providing patch weights. Then we show a "ResNet-50" augmented by adding our attention-based aggregation layer. Its hierarchical design leads to a low-resolution attention map with artefacts: We need an architecture producing a higher-resolution feature maps in order to better leverage the proposed attention-based pooling.

### 5.3.2 Classification results

We first compare our model with competing approaches on the validation set of ImageNet-1k (Imnet-val / Top-1) and ImageNet-v2 in Table 5.1. We report the compute requirement as reflected by FLOPs, the peak memory usage, the number of parameters, and a throughput at inference time measured for a constant batch-size of 256 images.

We compare with various models, including classic models like ResNet-50 [94] revisited with modern training recipes such as the one recently proposed by [215]. Note however that different models may have received a different optimization effort, therefore the results on a single criterion are mostly indicative. That being pointed out, we believe that the PatchConvNet results show that a simple columnar architecture is a viable choice compared to other attention-based approaches that are more difficult to optimize or scale.

**Higher-resolution.** There is a fine interplay between model size and resolution when it comes to the specific optimization of FLOPs and accuracy. We refer to the findings of [13] who discussed some of these relationships, for instance the fact that small networks are better associated with smaller resolution. In our work, we did not specifically optimize the Pareto curve. Since this trade-off is only one out of multiple criteria depending on the context, we prefer to report most of our results at the 224 and 384 resolutions. Table 5.1 shows that our model significantly benefits from larger resolution images. See also Figures 5.3 and 5.4 where we analyze PatchConvNet as a function of the image size. Table 5.2 we analyze PatchConvNet pre-trained on ImageNet-21k with different fine-tuning resolution. All networks are pre-trained on ImageNet-21k during 90

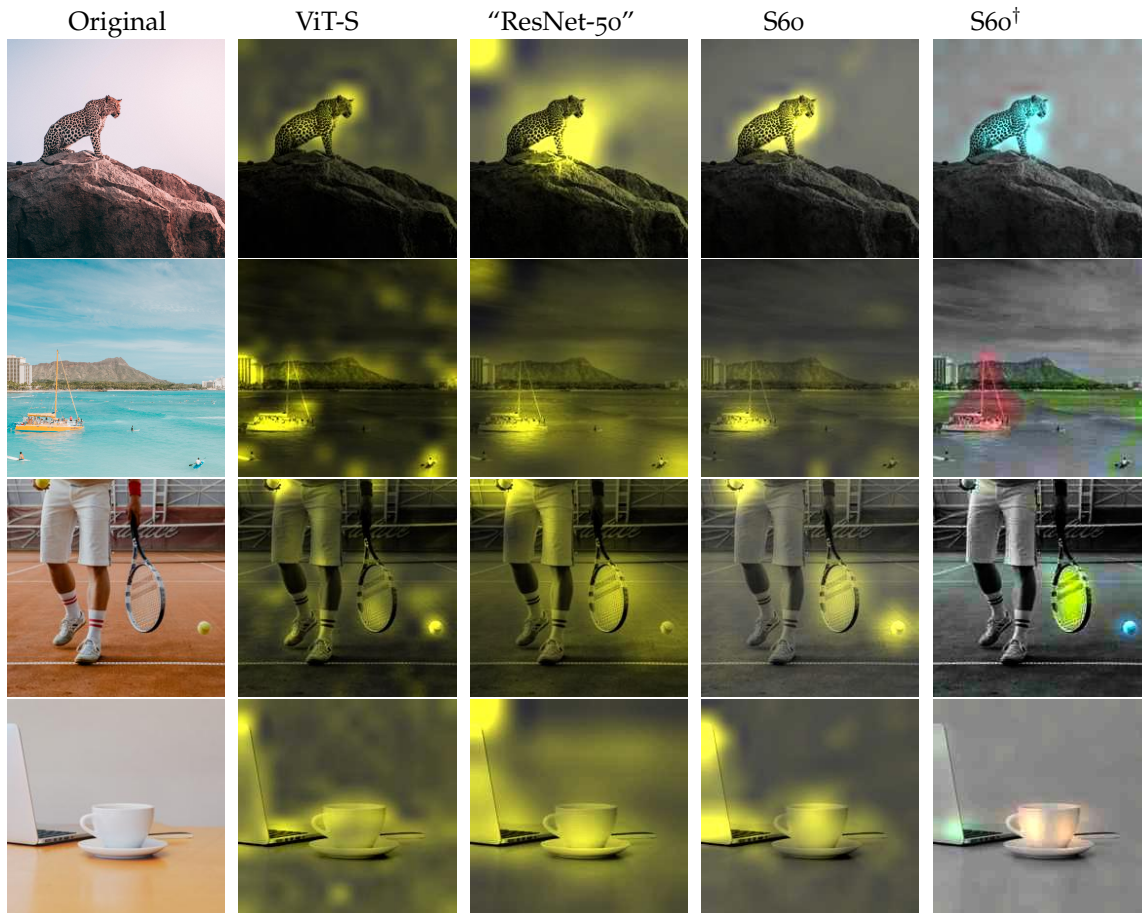


Figure 5.5 – We augment convolutional neural networks with a learned attention-based aggregation layer. We visualize the attention maps for classification for diverse models. We first extract attention maps from a regular ViT-S [61, 193]. Then we consider convnets in which we replace the average pooling by our learned attention-based aggregation layer. Unlike ViT, this layer directly provides the contribution of the patches in the weighted pooling. This is shown for a “ResNet-50 [94]”, and with our new simple patch-based model (S60) that we introduce to increase the attention map resolution. We can specialize this attention per class, as shown with S60†.

epochs at resolution 224, finetuned on ImageNet-1k at resolution 384 and then fine-tuned at bigger resolution.

**Transfer Learning experiments** We evaluate our architecture on 6 transfer learning tasks. The datasets used are summarized Table 5.4. For fine-tuning we used the procedure used in Chapter 3. Our results are summarized Table 5.5. We can observe that our architecture achieves competitive performance on transfer learning tasks.

### 5.3.3 Segmentation results and detection

**Semantic segmentation** We evaluate our models with semantic segmentation experiments on the ADE20k dataset [247]. This dataset consist of 20k training and 5k validation images with labels from over 150 categories. For the training, we adopt the same schedule as in Swin: 160k iterations with UPerNet [222]. At test time we evaluate with a single scale similarly to XciT and multi-scale as in Swin. As our approach is not pyramidal we only use the final output of our network in



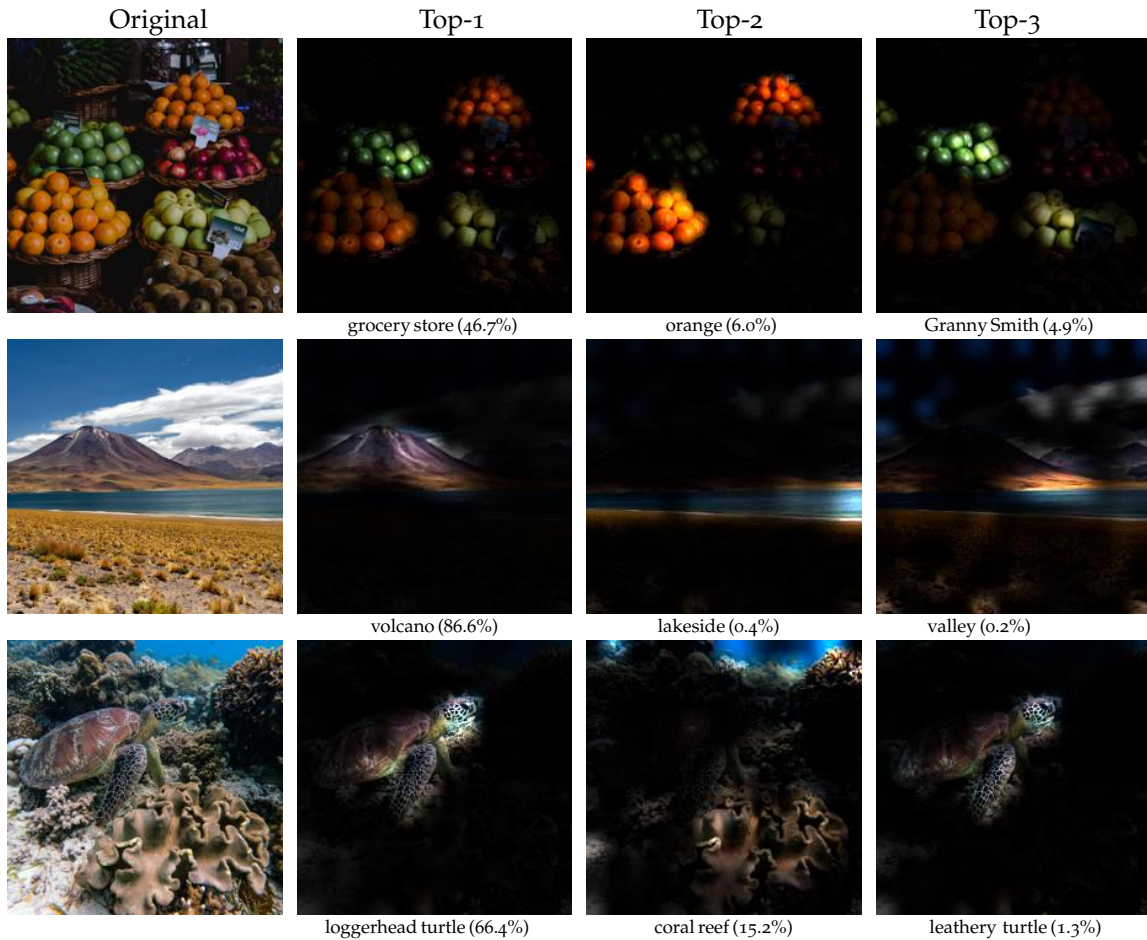


Figure 5.6 – We provide three images for which the attention-based aggregation stage is specialized so as to provide one attention map per ImageNet classes. We display the attention for the top-3 classes w.r.t. the model prediction.

UPerNet. Unlike concurrent approaches we only use the final output of our network at different levels in UPerNet which simplifies the approach.

Our results are reported in Table 5.3. We can observe that our approach although simpler is at the same level as the state-of-the-art Swin architecture and outperforms the XCiT architecture w.r.t. FLOPs-mIoU trade-off.

**Detection & instance segmentation** We have evaluated our models on detection and instance segmentation tasks on COCO [129]. We adopt the Mask R-CNN [93] setup with the commonly used  $\times 3$  schedule. Similar to segmentation experiments, as our approach is not pyramidal, we only use the final output of our network in Mask R-CNN [93]. In Table 5.6, our results show that PatchConvNet is on par with Swin [134] and XCiT [66] in terms of FLOPs-AP tradeoff.

### 5.3.4 Ablations

All our ablation have been carried out with “Seed 0”, i.e., we report only one result without handpicking. For this reason one must keep in mind that there is a bit of noise in the performance measurements: On ImageNet-1k val, we have measured with the seeds 1 to 10 a standard deviation

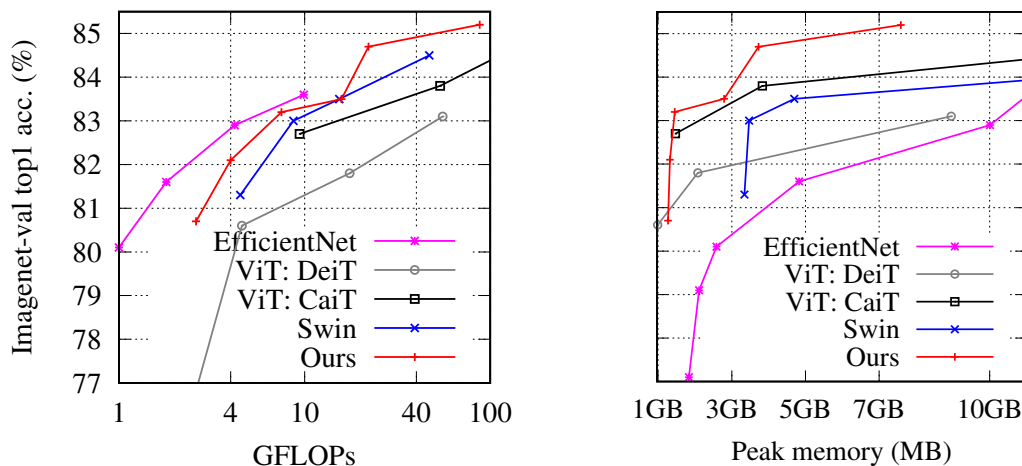


Figure 5.7 – Trade-offs for ImageNet-1k top 1 accuracy vs. FLOPs requirement and peak memory requirements (for a batch of 256 images). Non-hierarchical architectures are comparatively inferior w.r.t. the accuracy-FLOPs trade-off than hierarchical ones, but offer better operating points in terms of accuracy-memory compromise at inference time.

of  $\pm 0.11\%$  in top-1 accuracy for a S60 model, which concurs with measurements done on ResNet-50 trained with modern training procedures [215].

**Stochastic depth.** Our main parameter is the stochastic depth, whose effect is analyzed in Fig. 5.8. This regularization slows down the training, yet with long enough schedules, higher values of the *drop-path* hyperparameter lead to better performance at convergence. We train with the values reported in Table 5.8. When fine-tuning at higher resolutions or from ImageNet-21k, we reduce this *drop-path* by 0.1.

**Architectural ablation.** In Table 5.7, we have conducted various ablations of our architecture with the S60 model. We compare the impact of class attention *vs.* average-pooling. Average-pooling is the most common aggregation strategy in ConvNet while class attention is only used with transformers [197]. We compare also convolutional stem *vs.* linear projection for the patch extraction in the image, LayerNorm *vs.* BatchNorm and Multi-head class attention as used in CaiT [197] *vs.* single-head class attention. Our single-head design reduces the memory consumption and simplifies attention map visualization.

**LayerNorm vs BatchNorm.** LayerNorm is the most used normalisation in transformers while BatchNorm is the most used normalisation with ConvNets. For simplicity we have used LayerNorm as it does not require (batch) statistics synchronisation during training, which tends to slow the training, especially on an infrastructure with relatively high synchronisation costs.

In Table 5.8 we compare the effects of LayerNorm with those of BatchNorm. We can see that BatchNorm increases the PatchConvNet top-1 accuracy. This difference tends to be lower for the deeper models.

**Attention-based pooling with ConvNets.** Interestingly, our learned aggregation stage increases the performance of a very competitive ResNet model. When adopting the recent training recipe from Wightman *et al.* [215], we obtain 80.1% top-1 accuracy on ImageNet-1k by adding a learned pooling to a ResNet50. This is an improvement of +0.3% to the corresponding 300-epoch baseline based on average pooling. The class attention only slightly increases the number of FLOPs of the models: 4.6B vs 4.1B.

Table 5.1 – **Classification with ImageNet-1k training.** We compare architectures based on convolutional networks [6, 25, 94, 156, 187]), Transformers [193, 134] and feedforward networks [191, 192] with comparable FLOPs and number of parameters. All models are trained on ImageNet-1k only without distillation nor self-supervised pre-training. We report Top-1 accuracy on the validation set of ImageNet-1k and ImageNet-V2 with different measures of complexity: throughput, FLOPs, number of parameters and peak memory usage. The throughput and peak memory are measured on a single V100-32GB GPU with batch size fixed to 256 and mixed precision. For ResNet [94] and RegNet [156] we report the improved results from [215]. Note that different models may have received a different optimization effort. †R indicates that the model is fine-tuned at the resolution  $R$ .

Architecture	nb params ( $\times 10^6$ )	throughput (im/s)	FLOPs ( $\times 10^9$ )	Peak Mem (MB)	Top-1 Acc.	V2 Acc.
<b>“Traditional” ConvNets</b>						
ResNet-50	25.6	2587	4.1	2182	80.4	68.7
RegNetY-4GF	20.6	1779	4.0	3041	81.5	70.7
RegNetY-8GF	39.2	1158	8.0	3939	82.2	71.1
RegNetY-16GF	83.6	714	16.0	5204	82.9	72.4
EfficientNet-B4	19.0	573	4.2	10006	82.9	72.3
EfficientNet-B5	30.0	268	9.9	11046	83.6	73.6
NFNet-F0	71.5	950	12.4	4338	83.6	72.6
NFNet-F1	132.6	337	35.5	6628	84.7	74.4
<b>Vision Transformers and derivatives</b>						
ViT: DeiT-S	22.0	1891	4.6	987	80.6	69.4
ViT: DeiT-B	86.6	831	17.5	2078	81.8	71.5
Swin-T-224	28.3	1109	4.5	3345	81.3	69.5
Swin-S-224	49.6	718	8.7	3470	83.0	71.8
Swin-B-224	87.8	532	15.4	4695	83.5	–
<b>Vision MLP</b>						
Mixer-L/16	208.2	322	44.6	2614	71.8	56.2
Mixer-B/16	59.9	993	12.6	1448	76.4	63.2
ResMLP-S24	30.0	1681	6.0	844	79.4	67.9
ResMLP-B24	116.0	1120	23.0	930	81.0	69.0
<b>Patch-based ConvNets</b>						
ResMLP-S12 conv3x3	16.7	3217	3.2	763	77.0	65.5
ConvMixer-768/32	21.1	271	20.9	2644	80.2	–
ConvMixer-1536/20	51.6	157	51.4	5312	81.4	–
Ours-S60	25.2	1125	4.0	1321	82.1	71.0
Ours-S120	47.7	580	7.5	1450	83.2	72.5
Ours-B60	99.4	541	15.8	2790	83.5	72.6
Ours-B120	188.6	280	29.9	3314	84.1	73.9

We point out that we have not optimized the training recipes further (either without or with class-attention). This result is reported for a single run (Seed 0) in both cases.

**Patch pre-processing.** In the vanilla patch-based approaches as vision transformers [61, 193] and MLP-style models [191, 192], the images patches are embedded by one linear layer. Recent works [86, 225] show that replacing this linear patch pre-processing by a few convolutional layers allows to have a more stable architecture [225] with better performance. So, in our work we choose to use a convolutional stem instead of pure linear projection. We provide in Table 5.7 an ablation of this component.

Table 5.2 – **ImageNet-21k pre-training**: Comparison of PatchConvNet fine-tuned at different resolutions on ImageNet-1k. We report peak memory (MB) and throughput (im/s) on one GPU V100 with batch size 256 and mixed precision. Larger resolution provides classification improvement with the same model, but significantly increase the resource requirements. [*italic: results obtained with a longer training*].

Model	GFLOPs	Peak Mem	throughput (im/s)	Res	Imnet-val top-1 Acc
S60	4.0	1322	1129	224	82.9 [83.5]
S60	6.6	2091	692	288	84.0 [84.4]
S60	11.8	3604	388	384	84.6 [84.9]
S60	20.9	6296	216	512	85.0 [85.4]
B60	15.8	2794	547	224	85.0 [85.4]
B60	26.1	4235	328	288	85.7
B60	46.5	7067	185	384	86.1 [86.5]
L60	28.1	3913	394	224	85.6
L60	46.4	5801	237	288	86.1
L60	82.5	9506	132	384	86.4
B120	29.8	3313	280	224	86.0
B120	49.3	4752	169	288	86.6
B120	87.7	7587	96	384	86.9
L120	53.0	4805	204	224	86.1
L120	87.5	6693	123	288	86.6
L120	155.5	10409	68	384	87.1

Table 5.3 – **ADE20k semantic segmentation** performance using UperNet [223] (in comparable settings [59, 66, 134]). All models are pre-trained on ImageNet-1k except models with † symbol that are pre-trained on ImageNet-21k (Swin-B at resolution  $640 \times 640$ ).

Backbone	UperNet			
	#params ( $\times 10^6$ )	FLOPs ( $\times 10^9$ )	Single scale mIoU	Multi scale mIoU
ResNet50	66.5	–	42.0	–
DeiT-S	52.0	1099	–	44.0
XciT-T12/16	34.2	874	41.5	–
XciT-S12/16	54.2	966	45.9	–
Swin-T	59.9	945	44.5	46.1
Ours-S60	57.1	952	<b>46.0</b>	<b>46.9</b>
XciT-M24/16	112.2	1213	47.6	–
XciT-M24/8	110.0	2161	48.4	–
Swin-B	121.0	1188	48.1	49.7
Ours-B60	140.6	1258	48.1	48.6
Ours-B120	229.8	1550	<b>49.4</b>	<b>50.3</b>
Swin-B†	121.0	1841	50.0	51.6
CSWin-B†	109.2	1941	51.8	52.6
Ours-S60†	57.1	952	48.4	49.3
Ours-B60†	140.6	1258	50.5	51.1
Ours-B120†	229.8	1550	51.9	52.8
Ours-L120†	383.7	2086	<b>52.2</b>	<b>52.9</b>

Table 5.4 – Datasets used for our transfer learning tasks.

Dataset	Train size	Test size	#classes
iNaturalist 2018 [104]	437,513	24,426	8,142
iNaturalist 2019 [103]	265,240	3,003	1,010
Flowers-102 [144]	2,040	6,149	102
Stanford Cars [121]	8,144	8,041	196
CIFAR-100 [124]	50,000	10,000	100
CIFAR-10 [124]	50,000	10,000	10

Table 5.5 – Results in transfer learning.

Model	CIFAR-10	CIFAR-100	Flowers	Cars	iNat-18	iNat-19	FLOPs
ResNet-50 [215]	98.3	86.9	97.9	92.7	–	73.9	4.1B
Grafit [198]	–	–	98.2	92.5	69.8	75.9	4.1B
EfficientNet-B7 [187]	98.9	91.7	98.8	94.7	–	–	37.0B
ViT-B/16 [61]	98.1	87.1	89.5	–	–	–	55.5B
ViT-L/16 [61]	97.9	86.4	89.7	–	–	–	190.7B
DeiT-B [193]	99.1	90.8	98.4	92.1	73.2	77.7	17.5B
CaiT-S-36 [197]	99.2	92.2	98.8	93.5	77.1	80.6	13.9B
CaiT-M-36 [197]	<b>99.3</b>	<b>93.3</b>	99.0	93.5	76.9	81.7	53.7B
Ours-S60	99.2	91.4	98.8	94.0	72.9	78.1	4.0B
Ours-B120	99.2	91.1	99.0	94.4	74.3	79.5	29.9B
Ours-S60 @ 320	99.1	91.4	98.9	94.5	76.8	81.4	8.2B
Ours-B120 @ 320	99.1	91.2	<b>99.1</b>	<b>94.8</b>	<b>79.6</b>	<b>82.5</b>	60.9B

Table 5.6 – COCO object detection and instance segmentation performance on the mini-val set. Backbones [66, 94, 134, 229, 211, 244] are all pre-trained on ImageNet-1k, use Mask R-CNN [93] and the same 3× train schedule.

Backbone	#params	GFLOPs	AP <sup>b</sup>	AP <sub>50</sub> <sup>b</sup>	AP <sub>75</sub> <sup>b</sup>	AP <sup>m</sup>	AP <sub>50</sub> <sup>m</sup>	AP <sub>75</sub> <sup>m</sup>
ResNet50	44.2M	180	41.0	61.7	44.9	37.1	58.4	40.1
ResNet101	63.2M	260	42.8	63.2	47.1	38.5	60.1	41.3
ResNeXt101-64	101.9M	424	44.4	64.9	48.8	39.7	61.9	42.6
PVT-Small	44.1M	–	43.0	65.3	46.9	39.9	62.5	42.8
PVT-Medium	63.9M	–	44.2	66.0	48.2	40.5	63.1	43.5
XCiT-S12/16	44.4M	295	45.3	67.0	49.5	40.8	64.0	43.8
XCiT-S24/16	65.8M	385	46.5	68.0	50.9	41.8	65.2	45.0
ViL-Small	45.0M	218	43.4	64.9	47.0	39.6	62.1	42.4
ViL-Medium	60.1M	294	44.6	66.3	48.5	40.7	63.8	43.7
ViL-Base	76.1M	365	45.7	67.2	49.9	41.3	64.4	44.5
Swin-T	47.8M	267	46.0	68.1	50.3	41.6	65.1	44.9
Ours-S60	44.9M	264	46.4	67.8	50.8	41.3	64.8	44.2
Ours-S120	67.4M	339	47.0	69.0	51.4	41.9	65.6	44.7

Table 5.7 – Ablation of our model: we modify each time a single architectural characteristic in our Patch-ConvNet model S60, and measure how it affects the classification performance on ImageNet-1k. Batch-normalization improves the performance a bit. The convolutional stem is key for best performance, and the class-attention brings a slight improvement in addition to enabling attention-based visualisation properties.

↓ Modification to the architecture	Top-1 acc.
none	82.1
class-attention → average pooling	81.9
conv-stem → linear projection	80.0
layer-normalization → batch-normalization	82.4
single-head attention → multi-head attention	81.9
a single class-token → one class-token per class	81.1

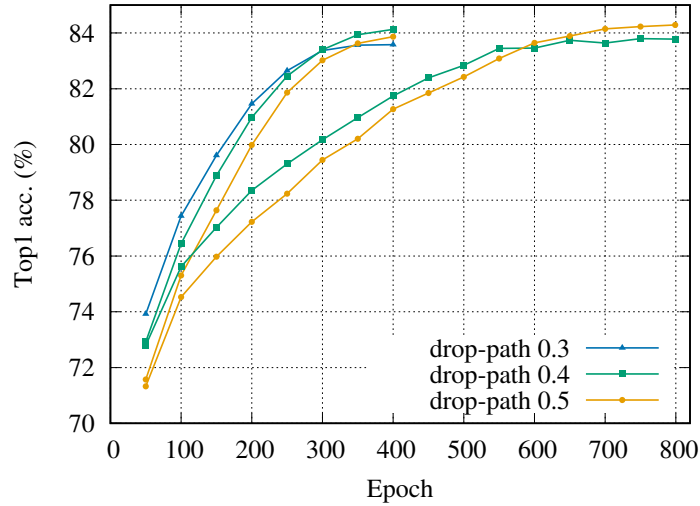


Figure 5.8 – Effect of stochastic depth on the performance for varying training duration for a PatchConvNet-B120 model trained @ resolution 224. The corresponding hyper-parameter (*drop-path*) is selected among 0.3, 0.4 or 0.5 in that case, which means that we randomly drop up to half of the layers. Smaller values of the drop-rate converge more rapidly but saturate.

Table 5.8 – Comparison of PatchConvNet with Layer-Normalization and Batch-Normalization: Performance on ImageNet-1k val after pre-training on ImageNet-1k train only. The *drop-path* parameter value is obtained by cross-validation on ImageNet-1k for each model. Batch-Normalization usually provides a slight improvement in classification, but with large models the need for synchronization can significantly slow down the training (in some cases like training a B120 model on AWS, it almost doubled the training time). Therefore we do not use it by default in this chapter.

Model	<i>drop-path</i>	ImageNet-val Top-1 acc.	
		LayerNorm	BatchNorm
S20	0.0	78.7	78.8
S36	0.05	80.7	81.2
S60	0.15	82.1	82.4
S120	0.2	83.2	83.4
B36	0.2	82.8	83.5
B60	0.3	83.5	83.9
B120	0.4	84.1	84.3

## 5.4 Conclusion

In this chapter, we introduced an attention-based pooling layer which offers visualization properties and interpretability by design. In addition, we proposed a full patch-based ConvNet with no pyramidal structure design in order to better exploit our pooling layer. We demonstrated its interest on several computer vision tasks: classification, segmentation, detection.

In summary, we make the following contributions:

- We revisit the final pooling layer in convnets and introduce a simple learned, attention-based pooling, which provides a direct visualization and interpretability of the decision;
- We propose a slight adaptation of our attention-based pooling in order to have one attention map per class, offering an interpretability of the predictions per class;
- We propose an architecture, PatchConvNet, with a simple patch-based design (two parameters: depth and width), which we design so that it offers better visualizations of the class attention maps (see Figure 5.5) by maintaining a relatively high resolution across all layers.

After studying different architectures for computer vision and in particular for image classification, we will study the interaction between the architecture and the training procedure.

## ARCHITECTURE AND TRAINING INTERACTION REVISITED

After their vast success in NLP, transformer models [204] and their derivatives are increasingly popular in computer vision. Because they incorporate as priors only the co-localisation of pixels in patches, transformers have to learn about the structure of images while optimizing the model such that it processes the input with the objective of solving a given task. This can be either reproducing labels in the supervised case, or other proxy tasks in the case of self-supervised approaches. Nevertheless, despite their huge success, there has been only few works in computer vision studying how to efficiently train vision transformers, and in particular on a midsize dataset like ImageNet-1k. Since the work of Dosovistky et al. [61], the training procedures are mostly variants from the proposal of DeiT (Chapter 3) and Steiner et al. [183]. In contrast, multiple works have proposed alternative architectures by introducing pooling, more efficient attention, or hybrid architectures re-incorporating convolutions and a pyramid structure. These new designs, while being particularly effective for some tasks, are less general. One difficult question to address is whether the improved performance is due to a specific architectural design, or because it facilitates the optimization as suggested it is the case for convolutions with ViTs [224].

Recently, self-supervised approaches inspired by the popular BeiT pre-training have raised hopes for a BeiT moment in computer vision. There are some analogies between the fields of NLP and computer vision, starting with the transformer architecture itself. However, these fields are not identical in every way: The modalities processed are of different nature (continuous versus discrete). Computer vision offers large annotated databases like ImageNet [168], and fully supervised pre-training on ImageNet is effective for handling different downstream tasks such as transfer learning [147] or semantic segmentation. Without further work on fully supervised approaches on ImageNet it is difficult to conclude if the intriguing performance of self-supervised approaches like BeiT [11] is due to the training, e.g. data augmentation, regularization, optimization, or to an underlying mechanism that is capable of learning more general implicit representations.

In this chapter, we propose to revisit the training procedure for vanilla ViT architectures. We investigate how to fully exploit the potential of transformers and discuss the importance of BeiT-like pre-training. Our work builds upon the recent state of the art on fully supervised and self-supervised approaches, with new insights regarding data-augmentation. We propose new training recipes for vision transformers on ImageNet-1k and ImageNet-21k. The whole procedure is called DeiT III.

This chapter is organised as follows: first, we describe the related work. Then, we detail our training strategies with our new training components. In section 6.3, we provide image classification and semantic segmentation experiments.

**Publication.** Chapter 6 is based on the papers “ResNet strikes back: An improved training procedure in timm”, Ross Wightman, Hugo Touvron, Hervé Jégou, *NeurIPS workshop 2021* (see RSB paper [215]) and “DeiT III: Revenge of the ViT”, Hugo Touvron, Matthieu Cord, Hervé Jégou, *ECCV 2022* (see DeiT III paper [194]). The code associated is publicly available at <https://github.com/facebookresearch/deit> and <https://github.com/rwightman/pytorch-image-models>.



## 6.1 Related work

**Training procedures:** The first procedure proposed in the ViT paper [61] was mostly effective for larger models trained on large datasets. In particular the ViT were not competitive with convnets when trained from scratch on ImageNet. Chapter 3 showed that by adapting the training procedure, it is possible to achieve a performance comparable to that of convnets with Imagenet training only. After this Data Efficient Image Transformer procedure (DeiT), only few adaptations have been proposed to improve the training of vision transformers. Steiner et al. [183] published a complete study on how to train vision transformers on different datasets by doing a complete ablation of the different training components. Their results on ImageNet [168] are slightly inferior to those of DeiT but they report improvements on ImageNet-21k compared to Dosovitskiy et al. [61]. The self-supervised approach referred to as masked auto-encoder (MAE) [92] proposes an improved supervised baseline for the larger ViT models.

**BerT pre-training:** In the absence of a strong fully supervised training procedure, BerT [56]-like approaches that train ViT with a self-supervised proxy objective, followed by full finetuning on the target dataset, seem to be the best paradigm to fully exploit the potential of vision transformers. Indeed, BeiT [11] or MAE [92] significantly outperform the fully-supervised approach, especially for the largest models. Nevertheless, to date these approaches have mostly shown their interest in the context of mid-size datasets. For example MAE [92] report its most impressive results when pre-training on ImageNet-1k with a full finetuning on ImageNet-1k. When pre-training on ImageNet-21k and finetuning on ImageNet-1k, BeiT [11] requires a full 90-epochs finetuning on ImageNet-21k followed by another full finetuning on ImageNet-1k to reach its best performance, suggesting that a large labeled dataset is needed so that BeiT realizes its best potential. A recent work suggests that such auto-encoders are mostly interesting in a data starving context [65], but this questions their advantage in the case where more labelled data is actually available.

**Data augmentation:** For supervised training, the community commonly employs data augmentations offered by automatic design procedures such as RandAugment [44] or Auto-Augment [45]. These data augmentations seem to be essential for training vision transformers [193]. Nevertheless, papers like TrivialAugment [142] and Uniform Augment [131] have shown that it is possible to reach interesting performance levels when simplifying the approaches. However, these approaches were initially optimized for convnets. In our work, we propose to go further in this direction and drastically limit and simplify data augmentation: we introduce a data augmentation policy that employs only 3 different transformations randomly drawn with uniform probability. That's it!

## 6.2 Revisit training & pre-training for Vision Transformers

We present our training procedure for vision transformers and compare it with existing approaches. We detail the different ingredients in Table 6.1. Building upon Wightman et al. [215] and Touvron et al. [193], we introduce several changes that have a significant impact on the final model accuracy.

### 6.2.1 Regularization & loss

**Stochastic depth** is a regularization method that is especially useful for training deep networks. We use a uniform drop rate across all layers and adapt it according to the model size [197]. Table 6.14 (6.3.6) gives the stochastic depth drop-rate per model.

Table 6.1 – Summary of our training procedures with ImageNet-1k and ImageNet-21k. We also provide DeiT [193], Wightman et al [215] and Steiner et al. [183] baselines for reference. Adapt. means the hparams is adapted to the size of the model. For finetuning to higher resolution with model pre-trained on ImageNet-1k only we use the finetuning procedure from Chapter 3 see section 6.3.6 for more details.

Procedure → Reference	Previous approaches				Ours		
	ViT [61]	Steiner et al. [183]	DeiT [193]	Wightman et al. [215]	ImNet-1k	ImNet-21k Pretrain. Finetune.	
Batch size	4096	4096	1024	2048	2048	2048	2048
Optimizer	AdamW	AdamW	AdamW	LAMB	LAMB	LAMB	LAMB
LR	$3.10^{-3}$	$3.10^{-3}$	$1.10^{-3}$	$5.10^{-3}$	$3.10^{-3}$	$3.10^{-3}$	$3.10^{-4}$
LR decay	cosine	cosine	cosine	cosine	cosine	cosine	cosine
Weight decay	0.1	0.3	0.05	0.02	0.02	0.02	0.02
Warmup epochs	3-4	3-4	5	5	5	5	5
Label smoothing $\epsilon$	0.1	0.1	0.1	×	×	0.1	0.1
Dropout	✓	✓	×	×	×	×	×
Stoch. Depth	×	✓	✓	✓	✓	✓	✓
Repeated Aug	×	×	✓	✓	✓	×	×
Gradient Clip.	1.0	1.0	×	1.0	1.0	1.0	1.0
H. flip	✓	✓	✓	✓	✓	✓	✓
RRC	✓	✓	✓	✓	✓	×	×
Rand Augment	×	Adapt.	9/0.5	7/0.5	×	×	×
3 Augment (ours)	×	×	×	×	✓	✓	✓
LayerScale	×	×	×	×	✓	✓	✓
Mixup alpha	×	Adapt.	0.8	0.2	0.8	×	×
Cutmix alpha	×	×	1.0	1.0	1.0	1.0	1.0
Erasing prob.	×	×	0.25	×	×	×	×
ColorJitter	×	×	×	×	0.3	0.3	0.3
Test crop ratio	0.875	0.875	0.875	0.95	1.0	1.0	1.0
Loss	CE	CE	CE	BCE	BCE	CE	CE

**LayerScale.** We use LayerScale [197] introduced in Chapter 3. As previously mentioned, this method was introduced to facilitate the convergence of deep transformers. With our training procedure, we do not have convergence problems, however we observe that LayerScale allows our models to attain a higher accuracy for the largest models. In Chapter 3, the initialization of LayerScale is adapted according to the depth. In order to simplify the method we use the same initialization ( $10^{-4}$ ) for all our models.

**Binary Cross entropy.** Wightman et al. [215] adopt a binary cross-entropy (BCE) loss instead of the more common cross-entropy (CE) to train ResNet-50. They conclude that the gains are limited compared to the CE loss but that this choice is more convenient when employed with Mixup [242] and CutMix [237]. For larger ViTs and with our training procedure on ImageNet-1k, the BCE loss provides us a significant improvement in performance, see an ablation in Table 6.4. We did not achieve compelling results during our exploration phase on Imagenet21k, and therefore keep CE when pre-training with this dataset as well as for the subsequent fine-tuning.

**The optimizer** is LAMB [234], a derivative of AdamW [136]. It includes gradient clipping by default in Apex’s [1] implementation.

## 6.2.2 Data-augmentation

Since the advent of AlexNet, there has been significant modifications to the data-augmentation procedures employed to train neural networks. Interestingly, the same data augmentation, like

ColorJitter	Data-Augmentation			ImageNet-1k		
	Grayscale	Gaussian Blur	Solarization	Val	Real	V2
0.3	✗	✗	✗	81.4	86.1	70.3
0.3	✓	✗	✗	81.0	86.0	69.7
0.3	✓	✓	✗	82.7	87.6	72.7
0.3	✓	✓	✓	<b>83.1</b>	<b>87.7</b>	72.6
0.0	✓	✓	✓	<b>83.1</b>	<b>87.7</b>	72.0

Table 6.2 – Ablation of the components of our data-augmentation strategy with ViT-B on ImageNet-1k.

RandAugment [44], is widely employed for ViT while their policy was initially learned for convnets. Given that the architectural priors and biases are quite different in these architectures, the augmentation policy may not be adapted, and possibly overfitted considering the large amount of choices involved in their selection. We therefore revisit this prior choice.

**3-Augment:** We propose a simple data augmentation inspired by what is used in self-supervised learning (SSL). We consider the following transformations:

- Grayscale: This favors color invariance and give more focus on shapes.
- Solarization: This adds strong noise on the colour to be more robust to the variation of colour intensity and so focus more on shape.
- Gaussian Blur: In order to slightly alter details in the image.

For each image, we select only one of these data-augmentations with a uniform probability over 3 different ones. In addition to these 3 augmentation choices, we include the common color-jitter and horizontal flip. Figure 6.1 illustrates the different augmentations used in our 3-Augment approach. In Table 6.2 we provide an ablation on our different data-augmentation components.

### 6.2.3 Cropping

**Random Resized Crop (RRC)** was introduced in the GoogleNet [184] paper. It serves as a regularisation to limit model overfitting, while favoring that the decision done by the model is invariant to a certain class of transformations. This data augmentation was deemed important on Imagenet1k to prevent overfitting, which happens to occur rapidly with modern large models.

This cropping strategy however introduces some discrepancy between train and test images in terms of the aspect ratio and the apparent size of objects [199]. Since ImageNet-21k includes significantly more images, it is less prone to overfitting. Therefore we question whether the benefit of the strong RRC regularization compensates for its drawback when training on larger sets.

**Simple Random Crop (SRC)** is a much simpler way to extract crops. It is similar to the original cropping choice proposed in AlexNet [123]: We resize the image such that the smallest side matches the training resolution. Then we apply a reflect padding of 4 pixels on all sides, and finally we apply a square Crop of training size randomly selected along the x-axis of the image.

Figure 6.2 visualizes cropping boxes sampled for RRC and SRC. RRC provides a lot of diversity and very different sizes for crops. In contrast SRC covers a much larger fraction of the image overall and preserve the aspect ratio, but offers less diversity: The crops overlaps significantly. As a result, when training on ImageNet-1k the performance is better with the commonly used RRC. For instance a ViT-S reduces its top-1 accuracy by  $-0.9\%$  if we do not use RRC.

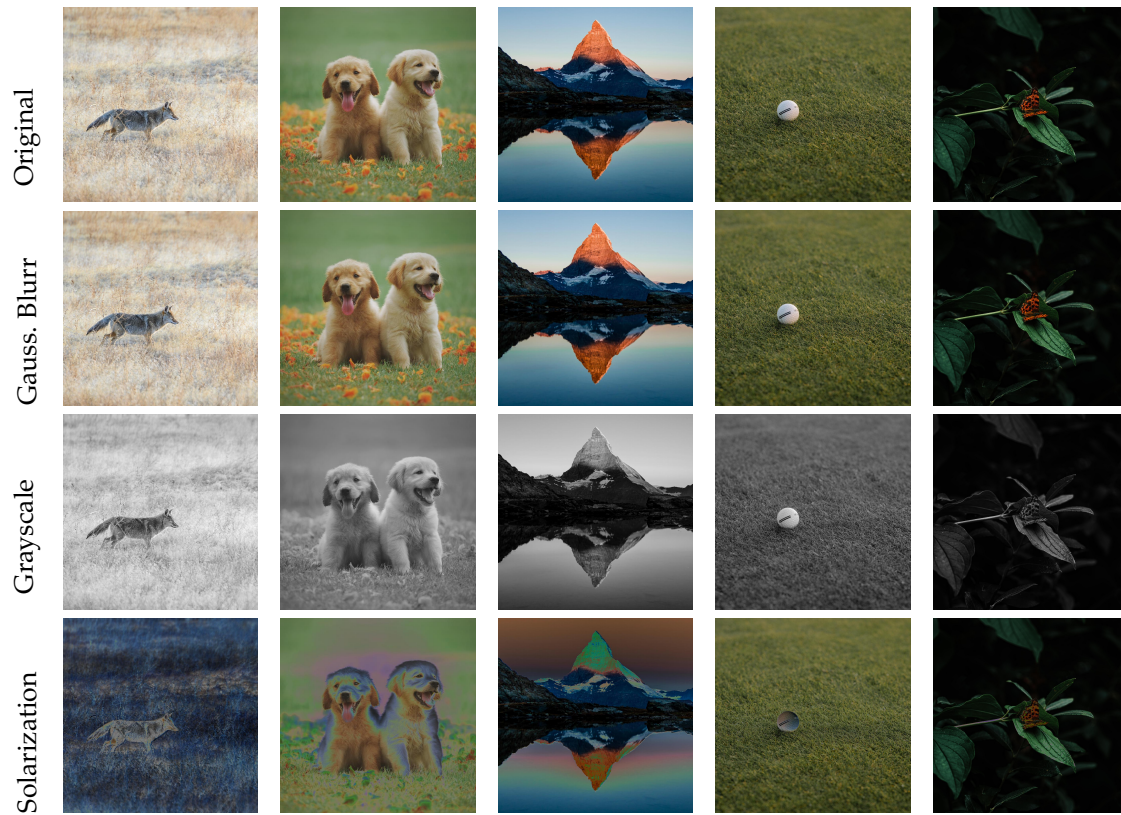


Figure 6.1 – Illustration of the 3 types of data-augmentations used in 3-Augment.



Figure 6.2 – Example of crops selected by two strategies: Resized Crop and Simple Random Crop.



Figure 6.3 – Illustration of Random Resized Crop (RRC) and Simple Random Crop (SRC). The usual RRC is a more aggressive data-augmentation than SRC: It has a more important regularizing effect and avoids overfitting by giving more variability to the images. At the same time it introduces a discrepancy of scale and aspect-ratio. It also leads to labeling errors, for instance when the object is not in the cropped region (e.g., train or boat). On Imagenet1k this regularization is overall regarded as beneficial. However our experiments show that it is detrimental on ImageNet-21k, which is less prone to overfitting.

However, in the case of ImageNet-21k ( $\times 10$  bigger than ImageNet-1k), there is less risk of overfitting and increasing the regularisation and diversity offered by RRC is less important. In this context, SRC offers the advantage of reducing the discrepancy in apparent size and aspect ratio. More importantly, it gives a higher chance that the actual label of the image matches that of the crop: RRC is relatively aggressive in terms of cropping and in many cases the labelled object is not even present in the crop, as shown in Figure 6.3 where some of the crops do not contain the labelled object. For instance, with RRC there is a crop no zebra in the left example, or no train in three of the crops from the middle example. This is more unlikely to happen with SRC, which covers a much larger fraction of the image pixels. In Table 6.5 we provide an ablation of random resized crop on ImageNet-21k, where we see that these observations translate as a significant gain in performance.

## 6.3 Experiments

This section includes multiple experiments in image classification, with a special emphasis on ImageNet-1k [54, 159, 168]. We also report results for downstream tasks in fine-grained classification and segmentation. We include a large number of ablations to better analyze different effects, such as the importance of the training resolution and longer training schedules. We provide additional results in the appendices.

### 6.3.1 Baselines and default settings

The main task that we consider in this chapter for the evaluation of our training procedure is image classification. We train on ImageNet-1k train and evaluate on ImageNet-1k val, with results on ImageNet-V2 to control overfitting. We also consider the case where we can pretrain on ImageNet-21k, Finally, we report transfer learning results on 6 different datasets/benchmarks.

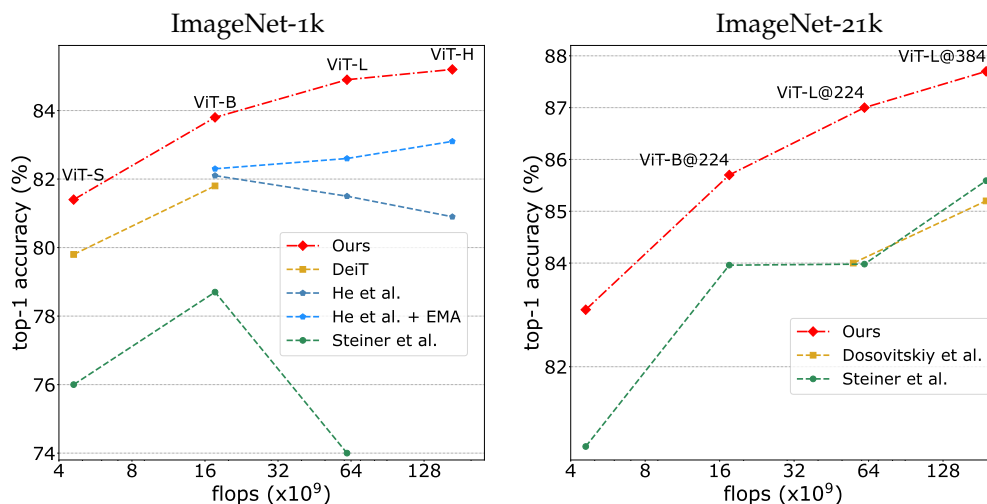


Figure 6.4 – Comparison of training recipes for (left) vanilla vision transformers trained on ImageNet-1k and evaluated at resolution  $224 \times 224$ , and (right) pre-trained on ImageNet-21k at  $224 \times 224$  and finetuned on ImageNet-1k at resolution  $224 \times 224$  or  $384 \times 384$ .

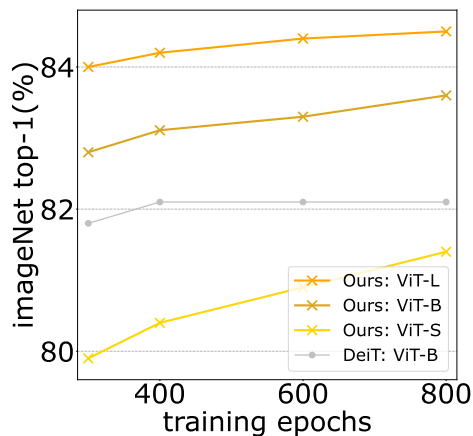


Figure 6.5 – Top-1 accuracy on ImageNet-1k only at resolution  $224 \times 224$  with our training recipes and a different number of epochs

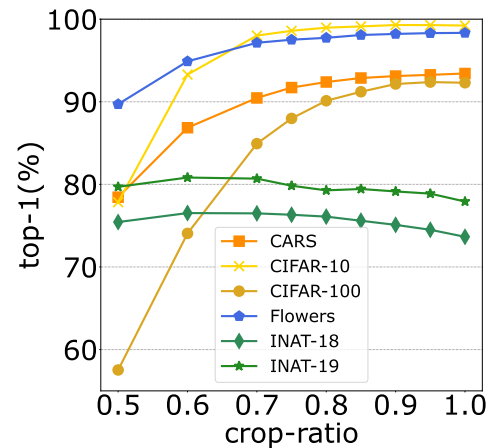


Figure 6.6 – Transfer learning performance on 6 datasets with different test-time crop ratio. ViT-B pre-trained on ImageNet-1k at resolution  $224$ .

**Default setting.** When training on ImageNet-1k only, by default we train during 400 epochs with a batch size 2048, following prior works [197, 224]. Unless specified otherwise, both the training and evaluation are carried out at resolution  $224 \times 224$  (even though we recommend to train at a lower resolution when targeting  $224 \times 224$  at inference time).

When pre-training on ImageNet-21k, we pre-train by default during 90 epochs at resolution  $224 \times 224$ , followed by a finetuning of 50 epochs on ImageNet-1k. In this context, we consider two fine-tuning resolutions:  $224 \times 224$  and  $384 \times 384$ .

## 6.3.2 Ablations

### 6.3.2.1 Impact of training duration

In Figure 6.5 we provide an ablation on the number of epochs, which shows that ViT models do not saturate as rapidly as the Chapter 3 training procedure [193] when we increase the number of epochs beyond the 400 epochs adopted for our baseline.

For ImageNet-21k pre-training, we use 90 epochs for pre-training as in a few works [134, 195]. We finetune during 50 epochs on ImageNet-1k [195] and marginally adapt the stochastic depth parameter. We point out that this choice is mostly for the sake of consistency across models: we observe that training 30 epochs also provides similar results.

### 6.3.2.2 Data Augmentation

In Table 6.3 we compare our handcrafted data augmentation 3-Augment with existing learned augmentation methods. With the ViT architecture, our data augmentation is the most effective while being simpler than the other approaches. Since previous augmentations were introduced on convnets, we also provide results for a ResNet-50. In this case, previous augmentation policies have similar (RandAugment, Trivial-Augment) or better results (Auto-Augment) on the validation set. This is no longer the case when evaluating on the independent set V2, for which the Auto-Augment better accuracy is not significant.

Method	Learned augm. methods	# Nb of DA	Model	ImageNet-1k		
				Val	Real	V2
Auto-Augment [45]	✓	14	ResNet50	79.7	85.6	67.9
			ViT-B	82.8	87.5	71.9
			ViT-L	84.0	<b>88.6</b>	74.0
RandAugment [44]	✓	14	ResNet50	79.5	85.5	67.6
			ViT-B	82.7	87.4	72.2
			ViT-L	84.0	88.3	73.8
Trivial-Augment [142]	✗	14	ResNet50	79.5	85.4	67.6
			ViT-B	82.3	87.0	71.2
			ViT-L	83.6	88.1	73.7
3-Augment (Ours)	✗	3	ResNet50	79.4	85.5	67.8
			ViT-B	<b>83.1</b>	<b>87.7</b>	<b>72.6</b>
			ViT-L	<b>84.2</b>	<b>88.6</b>	<b>74.3</b>

Table 6.3 – Comparison of some existing data augmentation methods with our simple 3-Augment proposal inspired by data augmentation used with self-supervised learning.

Model	Loss	LayerScale	Data Aug.	Epochs	ImageNet-1k		
					val	real	v2
ViT-S	CE	✗	RandAugment	300	79.8	85.3	68.1
	BCE	✗	RandAugment	300	79.8	85.9	68.2
	BCE	✓	RandAugment	300	80.1	<b>86.1</b>	69.1
	BCE	✓	RandAugment	400	<b>80.7</b>	86.0	69.3
	BCE	✓	3-Augment	400	80.4	<b>86.1</b>	<b>69.7</b>
ViT-B	CE	✗	RandAugment	300	80.9	85.5	68.5
	BCE	✗	RandAugment	300	82.2	87.2	71.4
	BCE	✓	RandAugment	300	82.5	87.5	71.4
	BCE	✓	RandAugment	400	82.7	87.4	72.2
	BCE	✓	3-Augment	400	<b>83.1</b>	<b>87.7</b>	<b>72.6</b>
ViT-L	BCE	✗	RandAugment	300	83.0	87.9	72.4
	BCE	✗	RandAugment	400	83.3	87.7	72.5
	BCE	✓	RandAugment	400	84.0	88.3	73.8
	BCE	✓	3-Augment	400	<b>84.2</b>	<b>88.6</b>	<b>74.3</b>

Table 6.4 – Ablation on different training component with training at resolution  $224 \times 224$  on ImageNet-1k. We perform ablations with ViT-S, ViT-B and ViT-L. We report top-1 accuracy (%) on ImageNet validation set, ImageNet real and ImageNet v2.

Crop.	LS	Mixup	Aug. policy	#Imnet21k epochs	finetuning resolution	Imagenet-1k val top-1			Imagenet-1k v2 top-1		
						ViT-S	ViT-B	ViT-L	ViT-S	ViT-B	ViT-L
RRC	✗	0.8	RA	90	224 <sup>2</sup>	81.6	84.6	86.0	70.7	74.7	76.4
SRC	✗	0.8	RA	90	224 <sup>2</sup>	82.1	84.8	86.3	71.8	75.0	76.7
SRC	✓	0.8	RA	90	224 <sup>2</sup>	82.4	85.0	86.4	72.4	75.7	77.4
SRC	✓	✗	RA	90	224 <sup>2</sup>	82.3	85.1	86.5	72.4	75.6	77.2
SRC	✓	✗	3A	90	224 <sup>2</sup>	82.6	85.2	86.8	72.6	76.1	78.3
SRC	✓	✗	3A	240	224 <sup>2</sup>	<b>83.1</b>	<b>85.7</b>	<b>87.0</b>	<b>73.8</b>	<b>76.5</b>	<b>78.6</b>
SRC	✓	✗	3A	240	384 <sup>2</sup>	84.8	86.7	87.7	75.1	77.9	79.1

Table 6.5 – Ablation path: **augmentation and regularization** with ImageNet-21k pre-training (at resolution  $224 \times 224$ ) and ImageNet-1k fine-tuning. We measure the impact of changing Random Resize Crop (RRC) to Simple Random Crop (SRC), adding LayerScale (LS), removing Mixup, replacing RandAugment (RA) by 3-Augment (3A), and finally employing a longer number of epochs during the pre-training phase on ImageNet-21k. All experiments are done with Seed 0 with fixed hparams except the drop-path rate of stochastic depth, which depends on the model and is increased by 0.05 for the longer pre-training. We report 2 digits top-1 accuracy but note that the standard standard deviation is around 0.1 on our ViT-B baseline. Note that all these changes are neutral w.r.t. complexity except in the last row, where the fine-tuning at resolution  $384 \times 384$  significantly increases the complexity.



Model	epochs		Resolution		ImageNet top-1 acc		
	Train.	FT	Train.	FT	val	real	v2
ViT-B	400	20	128 × 128	224 × 224	83.2	<b>88.1</b>	<u>73.2</u>
		–	160 × 160		<u>83.3</u>	<u>88.0</u>	<b>73.4</b>
			192 × 192		<b>83.5</b>	<u>88.0</u>	72.8
	800	–	224 × 224	83.1	87.7	72.6	
			128 × 128	83.5	<b>88.3</b>	73.4	
		20	160 × 160	83.6	<u>88.2</u>	<u>73.5</u>	
ViT-L	400	20	128 × 128	224 × 224	83.9	<b>88.8</b>	<u>74.3</u>
		–	160 × 160		<u>84.4</u>	<b>88.8</b>	<u>74.3</u>
			192 × 192		<b>84.5</b>	<b>88.8</b>	<b>75.1</b>
	800	–	224 × 224	84.2	88.6	<u>74.3</u>	
			128 × 128	84.5	<b>88.9</b>	74.7	
		20	160 × 160	<u>84.7</u>	<b>88.9</b>	<b>75.2</b>	
ViT-H	400	20	126 × 126	224 × 224	84.7	<u>89.2</u>	75.2
		–	154 × 154		<b>85.1</b>	<b>89.3</b>	<u>75.3</u>
			182 × 182		<b>85.1</b>	89.2	<b>75.4</b>
	800	–	224 × 224	84.8	89.1	<u>75.3</u>	
			126 × 126	<u>85.1</u>	<b>89.2</b>	75.6	
		20	154 × 154	<b>85.2</b>	<b>89.2</b>	<b>75.9</b>	
ViT-H-52	400	20	126 × 126	224 × 224	<u>85.1</u>	88.9	<b>75.9</b>
		–	224 × 224	84.9	89.1	75.6	
ViT-H-26×2	400	20	126 × 126	224 × 224	84.9	89.2	75.6
ViT-H-26×2	400	20	126 × 126	224 × 224	84.9	89.1	75.3

Table 6.6 – We compare ViT architectures pre-trained on ImageNet-1k only with different training resolution followed by a fine-tuning at resolution  $224 \times 224$ . We benefit from the FixRes effect [199] and get better performance with a lower training resolution (e.g resolution  $160 \times 160$  with patch size 16 represent 100 tokens vs 196 for  $224 \times 224$ . This represents a reduction of 50% of the number of tokens).

### 6.3.2.3 Impact of training resolution

In Table 6.6 we report the evolution of the performance according to the training resolution. We observe that we benefit from the FixRes [199] effect. By training at resolution  $192 \times 192$  (or  $160 \times 160$ ) we get a better performance at 224 after a slight fine-tuning than when training from scratch at  $224 \times 224$ .

We observe that the resolution has a regularization effect. While it is known that it is best to use a smaller resolution at training time [199], we also observe with the training curves that using smaller resolution reduces the overfitting of the larger models. This is also illustrated by our results Table 6.6 with ViT-H and ViT-L. This is especially important with longer training, where models overfit without a stronger regularisation. This smaller resolution implies that there are less patches to be processed, and therefore it reduces the training cost and increases the performance. In that respect its effect is comparable to that of MAE [92]. We also report results with ViT-H 52 layers and ViT-H 26 layers parallel [196] models with 1B parameters. Due to the lower resolution training it is easier to train these models.

#### 6.3.2.4 Comparison with previous training recipes for ViT

In Figure 6.4, we compare training procedures used to pre-train the ViT architecture either on ImageNet-1k and ImageNet-21k. Our procedure outperforms existing recipes with a large margin. For instance, with ImageNet-21k pre-training we have an improvement of +3.0% with ViT-L in comparison to the best approach. Similarly, when training from scratch on ImageNet-1k we improve the accuracy by +2.1% for ViT-H compared to the previous best approach, and by +4.3% with the best approach that does not use EMA. See also detailed results in our appendices.

### 6.3.3 Image Classification

**ImageNet-1k.** In Table 6.7 we compare ViT architectures trained with our training recipes on ImageNet-1k with other architectures. We include a comparison with the recent SwinTransformers [134] and ConvNeXts [135].

**ImageNet-21k.** In Table 6.8 we compare ViT architecture pre-trained on ImageNet-21k with our training recipe then finetuned on ImageNet-1k. We can observe that the findings are similar to what we obtained on ImageNet-1k only.

**Comparison with Bert-like pre-training.** In Table 6.9 we compare ViT models trained with our training recipes with ViT trained with different Bert-like approaches. We observe that for an equivalent number of epochs our approach gives comparable performance on ImageNet-1k and better on ImageNet-v2 as well as in segmentation on Ade. For Bert like pre-training we compare our method with MAE [92] and BeiT [11] because they remain relatively simple approaches with very good performance. As our approach does not use distillation or multi-crops we have not made a comparison with approaches such as PeCo [60] which use an auxiliary model as a psycho-visual loss and iBoT [249], which uses multi-crop and an exponential moving average of the model.

#### 6.3.4 Significance of measurements

In this subsection we study first the Significance of measurements with ResNet-50 architectures and A2 training strategy from Wigthman et al. [215] We then extend this study to the ViT architecture with our new training procedure.

**Seed experiments** For a fixed set of choices and hyper-parameters, there is some inherent variability on the performance due to the presence of random factors in several stages. It is the case for the weight initialization, but also for the optimization procedure itself. For instance the order in which the images are fed to the network through batches depends on a random generator. This variability raises the question of the significance of accuracy measurements. For this purpose, we measure the distribution of performance when changing the random generator choices. This is conveniently done by changing the seed, as previously done by Picard [152], who concludes to the existence of outliers significantly outperforming or underperforming the average outcome of a training procedure. In Figure 6.7, we report several statistics on the performance with the A2 training [215] procedure and ResNet50 architecture when considering 100 distinct seeds (from 1 to 100, note that we have used seed=0 in all other experiments). In these experiments, we focus on the performance reached at the end of the training: we do not select the maximum obtained by intermediate checkpoints in the last epochs. This would have a similar effect as a seed selection, but the measures would not be IID and less disentangled from the training duration itself.

Table 6.7 – **Classification with ImageNet-1k training.** We compare architectures with comparable FLOPs and number of parameters. All models are trained on ImageNet1k only without distillation nor self-supervised pre-training. We report Top-1 accuracy on the validation set of ImageNet1k and ImageNet-V2 with different measure of complexity: throughput, FLOPs, number of parameters and peak memory usage. The throughput and peak memory are measured on a single V100-32GB GPU with batch size fixed to 256 and mixed precision. For ResNet [94] and RegNet [156] we report the improved results from Wightman et al. [215]. Note that different models may have received a different optimization effort.  $\uparrow$ R indicates that the model is fine-tuned at the resolution  $R$  and -R indicates that the model is trained at resolution  $R$ .

Architecture	nb params ( $\times 10^6$ )	throughput (im/s)	FLOPs ( $\times 10^9$ )	Peak Mem (MB)	Top-1 Acc.	V2 Acc.
<b>“Traditional” ConvNets</b>						
ResNet-50 [94, 215]	25.6	2587	4.1	2182	80.4	68.7
ResNet-101 [94, 215]	44.5	1586	7.9	2269	81.5	70.3
ResNet-152 [94, 215]	60.2	1122	11.6	2359	82.0	70.6
RegNetY-4GF [156, 215]	20.6	1779	4.0	3041	81.5	70.7
RegNetY-8GF [156, 215]	39.2	1158	8.0	3939	82.2	71.1
RegNetY-16GF [156, 193]	83.6	714	16.0	5204	82.9	72.4
EfficientNet-B4 [187]	19.0	573	4.2	10006	82.9	72.3
EfficientNet-B5 [187]	30.0	268	9.9	11046	83.6	73.6
EfficientNetV2-S [188]	21.5	874	8.5	4515	83.9	74.0
EfficientNetV2-M [188]	54.1	312	25.0	7127	85.1	75.5
EfficientNetV2-L [188]	118.5	179	53.0	9540	85.7	76.3
<b>Vision Transformers derivative</b>						
PiT-S-224 [99]	23.5	1809	2.9	3293	80.9	–
PiT-B-224 [99]	73.8	615	12.5	7564	82.0	–
Swin-T-224 [134]	28.3	1109	4.5	3345	81.3	69.5
Swin-S-224 [134]	49.6	718	8.7	3470	83.0	71.8
Swin-B-224 [134]	87.8	532	15.4	4695	83.5	–
Swin-B-384 [134]	87.9	160	47.2	19385	84.5	–
<b>Vision MLP &amp; Patch-based ConvNets</b>						
Mixer-B/16 [191]	59.9	993	12.6	1448	76.4	63.2
ResMLP-B24 [192]	116.0	1120	23.0	930	81.0	69.0
PatchConvNet-S60-224 [195]	25.2	1125	4.0	1321	82.1	71.0
PatchConvNet-B60-224 [195]	99.4	541	15.8	2790	83.5	72.6
PatchConvNet-B120-224 [195]	188.6	280	29.9	3314	84.1	73.9
ConvNeXt-B-224 [135]	88.6	563	15.4	3029	83.8	73.4
ConvNeXt-B-384 [135]	88.6	190	45.0	7851	85.1	74.7
ConvNeXt-L-224 [135]	197.8	344	34.4	4865	84.3	74.0
ConvNeXt-L-384 [135]	197.8	115	101.0	11938	85.5	75.3
<b>Our Vanilla Vision Transformers</b>						
ViT-S	22.0	1891	4.6	987	81.4	70.5
ViT-S $\uparrow$ 384	22.0	424	15.5	4569	83.4	73.1
ViT-B	86.6	831	17.5	2078	83.8	73.6
ViT-B $\uparrow$ 384	86.9	190	55.5	8956	85.0	74.8
ViT-L	304.4	277	61.6	3789	84.9	75.1
ViT-L $\uparrow$ 384	304.8	67	191.2	12866	85.8	76.7
ViT-H	632.1	112	167.4	6984	85.2	75.9

Table 6.8 – **Classification with ImageNet-21k training.** We compare architectures with comparable FLOPs and number of parameters. All models are trained on ImageNet-21k without distillation nor self-supervised pre-training. We report Top-1 accuracy on the validation set of ImageNet-1k and ImageNet-V2 with different measure of complexity: throughput, FLOPs, number of parameters and peak memory usage. The throughput and peak memory are measured on a single V100-32GB GPU with batch size fixed to 256 and mixed precision. For Swin-L we decrease the batch size to 128 in order to avoid out of memory error and re-estimate the memory consumption.  $\uparrow R$  indicates that the model is fine-tuned at the resolution  $R$ .

Architecture	nb params ( $\times 10^6$ )	throughput (im/s)	FLOPs ( $\times 10^9$ )	Peak Mem (MB)	Top-1 Acc.	V2 Acc.
<b>“Traditional” ConvNets</b>						
R-101x3 $\uparrow$ 384 [120]	388	–	204.6	–	84.4	–
R-152x4 $\uparrow$ 480 [120]	937	–	840.5	–	85.4	–
EfficientNetV2-S $\uparrow$ 384 [188]	21.5	874	8.5	4515	84.9	74.5
EfficientNetV2-M $\uparrow$ 480 [188]	54.1	312	25.0	7127	86.2	75.9
EfficientNetV2-L $\uparrow$ 480 [188]	118.5	179	53.0	9540	86.8	76.9
EfficientNetV2-XL $\uparrow$ 512 [188]	208.1	–	94.0	–	87.3	77.0
<b>Patch-based ConvNets</b>						
ConvNeXt-B [135]	88.6	563	15.4	3029	85.8	75.6
ConvNeXt-B $\uparrow$ 384 [135]	88.6	190	45.1	7851	86.8	76.6
ConvNeXt-L [135]	197.8	344	34.4	4865	86.6	76.6
ConvNeXt-L $\uparrow$ 384 [135]	197.8	115	101	11938	87.5	77.7
ConvNeXt-XL [135]	350.2	241	60.9	6951	87.0	77.0
ConvNeXt-XL $\uparrow$ 384 [135]	350.2	80	179.0	16260	87.8	77.7
<b>Vision Transformers derivative</b>						
Swin-B [134]	87.8	532	15.4	4695	85.2	74.6
Swin-B $\uparrow$ 384 [134]	87.9	160	47.0	19385	86.4	76.3
Swin-L [134]	196.5	337	34.5	7350	86.3	76.3
Swin-L $\uparrow$ 384 [134]	196.7	100	103.9	33456	87.3	77.0
<b>Vanilla Vision Transformers</b>						
ViT-B/16 [183]	86.6	831	17.6	2078	84.0	–
ViT-B/16 $\uparrow$ 384 [183]	86.7	190	55.5	8956	85.5	–
ViT-L/16 [183]	304.4	277	61.6	3789	84.0	–
ViT-L/16 $\uparrow$ 384 [183]	304.8	67	191.1	12866	85.5	–
<b>Our Vanilla Vision Transformers</b>						
ViT-S	22.0	1891	4.6	987	83.1	73.8
ViT-B	86.6	831	17.6	2078	85.7	76.5
ViT-B $\uparrow$ 384	86.9	190	55.5	8956	86.7	77.9
ViT-L	304.4	277	61.6	3789	87.0	78.6
ViT-L $\uparrow$ 384	304.8	67	191.2	12866	87.7	79.1
ViT-H	632.1	112	167.4	6984	87.2	79.2

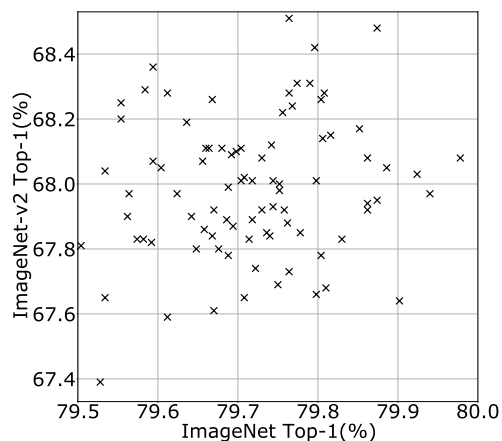
Pretrained data	Model	Method	# pre-training epochs	# finetuning epochs	ImageNet				
					Val	Real	V2		
INET-1k	ViT-B	BeiT	300	100 <sup>(1k)</sup>	82.9	-	-		
			800	100 <sup>(1k)</sup>	83.2	-	-		
		MAE*	1600	100 <sup>(1k)</sup>	<u>83.6</u>	<u>88.1</u>	<u>73.2</u>		
		Ours	400 <sup>(1k)</sup> 800 <sup>(1k)</sup>	20 <sup>(1k)</sup> 20 <sup>(1k)</sup>	83.5 <b>83.8</b>	88.0 <b>88.2</b>	72.8 <b>73.6</b>		
	ViT-L	BeiT	800	30 <sup>(1k)</sup>	<u>85.2</u>	-	-		
			MAE	400 800 1600	50 <sup>(1k)</sup> 50 <sup>(1k)</sup> 50 <sup>(1k)</sup>	84.3 84.9 85.1	- - -	- - -	
		MAE*	1600	50 <sup>(1k)</sup>	<b>85.9</b>	<b>89.4</b>	<b>76.5</b>		
		Ours	400 <sup>(1k)</sup> 800 <sup>(1k)</sup>	20 <sup>(1k)</sup> 20 <sup>(1k)</sup>	84.5 84.9	<u>88.8</u> 88.7	<u>75.1</u> <u>75.1</u>		
		INET-21k	ViT-B	BeiT	150	50 <sup>(1k)</sup>	83.7	88.2	73.1
					150 + 90 <sup>(21k)</sup>	50 <sup>(1k)</sup>	<u>85.2</u>	<u>89.4</u>	<u>75.4</u>
Ours	90 <sup>(21k)</sup> 240 <sup>(21k)</sup>			50 <sup>(1k)</sup> 50 <sup>(1k)</sup>	85.2 <b>85.7</b>	89.4 <b>89.5</b>	76.1 <b>76.5</b>		
ViT-L	BeiT		150	50 <sup>(1k)</sup>	86.0	89.6	76.7		
			150 + 90 <sup>(21k)</sup>	50 <sup>(1k)</sup>	<b>87.5</b>	<b>90.1</b>	<b>78.8</b>		
	Ours		90 <sup>(21k)</sup> 240 <sup>(21k)</sup>	50 <sup>(1k)</sup> 50 <sup>(1k)</sup>	86.8 <u>87.0</u>	89.9 <u>90.0</u>	78.3 <u>78.6</u>		

Table 6.9 – Comparison of self-supervised pre-training with our approach. As our approach is fully supervised, this table is given as an indication. All models are evaluated at resolution  $224 \times 224$ . We report Image classification results on ImageNet val, real and v2 in order to evaluate overfitting. <sup>(21k)</sup> indicate a finetuning with labels on ImageNet-21k and <sup>(1k)</sup> indicate a finetuning with labels on ImageNet-1k. \* design the improved setting of MAE using pixel (w/ norm) loss.

The standard deviation is typically around 0.1 on ImageNet-val, see Figure 6.7. This concurs with statistics reported in the literature for ResNet and other convnets [156]. The variance is higher on ImageNet-V2 (std=0.23), which consists of a smaller set (10000 vs 50000 for -val) of images not present in the validation set. The mean 79.72% shows that our main weights (seed 0) overestimates the average performance by about +0.13%.

**Peak performance and control of overfitting** To prevent to over-estimate too much the accuracy on validation, during our exploration process we have selected only the final checkpoint and we use relatively coarse grid for hyper-parameters search to prevent introducing an additional seed effect. However optimizing over a large number of choices typically leads to overfitting. In Figure 6.7, we observe that the maximum (or peak performance) is close to 80.0% with the A2 training procedure. Note, Figure 6.8 provides the distribution of accuracy as an histogram;

One question is whether this model is intrinsically better than the average ones, or if it was just lucky on this particular measurement set. To attempt to answer this question, we measure how the performance transfers to another measurement dataset: we compute for all the seeds the couples (ImageNet-val top-1 acc., ImageNet-V2 top-1 acc.), and plot them as a point cloud in Figure 6.7. We observe that the correlation between the performance on ImageNet-val and -V2 is limited. Noticeably the best performance is not achieved by the same seed on the two datasets. This observation suggests some significant measurement noise, which advocates to report systematically the performance on different datasets, and more particularly one making a clear distinction between validation and test.



dataset ↓	Top-1 accuracy (%)				
	mean	std	max	min	seed 0
ImageNet-val	79.72	0.10	79.98	79.50	79.85
ImageNet-real	85.37	0.08	85.55	85.21	85.45
ImageNet-V2	67.99	0.23	68.69	67.39	67.90

Figure 6.7 – Top ↑: Statistics for ResNet-50 trained with A2 and 100 different seeds. The column "seed 0" corresponds to the weights that we take as reference. Its performance is +0.13% above the average top-1 accuracy on Imagenet-val.

← Left: Point cloud plotting the ImageNet-val top-1 accuracy vs ImageNet-V2 for all seeds. Note that the outlying seed that achieves 68.5% top-1 accuracy on ImageNet-V2 has an average performance on ImageNet-val.

**Variability along epochs and discussion on early stopping.** Figure 6.9 shows how the performance variability evolves along epochs, where we observe the variance of the score is very high until the last 100 epochs. In Figure 6.10, we additionally measure the performance early in the training and compare it to the final performance. It is only towards the end of the training that one can determine the most interesting seeds. We conclude that we can not apply an early stopping rule based on early results.

**Comparing architectures and training procedures: a show-case of contradictory conclusions**  
 In this paragraph we case how difficult it is to compare two architectures, even under the same training procedure, or conversely how it is difficult to compare different procedures with a single architecture. We choose ResNet-50 and DeiT-S. The latter [193] is essentially a ViT parameterized so that it has approximately the same number of parameters as a ResNet-50. For each architecture, we have put a significant effort in optimizing the procedure to maximize the performance on Imagenet-val with the same 300 epochs training schedule and same batch size. Under this constraint, the best training procedure that we have designed for ResNet-50 is A2. We denote by T2 the corresponding training procedure for DeiT-S. Note that this training procedure achieves a significantly better performance on Imagenet-val than the one initially proposed for DeiT-S (80.4% versus 79.8% in the original paper).

		test set →		ImageNet-v2	
		ImageNet-val		ImageNet-v2	
↓ architecture	training →	A2	T2	A2	T2
ResNet-50		79.9	79.2	67.9	67.9
DeiT-S		79.6	80.4	68.1	69.2

As one can see, by choosing the procedure optimized for any of the two architectures, one may conclude that this architecture is better based on ImageNet-val accuracy: with A2 training, ResNet50 is better than DeiT-S, with T2 training, DeiT-S is better than ResNet50. The measurements on ImageNet-v2 would lead to a different conclusion, as DeiT-S is better for both procedure. But even in that case, by focusing on A2 one may conclude that the difference between ResNet-50 and DeiT-S with A2 training is not statistically significant: 67.9% vs 68.1%. Conversely, if the goal is to compare A2 to T2, we could draw different conclusions on ImageNet-val if considering a single architecture. This highlights the difficulty of comparing two architectures in a fair way. The training procedure has interactions with the choice of architecture. It is completely possible to overfit an architecture on a training procedure.

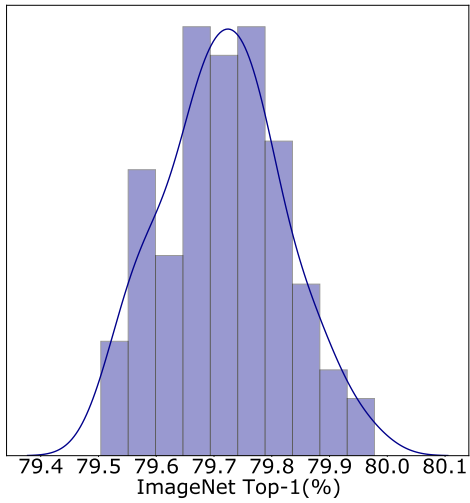


Figure 6.8 – Distribution of the performance on ImageNet-val with the A2 procedure. It is measured with 100 different seeds. We also depict the Gaussian-fit of this distribution.

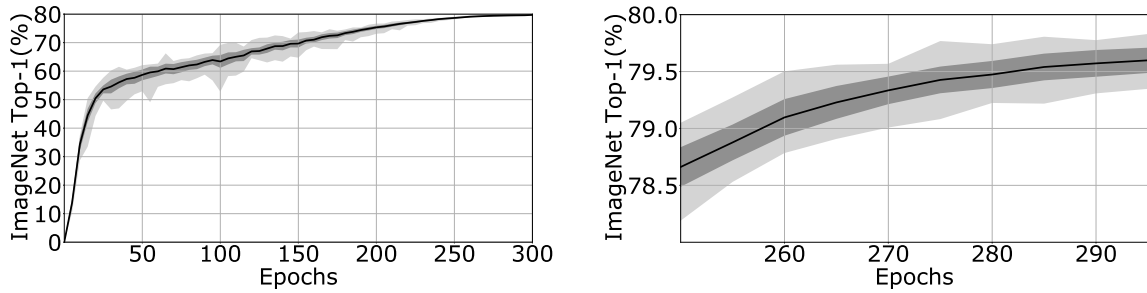


Figure 6.9 – We show how the mean, standard deviation, minimum and maximum of the top-1 accuracy on ImageNet-val evolves during training with the A2 procedure (ResNet-50 architecture). **(Left)** For all 300 training epochs. **(Right)** Same but for the last epochs. We note that the variance in accuracy is high at the beginning, see for instance at epoch 100, where the difference in performance can be as large as 10% in accuracy. Towards the end of the training, most of the networks converge to similar values and the range significantly decreases in the last 50 epochs. *Credit:* this figure and experiment was inspired by Picard [152].

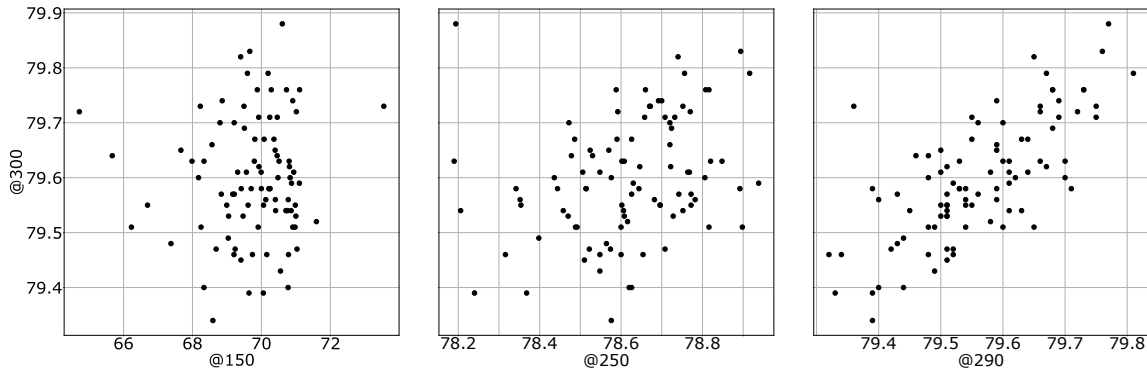


Figure 6.10 – Final accuracy @ Epoch 300 versus accuracy at epochs 150, 250 and 290, for 100 networks trained with A2 training. It is only close to the end of the training that we start observing a correlation between temporary and final performance. We can therefore not apply early stopping rules based on an early validation accuracy.

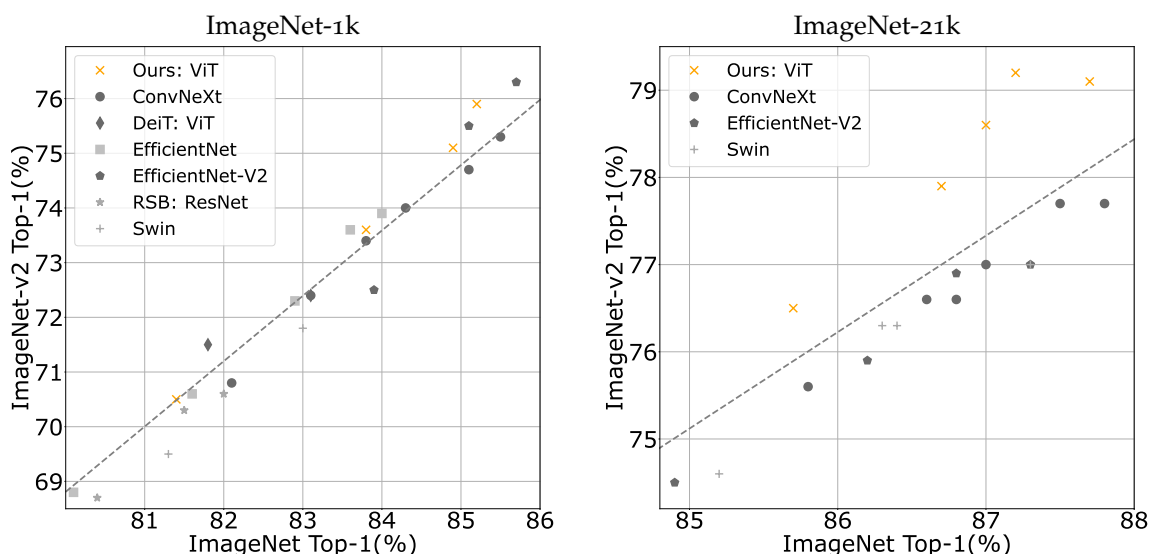


Figure 6.11 – Generalization experiment: top-1 accuracy on ImageNet-1k val versus ImageNet-v2 for models in Table 6.7 and Table 6.8. We display a linear interpolation of all points in order to compare the generalization capability (or level of overfitting) for the different models.

After highlighting the interest of overfitting control with the ResNet50 and the training procedure of Wightman et al [215], we study it with our training procedure and the ViT architecture.

**Overfitting evaluation with DeiT III** The comparison between ImageNet-val and -v2 is a way to quantify overfitting [200], or at least the better capability to generalize in a nearby setting without any fine-tuning<sup>1</sup>. In Figure 6.11 we plot ImageNet-val top-1 accuracy vs ImageNet-v2 top-1 accuracy in order to evaluate how the models performed when evaluated on a test set never seen at validation time. Our models overfit significantly than all other models considered, especially on ImageNet-21k. This is a good behaviour that validates the fact that our restricted choice of hyper-parameters and variants in our recipe does not lead to (too much) overfitting.

### 6.3.5 Downstream tasks and other architectures

#### 6.3.5.1 Transfer Learning

Dataset	Train size	Test size	#classes
iNaturalist 2018 [104]	437,513	24,426	8,142
iNaturalist 2019 [103]	265,240	3,003	1,010
Flowers-102 [144]	2,040	6,149	102
Stanford Cars [121]	8,144	8,041	196
CIFAR-100 [124]	50,000	10,000	100
CIFAR-10 [124]	50,000	10,000	10

Table 6.10 – Datasets used for our different transfer-learning tasks.

In order to evaluate the quality of the ViT models learned through our training procedure we evaluated them with transfer learning tasks. We focus on the performance of ViT models

<sup>1</sup>. Caveat: The measures are less robust with -V2 as the number of test images is 10000 instead of 50000 for ImageNet-val. This translates to a higher standard deviation (about 0.2%).



Table 6.11 – We compare Transformers based models on different transfer learning tasks with ImageNet-1k pre-training. We report results with our default training on ImageNet-1k (400 epochs at resolution  $224 \times 224$ ). We also report results with convolutional architectures for reference. For consistency we keep our crop ratio equal to 1.0 on all datasets. Other works use 0.875, which is better for iNat-19 and iNat-18, see Figure 6.6.

Model	CIFAR-10	CIFAR-100	Flowers	Cars	iNat-18	iNat-19
Grafit ResNet-50 [198]	–	–	98.2	92.5	69.8	75.9
ResNet-152 [40]	–	–	–	–	69.1	–
ViT-B/16 [61]	98.1	87.1	89.5	–	–	–
ViT-L/16 [61]	97.9	86.4	89.7	–	–	–
ViT-B/16 [183]	–	87.8	96.0	–	–	–
ViT-L/16 [183]	–	86.2	91.4	–	–	–
DeiT-B	99.1	90.8	98.4	92.1	73.2	77.7
Ours ViT-S	98.9	90.6	96.4	89.9	67.1	72.7
Ours ViT-B	99.3	92.5	98.6	93.4	73.6	78.0
Ours ViT-L	<b>99.3</b>	<b>93.4</b>	<b>98.9</b>	<b>94.5</b>	<b>75.6</b>	<b>79.3</b>

pre-trained on ImageNet-1k only at resolution  $224 \times 224$  during 400 epochs on the 6 datasets shown in Table 6.10. Our results are presented in Table 6.11. In Figure 6.6 we measure the impact of the crop ratio at inference time on transfer learning results. We observe that on iNaturalist this parameter has a significant impact on the performance. As recommended in the paper Three Things [196] we finetune only the attention layers for transfer learning experiments on Flowers, this improves performance by 0.2%.

### 6.3.5.2 Semantic segmentation

We evaluate our ViT baselines models (400 epochs schedules for ImageNet-1k models and 90 epochs for ImageNet-21k models) with semantic segmentation experiments on ADE20k dataset [247]. This dataset consists of 20k training and 5k validation images with labels over 150 categories. For the training, we adopt the same schedule as in Swin: 160k iterations with UperNet [223]. At test time we evaluate with a single scale and multi-scale. Our UperNet implementation is based on the XCiT [66] repository. By default the UperNet head uses an embedding dimension of 512. In order to save compute, for small and tiny models we set it to the size of their working dimension, i.e. 384 for small and 192 for tiny. We keep the 512 by default as it is done in XCiT for other models. Our results are reported in Table 6.12. We observe that vanilla ViTs trained with our training recipes have a better FLOPs-accuracy trade-off than recent architectures like XCiT or Swin.

### 6.3.5.3 Training with others architectures

In Table 6.13 we measure the top-1 accuracy on ImageNet-val, ImageNet-real and ImageNet-v2 with different architecture train with our training procedure at resolution  $224 \times 224$  on ImageNet-1k only. We can observe that for some architectures like PiT or CaiT our training method will improve the performance. For some others like TNT our approach is neutral and for architectures like Swin it decreases the performance. This is consistent with the findings of Wightman et al. [215] and illustrates the need to improve the training procedure in conjunction to the architecture to obtain robust conclusions. Indeed, adjusting these architectures while keeping the training procedure fixed can probably have the same effect as keeping the architecture fixed and adjusting the training procedure. That means that with a fixed training procedure we can have an overfitting of an architecture for a given training procedure. In order to take overfitting into account we perform our measurements on the ImageNet val and ImageNet-v2 to quantify the amount of overfitting.

Table 6.12 – ADEzok semantic segmentation performance using UperNet [223] (in comparable settings [59, 66, 134]). All models are pre-trained on ImageNet-1k except models with † symbol that are pre-trained on ImageNet-21k. We report the pre-training resolution used on ImageNet-1k and ImageNet-21k.

Backbone	Pre-training		UperNet		
	resolution	#params ( $\times 10^6$ )	FLOPs ( $\times 10^9$ )	Single scale mIoU	Multi-scale mIoU
ResNet50	224 $\times$ 224	66.5	–	42.0	–
DeiT-S	224 $\times$ 224	52.0	1099	–	44.0
XciT-T12/16	224 $\times$ 224	34.2	874	41.5	–
XciT-T12/8	224 $\times$ 224	33.9	942	43.5	–
Swin-T	224 $\times$ 224	59.9	945	44.5	46.1
Our ViT-T	224 $\times$ 224	10.9	148	40.1	41.8
Our ViT-S	224 $\times$ 224	41.7	588	<b>45.6</b>	<b>46.8</b>
XciT-M24/16	224 $\times$ 224	112.2	1213	47.6	–
XciT-M24/8	224 $\times$ 224	110.0	2161	48.4	–
PatchConvNet-B60	224 $\times$ 224	140.6	1258	48.1	48.6
PatchConvNet-B120	224 $\times$ 224	229.8	1550	49.4	50.3
MAE ViT-B	224 $\times$ 224	127.7	1283	48.1	–
Swin-B	384 $\times$ 384	121.0	1188	48.1	49.7
Our ViT-B	224 $\times$ 224	127.7	1283	49.3	50.2
Our ViT-L	224 $\times$ 224	353.6	2231	<b>51.5</b>	<b>52.0</b>
PatchConvNet-B60†	224 $\times$ 224	140.6	1258	50.5	51.1
PatchConvNet-L120†	224 $\times$ 224	383.7	2086	52.2	52.9
Swin-B† (640 $\times$ 640)	224 $\times$ 224	121.0	1841	50.0	51.6
Swin-L† (640 $\times$ 640)	224 $\times$ 224	234.0	3230	–	53.5
Our ViT-B†	224 $\times$ 224	127.7	1283	51.8	52.8
Our ViT-B†	384 $\times$ 384	127.7	1283	53.4	54.1
Our ViT-L†	224 $\times$ 224	353.6	2231	53.8	54.7
Our ViT-L†	320 $\times$ 320	353.6	2231	<b>54.6</b>	<b>55.6</b>

### 6.3.6 Experimental details

**Fine-tuning at higher resolution** When pre-training on ImageNet-1k at resolution 224  $\times$  224 we fix the train-test resolution discrepancy by finetuning at a higher resolution [199]. Our finetuning procedure is inspired by the Chapter 3 finetuning procedure, except that we adapt the stochastic depth rate according to the model size [197]. We fix the learning rate to  $lr = 1 \times 10^{-5}$  with batch-size=512 during 20 epochs with a weight decay of 0.1 without repeated augmentation. Other hyper-parameters are similar to those employed in Chapter 3 fine-tuning.

**Stochastic depth** We adapt the stochastic depth drop rate according to the model size. We report stochastic depth drop rate values in Table 6.14.

## 6.4 Additional Ablations

**Number of training epochs** In Table 6.15 we provide an ablation on the number of training epochs on ImageNet-1k. We do not observe a saturation when the increase of the number of training epochs, as observed with Bert like approaches [11, 92]. For longer training we increase the weight decay from 0.02 to 0.05 and we increase the stochastic depth drop-rate by 0.05 every 200 epochs to prevent overfitting.

Model	Params ( $\times 10^6$ )	Flops ( $\times 10^9$ )	ImageNet-1k			
			orig.	val	real	v2
ViT-S [193]	22.0	4.6	79.8	80.4	86.1	69.7
ViT-B [61, 193]	86.6	17.6	81.8	83.1	87.7	72.6
PiT-S [99]	23.5	2.9	80.9	80.4	86.1	69.2
PiT-B [99]	73.8	12.5	82.0	82.4	86.8	72.0
TNT-S [89]	23.8	5.2	81.5	81.4	87.2	70.6
TNT-B [89]	65.6	14.1	82.9	82.9	87.6	72.2
ConViT-S [52]	27.8	5.8	81.3	81.3	87.0	70.3
ConViT-B [52]	86.5	17.5	82.4	82.0	86.7	71.3
Swin-S [134]	49.6	8.7	83.0	82.1	86.9	70.7
Swin-B [134]	87.8	15.4	83.5	82.2	86.7	70.7
CaiT-B12 [197]	100.0	18.2	–	83.3	87.7	73.3

Table 6.13 – We report the performance reached with our training recipe with 400 epochs at resolution  $224 \times 224$  for other transformers architectures. We have not performed an extensive grid search to adapt the hyper-parameters to each architecture. Our results are overall similar to the ones achieved in the papers where these architectures were originally published (reported in column ‘orig.’), except for Swin Transformers, for which we observe a drop on ImageNet-val.

Model	# Params ( $\times 10^6$ )	FLOPs ( $\times 10^9$ )	Stochastic depth drop-rate	
			ImageNet-1k	ImageNet-21k
ViT-T	5.7	1.3	0.0	0.0
ViT-S	22.0	4.6	0.0	0.0
ViT-B	86.6	17.5	0.1	0.1
ViT-L	304.4	61.6	0.4	0.3
ViT-H	632.1	167.4	0.5	0.5

Table 6.14 – Stochastic depth drop-rate according to the model size. For 400 epochs training on ImageNet-1k and 90 epochs training on ImageNet-21k. See section 6.4 for further adaption with longer training.

Model	epochs	ImageNet top1 acc.		
		val	real	v2
ViT-S	300	79.9	86.1	68.8
	400	80.4	86.1	69.7
	600	80.8	86.7	69.9
	800	81.4	87.0	70.5
ViT-B	300	82.8	87.6	72.1
	400	83.1	87.7	72.6
	600	83.2	87.8	73.3
	800	83.7	88.1	73.1
ViT-L	300	84.1	88.5	74.1
	400	84.2	88.6	74.3
	600	84.4	88.6	74.6
	800	84.5	88.8	75.0
ViT-H	300	84.6	89.0	74.9
	400	84.8	89.1	75.3

Table 6.15 – Impact on the performance of the number of training epochs on ImageNet-1k.

## 6.5 Conclusion

This chapter makes a simple contribution: it proposes improved baselines for vision transformers trained in a supervised fashion that can serve (1) as a comparison basis for new architectures; (2) for other training approaches such as those based on self-supervised learning.

To summarize the main ingredients are as follows:

- We build upon the work of Wightman et al. [215] introduced for ResNet50. In particular we adopt a binary cross entropy loss for ImageNet-1k training. We adapt this method by including ingredients that significantly improve the training of large ViT [197], namely stochastic depth [108] and LayerScale [197], explained in chapter 3.
- **3-Augment**: is a simple data augmentation inspired by that employed for self-supervised learning. It works better than the usual automatic/learned data-augmentation employed to train vision transformers like RandAugment [44].
- **Simple Random Cropping** is more effective than Random Resize Cropping when pre-training on a larger set like ImageNet-21k.
- **A lower resolution** at training time. This choice reduces the train-test discrepancy [199] but has not been much exploited with ViT. We observe that it also has a regularizing effect for the largest models by preventing overfitting. For instance, for a target resolution of  $224 \times 224$ , a ViT-H pre-trained at resolution  $126 \times 126$  (81 tokens) achieves a better performance on ImageNet-1k than when pre-training at resolution  $224 \times 224$  (256 tokens). This is also less demanding at pre-training time, as there are 70% fewer tokens. From this perspective it offers similar scaling properties as masked-autoencoders [92].

Our “new” training strategies do not saturate with the largest models, making another step beyond the Data-efficient image Transformer (DeiT) introduced in chapter 3. As a result, we obtain a competitive performance in image classification and segmentation, even when compared to recent popular architectures such as SwinTransformers [134] or modern convnet architectures like ConvNext [135]. Below we point out a few interesting outcomes.

- We leverage models with more capacity even on midsize datasets. For instance we reach 85.2% in top-1 accuracy when training a ViT-H on ImageNet1k only, which is an improvement of +5.1% over the best ViT-H with supervised training procedure reported in the literature at resolution  $224 \times 224$ .
- Our training procedure for ImageNet-1k allows us to train a **billion-parameter ViT-H** (52 layers) without any hyper-parameter adaptation, just using the same stochastic depth drop-rate as for the ViT-H. It attains 84.9% at  $224 \times 224$ , i.e., +0.2% higher than the corresponding ViT-H trained in the same setting.
- Without sacrificing performance, we **divide by more than 2** the number of GPUs required and the training time for ViT-H, making it effectively possible to train such models with a reduced amount of resources. This is thanks to our pre-training at lower resolution, which reduces the peak memory.
- For ViT-B and ViT-L models, our supervised training approach is on par with Bert-like self-supervised approaches [11, 92] with their default training setting and when using the

same level of annotations and less epochs, both for the tasks of image classification and of semantic segmentation.

- With this improved training procedure, a vanilla ViT closes the gap with recent state-of-the-art architectures, often offering better compute/performance trade-offs. Our models are also comparatively better on the additional test set ImageNet-V2 [159], which indicates that our trained models generalize better to another validation set than most prior works.
- An ablation on the effect of the crop ratio employed in transfer learning classification tasks. We observe that it has a noticeable impact on the performance but that the best value depends a lot on the target dataset/task.

We hope that this very strong baseline will stimulate the discussion about good practice to learn huge foundation models. Our experiments have also gathered a few insights on how to train ViT for larger models with reduced resources without hurting accuracy, allowing us to train a one-billion parameter model with 4 nodes of 8 GPUs.

## CONCLUSION

We summarise in the following our main contributions:

**Grafit.** In the coarse-to-fine representation learning context presented in Chapter 2, we make the following contributions: We propose Grafit, a method to learn image representations at a finer granularity than the one offered by the annotation at training time. Inspired by the recent self-supervised BYOL [87] instance learning approach, we carefully design a joint learning scheme integrating instance and coarse-label based classification losses. For the latter one, we exploit a knn strategy but with a dedicated process to manage the memory both at train-time and for inference at test-time. We propose two original use-cases to deeply evaluate coarse-trained fine-grained testing evaluation, for which Grafit exhibits outstanding performance. For instance, we improve by **+16.3%** the top-1 accuracy for on-the-fly classification on ImageNet. This improvement is still +9.5% w.r.t. our own stronger baseline. Grafit also improves transfer learning: our experiments show that our representation discriminates better at a finer granularity.

**Transformers: DeiT & CaiT.** In Chapter 3 we show how to train transformer-based image classification neural networks on ImageNet only. With DeiT, we report large improvements over previous ViT [61] results. We introduce a novel distillation procedure based on a token-based strategy. With CaiT, we propose a new method to train deeper models called LayerScale and a new architecture designed to extract the information inside the architecture called class attention. In Chapter 6 we revisit the training strategy for vision transformers. We propose a new data-augmentation approach called 3Augment. We adapt the cropping approach according to the datasets size. We called our whole training procedure DeiT III, with which we achieve competitive performance on image classification and semantic segmentation with vanilla vision transformers. We show the importance of the interaction between architecture and training procedure by comparing our approach with BeiT like pre-training and discussing the performance of DeiT III when applied to different architectures. We have also proposed additional improvements (not detailed in Chapter 6) for the ViT architectures in our last paper *Three things everyone should know about Vision Transformers*, we refer to the publication for more details.

**ResMLP & PatchConvnet.** We further investigate patch-based residual architectures, alternating feed-forward blocks and linear patch interaction layers. Thanks to modern training strategies similar to ours proposed for transformers, we achieve an unexpectedly high performance on ImageNet classification benchmarks. In Chapter 5, we introduce an attention-based pooling layer, which offers visualization properties and interpretability by design. In addition, we propose a full patch-based ConvNet with no pyramidal structure design to best exploit our pooling layer. We demonstrate its interest on several computer vision tasks: classification, segmentation and detection.

In this thesis, we have studied the training procedures and architectures that can be used in computer vision, mostly for image classification. We have shown that it is possible to achieve a competitive performance in image classification with transformer architectures without using hundreds of millions of annotated images. We also showed that a MLP-like architecture could

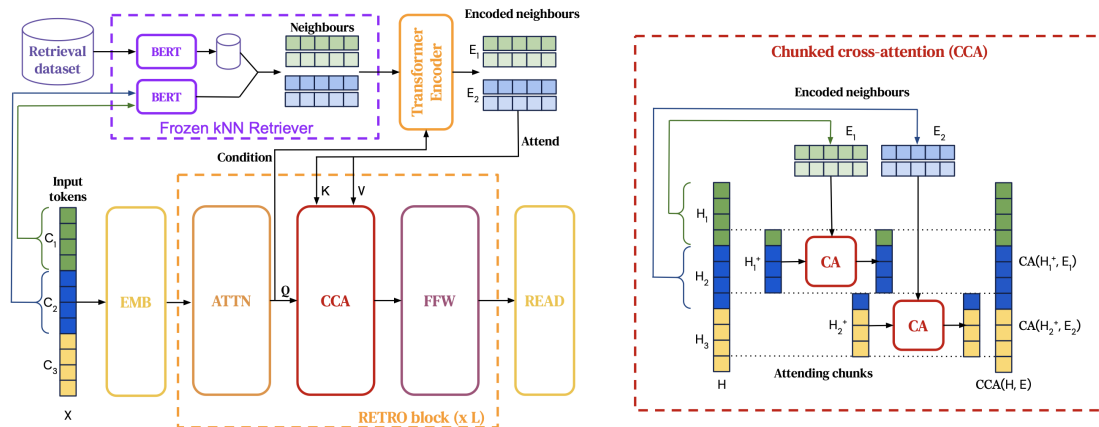


Figure 7.1 – Illustration of transformers with retrieval mechanisms as in used in the Retro transformers. Credit: Borgeaud et al. *“Improving language models by retrieving from trillions of tokens”*.

obtain reasonable performance on different computer vision tasks and on NLP tasks. We have highlighted the impact of the interactions between the training procedures and the architectures, and thus the need to revisit the baselines with modern procedures to get consistent conclusions.

**Perspectives.** With a proper training procedure [183, 193, 215], Transformers achieve interesting performance/complexity trade-offs. Although we have studied the training procedures (see Chapter 3 and 6), much remains to be done. Transformers seem more efficient than convnet when we have a large dataset (like JFT-300M or JFT-3B) as shown by Dai et al. [49]. Although there are some studies interested in the scaling of vision transformers [49, 162, 239], we are still far from obtained effective models with more than 100B parameters as it is the case in NLP [27, 39, 157]. BerT-like approaches [11, 92] also seem very promising for training bigger models. Indeed, the masking task seems to be less prone to overfitting, which is essential for this kind of models. Let us mention two challenging use-cases:

- **Vision transformers by retrieving from trillions of tokens.** Transformers exploiting an extra token base with retrieval in its attention process as illustrated in Figure 7.1 with the retro method [22], are very successful approaches in NLP. This kind of approach is not yet used in vision. However, this could have many advantages, for example, for classification tasks with non-fixed classes. Indeed, using an external memory allows the model to include parts that are non-parametric (as in Chapter 2). This allows the behaviour of the network to be modified without the need to re-train the weights. Indeed, it is only necessary to update the external database. This also has the advantage of not having to encode all the useful information in the weights of the network, which can be difficult if we have few data for certain concepts we want to encode.
- **Vision & language joint representation.** Since Radford et al. [154] (see Figure 7.2) joint models for vision and language become more and more popular. Indeed, since transformers have become very powerful in vision, using this architecture together in vision and language seems very promising. Being able to align text and vision representations allows to exploit new properties by contextualising the output of the model in some way thanks to the text. Models combining vision and language appear to be able to handle different tasks without requiring a complete fine tuning as illustrated in the Flamingo [5] paper. These approaches are thus very promising given the different tasks they can handle. However, this field is

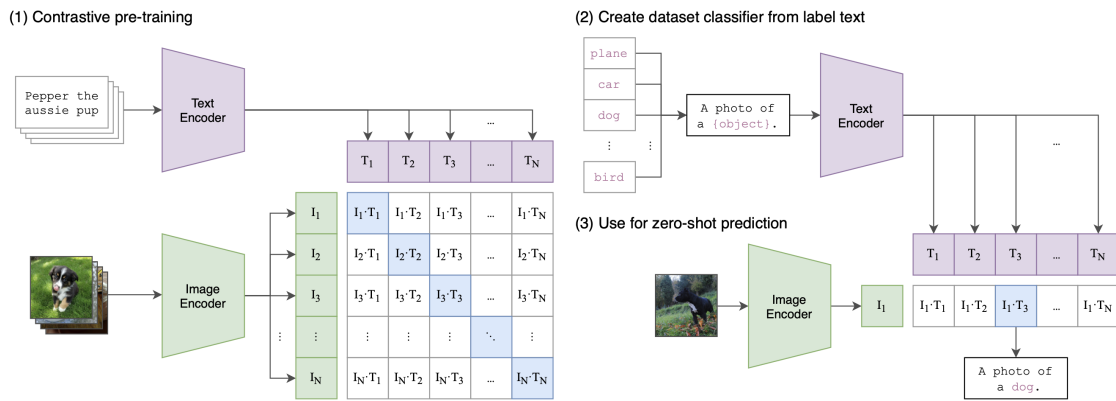


Figure 7.2 – Illustration of the Clip method. Credit: Radford et al. "Learning Transferable Visual Models From Natural Language Supervision".

quite new, and the architectures and training procedures are far from being fully understood. Having a better knowledge of how to scale these models, as well as finding the right way to design these architectures and the way to train them, seems to be a very important aspect. Indeed, as we have highlighted in Chapter 6, the interaction between architecture and training should play a key role in the performance of such models.



## List of main PhD publications

We detail below the publications associated with this thesis. Some of these are not covered in this manuscript and we refer the reader to the articles for more details.

### Main PhD publications

- [1] *DeiT III: Revenge of the ViT*  
**Hugo Touvron**, Matthieu Cord, Hervé Jégou.  
*ECCV 2022*
- [2] *Three things everyone should know about Vision Transformers*  
**Hugo Touvron**, Matthieu Cord, Alaa El-Nouby, Jakob Verbeek, Hervé Jégou.  
*ECCV 2022*
- [3] *ResMLP: Feedforward networks for image classification with data-efficient training*  
**Hugo Touvron**, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Gautier Izacard, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, Hervé Jégou.  
*arXiv 2021, submitted TPAMI (minor revision)*
- [4] *Augmenting Convolutional networks with attention-based aggregation*  
**Hugo Touvron**, Matthieu Cord, Alaaeldin El-Nouby, Piotr Bojanowski, Armand Joulin, Gabriel Synnaeve, Hervé Jégou.  
*arXiv 2021 (under review NeurIPS 2022)*
- [5] *Resnet strikes back: An improved training procedure in timm*  
Ross Wightman, **Hugo Touvron**, Hervé Jégou.  
*NeurIPS workshop 2021 (Spotlight & Best Paper Award)*
- [6] *Going deeper with image transformers*  
**Hugo Touvron**, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, Hervé Jégou.  
*ICCV 2021 (Oral)*
- [7] *Grafit: Learning fine-grained image representations with coarse labels*  
**Hugo Touvron**, Alexandre Sablayrolles, Matthijs Douze, Matthieu Cord, Hervé Jégou.  
*ICCV 2021*
- [8] *Training data-efficient image transformers & distillation through attention*  
**Hugo Touvron**, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, Hervé Jégou.  
*ICML 2021*

### Others publications or preprints

- [1] *Are Large-scale Datasets Necessary for Self-Supervised Pre-training?*  
Alaaeldin El-Nouby, Gautier Izacard, **Hugo Touvron**, Ivan Laptev, Hervé Jégou, Edouard Grave.  
*arXiv 2021*

- [2] *XCiT: Cross-Covariance Image Transformers*  
Alaaeldin El-Nouby, **Hugo Touvron**, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, Hervé Jégou.  
*NeurIPS 2021*
- [3] *Emerging properties in self-supervised vision transformers*  
Mathilde Caron, **Hugo Touvron**, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, Armand Joulin.  
*ICCV 2021*
- [4] *LeViT: a Vision Transformer in ConvNet's Clothing for Faster Inference*  
Ben Graham, Alaaeldin El-Nouby, **Hugo Touvron**, Pierre Stock, Armand Joulin, Hervé Jégou, Matthijs Douze.  
*ICCV 2021*
- [5] *Convit: Improving vision transformers with soft convolutional inductive biases*  
Stéphane d'Ascoli, **Hugo Touvron**, Matthew Leavitt, Ari Morcos, Giulio Biroli, Levent Sagun.  
*ICML 2021*
- [6] *Powers of layers for image-to-image translation*  
**Hugo Touvron**, Matthijs Douze, Matthieu Cord, Hervé Jégou.  
*arXiv 2020*
- [7] *Fixing the train-test resolution discrepancy: Fixefficientnet*  
**Hugo Touvron**, Andrea Vedaldi, Matthijs Douze, Hervé Jégou.  
*arXiv 2020*
- [8] *Fixing the train-test resolution discrepancy*  
**Hugo Touvron**, Andrea Vedaldi, Matthijs Douze, Hervé Jégou.  
*NeurIPS 2019*





Table 7.3 – Author and Creative Commons Copyright notice for images in Figure 2.9.

slsfirefight: CC BY-NC 4.0, J. Maughn: CC BY-NC 4.0, Tim Hite: CC BY 4.0, Mikael Behrens: CC BY-NC 4.0, colinmorita: CC BY-NC 4.0, Mary Joyce: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, icosahedron: CC BY 4.0, Donna Pomeroy: CC BY-NC 4.0, tnewman: CC BY-NC 4.0, phylocode: CC BY-NC 4.0, Marisa or Robin Agarwal: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Tom Benson: CC BY-NC-ND 4.0, tam topes: CC BY-NC 4.0, James Maughn: CC BY-NC 4.0, Mark Rosenstein: CC BY-NC-SA 4.0, Marisa or Robin Agarwal: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, slsfirefight: CC BY-NC 4.0, Thomas Koffel: CC BY-NC 4.0, Chris Evers: CC BY-NC 4.0, jaliya: CC BY-NC 4.0, Chris Evers: CC BY-NC 4.0, Victor W Fazio III: CC BY-NC-ND 4.0, Chris Evers: CC BY-NC 4.0, J. Maughn: CC BY-NC 4.0, Andrew Cannizzaro: CC BY 4.0, 116916927065934112165: CC BY-NC-ND 4.0, gyrrlfalcon: CC BY-NC 4.0, kolasafamily: CC BY-NC 4.0, summermule: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, David R: CC BY-NC-ND 4.0, summermule: CC BY-NC 4.0, slsfirefight: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, petecorradino: CC BY-NC 4.0, Mike Leveille: CC BY-NC 4.0, greglasley: CC BY-NC 4.0, tegmort: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Tom Benson: CC BY-NC-ND 4.0, flyfisherking: CC BY-NC 4.0, Jean-Lou Justine: CC BY 4.0, Judith Lopez Sikora: CC BY-NC 4.0, kolasafamily: CC BY-NC 4.0, nudibranchmom: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, R.J. Adams: CC BY-NC 4.0, monicamares: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, nudibranchmom: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Mike Leveille: CC BY-NC 4.0, nudibranchmom: CC BY-NC 4.0, slsfirefight: CC BY-NC 4.0, J. Maughn: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Tom Benson: CC BY-NC-ND 4.0, JJ Johnson: CC BY-NC 4.0, James Maughn: CC BY-NC 4.0, petecorradino: CC BY-NC 4.0, summermule: CC BY-NC 4.0, pfaucher: CC BY-NC 4.0, Amy: CC BY-NC 4.0, enzedfred: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, BJ Stacey: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Ken-ichi Ueda: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Brian Gratwicke: CC BY 4.0, slsfirefight: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, summermule: CC BY-NC 4.0, KK: CC BY-NC 4.0, John Karges: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Javier Solís: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Mike Leveille: CC BY-NC 4.0, Ken-ichi Ueda: CC BY-NC 4.0, J. Maughn: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, David J Barton: CC BY-NC 4.0, James Maughn: CC BY-NC 4.0, greglasley: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, James Maughn: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, slsfirefight: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, John Karges: CC BY-NC 4.0, 104623964081378888743: CC BY-NC-ND 4.0, Judith Lopez Sikora: CC BY-NC 4.0, Amy: CC BY-NC 4.0, Marisa or Robin Agarwal: CC BY-NC 4.0, summermule: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Jennifer Rycenga: CC BY-NC 4.0, David J Barton: CC BY-NC 4.0, thylacine: CC BY-NC 4.0, greglasley: CC BY-NC 4.0, J. Maughn: CC BY-NC 4.0, Javier Solís: CC BY-NC 4.0, redhat: CC BY-NC 4.0, timputtre: CC BY-NC 4.0, icosahedron: CC BY 4.0, rbrummitt: CC BY-NC 4.0, icosahedron: CC BY 4.0, David J Barton: CC BY-NC 4.0, slsfirefight: CC BY-NC 4.0, 104623964081378888743: CC BY-NC-ND 4.0, Ken-ichi Ueda: CC BY-NC-SA 4.0, Donna Pomeroy: CC BY-NC 4.0, sakuraisomi: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, J. Maughn: CC BY-NC 4.0, kestrel: CC BY-NC 4.0, BJ Stacey: CC BY-NC 4.0, summermule: CC BY-NC 4.0, thylacine: CC BY-NC 4.0, icosahedron: CC BY 4.0, KK: CC BY-NC 4.0, James Maughn: CC BY-NC 4.0, Javier Solís: CC BY-NC 4.0, rbrummitt: CC BY-NC 4.0, J. Maughn: CC BY-NC 4.0, greglasley: CC BY-NC 4.0, greglasley: CC BY-NC 4.0, timputtre: CC BY-NC 4.0.

## BIBLIOGRAPHY

- [1] Apex. <https://nvidia.github.io/apex/index.html>. Accessed: 2022-01-01.
- [2] Authors:copyright for Figure 1 images from inaturalist-2018, top to down, left to right, employed for illustration of research work. Diana-Terry Hibbitts: CC BY-NC 4.0, Ronald Werson:CC BY-NC-ND 4.0, Greg Lasley: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Stefano Tito: CC BY-NC 4.0, Marion Zöller: CC BY-NC 4.0, Stefano Tito: CC BY-NC 4.0, Ronald Werson: CC BY-NC-ND 4.0, Ronald Werson: CC BY-NC-ND 4.0, Donna Pomeroy: CC BY-NC 4.0, Chris van Swaay: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Giuseppe Cagnetta: CC BY-NC 4.0, Chris van Swaay: CC BY-NC 4.0, martinswarren: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Robin Agarwal: CC BY-NC 4.0, Fernando de Juana: CC BY-NC 4.0, Giuseppe Cagnetta: CC BY-NC 4.0, Ronald Werson: CC BY-NC-ND 4.0, Marion Zöller: CC BY-NC 4.0, martinswarren: CC BY-NC 4.0, Ronald Werson: CC BY-NC-ND 4.0, Donna Pomeroy: CC BY-NC 4.0, Donna Pomeroy: CC BY-NC 4.0, Marion Zöller: CC BY-NC 4.0, martinswarren: CC BY-NC 4.0, note = Accessed: 2020-06-10.
- [3] Pytorch. <https://pytorch.org/vision/stable/index.html>. Accessed: 2021-08-01.
- [4] Samira Abnar, Mostafa Dehghani, and Willem Zuidema. Transferring inductive biases through knowledge distillation. *arXiv preprint arXiv:2006.00555*, 2020.
- [5] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikołaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. *arXiv preprint arXiv:2204.14198*, 2022.
- [6] Anonymous. Patches are all you need? In *Submitted to The Tenth International Conference on Learning Representations*, 2022. under review.
- [7] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.
- [8] Artem Babenko, Anton Slesarev, Alexander Chigorin, and Victor S. Lempitsky. Neural codes for image retrieval. *arXiv preprint arXiv:1404.1777*, 2014.
- [9] Thomas C. Bachlechner, Bodhisattwa Prasad Majumder, H. H. Mao, G. Cottrell, and Julian McAuley. Rezero is all you need: Fast convergence at large depth. *arXiv preprint arXiv:2003.04887*, 2020.
- [10] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [11] Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.
- [12] Hangbo Bao, Li Dong, and Furu Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.
- [13] Irwan Bello, W. Fedus, Xianzhi Du, E. D. Cubuk, A. Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting ResNets: Improved training and scaling strategies. *arXiv preprint arXiv:2103.07579*, 2021.

- [14] Irwan Bello, Barret Zoph, Ashish Vaswani, Jonathon Shlens, and Quoc V. Le. Attention augmented convolutional networks. *International Conference on Computer Vision*, 2019.
- [15] Maxim Berman, Hervé Jégou, Andrea Vedaldi, Iasonas Kokkinos, and Matthijs Douze. Multi-grain: a unified image embedding for classes and instances. *arXiv preprint arXiv:1902.05509*, 2019.
- [16] David Berthelot, N. Carlini, E. D. Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring. *arXiv preprint arXiv:1911.09785*, 2019.
- [17] David Berthelot, Nicholas Carlini, I. Goodfellow, Nicolas Papernot, A. Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. In *NeurIPS*, 2019.
- [18] Lucas Beyer, Olivier J. Hénaff, Alexander Kolesnikov, Xiaohua Zhai, and Aaron van den Oord. Are we done with ImageNet? *arXiv preprint arXiv:2006.07159*, 2020.
- [19] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2006.
- [20] Théodore Bluche. *Deep neural networks for large vocabulary handwritten text recognition*. PhD thesis, Université Paris-Sud, 2015.
- [21] Piotr Bojanowski and Armand Joulin. Unsupervised learning by predicting noise. *arXiv preprint arXiv:1704.05310*, 2017.
- [22] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George van den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, Diego de Las Casas, Aurelia Guy, Jacob Menick, Roman Ring, T. W. Hennigan, Saffron Huang, Lorenzo Maggiore, Chris Jones, Albin Cassirer, Andy Brock, Michela Paganini, Geoffrey Irving, Oriol Vinyals, Simon Osindero, Karen Simonyan, Jack W. Rae, Erich Elsen, and L. Sifre. Improving language models by retrieving from trillions of tokens. *arXiv preprint arXiv:2112.04426*, 2021.
- [23] Lukas Bossard, Matthieu Guillaumin, and Luc Van Gool. Food-101 – mining discriminative components with random forests. In *European Conference on Computer Vision*, 2014.
- [24] Stéphane Boucheron, Olivier Bousquet, and Gabor Lugosi. Theory of classification: A survey of some recent advances. *ESAIM: probability and statistics*, 2005.
- [25] A. Brock, Soham De, S. L. Smith, and K. Simonyan. High-performance large-scale image recognition without normalization. *arXiv preprint arXiv:2102.06171*, 2021.
- [26] Andrew Brock, Soham De, and Samuel L Smith. Characterizing signal propagation to close the performance gap in unnormalized resnets. *arXiv preprint arXiv:2101.08692*, 2021.
- [27] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.
- [28] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *European Conference on Computer Vision*, 2020.
- [29] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *arXiv preprint arXiv:2006.09882*, 2020.

- [30] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *arXiv preprint arXiv:2104.14294*, 2021.
- [31] Clément Chatelain. *Extraction de séquences numériques dans des documents manuscrits quelconques*. PhD thesis, Université de Rouen, 2006.
- [32] Aditya Chattopadhyay, Anirban Sarkar, Prantik Howlader, and Vineeth N. Balasubramanian. Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018.
- [33] Hila Chefer, Shir Gur, and Lior Wolf. Transformer interpretability beyond attention visualization. *Conference on Computer Vision and Pattern Recognition*, 2021.
- [34] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.
- [35] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.
- [36] J. H. Cho and B. Hariharan. On the efficacy of knowledge distillation. *International Conference on Computer Vision*, 2019.
- [37] Minsu Cho, Suha Kwak, Cordelia Schmid, and Jean Ponce. Unsupervised object discovery and localization in the wild: Part-based matching with bottom-up region proposals. *Conference on Computer Vision and Pattern Recognition*, 2015.
- [38] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Conference on Computer Vision and Pattern Recognition*, 2017.
- [39] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek B Rao, Parker Barnes, Yi Tay, Noam M. Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Benton C. Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier García, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayanan Pillai, Marie Pellat, Aitor Lewkowycz, Erica Oliveira Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Díaz, Orhan Firat, Michele Catasta, Jason Wei, Kathleen S. Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.
- [40] P. Chu, Xiao Bian, Shaopeng Liu, and Haibin Ling. Feature space augmentation for long-tailed data. *arXiv preprint arXiv:2008.03673*, 2020.
- [41] Tan Kiat Chuan, Liu Yulong, Ambrose Barbara, Tulig Melissa, and Belongie Serge. The herbarium challenge 2019 dataset. *arXiv preprint arXiv:1906.05372*, 2019.
- [42] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep big multilayer perceptrons for digit recognition. In *Neural networks: tricks of the trade*, pages 581–598. Springer, 2012.
- [43] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *In Workshop on Statistical Learning in Computer Vision, ECCV*, 2004.



- [44] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. RandAugment: Practical automated data augmentation with a reduced search space. *arXiv preprint arXiv:1909.13719*, 2019.
- [45] Ekin Dogus Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.
- [46] Yin Cui, Zeqi Gu, Dhruv Kumar Mahajan, Laurens van der Maaten, Serge J. Belongie, and Ser-Nam Lim. Measuring dataset granularity. *arXiv preprint arXiv:1912.10154*, 2019.
- [47] Yin Cui, Yang Song, Chen Sun, Andrew Howard, and Serge J. Belongie. Large scale fine-grained categorization and domain-specific transfer learning. *Conference on Computer Vision and Pattern Recognition*, 2018.
- [48] George V. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, pages 303–314, 1989.
- [49] Zihang Dai, Hanxiao Liu, Quoc V. Le, and Mingxing Tan. Coatnet: Marrying convolution and attention for all data sizes. *arXiv preprint arXiv:2106.04803*, 2021.
- [50] Alexander D’Amour, Katherine Heller, Dan Moldovan, Ben Adlam, Babak Alipanahi, Alex Beutel, Christina Chen, Jonathan Deaton, Jacob Eisenstein, Matthew D. Hoffman, Farhad Hormozdiari, Neil Houlsby, Shaobo Hou, Ghassen Jerfel, Alan Karthikesalingam, Mario Lucic, Yian Ma, Cory McLean, Diana Mincu, Akinori Mitani, Andrea Montanari, Zachary Nado, Vivek Natarajan, Christopher Nielson, Thomas F. Osborne, Rajiv Raman, Kim Ramasamy, Rory Sayres, Jessica Schrouff, Martin Seneviratne, Shannon Sequeira, Harini Suresh, Victor Veitch, Max Vladymyrov, Xuezhi Wang, Kellie Webster, Steve Yadlowsky, Taedong Yun, Xiaohua Zhai, and D. Sculley. Underspecification presents challenges for credibility in modern machine learning, 2020.
- [51] Stéphane d’Ascoli, Levent Sagun, Joan Bruna, and Giulio Biroli. Finding the needle in the haystack with convolutions: on the benefits of architectural bias. In *NeurIPS*, 2019.
- [52] Stéphane d’Ascoli, Hugo Touvron, Matthew L. Leavitt, Ari S. Morcos, Giulio Biroli, and Levent Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *ICML*, 2021.
- [53] Soham De and Samuel L Smith. Batch normalization biases residual blocks towards the identity function in deep networks. *arXiv e-prints*, pages arXiv–2002, 2020.
- [54] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.
- [55] Jia Deng, J. Krause, A. Berg, and Li Fei-Fei. Hedging your bets: Optimizing accuracy-specificity trade-offs in large scale visual recognition. *Conference on Computer Vision and Pattern Recognition*, 2012.
- [56] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*, 2019.
- [57] Terrance Devries and Graham W. Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.
- [58] Jeff Donahue and Karen Simonyan. Large scale adversarial representation learning. In *NeurIPS*, 2019.
- [59] Xiaoyi Dong, Jianmin Bao, Dongdong Chen, Weiming Zhang, Nenghai Yu, Lu Yuan, Dong Chen, and Baining Guo. Cswin transformer: A general vision transformer backbone with cross-shaped windows. *arXiv preprint arXiv:2107.00652*, 2021.

- [60] Xiaoyi Dong, Jianmin Bao, Ting Zhang, Dongdong Chen, Weiming Zhang, Lu Yuan, Dong Chen, Fang Wen, and Nenghai Yu. Peco: Perceptual codebook for bert pre-training of vision transformers. *arXiv preprint arXiv:2111.12710*, 2021.
- [61] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.
- [62] Simon S Du, Yining Wang, Xiyu Zhai, Sivaraman Balakrishnan, Ruslan Salakhutdinov, and Aarti Singh. How many samples are needed to estimate a convolutional neural network? In *NeurIPS*, 2018.
- [63] Thibaut Durand, Taylor Mordan, Nicolas Thome, and Matthieu Cord. WILDCAT: weakly supervised learning of deep convnets for image classification, pointwise localization and segmentation. In *Conference on Computer Vision and Pattern Recognition*, 2017.
- [64] Alexandre Défossez, Yossi Adi, and Gabriel Synnaeve. Differentiable model compression via pseudo quantization noise. *arXiv preprint arXiv:2104.09987*, 2021.
- [65] Alaaeldin El-Nouby, Natalia Neverova, Ivan Laptev, and Hervé Jégou. Training vision transformers for image retrieval. *arXiv preprint arXiv:2102.05644*, 2021.
- [66] Alaaeldin El-Nouby, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, et al. Xcit: Cross-covariance image transformers. *arXiv preprint arXiv:2106.09681*, 2021.
- [67] Alaaeldin El-Nouby, Hugo Touvron, Mathilde Caron, Piotr Bojanowski, Matthijs Douze, Armand Joulin, Ivan Laptev, Natalia Neverova, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Xcit: Cross-covariance image transformers. *arXiv preprint arXiv:2106.09681*, 2021.
- [68] Amir Erfan Eshratifar, David Eigen, Michael J. Gormish, and Massoud Pedram. Coarse2fine: A two-stage training method for fine-grained visual classification. *arXiv preprint arXiv:1909.02680*, 2019.
- [69] Angela Fan, Edouard Grave, and Armand Joulin. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*, 2019. ICLR 2020.
- [70] Angela Fan, Pierre Stock, Benjamin Graham, Edouard Grave, Rémi Gribonval, Hervé Jégou, and Armand Joulin. Training with quantization noise for extreme model compression. In *International Conference on Learning Representations*, 2021.
- [71] Haoqi Fan, Bo Xiong, Karttikeya Mangalam, Yanghao Li, Zhicheng Yan, Jitendra Malik, and Christoph Feichtenhofer. Multiscale vision transformers. *arXiv preprint arXiv:2104.11227*, 2021.
- [72] Christiane Fellbaum. *WordNet: An Electronic Lexical Database*. Bradford Books, 1998.
- [73] Ruth C Fong and Andrea Vedaldi. Interpretable explanations of black boxes by meaningful perturbation. In *International Conference on Computer Vision*, 2017.
- [74] Jerome Fournier, Matthieu Cord, and Sylvie Philipp-Foliguet. Retin: A content-based image indexing and retrieval system. *Pattern Analysis and Applications Journal*, 2001.
- [75] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.
- [76] Kuniyiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 2004.
- [77] Carolina Galleguillos, Brian McFee, Serge J. Belongie, and Gert R. G. Lanckriet. From region similarity to category discovery. *Conference on Computer Vision and Pattern Recognition*, 2011.

- [78] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.
- [79] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252. PMLR, 06–11 Aug 2017.
- [80] Spyros Gidaris, Andrei Bursuc, Nikos Komodakis, Patrick Pérez, and Matthieu Cord. Learning representations by predicting bags of visual words. In *Conference on Computer Vision and Pattern Recognition*, 2020.
- [81] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.
- [82] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Machine Learning*, 2011.
- [83] J. Goldberger, S. Roweis, Geoffrey E. Hinton, and R. Salakhutdinov. Neighbourhood components analysis. In *NeurIPS*, 2004.
- [84] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [85] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.
- [86] Ben Graham, Alaaeldin El-Nouby, Hugo Touvron, Pierre Stock, Armand Joulin, Hervé Jégou, and Matthijs Douze. Levit: a vision transformer in convnet’s clothing for faster inference. *arXiv preprint arXiv:2104.01136*, 2021.
- [87] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.
- [88] Yanming Guo, Yu Liu, Erwin M. Bakker, Yuanhao Guo, and Michael S. Lew. Cnn-rnn: a large-scale hierarchical image classification framework. *Multimedia Tools and Applications*, 2017.
- [89] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *arXiv preprint arXiv:2103.00112*, 2021.
- [90] Boris Hanin and David Rolnick. How to start training: The effect of initialization and architecture. *NeurIPS*, 31, 2018.
- [91] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. 2001.
- [92] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Doll’ar, and Ross B. Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.
- [93] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *International Conference on Computer Vision*, 2017.
- [94] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Conference on Computer Vision and Pattern Recognition*, 2016.
- [95] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

- [96] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. *arXiv preprint arXiv:1812.01187*, 2018.
- [97] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition*, 2019.
- [98] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). *arXiv preprint arXiv:1606.08415*, 2016.
- [99] Byeongho Heo, Sangdoon Yun, Dongyoon Han, Sanghyuk Chun, Junsuk Choe, and Seong Joon Oh. Rethinking spatial dimensions of vision transformers. *arXiv preprint arXiv:2103.16302*, 2021.
- [100] Geoffrey E. Hinton, Oriol Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.
- [101] Elad Hoffer, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoeftler, and Daniel Soudry. Augment your batch: Improving generalization through instance repetition. In *Conference on Computer Vision and Pattern Recognition*, 2020.
- [102] Grant Van Horn, Oisín Mac Aodha, Yang Song, Alexander Shepard, Hartwig Adam, Pietro Perona, and Serge J. Belongie. The inaturalist challenge 2017 dataset. *arXiv preprint arXiv:1707.06642*, 2017.
- [103] Grant Van Horn, Oisín Mac Aodha, Yang Song, Alexander Shepard, Hartwig Adam, Pietro Perona, and Serge J. Belongie. The iNaturalist species classification and detection dataset. *arXiv preprint arXiv:1707.06642*, 2017.
- [104] Grant Van Horn, Oisín Mac Aodha, Yang Song, Alexander Shepard, Hartwig Adam, Pietro Perona, and Serge J. Belongie. The inaturalist challenge 2018 dataset. *arXiv preprint arXiv:1707.06642*, 2018.
- [105] A. Howard, Mark Sandler, G. Chu, Liang-Chieh Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Quoc V. Le, and H. Adam. Searching for MobileNetV3. *International Conference on Computer Vision*, 2019.
- [106] Yen-Chang Hsu, Zhaoyang Lv, and Zsolt Kira. Deep image category discovery using a transferred similarity function. *arXiv preprint arXiv:1612.01253*, 2016.
- [107] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.
- [108] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, 2016.
- [109] Xiao Shi Huang, Felipe Perez, Jimmy Ba, and Maksims Volkovs. Improving transformer optimization through better initialization. In *International Conference on Machine Learning*, pages 4475–4483. PMLR, 2020.
- [110] Mi-Young Huh, Pulkit Agrawal, and Alexei A. Efros. What makes imagenet good for transfer learning? *arXiv preprint arXiv:1608.08614*, 2016.
- [111] Maximilian Ilse, Jakub M. Tomczak, and Max Welling. Attention-based deep multiple instance learning. *International Conference on Machine Learning*, 2018.
- [112] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.
- [113] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

- [114] Andrew Jaegle, Sebastian Borgeaud, Jean-Baptiste Alayrac, Carl Doersch, Catalin Ionescu, David Ding, Skanda Koppula, Andrew Brock, Evan Shelhamer, Olivier J. H'enaff, Matthew M. Botvinick, Andrew Zisserman, Oriol Vinyals, and João Carreira. Perceiver io: A general architecture for structured inputs & outputs. *arXiv preprint arXiv:2107.14795*, 2021.
- [115] Xu Ji, Andrea Vedaldi, and João F. Henriques. Invariant information clustering for unsupervised image classification and segmentation. *International Conference on Computer Vision*, 2019.
- [116] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *International Conference on Learning Representations*, 2017.
- [117] Yuehaw Khoo and Lexing Ying. Switchnet: a neural network model for forward and inverse scattering problems. *SIAM Journal on Scientific Computing*, 41(5):A3182–A3201, 2019.
- [118] Diederik P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible  $1 \times 1$  convolutions. *arXiv preprint arXiv:1807.03039*, 2018.
- [119] Diederik P. Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible  $1 \times 1$  convolutions. In *NeurIPS*, 2018.
- [120] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. *arXiv preprint arXiv:1912.11370*, 6:3, 2019.
- [121] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *IEEE Workshop on 3D Representation and Recognition*, 2013.
- [122] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.
- [123] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012.
- [124] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, CIFAR, 2009.
- [125] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Mallocci, Tom Duerig, and Vittorio Ferrari. The open images dataset v4. *International Journal of Computer Vision*, 2020.
- [126] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.
- [127] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam R. Kosiosek, Seungjin Choi, and Yee Whye Teh. Set transformer. *International Conference on Machine Learning*, 2019.
- [128] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. *Conference on Computer Vision and Pattern Recognition*, 2019.
- [129] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, 2014.
- [130] Zhouhan Lin, Roland Memisevic, and Kishore Konda. How far can we go without convolution: Improving fully-connected networks. *arXiv preprint arXiv:1511.02580*, 2015.
- [131] Tom Ching LingChen, Ava Khonsari, Amirreza Lashkari, Mina Rafi Nazari, Jaspreet Singh Sambee, and Mario A. Nascimento. Uniformaugmt: A search-free probabilistic data augmentation approach. *arXiv preprint arXiv:2003.14348*, 2020.

- [132] Ruiyang Liu, Yinghui Li, Dun Liang, Linmi Tao, Shi-Min Hu, and Hai-Tao Zheng. Are we ready for a new paradigm shift? a survey on visual deep mlp, 2021.
- [133] Yang Liu, Yao Zhang, Yixin Wang, Feng Hou, Jin Yuan, Jiang Tian, Yang Zhang, Zhongchao Shi, Jianping Fan, and Zhiqiang He. A survey of visual transformers, 2021.
- [134] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. *arXiv preprint arXiv:2103.14030*, 2021.
- [135] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *arXiv preprint arXiv:2201.03545*, 2022.
- [136] I. Loshchilov and F. Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.
- [137] David G. Lowe. Object recognition from local scale-invariant features. *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999.
- [138] Wei-Ying Ma and B. S. Manjunath. Netra: A toolbox for navigating large image databases. 1999.
- [139] Martin Renqiang Min, L. V. D. Maaten, Zineng Yuan, A. Bonner, and Z. Zhang. Deep supervised t-distributed embedding. In *International Conference on Machine Learning*, 2010.
- [140] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature communications*, 9(1):1–12, 2018.
- [141] Yair Movshovitz-Attias, A. Toshev, T. Leung, S. Ioffe, and S. Singh. No fuss distance metric learning using proxies. *International Conference on Computer Vision*, 2017.
- [142] Samuel Müller and Frank Hutter. Trivialaugmt: Tuning-free yet state-of-the-art data augmentation. *arXiv preprint arXiv:2103.10158*, 2021.
- [143] Behnam Neyshabur. Towards learning convolutions from scratch. In *NeurIPS*, 2020.
- [144] M-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.
- [145] Roman Novak, Lechao Xiao, Jaehoon Lee, Yasaman Bahri, Greg Yang, Jiri Hron, Daniel A Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Bayesian deep convolutional networks with many channels are Gaussian processes. In *International Conference on Learning Representations*, 2019.
- [146] Alexander Novikov, Dmitrii Podoprikin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Neurips*, 2015.
- [147] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition*, 2014.
- [148] Maxime Oquab, Léon Bottou, Ivan Laptev, and Josef Sivic. Is object localization for free?-weakly-supervised learning with convolutional neural networks. In *Conference on Computer Vision and Pattern Recognition*, 2015.
- [149] Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. Scaling neural machine translation. In *ACL*, pages 1–9. Association for Computational Linguistics, 2018.
- [150] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pages 8026–8037, 2019.

- [151] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Conference on Computer Vision and Pattern Recognition*, 2007.
- [152] David Picard. `torch.manual_seed(3407)` is all you need: On the influence of random seeds in deep learning architectures for computer vision. *arXiv preprint arXiv:2109.08203*, sep 2021.
- [153] Filip Radenović, Giorgos Tolias, and Ondrej Chum. Fine-tuning CNN image retrieval with no human annotation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(7):1655 – 1668, 2018.
- [154] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.
- [155] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.
- [156] Ilija Radosavovic, Raj Prateek Kosaraju, Ross B. Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. *Conference on Computer Vision and Pattern Recognition*, 2020.
- [157] Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2020.
- [158] Prajit Ramachandran, Niki Parmar, Ashish Vaswani, I. Bello, Anselm Levskaya, and Jonathon Shlens. Stand-alone self-attention in vision models. In *Neurips*, 2019.
- [159] B. Recht, Rebecca Roelofs, L. Schmidt, and V. Shankar. Do ImageNet classifiers generalize to ImageNet? In *International Conference on Machine Learning*, 2019.
- [160] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 2016.
- [161] Oren Rippel, Manohar Paluri, Piotr Dollár, and Lubomir D. Bourdev. Metric learning with adaptive density discrimination. *International Conference on Learning Representations*, 2016.
- [162] Carlos Riquelme, Joan Puigcerver, Basil Mustafa, Maxim Neumann, Rodolphe Jenatton, André Susano Pinto, Daniel Keysers, and Neil Houlsby. Scaling vision with sparse mixture of experts. In *NeurIPS*, 2021.
- [163] Marko Ristin, Juergen Gall, Matthieu Guillaumin, and Luc Van Gool. From categories to subcategories: Large-scale image classification with partial class label refinement. *Conference on Computer Vision and Pattern Recognition*, 2015.
- [164] A. Romero, Nicolas Ballas, S. Kahou, Antoine Chassang, C. Gatta, and Yoshua Bengio. Fit-nets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2015.
- [165] F Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. 1962.
- [166] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 2019.
- [167] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
- [168] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

- [169] R. Salakhutdinov and Geoffrey E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *AISTATS*, 2007.
- [170] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. 1986.
- [171] Mark Sandler, Jonathan Baccash, Andrey Zhmoginov, and Andrew Howard. Non-discriminative data or weak model? on the relative importance of data and model resolution, 2019.
- [172] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128, 2019.
- [173] Noam Shazeer, Zhenzhong Lan, Youlong Cheng, N. Ding, and L. Hou. Talking-heads attention. *arXiv preprint arXiv:2003.02436*, 2020.
- [174] Zhuoran Shen, Irwan Bello, Raviteja Vemulapalli, Xuhui Jia, and Ching-Hui Chen. Global self-attention networks for image recognition. *arXiv preprint arXiv:2010.03019*, 2020.
- [175] Zhuoran Shen, Irwan Bello, Raviteja Vemulapalli, Xuhui Jia, and Ching-Hui Chen. Global self-attention networks for image recognition. *arXiv preprint arXiv:2010.03019*, 2020.
- [176] Patrice Y Simard, David Steinkraus, John C Platt, et al. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, 2003.
- [177] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.
- [178] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *Workshop at International Conference on Learning Representations*, 2014.
- [179] Josef Sivic and Andrew Zisserman. Video google: a text retrieval approach to object matching in videos. *Proceedings Ninth IEEE International Conference on Computer Vision*, 2003.
- [180] Kihyuk Sohn, David Berthelot, C. Li, Zizhao Zhang, N. Carlini, E. D. Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685*, 2020.
- [181] R. Srivastava, Klaus Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015.
- [182] R. Srivastava, Klaus Greff, and J. Schmidhuber. Training very deep networks. In *NeurIPS*, 2015.
- [183] Andreas Steiner, Alexander Kolesnikov, Xiaohua Zhai, Ross Wightman, Jakob Uszkoreit, and Lucas Beyer. How to train your vit? data, augmentation, and regularization in vision transformers. *arXiv preprint arXiv:2106.10270*, 2021.
- [184] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Conference on Computer Vision and Pattern Recognition*, 2015.
- [185] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *Conference on Computer Vision and Pattern Recognition*, 2016.
- [186] Fariborz Taherkhani, Hadi Kazemi, Ali Dabouei, Jeremy Dawson, and Nasser M. Nasrabadi. A weakly supervised fine label classifier enhanced by coarse supervision. In *International Conference on Computer Vision*, 2019.
- [187] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.



- [188] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, 2021.
- [189] Eu Wern Teh, Terrance Devries, and Graham W. Taylor. Proxynca++: Revisiting and revitalizing proxy neighborhood component analysis. *European Conference on Computer Vision*, 2020.
- [190] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Lijia Li. Yfcc100m: the new data in multimedia research. *Commun. ACM*, 2016.
- [191] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-Mixer: An all-MLP architecture for vision. *arXiv preprint arXiv:2105.01601*, 2021.
- [192] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, M. Cord, Alaaeldin El-Nouby, Edouard Grave, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. ResMLP: feedforward networks for image classification with data-efficient training. *arXiv preprint arXiv:2105.03404*, 2021.
- [193] Hugo Touvron, M. Cord, M. Douze, F. Massa, Alexandre Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. *International Conference on Machine Learning*, 2021.
- [194] Hugo Touvron, Matthieu Cord, and Hervé Jégou. Deit iii: Revenge of the vit. *arXiv preprint arXiv:2204.07118*, 2022.
- [195] Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Piotr Bojanowski, Armand Joulin, Gabriel Synnaeve, and Hervé Jégou. Augmenting convolutional networks with attention-based aggregation. *arXiv preprint arXiv:2112.13692*, 2021.
- [196] Hugo Touvron, Matthieu Cord, Alaaeldin El-Nouby, Jakob Verbeek, and Hervé Jégou. Three things everyone should know about vision transformers. *arXiv preprint arXiv:2203.09795*, 2022.
- [197] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers. *International Conference on Computer Vision*, 2021.
- [198] Hugo Touvron, Alexandre Sablayrolles, M. Douze, M. Cord, and H. Jégou. Graft: Learning fine-grained image representations with coarse labels. *International Conference on Computer Vision*, 2021.
- [199] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Herve Jegou. Fixing the train-test resolution discrepancy. *Neurips*, 2019.
- [200] Hugo Touvron, Andrea Vedaldi, Matthijs Douze, and Hervé Jégou. Fixing the train-test resolution discrepancy: Fixefficientnet. *arXiv preprint arXiv:2003.08237*, 2020.
- [201] Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. Do deep convolutional nets really need to be deep and convolutional? In *International Conference on Learning Representations*, 2017.
- [202] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 2008.
- [203] Wouter Van Gansbeke, Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Scan: Learning to classify images without labels. In *European Conference on Computer Vision*, 2020.

- [204] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.
- [205] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.
- [206] Huy V. Vo, Francis Bach, Minsu Cho, Kai Han, Yann LeCun, Patrick Pérez, and Jean Ponce. Unsupervised image matching and object discovery as optimization. *Conference on Computer Vision and Pattern Recognition*, pages 8279–8288, 2019.
- [207] Huy V. Vo, Patrick Pérez, and Jean Ponce. Toward unsupervised, multi-object discovery in large-scale image collections. *arXiv preprint arXiv:2007.02662*, 2020.
- [208] Apoorv Vyas, Angelos Katharopoulos, and Franccois Fleuret. Fast transformers with clustered attention. *arXiv preprint arXiv:2007.04825*, 2020.
- [209] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical report, California Institute of Technology, 2011.
- [210] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021.
- [211] Wenhai Wang, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao. Pyramid vision transformer: A versatile backbone for dense prediction without convolutions. *arXiv preprint arXiv:2102.12122*, 2021.
- [212] X. Wang, Ross B. Girshick, A. Gupta, and Kaiming He. Non-local neural networks. *Conference on Computer Vision and Pattern Recognition*, 2018.
- [213] Longhui Wei, An Xiao, Lingxi Xie, X. Chen, Xiaopeng Zhang, and Q. Tian. Circumventing outliers of autoaugment with knowledge distillation. In *European Conference on Computer Vision*, 2020.
- [214] Ross Wightman. Pytorch image models. <https://github.com/rwightman/pytorch-image-models>, 2019.
- [215] Ross Wightman, Hugo Touvron, and Hervé Jégou. Resnet strikes back: An improved training procedure in timm. *arXiv preprint arXiv:2110.00476*, 2021.
- [216] Bichen Wu, Chenfeng Xu, Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Masayoshi Tomizuka, Kurt Keutzer, and Péter Vajda. Visual transformers: Token-based image representation and processing for computer vision. *arXiv preprint arXiv:2006.03677*, 2020.
- [217] Cinna Wu, M. Tygert, and Y. LeCun. A hierarchical loss and its problems when classifying non-hierarchically. *PLoS ONE*, 2019.
- [218] Haiping Wu, Bin Xiao, Noel C. F. Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers. *arXiv preprint arXiv:2103.15808*, 2021.
- [219] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.
- [220] Zhirong Wu, Alexei A Efros, and Stella Yu. Improving generalization via scalable neighborhood component analysis. In *European Conference on Computer Vision*, 2018.
- [221] L. Xiao, Y. Bahri, Jascha Sohl-Dickstein, S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10, 000-layer vanilla convolutional neural networks. *arXiv preprint arXiv:1806.05393*, 2018.

- [222] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *European Conference on Computer Vision*, 2018.
- [223] Tete Xiao, Yingcheng Liu, Bolei Zhou, Yuning Jiang, and Jian Sun. Unified perceptual parsing for scene understanding. In *European Conference on Computer Vision*, 2018.
- [224] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross Girshick. Early convolutions help transformers see better. *arXiv preprint arXiv:2106.14881*, 2021.
- [225] Tete Xiao, Mannat Singh, Eric Mintun, Trevor Darrell, Piotr Dollár, and Ross B. Girshick. Early convolutions help transformers see better. *arXiv preprint arXiv:2106.14881*, 2021.
- [226] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, A. Yuille, and Quoc V. Le. Adversarial examples improve image recognition. *Conference on Computer Vision and Pattern Recognition*, 2020.
- [227] Qizhe Xie, Zihang Dai, Eduard H. Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation for consistency training. *arXiv preprint arXiv:1904.12848*, 2019.
- [228] Qizhe Xie, Eduard H. Hovy, Minh-Thang Luong, and Quoc V. Le. Self-training with noisy student improves imagenet classification. *arXiv preprint arXiv:1911.04252*, 2019.
- [229] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Conference on Computer Vision and Pattern Recognition*, 2017.
- [230] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *Conference on Computer Vision and Pattern Recognition*, 2017.
- [231] Saining Xie, Tianbao Yang, X. Wang, and Y. Lin. Hyper-class augmented and regularized deep learning for fine-grained image classification. In *Conference on Computer Vision and Pattern Recognition*, 2015.
- [232] Ismet Zeki Yalniz, Hervé Jégou, Kan Chen, Manohar Paluri, and Dhruv Kumar Mahajan. Billion-scale semi-supervised learning for image classification. *arXiv preprint arXiv:1905.00546*, 2019.
- [233] Xueting Yan, Ishan Misra, Abhinav Gupta, Deepti Ghadiyaram, and Dhruv Kumar Mahajan. Clusterfit: Improving generalization of visual representations. *Conference on Computer Vision and Pattern Recognition*, 2020.
- [234] Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. Large batch optimization for deep learning: Training BERT in 76 minutes. In *International Conference on Learning Representations*, 2020.
- [235] L. Yuan, Y. Chen, Tao Wang, Weihao Yu, Yujun Shi, F. Tay, Jiashi Feng, and S. Yan. Tokens-to-token vit: Training vision transformers from scratch on imagenet. *arXiv preprint arXiv:2101.11986*, 2021.
- [236] L. Yuan, F. Tay, G. Li, T. Wang, and Jiashi Feng. Revisit knowledge distillation: a teacher-free framework. *Conference on Computer Vision and Pattern Recognition*, 2020.
- [237] Sangdoon Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization strategy to train strong classifiers with localizable features. *arXiv preprint arXiv:1905.04899*, 2019.
- [238] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision*, 2014.
- [239] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. *arXiv preprint arXiv:2106.04560*, 2021.

- [240] Xiaohua Zhai, A. Oliver, A. Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. *International Conference on Computer Vision*, 2019.
- [241] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Zhi Zhang, Haibin Lin, Yue Sun, Tong He, Jonas Muller, R. Manmatha, Mu Li, and Alexander Smola. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020.
- [242] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
- [243] Hongyi Zhang, Yann Dauphin, and Tengyu Ma. Fixup initialization: Residual learning without normalization. *arXiv preprint arXiv:1901.09321*, 2019.
- [244] Pengchuan Zhang, Xiyang Dai, Jianwei Yang, Bin Xiao, Lu Yuan, Lei Zhang, and Jianfeng Gao. Multi-scale vision longformer: A new vision transformer for high-resolution image encoding. *arXiv preprint arXiv:2103.15358*, 2021.
- [245] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020.
- [246] Bolei Zhou, Aditya Khosla, Àgata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. *Conference on Computer Vision and Pattern Recognition*, 2016.
- [247] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. *Conference on Computer Vision and Pattern Recognition*, 2017.
- [248] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 2018.
- [249] Jinghao Zhou, Chen Wei, Huiyu Wang, Wei Shen, Cihang Xie, Alan Loddon Yuille, and Tao Kong. ibot: Image bert pre-training with online tokenizer. *arXiv preprint arXiv:2111.07832*, 2021.