Towards Disentangling Information Paths with Coded ResNeXt

Anonymous Author(s)
Affiliation
Address
email

Abstract

The conventional, widely used treatment of deep learning models as black boxes provides limited or no insights into the mechanisms that guide the neural network decisions. Significant research effort has been dedicated to building interpretable models to address this issue. Most efforts either focus on the high-level features associated with the last layers, or attempt to interpret the output of a single layer. In this paper, we take a novel approach to enhance the transparency of the function of the whole network. We propose a neural network architecture for classification in which the information that is relevant to each class flows through specific paths. These paths are designed in advance before training leveraging coding theory. Moreover, the paths do not depend on the semantic similarities between classes. A key property is that each path can be used as an autonomous single-purpose model. This enables to obtain, without any additional training and for any class, a lightweight binary classifier that has at least 60% fewer parameters than the original network. Furthermore, our coding theory based approach allows the neural network to make early predictions at intermediate layers during inference, without requiring its full evaluation. Remarkably, the proposed architecture provides all the aforementioned properties while significantly improving the overall accuracy. We demonstrate these properties on a slightly modified ResNeXt model tested on CIFAR-10/100 and ImageNet-1k.

1 Introduction

Most successful deep learning architectures for image classification consist of a certain building block applied sequentially several times: one block follows another until a linear operation finally outputs the model prediction. In deep convolutional neural networks (CNNs), the block consists of multiple convolutional operations [35, 36] applied sequentially. Nonetheless, there are numerous proposals placing the convolutional layers in parallel, forming multi-branch designs. For instance, inception models [71, 72] use blocks with multiple branches, each applying some convolutional operations on the block's input and finally concatenating at the end the output of all branches. The multi-branch design framework can also accommodate skip connections [22], as initially done in ResNeXt networks [81], and later refined using squeeze-excitation in [25], or a split-attention mechanism in [85]. A first question we ask in this work is: *what is the purpose of multi-branch architectures?*

Initially, in AlexNet [32] two branches were employed to allow the distribution of the model across two GPUs, which at that time had limited memory. Nowadays, multi-branch architectures are commonly used for distributing the parameters of a block into branches such that each one applies a separate transformation to the input. However, rare are the cases where each branch is shown to contribute in a different way. One example is SKNet [39], in which each branch is associated with different receptive field size and zooming in or out of an input image activates the appropriate branch. Nonetheless, the value of multi-branch networks is mostly justified by achieving a higher accuracy.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Multi-branched blocks are also used for network architecture search, where the block/cell architecture is optimized selecting the number of branches, the operation that each performs, and how they are combined [41, 54, 56, 87, 89]. Still, the focus therein was on improving accuracy.

In this work, we investigate how to ensure that each branch provably contributes in a different way in a multi-branch architecture. We propose a novel way to organize in a class-wise manner the transformations carried out by the branches. Before the training starts, we assign each branch to a specific set of classes. This set remains fixed and we train the branches to activate only to classes within that set using a loss function that pushes the output to be zero for the samples that do not belong to their assigned set of classes. Thanks to this assignment, once the network is trained, for any given class there is a unique path traversing the network through which the information related to that class flows. To showcase the unique features and the advantages of our idea, we use the state-of-the-art multi-branch architecture ResNeXt [81], to which we add a small number of modifications.

Our main contributions can be summarized as follows:

- We modify the ResNeXt architecture so that it functions in a more transparent way by forcing the information related to each class to flow through well-defined network paths.
- As a proof of concept, we show that in order to form those paths, it is not necessary to rely on the semantics of each class and the similarity between classes.
- Without any additional training, we can obtain a single-purpose model per class operating as a binary classifier that has 60% fewer parameters as compared to the complete network.
- We demonstrate that the intermediate layers can be used both for making early predictions and for providing a confidence level for correctness of the network's final prediction.
- The proposed Coded ResNeXt significantly improves ResNeXt accuracy across all tested datasets.

2 Related Work

There have been numerous attempts to understand how deep neural networks actually work. For instance, activation maximization tries to find the input that increases neuron activation [13, 52, 50, 82]. Saliency maps [66, 64, 67, 69, 53, 57] find the pixels that have the largest influence on the model prediction. However, such approaches cannot really explain how the network functions, and they mainly serve as post hoc visualization methods [58]. There are many proposals interpretable by design [76, 86, 6, 48, 30], but most of them focus on enhancing interpretability only in the last layers.

Another direction for enhancing interpretability is through disentanglement. While there is not yet a generally accepted definition, disentanglement aims at separating the main factors that are present in the data distribution [4, 23, 43]. Still, the efforts concentrate on disentangling the factors at a single vector/tensor (the latent representation), which is either the input or the output of a network. This applies to (variational) autoencoders [24, 5, 7, 29, 14], generative adversarial networks [8, 45, 28], normalizing flows [15, 62], or even architectures that aim to decompose the content to the style representation of an image [19, 17]. In contrast, we approach disentanglement as "the way information travels through the network" [59] and look at the neural network architectures for classification as a whole. Specifically, our goal is to control the paths through which information flows by assigning each part of the network to a specific subsets of classes. Related to our work is [78], where they interpret a deep neural network by identifying such information paths; one key difference is that they use a post hoc method, so the paths are identified and not designed.

There are two additional lines of work related to ours. The first line proposes to dynamically control the path that a sample follows through the network by letting an extra network choose which parts to be pruned/omitted [38, 9, 2, 18, 40, 77, 74, 80, 42]. This brings memory and speedup gains during inference, since the samples do not pass through the entire network. Our proposal has a few key differences with these works. First, we do not use any extra network that has to be trained to learn good paths. Second, our paths are defined prior to training (and thus not learnt). Most importantly, these works do no guarantee that two samples of the same class follow the same path, so it is not possible to extract class-specific network portions to use as single-purpose models. The second line concerns the models that can make early predictions without evaluating the whole network [63]. Similarly to these works, we also apply loss functions to the middle layers [37, 65]; however, in our case this not only allows us to make early predictions, but also to control the information flow of each class, enabling us to e.g. extract lightweight binary classifiers from the trained network.

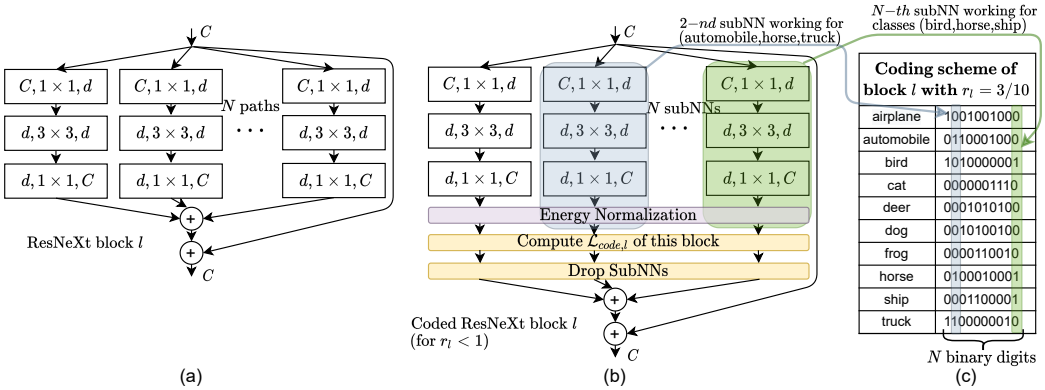


Figure 1: Building block of ResNeXt and our proposed variant. **(a)**: ResNeXt block. A layer is shown as (# in channels, kernel size, # out channels). **(b)**: Coded ResNeXt block. With light violet color we depict the architectural addition and with beige the algorithmic ones. The energy normalization keeps the total sum of the subNNs' output energies constant. Each subNN's output can be zeroed by the dropSubNNs operation with probability p_{drop}. Depending on the class of the sample, the loss $\mathcal{L}_{code,l}$ pushes the total energy to be allocated among subNNs with a specific order. **(c)**: An example of the order we used for CIFAR-10. We name this table as the coding scheme of the Coded ResNeXt block. The ratio $r_l = 3/10$ means that for the l-th block $N_{act,l} = 3$ out of $N = 10$ subNNs will work for each class. The $\mathcal{L}_{code,l}$ tries to match the output energy of the subNNs to their corresponding digit, depending on the binary codeword of the class.

While in this work we assign classes to parts of the networks arbitrarily, an interesting possibility is to exploit semantic/visual similarities for the assignment, similarly to [11, 12]. This may lead to improvements in both performance *and* interpretability; however, it comes with some caveats. In particular, it is not always available or straightforward to obtain the semantic relationships. In [11], for example, to perform classification on ImageNet they had to resort to another database (WordNet [16]).

3 Coded ResNeXt

3.1 The block

The typical ResNeXt block [81] 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, respectively) and outputs y of the same dimensions. It consists of N paths/branches (N is called cardinality in [81]). Each branch, which is called *sub-neural network* (subNN) here, performs transformation $\mathcal{T}_n,\ n \in \{1, \cdots N\}$ which is 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

For the Coded ResNeXt block depicted in Fig. 1(b), the sole architectural change we introduce is the *Energy Normalization* applied just before aggregating the transformed inputs $\mathcal{T}_n(x) \in \mathbb{R}^{C \times H \times W}$. For convenience, let $t_n := \mathcal{T}_n(x)$. 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. Energy Normalization simply divides the outputs of all branches by a scalar value equal to the square root of the total mean energy:

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

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 normalizes the total energy, since after it the sum of the energy of all subNNs becomes $\sum_{n=1}^{N} \mathcal{E}(\bar{t_n}) = N$.

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* $\mathcal{L}_{code}$. Consider a classification problem of K classes. Let l be the index of the position of a ResNeXt block within the 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 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 the ratio

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

which measures how much each class uses the block's total computational resources. We term the mapping of the classes to codewords, as in Fig. 1(c), the *coding scheme* of the block.

Given an input of class k, the coding loss forces the mean energies of the subNNs inactive for class k to zero and those of the active subNNs to positive values. The coding loss for the l-th block is

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

Note that after the Energy Normalization, the total subNNs mean energy is $\sum_{n=1}^{N} \mathcal{E}(\bar{t_n}) = N$, while the codeword has $N_{act,l} = r_l N$ ones, hence we multiply $\mathcal{E}(\bar{t_l})$ by r_l. We remark that the choice of setting the exponent to 4 is carefully made. For example, setting it to 2, the accuracy for CIFAR-10 drops from 94.4% to 93.1%, which further drops to 87.1% if the absolute value is used. The exponent of 2 is much more demanding than the one of 4 on matching precisely the output energies to the rules of the coding scheme. The exponent of 2 seems to considerably restrict the flexibility of the function of the subNNs, thus degrading the overall performance. This trend is exacerbated with using the absolute value.

3.1.3 DropSubNNs

The second algorithmic addition is a type of dropout [68], similar to techniques such as SpatialDropout [73], StochasticDepth [26], and DropPath [34]. Seeing each subNN as one more complicated neuron, we apply dropout to it, so 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 [68] on the subNN level, according to which subNNs collaborate in groups instead of trying to independently produce useful features. In our implementation, we apply the same random mask to all blocks that have 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 dropSubNNs are applied only to blocks whose subNNs we want to specialize in some subsets of classes. Thus, 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 is a Coded ResNeXt model where all blocks have $r_l = N/N$.

3.2.1 Coding Scheme Construction

We remark that the coding scheme is constructed before training, and that the subNNs are trained to comply with this fixed scheme. In general, the coding scheme can be arbitrary, and can possibly incorporate semantic similarities between classes. However, the primary goal of this work is to make a proof of concept that *it is possible to specialize subNNs to subsets of classes defined before training, even in the case when the classes within those subsets may not be semantically related*. Specifically, we found that even when the coding schemes are designed in an agnostic way with respect to the nature of the classes, good performance is guaranteed if some general rules are followed.

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. A general rule we

stage	**Coded ResNeXt-29 (10×11d)** for CIFAR-10	**Coded ResNeXt-29 (20×6d)** for CIFAR-100	**Coded ResNeXt-50 (32×4d)** for ImageNet
s0	conv $3\times3, 64$	conv $3\times3, 64$	conv $7\times7, 64$, str. 2, 3×3 max pool, str. 2
s1	$[256, 11, 10/10]\times3$	$[256, 6, 20/20] \times 3$	$[256, 4, 32/32]\times3$
s2	$[512, 22, \mathbf{5/10}]\times3$	$[512, 12, \mathbf{8/20}] \times 3$	$[512, 8, 32/32]\times4$
s3	$[1024, 44, \mathbf{3/10}]\times3$	$[1024, 24, \mathbf{4/20}] \times 3$	$[1024, 16, \mathbf{16/32}]\times6$
s4	global avg. pool, 10-d fc	global avg. pool, 100-d fc	$[2048, 32, \mathbf{8/32}]\times3$
			global avg. pool,1000-d fc

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 and d being the bottleneck width. For CIFAR architectures, stages s1, s2, s3 have approximately 0.2, 0.9, 3.5 million parameters respectively (in total $4.7M$). For ImageNet, s1, s2, s3 and s4 have $0.2M$, $1.2M$, $7.0M$ and $14.5M$ respectively (in total $25.0M$).

follow is that the deeper in the network a block is (i.e., the larger l is), the smaller is 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. [1] This rule is not only intuitive, but works better in practice. For instance, in CIFAR-100, changing the proposed ratios ($20/20$, $8/20$, $4/20$) to ($8/20$, $8/20$, $8/20$) drops the accuracy from 78.8% to 77.9%, and inverting the order to ($4/20$, $8/20$, $20/20$) gives 76.7%.

Given a block l with ratio r_l, we would like the coding scheme to satisfy the following three rules:

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

Moreover, we want to avoid under- or over-utilizing any subNN, in the sense of assigning too few or too many classes for it to process. As a result, the second rule is:

B. Seeing the coding scheme as a binary table, as in Fig. 1(c), the sum of each column should be approximately the same.

Finally, we aim at making the set of subNNs dedicated to work for a class to be as different as possible from the sets assigned to the rest of the classes. This translates to:

C. The minimum Hamming distance between all pairs of codewords should be as high as possible.

Given r_l and N, many coding schemes that follow the above rules may exist. For example, permutating the rows and/or the columns of the binary matrix in Fig. 1(c) gives new valid coding schemes. We experimentally checked that any scheme that satisfies the above properties gives similar results. Finding any of those schemes is a NP-Hard problem. In Appendix A we provide an algorithm that finds good coding schemes according to the rules above which we used to generate the codes of all our experiments. On a high-level, the algorithm first constructs the set of all binary codewords of length N with N_{act} ones (rule A). Secondly, it extracts from it a subset containing only codewords whose mutual Hamming distance is always higher than a given threshold (rule C). Finally, it extracts multiple combinations of K codewords from that subset and checks which one is a good coding scheme in terms of how well the rule B is satisfied.

3.2.2 Architecture and Total Loss

We succinctly describe a Coded ResNeXt block as $[C_{out}, d, r_l]$, with C_{out} being the number of channels the block outputs and d being the bottleneck width as in ResNeXt [81]. A conventional ResNeXt block is expressed as $[C_{out}, d, N/N]$. Table 1 presents the networks trained for CIFAR-10 (C10), CIFAR-100 (C100) [31], and ImageNet 2012 [60] classification datasets. In CIFAR-10/100 we chose N to be small yet sufficiently high to enable reducing r_l to less than 0.25 and still obtaining a coding scheme with minimum Hamming distance not less than 4. For ImageNet we used the default values of ResNeXt-50. Remarkably, even though the number of classes increases exponentially across

[1] In fact, the last linear layer of the ResNeXt can be seen as K subNNs, each performing a simple linear combination, and the coding scheme has the lowest possible ratio $r_l = 1/K$ (i.e. codewords are one-hot vectors).

datasets ($K \in \{10, 100, 1000\}$), a strong coding scheme can be found to efficiently share the subNNs between classes, so that (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 cross entropy loss and B_{code} the set of indices l pointing to the blocks with ratio $r_l < 1$. Let μ be a loss-balancing constant; then the total loss used to train the network is

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

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. To demonstrate this we show that when the 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, given a class, we keep only the subNNs assigned to that class. That way, we retrieve a lightweight single-purpose binary classifier accurately deciding whether the input sample belongs or not to the class. Finally, we show that it is possible to get good predictions from intermediate blocks without evaluating the whole network. Those predictions can also be used to provide confidence on whether the final network's prediction is correct.

4.1 Setup and Validation Accuracy

In order to make a fair comparison with ResNeXt, on ImageNet [60] we follow the training process proposed by timm library [1]. The epochs are 250 (first 5 as warmup [20] and last 10 cooling down), the batch size is 1536, and the learning rate 0.6. RandAugment [10] of 2 layers and magnitude 7 (varied with a standard deviation of 0.5) is used and also random erasing augmentation [88] with probability 0.4 and 3 recounts. We diverge from the timm's proposed process only on the resolution of the input *training* images. We reduce the resolution from 224 to 160, since on the TPU-v2 of Google COLAB (the platform used for our experiments) the training would take more than three weeks. Still, timm reports 79.77% accuracy for the ResNeXt-50, which is clearly smaller than the one of Coded ResNeXt-50, despite being trained with lower resolution. In Appendix B we give further details, including the training of CIFAR.

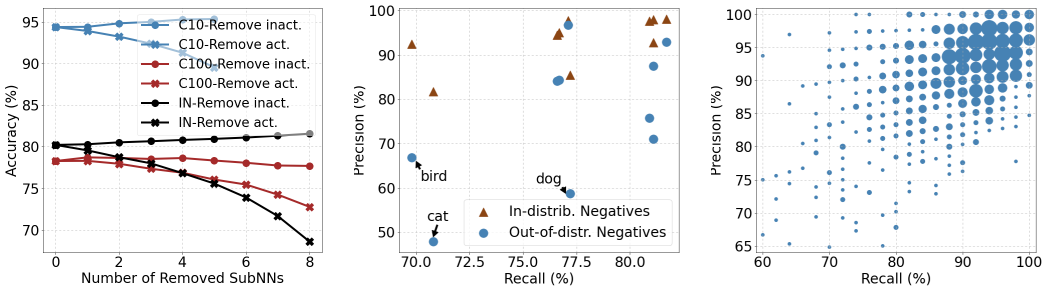
	(μ, p_{drop})	Coded ResNeXt	ResNeXt
CIFAR-10	(6, 0.1)	**94.41%**	93.66%
CIFAR-100	(6, 0.1)	**78.76%**	76.86%
ImageNet	(2, 0.1)	**80.24%**	79.50%

Table 2: Default hyperparameters and validation accuracies. For Coded ResNeXt on ImageNet 80.24% is the mean of 3 runs, which gave almost identical results $(80.21\%, 80.25\%, 80.26\%)$.

In Table 2 we compare the accuracy of Coded ResNeXt against the corresponding ResNeXt (i.e., when setting all ratios r_l equal to 1). We observe a clear improvement of the accuracy across all datasets. Surprisingly, forcing the subNNs to specialize to specific set of classes yields significant gains even if the assignment of classes to subNNs is done in a way agnostic to the semantics of the classes. Table 2 presents the default values used for the introduced hyperparameters (μ, p_{drop}) and the achieved validation accuracy. In Appendix D we perform an ablation study on those hyperparameters.

4.2 Specialization

A key idea of our work is to specialize each subNN to specific subset of classes; hence the first experiment is designed to test whether our architecture succeeds in achieving specialization. Assuming a subNN is assigned to activate for some class, then if this subNN helps indeed on the classification process of images belonging to that class, removing this active subNN should negatively impact this process. 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.



(a) Removing subNNs of a block. (b) Precision-Recall on CIFAR-10. (c) Precision-Recall on ImageNet.

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. The larger the marker, the more points fall into that area.

For the first experiment we pick a block l from which we randomly remove subNNs[2] 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. For illustration, in Fig. 2a we pick the last block of stage s2 in the architecture for CIFAR (see Table 1) and the second of stage s4 for that for ImageNet. Figures with respect to other blocks are presented in Appendix F.

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 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 a priori 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. We believe that the reason behind this is that, in that case, 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, instead of randomly removing few subNNs from one block, given a class k, we remove from *all* blocks all subNNs not assigned to class k. The rationale behind this is to check whether by keeping only the subNNs specialized on one class we can obtain a binary classifier (BC) capable of recognizing that class among the others.

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. We also remove from the final linear layer everything except its first row of parameters so as to keep only the first logit corresponding to the first class. That way, we retrieve a sub-model whose output is one dimensional. Figure 3 depicts with blue 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). Clearly, the extracted sub-models do operate as BCs giving high output when fed with samples of the class for which they are specialized. 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 (OOD) predictions, since the sub-model has never been trained on such samples. Nevertheless, as

[2]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.

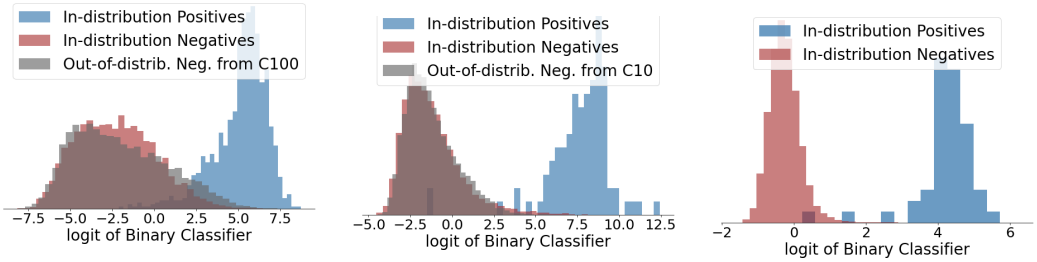


Figure 3: Output distribution of Binary Classifier (BC) of the first class (airplanes, apples, tench) of each dataset (CIFAR-10, CIFAR-100, ImageNet, respectively).

Fig. 3 shows, the extracted BC still perform very well. We remark that the functional lottery ticket hypothesis [84] possibly justifies the good OOD performance of the extracted BC. This hypothesis states that every full network contains a subnetwork that can achieve better OOD performance.

Therefore, *with a single training of a large multi-purpose neural network, we can straightforwardly extract multiple single-purpose models that are considerably lighter (*38%*,* 27%*, and* 35% *of the initial parameters for CIFAR-10, CIFAR-100, and ImageNet architectures, respectively)*. Given a threshold distinguishing between positive and negative predictions, each of those models becomes a BC. We set that threshold to the value maximizing the F1-score of the BC when fed with samples from the training dataset. In Figs. 2b and 2c[3] the performance of the BCs (on the validation set and the out-of-distribution set) is depicted in precision-recall plots. Notably, for CIFAR-10, the worst performance is obtained by the BC for "cats" 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" any sample of them during training.

It is known that in ResNet, complete blocks can be removed without severely degrading the accuracy [75]. Hence, it is reasonable to ask whether the conventional ResNeXt is also robust to the removal of subNNs and whether good BCs can be extracted from it without the need for our proposed modifications. Interestingly, this is not the case and the answer is negative. In CIFAR-10 for instance, the extracted BCs from the Coded ResNeXt give on average precision 93% and recall 77% (in-distribution). Attempting to extract likewise BCs from a ResNeXt leads to precision 13% and recall 56%. Finally, given a class, the complete ResNeXt architecture can be seen as a BC by considering its output to be only the corresponding logit. Comparing such BCs to the extracted BCs seems unfair since the extracted BCs not only have 2.5 times fewer parameters but also have never been trained as independent models. Nonetheless, it may serve as a baseline. For CIFAR-10 this baseline gives BCs with average precision 79% and recall 94%. Additional details and plots are provided in Appendix F.

4.3.1 Why ResNeXt?

In this subsection, we provide insights on why the subNNs achieve specialization and we highlight why ResNeXt serves as the appropriate architecture upon which to build our idea. The objective of our work is to construct networks in which the per-class information is forced to flow through specific paths (determined here by the coding schemes). To achieve this, we employ (i) an operation (energy normalization) that limits how many subNNs can be activated; and (ii) a loss function forcing which ones should be activated. Intuitively, those operations should suffice for constraining the information to flow through the active subNNs. A natural question that arises is how accurate this is. Let us assume that it is accurate. Then, keeping those operations unaltered and changing only the way the subNNs' outputs are passed to the subsequent blocks should not impact the flow of information. However, if instead of aggregating them by *summation*, they are *concatenated*, the performance of the extracted BCs becomes poor (precision $< 20\%$). It seems that the concatenation inhibits

[3]The validation set of ImageNet has 50 positives and $999{*}50{=}49950$ negatives per class. In Figure 2c we consider only $9{*}50{=}450$ randomly selected negatives to compute the precision and recall. We do that (i) in order to keep the same ratio of positives versus negatives as in CIFAR-10 and allow comparison, and (ii) because the dataset is very skewed; e.g., even a very conservative threshold that misclassifies only 1% of the negatives results into approximately 500 false positives. Since they are only 50 positives, the precision becomes 10%.

the "information" to pass only through the designated paths, since the performance degrades when inactive subNNs (which in theory should not participate in those paths) are removed. Let us see why.

When concatenating the outputs of the l-th block, the information about which are the inactive subNNs is preserved, thus the $l+1$-th block may depend its function on which subNNs of the l-th block provide zero output. This allows information to "leak" from the inactive subNNs. On the contrary, the information that some subNNs provide zero output is lost when adding them to the final output of the block. For that reason, the ResNeXt architecture (which aggregates the outputs by summation) is very well suited for developing our idea of controlling the information paths.

4.4 Early Decoding

Coded ResNeXt improves accuracy over ResNeXt while enabling the extraction of multiple lighter single-purpose models with a single training and providing transparency on how information flows throughout the network. Here we show that leveraging coding theory to design when and which subNNs should activate allows exploiting Coded ResNeXt in other ways.

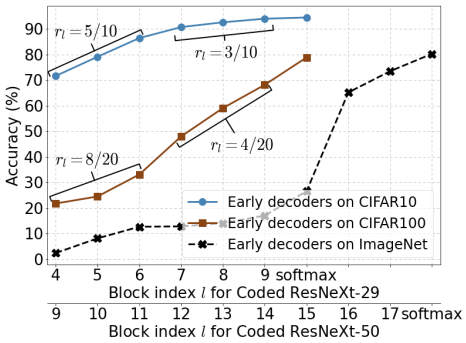


Figure 4: Accuracy of the early decoders.

Given block l with $r_l < 1$, i.e., $l \in B_{code}$, the coding scheme maps each class $k \in \{1, \cdots, K\}$ in a *one-to-one* fashion to a codeword $w_{l,k}$ and then the training pushes the energies $v_l \in \mathbb{R}^N_{\geq 0}$ of the block's subNNs output to match that codeword. This allows for each $l \in B_{code}$ to measure the vector $v_l \in \mathbb{R}^N_{\geq 0}$, find the codeword $w_{l,k}, k \in \{1, \cdots, K\}$ having the minimum distance to v_l, and consequently predict the class of the sample. As a result, each block $l \in B_{code}$ becomes an early decoder predicting label: $\arg\min_k ||v_l - w_{l,k}||_2,\ k \in \{1, \cdots, K\}$ with $|| \cdot ||_2$ being the L2 norm.

In Fig. 4 we depict the accuracy of every block $l \in B_{code}$ when functioning as an early decoder. Interestingly, as a sample passes from one block to the next one, the probability of being correctly decoded increases. In Appendix D we show that as the coefficient μ of $\mathcal{L}_{code,l}$ increases, the early decoders improve, but past a certain point this comes at the expense of the overall accuracy.

Finally, the early decoders can be used to improve the measurement of the confidence that the network's final prediction is correct. Indeed, the more decoders agree with the final prediction, the higher is the probability to be correct. Due to space limitation, we analyze this utility in Appendix C.

5 Conclusion

In this paper, we proposed a network architecture in which the information related to each class flows through distinctive and clearly defined paths. We depart from the ResNeXt architecture and apply few yet crucial modifications that attain both higher accuracy and have several additional attractive properties. First, we specialize each part of the model on a specific and fixed subset of classes, which –given a certain class– enables to obtain a binary classifier for that class by keeping only the relevant parts of the network. Second, it allows to obtain early predictions without the need for entirely evaluating the network. Third, if fully evaluated, a confidence level can be produced on the correctness of the final prediction. While in this work we achieved specialization without having to rely on the semantic similarities between classes, we conjecture that further gains (higher accuracy, lighter binary classifiers, smaller network, etc.) can be obtained by exploiting those similarities. Finally, while in this work we have shown that our method outperforms ResNeXt in classification, future directions include comparing in other computer vision tasks such as detection, few-shot learning and robustness against adversarial attacks.

References

[1] timmdocs, training scripts. `https://timm.fast.ai/training_scripts`. Accessed: 25-4-2022.

[2] Babak Ehteshami Bejnordi, Tijmen Blankevoort, and Max Welling. Batch-shaping for learning conditional channel gated networks. *arXiv preprint arXiv:1907.06627*, 2019.

[3] 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.

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1798–1828, 2013.

[5] Christopher P Burgess, Irina Higgins, Arka Pal, Loic Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. Understanding disentangling in β-vae. *arXiv preprint arXiv:1804.03599*, 2018.

[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.

[7] Ricky TQ Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. Isolating sources of disentanglement in variational autoencoders. *Advances in neural information processing systems*, 31, 2018.

[8] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. *Advances in neural information processing systems*, 29, 2016.

[9] Zhourong Chen, Yang Li, Samy Bengio, and Si Si. You look twice: Gaternet for dynamic filter selection in CNNs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9172–9180, 2019.

[10] 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.

[11] Jia Deng, Nan Ding, Yangqing Jia, Andrea Frome, Kevin Murphy, Samy Bengio, Yuan Li, Hartmut Neven, and Hartwig Adam. Large-scale object classification using label relation graphs. In *European conference on computer vision*, pages 48–64. Springer, 2014.

[12] Jia Deng, Sanjeev Satheesh, Alexander Berg, and Fei Li. Fast and balanced: Efficient label tree learning for large scale object recognition. *Advances in Neural Information Processing Systems*, 24, 2011.

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

[14] Babak Esmaeili, Hao Wu, Sarthak Jain, Alican Bozkurt, Narayanaswamy Siddharth, Brooks Paige, Dana H Brooks, Jennifer Dy, and Jan-Willem Meent. Structured disentangled representations. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2525–2534. PMLR, 2019.

[15] Patrick Esser, Robin Rombach, and Bjorn Ommer. A disentangling invertible interpretation network for explaining latent representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9223–9232, 2020.

[16] Christiane Fellbaum et al. Wordnet: An electronic lexical database mit press. *Cambridge, Massachusetts*, 1998.

[17] Aviv Gabbay and Yedid Hoshen. Demystifying inter-class disentanglement. *arXiv preprint arXiv:1906.11796*, 2019.

[18] Xitong Gao, Yiren Zhao, Łukasz Dudziak, Robert Mullins, and Cheng-zhong Xu. Dynamic channel pruning: Feature boosting and suppression. *arXiv preprint arXiv:1810.05331*, 2018.

[19] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2414–2423, 2016.

[20] 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.

[21] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning*, pages 1321–1330. PMLR, 2017.

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

[23] 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.

[24] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. β-vae: Learning basic visual concepts with a constrained variational framework. 2016.

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

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

[27] 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.

[28] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4401–4410, 2019.

[29] Hyunjik Kim and Andriy Mnih. Disentangling by factorising. In *International Conference on Machine Learning*, pages 2649–2658. PMLR, 2018.

[30] 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.

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

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

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

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

[35] 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.

[36] 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.

[37] Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, and Zhuowen Tu. Deeply-supervised nets. In *Artificial intelligence and statistics*, pages 562–570. PMLR, 2015.

[38] Changlin Li, Guangrun Wang, Bing Wang, Xiaodan Liang, Zhihui Li, and Xiaojun Chang. Dynamic slimmable network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8607–8617, 2021.

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

[40] Ji Lin, Yongming Rao, Jiwen Lu, and Jie Zhou. Runtime neural pruning. *Advances in neural information processing systems*, 30, 2017.

[41] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[42] Lanlan Liu and Jia Deng. Dynamic deep neural networks: Optimizing accuracy-efficiency trade-offs by selective execution. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[43] X Liu, P Sanchez, S Thermos, AQ O'Neil, and SA Tsaftaris. Learning disentangled representations in the imaging domain. *arXiv preprint arXiv:2108.12043*, 2021.

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

[45] Sudipto Mukherjee, Himanshu Asnani, Eugene Lin, and Sreeram Kannan. Clustergan: Latent space clustering in generative adversarial networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4610–4617, 2019.

[46] Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. Obtaining well calibrated probabilities using bayesian binning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

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

[48] 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.

[49] 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.

[50] 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.

[51] Alexandru Niculescu-Mizil and Rich Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22nd international conference on Machine learning*, pages 625–632, 2005.

[52] 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.

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

[54] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

[55] John Platt et al. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74, 1999.

[56] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789, 2019.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[62] Aadhithya Sankar, Matthias Keicher, Rami Eisawy, Abhijeet Parida, Franz Pfister, Seong Tae Kim, and Nassir Navab. Glowin: A flow-based invertible generative framework for learning disentangled feature representations in medical images. *arXiv preprint arXiv:2103.10868*, 2021.

[63] Simone Scardapane, Michele Scarpiniti, Enzo Baccarelli, and Aurelio Uncini. Why should we add early exits to neural networks? *Cognitive Computation*, 12(5):954–966, 2020.

[64] 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.

[65] Robik Shrestha, Kushal Kafle, and Christopher Kanan. Occamnets: Mitigating dataset bias by favoring simpler hypotheses. *arXiv preprint arXiv:2204.02426*, 2022.

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

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

[68] 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.

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

[70] 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.

[71] 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.

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

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

[74] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–18, 2018.

[75] Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. *Advances in neural information processing systems*, 29, 2016.

[76] 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.

[77] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 409–424, 2018.

[78] Yulong Wang, Hang Su, Bo Zhang, and Xiaolin Hu. Interpret neural networks by identifying critical data routing paths. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, June 2018.

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

[80] Zuxuan Wu, Tushar Nagarajan, Abhishek Kumar, Steven Rennie, Larry S Davis, Kristen Grauman, and Rogerio Feris. Blockdrop: Dynamic inference paths in residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8817–8826, 2018.

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

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

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

[84] Dinghuai Zhang, Kartik Ahuja, Yilun Xu, Yisen Wang, and Aaron Courville. Can subnetwork structure be the key to out-of-distribution generalization? In *International Conference on Machine Learning*, pages 12356–12367. PMLR, 2021.

[85] 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.

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

[87] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432, 2018.

[88] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13001–13008, 2020.

[89] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] The datasets are publicly available and the code is included in the supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Coding schemes

In this section we present the construction methodology of coding schemes. We drop the subscript l from r_l and $N_{act,l}$ and show the general rule for an arbitrary block for which these values are given. For the reader's convenience, we repeat the three rules the coding scheme should comply with. 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 length.
B. Seeing the coding scheme as a binary table, with each row representing a class and each column a subNN as in Table 3, we require the sum of each column to be approximately the same.[4]
C. The Hamming distance of any pair of codewords should be as high as possible.

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 second rule is fully satisfied if $S_{min} = S_{max}$. This is not realizable for all ratios r. The ratio r must be chosen taking into account the number of classes K 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 $KN_{act} = NS_{opt} \Leftrightarrow S_{opt} = rK \in \mathbb{N}$. Hence, a necessary condition to be able to find a coding scheme with $S_{min} = S_{max}$ is that $rK \in \mathbb{N}$.

The number of possible combinations for 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 $C(N, rN) \geq K$. Therefore the chosen r should satisfy this inequality.

Let H_{min} be the minimum Hamming distance within the set of all possible pairs of codewords in the coding scheme. In CIFAR-10 for instance, as can be verified from Table 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, and the larger the H_{min} that 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.

	$r = 5/10$	$r = 3/10$
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.

After choosing r, N such that $rK \in \mathbb{N}$ and $C(N, rN) \geq K$, we proceed to find the coding scheme. Finding a coding scheme that satisfies the three rules mentioned above is very challenging. Actually, there is no known way to compute even the "basic" function $A_2(N, \mathsf{d})$ that gives the maximum number of binary codewords of length N with minimum Hamming distance d. Moreover, computing the function $\mathsf{D}(N, K)$ that gives 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} "1"s increases the difficulty. Lastly, in addition to knowing the existence of such a coding scheme, we are interested in realizing it, i.e. in generating a valid set of codewords for that scheme. For that, we have to resort to heuristics that satisfy to the largest extent the three aforementioned rules. In Algorithm 1 we give the pseudocode of the algorithm used to generate the coding schemes of 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.

Algorithm 1 starts by creating a list of length $C(N, rN)$ with all possible codewords that satisfy rule A. Each codeword can be mapped to the integer whose binary representation matches the codeword. These integers are used to sort the list of codewords in line 12. This step is crucial, since randomly picking codewords is very inefficient for creating the set G in the subsequent lines. The set G is a set of codewords in which all possible pair of codewords belonging in this set have Hamming distance

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

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: $L_{sorted} \leftarrow \text{sort}(L)$
13: $G \leftarrow \{\}$
14: **for** w in L_{sorted} **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** $BestSchemeFound$

between them at least H_{min}. The larger G is, the easier it is to find a subset of cardinality K that satisfies all three rules above.

We now give some intuition on why picking codewords randomly from the set L (see line 11) would result in a much smaller set G than the method proposed that picks them sequentially from L_{sorted}. Consider the following analogy: imagine having disks with radius of H_{min} instead of codewords. The problem is to fit as many non-overlapping disks as possible inside a square. If we start filling the square by randomly placing the disks inside the square, this will quickly result in no extra disk actually fitting within the space left by the already placed ones, even though there is still a lot of space unoccupied. On the other hand, if the disks are placed in an ordered way, for example starting from the edges and progressively placing them as close as possible to the already placed disks, then many more disks will eventually fit.

Algorithm 1 is a simplified version of our implementation. In line 20, the number of possible sets C that can be chosen from G can be extremely large. In that case, we resort to additional heuristics for picking only good candidates for C. Further details can be found in our Python code. 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 B., 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 $S_{min} = 249 \approx S_{max} = 251$.

B Implementation details and computational costs

We present here some additional implementation details and show the computational costs for training a Coded ResNeXt network with respect to the costs of training a conventional ResNeXt.

As shown in Fig. 1(a) and (b), each path/subNN of the ResNeXt and Coded ResNeXt blocks consists of three layers. Each of the first two layers are composed of a convolutional operation followed by a batch normalization (BN) [27], and a rectified linear unit (ReLU) [47]. For the ResNeXt block, in the last layer, after the convolutional operation the output of all paths/subNNs are aggregated by summation and then the BN and ReLU follow. Similarly, for Coded ResNeXt, there are a BN and a ReLU operation that come after aggregating the output of all subNNs. In Fig. 1(a) and (b) those two operations would be depicted between the two summations.

We run all our experiments on Google's Colab TPU-v2 ($N_w = 8$ cores with precision bfloat16). We used PyTorch's implementation of stochastic gradient descent with Nesterov momentum [49, 70] equal to 0.9. We used cosine scheduler [44] that decayed the learning rate until 10^{-5}.

For CIFAR dataset the batch size is picked relatively high to harness TPU speed; the batch size per core is set to $B_w = 64$ (i.e., effective 512). Training on CIFAR is performed for 300 epochs with initial learning rate 0.1 and weight decay [33] equal to $5 \cdot 10^{-4}$. For data augmentation we used RandAugment [10] with $(N_{aug}, M_{aug}) = (3, 4)$ for CIFAR-10 and $(1, 2)$ for CIFAR-100[5], which we applied after the standard pad-and-crop and horizontal flips with probability 0.5.

As mentioned in Section 4.1, in order to make a fair comparison with ResNeXt, on ImageNet [60] we follow the training process proposed by the timm library [1][6]. The epochs are 250 (first 5 as warmup [20] and last 10 cooling down), the effective batch size is 1536 (192 per core), the learning rate is 0.6, and weight decay is 10^{-4}. The input image is first randomly resized and cropped using the standard values of scale and ratio [71] and then horizontally flipped with probability equal to 0.5. RandAugment [10] follows with $N_{aug} = 2$ layers and magnitude $M_{aug} = 7$ (varied using an additive gaussian noise of a standard deviation equal to 0.5) and also random erasing augmentation [88] with probability 0.4 and 3 recounts. We diverge from the timm's proposed process only on the resolution of the input *training* images. We reduce the resolution to 160 (instead of 224) because on the TPU-v2 of Google COLAB the training would require more than three weeks. With the reduced resolution it required approximately 10 days. The final evaluation of the trained model on the validation set is done using resolution equal to 224.

As far as the computational costs are concerned, we focus on ImageNet since it is considerably more demanding in terms of computational resources than the CIFAR datasets, and also because for the implementation we use a library dedicated to ImageNet training [79]. 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 that our implementation may have on the performance.

We would like to clarify that in a ResNeXt block (or Coded ResNeXt block with ratio $r = 1$) the last convolutional layer of the block does not have 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 $r = 1$ and also coincides with the way ResNeXt is implemented in the original work [81].

The throughput and average RAM consumption of Table 4 have been measured on the second epoch, the reason being that "generally the first epoch is slow with Pytorch XLA" [79]. For ResNeXt-50 we measure the FLOPs using the library fvcore. For Coded ResNeXt we add to 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 in element-wise manner the tensor to the power of 2, second it takes the mean, and finally it performs

[5] Those values are chosen in [10] for Wide-ResNet-28-2 [83] which, out of all models presented in that work, seems most similar to ResNeXt-29.

[6] It is possible that using more recent guidelines like the ones proposed for ResNet in [3] would give even higher accuracy. However, in this case, we would not have available an already publicly reported accuracy for ResNeXt. Therefore, we prefer to follow the recipe of timm's library.

	GFlops	#Params	Throughput	RAM
ResNeXt-50 (32×4d)	2.196	$25.0 \cdot 10^6$	$378 \frac{\text{samples}}{\text{sec}}$	12.3GB
Coded ResNeXt-50 (32×4d)	2.269	$25.0 \cdot 10^6$	$375 \frac{\text{samples}}{\text{sec}}$	12.7GB

Table 4: Computational costs on ImageNet.

an element-wise division with the square root of the total mean energy. We see in Table 4 that for all metrics, Coded ResNeXt does not introduce any significant additional computational cost when trained on TPU.[7]

One epoch of Imagenet on Google COLAB TPU-v2 takes roughly 55 minutes. Due to the platform's constraints, we had to split the training in sessions of 24 hours (i.e., in each session around 25 epochs were executed[8]), save the checkpoint at the end of each session, and start the next session from the latest checkpoint stored. Due to lack of powerful resources, for ImageNet and with the training setup described above we tested one additional set of hyperparameters, which was $(\mu, p_{drop}) = (4, 0.1)$ (instead of $(2, 0.1)$). With $(4, 0.1)$ we observed better binary classifiers and early decoders, but lower accuracy. Specifically, the accuracy was 79.81% (instead of 80.24%). The accuracy of the early decoders were $(4.6\%, 22.0\%, 32.0\%, 37.3\%, 40.0\%, 43.8\%, 55.6\%, 72.5\%, 75.7\%)$ (instead of $(2.38\%, 8.12\%, 12.7\%, 12.8\%, 14.0\%, 17.0\%, 26.5\%, 65.1\%, 73.4\%)$). Using a simpler training procedure that did not require that long period of training (for 150 epochs, with $(\mu, p_{drop}) = (1, 0.1)$ and without random erasing augmentation) we tested also other combinations of coding schemes. In contrast to the proposed schemes which at stages (s3, s4) have ratios $(16/32, 8/32)$ and gave an accuracy of 78.1%, we tested also the following ones which led to a more than 1% decrease of accuracy: one where stages (s2, s3, s4) had ratios $(24/32, 16/32, 8/32)$ and a second one where in stages s3 we used for the first 3 blocks $r = 16/32$ and for the last 3 blocks $r = 8/32$, and for stage s4 $r = 4/32$.

C Using early decoders to improve confidence calibration

In Section 4.4 we showed that the way the coding schemes are designed brings additional useful properties. It is possible to stop the evaluation of the network at an intermediate block, measure the energies of the output of each subNN of that block and predict the class of the input image. Therefore, each block trained to comply with a coding scheme can be used as an "early decoder" and produce early predictions. However, the final network's predictions are more accurate. Nonetheless, even when the entire network is evaluated, the early predictions can still be of use. In that case, they can provide a confidence estimation on the correctness of the network's final prediction. As shown in Table 5, the more early decoders agree with the final prediction the higher are the chances that this prediction is correct. Therefore, Table 5 can be used to obtain confidence levels on the output of a network by verifying how many early decoders agree with the final network's prediction and checking the corresponding line in the table.

number of early decoders agreeing with final prediction	CIFAR-10	CIFAR-100	ImageNet
0	52.0%	48.7%	46.6%
1	67.7%	69.5%	70.5%
2	77.2%	77.2%	87.3%
3	83.2%	87.7%	90.7%
4	89.9%	90.7%	90.9%
5	92.1%	90.6%	91.1%
≥ 6	98.1%	95.8%	92.6%

Table 5: Accuracy of final prediction (%) given the number of early decoders giving the same prediction.

In this section, we will also show that the predictions of the early decoders can also be employed by confidence calibration methods as a source of extra features. Confidence calibration is the problem of predicting probability estimates that are representative of the true correctness likelihood [21]. The

[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.

[8] For the total required time it should be taken into account that around 45 minutes are needed to fetch the dataset from Google Drive to Google COLAB per session.

softmax operation, which is commonly used as the final operation of the neural network models for classification, provides for each class the likelihood that the input sample belongs to that class. However, it is found in [21] that recent state-of-the-art models provide overconfident predictions. Even when those models predict a wrong class, they erroneously estimate a high probability that their prediction is actually correct. Calibrating the probabilities associated to the predicted class allows to reflect the true correctness likelihood.

Assume a multi-class classification problem where the input $X \in \mathcal{X}$ and label $Y \in \{1, \cdots, K\}$ are random variables following some ground truth joint distribution $\pi(X, Y)$. Let $g : \mathcal{X} \mapsto \mathbb{R}^K$ be the neural network trained for this problem. For a sample $(x, y) \sim \pi$, it outputs the logits $z = g(x)$ and predicts that the correct label of x is $\hat{y} = \arg\max z$. Calibrating perfectly this neural network means finding a function $h : \mathbb{R}^K \mapsto [0, 1]$ such that

$$\mathbb{P}(\hat{Y} = Y | \hat{P} = p) = p, \tag{7}$$

with $\hat{P} = h(g(X)), \hat{Y} = \arg\max g(X)$ and $\forall p \in [0, 1]$. The above probability is taken over the joint distribution π. One example can help us understand Eq. (7). Assume that we found an h that is a perfect calibrator, which means it satisfies Eq. (7). Suppose that for $p = 0.8$, there are exactly 100 input samples $x \in X$ such that $\hat{p} = h(g(x)) = 0.2$. Then Eq. (7) says that the neural network $g(x)$ predicts the correct label, i.e. $\hat{y} = y$, for only 80 out of those 100 samples. In other words, the function h correctly estimates that out of all those samples that gave 0.8 likelihood to be correct, exactly 80% of them were indeed correctly labeled by the neural network $g(x)$.

Since in practice we have datasets with finite number of samples, we have to estimate the performance of the calibrating function h by approximating the probability in Eq. (7) [51, 46]. To do that, we break the interval $[0, 1]$ into M interval bins of equal size and let those bins be $I_m = (\frac{m-1}{M}, \frac{m}{M}], m \in \{1, \cdots M\}$. If D is a given dataset, then let $B_m = \{(x, y) \in D : h(g(x)) \in I_m\}$ be the subset of samples for which the calibrating function h gives that the predicted label is correct with a probability belonging to the interval I_m. An unbiased estimator of $\mathbb{P}(\hat{Y} = Y | \hat{P} \in I_m)$ is

$$\text{acc}(B_m) = \frac{1}{|B_m|} \sum_{(x,y) \in B_m} \mathbb{1}(\hat{y} = y) \tag{8}$$

with $\mathbb{1}$ being the indicator function. The average confidence within bin B_m is defined as

$$\text{conf}(B_m) = \frac{1}{|B_m|} \sum_{(x,y) \in B_m} \hat{p} \tag{9}$$

with $\hat{p} = h(g(x))$. The function $\text{acc}(\cdot)$ approximates the left-hand side of Eq. (7) while $\text{conf}(\cdot)$ does it for the right-hand side. Therefore, the closer those two values are the better the calibration function h is.

Two metrics used to evaluate the calibration are [46]:

- Expected Calibration Error (ECE), which approximates $\mathbb{E}[|\mathbb{P}(\hat{Y} = Y | \hat{P} = p) - p|]$ as

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{|B|} |\text{acc}(B_m) - \text{conf}(B_m)|,$$

- and Maximum Calibration Error (MCE), which approximates $\max_{p \in [0,1]} |\mathbb{P}(\hat{Y} = Y | \hat{P} = p) - p|$ as

$$\text{MCE} = \max_{m \in 1, \cdots, M\}} |\text{acc}(B_m) - \text{conf}(B_m)|.$$

ECE measures on average how well calibrated the function h is and MCE provides the maximum error of the predicted likelihoods. MCE can be valuable a metric in high-risk applications where good reliability guarantees should be provided even in the worst-case scenario.

Surprisingly, in [21] it was found that the calibration function that works the best is simply a softmax function with a single parameter T regulating the scale (also known as "temperature"). This is an extension of Platt scaling [55, 51] and the calibration function used is

$$h_T(z)_k = \frac{e^{z_k/T}}{\sum_{j=1}^{K} e^{z_j/T}}, k \in \{1, \cdots, K\} \tag{10}$$

	h_T	h_{aug}	h_T	h_{aug}
CIFAR-10	(0.70%, 24.35%)	(0.70%, **24.14**%)	(0.80%, 24.83%)	(0.80%, **23.64**%)
CIFAR-100	(1.75%, 10.28%)	(**1.71**%, **9.31**%)	(1.82%, 11.37%)	(**1.79**%,**10.44**%)
ImageNet	(**2.98**%, 8.32%)	(3.00%, **7.70**%)	(**3.01**%, 8.56%)	(3.05%, **8.10**%)

(a) Split set into: train set 25%, test set 75%. (b) train set 50%, test set 50%

Table 6: The performance in terms of (ECE(%), MCE(%)) of the two calibration methods using temperature scaling. The first one h_T is proposed in [21] and has only one parameter. The second h_{aug} has additionally one parameter per early decoder and takes into account the prediction of the early decoders.

where $z \in \mathbb{R}^K$ is a vector (representing the logits produced by the model, i.e. $z = g(x)$) and z_k is the k-th element of that vector. The value of the parameter T is chosen by minimizing the negative log-likelihood on a hold-out validation set [21]. The minimization is performed using gradient descent updating only the parameter T. We emphasize that since the parameters of the network are kept fixed, the network predictions are not changed. h_T calibrates the likelihood probabilities associated to the class predicted by the network.

We now describe a simple way to take into account the predictions of the early decoders. Given an input sample, suppose that $\hat{e}_i, i \in \{1, \cdots, N_{dec}\}$ is the prediction of the i-th early decoder, N_{dec} is the number of early decoders and $\hat{y}$ the final prediction. Our proposed architecture for CIFAR has $N_{dec} = 6$ and for ImageNet $N_{dec} = 9$. We intend to adjust the parameterized function h_T by adding another N_{dec} parameters, which we denote τ_i, that account for the predictions of the early decoders. Specifically the calibration function we employ is

$$h_{aug}(z)_k = \frac{e^{z_k/T_{aug}}}{\sum_{j=1}^{K} e^{z_j/T_{aug}}}, \text{ with } T_{aug} = T\left(1 + \sum_{i=1}^{N_{dec}} \mathbb{1}(\hat{e}_i = \hat{y})\tau_i\right) \text{ and } k \in \{1, \cdots, K\} \quad (11)$$

which has $N_{dec} + 1$ parameters. If the i-th early decoder agrees with the final prediction $\hat{y}$ then the temperature is adjusted by adding the term τ_i. The parameters are optimized in the same way as for h_T, using gradient descent with respect to the negative log-likelihood. We initialize all the τ_i to be zero and T to be one. The learning rate is 0.1 and we perform 600 iterations.

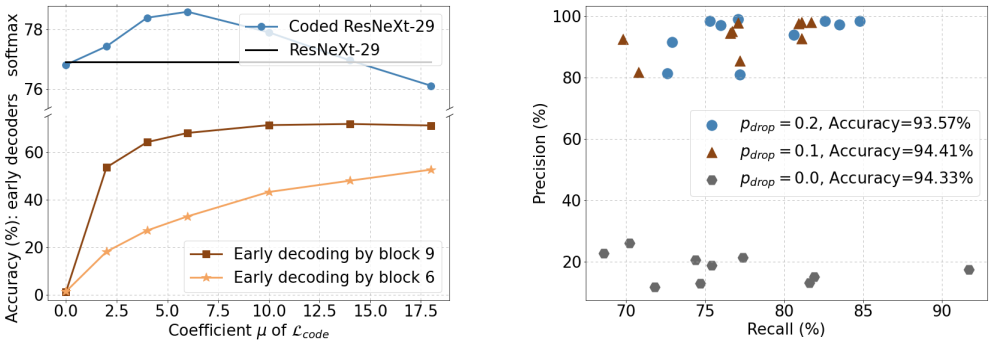
We start the experimental procedure by randomly splitting the validation set of each dataset into two sets. The first contains a quarter of the total samples and the second the rest. The splitting is performed in a way such that within the set all classes have approximately the same number of samples. We use the first set to train the parameters of h_T and h_{aug}. We test their performance on the second set by estimating the Expected Calibration Error (ECE) and the Maximum Calibration Error (MCE), using $M = 15$ bins as in [21]. We repeat the procedure another 40 times, each time with a different random split of the validation dataset. The results are shown in Table 6a. In Table 6b we conduct the same experiment but now we split the validation set into two sets of equal size. Across all datasets and for both splitting ratios we see that h_{aug} improves the MCE with almost no impact on ECE with respect to h_T.

D Ablation Study on Coding Loss and dropSubNN

In this section, we 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.

In Fig. 5a we show that increasing μ and so pushing more the energies of the subNNs to comply with the coding scheme, is at first 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, and the final accuracy of the whole network's predictions starts declining.

The second experiment concerns the dropSubNN and its hyperparameter p_{drop}. Dropping randomly some subNNs during training inhibits their "co-adaptation"[68], as they learn not to depend on the others and to perform well even in the absence of some of them. Figure 5b shows that the dropSubNN is essential for the binary classifiers. A small value of $p_{drop} = 0.1$ can greatly boost their performance



(a) Impact of coefficient μ of the coding loss on CIFAR-100.

(b) Impact of the probability p_{drop} on the performance of the binary classifiers.

Figure 5: Ablation study on the two hyperparameters introduced in this paper, i.e. the coefficient μ balancing the losses in Eq. (6) and the probability p_{drop} of dropping subNNs.

and even slightly improve the overall accuracy. Further increasing it to $p_{drop} = 0.2$ degrades the accuracy without improving much the binary classifiers.

E Limitations

A limitation regarding our proposed idea is that it cannot be applied to *any* type of multi-branch architecture. As we explain in Section 4.3.1 the method works for ResNeXt because the output of all branches of every block are conveniently aggregated by summation. This avoids having the blocks relying on the output of the inactive subNNs to learn the classification, which would then prevent the extraction of the binary classifiers by just keeping the active subNNs. Instead, since (i) the blocks receive the aggregated outputs of the previous block and (ii) the outputs of the inactive subNNs are pushed to zero by the coding loss, during training the blocks are pushed to solely rely on the aggregated signal provided by the active subNNs. Interestingly, there are other architectures that have a structure similar to that of ResNeXt and where the idea of our paper could also be applied. An example is MobileNetV2 [61], which uses a type of block called MBConv. To see how our idea could be applied to this kind of blocks, let us define the following neural network module in PyTorch:

Pytorch sequential code

Require: channels_in, channels_mid, channels_out, N, s

```
module = nn.Sequential(
    nn.Conv2d(channels_in, channels_mid, kernel_size=1),
    nn.BatchNorm2d(channels_mid),
    nn.ReLU(),
    nn.Conv2d(channels_mid, channels_mid, kernel_size=3, padding=1, groups=N, stride=s),
    nn.BatchNorm2d(channels_mid),
    nn.ReLU(),
    nn.Conv2d(channels_mid, channels_out, kernel_size=1),
    nn.BatchNorm2d(channels_out)
    )
```

A typical ResNeXt block is formed by simply adding a residual connection to the above module, i.e. module(x)+x where x is the input of the block.[9] This ResNeXt block has N branches/subNNs. The MBConv uses an almost identical module, with the slight differences that (i) the activation function ReLU6() is used instead of ReLU(), and (ii) channels_mid takes a larger value than channels_out, whereas in ResNeXT it takes a smaller one. Finally, it forces N = channels_mid so the second

[9] We assumed that channels channels_in = channels_out, and therefore there is no expansions of the number channels which usually happens in the beginning of each stage.

convolution is depth-wise. Overall, the MBConv can be seen as a type of ResNeXt block but with the number of subNNs equal to the channels_mid. We therefore expect a "Coded-MobileNetV2" to behave the same as Coded-ResNeXt and exhibit the same properties.

Another limitation of our work is given by the trade-off between the specialization of the subNNs and the performance of the entire neural network model. This relationship is depicted clearly in Fig. 5a. To force the subNNs to specialize in a specific set of classes we introduced the "coding loss" $\mathcal{L}_{code}$. The balance between this loss and the cross entropy one (i.e. $\mathcal{L}_{class}$) is regulated with the hyperparameter μ. As μ increases more emphasis is given on the subNNs activating according to the coding scheme. As shown in Fig. 5a, increasing μ at first helps the total performance (accuracy) of the entire network. However, after a certain point, increasing the specialization of the subNNs comes at the expense of the performance of the entire network.

We came across this trade-off also when designing the ratios r of the coding schemes. Ideally, we would like the ratios to be as small as possible. This would mean that each subNN would be assigned to specialize to smaller set of classes. Consequently, the information paths of the classes would be more disentangled, in the sense that the paths associated to two different classes would have less shared parameters. Moreover, the extracted binary classes would have even less parameters. Unfortunately, we could not decrease more the ratios without negatively impacting the total model's accuracy. For example, in one of our initial experiments (with fewer epochs) on ImageNet, we found that if we used the proposed ratios of $(32/32, 16/32, 8/32)$ in stages $(c3, c4, c5)$ we get an accuracy of 78.1%, but trying to increase the subNNs' specialization by setting the ratios to $(16/32, 8/32, 4/32)$ drops the accuracy to 76.6%.

Nonetheless, we are optimistic that there exist ways in which a more drastic specialization of the subNNs is achieved without compromising the performance of the entire network. One alternative that was the subject of some of our tests was, during training, zeroing the gradients[10] of the subNNs that according to the coding scheme should remain inactive. That way, the subNNs are updated only by gradients coming from samples of the subset of classes that the coding scheme had assigned to them. Those experiments were performed in a network whose architecture was identical to Coded ResNeXt with the sole difference that the blocks had no skip/residual connections. In that architecture, replacing after Energy Normalization the $\mathcal{L}_{code}$ with an operation that zeroes the gradients according to the coding schemes yielded excellent specialization without degrading the overall network's performance. Unfortunately, this idea did not work well when we added the skip connections. The accuracy of the network was significantly lower when the architecture had skip connections and the gradients of inactive subNNs were zeroed during training.

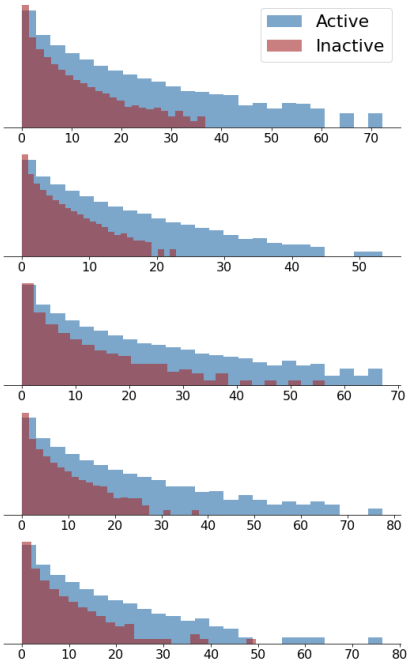
F Additional information for binary classifiers and activation of subNNs

In this section, we provide additional details regarding the binary classifiers that can be extracted after training a Coded ResNeXt, the activation distributions of the subNNs, and their specialization.

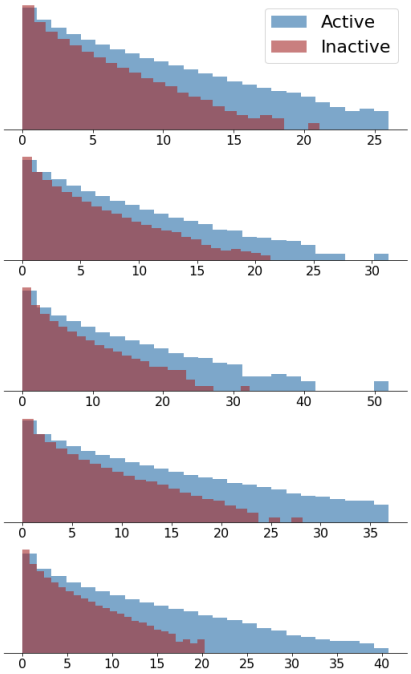
F.1 Activation distribution of subNNs

In this subsection we investigate the distribution of the output signal of the subNNs. A sample that passes through a subNN can either belong to the set of classes for which the subNN has to activate, or to the set for which it has to stay inactive. We would like to see how the subNNs respond in these two scenarios by plotting the distribution of the absolute value of the output in each case. In Fig. 6 we plot for CIFAR-10 and CIFAR-100 those distributions for the first 5 subNNs of the second block of stage s3. We show in red how those subNNs react when the sample belongs to the set of classes for which the subNN has to stay inactive according to the coding scheme, and with blue when they have to activate. Since the distributions were very skewed with many values around zero, we show the y-axis in log scale to better appreciate their shape in the complete range. We see that the distribution of the active subNNs has a bigger tail, which confirms that the subNNs output higher values when receiving a sample of their assigned classes.

[10] This can be implemented on PyTorch using the detach function.



(a) Distribution of the absolute value of the output of the first 5 subNNs from the second block of stage s3 on the architecture for CIFAR-10.



(b) Distribution of the absolute value of the output of the first 5 subNNs from the second block of stage s3 on the architecture for CIFAR-100.

Figure 6: Output distribution of active vs inactive subNNs

F.2 Additional details on Binary Classifiers

For convenience, we provide here 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.

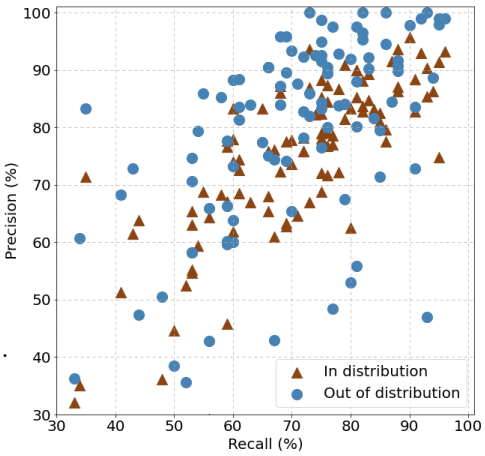
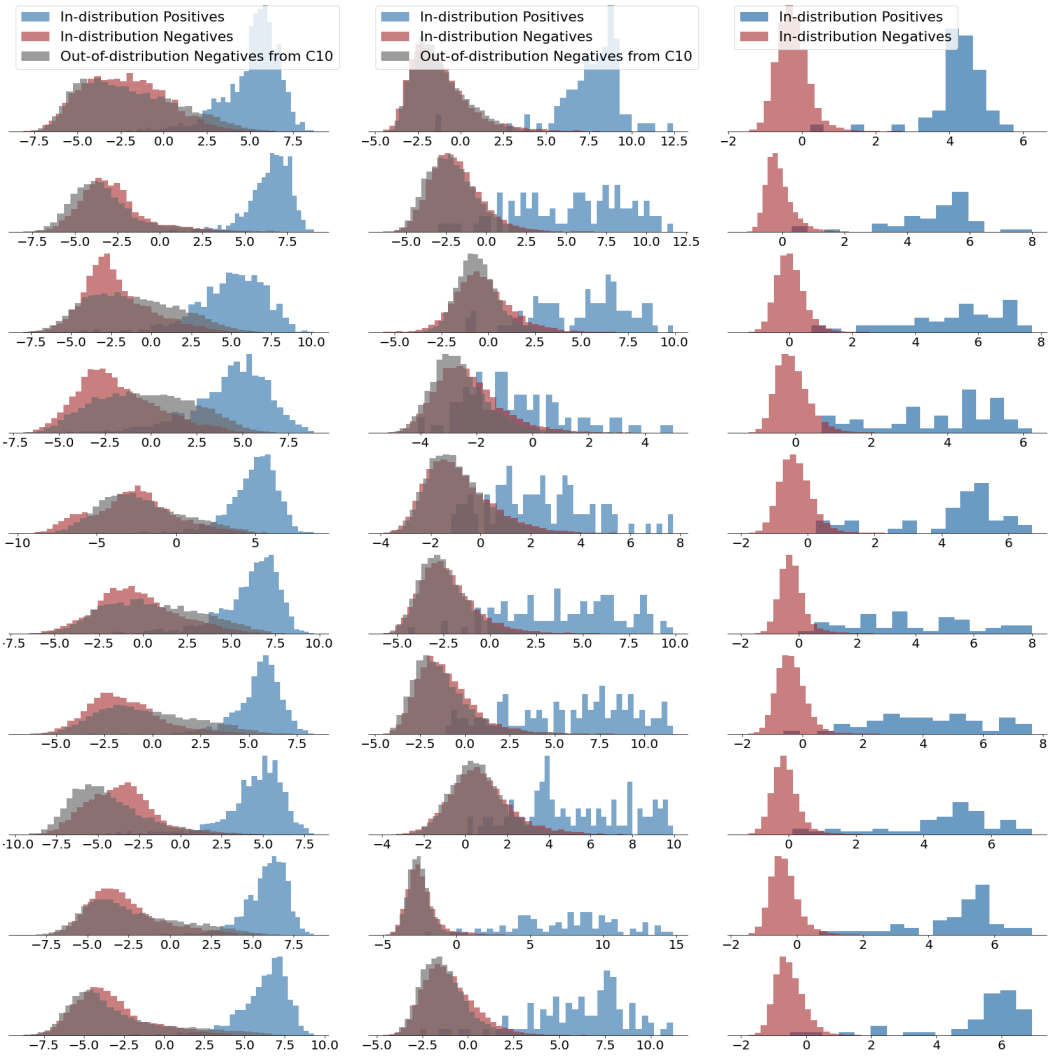


Figure 8: Precision-Recall on CIFAR-100.

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



(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

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.

Regarding plots Fig. 2b, Fig. 8 and Fig. 2c, which depict, respectively, the precision and recall achieved by the binary classifiers trained on CIFAR-10, CIFAR-100 and ImageNet, we would like to make some remarks.[11] 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

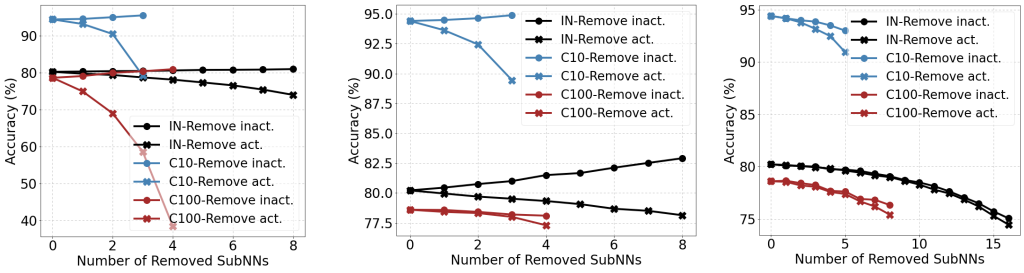
[11] In Fig. 8, which corresponds to CIFAR-100, we followed the same procedure as in Section 4.3 for ImageNet. Specifically, we randomly sample from the set of negatives a subset of size 9 times bigger than the size of positives. We use that subset of negatives to evaluate the precision and recall of the binary classifier.

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 $\frac{5000}{500} = 10$ times fewer positives but have "seen" negatives from $\frac{100-1}{10-1} = 11$ times more classes. The observation that CIFAR-100 binary classifiers perform significantly better than the ones of CIFAR-10 with out-of-distribution negatives, agrees with our intuition that a model should work better for out-of-distribution data if it has been trained with a high variety of classes where the number of samples per class is small than if it is trained with the same amount of samples but fewer classes and more examples per class.

F.3 More experiments on removing subNNs

In this subsection repeat the experiment described in Section 4.1 for three more blocks. We remind the reader that in that experiment we pick a block l from which we randomly remove subNNs 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. We see that the trend depicted in Fig. 2a, where the performance drops more when active subNNs are removed from a block than when the inactive are removed, is maintained also for the blocks in Fig. 9.



(a) Removing subNNs from the last block.

(b) Removing subNNs from the third to last block.

(c) Removing subNNs from the fifth to last block.

Figure 9: Performance when removing active versus inactive subNNs from a specific block.

However, we see that this trend becomes less visible the earlier is the block in which we perform the experiment. For example in Fig. 9c we see that removing either active or inactive subNNs leads to small differences. Removing active subNNs from the earlier blocks has much less negative effect than from the blocks deeper in the architecture. The subNNs in the early blocks produce low level features, which most of the times are useful in general for any class. It is challenging to force those subNNs to produce low level features that are exclusively useful to a specific subset of classes. This difficulty is more pronounced in this study, where we design the coding schemes without taking into account the semantics of each class. Stated differently, it is hard, given a set of classes and without looking at the semantic similarities between the classes, to split the set into two subsets and push subNNs to produce low level features that are useful only for one of those subsets and useless for the other. Nonetheless, we expect that increasing μ, which would push the subNNs towards higher specialization, would lead to having the performance degradation due to removing active subNNs observed in the latter blocks also in early blocks.