BranchConnect: Image Categorization with Learned Branch Connections

Karim Ahmed Lorenzo Torresani
Department of Computer Science, Dartmouth College
karim@cs.dartmouth.edu, LT@dartmouth.edu

Abstract

We introduce an architecture for image categorization that enables the end-to-end learning of separate visual features for the different classes to distinguish. The proposed model consists of a deep CNN shaped like a tree. The stem of the tree includes a sequence of convolutional layers common to all classes. The stem then splits into multiple branches implementing parallel feature extractors, which are ultimately connected to the final classification layer via learned gated connections. These learned gates determine for each individual class the subset of features to use. Such a scheme naturally encourages the learning of a heterogeneous set of specialized features through the separate branches and it allows each class to use the subset of features that are optimal for its recognition. We show the generality of our proposed method by reshaping several popular CNNs from the literature into our proposed architecture. Our experiments on the CIFAR100, CIFAR10, ImageNet, and Synth datasets show that in each case our resulting model yields a substantial improvement in accuracy over the original CNN. Our empirical analysis also suggests that our scheme acts as a form of beneficial regularization improving generalization performance.

1. Introduction

In this paper we consider the problem of image categorization in scenarios involving a large number of classes. The recent introduction of several large-scale class-labeled image datasets [4, 14, 11, 27] has stimulated active research on this topic. Deep convolutional neural networks (CNNs) [15, 20, 6, 22, 7] have emerged as highly effective models for visual recognition tasks. One of the reasons behind the success of deep networks in this setting is that they enable the learning of features that are explicitly optimized to discriminate as well as possible the given classes.

It can be noted that the final fully-connected layer in a CNN can be viewed as implementing a separate linear classifier of the learned features for each of the classes to distinguish. But these class-specific classifiers operate all on the *same* feature representation, corresponding to the activations of the preceding layer. Under this model, the only obvious way to make the final classifiers more accurate is to either increase the number of features (e.g., by adding more neurons in the preceding layer) or make the features more expressive (e.g., by increasing the depth of the network). However, both these strategies increase the training cost and render the model more prone to overfitting. Furthermore, it can be argued that a single, jointly learned representation optimized on the overall problem of discriminating a large number of classes neglects to capture the uneven separability between classes, as some categories are inherently more difficult to distinguish than others or may require fine-grained distinction between similar classes.

To address these limitations, we propose a CNN architecture that makes it possible to learn "dedicated" or "specialized" features for each class but it still does so by means of a single optimization over all classes. This goal is realized through two mechanisms: a multi-branch network and learned gated connections between each class-specific classifier and the branches of the network. Let us consider each of these two mechanisms in turn. The multi-branch net has a tree-structure. It starts with a single stem corresponding to a sequence of convolutional layers common to all classes. The stem then splits into multiple branches implementing parallel feature extractors. Each branch computes a different representation, specialized for a subset of classes. In our scheme, the number of branches, M, is chosen to be smaller than the number of classes, C, (i.e., $M < C$) in order to parsimoniously share the features. Our model allows each class to use a subset of the M features. The selection of features for each class is learned as part of the optimization by means of gated connections linking the C neurons in the final fully-connected layer to the M branches. The gate of a class effectively controls the subset of features used to recognize that category. Because each class can leverage a different set of features, the representation can be specialized to distinguish fine-grained categories that would be difficult to tell apart on the basis of a single global representation for all classes. At the same time, because all M features originate from a common convolutional stem, the number of parameters in our model remains small compared to an ensemble of separate models. Furthermore, a single model allows us to obtain a diversified set of features by means of *one* optimization over the C classes. The optimization learns jointly the weights in the network as well as the gates defining the subset of features for each class.

In summary, this work provides several contributions:

- We present a multi-branch, gated architecture that enables the learning of different features for the different classes in a large-scale categorization problem.
- The learning is end-to-end and it is posed as optimization of a single learning objectives over all classes.
- The training runtime is comparable to that of learning a traditional single-column network with the same depth and number of parameters.
- We demonstrate the generality of our approach by reporting results on 4 different datasets. In each case, our adaptation of traditional CNNs into BranchConnect yields a substantial improvement in accuracy.
- We perform a sensitivity analysis of the effect of varying the number of branches and the number of branch features used by each class.
- We present an empirical study suggesting that our proposed scheme acts as a beneficial regularizer, yielding larger training error but improved test performance compared to a flat architecture of the same depth without branches and gated connections.

2. Related Work

Our approach bears closest similarity with deep learning methods that learn models or representations specialized for the different categories in a classification problem. Hinton et al. [8] achieve this goal by means of an ensemble of deep networks consisting of one full model trained to discriminate all classes (the generalist) and many specialists that improve the recognition of fine-grained classes confused by the full model. The approach of Farley et al. [24] explores a similar idea but with the difference that the specialists are trained on top of the low-level features already learned by the generalist, thus giving rise to a unified model. Yan et al. [25] and Ahmed et al. [1] propose a hierarchical decomposition of the categories where learned clusters of categories are first distinguished by a coarse-category classifier and then fine-grained classes within each cluster are discriminated by an expert. All these prior approaches [8, 24, 25, 1] differ from ours in the fact that they decompose the training into multiple stages, where first a full generalist is trained over all classes, and then separate expert networks or subnetworks are learned for different subsets of categories. Instead, our method learns class-specific representations by optimizing a single learning objective over all classes and does not require costly pretraining of a generalist. This enables fast training of our model, with computational cost equivalent to that of training a traditional single-column CNN of the same size. In the experimental section we compare our approach to HD-CNN [25] and Network of Experts [1] and demonstrate that, for the same depth and number of parameters, our model yields significantly higher accuracy. Unfortunately, we cannot directly compare with the methods described in [8, 24] as their models were trained on an internal Google dataset involving 100M labeled images. Neither the models, nor the softwares, nor the dataset have been publicly released.

Aljundi et al. [2] propose the use of autoencoders to learn a representation that is specialized for each different task in a lifelong-learning setting, where new tasks are added to the model sequentially giving rise to an ensemble of experts. This work also adopts a gating mechanism. But the gates are used to route the input example to the most suitable expert model. Instead, our approach is designed to operate in a scenario where all the data is fully available at the time when training is started, and it involves the use of gated connections to diversify the features used by different classes rather than to route the computation to separate models.

Our model also shares similarities with the mechanism of skipped connections popularized by fully convolutional networks [18] and used also in residual learning [7]. Skipped connections enable the additive fusion of activations produced by different layers in the network. However, skipped connections define a priori the layers to add together in the architecture. Instead, our approach learns the branch layers to combine via gated connections that are trained simultaneously with the weights of the network.

Our approach is also related to dropout [21], which has been show to act as a regularizer preventing overfitting. In our experiments we demonstrate that our BranchConnect also produces a regularization effect. However, while in dropout connections are randomly dropped during each training iteration, our method explicitly learns the connections that are most beneficial to eliminate and those that should be preserved. Stochastic pooling [26] is another regularization mechanism leveraging stochasticity during learning. Our gating function is closely related to stochastic pooling, as both rely on sampling a multinomial distribution during training. However, while in stochastic pooling the multinomial is defined from activations in a region, in our case the parameters of the multinomial are learned via backpropagation and do not change with the input. Furthermore, while stochastic pooling is used to reduce the dimensionality of activation volumes in deep CNNs, the gated connections of BranchConnect are designed to differentiate the features for each category in a classification problem.

Finally, our work falls in the broad genre of multi-branch

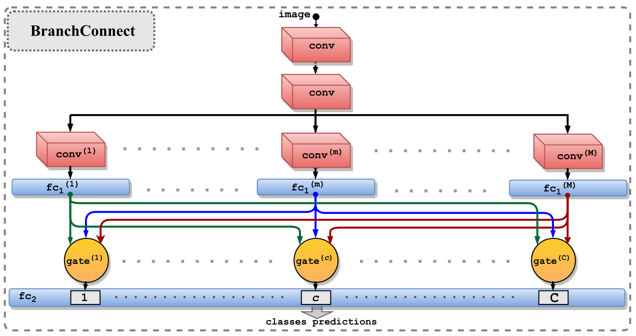


Figure 1: The architecture of BranchConnect for classification of C classes. The branches implement $M < C$ parallel feature extractors. Class-specific gates connect the M branches to the C classes in the last fully-connected layer.

networks, which have been adopted in previous work to address a wide array of tasks ranging from combining decision trees with convolutional networks [13], to autoencoding [9] as well as feature pooling [16].

3. Technical Approach

In this section we present our proposed technical approach. We begin by introducing the notation that will be used throughout this paper. We assume we are given a training dataset $\mathscr{D}$ of N class-labeled images: $\mathscr{D} = \{(x^1, y^1), \ldots, (x^N, y^N)\}$ where x^i represents the i-th RGB image and $y^i \in \{1, \ldots, C\}$ denotes its associated class label, with C indicating the total number of classes.

In subsection 3.1 we describe the architecture of BranchConnect. In subsection 3.2 and subsection 3.3 we discuss the training and inference procedures, respectively.

3.1. The architecture of BranchConnect

The architecture of BranchConnect is a tree-structured network, as illustrated in Figure 1. It consists of a *stem* that splits into $M < C$ branches, where M is a hyper-parameter that controls the complexity of our model. The stem consists of a sequence of convolutional layers possibly interleaved by pooling layers. Each branch contains one or more convolutional/pooling layers, followed by zero, one or more fully-connected layers (in our experiments we present results for a variety of models). The branches have identical architecture but different parameters. The BranchConnect model culminates into a fully-connected layer of C neurons using the softmax activation function to define a proper distribution over the C classes to discriminate. This last layer takes as input the activations from the M branches and it is equivalent in role to the last fully-connected layer of a traditional CNN for categorization. However, in BranchConnect each of these C neurons has a dedicated *branch gate* that controls the input effectively fed to the neuron. More specifically, let us consider the c-th neuron in the last fully-connected layer of C units. We refer to this neuron as the *neuron classifier* of class c, since it is responsible for computing the probability that the input image belongs to class c. The branch gate of this class is a *learned* binary vector $\mathbf{g}_c^b = \left[g_{c,1}^b, g_{c,2}^b, \ldots, g_{c,M}^b\right]^\top \in \{0,1\}^M$ specifying the branches taken into consideration by the neuron classifier to predict the probability of class c. If $g_{c,m}^b = 1$, then the activation volume produced by the m-th branch is fed as input to the neuron of class c. If $g_{c,m}^b = 0$, then the computation from the m-th branch is ignored by the classifier for class c. Thus, if we denote with E_m the output activation tensor computed by the last layer of the m-th branch, the input F_c to the c-th neuron will be given by the following equation:

$$F_c = \sum_{m=1}^{M} g_{c,m}^b \cdot E_m \tag{1}$$

The interpretation is that the branch gate $\mathbf{g}_c^b$ adds *selectively* the information from the M branches by choosing the branches that are most salient for the classification of class c. Under this scheme, each branch can therefore specialize to compute features that are relevant only to a subset of the classes. We also point out that depending on the constraints posed over $\mathbf{g}_c^b$, different interesting models can be realized. For example, by introducing the constraint that $\sum_m g_{c,m}^b = 1$, only one branch will be *active* for each neuron c (since $g_{c,m}^b$ must be either 0 or 1). Such a model would effectively partition the set of C classes into M disjoint clusters, where branch m is trained to discriminate among the classes in cluster m. It can be noted that at the other end of the spectrum, if we set $g_{c,m}^b = 1$ for all branches m and classes c, then all classifiers in the last layer would be operating on the same input. In our experiments we will demonstrate that the best results are achieved for a middle ground between these two extremes, i.e., by connecting each neuron classifier to exactly K branches where K is a cross-validated hyper-parameter such that $1 < K < M$. As discussed in the next section, the gate $\mathbf{g}_c^b$ of each class c is learned simultaneously with all the other weights in the network via backpropagation.

We point out that Eq. 1 uses *additive* selective fusion of the output produced by the branches. We have also tried stacking (rather than adding) the feature maps of the active branches but this increases by a multiplicative factor the parameters in the last layer and in our experiments this approach produced results inferior to the additive scheme.

3.2. Training BranchConnect

The training of our model is end-to-end and it is done by optimizing via backpropagation a given learning objective ℓ over the C classes of dataset $\mathscr{D}$. However, in the case of BranchConnect, the objective is optimized with respect to not only the weights of the network but also the branch gates, which are viewed as additional parameters in the model.

In BranchConnect, the weights of the convolutional and fully connected layers are real values, as in traditional CNNs. Instead, the branch gates are binary, which render optimization more challenging. To learn these binary parameters, we adopt a procedure inspired by the algorithm proposed in [3] to train neural networks with binary weights. During training we store and update a real-valued version $\mathbf{g}_c^r \in [0,1]^M$ of the branch gates, with entries clipped to lie in the continuous interval from 0 to 1.

In general, the training of a CNN consists of three steps: 1) forward propagation, 2) backward propagation, and 3) parameters update. We stochastically binarize the real-valued branch gates into binary-valued vectors $\mathbf{g}_c^b \in \{0,1\}^M$ only during the forward propagation and backward propagation (steps 1 and 2), whereas during the parameters update (step 3), the method updates the real-valued branch gates $\mathbf{g}_c^r$. The remaining weights of the convolutional and fully connected layers are optimized using standard backpropagation. In the next subsections we discuss in further detail the gate training procedure, under the assumption that at any time there can be only K active entries in the binary branch gate $\mathbf{g}_c^b$, where K is a predefined integer hyperparameter with $1 \leq K \leq M$. In other words, we impose the following constraints:

$$\sum_{m=1}^{M} g_{c,m}^b = K, \forall c \in \{1,\dots,C\}$$

$g_{c,m}^b \in \{0,1\}, \forall c \in \{1,\dots,C\}$ and $\forall m \in \{1,\dots,M\}$.

These constraints imply that each classifier neuron in the last layer receives input from exactly K branches. The entire training procedure for the branch gates is summarized in Algorithm 1 and discussed in detail below.

Branch Gates: Forward Propagation

During the forward propagation, our algorithm first normalizes the current M real-valued branch gates $g_{c,m}^r$ for each class c to sum up to 1. This is done so that $\text{Mult}(g_{c,1}^r, g_{c,2}^r, \dots, g_{c,M}^r)$ defines a proper multinomial distribution over the M branch connections of the c-th neuron classifier. Then, the binary branch gate $\mathbf{g}_c^b$ is stochastically generated by drawing K *distinct* samples $i_1, i_2, \dots, i_K \in \{1,\dots,M\}$ from the multinomial distribution over the branch connections. Then, the entries corresponding to the K samples are activated in the binary branch gate vector, i.e., $g_{c,i_k}^b \leftarrow 1$, for $k = 1, ..., K$. The input activation volume to the neuron classifier for each class c is then computed according to Eq. 1 from the sampled binary branch gates and the final prediction is obtained.

We have also experimented with a deterministic procedure that sets the active branch connections in $\mathbf{g}_c^b$ to correspond to the K largest values in $\mathbf{g}_c^r$. However, we found that this often causes the binary gate vector $\mathbf{g}_c^b$ to remain stuck at the initial configuration. As also reported in [3], we found the stochastic assignment of binary gates according to the real-valued probabilities to yield much better performance. In all our experiments we initialize the real-valued scalar branch gates $g_{c,m}^r$ to 0.5. This allows the training procedure to explore different connections in the first few iterations.

Branch Gates: Backward Propagation

In the backward propagation step, our method first computes the gradient of the mini-batch loss with respect to the input volume of each neuron classifier, i.e., $\frac{\partial \ell}{\partial F_c}$. Then, the gradient $\frac{\partial \ell}{\partial E_m}$ with respect to each branch output is obtained via back-propagation from $\frac{\partial \ell}{\partial F_c}$ and the current binary branch gates $g_{c,m}^b$.

Branch Gates: Parameters Update

As shown in Algorithm 1, in the parameter update step our algorithm computes the gradient with respect to the binary branch gates for each branch. Then, using these computed gradients and the given learning rate, it updates the real-valued branch gates via gradient descent. At this time we clip the updated real-valued branch gates to constrain them to remain within the valid interval $[0,1]$. The same clipping strategy was adopted for the binary weights in the work of Courbariaux et al. [3].

3.3. Inference with BranchConnect

In order to perform test-time inference on new samples given a trained model with real-valued branch gates $\mathbf{g}_c^r$, we adopt a deterministic strategy, rather than the stochastic approach used during training. We simply set to 1 the entries of $\mathbf{g}_c^b$ that correspond to the largest K values of $\mathbf{g}_c^r$, and leave all other entries set to 0.

We have also experimented with using the non-binary gates $\mathbf{g}_c^r$ for inference at test time but found this approach to yield much lower performance. This is understandable given that the learning objective is computed and minimized using binary rather real-valued gates.

4. Experiments

We demonstrate the effectiveness and the generality of our approach by presenting experiments using BranchConnect models built from different CNN architectures on four different datasets: CIFAR-100 [14], CIFAR-10 [14], ImageNet [4], and the Synthetic Word Dataset (Synth) [11, 10].

Algorithm 1 Training Branch Gates with BranchConnect.

```
Input: a minibatch of labeled examples $(x^i, y^i)$, $C$: number of classes, $M$: number of branches, $K$: the number of active branch connections per class, $\eta$: learning rate, $\ell$: the loss over the minibatch, $\mathbf{g}^r_c \in [0,1]^M$: real-valued branch gates from previous training iteration.
Output: updated $\mathbf{g}^r_c$, for all classes $c = 1, \ldots, C$
1. Forward Propagation:
for $c \leftarrow 1$ to $C$ do
    Normalize the real-valued branch gates of class $c$ to sum up to 1:
    $g^r_{c,m} \leftarrow \frac{g^r_{c,m}}{\sum_{m'=1}^{M} g^r_{c,m'}}$, for $m = 1, \ldots, M$
    Reset binary branch gates: $\mathbf{g}^b_c \leftarrow \mathbf{0}$
    Draw $K$ distinct samples from multinomial branch gate distribution:
    $i_1, i_2, \ldots, i_K \leftarrow \text{Mult}(g^r_{c,1}, g^r_{c,2}, \ldots, g^r_{c,M})$
    Set active binary branch gates based on drawn samples:
    $g^b_{c,i_k} \leftarrow 1$ for $k = 1, ..., K$
    Compute input $F_c$ to the $c$-th neuron, given branch activations $E_m$:
    $F_c \leftarrow \sum_{m=1}^{M} g^b_{c,m} \cdot E_m$
end for
2. Backward Propagation:
for $c \leftarrow 1$ to $C$ do
    Compute $\frac{\partial \ell}{\partial F_c}$ from $\ell$ and neuron classifier parameters
    Compute $\frac{\partial \ell}{\partial E_m}$ given $\frac{\partial \ell}{\partial F_c}$, $g^b_{c,m}$ for $m = 1, ..., M$
end for
2. Parameter Update:
for $c \leftarrow 1$ to $C$ do
    Compute $\frac{\partial \ell}{\partial g^b_{c,m}}$ given $\frac{\partial \ell}{\partial F_c}$ and $E_m$, for $m = 1, ..., M$
    $g^r_{c,m} \leftarrow \text{clip}(g^r_{c,m} - \eta \cdot \frac{\partial \ell}{\partial g^b_{c,m}})$ for $m = 1, ..., M$
end for
```

4.1. Reshaping a traditional CNN into a multi-branch net with BranchConnect

In order to show the benefits of BranchConnect we present results obtained by reshaping several traditional CNNs from the literature into the form of our multi-branch architecture. We refer to the original architectures as the *base* models. Note that our approach requires only the *specification* of the base CNN architecture, i.e., no pre-trained parameters are needed.

We evaluate a simple, single recipe to reshape each base model into an BranchConnect network. Let P_c be the total number of convolutional/pooling layers of the base model, and P_f the number of fully-connected layers following the convolutional/pooling layers. The stem of BranchConnect is formed by using P_c-1 convolutional/pooling layers identical in specifications to the first $P_c - 1$ layers of the base model. Then, we place in each branch the remaining convolutional layer of the base model followed by $P_f - 1$ fully connected layers identical in specifications to the first $P_f - 1$ fully connected layers of the base models (i.e, all fc layers except the last one). The M branches have identical architecture but distinct parameters. Then, we place a final fully connected layer of size C at the top. This layer is shared among all branches and is responsible for the final prediction. Neuron c in this layer is connected through learned gate $\mathbf{g}^b_c$ to the last layer of the M branches (see Figure 1).

4.2. CIFAR-100

CIFAR-100 is a dataset of 32x32 color images spanning $C = 100$ classes. The training set contains 50,000 examples and the test set includes 10,000 images. We use this dataset to conduct a comprehensive study using different network architectures and settings.

4.2.1 Accuracy Gain for Different Architectures

We begin by showing that BranchConnect yields consistent improvements irrespective of the specific architecture. To demonstrate this, we take five distinct architectures from prior work [7, 25, 1] and reshape them as BranchConnect.

The architectures are listed below (full specifications are listed in Appendix A). For this preliminary set of experiments we fix the number of BranchConnect branches M to 10. For each architecture, we train 10 separate models for values of K (the number of active branches per class) ranging from 1 to 10. As already discussed in section 3, we build the BranchConnect network from each base model by placing all convolutional layers except the last one in the stem. Each branch then contains one convolutional layer (identical in specifications to the last convolutional layer of the base model) followed by fully-connected layers (identical to those in the base model), except for the last one. The last fully-connected layer is shared among all branches. (see Figure 1).

Here are the five base models for this experiment:
1) *AlexNet-Quick.* This is a slightly modified version of the AlexNet model [15] adapted by Ahmed et al [1] to work on the 32x32 images of CIFAR-100. It consists of 3 convolutional layers and 2 fully-connected layers. Thus, our BranchConnect net constructed from *AlexNet-Quick* includes two convolutional layers in the stem, while each branch contains one convolutional layer with the same specification as the third convolutional layer in the base model and one fully-connected layer.
2) *AlexNet-Full.* This model is also taken from [1]. This base CNN is slightly different from *AlexNet-Quick* as it has only one fully-connected layer instead of two layers, and it uses local response normalization layers. The accuracy of this base model is higher than *AlexNet-Quick*. The corresponding BranchConnect model consists of a stem that contains the first two convolutional layers. Each branch consists of only one convolutional layer with the same specification as the third convolutional layer in the base model.
3) *NIN.* This model is a "Network In Network" (NIN) [17]. NIN models do not use fully-connected layers. Instead, they employ as final layer a convolutional layer where the number of filters is equal to the number of classes (C). The final prediction is obtained by performing global average pooling over the feature maps of this layer. Thus, we build BranchConnect by placing in the stem all convolutional layers, ex-

Table 1: Classification accuracy (%) (single crop) on CIFAR-100 for 5 base architectures. BranchConnect uses $M = 10$ branches. G:K/M means that each gate has K active connections. We report performance for $K = 1$ and when choosing the best value of K.[1]

	Method	depth	#params	Accuracy
AlexNet-Quick	Base Model V1	5	0.15M	44.3
	Base Model V2	5	1.20M	40.26
	NofE [1]	6	1.27M	49.09
	BranchConnect G:1/10	5	1.20M	**53.28**
	BranchConnect G:5/10	5	1.20M	**54.62**
Alexnet-Full	Base Model V1	4	0.18M	54.04
	Base Model V2	4	0.64M	50.42
	NofE [1]	5	1.12M	56.24
	BranchConnect G:1/10	4	0.64M	**57.34**
	BranchConnect G:6/10	4	0.64M	**60.27**
NIN [17]	Base Model V1	9	1.38M	64.73
	Base Model V2	9	1.61M	65.24
	HD-CNN [25]	n/a	n/a	65.64
	NofE [1]	11	4.66M	65.91
	BranchConnect G:1/10	9	1.61M	**66.10**
	BranchConnect G:5/10	9	1.61M	**66.45**
ResNet56 [7]	Base Model V1	56	0.86M	69.66
	Base Model V2	56	1.47M	70.72
	BranchConnect G:1/10	56	1.47M	**71.24**
	BranchConnect G:5/10	56	1.47M	**71.98**
ResNet56-4X [1]	Base Model V1 [1]	56	13.6M	72.23
	Base Model V2	56	25.4M	73.12
	NofE [1]	58	25.5M	74.71
	BranchConnect G:1/10	56	25.4M	**75.55**
	BranchConnect G:5/10	56	25.4M	**75.72**

cept the last two. Each branch then contains only one convolutional layer. Finally the branches are connected via our binary gates to the final convolutional layer of C filters followed by global average pooling for the final prediction.
4) *ResNet56*. This is the 56-layer residual network originally described in [7]. In the BranchConnect model, the stem contains all the residual blocks except the last block which is included in the branches. The final fully-connected layer is shared among all branches.
5) *ResNet56-4X*. This model is identical in structure to *ResNet56* but it uses 4 times as many filters in each convolutional layer and was shown in [1] to yield higher accuracy on CIFAR-100.

The results achieved with these 5 architectures are shown in Table 5. For each architecture, we report the accuracy of the base model *"Base Model V1"* as well as that obtained with our BranchConnect. We report two numbers for BranchConnect: the first using only *one* active branch per class (i.e., $K = 1$), the second obtained by choosing the best value of K (ranging from 1 to 10) for each architecture. We also include for comparison the performance achieved by Network of Experts (NofE) [1], which also builds a branched architecture from the given base model. However, NofE does so by performing hierarchical decomposition of the classes using two separate training stages and it connects each class to only one branch by construction. For the case of NIN, we also include the performance reported in [25] for the hierarchical HD-CNN built from this base model.

We can see from Table 5 that BranchConnect outperforms the base model *"Base Model V1"* for all five architectures. BranchConnect does also considerably better than NofE [1] and HD-CNN [25], which are the most closely related approaches to our own. For all architecture the peak performance of BranchConnect is achieved when setting the number of active branches (K) to be greater than 1 (the best accuracy is achieved with $K = 6$ for *AlexNet-Full* and with $K = 5$ for the other four models).

It can be noted that BranchConnect involves more parameters than the base models *"Base Model V1"*. Thus, one could argue that the improved performance of BranchConnect is merely the result of a larger learning capacity. To disprove this hypothesis we report the results for another version of the base models named *"Base Model V2"*. These base models were built by increasing uniformly the number of filters and the number of units in the convolutional layers and the first fully connected layer of *"Base Model V1"* in order to match exactly the total number of parameters in the BranchConnect models. BranchConnect with the same number of parameters and overall depth achieves much better accuracy than *"Base Model V2"*. In subsection 4.2.3 we will show that this is due to a regularization effect induced by our architecture. In Appendix A we also report results obtained by shrinking the numbers of parameters in BranchConnect to match those in the original *"Base Models V1"*. Even in this scenario, BranchConnect consistently outperforms the base models.

4.2.2 Varying the number of branches (M) and the number of active connections (K)

In this subsection we study the effect of varying the number of branches (M) in addition to the number of active connections (K). Due to lack of space, here we report results only using the *AlexNet-Quick* base architecture but we found the overall trend on this base model to generalize also to other architectures. Figure 2 shows the accuracy achieved by BranchConnect for $M = 10$, $M = 20$ and $M = 30$ branches. For each of these 3 architectures, we retrained our model using a varying number of active connections (K) ranging from 1 to M. It can be seen that best

[1]Note that some of the accuracies for NofE and Base Models listed here differ slightly from those reported in [1]. The differences are merely due to the fact that in [1] some of the architectures were tested using multiple image crops, while our evaluation uses a single crop for all architectures.

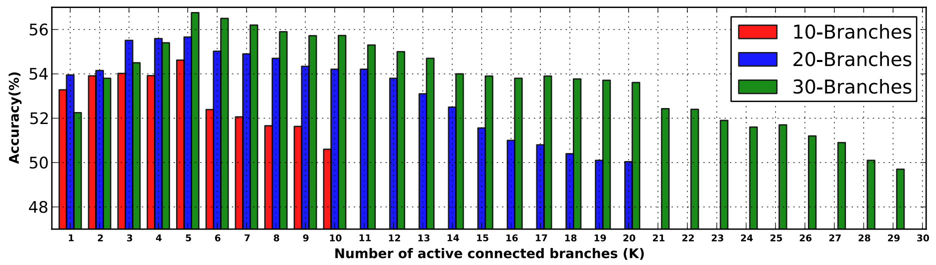


Figure 2: Accuracy of BranchConnect using different number of branches ($M = \{10, 20, 30\}$) and number of active connections (K) on CIFAR-100. Models were built from *AlexNet-Quick*.

performance is achieved with the model having $M = 30$ branches and that the model with $M = 20$ branches does better than that using $M = 10$ branches. This makes intuitive sense, as increasing the number of branches allows our model to further diversify the features used by the different classes. However, we can notice that in all three cases (10, 20, 30 branches), the peak accuracy is obtained at or around $K = 5$. Moving away from this peak point, the accuracy drops nearly monotonically. When the number of active connections K is near or equal to the maximum value (M), the models perform poorly compared to the case where $K = 5$ branches are connected. The case where only one branch is connected ($K = 1$) also gives inferior results.

4.2.3 BranchConnect acts as a regularizer

We observed in Table 5 BranchConnect nets achieve higher accuracy than "Base Models V2" despite having the same depths and numbers of parameters. We hypothesize that this happens because of a regularization effect induced by the branched structure of our network. While a single-column CNN performs feature extraction "horizontally" through the layers, BranchConnect spreads some of the feature computation "vertically" through the parallel branches. This implies that although BranchConnect has the same number of parameters as "Base Model V2," the number of feature maps extracted through each horizontal path from the input image to the end of a branch gate is smaller. This suggests that the branch features should be less prone to overfit. We confirm this hypothesis through two experiments.

In Figure 5 we plot the training loss versus the test loss of four different models while varying the number of training iterations. The four models were trained on CIFAR-100 and built from *AlexNet-Quick* (Appendix A includes the plot for *AlexNet-Full*). The four models are: 1) "Base Model V1", 2) "Base Model V2", 3) our BranchConnect using $K = 1$ active connections out of $M = 10$ branches, 4) a new model (*"Random Connect"*) with $M = 10$ branches where each class was permanently connected to only $K = 1$ branch chosen at random from the given 10 branches. For the same training loss value, BranchConnect systematically yields lower test loss value than the base model and "Random Connect." Also, upon convergence BranchConnect achieves lower test loss but higher training loss compare to the other two models. These results support a *regularization* explanation for the effect of BranchConnect. Furthermore, the overall poor test accuracy achieved by "Random Connect" emphasizes the importance of learning the active connections rather than hardcoding them a priori.

In a second experiment, we study the effect of increasing network depth on test accuracy. In the case of single-column CNNs, if the net has already enough learning capacity, further increasing its depth makes it prone to poor local minima [5]. But if BranchConnect acts as a regularizer, it should be more resilient to increasing depths. This is confirmed by the box plots in Figure 4, which were obtained by training both the base model (*AlexNet-Quick* "Base Model V1") and the corresponding BranchConnect model (using $M = 10$ and $K = 5$) starting from 7 different initializations. The figure shows the resulting distribution of test accuracy values for each model when the depth is increased up to 4 layers (top and bottom quartiles in box, maximum and minimum values above and below the box). For the base model, the accuracy goes down sharply as the depth is increased. It should also be noted that we were unable to train a base model with 4 additional layers. On the other hand, we can notice that the BranchConnect models are much more robust to the increase in depth. We were also able to train with good performance a BranchConnect model with 4 extra layers. Furthermore the variance in test accuracy is consistently lower for the case of BranchConnect for all depths.

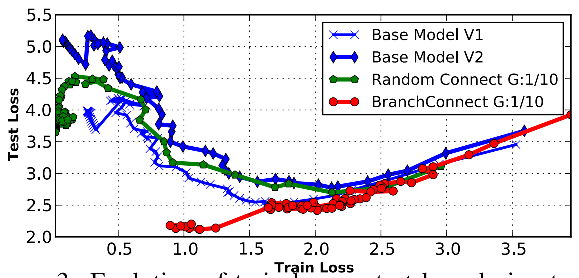


Figure 3: Evolution of train loss vs test loss during training on CIFAR-100. The training trajectory is from right to left. BranchConnect yields lower test loss for the same train loss compared to the base models and a net with randomly-chosen active connections.

Table 2: Classification accuracy (%) on CIFAR-10 dataset. G:K/M denotes K active connections out of a total of M.

Architecture	Method	Accuracy
AlexNet-Quick	Base Model	76.86
	BranchConnect G:3/5	**82.84**
AlexNet-Full	Base Model	82.78
	BranchConnect G:3/5	**85.00**
ResNet-56 [7]	Base Model	92.04
	BranchConnect G:3/5	**92.46**

Table 3: Top-1 single crop validation accuracy (%) on ImageNet. G:K/M denotes K active connections out of M.

Architecture	Method	Accuracy
AlexNet [4]	Base Model	58.71
	NofE [1]	61.29
	BranchConnect G:5/10	**63.49**
ResNet50 [7]	Base Model [7]	76.15
	BranchConnect G:5/10	**77.39**
	BranchConnect G:8/15	**77.68**
ResNet101 [7]	Base Model [7]	77.37
	BranchConnect G:5/10	**78.19**

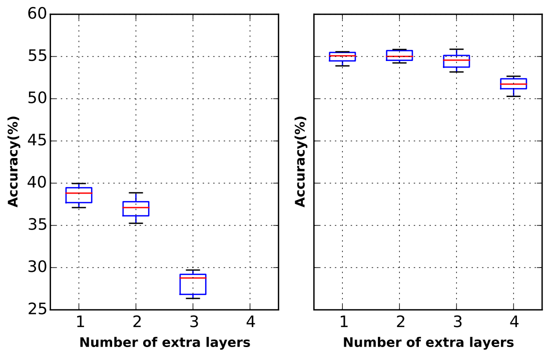


Figure 4: The effect of increasing the network depth on test accuracy (using CIFAR-100). Boxes show the distribution of test accuracy values of the same model trained using different initialization seeds. Left: base model (*AlexNet-Quick*). Right: BranchConnect. Our model is less sensitive to variations in the random initialization and yields stable performance even for increased depth.

4.3. CIFAR-10

In this subsection we test our approach on CIFAR-10, which includes 10 classes. This dataset is divided into 50,000 images for training, and 10,000 images for testing. Table 2 shows the classification accuracy of our BranchConnect models based on three different base architectures.

4.4. ImageNet

We evaluate our approach on the ImageNet 2012 large-scale classification dataset [4], which includes images of 1000 classes. The training set contains 1.28M images. We use the validation set which consist of 50K images to evaluate the performance. In Table 3, we report the Top-1 accuracies of different models.

4.5. Synth dataset

Finally, we evaluate our approach on a text recognition task using the Synth dataset [10]. The dataset contains a total of 9M images of size 32x100. Each image contains a word drawn from a 90K dictionary. The dataset is divided into 900K images for testing, 900K images for validation, and the remaining of the images are used for training. The recognition task is to classify each of the 900K testing images into one of the 90K words (i.e., $C = 90K$). The very large number of classes renders this dataset an interesting benchmark to test our BranchConnect approach. Our branched models are built from the base architecture "DICT+2-90K" used by Jaderberg et al. [11]. This base architecture has 5 convolutional layers and 3 fully-connected layers. Due to the large number of the classes, the training of these models was performed by adding the classes incrementally as described in [11]. In Table 4 we show the test accuracy of the base architecture and the BranchConnect models. Additionally, we show the results of the models learned from the Synth dataset when tested on other smaller datasets: IC03 [19], SVT [23], and IC13 [12].

Table 4: Word recognition accuracy (%) for models trained on Synth, using DICT+2-90k [11] as base model.

Model	Test Dataset			
	Synth [10]	IC03 [19]	SVT [23]	IC13 [12]
Base Model [10]	95.2	93.1	80.7	90.8
BranchConnect G:7/10	95.6	93.7	83.4	92.1

5. Conclusions

In this paper we presented BranchConnect—a multi-branch, gated architecture that enables the learning of separate features for each class in large-scale classification problems. The training of our approach is end-to-end and it is posed as a single optimization that simultaneously learns the network weights and the branch connections for each class. We demonstrated the benefits of our method by adapting several popular CNNs into the form of BranchConnect. We also provided empirical analysis suggesting that BranchConnect induces a beneficial form of regularization, reducing overfitting and improving generalization.

Future work will focus on more sophisticated combination schemes. The learned gates in our model can be viewed as performing a rudimentary form of architecture learning, limited to the last layer. We plan to study the applicability of this mechanism for more general forms of model learning.

References

[1] K. Ahmed, M. H. Baig, and L. Torresani. Network of experts for large-scale image categorization. In *ECCV 2016*, 2016. 2, 5, 6, 8, 11

[2] R. Aljundi, P. Chakravarty, and T. Tuytelaars. Expert gate: Lifelong learning with a network of experts. *CoRR*, abs/1611.06194, 2016. 2

[3] M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 3123–3131. Curran Associates, Inc., 2015. 4

[4] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pages 248–255, 2009. 1, 4, 8

[5] D. Erhan, Y. Bengio, A. C. Courville, P. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11:625–660, 2010. 7

[6] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034, 2015. 1, 15, 17, 18

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, 2016. 1, 2, 5, 6, 8, 11

[8] G. E. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *CoRR*, abs/1503.02531, 2015. 2

[9] O. Irsoy and E. Alpaydin. Autoencoder trees. In *Proceedings of The 7th Asian Conference on Machine Learning, ACML 2015, Hong Kong, November 20-22, 2015.*, pages 378–390, 2015. 3

[10] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Reading text in the wild with convolutional neural networks. *arXiv preprint arXiv:1412.1842*, 2014. 4, 8

[11] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman. Synthetic data and artificial neural networks for natural scene text recognition. *arXiv preprint arXiv:1406.2227*, 2014. 1, 4, 8

[12] D. Karatzas, F. Shafait, S. Uchida, M. Iwamura, L. G. i Bigorda, S. R. Mestre, J. Mas, D. F. Mota, J. Almazán, and L. de las Heras. ICDAR 2013 robust reading competition. In *2013 12th International Conference on Document Analysis and Recognition, Washington, DC, USA, August 25-28, 2013*, pages 1484–1493, 2013. 8

[13] P. Kontschieder, M. Fiterau, A. Criminisi, and S. R. Bulò. Deep neural decision forests. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 4190–4194, 2016. 3

[14] A. Krizhesvsky. Learning multiple layers of features from tiny images, 2009. Technical Report https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf. 1, 4

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, pages 1106–1114, 2012. 1, 5

[16] C. Lee, P. W. Gallagher, and Z. Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pages 464–472, 2016. 3

[17] Lin, Min, Q. Chen, and S. Yan. Network in network. In *International Conference on Learning Representations, 2014 (arXiv:1409.1556).*, 2014. 5, 6, 11

[18] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 3431–3440, 2015. 2

[19] S. M. Lucas, A. Panaretos, L. Sosa, A. Tang, S. Wong, and R. Young. ICDAR 2003 robust reading competitions. In *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*, pages 682–687, 2003. 8

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1

[21] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 2

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*, pages 1–9, 2015. 1

[23] K. Wang, B. Babenko, and S. J. Belongie. End-to-end scene text recognition. In *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 1457–1464, 2011. 8

[24] D. Warde-Farley, A. Rabinovich, and D. Anguelov. Self-informed neural network structure learning. *CoRR*, abs/1412.6563, 2014. 2

[25] Z. Yan, H. Zhang, R. Piramuthu, V. Jagadeesh, D. DeCoste, W. Di, and Y. Yu. HD-CNN: hierarchical deep convolutional neural networks for large scale visual recognition. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 2740–2748, 2015. 2, 5, 6

[26] M. D. Zeiler and R. Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013. 2

[27] B. Zhou, À. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 487–495, 2014. 1

A. Appendix

This supplementary material is organized as follows: in subsection A.1, we show additional experiments on CIFAR-100; in subsection A.2 we show additional plots that further support the interpretation of BranchConnect as a regularizer; in subsection A.3 we provide the specifications of the networks used in the paper including the details of the training process and data preprocessing.

A.1. Additional Experiments on CIFAR-100

In Table 5 we report additional experiments on the CIFAR-100 dataset. In the main paper we presented results of BranchConnect networks having the same number of parameters as *"Base Models V2"* models, which are larger capacity versions of *"Base Models V1"*. Here we report results of additional BranchConnect models (*"BranchConnect V1"*) that are obtained by shrinking the numbers of parameters to match those in the original *"Base Models V1"*. Once again, we see that BranchConnect networks consistently outperform the base models of the same capacity, both in the case of *"Base Models V1"* as well as *"Base Models V2"*.

A.2. Regularization Effect Induced by BranchConnect

In the main paper we provided a figure that shows the training loss versus the test loss of different models that were trained on CIFAR-100 and built from *AlexNet-Quick* architecture (Figure 3 in the paper). Here we show the training loss versus the test loss of different models that were built from other architectures.

In Figure 5a we show the training loss versus the test loss for three different models that were trained on CIFAR-100 and built from the *AlexNet-Full* architecture: 1) "Base Model V1", 2) "Base Model V2", 3) our BranchConnect using $K = 1$ active connection out of $M = 10$ branches. In Figure 5b we show the training loss versus the test loss for two models that were trained on CIFAR-100 and built from the *NIN* architecture: 1) the BranchConnect model using $K = 1$ active connection out of $M = 10$ branches, 2) "Base Model V2" which has the same number of parameters as the BranchConnect network, In all figures, we notice that for the same training loss value, our BranchConnect yields lower test loss value than the base models. Additionally, we notice that upon convergence BranchConnect achieves lower test loss but higher training loss compared to the other models. These results support a *regularization* explanation for the effect of BranchConnect.

A.3. Network Specifications

In this subsection, we provide the network specification and the learning policy of the models used in our experiments on CIFAR100, CIFAR10, ImageNet and Synth datasets. Tables 6, 7, 8, 9, and 10 list the specifications of the original base models ("Base Model V1" in the paper) and the BranchConnect models built from the five architectures that were used to train CIFAR100 dataset: AlexNet-Quick, AlexNet-Full, NIN, Resnet56, and Resnet56-4X (classification accuracy of different models are given in Table 1 of the paper).

For training on CIFAR10, we used the same architectures presented in Tables 6, 7, and 10 except that the number of output units in the last fully-connected layer was set to 10 instead of 100 (classification accuracy of different models are given in Table 2 of the paper).

For training on ImageNet, we used the network specification and the learning policy as listed in Tables 11, 12, and 13.

In Table 14 we show the the specifications of the base model and the corresponding BranchConnect model used on the Synth dataset.

Each table lists the specification in two columns starting from top (input) to down (output). The first column shows the architecture and the training policy of the base model, while the second column shows the architecture and the training policy of the corresponding BranchConnect model. For all of the BranchConnect models, we show the layers included in the stem of BranchConnect, and the layers included in the branches. We used rectified linear units (ReLU) in all architectures. In the second part of each table, we show the input data preprocessing done before training each model. The third part shows the learning policy for each model.

Notation used to describe layers:

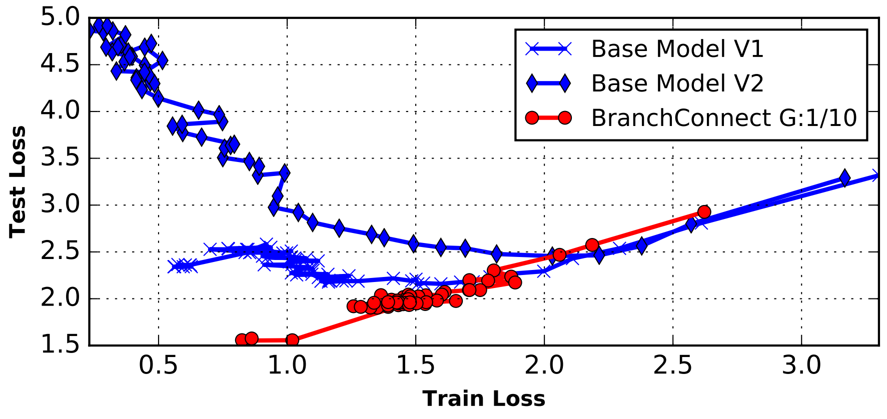
- **Convolutional layer:**
  CONV: <filter width> × <filter height>, <number of filters>
  Example: CONV: 5×5,32 means a convolutional layer with 32 filters of size 5×5.
- **Pooling layer:**
  POOL: <filter width> × <filter height>,<type>,<stride>
  Example: POOL: 3×3,Ave,2 denotes an average pooling layer of filter size 3×3 and stride 2. Two types are used: "Max" means maximum pooling, and "Ave" means Average pooling.
- **LRN:** Local Response Normalization layer.
- **Fully Connected layer:**
  FC: <number of output units>
  Example: FC:100 indicates a fully connected layer with 100 output units.
- **Fully Connected layer with Gates:**
  FC_Gates: <number of output units>
  As described in the paper, this type of fully-connected layer includes gates and it is used to build the BranchConnect models. For example, FC_Gates:100 indicates a fully connected layer with 100 output units

and also 100 gates, where each gate is connected to each output unit.

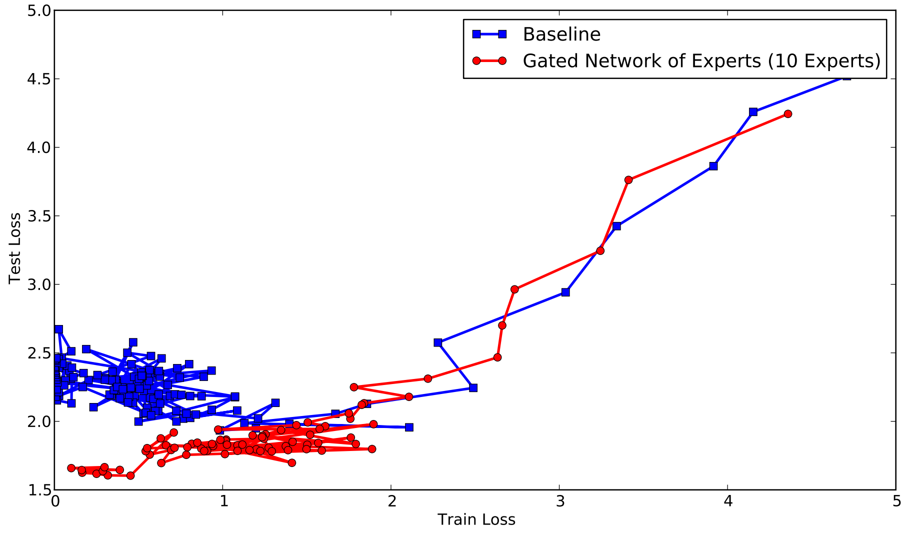
- **Convolution Layer with Gates:**
  CONV_Gates: <filter width> × <filter height>, <number of filters>
  Similar to fully-connected layer with gates, this is layer is connected to gates and it is used as an output layer for the BRANCHCONNECT models that were built using NIN architecture.
- **Residual block:**
  Residual blocks are the building units of the Residual Networks [7]. In Tables 9 and 10 we use big braces to refer to a residual block. For instance, the following: $\left\{\begin{array}{c}\text{CONV: } 3\times3{,}64\\ \text{CONV: } 3\times3{,}64\end{array}\right\}\times 9$, denotes a group of 9 residual blocks where each block consists of two layers CONV: 3×3,64, with batch normalization and scaling layers as proposed in [7].

Table 5: Classification accuracy (%) (single crop) on CIFAR-100 for 3 different base architectures. *"Base Models V1"* represent the original model architectures from prior work [17, 1], and "BRANCHCONNECT *V1"* are the equivalent BRANCHCONNECT models obtained by shrinking the numbers of parameters in BRANCHCONNECT to match those in these original base models. Conversely, "BRANCHCONNECT *V2"* represent higher-capacity BRANCHCONNECT models whose branches have the same specifications as the last layer(s) of *"Base Models V1"*. In order to compare "BRANCHCONNECT *V2"* with base models of the same capacity, we build *"Base Models V2"* obtained from *"Base Models V1"* by increasing uniformly the number of filters and the number of units. These results show that BRANCHCONNECT networks consistently outperform base models of the same capacity.

	Method	depth	#params	Accuracy
AlexNet-Quick	Base Model V1	5	0.15M	44.3
	BRANCHCONNECT G:1/10 V1	5	0.15M	**49.14**
	BRANCHCONNECT G:5/10 V1	5	0.15M	**52.81**
	Base Model V2	5	1.20M	40.26
	BRANCHCONNECT G:1/10 V2	5	1.20M	**53.28**
	BRANCHCONNECT G:5/10 V2	5	1.20M	**54.62**
Alexnet-Full	Base Model V1	4	0.18M	54.04
	BRANCHCONNECT G:1/10 V1	4	0.18M	**55.61**
	BRANCHCONNECT G:6/10 V1	4	0.18M	**56.73**
	Base Model V2	4	0.64M	50.42
	BRANCHCONNECT G:1/10 V2	4	0.64M	**57.34**
	BRANCHCONNECT G:6/10 V2	4	0.64M	**60.27**
NIN [17]	Base Model V1	9	1.38M	64.73
	BRANCHCONNECT G:1/10 V1	9	1.38M	**65.46**
	BRANCHCONNECT G:5/10 V1	9	1.38M	**65.99**
	Base Model V2	9	1.61M	65.24
	BRANCHCONNECT G:1/10 V2	9	1.61M	**66.10**
	BRANCHCONNECT G:5/10 V2	9	1.61M	**66.45**



(a) **AlexNet-Full**



(b) **Network In Network (NIN)**

Figure 5: Evolution of train loss vs test loss during training on CIFAR-100. The models were built from different architectures: a) AlexNet-Full architecture, b) Network In Network (NIN) architecture. The training trajectory is from right to left. BRANCHCONNECT yields lower test loss for the same train loss compared to the base models and a net with randomly-chosen active connections.

Table 6: **CIFAR100 AlexNet-Quick**

Base Model		BRANCHCONNECT
CONV: 5×5,32	**Stem**	CONV: 5×5,32
POOL: 3×3,Max,2		POOL: 3×3,Max,2
CONV: 5×5,32		CONV: 5×5,32
POOL: 3×3,Ave,2		POOL: 3×3,Ave,2
CONV: 5×5,64	**Branch**	CONV: 5×5,64
POOL: 3×3,Ave,2		POOL: 3×3,Ave,2
FC: 64		FC: 64
FC: 100		FC_Gates: 100
Data preprocessing		
Input: 32 × 32		Input: 32 × 32
Image mean subtraction		Image mean subtraction
Learning Policy		
• Learning rate: 0.001, 0.0001, 0.00001		• Learning rate: 0.001, 0.0001, 0.00001
• Momentum: 0.9		• Gate Learning rate: 10×Learning rate
• Weight decay: 0.004		• Momentum: 0.9
• Weight initialization: Random		• Weight decay: 0.004
		• Weight initialization: Random

Table 7: **CIFAR100 AlexNet-Full**

Base Model		BRANCHCONNECT
CONV: 5×5,32	**Stem**	CONV: 5×5,32
POOL: 3×3,Max,2		POOL: 3×3,Max,2
LRN		LRN
CONV: 5×5,32		CONV: 5×5,32
POOL: 3×3,Ave,2		POOL: 3×3,Ave,2
LRN		LRN
CONV: 5×5,64	**Branch**	CONV: 5×5,64
POOL: 3×3,Ave,2		POOL: 3×3,Ave,2
FC: 100		FC_Gates: 100
Data preprocessing		
Input: 32 × 32		Input: 32 × 32
Image mean subtraction		Image mean subtraction
Learning Policy		
• Learning rate: 0.001, 0.0001, 0.00001		• Learning rate: 0.001, 0.0001, 0.00001
• Momentum: 0.9		• Gate Learning rate: 10×Learning rate
• Weight decay: 0.004		• Momentum: 0.9
• Weight initialization: Random		• Weight decay: 0.004
		• Weight initialization: Random

Table 8: **CIFAR100 - Network In Network (NIN)**

Base Model		**BranchConnect**
CONV: 3×3,192 CONV: 3×3,184 CONV: 1×1,158 POOL: 3×3,Max,2	**Stem**	CONV: 3×3,192 CONV: 3×3,184 CONV: 1×1,158 POOL: 3×3,Max,2
CONV: 3×3,192 CONV: 3×3,192 CONV: 1×1,192 POOL: 3×3,Max,2 CONV: 3×3,192		CONV: 3×3,192 CONV: 3×3,192 CONV: 1×1,192 POOL: 3×3,Max,2 CONV: 3×3,192 POOL: 3×3,Ave,2
CONV: 1×,192	**Branch**	CONV: 1×,192
CONV: 1×1,100 POOL: 1×1,Ave,11		CONV_Gates: 1×1,100 POOL: 1×1,Ave,11
Data preprocessing		
Input: 32×32 Image mean subtraction		Input: 32×32 Image mean subtraction
Learning Policy		
• Learning rate: 0.025, 0.0125, 0.0001 • Momentum: 0.9 • Weight decay: 0.0005 • Weight initialization: Random		• Learning rate: 0.025, 0.0125, 0.0001 • Gate Learning rate: 10×Learning rate • Momentum: 0.9 • Weight decay: 0.0005 • Weight initialization: Random

Table 9: **CIFAR100 ResNet56-4X**

Base Model		**BranchConnect**
CONV: 3×3,64	**Stem**	CONV: 3×3,64
{CONV: 3×3,64; CONV: 3×3,64} ×9	**Stem**	{CONV: 3×3,64; CONV: 3×3,64} ×9
{CONV: 3×3,128; CONV: 3×3,128} ×9	**Stem**	{CONV: 3×3,128; CONV: 3×3,128} ×9
{CONV: 3×3,256; CONV: 3×3,256} ×9	**Stem**	{CONV: 3×3,256; CONV: 3×3,256} ×8
POOL: 7×7,Ave,1	**Branch**	{CONV: 3×3,256; CONV: 3×3,256} ×1 POOL: 7×7,Ave,1
FC: 100		FC_Gates: 100
Data preprocessing		
Input: 28 × 28 Image mean subtraction Image mirroring		Input: 28 × 28 Image mean subtraction Image mirroring
Learning Policy		
• Learning rate: 0.1, 0.01, 0.001 • Momentum: 0.9 • Weight decay: 0.0001 • Weight initialization: MSRA [6]		• Learning rate: 0.1, 0.01, 0.001 • Gate Learning rate: 10×Learning rate • Momentum: 0.9 • Weight decay: 0.0001 • Weight initialization: MSRA [6]

Table 10: **CIFAR100 ResNet56**

Base Model		**BranchConnect**
CONV: 3×3,16	**Stem**	CONV: 3×3,16
{CONV: 3×3,16; CONV: 3×3,16} ×9	**Stem**	{CONV: 3×3,16; CONV: 3×3,16} ×9
{CONV: 3×3,32; CONV: 3×3,32} ×9	**Stem**	{CONV: 3×3,32; CONV: 3×3,32} ×9
{CONV: 3×3,64; CONV: 3×3,64} ×9	**Stem**	{CONV: 3×3,64; CONV: 3×3,64} ×8
POOL: 7×7,Ave,1	**Branch**	{CONV: 3×3,64; CONV: 3×3,64} ×1 POOL: 7×7,Ave,1
FC: 100		FC_Gates: 100
Data preprocessing		
Input: 28 × 28 Image mean subtraction Image mirroring		Input: 28 × 28 Image mean subtraction Image mirroring
Learning Policy		
• Learning rate: 0.1, 0.01, 0.001 • Momentum: 0.9 • Weight decay: 0.0001 • Weight initialization: MSRA [6]		• Learning rate: 0.1, 0.01, 0.001 • Gate Learning rate: 10×Learning rate • Momentum: 0.9 • Weight decay: 0.0001 • Weight initialization: MSRA [6]

Table 11: **ImageNet AlexNet**

Base Model		**BranchConnect**
CONV: 11×11,96 CONV: 11×11,96 LRN POOL: 3×3,Max,2	**Stem**	11×11,96 11×11,96 LRN POOL: 3×3,Max,2
CONV: 3×3,384 CONV: 3×3,384 CONV: 3×3,256 LRN POOL: 3×3,Max,2		CONV: 3×3,384 CONV: 3×3,384
FC: 4096 FC: 4096	**Branch**	CONV: 3×3,256 LRN POOL: 3×3,Max,2 FC: 1024 FC: 1024
FC: 1000		FC_Gates: 1000
Data preprocessing		
Input: 227 × 227 Image mean subtraction		Input: 227 × 227 Image mean subtraction
Learning Policy		
• Learning rate: 0.01, 0.001, 0.001 • Momentum: 0.9 • Weight decay: 0.0005 • Weight initialization: Random		• Learning rate: 0.01, 0.001, 0.001 • Gate Learning rate: 10×Learning rate • Momentum: 0.9 • Weight decay: 0.0005 • Weight initialization: Random

Table 12: **ImageNet ResNet-50**

Base Model		**BranchConnect**
CONV: 7×7,64 POOL: 3×3,Max,2	**Stem**	CONV: 7×7,64 POOL: 3×3,Max,2
$\left\{\begin{array}{l}\text{CONV: }1\times1,64\\ \text{CONV: }3\times3,64\\ \text{CONV: }1\times1,256\end{array}\right\}\times3$		$\left\{\begin{array}{l}\text{CONV: }1\times1,64\\ \text{CONV: }3\times3,64\\ \text{CONV: }1\times1,256\end{array}\right\}\times3$
$\left\{\begin{array}{l}\text{CONV: }1\times1,128\\ \text{CONV: }3\times3,128\\ \text{CONV: }1\times1,512\end{array}\right\}\times4$		$\left\{\begin{array}{l}\text{CONV: }1\times1,128\\ \text{CONV: }3\times3,128\\ \text{CONV: }1\times1,512\end{array}\right\}\times4$
$\left\{\begin{array}{l}\text{CONV: }1\times1,256\\ \text{CONV: }3\times3,256\\ \text{CONV: }1\times1,1024\end{array}\right\}\times6$		$\left\{\begin{array}{l}\text{CONV: }1\times1,256\\ \text{CONV: }3\times3,256\\ \text{CONV: }1\times1,1024\end{array}\right\}\times6$
$\left\{\begin{array}{l}\text{CONV: }1\times1,512\\ \text{CONV: }3\times3,512\\ \text{CONV: }1\times1,2048\end{array}\right\}\times3$		$\left\{\begin{array}{l}\text{CONV: }1\times1,512\\ \text{CONV: }3\times3,512\\ \text{CONV: }1\times1,2048\end{array}\right\}\times2$
POOL: 7×7,Ave,1	**Branch**	$\left\{\begin{array}{l}\text{CONV: }1\times1,512\\ \text{CONV: }3\times3,512\\ \text{CONV: }1\times1,1024\end{array}\right\}\times1$ POOL: 7×7,Ave,1
FC: 1000		FC_Gates: 1000

Data preprocessing	
Input: 224 × 224 Image mean subtraction Image mirroring	Input: 224 × 224 Image mean subtraction Image mirroring
Learning Policy	
• Learning rate: 0.1, 0.01, 0.001 • Momentum: 0.9 • Weight decay: 0.0001 • Weight initialization: MSRA [6]	• Learning rate: 0.1, 0.01, 0.001 • Gate Learning rate: 10×Learning rate • Momentum: 0.9 • Weight decay: 0.0001 • Weight initialization: MSRA [6]

Table 13: **ImageNet ResNet-101**

Base Model		BRANCHCONNECT
CONV: 7×7,64 POOL: 3×3,Max,2	**Stem**	CONV: 7×7,64 POOL: 3×3,Max,2
$\left\{\begin{array}{l}\text{CONV: }1\times1,64\\\text{CONV: }3\times3,64\\\text{CONV: }1\times1,256\end{array}\right\}\times3$		$\left\{\begin{array}{l}\text{CONV: }1\times1,64\\\text{CONV: }3\times3,64\\\text{CONV: }1\times1,256\end{array}\right\}\times3$
$\left\{\begin{array}{l}\text{CONV: }1\times1,128\\\text{CONV: }3\times3,128\\\text{CONV: }1\times1,512\end{array}\right\}\times4$		$\left\{\begin{array}{l}\text{CONV: }1\times1,128\\\text{CONV: }3\times3,128\\\text{CONV: }1\times1,512\end{array}\right\}\times4$
$\left\{\begin{array}{l}\text{CONV: }1\times1,256\\\text{CONV: }3\times3,256\\\text{CONV: }1\times1,1024\end{array}\right\}\times23$		$\left\{\begin{array}{l}\text{CONV: }1\times1,256\\\text{CONV: }3\times3,256\\\text{CONV: }1\times1,1024\end{array}\right\}\times23$
$\left\{\begin{array}{l}\text{CONV: }1\times1,512\\\text{CONV: }3\times3,512\\\text{CONV: }1\times1,2048\end{array}\right\}\times3$		$\left\{\begin{array}{l}\text{CONV: }1\times1,512\\\text{CONV: }3\times3,512\\\text{CONV: }1\times1,2048\end{array}\right\}\times2$
POOL: 7×7,Ave,1	**Branch**	$\left\{\begin{array}{l}\text{CONV: }1\times1,512\\\text{CONV: }3\times3,512\\\text{CONV: }1\times1,1024\end{array}\right\}\times1$ POOL: 7×7,Ave,1
FC: 1000		FC_Gates: 1000

Data preprocessing	
Input: 224 × 224 Image mean subtraction Image mirroring	Input: 224 × 224 Image mean subtraction Image mirroring
Learning Policy	
• Learning rate: 0.1, 0.01, 0.001 • Momentum: 0.9 • Weight decay: 0.0001 • Weight initialization: MSRA [6]	• Learning rate: 0.1, 0.01, 0.001 • Gate Learning rate: 10×Learning rate • Momentum: 0.9 • Weight decay: 0.0001 • Weight initialization: MSRA [6]

Table 14: **DICT+2-90k**

Base Model		**BranchConnect**
CONV: 5×5,64 POOL: 2×2,Max,2	**Stem**	CONV: 5×5,64 POOL: 2×2,Max,2
CONV: 5×5,128 POOL: 2×2,Max,2	**Stem**	CONV: 5×5,128 POOL: 2×2,Max,2
CONV: 3×3,256 POOL: 2×2,Max,2	**Stem**	CONV: 3×3,256 POOL: 2×2,Max,2
CONV: 3×3,512	**Stem**	CONV: 3×3,512
CONV: 3×3,512	**Branch**	CONV: 3×3,512
FC: 4096	**Branch**	FC: 256
FC: 4096	**Branch**	FC: 256
FC: 90000		FC_Gates: 90000
Data preprocessing		
Input: 32×100 Image mean subtraction		Input: 32×100 Image mean subtraction
Learning Policy		
• Incremental learning starting with 5000 classes till convergence then adding another 5000 till all 90000 are added. • Dropout after fully-connected layers. • Learning rate: 0.01, 0.001, 0.0001 • Momentum: 0.9 • Weight decay: 0.0005 • Weight initialization: Random		• Incremental learning starting with 5000 classes till convergence then adding another 5000 till all 90000 are added. • No Dropout used. • Learning rate: 0.01, 0.001, 0.0001 • Gate Learning rate: 10×Learning rate • Momentum: 0.9 • Weight decay: 0.0005 • Weight initialization: Random