



HAL
open science

Content combination strategies for Image Classification

Rémy Sun

► **To cite this version:**

Rémy Sun. Content combination strategies for Image Classification. Computer Science [cs]. Sorbonne Université, 2022. English. NNT: . tel-03952815v1

HAL Id: tel-03952815

<https://hal.science/tel-03952815v1>

Submitted on 23 Jan 2023 (v1), last revised 6 Feb 2023 (v2)

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)

Content combination strategies for Image Classification
Stratégies de combinaison de contenus pour la classification d'images

Présentée par
Rémy Sun

Dirigée par
Pr. Matthieu CORD et Pr. Nicolas THOME

Pour obtenir le grade de
DOCTEUR de SORBONNE UNIVERSITÉ

Présentée et soutenue publiquement le 17/10/2022

Devant le jury composé de :

Pr. Frédéric PRECIOSO <i>Professeur, Université Côte d'Azur</i>	Rapporteur
Pr. Céline HUDELLOT <i>Professeur, Centrale-Supélec</i>	Rapporteuse
Pr. Elisa FROMONT <i>Professeur, Université Rennes 1</i>	Examinatrice
Pr. Catherine ACHARD <i>Professeur, Sorbonne Université</i>	Examinatrice
Dr. Hervé JÉGOU <i>Directeur, Research Scientist, Meta, FAIR</i>	Examineur
Pr. Matthieu CORD <i>Professeur, Sorbonne Université</i>	Directeur de thèse
Pr. Nicolas THOME <i>Professeur, Sorbonne Université</i>	Co-Directeur de thèse
M. Clément MASSON <i>Research engineer</i>	Encadrant (Invité)

CONTENTS

CONTENTS	iii
LIST OF FIGURES	v
LIST OF TABLES	vii
REMERCIEMENTS	ix
ABSTRACT	i
RÉSUMÉ	iii
ACRONYMS	v
COMMON NOTATION CONVENTIONS	vii
1 INTRODUCTION	1
1.1 Learning to classify images with neural networks	2
1.2 Thesis Focus: Mixing Samples Data augmentations	7
1.3 Positioning	8
1.4 Summary table of publications	11
2 IN-CLASS MIXING SAMPLES DATA AUGMENTATIONS	13
2.1 Introduction	13
2.2 Related Work	16
2.3 SAMOSA: Adding non-semantic variations to an image	22
2.4 SciMix: Embedding semantic content into other contexts	31
2.5 Results	36
2.6 Conclusion	46
3 MSDA AS COMPRESSED REPRESENTATIONS FOR MIMO TRAINING	49
3.1 Introduction	49
3.2 Related Work	51
3.3 MixMo: Multi-Input Multi-Output MSDA	53
3.4 MixShare: Feature sharing between MIMO subnetworks	69
3.5 Conclusion	76
4 MULTI-INPUT MULTI-OUTPUT MSDA, UNMIXING AND ATTENTION MECHANISMS	79
4.1 Introduction	79
4.2 Related Work	80
4.3 MixViT: A MIMO MSDA formulation of Vision Transformers	83
4.4 Results of the MixViT framework	89
4.5 Conclusion	95
5 CONCLUSION	97
5.1 Main contributions	97
5.2 Perspectives	99

BIBLIOGRAPHY	103
A APPENDIX	117
A.1 Experimental details	117
A.2 Pseudo-code algorithms	133
A.3 Additional experiments and material for SAMOSA	139
A.4 Additional experiments and material for SciMix	144
A.5 Additional experiments and material for MixMo	148
A.6 Additional experiments and material for MixShare	156
A.7 Additional experiments and material for MixViT	159

LIST OF FIGURES

CHAPTER 1: INTRODUCTION		1
Figure 1.1	Illustration of Neural Networks.	3
Figure 1.2	Examples of layer functions.	4
Figure 1.3	Illustration of Mixing Sample Data Augmentations.	7
CHAPTER 2: IN-CLASS MIXING SAMPLES DATA AUGMENTATIONS		13
Figure 2.1	Illustration of hybrids yielded by standard MSDA and ours.	15
Figure 2.2	Overview of the Mean Teacher framework.	17
Figure 2.3	Overview of the FixMatch framework.	18
Figure 2.4	Overview of the AdaIN framework	20
Figure 2.5	Example of an Image-to-Image translation framework.	21
Figure 2.6	Illustration of the separation of contents in MSDA.	23
Figure 2.7	Overview of the SAMOSA framework.	24
Figure 2.8	Overview of the asymmetrical SAMOSA decoder.	26
Figure 2.9	Optimization of \mathcal{L}_{SAMOSA} in the modules.	27
Figure 2.10	Hybridization in SAMOSA.	28
Figure 2.11	Examples of hybrids obtained by SAMOSA.	30
Figure 2.12	Overview of the SciMix generator architecture.	33
Figure 2.13	Examples of SAMOSA hybrids.	41
Figure 2.14	Examples of SciMix hybrids.	41
Figure 2.15	SciMix hybrids on CUB-200.	45
CHAPTER 3: MSDA AS COMPRESSED REPRESENTATIONS FOR MIMO TRAINING		49
Figure 3.1	Overview of the MIMO framework.	52
Figure 3.2	Overview of the MixMo framework.	54
Figure 3.3	Illustration of MixMo’s training scheme.	57
Figure 3.4	Influence of MixMo on network utilization.	58
Figure 3.5	Parameter efficiency of MixMo.	62
Figure 3.6	Ensemble effectiveness of MixMo.	63
Figure 3.7	Influence of training time on MixMo.	64
Figure 3.8	Influence of the mixing probability in MixMo.	65
Figure 3.9	Influence of the weighting function in MixMo.	66
Figure 3.10	MixMo performance for $M \geq 2$	67
Figure 3.11	Influence of the feature maps for the subnetworks.	71

Figure 3.12	Usage of features by MixMo subnetworks.	72
Figure 3.13	Overview of the MixShare framework.	72
Figure 3.14	Usage of features by MixMo subnetworks with unmixing.	73
CHAPTER 4: MULTI-INPUT MULTI-OUTPUT MSDA, UNMIX- ING AND ATTENTION MECHANISMS		79
Figure 4.1	Overview of the seminal ViT framework.	82
Figure 4.2	Overview of MIMO architectures.	83
Figure 4.3	Overview of our MixViT framework.	87
Figure 4.4	Influence of batch augmentations on MixViT.	92
CHAPTER 5: CONCLUSION		97
APPENDIX A: APPENDIX		117
Figure A.1	Examples of SAMOSA hybrids with MixMatch.	139
Figure A.2	Hybrids for MNIST-M in SAMOSA.	142
Figure A.3	Additional SciMix hybrids on SVHN.	147
Figure A.4	Illustration of common MSDA procedures.	149
Figure A.5	Plots of the reweighting operation.	151
Figure A.6	Ensemble effectiveness with CutMix in MixMo.	153
Figure A.7	α trade-off in MixMo.	155
Figure A.8	Influence of feature maps in the core network.	157
Figure A.9	Transposition of traditional MIMO architectures to vision transformers.	159

LIST OF TABLES

Table 1.1	Table of publications and thesis chapters.	11
CHAPTER 2: IN-CLASS MIXING SAMPLES DATA AUGMENTATIONS		13
Table 2.1	Gains from using SAMOSA on CIFAR10.	37
Table 2.2	Gains from using SciMix.	39
Table 2.3	Ablation on the gains from in-class MSDA.	42
Table 2.4	Quantitative comparison of SAMOSA and SciMix	43
Table 2.5	Transfer rates of semantic and non-semantic contents.	44
Table 2.6	Comparison of SciMix with other MSDA on very few labels.	44
Table 2.7	Gains from using SciMix on CUB-200	45
CHAPTER 3: MSDA AS COMPRESSED REPRESENTATIONS FOR MIMO TRAINING		49
Table 3.1	Performance of the MixMo framework on CIFAR-10 and 100.	61
Table 3.2	Robustness comparison on CIFAR-100-c.	63
Table 3.3	Influence of the MSDA used in the mixing block.	65
Table 3.4	Importance of the encoder/decoders in MixMo.	66
Table 3.5	MixMo performance on Tiny ImageNet	68
Table 3.6	Performance of MixShare.	74
CHAPTER 4: MULTI-INPUT MULTI-OUTPUT MSDA, UNMIXING AND ATTENTION MECHANISMS		79
Table 4.1	Performance of MIMO formulations of ViTs.	85
Table 4.2	Gains from using MixViT.	91
Table 4.3	Comparison of MixViT and MixMo.	91
Table 4.4	CIFAR-100 accuracy with MSDA/BA.	92
Table 4.5	Comparison of MixViT against SOTA ViT performances.	93
Table 4.6	Ablation on MixViT design choices.	94
APPENDIX A: APPENDIX		117
Table A.1	General structure of E_c in SAMOSA.	120
Table A.2	General structure of E_r in SAMOSA.	120
Table A.3	General structure of D in SAMOSA.	121
Table A.4	General structure of C in SAMOSA.	121
Table A.5	WideResBlock of F filters ands stride s.	121
Table A.6	DeconvBlock of F filters ands stride s.	122

Table A.7	General structure of C in SciMix.	124
Table A.8	General structure of E_c in SciMix.	125
Table A.9	General structure of E_r in SciMix.	125
Table A.10	General structure of G in SciMix.	125
Table A.11	General structure of D in SciMix.	126
Table A.12	Accuracy at the end of training for MixMo experiments. .	128
Table A.13	Component separation in SAMOSA.	141
Table A.14	Comparison of various architectural variants of SciMix . .	145
Table A.15	Comparison of various losses in SciMix data augmentation.	147
Table A.16	Proportion (%) of active filters in MixMo networks.	152
Table A.17	Summary table of split advantages in MixMo.	154
Table A.18	Impact of initialization when transposing MIMO trans- formers.	160
Table A.19	Comparison between MixMo and MixViT on CIFAR-100. .	162

REMERCIEMENTS

Il me reste le souvenir d'un matin à la cafétéria de l'ISTIC avec Luc Bougé où, sans doute exaspéré par une autre de mes interminables tergiversations, il m'a donné l'un des plus précieux conseils de ma vie. Les années et les expériences m'auront malheureusement soutiré l'exacte formulation, mais je permettrai de relater ici ce que j'en ai retiré: la recherche - comme la vie - s'écrit de jour en jour, au fil des hasards, des idées, des coïncidences, des succès, des imprévus, des tribulations, des opportunités, des convictions. Et au travers des personnalités dont on a la chance de croiser le chemin.

J'aimerais - avant de commencer en bonne et dûe forme - prendre le temps de remercier les personnes qui m'ont permis d'aboutir à l'écriture de ce manuscrit.

Je remercie avant toute chose mon directeur de thèse Matthieu Cord, et mon co-directeur de thèse Nicolas Thome. Matthieu, je te remercie pour m'avoir accueilli dans ton équipe, accompagné dans mes travaux, et accommodé dans mes excentricités en ces temps troublés. Si j'ai pu conduire cette thèse en toute tranquillité d'esprit, c'est grâce à ton accessibilité et ta bienveillance. J'ai beaucoup grandi à ton contact et par ton exemple, aussi bien scientifiquement qu'humainement. Comme tu le sais bien, il m'est facile de me perdre dans mes investigations mais je pense avoir au moins appris à le reconnaître et surveiller après 3 ans. Nicolas, je te suis profondément reconnaissant pour ton importante contribution dans les travaux de cette thèse, et ton infatigable aplomb. J'ai - moi-même et mes travaux - grandement bénéficié de tes nombreux questionnements et remarques. Et à vous deux, j'ai toujours pris à coeur vos réserves et critiques, même si j'ai pu sembler obtus par moments.

Mes remerciements vont aussi à Thales. Par Thales, je pense certes à la société qui m'a financé durant ces travaux, mais surtout aux personnes que j'ai pu côtoyé ces trois dernières années. D'abord, Clément pour m'avoir suivi au jour le jour, gardé attentif dans mes raisonnements et avoir toujours su se rendre disponible quand j'en ai eu besoin. Gilles aussi, pour son suivi et pour m'avoir fourni l'opportunité de discuter de mes travaux au sein de l'entreprise. Mes collègues doctorants au service François, Thomas et Thomas avec qui j'ai pu échanger et commiserer. Et bien sûr les collègues qui m'ont écouté parler d'apprentissage machine et échangé une plaisanterie ou deux au déjeuner même s'ils sont trop nombreux pour tous être cités.

Je remercie le jury pour l'intérêt porté à mes travaux, et au temps consacré à l'examen de mes travaux au milieu de leurs nombreuses obligations. Je souhaite

en particulier témoigner de ma reconnaissance à Frédéric Precioso et Céline Hudelot qui ont accepté de prendre le temps de rapporter ce (long) manuscrit. Je remercie aussi vivement les examinateurs Hervé Jégou, Elisa Fromont et Catherine Achard qui ont accepté d'examiner cette thèse.

Je tiens à remercier également les "chordettes", les autres doctorants de Matthieu (alphabétiquement !). Alexandre, avec qui j'aurai pas mal réfléchi. Antoine, toujours affable et plaisant. Arthur, souvent prêt à offrir son aide. Asya, dont l'anglais natif a parfois été bien utile. Corentin, pour les discussions et les indications. Fabio, qui a apporté un bout d'Italie à Jussieu. Guillaume, pour ses manipulations d'images qui m'ont toujours intéressé. Hugo, et son expertise sur les transformers. Mostafa, pour sa bonne humeur. Rémi, parce qu'il sait ce qu'est un montage 10-20. Yifu, pour sa présence rassurante durant mes premiers mois de thèse. Et les doctorants de Nicolas aussi. Charles, avec qui je devrais retourner chez Xu. Laura, qui a une oreille attentive. Olivier, pour son aide avec l'infrastructure du CNAM. Thuy, pour sa gentillesse. Vincent, qui m'a fait découvrir des techniques intéressantes. Je profite de l'occasion pour remercier les équipes scientifiques et administratives, au CNAM et surtout dans l'équipe MLIA à Jussieu. L'équipe MLIA m'a fourni un excellent environnement durant cette thèse. C'est aussi bien le fait de l'implication des permanents, du service informatique (que j'ai parfois dû embêter plus que de raison), et des nombreux doctorants (trop nombreux pour en faire l'inventaire) de tous horizons.

Enfin, je souhaite remercier mes professeurs de l'Ecole Normale Supérieure de Rennes à qui je dois aussi beaucoup. Luc Bougé, qui m'a accompagné à une période de ma vie où je ne savais pas vraiment où aller. David Pichardie, qui s'est toujours montré encourageant dans mes démarches. Hervé Jégou, même s'il n'a jamais été mon professeur, dont les conseils inestimables m'ont permis de construire mon projet et mon parcours. Aussi, David Cachera, Martin Quinson, Sophie Pinchinat et François Schwarzentruher qui m'ont appris bien des leçons sur l'enseignement et la méthodologie. Comprises des années plus tard certes, mais comprises tout de même. Je remercie aussi mes directeurs de stage de recherche: François Coste qui m'a fait découvrir l'apprentissage machine alors que je voulais faire de la bio-informatique, Christoph Lampert qui m'a fait comprendre ce qu'est la recherche en apprentissage, et Michel Besserve qui m'a pour la première fois forcé à vraiment réfléchir à ce qu'est un "bon mécanisme".

A MA FAMILLE

A ma petite soeur, qui m'a souvent mis la pression avec ses nombreux talents et sa lubie de me croire bien plus malin que je ne le suis réellement. A ma grand-mère pour qui ce manuscrit arrive 16 ans trop tard, qui m'aura décidément rempli la tête d'idées saugrenues. Et à mes parents qui ont fait les sacrifices nécessaires pour me soutenir et m'encourager dans mes choix tout au long de ma vie. Et si je ne l'ai jamais dit, je le sais, que j'ai eu beaucoup de chance de toujours pouvoir poser mes questions les plus étranges à la maison.

ABSTRACT

In this thesis, we tackle the question of deep image classification, a fundamental issue for computer vision and visual understanding in general. Faced with neural networks' need for large training datasets, we look into the common practice of engineering new examples to augment the dataset. We take this as an opportunity to teach neural algorithms to reconcile information mixed from different samples with Mixing Sample Data Augmentation so as to better understand the problem. To this end, we study both how to edit the content in a mixed image, and what the model should predict for the mixed images.

We first propose a new type of data augmentation that helps model generalize by embedding the semantic content of samples into the non-semantic context of other samples to generate in-class mixed samples. To this end, we design new neural architectures capable of generating such mixed samples, and then show the resulting mixed inputs help train stronger classifiers in a semi-supervised setting where few labeled samples are available.

In a second part, we show input mixing can be used as an input compression method to train multiple subnetworks in a base network from compressed inputs. Indeed, by formalizing the seminal multi-input multi-output (MIMO) framework as a mixing data augmentation and changing the underlying mixing mechanisms, we obtain strong improvements of over standard models and MIMO models. Furthermore, we shine a light on these models' tendency to train subnetworks that share no features and propose a solution by leveraging knowledge on the underlying input mixing.

Finally, we adapt this MIMO technique to the emerging Vision Transformer (ViT) models. Our work shows ViTs present unique challenges for MIMO training, but that they are also uniquely suited for it. We leverage ViTs' unique token based representations to introduce a source attribution mechanism that allows us to only train subnetworks in the last layers of the model. This causes the subnetworks to train a very strong shared feature extracting base while still being somewhat diverse and beneficial to model performance.

RÉSUMÉ

Dans cette thèse, nous nous attaquons au problème de la classification d'images, un problème fondamental pour la vision par ordinateur et le raisonnement visuel en général. Face la quantité énorme de données nécessaires pour entraîner des réseaux profonds, nous nous intéressons aux différentes façons d'augmenter artificiellement la taille du jeu de données. Plus précisément, nous mettons cette technique à profit pour apprendre aux algorithmes neuronaux à réconcilier l'information mixée à partir de différents exemples par le biais des augmentations de données mixantes afin de mieux comprendre le problème sous-jacent. A cette fin, nous étudions à la fois comment éditer le contenu mixé dans un exemple mixte et ce qu'un modèle devrait prédire sur une telle image.

Nous proposons d'abord un nouveau type d'augmentation qui aide le modèle à généraliser en incrustant le contenu sémantique d'un exemple dans le contexte non-sémantique d'un autre pour générer des exemples mixtes appartenant à une unique classe. Pour ce faire, nous proposons de nouvelles architectures permettant de générer de tels exemples, et montrons ensuite comment ces exemples mixtes aident à entraîner de meilleurs classificateurs dans un contexte semi-supervisé.

Dans un second temps, nous montrons que le mixage d'image peut être utilisé comme un schéma de compression d'entrées permettant d'entraîner de multiples sous-réseaux au sein d'un réseau de base. En effet, en formalisant la méthode séminale "multi-input multi-output" (MIMO) comme un schéma d'augmentation de données par mixage d'images et en changeant le mécanisme de mixage sous-jacent nous obtenons des gains en performance par rapports aux modèles classiques. De plus, nous mettons en lumière la tendance de ces modèles à entraîner des sous-réseaux ne partageant aucune feature et proposons une solution exploitant notre compréhension de mécanisme de mixage des entrées en jeu dans ces méthodes.

Finalement, nous adaptons ces derniers modèles MIMO aux récents modèles Vision Transformer. Nos travaux montrent que ces nouvelles architectures présentent leurs propres uniques incompatibilités avec l'entraînement MIMO, mais qu'elles y sont aussi extrêmement adaptées à d'autres égards. Nous tirons avantage de la représentation par tokens des ViTs pour introduire un nouveau mécanisme d'attribution de source qui permet d'entraîner des sous-réseaux uniquement dans les dernières couches du modèle. Cela mène les sous-réseaux à entraîner un tronc commun très robuste tout en conservant des sous réseaux relativement diversifiés et bénéficie à la performance des modèles.

ACRONYMS

AI	Artificial Intelligence
MSDA	Mixing Sample Data Augmentation
MIMO	Multi-input Multi-output
CV	Computer Vision
NLP	Natural Language Processing
DA	Data Augmentation
GPU	Graphics Processing Unit
FC	Fully Connected
ML	Machine Learning
DL	Deep Learning
MLP	Multi-Layer Perceptron
CNN	Convolutional Neural Network
ViT	Vision Transformers
ReLU	Rectified Linear Unit
BN	Batch Normalization
MSE	Mean-Squared Error
SGD	Stochastic Gradient Descent
SGLD	Stochastic Gradient Langevin Dynamic
SL	Supervised Learning
SSL	Semi-Supervised Learning
GAN	Generative Adversarial Network
GAP	Global Average Pooling
SA	Self-Attention
CA	Class-Attention
MHSA	Multi-Head Self-Attention
GPSA	Gated Positional Self-Attention

COMMON NOTATION CONVENTIONS

\mathcal{D}	A distribution or a dataset
x	A model input, usually an image
y	A label
z	An internal latent variable
h	Intermediate activation maps
f	A classifier function, typically a neural network
$d(\cdot)$	Dense layer functions
$c(\cdot)$	Convolutional layer functions
θ	Model parameters
\mathcal{R}	A risk
Ω	A regularizer term
\mathcal{L}	An aggregated loss function over multiple samples
l	A loss function on an individual instance
W	Linear weights or convolutional kernels
b	Bias terms
λ	A ponderation coefficient or ratio
\mathcal{M}	A mask (for mixing)
$\mathbf{1}$	An indicator function
M	Number of inputs in MIMO frameworks
L	Number layers
t	A transformer token
\odot	Hadamard product
$*$	Convolutional product

INTRODUCTION

Tomorrow owes you the sum of
your yesterdays. No more than that.
And no less.

Robin Hobb

Artificial Intelligence is a fashionable term to use in our day and age. One can find it most everywhere: from the newspaper stand to the university library, the morning radio to the evening entertainment, and - most importantly - from the common everyday chatter to the daydreams of the collective unconscious. It is after all easy to get lost in the many successes of Artificial intelligence. Already, Go (Silver et al. 2016) and Chess (Silver et al. 2018) champions learn strategies yet unheard of from neural network. New antibiotics are discovered (Stokes et al. 2020) owing to recent advances. Tomorrow promises this and more. Better automatic translation (Vaswani et al. 2017), more accurate film recommendations (Töscher and Jahrer 2009), and even conversation partners (Shang et al. 2015) are only a few examples of what the future might hold.

It is however important to keep in mind that this state of affairs is only the results of a slow accretion decades in the making. This remains true even if we were to discount earlier inventions such as Charles Babbage’s Analytical Engine (Babbage 1982) in the 1800s, Kurt Gödel’s theoretical work (Gödel 1931) and Alan Turing’s eponymous machine (Turing et al. 1936) under the pretext that they pertain to Computer Science per se. Indeed, much work has been done since the famed 1956 Dartmouth workshop that first coined the term of Artificial Intelligence (McCarthy et al. 1956). We do not purport here to provide definitions or history. Rather, we will say only this: the road from Dartmouth to today has seen many different paradigms (Nilsson 1982; Jordan 1999; Devroye et al. 2013) rule over the years, and the infamous “AI winters” researchers have had to push through.

For all that statistical learning (Devroye et al. 2013) has become the dominant paradigm in AI with the subfields of Machine Learning (Bishop 2006) and Deep Learning (Goodfellow et al. 2016), much remains to be done. Beyond the many

ethical concerns (Gebru 2019) raised by neural networks, a lot of questions remain open: Why do very large networks perform well? (Nakkiran et al. 2021) How should we optimize networks? (Kingma and Ba 2014) What deep architecture should we use? (He et al. 2016b; Vaswani et al. 2017) If Artificial Intelligence is to hold its promises for tomorrow, we must tackle these issues today.

This thesis attempts to add a few points to the colossal sum that has been tallying up ever since Dartmouth. Our work focuses on finding new techniques to classify images with Deep Learning (and Machine Learning). As the neural networks trained in Deep Learning are very reliant on the quality and quantity of the training data, it is common practice to engineer new examples to augment the dataset. We take this as an opportunity to teach neural algorithms to reconcile information mixed from different samples so as to better understand the problem. To this end, we study both how to edit the content in a mixed image, and what the model should predict for the mixed images. After a brief introduction to classifying images with neural networks (Section 1.1), we offer an overview of current Mixing Sample Data Augmentation (Section 1.2) and discuss our contributions (Section 1.3).

1.1 Learning to classify images with neural networks

Classifying images as a general problem can simply be understood as an issue of being able to associate any image x of a data distribution \mathcal{D} (Devroye et al. 2013) with the corresponding label y (e.g. associating a picture of a dog with the class “dog”). Formally, this is simply an issue of finding a mapping or function f such that

$$\forall (x, y) \in \mathcal{D}, f(x) = y. \quad (1.1)$$

The heart of the issue therefore lies in finding the right function for the data distribution. Alas, covering the entire set of possible functions f is unrealistic and would raise a number of additional issues beyond the computational considerations.

Machine Learning often aims to seek the best solution f from among a restricted and tractable set of functions (Bishop 2006). In fact, the common view is to consider parameterized functions $\{f_\theta\}$ as a family of possible functions, and strive to find the best possible θ . The reader might note here that this introduces an additional wrinkle: it is unlikely that a function f_θ that perfectly matches all images to their labels exists. If we add an additional criterion \mathcal{L} that quantifies

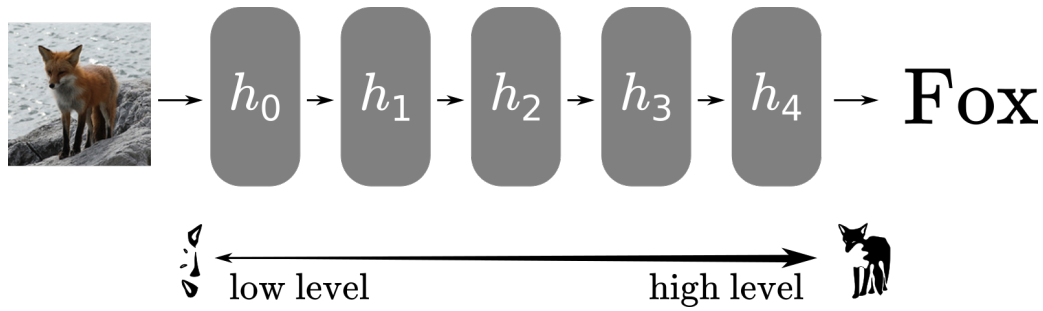


Figure 1.1. – **Illustration of Neural Networks.** Neural Networks use successive layer functions to extract more and more refined representations.

the gap between predictions and labels on the problem distribution, the goal therefore becomes to find - if it exists - an optimal θ such that

$$\arg \min_{\theta} \mathcal{L}(\{f_{\theta}(x), y\}_{(x,y) \in \mathcal{D}}). \quad (1.2)$$

In this thesis, we consider a family of parameterized functions commonly referred to as Neural Networks (Goodfellow et al. 2016). While there exist many other good families of parameterized functions (e.g. Support Vector Machines (Bishop 2006), Graphical models (Jordan 1999), Convex functions (Boyd et al. 2004), ...) for machine learning problems, Neural Networks have emerged in the last decade as the most powerful solution for Image Classification (Szeliski 2022) and present a number of interesting properties.

1.1.1 Neural networks

As per Goodfellow et al. 2016, Neural networks are typically built from multiple successive layers (see Figure 1.1), each of which are themselves parameterized functions h_{θ}^i . Therefore, the larger function f_{θ} can be understood to be

$$f_{\theta} = h_{\theta}^0 \circ h_{\theta}^1 \circ \dots \circ h_{\theta}^{L-1}, \quad (1.3)$$

with each layer h_{θ}^i performing fairly basic operations. For instance, as seen in Figure 1.2a the historical dense layer d_{θ} function with input size d and output size d' simply computes

$$d_{\theta}(x) = W_{\theta}x^T + b_{\theta}, \quad (1.4)$$

where W_{θ} is a $d' \times d$ matrix and b a d' -dimensional vector.

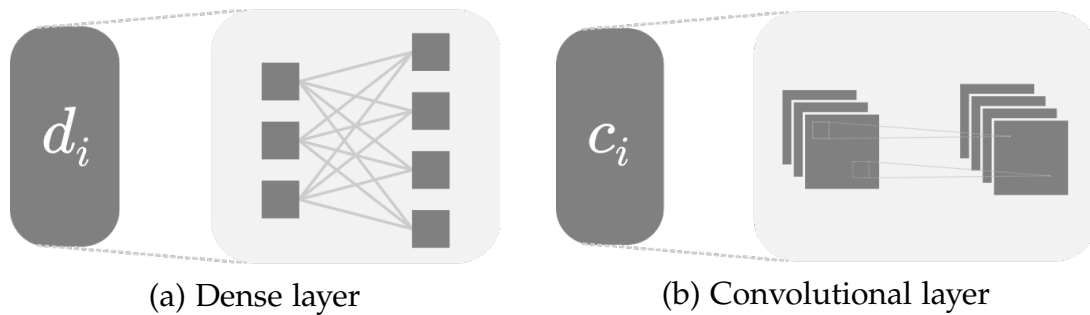


Figure 1.2. – Examples of layer functions.

As such, the general idea in neural networks is the following: each layer is very expressive (proven to be universal approximators for asymptotic sizes in Cybenko 1989), and each successive layer can build upon the representation given by its predecessor. Deep neural networks are intrinsically hierarchical in the functions they learn. They can afford to learn very low level notions in the first few layers, with the last layers learning very complex patterns from the low level motifs observed by the first few layers.

In practice, neural networks architectures are designed by hand and regarded as hyper-parameters instead of function parameters that must be optimized. As such, the default setting in Machine Learning is often that of a set neural architecture that defines a range of possible functions (Goodfellow et al. 2016). The optimization process only seeks to find the optimal parameters for this fixed architecture.

1.1.2 Example: Convolutional neural networks

It is perhaps easier to illustrate neural networks for computer vision on a concrete family of architectures: Convolutional neural networks (CNN). They have been for a number of years the gold standard architecture for Computer Vision, and remain competitive with the emerging Vision Transformer architectures (Dosovitskiy et al. 2021). As Chapter 2 and Chapter 3 build upon Convolutional architectures, we feel it is beneficial to provide a quick introduction here.

Convolutional neural networks (Goodfellow et al. 2016) mostly use two types of parameterized neural network layers: dense layers (introduced earlier) for classification, and convolutional layers for feature extraction. Contrarily to dense layers that operate on flat d -dimensional vectors, convolutional layers (see Figure 1.2b) for computer vision typically operate on spatial inputs with dimensions $C \times H \times W$ where H and W are spatial dimensions and C denotes a number of parallel channels (e.g. the RGB channels in a standard image). Beyond this, con-

volutional layers are however fairly similar to dense layers and a convolutional layer c can indeed be formalized in a fairly similar manner

$$c_{\theta}(x) = W_{\theta} * x^T + b_{\theta}, \quad (1.5)$$

where $*$ is the convolution (or more exactly, correlation) operation instead of a matrix product operation. The most significant difference here - which goes hand in hand with the $*$ operation - is that W_{θ} is a small $C \times C' \times K \times K$ convolution kernel instead of a weight matrix for an input with C channels and output with C' channels.

1.1.3 Supervised Learning

Given a family of parameterized functions - typically a neural network as defined previously - we simply have to find the function that most closely matches the distribution we want to approximate with the function. This therefore leaves the question of how we can quantify the gap between the distribution and the predictions given by the function.

Ideally, if we quantify the distance between a prediction \hat{y} and the true label y with a cross-entropy function (Bishop 2006)

$$l_{CE}(\hat{y}, y) = - \sum_{i=0}^{\#Classes-1} y_i \log(\hat{y}_i), \quad (1.6)$$

Supervised Machine Learning (Bishop 2006) aims to find f_{θ} that minimizes the expected value - or risk - of this distance over the entire distribution

$$\mathcal{R}(\mathcal{D}) = \mathbb{E}_{(x,y) \in \mathcal{D}} [l_{CE}(f_{\theta}(x), y)]. \quad (1.7)$$

In practice, we are not able to access the full distribution \mathcal{D} : if we were, we would have no need to learn a function f_{θ} ! As such, we are typically limited to training on a training dataset \mathcal{D}_{train} and we seek a solution that minimizes the empirical risk $\hat{\mathcal{R}}_{train}$ (Bishop 2006):

$$\arg \min_{\theta} \frac{1}{\#\mathcal{D}_{train}} \sum_{(x,y) \in \mathcal{D}_{train}} l_{\theta}(f_{\theta}(x), y). \quad (1.8)$$

The most widely adopted way to solve this optimization problem with neural networks is to perform stochastic gradient descent (Goodfellow et al. 2016). Gradient descent is an iterative procedure that slowly modifies the parameters

θ following the first order moment (gradient) of the empirical risk. Indeed, SGD samples (mini-)batches B of $\#B$ images from the training dataset to compute a loss objective over the batch

$$\hat{\mathcal{R}}_{\theta}(B) = \frac{1}{\#B} \sum_{(x,y) \in B} l_{CE}(f_{\theta}(x), y), \quad (1.9)$$

such that we can update the parameters with a step - or learning rate - η :

$$\theta_{update} = \theta - \eta \frac{\partial \hat{\mathcal{R}}_{\theta}(B)}{\partial \theta}. \quad (1.10)$$

1.1.4 Discussion

Most recent works have followed this general structure, and tend to focus on improving some components. The following provides a brief inventory of the most common lines of research.

Many neural architectures have been proposed over the years ranging from Convolutional Neural Network variants (He et al. 2016b) to the recent Vision Transformers (Vaswani et al. 2017). By designing and choosing different neural networks, we modify the set of possible functions f_{θ} that can possibly be chosen by the algorithm. In practice, this is often the occasion to include or exclude explicit biases in the type of content the functions should focus on.

The optimization process used to minimize the empirical risk is also often re-evaluated with alternative schemes. These schemes can implement completely new dynamics like Stochastic Gradient Langevin Dynamics (Welling and Teh 2011) or simply adapt the learning rate of SGD like Adam (Kingma and Ba 2014).

Interestingly, the training objective itself is a very common target of research. Beyond adding a number of regularizer terms (Krogh and Hertz 1991), the loss function used to evaluate the performance of the model on the training set has also been modified (Szegedy et al. 2016) a number of times. In fact, even the Empirical Risk Minimization paradigm sees some opposition in the form of Invariant Risk Minimization (Arjovsky et al. 2019) variants.

In any case, minimizing the empirical risk instead of the true risk means we have to hope the training data can properly represent the distribution \mathcal{D} . As such, the quality of this training set or distribution \mathcal{D}_{train} is of paramount importance. In practice, this can only be the case with very large training datasets which are rare and very expensive to curate. While there exist limited training settings like semi-supervised learning (Chapelle et al. 2006) that seek to alleviate the need for very large and expensive training datasets, techniques have also emerged over the

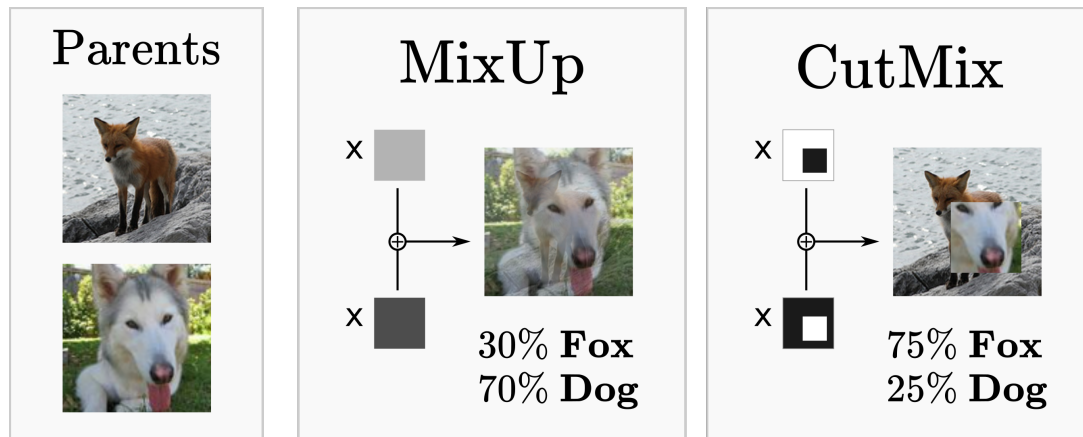


Figure 1.3. – **Illustration of Mixing Sample Data Augmentations.** MSDA combine contents from multiple images (e.g. a dog and a fox) to create new training data.

years to directly inflate the size of the training dataset (Perez and J. Wang 2017). This thesis studies this problem in particular.

1.2 Thesis Focus: Mixing Samples Data augmentations

Deep neural networks therefore work by modeling the training distribution they are fed. This requires the training distribution to be representative of the general application data, which is typically a difficult proposition as discussed above.

It is therefore mandatory to artificially augment the size of the training dataset through a variety of procedures often referenced under the common umbrella of Data Augmentation (Perez and J. Wang 2017; Gontijo-Lopes et al. 2021). Data Augmentation seeks to generate “new” training samples by perturbing samples and training the models to generalize to those perturbed samples. This procedure is however typically limited as modifying one sample too much introduces perturbations on the label of the new artificial samples that we cannot predict.

Mixing Sample Data Augmentation (MSDA) creates new samples (Figure 1.3) by combining characteristics from two or more images (H. Zhang et al. 2018; Y. Yang and Soatto 2019). This presents two advantages: the generated samples can be much more different from samples in the training set, and we have an idea of the perturbation induced on the label of one sample. As such, MSDA allows for a much more thorough augmentation of the available training data.

For instance, the seminal MSDA technique - MixUp (H. Zhang et al. 2018) - expands the training set to the convex hull of the training data. MixUp generates a new mixed labeled sample (\hat{x}, \hat{y}) from two images (x_0, y_0) and (x_1, y_1) (see Equation 1.11 and Equation 1.12) following a mixing ratio $\lambda \sim \beta(\alpha, \alpha)$:

$$\hat{x} = \lambda x_0 + (1 - \lambda)x_1, \quad (1.11)$$

with soft in-between class label

$$\hat{y} = \lambda y_0 + (1 - \lambda)y_1. \quad (1.12)$$

MixUp - or interpolation - based mixing augmentations are typically thought to help regularize the model by introducing noise on the samples (Carratino et al. 2020).

Another family of diversity inducing MSDA was introduced by the CutMix Augmentation (Y. Yang and Soatto 2019). Colloquially, CutMix extracts a square patch from one of the inputs (x_1 or x_2) and pastes it onto the other input. Formally, we draw a binary mask \mathcal{M} such that a square of values on the mask is set to 1 and the rest to 0, such that

$$\hat{x} = \mathcal{M} \odot x_0 + (1 - \mathcal{M}) \odot x_1. \quad (1.13)$$

Similarly to MixUp, CutMix follows a mixing ratio $\lambda \sim \beta(\alpha, \alpha)$. In CutMix, the mixing ratio dictates the area of the square mask used for mixing (ie, $Avg(\mathcal{M}) = \lambda$). Interestingly, it must be noted that this wider mask formalism also applies to MixUp with $\mathcal{M} = \lambda 1_{H \times W}$.

1.3 Positioning

This thesis was funded by Thales Land and Air Systems and Association Nationale de la Recherche et de la Technologie (ANRT) in the context of a CIFRE PhD. Thales Land and Air Systems builds electrical systems for aerospace, defense, transport and security. As part of its operations, the company's product must often work with a limited number of labeled samples for some objects that are either new or rarely appear. As such, it is especially important for Thales to find ways to inflate the size of its datasets. In particular, combining the contents of these rare images with other images could be particularly useful in a context like semi-supervised learning.

Mixing Samples Data Augmentations have the very peculiar characteristic of using soft labels \hat{y} instead of hard labels. More precisely, MSDA samples contain

information from multiple samples: they are between-class samples instead of true in-distribution samples. At first blush, soft labels are an elegant way to reflect this state of affairs, and bring some additional regularizing benefits like label smoothing (Szegedy et al. 2016).

In this thesis, we examine the issue of in-between MSDA samples and soft labels more closely to better understand the underlying mechanisms. Our work does not aim to invalidate the current paradigm, but instead explores possible avenues opened by challenging the use of in-between class mixed samples with soft labels. To the best of our knowledge, this question has been left mostly unexplored in the literature. We explore this problem along three main directions:

- **In-class MSDA samples** We first consider in [Chapter 2](#) how in-class mixed samples could be obtained and leveraged to train stronger classifiers in difficult training scenarios. While some works in the literature have attempted to mix samples from the same class such that the mixed samples remain within a single class, we study a new paradigm that transposes the semantic content of one sample into the non-semantic context of another sample. To this end, we introduce generative frameworks that allow us to generate such mixed samples and then verify that these augmented samples can be leveraged to improve the performance of models in semi-supervised settings. The work here has led to two publications:
 - SUN Rémy, MASSON Clément, HENAFF Gilles, THOME Nicolas. and CORD Matthieu (2021) “Semantic Augmentation by Mixing Contents for Semi-Supervised Learning”. Submitted as a research article to Pattern Recognition; presented as a short paper at NeurIPS 2020 workshop on Self-Supervised Learning.
 - SUN Rémy, MASSON Clément, HENAFF Gilles, THOME Nicolas and CORD Matthieu. (2022) “Swapping Semantic Contents for Mixing Images”, in International Conference on Pattern Recognition (Oral).
- **MSDA samples as compressed representations for MIMO training** Taking a step back from the issue of what class the mixed sample should be, we observe that mixed samples contain information from two distinct samples in [Chapter 3](#). As such, the notion of between-class samples and soft labels simply hides a more complex optimization problem. We leverage this vision of MSDA to generalize the recent Multi-Input Multi-Output (Havasi et al. 2021) pseudo-ensembling framework. We then introduce an unmixing mechanism to help feature sharing between the subnetworks induced by MIMO MSDA. The contributions in this chapter have concluded in two publications:

- RAME Alexandre* (equal contribution), SUN Rémy* (equal contribution), and CORD Matthieu. (2022) “MixMo: Mixing Multiple Inputs for Multiple Outputs via Deep Subnetworks”, in International Conference on Computer Vision.
- SUN Rémy, RAME Alexandre, MASSON Clément, THOME Nicolas and CORD Matthieu. (2022) “Towards efficient feature sharing in MIMO architectures”, in CVPR 2022 workshop on efficient deep learning for computer vision.
- **MIMO MSDA, unmixing and attention** [Chapter 4](#) finally looks into the link between our previously introduced unmixing and the attention mechanism at the heart of Vision Transformers. As this link causes MIMO MSDA to behave intrinsically differently than on CNNs, we modify the architecture to accommodate it. Interestingly, the feature sharing between subnetworks this entails causes this new MIMO MSDA to be closer to standard regularizing MSDA. We have summarized our finding on the issue in a publication:
 - SUN Rémy, MASSON Clément, THOME Nicolas and CORD Matthieu. (2022) “Adapting Multi-Input Multi-Output schemes to Vision Transformers”, submitted to ICLR 2023, in CVPR 2022 workshop on transformers and attention.

1.4 Summary table of publications

Table 1.1 gives a summary of the publications developed during this thesis.

Publication	Chapter reference
SUN Rémy, MASSON Clément, HENAFF Gilles, THOME Nicolas. and CORD Matthieu (2021) "Semantic Augmentation by Mixing Contents for Semi-Supervised Learning". Submitted as a research article to Pattern Recognition; presented as a short paper at NeurIPS 2020 workshop on Self-Supervised Learning.	2
SUN Rémy, MASSON Clément, HENAFF Gilles, THOME Nicolas and CORD Matthieu. (2022) "Swapping Semantic Contents for Mixing Images", in International Conference on Pattern Recognition (Oral).	2
RAME Alexandre* (equal contribution), SUN Rémy* (equal contribution), and CORD Matthieu. (2022) "MixMo: Mixing Multiple Inputs for Multiple Outputs via Deep Subnetworks", in International Conference on Computer Vision.	3
SUN Rémy, RAME Alexandre, MASSON Clément, THOME Nicolas and CORD Matthieu. (2022) "Towards efficient feature sharing in MIMO architectures", in CVPR 2022 workshop on efficient deep learning for computer vision.	3
SUN Rémy, MASSON Clément, THOME Nicolas and CORD Matthieu. (2022) "Adapting Multi-Input Multi-Output schemes to Vision Transformers", submitted to ICLR 2023, in CVPR 2022 workshop on transformers and attention.	4

Table 1.1. – Table of publications and thesis chapters.

IN-CLASS MIXING SAMPLES DATA AUGMENTATIONS

Contents

2.1	Introduction	13
2.2	Related Work	16
2.2.1	Semi-Supervised Learning	16
2.2.2	Hybrid generation in the literature	19
2.2.3	Positioning	21
2.3	SAMOSAs: Adding non-semantic variations to an image	22
2.3.1	Overview of the SAMOSAs framework	23
2.3.2	Adding a Non-Supervised Reconstruction Module	25
2.3.3	Learning Scheme	27
2.3.4	Making use of the SAMOSAs framework in Semi-Supervised Learning	29
2.3.5	Take-aways from the SAMOSAs project	29
2.4	SciMix: Embedding semantic content into other contexts	31
2.4.1	Learning to generate hybrids	32
2.4.2	Training a classifier by leveraging our Data Augmentation	35
2.5	Results	36
2.5.1	Performance improvements	37
2.5.2	Improvements of SciMix over SAMOSAs	39
2.5.3	Further analysis of SciMix results	43
2.5.4	Pushing SciMix on CUB-200	45
2.6	Conclusion	46

2.1 Introduction

Mixing samples data augmentations combine contents from multiple images to improve the performance of neural networks. The artificial hybrids created by traditional MSDA techniques are however clearly not natural images, which can also be observed from the soft targets used to train models on those samples.

MSDA techniques traditionally mix contents from source images indiscriminately, and therefore incorporate semantic content from both. This is the core reason the generated images are out of the target distribution and use non-binary labels: the hybrids literally contain some semantic content from the first image’s class and some from the other’s. There are benefits to this, and MixUp-based (H. Zhang et al. 2018) mixing techniques at the least have been proven to act as noisy regularizers (Carratino et al. 2020; Thulasidasan et al. 2019). Nevertheless, the fact remains that these MSDA do not truly extend the support of the training distribution as is the purview of standard data augmentation.

In this chapter, we explore the idea of generating mixed samples that inherit semantic content from only one of their parents, and take other non-semantic characteristics from their other parent. As the ultimate goal here is to improve networks for a given classification task, we define “semantic” with regard to the information used by a classifier trained on the problem. This has the distinct benefit of directly teaching models a measure of generalization: by showing a similar semantic content with different non-semantic attributes, we help the model become invariant to the non-semantic content.

This type of augmentation is particularly useful in settings like semi-supervised learning where we have access to a lot of training samples but few labels. Indeed, the role of unlabeled images in semi-supervised learning is to regularize solutions found by the classifier on the few labeled images available. As such, our mixing paradigm very naturally fits into semi-supervised design philosophies and helps regularize the model by teaching it to ignore some non-semantic characteristics.

To this end, we propose novel neural networks that take two images as inputs and yield an artificial sample that integrates semantic content from one of the sample with general non-semantic characteristics from the other sample from the available (labeled and unlabeled) data as illustrated in [Figure 2.1](#). We can then use the trained generator to generate more artificial samples to train a semi-supervised classifier.

Through the course of this thesis, we have proposed two different strategies to identify and combine contents from multiple samples to improve the performance of existing semi-supervised algorithms:

- **SAMOSA** In a first approach, we take the semantic parent sample as the basis and add some non-semantic characteristics from the non-semantic parent. We develop an autoencoder architecture that separates semantic content from non-semantic content in an image before recombining them. The autoencoder is trained to identify semantic and non-semantic contents with no preconceptions other than sharing some features with a classifier, and an asymmetric generator structure that uses one input to decide *what* to

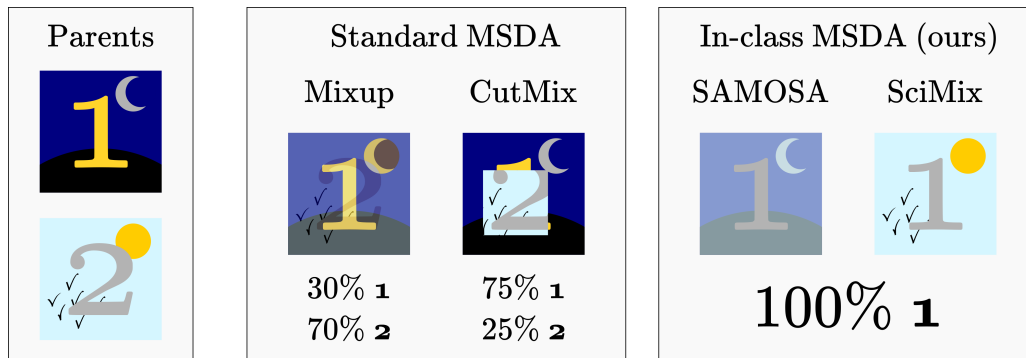


Figure 2.1. – **Illustration of typical hybrids obtained by standard mixing methods and our methods.** Standard mixing augmentations mix contents indiscriminately whereas our method mixes the semantic number from one sample with the background information from another.

reconstruct and the other *how* to reconstruct. In this approach, we train the generator alongside the classifier we seek to improve, thereby regularizing the classifier.

- **SciMix** In a second approach, we propose a much stronger hybridizer that embeds semantic content from one sample into the general (non-semantic) context of the other sample. Here, we add explicit loss terms to guide hybridization but still refrain from explicitly defining non-semantic content. Contrarily to SAMOSA, we separate the generator and classifier by first training the generator, then using a fully trained generator to generate hybrids on the fly as we train a semi-supervised classifier from scratch.

We first provide some context on Semi-Supervised Learning in [Section 2.2](#). We then discuss our proposed SAMOSA ([Section 2.3](#)) and SciMix ([Section 2.4](#)) frameworks. Finally, we experimentally validate our methods in [Section 2.5](#).

The work conducted in this chapter of this thesis has led to two publications:

- SUN Rémy, MASSON Clément, HENAFF Gilles, THOME Nicolas. and CORD Matthieu (2021) “Semantic Augmentation by Mixing Contents for Semi-Supervised Learning”. Submitted as a research article to Pattern Recognition; presented as a short paper at NeurIPS 2020 workshop on Self-Supervised Learning.
- SUN Rémy, MASSON Clément, HENAFF Gilles, THOME Nicolas and CORD Matthieu. (2022) “Swapping Semantic Contents for Mixing Images”, in International Conference on Pattern Recognition (Oral).

2.2 Related Work

This chapter tackles semi-supervised learning problems and develops generative models to generate our in-class hybrids. As such, the following provides some context on those topics.

2.2.1 Semi-Supervised Learning

We first introduce the semi-supervised (Chapelle et al. 2006) problem before providing a quick overview of the relevant literature. In semi-supervised learning, the dataset $\mathcal{D} = \mathcal{D}_l \cup \mathcal{D}_u$ contains two sub-datasets: a labeled \mathcal{D}_l dataset and an unlabeled \mathcal{D}_u dataset. At the core of semi-supervised learning therefore lies the question of how to find ways to leverage \mathcal{D}_u to extract information relevant to the task of interest.

This has been achieved in a number of ways, ranging from generative modeling (C. Li et al. 2017) to graph based methods (Isken et al. 2019; Chapelle et al. 2006). While generative models have mostly relied on unsupervised generative training (C. Li et al. 2017), label propagation methods have been used in a number of graph based methods to infer labels for unlabeled data (Isken et al. 2019).

Recent advances in self-supervised learning are often hijacked for Semi Supervised Learning by jointly performing supervised training with labeled samples and self-supervised learning on unlabeled samples (Zhai et al. 2019; B. Kim et al. 2021). While these methods currently hold the state-of-the-art on larger datasets like ImageNet, the self-supervised objectives involved require a lot of unlabeled data to train on. As such, semi-supervised learning on smaller datasets remains the province of other more specialized semi-supervised frameworks.

Specialized semi-supervised frameworks have focused in recent years on two techniques: consistency training (Tarvainen and Valpola 2017; Miyato et al. 2019; Laine and Aila 2017; Luo et al. 2018) and pseudo-labeling (Isken et al. 2019; Sohn et al. 2020; Rizve et al. 2021). We illustrate these two families of techniques through Mean Teacher and FixMatch, two well established semi-supervised methods.

2.2.1.1 Mean Teacher and consistency based methods

The Mean Teacher (Tarvainen and Valpola 2017) framework makes use of two key components: a student network f_θ (the one we want to train) and a teacher network \hat{f}_θ . The Mean Teacher framework's name stems from the fact this teacher network is in fact an exponential moving average of the student network such that $\hat{f}_\theta = EMA(f_\theta) = f_{EMA(\theta)}$.

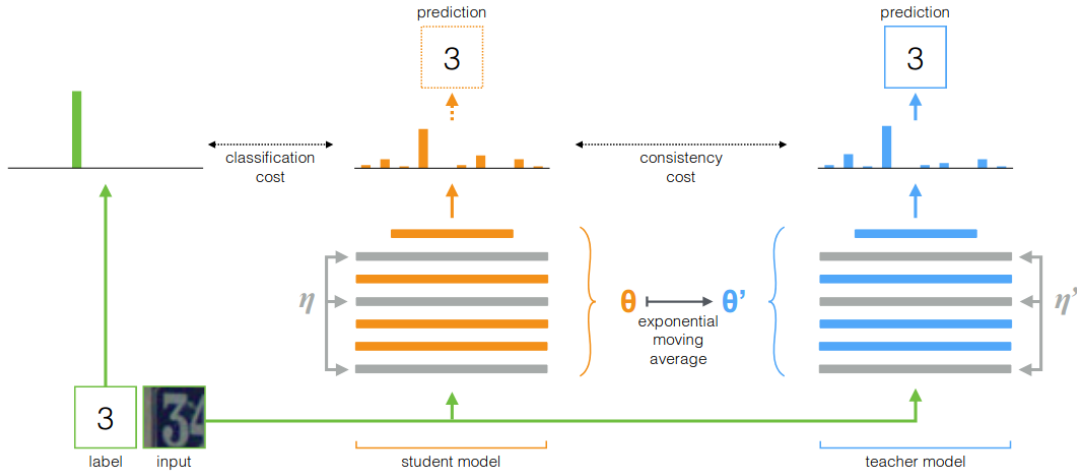


Figure 2.2. – **Overview of the Mean Teacher framework.** Figure from the seminal paper (Tarvainen and Valpola 2017).

Given a teacher and student network, the Mean Teacher framework is a standard consistency based method that optimizes a consistency loss over unlabeled samples

$$\mathcal{L}_{cons}(\mathcal{D}_u) = \sum_{x \in \mathcal{D}_u} \|\hat{f}_\theta(x) - f_\theta(x)\|^2, \quad (2.1)$$

with the network optimizing the overall training loss

$$\mathcal{L}_{MeanTeacher} = \mathcal{L}_{Sup}(\mathcal{D}_l) + \lambda \mathcal{L}_{cons}(\mathcal{D}_u), \quad (2.2)$$

where λ is a trade-off term.

Mean Teacher (Tarvainen and Valpola 2017) constitutes a first significant milestone for deep semi-supervised learning, cementing consistency based techniques as a staple of the field after the early successes of Laine and Aila 2017. A large number of consistency based methods have been developed following its success (Robert et al. 2018; Verma et al. 2019b), and the Mean Teacher framework remained a standard baseline for a number of years. More disruptive mixing data augmentation techniques have proven effective for consistency based training (Berthelot et al. 2019; Berthelot et al. 2020; Verma et al. 2019b) for a time. To this day, a number of more preliminary exploration into semi-supervised learning for other domains still largely adapt the Mean Teacher framework (e.g. Cross-Domain Object Detection (Jinhong Deng et al. 2021), Medical Image Segmentation (K. Wang et al. 2022), ...).

Mean Teacher and its affiliated consistency based methods have recently lost in prominence with the advent of deep pseudo-labeled frameworks like FixMatch.

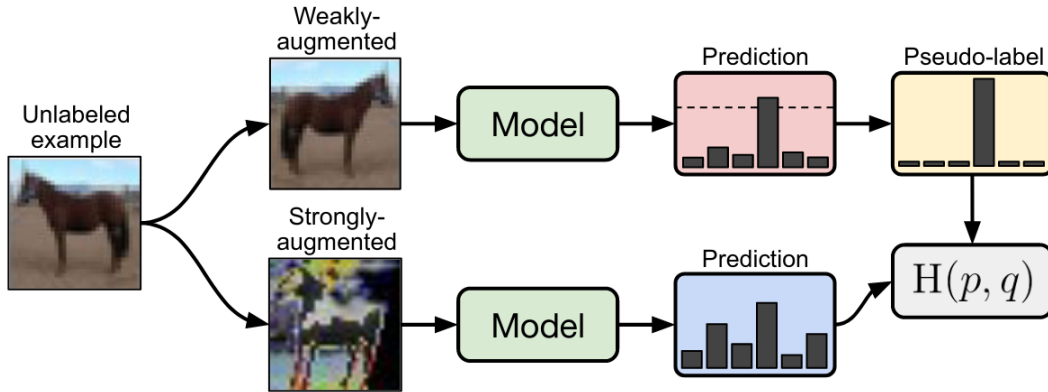


Figure 2.3. – **Overview of the FixMatch framework.** Figure from the seminal paper (Sohn et al. 2020).

2.2.1.2 FixMatch and pseudo-labeling based methods

The FixMatch (Sohn et al. 2020) framework only uses one network - the one we wish to train - and instead focuses on the data augmentations applied to inputs. For one input x , FixMatch considers a weakly augmented version x_{weak} (i.e. kept close to the original input x) and a strongly augmented version x_{strong} (i.e. augmented with transformations of significant magnitude).

The core idea of FixMatch is to compute pseudo-labels from the weakly augmented versions of the inputs and use those pseudo-labels to train the model on the strongly augmented inputs. Formally, this means computing probabilities $p_{weak,x} = f_{\theta}(x)$, converting them to pseudo-labels $\hat{p}_x = \mathbf{1}(\max p_{weak,x} \geq \tau) \arg \max(p_{weak,x})$.

As far as the optimization goes, the framework minimizes a simple pseudo-labeling loss on the unlabeled data

$$\mathcal{L}_{pseudo}(\mathcal{D}_u) = \sum_{x \in \mathcal{D}_u} l_{CE}(f(x_{strong}), \hat{p}_x), \quad (2.3)$$

with the network optimizing the overall training loss

$$\mathcal{L}_{FixMatch} = \mathcal{L}_{Sup}(\mathcal{D}_l) + \lambda \mathcal{L}_{pseudo}(\mathcal{D}_u), \quad (2.4)$$

where λ is a trade-off term.

FixMatch (Sohn et al. 2020) constitutes the current gold standard for semi-supervised learning and semi-supervised pseudo labeling methods (Chapelle et al. 2006; Iscen et al. 2019) in particular. While refinements to the FixMatch method have since been proposed (J. Li et al. 2021; B. Zhang et al. 2021), these methods most often rely heavily on the backbone set by FixMatch. Interestingly, FixMatch

has started to replace Mean Teacher as the semi-supervised framework of choice in more applied semi-supervised works (e.g. Object Detection (Y.-C. Liu et al. 2022), ...)

2.2.2 Hybrid generation in the literature

A number of different techniques might allow the generation of edited samples, though few match our use case. For instance, while there is a large body of work on disentangled generation (Higgins et al. 2017; Robert et al. 2019), it does typically involve identifying the disentangled factors in an existing image. We are first and foremost interested in generative processes that allow us to combine contents from images. Here, we will discuss a few methods that do allow such combinations like Style Transfer based methods and Image-to-Image translation works.

2.2.2.1 AdaIN and style transfer

Style transfer seeks to modify an existing image to incorporate some “stylistic” characteristics from another image (e.g. see an image in the style of a painting (Gatys et al. 2016)). The core idea is to make use of a specific “perceptual” loss that measures the distance between images at different representation levels in a neural classifier: we can ask the image to be close to one image for some representation spaces and close to another in the remaining intermediate representations (Gatys et al. 2016). Although the early techniques required computationally intensive gradient ascent procedures, this particular type of transformation has become more widespread with the advent of lightweight techniques like AdaIN (X. Huang and Belongie 2017).

AdaIN trains a network to encode images into a latent space, manipulate the latent space and decode the latent representation into a new image using a perceptual loss that mirrors the latent modification. The heart of the techniques lies in the peculiar manipulation of the latent space: when trying to generate an image c in the style of an image s , we normalize “ c ”’s latent z_c and replace the per-map moments (mean μ and standard deviation σ) with those of the stylistic latent s to obtain a hybrid latent z_h :

$$z_h = \text{AdaIN}(z_c, z_s) = \frac{z_c - \mu(z_c)}{\sigma(z_c)} \odot \sigma(z_s) + \mu(z_s). \quad (2.5)$$

At training, we go on to train the model to reconstruct an image close to c in content and s in style according to the perceptual loss. This way, AdaIN can

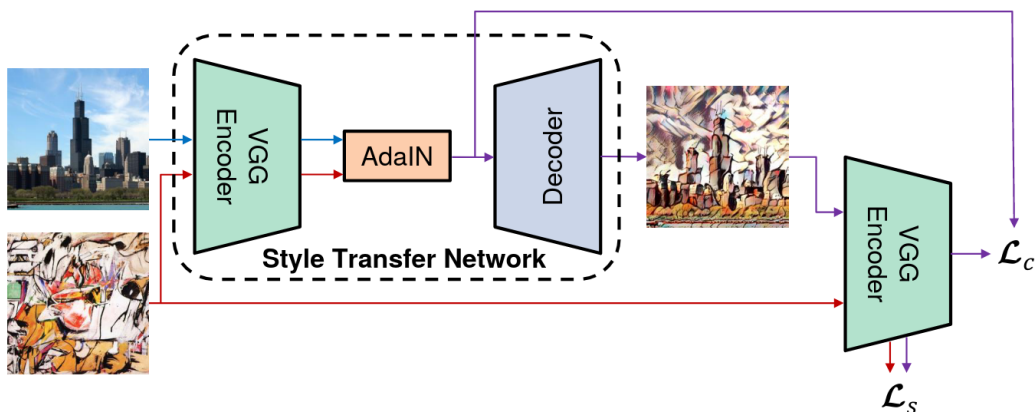


Figure 2.4. – **Overview of the AdaIN framework.** Figure from the seminal paper (X. Huang and Belongie 2017).

generate hybrids at a low cost at test time by simply swapping the moments of the latent representation and retrieving the decoded image.

It is worth noting that AdaIN has had an enduring legacy beyond style transfer as the map modulation mechanism it introduces is central to the StyleGAN (Karras et al. 2019; Karras et al. 2020) family of generative networks. StyleGAN networks learn to generate very complex images solely by modulating feature maps at every layer of a Convolutional Neural Network that takes a constant vector as input. While StyleGAN networks do not combine multiple samples, they have been used in recent advances in another hybridizing technique: Image-to-Image translation.

2.2.2.2 (Unsupervised) image-to-image translation

Image-to-Image translation (Zhu et al. 2017) typically aims to transpose an image from one domain into another domain (e.g. from day to night). Although traditional approaches rely on knowing the domains of interest, unsupervised Image-to-Image translation approaches simply attempt to translate one image into the domain of another without explicit domain knowledge.

Interestingly, work on this matter regularly draws inspiration from advances in style transfer. Bi-modal auto-encoding architectures appear fairly early on in this field (M.-Y. Liu et al. 2017; X. Huang et al. 2018). More recent works in few-shot translation (M.-Y. Liu et al. 2019) and unsupervised translation (Baek et al. 2021) have even started associating the domain (or class) information to a style code fed as input to a StyleGAN inspired decoder. On a more supervised note, style based methods have been used to combine textures (domain information)

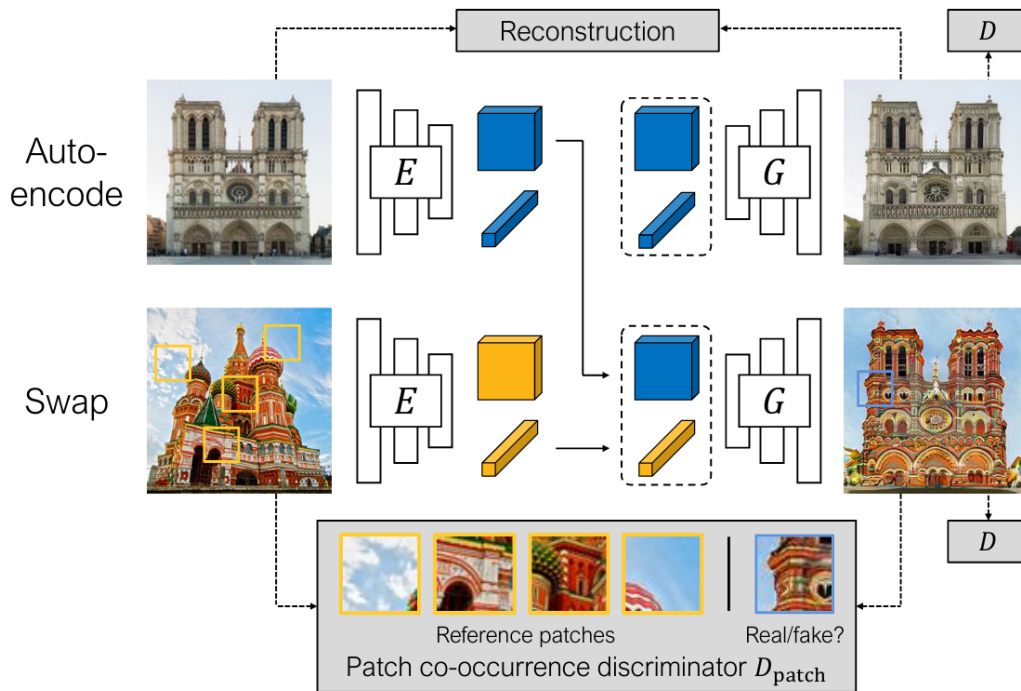


Figure 2.5. – **Example of a style based Image-to-Image translation framework.** Figure from the Swapping Auto-encoders paper (Park et al. 2020).

with structural information (image information) (Park et al. 2020) as shown on Figure 2.5 thanks to StyleGAN’s peculiar structure.

2.2.3 Positioning

In this chapter, we propose a new type of mixing data augmentation that mixes the semantic content of one sample with other non-semantic content for low label settings like semi-supervised learning. We posit that such data augmentations can help models learn invariance to some non-semantic variations, and that this type of invariance is harder to learn in a semi-supervised setting.

As such, we aim here to generate hybrids that satisfy our requirements and demonstrate their usefulness in a semi-supervised setting. Our goal is not to replace any established semi-supervised algorithm, and the augmentations we propose are deployed alongside standard semi-supervised algorithm. In fact, we mainly evaluate the success of our contributions by the gains obtained by combining our augmentations with a backbone method like Mean Teacher (Tarvainen and Valpola 2017) or FixMatch (Sohn et al. 2020).

To generate the hybrids, we use an asymmetric decoder/generator that gives different roles to its two inputs in the generative process. In SAMOSA, we design an architecture that follows similar principles as AdaIN (X. Huang and Belongie 2017) and StyleGAN (Karras et al. 2020) with one of the inputs influencing how the model generates from the other input. While the mechanism is similar, it comports some differences in how the internal maps are chosen as it was developed independently. SciMix on the other hand directly makes use of StyleGANv2 (Karras et al. 2020) generator and adapts ideas from Swapping Auto-encoders (Park et al. 2020).

Crucially, the generators we train are specifically designed to help train a classifier whereas the discussed methods focus entirely on generating new images. Contrarily to those methods, the hybridization is mostly guided by semantic information from a classifier (as opposed to specialized perceptual loss terms).

2.3 SAMOSA: Adding non-semantic variations to an image

Our final goal here is to generate in-class hybrids that mix contents from two samples that inherit semantic content from one parent and non-semantic content from another as shown in Figure 2.6. The underlying idea behind these mixed samples is that such samples can decline the semantic content of the available samples in different non-semantic context given by the large unlabeled set we have in problems like semi-supervised learning.

Our SAMOSA framework can mix the contents of two inputs such that the mixed sample retains most of one sample’s semantic content, but integrates some non-semantic characteristic from the other. To this end, SAMOSA learns to separate and identify semantic content from non-semantic content in images, and how to re-combine the extracted representations to create plausible hybrids (e.g. Figure 2.6b).

The main challenge when performing such mixing lies in this proper separation of semantic and non-semantic content. SAMOSA introduces a novel neural architecture that separates input information into semantic information useful to a classifier, and auxiliary information necessary for reconstruction. Furthermore, our SAMOSA framework leverages its novel asymmetrical decoder (inspired by work in generative modeling and edition (X. Huang and Belongie 2017; Karras et al. 2019)) to mix any two extracted semantic and non-semantic content. Figure 2.6b shows how SAMOSA combines a bird picture with color tones from a plane picture.

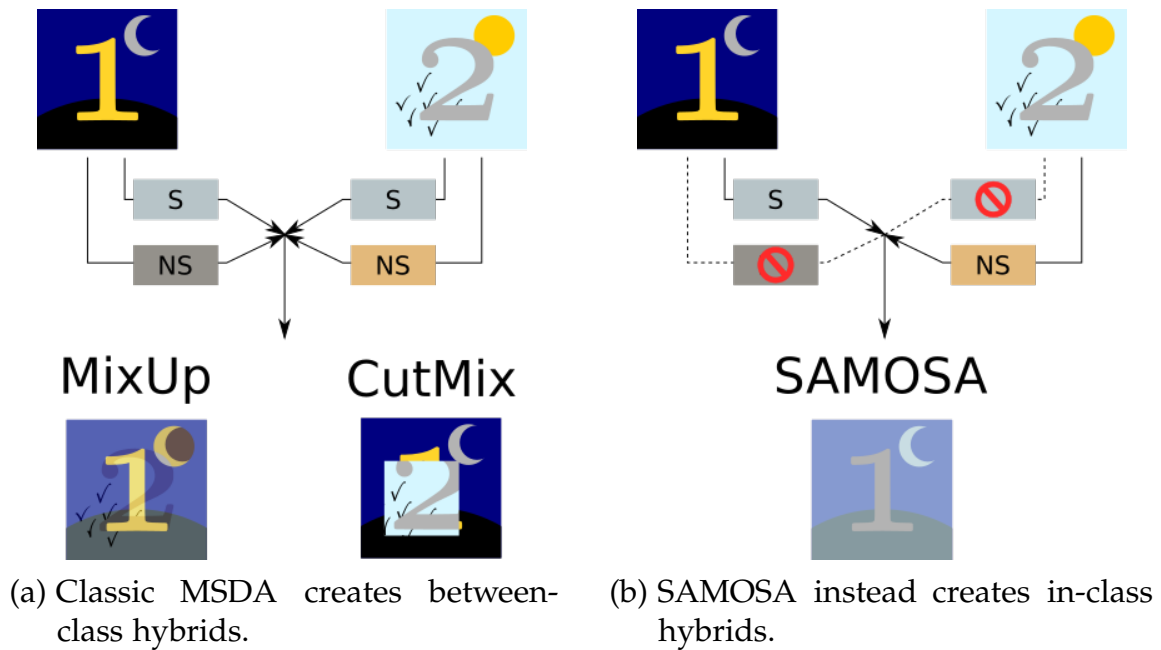


Figure 2.6. – **Content combination in classic MSDA and ours.** While classical mixing combines all content (i.e. semantic “S” + non-semantic “NS”) from both parents, SAMOSA mixes semantic content (“S”) from one parent and non semantic content (“NS”) from the other.

The main contributions we seek with SAMOSA are therefore twofold:

- Create a generative framework that allows the creation of hybrids such that the mixed sample inherits the semantic content of one parent sample and non-semantic characteristics of the other.
- Offer a first proof of concept regarding the usefulness of this new mixing paradigm for semi-supervised learning by combining it with well known semi-supervised algorithms.

2.3.1 Overview of the SAMOSA framework

We detail in this section our proposed SAMOSA Framework. After a brief overview of the general framework, we give a detailed account of our novel asymmetrical decoder in Sec. 2.3.2 and detail SAMOSA’s atypical learning scheme in Sec. 2.3.3. Finally, we discuss how our SAMOSA framework can be used in a SSL setting in Sec. 2.3.4.

First and foremost, we introduce in this paper a novel architecture presented in Figure 2.7. It is composed of two encoders E_c and E_r (one semantic - with

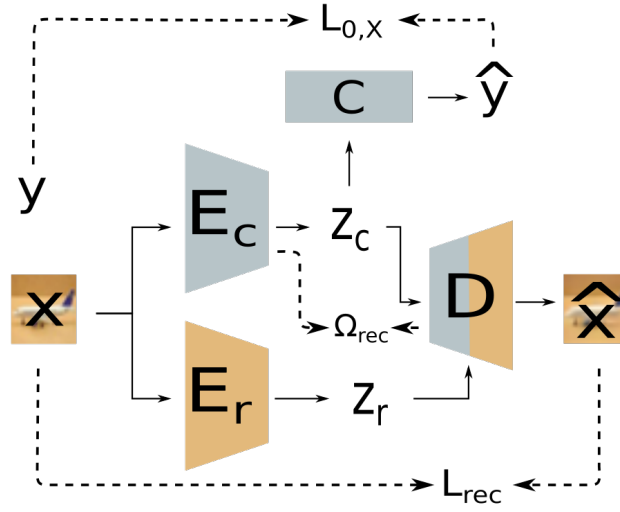


Figure 2.7. – **Overview of the SAMOSA framework.** z_c and z_r are extracted from input x . z_c alone is used to classify the input (to optimize the base $\mathcal{L}_{0,X}$ loss). \hat{x} (which reconstructs x due to Ω_{SAMOSA} 's sub-loss \mathcal{L}_{rec}) is computed from both extracted features z_c and z_r . E_c is regularized by the learned decoder (through the Ω_{rec} regularizer of Ω_{SAMOSA})

regards to the classification process - and one non semantic), a simple classifier C and a bi-modal decoder D that takes inputs from a semantic modality z_c and a non-semantic modality z_r . SAMOSA is meant to be added on top of existing semi-supervised learning algorithms for neural architectures. In this sense, an input x is mapped to a feature representation $z_c = E_c(x)$, which is then used to obtain a classifier prediction $\hat{y} = C(z_c) = C(E_c(x))$. We further elaborate on the peculiarities of our additional reconstruction modules E_r and D in Section 2.3.2.

To train such an architecture, we optimize the modules necessary for classification (E_c and C) to minimize a two component loss \mathcal{L}_{SAMOSA} (Equation 2.6, Figure 2.7). Our novel regularizer Ω_{SAMOSA} (in Equation 2.6) differs substantially from standard reconstruction regularizers by leveraging peculiarities of SAMOSA's architecture. On the other hand, the base loss $\mathcal{L}_{0,X}$ term in Equation 2.6 acts as a proxy to represent the base method (which we seek to improve) "X"'s training process (see Figure 2.7). For instance, the three base losses we considered are $\mathcal{L}_{0,MT}$ from Mean Teacher (Tarvainen and Valpola 2017), $\mathcal{L}_{0,Mix}$ from MixMatch (Berthelot et al. 2019) and $\mathcal{L}_{0,Fix}$ from FixMatch (Sohn et al. 2020). As the basic algorithms we consider are meant to function on standard classifier models, $\mathcal{L}_{0,X}$ is only minimized for $E_c \circ C$. The training and manipulation of the remaining modules as well as the regularizer term Ω_{SAMOSA} are specific to SAMOSA, and are elaborated upon in Section 2.3.3.

$$\begin{aligned} \mathcal{L}_{SAMOSA}(\{x_l, y_l\}_{\mathcal{D}_l} \cup \{x_u\}_{\mathcal{D}_u}) = & \mathcal{L}_{0,X}(\{x_l, y_l\} \cup \{x_u\}) \\ & + \Omega_{SAMOSA}(\{x_l\} \cup \{x_u\}). \end{aligned} \quad (2.6)$$

As such, the method trains a standard classifier $E_c \circ C$ according to a base SSL method X with loss $\mathcal{L}_{0,X}$. Our contribution consists in adding a reconstruction regularizer Ω_{SAMOSA} , a non-semantic encoder E_r and a special bi-modal decoder D to be optimized and trained simultaneously with the base SSL classifier. The bi-modal decoder in particular requires careful design to mix semantic and non-semantic content.

2.3.2 Adding a Non-Supervised Reconstruction Module

Our goal is to mix the semantic content of one sample with the non-semantic content of another. This requires both separating the two contents from samples and reconstructing from those contents in a modular fashion. To some extent, this has been achieved in style transfer (X. Huang and Belongie 2017) and generative modeling (Higgins et al. 2017; Karras et al. 2019). (X. Huang and Belongie 2017) and (Karras et al. 2019) for instance have shown manipulating activation statistics of intermediate activation maps in an autoencoder can be used to train a model capable of reconstructing an input image in a number of different ways. Those methods however either explicitly define what “style” (which we liken to non semantic information) is through a specifically designed loss functions and targets (X. Huang and Belongie 2017), or perform adversarial optimization that does not allow for specific reconstructions (Karras et al. 2019). We propose here an architecture operating along similar principles, but that can reconstruct inputs without any pre-conception of what constitutes non-semantic information.

We retain the base model’s E_c as our semantic encoder, and add a separate encoder E_r for the remaining non-semantic information. A novel asymmetrical bi-modal decoder D is then used to reconstruct the input images from the outputs of the encoders E_c and E_r . Practically, an input x is mapped to an additional non semantic feature representation $z_r = E_r(x)$, which is then used in conjunction with z_c to obtain a reconstructed image $\hat{x} = D(z_c, z_r) = D(E_c(x), E_r(x))$. This last reconstruction process is facilitated by the very peculiar structure of D .

Asymmetrical decoder D Crucially, we design a novel decoder module (Figure 2.8) to combine semantic and non semantic feature spaces. An immediate concern when reconstructing from two latent spaces as we propose is that an unconstrained non semantic feature space is liable to store all the necessary in-

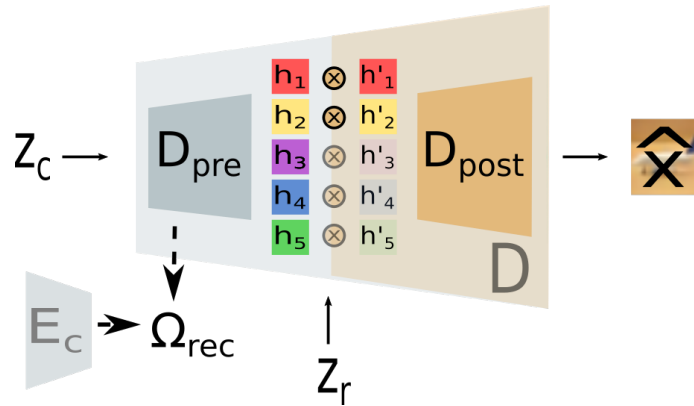


Figure 2.8. – **Overview of the asymmetrical SAMOSA decoder.** Our proposed asymmetrical decoder D reconstructs x from z_c , with z_r modulating which parts of D are active.

formation to reconstruct the input, thereby leaving a decoder free to ignore the semantic feature space z_c . Previous work (Robert et al. 2018) ran into this issue when generating two partial reconstructions - one from semantic features and one from non-semantic features - and summing the two to obtain a complete reconstruction. This was addressed by forcefully stopping gradient flows of one partial reconstruction right before combination (depending on which partial reconstruction needs more training). However, this method led to both E_c and E_r each contributing very similar information as this process only ensures the two modalities contribute to the reconstruction. Conversely, we design an asymmetrical decoder that uses the two input modalities differently.

To prevent z_r from encoding all the information, we shift its role from affecting *what* is on the reconstruction to affecting *how* the semantic latent space z_c is translated to a reconstruction. As figured in Figure 2.8, D can be broken down into two sub decoders D_{pre} and D_{post} such that $h = D_{pre}(z_c) \in \mathbb{R}^{S \times H \times W}$ can be construed as a stack of S intermediate reconstruction maps. z_r serves as a set of S gating weights $\in [0, 1]$ (through the use of a final linear projection and softmax activation) such that only some intermediate activation maps $h' = z_r \odot h$ are used to compute the final reconstruction $D_{post}(h')$ (e.g. only the red and yellow maps remain in Figure 2.8). While this can be seen as a rescaling of feature maps (like in style transfer) (X. Huang and Belongie 2017; Karras et al. 2019; X. Huang et al. 2018), the absence of style targets might lead to z_r selecting all maps with a method such as AdaIN. To address this, we ensure only a few maps are selected by z_r for each sample using a softmax activation.

This architecture allows us to reconstruct samples while avoiding the pitfall of forcing the classifier’s feature extractor E_c to keep irrelevant information. Furthermore, we propose a learning scheme that pushes the semantic encoder E_c

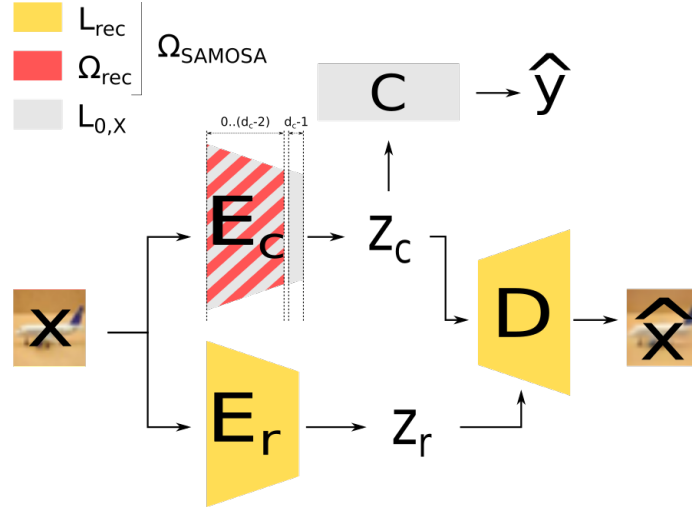


Figure 2.9. – **Optimization of \mathcal{L}_{SAMOSA} in the modules.** The base SSL loss $\mathcal{L}_{0,X}$ loss optimizes the classifier modules while the reconstruction loss \mathcal{L}_{rec} optimizes the additional modules E_r and D . E_c is benefited from the reconstruction module through the Ω_{rec} regularizer

to leverage the decoder D to identify what it should keep track of through the second term Ω_{SAMOSA} of the loss given in Equation 2.6.

2.3.3 Learning Scheme

Model optimization SAMOSA relies on a regularizer term Ω_{SAMOSA} to leverage its peculiar architecture:

$$\begin{aligned} \Omega_{SAMOSA}(\{x_l\}_{\mathcal{D}_l} \cup \{x_u\}_{\mathcal{D}_u}) = & \lambda_{rec} \mathcal{L}_{rec}(\{x_l\} \cup \{x_u\}) \\ & + \lambda_{SAMOSA} \Omega_{rec}(\{x_l\} \cup \{x_u\}), \end{aligned} \quad (2.7)$$

with the loss \mathcal{L}_{rec} used to optimize E_r and D for reconstruction of inputs, and the auxiliary regularizer Ω_{rec} used to refine E_c through knowledge learned by D . This differs significantly from traditional work in SSL that uses reconstruction for regularization as we do not directly optimize the classifier for reconstruction. Rather, we leverage our asymmetrical decoder’s peculiar structure to regularize the classifier so that it solely learns to reconstruct information identified as semantic by our framework.

$\mathcal{L}_{rec} = \frac{1}{\#\mathcal{D}} \sum_{x \in \mathcal{D}} \|D(E_c(x), E_r(x)) - x\|_2^2$ (figured on Figure 2.9) tries to match inputs x to model reconstructions $D(E_c(x), E_r(x))$ through the L2 distance between the two. E_c is deliberately not optimized here as skip connections (He et al. 2016b)

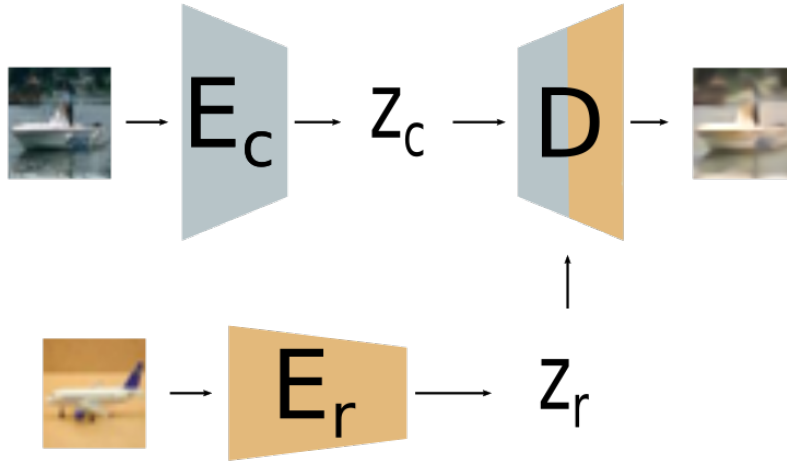


Figure 2.10. – **Hybridization in SAMOSA.** A trained model can then be used to combine semantic content from a boat picture and non semantic content from a plane picture.

in modern neural networks already let a lot of input information trickle down to their feature space. In our experiments, we found optimizing E_c for reconstruction led D to rely entirely on E_c and ignore E_r .

$\Omega_{rec} = \frac{1}{\#\mathcal{D}} \sum_{x \in \mathcal{D}} \|E_c^{(0..(d_c-2))}(x) - D^{(0)}(E_c(x), E_r(x))\|_2^2$ leverages our decoder’s asymmetrical structure to regularize E_c (Figure 2.9). Importantly, the first few intermediate reconstructions are purely semantic as they are prior to re-modulation by z_r (the style input $E_r(x)$ in $D(E_c(x), E_r(x))$ is of no effect). Therefore, training E_c to match these early decoder features provides a novel reconstruction regularizer for the feature extractor that is not polluted by non-semantic information (i.e. information injected by $E_r(x)$ in the reconstruction). In practice, Ω_{rec} ties the last intermediate features $E_c^{(0..(d_c-2))}(x)$ extracted by E_c (layer $E_c^{(d_c-2)}$) to the first intermediate reconstructions $D^{(0)}(E_c(x), E_r(x))$ generated by D (layer $D_{(0)}$). Here, d_c refers to the depth of E_c , $E_c^{(0..(d_c-2))}$ to the composition of the first $d_c - 1$ convolutional layers of E_c (all but the last one), and $D^{(0)}$ to the first convolutional layer of D .

This training process yields an architecture capable of generating hybrids that incorporates non-semantic content from a sample x_2 into a sample x_1 while preserving x_1 ’s semantic content. We now discuss how this can be put to use in a Semi-Supervised Learning setting.

2.3.4 Making use of the SAMOSA framework in Semi-Supervised Learning

We introduce a novel asymmetrical decoder that is modular *by design* with regard to semantic and non semantic content, as well as propose an adapted training scheme. In practice, the learning scheme itself can be used to regularize classifiers, but the trained models can also be used to generate augmented samples to train models on. For instance, a model could be trained to optimize \mathcal{L}_{SAMOSA} , then used to generate a set of artificial labeled samples through SAMOSA hybridization and the model could then be re-trained on the augmented dataset.

Indeed, generating hybrids given a trained model is straightforward (Algorithm A.3 and Figure 2.10). Specifically, given samples $x^{(1)}$ (with known label $y^{(1)}$) and $x^{(2)}$, we extract the relevant features $z_c^{(1)} = E_c(x^{(1)})$, $z_r^{(1)} = E_r(x^{(1)})$, $z_c^{(2)} = E_c(x^{(2)})$ and $z_r^{(2)} = E_r(x^{(2)})$. $x_h = D(z_c^{(1)}, z_r^{(2)})$ is now a sample with class $y^{(1)}$. As a conservative measure, we only keep the generated hybrid if $C(E_c(x_h)) = y^{(1)}$ to avoid disturbing decision boundaries too much. Note that with this, we generate a strong augmentation of x_1 and teach the classifier to group x_1 with its strongly augmented version in a similar line to work in contrastive representation learning (He et al. 2020).

Integrating SAMOSA into classical semi-supervised learning frameworks As previously discussed, SAMOSA can be deployed in SSL systems in a variety of ways, two of which are explored experimentally paper. We study a first framework that trains a SSL model to optimize \mathcal{L}_{SAMOSA} , generates hybrids using labeled samples for the semantic component and unlabeled samples for the non-semantic component, and re-trains the model on the augmented set (Appendix Algorithm A.1). We also show a more intricate incorporation of SAMOSA in the MixMatch framework (Appendix Algorithm A.2) by occasionally replacing the MixUp procedure with our in-class hybridization in the training of a MixMatch model optimizing \mathcal{L}_{SAMOSA} .

2.3.5 Take-aways from the SAMOSA project

The SAMOSA framework constitutes a first step in our quest to generate in-class mixed samples, and it learns to generate convincing hybrids on toy-datasets like the MNIST-M dataset (see Figure 2.11). On more complex real data like the CIFAR-10 and SVHN datasets the framework also succeeds in identifying and transferring some non-semantic characteristics (e.g. lighting and coloration) to the main semantic image as shown on Figure 2.11. Interestingly, these slight

alterations already led to some improvements for semi-supervised frameworks on standard benchmarks at the time (CIFAR-10 and SVHN).

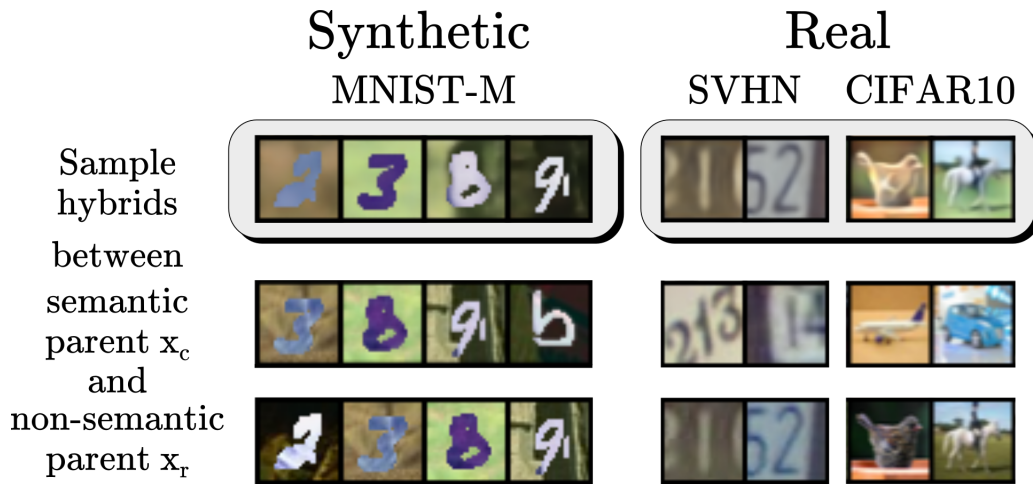


Figure 2.11. – Examples of hybrids obtained by SAMOSA (Mean Teacher backbone) on the synthetic MNIST-M dataset as well as the real SVHN and CIFAR10 datasets.

While SAMOSA does technically fulfill most of the objectives we outlined in this chapter, we found significant shortcomings in our approach:

- **The model only slightly modifies the semantic content** While the hybrids on Figure 2.11 indeed succeed in adding some non-semantic variations to the semantic content of samples in the dataset, the resulting images remain very close to their semantic parent. As such, SAMOSA does not truly generate mixed samples that combine a significant amount of content from multiple images.
- **The classifier does not benefit from hybrids it helped generate** The non-semantic content is identified from the content that is not useful to the classifier. Therefore, retraining the classifier with the hybrids generated by the framework only yields minimal performance improvements. In fact, the majority of the improvements observed stem from the regularization induced by the reconstruction on the classifier’s feature space.
- **Tying generator and classifier complicates training** Training the generator along with the classifier makes scheduling delicate and significantly increases training time as well as overhead: we typically conduct experiments on 2 gpus, even for very simple datasets like the CIFAR-10 and SVHN datasets. Furthermore, training some of the more complex semi-supervised classifiers like MixMatch and FixMatch proves fairly difficult. Moreover, the

strong augmentations that are part and parcel of these techniques interfere with the generative process and cause more variance in results.

To address this, we have since developed the SciMix framework which embeds the semantic content of one image into the context of another, and uses the resulting data augmentation to train semi-supervised classifiers from scratch.

2.4 SciMix: Embedding semantic content into other contexts

Our SAMOSA framework properly generates mixed samples that only contain semantic content from one image, and the slight non-semantic variations induced by the mixing process seem to help the model find more general solutions. The non-semantic variations however remain minimal, and the mixed samples therefore cannot truly be characterized as mixing content from both samples.

SciMix takes an opposite view to SAMOSA. Rather than attempt to identify and add non-semantic characteristics to a semantic parent, we directly embed semantic content into the general non-semantic context of the non-semantic parent. This ensures the mixed images contain a significant amount of content from the non-semantic parent. The work of the hybridizer becomes identifying the semantic content (with the help of a classifier) and adding it to the other image.

The main contributions we propose with SciMix after developing SAMOSA are therefore:

- A better exploration of our new mixing paradigm designed to create artificial in-distribution samples that embed the non-semantic content of one sample into the non-semantic context of another.
- A new auto-encoding architecture and associated learning scheme that trains a generator (in lieu of a decoder) to mix semantic and non-semantic contents. In particular, we purposefully train a model to separate semantic and non-semantic contents into two representations, and train a style-inspired generator to embed the semantic content (“style” code) into the non-semantic background (traditional input). In contrast to SAMOSA, we use the semantic latent representations as a style or modulating input to the asymmetrical generator.
- A new learning process to leverage our new mixing data augmentation separately from the hybridization process. We show mixed samples can

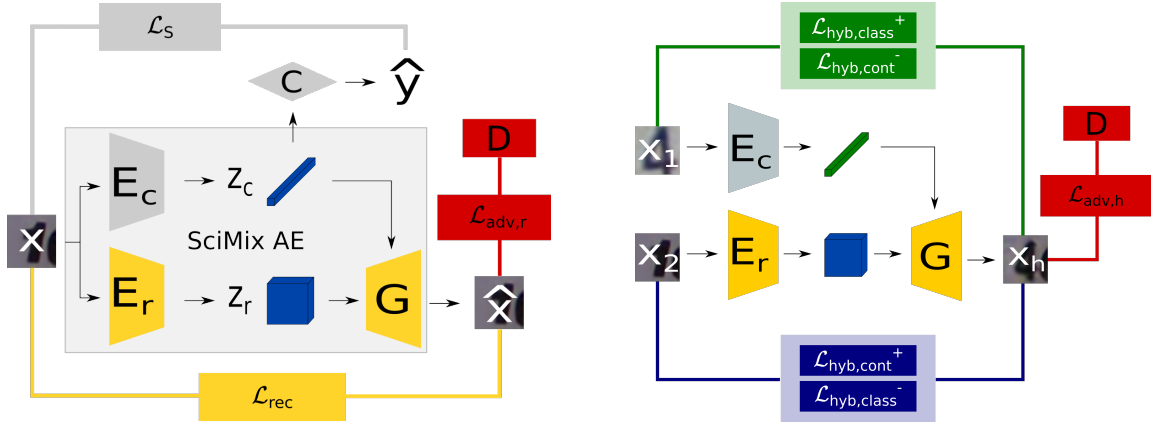
be used to optimize an additional supervised objective that significantly improves classifier performance.

2.4.1 Learning to generate hybrids

Auto-encoding architecture Our framework is based on a novel auto-encoder architecture presented in [Figure 2.12a](#) that treats semantic information as a global characteristic of an encoded image. An input x is projected into a semantic latent space z_c by an encoder E_c as well as a complementary non-semantic latent space z_r by an encoder E_r . While our framework should primarily be understood as an auto-encoder framework, the distinct nature of the latent spaces z_c and z_r requires more careful consideration. We choose in this paper to focus on the definition and exploitation of the semantic features z_c , and simply treat z_r as information irrelevant to z_c . In other words, we design the framework so that the semantic information z_c controls what the generator G reconstructs. As the notion of semantic information is fundamentally tied to that of the tasks under consideration, we define z_c with respects to a classifier. More precisely, we treat the semantic latent space z_c as the feature space of a classifier. To this end, we add a linear neural layer C on top of z_c (see [Figure 2.12a](#)) that outputs a class prediction \hat{y} . Note that $E_c \circ C$ therefore constitutes a standard CNN classifier (He et al. 2016b; Zagoruyko and Komodakis 2016). We ensure the classifier $E_c \circ C$ correctly learns semantic information through classification loss term \mathcal{L}_S on its output¹.

Contrarily to (Park et al. 2020), we complete the encoding with a generator G that computes a reconstruction \hat{x} using the non-semantic features z_r as direct inputs and the semantic features z_c as style codes. This makes the non-semantic content z_r easier to transfer, which is fortunate considering we can ensure semantic transfer more easily through a classifier $E_c \circ C$ (as opposed to SAMOSA). As we treat the model as an autoencoder, we use a reconstruction loss \mathcal{L}_{rec} to tie the reconstruction \hat{x} to the input x . We further refine this reconstruction by using an adversarial critic to smooth out details (Park et al. 2020). We add a discriminator network D (see [Fig. 2.12a](#)) to predict whether the considered image is a real sample or a reconstruction. Conversely, E_r , E_c and G are trained to fool D into

¹. \mathcal{L}_S can correspond to any classifier training framework (e.g. supervised training, FixMatch (Sohn et al. 2020)). In semi-supervised experiments, we use Mean Teacher (Tarvainen and Valpola 2017) as a simple classifier guide in order to leverage SSL datasets.



(a) Semantic auto-encoder of SciMix's generator. x is encoded into z_c (made semantic by \mathcal{L}_S) and z_r , which are decoded into \hat{x} optimized by $\mathcal{L}_{AE} = \mathcal{L}_{rec} + \mathcal{L}_{adv,r}$.

(b) Hybridization in SciMix's generator. Semantic components are extracted from x_1 and x_2 to obtain a hybrid optimized through \mathcal{L}_{hyb} (detailed in 5 sub-losses here).

Figure 2.12. – **Overview of the SciMix generator architecture.** SciMix trains an auto-encoder with two latent spaces, one of which is semantic

seeing reconstructions \hat{x} as real images. The resulting loss $\mathcal{L}_{adv,r}$ serves to improve reconstructions learned through the reconstruction loss.

$$\begin{aligned} \mathcal{L}_{AE} = \mathcal{L}_{rec} + \mathcal{L}_{adv,r} = & \sum_{x \in \mathcal{D}} \|x - G(E_c(x), E_r(x))\|_2 \\ & + \sum_{x \in \mathcal{D}} -\log(D(G(E_c(x), E_r(x))))). \end{aligned} \quad (2.8)$$

Finally, our learning scheme is based on the minimization of the loss \mathcal{L}_{gen} composed of \mathcal{L}_S and \mathcal{L}_{AE} , plus an additional loss \mathcal{L}_{hyb} term:

$$\mathcal{L}_{gen} = \mathcal{L}_{AE} + \mathcal{L}_S + \mathcal{L}_{hyb}, \quad (2.9)$$

which is described in the following hybridizing scheme.

Hybridization losses Simply training the auto-encoder architecture, even with z_c as the feature space of a classifier, is not enough to ensure that hybrids correctly inherit characteristics from their parents. To force the model to properly inject semantic content into the general background of known samples, we design explicit hybridization losses (studied more closely in Appendix Section A.4.2)

$$\begin{aligned} \mathcal{L}_{hyb} = & \mathcal{L}_{hyb,class}^+ + \mathcal{L}_{hyb,cont}^+ \\ & + \mathcal{L}_{hyb,class}^- + \mathcal{L}_{hyb,cont}^- + \mathcal{L}_{adv,h}. \end{aligned} \quad (2.10)$$

$\mathcal{L}_{hyb,class}^+$ explicitly trains our model to rely on z_c to generate the main semantic object in the generated reconstruction/hybrid. Indeed, we rely on the classifier $E_c \circ C'$'s ability to identify and classify the main object in inputs (see Figure 2.12b).

Put plainly, we generate hybrids $x_h = G(E_c(x_1), E_r(x_2))$ from pairs of samples in a batch and obtain logits predictions $C(E_c(x_h))$ for those hybrids. $\mathcal{L}_{hyb,class}^+$ optimizes the model so that this prediction on the logits match the prediction on the semantic parent x_1 of x_h . Importantly, we only optimize the hybridization process that generates x_h : we do not optimize the classifier's prediction on x_h or x_1 . The idea is that the autoencoder learns to place x_h in the right class manifold, while said class manifold does not move to accommodate x_h :

$$\mathcal{L}_{hyb,class}^+ = \sum_{x \in \mathcal{D}} \|C(E_c(x_1)) - C(E_c(G(E_c(x_1), E_r(x_2))))\|_2. \quad (2.11)$$

Similarly, $\mathcal{L}_{hyb,cont}^+$ optimizes the model so that a generated hybrids x_h 's non-semantic representation $z_{r,h}$ matches its non semantic parent x_2 's non semantic component $z_{r,2}$. As in the semantic case, we only optimize the generative process that leads to the generation of x_h but do not optimize E_r to project x_h close to its non-semantic parent:

$$\mathcal{L}_{hyb,cont}^+ = \sum_{x \in \mathcal{D}} \|E_r(x_2) - E_r(G(E_c(x_1), E_r(x_2)))\|_2. \quad (2.12)$$

We also train hybrids to differ from their parents through the negative semantic hybridization loss $\mathcal{L}_{hyb,class}^- = \sum_{x \in \mathcal{D}} -\|C(E_c(x_2)) - C(E_c(G(E_c(x_1), E_r(x_2))))\|_2$ and the negative non-semantic hybridization loss $\mathcal{L}_{hyb,cont}^- = \sum_{x \in \mathcal{D}} -\|E_r(x_1) - E_r(G(E_c(x_1), E_c(x_2)))\|_2$. In practice, this means maximizing the distance between hybrids and semantic (resp. non-semantic) parent in non-semantic (resp. semantic) space.

To ensure the quality of generated hybrids, we train the discriminator D to also recognize hybrids as synthetic images. With this discriminator we can simply add an adversarial loss term $\mathcal{L}_{adv,h}$ to ensure hybrids look realistic (as far as D is concerned).

2.4.2 Training a classifier by leveraging our Data Augmentation

We now have a novel mixing data augmentation that can embed the semantic content of one sample to the non-semantic context of other samples, given a trained generator. This provides a useful and new way to improve any standard training method “X” by adding a single additional loss term $\mathcal{L}_{\text{contradict}}$: $\mathcal{L}_{\text{SciMix}} = \mathcal{L}_X + \mathcal{L}_{\text{contradict}}$.

Generating hybrids given a trained autoencoder Generating hybrids given a trained model is straightforward (Figure 2.12b shows how a hybrid is mixed). Specifically, given samples $x^{(1)}$ (with known label $y^{(1)}$) and $x^{(2)}$, we extract the relevant features $z_c^{(1)} = E_c(x^{(1)})$, $z_r^{(1)} = E_r(x^{(1)})$, $z_c^{(2)} = E_c(x^{(2)})$ and $z_r^{(2)} = E_r(x^{(2)})$. $x_h = G(z_c^{(1)}, z_r^{(2)})$ is now a sample with class $y^{(1)}$. As a conservative measure, we only keep the generated hybrid if $C(E_c(x_h)) = y^{(1)}$ to avoid disturbing decision boundaries too much. Note that with this, we generate a strong augmentation of x_1 and teach the classifier to group x_1 with its strongly augmented version in a similar line to work in contrastive representation learning (He et al. 2020).

Training a new classifier f We now propose a way to leverage our novel hybrids to improve the training of standard models such as Mean Teacher (Tarvainen and Valpola 2017) or FixMatch (Sohn et al. 2020). To this end, we compute hybrids that mix the semantic content of each sample in the batch with the non-semantic content of other samples in the batch. We leverage those hybrids by optimizing an additional loss:

$$\mathcal{L}_{\text{contradict}} = \sum_{x_c, x_r \in \mathcal{B}, \text{perm}(\mathcal{B})} [l_{\text{MSE}}(f(x_h), \alpha * f(x_c) + (1 - \alpha)f(x_r))]. \quad (2.13)$$

This new loss takes advantage of our mixing paradigm by mostly imputing the semantic parent’s label to our hybrids, with only a slight dependence on the non-semantic parent to acknowledge the imperfection of the mixing process. Contrarily to standard mixing augmentations, the ratio $\alpha > 0.5$ is a fixed hyperparameter (in the spirit of label smoothing (Szegedy et al. 2016)).

2.5 Results

We detail here the results experimental investigation we conducted to evaluate the performance of SAMOSA and SciMix. To this end, we mostly studied the semi-supervised problem on the CIFAR10 (Krizhevsky and Hinton 2009) and SVHN (Netzer et al. 2011) datasets.

We applied our frameworks over well studied semi-supervised backbones like Mean Teacher (Tarvainen and Valpola 2017), MixMatch (Berthelot et al. 2019), and FixMatch (Sohn et al. 2020) (refer to Section 2.3.4 and Section 2.4.2 for how we apply our frameworks to these methods). More precisely, we chose Mean Teacher for both SAMOSA and SciMix as a reference consistency-based baseline. Beyond its widespread use in SSL, consistency induces a stabilization we feel would help extract invariant semantic features. For SAMOSA, we use MixMatch as a backbone to illustrate interactions of SAMOSA’s generator with more modern methods that make use of mixing techniques (such as CutMix, CowMix, ICT and ReMixMatch). Since SciMix separates the generator from the trained semi-supervised classifier, this interaction is more minimal. We therefore use FixMatch instead of MixMatch as a backbone. FixMatch is a state-of-the-art SSL method based on strong augmentation, and often serves as the reference or backbone in the literature (J. Li et al. 2021; B. Zhang et al. 2021).

For both frameworks, we operate on a standard WideResNet-28-2 (Zagoruyko and Komodakis 2016) for our classifiers (both f and $E_c \circ C$). E_r follows the same architecture as E_c . In the case of SAMOSA, hybrids are generated by a decoder D that follows an inverted 13-layer 4-4-4 CNN architecture, with D_{pre} being a 4-4 block and D_{post} being made up of the last 4 block and final convolution. For SciMix, the hybrids are generated by a generator G that is a StyleGanv2 (Karras et al. 2020) architecture. Hyperparameters and optimizers were generally taken to follow settings reported in the base methods’ original papers (Tarvainen and Valpola 2017; Berthelot et al. 2019; Sohn et al. 2020).

We report the *mean* \pm *std* classification accuracy over 3 seeded runs for varying numbers of labeled samples in a dataset (the rest are treated as unlabeled). The SciMix generators used to train a model with N labeled samples are also trained with only N labeled samples. One generator is trained per setting, and classifiers trained with the generator’s mixed samples are trained on the same split of labeled/unlabeled data to avoid information leakage. More details for all experiments are given in Appendix.

Method	CIFAR10		
	250	500	1000
Purely supervised (lower bound)	27.8 ± 0.9	35.4 ± 1.9	43.5 ± 2.4
Mean Teacher ¹ , (Tarvainen and Valpola 2017)	61.3 ± 3.3	76.4 ± 3.1	87.6 ± 0.3
SAMOSA w/ Mean Teacher (ours)	68.1 ± 3.3	82.4 ± 1.3	88.7 ± 0.3
MixMatch ¹² , (Berthelot et al. 2019)	82.4 ± 0.4	86.8 ± 0.2	90.4 ± 0.1
SAMOA w/ MixMatch (ours)	84.1 ± 2.2	89.4 ± 0.8	90.7 ± 0.3

Table 2.1. – **Comparative accuracies (%) with SAMOSA as an add-on module on CIFAR10.**

2.5.1 Performance improvements

As the final objective of this project is to improve the performance of semi-supervised classifiers, we first discuss how SAMOSA and SciMix help regularize the networks. It is worth noting the experimental settings and codebases differ significantly between the two. Direct comparison of the performances is therefore not straightforward and will be discussed in [Section 2.5.2](#).

2.5.1.1 SAMOSA

We show here that SAMOSA can improve quantitative performance when added to Mean Teacher and MixMatch ([Table 2.1](#)). In each table, we also report for reference the accuracy of models trained in a purely supervised fashion on the available labeled samples as a lower bound.

Mean Teacher (MT) We evaluate a first application of SAMOSA for augmentation to show improvements on Mean Teacher (the procedure is detailed more closely in [Appendix Algorithm A.1](#)). We train the model normally for 300 epochs (with the reconstruction module), then we hybridize every labeled sample with 10 unlabeled samples. For every generated hybrid, we keep the artificial example only if it still gets predicted by the model as being part of the right class, otherwise the hybrid is replaced by its semantic “parent”. The model is then retrained with this additional labeled data over 300 epochs. Afterwards, another hybridization procedure is repeated and a final training is conducted, still over the same number of epochs.

The model is trained using a SGD optimizer with cosine learning rate (base 0.2 learning rate) for 300 epochs over the unlabeled samples for CIFAR10 for one training cycle. After each training and subsequent augmentation step, the learning rate is reset and training resumes (for an overall 900 epochs). In the following training passes, the model is only optimized over the augmented dataset (instead

of the true dataset) from epoch 150 to 250 of each cycle (following discussions in (Perez and J. Wang 2017; Gontijo-Lopes et al. 2021)) with $\lambda_{recons} = 0.25$ and $\lambda_{SAMOSA} = 0.5/0.1$. Exact details in the supplementary material.

As a baseline, we check the performance of the model trained under the same procedure (3 training cycles) but with no reconstruction regularizer, and no artificial samples. Table 2.1 shows improvements from using the SAMOSA framework on top of Mean Teacher. Notably, we have very noticeable gains for 250 labels, which suggests the method is particularly useful when labeled information is lacking. To explore performance with very few labels, we furthermore tested the model with 100 labels on SVHN. An important accuracy gain from 62.5 ± 3.7 to 66.2 ± 2.2 is observed which is significant given such very low label settings are especially interesting in applied settings.

MixMatch (Mix) We showcase a more intricate use of SAMOSA on MixMatch by directly incorporating our augmentation process in MixMatch’s native hybridization (MixUp) as detailed in Appendix Algorithm A.2. We train the reconstruction module along the base classifier model, as well as optimize for the reconstruction regularizer. Every batch, with probability $p = 0.2$, we replace the MixUp examples with Hybrids generated from our reconstruction module. For every reconstructed hybrid, we keep the label/pseudolabel corresponding to its semantic “parent”. Contrarily to the Mean Teacher case, we generate hybrids that have both labeled and unlabeled samples as semantic “parents” (as is done by MixUp in MixMatch) and leverage the pseudo-label MixMatch naturally generates throughout its course for MixUp. Exact details are given in the supplementary material, but are similar to the Mean Teacher ones. We follow (Berthelot et al. 2019) and report results from a weight averaged model.

As a baseline comparison, we check the performance of the model trained under the same procedure (which is basically the normal training procedure) but with no reconstruction regularizer, and no artificial samples. As can be seen from Table 2.1, sizable gains are achieved on both the 500 and 250 labels CIFAR10 settings, and are consistent even when 3 runs are not enough to definitely verify improvements. Interestingly, adding SAMOSA to MixMatch increases the variance of the model which is not the case in the Mean Teacher case. The use of MixUp hybrids in MixMatch strongly influences the hybrids generated by SAMOSA (discussed in Section A.3.1). This and the random nature of MixUp could lead to a stronger variability in the quality of hybrids learned by a SAMOSA generator. Considering how reliant SAMOSA’s classifier is on the quality of the generated hybrids for regularization, we believe this explains the higher variance on the MixMatch variant of SAMOSA.

2. Different setting from (Berthelot et al. 2019) for fast training.

(a) Mean Teacher

Method	CIFAR10			SVHN	
	100	250	500	60	100
Mean Teacher, (Tarvainen and Valpola 2017)	40.5 ± 6.4	63.1 ± 0.9	72 ± 3	48.7 ± 23.0	82.3 ± 5.5
SciMix w/ Mean Teacher	46.4 ± 1.2	68.0 ± 1	77.2 ± 0.5	83.4 ± .5	87.3 ± 2.8

(b) FixMatch

Method	CIFAR10	SVHN
	100	60
FixMatch, (Berthelot et al. 2019)	88.6 ± 0.7	96.4 ± 0.3
SciMix w/ FixMatch	90.7 ± 0.2	96.5 ± 0.9

Table 2.2. – **Gains from using SciMix.** Mixing samples with SciMix as a data augmentation improves the performance of Mean Teacher and FixMatch. Significant accuracy (%) gains are observed on CIFAR10 (with 100, 250 and 500 labels) and SVHN (with 60 and 100 labels).

2.5.1.2 SciMix

Table 2.2 shows that adding our optimization on hybrids with $\mathcal{L}_{contradict}$ - as described in Section 2.4.2 - does indeed lead to improved performance on CIFAR10 and SVHN. Indeed, training with SciMix hybrids leads to significant accuracy gains with a Mean Teacher classifier with a wide range of labeled samples. We also observe improvements over FixMatch on very low labeled settings (FixMatch performance quickly saturates on higher labeled settings). Interestingly, two concurrent behaviors can be observed on Mean Teacher: SciMix hybrids become more useful when less labeled samples are available but the quality of generated hybrids becomes unreliable with few labeled samples. Indeed, at 100 labeled samples on CIFAR10, the Mean Teacher classifier used to train the generator is too weak to provide very useful hybrids. With a strong SciMix hybridizer (trained with all samples), SciMix data augmentation would bring a Mean Teacher classifier to an accuracy of 60.4 ± 1.7 with 100 labels on CIFAR10.

2.5.2 Improvements of SciMix over SAMOSA

As discussed in Section 2.3.5, SciMix was created to address many of the problems observed in SAMOSA:

- **The model only slightly modifies the semantic content** Hybrids generated by SAMOSA on real datasets only incorporate minor non-semantic characteristics like color tint. As such, the generated hybrids do not truly mix the semantic contents of some sample with the non-semantic contents of others.

SciMix solves this issue as can be observed from the hybrids generated by both frameworks. [Section 2.5.2.1](#) shows examples of hybrids generated by both frameworks and discusses the improvements brought by SciMix.

- **The classifier does not benefit from hybrids it helped generate** The classifier trained by SAMOSA benefits little from training on the hybrids generated by SAMOSA. We posit that this stems both from the implicit regularization of the classifier by the generative objectives, and from the fact the hybrids incorporate non-semantic information identified by the classifier itself.

[Section 2.5.2.2](#) highlights this behavior for SAMOSA, and recovers a similar observation for a variant of SciMix which re-uses the weights of the classifier used to train the hybrid generator. In the latter cases, we show keeping the weights of the previous classifiers in fact leads to worse results than randomly initializing the weights of our semi-supervised classifier.

- **Tying generator and classifier complicates training** A significant issue when attempting to train new semi-supervised algorithms with the SAMOSA framework is that the schedules of the generator and classifier must be synchronized. Furthermore, training an autoencoder alongside our classifiers adds a large overhead to the SAMOSA framework which makes deployment of the framework unrealistic and cumbersome.

Appendix [Section A.4.1](#) discusses the costs incurred by the SAMOSA approach in more details and gives some perspective on the types of schedules we use to train semi-supervised classifiers and hybrid generators.

2.5.2.1 Comparison of SAMOSA and SciMix hybrids

Comparing hybrids obtained by SAMOSA [Figure 2.13](#) to those generated by SciMix [Figure 2.14](#) shows the benefits of our SciMix framework.

While SAMOSA’s hybrids only incorporate minor non-semantic variations like lighting or coloration into the general content of their semantic parents, SciMix hybrids inherit large amounts of content from their non-semantic parent as well. For instance, the street numbers on [Figure 2.14](#) inherit nearly all the non-semantic content of their non-semantic parent (background, color, font, ...), with only the actual semantic street number being inherited from the semantic parent. Interestingly, even the non-central digits are correctly identified as non-semantic by the framework and kept from the non-semantic parent.

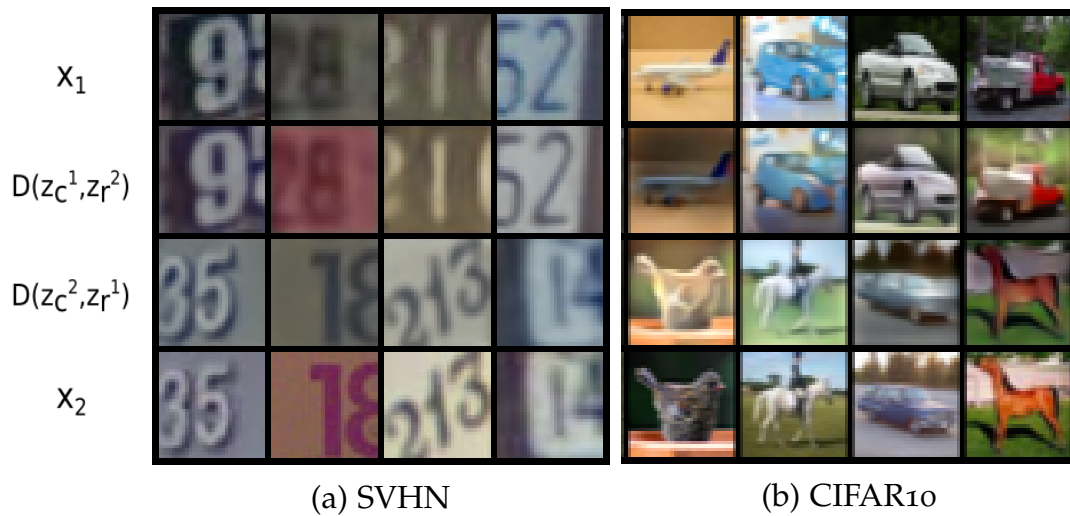


Figure 2.13. – **Examples of SAMOSA Hybrids** between true samples x_1 and x_2 . Results for the Mean-Teacher SAMOSA trained on SVHN (100 labels) and CIFAR10 (1000 labels). (Line 2 shows hybrids with the semantic content of Line 1 and non-semantic content of Line 4. Line 3 shows the opposite.)



Figure 2.14. – **Examples of SciMix hybrids**. The hybrids properly mix semantic and non-semantic contents. Same format as [Figure 2.13](#)

This stems from the fundamental difference in design between SAMOSA and SciMix: where SAMOSA attempts to discover non-semantic variations in the content of images, SciMix works to embed the semantic content of one sample into the general context of the other. Although this approach requires inverting the roles of the semantic and non-semantic latent codes in the asymmetric generator and

(a) SAMOSA on CIFAR 10		(b) SciMix on SVHN	
Method	CIFAR 10	Method	SVHN
	500		60
Mean Teacher, (Tarvainen and Valpola 2017)	76.4 ± 3.1	Mean Teacher, (Tarvainen and Valpola 2017)	48.7 ± 23.0
SAMOSA Gen. classifier w/ Mean Teacher (no hybrids)	82.1 ± 1.5	SciMix Gen. classifier w/ Mean Teacher (no hybrids)	72.0 ± 2.7
SAMOSA Gen. classifier w/ Mean Teacher	82.4 ± 1.3	SciMix Gen. classifier w/ Mean Teacher	72.6 ± 2.3
SAMOSA New classifier w/ Mean Teacher	N/A	SciMix New classifier w/ Mean Teacher	$83.4 \pm .5$

Table 2.3. – **Ablation on the gains from in-class MSDA.** We compare the classifier trained with only Mean Teacher, trained with the generator but no hybrids, trained with generator and then with hybrids, and finally of a new classifier trained with the hybrids. Although they were evaluated on different settings, we show results for both SAMOSA and SciMix.

introducing explicit hybridization objectives, this enables the model to generate hybrids that satisfy the requirements outlined at the outset of this chapter.

2.5.2.2 Regularization vs. Data Augmentation

SAMOSA trains the evaluated semi-supervised classifier alongside the generative framework. While this is efficient in a sense, Table 2.3a shows classifiers trained in this manner benefit very little from further training with hybrids generated by the framework. This is not wholly surprising: hybrids only incorporate non-semantic variations that do not impact the classifier by construction, and the classifier already implicitly benefits from the hybridization due to the regularization induced by the generative framework. SciMix sidesteps this issue entirely by training a separate classifier with the generative framework.

The first three lines of Table 2.3b show the same behavior as SAMOSA: the classifier SciMix trains to guide the generator does not reliably benefit from training on the generated hybrids. Contrarily to SAMOSA however, SciMix uses its hybrids to train a new classifier from scratch which leads to much more reliable improvements as can be seen from the last line of Table 2.3b.

2.5.2.3 Comparison of quantitative results in comparable settings

As mentioned in Section 2.5.1, comparing the quantitative performances of SAMOSA and SciMix is not straightforward. Indeed, fair comparison of baselines to SAMOSA typically require using fairly complex training procedures. For instance, on the Mean Teacher setting, SAMOSA trains the network three times to incorporate new hybrids in each training cycle.

Method	CIFAR10			
	250		500	
	SAMOS A	SciMix	SAMOS A	SciMix
Mean Teacher	57.7 ± 4.0	63.1 ± 0.9	69.6 ± 1.4	72 ± 3
In-class MSDA w/ Mean Teacher	60.4 ± 0.3	68.0 ± 1	75.7 ± 3.4	77.2 ± 0.5
Gains	+2.7	+4.9	+6.1	+5.2

Table 2.4. – **Comparison of SAMOSA and SciMix** in mostly comparable conditions on the Mean Teacher backbone.

Interestingly, the training settings of SciMix and SAMOSA with the Mean Teacher backbone are otherwise fairly similar. As it was established that explicitly training on the hybrids contributes fairly little to the overall performance of the SAMOSA framework, we feel it is fair to compare results at the end of 1 training cycle with results obtained by SciMix

Table 2.4 shows comparable gains from SAMOSA and SciMix with 500 labeled samples on CIFAR 10, and significantly better gains from SciMix with 250 samples. This suggests the stronger hybrids generated by SciMix offer better regularization than SAMOSA in very low-label situations where the classifier struggles to learn invariance to non-semantic content. It must however also be noted the gains could be partially attributed to the fact SciMix does not use the same classifier as its generator.

2.5.3 Further analysis of SciMix results

We provide here two further results obtained from our study of the SciMix Data Augmentation. The following experiments compare the quality of the SciMix hybrids against different mixing alternatives.

2.5.3.1 Preservation of semantic and non-semantic characteristics in hybrids

We quantify how well the generated hybrids x_h inherit properties from the semantic and non-semantic parents x_c and x_r through the metrics s_c and s_r . The semantic transfer rate s_c is the accuracy of an “oracle” classifier (trained on the entire dataset, as a proxy for human evaluation of hybrid labels) over a dataset built from hybrids that are assumed to have inherited the label of their semantic parent. Conversely, the non-semantic preservation rate s_r is the proportion of hybrids x_h that are closer to the non-semantic parent x_r in pixel space (ie, $\|x_h - x_c\| > \|x_h - x_r\|$).

Method		SVHN
Semantic transfer s_c	FDA (Y. Yang and Soatto 2020)	77.0
	AdaIN (X. Huang and Belongie 2017)	92.1
	MixUp (H. Zhang et al. 2018; Inoue 2018)	94.2
	SciMix	95.2
Non-Semantic preservation s_r	FDA (Y. Yang and Soatto 2020)	22.4
	AdaIN (X. Huang and Belongie 2017)	60.0
	MixUp (H. Zhang et al. 2018; Inoue 2018)	00.0
	SciMix	98.2

Table 2.5. – **Transfer rates of semantic and non-semantic contents.** SciMix hybrids properly mix semantic and non-semantic contents.

Table 2.5 shows SciMix compares favorably to texture altering hybrids (FDA, AdaIN) and standard mixing augmentations (MixUp with the label of the dominant samples as suggested in (Inoue 2018)) on a strong generator (SVHN 250 labels). While most existing hybridizations do tend to preserve the semantic content, SciMix shines in that it transfers semantic content while keeping hybrids very close to their non-semantic parent. Indeed, other hybrids remain much closer to their semantic parent (always the case - by design - for MixUp).

Method	CIFAR ₁₀	SVHN
	250	60
Mean Teacher, (Tarvainen and Valpola 2017)	63.1 ± 0.9	48.7 ± 23.0
Mean Teacher + MixUp (H. Zhang et al. 2018)	64.8 ± 3.5	61.8 ± 1.0
Mean Teacher + CutMix (Y. Yang and Soatto 2019)	55.0 ± 7.5	17.3 ± 3.7
SciMix w/ Mean Teacher (ours)	68 ± 1.0	83.4 ± 0.5

Table 2.6. – **Comparison of SciMix with other MSDA on very few labels.**

2.5.3.2 Comparison to Mixing Data Augmentation

We now show that in very low label settings, the “artificial” labeled samples SciMix can outperform the regularization offered by more traditional mixing Data Augmentation. Table 2.6 shows that SciMix mostly outperforms MixUp and CutMix for CIFAR₁₀ with 250 labeled samples (the generator is too weak with 100 labels) and SVHN with 60 labeled samples (hardest setting). While MixUp does perform similarly to SciMix on CIFAR₁₀, this is likely due to the low performance of the classifier $E_c \circ C$ trained with only 250 labels.



Figure 2.15. – SciMix hybrids mix semantic and non-semantic contents on CUB-200.

Method	CUB-200	
	64×64	96×96
Supervised	58.9 ± 1.0	65.2 ± 0.8
SciMix w/ Supervised	60.2 ± 0.6	65.6 ± 0.9

Table 2.7. – Impact of SciMix data augmentation for the CUB-200 dataset at resolutions 64×64 and 96×96 .

2.5.4 Pushing SciMix on CUB-200

We finally push SciMix further by studying versions of the more complex **Caltech-UCSD Birds 200** (CUB-200) dataset (Welinder et al. 2010) (6033 pictures of 200 bird species). Given CUB-200 inherently presents few labels, we directly study how fully supervised training benefits from SciMix on low labels settings (\mathcal{L}_S is a standard cross-entropy loss). Furthermore, we take advantage of CUB-200’s higher native resolution to go beyond the limitations of 32×32 images in CIFAR 10 and SVHN: we use SciMix on commonly studied input sizes 64×64 (e.g. Tiny ImageNet (Chrabaszcz et al. 2017)) and 96×96 (e.g. STL-10 (Coates et al. 2011)).

Quality of generated hybrids As can be observed on Figure 2.15a, the SciMix autoencoder learns to generate interesting hybrids for different resolutions of the CUB-200 dataset. While the lower resolution 64×64 hybrids inherit more semantic characteristics of the relevant parent, 96×96 hybrids still retain semantic patterns tied to the semantic parent’s class.

Interestingly, SciMix has no difficulty producing hybrids close to their non-semantic parent as can be shown in [Figure 2.15b](#) by reprising the analysis of the non-semantic transfer rate s_r from [Section 2.5.3.1](#). Analyzing the semantic transfer rate s_c proves more difficult as our best “oracle” classifiers remain unreliable (around 60% accuracy). Nevertheless, the semantic transfer rates s_c in [Figure 2.15b](#) indicate hybrids generated by SciMix are properly classified by the oracle classifier as having inherited their semantic parent’s class about 40% of the time (orders of magnitude more than attributable to random chance).

Performance gains [Table 2.7](#) shows a fully supervised version of SciMix data augmentation improves supervised models on both 64×64 and 96×96 versions of CUB-200. This demonstrates that while SciMix augmentation strongly benefits from a large amount of unlabeled data, it can still generate hybrids diverse enough to benefit training with only a small set of data to generate hybrids from.

2.6 Conclusion

In this chapter, we explore an alternative MSDA paradigm that creates in-class samples by combining the semantic content of one sample and non-semantic content of another. This new paradigm helps teach invariance to a number of non-semantic characteristics, which is particularly useful in low-label settings like semi-supervised learning. To test this hypothesis, we design neural frameworks that learn to separate semantic and non-semantic contents in images and recombine them into in-class hybrids before applying them in semi-supervised scenarios.

We first propose SAMOSA, an auto-encoding framework that identifies and adds some non-semantic variations to a base semantic image with the aid of an asymmetric generator inspired by style transfer techniques. SAMOSA takes advantage of the generative process to train a semi-supervised classifier both to guide hybridization and to regularize the classifier.

We then take the opposite view with SciMix by training a hybridizer that identifies and embeds semantic content into a base image’s non-semantic context. By inverting the roles of semantic and non-semantic content in the process, non-semantic content does not have to be discovered by the framework: it remains non-semantic unless specifically identified as semantic. Contrarily to SAMOSA, SciMix uses a powerful StyleGanV2 (Karras et al. 2020) asymmetrical generator and explicit hybridizing losses to learn to embed semantic content into non-semantic context. We also streamline the training setting with SciMix by training a

semi-supervised classifier with SciMix hybrids after a hybrid generator is trained, with no additional regularization from the generative framework.

Experiments show both SAMOSA and SciMix generate hybrids that benefit the training of standard semi-supervised frameworks like Mean Teacher or FixMatch. While SAMOSA’s hybrids only incorporate surface level non-semantic characteristics, qualitative evaluation shows SciMix generates hybrids that inherit significant amounts of contents from both of their parents. After showing the SciMix framework also streamlines a number of issues in the SAMOSA framework, we further validate the quality of the SciMix hybrids and the importance of its components. Finally, we show SciMix can also generate 64×64 or 96×96 hybrids on a fine grained setting with few labels per class and no unlabeled data like CUB-200.

All told, SciMix introduces a realistic way to generate in-class hybrids as we set out to do in this chapter. Empirical evidence suggests these in-class hybrids do help train semi-supervised models, especially in low label settings. Unfortunately, this also highlights the current limitation of the method: as we are reliant on an auxiliary classifier to guide the generative framework, the quality of our hybrids decreases in lower label settings. There are fortunately a number of solutions to this problem like managing to train stronger semi-supervised auxiliary classifiers along the generative framework which could be explored in further work.

This alternative point of view on mixing data augmentation addresses the question of soft labels by generating samples that only belong to one class and therefore use hard labels. It does not however directly address the mixing of different semantic contents in a single mixed image. The following chapters of this thesis will endeavor to treat mixed semantic contents in MSDA.

MIXING SAMPLES DATA AUGMENTATIONS AS COMPRESSED REPRESENTATIONS FOR MULTI-INPUT MULTI-OUTPUT TRAINING

Contents

3.1	Introduction	49
3.2	Related Work	51
3.2.1	Ensembling	51
3.2.2	MIMO	52
3.2.3	Positioning	53
3.3	MixMo: Multi-Input Multi-Output MSDA	53
3.3.1	General overview	55
3.3.2	Mixing inputs and balancing concurrent subnetworks	56
3.3.3	From manifold mixing to MixMo	59
3.3.4	Main experimental results	59
3.3.5	MixMo analysis on CIFAR-100 w/ WRN-28-10	64
3.3.6	Pushing MixMo further: Tiny ImageNet	67
3.3.7	Takeaways from the MixMo project	67
3.4	MixShare: Feature sharing between MIMO subnetworks	69
3.4.1	MIMO Subnetworks do not share features	70
3.4.2	How can subnetworks share features?	73
3.5	Conclusion	76

3.1 Introduction

Mixing samples data augmentations are often treated as samples with semantic content from multiple classes (H. Zhang et al. 2018). While this is a mostly accurate assessment, it fails to capture the exact nature of mixing samples data augmentations: the mixed samples contain information on multiple samples, which can be of differing classes.

The soft labels used by MSDA techniques therefore only constitute a reduction of the information in a mixed samples. It is not that the sample represents something that is in-between two classes, rather the sample contains information on two samples that are each natural in-class samples. Perhaps surprisingly, the soft “summarized” labels have only recently started to come under scrutiny, and only in so far that the proportions of the classes in the mixed sample have been re-evaluated (J.-N. Chen et al. 2022).

In this chapter, we explore the idea of providing multiple predictions for a mixed sample - one for each original parent sample - instead of a single aggregated soft prediction. This significantly changes the effect of the augmentation: instead of teaching the model to make a single in-between prediction, we encourage the model to see multiple interpretations of the same mixed input.

The MIMO architecture (Havasi et al. 2021) slightly modifies standard convolutional networks to take multiple inputs (typically $M=2$) and yield M predictions. The M input/output pairs trained by MIMO each define a subnetwork with its own independent predictions. For all intents and purposes, providing multiple interpretations - one for each parent sample - of the same mixed input leads to separating the network into smaller subnetworks that predict the labels of the individual parent samples. Interestingly, we recover here our suggested MSDA scheme as MIMO combines the inputs similarly to a mixing augmentation and then makes separate predictions for each mixed image.

We propose to study the role and influence of mixing samples data augmentations in Multi-Input Multi-Output models. As a first step, we seek to improve the training process of independent subnetworks in MIMO frameworks to facilitate better performance of the ensembled subnetwork predictions. To a lesser extent, we aim to train cooperating subnetworks that can yield independent predictions while still benefiting from the regularizing effects of MSDA.

Through the course of this thesis, we have made 2 contribution towards this goal:

- **MixMo** We replace the suboptimal summing mixing mechanism at the heart of the seminal MIMO architecture by a more appropriate noiseless CutMix scheme that allows better predictions on the original network inputs. This modification, along with a rebalancing of subnetwork training losses in line with MSDA practices, aims to improve the individual performance of the trained subnetworks as well as the overall ensembled accuracy.
- **MixShare** We then put in evidence MIMO architectures’ peculiar tendency to train subnetworks that share no features or parameters. To recover some regularizing benefits from the MSDA procedure and train on smaller mod-

els, we introduce an unmixing mechanism that addresses the root cause of this behavior.

After introducing the MIMO architecture in more detail along with some literature on ensembling in [Section 3.2](#), we discuss the main contribution of this chapter in the form of the MixMo framework ([Section 3.3](#)). Finally, we will provide a quick investigation of MIMO models’ peculiar lack of feature sharing between subnetworks and the possible solutions in [Section 3.4](#).

The work conducted in this chapter of this thesis has led to two publications:

- RAME Alexandre* (equal contribution), SUN Rémy* (equal contribution), and CORD Matthieu. (2022) “MixMo: Mixing Multiple Inputs for Multiple Outputs via Deep Subnetworks”, in International Conference on Computer Vision.
- SUN Rémy, RAME Alexandre, MASSON Clément, THOME Nicolas and CORD Matthieu. (2022) “Towards efficient feature sharing in MIMO architectures”, in CVPR 2022 workshop on efficient deep learning for computer vision.

3.2 Related Work

This chapter mostly focuses on a MSDA-centric view of the Multi-Input Multi-Output architecture (Havasi et al. 2021). As the seminal paper is deeply rooted in the larger problem of ensembling, we provide here a short introduction to ensembling before discussing the MIMO framework proper.

3.2.1 Ensembling

Aggregating predictions from a diverse set of neural networks (*i.e.*, with different failure cases) strongly improves generalization (T. G. Dietterich 2000; Hansen and Salamon 1990; Lakshminarayanan et al. 2017). An ensemble of several small networks usually performs better than one large network empirically (Lobacheva et al. 2020).

Ensembling’s fundamental drawback is the inherent **computational and memory overhead**, which increases linearly with the number of members. This bottleneck is typically addressed by sacrificing either *individual* performance or *diversity* in a complex *trade-off*. Averaging predictions from several checkpoints on the training process, *i.e.* snapshot ensembles (G. Huang et al. 2017), fails to explore

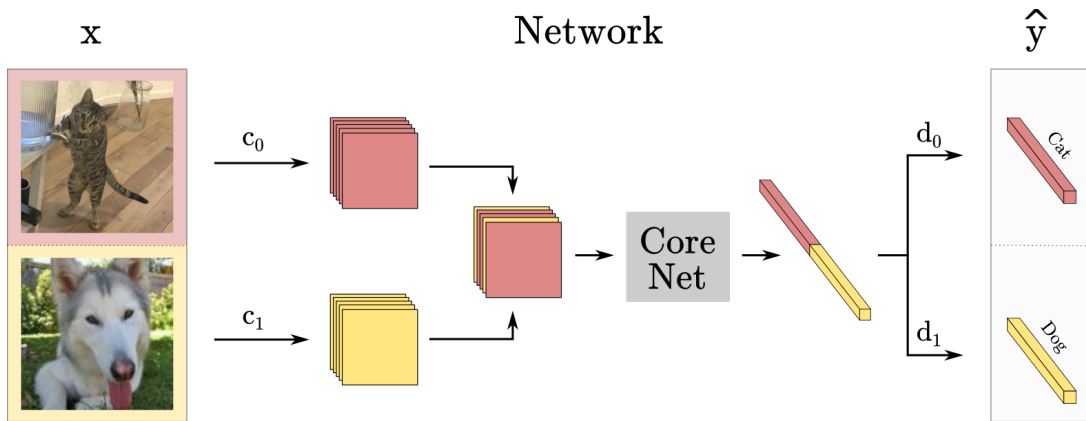


Figure 3.1. – Overview of the MIMO framework.

multiple local optima (Ashukha et al. 2020). So does Monte Carlo Dropout (Gal and Ghahramani 2016). The recent BatchEnsemble (Dusenberry et al. 2020) is parameter-efficient, yet requires multiple forward passes. TreeNets (S. Lee et al. 2015) reduce training and inference cost by sharing low-level layers. This however always comes at the cost of reduced diversity.

The multi-input multi-output strategies we focus on here aim to address this issue.

3.2.2 MIMO

Multi-input multi-output (MIMO) strategies (Havasi et al. 2021; Rame et al. 2021) provide an interesting solution to this conundrum by ensembling for virtually free. Through their multiple inputs and outputs, MIMO frameworks train independent subnetworks within a base network. Thanks to the sparse nature of large neural networks (Malach et al. 2020), the resulting subnetworks yield strong and diverse predictions that can be ensembled. As shown on Figure 3.1 with $M = 2$, the M inputs are embedded by M subnetworks with no structural differences. Thus, we have M (inputs, labels) pairs in training: $\{(x_i, y_i)\}_{0 \leq i < M}$. More precisely, M **images are fed to the network at once**. The M inputs are encoded by M distinct convolutional layers $\{c_i\}_{0 \leq i < M}$ into a shared latent space before being aggregated through a summing operation. This representation is then processed by the core network into a single feature vector, which is classified by M dense layers $\{d_i\}_{0 \leq i < M}$. Diverse subnetworks appear as d_i learns to classify y_i from input x_i . At inference, we can ensemble M predictions by feeding the same image M times to the model.

As such, the multi-input multi-output MIMO achieves **ensemble almost “for free”**: all of the layers except the first convolutional and last dense layers are

shared ($\approx +1\%$ #parameters). (Soflaei et al. 2020) motivated a related Aggregated Learning to learn concise representations with arguments from information bottleneck (Tishby 2001). The idea is that over-parameterized CNNs (Frankle and Carbin 2019; Molchanov et al. 2017; Pensia et al. 2020) can fit multiple subnetworks (Veit et al. 2016). Interestingly, MIMO does not need structural differences among subnetworks: they learn to build their own paths while being as diverse as in DE.

3.2.3 Positioning

In this chapter, we formulate a new variant of MSDA that reprises the multi-input multi-output framework by providing separate predictions for each of the original parent samples. By borrowing techniques from mixing data augmentation, we seek to improve the subnetworks in MIMO architectures in order to obtain even better performance. As such, our work directly follows from the seminal MIMO (Havasi et al. 2021) and only loosely borrows ideas from the rest of the ensembling literature.

More precisely, we first develop the MixMo framework by analyzing the MIMO framework through the lens of MSDA. We then develop the MixShare framework to understand and solve issues that emerge when trying to train many subnetworks in a small network with MixMo and MIMO.

3.3 MixMo: Multi-Input Multi-Output MSDA

We first propose **MixMo**, a new generalized multi-input multi-output framework: we train a base network with $M \geq 2$ inputs and outputs as shown on Figure 3.2. This way, we fit M independent subnetworks (Gao et al. 2019; Havasi et al. 2021; Soflaei et al. 2020) defined by an input/output pair and a subset of network weights. This is possible as large networks only leverage a subset of their weights (Frankle and Carbin 2019). Rather than pruning (ie, eliminating) inactive filters (Yann Lecun et al. 1990; H. Li et al. 2017), we seek to fully use the available neurons and over parameterization through multiple subnetworks.

The key divergent point between MixMo variants lies in the **multi-input mixing block** that seeks to combine inputs in a way that favors the emergence of strong and diverse subnetworks. Should the merging be a basic summation or a concatenation, we would recover MIMO (Havasi et al. 2021) or respectively Aggregated Learning (Soflaei et al. 2020) - which both featured this multi-input multi-output strategy.

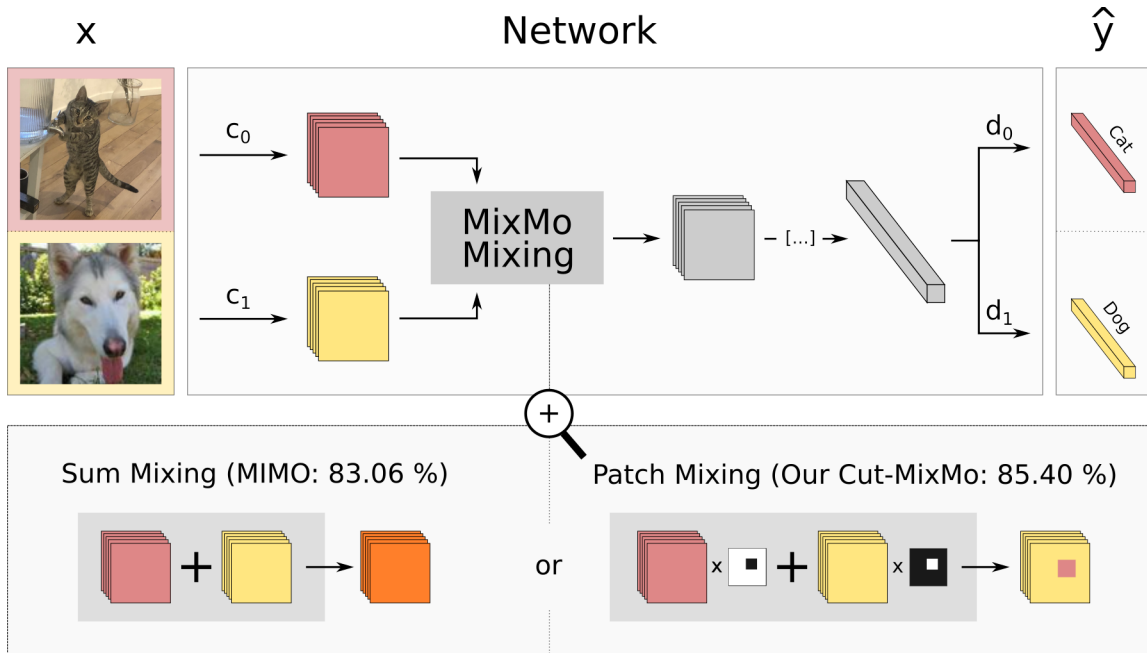


Figure 3.2. – **MixMo overview.** We embed $M = 2$ inputs into a shared space with convolutional layers (c_1, c_2), mix them, pass the embedding through further layers and output 2 predictions via dense layers (d_1, d_2). The key point of our MixMo is the mixing block. Mixing with patches performs better than basic summing: 85.40% vs. 83.06% (MIMO (Havasi et al. 2021)) on CIFAR-100 with WRN-28-10.

Our main intuition is simple: we see summing as a balanced and restrictive form of Mixup (H. Zhang et al. 2018) where $\lambda = \frac{1}{M}$. By analogy, we draw from the considerable MSDA literature to design a more appropriate mixing block. In particular, we leverage binary masking methods to ensure subnetworks diversity. Our framework allows us to create a new Cut-MixMo variant inspired by CutMix (Y. Yang and Soatto 2019), and illustrated in Figure 3.2: a patch of features from the first input is pasted into the features from the second input.

This asymmetrical mixing also raises new questions regarding information flow in the network’s features. We tackle the imbalance between the multiple classification training tasks via a new weighting scheme. Conversely, MixMo’s double nature as a new mixing augmentation in features yields important insights on traditional MSDA.

In summary, our contributions are threefold:

1. We propose a general framework, MixMo, connecting two successful fields: mixing samples data augmentations & multi-input multi-output ensembling.

2. We identify the appropriate mixing block to best tackle the diversity/individual accuracy trade-off in subnetworks: our easy to implement CutMixMo benefits from the synergy between CutMix and ensembling.
3. We design a new weighting of the loss components to properly leverage the asymmetrical inputs mixing.

3.3.1 General overview

We leverage a training classification dataset D of i.i.d. pairs of associated image/label $\{x_i, y_i\}_{i=1}^{|D|}$. We randomly sample a subset of $|B|$ samples $\{x_i, y_i\}_{i \in B}$ that we randomly shuffle via some permutation π . Our training batch therefore is $\{(x_i, x_j), (y_i, y_j)\}_{i \in B, j=\pi(i)}$. The loss $\mathcal{L}_{\text{MixMo}}$ is averaged over these $|B|$ samples: the networks' weights are updated through backpropagation and gradient descent.

Let's focus on the training sample $\{(x_0, x_1), (y_0, y_1)\}$. In MixMo, both inputs are **separately encoded** (see Figure 3.2) into the shared latent space with two different convolutional layers (with 3 input channels each and no bias term): x_0 via c_0 and x_1 via c_1 . To recover a strictly equivalent formulation to MIMO (Havasi et al. 2021), we simply sum the two encodings: $c_0(x_0) + c_1(x_1)$. Indeed, MIMO merges inputs through channel-wise concatenation in pixels: MIMO's first convolutional layer (with 6 input channels and no bias term) hides the summing operation in the output channels.

Explicitly highlighting the underlying mixing leads us to consider a **generalized multi-input mixing block** \mathcal{M} . This manifold mixing presents a unique opportunity to tackle the ensemble diversity/individual accuracy trade-off and to improve overall ensemble results (see Section 3.3.2.1). The shared representation $\mathcal{M}(c_0(x_0), c_1(x_1))$ feeds the next convolutional layers. We note κ the **mixing ratio** between inputs.

The core network \mathcal{C} handles features that represent both inputs simultaneously. The dense layer d_0 predicts $\hat{y}_0 = d_0[\mathcal{C}(\mathcal{M}\{c_0(x_0), c_1(x_1)\})]$ and targets y_0 , while d_1 targets y_1 . Thus, the **training loss** is the sum of two cross-entropies \mathcal{L}_{CE} weighted by parametrized function w_r (defined in Section 3.3.2.2) to balance the asymmetry when $\kappa \neq 0.5$:

$$\mathcal{L}_{\text{MixMo}} = w_r(\kappa)\mathcal{L}_{\text{CE}}(y_0, \hat{y}_0) + w_r(1-\kappa)\mathcal{L}_{\text{CE}}(y_1, \hat{y}_1). \quad (3.1)$$

At inference, the same input x is repeated twice: the core network \mathcal{C} is fed the sum $c_0(x) + c_1(x)$ that preserves maximum information from both encodings. Then, the diverse predictions are averaged: $\frac{1}{2}(\hat{y}_0 + \hat{y}_1)$. This allows us to benefit from ensembling in a single forward pass.

3.3.2 Mixing inputs and balancing concurrent subnetworks

Our MixMo framework relies on two key components: a mixing block that combines the inputs in a manner that allows separate predictions later on, and a balancing mechanism to train strong concurrent subnetworks.

3.3.2.1 Mixing block

The mixing block \mathcal{M} - which combines both inputs into a shared representation - is the cornerstone of MixMo. Our main intuition was to analyze MIMO as a simplified Mixup variant where the mixing ratio κ is fixed to 0.5. Our generalized MixMo framework encompasses a wider range of variants inspired by MSDA mixing methods. Our first main variant - Linear-MixMo - fully extends Mixup. The mixing block is $\mathcal{M}_{\text{Linear-MixMo}}(l_0, l_1) = 2[\kappa l_0 + (1 - \kappa)l_1]$, where $l_0 = c_0(x_0)$, $l_1 = c_1(x_1)$ and $\kappa \sim \text{Beta}(\alpha, \alpha)$ with α the concentration parameter. The second and more effective variant **Cut-MixMo** adapts the patch mixing from CutMix:

$$\mathcal{M}_{\text{Cut-MixMo}}(l_0, l_1) = 2[\mathbf{1}_{\mathcal{M}} \odot l_0 + (\mathbf{1} - \mathbf{1}_{\mathcal{M}}) \odot l_1], \quad (3.2)$$

where $\mathbf{1}_{\mathcal{M}}$ is a binary mask with area ratio $\kappa \sim \text{Beta}(\alpha, \alpha)$, valued at 1 either on a rectangle or on the complementary of a rectangle. In brief, a patch from $c_0(x_0)$ is pasted onto $c_1(x_1)$, or vice versa. This binary mixing in Cut-MixMo advantageously replaces the linear interpolation in MIMO and Linear-MixMo: subnetworks are more accurate and more diverse, as shown empirically in [Figure 3.8](#).

First, binary mixing in \mathcal{M} trains stronger **individual** subnetworks for the same reasons why CutMix improves over Mixup. In a nutshell, linear MSDAs (Verma et al. 2019a; H. Zhang et al. 2018) produce noisy samples (Carratino et al. 2020) that lead to robust representations. As MixMo tends to distribute different inputs on non-overlapping channels (as discussed later in [Figure 3.4a](#)), this regularization hardly takes place anymore in $\mathcal{M}_{\text{Linear-MixMo}}$. On the contrary, by masking features, we simulate common object occlusion problems. This spreads subnetworks' focus across different locations: the two classifiers are forced to find information relevant to their assigned input at disjoint locations. This occlusion remains effective as the receptive field in this first shallow latent space remains small.

Secondly, linear interpolation is fundamentally ill-suited to induce diversity as full information is preserved from both inputs. CutMix on the other hand explicitly increases dataset diversity by presenting patches of images that do not normally appear together. Such benefits can be directly transposed to $\mathcal{M}_{\text{Cut-MixMo}}$: binary mixing with patches increases randomness and **diversity between the subnetworks**. Indeed, in a similar spirit to bagging (Breiman 1996), different

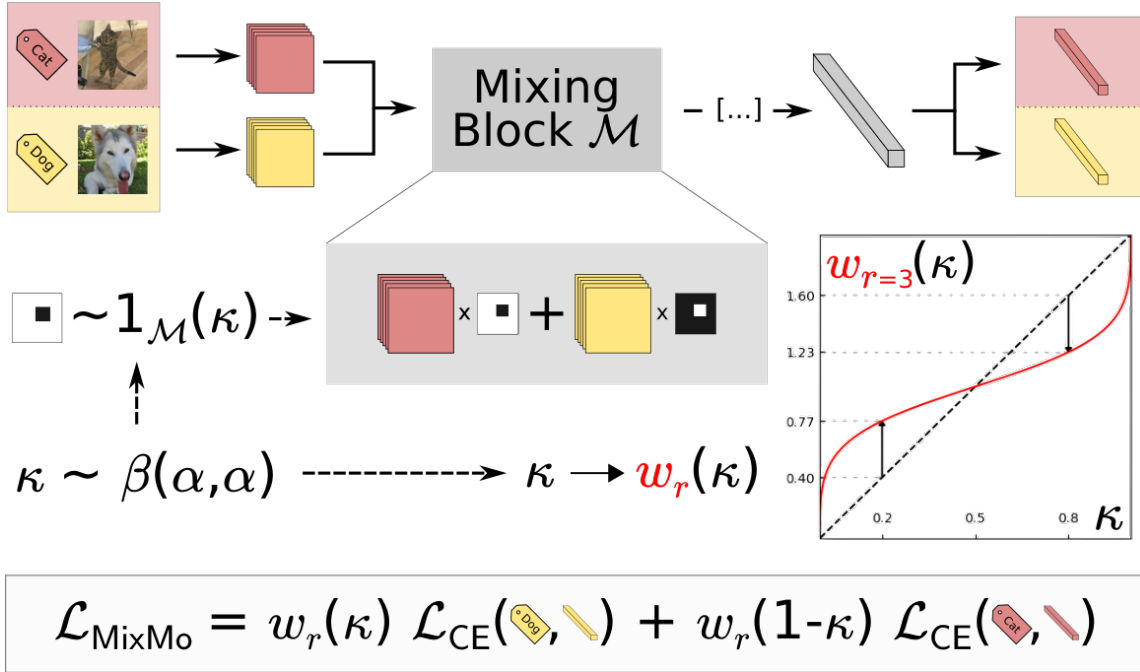


Figure 3.3. – **Overview of Cut-MixMo training.** We sample a mixing mask given κ , and balance the losses with $w_r(\kappa)$ from Equation 3.3.

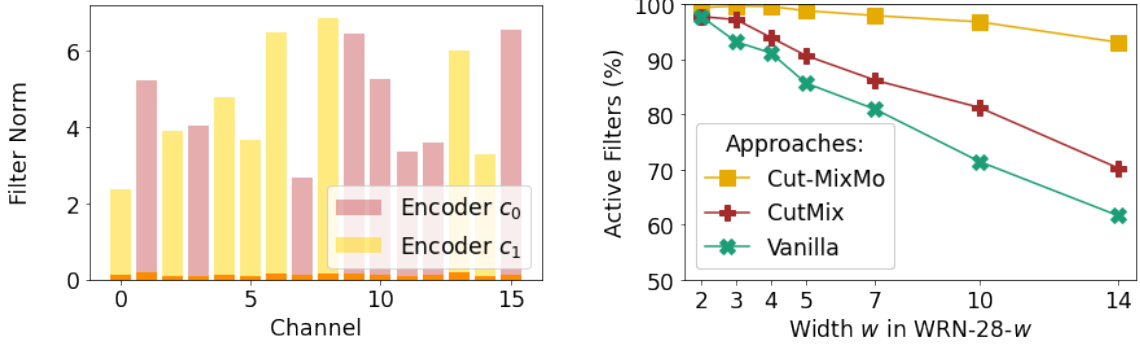
samples are given to the subnetworks. By deleting asymmetrical complementary locations from the two inputs, subnetworks will not rely on the same region and information. Overall, they are less likely to collapse on close solutions.

3.3.2.2 Loss weighting w_r

Asymmetries in the mixing mechanism can cause one input to overshadow the other. Notably when $\kappa \neq 0.5$, the predominant input may be easier to predict. We seek a weighting function w_r to **balance the relative importance** of the two \mathcal{L}_{CE} in $\mathcal{L}_{\text{MixMo}}$. This weighting modifies the effective learning rate, how gradients flow in the network and overall how mixed information is represented in features. In this paper, we propose to weight via the parametrized:

$$w_r(\kappa) = 2 \frac{\kappa^{1/r}}{\kappa^{1/r} + (1-\kappa)^{1/r}}. \quad (3.3)$$

This defines a family of functions indexed by the parameter r , visualized for $r = 3$ in red on Figure 3.3. See Appendix Section A.5.3 for complementary visualizations. This power law provides a natural relaxation between **two extreme configurations**. The first extreme, $r = 1$, $w_1(\kappa) = 2\kappa$, is in line with linear label interpolation in MSDA. The resulting imbalance in each subnetwork’s contribution to $\mathcal{L}_{\text{MixMo}}$ causes lopsided updates. While it promotes diversity, it also reduces



(a) Filters l_1 -norms of the input encoders c_0 and c_1 . (b) Proportion of active filters in the core network vs. width w .

Figure 3.4. – **Influence of MixMo on network utilization.**(a) The encoders have separate channels: the two subsequent classifiers can differentiate the two inputs. (b) Less filters are strongly active ($\|f_i\|_1 \geq 0.4 \times \max_{f \in \text{layer}} \|f\|_1$) in wider networks: Cut-MixMo reduces this negative point.

regularization: the overshadowed input has a reduced impact on the loss. The opposite extreme, $r \rightarrow \infty$, $w_\infty(\kappa) \rightarrow 1$, removes reweighting. Consequently, w_r inflates the importance of hard under-represented inputs, *à la* Focal Loss (Lin et al. 2017). However, minimizing the role of the predominant inputs destabilizes training. Overall, we empirically observe that moderate values of r perform best as they trade off pros and cons from both extremes.

Interestingly, the proper weighting of loss components is also a central theme in **multi-task learning** (Caruana 1997; Z. Chen et al. 2018). While it aims at predicting several tasks from a shared input, MixMo predicts a shared task from several different inputs. Beyond this inverted structure, we have similar issues: *e.g.* gradients for one task can be detrimental to another conflicting task. Fortunately, MixMo presents an advantage: the exact ratios κ and $1 - \kappa$ of each task are known exactly.

3.3.2.3 Generalization to $M \geq 2$ subnetworks

MixMo is easily extended by optimizing $\mathcal{L}_{\text{MixMo}} = \sum_{0 \leq i < M} M \frac{\kappa_i^{1/r}}{\sum_j \kappa_j^{1/r}} \mathcal{L}_{\text{CE}}(y_i, \hat{y}_i)$ with $\{\kappa_i\} \sim \text{Dir}(\alpha)$ from a Dirichlet distribution (see Appendix Section A.5.2). The key change is that \mathcal{M} now needs to handle more than 2 inputs: $\{c_i(x_i)\}_{0 \leq i < M}$. While linear interpolation is easily generalized, Cut-MixMo has several possible extensions: in our experiments, we first linearly interpolate between $M - 1$ inputs and then patch in a region from the M -th.

3.3.3 From manifold mixing to MixMo

We have discussed at length how we extend multi-input multi-output frameworks by borrowing mixing protocols from MSDA. Now we reversely point out how our **MixMo diverges from MSDA schemes**. At first glimpse, the idea is the same as manifold mixing (Faramarzi et al. 2020; B. Li et al. 2021; Verma et al. 2019a): $M = 2$ inputs are encoded into a latent space to be mixed before being fed to the rest of the network. Yet, while they mix at varying depths, we only mix in the shallowest space. Specifically, we only mix in features - and not in pixels - to allow separate encodings of the inputs: they need to remain distinct in the mixed representation for the subsequent classifiers.

Hence our two key differences: *first*, MixMo uses two separated encoders (one for each input), and *second*, it outputs two predictions instead of a single one. Indeed, MSDAs use a single classifier that targets a unique soft label reflecting the different classes via linear interpolation. MixMo instead chooses to fully leverage the composite nature of mixed samples and trains **separated dense layers**, d_0 and d_1 , ensembled “for free” at test time.

Section 3.3.5.4 demonstrates that MixMo works because it also uses two **different encoders** c_0 and c_1 . While training two classifiers may seem straightforward in MSDA, it actually raises a troubling question: which input should each classifier predicts? Having two encoders provides a simple solution: the network is divided in two subnetworks, one for each input. Their separability is easily observed: Figure 3.4a shows the l_1 -norm of the 16 filters for the two encoders (WRN-28-10 on CIFAR-100). Each filter norm is far from zero in only one of the two encoders: $c_0(x_0)$ and $c_1(x_1)$ separate the inputs in different dimensions which allows subsequent layers to treat them differently.

This leads MixMo to use most available filters. Following the structured pruning literature (H. Li et al. 2017), we consider in Figure 3.4b that a filter (in a layer of the core network) is active if its l_1 -norm is at least 40% of the l_1 -norm from its layer’s most active filter (see Appendix Section A.5.4). This illustrates the known increase in sparsity in wider networks. Conversely, having 2 subnetworks in MixMo enables the weights ignored by one subnetwork to be leveraged by the other.

3.3.4 Main experimental results

We evaluate MixMo efficiency on standard image classification datasets: CIFAR-10,100 (Krizhevsky and Hinton 2009) and Tiny ImageNet (Chrabaszcz et al. 2017). We equally track accuracies (Top{1,5}, \uparrow) and the *calibrated Negative Log-Likelihood*

($\text{NLL}_{c_t} \downarrow$). Indeed, (Ashukha et al. 2020) shows that we should compare in-domain uncertainty estimations after temperature scaling (TS) (Guo et al. 2017): we thus split the test set in two and calibrate (after averaging in ensembles) with the temperature optimized on the other half, as in (Lobacheva et al. 2020; Rame and Cord 2021). We nonetheless report NLL (without TS) along with the Expected Calibration Error (Naeini et al. 2015) in Appendix Section A.1.3.2.

3.3.4.1 Implementation details

We mostly study the Linear-MixMo and Cut-MixMo variants with $M=2$. We set **hyper-parameter** $r=3$ (see Section 3.3.5.3). $\alpha=2$ performs better than 1 (see Appendix Section A.5.7). In contrast, MIMO (Havasi et al. 2021) refers to linear summing, like Linear-MixMo, but with $\kappa=0.5$ instead of $\kappa \sim \text{Beta}(\alpha, \alpha)$.

Different mixing methods create a strong **train-test distribution gap** (Carratino et al. 2020; Gontijo-Lopes et al. 2021). Thus, in Cut-MixMo we actually substitute $\mathcal{M}_{\text{Cut-MixMo}}$ for $\mathcal{M}_{\text{Linear-MixMo}}$ with probability $1 - p$ to accommodate for the summing in \mathcal{M} at inference. We set the probability of patch mixing during training to $p=0.5$, with linear descent to 0 over the last twelfth of training epochs (see pseudocode in Appendix Algorithm A.6).

When combined with CutMix, the pixels inputs are: $(m_x(x_i, x_k, \lambda), m_x(x_j, x_{k'}, \lambda'))$ with interpolated targets $(\lambda y_i + (1 - \lambda)y_k, \lambda' y_j + (1 - \lambda')y_{k'})$, where k, k' are randomly sampled and $\lambda, \lambda' \sim \text{Beta}(1, 1)$.

MIMO duplicates samples b times via **batch repetition**: x_i will be associated with $x_{\pi(i)}$ and $x_{\pi'(i)}$ in the same batch if $b=2$. As the batch size remains fixed, the count of unique samples per batch and the learning rate is divided by b . Conversely, the number of steps is multiplied by b . Overall, this stabilizes training but multiplies its cost by b . We thus indicate an estimated (training/inference) overhead (wrt. vanilla training) in the *time* column of our tables. Note that some concurrent approaches also lengthen training: e.g. GradAug (T. Yang et al. 2020) via multiple subnetworks predictions ($\approx \times 3$).

We provide more details in Appendix Section A.1.3.1 and provide our open source PyTorch (Paszke et al. 2019) implementation.

3.3.4.2 Main results on CIFAR-100 and CIFAR-10

Table 3.1 reports averaged scores over 3 runs for our main experiment on CIFAR with WRN-28-10 (Zagoruyko and Komodakis 2016). We re-use the hyperparameters given in MIMO (Havasi et al. 2021). Cut-MixMo reaches (85.40% Top1, 0.535 NLL_c) on CIFAR-100 with $b=4$: it surpasses our Linear-MixMo (83.08%, 0.656) and MIMO (83.06%, 0.661). Cut-MixMo sets a new state of the art when

Dataset	CIFAR-100			CIFAR-10		
Approach	Time Tr./Inf.	Top1 %, \uparrow	Top5 %, \uparrow	NLL _c 10 ⁻² , \downarrow	Top1 %, \uparrow	NLL _c 10 ⁻² , \downarrow
Vanilla		81.63	95.49	73.9	96.34	12.6
Mixup		83.44	95.92	65.7	97.07	11.2
Manifold Mixup [†]	1/1	81.96	95.51	73.4	97.45	12.2
CutMix		84.05	96.09	64.8	97.23	9.9
ResizeMix [†]		84.31	-	-	97.60	-
Puzzle-Mix [†]	2/1	84.31	96.46	66.8	-	-
GradAug [†] + CutMix [†]	3/1	84.14	96.43	-	-	-
		85.51	96.86	-	-	-
Mixup BA [†]	7/1	84.30	-	-	97.80	-
DE (2 Nets) + CutMix	2/2	83.17	96.37	66.4	96.67	11.1
		85.74	96.82	57.1	97.52	8.6
MIMO	2/1	82.40	95.78	68.8	96.38	12.1
Linear-MixMo + CutMix		82.54	95.99	67.6	96.56	11.4
		84.69	97.12	57.2	97.32	9.4
Cut-MixMo + CutMix		84.38	96.94	56.3	97.31	8.9
		85.18	97.20	54.5	97.45	8.4
MIMO	4/1	83.06	96.23	66.1	96.74	11.4
Linear-MixMo + CutMix		83.08	96.26	65.6	96.91	10.8
		85.47	97.04	55.8	97.68	8.7
Cut-MixMo + CutMix		85.40	97.22	53.5	97.51	8.1
		85.77	97.42	52.4	97.73	7.9

Table 3.1. – **Main results:** WRN-28-10 on CIFAR. **Bold** highlights best scores, [†] marks approaches not re-implemented.

combined with CutMix (85.77%, 0.524). Results remain strong when $b=2$: Cut-MixMo (84.38%, 0.563) proves better on its own than traditional DE (Lakshminarayanan et al. 2017), and MSDAs like MixUps (H. Zhang et al. 2018; Verma et al. 2019a) or the stronger CutMix variant (Y. Yang and Soatto 2019). On CIFAR-10, we see similar trends: Cut-MixMo reaches 0.081 in NLL_c, 0.079 with CutMix. Yet, the costlier batch augmented Mixup BA (Hoffer et al. 2020) edges it out in Top1.

Figure 3.5 shows how MixMo grows stronger than DE (green curves) as width w in WRN-28- w increases. The parameterization becomes appropriate at $w=4$: Cut-MixMo (yellow curves) then matches DE - with half the parameters - in Figure 3.5a

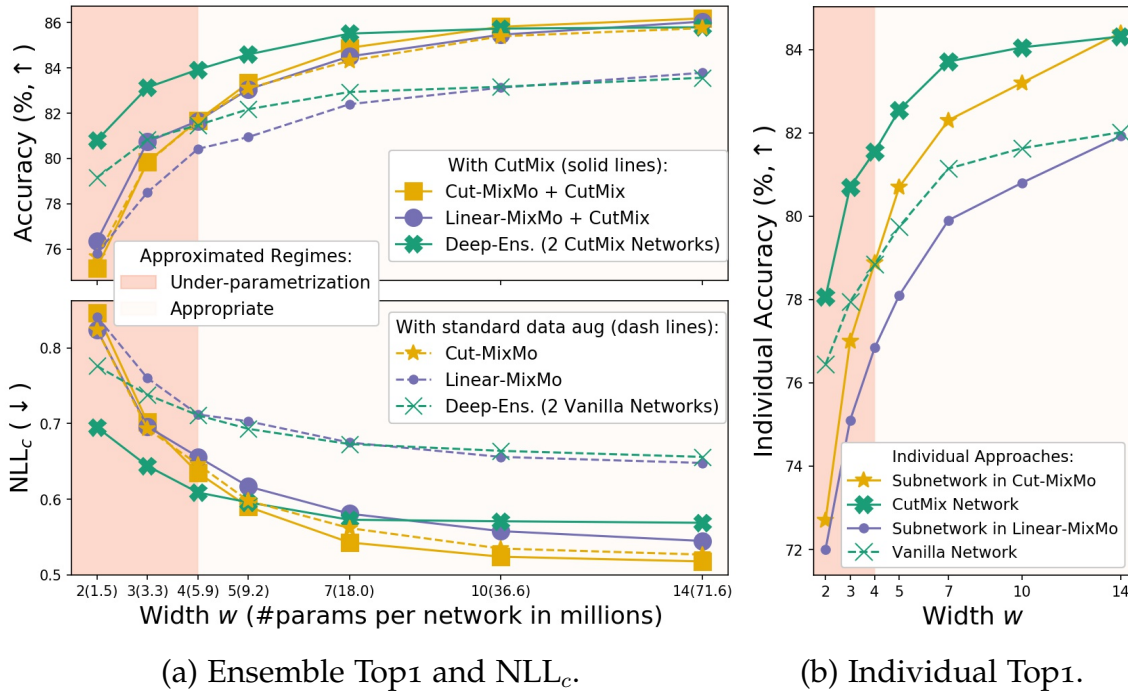


Figure 3.5. – **Parameter efficiency of MixMo** (metrics/#params). CIFAR-100 with WRN-28- w , $b=4$. Comparisons between (a) ensemble and some of their (b) individual counterparts.

and its subnetworks match a vanilla network in Figure 3.5b. Beyond, MixMo better uses over-parameterization: Cut-MixMo+CutMix surpasses DE+CutMix in NLL_c for $w \geq 5$, and this is true in Top1 for $w \geq 10$.

Compared to our strong Linear-MixMo+CutMix (purple curves), Cut-MixMo performs similarly in Top1, and better with CutMix for $w \geq 4$. While Linear-MixMo and DE learn from occlusion, Cut-MixMo also benefits from CutMix, notably from the induced label smoothing. Overall, Cut-MixMo, even without CutMix, significantly better estimates uncertainty.

3.3.4.3 Robustness to image corruptions

Deep networks' results decrease when facing unfamiliar samples. To measure robustness to train-test distribution gaps, (Hendrycks and T. Dietterich 2019) corrupted CIFAR-100 test images into CIFAR-100-c (more details in Appendix Section A.1.3.1). As in Puzzle-Mix (J.-H. Kim et al. 2020), we report WRN-28-10 results with and without AugMix (Hendrycks et al. 2020), a pixels data augmentation technique specifically introduced for this task. Table 3.2 shows that Cut-MixMo ($b=4$) best complements AugMix and reaches 71.1% Top1.

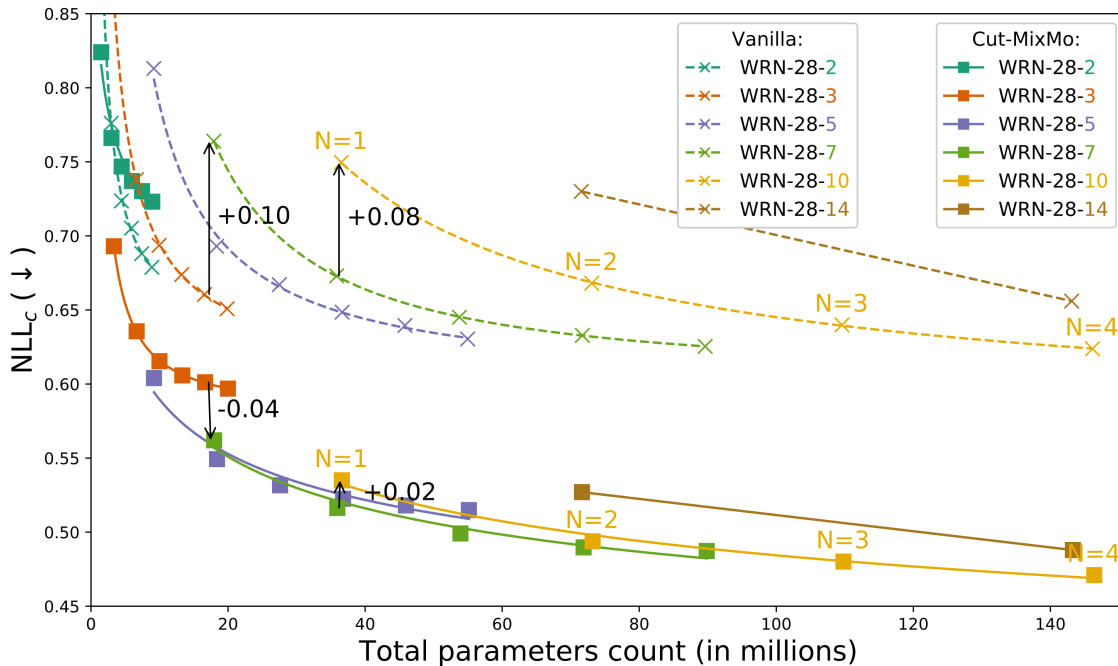


Figure 3.6. – **Ensemble effectiveness** ($NLL_c/\#\text{params}$), for different widths w in WRN-28- w and numbers of members N . Standard data augmentations on CIFAR-100 with $b = 4$. Curves interpolated through power laws (Lobacheva et al. 2020).

3.3.4.4 Ensemble of MixMo

Since MixMo adds very little parameters ($\approx +1\%$), we can combine independently trained MixMo like in DE. This ensembling of ensemble of subnetworks leads in practice to the averaging of $M \times N = 2 \times N$ predictions. Figure 3.6 compares ensembling for vanilla networks and Cut-MixMo on CIFAR-100. We first recover the Memory Split Advantage (Chirkova et al. 2020; Lobacheva et al. 2020) (MSA): at similar parameter counts, $N=5$ vanilla WRN-28-3 do better than a single vanilla WRN-28-7 (+0.10 in NLL_c). **Cut-MixMo challenges this MSA:** we bridge the gap between using one network or several smaller networks (-0.04 on same setup). Visually, Cut-MixMo’s curves remain closer to the lower envelope: performances are less dependent on how the memory budget is split. This is be-

Approach AugMix	1 Net.		CutMix	Puzzle-Mix [†]		DE (2 Nets)		MIMO	Linear-MixMo		Cut-MixMo	
	-	✓		-	✓	-	✓		-	✓	-	✓
Top1 ↑	52.2	67.8	51.93	58.09	70.46	53.8	69.9	53.6	55.6	70.4	57.0	71.1
Top5 ↑	73.7	87.5	72.03	77.3	87.7	74.9	88.9	74.9	76.1	89.4	77.4	89.5
NLL ↓	2.50	1.38	2.13	1.96	1.34	2.27	1.24	2.66	2.33	1.22	2.04	1.16

Table 3.2. – **Robustness comparison on CIFAR-100-c.**

cause Cut-MixMo is effective mainly for larger architectures by better leveraging their parameters.

We also recover that wide vanilla networks tend to be less diverse (Neal et al. 2018), and thus gain less from ensembling (Lobacheva et al. 2020): $N=2$ vanilla WRN-28-14 (83.47% Top1, 0.656 NLL_c) perform not much better than $N=2$ WRN-28-7 (82.94%, 0.673). Contrarily, **Cut-MixMo facilitates the ensembling of large networks** with (86.58%, 0.488) vs. (85.50%, 0.516) (more comparisons in Appendix Section A.5.5).

3.3.5 MixMo analysis on CIFAR-100 w/ WRN-28-10

3.3.5.1 Training time

We have just seen that CutMix improves Linear-MixMo at varying widths w , but not enough to match Cut-MixMo in NLL_c : CutMix can not fully compensate for the advantages from patch mixing over linear interpolation. We recover this finding in Figure 3.7, this time at varying batch repetition $b \in \{1, 2, 4\}$ when $w=10$. Moreover, Cut-MixMo outperforms DE for the **same training time**. Indeed, MixMo variants trained with a given b matches the training time of DE with $N=b$ networks. In the rest of this section, we set $b=2$.

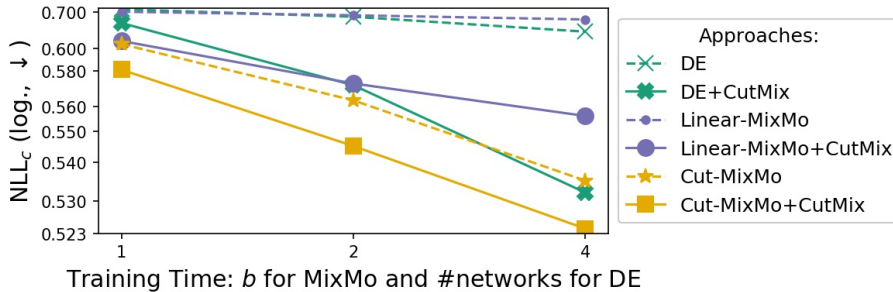


Figure 3.7. – $NLL_c(\downarrow)$ improves with longer training, via batch repetitions (MixMo) or additional networks (DE).

3.3.5.2 The mixing block

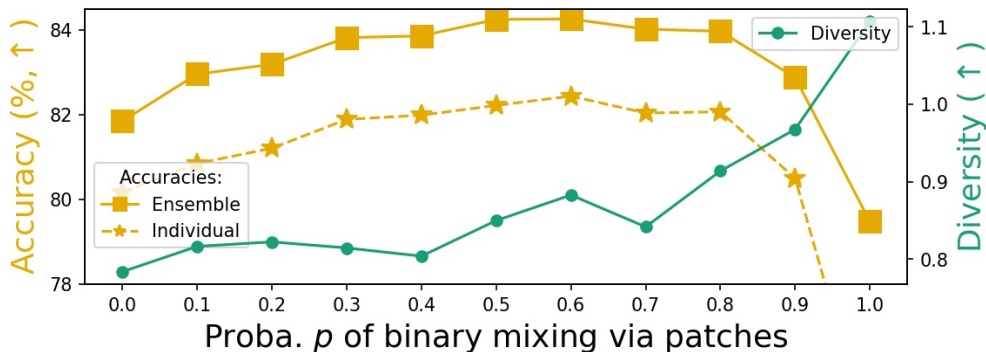
Table 3.3 compares performance for several mixing blocks (Faramarzi et al. 2020; Harris et al. 2020; Summers and Dinneen 2019; Y. Yang and Soatto 2019). No matter the **shape** (illustrated in Appendix Section A.5.1), binary masks perform better than linear mixing: the cow-spotted mask (84.17%, 0.561) (G. French et al. 2020a; G. French et al. 2020b) notably performs well. The basic CutMix patching (84.38%, 0.563) is nevertheless more accurate and was our main focus.

<i>Mapproach</i>	MixUp	Hor. Concat.	Vert. Concat.	PatchUp 2D	FMix	CowMask	CutMix
Top1 \uparrow	82.5	82.78	84.00	84.16	83.76	84.17	84.38
NLL _c \downarrow	0.676	0.627	0.573	0.581	0.602	0.561	0.563

Table 3.3. – Influence of the MSDA used in the mixing block \mathcal{M} .

We further study the impact of patch mixing through the lens of the *ensemble diversity/individual accuracy trade off*. As in (Rame and Cord 2021), we measure diversity via the pairwise ratio-error (Aksela 2003) (d_{re} , \uparrow), defined as the ratio between the number of different errors and simultaneous errors for two predictors. In Figure 3.8 and Figure 3.9, we average metrics over the last 10 epochs.

As argued in Section 3.3.2.1, patch mixing increases diversity compared to linear mixing in Figure 3.8. As the probability p of patch mixing grows, so does diversity: from $d_{re}(p=0.0)\approx 0.78$ (Linear-MixMo) to $d_{re}(p=0.5)\approx 0.85$ (Cut-MixMo). In contrast, DE has $d_{re}\approx 0.76$ while MIMO has $d_{re}\approx 0.77$ on the same setup. Increasing p past 0.6 boosts diversity even more at the cost of subnetworks’ accuracies: this is due to underfitting and an increased test-train distribution gap. $p \in [0.5, 0.6]$ is thus the best trade off.

Figure 3.8. – Diversity/accuracy as function of p with $r = 3$.

3.3.5.3 Weighting function w_r

We analyze the impact of the parameter r in the reweighting function w_r . Higher values tend to remove reweighting, as shown in Appendix Section A.5.3: they strongly decrease diversity in Figure 3.9. The opposite extreme with $r=1$ increases diversity via lopsided gradient updates but it degrades accuracy. We speculate it under-emphasizes hard samples. The range $r \in [3, 6]$ strikes a good balance: results remain high and stable.

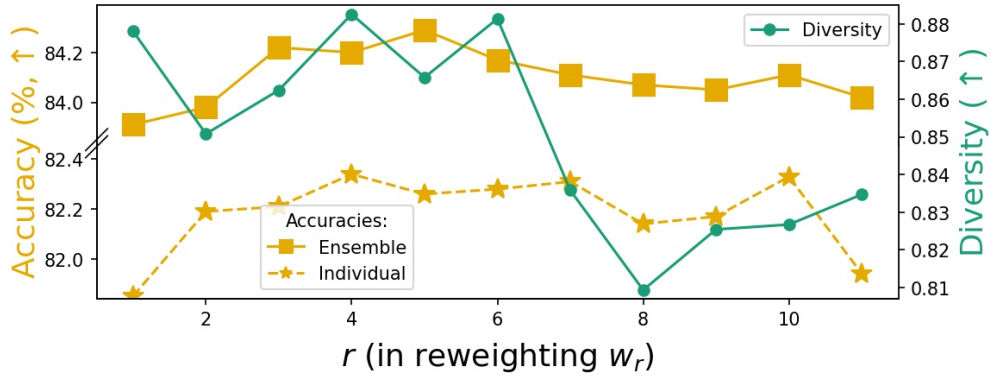


Figure 3.9. – Diversity/accuracy as function of r with $p = 0.5$.

3.3.5.4 Multiple encoders and classifiers

In Section 3.3.3, we compared MixMo and MSDA. Table 3.4 confirms the need for **2 encoders and 2 classifiers**. With 1 classifier and linearly interpolated labels (in the same spirit as (J. Chen et al. 2020)), the 2 encoders perform worse than 1 encoder. With 1 shared encoder and 2 classifiers, it is not clear which input each classifier should target. In the first naive \ominus , we randomly associate the 2 classifiers and the 2 inputs (encoded with the same encoder). This \ominus variant yields poor results. In \otimes , the first classifier tries to predict the label from the predominant input, the second targets the other input: \otimes reaches 0.598 vs. 0.563 for Cut-MixMo.

# enc.	# clas.	nll _c ↓
1	1	0.604
2	1	0.666
1	2^{\ominus}	0.687
1	2^{\otimes}	0.598
2	2	0.563

Table 3.4. – Ablation on the importance of the encoder/decoders in MixMo.

3.3.5.5 Generalization to $M \geq 2$ subnetworks

We try to generalize MixMo to more than $M = 2$ subnetworks in Figure 3.10. Cut-MixMo’s subnetworks perform at 82.3% when $M=2$ vs. 79.5% when $M=3$. In MIMO, it’s 79.8% vs. 77.7%. Because subnetworks do not share features, higher M degrades their results: only two can fit seamlessly. Ensemble Top1 overall decreases in spite of the additional predictions, as already noticed in MIMO (Havasi et al. 2021).

This reflects MixMo’s strength in over-parametrized regimes, but also its limitations with fewer parameters when subnetworks underfit (recall previous Figure 3.5). Facing similar findings, MIMO (Havasi et al. 2021) introduced input repetition so that subnetworks share their features, at the cost of drastically reduc-

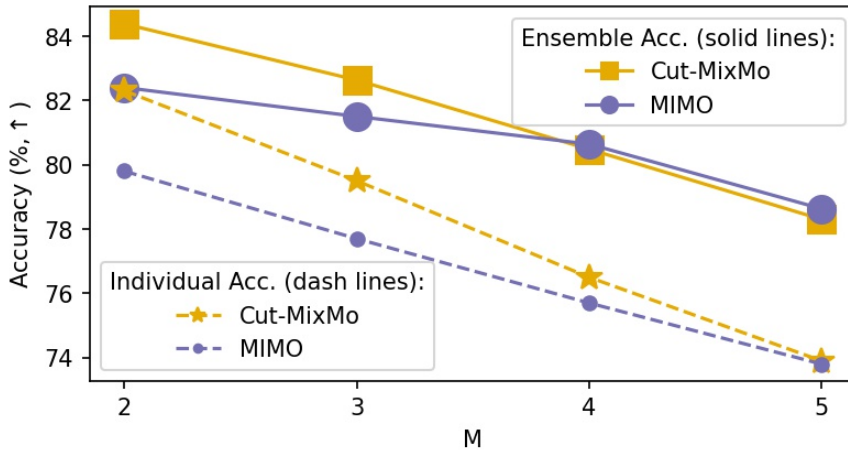


Figure 3.10. – **Ensemble/individual** accuracies for $M \geq 2$.

ing diversity. Our generalization may be extended by future approaches whose mixing blocks (perhaps not inspired by MSDA) would tackle these issues.

3.3.6 Pushing MixMo further: Tiny ImageNet

At a larger scale and with more varied 64×64 images, Cut-MixMo reaches a new state of the art of 70.24% on Tiny ImageNet (Chrabaszcz et al. 2017) in Table 3.5. We re-use the hyper-parameters given in previous state of the art Puzzle-Mix (J.-H. Kim et al. 2020). With $w=1$, PreActResNet-18 (He et al. 2016b) is not sufficiently parametrized for MixMo’s advantages to express themselves on this challenging dataset. MixMo’s full potential shines with wider networks: with $w=2$ and 44.9M parameters, Cut-MixMo reaches (69.13%, 1.28) vs. (67.76%, 1.33) for CutMix. Compared to DE with 3 networks, Cut-MixMo performs {worse, similarly, better} for width $w \in \{1, 2, 3\}$. At (almost) the same numbers of parameters, Cut-MixMo when $w=2$ performs better (69.13%, 1.28) than DE with 4 networks when $w=1$ (67.51%, 1.31).

3.3.7 Takeaways from the MixMo project

With MixMo, we have shown MIMO architectures can in fact be interpreted as a variation on mixing sample data augmentations. Under this particular angle, the seminal formulation of the framework is fairly sub-optimal: summing the encoded inputs is a poor compression process that obfuscates them both. Rather, MIMO methods should rely on binary mixing methods that helps diversify the subnetwork predictions like CutMix.

Width w (# params)		$w = 1$ (11.2M)		$w = 2$ (44.9M)		$w = 3$ (100.5M)	
Approach	Time	Top1	NLL _c	Top1	NLL _c	Top1	NLL _c
	Tr./Inf.	%, \uparrow	\downarrow	%, \uparrow	\downarrow	%, \uparrow	\downarrow
Vanilla		62.56	1.53	64.80	1.51	65.78	1.53
Mixup		63.74	1.62	66.62	1.50	67.27	1.51
Manifold Mixup [†]	1/1	58.70	1.92	-	-	-	-
Co-Mixup [†]		64.15	-	-	-	-	-
CutMix		65.09	1.58	67.76	1.33	68.95	1.29
Puzzle-Mix [†]	2/1	64.48	1.65	-	-	-	-
DE (2 Nets)	2/2	65.53	1.39	68.06	1.37	68.38	1.36
DE (3 Nets)	3/3	66.76	1.34	69.05	1.29	69.36	1.28
DE (4 Nets)	4/4	67.51	1.31	69.94	1.24	69.72	1.26
Linear-MixMo		61.58	1.61	66.62	1.41	68.18	1.36
Cut-MixMo	2/1	63.78	1.48	68.30	1.30	69.89	1.26
Linear-MixMo		62.91	1.51	67.03	1.41	68.38	1.38
Cut-MixMo	4/1	64.44	1.48	69.13	1.28	70.24	1.19

Table 3.5. – Results on Tiny ImageNet (PreActResNet-18- w).

Basing a MIMO architecture on CutMix - along with a few key additions like reintroducing MSDA style ponderation on the subnetwork training losses - leads to a much improved performance over the CIFAR datasets on large enough WideResNets. Importantly, MIMO architectures prove orthogonal to standard MSDA techniques as we also see significant improvements when the model inputs are mixed inputs themselves. While we did not combine MixMo with the in-class MSDA scheme developed in Chapter 2, the framework should prove orthogonal to in-class mixing on the inputs. It must be noted however that in-class mixing cannot be used to mix the inputs in the mixing block \mathcal{M} as we need to mix semantic content from both inputs. MixMo also performs well on more complex settings like the corrupted CIFAR-100-C or the more complex TinyImageNet.

These last experiments on TinyImageNet (Section 3.3.6) however shine a light on the main issue with MIMO models: subnetworks can only be trained efficiently if the base model is wide enough. Indeed, we need to triple the width of ResNet-18 models before we see solid gains from using MixMo. Worse, MixMo performs worse than a single-input single-output model on TinyImageNet with a standard width ResNet-18. This same problem also appears on CIFAR-100 with narrower models.

As such, it seems of paramount importance to confirm the root cause of this shortcoming and identify strategies to address it. As it stands, MIMO architectures must remain limited to very wide architectures which restricts their applicability. Moreover, it seems likely that the difficulties observed in [Section 3.3.5.5](#) and (Havasi et al. 2021) when trying to learn more than 2 subnetworks are linked to this width issue.

3.4 MixShare: Feature sharing between MIMO subnetworks

Our MixMo framework (Rame et al. 2021) (see [Section 3.3](#)) improves the performance of MIMO (Havasi et al. 2021) architectures by treating the whole model as a mixed sample with separate predictions for each of the original inputs. Our work however highlights significant limitations of MIMO architectures: multi-input multi-output architectures require large base models and struggle to fit more than 2 subnetworks. Indeed, [Section 3.3.4.4](#) shows a significant drop in performance on CIFAR 100 when going from 2 subnetworks to 4 subnetworks.

This scaling issue is explained by analyzing the features inside the network, as we show at the beginning of this paper by extending our previous study of subnetwork behavior. Our analysis shows that the aforementioned scaling issues stem from how subnetworks share no features in the base network: each channel or feature is almost exclusively used by one subnetwork. As such, we can explain the scaling issue since each additional subnetwork significantly reduces the effective size of the individual subnetworks. Beyond causing issues on smaller architectures or harder datasets, this leads to very wasteful use of network parameters. This is especially unfortunate as the subnetworks could at the very least share generic features in the first layers. We see this as a missed opportunity, one that can significantly improve multi-input multi-output models' applicability to real world settings like mobile devices.

We therefore develop the MixShare framework along with the following contributions:

- We carefully study the repartition of features in MIMO subnetworks.
- We develop the unmixing mechanism to allow the final network features to describe each of the original inputs.
- We discuss necessary adjustments to obtain good performance with unmixing. In particular, we show particular care must be given to progressively

weakening the unmixing process over training and to properly initializing input encoders.

3.4.1 MIMO Subnetworks do not share features

In this section, we strive to pinpoint the cause of multi-input multi-output architectures’ scaling issues. To this end, we consider the following question: how do subnetworks behave in multi-input multi-output architectures ?

Following [Section 3.3.3](#), we check how the inputs are organized in the C feature maps of the mixing space by considering the L_1 norm of the C encoder kernels for each subnetwork (see [Figure 3.11](#)). This tells us whether a feature map contains more information about one input, and we can visualize which maps are used by which subnetwork through histograms h_0 and h_1 of feature influence for each subnetwork. Quantitatively, we can approximate the feature sharing rate through the ratio of $\frac{\min(h_0, h_1)}{\max(h_0, h_1)}$. In the same spirit, we consider the L_1 norm of the columns of classifier weight matrices to quantify the importance of each feature to each classifier.

We conduct our study on a WideResNet-28-2 ([Zagoruyko and Komodakis 2016](#)) using the more realistic batch repetition 2 setting from ([Rame et al. 2021](#)) on the CIFAR 100 dataset ([Krizhevsky and Hinton 2009](#)) (see [Appendix](#)). We choose to consider this situation as it perfectly showcases the issues encountered by MIMO methods on smaller architectures. To complement this, we also show results on the slightly larger WideResNet-28-5 later on.

[Figure 3.12](#) shows the subnetworks are fully independent in the core network: each channel in the input block encodes information about only one input, as the corresponding kernel of the other encoder’s L_1 is very low. A similar behavior is observed in the output block, and further analysis of input influence on intermediary feature maps shows this behavior remains consistent within the network (See [Appendix](#)).

Multi-input multi-output architectures’ scaling issues become much easier to understand in light of this: the amount of weights available to each of the underlying subnetworks decreases quadratically with the number of subnetwork. Indeed, since feature maps of different subnetworks cannot communicate, only $\frac{1}{M}$ weights can be non-zero. This fraction of non-zero weights must then be distributed between the M subnetworks. Furthermore, the subnetworks likely extract similar generic features, at least in the first layers. Since the subnetworks share no features, this means those features are unnecessarily replicated for each subnetwork.

This is not wholly surprising or undesirable behavior as MIMO strives to train M independent subnetworks to obtain diverse ensembles. By avoiding overlap

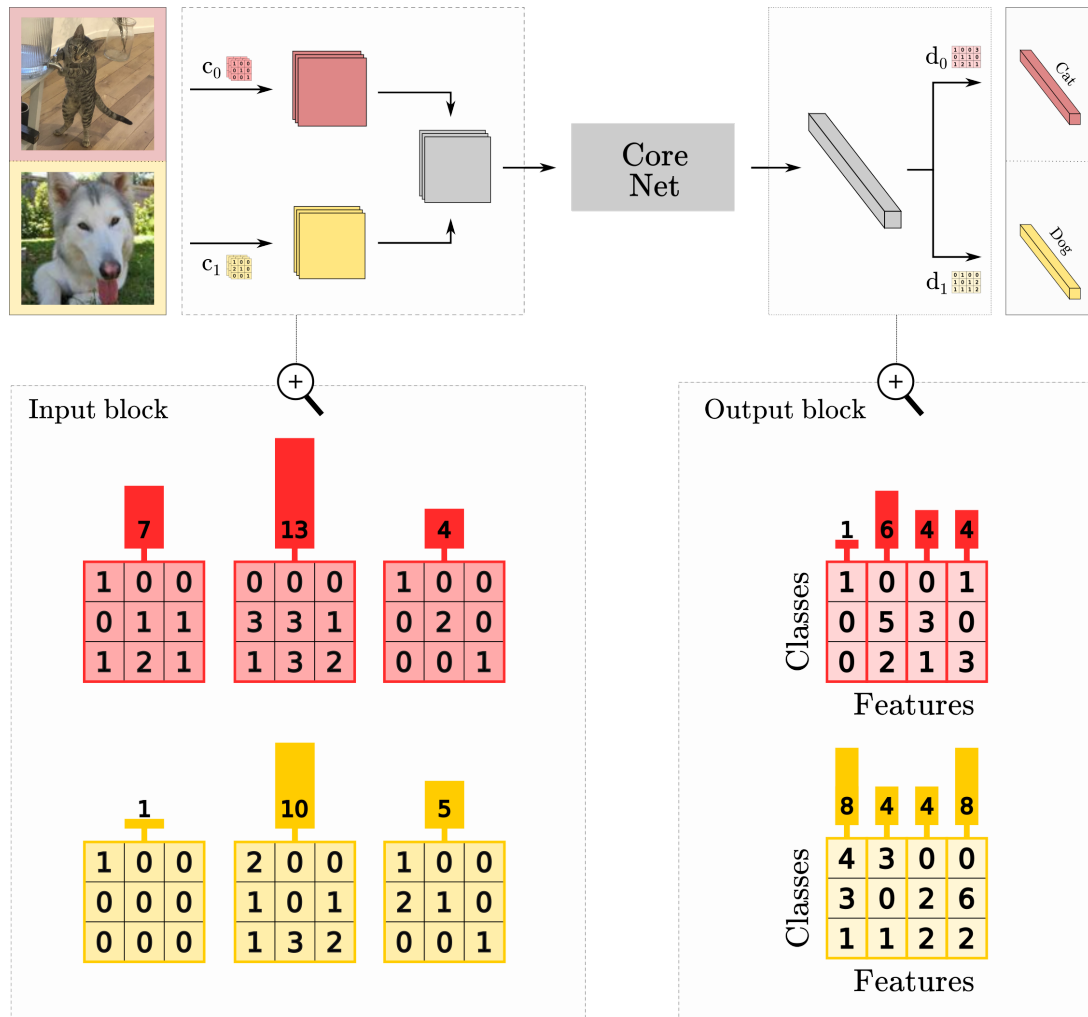


Figure 3.11. – **Influence of the feature maps for the subnetworks.** We study the influence of features in the input and output block on the subnetworks (L_1 norm figured by bars). For the input block, we consider the L_1 norm of the feature kernels for the relevant encoders. For the output block, we look at the L_1 norm of columns in the classifier weights matrices. On the figure, this shows us the first input block kernel is mostly used by the first (red) subnetwork (7 vs. 1). Similarly, the first feature of the output block proves important only to the first (red) classifier.

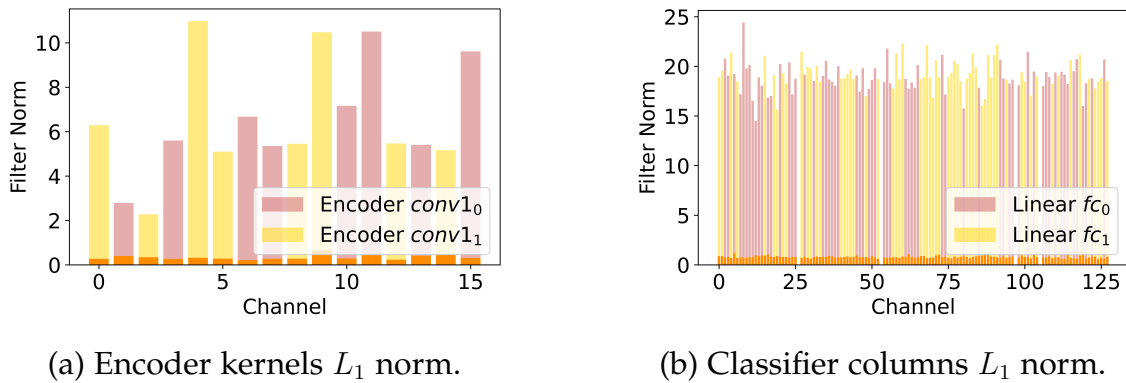


Figure 3.12. – **Usage of features by MixMo subnetworks.** Features are used by one subnetwork or the other, never both at the same time: the overlap (orange) is very low.

between subnetworks, the subnetworks act as a standard ensemble of smaller models, with the base model size acting as hard cap on the number of the parameters used by the ensemble.

While it is true not sharing any features ensures subnetworks’ independence, it seems unnecessary. Indeed, the subnetworks are highly unlikely to extract completely different features. As such, subnetworks should benefit from sharing features at least in the early layers even if the classifier still consider fairly different features.

At first blush, nothing in the MIMO training protocol explicitly requires the subnetworks not share any features. Why do the subnetworks avoid sharing features? How could we encourage them to share some parameters?

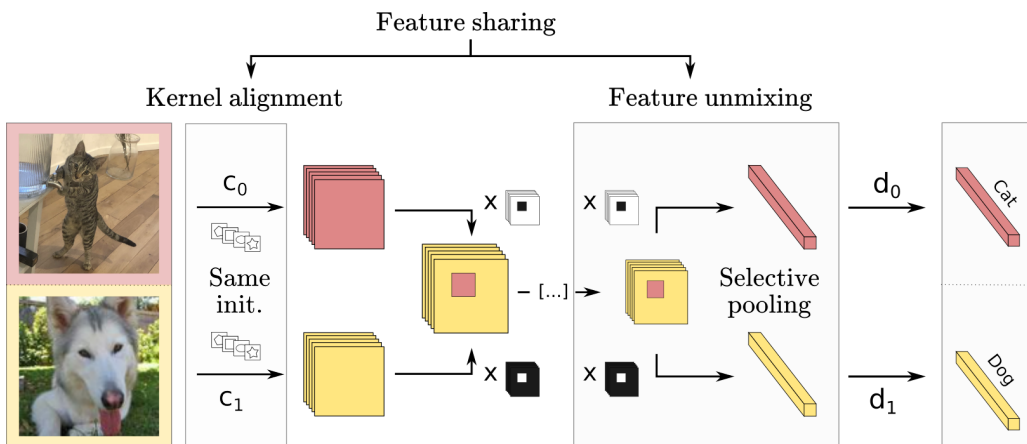


Figure 3.13. – **Overview of the MixShare framework.** Two steps are necessary to allow feature sharing: 1) Ensure the subnetworks share a “common language” by initializing the convolutional encoders to be similar. 2) Extract descriptions of each input from model features.

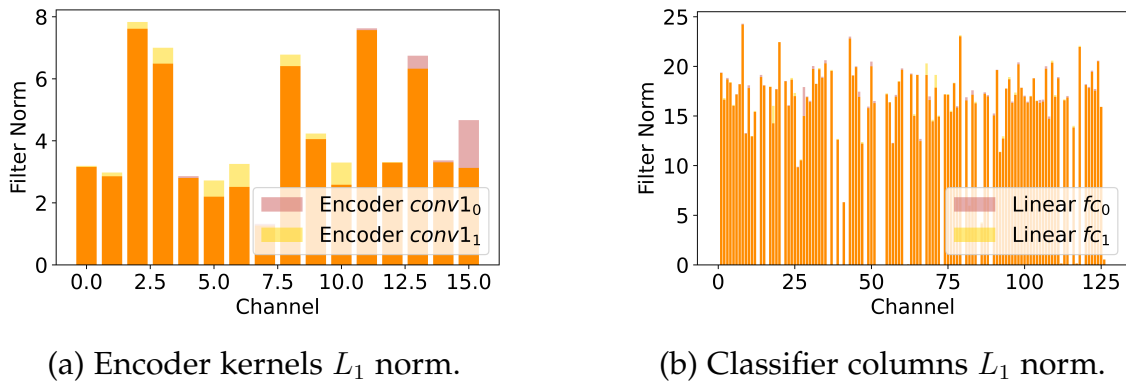


Figure 3.14. – **Usage of features by MixMo subnetworks with unmixing.** Applying unmixing leads to features being used by both subnetworks: the overlap (orange) is very high.

3.4.2 How can subnetworks share features?

We discuss here the obstacles preventing feature sharing in multi-input multi-output architectures, and propose solutions to correct this behavior.

3.4.2.1 Unmixing: extracting features for each input

We build upon an intuition put forth in MixMo (Rame et al. 2021): the lack of feature sharing is caused by the need for individual classifier at the end of the network to extract class information for one input specifically. Indeed, the M classifiers have access to the exact same set of extracted features. If two classifiers use the same feature, that feature needs to describe the state of two different inputs. This is an issue when one accounts for the fact inputs are in fact drawn independently and there can therefore be no meaningful feature describing the state of two inputs simultaneously.

Let us now consider how the classifiers should ideally behave on shared features. Since each classifier is paired to one of the input pathways, they should be able to extract two different interpretations of the shared features that still encode the same functional information (see Figure 3.13). For instance, the shared feature should encode for the presence of flowers but each classifier should be able to infer from the feature whether its personal input contains flowers.

While this is not the case in traditional CNNs, MixMo (Rame et al. 2021) introduces a modification to the seminal MIMO architecture that causes feature maps to encode information about the different inputs separately. Indeed, since MixMo mixes inputs according to some binary mixing augmentation scheme (typically

Method	28-2			28-5		
	Acc. ens.	Acc. Ind.	Classifier share rate	Acc. ens.	Acc. Ind.	Classifier share rate
MixMo	74.8 ± 0.2	71.6 ± 0.2	4.9 ± 0.4	81.9 ± 0.1	79.4 ± 0.2	8.2 ± 0.1
MixMo + Unmix	69.7 ± 15.4	69.7 ± 15.4	99.1 ± 0.4	79.4 ± 2.7	79.4 ± 2.7	98.8 ± 0.7
MixMo + Unmix + kernel init.	79.0 ± 0.1	79.0 ± 0.1	99.4 ± 0.1	82.1 ± 0.2	82.1 ± 0.2	99.3 ± 0.1
MixShare (partial on 25% features)	73.3 ± 0.5	71.5 ± 1.0	60.6 ± 6.5	79.9 ± 0.3	78.8 ± 0.4	60.0 ± 2.3
MixShare (fadeout to 100 epochs)	79.0 ± 0.1	76.7 ± 1.3	64.4 ± 6.0	82.4 ± 0.3	81.6 ± 0.5	62.7 ± 8.3

Table 3.6. – **Performance of MixShare.** Overall ensemble accuracy (%), average subnetwork accuracy (%) and classifier (output block) sharing rate (see Sec. 3.4.1) for MixMo variants, *mean ± std* reported over 3 runs. Mixshare with fadeout unmixing yields both strong individual models and ensemble gains.

CutMix (Y. Yang and Soatto 2019)), each pixel on the final feature maps encodes information about one of the inputs.

This is fortunate as it provides us with a fairly natural solution: unmixing. **Unmixing** (illustrated in Figure 3.13) recycles the binary masks generated for input mixing in order to filter the feature maps so that only information relevant to a specific input is contained in the unmixed version. This way, a single feature map can describe each of the inputs.

Figure 3.14 shows that applying unmixing causes the subnetworks to share features, both in the input and output block. In fact, every feature in the unmixed model is used by all subnetworks which proves unmixing indeed solves the core obstacle to feature sharing in MIMO networks.

Introducing unmixing however leads to unstable and generally worse performance as seen in Table 3.6. Crucially, even individual subnetwork accuracy suffers from unmixing which suggests an underlying issue.

3.4.2.2 Aligning encoder kernels to allow efficient feature sharing

Intuitively, feature sharing should at the very least lead to higher individual subnetwork accuracy as the subnetworks use more parameters. As such, we now investigate why unmixing degrades performance so dramatically.

By extracting multiple possible interpretations of a single feature, unmixing introduces a new problem in the model. Indeed, we need our interpretations of the same feature to encode the same functional characteristics (e.g. flower detection). The issue is that a randomly initialized multi-input multi-output network typically leads to having multiple interpretations of the same feature.

Indeed, the encoders computing the mixed representations are very different. For an input feature, the mixed feature map could contain information about

horizontal borders on input 1 and vertical borders on input 2. As such, there is no consistent interpretation for our mixed features.

We can unify the interpretation of unmixed features at the start by simply **aligning the kernels of the encoders**. Indeed, as long as each feature encodes the same sort of information for each encoder, there should be no ambiguity introduced by the unmixing process.

Table 3.6 shows that fixing the initialization scheme of the encoders to the same value does indeed lead the model to outperform normal mixmo models.

3.4.2.3 Towards partial feature sharing

While proper unmixing does allow feature sharing in multi-input multi-output networks, Table 3.6 and Figure 3.14 show it leads to subnetworks sharing all features: the subnetworks are identical. This is even less desirable than fully separated subnetworks as it makes ensembling pointless (Pang et al. 2019; Rame and Cord 2021).

Ideally, subnetworks would share some parameters but still remain distinct functionally. This way, we would be able to strike a compromise between fully separated and fully shared subnetworks. The issue with this however, is that removing obstacles to feature sharing makes it unnecessary for subnetworks to separate in any way.

In this preliminary work, we discuss two solutions: **partial unmixing** and **fadeout unmixing**. **Partial unmixing** is a straightforward solution where we only apply unmixing to a fixed subset of the final feature maps (e.g. 25%). In **Fadeout unmixing** we start training the network with proper unmixing but progressively reduce the strength of unmixing so that there is no unmixing towards the end of the procedure. For instance, we use the unmixing mask $M + r(1 - M)$ (instead of M) with $r = \min(1, epoch/100)$ if we want to stop unmixing by epoch 100. As such, fadeout unmixing initializes the network in a shared state and progressively pushes the subnetworks to develop independent features.

We now propose the full MixShare framework by combining proper kernel initialization and partial/fadeout unmixing along with slight adjustments to standard MIMO procedures like input repetition (Havasi et al. 2021) and loss balancing (Rame et al. 2021) (see Appendix). Table 3.6 shows that both MixShare variants succeed in causing partial feature sharing. **Partial** fails to train strong individual subnetworks, but still showcases ensemble benefits. **Fadeout** on the other hand leads to strong performances and retains significant ensembling benefits on medium sized networks like a WideResNet 28-5.

3.5 Conclusion

In this chapter, we study how training models to output separate predictions on a MSDA sample’s parent inputs leads to the emergence of concurrent subnetworks. Training multiple subnetworks within the same base networks has a number of benefits: the subnetworks’ predictions can be ensembled, more features in the network see use, and features shared by the subnetworks should in theory be more general as they can be used to describe two independent inputs.

We first proposes MixMo, which analyses MIMO architectures through the lens of Mixing Samples Data Augmentations. Given this point of view, we replace the sum based input mixing block by a CutMix based one to train stronger and more diverse subnetworks. We also recognize in the multiple outputs of the MIMO architecture a deconstruction of the standard soft labels. As such, we dynamically balance the weights of the subnetwork training losses to reflect the input mixing proportions.

As MixMo proves to have difficulty on smaller networks, we study the source of the problem with a new MixShare framework. After confirming that subnetworks share absolutely no features, we seek to solve MIMO models’ dependency on large base networks by enabling feature sharing. We find introducing a novel unmixing mechanism leads - with proper initialization of the input encoders - the subnetworks to share all features. Unmixing uses knowledge on the input mixing process to provide different interpretations of the same final network feature, one for each input.

Experiments show MixMo improves on both ensembling and MSDA baselines for ResNet based architectures on the CIFAR datasets and TinyImageNet, with and without standard MSDA on the inputs. Interestingly, further experiments show MixMo make very efficient use of the base models’ features and trains fairly robust models (as measured on the CIFAR-100-C dataset). MixShare succeeds in allowing feature sharing between subnetworks, but the subnetworks collapse to be identical. With a weakened version of unmixing to prevent that, we recover very good performance for MixShare on smaller architectures.

All in all, breaking down the soft label prediction for a mixed sample into distinct predictions succeeds in improving model performance and proves to be orthogonal to standard mixing augmentations. While the framework has difficulty scaling down to smaller networks, enabling feature sharing between networks seems to alleviate the issue. The procedure outlined in MixShare is however sensitive and requires careful tuning. As such, the next step should be to find a better way to balance feature sharing.

The unmixing mechanism we introduce in MixShare does not integrate very well with the way convolutional neural networks work. Indeed, CNNs are fundamentally built to learn one feature per feature map and activations late in the network tend to encompass the whole input in theory. Interestingly, this type of interaction is instead very natural in the context of attention mechanisms. As such, the recently proposed Vision Transformer architectures should prove much more adapted to training MIMO subnetworks that share features and cooperate during training.

MULTI-INPUT MULTI-OUTPUT MSDA, UNMIXING AND ATTENTION MECHANISMS

Contents

4.1	Introduction	79
4.2	Related Work	80
4.2.1	Attention	80
4.2.2	Vision Transformers	81
4.2.3	Positioning	82
4.3	MixViT: A MIMO MSDA formulation of Vision Transformers	83
4.3.1	Transposing MIMO frameworks to vision transformers	84
4.3.2	Overview of MixViT	86
4.3.3	MixViT framework	86
4.4	Results of the MixViT framework	89
4.4.1	MixViT significantly improves the performance of vision transformers	90
4.4.2	Comparison against CNN-based MIMO methods	91
4.4.3	Implicit regularization for simpler training settings	92
4.4.4	Ablations	94
4.5	Conclusion	95

4.1 Introduction

We have shown in [Chapter 3](#) that predicting separately the original samples in a MSDA image leads to training concurrent subnetworks within the base network. The unmixing mechanism necessary to allow the subnetworks to share features is a bit difficult to optimize with on Convolutional Neural Networks however. It is however very reminiscent of the attention mechanism at the heart of the emerging Vision Transformers.

Vision Transformers (ViT) have started to overtake Convolutional Neural Networks on a large number of vision benchmarks over the last few years. ViTs present quite a few benefits over traditional CNNs: they often outperform CNNs

for a similar number of parameters, can learn non local relationships much more easily, accommodate variable input sizes, and their attention mechanism is inherently explainable. Importantly, transformers have been shown (Touvron et al. 2021a) to require strong regularization to work. The MIMO paradigm is therefore well adapted to these models as training multiple subnetworks has strong regularizing effects (Sun et al. 2022; Cygert and Czyżewski 2022), and the attention mechanism natively performs a process analogous to unmixing. In spite of this, no previous work has successfully adapted MIMO frameworks to Vision Transformers.

In this chapter, we study how Multi-Input Multi-Output MSDA can be adapted to the Vision Transformer architecture, and in particular how differently the trained subnetworks behave compared to a MIMO CNN. Indeed, since ViTs naturally perform some sort of unmixing, a MIMO ViT should behave quite differently considering the difficulties observed when training the MixShare model. We propose the **MixViT** framework, deferring the separation of subnetworks to the last layers of the network. We achieve this separation by using the same encoder for the two inputs and instead adding a source attribution mechanism towards the end of the network to indicate which subnetwork should process a patch token.

After a brief introduction to the attention mechanism and common Vision Transformer frameworks in Section 4.2, we will discuss our MixViT framework (Section 4.3.2) and our experimental results (Section 4.4).

The work conducted in this chapter of this thesis has led to one publication:

- SUN Rémy, MASSON Clément, THOME Nicolas and CORD Matthieu. (2022) “Adapting Multi-Input Multi-Output schemes to Vision Transformers”, submitted to ICLR 2023, in CVPR 2022 workshop on transformers and attention.

4.2 Related Work

As MixViT departs from Convolutional Neural Networks to work on Vision Transformers, we provide here a brief introduction to the attention mechanism and Vision Transformers.

4.2.1 Attention

At a very high level, the attention mechanism aims to represent an input token (ie, a d -dimensional vector) by how it relates to a set of contextual tokens: it looks to the “attention” the token must pay to part of a context (Bahdanau et al. 2015).

Let us consider an input t represented by a query token q and context tokens $\{c_i\}_{i=0\dots N-1}$ represented by value tokens $\{v_i\}_{i=0\dots N-1}$ and key tokens $\{k_i\}_{i=0\dots N-1}$. In essence, the attention mechanism will first compute the attention weights

$$\alpha_i = q^T k_i, \quad (4.1)$$

before computing the attended representation o of the input token

$$o = \text{Attention}(t, \{c_i\}_{i=0\dots N-1}) = \sum_{i=0}^{N-1} \alpha_i v_i. \quad (4.2)$$

In practice, transformers formalize this problem with three learnable matrices W_q , W_k and W_v such that $q = W_q^T t$, $k_i = W_k^T c_i$ and $v_i = W_v^T c_i$ (Vaswani et al. 2017).

In Vision Transformers, the base attention block most often used is the self-attention block (Cheng et al. 2016). The Self-Attention (SA) block takes as input a set of tokens $\{t_i\}_{i=0\dots N-1}$, and outputs a set of refined attended tokens $\{o_i\}_{i=0\dots N-1}$. The peculiarity of self-attention is that the tokens $\{t_i\}_{i=0\dots N-1}$ play the roles of input (query) token and context tokens both:

$$o_j = \text{Attention}(t_j, \{t_i\}_{i=0\dots N-1}). \quad (4.3)$$

Over the years, many variants of this SA block have been developed. Multi-Head Self-Attention (MHSA) blocks - the actual version used in the seminal ViT (Dosovitskiy et al. 2021) - simply compute multiple self-attention mechanisms (“heads”) in parallel and aggregate the resulting representations with a dense projection. Gated Positional Self-Attention (GPSA) blocks (D’Ascoli et al. 2021) take into account the relative position of tokens when computing the attention, and use a special initialization scheme so the block emulates a convolutional operation at the start of training. Class-Attention (CA) blocks (Touvron et al. 2021b) only computes the attention of an additional “class” token with the input tokens, keeping the representation of the standard tokens fixed.

4.2.2 Vision Transformers

While this attention mechanism and the Transformer architecture (Vaswani et al. 2017) were initially developed for Natural Language Processing, they have recently been adapted to Computer Vision problems (Dosovitskiy et al. 2021). As can be seen from Figure 4.1, only minimal changes must be made to accommodate the change in modality: the image is separated into N patches, and the patches are embedded by a shared embedding layer into (vector) tokens. These tokens

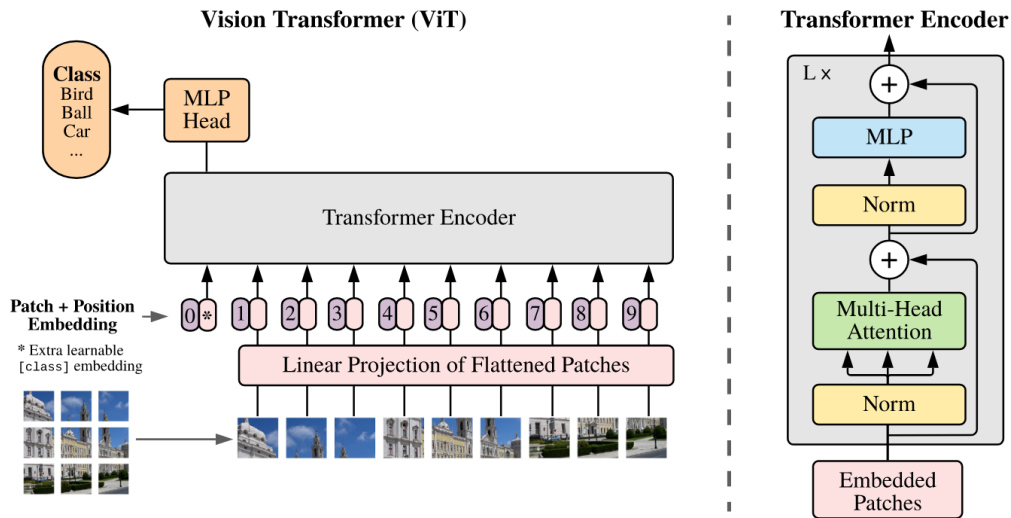


Figure 4.1. – **Overview of the seminal ViT framework.** The image is divided into patches that are tokenized and fed to transformer blocks.

are then passed as input to L transformer blocks. Transformers blocks themselves are simply made from an attention block and a few linear layers along with some residual connections.

Classification is performed by inserting an additional learnable classification token into the set of patch tokens. This classification token progressively aggregates information from the patch tokens through successive layers, and a linear classification layer outputs predictions from the feature representation of this classification token after L transformer blocks.

This base framework offers a general skeleton whose components are often modified to solve a number of underlying issues. For instance, a number of architectures have modified the attention mechanism to emulate convolutional neural networks (D’Ascoli et al. 2021; Z. Liu et al. 2021) since Vision Transformers need help learning local structures inherent to vision problems. Others have taken to inserting the classification token in the last layers of the model (D’Ascoli et al. 2021; Touvron et al. 2021b) or altogether replacing it by a pooling step on the patch token features (Zhai et al. 2022).

4.2.3 Positioning

In this chapter, our focus remains on Multi-Input Multi-Output Mixing Sample Data Augmentation. We are only interested in Vision Transformers in so far that developing a MIMO MSDA framework for this new architecture has partic-

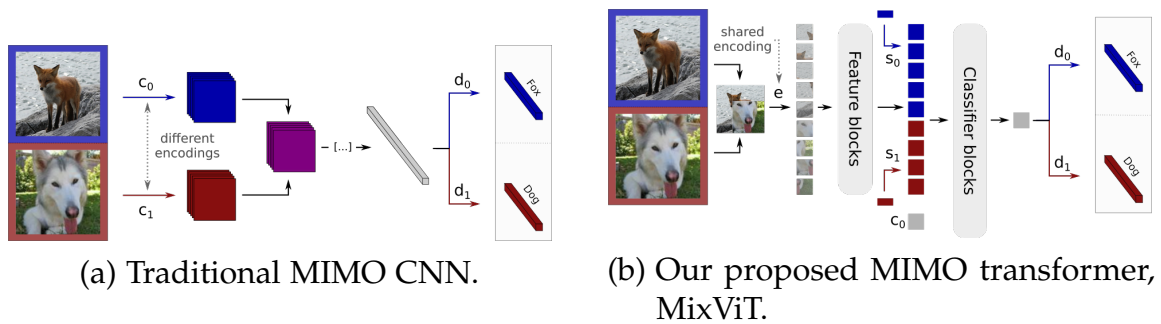


Figure 4.2. – **Overview of MIMO architectures.** MixViT, our transformer only separates subnetworks in the last layers which facilitates feature sharing between subnetworks and makes it possible to avoid summing the encoded inputs (purple in (a)) at inference (see Sec. 4.3.1).

ular implications in terms of feature sharing. Indeed, the use of a classification token along with the attention mechanism used at all stages of ViTs natively emulates the unmixing mechanism introduced in Chapter 3. While we add new components to the architecture, our work is largely meant to be adaptable to most existing Vision Transformer backbones.

4.3 MixViT: A MIMO MSDA formulation of Vision Transformers

Given the clear differences between Vision Transformers and standard Convolutional neural networks, careful thought must be given to a MIMO formulation of ViTs.

We find training properly designed MIMO vision transformers is both inexpensive and extremely beneficial in terms of performance. We observe this by proposing MixViT, the first MIMO transformer that outperforms its single-input single-output backbone. As shown on Figure 4.2b, we achieve this by deferring the separation of subnetworks to the last layers of the model, and introducing a novel “source attribution” mechanism to facilitate this separation. MixViT mixes patches from M inputs before embedding them as tokens with a linear projection e shared across subnetworks and feeding them to feature extraction blocks. We then use our “source attribution” to associate each attended token to the input/subnetwork they pertain to, and aggregate features with a class token. Finally, we get M predictions from the class token features with M different dense layers.

Our contributions are therefore as follows:

- We identify and address vision transformers’ difficulty in sharing tokens across subnetworks in [Section 4.3.1](#).
- Through our MixViT framework, we show subnetworks in vision transformers can - and should - only be located in the very last layers of the model. As such, MixViT adds only marginal costs at training and inference while strongly regularizing the model.
- We introduce a novel source attribution mechanism to facilitate this late separation of subnetworks.

4.3.1 Transposing MIMO frameworks to vision transformers

We start with a straightforward formulation of MIMO (Havasi et al. 2021; Rame et al. 2021) vision transformers. This section shows these translations fail on vision transformers, and highlights how vision transformer struggle to share tokens across subnetworks. We directly introduce the generalized MIMO strategy introduced in MixMo (Rame et al. 2021). For simplicity, **we discuss the usual case where $M = 2$** subnetworks (Havasi et al. 2021; Rame et al. 2021) are trained unless specified otherwise in this paper.

The following experiments consider the CIFAR-100 dataset (Krizhevsky and Hinton 2009) with a variant of the ConViT (D’Ascoli et al. 2021) architecture (see [Section 4.4](#)), but the conclusions hold for other settings. The backbone single-input single-output model feeds image patches to 5 Gated-Positional Self-Attention (GPSA (D’Ascoli et al. 2021)) blocks (which initially imitate convolutions) for feature extraction, before adding a classification token and applying 2 Class-Attention (CA (Touvron et al. 2021b)) blocks. Classification is then performed as usual by a dense layer on the attended classification token’s features.

4.3.1.1 MIMO and MixMo formulations of Vision Transformers

At first blush, MIMO architectures can easily be transposed to vision transformers by analogy with convolutional neural networks (figured in [Figure 4.2a](#)). As such, this section looks into the baseline MIMO (Havasi et al. 2021) and MixMo (Rame et al. 2021) frameworks applied to ViTs. We outline below how vision transformers can replicate the steps outlined in [Figure 4.2a](#) step by step, and provide an illustration of MIMO transformers in [Figure A.9](#) of [Appendix Section A.7.1](#).

We consider (like in [Figure 4.2a](#)) M (inputs, labels) pairs $\{(x_i, y_i)\}_{0 \leq i < M}$ during training: **we feed M inputs to the model as a combined input**. To this end, we break down each image into small 4×4 patches and embed the patches into N

Backbone	Method	Inference Mix.	# FLOPS	Accuracy (%)	Subnetwork Acc. (%)
ConViT	(1) Single-input (D’Ascoli et al. 2021)	N/A	1×	79.5 ± 0.1	-
	(2) MIMO (Havasi et al. 2021)	Sum	1×	77.1 ± 0.1	74.8 ± 0.1
	(3) MixMo (Rame et al. 2021)	Sum	1×	77.0 ± 0.2	75.2 ± 0.1
	(4) MixMo (Rame et al. 2021)	Separate	2×	78.5 ± 0.2	76.4 ± 0.1
	(5) CutMix-only MixMo	Separate	2×	80.1 ± 0.2	78.3 ± 0.3

Table 4.1. – **Performance of MIMO formulations of ViTs.** CIFAR-100 accuracy and estimated FLOPS of the backbone and MIMO formulations.

384-dimensional tokens using M linear embedding layers e_i (one for each input). Corresponding tokens (at the same position in the two images) are then mixed into one common set of tokens (either through sums (Havasi et al. 2021) or other mixing schemes (Rame et al. 2021)). The core network (feature blocks + classification blocks) then processes this representation until a final attended classification token is obtained. M dense layers $\{d_i\}_{0 \leq i < M}$ - one for each subnetwork - then yield M predictions from this attended classification. At test time, M predictions can be obtained for one image by embedding the image patches with the M linear embeddings and summing the sets of embedded patch tokens.

In the seminal MIMO framework, we mix the embedded tokens by simply summing them. The CutMix-based MixMo formulation interpolates between the tokens following either a randomly drawn square mask with ratio $\lambda \sim \beta(\alpha, \alpha)$, or a linear scheme (similar to summing).

4.3.1.2 MIMO frameworks do not translate well to transformers

Lines (2) and (3) in Table 4.1 show our direct transpositions of the seminal MIMO and MixMo frameworks do not outperform their single-input single-output ConViT backbone (line (1)): both frameworks fail to even reach the same performance. While the subnetworks still provide meaningful gains when ensembled, the individual performance of the subnetworks is very poor which brings down the overall accuracy. Applying the same solution as in MixShare does not seem to address the issue here as we study in Section A.7.2.

We posit that transformers have difficulty encoding multiple inputs within the same patch token. Indeed, we find spreading the inputs across separate tokens solves the problem. More precisely, instead of mixing the patch tokens of the inputs, we only keep the patch tokens relevant to one of the subnetworks and extract the corresponding predictions. By performing separate evaluations - one for each subnetwork - we can recover predictions from each subnetwork. Table 4.1’s line (4) shows performing inference in this way on the models trained in line (3) significantly improves performance. Furthermore, the model outperforms the

baseline when no summing is performed during training in line (5) (the patches are always mixed following a CutMix (Y. Yang and Soatto 2019) scheme).

From this, we can posit that the issue stems from sharing tokens between sub-networks. While Sum mixing has already been heavily criticized as sub-optimal by MixMo (Rame et al. 2021), it however remains necessary in MIMO frameworks at inference since both subnetworks need to see the whole input in one forward pass. Importantly, our corrected inference process needs to perform two forward passes, which is not realistic as it doubles the inference overhead.

In this paper, we show that - contrarily to CNN-based MIMO models - transformers can avoid this issue by deferring the subnetwork separation and input encoding to much later in the network (see Section 4.3.2 and Figure 4.2b). Interestingly, this simple restructuring even improves performances and allows us to introduce new subnetwork-encoding mechanisms.

4.3.2 Overview of MixViT

We propose in this paper MixViT (Figure 4.3), a multi-input multi-output framework suited to vision transformers that trains distinct subnetworks in the last layers of the network to address the issues raised in Section 4.3.1 and favorize feature sharing among subnetworks. MixViT relies on a novel “source attribution” mechanism to map the patch tokens to the different subnetworks after the feature extraction blocks. This section starts by introducing MixViT (Section 4.3.3) and source attribution (Section 4.3.3.1), before discussing MixViT’s added overhead in Section 4.3.3.2.

The only assumption our formulation of MixViT requires is a separation of the transformer into feature extraction blocks that use Self-Attention blocks and classifier blocks that rely on Class-Attention blocks (see Figure 4.2b). A number of modern vision transformers follow this structure like the ConViT (D’Ascoli et al. 2021) variant studied in Section 4.3.1 and the CaiT (Touvron et al. 2021b) architecture. Moreover, MixViT easily generalizes to other architectures.

4.3.3 MixViT framework

At training time, the $M \times N$ patches extracted from the M inputs are **mixed** into N patches following a corrected CutMix (Y. Yang and Soatto 2019) scheme (ie, so that each position takes only one patch on its own without interpolation) with masks $\{\mathcal{M}_i\}$ (see Figure 4.3) with a ratio $\lambda \sim \beta(\alpha, \alpha)$. The N mixed patches are then encoded into d -dimensional tokens by a linear layer e . Learnable positional

embeddings are added to the tokens and the tokens are fed to the L Self-Attention blocks that serve as feature extraction blocks. This yields N attended tokens t but no indication which input/subnetwork each token belongs to.

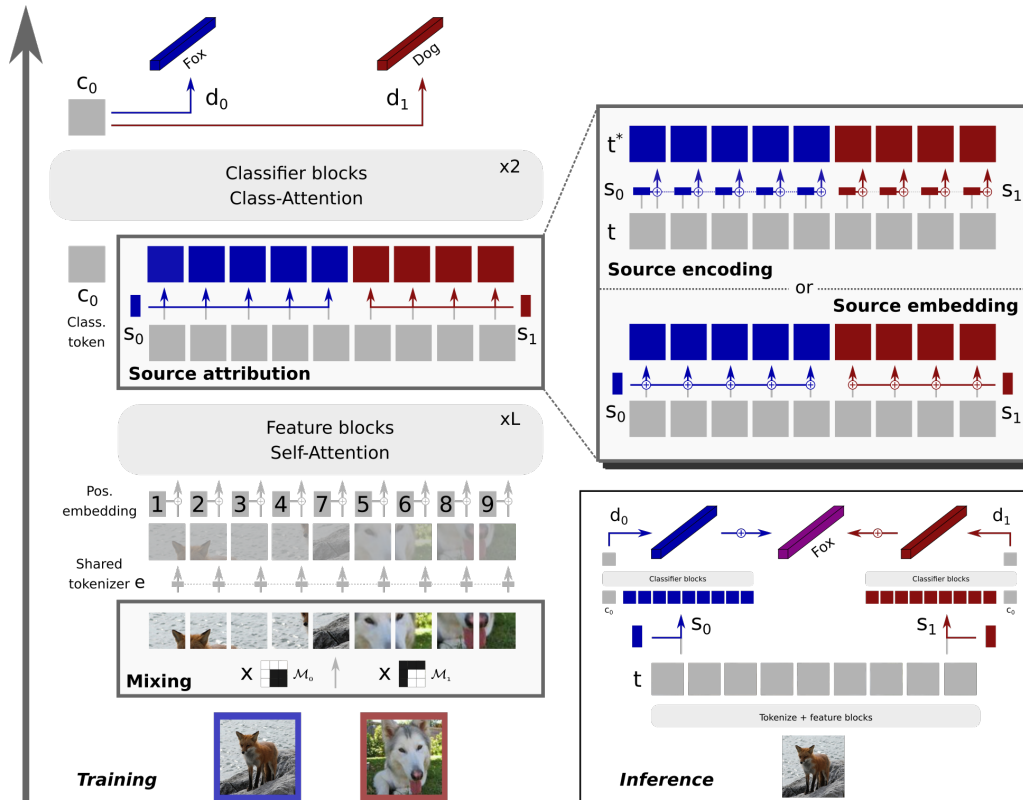


Figure 4.3. – **Overview of our MixViT framework.** At training, we insert a mixing and a source attribution step in the normal flow of the network. Source attribution associates tokens with the relevant input/subnetwork, and can be done through either source encoding or source embedding. At inference, we perform multiple passes on the last layer using differently sourced tokens.

We propose a novel **source attribution** mechanism s_i to specify which input or subnetwork each patch pertains to. The source attribution adds subnetwork-specific information to the tokens following either a **source encoding** or **source embedding** mechanism as we elaborate on in Section 4.3.3.1. The sourced tokens t^* are passed - along with a learnable classification token c_0 - to the remaining classifier blocks. After the class-attention blocks, we retrieve the classification token’s features for classification. M predictions p_i are then obtained from this feature representation through the use of M different dense layers d_0 . We then use the input targets y_i to optimize the subnetworks’ cross-entropy losses weighted by the mixing ratio λ following MixMo (Rame et al. 2021):

$$\mathcal{L}_{MixViT} = \lambda \cdot l_{CE}(y_0, p_0) + (1 - \lambda) \cdot l_{CE}(y_1, p_1). \quad (4.4)$$

At inference, MixViT only has to perform feature extraction on the image’s patch tokens once as shown on [Figure 4.3](#): **no mixing** is needed since the tokens are subnetwork-agnostic. We then collect predictions for each subnetwork i by using s_i to specialize the general feature patch tokens t for the subnetwork through **source attribution**, and then passing these specialized tokens to the classification blocks along with c_0 (see [Figure 4.3](#)). This allows us to keep subnetwork-specific tokens as was shown to be required in [Section 4.3.1](#) without having to perform multiple full forward passes in the network.

4.3.3.1 Source attribution mechanisms

We move the subnetworks to the end of vision transformers by introducing source attribution mechanisms s_i after the feature extraction blocks to specialize the **generic patch tokens** t into **subnetwork-specific tokens** t^* . Conceptually, this replaces the separate patch encoders used in our naive formulation of a MIMO ViT in [Section 4.3.1](#).

This is most naturally achieved through the use of subnetwork-specific **source encoders**, similarly to the different image encoders used by MIMO and MixMo. More precisely, we define our source encoders s_i such that $s_i = \text{Linear}(W_i^s; b_i^s)$ are Linear layers. Intuitively, each patch token is projected by the relevant source encoder s_i . In practice, we project the feature tokens t with the Linear layers s_i into specialized tokens, and mix these specialized tokens following the binary masks \mathcal{M}_i used on the inputs. The resulting projection is added to original residual token to obtain our subnetwork-specific token t^* as figured in [Equation 4.5](#) (\odot is the Hadamard product):

$$t^* = t + \sum_{i=0}^{M-1} \mathcal{M}_i \odot s_i(t) = t + \sum_{i=0}^{M-1} \mathcal{M}_i \odot (t \times W_i^{sT} + b_i^s). \quad (4.5)$$

We also propose an alternative lightweight **source embedding** mechanism to avoid incurring complications from training additional Linear layers (see [Equation 4.6](#)). Source embeddings s_i act similarly to positional embeddings, and are d -dimensional vectors that are added to feature tokens to specify which subnetwork the patch is relevant to. This therefore only adds Md parameters and $\mathcal{O}(NMd)$ operations which is negligible in ViTs. Interestingly, source embeddings are actually a particular case of source encodings where the weight matrix is null such that $s_i = \text{Linear}(0^{d \times d}; b_i^s)$ and

$$t^* = t + \sum_{i=0}^{M-1} \mathcal{M}_i \odot s_i(t) = t + \sum_{i=0}^{M-1} \mathcal{M}_i \odot (t \times 0^{d \times d^T} + b_i^s). \quad (4.6)$$

4.3.3.2 Regarding computational overhead

Deferring the separation of subnetworks to the last layers allows us to perform separate evaluations for each subnetwork at a very low computational cost. Consider a transformer on a C -class problem with L self-attention blocks and 2 class-attention blocks taking N d -dimensional patch tokens. **As a rule of thumb**, one forward pass of MixViT roughly - at the most - incurs $\frac{L+2M}{L+2} \times$ as many computations as a normal model's during inference, and $1 \times$ at training.

Indeed, the most intensive computations are tied to Linear $d \times d$ weight multiplications: d is usually much larger than N , C or M . While source encoding does incur M such computations per token, each Self-Attention layer incurs 6 per token and Class-Attention slightly more than 3 per token: even for our small ConViT (see Section 4.3.1) with $M = 2$ this represents about $\frac{6 \times 5 + 3 \times 2 + 2}{6 \times 5 + 3 \times 2} \simeq 1.06 \times$ multiplications. The amount of additional computations is therefore bounded by the ratio of passes through attention blocks. This approximation could easily be lowered however: it assumes class-attention is as expensive as self-attention which is very pessimistic (see Appendix Section A.7.3 for tighter approximations and asymptotic complexity).

4.4 Results of the MixViT framework

We first show our proposed MixViT framework strongly improves on whichever backbone it builds upon in Section 4.4.1, and outperforms MIMO CNNs (Section 4.4.2) on TinyImageNet. Furthermore, Section 4.4.3 shows MixViT is much more resilient on simpler training settings. We then compare MixViT to state-of-the-art transformers (Section 4.4.3.1). Finally, we demonstrate in Section 4.4.4 that separating subnetworks at the end of the model with our source attribution mechanism is indeed very beneficial.

Setting In this paper, we run experiments on the CIFAR-10/100 (Krizhevsky and Hinton 2009), TinyImageNet (Chrabaszcz et al. 2017) and ImageNet-100 (Tian et al. 2020) datasets. Most experiments use a variant of the ConViT (D'Ascoli et al. 2021) architecture with 5 GPSA blocks and 2 CA blocks with an embedding dimension of 384 and 12 heads (see Appendix). On the more complex datasets, we also use deeper CaiT (Touvron et al. 2021b) architectures. Our transformers

work on $8 \times 8 = 64$ patches for CIFAR and TinyImageNet, and $14 \times 14 = 196$ patches on ImageNet-100. We report the best epoch overall ensemble accuracy as *mean* \pm *std* over three seeded runs.

Implementation details We mix inputs with a corrected CutMix scheme detailed in Appendix, and start training the subnetworks individually towards the end of training to ready the subnetworks to work on whole images. We largely adapt the DeiT (Touvron et al. 2021a) training settings that are widely used in the literature (Z. Zhang et al. 2022; Touvron et al. 2021b; S. H. Lee et al. 2021) (see Appendix). Unless specified otherwise, we therefore train with the AdamW (Loshchilov and Hutter 2019) optimizer for 150 epochs with weight decay, RandErase (Zhong et al. 2020), AutoAugment (Cubuk et al. 2019), stochastic depth (G. Huang et al. 2016), label smoothing (Szegedy et al. 2016) and 3 Batch augmentations (Hoffer et al. 2020). We also apply CutMix (Y. Yang and Soatto 2019) and MixUp (H. Zhang et al. 2018) but find applying them simultaneously instead of alternatively yields better result. Finally, we keep the 0.001 learning rate but adopt a step decay instead of cosine.

4.4.1 MixViT significantly improves the performance of vision transformers

Table 4.2 shows both of our MixViT variants provide significant improvements over their single-input single-output counterpart across a multitude of architectures and datasets. For instance, on TinyImageNet, our larger CaiT-XS model goes from an accuracy of 68.6% as a single-input single-output model to 70.9% accuracy with our MixViT-encoding protocol. Interestingly, while MixViT shows solid gains from ensembling, a large parts of the gains are due to a much higher accuracy of the individual subnetworks: the framework also has a strong regularizing effect, possibly due to the feature extractor being shared.

Our lightweight embedding-based variant of MixViT actually outperforms the more complete MixViT-encoding model on CIFAR-10/100 (Table 4.2d and Table 4.2c). MixViT-encoding however overtakes it on the more complex TinyImageNet (Table 4.2a). While the difference is minimal on our small ConViT model, MixViT-encoding proves much stronger on the deeper CaiT architectures. Interestingly, we observe this difference is deepened by both model depth (our ConViT has 7 layers against our CaiTs’ 26) and model width (Cait-XXS uses 192-dimensional embeddings against Cait-XS’s 288). This demonstrates the benefits of both our proposed source attribution mechanisms: source embeddings are much faster and easier to train, but source encodings are more powerful overall.

(a) TinyImageNet			(b) ImageNet-100		
Backbone	Method	Accuracy (%)	Backbone	Method	Accuracy (%)
ConViT	Single-input/output	65.4 ± 0.2	ConViT	Single-input/output	86.6 ± 0.2
	MixVit-embedding (ours)	70.0 ± 0.1		MixVit-embedding (ours)	88.8 ± 0.3
	MixVit-encoding (ours)	70.2 ± 0.2		MixVit-encoding (ours)	88.6 ± 0.5
CaiT XXS-24	Single-input/output	66.2 ± 0.3	CaiT XXS-24	Single-input/output	85.8 ± 0.2
	MixVit-embedding (ours)	68.6 ± 0.3		MixVit-embedding (ours)	87.2 ± 0.3
	MixVit-encoding (ours)	69.6 ± 0.2		MixVit-encoding (ours)	86.7 ± 0.2
CaiT XS-24	Single-input/output	68.6 ± 0.1	CaiT XS-24	Single-input/output	87.6 ± 0.4
	MixVit-embedding (ours)	69.4 ± 0.1		MixVit-embedding (ours)	89.7 ± 0.3
	MixVit-encoding (ours)	70.9 ± 0.2		MixVit-encoding (ours)	89.3 ± 0.2
(c) CIFAR-100			(d) CIFAR-10		
Backbone	Method	Accuracy (%)	Backbone	Method	Accuracy (%)
ConViT	Single-input/output	79.5 ± 0.1	ConViT	Single-input/output	96.1 ± 0.1
	MixViT-embedding (ours)	82.4 ± 0.1		MixViT-embedding (ours)	96.5 ± 0.2
	MixViT-encoding (ours)	82.1 ± 0.1		MixViT-encoding (ours)	96.5 ± 0.2

Table 4.2. – **Gains from using MixViT.** MixViT significantly improves the performance of vision transformers

4.4.2 Comparison against CNN-based MIMO methods

On the complex TinyImageNet dataset MixViT largely outperforms its much larger CNN-based competition (Table 4.3) in spite of a much shorter training scheme (150 epochs vs. 1000). Interestingly, this highlights a fundamental difference between traditional MIMO frameworks and our MixViT models. MixMo learns largely independent subnetworks that share no features

(Rame et al. 2021; Sun et al. 2022) and therefore struggles to fit good subnetworks on narrow architectures. As such, MixMo needs to consider a wide ResNet-18-3 (He et al. 2016b) backbone to start showing improvements. MixViT simply does not have this issue as the subnetworks share features and a large part of the gains come from how the subnetworks learn features they can both use.

Backbone	Method	# Params	Accuracy (%)
ResNet 18	Single-input/output	11M	65.1
	MixMo	11M	64.4
ResNet 18-3	Single-input/output	100M	69.0
	MixMo	100M	70.2
CaiT XS-24	Single-input/output	26M	68.6 ± 0.1
	MixVit-embedding	26M	69.4 ± 0.1
	MixVit-encoding	26M	70.9 ± 0.2

Table 4.3. – **Comparison of MixViT and MixMo.** MixViT outperforms CNN-based MixMo models on Tiny-ImageNet.

4.4.3 Implicit regularization for simpler training settings

Typical training schemes for vision transformers often require both heavy augmentation (Y. Yang and Soatto 2019; H. Zhang et al. 2018) (in the form of mixing augmentations) and batch augmentations (Hoffer et al. 2020). This is concerning as these procedures can significantly extend training costs and time.

MixViT inherently provides some of the benefits of MSDA and batch augmentation. Indeed, we show mixed inputs to our feature extraction blocks which can emulate the effects of MSDA. Moreover, each sample in a batch appears as part of M input tuples: it has to be fed to each of the M subnetworks once. As such, each of the batch’s sample is processed multiple times by the transformer’s feature extraction blocks while mixed with other samples.

Model	MSDA	BA	Accuracy (%)
ConViT			72.3 ± 0.3
ConViT	✓		75.3 ± 0.1
ConViT		✓	74.9 ± 0.3
ConViT	✓	✓	79.5 ± 0.1
MixViT-embedding			79.0 ± 0.5
MixViT-embedding	✓		77.3 ± 0.2
MixViT-embedding		✓	81.5 ± 0.4
MixViT-embedding	✓	✓	82.4 ± 0.1
MixViT-encoding			77.7 ± 0.1
MixViT-encoding	✓		75.8 ± 0.1
MixViT-encoding		✓	81.7 ± 0.3
MixViT-encoding	✓	✓	82.1 ± 0.1

Table 4.4. – CIFAR-100 accuracy with MSDA/BA.

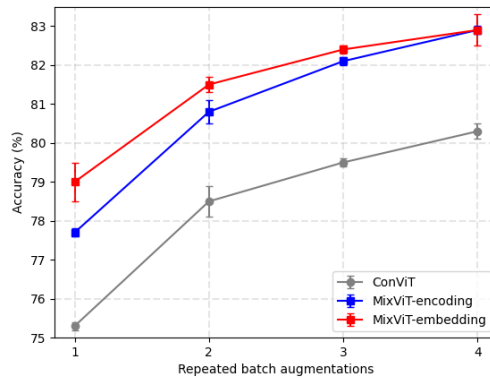


Figure 4.4. – MixViT performs well even without any batch repetitions on CIFAR-100.

Table 4.4 shows both MixViT variants prove much more resilient than their single-input single-output backbone on less regularized settings on CIFAR-100. It is particularly interesting to note that even with no batch augmentations and MSDA augmentations, our MixViT-embedding model matches the fully regularized variants of their single-input single-output backbone. While MSDA seems detrimental to training MixViT models with no batch augmentation to stabilize training, this is not a problem as MixViT models perform very well without MSDA inputs.

Furthermore, 4.4 demonstrates MixViT models behave much better than their single-input single-output counterpart with low amounts of batch augmentation. This is particularly important as batch augmentation linearly increases computations and has been a key fixture of vision transformer training since its introduction in DeiT (Touvron et al. 2021a). This linear cost has led some modern works

Models	# Params	CIFAR100	CIFAR10	TinyImageNet	ImageNet-100	
		Accuracy (%)	Accuracy (%)	Accuracy (%)	Accuracy (%)	
Reported results (from [.])						
Dytox ((Douillard et al. 2022), joint training)	10M	76.0	-	-	79.1	
CvT-7/4 ((Hassani et al. 2021))	3M	73.0	92.6	-	-	
CCT-7/3x1 ((Hassani et al. 2021))	4M	77.1	94.8	-	-	
DeiT-T (Touvron et al. 2021a) ((Z. Zhang et al. 2022))	5M	67.5	88.4	-	-	
PVT-T (W. Wang et al. 2021) ((Z. Zhang et al. 2022))	13M	69.6	90.5	-	-	
CaiT-XXS-24 (Touvron et al. 2021b) ((S. H. Lee et al. 2021))	9M	76.9	94.9	64.4	-	
SL-CaiT-XXS-24 ((S. H. Lee et al. 2021))	9M	80.3	95.8	67.1	-	
CvT-13 (Wu et al. 2021) ((Y. Liu et al. 2021))	20M	73.5	89.0	-	85.6	
CvT-13 + \mathcal{L}_{DrLoc} ((Y. Liu et al. 2021))	20M	74.5	90.3	-	86.1	
Our experiments						
Best Backbone	MixViT-embedding	12/26M	82.4 \pm 0.1	96.5 \pm 0.2	69.4 \pm 0.1	89.7 \pm 0.3
	MixViT-encoding	12/26M	82.1 \pm 0.1	96.5 \pm 0.2	70.9 \pm 0.2	89.3 \pm 0.2

Table 4.5. – Comparison of MixViT against similarly sized transformer results reported in the literature.

to dispense with it (Z. Liu et al. 2021), but the procedure remains widely used in the literature (Z. Zhang et al. 2022; W. Wang et al. 2021). As such, MixViT’s ability to provide strong performance with no batch augmentation could strongly simplify training schemes.

4.4.3.1 Comparison to similar transformers

In this section, we provide some context on the performance of our models with regard to the existing literature. We provide in Table 4.5 a comparison to the performance of normally trained transformers of similar sizes. Our results over 150 training epochs should allow a mostly fair comparison: vision transformers are typically trained for comparable or longer amounts of time (100 (Y. Liu et al. 2021; Touvron et al. 2021a) to 200 (Hassani et al. 2021) or more (Z. Zhang et al. 2022; Douillard et al. 2022) epochs) on CIFAR, TinyImageNet and ImageNet-100.

CIFAR-100 Results in Table 4.5 show our models largely outperform small transformers on the CIFAR datasets. In fact our MixViT-encoding’s performance of 83.4% accuracy on the extended 300 epochs training setting (see Appendix Section A.7.4) is - by a large margin - **the best transformer performance on CIFAR-100**. Indeed, the best scores in the literature are an 82.7% accuracy for a CCT-7/3x1 (Hassani et al. 2021) model trained for 5000 epochs and an 82.6% accuracy for a NesT-B (Z. Zhang et al. 2022) model with 90M parameters trained over 300 epochs.

TinyImageNet Table 4.5 this time demonstrates MixViT largely outperforms transformers on the more complex TinyImageNet dataset. Interestingly, the best reported transformers on the dataset are in line with our single-input single-output backbones. Therefore, the gains on the state-of-the-art can largely be attributed to MixViT’s benefits. Interestingly, to the best of our knowledge, our

MixViT-encoding variant trained with a CaiT-XS backbone showcases an accuracy of 70.9% which **improves on the previous state-of-the-art** held by MixMo’s ResNet-18-3 model (70.2% accuracy).

ImageNet-100 Table 4.5 shows MixViT outperforms the competing architectures for comparable training times. This suggests MixViT has little trouble accommodating larger 224×224 images.

4.4.4 Ablations

We verify in Table 4.6 the benefits of our source attribution design (Section 4.4.4.1), our inference scheme (Section 4.4.4.2) and our choice to mix inputs (Section 4.4.4.3) on CIFAR-100.

Method	# FLOPS	Accuracy (%)
Importance of source attribution		
Late source embedding	$\times 1.1$	82.4 ± 0.1
Late source encoding	$\times 1.1$	82.1 ± 0.1
No source attribution	$\times 1.1$	81.5 ± 0.2
Early input source embedding	$\times 2$	82.2 ± 0.2
Early source encoding	$\times 2$	82.2 ± 0.2
Inference scheme		
Separate inference	$\times 1.1$	82.4 ± 0.1
Sum inference	$\times 1$	81.9 ± 0.1
Alternating inference	$\times 1$	82.0 ± 0.4
Single subnetwork inference	$\times 1$	82.0 ± 0.2
Mixing scheme		
Binarized CutMix	$\times 1.1$	82.4 ± 0.1
All tokens	$\times 2$	81.5 ± 0.2

Table 4.6. – **Ablation on the importance of MixViT design choices.** Default settings are highlighted in grey for reference.

4.4.4.1 Effect of source attribution

Table 4.6 shows MIMO training seems to inherently benefit the model: networks trained without source attribution are significantly better than standard ConViT networks. While such models learn completely identical subnetworks (as there is no source attribution), having to learn features useful for the classification of multiple images simultaneously provides ample regularization.

We separate subnetworks in the last layers instead of separately encoding the inputs early in the network to avoid having to perform multiple forward passes for inference (see Section 4.3.1). We however find in Table 4.6 that displacing the separation back to early in the model slightly deteriorates performance: the model benefits from wholly sharing feature blocks between subnetworks.

4.4.4.2 Inference scheme

In stark contrast to our findings in Section 4.3.1, we find summing the 2 source attributed versions of each patch token at the end of our network before performing a single forward pass yields reasonably good results (Table 4.6). While these results are indeed worse than separate inferences, they are also a far cry from the disastrous MixMo ConViT results in Section 4.3.1. We posit late patch representations are therefore significantly more suited than early patch representations to this sort of combination. Interestingly, we observe similar performance when drawing at random which patches should be attributed to which subnetwork. It is also worth noting that the regularization induced by our MixViT scheme is strong enough for the performance of individual subnetworks to be reasonable on its own.

4.4.4.3 Input mixing scheme

We initially chose to compress the inputs into a single set of N tokens through a CutMix scheme to avoid the quadratic cost that would come with using all $M \times N$ tokens from the M inputs. Table 4.6 actually indicates that far from deteriorating performance, our compressed scheme performs better than a model trained with all tokens. We explain this by the fact the CutMix compression introduces additional regularization on the shared feature extractor and slightly more diverse subnetworks.

4.5 Conclusion

In this chapter, we deployed a MIMO MSDA formulation of Vision Transformers, which involves a particular influence of the attention mechanism with the separation of the subnetworks. Since the attention mechanism natively mimics the unmixing mechanism introduced at the end of the previous chapter, MIMO ViTs naturally share features. While this behavior is desirable to an extent, it means special care must be given to designing a MIMO ViT.

Our contribution in this chapter is the MixViT framework. The MixViT framework significantly modifies the original MIMO workflow to account for these

complications. Contrarily to previous approaches, we only separate the subnetworks in the last layers of the model. We do this by using the same encoder for both inputs and instead adding a source attribution mechanism in the last layers when we want the subnetworks to separate. As such, our subnetworks share the same early features by design.

Experiments show MixViT improves both ConViT and CaiT models on a large number of datasets (TinyImageNet, ImageNet-100, CIFAR), providing strong implicit regularization. As expected, the subnetworks trained by MixViT share features which explains the strong implicit regularization and the lesser reliance of MixViT on costly techniques like batch augmentation compared to single-input Vision Transformers. Contrarily to MixShare however, the subnetworks do not collapse to the same value and instead learn different functions on their own without any need for sensitive scheduling. This allows MixViT to completely outperform our CNN-based MixMo on the more complex TinyImageNet dataset. Indeed, MixViT does not need to use very wide networks since features can be shared between subnetworks which enables us to easily train on TinyImageNet.

Therefore, the attention mechanism and token representation of Vision Transformers does prove particularly suited to training subnetworks through MIMO MSDA. With the MixViT formulation, we recover the regularization effect characteristic of standard MSDA techniques. Additionally, sharing the first layers of the model between the subnetworks has a similar effect to initializing input encoders at the same value like in MixShare. As such, applying the MixViT formula - the source encoding variant - back to CNNs might solve the issues observed in MixShare.

CONCLUSION

In this thesis, we explore how combining contents from multiple inputs helps train stronger image classifiers. To this end, we challenge the standard practice of using soft hybrid labels in Mixing Samples Data Augmentation that has been part and parcel of the technique since its inception. While soft labels are used for good reason in MSDA, their ubiquitousness limits the potential of content mixing methods. As such, this thesis aims to explore alternative MSDA formulations and their implications.

After providing a quick summary of this thesis' contributions to the problem, we offer a few insights into possible future work.

5.1 Main contributions

The main contributions of this thesis study various manners in which altering soft label MSDA can lead to interesting behaviors. In [Chapter 2](#), we circumvent the issue of soft labels by generating in-class mixed samples that only contain the semantic content of one of the samples. We then directly challenge soft labels in [Chapter 3](#) by training subnetworks to the network to output separate predictions, one for each input in the mixed sample. Finally, we highlight in [Chapter 4](#) how the attention mechanism at the heart of the emerging Vision Transformers is fundamentally suited to training concurrent subnetworks from MSDA samples.

In-class mixing samples data augmentations We propose an alternative MSDA paradigm that creates in-class samples by combining the semantic content of one sample and non-semantic content of another in [Chapter 2](#). We use it to teach invariance to some non-semantic characteristics in semi-supervised learning.

Our first approach **SAMOSA** is an auto-encoding framework that identifies and adds some non-semantic variations to a base semantic image with the aid of an asymmetric generator inspired by style transfer techniques. SAMOSA generates hybrids that mostly match their semantic parent with some slight non-semantic variations like lighting. While this seems sufficient to improve classifiers, this does

not take full advantage of the non-semantic parent’s content. We then take the opposite view with **SciMix** by training a hybridizer that identifies and embeds semantic content into a base image’s non-semantic context using a powerful StyleGanV2 generator. Contrarily to SAMOSA, SciMix generates hybrids that inherit a lot of content from their non-semantic parent. While both methods provide significant gains when combined with classical semi-supervised methods like Mean Teach, SciMix seems to help more on lower label settings.

Mixing Samples Data Augmentations as compressed representations for Multi-Input Multi-output training We then study how subnetworks can be trained within the model by training the model to output separate predictions for each of the original inputs in a MSDA sample. By incorporating the MSDA formalism into the Multi-Input Multi-Output (MIMO) framework we obtain a new mixing augmentation paradigm that offers self-ensembling properties and is orthogonal to traditional mixing augmentations.

We first propose **MixMo**, which projects each original parent sample with a different convolutional layer before mixing them, and outputs two predictions - one for each input - instead of a soft label. In this formulation, using a sum operation to mix samples recovers the seminal MIMO framework. Interestingly, we find replacing this sum operation with a CutMix operation leads to strong performance improvements on its own while still retaining ensembling benefits. We then develop the **MixShare** framework to address MixMo’s difficulties on smaller networks. As this is likely due to the lack of feature sharing between MixMo subnetworks, we aim with MixShare to facilitate feature sharing. We introduce a novel unmixing mechanism that leads the subnetworks to share all features by “reversing” the mixing process used on the multiple inputs. This, along with proper initialization of the model and scheduling of the unmixing process, allows smaller networks to successfully train concurrent subnetworks.

Multi-Input Multi-Output MSDA, unmixing and attention mechanisms Finally, we look at the interaction between MIMO MSDA and Vision Transformers. This requires particular care as Transformers’ attention mechanism provides a native implementation of the unmixing mechanism introduced by MixShare, and our work on MixShare demonstrates feature sharing between subnetworks significantly complicates training.

As such, we propose the **MixViT** framework that significantly modifies the original MIMO workflow: we only separate the subnetworks in the last layers of the model. We do this by using the same encoder for both inputs and instead adding a source attribution mechanism in the last layers when we want the subnetworks to separate. As such, our subnetworks share the same early features by design

which is reminiscent of the initialization scheme we had to use in MixShare. The resulting framework yields strong improvements over the single-input single-output baseline for a minimal extra cost. Interestingly, MixViT strongly benefits from implicit regularization due to feature sharing between the subnetworks and only minimally benefits from subnetwork ensembling contrarily to MixMo.

5.2 Perspectives

At the start of this thesis, we asked how combining contents without resorting to soft labels could benefit image classification. We find selecting the contents we mix leads to a MSDA that helps train invariance in models for low label setting, while separating the predictions creates a paradigm closer to ensembling and multi-task learning. Both of these behaviors open new avenues when compared to standard MSDA's behavior as regularizer. We endeavor here to outline a few possible ways the three content mixing paradigms could further develop.

Regarding in-class MSDA SciMix heavily relies on a classifier to indicate semantic characteristics to embed in the non-semantic object. Classifiers are however typically limited in this aspect: logits for objects of the same class are mostly similar. As such, pretrained natural language image models like CLIP (Radford et al. 2021) might be significantly more suited to the problem compared to using an auxiliary classifier. Indeed, this sort of embedding provides significantly more nuance in expressing the content of an image, and has successfully been used to guide image modifications (Couairon et al. 2022).

Interestingly, in-class hybrids say interesting things about the auxiliary classifier they rely on in the generative process: the hybrids show what the classifier identifies as semantic in the image and dataset. As such, we could leverage this to help distill model knowledge (Ba and Caruana 2014; Y. Li et al. 2021) for compression purposes, and to interpret how the auxiliary classifier makes its decision (Zhiheng Li and Xu 2021).

Regarding MIMO MSDA Feature sharing in MIMO CNNs is sensitive and subject to significant tuning in the scheduling of unmixing. Interestingly, re-transposing the MixViT structure to CNNs might solve some of the more problematic issues. In any case, we have little to no control as to the degree of feature sharing between the subnetworks. Mixture of Experts (Masoudnia and Ebrahimpour 2014) are a well studied technique for CNNs and Transformers that would allow us to force some features to be dedicated to one of the subnetworks.

Sharing features between subnetworks has interesting implications regarding the genericity of the features trained. Indeed, a shared feature is likelier to serve multiple roles in the discriminative process: the feature might be needed to classify a duck for the first network at the same time it helps classify a boat in the other network. Similar ideas can be found in the traditional Multi-Task literature (Caruana 1997), but seeing different inputs at once significantly changes the training dynamics of such a process.

On MSDA in general Traditional MSDA themselves present several shortcomings like their usual inability to accommodate more than 2 inputs or the relative failure of saliency based mixing methods (Qin et al. 2020). The latter in particular is concerning as experimental results in Qin et al. 2020 suggest it is pointless to specifically mix semantic content from both inputs: non-semantic content should also be mixed according to the experiments. Interestingly, recent results (J.-N. Chen et al. 2022) show it is in fact better to recalibrate the mixed target to accommodate for the presence of non-semantic information in the mixing process.

It is our belief properly designed MSDA techniques are particularly suited to bridging the gap between different tasks or modalities. In fact, Continual learning (Zhizhong Li and Hoiem 2017), which can be understood as a form of asynchronous multi-task learning, has recently taken to use MixUp (Douillard et al. 2022; Madaan et al. 2021) with good success. We think mixing contents of different images can help models find proper optima for joint objectives that are such that the individual objectives have very different landscapes. After all, what better way to reconcile contradictory views than to see multiple instances at once?

Combining contents beyond MSDA This thesis has restricted itself to combining contents in data augmentations, in part with the understanding data augmentation can be used in turn to encapsulate a number of desirable behaviors (Y. Li et al. 2021; Zhiheng Li and Xu 2021). Nevertheless, understanding how to combine contents and exploit this combination has much broader implications. Indeed, the issue lies at the heart of many larger issues we have only alluded to with our MSDA approaches. For instance, combining different types of (often conflicting) knowledge within the network is of particular importance in fields like Multi-task learning (Caruana 1997) and Continual learning (Zhizhong Li and Hoiem 2017). This is particularly so in the latter where methods are starting to combine contents from different subnetworks (Yan et al. 2021). Similarly, combining contents from a stronger network trained with privileged information (Vapnik and Vashist 2009) with contents from a standard networks will require particular care.

Most remarkably however, combining contents has long been a central problem in multimodal learning (Ramachandram and Taylor 2017) where very different types of contents like text and images must be fused. This problem is coming to the forefront as we strive to train more and more general models if the recent rise of foundation models (Bommasani et al. 2021) is any indication. If neural networks are to take a larger place in our future, then it is perhaps time for them to combine information as we ourselves do everyday.

BIBLIOGRAPHY

- Aksela, Matti (2003). "Comparison of classifier selection methods for improving committee performance". In: *MCS* (cit. on p. 65).
- Arbelaez, Pablo, Michael Maire, Charless Fowlkes, and Jitendra Malik (2011). "Contour Detection and Hierarchical Image Segmentation". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (cit. on p. 141).
- Arjovsky, Martin, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz (2019). "Invariant risk minimization". In: *arXiv preprint library* (cit. on p. 6).
- Ashukha, Arsenii, Alexander Lyzhov, Dmitry Molchanov, and Dmitry Vetrov (2020). "Pitfalls of In-Domain Uncertainty Estimation and Ensembling in Deep Learning". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 52, 60, 126, 127, 129).
- Ba, Jimmy and Rich Caruana (2014). "Do deep nets really need to be deep?" In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 99).
- Babbage, Henry P (1982). "Babbage's analytical engine". In: *The Origins of Digital Computers* (cit. on p. 1).
- Baek, Kyungjune, Yunje Choi, Youngjung Uh, Jaejun Yoo, and Hyunjung Shim (2021). "Rethinking the Truly Unsupervised Image-to-Image Translation". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on p. 20).
- Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural machine translation by jointly learning to align and translate". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 80).
- Berthelot, David, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel (2020). "ReMixMatch: Semi-Supervised Learning with Distribution Matching and Augmentation Anchoring". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 17).
- Berthelot, David, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel (2019). "MixMatch: A Holistic Approach to Semi-Supervised Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 17, 24, 36–39, 117–119, 123, 124, 134).
- Bishop, Christopher (2006). *Pattern Recognition and Machine Learning*. Springer (cit. on pp. 1–3, 5).
- Bommasani, Rishi, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. (2021). "On the opportunities and risks of foundation models". In: *Technical report* (cit. on p. 101).

- Boyd, Stephen, Stephen P Boyd, and Lieven Vandenbergh (2004). *Convex optimization*. Cambridge university press (cit. on p. 3).
- Breiman, Leo (1996). “Bagging predictors”. In: *Machine learning* (cit. on p. 56).
- Carratino, Luigi, Moustapha Cissé, Rodolphe Jenatton, and Jean-Philippe Vert (2020). “On Mixup Regularization”. In: *arXiv preprint library* (cit. on pp. 8, 14, 56, 60).
- Caruana, Rich (1997). “Multitask learning”. In: *Machine learning* (cit. on pp. 58, 100).
- Chapelle, Olivier, Bernhard Schlkopf, and Alexander Zien (2006). *Semi-Supervised Learning*. The MIT Press (cit. on pp. 6, 16, 18).
- Chen, Jie-Neng, Shuyang Sun, Ju He, Philip H.S. Torr, Alan Yuille, and Song Bai (2022). “TransMix: Attend To Mix for Vision Transformers”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 50, 100).
- Chen, John, Samarth Sinha, and Anastasios Kyrillidis (2020). “ImCLR: Implicit Contrastive Learning for Image Classification”. In: *arXiv preprint library* (cit. on p. 66).
- Chen, Zhao, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich (2018). “Gradnorm: Gradient normalization for adaptive loss balancing in deep multi-task networks”. In: *International Conference on Machine Learning (ICML)* (cit. on p. 58).
- Cheng, Jianpeng, Li Dong, and Mirella Lapata (2016). “Long short-term memory-networks for machine reading”. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)* (cit. on p. 81).
- Chirkova, Nadezhda, Ekaterina Lobacheva, and Dmitry P. Vetrov (2020). “Deep Ensembles on a Fixed Memory Budget: One Wide Network or Several Thinner Ones?” In: *arXiv preprint library* (cit. on pp. 63, 129, 153).
- Chrabaszcz, Patryk, Ilya Loshchilov, and Frank Hutter (2017). “A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets”. In: *arXiv preprint library* (cit. on pp. 45, 59, 67, 89, 126).
- Coates, Adam, Andrew Ng, and Honglak Lee (2011). “An Analysis of Single-Layer Networks in Unsupervised Feature Learning”. In: *International Conference on Artificial Intelligence and Statistics (AISTATS)* (cit. on p. 45).
- Couairon, Guillaume, Asya Grechka, Jakob Verbeek, Holger Schwenk, and Matthieu Cord (2022). “FlexIT: Towards Flexible Semantic Image Translation”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 99).
- Cubuk, Ekin D., Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le (2019). “AutoAugment: Learning Augmentation Strategies From Data”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 90).

- Cybenko, George (1989). "Approximation by superpositions of a sigmoidal function". In: *Mathematics of control, signals and systems* (cit. on p. 4).
- Cygert, Sebastian and Andrzej Czyżewski (2022). "Robust Object Detection with Multi-input Multi-output Faster R-CNN". In: *Image Analysis and Processing (ICIAP)* (cit. on p. 80).
- D'Ascoli, Stéphane, Hugo Touvron, Matthew L Leavitt, Ari S Morcos, Giulio Biroli, and Levent Sagun (2021). "ConViT: Improving Vision Transformers with Soft Convolutional Inductive Biases". In: *International Conference on Machine Learning (ICML)* (cit. on pp. 81, 82, 84–86, 89, 131, 160, 162).
- Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei (2009). "ImageNet: A Large-Scale Hierarchical Image Database". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 126, 155).
- Deng, Jinhong, Wen Li, Yuhua Chen, and Lixin Duan (2021). "Unbiased mean teacher for cross-domain object detection". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 17).
- DeVries, Terrance and Graham W Taylor (2017). "Improved regularization of convolutional neural networks with cutout". In: *arXiv preprint library* (cit. on p. 149).
- Devroye, Luc, László Györfi, and Gábor Lugosi (2013). *A probabilistic theory of pattern recognition*. Springer Science & Business Media (cit. on pp. 1, 2).
- Dietterich, Thomas G (2000). "Ensemble methods in machine learning". In: *MCS* (cit. on p. 51).
- Dosovitskiy, Alexey, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby (2021). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations* (cit. on pp. 4, 81).
- Douillard, Arthur, Alexandre Ramé, Guillaume Couairon, and Matthieu Cord (2022). "DyTox: Transformers for Continual Learning with DYnamic TOken eXpansion". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 93, 100, 131).
- Dusenberry, Michael, Ghassen Jerfel, Yeming Wen, Yian Ma, Jasper Snoek, Katherine Heller, Balaji Lakshminarayanan, and Dustin Tran (2020). "Efficient and scalable bayesian neural nets with rank-1 factors". In: *International Conference on Machine Learning (ICML)* (cit. on p. 52).
- Faramarzi, Mojtaba, Mohammad Amini, Akilesh Badrinaaraayanan, Vikas Verma, and Sarath Chandar (2020). "PatchUp: A Regularization Technique for Convolutional Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)* (cit. on pp. 59, 64, 148).

- Frankle, Jonathan and Michael Carbin (2019). “The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks”. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 53).
- French, Geoff, Timo Aila, Samuli Laine, Michal Mackiewicz, and Graham Finlayson (2020a). “Semi-supervised semantic segmentation needs strong, high-dimensional perturbations”. In: *Proceedings of the British Machine Vision Conference (BMVC)* (cit. on pp. 64, 149).
- French, Geoff, Avital Oliver, and Tim Salimans (2020b). “Milking CowMask for Semi-Supervised Image Classification”. In: *arXiv preprint library* (cit. on pp. 64, 149).
- Gal, Yarin and Zoubin Ghahramani (2016). “Dropout as a bayesian approximation: Representing model uncertainty in deep learning”. In: *International Conference on Machine Learning (ICML)* (cit. on p. 52).
- Ganin, Yaroslav and Victor Lempitsky (2015). “Unsupervised Domain Adaptation by Backpropagation”. In: *International Conference on Machine Learning (ICML)* (cit. on p. 141).
- Gao, Yuan, Zixiang Cai, and Lei Yu (2019). “Intra-Ensemble in Neural Networks”. In: *arXiv preprint library* (cit. on p. 53).
- Gatys, Leon A, Alexander S Ecker, and Matthias Bethge (2016). “Image style transfer using convolutional neural networks”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 19).
- Gebru, Timnit (2019). “Oxford handbook on AI ethics book chapter on race and gender”. In: *arXiv preprint library* (cit. on p. 2).
- Ghiasi, Golnaz, Tsung-Yi Lin, and Quoc V Le (2018). “Dropblock: A regularization method for convolutional networks”. In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 148).
- Gödel, Kurt (1931). “Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I”. In: *Monatshefte für mathematik und physik* (cit. on p. 1).
- Gontijo-Lopes, Raphael, Sylvia J. Smullin, Ekin D. Cubuk, and Ethan Dyer (2021). “Affinity and Diversity: Quantifying Mechanisms of Data Augmentation”. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 7, 38, 60, 118).
- Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press (cit. on pp. 1, 3–5).
- Granzio, Diego, Stefan Zohren, and Stephen Roberts (2020). “Learning rates as a function of batch size: A random matrix theory approach to neural network training”. In: *arXiv preprint library* (cit. on p. 132).
- Guo, Chuan, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger (2017). “On Calibration of Modern Neural Networks”. In: *International Conference on Machine Learning (ICML)* (cit. on pp. 60, 129).

- Hansen, Lars Kai and Peter Salamon (1990). “Neural network ensembles”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (cit. on p. 51).
- Harris, Ethan, Antonia Marcu, Matthew Painter, Mahesan Niranjan, Adam Prügell-Bennett, and Jonathon Hare (2020). “FMix: Enhancing Mixed Sample Data Augmentation”. In: *arXiv preprint library* (cit. on pp. 64, 148).
- Hassani, Ali, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi (2021). “Escaping the Big Data Paradigm with Compact Transformers”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshop* (cit. on p. 93).
- Havasi, Marton, Rodolphe Jenatton, Stanislav Fort, Jeremiah Liu, Jasper Roland Snoek, Balaji Lakshminarayanan, Andrew Mingbo Dai, and Dustin Tran (2021). “Training independent subnetworks for robust prediction”. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 9, 50–55, 60, 66, 69, 75, 84, 85, 126, 127, 129, 155, 156, 159).
- He, Kaiming, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick (2020). “Momentum Contrast for Unsupervised Visual Representation Learning”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 29, 35).
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016a). “Deep residual learning for image recognition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 127).
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016b). “Identity Mappings in Deep Residual Networks”. In: *Proceedings of the IEEE European Conference on Computer Vision (ECCV)* (cit. on pp. 2, 6, 27, 32, 67, 91).
- Hendrycks, Dan and Thomas Dietterich (2019). “Benchmarking Neural Network Robustness to Common Corruptions and Perturbations”. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 62, 127).
- Hendrycks, Dan, Norman Mu, Ekin Dogus Cubuk, Barret Zoph, Justin Gilmer, and Balaji Lakshminarayanan (2020). “AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty”. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 62).
- Higgins, Irina, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner (2017). “Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework”. In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 19, 25).
- Hoffer, Elad, Tal Ben-Nun, Itay Hubara, Niv Giladi, Torsten Hoefler, and Daniel Soudry (2020). “Augment Your Batch: Improving Generalization Through Instance Repetition”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 61, 90, 92, 127).

- Huang, Gao, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E. Hopcroft, and Kilian Q Weinberger (2017). "Snapshot Ensembles: Train 1, get M for free". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 51).
- Huang, Gao, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger (2016). "Deep Networks with Stochastic Depth". In: *Proceedings of the IEEE European Conference on Computer Vision (ECCV)* (cit. on p. 90).
- Huang, Xun and Serge Belongie (2017). "Arbitrary Style Transfer in Real-Time With Adaptive Instance Normalization". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on pp. 19, 20, 22, 25, 26, 44).
- Huang, Xun, Ming-Yu Liu, Serge Belongie, and Jan Kautz (2018). "Multimodal Un-supervised Image-to-image Translation". In: *Proceedings of the IEEE European Conference on Computer Vision (ECCV)* (cit. on pp. 20, 26).
- Inoue, Hiroshi (2018). "Data Augmentation By Pairing Samples for Images Classification". In: *arXiv preprint library* (cit. on p. 44).
- Iscen, Ahmet, Giorgos Tolias, Yannis Avrithis, and Ondrej Chum (2019). "Label Propagation for Deep Semi-Supervised Learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 16, 18).
- Jiang, Zi-Hang, Qibin Hou, Li Yuan, Daquan Zhou, Yujun Shi, Xiaojie Jin, Anran Wang, and Jiashi Feng (2021). "All Tokens Matter: Token Labeling for Training Better Vision Transformers". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 131).
- Jordan, Michael Irwin (1999). *Learning in graphical models*. MIT press (cit. on pp. 1, 3).
- Karras, Tero, Samuli Laine, and Timo Aila (2019). "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 20, 22, 25, 26).
- Karras, Tero, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila (2020). "Analyzing and Improving the Image Quality of StyleGAN". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 20, 22, 36, 46, 124).
- Kim, Byoungjip, Jinho Choo, Yeong-Dae Kwon, Seongho Joe, Seungjai Min, and Youngjune Gwon (2021). "SelfMatch: Combining Contrastive Self-Supervision and Consistency for Semi-Supervised Learning". In: (cit. on p. 16).
- Kim, Jang-Hyun, Wonho Choo, and Hyun Oh Song (2020). "Puzzle mix: Exploiting saliency and local statistics for optimal mixup". In: *International Conference on Machine Learning (ICML)* (cit. on pp. 62, 67, 126, 127).
- Kim, JangHyun, Wonho Choo, Hosan Jeong, and Hyun Oh Song (2021). "Co-Mixup: Saliency Guided Joint Mixup with Supermodular Diversity". In: *Pro-*

- ceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 150).
- Kingma, Diederik P and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 2, 6).
- Krizhevsky, Alex and Geoffrey Hinton (2009). *Learning multiple layers of features from tiny images*. Tech. rep. (cit. on pp. 36, 59, 70, 84, 89, 122, 126).
- Krogh, Anders and John Hertz (1991). "A simple weight decay can improve generalization". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 6).
- Laine, Samuli and Timo Aila (2017). "Temporal Ensembling for Semi-Supervised Learning". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 16, 17).
- Lakshminarayanan, Balaji, Alexander Pritzel, and Charles Blundell (2017). "Simple and scalable predictive uncertainty estimation using deep ensembles". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 51, 61, 126).
- Lecun, Yann. (1998). "THE MNIST DATABASE of handwritten digits". In: <http://yann.lecun.com/exdb> (cit. on p. 141).
- Lecun, Yann, J. S. Denker, Sara A. Solla, R. E. Howard, and L.D. Jackel (1990). "Optimal brain damage". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 53).
- Lee, Seung Hoon, Seunghyun Lee, and Byung Cheol Song (2021). "Vision Transformer for Small-Size Datasets". In: *arXiv preprint library* (cit. on pp. 90, 93, 132).
- Lee, Stefan, Senthil Purushwalkam, Michael Cogswell, David J. Crandall, and Dhruv Batra (2015). "Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks". In: *arXiv preprint library* (cit. on p. 52).
- Li, Boyi, Felix Wu, Ser-Nam Lim, Serge Belongie, and Kilian Q. Weinberger (2021). "On Feature Normalization and Data Augmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 59).
- Li, Chongxuan, Taufik Xu, Jun Zhu, and Bo Zhang (2017). "Triple Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 16).
- Li, Hao, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf (2017). "Pruning Filters for Efficient ConvNets". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 53, 59, 152).
- Li, Junnan, Caiming Xiong, and Steven C.H. Hoi (2021). "CoMatch: Semi-Supervised Learning With Contrastive Graph Regularization". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on pp. 18, 36).

- Li, Yuhang, Feng Zhu, Ruihao Gong, Mingzhu Shen, Xin Dong, Fengwei Yu, Shaoqing Lu, and Shi Gu (2021). "Mixmix: All you need for data-free compression are feature and data mixing". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on pp. 99, 100).
- Li, Zhiheng and Chenliang Xu (2021). "Discover the Unknown Biased Attribute of an Image Classifier". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on pp. 99, 100).
- Li, Zhizhong and Derek Hoiem (2017). "Learning without forgetting". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (cit. on p. 100).
- Lin, Tsung-Yi, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár (2017). "Focal loss for dense object detection". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on p. 58).
- Liu, Ming-Yu, Thomas Breuel, and Jan Kautz (2017). "Unsupervised Image-to-Image Translation Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 20).
- Liu, Ming-Yu, Xun Huang, Arun Mallya, Tero Karras, Timo Aila, Jaakko Lehtinen, and Jan Kautz (2019). "Few-Shot Unsupervised Image-to-Image Translation". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 20).
- Liu, Yahui, Enver Sangineto, Wei Bi, Nicu Sebe, Bruno Lepri, and Marco De Nadai (2021). "Efficient Training of Visual Transformers with Small Datasets". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 93).
- Liu, Yen-Cheng, Chih-Yao Ma, and Zsolt Kira (2022). "Unbiased Teacher v2: Semi-Supervised Object Detection for Anchor-Free and Anchor-Based Detectors". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 19).
- Liu, Ze, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo (2021). "Swin Transformer: Hierarchical Vision Transformer Using Shifted Windows". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on pp. 82, 93).
- Lobacheva, Ekaterina, Nadezhda Chirkova, Maxim Kodryan, and Dmitry P Vetrov (2020). "On Power Laws in Deep Ensembles". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 51, 60, 63, 64, 129, 153, 154).
- Loshchilov, Ilya and Frank Hutter (2019). "Decoupled Weight Decay Regularization". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 90).
- Luo, Yucen, Jun Zhu, Mengxi Li, Yong Ren, and Bo Zhang (2018). "Smooth Neighbors on Teacher Graphs for Semi-Supervised Learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 16).

- Madaan, Divyam, Jaehong Yoon, Yuanchun Li, Yunxin Liu, and Sung Ju Hwang (2021). "Representational continuity for unsupervised continual learning". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 100).
- Malach, Eran, Gilad Yehudai, Shai Shalev-Schwartz, and Ohad Shamir (2020). "Proving the lottery ticket hypothesis: Pruning is all you need". In: *International Conference on Machine Learning (ICML)* (cit. on p. 52).
- Masoudnia, Saeed and Reza Ebrahimpour (2014). "Mixture of experts: a literature survey". In: *Artificial Intelligence Review* (cit. on p. 99).
- McCarthy, John, Marvin L Minsky, Nathaniel Rochester, and Claude E Shannon (1956). "A proposal for the dartmouth summer research project on artificial intelligence". In: *Dartmouth Summer Research Project on Artificial Intelligence* (cit. on p. 1).
- Miyato, T., S. Maeda, M. Koyama, and S. Ishii (2019). "Virtual Adversarial Training: A Regularization Method for Supervised and Semi-Supervised Learning". In: (cit. on p. 16).
- Molchanov, Pavlo, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz (2017). "Pruning convolutional neural networks for resource efficient transfer learning". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 53).
- Naeini, Mahdi Pakdaman, Gregory Cooper, and Milos Hauskrecht (2015). "Obtaining well calibrated probabilities using bayesian binning". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)* (cit. on p. 60).
- Nakkiran, Preetum, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever (2021). "Deep double descent: Where bigger models and more data hurt". In: *Journal of Statistical Mechanics: Theory and Experiment* (cit. on p. 2).
- Neal, Brady, Sarthak Mittal, Aristide Baratin, Vinayak Tantia, Matthew Scicluna, Simon Lacoste-Julien, and Ioannis Mitliagkas (2018). "A Modern Take on the Bias-Variance Tradeoff in Neural Networks". In: *arXiv preprint library* (cit. on pp. 64, 154).
- Netzer, Yuval, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng (2011). "Reading digits in natural images with unsupervised feature learning". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 36, 122).
- Nilsson, Nils J (1982). *Principles of artificial intelligence*. Springer Science & Business Media (cit. on p. 1).
- Nixon, Jeremy, Michael W Dusenberry, Linchuan Zhang, Ghassen Jerfel, and Dustin Tran (2019). "Measuring Calibration in Deep Learning." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshop* (cit. on p. 129).

- Pang, Tianyu, Kun Xu, Chao Du, Ning Chen, and Jun Zhu (2019). "Improving Adversarial Robustness via Promoting Ensemble Diversity". In: *International Conference on Machine Learning (ICML)* (cit. on p. 75).
- Park, Taesung, Jun-Yan Zhu, Oliver Wang, Jingwan Lu, Eli Shechtman, Alexei Efros, and Richard Zhang (2020). "Swapping Autoencoder for Deep Image Manipulation". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 21, 22, 32).
- Paszke, Adam, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala (2019). "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 60, 130).
- Pensia, Ankit, Shashank Rajput, Alliot Nagle, Harit Vishwakarma, and Dimitris Papailiopoulos (2020). "Optimal Lottery Tickets via SubsetSum: Logarithmic Over-Parameterization is Sufficient". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 53).
- Perez, Luis and Jason Wang (2017). "The Effectiveness of Data Augmentation in Image Classification Using Deep Learning". In: *arXiv preprint library* (cit. on pp. 7, 38, 118).
- Qin, Jie, Jiemin Fang, Qian Zhang, Wenyu Liu, Xingang Wang, and Xinggang Wang (2020). "ResizeMix: Mixing Data with Preserved Object Information and True Labels". In: *arXiv preprint library* (cit. on pp. 100, 127).
- Radford, Alec, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. (2021). "Learning transferable visual models from natural language supervision". In: *International Conference on Machine Learning (ICML)* (cit. on p. 99).
- Ramachandram, Dhanesh and Graham W Taylor (2017). "Deep multimodal learning: A survey on recent advances and trends". In: *IEEE signal processing magazine* (cit. on p. 101).
- Rame, Alexandre and Matthieu Cord (2021). "DICE: Diversity in Deep Ensembles via Conditional Redundancy Adversarial Estimation". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 60, 65, 75, 129).
- Rame, Alexandre, Remy Sun, and Matthieu Cord (2021). "MixMo: Mixing Multiple Inputs for Multiple Outputs via Deep Subnetworks". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on pp. 52, 69, 70, 73, 75, 84–87, 91, 129, 156, 158–160, 162).
- Rizve, Mamshad Nayeem, Kevin Duarte, Yogesh S Rawat, and Mubarak Shah (2021). "In Defense of Pseudo-Labeling: An Uncertainty-Aware Pseudo-label

- Selection Framework for Semi-Supervised Learning". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 16).
- Robert, Thomas, Nicolas Thome, and Matthieu Cord (2018). "HybridNet: Classification and Reconstruction Cooperation for Semi-Supervised Learning". In: *Proceedings of the IEEE European Conference on Computer Vision (ECCV)* (cit. on pp. 17, 26, 118, 119, 122, 123).
- Robert, Thomas, Nicolas Thome, and Matthieu Cord (2019). "DualDis: Dual-Branch Disentangling with Adversarial Learning". In: *arXiv preprint library* (cit. on p. 19).
- Shang, Lifeng, Zhengdong Lu, and Hang Li (2015). "Neural responding machine for short-text conversation". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL Short Papers)* (cit. on p. 1).
- Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* (cit. on p. 1).
- Silver, David, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* (cit. on p. 1).
- Soflaei, Masoumeh, Hongyu Guo, Ali Al-Bashabsheh, Yongyi Mao, and Richong Zhang (2020). "Aggregated Learning: A Vector-Quantization Approach to Learning Neural Network Classifiers". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)* (cit. on p. 53).
- Sohn, Kihyuk, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li (2020). "Fix-Match: Simplifying Semi-Supervised Learning with Consistency and Confidence". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 16, 18, 21, 24, 32, 35, 36, 122, 124).
- Stokes, Jonathan M, Kevin Yang, Kyle Swanson, Wengong Jin, Andres Cubillos-Ruiz, Nina M Donghia, Craig R MacNair, Shawn French, Lindsey A Carfrae, Zohar Bloom-Ackermann, et al. (2020). "A deep learning approach to antibiotic discovery". In: *Cell* (cit. on p. 1).
- Summers, Cecilia and Michael J Dinneen (2019). "Improved Mixed-Example Data Augmentation". In: *Proceedings of the IEEE Winter Conference on Application of Computer Vision (WACV)* (cit. on pp. 64, 148).
- Sun, Remy, Alexandre Rame, Clement Masson, Nicolas Thome, and Matthieu Cord (2022). "Towards efficient feature sharing in MIMO architectures". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshop* (cit. on pp. 80, 91).

- Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2016). "Rethinking the Inception Architecture for Computer Vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on pp. 6, 9, 35, 90).
- Szeliski, Richard (2022). *Computer vision: algorithms and applications*. Springer Science & Business Media (cit. on p. 3).
- Tarvainen, Antti and Harri Valpola (2017). "Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 16, 17, 21, 24, 32, 35–37, 39, 42, 44, 118, 119, 123, 124).
- Thulasidasan, Sunil, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak (2019). "On mixup training: Improved calibration and predictive uncertainty for deep neural networks". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 14).
- Tian, Yonglong, Dilip Krishnan, and Phillip Isola (2020). "Contrastive multiview coding". In: *Proceedings of the IEEE European Conference on Computer Vision (ECCV)* (cit. on p. 89).
- Tishby, Naftali (2001). "The information bottleneck method". In: *Allerton Conference on Communication, Control and Computation* (cit. on p. 53).
- Töscher, Andreas and Michael Jahrer (2009). "The BigChaos Solution to the Netflix Grand Prize". In: (cit. on p. 1).
- Touvron, Hugo, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Herve Jegou (2021a). "Training data-efficient image transformers and distillation through attention". In: *International Conference on Machine Learning (ICML)* (cit. on pp. 80, 90, 92, 93, 132).
- Touvron, Hugo, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou (2021b). "Going Deeper With Image Transformers". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on pp. 81, 82, 84, 86, 89, 90, 93, 131).
- Turing, Alan Mathison et al. (1936). "On computable numbers, with an application to the Entscheidungsproblem". In: *Journal of Math* (cit. on p. 1).
- Vapnik, Vladimir and Akshay Vashist (2009). "A new learning paradigm: Learning using privileged information". In: *Neural networks* (cit. on p. 100).
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is all you need". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 1, 2, 6, 81).
- Veit, Andreas, Michael Wilber, and Serge Belongie (2016). "Residual networks behave like ensembles of relatively shallow networks". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on p. 53).

- Verma, Vikas, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio (2019a). "Manifold Mixup: Better Representations by Interpolating Hidden States". In: *International Conference on Machine Learning (ICML)* (cit. on pp. 56, 59, 61, 127, 155).
- Verma, Vikas, Alex Lamb, Juho Kannala, Yoshua Bengio, and David Lopez-Paz (2019b). "Interpolation Consistency Training for Semi-supervised Learning". In: *International Joint Conference on Artificial Intelligence* (cit. on p. 17).
- Wang, Kaiping, Bo Zhan, Chen Zu, Xi Wu, Jiliu Zhou, Luping Zhou, and Yan Wang (2022). "Semi-supervised medical image segmentation via a tripled-uncertainty guided mean teacher model with contrastive learning". In: *Medical Image Analysis* 79 (cit. on p. 17).
- Wang, Wenhai, Enze Xie, Xiang Li, Deng-Ping Fan, Kaitao Song, Ding Liang, Tong Lu, Ping Luo, and Ling Shao (2021). "Pyramid Vision Transformer: A Versatile Backbone for Dense Prediction Without Convolutions". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on p. 93).
- Welinder, P., S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona (2010). *Caltech-UCSD Birds 200*. Tech. rep. (cit. on p. 45).
- Welling, Max and Yee W Teh (2011). "Bayesian learning via stochastic gradient Langevin dynamics". In: *International Conference on Machine Learning (ICML)* (cit. on p. 6).
- Wen, Yeming, Ghassen Jerfel, Rafael Muller, Michael W Dusenberry, Jasper Snoek, Balaji Lakshminarayanan, and Dustin Tran (2021). "Combining Ensembles and Data Augmentation Can Harm Your Calibration". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on p. 129).
- Wu, Haiping, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang (2021). "CvT: Introducing Convolutions to Vision Transformers". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on p. 93).
- Yan, Shipeng, Jiangwei Xie, and Xuming He (2021). "Der: Dynamically expandable representation for class incremental learning". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 100).
- Yang, Taojiannan, Sijie Zhu, and Chen Chen (2020). "GradAug: A New Regularization Method for Deep Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 60, 127).
- Yang, Yanchao and Stefano Soatto (2019). "CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on pp. 7, 8, 44, 54, 61, 64, 74, 86, 90, 92, 126, 127, 129, 132, 149).
- Yang, Yanchao and Stefano Soatto (2020). "FDA: Fourier Domain Adaptation for Semantic Segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 44).

- Zagoruyko, Sergey and Nikos Komodakis (2016). "Wide Residual Networks". In: *Proceedings of the British Machine Vision Conference (BMVC)* (cit. on pp. 32, 36, 60, 70, 119, 124, 162).
- Zhai, Xiaohua, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer (2022). "Scaling vision transformers". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (cit. on p. 82).
- Zhai, Xiaohua, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer (2019). "S4L: Self-Supervised Semi-Supervised Learning". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on p. 16).
- Zhang, Bowen, Yidong Wang, Wenxin Hou, Hao Wu, Jindong Wang, Manabu Okumura, and Takahiro Shinozaki (2021). "FlexMatch: Boosting Semi-Supervised Learning with Curriculum Pseudo Labeling". In: *Advances in Neural Information Processing Systems (NeurIPS)* (cit. on pp. 18, 36).
- Zhang, Hongyi, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz (2018). "mixup: Beyond empirical risk minimization". In: *Proceedings of the International Conference on Learning Representations (ICLR)* (cit. on pp. 7, 8, 14, 44, 49, 54, 56, 61, 90, 92, 132, 148).
- Zhang, Zizhao, Han Zhang, Long Zhao, Ting Chen, and Tomas Pfister (2022). "Aggregating Nested Transformers". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)* (cit. on pp. 90, 93).
- Zhong, Zhun, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang (2020). "Random Erasing Data Augmentation". In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)* (cit. on p. 90).
- Zhu, Jun-Yan, Taesung Park, Phillip Isola, and Alexei A Efros (2017). "Unpaired image-to-image translation using cycle-consistent adversarial networks". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)* (cit. on p. 20).

APPENDIX

In this Appendix, we offer additional material to contextualize the work presented in this thesis. We start with an inventory of the experimental settings (Section A.1) and pseudo-code algorithms (Section A.2) for the five papers presented in this thesis. Afterwards, we give additional details and material for SAMOSA in Section A.3, SciMix in Section A.4, MixMo in Section A.5, MixShare in Section A.6 and MixViT in Section A.7.

A.1 Experimental details

We outline experimental details for SAMOSA in Section A.1.1, SciMix in Section A.1.2, MixMo in Section A.1.3, MixShare in Section A.1.4 and MixViT in Section A.1.5.

A.1.1 Experimental details for SAMOSA experiments

Implementation was coded in python3 using the pytorch framework, parameters follow their default pytorch value unless specified otherwise. The code base was largely adapted from <https://github.com/ThomasRobertFr/hybridnet> for the Mean Teacher implementation and general code backbone, MixMatch implementation was adapted almost directly from <https://github.com/YU1ut/MixMatch-pytorch> (with some modifications to follow (Berthelot et al. 2019) more closely).

For contractual reasons it is not possible to provide the experimental code, but code snippets can be provided on demand to illustrate the more important parts of the algorithm and experimental setting.

Most experiments were run on a *nationwide* computation cluster with 261 Nodes of 2 Intel Cascade Lake 6248 CPUs and 4 Nvidia V100 SXM2 32 Go GPUs operating RedHat version 8.1 with conda 4.8.0. The SVHN Mean Teacher based experiments were run on another workstation with an Intel Xeon Silver 4108 CPU and 4 Nvidia Titan Xp 12 Go GPUs operating Debian Bullseye.

General Setting We operate on a standard WideResNet-28-2 (1.5M parameters) which is widely used in the Semi-Supervised Learning literature as a base model ($E_c \circ C$). E_r follows the same architecture as E_c with an additional final linear layer and softmax activation to obtain activation gates. The skeleton of D follows an inverted 13-layer 4-4-4 CNN architecture, with D_{pre} being a 4-4 block and D_{post} being made up of the last 4 block and final convolution. Downsampling/Upsampling convolutions are stabilized with an additional Batch Normalization layer in MixMatch experiments. Weights were initialized according to default pytorch layer initializations (Kaiming initializations). Batch Normalization layers followed standard pytorch configuration (momentum = 0.1). Gradients are clipped to unit norm to stabilize training, optimizers details vary from setting to setting (detailed below).

Samples are randomly flipped horizontally (only for CIFAR10) and shifted by up to 4 pixels both horizontally and vertically with reflect padding. The resulting augmented samples are then standardized channel-wise according to train set statistics. No holdout validation set is kept for either dataset but hyper-parameters are mostly directly adapted from (Robert et al. 2018; Berthelot et al. 2019).

Reduced settings parameters for the Case Study were identical to normal settings outlined below with the sole differences being that they were run on only one gpu with halved learning rates and batch sizes.

Mean Teacher Based SAMOSA on CIFAR/SVHN/MNISTM The model is trained over 2 gpus using a SGD optimizer with weight decay $5e - 4$, Nesterov momentum 0.9 and cosine learning rate (base 0.2 learning rate) for 300/150/300 epochs over the unlabeled samples (the rate is set to fully decrease over 350/200 epochs), with the reconstructive modules E_r and D optimized by a secondary SGD optimizer following the same parameters.

After each training and subsequent augmentation step, the model learning rate is reset and training resumes (leading to training for an overall 900 epochs). $\lambda_{recons} = 0.25$, $\lambda_{SAMOSA} = 0.5/0.1/0.1$ during training, consistency targets are optimized with weight $\lambda_{cons} = 300$, with batches of 124/30 labeled samples and 388/500/500 unlabeled samples (similarly to (Robert et al. 2018)). In the second and third training passes, the model is only optimized over the augmented dataset from epoch 150/50/150 to 250/100/250 of each cycle, and is optimized over the true dataset for the remaining epochs (following discussions in (Perez and J. Wang 2017; Gontijo-Lopes et al. 2021)).

Consistency is actually optimized following the dual head trick used in (Tarvainen and Valpola 2017) where the classifier outputs two different decision y_{preds} and y_{cons} . y_{cons} is optimized to reconstruct the EMA target's y_{preds}^{EMA} (weight $\lambda_{cons} = 300$), and y_{preds} is tied to reconstruct y_{cons} (weight $\lambda_{logitdist} = 0.01$). This

limits the need for precise scheduling of consistency weights by having a “buffer” prediction head. The EMA model is updated with decay 0.97 (following a linear start for early iterations).

MixMatch Based SAMOSA on CIFAR10 The classifier $E_c \circ C$ is trained over 2 gpus using a AdamW optimizer with weight decay 0.02, $\beta = (0.9, 0.999)$ and constant learning rate 0.002 for 900 epochs over the unlabeled samples, with the reconstructive modules E_r and D optimized with a SGD optimizer with weight decay $5e-4$, Nesterov momentum 0.9 and cosine learning rate (base learning rate 0.2) for 900 epochs (set to fully decrease over 950 epochs).

$\lambda_{recons} = 0.25$, $\lambda_{SAMOSA} = 0.5$, pseudolabel targets are optimized with weight $\lambda_u = 37.5$ (linear ramp over 16384 iterations first), with batches of 256 labeled samples and 256 unlabeled samples (following the proportions in (Berthelot et al. 2019)), in the CIFAR10 250 label case the batch sizes are instead 250 and 250). Similarly to the Mean Teacher setting, we only incorporate hybrid samples from epoch 100 to epoch 800. Outside of this window, the model is optimized according to the standard MixMatch procedure (only on Mixed Up examples).

Hyperparameter Ranges Investigated Values for λ_{cons} , λ_{logit_dist} , λ_u , $\lambda_{recons,int}$, λ , β , as well as decay and learning rates were directly taken from (Robert et al. 2018; Berthelot et al. 2019) with no parameter search. Batch sizes for the Mean Teacher experiments were also taken from (Robert et al. 2018). Batch sizes for the Mix-Match experiments were taken to balance out labeled and unlabeled samples while being as large as possible (for faster training). We tried turning hybrid augmentation on $\{0,50,100,150\}$ epochs into training and turning it off $\{100,50,0\}$ epochs before the end of training on the Mean Teacher CIFAR10 1000 labels setting, and adapted the training schedule for other settings by taking the optimal setting with regards to test set accuracy on the CIFAR10 1000 label setting.

A.1.1.1 Architectures

We operate on a standard WideResNet-28-2 (Zagoruyko and Komodakis 2016) for our classifiers (both f and $E_c \circ C$). E_r follows the same architecture as E_c . Note that we keep the same architecture for both the generator and trained classifier to avoid any interference due to different function classes engendered by the models. The skeleton of D follows the one used in (Robert et al. 2018). Hyperparameters and optimizers were generally taken to follow settings reported in the base methods’ original papers (Tarvainen and Valpola 2017; Berthelot et al. 2019).

More precisely, we outline in Table A.1, Table A.2, Table A.4, Table A.3, Table A.5 and Table A.6 the detailed make up of the neural networks used in this paper.

A.1.1.2 Main modules

Layer	Details	Output shape
Input	Preprocessed image x	$3 \times 32 \times 32$
Conv2d	16 filters, 3×3 kernels, pad 1	$16 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	64 filters, 3×3 kernels, pad 1, stride 2	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	128 filters, 3×3 kernels, pad 1, stride 2	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$

Table A.1. – General structure of E_c .

Layer	Details	Output shape
Input	Preprocessed image x	$3 \times 32 \times 32$
Conv2d	16 filters, 3×3 kernels, pad 1	$16 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	64 filters, 3×3 kernels, pad 1, stride 2	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	128 filters, 3×3 kernels, pad 1, stride 2	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
Flatten	$C \times H \times W \rightarrow CHW$	8192
Linear	64 units	64
Softmax		64

Table A.2. – General structure of E_r .

Layer	Details	Output shape
Input	Semantic features z_c , (Style weights z_r)	$128 \times 8 \times 8$, (64)
DeconvBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
DeconvBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
DeconvBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
DeconvBlock	64 filters, 3×3 kernels, pad 1, stride 2	$64 \times 16 \times 16$
DeconvBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
DeconvBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
DeconvBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
DeconvBlock h	64 filters, 3×3 kernels, pad 1, stride 2	$64 \times 32 \times 32$
Auxiliary modulation h'	$z_r \odot h$	64×32
DeconvBlock	64 filters, 3×3 kernels, pad 1, takes h'	$64 \times 32 \times 32$
DeconvBlock	64 filters, 3×3 kernels, pad 1	$64 \times 32 \times 32$
DeconvBlock	64 filters, 3×3 kernels, pad 1	$64 \times 32 \times 32$
DeconvBlock h	16 filters, 3×3 kernels, pad 1	$16 \times 32 \times 32$
Deconv2d	3 filters, 3×3 kernels, pad 1	$3 \times 32 \times 32$

Table A.3. – General structure of D .

Layer	Details	Output shape
Input	Semantic features z_c	$128 \times 8 \times 8$
BatchNormalization2d		$128 \times 8 \times 8$
LeakyReLU	Negative slope 0.1	$128 \times 8 \times 8$
Global Average Pooling f	128 units	128
Linear	10 units, primary output, takes f as input	10
Linear	10 units, stability output, takes f as input	10

Table A.4. – General structure of C .

A.1.1.3 Building blocks

Layer	Details	Output shape
Input	Preprocessed image x	$C \times H \times W$
BatchNormalization2d		$C \times H \times W$
LeakyReLU	Negative slope 0.1	$C \times H \times W$
Conv2d	F filters, 3×3 kernels, pad 1, stride s	$F \times \frac{C}{s} \times \frac{H}{s}$
BatchNormalization2d		$F \times \frac{C}{s} \times \frac{H}{s}$
LeakyReLU	Negative slope 0.1	$F \times \frac{C}{s} \times \frac{H}{s}$
Conv2d r	F filters, 3×3 kernels, pad 1	$F \times \frac{C}{s} \times \frac{H}{s}$
Conv2d x (replaces)	iff $F \neq C$, F filters, 1×1 kernels, stride s , takes x	$F \times \frac{C}{s} \times \frac{H}{s}$
BatchNormalization2d x (replaces)	iff $F \neq C$, takes x , only for MixMatch	$F \times \frac{C}{s} \times \frac{H}{s}$
Fusion	$x + r$	$F \times \frac{C}{s} \times \frac{H}{s}$

Table A.5. – WideResBlock of F filters and stride s .

Layer	Details	Output shape
Input	Preprocessed image x	$C \times H \times W$
BatchNormalization2d		$C \times H \times W$
LeakyReLU	Negative slope 0.1	$C \times H \times W$
Deconv2d	F filters, 3×3 kernels, pad 1, stride s	$F \times sC \times sH$

Table A.6. – **DeconvBlock of F filters and stride s .**

A.1.2 Experimental details for SciMix experiments

We first detail how we obtain experimental results presented in [Section 2.4](#). All experiments were run using the Pytorch framework. Afterwards, we present additional examples of hybrids on SVHN and pseudo-code of SciMix’s main components.

A.1.2.1 Datasets

CIFAR10 dataset The CIFAR10 dataset (Krizhevsky and Hinton 2009) is a subset of the TinyImages dataset comprised 32×32 RGB images from ten classes: airplane, car, truck, boat, bird, cat, deer, dog, frog and horse. Available samples are split between 50000 training samples and 10000 test samples.

SVHN dataset The SVHN dataset (Netzer et al. 2011) is comprised of 32×32 RGB images of street numbers (divided along ten classes: one per digit). Available samples are split between 73257 training samples and 26032 test images.

Samples are randomly flipped horizontally (only for CIFAR10) and shifted by up to 4 pixels both horizontally and vertically with reflect padding. The resulting augmented samples are then standardized channel-wise according to train set statistics. No holdout validation set is kept for either dataset but hyper-parameters are mostly directly adapted from (Robert et al. 2018; Sohn et al. 2020).

A.1.2.2 Generator training

We train the generator architecture following the procedure described in [Section 2.4.1](#), using a Mean Teacher loss \mathcal{L}_S to train the classifier of the model. For each setting (eg CIFAR10 100 labels) we train one seeded generator to be used for data augmentation experiments. Most hyperparameters are shared across experimental settings. When parameters differ across settings, we detail the 5 values as CIFAR 10 100/250/500 / SVHN 60/100.

The autoencoder modules E_c, E_r, C, D and G are trained on a single gpu using a SGD optimizer with weight decay $5e - 4$, Nesterov momentum 0.9 and cosine learning rate (base 0.1/0.1/0.1/0.2/0.2 learning rate) for 300 epochs over the unlabeled samples (the rate is set to fully decrease over 350 epochs), consistency targets are optimized with weight $\lambda_{cons} = 300$, with batches of 60 labeled samples and 196 unlabeled samples (similarly to (Robert et al. 2018)).

Consistency is actually optimized following the dual head trick used in (Tarvainen and Valpola 2017) where the classifier outputs two different decision y_{preds} and y_{cons} . y_{cons} is optimized to reconstruct the EMA target’s y_{preds}^{EMA} (weight $\lambda_{cons} = 300$), and y_{preds} is tied to reconstruct y_{cons} (weight $\lambda_{logitdist} = 0.01$). This limits the need for precise scheduling of consistency weights by having a “buffer” prediction head. The EMA model is updated with decay 0.97 (following a linear start for early iterations).

To simplify notations in the main paper, loss coefficients are kept implicit. In this section, we will simply write out the corresponding hyperparameters λ_{term} (eg λ_S for \mathcal{L}_S). For all main experiments - excluding model analysis experiments - in the paper, the hyperparameters considered are the same: $\lambda_{recons} = 1, \lambda_{adv,r} = 0.5, \lambda_S = 1, \lambda_{hyb,class}^+ = 0.5, \lambda_{hyb,cont}^+ = 0.5, \lambda_{hyb,class}^- = 0.05, \lambda_{hyb,cont}^- = 0.05, \lambda_{adv,h} = 0.5$. More precisely, we only start optimizing $\mathcal{L}_{hyb,cont}^+$ and $\mathcal{L}_{hyb,cont}^-$ after epoch 50 since residual networks (like E_r) tend to start close to the identity function.

A.1.2.3 Training a classifier with SciMix data augmentation

We train the generator architecture following the procedure described in Section 2.4.2 to train a new classifier model f from scratch. Using a pretrained generator (one per setting), we generate hybrids to train a new model. Accuracy results are obtained from an EMA average of the classifier model following (Berthelot et al. 2019). Most hyperparameters are shared across experimental settings. When parameters differ across settings, we detail the 5 values as CIFAR10 100/250/500 / SVHN 60/100.

Mean Teacher The classifier f is trained on a single gpu using a SGD optimizer with weight decay $5e - 4$, Nesterov momentum 0.9 and cosine learning rate (base 0.1/0.1/0.1/0.2/0.2 learning rate) for 300 epochs over the unlabeled samples (the rate is set to fully decrease over 350 epochs) during training, consistency targets are optimized with weight $\lambda_{cons} = 300$, with batches of 60 labeled samples and 196 unlabeled samples (similarly to (Robert et al. 2018)).

Consistency is actually optimized following the dual head trick used in (Tarvainen and Valpola 2017) where the classifier outputs two different decision y_{preds} and y_{cons} . y_{cons} is optimized to reconstruct the EMA target’s y_{preds}^{EMA} (weight

$\lambda_{cons} = 300$), and y_{preds} is tied to reconstruct y_{cons} (weight $\lambda_{logitdist} = 0.01$). This limits the need for precise scheduling of consistency weights by having a “buffer” prediction head. The EMA model is updated with decay 0.97 (following a linear start for early iterations).

We keep the coefficient of the hybridizing loss $\mathcal{L}_{contradict} = 1$, and only apply it from epoch 50 to 150 as per standard data augmentation practices. To stabilize the estimation of $\mathcal{L}_{contradict}$, it is computed across three batches of hybrids (in the same spirit as recent advances in data augmentation such as Batch Augment).

FixMatch The classifier f is trained on a single gpu using a SGD optimizer with weight decay $5e - 4$, Nesterov momentum 0.9 and cosine learning rate (base 0.015/0.015/0.015/0.03/0.03 learning rate) for 1000 epochs over the unlabeled samples (the rate is set to fully decrease over 1050 epochs) during training with batches of 32 labeled samples and 224 unlabeled samples (similarly to (Sohn et al. 2020)). Pseudolabel targets are optimized with weight $\lambda_u = 1$.

We keep the coefficient of the hybridizing loss $\mathcal{L}_{contradict} = 1$, and only apply it from epoch 100 to 200 as per standard data augmentation practices. To stabilize the estimation of $\mathcal{L}_{contradict}$, it is computed across three batches of hybrids (in the same spirit as recent advances in data augmentation such as Batch Augment).

A.1.2.4 Architectures

We operate on a standard WideResNet-28-2 (Zagoruyko and Komodakis 2016) for our classifiers (both f and $E_c \circ C$). E_r follows the same architecture as E_c . Note that we keep the same architecture for both the generator and trained classifier to avoid any interference due to different function classes engendered by the models. The skeleton of G follows a StyleGanv2 (Karras et al. 2020) architecture. Hyperparameters and optimizers were generally taken to follow settings reported in the base methods’ original papers (Tarvainen and Valpola 2017; Berthelot et al. 2019).

More precisely, we outline in Table A.7, Table A.8, Table A.9, Table A.10 and Table A.11 the detailed make up of the neural networks used in this paper.

Layer	Details	Output shape
Input	Semantic features z_c	$128 \times 8 \times 8$
Linear	10 units, primary output, takes z_c as input	10
Linear	10 units, stability output, takes z_c as input	10

Table A.7. – General structure of C .

Layer	Details	Output shape
Input	Preprocessed image x	$3 \times 32 \times 32$
Conv2d	16 filters, 3×3 kernels, pad 1	$16 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	64 filters, 3×3 kernels, pad 1, stride 2	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	128 filters, 3×3 kernels, pad 1, stride 2	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
BatchNormalization2d		$128 \times 8 \times 8$
LeakyReLU	Negative slope 0.1	$128 \times 8 \times 8$
Global Average Pooling f	128 units	128

Table A.8. – General structure of E_c .

Layer	Details	Output shape
Input	Preprocessed image x	$3 \times 32 \times 32$
Conv2d	16 filters, 3×3 kernels, pad 1	$16 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
WideResBlock	64 filters, 3×3 kernels, pad 1, stride 2	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
WideResBlock	128 filters, 3×3 kernels, pad 1, stride 2	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
WideResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
Conv2d	8 filters, 1×1 kernels, pad 0	$8 \times 8 \times 8$

Table A.9. – General structure of E_r .

Layer	Details	Output shape
Input	Non-semantic features z_r , (Style weights z_c)	$8 \times 8 \times 8, 128$
StyleResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 8 \times 8$
StyleResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 8 \times 8$
StyleResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 8 \times 8$
StyleResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 16 \times 16$
StyleResBlock	32 filters, 3×3 kernels, pad 1	$32 \times 32 \times 32$
Deconv2d	3 filters, 3×3 kernels, pad 1	$3 \times 32 \times 32$

Table A.10. – General structure of G .

Layer	Details	Output shape
Input	Preprocessed image x	$3 \times 32 \times 32$
Conv2d	32 filters, 3×3 kernels, pad 1	$32 \times 16 \times 16$
ResBlock	64 filters, 3×3 kernels, pad 1	$64 \times 8 \times 8$
ResBlock	128 filters, 3×3 kernels, pad 1	$128 \times 4 \times 4$
ResBlock	32 filters, 3×3 kernels, pad 1	$256 \times 2 \times 2$
Linear	256 units	256
Linear	1 units	1

Table A.11. – General structure of D .

A.1.3 Experimental details for MixMo experiments

A.1.3.1 Implementation details

We first used the popular image classification **datasets** CIFAR-100 and CIFAR-10 (Krizhevsky and Hinton 2009). They contain 60k 32×32 natural and colored images in respectively 100 classes and 10 classes, with 50k training images and 10k test images. At a larger scale, we study Tiny ImageNet (Chrabaszcz et al. 2017), a downsampled version of ImageNet (J. Deng et al. 2009). It contains 200 different categories, 100k 64×64 training images (*i.e.* 500 images per class) and 10k test images.

Our code was adapted from the official MIMO (Havasi et al. 2021) implementation¹. For CIFAR, we re-use the **hyper-parameters** from MIMO (Havasi et al. 2021). The optimizer is SGD with learning rate of $\frac{0.1}{b} \times \frac{\text{batch-size}}{128}$, batch size 64, linear warmup over 1 epoch, decay rate 0.1 at steps $\{100, 200, 225\}$, l_2 regularization $3e-4$. We follow standard MSDA practices (Ashukha et al. 2020; J.-H. Kim et al. 2020; Y. Yang and Soatto 2019) and set the maximum number of epochs to 300. For Tiny ImageNet, we adapt PreActResNet-18- w , with $w \in \{1, 2, 3\}$ times more filters. We re-use the hyper-parameters from Puzzle-Mix (J.-H. Kim et al. 2020). The optimizer is SGD with learning rate of $\frac{0.2}{b}$, batch size 100, decay rate 0.1 at steps $\{600, 900\}$, 1200 epochs maximum, weight decay $1e-4$. Our experiments ran on a single NVIDIA 12Go-TITAN X Pascal GPU. All results without a † were obtained with these training configurations. We will soon release our code and pre-trained models to facilitate reproducibility.

Batch repetition increases performances at the cost of **longer training**, which may be discouraging for some practitioners. Thus in addition to $b = 4$ as in MIMO (havasi2020raining), we often consider the quicker $b = 2$. Note that most of our concurrent approaches also increase training time: DE (Lakshminarayanan et al.

1. <https://github.com/google/edward2/>

2017) via several independent trainings, Puzzle-Mix (J.-H. Kim et al. 2020) via saliency detection ($\approx \times 2$), GradAug (T. Yang et al. 2020) via multiple subnetworks predictions ($\approx \times 3$) or Mixup BA (Hoffer et al. 2020) via 10 batch augmentations ($\approx \times 7$ with our hardware on a single GPU).

MixMo operates in the features space and is complementary with **pixels augmentations**, *i.e.* cropping, AugMix. The standard vanilla pixels data augmentation (He et al. 2016a) consists of 4 pixels padding, random cropping and horizontal flipping. When combined with CutMix, notably to benefit from multilabel smoothing, the input may be of the form: $(m_x(x_i, x_k, \lambda), x_j)$, where x_k is randomly chosen in the whole dataset, and not only inside the current batch². Moreover, $\mathcal{M}_{\text{Cut-MixMo}}$ modifies by $\mathbf{1}_{\mathcal{M}}$ the visible part from mask $\mathbf{1}_m$ (of area λ). We thus modify targets accordingly: $(\lambda' y_i + (1 - \lambda') y_k, y_j)$ where $\lambda' = \frac{\sum \mathbf{1}_m \odot \mathbf{1}_{\mathcal{M}}}{\sum \mathbf{1}_{\mathcal{M}}}$. To fully benefit from b , we force the repeated x_i to remain predominant in its b appearances: *i.e.*, we swap x_i and x_k if $\lambda' < 0.5$. We see CutMix as a perturbation on the main batch sample.

Distributional uncertainty measures help when there is a mismatch between train and test data distributions. Thus (Hendrycks and T. Dietterich 2019) introduced **CIFAR-100-c** on which **AugMix** performs best. AugMix sums the pixels from a chain of several augmentations and is complementary to our approach in features. We use default parameters³: the severity is set 3, the mixture’s width to 3 and the mixture’s depth to 4. We exclude operations in AugMix which overlap with CIFAR-100-c corruptions: thus, [equalize, posterize, rotate, solarize, shear_x, shear_y, translate_x, translate_y] remain. We disabled the Jensen-Shannon Divergence loss between predictions for the clean image and for the same image AugMix augmented: that would otherwise triple the training time. For comparison of out-of-domain uncertainty estimations, we report NLL as in (Havasi et al. 2021): indeed, the recommendation of (Ashukha et al. 2020) to apply TS only stands for in-domain test set.

A.1.3.2 Evaluation setting and metrics

We reproduce the **experimental setting** from CutMix (Y. Yang and Soatto 2019), Manifold Mixup (Verma et al. 2019a) and other works such as the recent state-of-the-art ResizeMix (Qin et al. 2020): in absence of a validation dataset, results are reported at the epoch that yields the best test accuracy. For fair comparison, we apply this early stopping for all concurrent approaches. Nonetheless, for the sake of completeness, Table A.12 shows results without early stopping on the main experiment (CIFAR with a standard WRN-28-10). We recover the exact same ranking among methods as in our main Table 3.1.

2. Following <https://github.com/ildoonet/cutmix>

3. <https://github.com/google-research/augmix/blob/master/cifar.py>

Dataset		CIFAR-100					CIFAR-10			
Approach	Time Tr./Inf.	Top1 %, \uparrow	Top5 %, \uparrow	NLL _c 10 ⁻² , \downarrow	NLL 10 ⁻² , \downarrow	ECE 10 ⁻² , \downarrow	Top1 %, \uparrow	NLL _c 10 ⁻² , \downarrow	NLL 10 ⁻² , \downarrow	ECE 10 ⁻² , \downarrow
Vanilla		81.47	95.57	73.6	76.2	6.47	96.31	12.5	14.1	1.95
Mixup	1/1	83.15	95.75	66.3	67.3	1.62	97.00	11.3	11.5	0.97
Hard PatchUp [†]		83.87	-	-	66.0	-	97.47	-	11.4	-
CutMix		83.74	96.18	65.4	66.1	4.95	97.21	9.7	10.8	1.51
Puzzle-Mix [†]		2/1	84.05	96.08	66.9	68.1	2.76	-	-	-
GradAug [†]	3/1	83.98	96.28	-	-	-	-	-	-	-
+ CutMix [†]		85.25	96.85	-	-	-	-	-	-	-
Mixup BA [†]	7/1	84.30	-	-	-	-	97.80	-	-	-
DE (2 Nets)	2/2	83.15	96.30	66.0	67.2	5.15	96.58	11.1	12.2	1.82
+ CutMix		85.46	96.90	57.4	57.5	3.62	97.51	8.7	9.0	1.16
MIMO ($M = 2$)	2/1	82.04	95.75	69.1	72.4	6.32	96.33	12.1	13.4	1.89
Linear-MixMo		81.88	95.97	67.8	70.3	6.20	96.55	11.4	12.5	1.67
+ CutMix		84.55	96.95	57.4	57.5	2.54	97.34	8.9	9.3	1.34
Cut-MixMo		84.07	96.97	56.6	57.9	4.19	97.26	8.7	9.1	0.98
+ CutMix		85.17	97.28	54.4	54.5	2.13	97.33	8.5	8.6	0.88
MIMO ($M = 2$)	4/1	82.74	95.90	67.0	74.0	7.56	96.66	11.5	13.6	1.98
MIMO [†] ($M = 3$)		82.0	-	-	69.0	2.2	96.4	-	12.3	1.0
Linear-MixMo		82.53	96.08	65.8	68.5	6.64	96.78	10.8	11.8	1.80
+ CutMix		85.24	96.97	56.3	56.4	3.53	97.53	8.8	8.6	1.19
Cut-MixMo		85.32	97.12	53.6	54.8	4.53	97.42	8.1	8.4	1.15
+ CutMix		85.59	97.33	53.2	53.3	1.95	97.70	8.0	8.2	0.98

Table A.12. – WRN-28-10 on CIFAR without early stopping.

Following recent works in ensembling (Chirkova et al. 2020; Lobacheva et al. 2020; Rame and Cord 2021), we have mainly focused on the NLL_c metric for in-domain test set. Indeed, (Ashukha et al. 2020) have shown that “comparison of [...] ensembling methods without temperature scaling (TS) (Guo et al. 2017) might not provide a fair ranking”. Nevertheless in Table A.12, we found that *Negative Log-Likelihood* (NLL) (without TS) leads to similar conclusions as NLL_c (after TS).

The TS even mostly seems to benefit to poorly calibrated models, as shown by the calibration criteria *Expected Calibration Error* (ECE, ↓, 15 bins). ECE measures how confidences match accuracies. MixMo attenuates over-confidence in large networks and thus reduces ECE. In our case, combining ensembling and data augmentation improves calibration (Wen et al. 2021). Note that the appropriate measure of calibration is still under debate (Nixon et al. 2019). Notably, (Ashukha et al. 2020) have also stated that, despite being widely used, ECE is biased and unreliable: we can confirm that we found ECE to be dependant to hyper-parameters and implementation details. Due to space constraints and these pitfalls, we have not included this controversial metric in the main paper.

A.1.4 Experimental details for MixShare experiments

The code for this work was directly adapted from the official MixMo (Rame et al. 2021) codebase: <https://github.com/alexrame/mixmo-pytorch>.

We followed similar experimental settings on CIFAR 100 as MixMo (Rame et al. 2021) and present here the adapted setting description:

We used standard architecture WRN-28- w , with a focus on $w = 2$. We re-use the **hyper-parameters** configuration from MIMO (Havasi et al. 2021) with batch repetition 2 (bar2). The optimizer is SGD with learning rate of $\frac{0.1}{b} \times \frac{\text{batch-size}}{128}$, batch size 64, linear warmup over 1 epoch, decay rate 0.1 at steps $\{75, 150, 225\}$, l_2 regularization $3e-4$. We follow standard MSDA practices (Y. Yang and Soatto 2019) and set the maximum number of epochs to 300. Our experiments ran on a single NVIDIA 12Go-TITAN X Pascal GPU.

All experiments were run three times on three fixed seeds from the same version of the codebase. Qualitative results presented in Figure 3.12 and Figure 3.14 are obtained by visualizing results for the first set of random seeds. Quantitative results presented in Table 3.6 are given in the form of $mean \pm std$ over the three runs.

A.1.5 Experimental details for MixViT experiments

We give here a detailed inventory of hyperparameters and training procedure used in this paper. For contractual reasons, it is not possible to provide the codebase used in this paper, but we provide indications below.

Our experiments were run through a codebase implemented using the PyTorch (Paszke et al. 2019) library. Our codebase built upon the public official implementation of MixMo (<https://github.com/alexrame/mixmo-pytorch>) and adapted components from multiple public codebases:

- We took ConViT building blocks from github.com/facebookresearch/convit/blob/main/convit.py
- We took CaiT building blocks from github.com/facebookresearch/deit/blob/main/cait_models.py
- We took our AutoAugment implementation from github.com/DeepVoltaire/AutoAugment
- We took a number of primitive functions from the timm library github.com/rwightman/pytorch-image-models

The MixMo codebase should provide everything necessary to train multi-input multi-output networks. The main adaptation that must be made is the addition of the transformer networks and specific MixViT routines in the transformer networks.

All our experiments were run three times on three fixed seeds from the same codebase. Quantitative results are given in the form of $mean \pm std$ over the three runs.

Computational resources We used approximately 5000 GPU hours in this work on a nation-wide cluster, with access to the cluster granted for a set number of GPU hours on the basis of a grant application. The cluster provides V100 GPUs with Intel Cascade Lake 6248 CPUs. Most experiments ran on a single GPU, with the ImageNet-100 being run on 4 GPUs at once.

FLOPS estimations We provide an estimate of the relative number of FLOPS used by different methods in [Table 4.1](#) and [Table 4.6](#). This number is traditionally computed for pytorch models on a theoretical basis by standardized libraries like `ptflops` or `fvcore`. These libraries however do not support custom attention layers and are therefore un-useable for our purposes.

Fortunately, estimating the ratio of FLOPS is straightforward in [Table 4.1](#): the additional overhead of MIMO and MixMo is known to be approximately $1\times$ on CNNs (which also holds on transformers) and separate inference just leads to inferring twice, hence the $2\times$ ratio. We use the upper bound derived in [Appendix Section A.7.3](#) for MixViT and its variants that follow similar computational flows. For the last line of [Table 4.6](#), we give a rough approximation derived by considering the impact of the larger number of tokens on the attention layers.

Corrected CutMix Scheme An issue that arises when considering a CutMix scheme to mix patch tokens is that CutMix masks have binary values for each pixel value whereas a patch contains multiple patches. It stands to reason therefore that we will often have situations where the patch contains both masked pixels and unmasked pixels. While the natural solution is simply to take a continuous mask value by averaging over the patch, we have seen in [Section 4.3.1](#) that subnetworks have trouble sharing tokens.

To address this, we correct the CutMix scheme by using a majority vote within the patch: if most pixels are masked, then we mask the patch and vice-versa. Ties (where we have as many masked and unmasked pixels) are broken by choosing randomly for one set of inputs which input will be assigned the problematic patches. It worth noting that this scheme is for all intents and purposes the MixToken (Jiang et al. 2021) scheme, with some practical differences in the sort of mask generated.

Architectural details We used a modified variant of the ConViT (D’Ascoli et al. 2021) architecture for our experiments hidden dimension 384, 12 attention heads, 5 GPSA blocks and 2 SA Blocks similarly to dytox (Douillard et al. 2022). Patch size was taken to be 4×4 on CIFAR, 8×8 on TinyImageNet and 16×16 on ImageNet-100. It is worth noting GPSA blocks should in theory have a square number of heads to exactly emulate convolutional kernels at initialization, but we did not find this to be an issue experimentally.

We took the CaiT models directly from the seminal paper (Touvron et al. 2021b) and used the same patch size as for our ConViT model (which also coincides with CaiT’s default settings).

Training details In general, we train our models over 150 epochs using the AdamW optimizer with a learning rate of 0.001, weight decay of 0.05 and linear warmup over 10% of training. We follow a step decay schedule with decay rate 0.1 at steps {125, 140}

We used a base batch size B of 128 on CIFAR and TinyImageNet and 1048 on ImageNet. The effective batch size b used was dependant on memory constraints,

and the learning rate was scaled accordingly with a ratio $\sqrt{\frac{b}{B}}$ following heuristics derived from (Granzio et al. 2020).

Following DeiT, we train with a lot of regularization. We use 0.05 weight decay, 0.05 stochastic depth for CaiT models, 0.1 label smoothing, 3 Batch augmentations. We use the CIFAR configuration of AutoAugment for CIFAR and TinyImageNet, and the ImageNet configuration for ImageNet-100. We also apply CutMix (Yang and Soatto 2019) and MixUp (H. Zhang et al. 2018) but find applying them simultaneously instead of alternatively yields better result and is more stable. Practically speaking, we draw CutMix mask with $\alpha = 1.0$ but instead of the mask having binary values (0 or 1), it has continuous values like a $\alpha = 0.8$ MixUp scheme.

Similarly to MixMo, we progressively adapt the input mixing scheme to reflect the mechanism used at inference. In the MixMo case, this means using a summing scheme instead of CutMix with a probability p that linearly grows over the last twelfth of training. In the MixViT case, we use masks that take all tokens from one of the inputs with a probability p that linearly grows over the last twelfth of training. We directly inherited the mechanism from MixMo and did not tune the point at which we start increasing the probability p linearly.

Choice of hyperparameters We tuned the parameter $\alpha \in \{0.25, 0.5, 1.0, 2.0\}$ used to draw MixViT’s input mixing CutMix mask on CIFAR-100 and kept it fixed for all other experiments. $\alpha = 0.5$ was chosen as it was more stable and not necessarily for optimality purposes. We did not tune the root parameter used for loss balancing in MixMo and instead discarded it early on in our preliminary experiments (equivalent to choosing root $r = 1$).

While cosine decay is standard in the transformer literature, we failed to converge with it on CIFAR-100 preliminary experiments. As we wished to ensure our single-input single-output backbone had converged for fair comparison with MixViT, we adopted a step decay schedule and roughly set decay steps by considering the training loss on CIFAR-100. Similarly, we train on 150 epochs instead of 100 because our models clearly failed to converge in 100 epochs on CIFAR-100.

We largely inherited all other parameters(patch sizes, batch sizes, 0.001 learning rate, 3 batch augmentations, 0.05 weight decay, AdamW optimizer, 0.05 stochastic depth dropout on CaiT, 0.1 label smoothing, CutMix $\alpha = 1.0$, MixUp $\alpha = 0.8$, RandErase and AutoAugment configurations) from the literature (Touvron et al. 2021a; S. H. Lee et al. 2021). We did double weight decay and stochastic depth on TinyImageNet and ImageNet-100 for MixViT to avoid overfitting and checked the performance of the backbone with those parameters, but did not look for an optimal value.

A.2 Pseudo-code algorithms

We provide here pseudo code for important components of SAMOSA in Section A.2.1, SciMix in Section A.2.2, MixMo in Section A.2.3, MixShare in Section A.2.4 and MixViT in Section A.2.5.

A.2.1 Pseudo-code for the SAMOSA framework

Algorithm A.3 (and Figure 2.10) presents how hybrids can be generated from the SAMOSA auto-encoding framework. We then propose a general application of SAMOSA hybrids to the Mean Teacher framework in Algorithm A.1, and a closer integration with the MixMatch framework in Algorithm A.2

Algorithm A.1 Skeleton of SAMOSA integration with the Mean Teacher Framework. Additions to the Mean Teacher Framework are in blue.

Require: Dataset $\mathcal{D} = \mathcal{D}_l \cup \mathcal{D}_u$, Number of complete training cycles n_{cycles} , Number of epochs in a cycle n_{epochs} .

```

for cycle in 1...n_cycles do
   $\mathcal{D}_{l,0} = \mathcal{D}_l$ 
  for epoch in 1...n_epochs do
    for  $\mathcal{B} = \mathcal{B}_l \cup \mathcal{B}_u$  in Batch( $\mathcal{D}$ ) do
      Compute  $\mathcal{L}_{SAMOSA}(\mathcal{B}) = \mathcal{L}_{0,MT}(\mathcal{B}) + \Omega_{SAMOSA}(\mathcal{B})$ 
      Optimization step for  $\mathcal{L}_{SAMOSA}(\mathcal{B})$ 
    end for
  end for
  for epoch in 1...10 do
     $\mathcal{D}_h = \emptyset$ 
    for  $\mathcal{B}_l, \mathcal{B}_u$  in zip(Batch( $\mathcal{D}_{l,0}$ ), Batch( $\mathcal{D}_u$ )) do
       $\mathcal{O} = Hybridize(\mathcal{B}_l, \mathcal{B}_u)$ 
       $\mathcal{D}_h = \mathcal{D}_h \cup \mathcal{O}$ 
    end for
  end for
   $\mathcal{D}_l = \mathcal{D}_{l,0} \cup \mathcal{D}_h$ 
end for

```

Algorithm A.2 Skeleton of SAMOSA integration with the MixMatch Framework. Additions to the MixMatch Framework are in [blue](#).

Require: Dataset $\mathcal{D} = \mathcal{D}_l \cup \mathcal{D}_u$, Number of epochs during training n_{epochs}

```

 $\mathcal{D}_{l,0} = \mathcal{D}_l$ 
for epoch in  $1 \dots n_{epochs}$  do
  for  $\mathcal{B} = \mathcal{B}_l \cup \mathcal{B}_u$  in  $\text{Batch}(\mathcal{D})$  do
     $\mathcal{X}_l, \mathcal{Y}_l := \mathcal{B}_l$ 
     $\mathcal{X}_u := \mathcal{B}_u$ 
    Estimate pseudo-targets  $\mathcal{Y}_u$  as per (Berthelot et al. 2019)
     $\mathcal{W} := \text{Concat}(\{(x_l, y_l)\}, \{x_u, y_u\})$ 
     $\tilde{\mathcal{W}} := \text{Shuffle}(\mathcal{W})$ 
     $p \sim \text{Random}(0, 1)$ 
    if  $p < \frac{1}{5}$  then
       $\tilde{\mathcal{B}} = \text{Hybridize}(\mathcal{W}, \tilde{\mathcal{W}})$ 
    else
       $\tilde{\mathcal{B}} = \text{MixUp}_{biased}(\mathcal{W}, \tilde{\mathcal{W}})$ 
    end if
    Compute  $\mathcal{L}_{SAMOSA}(\tilde{\mathcal{B}}) = \mathcal{L}_{0,Mix}(\tilde{\mathcal{B}}) + \Omega_{SAMOSA}(\tilde{\mathcal{B}})$ 
    Optimization step for  $\mathcal{L}_{SAMOSA}(\tilde{\mathcal{B}})$ 
  end for
end for

```

Algorithm A.3 Algorithm for the hybridization procedure.

Require: Batch $\mathcal{B} = \mathcal{B}_l \cup \mathcal{B}_u$, Modules E_c, E_r, C, D

```

function HYBRIDIZE( $\mathcal{B}_l, \mathcal{B}_u$ )
   $\mathcal{O} = \emptyset$ 
  for  $(x^{(1)}, y^{(1)}), x^{(2)} \in \text{zip}(\mathcal{B}_l, \mathcal{B}_u)$  do
     $z_c^{(1)} = E_c(x^{(1)})$ 
     $z_r^{(2)} = E_r(x^{(2)})$ 
     $x_h = D(x^{(1)}, x^{(2)})$ 
    if  $\text{argmax}(C(E_c(x_h))) == y^{(1)}$  then
       $x_h = x^{(1)}$ 
    end if
     $\mathcal{O} = \mathcal{O} \cup \{(x_h, y^{(1)})\}$ 
  end for
  return  $\mathcal{O}$ 
end function

```

A.2.2 Pseudo-code for the SciMix framework

We provide here an outline in Pseudo Code of the SciMix framework. [Algorithm A.4](#) shows the hybridization procedure while [Algorithm A.5](#) shows how the data augmentation is leveraged to improve upon pre-existing frameworks.

Algorithm A.4 Algorithm for the hybridization procedure.

Require: Batch $\mathcal{B} = \mathcal{B}_l \cup \mathcal{B}_u$, Modules E_c, E_r, G

```

function HYBRIDIZE( $\mathcal{B}_l, \mathcal{B}_u$ )
   $\mathcal{O} = \emptyset$ 
  for  $(x^{(1)}, y^{(1)}), x^{(2)} \in \text{zip}(\mathcal{B}_l, \mathcal{B}_u)$  do
     $z_c^{(1)} = E_c(x^{(1)})$ 
     $z_r^{(2)} = E_r(x^{(2)})$ 
     $x_h = G(z_c^{(1)}, z_r^{(2)})$ 
     $\mathcal{O} = \mathcal{O} \cup \{(x_h, y^{(1)})\}$ 
  end for
  return  $\mathcal{O}$ 
end function

```

Algorithm A.5 Training a model following method X with the additional hybridization loss.

Require: Dataset $\mathcal{D} = \mathcal{D}_l \cup \mathcal{D}_u$, Number of epochs n_{epochs} .

```

for epoch in  $1 \dots n_{epochs}$  do
  for  $\mathcal{B} = \mathcal{B}_l \cup \mathcal{B}_u$  in Batch( $\mathcal{D}$ ) do
    Compute method X's batch  $\mathcal{B}_X = \text{Preprocess}_X(\mathcal{B})$ 
    Compute hybrids  $\mathcal{B}_h = \text{Hybridize}(\mathcal{B}, \pi(\mathcal{B}))$ 
    Compute  $\mathcal{L}_{SciMix}(\mathcal{B}) = \mathcal{L}_X(\mathcal{B}_X) + \mathcal{L}_{contradict}(\mathcal{B}_h)$ 
    Optimization step for  $\mathcal{L}_{SciMix}(\mathcal{B})$ 
  end for
end for

```

A.2.3 Pseudo-code for the MixMo framework

Algorithm A.6 Procedure for Cut-MixMo with $M = 2$ subnetworks

Require: Dataset $D = \{x_i, y_i\}_{i=1}^{|D|}$, probability p of applying binary mixing via patches, reweighting coefficient r , concentration parameter α , batch size b_s , batch repetition b , optimizer g , learning rate l_r . Parameters: first convolutions $\{c_0, c_1\}$, dense layers $\{d_0, d_1\}$ and core network \mathcal{C} , randomly initialized.

for epoch from 1 to #epochs do

for step from 1 to $\frac{|D| \times b}{b_s}$ do

 Randomly select $\frac{b_s}{b}$ samples

 Duplicate these samples b times to create batch $\{x_i, y_i\}_{i \in B}$ of size b_s

 Randomly shuffle B with π to create $\{(x_i, x_j), (y_i, y_j)\}_{i \in B, j = \pi(i)}$

 # Define the mixing mechanism at the batch level

if epoch $> \frac{11}{12} \times \text{\#epochs}$ then

$p_e = p \frac{\text{\#epochs} - \text{epoch}}{\frac{1}{12} \times \text{\#epochs}}$

else

$p_e = p$

end if

 Sample $1_{\text{binary}} \sim \text{Ber}(p_e)$ from Bernoulli, Sample $1_{\text{outside}} \sim \text{Ber}(0.5)$

 # Forward and loss

for $i \in B$ do

 Sample $\kappa_i \sim \text{Beta}(\alpha, \alpha)$, $l_i^0 = c_0(x_i)$ and $l_i^1 = c_1(x_{\pi(i)})$

if 1_{binary} then

 Sample $1_{\mathcal{M}}$ a rectangular binary mask with average κ_i

if 1_{outside} then

$1_{\mathcal{M}} \leftarrow 1 - 1_{\mathcal{M}}, \kappa_i \leftarrow 1 - \kappa_i$

end if

$l_i = 2 [1_{\mathcal{M}} \odot l_0 + (1 - 1_{\mathcal{M}}) \odot l_1]$

else

$l_i = 2 [\kappa_i l_0 + (1 - \kappa_i) l_1]$

end if

 Extract features $f_i \leftarrow \mathcal{C}(l_i)$ from core network

 Compute predictions $\hat{y}_i^0 \leftarrow d_0(f_i)$ and $\hat{y}_i^1 \leftarrow d_1(f_i)$

 Compute weights $w_i \leftarrow 2 \frac{\kappa_i^{1/r}}{\kappa_i^{1/r} + (1 - \kappa_i)^{1/r}}$

 Compute loss $\mathcal{L}_i \leftarrow w_i \mathcal{L}_{\text{CE}}(y_i, \hat{y}_i^0) + (2 - w_i) \mathcal{L}_{\text{CE}}(y_{\pi(i)}, \hat{y}_i^1)$

end for

 Aggregate loss $\mathcal{L}_{\text{MixMo}} \leftarrow \frac{1}{|B|} \sum \mathcal{L}_i$

$c_0, c_1, \mathcal{C}, d_0, d_1 \leftarrow g(\text{gradient} = \nabla \mathcal{L}_{\text{MixMo}}, \text{learning rate} = \frac{l_r}{b})$

end for

end for

A.2.4 Pseudo-code for the MixShare framework

Algorithm A.7 Procedure for MixShare training with $M = 2$ subnetworks

Require: Dataset $D = \{x_i, y_i\}_{i=1}^{|D|}$, reweighting coefficient r , concentration parameter α , batch size b_s , batch repetition b , optimizer g , learning rate l_r . Parameters: first convolutions $\{c_0, c_1\}$ (**initialized to the same value**), dense layers $\{d_0, d_1\}$ and core network \mathcal{C} , randomly initialized.

for epoch from 1 to #epochs do

for step from 1 to $\frac{|D| \times b}{b_s}$ do

 Prepare repeated batch B

 # Define the mixing mechanism at the batch level

 Sample $1_{outside} \sim \text{Ber}(0.5)$

 # Forward and loss

for $i \in B$ do

 Sample $\kappa_i \sim \text{Beta}(\alpha, \alpha)$, $l_i^0 = c_0(x_i)$ and $l_i^1 = c_1(x_{\pi(i)})$

 Sample $\mathbf{1}_{\mathcal{M}}$ a rectangular binary mask with average κ_i

if $1_{outside}$ then

$\mathbf{1}_{\mathcal{M}} \leftarrow \mathbf{1} - \mathbf{1}_{\mathcal{M}}$, $\kappa_i \leftarrow 1 - \kappa_i$

end if

$l_i = 2 [\mathbf{1}_{\mathcal{M}} \odot l_0 + (\mathbf{1} - \mathbf{1}_{\mathcal{M}}) \odot l_1]$

 Extract features $f_i \leftarrow \mathcal{C}(l_i)$ from core network

 # Unmix and predict

Compute $f'_0 \leftarrow \text{GlobalAveragePool}(\mathbf{1}_{\mathcal{M}} \odot f_i)$

Compute $f'_1 \leftarrow \text{GlobalAveragePool}((\mathbf{1} - \mathbf{1}_{\mathcal{M}}) \odot f_i)$

Compute predictions $\hat{y}_i^0 \leftarrow d_0(f'_0)$ and $\hat{y}_i^1 \leftarrow d_1(f'_1)$

 Compute weights $w_i \leftarrow 2 \frac{\kappa_i^{1/r}}{\kappa_i^{1/r} + (1 - \kappa_i)^{1/r}}$

 Compute loss $\mathcal{L}_i \leftarrow w_i \mathcal{L}_{\text{CE}}(y_i, \hat{y}_i^0) + (2 - w_i) \mathcal{L}_{\text{CE}}(y_{\pi(i)}, \hat{y}_i^1)$

end for

 Aggregate loss $\mathcal{L}_{\text{MixShare}} \leftarrow \frac{1}{|B|} \sum \mathcal{L}_i$

$c_0, c_1, \mathcal{C}, d_0, d_1 \leftarrow g(\text{gradient} = \nabla \mathcal{L}_{\text{MixMo}}, \text{learning rate} = \frac{l_r}{b})$

end for

end for

A.2.5 Pseudo-code for the MixViT framework

We provide in [Algorithm A.8](#) and [Algorithm A.9](#) the training and inference procedure for MixViT. PatchEmbed simply embeds the inputs into patch representations as is standard in transformer frameworks, MixWithMasks simply mixes the inputs following the masks given, AddPosEmbed just adds the positional embeddings to the mixed tokens. SourceAttribution implements [Equation 4.5](#) or [Equation 4.6](#) depending on which variant of MixViT is considered. SourceAttributionTiled is similar, except it creates M sets of tokens, each equivalent to $\text{SourceAttribution}(x, \mathcal{T}_i)$ with \mathcal{T}_i being made up null masks $\mathbf{0}_N$ except for the one at position i being a unit mask $\mathbf{1}_N$.

Algorithm A.8 Mixvit forward at train- ing.	Algorithm A.9 MixViT forward at infer- ence.
Require: Inputs x_i , Masks \mathcal{M}_i Self-Attention blocks SA_k Class-Attention blocks CA_k Classification token c_0 Dense classification layers d_i function FORWARD # Format the input $\{x_i\} := \{\text{PatchEmbed}(x_i)\}$ $x := \text{MixWithMasks}(\{x_i\}, \{\mathcal{M}_i\})$ $x := \text{AddPosEmbed}(x_i)$ # Standard forward evaluations for $k=0, \dots, L-1$ do $x := SA_k(x)$ end for # Attribute sources to patches $x := \text{SourceAttribution}(x, \{\mathcal{M}_i\})$ # Standard class-attention $x := \text{Concat}([c_0; x])$ for $k=0, 1$ do $x := CA_k(x)$ end for # Predict from features $c_0 := x[0]$ $\{p_i\} := d_i(c_0)$ return $\{p_i\}$ end function	Require: Inputs x_i , Masks \mathcal{M}_i Self-Attention blocks SA_k Class-Attention blocks CA_k Classification token c_0 Dense classification layers d_i function FORWARD # Format the input $\{x_i\} := \{\text{PatchEmbed}(x_i)\}$ # No mixing needed ! $x := \text{AddPosEmbed}(x_i)$ # Standard forward evaluations for $k=0, \dots, L-1$ do $x := SA_k(x)$ end for # Create sets of sourced tokens $\{x_i\} := \text{SourceAttributionTiled}(x)$ # Standard class-attention $\{x_i\} := \{\text{Concat}([c_0; x_i])\}$ for $k=0, 1$ do $\{x_i\} := \{CA_k(x_i)\}$ end for # Predict from features $\{c_{0,i}\} := \{x_i[0]\}$ $\{p_i\} := \{d_i(c_{0,i})\}$ return $\{p_i\}$ end function

procedure and our hybridization procedure, and suggests there is indeed complementarity between our method and other mixing augmentation methods.

To verify these intuitions, we investigate this from a more quantitative point of view.

A.3.2 Case Study: Component Separation

We now investigate the composition of generated hybrids on reduced settings (MT Base) to verify the model’s ability to generate hybrids that correctly inherit their parent’s semantic and non-semantic components. To this end, at various points during training, we generate a study dataset of hybrid samples \mathcal{D}_H . The dataset is generated by mixing every sample in the labeled set with ten random unlabeled samples such that $\#\mathcal{D}_H = 10 \times \#\mathcal{D}_l$.

Inheritance of semantic and non-semantic features We start by assessing how well generated hybrids inherit semantic/non-semantic features with respect to our model’s learned projections. The quality of the inherited semantic component can be approximated straightforwardly by considering the accuracy s_c of our trained classifier on hybrids (ie, checking how many hybrids are correctly classified as belonging to the same class as their semantic parent). As we do not have access to such a clear criterion for the non-semantic component, we use a proxy metric in the non-semantic latent space. We consider the distance $d_l := \|z_r^h - z_r^1\|_2^2$ (resp. $d_r := \|z_r^h - z_r^2\|_2^2$) between the extracted non-semantic feature $z_r^h = E_r(x_h)$ of a hybrid x_h and those of its semantic parent z_r^1 (resp. non-semantic parent z_r^2). If $d_l \geq d_r$, then we conclude the hybrid correctly inherited its non-semantic parent’s style component. As such, we can define a non-semantic separation accuracy s_r by the proportion of hybrids in \mathcal{D}_H correctly identified as being closer to their non-semantic parent. In other words, we monitor whether the hybrid’s non-semantic content is indeed closer to its non-semantic parent’s.

The accuracy of the semantic and non-semantic separation tasks are presented in (Table A.13a) along with the average distances in non-semantic space to the hybrid’s parent samples d_l and d_r at the end of training. On both datasets, we can observe that hybrids mostly inherit the correct semantic and non semantic characteristics at the end of training. In particular, non-semantic features of hybrids are about 10 times closer to their non-semantic parents’ compared to their semantic parents’.

Importantly, the observed inheritance of semantic/non-semantic features significantly improves over the course of the entire training. For instance, with 1000 labels on CIFAR10, semantic accuracy s_c on generated hybrids at the end of the

Method	CIFAR10	SVHN	Method	MNIST-M
	1000	250		100
Accuracy s_c (%)	97.3 ± 0.6	100 ± 0.0	Accuracy s_c (%)	99.9 ± 0.2
Accuracy s_r (%)	100 ± 0	98.2 ± 0.3	Accuracy s_r (%)	96.8 ± 3.1
Ratio of mean $\frac{d_c}{d_r}$	15.5 ± 2.6	7.6 ± 1.4	Ratio of mean $\frac{d_c}{d_r}$	11.0 ± 4.0

(a) Component separation (CIFAR10 and SVHN). (b) Background inheritance (MNIST-M).

Table A.13. – **Identification of semantic and non-semantic parents on a hybrid dataset \mathcal{D}_H at the end of training on multiple datasets.** Both the semantic separation s_c and the non-semantic separation s_r accuracies show the model properly incorporates semantic and non-semantic information during hybridization. The ratio of the average non-semantic distances $\frac{d_c}{d_r}$ between hybrids and their semantic/non-semantic parent is given to complement non-semantic separation scores s_r .

first training cycle (300 epochs, no hybrid augmentation yet) is 74.1 ± 2.5 , 93.0 ± 2.5 at the end of the second (trained with hybrid augmentation) and 97.3 ± 0.6 at the end of training. In theory, two inputs reconstructed from the same semantic features but different non-semantic features should lead to extracting the same semantic features. However, in practice generated hybrids constitute new samples an overfit model could have trouble accommodating, or present combinations of semantic/non-semantic features that interfere with each other. As per the previous results, our augmentation strategy helps the model deal with those new problematic samples by presenting them as training samples.

Inheritance of non-semantic background in MNIST-M To better understand non-semantic features, we run an additional experiment by generating an MNIST-M-style dataset (Ganin and Lempitsky 2015) by combining each digit picture in the MNIST (Yann. Lecun 1998) dataset with a random crop from the BSD 500 dataset (Arbelaez et al. 2011) (Figure A.2). A model is trained following our standard procedure over 50000 training samples (100 labeled samples), and we track the hybrids generated during training as outlined previously.

Once again, we assess the correct inheritance of semantic content from the semantic parent by tracking the classification accuracy s_c over hybrids. This experiment however differs from the previous one in how the non-semantic distances d_l and d_r are computed. Instead, of considering the distances in z_r latent space we leverage the construction of MNIST-M to propose a more interpretable criterion.



Figure A.2. – Hybrids for MNIST-M (format: see Fig. A.1).

The MNIST-M dataset presents one known non-semantic feature: the background of the samples. We therefore verify experimentally that the background of hybrids generated by our procedure closely matches the background of their non-semantic parents (more closely than the one of their semantic parent) instead of considering z_r distances. By construction of MNIST-M, we know which pixels in images correspond to a digit and which correspond to a BSD 500 background: we know which MNIST sample was used to generate the sample. As such, we can have access to a mask that zeroes out pixels corresponding to the digit and does not alter background pixels for MNIST-M images. As qualitative studies (Figure A.2) suggest hybrid samples do correctly inherit digit outline from their semantic parent, we approximate the background of hybrids to be the same as that of their semantic parent. Therefore, we calculate the background of hybrid samples $b_h = m_1 * x_h$ by applying a mask m_1 that zeroes out pixels corresponding to the digit in the semantic parent (known by construction). The backgrounds b_1 and b_2 of the parent samples are also known by construction (corresponding to the BSD 500 backgrounds used to generate samples). Similarly to our previous procedure, if $d_l := \|b_h - m_1 * b_1\|_2^2 \geq d_r := \|b_h - m_1 * b_2\|_2^2$, then we conclude the hybrid correctly inherited its non-semantic parent’s style component.

The separation accuracies s_c and s_r as well as the distances between the hybrid’s background and its parents’ are given in Table A.13b. Results suggest a clear separation of semantic and non-semantic content in hybrids. The nature of the pixel distances tracked in this experiment strongly correlate the model’s notion of non-semantic features with the known background modularity as expected. As such, the results strongly suggest that at least in simple cases, SAMOSA is capable of correctly identifying and separating the semantic and non-semantic factors in training data.

A.3.3 Semantic content in z_r

In addition to verifying z_r does indeed cause non-semantic changes in reconstruction/hybridization, we also observe that it does not contain a lot of semantic information. Training a classification head (with all 50000 labeled training samples) on the non-semantic space of a trained Mean Teacher based SAMOSA model (1000 label CIFAR 10 setting) does lead to low classification accuracy (about 30%). This contrasts with the 88% accuracy obtained by a linear layer trained on the semantic space z_c with only 1000 labeled samples in previous experiments. As such, we verify that z_r does not extract specifically semantic features in accordance with SAMOSA’s design.

What little semantic information remains in z_r is ignored by the decoder D as generated hybrids only inherit the class of their non-semantic parents in about 10% of cases (random chance). Furthermore, we can verify that expunging semantic content from z_r is pointless. z_r can be made wholly non-semantic by training a linear classifier to classify from z_r , and training E_r to fool this classifier. While a model trained this way retains almost no semantic information in non-semantic space z_r (about 18% accuracy for a linear classifier trained on the frozen projection), such a model fails to better classify samples (accuracy of 88.5 ± 0.1 vs. 88.7 ± 0.3).

A.4 Additional experiments and material for SciMix

We provide the following additional experiments and material for MixShare:

- In [Section A.4.1](#), we discuss the difference in overhead between the SciMix and SAMOSA frameworks.
- In [Section A.4.2](#), we show how various variants of the SciMix generative framework perform.
- In [Section A.4.3](#), we show how various variants of the SciMix data augmentation perform.
- In [Section A.4.4](#), shows some more hybrids for SciMix on SVHN.

A.4.1 Overhead of SAMOSA and SciMix

Beyond interfering with augmentation process, training the semi-supervised classifier alongside the generative framework makes the training process itself significantly more complex. The main pain points with this are:

- **Additional training parameters when training the classifier** In the SAMOSA experiments, we typically train a 1.5M parameter WideResNet-28-2 classifier. Due to the generative framework, we must additionally train a 1.5M parameter non-semantic encoder and a 2M parameter decoder/generator. This represents over 3 times as many parameters, and requires training over two GPUs to obtain results in reasonable time (since SAMOSA must train three times to incorporate hybrids).

While SciMix does have to train a generator along an auxiliary classifier, it only needs to be trained once and we do not need refine the auxiliary classifier with additional hybrids. Training the actual semi-supervised model only involves the 1.5M parameters of WideResNet-28-2

- **Difficulty in synchronizing classifier and generator training** It must be noted that semi-supervised learning algorithms tend to use fairly diverse training settings, with some of them running fairly long. For reference, we train Mean Teacher over 300 epochs with SGD, MixMatch over 900 epochs with AdamW and FixMatch over 1000 epochs using SGD. The training settings for MixMatch and FixMatch in our experiments are in fact already shortened by a factor of 10. The generative framework for its part trains well with SGD over 300 epochs, with similar settings as the Mean Teacher framework. Beyond the fact it largely increases the already significant training

times of FixMatch and MixMatch, the generative modules do not adapt well to their training settings.

SciMix’s use of an auxiliary classifier solves this problem for the most part: we always use a Mean Teacher algorithm to train the generative model. While this means we essentially use a weaker model to teach a stronger model on the FixMatch backbone, experimental results suggest the hybrids still provide useful regularization. Interestingly, as the auxiliary classifier still benefits from the reconstruction/hybridization process, it provides stronger performance than a standard Mean Teacher algorithm. On another note, we could have trained the auxiliary classifier with a FixMatch algorithm on very shortened schedule as our goal is not to get the best possible performance out of the auxiliary classifier.

A.4.2 Model analysis: Importance of the learning scheme


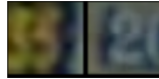
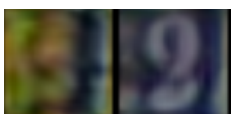
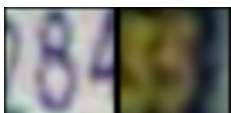
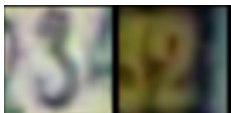
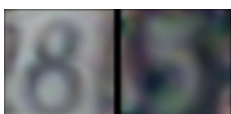
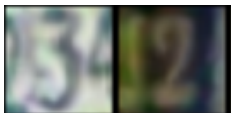
Hybrids parent pairs x_r/x_c					
Method	Accuracy	s_c	s_r	sample hybrids	
Structural z_c	39.5 ± 9.3	16.0	56.0		
No \mathcal{L}_{hyb}	48.5 ± 14.2	11.7	99.8		
Basic \mathcal{L}_{hyb}	66.3 ± 1.0	66.2	99.4		
Non Frozen criterion \mathcal{L}_{hyb}	81.8 ± 3.0	75.1	73.8		
Full Setup	83.4 ± 0.5	76.7	98.8		

Table A.14. – **Comparison of various architectural variants** (Section 2.4.1) along with samples of generated hybrids for each variant on SVHN 60 labels.

To better explore how our framework facilitates the incrustation of semantic content in the general context of existing samples, we first propose a rapid ablation study on the quality of the samples generated by variants of our auto-encoder on our hardest SVHN setting (60 labels). We evaluate this through the classification accuracy of models trained with the generator’s mixed samples, the semantic transfer s_c , the non-semantic transfer s_r , and a visual evaluation of two hybrids (e.g. the leftmost hybrid mixes a blue 8 x_c with a yellow 3 x_r).

We consider 4 variations on SciMix’s generator to demonstrate the merits of our chosen method: **z_r as a style code**. We flip the roles of z_c and z_r , to demonstrate z_c is better used as a style code in SciMix. **No \mathcal{L}_{hyb}** . We train without an explicit optimization loss, to show \mathcal{L}_S and \mathcal{L}_{rec} are not sufficient to create good hybrids in SciMix. **Basic \mathcal{L}_{hyb}** . We demonstrate the orthogonalization losses $\mathcal{L}_{hyb,class}^-$ and $\mathcal{L}_{hyb,cont}^-$ contribute to the generator by considering a variant that does not optimize them. **No frozen criterion \mathcal{L}_{hyb}** . We do not force the generator to only optimize the generation of the hybrids when optimizing \mathcal{L}_{hyb} (the projection of the hybrids in the latent spaces is also modified).

Table A.14 shows that without explicit hybridization optimization, the model fails to properly transfer semantic characteristics (low s_c score). Since the model does properly transfer semantic content with only $\mathcal{L}_{hybrids}^+$, the addition of the orthogonalization constraints $\mathcal{L}_{hybrids}^-$ is not necessary to obtain useful hybrids. However, this orthogonalization increases the diversity in the generated hybrids and therefore leads to a better augmentation procedure. Not freezing the projection heads when training for hybridization on the other hand leads to a general deterioration of the training process and can therefore be felt in both transfer rates. Predictably, using z_r as a global style code leads a very poor correspondence of hybrids to their non-semantic parents as things like backgrounds can become very complicated to reproduce with a modulation based generator.

A.4.3 Model analysis: Leveraging the hybrids as Data Augmentation

We considered 4 alternative methods to exploit the hybrids generated by our method: **Labeled** Only hybrids with a labeled semantic parent are considered (supervised training with a hard label) **Pseudo-label** Hybrids are treated as labeled samples (hard labels), with the labels inherited from the semantic parent’s pseudo-labels. **Consistency** Hybrids are made to follow their semantic parent’s prediction. **Contradict** Hybrids are made to match both their semantic parent’s consistency target and their non-semantic parent targets.

Method	Accuracy
Baseline	48.7 ± 23.0
$\mathcal{L}^{\text{labeled}}$	29.9 ± 17.6
$\mathcal{L}^{\text{pseudo-label}}$	62.8 ± 5.4
$\mathcal{L}^{\text{consistency}}$	77.8 ± 4.8
$\mathcal{L}^{\text{contradict}}$	83.4 ± 0.4

Table A.15. – Comparison of various losses to leverage hybrids in $\mathcal{L}_{\text{SciMix}}$ (Section 2.4.2) on SVHN 60 labels.

Table A.15 shows that the best results are obtained with $\mathcal{L}^{\text{contradict}}$, but $\mathcal{L}^{\text{pseudo-label}}$ and $\mathcal{L}^{\text{cons}}$ also outperform the baseline method on SVHN 60 labels (hardest setting). The fact $\mathcal{L}^{\text{contradict}}$ outperforms other methods is interesting in that the loss does not actually treat the hybrids as pure labeled samples, but assumes some semantic content/noise is retained from the non-semantic samples. $\mathcal{L}^{\text{labeled}}$'s failure suggests that even with proper mixing, it not possible to improve models by only generalizing a few labeled samples.

A.4.4 Additional hybrids

We provide additional examples of hybrids on SVHN in Figure A.3.



Figure A.3. – Additional SciMix hybrids on SVHN (100 labels).

A.5 Additional experiments and material for MixMo

We provide the following additional experiments and material for MixMo:

- In [Section A.5.1](#), we provide a quick refresher on common MSDA techniques.
- In [Section A.5.2](#), we clarify our framework generalization with $M > 2$ sub-networks.
- In [Section A.5.3](#), we illustrate the reweighting of the loss components.
- In [Section A.5.4](#), we elaborate on our analysis of filters activity.
- In [Section A.5.5](#), we analyze ensembles of Cut-MixMo with CutMix that reach state of the art.
- In [Section A.5.6](#), we provide a preliminary study of MixMo on ImageNet.
- In [Section A.5.7](#), we study the importance of α .

A.5.1 Mixed sample data augmentations

We have drawn inspiration from MSDA techniques to design our mixing block \mathcal{M} . In particular, [Section 3.3.5.2](#) compared different \mathcal{M} based on recent papers. [Figure A.4](#) provides the reader a visual understanding of their behaviour, which we explain below.

MixUp (H. Zhang et al. 2018) linearly interpolates between pixels: $m_x(x_i, x_k, \lambda) = \lambda x_i + (1 - \lambda)x_k$. The remaining methods fall under the label of **binary MSDA**: $m_x(x_i, x_k, \lambda) = \mathbf{1}_m \odot x_i + (\mathbf{1} - \mathbf{1}_m) \odot x_k$ with $\mathbf{1}_m$ a mask with binary values $\{0, 1\}$ and area of ratio λ . They diverge in how this mask is created. The **horizontal concatenation**, also found in (Summers and Dinneen 2019), simply draws a vertical line such that every pixel to the left belongs to one sample and every pixel to the right belongs to the other. Similarly, we define a **vertical concatenation** with an horizontal line. **PatchUp** (Faramarzi et al. 2020) adapted DropBlock (Ghiasi et al. 2018): a canvas C of patches is created by sampling for every spatial coordinate from the Bernoulli distribution $\text{Ber}(\lambda')$ (where λ' is a recalibrated value of λ): if the drawn binary value is 1, a patch around that coordinate is set to 1 on the final binary mask $\mathbf{1}_m$. PatchUp was designed for in-manifold mixing with a different mask by channels. However, duplicating the same 2D mask in all channels for \mathcal{M} performs better in our experiments. **FMix** (Harris et al. 2020) selects a large contiguous region in one image and pastes it onto another. The binary mask is made

of the top- λ percentile of pixels from a low-pass filtered 2D map G drawn from an isotropic Gaussian distribution. **CowMix** (G. French et al. 2020a; G. French et al. 2020b) selects a cow-spotted set of regions, and is somehow similar to FMix with a Gaussian filtered 2D map G . **CutMix** (Y. Yang and Soatto 2019) was inspired by CutOut (DeVries and Taylor 2017). Formally, we sample a square with edges of length $R\sqrt{\lambda}$, where R is the length of an image edge. Note that this sometimes leads to non square rectangles when the initially sampled square overlaps with the edge from the original image. We adjust our λ a posteriori to fix this boundary effect. Regarding the hyper-parameters, we use in \mathcal{M} those provided in the seminal papers, except for sampling of κ where we set $\alpha = 2$ in all setups.

Note we consider both versions of MixUp (in-pixel and manifold) in this paper, but only the in-pixel version of CutMix. Indeed, the manifold version of CutMix was shown in the seminal CutMix paper (Y. Yang and Soatto 2019) to be inferior to the standard in-pixel variant.



Figure A.4. – Common MSDA procedures with $\lambda = 0.5$.

A.5.2 Generalization to $M > 2$ heads

We have mostly discussed our MixMo framework with $M = 2$ subnetworks. For better readability, we referred to the mixing ratios κ and $1 - \kappa$ with $\kappa \sim \text{Beta}(\alpha, \alpha)$. It's equivalent to a more generic formulation $(\kappa_0, \kappa_1) \in \text{Dir}_2(\alpha)$ from a symmetric Dirichlet distribution with concentration parameter α . This leads to the alternate equations $\mathcal{L}_{\text{MixMo}} = \sum_{i=0,1} w_r(\kappa_i) \mathcal{L}_{\text{CE}}(y_i, \hat{y}_i)$, where $w_r(\kappa_i) = 2 \frac{\kappa_i^{1/r}}{\sum_{j=0,1} \kappa_j^{1/r}}$.

Now generalization to the general case $M \geq 2$ is straightforward. We draw a tuple $\{\kappa_i\}_{0 \leq i < M} \sim \text{Dir}_M(\alpha)$ and optimize the training loss:

$$\mathcal{L}_{\text{MixMo}} = \sum_{i=0}^{M-1} w_r(\kappa_i) \mathcal{L}_{\text{CE}}(y_i, \hat{y}_i), \quad (\text{A.1})$$

where the new weighting naturally follows:

$$w_r(\kappa_i) = M \frac{\kappa_i^{1/r}}{\sum_{j=0}^{M-1} \kappa_j^{1/r}}, \forall i \in \{0, \dots, M-1\}. \quad (\text{A.2})$$

The remaining point is the generalization of the mixing block \mathcal{M} , that relies on the existence of MSDA methods for $M > 2$ inputs. The linear interpolation can be easily expanded as in Mixup:

$$\mathcal{M}_{\text{Linear-MixMo}}(\{l_i\}) = M \sum_{i=0}^{M-1} \kappa_i l_i, \quad (\text{A.3})$$

where $l_i = c_i(x_i)$. However, extensions for other masking MSDAs have only recently started to emerge (J. Kim et al. 2021). For example, CutMix is not trivially generalizable to $M > 2$, as the patches could overlap and hide important semantic components. In our experiments, a soft extension of Cut-MixMo performs best: it first linearly interpolates $M - 1$ inputs and then patches a region from the M -th:

$$\mathcal{M}_{\text{Cut-MixMo}}(\{l_i\}) = M[\mathbf{1}_{\mathcal{M}} \odot l_k + (\mathbf{1} - \mathbf{1}_{\mathcal{M}}) \odot \sum_{i=0, i \neq k}^{M-1} \frac{\kappa_i}{1 - \kappa_k} l_i], \quad (\text{A.4})$$

where $\mathbf{1}_{\mathcal{M}}$ is a rectangle of area ratio κ_k and k sampled uniformly in $\{0, 1, \dots, M-1\}$. However, it has been less successful than $M = 2$, as only two subnetworks can fit independently in standard parameterization regimes. Future work could design new framework components, such as specific mixing blocks, to tackle these limits.

A.5.3 Weighting function w_r

As outlined in [Section 3.3.2.2](#), the asymmetry in the mixing mechanism leads to asymmetry in the relative importance of the two inputs. Thus we reweight the loss components with function w_r , defined as $w_r(\kappa) = 2 \frac{\kappa^{1/r}}{\kappa^{1/r} + (1-\kappa)^{1/r}}$. It rescales the mixing ratio κ through the use of a $\frac{1}{r}$ root operator. In the main paper, we have focused on $r = 3$.

[Figure A.5](#) illustrates how w_r behaves for $r \in \{1, 2, 3, 4, 10\}$ and $r \rightarrow \infty$. The first extreme $r = 1$ matches the diagonal $w_r(\kappa) = 2\kappa$, without rescaling of κ , similarly to what is customary in MSDA. Our experiments in [Section 3.3.5.3](#) justified the initial idea to shift the weighting function closer to the horizontal and constant curve $w_r(\kappa) = 1$ with higher r . In the other experiments, we always set $r = 3$.

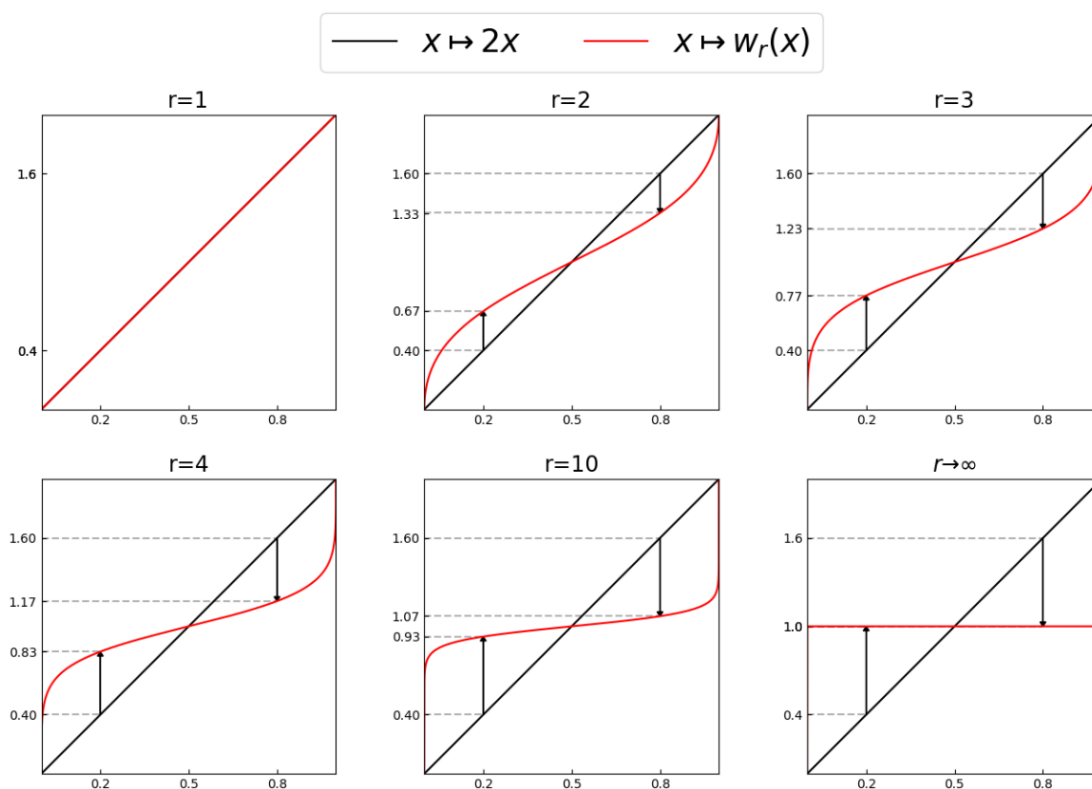


Figure A.5. – Curves of the reweighting operation that projects κ to the flattened ratio $w_r(\kappa)$

A.5.4 Filters activity

We argued in [Section 3.3.3](#) that MixMo better leverages additional parameters in wider networks. Concretely, a larger proportion of filters in large networks

Method	Width	$t_a = 0.2$	$t_a = 0.3$	$t_a = 0.4$	$t_a = 0.5$
Vanilla	2	98.9	98.8	97.8	93.3
	3	97.3	96.4	93.2	87.5
	4	96.5	95.2	91.2	81.6
	5	95.1	91.7	85.7	73.3
	7	92.6	88.2	81.0	69.5
	10	87.8	80.4	71.5	57.3
	14	83.9	74.0	61.6	46.8
CutMix	2	99.2	99.0	97.8	95.3
	3	98.7	98.5	97.2	93.4
	4	98.1	97.4	94.0	87.3
	5	97.0	96.1	90.7	80.6
	7	95.8	94.0	86.2	74.6
	10	93.5	88.4	81.3	67.0
	14	89.4	81.9	70.3	50.9
Cut-MixMo	2	100.0	100.0	99.4	97.3
	3	99.8	99.8	99.7	98.7
	4	99.7	99.7	99.6	98.7
	5	99.3	99.3	98.9	97.4
	7	98.9	98.8	98.0	95.2
	10	98.5	98.2	96.8	92.4
	14	97.5	96.3	93.1	82.6

Table A.16. – **Proportion (%) of active filters** in core network vs. width w for a WRN-28- w on CIFAR 100 and different activity thresholds t_a .

really help for classification as demonstrated in [Figure 3.4a](#) and [Figure 3.4b](#) in the main paper. Following common practices in the structured pruning literature (H. Li et al. 2017), we used the l_1 -norm of convolutional filters as a proxy for importance. These 3D filters are of shape $n_i \times k \times k$ with n_i the number of input channels and k the kernel size. In [Figure 3.4b](#), we arbitrarily defined a filter as active if its l_1 -norm is at least 40% of the highest filter l_1 -norm in that filter’s layer. We report the average percentage of active filters across all filters in the core network \mathcal{C} , for 3 learning strategies: vanilla, CutMix and Cut-MixMo.

The threshold $t_a = 0.4$ was chosen for visualization purposes. Nevertheless, the observed trend in activity proportions remains for varying thresholds in [Table A.16](#). For example, for the lax $t_a = 0.2$, CutMix uses 93.5% of filters vs. 98.5% for Cut-MixMo.

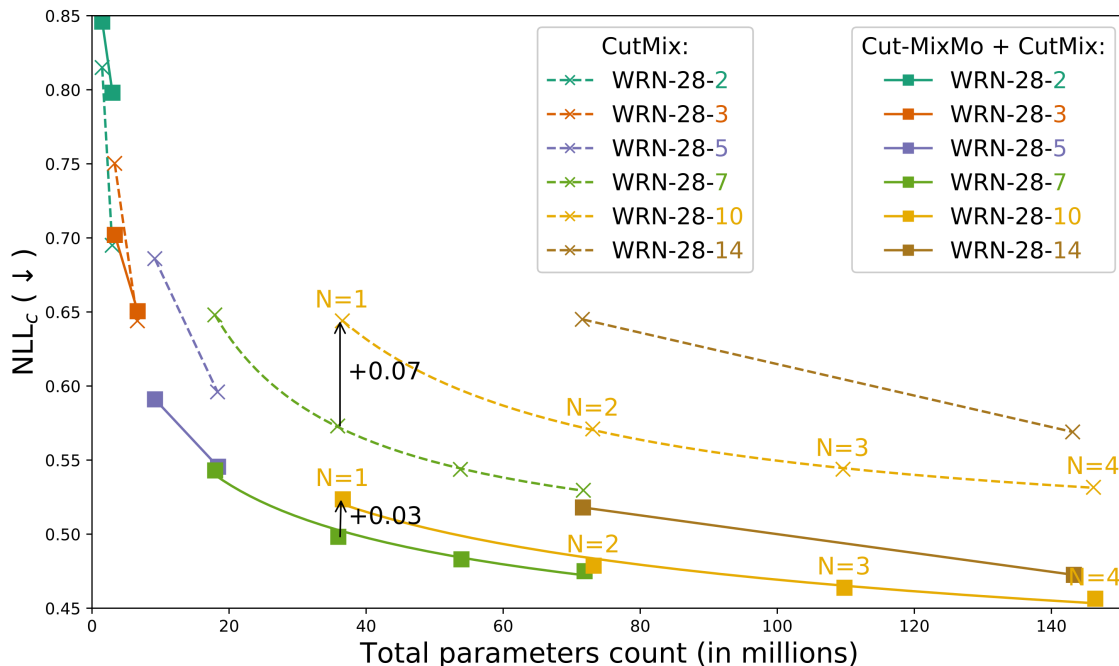


Figure A.6. – **Ensemble effectiveness** ($NLL_c/\#\text{params}$). We slide the width in WRN-28- w and numbers of members N . CutMix data augmentation. Interpolations through power laws (Lobacheva et al. 2020) when more than 2 points are available.

A.5.5 Ensemble of Cut-MixMo with CutMix

Figure A.6 plots performance for different widths w in WRN-28- w and varying number of ensembled networks N : two vertically aligned points have the same parameter budget. Indeed, the total number of parameters in our architectures has been used as a proxy for model complexity, as in (Chirkova et al. 2020; Lobacheva et al. 2020). The increase in the total number of weights in MixMo is visually almost unnoticeable. Precisely, with WRN-28-10, MixMo ($M=2$) has 36.60M weights vs. 36.53M standardly (+0.2%). Moreover, the number of flops is 5.9571G Flops for MixMo vs. 5.9565G Flops standardly (+0.01%). That’s why we state we achieve ensembling (almost) “for free”.

We compare ensembling with CutMix rather than standard pixels data augmentation, as previously done in 3.7 from Section Section 3.3.4.4. CutMix induces additional regularization and label smoothing: empirically, it improves all our approaches. For a fixed memory budget, a single network usually performs worse than an ensemble of several medium-size networks: we recover the **Memory Split Advantage** even with CutMix. However, Cut-MixMo challenges this by remaining closer to the lower envelope. In other words, parameters allocation (more net-

works or bigger networks) has less impact on results. This is due to Cut-MixMo’s ability to better use large networks.

In Table A.17, we summarize several experiments on CIFAR-100. Among other things, we can observe that large vanilla networks tend to gain less from ensembling (Lobacheva et al. 2020): *e.g.* 2 vanillas WRN-28-10 (83.17% Top1, 0.668 NLL_c) do not perform much better than 2 WRN-28-7 (82.94%, 0.673). This remains true even with CutMix: (85.74%, 0.571) vs. (85.52%, 0.573). We speculate this is related to wide networks’ tendency to converge to less diverse solutions, as studied in (Neal et al. 2018). Contrarily, **MixMo improves the ensembling of large networks**, with (86.04%, 0.494) vs. (85.50%, 0.517) on the same setup. When additionally combined with CutMix, we obtain state of the art (86.63%, 0.479) vs. (85.90%, 0.498). This demonstrates the importance of Cut-MixMo in cooperation with standard pixels data augmentation. It attenuates the drawbacks from over-parameterization This is of great importance for practical efficiency: it modifies the optimal network width for real-world applications.

Width w	Approach CutMix	1-Net		2-Nets		Linear-MixMo		Cut-MixMo		2-Cut-MixMos	
		-	✓	-	✓	-	✓	-	✓	-	✓
2	Top1	76.44	78.06	79.16	80.81	75.82	76.36	75.66	75.17	76.98	76.11
	NLL _c	0.921	0.815	0.776	0.695	0.841	0.824	0.824	0.846	0.7661	0.798
	# params	1.48M		2.95M		1.49M		2.99M			
3	Top1	77.95	80.70	80.85	83.14	78.51	80.74	79.81	79.85	80.78	81.20
	NLL _c	0.862	0.750	0.738	0.644	0.760	0.696	0.693	0.702	0.635	0.650
	# params	3.31M		6.62M		3.33M		6.66M			
4	Top1	78.84	81.55	81.48	83.93	80.43	81.66	81.68	81.69	82.57	82.58
	NLL _c	0.824	0.711	0.711	0.609	0.712	0.656	0.646	0.635	0.590	0.588
	# params	5.87M		11.74M		5.89M		11.79M			
5	Top1	79.75	82.55	82.18	84.60	80.95	83.06	83.11	83.34	83.97	84.31
	NLL _c	0.813	0.686	0.693	0.596	0.703	0.617	0.598	0.591	0.549	0.546
	# params	9.16M		18.32M		9.19M		18.39M			
7	Top1	81.14	83.71	82.94	85.52	82.4	84.51	84.32	84.94	85.50	85.90
	NLL _c	0.764	0.648	0.673	0.573	0.675	0.581	0.562	0.543	0.516	0.498
	# params	17.92M		35.85M		17.97M		35.94M			
10	Top1	81.63	84.05	83.17	85.74	83.08	85.47	85.40	85.77	86.04	86.63
	NLL _c	0.750	0.644	0.668	0.571	0.656	0.558	0.535	0.524	0.494	0.479
	# params	36.53M		73.07M		36.60M		73.21M			
14	Top1	82.01	84.31	83.47	85.80	83.79	86.05	85.76	86.19	86.58	87.11
	NLL _c	0.730	0.645	0.656	0.569	0.648	0.545	0.527	0.518	0.488	0.473
	# params	71.55M		143.1M		71.64M		143.28M			

Table A.17. – Summary: WRN-28- w on CIFAR-100. $b = 4$.

A.5.6 Preliminary ImageNet experiments

To further prove MixMo’s ability to scale to more complex problems, we also conduct a preliminary study of its behavior on the larger scale ImageNet dataset

(J. Deng et al. 2009). Following the protocol outlined in the seminal MIMO paper (Havasi et al. 2021), we consider variations on the standard ResNet-18 in the form of ResNet-18- w networks where w is multiplicative width factor.

These first experiments confirm that MixMo performs well when networks are overparameterized. For values of $w \geq 5$, our network at the end of training outperforms both Vanilla and CutMix baselines. For example, with a ResNet-18-5 backbone, Cut-MixMo (78.20% Top1, 0.867 NLLc) improves over Vanilla (76.47%, 1.121) and CutMix (77.40%, 1.263). This remains the case for a ResNet-18-7 backbone with Cut-MixMo (78.55% Top1, 0.846 NLLc) outperforming Vanilla (76.86%, 1.100) and CutMix (77.18%, 1.190).

A.5.7 Hyper-parameter α

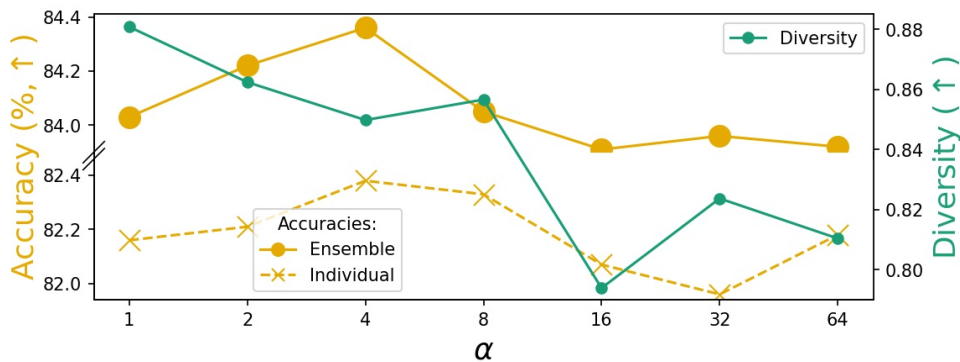


Figure A.7. – Diversity/accuracy as function of α .

In Fig. Figure A.7, we study the impact of different values of α , parameterizing the sampling law for $\kappa \sim \text{Beta}(\alpha, \alpha)$. For high values of α , the interval of κ narrows down around 0.5. Diversity is therefore decreased: we speculate this is because we do not benefit anymore from lopsided updates. The opposite extreme, when $\alpha=1$, is equivalent to uniform distribution between 0 and 1. Therefore diversity is increased, at the cost of lower individual accuracy due to less stable training. For simplicity, we set $\alpha=2$. Manifold-Mixup (Verma et al. 2019a) selected the same value on CIFAR-100. However, this value could be fine tuned on the target task: e.g. in Figure A.7, $\alpha=4$ seems to perform best for Cut-MixMo on CIFAR-100 with WRN-28-10 with $r=3$, $p=0.5$ and $b=2$.

A.6 Additional experiments and material for MixShare

We provide the following additional experiments and material for MixShare:

- In [Section A.6.1](#), we provide additional details regarding the adjustments necessary to implement the MixShare framework.
- In [Section A.6.2](#), we discuss the kernel initialization at more length.
- In [Section A.6.3](#), show how the features in the core network influence each of the subnetworks.

A.6.1 Complementary adjustments to MIMO procedures in MixShare

MIMO methods use a number of auxiliary procedures to train strong subnetworks. However, as MixShare differs significantly from standard MIMO frameworks, it does not use these frameworks to the same extent.

CutMix probability in the input block MixMo (Rame et al. 2021) only uses cutmix mixing in its input block about half the time, using a basic summing operation on the two encoded inputs the rest of the time. This is because the model will use a summing operation at test time. Therefore, the use of cutmix at training induces an strong train/test gap that needs to be bridged by the use of summing during training.

We cannot afford to use summing half the time as unmixing relies on the use of cutmix in the input block. However, since our two encoders are very similar (due to our kernel alignment), cutmix and summing (or averaging) behave very similarly and the train/test gap is therefore minimal.

Input Repetition A slight train/test gap still remains however since the model is rarely presented the same image as input to both subnetworks at training time. We solve this by reprising a procedure introduced in the seminal MIMO paper (Havasi et al. 2021): input repetition. In our case, we ensure 10% of inputs of our batches are made of repetition of the same image during training.

Loss rebalancing MixMo (Rame et al. 2021) introduced a re-weighting function of the subnetwork training losses that rescales the mixing ratios used in the inputs block. These ratios are rescaled to be less lopsided (closer to an even 50/50

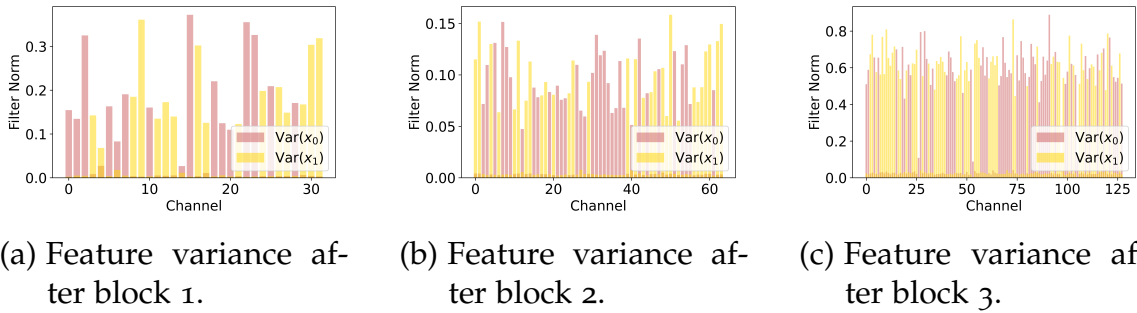


Figure A.8. – **Influence of feature maps in the core network.** Checking the variance of feature maps w.r.t. the two inputs at different levels of the network shows clear separation of features in standard multi-input multi-output architectures.

split) before being applied to their relevant subnetwork losses. This rescaling is necessary as it ensures all parameters receive sufficient training signal.

We however find in our experiments it is more beneficial to do away with this re-balancing and keep the original mixing ratios, which we explain by the large amount of features shared between subnetworks. Since features are shared, we do not need to worry about some features receiving too little training signal.

A.6.2 A more nuanced discussion on kernel alignment

While MixShare uses the exact same initialization of the encoder kernels for simplicity, it is interesting to note much weaker versions of kernel alignment are sufficient to obtain similar results.

Indeed, we found in our experiments that initializing the kernels to be simply co-linear is more than enough to ensure proper feature sharing. In fact, this leads to the exact same performance as using the same initialization and the encoder kernels quickly converge to similar values.

This further validates our intuition that MIMO models need a “common language” to benefit from sharing features: all that is required is for encoder kernels to extract the same “type” of features.

A.6.3 Analysis of subnetwork features within the core network

Section 3.4.1 studies what features each subnetwork uses in the input block and output block of the multi-input multi-output model. Studying the importance of features within the core networks for each subnetworks is more difficult as it is not

possible to consider the model weights. Reprising an analysis conducted in the Appendix of (Rame et al. 2021), we identify the influence of intermediate features on subnetworks with the variance of the feature with respect to the relevant input.

For the first subnetwork, if we consider the intermediate feature map (at one point in the network f) $\mathcal{M}_{int} = f_{int}(\mathcal{D}_{test}, d)$, such that \mathcal{M}_{int} is of shape $N \times C \times H \times W$ with \mathcal{D}_{test} the test set, d a fixed input, N the size of the test set, C the number of intermediate feature maps and $H \times W$ the spatial coordinates. We compute the importance of each of the C feature map with respect to the first subnetwork as $Mean(Var(\mathcal{M}_{int}, dim = 0), dim = (1, 2))$. The importance of intermediate features for the second subnetwork is obtained similarly by considering $\mathcal{M}_{int} = f_{int}(d, \mathcal{D}_{test})$.

Figure A.8 shows the resulting feature importance maps at after each of the three residual blocks in the core network. As can be observed, the subnetworks remain consistently separated in the core network.

A.7 Additional experiments and material for MixViT

We provide the following additional experiments and material for MixViT:

- In [Section A.7.1](#), we provide additional figures for [Section 4.3.1](#).
- In [Section A.7.2](#), we discuss how initialization scheme would interact with MIMO formulations of ViTs, and how it relates to MixViT’s mechanisms.
- In [Section A.7.3](#), we provide further notes on the overhead incurred by MixViT.
- In [Section A.7.4](#), we provide additional comparison of MixViT with MixViT on the CIFAR-100 dataset (which is much harder for vision transformers and CNNs).

A.7.1 Illustration of MIMO and MixMo transformers

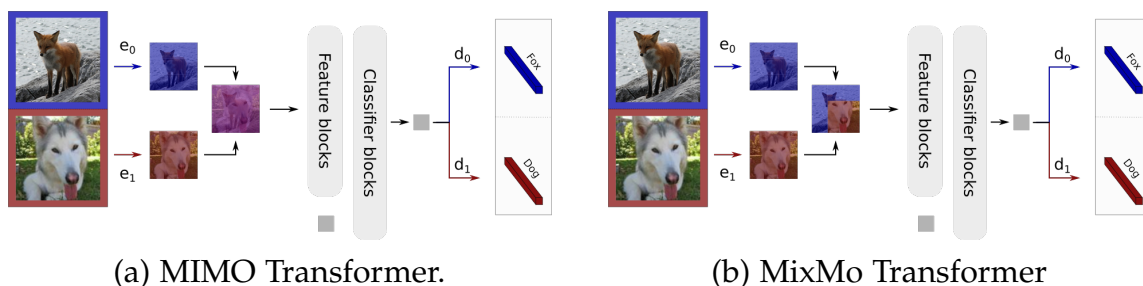


Figure A.9. – **Transposition of traditional MIMO architectures to vision transformers.**

[Figure A.9](#) shows how we transpose the seminal MIMO and MixMo approaches to vision transformers in [Section 4.3.1](#). As can be seen, the network embeds the images with separate encodings before mixing them into a single representation and proceeding normally.

A.7.2 On encoder initialization and MixViT

In [Section 4.3.1](#), we observe ViTs have trouble accommodating traditional MIMO techniques like the seminal MIMO (Havasi et al. [2021](#)) and MixMo (Rame et al. [2021](#)) frameworks. We find performing two parallel forwards at inference seems

Backbone	Method	Inference Mix.	# FLOPS	Accuracy (%)	Sub network Acc. (%)
ConViT	Single-input (D’Ascoli et al. 2021)	N/A	1×	79.5 ± 0.1	-
	MixMo (Rame et al. 2021)	Sum	1×	77.0 ± 0.2	75.2 ± 0.1
	MixMo (Rame et al. 2021)	Separate	2×	78.5 ± 0.2	76.4 ± 0.1
	MixMo + same init.	Sum	1×	77.1 ± 0.3	75.2 ± 0.3

Table A.18. – **Impact of initialization when transposing MIMO transformers.** CIFAR-100 accuracy and estimated FLOPS of the backbone and MIMO formulations.

to address the issue, which suggests Vision Transformers have trouble sharing tokens between inputs or subnetworks.

While the exact source of this behavior is not clear, the main difference between MIMO ViTs and MIMO CNNs seems to be that ViT subnetworks seem to naturally share features. This feature sharing in itself can likely be explained by the similarity between the unmixing mechanism we develop for MIMO CNNs in [Section 3.4](#) and the way the attention mechanism selectively aggregates information from the patch tokens into the class token. Interestingly, we observe in MixShare that such feature sharing strongly deteriorates model performance unless the input encoders are initialized to the same value. One might therefore reasonably assume reprising this initialization scheme could offer a simpler solution to the issue of MIMO ViTs.

We verify in [Table A.18](#) that - unlike in MixShare - initializing the patch embedding layers e_0 and e_1 (see [Figure A.9b](#)) to the same value does not solve the issue on its own. As such, the core problem is different than the one observed in MixShare and we do need to perform separate inference on MIMO and MixViT Vision Transformers.

It is worth noting however both source attribution in MixViT and encoder initialization in MixShare help the model rely on a largely shared feature representation: they both use a common encoder and add some subnetwork specific information. In MixViT, this directly follows from the definition of source attribution where we add a source attributed embedding or encoding to the general token representation.

The same thing arguably happens when using two different encoders initialized to the same value like in MixShare so long as the encoders weights remain close through training. Indeed, the linear weights (linear convolution kernels in CNNs) W_0 and W_1 can be decomposed as $W_0 = W + W_0^r$ and $W_1 = W + W_1^r$ with (for instance) $W_0^r = \frac{W_0 - W_1}{2}$, $W_1^r = \frac{W_1 - W_0}{2}$ and $W = W_0 - \frac{W_0 - W_1}{2} = \frac{W_0 + W_1}{2}$. If W_0 and W_1 are close, then the encoders can be thought to apply the same core weight W with the addition of smaller contributions from weights W_0^r and W_1^r . This approach is

nevertheless much more difficult to control than the one introduced in MixViT, which could be adapted to CNNs and help address the issues of MixShare.

A.7.3 Further notes on MixViT overhead

We provide in [Section 4.3.3.2](#) a very rough upper bound on the additional overhead incurred by MixViT at inference for one forward evaluation. We provide here a more nuanced discussion on the computations at play and show that - for $M = 2$ or at least bounded - the asymptotic complexity in L , N and d of MixViT is the same as that of a normal model.

Similarly to [Section 4.3.3.2](#), we consider a transformer on a C -class problem with L self-attention blocks and 2 class-attention blocks taking N d -dimensional patch tokens. We neglect the computational overhead of induced by adding bias terms, embedding the patches into tokens, classifying from the classification tokens features, normalization and source attribution.

How many computations do class-attention and self-attention need ? We assumed previously class-attention and self-attention are as expensive computationally which leads to a very pessimistic upper bound on the overhead.

Let us count the important operations in a Self-Attention layer:

1. We have to extract key, query and value representations for each token which causes $3Nd^2$ operations due to $3N$ weight multiplications.
2. We have to compute the similarity scores between N tokens which means N^2d operations.
3. We must compute the attended representation for all tokens which costs approximately N^2d operations as one attended token requires adding N d -dimensional tokens (we take the cost of multiplying the value representation by the attention weight to be atomic, but it is technically d).
4. Most implementations include a projection step to properly aggregate the multiple attention heads which means an additional Nd^2 cost.
5. Each token is then processed by the same 1 hidden layer multi-layer perceptron. While the hidden dimension of the perceptron can vary it is standard in the literature to take $4d$. Therefore we have to consider $2 \times 4 \times Nd^2$ operations as we consider 2 weights multiplications of dimensions $d \times 4d$.

This yields a cost of approximately $3Nd^2 + N^2d + N^2d + Nd^2 + 8Nd^2 = 13Nd^2 + 2N^2d$ in a Self-Attention layer whereas the cost of a Class-Attention layer is about $3Nd^2 + Nd + Nd + d^2 + 8d^2 = (3N + 9)d^2 + 2Nd$ operations. Indeed, Class-Attention follows a similar outline with the following differences: in 2) we only have Nd

operations (similarity to the class token), in 3) we only compute one attended representation for cost Nd , in 4) we only compute the projection for one token at cost d^2 and in step 5) we only apply the MLP to one token for $8d^2$ operations.

If we throw out the terms linear in d (as it is usually the largest term by far), Class-Attention therefore costs $\frac{3N+9}{13N} \times$ as many operations as Self-Attention layer. Note that - to the best of our knowledge - $N = 64$ at the least in the literature, so this ratio is closer at a maximum $0.25 \times$.

Tighter approximation of MixViT’s inference overhead We can now revise the very pessimistic approximation given Section 4.3.3.2 provided the estimations that Class-Attention costs at most $0.25 \times$ as many computations as Self-Attention. One forward evaluation of MixViT costs approximately $\frac{L+0.5M}{L+0.5} \times$ as many operations with this. For our modified ConViT, this indicates a ratio of only $1.1 \times$.

Asymptotic complexity in L , N and d Asymptotically, for a fixed number of subnetworks ($M=2$ usually), the complexity of MixViT and the underlying transformers are identical and equal to $\mathcal{O}(L(Nd^2 + N^2d))$. Indeed, as can be seen from the previous calculations, Self-Attention has a complexity of $\mathcal{O}(N^2d + Nd^2)$ and Class-Attention $\mathcal{O}(Nd^2)$. A normal forward pass is therefore on the order $\mathcal{O}(L(Nd^2 + N^2d))$ as we have $L + 2$ attention layers. MixViT-encoding induces M computations on the same order in the form $\mathcal{O}(Nd^2)$ source encodings. MixViT-encoding’s complexity is therefore $\mathcal{O}((L + M)Nd^2 + LN^2d)$ or $\mathcal{O}(L(Nd^2 + N^2d))$ if we consider M bounded.

A.7.4 Further comparison to MixMo on CIFAR-100

Table 4.3 shows MixViT matches the similarly sized WideResNet-28-5 (Zagoruyko and Komodakis 2016) on CIFAR-100 when given the same training budget. While MixMo’s (Rame et al. 2021) reported WideResNet-28-5 performance is better than our default setting MixViT performance, MixMo models train over 300 epochs with batch repetition 4. Batch repetition functions very

Models	# Params	Accuracy (%)
WideResNet-28-5*	9M	82.6
WideResNet-28-5 MixMo*	9M	83.3
ConViT (D’Ascoli et al. 2021)	12M	79.5 ± 0.1
MixViT-embedding w/ ConViT	12M	82.4 ± 0.1
MixViT-encoding w/ ConViT	12M	82.1 ± 0.1
ConViT* (D’Ascoli et al. 2021)	12M	81.4 ± 0.2
MixViT-embedding w/ ConViT*	12M	82.6 ± 0.2
MixViT-encoding w/ ConViT*	12M	83.2 ± 0.2

Table A.19. – **Comparison between MixMo and MixViT on CIFAR-100.** * indicates 300 training epochs with 4 batch augmentations.

similarly to Batch augmentation and is often needed to obtain good results with CNN-based MIMO models (which is less so the case for MixViT as per [Section 4.4.3](#)). If we reduce the learning rate and train with 4 batch augmentations, we close the gap in performance. This is particularly noteworthy as vision transformers are typically disadvantaged against CNNs on the CIFAR datasets as can be seen by the fact our baseline single-input single-output ConViT model only reaches 81.1% accuracy (vs. 82.6%) even on this longer training schedule.

