Convolutional network fabric pruning with label noise

Ilias Benjelloun, Bart Lamiroy, Efoevi Angelo Koudou

Convolutional Network Fabric Pruning With Label Noise

Ilias Benjelloun[1*], Bart Lamiroy[2,1*†] and Efoevi Angelo Koudou[3†]

[1*]Université de Lorraine, CNRS, LORIA, F-54000, Nancy, France.
[2*]Université de Reims Champagne Ardenne, CReSTIC EA 3804, F-51100, Reims, France.
[3]Université de Lorraine, CNRS, IECL, F-54000, Nancy, France.

*Corresponding author(s). E-mail(s): ilias.benjelloun@mines-nancy.org; bart.lamiroy@univ-reims.fr;
†These authors contributed equally to this work.

Abstract

This paper presents an iterative pruning strategy for Convolutional Network Fabrics (CNF) in presence of noisy training and testing data. With the continuous increase in size of neural network models, various authors have developed pruning approaches to build more compact network structures requiring less resources, while preserving performance. As we show in this paper, because of their intrinsic structure and function, Convolutional Network Fabrics are ideal candidates for pruning. We present a series of pruning strategies that can significantly reduce both the final network size and required training time by pruning either entire convolutional filters or individual weights, so that the grid remains visually understandable but that overall execution quality stays within controllable boundaries. Our approach can be iteratively applied during training so that the network complexity decreases rapidly, saving computational time. The paper addresses both data-dependent and data-independent strategies, and also experimentally establishes the most efficient approaches when training or testing data contain annotation errors.

Keywords: Pruning, Convolutional Networks, Label Noise, Classification

1 Introduction

Deep learning has been successful in solving very hard problems, but in the same time, the complexity of underlying neural networks has not ceased to increase. The hardware architectures required for achieving state-of-the-art performance are not necessarily available to all, or in constrained real-world situations like on hand-held devices. This is why research on pruning has gained momentum. Network pruning consists of determining parts of neural networks that do not significantly contribute to their final performance, and removing them. It has been first investigated in [1–3]. Pruning was mostly applied on pre-trained networks, followed by fine-tuning. More recently, other approaches have emerged, like iteratively alternating between training and pruning [4], or even pruning networks immediately after they have been initialized, so that one only needs to train a sparse architecture [5–7].

Furthermore, since designing the most appropriate neural net architecture for a given problem requires time, knowledge and experience for selecting and setting many hyperparameters, Convolutional Neural Fabrics (CNF) have been introduced [8] to facilitate the design of convolutional networks. CNFs embed an exponentially large number of chain-structured convolutional networks in one grid-like architecture, where a linear path from the input to the output of the grid corresponds to one CNN over many possible architecture choices. A CNF is defined by only two hyperparameters, and can be trained with the traditional backpropagation technique. Therefore, it facilitates the model selection process, since training the CNF is roughly similar to training multiple CNNs, while relying on a smaller number of hyperparameters. Although the CNF architecture is still huge in terms of the number of parameters, it still requires less resources compared to training each embedded CNN alone.

To the best of our knowledge, pruning has not seriously been investigated for CNFs. [8] shows how a pre-trained CNF can be pruned and fine-tuned with minor loss of accuracy, but this implies training a huge architecture during many epochs, which is time-consuming and requires big computational power. Our first contribution is to successfully apply an iterative pruning algorithm to a CNF, rapidly reducing the number of parameters to train. Furthermore, pruning is rather done on entire convolutional filters, so that the possibility to visualize the most influential paths in the network is not lost. This is an important asset of CNFs since it allows to select the most appropriate chained convolution architectures for a given problem.

A second contribution of this paper is to investigate the influence of data quality, and more specifically label or annotation noise on the pruning. Data is not always available with good quality annotations, and even with a good quality annotation process, inter-annotator agreement can vary, since the annotation problem is inherently ill-defined and partially subjective [9]. We study data-dependant *vs.* data-independent pruning criteria and how they influence the final quality of resulting networks.

Thus this work offers a way to build good convolutional models even for people:

- with a low level of knwoledge, as the CNF network structure needs only two hyperparameters to be defined, which is far simpler than traditional CNN;
- with low-quality resources: that is why we consider the data can be of low quality, i.e. noisy, and that the network must be pruned to ease its storage. Furthermore, all our experiments have been done using only one GPU, to ensure the feasability of the method with a low-quality architecture, and for that reason, we will not compare to other works done with multiple GPU.

The rest of the paper is organised as follows: the next section presents an overview of related work. Section 4 describes the pruning process and the choices made at various levels of the experimental design. The obtained results are presented and discussed in Section 5, and the last section concludes and presents some perspectives for improvement.

2 Background

Deep Neural Networks have largely grown in size to reach current state-of-the-art performance. With the improvement of hardware architectures, and large datasets available for many common tasks, it has become easier to build a network with good performance by using an over-parameterized model. This results in a network with high redundancy, trained and stored with unnecessary computation energy and memory consumption [10].

2.1 Network Pruning

To reduce the size of a network, one solution is to apply network pruning [1–3]. Network pruning corresponds to determining the weights of a network that contribute less to the output performance using various criteria, and remove them from the network. It is applied either on already trained networks [1, 3], and followed by fine-tuning, or in an iterative fashion [4], pruning some weights after enough training epochs and repeating the process, or else directly at the initialization of the network, which is thus trained for the first time after part of its weights have been pruned [11].

Recent advances have extended previous pruning strategies: [12] applies a pruning algorithm without loss in accuracy of the initial network; [5] shows the existence of a sub-network that, when trained in isolation, can perform as well as the whole trained network. Both use weight magnitude values for removing elements. On the other hand, [6, 7] focus on finding sub-networks with good performance in a randomly initialized untrained network.

There are mainly three criteria to decide which weights can be pruned. The actual number of weights to prune is usually a user-defined parameter.

- Magnitude-based or Zero-shot pruning: each weight is ranked based on its absolute value $|w|$. This criterion is data-independent, *i.e.* it is possible to compute without using any data.

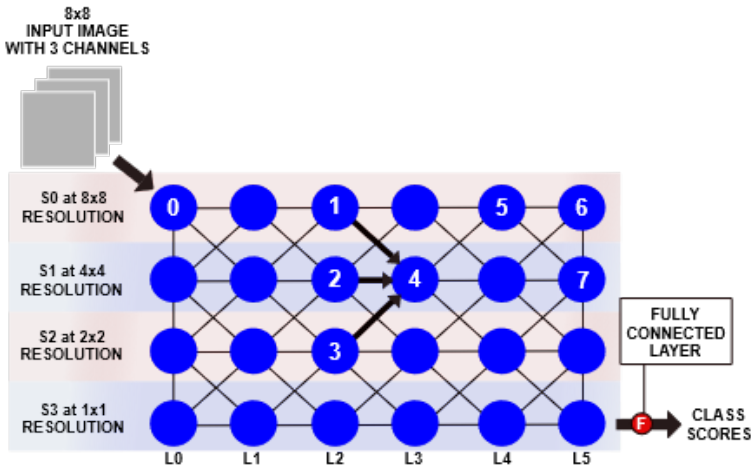

Fig. 1 A CNF architecture with 6 layers and 4 scales. Data passes through a first convolution to the input node in the top-left, which propagates it to the other scales by downsampling convolutions. The information flows through the grid along the links, toward the output node (bottom-right) and is passed to a fully connected layer.

- Sensitivity-based or Single-shot pruning [13]: each weight is ranked based on $|w\frac{\partial \mathcal{L}}{\partial w}|$, where $\mathcal{L}$ is the loss of the network. This criterion is data-dependent as we need some data samples to compute that loss.
- Hessian-based pruning [14]: the weights are selected using the hessian of the loss function. This criterion is also data-dependent.

2.2 Convolutional Neural Fabric

Designing an architecture by hand adapted to solving a given problem can be difficult given the high number of hyperparameters, and can lead to a time-consuming "test and retry" process. In [8], the authors focus on CNNs and propose a multidimensional model for image-related tasks. The model embeds an exponential number of chain-structured CNNs, which can be trained together at once by locally sharing weights. This simplifies the design and selection process of the best CNN architecture, but implies an increase in the size of the model to be trained. Thus, by applying network pruning techniques, the training time and memory storage of the model can be significantly reduced, allowing to be stored and run with modest hardware architectures.

A CNN architecture is determined by many hyper-parameters: number of convolutional layers ($ConvL$), their number of channels, filter size and stride for each $ConvL$, number of pooling layers ($PoolL$), their operator type and region size as well as the ordering between $ConvL$s and $PoolL$s, channel connectivity pattern between layers, and activation function types per layer. Exhaustive exploration of all possible architectures is not feasible in practice, and finding a good CNN (*e.g.* using a local-search strategy) remains a

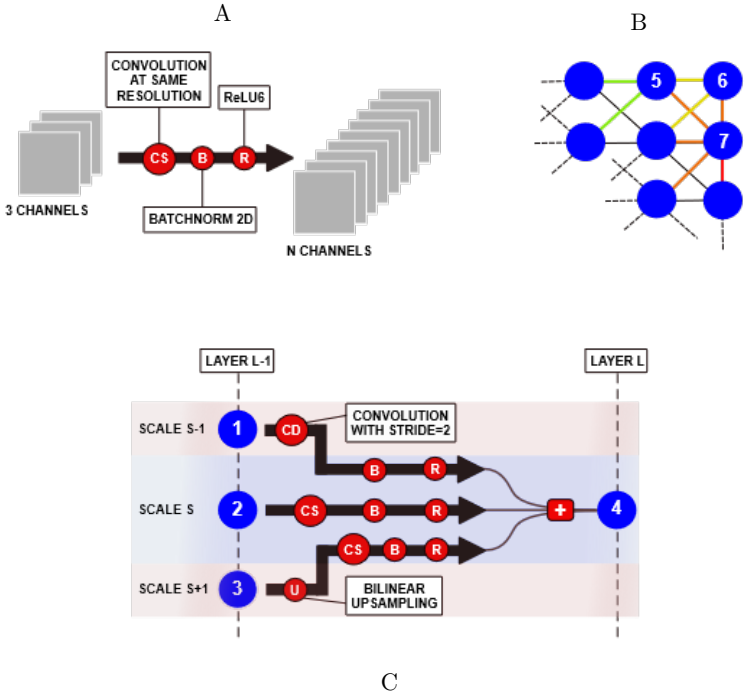

Fig. 2 Fig. A and C show how the activation tensor of node 0 is obtained from the input image, and how the activation tensor of node 4 is obtained from nodes 1, 2 and 3. Going from scale $S-1$ to S is done by setting the convolution stride to 2, while the opposite is done through bi-linear upsampling. Fig. B shows how parts of a CNF become obsolete (green, yellow and orange links) when particular links are pruned (red).

tough, time-consuming problem when one has to train and test individual architectures.

As described in [8], the set of architectures corresponding to all hyper-parameter choices can be embedded in a single three-dimensional grid-like structure, called a Convolutional Neural Fabric, and trained in one run. A CNF processes the information along three axes: a layer axis of size $\mathcal{L}$, a scale or resolution axis of size $\mathcal{S}$, and a channel axis of size $\mathcal{C}$. Since we do not change the number of channels inside a CNF, and we use a standard connectivity pattern between channels, the behaviour of our CNFs does not change across the channel axis. Therefore we will consider only a 2D version of a CNF, with a horizontal layer axis and a vertical scale axis (Fig. 1). The number of scales is set in function of the size of the input image such that the resolution is divided by 2 when going one scale down, and is equal to 1×1 at the smallest scale $\mathcal{S}-1$.

The input image is first transformed to the right number of channels by a starting convolution operation (Fig. 2.A). The result becomes the activation tensor at the input node of the grid, at position $(layer = 0, scale = 0)$ in the top-left corner. It is processed until reaching the output node at $(\mathcal{L}-1, \mathcal{S}-1)$. In the 2D grid, each node represents an activation tensor of size $(\mathcal{C}, W, H)$,

where (W, H) is the local resolution of the processed data. The resolution at scale 0 is the initial resolution of the input image, and decreases along the scale axis. Nodes are connected by links through a local connectivity pattern: node at position (l, s) is linked to the three closest nodes in the previous layer, *i.e.* at positions $(l-1, s \pm 1)$, and the three closest nodes in the next layer, at positions $(l+1, s \pm 1)$ wherever possible. Nodes at layers 0 and $\mathcal{L}-1$ must also spread the information along the scale axes, so *e.g.* node (l, s) is connected to node $(l, s+1)$ for $l \in \{0, \mathcal{L}\}$ when possible. Each link applies specific operations to the activation tensor of the node at its start, and passes the resulting tensor to the node at its end. Each node aggregates the tensors it receives to form its activation tensor (Fig. 2.C). For example, the activation tensor T of a node (l, s) in the middle of the grid is computed by $T_{l,s} = \sum_{i \in \{-1,0,1\}} T_{l-1,s+i}$.

The typical operations applied by a link are:

- a convolution, with a 3×3 kernel size, followed by an upsampling or a downsampling according to the direction of the link (up or down), or nothing if the link processes the information at the same scale;
- a batch normalization;
- the activation function (in our case ReLU6).

In their paper [8], the authors argue that the only determinant hyperparameters of a CNF are the number of channels and layers, and they show that those two hyperparameters become less critical if their values are large enough.

3 Pruning Algorithm

In what follows, we denote $\chi(e)$ the pruning criterion applied to a potentially prunable element e of the network. It returns a score expressing how much e contributes to the network performance. We are not interested in pruning at initialization, and focus on zero-shot and single-shot pruning. We divide our process in two steps.

First, we prune entire links from the CNF, removing them from the flow of information in the fabric. For example, in Fig. 1, if the link between nodes 3 and 4 is pruned, node 4 only receives two tensors to sum instead of three. This way we first determine an optimal substructure in the fabric identifying the most efficient processing paths for the problem at hand. Second, we apply pruning to the weights in the convolution filters of each remaining link in the fabric. Furthermore, and for obvious reasons, the first link transforming the input image into the right number of channels (Fig. 2), as well as the fully connected layer at the end of the network are never pruned.

3.1 Link Pruning

During the first step of pruning, removing one link can make other computations in the fabric unnecessary. For example, in Fig. 2.B, the removal of the red link prevents node 7 to transmit its activation to the following nodes since its sole output link disappears. This means all computations for the activation

tensor of node 7 become unnecessary. Therefore, removing the red link implies removing the input links of node 7 (in orange) as well. Since the output links of node 6 are part of these orange links, the same situation is repeated: the yellow links must also be removed, followed by the green ones, as node 5 becomes obsolete, too.

We also need to avoid that the removal of a link entirely cuts the information flow between the input and the output of the fabric. To avoid this situation, we ensure that the set of potentially pruned links $\mathcal{P}_L$ satisfies the condition that if all links in $\mathcal{P}_L$ are removed from the fabric, there still exists a path between the input and output nodes. We achieve this by first ranking the links based on their respective criterion value, then adding links one by one to the set, constantly checking above mentioned condition. Links violating the condition are skipped. We stop when enough links have been found, or when all links have been seen.

The question that remains is how a pruning criterion can be applied to a link. Indeed, they are initially defined to apply on the weights (cf. section 2.1), but links are complex objects. However, as each link contains a weight matrix, the pruning criteria can be adapted in a straightforward way. For a link l and the weight matrix of its convolution filter W^l_{conv}, the criterion $\chi(l)$ is the euclidean norm of the matrix obtained by applying χ element by element to W^l_{conv}: $\chi(l) = \left\|\chi(W^l_{conv})\right\|$.

3.2 Weight Pruning

The second step is similar to the first, but pruning now applies to the weights in the convolution filters of the remaining links. In practice, to remove a weight w from the weight matrix W, we apply a binary mask $\mathcal{M}_W$ on W with a zero at the same position as w, as to cancel any operation involving that weight. This is a convenience solution and will have an impact on the subsequent measurements we can do later on (cf. section 5.2).

Like for Link Pruning, we apply a condition to the weights to determine if we can safely remove them: a weight w cannot be removed if its mask $\mathcal{M}_W$ would result in containing only zeros. We do not want to zero out an entire convolution filter, since the equivalent is already done in the first step.

4 Experimental Setup

4.1 Data

We consider 4 different datasets for our experiments:

- The CIFAR10 dataset [15]: 60,000 32×32 RGB images, labeled in 10 classes, each containing 6.000 images. There are 5,000 training images and 1,000 test images per class.
- The CIFAR100 dataset [15]: similar to the previous one, but having 100 classes, with each 500 training images and 100 test images.

Algorithm 1 Pruning algorithm

input : $\mathcal{E}$: a set of fabric elements to potentially prune; either the links of the fabric, or the weights of the convolution filters;
n: the number of elements to prune;
χ: the criterion used to rank the elements by order of importance;
$\mathcal{C}$: condition applied on an element to ensure it can be pruned safely.

output: A set $\mathcal{P}_E$ of elements to prune

An empty list L to store the ranked elements;
for e *in* $\mathcal{E}$ **do**
  Compute the score $\chi(e)$ of e;
  Insert e in L so that L is ranked according to $\chi(e)$;
end
Set $\mathcal{P}_E$ to empty;
for e *in* L **do**
  if $\mathcal{C}(e)$ **then**
    Add e to $\mathcal{P}_E$;
  end
end

- The SVHN dataset [16]: 600,000 32×32 RGB images of digits. We only use the 73,257 images of the training set and the 26,032 of the testing, excluding the 531,131 extra training images.
- The VOC2012 classification dataset of the PASCAL VOC challenge [17]: 11,530 RGB images of 500 pixels in width and various heights, with a total of 27,450 ROI annotated objects and 6,929 segmentations. There is a total of 20 classes. Each image can contain multiple objects, and the annotation includes information about each object (class, bounding box, *etc.*). For that reason, the classification challenge is a multilabel problem (the goal is to find all the objects present in each image). As we only focus on one-label classification, we first apply a preprocessing step to collect only the images that are suited for that task. This way, we collect 9,340 samples out of 11,530. Preprocessing for each item d in the dataset is as follows:
  - if d contains only one object or many objects of the same type, keep d with the corresponding object label;
  - otherwise compute the area occupied by each different object using their bounding boxes found in the annotation of d; if the object with the largest surface is twice as big as the object with the second largest surface, keep d with the label of the bigger one, else discard d.

4.2 Protocol

In order to show how our CNF pruning algorithm behaves, we compare the performance of an unpruned baseline CNF with differently pruned CNFs trained on the same data. This gives us 4 baseline CNFs (one for each dataset) and 18 differently pruned CNFs for each baseline. Experiments are run on both the

standard datasets (*cf.* Section 5) and on the same datasets in which we artificially introduce labelling errors (*cf.* Section 6), so that we can assess how the mislabeling may influence the pruning process. We implemented our experiments in PyTorch, and we trained the models on the Grid5000 platform [18], using one node (physical machine) with one GPU per training.

4.2.1 Training Data

systematically consists in only 90% of the provided training sets (for those datasets providing training and test sets). The remaining 10% is kept for validation. The test set is used only for gradient computation when one-shot pruning is applied (*cf.* 4.2.3). We always make sure to split the data such that CNFs are consistently trained on data sets of the same size, independently of the pruning criterion. As the PASCAL VOC challenge does not provide any test set, we apply the following particular split for its data: 70% of the data is used for training, 20% for testing, and the remaining 10% for validation. When splitting the datasets, we keep the same proportion of each class in each split. The data augmentation scheme we apply is generally composed of normalizing, resizing, randomly cropping and flipping horizontally the images (Fig. 6 in appendix).

4.2.2 CNF Architecture

consistently has 8 layers and 64 channels for all but one dataset: the SVHN dataset. Our experiments have shown that 64 channels is too much for that particular problem. We reduced the number of channels to 32 when training on SVHN. We use a cross entropy loss for training, and we optimize the networks with stochastic gradient descent during 200 epochs, using an initial learning rate of 0.1, and dividing it by 10 once after epoch 80, and a second time after epoch 120.

4.2.3 Pruning

1. We set three desired sparsity values of the final network: 0.05, 0.03, 0.01; a sparsity of 0.05 means that we remove 95% of the elements in the CNF (95% of links and 95% of weights); we also always ensure that the number of remaining links is greater than the number of links contained in the longest linear path in the CNF, no matter the sparsity specified: this is for letting a chance to at least each linear path to be kept during link pruning;
2. We apply three different strategies;
   - early pruning: after 5 epochs of training, we apply link pruning, followed by weight pruning to reach the desired sparsity;
   - late pruning: after 75 epochs of training, we apply link pruning, followed by weight pruning to reach the desired sparsity;
   - iterative pruning: after 5 epochs of training, we apply link pruning followed by weight pruning, and this every 10 epochs until epoch 75; the

CIFAR10

Model	Param.	Test Err. (%)
CNF-B	>4M	6.44
CNF-P	0.14M	8.03
Maxout [19]	>5M	9.38
DSN [20]	0.97M	7.97
RCNN-96 [21]	0.67M	7.37
RCNN-160 [21]	1.86M	7.09

CIFAR100

Model	Param.	Test Err. (%)
CNF-B	>4M	27.54
CNF-P	0.24M	33.26
Maxout [19]	>5M	38.57
DSN [20]	0.97M	34.57
RCNN-96 [21]	0.67M	34.18
RCNN-160 [21]	1.86M	31.75

SVHN

Model	Param.	Test Err. (%)
CNF-B	0.29M	3.56
CNF-P	15K	4.43
Maxout [19]	>5M	2.47
DSN [20]	0.97M	1.92
RCNN-160 [21]	0.67M	1.80
RCNN-192 [21]	1.86M	1.77

Fig. 3 Results on CIFAR10, CIFAR100 and SVHN datasets for various reference methods. CNF-B is our CNF baseline, CNF-P is our best obtained pruned CNF.

number of links and weights pruned each time is calculated so that at the end, the right number has been removed according to the sparsity value specified;

3. For each strategy, we both test magnitude based (zero-shot) and sensitivity based (single-shot) pruning. We keep the hessian-based criterion for future work.

5 Results on Clean Data

Due to space limitation, we could not compile the results of all the experiments. An exhaustive listing of the results is provided in the appendix.

5.1 State-of-the-Art

Fig. 3 gives an overview of how our CNFs compare to similar state-of-the-art approaches. The pruned CNFs are at par with their competitors where classification error rate is concerned, with significantly smaller networks.

Fig. 3 does not report results for the PASCALVOC dataset. Since we diverted its data for one-label classification, while it was initially conceived for object detection, it makes our results unfit for making a significant comparison with those reported elsewhere. We would need to compute the Mean Average Precision, based on bounding boxes, for which our CNFs are currently unsuited. However, we believe that it can still be useful to show how

	MAGN	SBD
Clean Setting		
CIFAR10	100%	0%
CIFAR100	83%	17%
SVHN	33%	67%
PASCALVOC	33%	67%
OVERALL	62.5%	37.5%

	MAGN	SBD
Noise Setting		
CIFAR10	33%	67%
CIFAR100	50%	50%
SVHN	83%	17%
PASCALVOC	42%	58%
OVERALL	52%	48%

Fig. 4 Percentage of the time where the models with the corresponding choices gave the best results.

our pruned models perform with respect to the baseline CNF on different data distributions.

5.2 Pruning settings

Overall, the best models were obtained when applying late pruning, since more training epochs before pruning makes it easier to detect unimportant elements in the network. However, the iterative pruning strategy often gives comparable results to late pruning. We also observe that the magnitude-based (MAGN) criterion gives slightly better results compared to the sensitivity-based (SBD) criterion, as shown in Fig. 4. The percentage values are taken from the experiments on all datasets, for all possible settings. For example, in the clean data setting, 18 networks have been trained on CIFAR100: 9 were pruned using MAGN criterion, and 9 other using SBD criterion. Thus there are 9 couples of classifier for which the only difference is the criterion used (all other parameters are the same). Table 4 says that for 83% of those couples, the classifier pruned with MAGN was better.

For the training time, the earlier we remove elements of the CNF, the better. Therefore, applying iterative pruning with a MAGN criterion seems to be a good trade-off between loss in accuracy and faster decrease in model complexity, optimizing training time and memory consumption. This is particularly noticeable for pruning sizes of 95% and 97%. When pruning 99% of our models, we saw a clear drop in performance, and the results were too chaotic to base further conclusions on.

In PyTorch, the usual implementation of network pruning does not allow to see the gain in training time, as it is simply done by multiplying the pruned weights with zero. We however have access to a partial gain in training time, thanks to our step of link pruning. Indeed, when we prune the links, we modify directly the architecture of a CNF, thus effectively removing all operations applied by links. This allows to at least show significant improvement in the average training time: 4 hours for early pruning, 6 hours for iterative pruning and 8 hours for late pruning, when the baselines took more than 8 and up to 14 hours. With an implementation of weight pruning that effectively makes use of only the remaining weights, the differences in training time would be far more striking.

Table 1 Percentage of the time where the models with the corresponding choices gave the best results. *E.g* for early pruning, link pruning (LP+WP) always outperforms weight pruning (WP): 56% of the time for MAGN, 44% of the time for SBD.

	MAGN		**SBD**	
	LP+WP	WP	LP+WP	WP
EARLY	0.56	0.0	0.44	0.0
ITERATIVE	0.22	0.33	0.33	0.12
LATE	0.45	0.45	0.10	0.0

5.3 Link pruning

Link pruning allows to have insight in the best performing architectures for each dataset (Fig. 5). We can see for instance that the best for the CIFAR10 and CIFAR100 data seems to be processing it at different scales in parallel in early layers. For CIFAR100, in particular, the best results are obtained when the multiscale processing phase lasts for more layers.

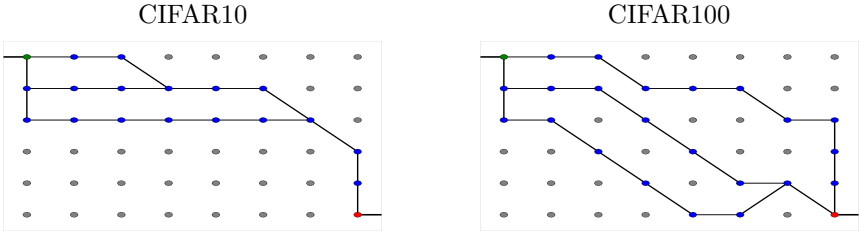

Fig. 5 Architecture examples of well-performing pruned models: MAGN-ITE-95 for CIFAR10 and CIFAR100

Moreover, we also tried to prune the CNFs without the link pruning step, directly removing weights in the convolution filters. When we compare the models obtained with link pruning with the models obtained without link pruning (table 1), we remark that:

- when early pruning is applied, adding the link pruning step gives models that clearly outperform those obtained without it;
- when iterative or late pruning is applied, the models obtained with or without link pruning have overall similar performance.

This indicates that in order to train small models without much resources (computational power, memory, etc), *i.e.* when pruning early on during training is required, applying a link pruning step in the pruning process is clearly beneficial for the performance of the trained models.

6 Results on Noisy Data

6.1 Label Noise

The last part of this paper consists of evaluating how data annotation quality impacts the outcome of the pruning process. Label noise [22] is known to be harmful to the optimization of supervised classifiers in general [23, 24] as it biases the training process. But in its broader sense (*i.e.* an undesired label given to an entity in the data set), is not necessarily limited to training data. It can also be present in validation sets, and even test sets. This has become apparent with the recent works concerning annotation bias [25]. Therefore, any process making use of data labels is influenced by annotation bias [9]. This could concern network pruning strategies as well, but to the best of our knowledge, no work has been reported on this subject. This work is part of a broader study on the impact of label noise on classifier bias [26]

For our experiments, we will artificially introduce annotation errors in our datasets, so that we can try and compare data-dependent and independent pruning criteria. Furthermore, we want the introduced errors be as close as possible to real world situations, such that the trained models are able to generalize the label noise structure to unseen examples, very much as if the data were mislabelled on subjective interpretations of an annotator, and that this biased interpretation was consequently learned by the machine.

Our goal is to measure if the pruning process exacerbates the bias, or rather remains insensitive to these kinds of noise.

6.2 Generating Annotation Errors

Let x be an entity in a dataset D belonging to class y_x with given label $\tilde{y}_x$. The label predicted for x by a given classifier is $\hat{y}_x$. We assume that our datasets are void of mislabelled entities: $\forall D, \forall x \in D, \tilde{y}_x = y_x$.

In [22], annotation errors are defined as the result of a stochastic process E that disturbs the label of an entity x with probability p_{x,y_x}. We note $\tilde{D}$ the result of E applied to D. E can take three different forms:

- uniformly random (Type 1): the probability of an entity being mislabelled does not depend on its class nor on its features ($p_{x,y_x} = p$);
- class dependent (Type 2): entities of two different classes can have different probabilities of being mislabelled ($p_{x,y_x} = p_{y_x}$);
- class and feature dependent (Type 3): even for two entities in the same class, the probability that they are mislabelled can vary depending on where they are located in the feature space.

Type 1 corresponds to swapping the label of each entity with a fixed probability, and by randomly choosing the noisy label among the remaining ones. For Type 2, one defines a transition matrix between the classes of the problem, such that cell (i, j) represents the probability of labelling an entity of

class i with j. When this matrix is symmetric, the process is called symmetry flipping, otherwise it is called pair flipping.

As pointed out before, our goal is to find for each dataset D, a noisy version $\tilde{D}$ such that the models trained on $\tilde{D}$ are effectively biased by the mislabelled entities. To that end, we first obtain a candidate $\tilde{D} = E(D)$, we train a baseline CNF on it and we look at two indicators:

- *clean fitting*: among the *clean* data samples, the fraction of correctly predicted entities, *i.e.* for which we predicted the *clean* label: $\frac{|\hat{y}_x = y_x = \tilde{y}_x|}{|y_x = \tilde{y}_x|}$
- *noisy fitting*: among the *noisy* data samples, the fraction of entities for which we predicted the *noisy* label: $\frac{|\hat{y}_x = \tilde{y}_x|}{|y_x \neq \tilde{y}_x|}$

Those indicators are measured for each classifier trained on the noisy data sets, using the test samples (which can also be mislabelled). The clean fitting represents how much a network managed to learn the underlying true task (specified by the remaining correctly labeled entities) without being too much impacted by the mislabelled entities, and the noisy fitting shows how much the network did overfit the label noise structure, and is thus biased by label noise. We consider that a noisy version of a dataset D is acceptable if the networks trained on it obtain high enough values for both indicators:

- if the clean fitting is too low, it means the label noise introduced has destroyed too much of the data structure for the classifiers to learn anything, and we do not want to study that setting;
- if the noisy fitting is too low, it means the classifiers do not overfit the noisy samples, so the label noise introduced does not have a coherent and learnable structure, while we do believe that real-world label noise is learnable.

.

We empirically determined that uniformly random (Type 1) or class dependant (Type 2) noise types are not satisfactory. Indeed, the noisy fitting indicator remains very low for all networks trained on datasets with such noise types. For our experiment, we therefore decided to only use class and feature dependant noise (Type 3). We generate Type 3 noise for a dataset $\tilde{D}$ by training an *artificial annotator* $\mathcal{A}$ on the data we want to perturbate. Doing so, we guarantee that the noisy labels effectively depend on the data features. To obtain a given fraction ϵ of mislabelled samples in $\tilde{D}$, we stop the training of $\mathcal{A}$ when its test error equals ϵ. In what follows we have generated noisy datasets for CIFAR10, CIFAR100, SVHN and PASCALVOC with $\epsilon = 10\%$ and $\epsilon = 20\%$.

Networks trained on those noisy datasets obtained higher values for the noisy fitting indicator, confirming the utility of Type 3 noise. Table 2 shows the overall values of the clean and noisy fittings for various networks trained on the four noisy datasets.

Table 2 Clean and noisy fitting value ranges for models (pruned or not) trained on the noisy versions of each dataset.

	CIFAR10	CIFAR100	SVHN	PASCALVOC
Clean Fitting	$> 80\%$	$> 60\%$	$\approx 97\%$	$\approx 50\%$
Noisy Fitting	$\approx 35\%$	15%-25%	40%-50%	10%-30%

6.3 Results

We have repeated the previous experiment conducted on clean data, using this time the noisy datasets. For each data set, we trained 1 baseline not pruned, and 18 networks pruned according to the same parameters than before. All networks are biased by label noise in various ways, as the noisy fitting indicator shows in table 2. The higher the value for one network, the more it did overfit the noisy samples.

First of all, we observed that the pruned networks were in general more biased by label noise than their corresponding baseline, while we were expecting a smaller model to be less prone to overfit the noise. Especially, the models obtained with the MAGN criterion seem to be slightly more biased in general.

Surprisingly, the results were overall better than expected for the SBD criterion, which is data-dependent, and worse than expected for the MAGN criterion, which is data-independent. We were expecting the gap in efficiency between the two criteria to expand when moving from the clean setting to the noisy setting, since we thought that the SBD criterion would prove even worse in presence of label noise, but overall we observed the opposite trend (Fig. 4). This may be due to the fact that the more a model overfits the structure of label noise in a dataset, the more it shows through its weights, and therefore through the MAGN criterion as well. The exception is for the SVHN dataset, where the SBD criterion become clearly worse in the noisy setting. However, the dimensions we chose for the models trained on SVHN are far smaller than for the models trained on the other datasets, making them potentially less prone to overfit the label noise structure. This could explain the greater efficiency of the MAGN criterion on SVHN models.

7 Conclusion

We showed how to adapt a pruning algorithm to a multidimensional neural network (CNF), obtained results comparable to the state-of-the-art with model sizes 5 to 50 times smaller, and tested the pruning algorithm for many different hyperparameter values. In particular, our pruned networks gave similar performances to those presented in initial CNF paper [8], even when we applied an iterative or early-pruning strategy, which allowed to greatly decrease the training time. We showed how including a link pruning step in the pruning process was beneficial to the pruned model, compared to classically applying pruning on the weights directly. We also refuted the initial idea that the presence of label noise in a dataset would make a data-independent criterion better

than a data-dependent one, by experimenting on datasets containing 10% and 20% of mislabelled entities.

References

[1] LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems, pp. 598–605 (1990)

[2] Hanson, S.J., Pratt, L.Y.: Comparing biases for minimal network construction with back-propagation. In: Advances in Neural Information Processing Systems, pp. 177–185 (1989)

[3] Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: Optimal brain surgeon. In: Advances in Neural Information Processing Systems, pp. 164–171 (1993)

[4] Carreira-Perpinan, M., Idelbayev, Y.: "learning-compression" algorithms for neural net pruning, pp. 8532–8541 (2018). https://doi.org/10.1109/CVPR.2018.00890

[5] Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. arXiv preprint arXiv:1803.03635 (2018)

[6] Zhou, H., Lan, J., Liu, R., Yosinski, J.: Deconstructing lottery tickets: Zeros, signs, and the supermask. arXiv preprint arXiv:1905.01067 (2019)

[7] Ramanujan, V., Wortsman, M., Kembhavi, A., Farhadi, A., Rastegari, M.: What's hidden in a randomly weighted neural network? arXiv preprint arXiv:1911.13299 (2019)

[8] Saxena, S., Verbeek, J.: Convolutional neural fabrics. In: Advances in Neural Information Processing Systems, pp. 4053–4061 (2016)

[9] Lamiroy, B.: Interpretation, Evaluation and the Semantic Gap ... What if we Were on a Side-Track? In: Lamiroy, B., Ogier, J.-M. (eds.) 10th IAPR International Workshop on Graphics Recognition, GREC 2013. LNCS, vol. 8746, pp. 213–226. Springer, Bethlehem, PA, United States (2013). https://hal.inria.fr/hal-01057362

[10] Denil, M., Shakibi, B., Dinh, L., Ranzato, M., De Freitas, N.: Predicting parameters in deep learning. In: Advances in Neural Information Processing Systems, pp. 2148–2156 (2013)

[11] Hayou, S., Ton, J.-F., Doucet, A., Teh, Y.W.: Pruning untrained neural networks: Principles and Analysis (2020)

[12] Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Advances in Neural Information

Processing Systems, pp. 1135–1143 (2015)

[13] Lee, N., Ajanthan, T., Torr, P.H.: Snip: Single-shot network pruning based on connection sensitivity. arXiv preprint arXiv:1810.02340 (2018)

[14] Wang, C., Zhang, G., Grosse, R.: Picking winning tickets before training by preserving gradient flow. arXiv preprint arXiv:2002.07376 (2020)

[15] Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)

[16] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning (2011)

[17] Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html

[18] Balouek, D., Carpen Amarie, A., Charrier, G., Desprez, F., Jeannot, E., Jeanvoine, E., Lèbre, A., Margery, D., Niclausse, N., Nussbaum, L., Richard, O., Pérez, C., Quesnel, F., Rohr, C., Sarzyniec, L.: Adding virtualization capabilities to the Grid'5000 testbed. In: Ivanov, I.I., van Sinderen, M., Leymann, F., Shan, T. (eds.) Cloud Computing and Services Science. Communications in Computer and Information Science, vol. 367, pp. 3–20. Springer, ??? (2013). https://doi.org/10.1007/978-3-319-04519-1_1

[19] Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. arXiv preprint arXiv:1302.4389 (2013)

[20] Lee, C.-Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-supervised nets. In: Artificial Intelligence and Statistics, pp. 562–570 (2015)

[21] Liang, M., Hu, X.: Recurrent convolutional neural network for object recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3367–3375 (2015)

[22] Frénay, B., Verleysen, M.: Classification in the presence of label noise: a survey. IEEE transactions on neural networks and learning systems **25**(5), 845–869 (2013)

[23] Zhang, J., Yang, Y.: Robustness of regularized linear classification methods in text categorization. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval, pp. 190–197 (2003)

[24] Nettleton, D.F., Orriols-Puig, A., Fornells, A.: A study of the effect of different types of noise on the precision of supervised learning techniques. Artificial intelligence review **33**(4), 275–306 (2010)

[25] Geva, M., Goldberg, Y., Berant, J.: Are We Modeling the Task or the Annotator? An Investigation of Annotator Bias in Natural Language Understanding Datasets (2019)

[26] Benjelloun, I.: Impact du bruit d'annotation sur l'Évaluation de classifieurs. PhD thesis, Université de Lorraine (2021). Thèse de doctorat dirigée par Lamiroy, Bart et Koudou, Angelo Efoevi Informatique Université de Lorraine 2021. http://www.theses.fr/2021LORR0267

A Appendices

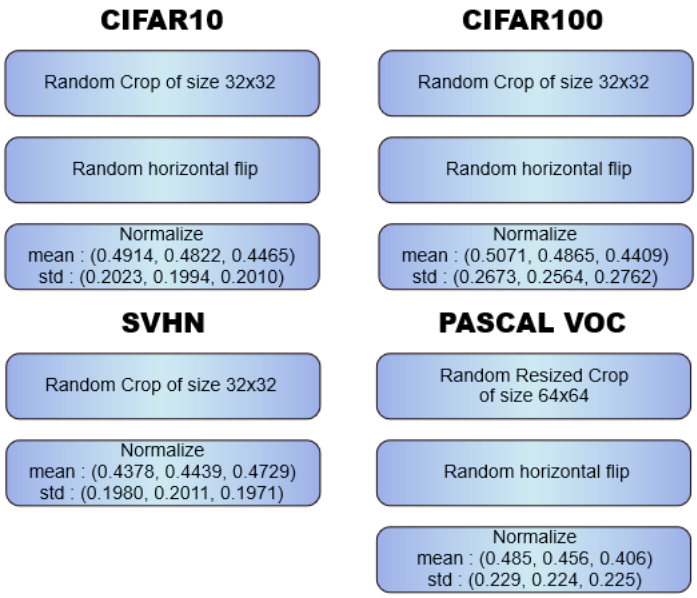

Fig. 6 Transformations applied to the 4 training sets for data augmentation.

Following are the results obtained on the 4 datasets for baseline CNFs (not pruned), and CNFs pruned in various ways, on clean and noisy data. The names of the models represent the different pruning settings: MAGN or SBD for magnitude-based or sensitivity-based pruning; EAR, ITE or LAT for early, iterative or late pruning; and the number represent the fraction of the CNF that has been pruned. For the noisy data setting, the results are still measured on the clean version of the testsets. Also, for all datasets but pascalvoc, and

only for the clean data setting, we tried to apply pruning by removing the link pruning step, directly pruning the weights of the models.

CIFAR10-CLEAN

Model	Nb. Param.	Test Err. with | without LP (%)
BASELINE	4523402	6.44
MAGN-EAR-95	228611	09.55 | 10.93
SBD-EAR-95	-	09.94 | 10.53
MAGN-ITE-95	-	08.03 | 07.35
SBD-ITE-95	-	08.99 | 07.67
MAGN-LAT-95	-	08.56 | 07.32
SBD-LAT-95	-	08.69 | 08.57
MAGN-EAR-97	138194	10.49 | 12.78
SBD-EAR-97	-	11.75 | 11.31
MAGN-ITE-97	-	09.97 | 08.79
SBD-ITE-97	-	10.31 | 06.39
MAGN-LAT-97	-	08.39 | 08.25
SBD-LAT-97	-	09.31 | 10.40
MAGN-EAR-99	47778	15.33 | 26.56
SBD-EAR-99	-	14.32 | 20.33
MAGN-ITE-99	-	13.40 | 17.11
SBD-ITE-99	-	16.20 | 19.84
MAGN-LAT-99	-	12.02 | 17.83
SBD-LAT-99	-	14.48 | 21.35

CIFAR100-CLEAN

Model	Nb. Param.	Test Err. with/without LP (%)
BASELINE	4529252	27.54
MAGN-EAR-95	234461	36.41 | 38.82
SBD-EAR-95	-	37.08 | 37.27
MAGN-ITE-95	-	34.72 | 31.62
SBD-ITE-95	-	35.21 | 31.77
MAGN-LAT-95	-	33.26 | 29.95
SBD-LAT-95	-	33.73 | 33.07
MAGN-EAR-97	144044	40.98 | 42.52
SBD-EAR-97	-	39.03 | 41.56
MAGN-ITE-97	-	34.76 | 33.78
SBD-ITE-97	-	37.75 | 35.77
MAGN-LAT-97	-	34.81 | 34.09
SBD-LAT-97	-	36.39 | 37.00
MAGN-EAR-99	53628	48.71 | 57.56
SBD-EAR-99	-	47.96 | 54.64
MAGN-ITE-99	-	42.99 | 50.66
SBD-ITE-99	-	46.55 | 54.13
MAGN-LAT-99	-	42.03 | 49.09
SBD-LAT-99	-	45.92 | 53.01

SVHN-CLEAN

Model	**Nb. Param.**	**Test Err. with/without LP (%)**
BASELINE	287594	03.56
MAGN-EAR-95	14997	05.01/07.90
SBD-EAR-95	-	05.20/06.40
MAGN-ITE-95	-	04.77/04.77
SBD-ITE-95	-	04.43/05.33
MAGN-LAT-95	-	05.00/04.82
SBD-LAT-95	-	04.47/06.03
MAGN-EAR-97	9258	06.71/18.94
SBD-EAR-97	-	05.74/14.25
MAGN-ITE-97	-	06.49/12.73
SBD-ITE-97	-	06.46/13.07
MAGN-LAT-97	-	04.90/11.46
SBD-LAT-97	-	05.90/20.14
MAGN-EAR-99	3519	14.45/80.41
SBD-EAR-99	-	17.77/80.41
MAGN-ITE-99	-	80.41/80.41
SBD-ITE-99	-	14.54/80.41
MAGN-LAT-99	-	17.94/80.41
SBD-LAT-99	-	19.88/80.41

PASCALVOC-CLEAN

Model	**Nb. Param.**	**Test Err. (%)**
BASELINE	5376340	**48.78**
MAGN-EAR-95	271876	53.44
SBD-EAR-95	-	52.33
MAGN-ITE-95	-	**51.22**
SBD-ITE-95	-	51.69
MAGN-LAT-95	-	51.69
SBD-LAT-95	-	51.64
MAGN-EAR-97	164413	52.70
SBD-EAR-97	-	56.41
MAGN-ITE-97	-	55.46
SBD-ITE-97	-	50.85
MAGN-LAT-97	-	53.07
SBD-LAT-97	-	52.70
MAGN-EAR-99	56951	59.27
SBD-EAR-99	-	84.16
MAGN-ITE-99	-	53.92
SBD-ITE-99	-	54.71
MAGN-LAT-99	-	54.24
SBD-LAT-99	-	55.14

CIFAR10-NOISY-10%

Model	**Nb. Param.**	**Test Err. (%)**
BASELINE	4523402	13.1
MAGN-EAR-95	228611	14.37
SBD-EAR-95	-	13.81
MAGN-ITE-95	-	13.49
SBD-ITE-95	-	14.24
MAGN-LAT-95	-	13.58
SBD-LAT-95	-	13.77
MAGN-EAR-97	138194	14.53
SBD-EAR-97	-	14.59
MAGN-ITE-97	-	14.05
SBD-ITE-97	-	13.66
MAGN-LAT-97	-	13.45
SBD-LAT-97	-	13.75
MAGN-EAR-99	47778	17.35
SBD-EAR-99	-	16.99
MAGN-ITE-99	-	16.02
SBD-ITE-99	-	17.96
MAGN-LAT-99	-	15.42
SBD-LAT-99	-	17

CIFAR100-NOISY-10%

Model	**Nb. Param.**	**Test Err. (%)**
BASELINE	4529252	30.68
MAGN-EAR-95	234461	39.24
SBD-EAR-95	-	39
MAGN-ITE-95	-	36.20
SBD-ITE-95	-	35.96
MAGN-LAT-95	-	35.14
SBD-LAT-95	-	35.27
MAGN-EAR-97	144044	41.33
SBD-EAR-97	-	40.84
MAGN-ITE-97	-	37.79
SBD-ITE-97	-	37.56
MAGN-LAT-97	-	36.37
SBD-LAT-97	-	37.81
MAGN-EAR-99	53628	48.17
SBD-EAR-99	-	47.98
MAGN-ITE-99	-	41.66
SBD-ITE-99	-	44.72
MAGN-LAT-99	-	43.68
SBD-LAT-99	-	47.43

SVHN-NOISY-10%

Model	**Nb. Param.**	**Test Err. (%)**
BASELINE	287594	7.91
MAGN-EAR-95	14997	7.79
SBD-EAR-95	-	8.1
MAGN-ITE-95	-	7.81
SBD-ITE-95	-	7.65
MAGN-LAT-95	-	7.58
SBD-LAT-95	-	7.59
MAGN-EAR-97	9258	8.59
SBD-EAR-97	-	8.41
MAGN-ITE-97	-	8.50
SBD-ITE-97	-	9.34
MAGN-LAT-97	-	8.1
SBD-LAT-97	-	8.47
MAGN-EAR-99	3519	16.22
SBD-EAR-99	-	15.9
MAGN-ITE-99	-	52.38
SBD-ITE-99	-	80.41
MAGN-LAT-99	-	14.13
SBD-LAT-99	-	19.53

PASCALVOC-NOISY-10%

Model	**Nb. Param.**	**Test Err. (%)**
BASELINE	5376340	52.07
MAGN-EAR-95	271876	52.86
SBD-EAR-95	-	54.13
MAGN-ITE-95	-	50.79
SBD-ITE-95	-	50.26
MAGN-LAT-95	-	52.44
SBD-LAT-95	-	51.75
MAGN-EAR-97	164413	54.13
SBD-EAR-97	-	54.08
MAGN-ITE-97	-	51.11
SBD-ITE-97	-	52.44
MAGN-LAT-97	-	52.28
SBD-LAT-97	-	52.49
MAGN-EAR-99	56951	80.83
SBD-EAR-99	-	56.73
MAGN-ITE-99	-	53.44
SBD-ITE-99	-	55.35
MAGN-LAT-99	-	53.65
SBD-LAT-99	-	53.07

CIFAR10-NOISY-20%

Model	**Nb. Param.**	**Test Err. (%)**
BASELINE	4523402	20.16
MAGN-EAR-95	228611	19.42
SBD-EAR-95	-	18.84
MAGN-ITE-95	-	19.58
SBD-ITE-95	-	18.16
MAGN-LAT-95	-	19.52
SBD-LAT-95	-	19.02
MAGN-EAR-97	138194	18.14
SBD-EAR-97	-	18.00
MAGN-ITE-97	-	19.11
SBD-ITE-97	-	18.95
MAGN-LAT-97	-	18.58
SBD-LAT-97	-	18.24

CIFAR100-NOISY-20%

Model	**Nb. Param.**	**Test Err. (%)**
BASELINE	4529252	38.51
MAGN-EAR-95	234461	42.66
SBD-EAR-95	-	40.98
MAGN-ITE-95	-	39.94
SBD-ITE-95	-	40.07
MAGN-LAT-95	-	38.78
SBD-LAT-95	-	40.75
MAGN-EAR-97	144044	43.69
SBD-EAR-97	-	42.65
MAGN-ITE-97	-	39.30
SBD-ITE-97	-	39.86
MAGN-LAT-97	-	40.03
SBD-LAT-97	-	42.13

SVHN-NOISY-20%

Model	**Nb. Param.**	**Test Err. (%)**
BASELINE	4523402	13.08
MAGN-EAR-95	228611	11.70
SBD-EAR-95	-	11.83
MAGN-ITE-95	-	11.84
SBD-ITE-95	-	11.85
MAGN-LAT-95	-	11.99
SBD-LAT-95	-	12.10
MAGN-EAR-97	138194	11.97
SBD-EAR-97	-	12.23
MAGN-ITE-97	-	12.36
SBD-ITE-97	-	14.79
MAGN-LAT-97	-	12.18
SBD-LAT-97	-	12.48

PASCALVOC-NOISY-20%

Model	**Nb. Param.**	**Test Err. (%)**
BASELINE	4529252	56.51
MAGN-EAR-95	234461	55.99
SBD-EAR-95	-	56.14
MAGN-ITE-95	-	53.50
SBD-ITE-95	-	56.04
MAGN-LAT-95	-	58.32
SBD-LAT-95	-	54.34
MAGN-EAR-97	144044	56.57
SBD-EAR-97	-	55.30
MAGN-ITE-97	-	58.26
SBD-ITE-97	-	56.20
MAGN-LAT-97	-	57.10
SBD-LAT-97	-	54.87