Coded ResNeXt: a network for designing disentangled information paths

Apostolos Avranas, Marios Kountouris
EURECOM
Sophia Antipolis, FRANCE

Abstract

To avoid treating neural networks as highly complex black boxes, the deep learning research community has tried to build interpretable models allowing humans to understand the decisions taken by the model. Unfortunately, the focus is mostly on manipulating only the very high-level features associated with the last layers. In this work, we look at neural network architectures for classification in a more general way and introduce an algorithm which defines before the training the paths of the network through which the per-class information flows. We show that using our algorithm we can extract a lighter single-purpose binary classifier for a particular class by removing the parameters that do not participate in the predefined information path of that class, which is approximately 60% of the total parameters. Notably, leveraging coding theory to design the information paths enables us to use intermediate network layers for making early predictions without having to evaluate the full network. We demonstrate that a slightly modified ResNeXt model, trained with our algorithm, can achieve higher classification accuracy on CIFAR-10/100 and ImageNet than the original ResNeXt, while having all the aforementioned properties.[1]

1. Introduction

Most successful deep learning architectures for image classification consist of a certain building block that is applied sequentially several times: one block succeeds another until a linear operation finally outputs the model prediction. In deep convolutional neural networks (CNNs), the block consists of a sequence of convolutional operations [23, 24], batch normalizations [17] and rectified linear units (ReLU) [27] activations. Notably, adding a skip connection to every block improves the performance and facilitates very deep architectures called Residual Networks (ResNets) [10]. Another approach relies on applying the convolutional layers in parallel, which results in a multi-branch design. For instance, inception models [43, 44] have blocks with multiple branches, each applying some transformation on the block's input. The input of the next block is then obtained by concatenating the outputs of all branches. The multi-branch design framework can also accommodate skip connections, as initially done in ResNeXt networks [48], and later refined using squeeze-excitation in [15], or a split-attention mechanism in [52]. The starting question of this work is: what is the purpose of multi-branch architectures?

Initially, in AlexNet [20], branches are used to allow "grouped" convolutions that could be distributed across multiple GPUs, which at the time had limited memory. Nowadays, multi-branch architectures are generally used to distribute the parameters of a block into smaller groups such that each group applies a separate transformation to the input. This has proved beneficial compared to keeping all parameters together in a single unique branch per block [48]. Nevertheless, rare are the cases where each branch of a multi-branch architecture is shown to contribute in a different way to the network performance. In most cases, the value of multi-branch architectures is mostly justified by showing an increase in the accuracy of the whole network. An example of the former is SKNet [25], where by zooming in and out the input images it was demonstrated that an attention mechanism [2] pays more attention to the branch with the appropriate receptive field size. Another interesting idea is related to capsules [13, 14], which group neurons into smaller units specialized in recognizing specific visual entities.

In this work, as a means to enhance interpretability, we investigate how to ensure that, in a multi-branch architecture, each branch provably contributes in a different way. In contrast to previous works [13, 14, 25], the role of branches is neither associated to some visual entity, nor to the size of the receptive field. We propose a novel way to organize in a class-wise manner the transformations carried out by the branches. Leveraging concepts from coding theory, we design how to assign each branch to a specific set of classes before training. Specifically, for each block in the network, a binary "codeword" of length equal to the number of branches of the block is assigned to each class. The

[1]Code available at: https://github.com/avranasa/Coded-ResNeXt

codeword of each class then indicates which of the branches in the block will work for that class. That way, by keeping only all the branches of the network assigned to that class, it is possible to form a path unique for that class that traverses the network and through which the information related to that class flows. To showcase the advantages of our idea, we use the the state-of-the-art multi-branch architecture ResNeXt [48] to which we add an architectural tweak.

Our main contributions can be summarized as follows:

- We develop an algorithm that provably controls the path through which the information flows, thus allowing us to design before training one path per class and force the information related to that class to pass through the assigned path.
- Without any additional training, these paths are used to extract for each class a binary classifier that has at least 60% less parameters than the complete network.
- We provide a design for the paths leveraging concepts from coding theory, which enables the utilization of the intermediate layers' output to make early predictions.
- Our algorithm is applied to a slightly modified ResNeXt architecture and we show that the aforementioned desirable properties are achieved while maintaining or even improving classification accuracy.

2. Related Work

Numerous attempts have tried to understand how complex, dense neural networks actually work. For instance, activation maximization is used to find the input that increases the activation of a neuron [8, 30, 31, 49]. Saliency maps [32, 33, 37–39, 41] try to find the pixels that influence the model's prediction the most. The pitfall of prior approaches is that they cannot really explain the reasoning process behind neural networks' decisions and they mainly serve as post hoc visualization methods. Building inherently interpretable models, beyond post hoc approaches, is our key challenge here [34]. There have been several recent efforts [6, 18, 28, 46, 53], but most of them concentrate on enhancing interpretability only in the last layers of the neural network. In [46], the final linear layer is replaced with a differentiable decision tree, and in [53] a loss is used to make each filter of the very high-level convolutional layer represent a specific object part. In [6], the model output is compared with learnt prototypes, whereas in [18] it represents concepts on which humans can intervene.

Differently from previous works, we investigate neural network architectures for classification in a more general way. Specifically, our aim is to control the paths through which information flows throughout the neural network. For that, we employ multi-branch architectures, in which each branch is assigned to specific paths. Roughly speaking, this resembles the idea of disentanglement [1, 5, 12]. In [35] it is stated that one of the ten challenges in interpretability is "disentanglement, which refers to the way to the way information travels through the network". Preferably "information about a specific concept traverses through one part of the network, while information about another concept traverses through a separate part". In this work, instead of concepts such as objects, parts, scenes, materials, textures, etc. [3], information is specifically related to the classes. Prior to training, our algorithm allows to assign for each class a unique path in the network throughout which the information related to that class will flow. The work [**?**] bears some resemblance to ours where they identify such information paths; a key difference is that they use a post hoc method so that the paths are identified and not designed.

3. Coded ResNeXt

3.1. The block

The typical ResNeXt block [48] is depicted in Fig. 1(a). It takes input $x \in \mathbb{R}^{C\times H\times W}$ (C is the number of input channels and H, W are the height and width of the input planes) and outputs y of the same dimensions. It consists of N paths/branches (N is called cardinality in [48]). Each branch, which here is called *sub-neural network* (subNN), performs transformations $\mathcal{T}_n,\ n \in \{1, \cdots N\}$ that are aggregated together with the input x, giving the block's output y:

$$y = x + \sum_{n=1}^{N} \mathcal{T}_n(x). \tag{1}$$

3.1.1 Energy Normalization

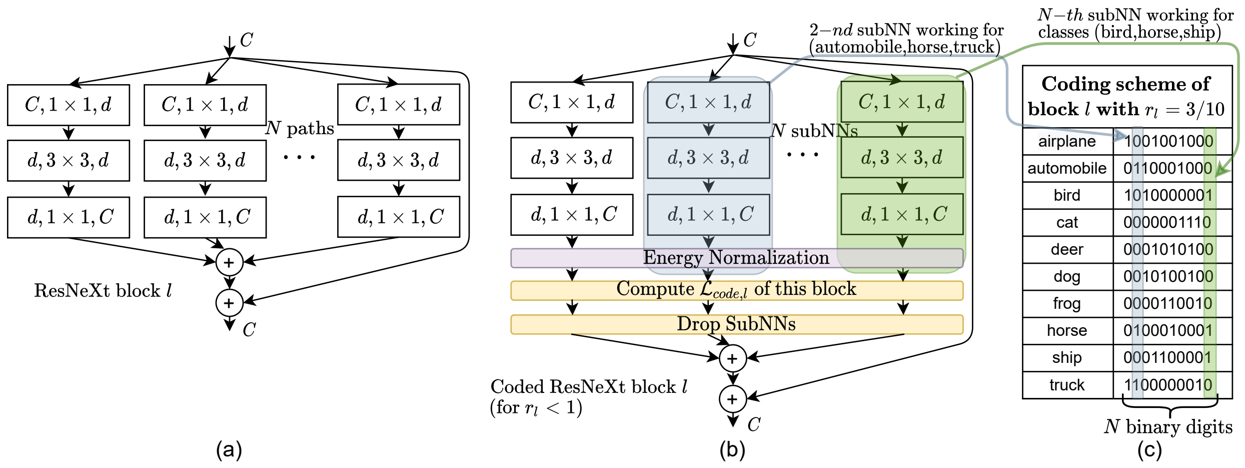
The *Energy Normalization* is the sole architectural change we introduce, and is applied just before aggregating the transformed inputs $t_n = \mathcal{T}_n(x) \in \mathbb{R}^{C\times H\times W}$. If $(t_n)_{c,h,w} \in \mathbb{R}$ is the element of t_n in position (c, h, w), then we define function $\mathcal{E}$ as:

$$\mathcal{E}(t_n) = \frac{1}{CHW}\sum_{c=1}^{C}\sum_{h=1}^{H}\sum_{w=1}^{W}\left((t_n)_{c,h,w}\right)^2, \tag{2}$$

which gives the mean energy of the output signal of the n-th subNN. The Energy Normalization step simply divides the outputs of all branches by a scalar value equal to the square root of the total mean energy:

$$\bar{t_n} = \frac{t_n}{\sqrt{\frac{1}{N}\sum_{i=1}^{N}\mathcal{E}(t_i)}}, \forall n \in \{1, \cdots, N\}. \tag{3}$$

Given that $\mathcal{E}(ax) = a^2\mathcal{E}(x)$ for scalar $a \in \mathbb{R}_{\geq 0}$, it is easy to see that this step is actually standardizing the total energy of the outputs of subNNs, since after that the sum of all subNNs mean energy is $\sum_{n=1}^{N}\mathcal{E}(\bar{t_n}) = N$.


C
$C, 1\times1, d$
$d, 3\times3, d$
$d, 1\times1, C$
N paths
ResNeXt block l
(a)
2−*nd* subNN working for (automobile,horse,truck)
N−*th* subNN working for classes (bird,horse,ship)
N subNNs
Energy Normalization
Compute $\mathcal{L}_{code,l}$ of this block
Drop SubNNs
Coded ResNeXt block l (for $r_l < 1$)
(b)
N binary digits
(c)

Coding scheme of block l with $r_l = 3/10$	
airplane	1001001000
automobile	0110001000
bird	1010000001
cat	0000001110
deer	0001010100
dog	0010100100
frog	0000110010
horse	0100010001
ship	0001100001
truck	1100000010

Figure 1. Building block of ResNeXt and our proposed variation. **(a)**: ResNeXt block. A layer is shown as (# in channels, filter size, # out channels). The layers are composed of a convolutional operation, batch normalization, and ReLU. The last layer's batch normalization is between the two summations and the ReLU comes after the last summation of the skip connection. The same applies to our Coded-ResNext block. **(b)**: Coded-ResNeXt block. With light violet color we depict the architectural addition, and with beige the algorithmic ones. The energy normalization keeps the total sum of the subNNs' output energies constant. Depending on the class of the sample, the loss $\mathcal{L}_{code,l}$ pushes the total energy to be allocated to specific subNNs with a prior to training order. Each subNN's output can be zeroed by the dropSubNNs operation with probability p_{drop}. **(c)**: The prior to training order in which the $\mathcal{L}_{code,l}$ allocates the total energy. We name this table as the coding scheme of the coded ResNeXt block. In the figure it refers to CIFAR-10 that has $K = 10$ classes, each mapped to a binary "codeword" of length equal to the number of subNNs N representing their desirable energies. The ratio $r_l = 3/10$ means that 3 subNNs out of $N = 10$ will be working for each class. The $\mathcal{L}_{code,l}$ tries to match the energy of the subNNs to their corresponding digit, depending on the codeword of the class.

3.1.2 Coding Loss

We present here our first algorithmic addition. After the Energy Normalization we compute a novel loss function coined *coding loss* $\mathcal{L}_{code}$. Assume we have an image classification problem of K classes and that l is the index indicating the position of a ResNeXt block within the whole network. As seen in Fig. 1(c), for that block, we assign to each class a binary codeword $w_{l,k}, k \in \{1, \cdots, K\}$ of length N, indicating which subNNs we want to operate/activate for that class. If the n-th subNN operates for class k, then the n-th digit of $w_{l,k}$ is $(w_{l,k})_n = 1$, and $(w_{l,k})_n = 0$ otherwise. To ensure that each class receives the same number $N_{act,l}$ of operating subNNs, all K codewords are designed with exactly $N_{act,l}$ ones. We define r_l as the ratio

$$r_l = \frac{N_{act,l}}{N}, \tag{4}$$

which measures how much each class uses the block's total computational resources. Given an input image of class k, the objective of coding loss is to push the mean energies of the subNNs inactive for class k to go to zero, and those of the active subNNs to take positive values. The coding loss for the l-th block is given by

$$\mathcal{L}_{code,l} = \frac{1}{N}\sum_{n=1}^{N}(r_l\mathcal{E}(\bar{t_n}) - (w_{l,k})_n)^4. \tag{5}$$

Note that after the energy normalization, the total subNNs' mean energy is $\sum_{n=1}^{N}\mathcal{E}(\bar{t_n}){=}N$ but the codeword has $N_{act,l}{=}r_l N$ ones, hence we multiply $\mathcal{E}(\bar{t_l})$ by r_l. The choice of the loss exponent 4 is justified in the discussion in Sec. 4.5.

3.1.3 DropSubNNs

The second algorithmic addition is a type of dropout [40] similar to techniques such as SpatialDropout [45], StochasticDepth [16], and DropPath [22]. Seeing each subNN as a more complicated neuron, we apply dropout to it such that its output is zeroed with a fixed probability p_{drop}. This method is coined as *DropSubNNs*. Our aim is to reduce the "co-adaptation" effect [40] on the subNN level, according to which subNNs will be collaborating in groups instead of trying to independently produce useful features.

The term *coding scheme* of a block corresponds to the mapping of the classes to codewords as in Fig. 1(c). In our implementation, if the subsequent blocks are designed with the same coding scheme, then no new mask is chosen for each of these blocks; the first block randomly generates one mask which is then reused to the subsequent blocks having the same coding scheme.

3.2. The network

The complete network is constructed as a sequence of blocks. The Energy Normalization, $\mathcal{L}_{code,l}$, and drop-SubNNs are applied only to blocks whose subNNs we want to specialize in some subsets of classes. So for blocks with $r_l = N/N = 1$, we use the conventional ResNeXt block as in Fig. 1(a). In that sense, the ResNeXt model can be seen as a Coded ResNeXt model where all blocks have ratios $r_l = N/N$.

3.2.1 Coding Scheme Construction

We construct one coding scheme per ratio r_l so that a coding scheme is uniquely characterized by the ratio r_l and any two blocks l, l' with $r_l = r_{l'}$ have exactly the same coding scheme. Our approach is based on the following intuitive rule: the deeper in the network a block is (i.e., the larger l is), the smaller the r_l assigned. The first blocks have $r_l = N/N$ so that their subNNs produce low-level features potentially useful for recognizing any of the classes. Deeper blocks have smaller r_l so that their subNNs specialize on a subset of classes. In fact, the last linear layer of the conventional ResNeXt architecture can be seen as K (number of classes) subNNs, each performing a simple linear combination, and where the coding scheme has the lowest possible ratio $r_l = 1/K$ with the codewords being one-hot vectors.

A natural approach could be to use coding schemes so that earlier blocks are tasked with differentiating between superclasses and later blocks are used to select classes within those superclasses. This approach would require the coding scheme to depend on the semantic similarity between the classes. In this work, our *primary goal is to demonstrate that it is possible to specialize subNNs to (defined before training) subsets of classes, even in the case that the classes within those subsets may not be semantically related*. Therefore, for a given r_l we construct the coding scheme in an agnostic way with respect to the nature of the classes. The following three rules are used:

A. The number of "1"s must be equal to $N_{act,l} = r_l N$ with N being the codeword length.

B. The Hamming distance[2] of any pair of codewords should be as high as possible.

C. Seeing the coding scheme as a binary table, as in Fig. 1(c), we require the sum of each column to be approximately the same.

[2]The Hamming distance of two binary words is equal to the number of different digits that they have.

The rationale behind the second rule is to assign each class to a set of classes being as different as possible from the rest, while the third rule aims to avoid overloading or underutilizing any subNN. We remark that finding such coding scheme is very challenging; there is no known way to compute even the function $A_2(N, \mathsf{d})$ which gives the maximum number of binary codewords of length N with minimum Hamming distance d. Moreover, computing $\mathsf{D}(N, K)$ giving the minimum possible Hamming distance of a coding scheme of K codewords is even harder. Using $\mathsf{D}(\cdot)$ one can evaluate $A_2(\cdot)$ through a binary search over K. In our case, the additional constraint of having $N_{act,l}$ "1"s increases the difficulty. Lastly, in addition to the existence of such coding scheme, we are interested in how to realize it. For that, we have to resort to heuristics when developing coding schemes that satisfy to the largest extent the aforementioned three rules, as explained in Appendix A.

3.2.2 Architecture and Total Loss

We compactly describe a Coded-ResNeXt block as $[C_{out}, d, r_l]$, with C_{out} being the number of channels the block outputs and d the bottleneck width as in ResNeXt [48]. A conventional ResNeXt block is expressed as $[C_{out}, d, N/N]$. Following [48], given the number of subNNs N, the bottleneck width d is determined so that the blocks have about the same number of parameters and FLOPs as the corresponding blocks of the original ResNet bottleneck architecture [10].

Table 1 presents the networks trained for CIFAR-10 (C10), CIFAR-100 (C100) [19], and ImageNet 2012 [36] classification datasets. In CIFAR-10/100 we tried to keep N low, but sufficiently high to enable reducing r_l to less than 0.25 and still obtaining a strong coding scheme with minimum Hamming distance larger or equal to 4. For ImageNet we used the default values of ResNeXt-50. Remarkably, even though the number of classes increases exponentially across datasets ($K \in \{10, 100, 1000\}$), the proposed coding methodology allows to efficiently share the subNNs between classes, so that both (a) random pairs of classes are assigned to very different subsets of subNNs; and (b) only a linear increase of the number of subNNs ($N \in \{10, 20, 32\}$) is needed.

Let $\mathcal{L}_{class}$ be the conventional negative cross entropy loss and B_{code} the set of indices pointing to the blocks with ratio $r_l < 1$. The total loss used in order to train the network is

$$\mathcal{L}_{tot} = \mathcal{L}_{class} + \mu \sum_{l \in B_{code}} \mathcal{L}_{code,l} \quad (6)$$

with μ being a constant balancing the two losses. For convenience of exposition and with some abuse of notation, in Eq. (6) both losses are actually the expected values over the distribution of the samples. As commonly done in practice,

stage	**Coded ResNeXt-29 (10×11d)** for CIFAR-10	**Coded ResNeXt-29 (20×6d)** for CIFAR-100	**Coded ResNeXt-50 (32×4d)** for ImageNet
c1	conv $3{\times}3, 64$	conv $3{\times}3, 64$	conv $7{\times}7, 64$, str. 2 $3{\times}3$ max pool, str. 2
c2	$\begin{bmatrix} 256, & 11, \\ 10/10 \end{bmatrix}{\times}3$	$\begin{bmatrix} 256, & 6, \\ 20/20 \end{bmatrix}{\times}3$	$\begin{bmatrix} 256, & 4, \\ 32/32 \end{bmatrix}{\times}3$
c3	$\begin{bmatrix} 512, & 22, \\ \mathbf{5/10} \end{bmatrix}{\times}3$	$\begin{bmatrix} 512, & 12, \\ \mathbf{8/20} \end{bmatrix}{\times}3$	$\begin{bmatrix} 512, & 8, \\ 32/32 \end{bmatrix}{\times}4$
c4	$\begin{bmatrix} 1024, & 44, \\ \mathbf{3/10} \end{bmatrix}{\times}3$	$\begin{bmatrix} 1024, & 24, \\ \mathbf{4/20} \end{bmatrix}{\times}3$	$\begin{bmatrix} 1024, & 16, \\ \mathbf{16/32} \end{bmatrix}{\times}6$
c5	global avg. pool 10-d fc, softmax	global avg. pool 100-d fc, softmax	$\begin{bmatrix} 2048, & 32, \\ \mathbf{8/32} \end{bmatrix}{\times}3$
			global avg. pool 1000-d fc, softmax

Table 1. Architecture for each dataset. A block is described by $[C_{out}, d, N_{act}/N]$, with C_{out} being the number of channels it outputs, d the bottleneck width, N the number of paths/subNNs, and N_{act} the number of active/operating subNNs per class. In the beginning of stages c3 and c4 in all datasets, and additionally for c5 in ImageNet, the feature map size is halved as in [10, 48]. For the CIFAR architectures, stages c2, c3, c4 have approximately 0.2, 0.9, 3.5 million parameters, respectively, and the total architecture has approximately 4.7 million parameters. For ImageNet, stages c2, c3, c4 and c5 have 0.2, 1.2, 7.0 and 14.5 million parameters, respectively, and the total number of parameters is 25.0 millions.

the gradients are computed on the *average* of the losses over the samples of the batch.

4. Experiments

In this section, we present various experimental results to assess the performance of the proposed Coded-ResNeXt. First, we show that our algorithm achieves subNN specialization. We demonstrate this by showing that when subNNs specialized on the class of interest are removed, the performance degrades, whereas it remains the same or even improves when the subNNs removed are not specialized for that class. To further prove the specialization of the subNNs, we test the performance of the following binary classifier: given a certain class, we keep only the subNNs assigned to that class, thus retrieving a lighter single-purpose binary classifier (whose decision is whether the input belongs or not to the class). Next, we show that it is possible to get good predictions from intermediate blocks without evaluating the whole network. Finally, we conduct an ablation study on the two hyperparameters introduced, namely μ that balances the two losses in Eq. (6) and the probability p_{drop} of dropping subNNs.

4.1. Setup and Validation Accuracy

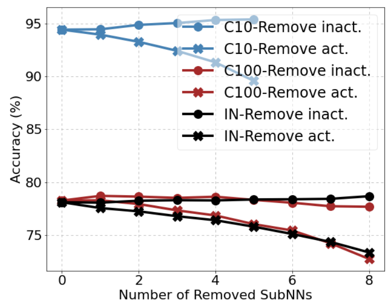
For data augmentation in ImageNet [36], we follow the guidelines in [4] for ResNet-RS-50 in order to train our (Coded-)ResNeXt-50. The input of the model is 160×160 randomly resized and cropped images, for which we use standard values of scale and ratio [43], followed by horizontal flips and RandAugment [7] of layers $N_{aug} = 2$ and magnitude $M_{aug} = 10$. For CIFAR-10/100 and (Coded-)ResNeXt-29, after the standard pad-and-crop and horizontal flips, RandAugment is used again with $(N_{aug}, M_{aug}) = (3, 4)$ for CIFAR-10 and $(1, 2)$ for CIFAR-100[3]. All our experiments are run on Google's Colab TPUv2 ($N_w = 8$ cores with bfloat16 precision). Batch size is picked relatively high to harness TPU speed; for CIFAR datasets, the batch size per core is set to $B_w = 64$ (i.e., effective 512), while for ImageNet, $B_w = 128$ (i.e., effective 1024). We use PyTorch's implementation of stochastic gradient descent with Nesterov momentum [29, 42] equal to 0.9. The weight decay [21] is equal to 10^{-4} for ImageNet and $5 \cdot 10^{-4}$ for CIFAR. Training on ImageNet is performed for 150 epochs using cosine scheduler [26] with initial learning rate defined by the rule $0.1\frac{N_w \cdot B_w}{256} = 0.4$ [9], warmed up for 5 epochs, and decayed until 10^{-5}. Training on CIFAR is performed for 300 epochs with the same scheduler but without warming up, with initial learning rate 0.1, and with decay until 10^{-4}. The implementation of Coded-ResNeXt for ImageNet is based on the timm library [47]. Compared to the corresponding ResNeXt, the throughput, the number of parameters, and the FLOPs are almost the same. Additional details are provided in Appendix B.

Table 2 presents the default values used (unless otherwise stated) for the introduced hyperparameters (μ, p_{drop}) and the achieved validation accuracy. The baseline is the corresponding ResNeXt. We observe a clear improvement in the CIFAR datasets, and in Imagenet our Coded-ResNeXt-50 achieves slightly higher accuracy. We remark that there exist several powerful techniques (label smoothing [44], squeeze excitation [15], exponential moving average, deeper networks, mixup [51], etc.), which are empirically known to improve ImageNet accuracy [4, 11] and which could have been used here to further boost our accuracy. However, we choose to keep our setup as simple as possible in order to clearly test the effect of our architectural and hyperparameter choices.

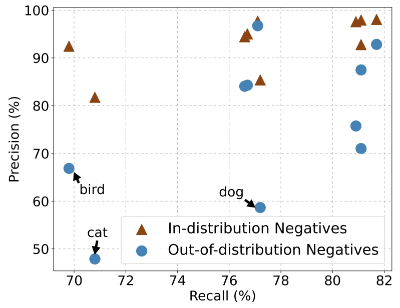
4.2. Specialization

The core idea of this work is to specialize each subNN to specific subset of classes; hence the first experiment is designed to test whether our proposed architecture actually succeeds in achieving specialization. Assuming a subNN is

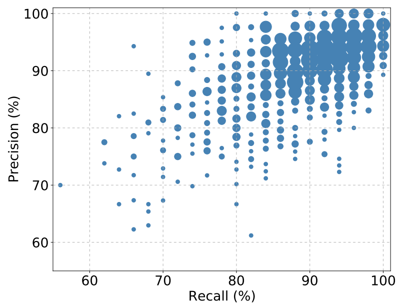
[3]Those values are chosen in [7] for Wide-ResNet-28-2 [50] which, out of all models presented in that work, seems most similar to ResNeXt-29.



(a) Removing subNNs of a specific block.



(b) Precision-Recall on CIFAR-10.



(c) Precision-Recall on ImageNet. The larger the marker, the more points fall into that area.

Figure 2. Demonstrating the specialization of subNNs to their assigned set of classes. **(a)** Performance when removing active versus inactive subNNs from a specific block. **(b)** Precision-Recall from all extracted binary classifiers trained on CIFAR-10. Out-of-distribution negatives are the validation set of CIFAR-100. **(c)** Precision-Recall from all extracted binary classifiers trained on ImageNet.

	(μ, p_{drop})	Coded ResNeXt top-1	ResNeXt top-1
CIFAR-10	(6, 0.1)	**94.41**	93.66
CIFAR-100	(6, 0.1)	**78.28**	76.86
ImageNet	(1, 0.1)	**78.12**	78.05

Table 2. Default hyperparameters and validation accuracy for each dataset.

assigned to actively work for some class, then if this indeed helps on the classification process of images belonging to that class, removing this active subNN should have a negative impact on the performance. On the other hand, if that subNN is not assigned to that class, then it should remain inactive during the process so removing it should have no impact (degradation) on the performance.

For the first experiment we pick a block l from which we randomly remove subNNs[4] in two ways. Given the class of the input image sampled from the validation set, the first way randomly removes $k \leq N_{act,l}$ subNNs from the set of active for that class subNNs. The second way randomly removes $k \leq N - N_{act,l}$ subNNs from the (complementary) set of inactive subNNs for that class. The block we choose in Fig. 2a is the last one of stage c3 (see Tab. 1) for CIFAR datasets and the second of stage c5 for ImageNet.

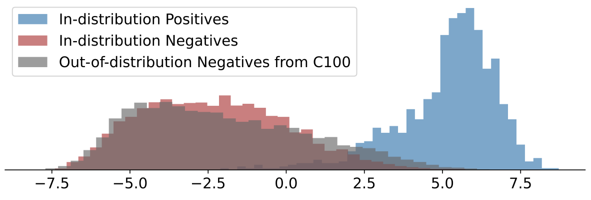
In Fig. 2a, we observe the same behavior across all datasets, which confirms that the more active subNNs are removed, the more the performance degrades. Interestingly, when removing inactive ones, the accuracy tends to increase. Our interpretation is that that even though the inactive subNNs are trained to output zero signal, this is never perfectly achieved in practice and their output always interferes with that of the active subNNs. Thus, taking out the interferers could improve the accuracy. Note that this higher accuracy of the neural network is not actually achievable since to remove a subNN we need to know the class of the input so as to know the set of (in)active subNNs for that class. Finally, we remark that even if all active subNNs are removed from one block the performance does not necessarily plummet because information can still pass from the previous block to the next one through the skip connection.

4.3. Binary Classifier

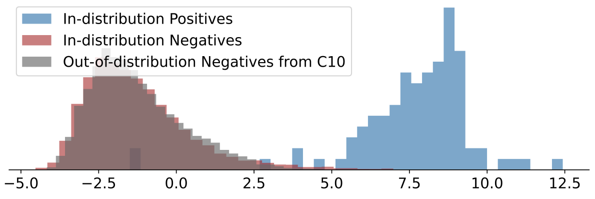
Having confirmed that the subNNs specialize on their assigned subset of classes, we proceed with testing this property to the extreme. For that, we do not only randomly remove few subNNs of one given block, but instead, given class a $k \in \{1, \cdots K\}$, we remove *all* subNNs assigned to classes *other* than k from all blocks. The rationale behind is to check whether one can keep solely the subNNs specialized on one class and obtain a binary classifier capable of recognizing that class among the others. *This binary classifier is considerably lighter than the initial network, since it has only* 38%, 27%, *and* 35% *of the initial parameters for CIFAR-10, CIFAR-100, and ImageNet architectures, respectively.*

In Fig. 3 we pick the first class of CIFAR-10/100 and ImageNet ("airplane", "apple", and "tench" respectively) and remove all inactive subNNs for that class. Additionally, we remove the softmax operation at the end of the network so that its outputs are logits $y \in \mathbb{R}^K$. We look only at the first element of the logits $(y)_1 \in \mathbb{R}$, which is equivalent to removing all except the first row of the parameters of the linear layer. That way we retrieve a sub-model whose output is one dimensional. Figure 3 depicts with blue

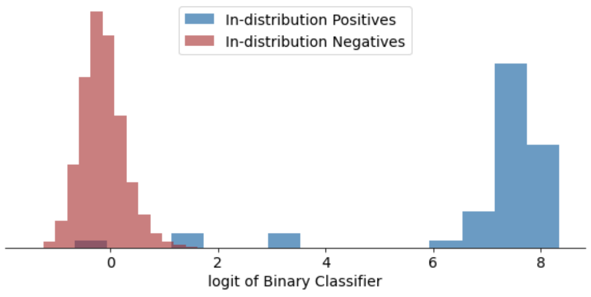
[4]Removing a subNN from a block in this architecture is equivalent to zeroing all of its parameters or to zeroing its output before the Energy Normalization. The latter changes neither the Energy normalization of the other subNNs (since its contribution to the denominator of Eq. (3) is zero), nor it affects anyhow the subsequent summation of the subNNs' outputs.



(a) Binary classifier for airplanes, CIFAR-10.



(b) Binary classifier for apples, CIFAR-100.



(c) Binary classifier for tench, ImageNet.

Figure 3. Output distribution of binary classifier of the first class of each dataset.

the output distribution when inputting samples of the validation set belonging to the first class of the dataset (i.e., in-distribution positives), and with red when the samples belong to some other class (i.e., in-distribution negatives). Evidently, the extracted sub-models indeed operate as binary classifiers giving high output when fed with samples of the class they are specialized in. To further showcase the specialization, for the sub-models trained on CIFAR-10 (resp. CIFAR-100) we input samples that belong to the validation set of CIFAR-100 (resp. CIFAR-10). Those are considered out-of-distribution predictions, since the sub-model has never been trained on such samples. Nevertheless, as Fig. 3 shows, the extracted model continues to perform well.

We show that with our algorithm it is possible to train a large multi-purpose neural network and extract from it a part serving as a "lighter", single-purpose model. This lighter model can be turned into a binary classifier by providing a threshold that gives a positive (resp. negative) prediction if the output of the model is greater (resp. lower) than the threshold value. We set that threshold to the value that maximizes the F1-score of the binary classifier when fed with samples from the training dataset. We can compactly depict its performance (on the validation set) as a point on a plot with the x and y axes being respectively the precision and recall of the binary classifier. In Fig. 2b we show the performance of all $K = 10$ binary classifiers extracted from the coded ResNeXt-29 trained on CIFAR-10. Notably, the worst performance is obtained with the "cat" classifier when fed with CIFAR-100's out-of-distribution samples. This seems reasonable, since we request from the classifier to distinguish cats from classes like leopard, lion, and tiger, but without having "seen" even one sample of them during training.

Figure 2c shows the performance of $K = 1000$ binary classifiers extracted from the Coded ResNeXt-50 trained on ImageNet. The validation set of ImageNet has 50 positives and $999 * 50 = 49950$ negatives per class. We compute the precision and recall by considering only $9 * 50 = 450$ randomly selected negatives. We do that in order to (i) keep the same ratio of positives versus negatives as in CIFAR-10 in order to compare the scores, and (ii) because the dataset is very skewed; e.g. even if setting the threshold very conservatively and allowing to misclassify a negative only 1% of the time, this translates into 500 false positives. Since they are only 50 positives, this means that the precision will be less than 10%. Additional details and plots are provided in Appendix C. In the next subsection, we provide some deeper insights on why the subNNs achieve specialization and we highlight why ResNeXt serves as the appropriate base architecture upon which our idea is built.

4.3.1 Why ResNeXt?

The core idea of our work is to construct a block of many parallel branches/subNNs, each being activated only when a certain rule, here the coding scheme, allows for it and in this way to control the paths through which the information related to each class flows. In order to achieve the desired behavior, we need (i) an operation (energy normalization) that limits how many subNNs on average can be activated; (ii) a loss function forcing the subNNs to comply with the rule of which one should be activated. Intuitively, one could think that for the l-th block with coding $r_l = N_{act,l}/N$, the energy normalization would allow the information to flow through only $N_{act,l}$ out of N paths and the coding loss $\mathcal{L}_{code,l}$ would determine the exact paths.

However, this is not particularly accurate. Let us assume that was the case, i.e., the paths to be activated are solely determined by the energy normalization followed by the coding loss. Then, keeping the energy normalization and coding loss unaltered and changing only the way the outputs of the subNNs are passed to the subsequent blocks, should still allow the extraction of good binary classifiers. It turns out that is not true. If instead of aggregating by *summation*, the subNNs' outputs are *concatenated*, the performance of the extracted binary classifiers becomes poor. Therefore, *it seems that when concatenating the outputs, the* *"information" does not pass only through the paths designated by $\mathcal{L}_{code,l}$*, since the performance degrades when all inactive subNNs (which in theory should not participate in those designated paths) are removed. Let us see why.

The reason is that when concatenating the outputs of the l-th block, the subsequent $l+1$-th block is allowed to take decisions not only based on the signal coming from the active subNNs of the l-th block, but also from the zero signal of the rest. Hence, since the $l+1$-th block also depends on which subNNs of the l-th block output a zero signal, we cannot remove them to construct a binary classifier. On the contrary, using summation to aggregate the outputs prohibits the $l+1$-th block to be dependent on the zero signal of the inactive subNNs of the l-th block. The information that some subNNs provide zero output is lost when adding them to the final output of the block. On the other hand, if the output of the subNNs is concatenated, the information related to which subNNs provide zero output is preserved. For that reason, the ResNeXt architecture (which aggregates the outputs by summation) is very well suited for developing our idea.[5]

4.4. Early Decoding

In this subsection, we keep all subNNs intact and investigate an additional advantage provided by our training algorithm. Given block l with $r_l < 1$, i.e. $l \in B_{code}$, the coding scheme maps each class in a *one-to-one* fashion to a codeword and then the training tries to match the energies of the block's subNNs to that codeword. A natural question is whether it is feasible, using only the energies of the subNNs, to retrieve the codeword that can be used to correctly predict the class of the sample.

In that experiment, we forward the samples of the validation set and compute the energies of the subNNs of the blocks with $r_l < 1$. For each $l \in B_{code}$ we obtain a vector $v_l \in \mathbb{R}^N_{\geq 0}$ containing those energies and we measure the Euclidean distance from v_l to all the codewords $w_{l,k}, k \in \{1, \cdots, K\}$. If $k^\star$ is the class whose codeword $w_{l,k^\star}$ has the smallest distance, then $k^\star$ is the prediction of the l-th block. As a result, each block $l \in B_{code}$ becomes an early decoder predicting label:

$$\arg\min_k ||v_l - w_{l,k}||_2, \; k \in \{1, \cdots, K\} \tag{7}$$

[5]In our initial experiments (before considering ResNeXt), we used an architecture like Coded ResNeXt, but without skip connections and instead of summing the outputs of subNNs (after the energy normalization step) we concatenated them. Algorithmically, we forced the information paths not with a (coding) loss function but by applying a mask in the backward propagation to *block the gradients* of the subNNs that according to the coding scheme should remain inactive. That way, the subNNs were updated only by gradients coming from samples of the subset of classes that the coding scheme had assigned to them. This approach resulted in both good binary classifiers and total multi-class accuracy for that architecture. Unfortunately, adding skip connections the accuracy dropped and that is why we replaced the blocking gradient approach by the coding loss.

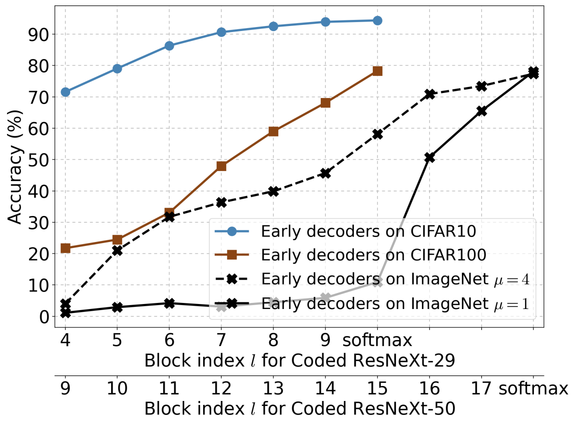


Figure 4. Accuracy of the early decoders of Coded ResNeXt-29 when trained on CIFAR-10 and CIFAR-100 and Coded ResNeXt-50 when trained on ImageNet.

with $||\cdot||_2$ being the L2 norm. This early decoding is enabled and reinforced by the second rule used to create the coding schemes (see Sec. 3.2.1). That rule forces the codewords to be as far apart from each other as possible, and the farther apart the codewords are, the more subNNs should erroneously be active or inactive for the early decoder to predict a wrong label. In Fig. 4 we depict the accuracy of every block $l \in B_{code}$ when functioning as early decoder. For Imagenet we can see that increasing the effect of the coding loss by changing the initial hyperparameter value μ=1 to μ=4 can greatly improve the early decoding, but at the expense of the final total accuracy being reduced to 77.6%. Interestingly, as a sample passes from one block to the next one, the probability of being correctly decoded is increased. This bears a strong resemblance to the decoding procedure in communication systems, where therein a received signal, which is distorted by noise, needs to be matched to the original message transmitted. This is usually achieved through iterative algorithms, which improve the prediction outcome at each iteration until converging to a prediction of the original message with high certainty.

4.5. Ablation Study on Coding Loss and dropSubNN

In this subsection, we investigate the choice of forth power in Eq. (5) for the coding loss and study the effect of the two hyperparameters introduced in the paper, namely the coefficient μ balancing the losses in Eq. (6) and the probability p_{drop} of dropping subNNs.

The role of a subNN in a block using coding is dual. On the one side, it has to provide useful features to the subsequent block for the set of classes it is assigned to; on the other side it should not interfere with the active subNNs for

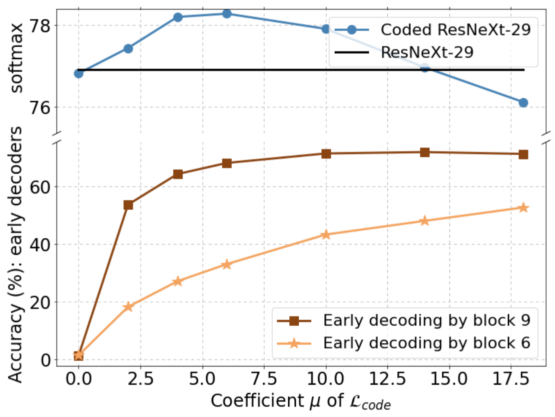


Figure 5. Impact of coefficient μ of the coding loss on CIFAR-100.

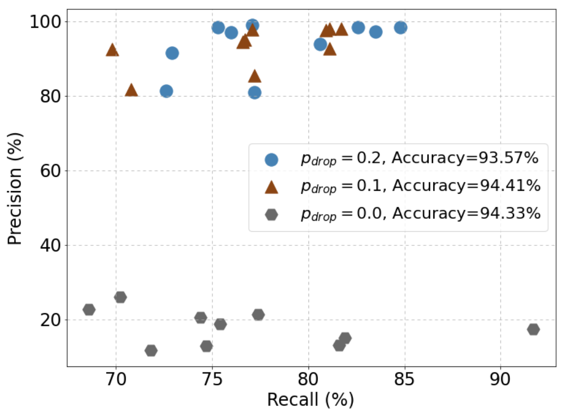


Figure 6. Impact of the probability p_{drop} on the performance of the binary classifiers. The accuracy of the baseline ResNeXt-29 is 93.66%; as we increase p_{drop} the gain in accuracy of our Coded-ResNeXt shrinks but the performance of the binary classifiers is improved.

the rest of the classes. However, the computational capacity of each subNN is limited, hence it is impossible to excel at both. The main task is certainly the first one, i.e., to forward useful features to the next block, whenever dictated by the coding scheme, since if not, the whole network may fail as a classifier. Consequently, we do not want the subNNs to overemphasize on the second(ary) task of not interfering, since this could degrade the performance on the main task. This is the main reason behind the choice of the fourth in the coding loss $\mathcal{L}_{code}$ Eq. (5). An exponent of 4 in the coding loss results in smaller values, closer to zero, than if an exponent of 2 or the absolute value were used. That way, the penalization in the coding loss function is more lenient when the subNNs energies are close to but not exactly equal to the codeword.

Our experiments performed using CIFAR-10 (with $(\mu, p_{drop}) = (6, 0.1)$) confirm the benefit of setting the exponent to 4, since the accuracy drops from 94.4% to 93.1% when the exponent is 2, and further drops to 87.1% when using absolute value. In Fig. 5 we show that as μ increases, "organizing" the energies according to the coding scheme is beneficial to the overall performance until a certain point. Past this point, forcing the subNNs to output a signal of a specific energy value provides only small diminishing gains on the early decoders. Furthermore, it disturbs the entire classification process, thus reducing the final accuracy of the whole network's predictions.

The last experiment concerns the dropSubNN. Dropping randomly some subNNs during training inhibits their "co-adaptation" [40], as they learn not to depend on the others and to perform well even in the absence of some of them. Figure 6 shows that the dropSubNN is essential for the binary classifiers. A small value of $p_{drop} = 0.1$ can greatly boost their performance and even slightly improve the overall accuracy. Further increasing $p_{drop} = 0.2$ degrades the accuracy without improving much the binary classifiers.

5. Conclusion

In this paper, we proposed a modification of ResNeXt, which exhibits several attractive properties. Our algorithm forces the branches of the ResNeXt blocks to specialize on specific subsets of classes. For any class k, we can exploit this specialization property by keeping only the branches assigned to k, thus extracting a binary model for identifying class k that is 60% lighter than the original scheme. Moreover, by leveraging coding theory for the assignment of the branches to classes, we enable the use of intermediate layers for making early predictions without having to evaluate the full network. Experiments show that those desirable properties can be achieved without compromising the accuracy, which remains similar or even improves compared to the conventional ResNeXt. We believe that this framework could lead to the development of novel architectures that provide both better interpretability and higher accuracy.

References

[1] Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *The Journal of Machine Learning Research*, 19(1):1947–1980, 2018. 2

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 1

[3] David Bau, Bolei Zhou, Aditya Khosla, Aude Oliva, and Antonio Torralba. Network dissection: Quantifying inter-

pretability of deep visual representations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6541–6549, 2017. 2

[4] Irwan Bello, William Fedus, Xianzhi Du, Ekin D Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. Revisiting resnets: Improved training and scaling strategies. In *NeurIPS*, 2021. 5

[5] Yoshua Bengio. *Learning deep architectures for AI*. Now Publishers Inc, 2009. 2

[6] Chaofan Chen, Oscar Li, Chaofan Tao, Alina Jade Barnett, Jonathan Su, and Cynthia Rudin. This looks like that: deep learning for interpretable image recognition. *arXiv preprint arXiv:1806.10574*, 2018. 2

[7] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPRW*, pages 702–703, 2020. 5

[8] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009. 2

[9] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 5

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016. 1, 4, 5

[11] Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *CVPR*, pages 558–567, 2019. 5

[12] Irina Higgins, David Amos, David Pfau, Sebastien Racaniere, Loic Matthey, Danilo Rezende, and Alexander Lerchner. Towards a definition of disentangled representations. *arXiv preprint arXiv:1812.02230*, 2018. 2

[13] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *Int. Conf. on Art. Neural Net.*, pages 44–51. Springer, 2011. 1

[14] Geoffrey E Hinton, Sara Sabour, and Nicholas Frosst. Matrix capsules with em routing. In *ICLR*, 2018. 1

[15] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *CVPR*, pages 7132–7141, 2018. 1, 5

[16] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, pages 646–661. Springer, 2016. 3

[17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Int. Conf. on Mach. Learn.*, pages 448–456. PMLR, 2015. 1

[18] Pang Wei Koh, Thao Nguyen, Yew Siang Tang, Stephen Mussmann, Emma Pierson, Been Kim, and Percy Liang. Concept bottleneck models. In *International Conference on Machine Learning*, pages 5338–5348. PMLR, 2020. 2

[19] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 4

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 25:1097–1105, 2012. 1

[21] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *NeurIPS*, pages 950–957, 1992. 5

[22] Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016. 3

[23] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *NeurIPS*, 2, 1989. 1

[24] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1

[25] Xiang Li, Wenhai Wang, Xiaolin Hu, and Jian Yang. Selective kernel networks. In *CVPR*, pages 510–519, 2019. 1

[26] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016. 5

[27] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Int. Conf. on Mach. Learn.*, 2010. 1

[28] Meike Nauta, Ron van Bree, and Christin Seifert. Neural prototype trees for interpretable fine-grained image recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14933–14943, 2021. 2

[29] Yurii E Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. akad. nauk Sssr*, volume 269, pages 543–547, 1983. 5

[30] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *Advances in neural information processing systems*, 29:3387–3395, 2016. 2

[31] Aaron Van Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1747–1756, New York, New York, USA, 20–22 Jun 2016. PMLR. 2

[32] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018. 2

[33] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016. 2

[34] Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019. 2

[35] Cynthia Rudin, Chaofan Chen, Zhi Chen, Haiyang Huang, Lesia Semenova, and Chudi Zhong. Interpretable machine learning: Fundamental principles and 10 grand challenges. *arXiv preprint arXiv:2103.11251*, 2021. 2

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 4, 5

[37] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *CVPR*, pages 618–626, 2017. 2

[38] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013. 2

[39] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *arXiv preprint arXiv:1706.03825*, 2017. 2

[40] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Mach. Learn. Research*, 15(56):1929–1958, 2014. 3, 9

[41] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Int. Conf. on Mach. Learn.*, pages 3319–3328. PMLR, 2017. 2

[42] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In *Int. Conf. on Mach. Learn.*, pages 1139–1147. PMLR, 2013. 5

[43] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015. 1, 5

[44] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, pages 2818–2826, 2016. 1, 5

[45] Jonathan Tompson, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler. Efficient object localization using convolutional networks. In *CVPR*, pages 648–656, 2015. 3

[46] Alvin Wan, Lisa Dunlap, Daniel Ho, Jihan Yin, Scott Lee, Henry Jin, Suzanne Petryk, Sarah Adel Bargal, and Joseph E Gonzalez. Nbdt: Neural-backed decision trees. *arXiv preprint arXiv:2004.00221*, 2020. 2

[47] Ross Wightman. Pytorch image models. `https://github.com/rwightman/pytorch-image-models/tree/bits_and_tpu`, 2019. 5, 13

[48] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *CVPR*, pages 1492–1500, 2017. 1, 2, 4, 5, 13

[49] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015. 2

[50] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016. 5

[51] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018. 5

[52] Hang Zhang, Chongruo Wu, Zhongyue Zhang, Yi Zhu, Haibin Lin, Zhi Zhang, Yue Sun, Tong He, Jonas Mueller, R Manmatha, et al. Resnest: Split-attention networks. *arXiv preprint arXiv:2004.08955*, 2020. 1

[53] Quanshi Zhang, Ying Nian Wu, and Song-Chun Zhu. Interpretable convolutional neural networks. In *CVPR*, pages 8827–8836, 2018. 2

A. Coding schemes

In this section we present the construction methodology of coding schemes. For the reader's convenience, we repeat the three rules the coding scheme should comply with. In this section we drop the subscript l from $r_l, N_{act,l}$, since we want to show for any block how in general we constructed its coding scheme. Given some block with ratio r and number of branches/subNNs N, these rules are as follows:

A. The number of "1"s must be equal to $N_{act} = rN$ with N being the codeword's length.

B. The Hamming distance of any pair of codewords should be as high as possible.

C. Seeing the coding scheme as a binary table, with each row representing a class and each column a subNN as in Tab. 3, we require the sum of each column to be approximately the same.[6]

The first rule is mandatory and we only consider codewords with number of "1"s equal to N_{act}. The other two rules serve as guidelines and we try to follow them to the maximum extent possible. Let S_{min} (resp. S_{max}) be the sum of the columns with the minimum (resp. maximum) sum. The third rule is fully satisfied if $S_{min} = S_{max}$. This is not realizable for all ratios r. The ratio r must be chosen taking the number of classes K into account as follows: The number of "1"s in the binary table is KN_{act}. Assuming a coding scheme with $S_{min} = S_{max} = S_{opt}$, the number of "1"s is also equal to NS_{opt}. This brings the equality $KN_{act} = NS_{opt} \iff S_{opt} = rK \in \mathbb{N}$. Hence, a necessary condition to be able to find a coding scheme of $S_{min} = S_{max}$ is $rK \in \mathbb{N}$.

The number of possible combinations of choosing p elements from a set of n distinct elements is given by $C(n,p) = \frac{n!}{p!(n-p)!}$. A coding scheme where each class is mapped to a distinct codeword exists if $C(N, rN) \geq K$.

Let H_{min} be the minimum Hamming distance among the pairs of the coding scheme. In CIFAR-10 for instance, as can be verified from Tab. 3, we have $H_{min} = 4$ for both $r = 5/10$ and $r = 3/10$ (check the pair horse-ship). Obviously, the higher $C(N, rN)$ is, the larger the set of acceptable codewords to choose for the coding scheme, so the larger H_{min} can be achieved. Nevertheless, not every value of H_{min} is achievable. There is a value above which there is no such coding scheme with K codewords.

Choosing r, N such that $rK \in \mathbb{N}$ and $C(N, rN) \geq K$, we proceed in finding the coding scheme. In Algorithm 1 we give the pseudocode for a simplified version of the algorithm generating the coding schemes. We used this algorithm to generate the coding schemes for CIFAR-100 and ImageNet. For CIFAR-10 the length of the codewords is $N = 10$ which is small enough to allow for the use of an approach similar to exhaustive search.

[6]Note that due to the first rule the sum of each row is equal to N_{act}

	$r = 5/10$	$r = 3/10$
airplane	1010100011	1001001000
automobile	0101010101	0110001000
bird	1101100010	1010000001
cat	0011001101	0000001110
deer	1010010101	0001010100
dog	1001001110	0010100100
frog	1011101000	0000110010
horse	0100011110	0100010001
ship	0110111000	0001100001
truck	0100110011	1100000010

Table 3. The coding schemes used in CIFAR-10.

Algorithm 1 Algorithm for generating a coding scheme

```
Require: $K, N, N_{act}, H_{min}$
1: function MinHamDist(codeword $w$, set $G$)
2:     $d \leftarrow \infty$
3:     for $w_g$ in $G$ do
4:         if HammingDistance$(w, w_g) < d$ then
5:             $d \leftarrow$ HammingDistance$(w, w_g)$
6:     return $d$

7: function Score(coding scheme $C$)
8:     $S_{min} \leftarrow \min\{$sum column of coding scheme $C\}$
9:     $S_{max} \leftarrow \max\{$sum column of coding scheme $C\}$
10:    return $S_{max} - S_{min}$            ▷ The lower, the better

11: $L \leftarrow$List of all codewords with $N_{act}$ ones
12: $L \leftarrow$ sort$(L)$
13: $G \leftarrow \{\}$
14: for $w$ in $L$ do
15:     if MinHamDist$(w, G) \geq H_{min}$ then
16:         $G \leftarrow G \cup \{w\}$
17: if cardinality of $G < K$ then
18:     exit                        ▷ Unable to find a coding scheme
19: $BestScore \leftarrow \infty$
20: for all $C \subseteq G$ with $|C| = K$ do
21:     $score \leftarrow$Score$(C)$
22:     if $score = 0$ then
23:         return $C$       ▷ Found solution satisfying rule C
24:     else if $score < BestScore$ then
25:         $BestScore \leftarrow score$
26:         $BestSchemeFound \leftarrow C$
27: return $BestSchemeFound$
```

The Algorithm 1 starts by creating a list of length $C(N, rN)$ (in the sense of the programming language

	GFlops	#Params	Throughput	RAM
ResNeXt-50 (32×4d)	2.196	$25.0 \cdot 10^6$	$439 \frac{\text{samples}}{\text{sec}}$	15.0GB
Coded-ResNeXt-50 (32×4d)	2.269	$25.0 \cdot 10^6$	$447 \frac{\text{samples}}{\text{sec}}$	15.7GB

Table 4. Computational costs on ImageNet.

Python) with all possible codewords that satisfy rule A. Each codeword is actually a binary number as well, so it can be seen as an integer. That way, it is possible to sort the list in the next step. This step is crucial since randomly picking codewords is very inefficient when creating afterwards the set G. The set G is a set containing codewords, where the Hamming distance of each pair of codewords is at least H_{min}. The larger the set G is, the easier is to find a subset of cardinality K that satisfies all three rules and could be the coding scheme. Randomly picking codewords from the list L would result in a much smaller set G than when picked in a sorted way. To understand why this holds true, we provide the following analogy.

Imagine having balls with radius of H_{min} instead of codewords. The problem is to fit as many balls as possible within a square. If we start filling the square by randomly placing the balls inside the square, this will quickly result in no extra ball actually fitting within the space left by the already placed balls, even though there is still a lot of space unoccupied. On the other hand, if balls are placed in an ordered way, starting from the edges and progressively placing them as close as possible to either the edges of the square or to the balls already placed, then many more balls will eventually fit in the square.

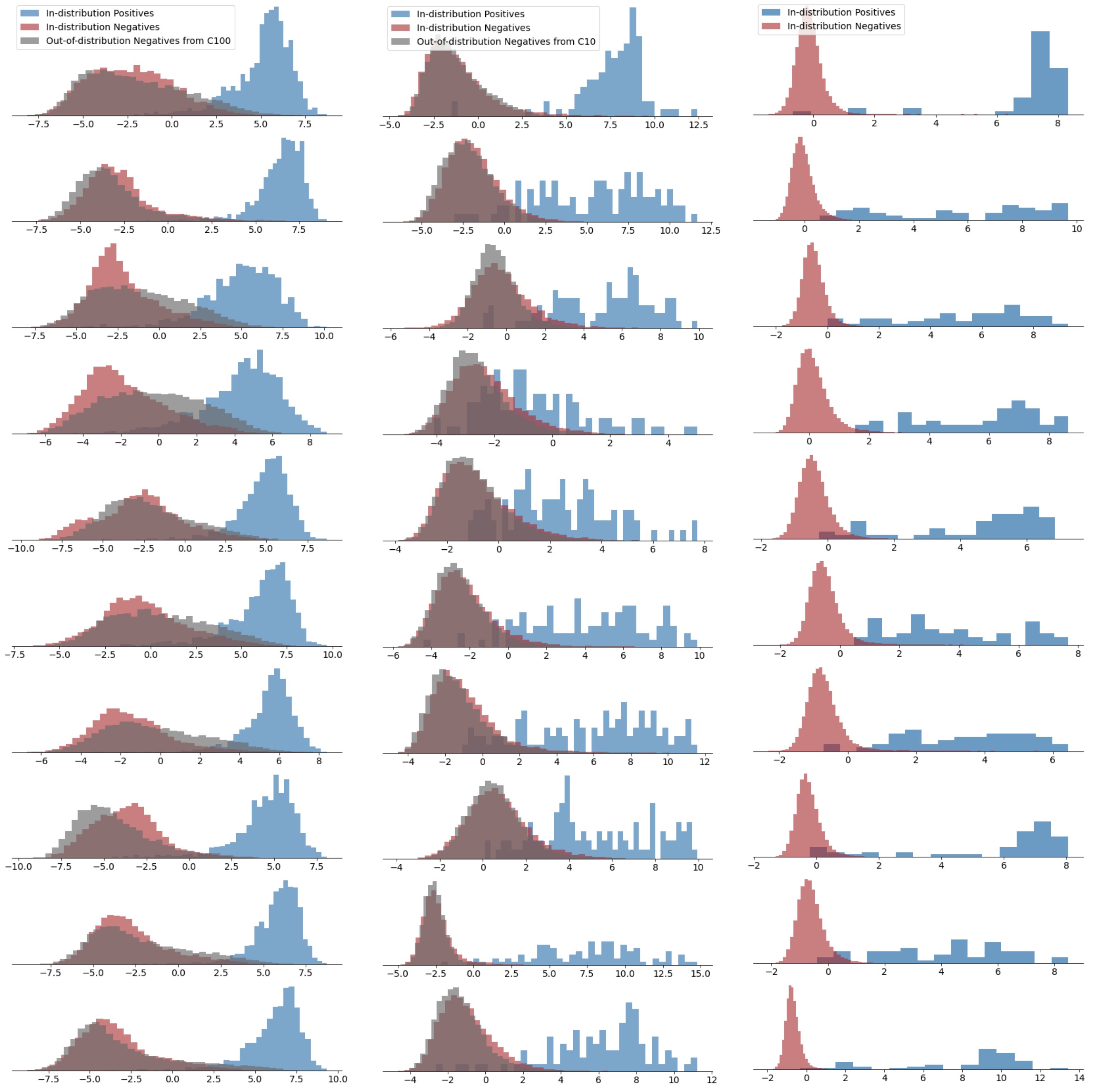
The Algorithm 1 is a simplified version of our implementation. In line 20, the number of possible sets C that can be chosen from G is most likely to be extremely big. In that case, we resort to additional heuristics for picking only good candidates for C. Further details can be found in our Python code made available. Finally the coding scheme for CIFAR-100 with $r = 8/20$ is retrieved using the arguments $(K, N, N_{act}, H_{min}) = (100, 20, 8, 8)$ in the Algorithm 1 and with $r = 4/20$ using $(K, N, N_{act}, H_{min}) = (100, 20, 4, 4)$. For both ratios the coding schemes found entirely satisfy rule C, i.e., $S_{min} = S_{max}$. The coding scheme for ImageNet with $r = 16/32$ is retrieved using arguments $(K, N, N_{act}, H_{min}) = (1000, 32, 16, 10)$ achieving $S_{min} = 499 \approx S_{max} = 501$. For the coding scheme with $r = 8/32$, we use arguments $(K, N, N_{act}, H_{min}) = (1000, 32, 8, 6)$, achieving $S_{min} = 249 \approx S_{max} = 251$. For further details on the exact coding scheme, please refer to our Python code.

B. Implementation details and computational costs

In this section we present some additional implementation details and show the computational costs for training a Coded-ResNeXt relative to the costs of training a conventional ResNeXt. We focus on ImageNet since it is considerably more demanding in terms of computational resources than the CIFAR datasets, and also because we use a dedicated for ImageNet training library [47] for the implementation. That way we can directly compare ResNeXt with Coded-ResNeXt and focus on the computational impact of Coded-ResNeXt's additional steps by minimizing the impact our implementation could have on the performance.

We would like first to clarify that in a ResNeXt block or Coded-ResNeXt block with ratio $r = 1$ the last convolution layer of the block is not necessary to be implemented as a grouped convolution followed by an aggregation by summation (as it is implied from Fig. 1). Since for those blocks we do not apply the additional operations of Energy Normalization, coding Loss and dropSubNNs, the grouped convolution with the subsequent aggregation by summation of the subNNs' outputs can be combined into a simple convolutional layer. This is how we implement ResNeXt and blocks of Coded-ResNeXt with ratio $r = 1$. This also coincides with the way ResNeXt is implemented in the original work [48] and it is illustrated in the Fig. 3 of that paper where they show that they propose Fig. 3c architecture instead of Fig. 3a. We refer the reader to [48] for more information and our Python code.

The throughput and average RAM consumption of Tab. 4 have been on the second epoch, the reason being that "generally the first epoch is slow with Pytorch XLA" [47]. For ResNeXt-50 we measure the FLOPs using the library fvcore. For Coded-ResNeXt we add on the flops computed for ResNeXt-50 the flops needed for the Energy Normalization step. For a block of N subNNs and output of dimensions $\mathbb{R}^{C \times H \times W}$, the energy normalization requires roughly $3 \times N \times C \times H \times W$ flops. The multiplication by 3 comes from the fact that the energy normalization step first raises the elements to the power of 2, element-wise, second it takes the mean of the squares, and finally it performs an element-wise division with the square root of the total mean energy. We see in Tab. 4 that for all metrics, Coded-ResNeXt does not introduce any significant additional com-



(a) All Binary Classifiers extracted from Coded-ResNeXt-29 (10×11d) trained on CIFAR-10.

(b) Binary Classifiers for the first 10 classes extracted from Coded-ResNeXt-29 (20×6d) trained on CIFAR-100.

(c) Binary Classifiers for the first 10 classes extracted from Coded-ResNeXt-50 (32×4d) trained on ImageNet.

Figure 7. Distribution of the output (logit) of binary classifiers

putational cost when trained on TPU.[7]

[7]We also tried training on a GPU provided by Google Colab, without relying on timm library. Coded-ResNeXt was more than two times slower compared to ResNeXt on that hardware. Nonetheless, the training of both ResNeXt and Coded-ResNeXt was faster on TPU, hence we kept TPU as our choice of hardware.

As mentioned in section Sec. 4.1 we trained on TPUv2 provided by Google Colab. One epoch in Imagenet took roughly 45 minutes. With an account of Colab-Pro+ a training session of 24 hours is required (i.e., around 30 epochs). We saved a checkpoint per epoch and after these 24 hours, we had to restart the session and start the training from the latest saved checkpoint. Due to lack of powerful resources, we only tested a limited set of hyperparameters, namely $(\mu, p_{drop}) \in \{(6, 0.1), (4, 0.1), (4, 0.05), (1, 0.1)\}$. The model with $(\mu, p_{drop}) = (4, 0.1)$ is the one used for the Fig. 4 which gave a bit lower validation accuracy (77.6%). We also tried two models with different coding schemes. One where in stage c3 we used coding ratio $r = 24/32$ and a second where in stage c4 we used for the first 3 blocks $r = 16/32$ and for the last 3 blocks $r = 8/32$, and for stage c5 $r = 4/32$. Finally, we note that we used 0.9 for the "momentum" (Pytorch's term) of all Batch-Normalization layers.

C. Additional details and plots on binary classifiers

In this section, we provide additional information and plots regarding the binary classifiers that can be extracted after training a Coded-ResNeXt. First, we remind the definitions of Precision and Recall. True positives TP (resp. true negatives TN) represent the number of positive (resp. negative) samples that the binary classifier correctly predicts as positives (resp. negatives). False positives FP (resp. false negatives FN) represent the number of negative (resp. positive) samples that the binary classifier erroneously predicts as positives (resp. negatives). The precision and recall are defined as follows

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}.$$

Precision measures out of predicted positives (i.e., $TP + FP$) how many of those predictions are correct. Recall measures out of all actual positives (i.e., $TP + FN$) how many were found by the binary classifier.

In Fig. 7 we give the output distributions of the binary classifiers for the first 10 classes of CIFAR-10/100 and ImageNet when fed with an input of in-distribution positive/negative samples, and out-of-distribution negative samples. The first row coincides with Fig. 3. We observe that depending on the threshold value a binary classifier uses (above which the classifier predicts positive and below negative), different values of precision and recall can be attained. Increasing the threshold value gives fewer false positives FP, but unfortunately also more false negatives FN, so increasing the threshold improves the precision but degrades the recall. Therefore, high recall can be exchanged for high precision by increasing the threshold, or the opposite by decreasing it.

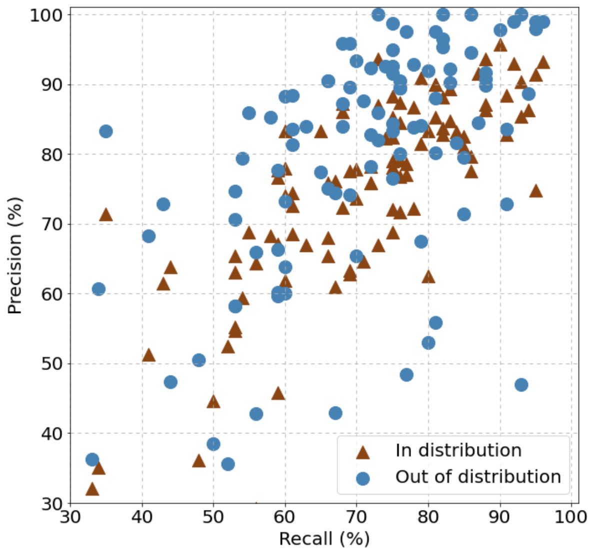


Figure 8. Precision-Recall on CIFAR-100.

Regarding plots Fig. 2b, Fig. 8 and Fig. 2c, which depict the precision and recall achieved by the binary classifiers trained on CIFAR-10, CIFAR-100 and ImageNet, we would like to make the following remarks:[8] first, when testing a binary classifier in the case of either in-distribution or out-of-distribution negatives, the set of samples of the validation set serving as positives remains the same. Since by definition recall depends only on this set of positives, its value remains unaltered in both testing cases. Second, in the case of ImageNet, we have only 50 samples of the validation set serving as positives for each binary classifier. Therefore, the true positives can only take integer values from 0 to 50 and $Recall \in \{0, \frac{1}{50}, \frac{2}{50}, \cdots, \frac{50}{50}\}$. This is the reason why in Fig. 2c, the points appear to follow some kind of grid and are aligned in specific vertical lines.

Finally and interestingly, we see that the binary classifiers trained on CIFAR-100 perform at the same level no matter whether the negatives come from in-distribution or out-of-distribution. This is in contrast to the binary classifiers trained on CIFAR-10, where out-of-distribution negatives clearly decrease the precision Figs. 2b, 7 and 8). Both CIFAR-10 and CIFAR-100 datasets have the same number of training samples (50000), but CIFAR-10 has 5000 samples per class and CIFAR-100 has 500 per class. Therefore, the binary classifiers of CIFAR-100 have been trained on

[8]In Fig. 8, which corresponds to CIFAR-100, we followed the same procedure as in Fig. 3c for ImageNet. Specifically, we randomly sample from the set of negatives a subset of size 9 times bigger than the size of positives. We use that subset of negatives to evaluate the precision and recall of the binary classifier.

$\frac{5000}{500} = 10$ times fewer positives but had "seen" negatives from $\frac{100-1}{10-1} = 11$ times more classes. The observation that CIFAR-100 binary classifiers perform significantly better with out-of-distribution negatives, agrees with our intuition that a model works better in unknown situations if trained with fewer samples per situation but covering a big range of situations than opposite, i.e., trained with many samples per situation but considering only few different situations.