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 [33, 34] applied sequentially. Nonetheless, there are numerous proposals placing the convolutional layers in parallel, forming multi-branch designs. For instance, inception models [65, 66] 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 [19], as initially done in ResNeXt networks [75], and later refined using squeeze-excitation in [23], or a split-attention mechanism in [79]. A first question we ask in this work is: *what is the purpose of multi-branch architectures?*

Initially, in AlexNet [30] 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 [36], 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 architectures is mostly justified by achieving a higher accuracy

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

of the whole network. 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 [38, 51, 53, 81, 83]. Nevertheless, 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. Leveraging concepts from coding theory, we design the assignment of each branch to a specific set of classes without having to rely on the semantics of each class. Specifically, for each block in the network, a binary "codeword" of length equal to the number of branches of the block is matched to each class, indicating which branches will activate for that class. That way, keeping only the network's branches assigned to a class, it is possible to form a unique path for that class, which traverses the whole network and 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 [75] to which we add an architectural tweak.

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

Numerous attempts have tried to understand how complex deep neural networks actually work. For instance, activation maximization is used for finding the input that increases the activation of a neuron [11, 49, 47, 76]. Saliency maps [60, 59, 61, 63, 50, 54] find the pixels that have the largest influence on the model prediction. The pitfall of such approaches is that they cannot really explain how the network functions and they mainly serve as post hoc visualization methods [55]. There are many interpretable by design proposals [70, 80, 6, 45, 28]; however, most of them concentrate on enhancing interpretability only in the last layers of the neural network. In [70], the final linear layer is replaced with a differentiable decision tree, and in [80] a loss is used to make each filter of the very high-level convolutional layer represent a specific object part. In [6], the model's output is compared with learnt prototypes, whereas in [28] it represents concepts on which humans can intervene.

Another direction for enhancing interpretability is through disentanglement. Although there is not yet a generally accepted definition, disentanglement aims at separating out the main factors that are present in the data distribution [4, 21, 40]. 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 is common across many proposed architectures such as (variational) autoencoders [22, 5, 7, 27, 12], generative adversarial networks [8, 42, 26], normalizing flows [13, 58], or even architectures that aim to decompose the content to the style representation of an image [16, 14]. In contrast to prior art, our work approaches disentanglement as "the way information travels through the network" [56] and looks at the neural network architectures for classification in a more general way. Specifically, our goal is to control the paths through which information flows throughout the network by assigning each part of the network to specific subset of classes. We only allow a small fraction (6%) of the parameters at the beginning of the network to be shared across all classes. Related to our work is [72], in which 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.

Finally, there is a stream of work proposing to dynamically control the path that a sample follows through the network [35, 9, 2, 15, 37, 71, 68, 74, 39]. An extra network, named gated network or controller, is usually trained to choose which parts (blocks or filters) to be pruned/omitted. During inference this brings memory and speedup gains since the samples do not pass through the entire network. Despite being related to our work, there are some key differences. First, we do not use any

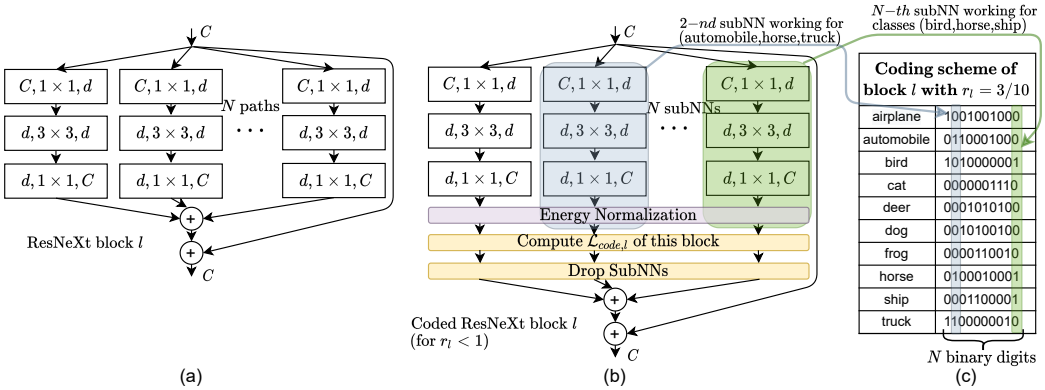


Figure 1: Building block of ResNeXt and our proposed variant. **(a)**: ResNeXt block. A layer is shown as (# in channels, kernel size, # out channels) and is composed of a convolutional operation, batch normalization (BN) [25], and ReLU [44]. **(b)**: Coded ResNeXt block. With light violet color we depict 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.

extra network that has to be trained to learn good paths. Second, in our work, the paths are defined in advance, prior to training (and thus not learnt). Most importantly, these works provide no guarantee that two samples with same semantics (for example of the same class) follow the same path, so it is not possible to extract any of those paths and use them as lightweight single purposed models.

3 Coded ResNeXt

3.1 The block

The typical ResNeXt block [75] 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 [75]). 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 [62], similar to techniques such as SpatialDropout [67], StochasticDepth [24], and DropPath [32]. 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 [62] 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 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 follow is that the deeper in the network a block is (i.e., the larger l is), the smaller the r_l being 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.

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

stage	**Coded ResNeXt-29 (10×11d)** for CIFAR-10	**Coded ResNeXt-29 (20×6d)** for CIFAR-100	**Coded ResNeXt-50 (32×4d)** for ImageNet
c1	conv $3\times3, 64$	conv $3\times3, 64$	conv $7\times7, 64$, str. 2, 3×3 max pool, str. 2
c2	$[256, 11, 10/10]\times3$	$[256, 6, 20/20] \times 3$	$[256, 4, 32/32]\times3$
c3	$[512, 22, \mathbf{5/10}]\times3$	$[512, 12, \mathbf{8/20}] \times 3$	$[512, 8, 32/32]\times4$
c4	$[1024, 44, \mathbf{3/10}]\times3$	$[1024, 24, \mathbf{4/20}] \times 3$	$[1024, 16, \mathbf{16/32}]\times6$
c5	global avg. pool, 10-d fc	global avg. pool, 100-d fc	$[2048, 32, \mathbf{16/32}]\times3$
			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 c2, c3, c4 have approximately 0.2, 0.9, 3.5 million parameters respectively (in total $4.7M$). For ImageNet, c2, c3, c4 and c5 have $0.2M$, $1.2M$, $7.0M$ and $14.5M$ respectively (in total $25.0M$).

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%.

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*. Although, for a given r_l, the coding scheme is designed in an agnostic way with respect to the nature of the classes, to ensure good performance we still have to design it smartly. For example, as stated in Section 3.1.2, we want to ensure that each class receives the same number of dedicated parameters and therefore of operating subNNs. This leads to the first rule:

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 sets assigned to the rest of the classes. This translates to:

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

Given r_l and N, many coding schemes that follow the above rules may exist. We have tested that any of those schemes give experimentally the same results. However, finding any of those schemes is a challenging NP-Hard problem. We present in Appendix A our algorithm for generating the coding schemes used here.

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 [75]. 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) [29], and ImageNet 2012 [57] 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 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 $r_l < 1$. The total loss used in order to train the network is

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

with μ being a constant balancing the two losses.

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 [57] we follow the training process proposed by timm library [1]. The epochs are 250 (first 5 as warmup [17] 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 [82] 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, which is the platform used for all our experiments, the training would require 1 month. Nonetheless, timm reports 79.77% accuracy for the ResNeXt-50, which is clearly smaller than the one of Coded ResNeXt-50 even though it is trained with reduced resolution. In Appendix B we describe the training process for CIFAR and give additional details.

	(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 $(80.21\%, 80.25\%, 80.26\%)$.

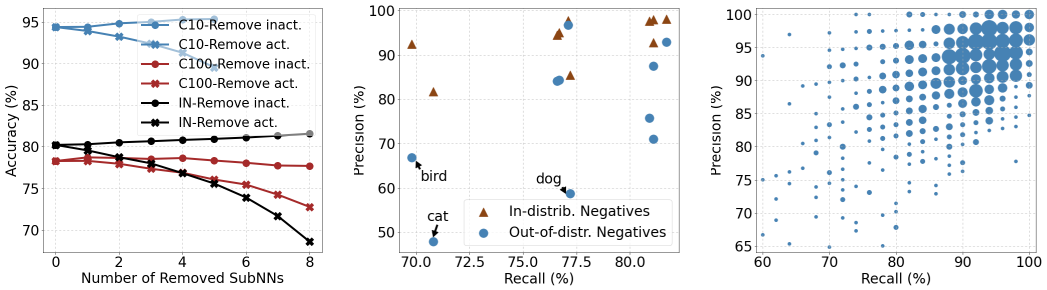
In Table 2 we compare the accuracy of Coded ResNeXt against the corresponding ResNeXt (i.e., when setting all ratios r_l equal to 1). We observe a clear improvement of the accuracy across all datasets. Surprisingly, forcing the subNNs to specialize to specific set of classes yields significant gains 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 proposed architecture actually succeeds in achieving specialization. Assuming a subNN is assigned to actively work 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.

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 c3 in the architecture for CIFAR (see Table 1) and the second of stage c5 for that for ImageNet. Figures with respect to other blocks are presented in Appendix F.

[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.



(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.

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.

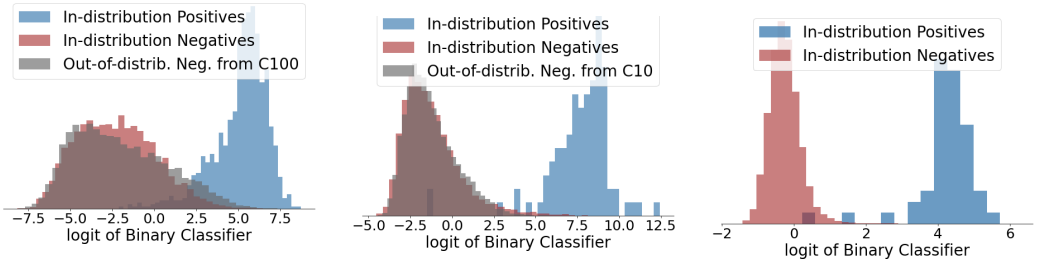


Figure 3: Output distribution of Binary Classifier (BC) of the first class (airplanes, apples, tench) of each dataset (CIFAR-10, CIFAR-100, ImageNet, respectively).

In Fig. 3 we pick the first class of CIFAR-10/100 and ImageNet ("airplane", "apple", and "tench", respectively) and remove all inactive subNNs for that class. 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 of the dataset (i.e., in-distribution negatives). Evidently, 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 predictions, since the sub-model has never been trained on such samples. Nevertheless, as Fig. 3 shows, the extracted BC model continues to perform well.

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 [69]. 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 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.

[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%.

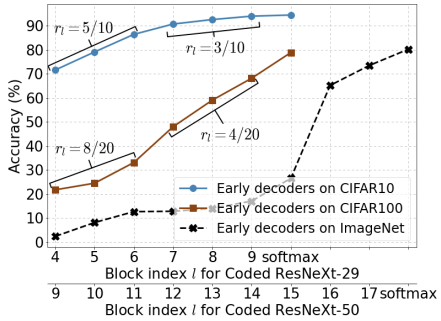


Figure 4: Accuracy of the early decoders.

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

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.

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:

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

with $|| \cdot ||_2$ being the L2 norm. This early decoding is enabled and reinforced by the rule C used to create the coding schemes (see Section 3.2.1). That rule forces the codewords to be as far apart from each other as possible. The farther apart the codewords are, the more subNNs should incorrectly be active or inactive for the early decoder to predict a wrong label. In Fig. 4 we depict the accuracy of every block $l \in B_{code}$ when functioning as 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, we illustrate another possible utility of the early decoders. In Table 3 we measure the accuracy of the network's final prediction given how many early decoders also provide the same prediction. It is obvious that the more decoders agree with the final prediction, the higher the probability to be correct. Therefore, early decoders provide a confidence estimation on the correctness of the network's prediction. Specifically, they can be a source of extra features used to improve state-of-the-art confidence calibration methods [18]. Due to space limitation, this is further analyzed 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 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 obtaining 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.

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

[12] 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.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

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

[21] 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.

[22] 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.

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

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

[25] 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.

[26] 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.

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

[28] 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.

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

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

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

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

[33] 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.

[34] 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.

[35] 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.

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

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

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

[39] 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.

[40] 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.

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

[42] 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.

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

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

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] 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.

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

[51] 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.

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

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

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

[62] 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.

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

[64] 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.

[65] 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.

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

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

[68] 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.

[69] 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.

[70] 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.

[71] 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.

[72] 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.

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

[74] 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.

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

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

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

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

[79] 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.

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

[81] 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.

[82] 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.

[83] 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]