A Branching and Merging Convolutional Network with Homogeneous Filter Capsules

Adam Byerly
Department of Electronic and Computer Engineering
Brunel University London
Uxbridge, UB8 3PH UK
Department of Computer Science and Information Systems
Bradley University
Peoria, Il, 61615 USA
abyerly@fsmail.bradley.edu

Tatiana Kalganova
Brunel University London
Uxbridge, UB8 3PH UK
tatiana.kalganova@brunel.ac.uk

Ian Dear
Brunel University London
Uxbridge, UB8 3PH UK
ian.dear@brunel.ac.uk

ABSTRACT

We present a convolutional neural network design with additional branches after certain convolutions so that we can extract features with differing effective receptive fields and levels of abstraction. From each branch, we transform each of the final filters into a pair of homogeneous vector capsules. As the capsules are formed from entire filters, we refer to them as filter capsules. We then compare three methods for merging the branches—merging with equal weight and merging with learned weights, with two different weight initialization strategies. This design, in combination with a domain-specific set of randomly applied augmentation techniques, establishes a new state of the art for the MNIST dataset with an accuracy of **99.84%** for an ensemble of these models, as well as establishing a new state of the art for a single model (**99.79%** accurate). These accuracies were achieved with a 75% reduction in both the number of parameters and the number of epochs of training relative to the previously best performing capsule network on MNIST. All training was performed using the Adam optimizer and experienced no overfitting.

Keywords Capsules, Convolutional Neural Network (CNN), Homogeneous Vector Capsules (HVCs), Homogeneous Filter Capsules (HFCs), MNIST

1 Introduction and Related Work

The best performing convolutional networks of the past several years have all explored multiple paths from input to classification [1][2][3][4][5][6]. The idea behind multiple path designs is to enable one or more of the following to contribute to the final classification: (a) different levels of abstraction, (b) different effective receptive fields, and (c) valuable information learned early to flow more easily to the classification stage. In [1] and [2], the authors' design branched and merged by filter concatenation many times as well as producing two different classifications that were summed together, each having been assigned predetermined weights. In this paper, we present a network design that produces three classifications and we report the results of experiments that sum them together with both predetermined equal weights and with weights learned via backpropagation.

Capsules (vector-valued neurons) have become a more active area of research since [7], which demonstrated near state of the art performance on MNIST [8] classification (at 99.75%) by using capsules and a routing algorithm to determine which capsules in a previous layer feed capsules in the subsequent layer. In [9], the authors extended this work by

conducting experiments with an alternate routing algorithm. In [10], the authors presented a capsule network design that used element-wise multiplication between capsules in subsequent layers with no routing mechanism, rather than matrix multiplication with a routing mechanism. They referred to this capsule design as homogeneous vector capsules. In this paper, we will be detailing a network design and experiments conducted with homogeneous vector capsules applied to the MNIST classification task which achieved a single model accuracy of 99.79% and an ensemble accuracy of 99.84%. Our design accomplishes this with a 75% reduction in both the number of parameters in the model and the number of epochs used to train a single model (relative to [7]).

Most (but not all [11][12]) of the state of the art MNIST results achieved over the past decade have used data augmentation [13][14][6]. In addition to the network design, a major part of our work involved applying an effective data augmentation strategy that included transformations informed specifically by the domain of the data. For example, we wanted to be sure we did not rotate our images into being more like a different class (e.g. rotating an image of the digit 2 by 180 degrees to create something that would more closely resemble a malformed 5). Nor did we want to translate the image content off of the canvas and perhaps cut off the left side of an 8 and thus create a 3.

2   Proposed Network Design

The starting point for our network design was a conventional convolutional neural network following many widely used practices. These include stacked 3×3 convolutions, each of which used ReLU [15] activation preceded by batch normalization [16]. We also followed the common practice of increasing the number of filters in each subsequent convolutional operation relative to the previous one. Specifically, our first convolution uses 32 filters and each subsequent convolution uses 16 more filters than the previous one, resulting in 160 filters present in the final convolution. Additionally, the final operation before classification was to softmax the logits and to use categorical cross entropy for calculating loss.

One common design element found in many convolutional neural networks which we intentionally avoided was the use of any pooling operations. We agree with Geoffrey Hinton's assessment [17] of pooling as an operation to be avoided due to the information it "throws away". Effectively, pooling is a form of down-sampling and, in the presence of sufficient computational power, should be avoided. With the MNIST data being 28×28, we have no need to down-sample as there exists sufficient computational power for images of this size. In choosing not to down-sample, we face the potential dilemma of how to reduce the dimensionality as we descend deeper into the network. This dilemma is solved by choosing not to zero-pad the convolution operations and thus each convolution operation by its nature reduces the dimensionality by 2 in both the horizontal and vertical dimensions. We deem choosing not to zero-pad as preferable in its own right in that zero padding effectively adds information not present in the original sample.

Beyond these design choices, we chose to employ two relatively novel design elements:

1) Rather than having a single monolithic design such that each operation in our network feeds into the next operation and only the next operation, we chose to create multiple branches. After the first two sets of three convolutions, in addition to feeding to the subsequent convolution, we also branched off the output to be forwarded on to an additional operation (detailed next). Thus, after all convolutions have been performed, we have three branches in our network.

   a) The first of which has been through three 3×3 convolutions and consists of 64 filters each having an effective receptive field of 7 of the original image pixels.

   b) The second of which has been through six 3×3 convolutions and consists of 112 filters each having an effective receptive field of 11 of the original image pixels.

   c) The third of which has been through nine 3×3 convolutions and consists of 160 filters each having an effective receptive field of 15 of the original image pixels.

2) For each branch, rather than flattening the outputs of the convolutions into scalar neurons, we instead transformed each filter into a vector to form the first capsule in a pair of homogeneous vector capsules. We then performed element-wise multiplication of each of these vectors with a set of weight vectors (one for each class) of the same length. This results in *nxm* weight vectors where *n* is the number of filters that were transformed into the first set of capsules and *m* is the number of classes. We summed across the filters to form the second capsule in the pair of homogeneous vector capsules. It is after this that we applied batch normalization and then ReLU activation. Because these capsules are formed one-to-one from entire filters, we see them as a sub-type of homogeneous vector capsules which we refer to as a Homogeneous Filter Capsules (HFCs). We reduce each vector to a single value per class by summing the components of the vector. These values can be thought of as the branch-level logits. As the filter size coming out of the first branch is 22×22, the length of the HFC vectors for this branch is 484. For the second branch, consisting of 16×16 sized filters,

the vectors are of length 256, and for the third branch, consisting of 10×10 sized filters, the vectors are of length 100. Figure 1a diagrams the transformation from filter maps through the homogeneous filter capsules to the output logits.

Before classifying, we needed to reconcile the three branch-level sets of logits with each image only belonging to one class. We accomplished this by stacking the branch-level logits into vectors of length 3, one for each class. We then reduced by summation each vector to a single value to form the final set of logits to be classified from. Figure 1b shows the high-level view of the entire network.

(a) The transformation from filter maps through the homogeneous filter capsules to the output logits. In this illustration, a single 3×3 filter map and 9-dimensional capsule are shown and represent the n filter maps and capsules, respectively.

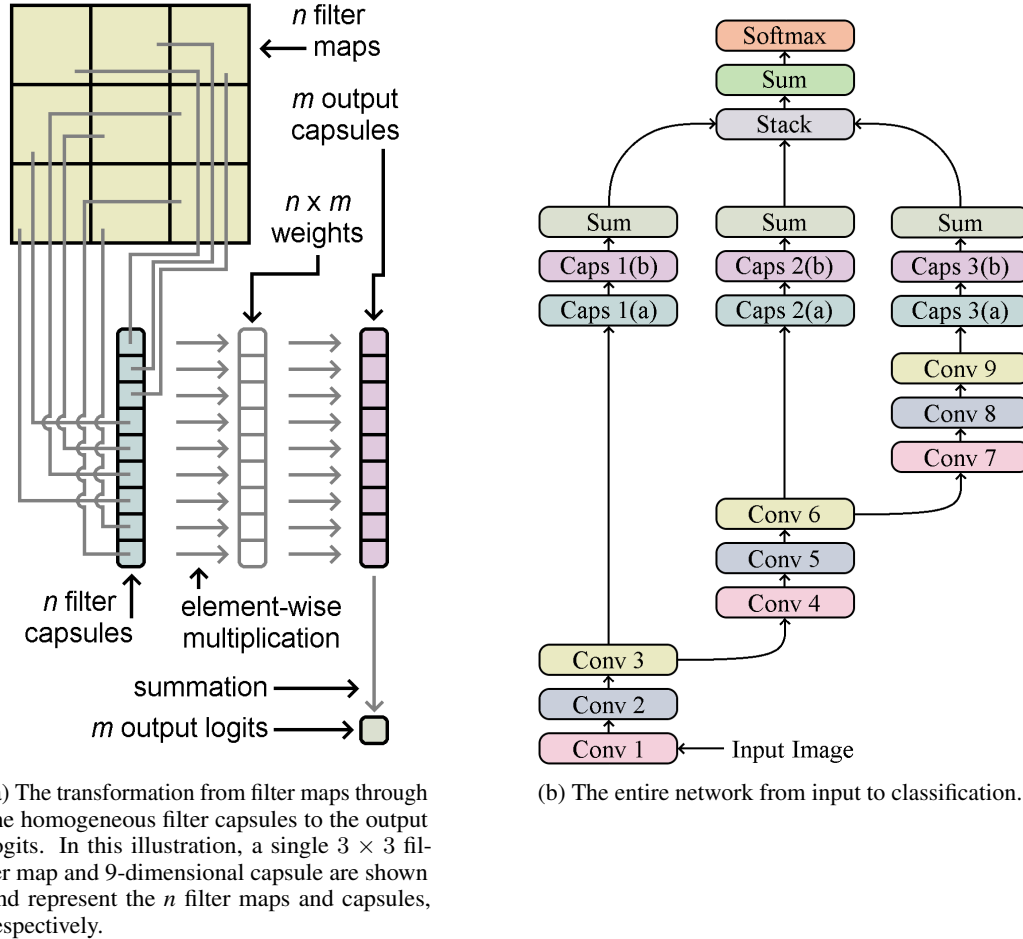(b) The entire network from input to classification.

Figure 1: Proposed Network Design

We used no weight decay regularization [18], a staple regularization method that improves generalization by penalizing the emergence of large weight values. Nor did we use any form of dropout regularization [19][14] which are regularization methods designed to stop the co-adaption of weights. This decision was made so that we could investigate the generalization properties of our novel network design elements in the absence of other techniques associated with good generalization. In addition, we intentionally left out any form of "routing" algorithm as in [7] and [9], preferring to rely on traditional trainable weights (see Table 1) and back-propagation.

3 Experimental Setup

3.1 Merge Strategies

In [1] and [2], the authors chose to give static, predetermined weights to both output branches and then added them together. In our case, we conducted three separate experiments of 32 trials each in order to investigate the effects of predetermined equal weighting of the branch outputs compared to learning the branch weights via backpropagation:

Table 1: Trainable Parameters

Parameter Type	Count
Convolutional Filters	756,000
Capsules	756,480
Batch Normalization	1,707
Total	1,514,187

The experiments for which the branch weights were learned required an additional 3 trainable parameters (one for each branch weight).

1) **Not learnable**. For this experiment, we merged the three branches together with equal weighting in order to investigate the effect of disallowing any one branch to have more impact than any other.

2) **Learnable with randomly initialized branch weights**. (Abbreviated as **Random Init.** subsequently.) For this experiment, we allowed the weights to be learned via backpropagation. We initialized the 3 trainable parameters using a Glorot uniform distribution [20], which for the case of a vector of 3 trainable parameters happens to be a random uniform distribution within the range [-1,1].

3) **Learnable with branch weights initialized to one**. (Abbreviated as **Ones Init.** subsequently.) For this experiment, we also allowed the weights to be learned via backpropagation. The difference with the **Random Init.** experiment being that we initialized the weights to 1. We conducted this experiment in addition to the **Random Init.** experiment in order to understand the difference between starting with random weights and starting with equal weights that are subsequently allowed to diverge during training.

3.2 Data Augmentation

By modern standards, in terms of dataset size, MNIST has a relatively low number of training images. As such, judicious use of appropriate data augmentation techniques is important for achieving a high level of generalizability in a given model. In terms of structure, hand-written digits show a wide variety in their rotation relative to some shared true "north", position within the canvas, width relative to their height, and the connectedness of the strokes used to create them. Throughout training for all trials, every training image in every epoch was subjected to a series of four operations in order to simulate a greater variety of the values for these properties.

1) **Rotation**. First, we randomly rotated each training image by up to 30 degrees in either direction. The amount of rotation applied to each training image was chosen by multiplying the value 30 by a sample drawn from a random normal distribution with mean 0 and standard deviation 0.33, clamped to a minimum of -1 (which would result in a left rotation of 30 degrees) and a maximum of 1 (which would result in a right rotation of 30 degrees). Whether to actually apply this rotation was chosen by drawing from a Bernoulli distribution with probability p of 0.5 (a fair coin toss).

2) **Translation**. Second, we randomly translated each training image within the available margin present in that image. In [7], the authors limited their augmentation to shifting the training images randomly by up to 2 pixels in either or both directions. The limit of only 2 pixels for the translation ensured that the translation is label-preserving. As the MNIST training data has varying margins of non-digit space in the available 28×28 pixel canvas, using more than 2 pixels randomly, would be to risk cutting off part of the digit and effectively changing the class of the image. For example, a 7 that was shifted too far left could become more appropriately classed as a 1, or an 8 or 9 shifted far enough down could be more appropriately classed as a zero. The highly structured nature of the MNIST training data allows for an algorithmic analysis of each image that will provide the translation range available for that specific image that will be guaranteed to be label-preserving. Figure 2 shows an example of an MNIST training image that has an asymmetric translation range that, as long as any translations are performed such that the digit part of the image is not moved by more pixels than are present in the margin, will be label preserving. In other words, the specific training example shown in Figure 2 could be shifted by up to 8 pixels to the left or 4 to the right and up to 5 up or 3 down, and after doing so, all of the pixels belonging to the actual digit will still be in the resulting translated image. The amount within this margin to actually translate a training image was chosen as follows:

   a) Whether to translate up or down and whether to translate left or right were drawn independently from a Bernoulli distribution with probability p of 0.5 (a fair coin toss).

   b) The amount of translation across the margin in each chosen direction was determined from the absolute values of two independent samples drawn from a normal distribution with mean 0 and standard deviation

0.33 and clamped to a maximum translation of the entire margin as to avoid translating out of the image's bounds.
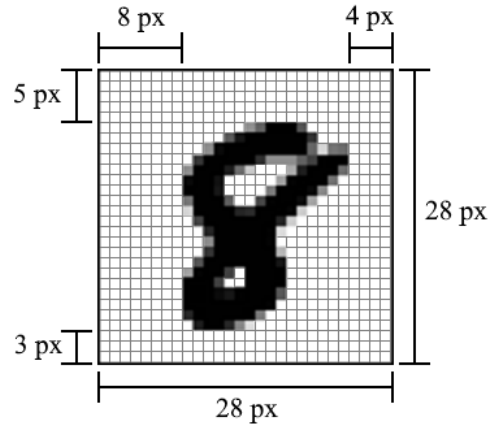

Figure 2: Example MNIST digit w/annotated margins.

3) **Width**. Third, we randomly adjusted each training image's width. MNIST images are normalized to be within a 20×20 central patch of the 28×28 canvas. This normalization is ratio-preserving, so all images are 20 pixels in the height dimension but vary in the number of pixels in the width dimension. This variance not only occurs across digits, but intra-class as well, as different peoples' handwriting can be thinner or wider than average. In order to train on a wider variety of these widths, we randomly compressed each image's width and then added equal zero padding on either side, leaving the digit's center where it was prior. This was inspired by a similar approach adopted in [6]. In their work, they created 6 additional versions of the MNIST training data set, where they normalized the width of the digits to 10, 12, 14, 16, 18, and 20 pixels. They then fed those data sets as well as the original MNIST data into 7 columns in their architecture. In our work, we compressed the width of each sample randomly within a range of 0–25%. The portion of that range of compression was the absolute value of a sample drawn from a random normal distribution with mean 0 and standard deviation 0.33, clamped to a maximum of 100% (that is 100% of the 25%).

4) **Random Erasure**. Fourth, we randomly erased (setting to 0) a 4×4 grid of pixels chosen from the central 20×20 grid of pixels in each training image. The X and Y coordinates of the patch were drawn independently from a random uniform distribution. This was inspired by the random erasing data augmentation method in [21]. The intention behind this method was to expose the model to a greater variety of (simulated) connectedness within the strokes that make up the digits. An alternative interpretation would be to see this as a kind of feature-space dropout.

3.3 Training

In [10], the authors show that Homogeneous Vector Capsules (HVCs) enable the use of adaptive gradient descent methods in convolutional neural networks, a practice previously deemed sub-optimal and prone to extreme overfitting. We followed the training methodology they used and trained with the Adam optimizer [22] using all of the default/recommended parameter values, including the base learning rate of 0.001. Also, as in both [10] and [7], we exponentially decayed the base learning rate. For our experiments, which trained for 300 epochs, we applied an exponential decay rate of 0.98 per epoch, clamped to a minimum of 1×10^{-6}. And like in [10], we were able to continue to train for many epochs without experiencing overfitting. (See Figure 3 and Figure 4.)

Test accuracy was measured using the exponential moving average of prior weights with a decay rate of 0.999. [23]

4 Experimental Results

4.1 Individual Models

For each of our three experiments, we ran 32 trials, each of which with weights randomly initialized prior to training and, due to the stochastic nature of the data augmentation, a different set of training images. As a result, training

progressed to different points in the loss surface resulting in a range of values for the top accuracies that were achieved on the test set. See Table 2.

Table 2: Individual Models

Experiment	Min	Max	Mean	SD
Not Learnable	99.71%	**99.79%**	0.997500	0.0002190
Random Init.	99.72%	**99.78%**	0.997512	0.0001499
Ones Init.	99.70%	99.77%	0.997397	0.0001885

32 trials of each individual model showed no statistically significant difference between the approach where the branch weights were not learnable and when the branch weights were learnable and randomly initialized. The approach with learnable branch weights initialized to one was inferior to the approach that did not use learnable branch weights (p-value 0.024045) as well as being inferior to the approach with learnable weights initialized randomly (p-value 0.004449). (SD is standard deviation)

The trial that achieved 99.79% test accuracy establishes a new state of the art for a single model where the previous state of the art was 99.77% [13]. 4 additional trials achieved 99.78% test accuracy, also surpassing the previous state of the art.

Although all three experiments produced similar results, the experiment that initialized the learnable branch weights randomly had a higher average accuracy across a greater number of epochs than the experiment that used non-learnable and equal branch weights and higher than all epochs of the experiment that initialized the learnable branch weights to one (see Figure 3). Additionally, the experiment that initialized the learnable branch weights randomly had a lower loss than either of the other two experiments across all epochs (see Figure 4).

4.2 Ensembles

Ensembling multiple models together and predicting based on the majority vote among the ensembled models routinely outperforms the individual models' performances. Ensembling can refer to either completely different model architectures with different weights or the same model architecture after being trained multiple times and finding different sets of weights that correspond to different locations in the loss surface. The previous state of the art of 99.82% was achieved using an ensemble of 30 different randomly generated model architectures [5]. Our ensembling method used the same architecture with different sets of weights learned during different trials. We matched the previous state of the art with 4,544 ensembles. We surpassed this with 44 ensembles that achieved an accuracy of 99.83% and established a new state of the art of 99.84% with one ensemble. See Table 3.

In order to find these ensembles, we ran 32 trials for each of the three experiments. For each trial, we saved the weights that were used to achieve the highest test accuracy throughout the 300 epochs of training. We then calculated the majority vote for all possible combinations of those weights across trials.

Table 3: Ensembles

Accuracy:	**99.84%**	**99.83%**	99.82%
Not Learnable	0	4	1,183
Random Init.	0	21	2,069
Ones Init.	1	19	1,292

Shown here are the number of ensembles that were generated that either matched the previous state of the art of 99.82% or exceeded it.
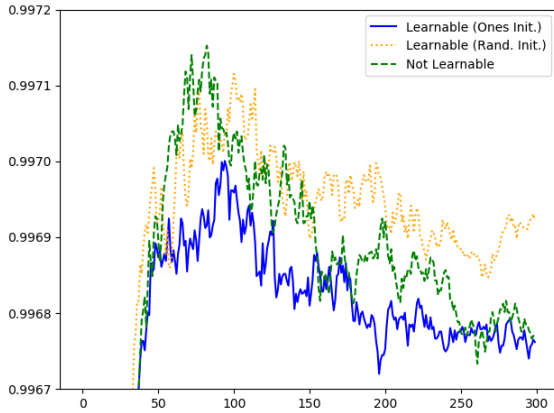
Figure 3: The average test accuracies of the trials for all three experiments across epochs.
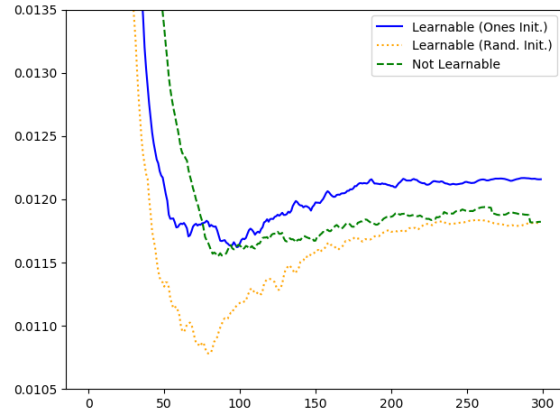

Figure 4: The average test losses of the trials for all three experiments across epochs.

4.3 Branch Weights

What follows are visualizations of the final branch weights (after 300 epochs of training) for each of the branches in all 32 trials of the experiment wherein the branch weights were initialized to one (see Figure 5) and of the experiment wherein the branch weights were initialized randomly (see Figure 6).

In Figure 5, we see that for all trials, the ratio between the learned branch weights is consistent, demonstrating that the amount of contribution from each branch plays a significant role. In Figure 6, as the weights were initialized randomly, the initial weights for some trials' branches were negative, leading the backpropagation algorithm to learn a negative weight for that specific weight of the branch. This does not mean that the branch prediction was consistently poor and thus down-weighted, but rather, the backpropagation algorithm copes with the initial negative value of the branch by learning the inverse relationship between weight of detected features and the predicted class. Regardless of the sign of the branch weight, the magnitude of each branch's weight is consistent across trials and consistent with the trials from the experiment in which the weights were initialized to one.

Figure 5: Final branch weights (after 300 epochs) for all 32 trials (across the x-axis) of the experiment for which the branch weights were initialized to one.

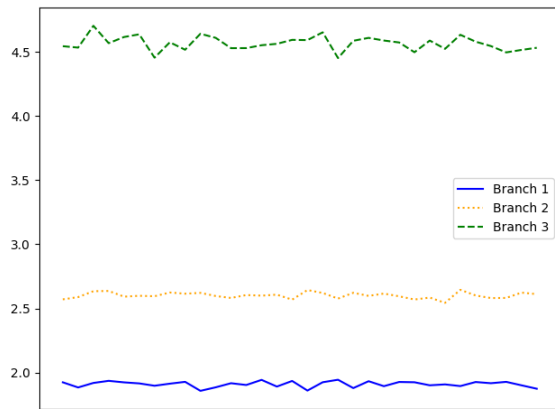Figure 6: Final branch weights (after 300 epochs) for all 32 trials (across the x-axis) of the experiment for which the branch weights were initialized randomly.

4.4 Troublesome Digits
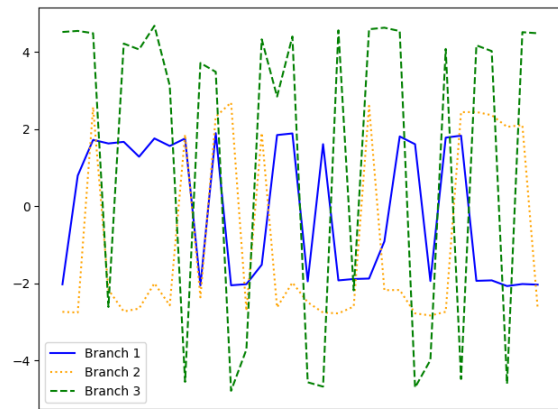
Across all 96 trials, there was total agreement on 9,907 out of the 10,000 test samples. There were 48 digits that were misclassified more often than not across all 96 trials, but only 21 digits were misclassified more often than not within

the 32 trials of any one experiment. This shows that although the accuracies of the models in the three experiments were quite similar, the different merge strategies of the three experiments did have a significant effect on classification.

Across all 96 trials, only 4 samples were misclassified in all models. Those samples, as numbered by the order they appear in the MNIST test dataset (starting from 0) are 1901, 2130, 2293, and 6576. The **Not Learnable** experiment had no trial that correctly predicted sample 3422. The **Random Init.** experiment had no trial that correctly predicted sample 2597 correctly. The **Ones Init.** experiment had no trial in which either 2597 or 3422 were predicted correctly. It is interesting that, in addition to the 4 that no trial predicted correctly, the 2 that the **Ones Init.** experiment predicted incorrectly were the fifth digits not predicted by the **Not Learnable** and **Random Init.** experiments.
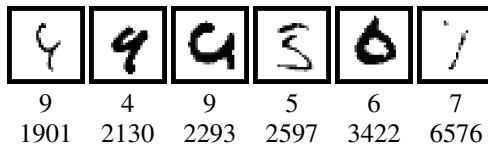

| 9 | 4 | 9 | 5 | 6 | 7 |
| 1901 | 2130 | 2293 | 2597 | 3422 | 6576 |

Figure 7: The Most Troublesome Digits

4.5 MNIST State of the Art

In Table 4 we present a comparison of previous state of the art MNIST results for both single model evaluations and ensembles along with the results achieved in our experiments.

Table 4: Current and Previous MNIST State of the Art Results

Paper	Year	Accuracy
Single Models		
Dynamic Routing Between Capsules[7]	2017	99.75%
Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures[11]	2016	99.75%
Batch-Normalized Maxout Network in Network[12]	2015	99.76%
APAC:Augmented PAttern Classification with Neural Networks[13]	2015	99.77%
Multi-Column Deep Neural Networks for Image Classification[6]	2012	99.77%
Using the method proposed in this paper (Branching & Merging CNN w/HFCs)	**2020**	**99.79%**
Ensembles		
Regularization of Neural Networks using DropConnect[14]	2013	99.79%
RMDL:Random Multimodel Deep Learning for Classification[5]	2018	99.82%
Using the method proposed in this paper (An ensemble of Branching & Merging CNN w/HFCs)	**2020**	**99.84%**

How long a model takes to train is an important factor to consider when evaluating a neural network. Indeed, it is an enabling factor during initial experimentation as faster training leads to a greater exploration of the design space. In Table 5 we present a comparison of the number of epochs of training used in experiments for the results achieved in the networks shown in Table 4. Across all trials, our design achieved peak accuracy in an average of 88.35 epochs, with a minimum peak achieved in 25 epochs and a maximum peak achieved at epoch 266. One trial achieved an accuracy of 99.78%, surpassing the previous state of the art, in 56 epochs. Since, we allowed all trials to run for up to 300 epochs, that is the number we report in Table 5.

4.6 Experiments With and Without Branching and Merging and Homogeneous Filter Capsules

In order to understand the effect of the novel design elements we introduced, we ran additional sets of paired experiments wherein the first set of experiments in a pair used the network design as described in this paper and the second set of experiments excludes the branching and merging and capsules. This second network design is the same network design described in this paper and as shown in 1b excluding the the first two branches, the stacking operation that combines the branches, and the homogeneous filter capsules in the main branch which were replaced with the typical flattening of the final convolution and a fully connected layer of neurons, one for each output class. This results in a very typical convolutional neural network with 9 3×3 convolutions. In our experiments, we labeled these two models M1 and M2, respectively (see Table 6).

8

Table 5: Epochs of Training

Paper	Epochs
Dynamic Routing Between Capsules[7]	1,200
APAC:Augmented PAttern Classification with Neural Networks[13]	15,000
Multi-Column Deep Neural Networks for Image Classification[6]	800
Regularization of Neural Networks using DropConnect[14]	1,200
RMDL:Random Multimodel Deep Learning for Classification[5]	120
Branching & Merging CNN w/HFCs	300

Neither [11] nor [12] report on how many epochs their designs were trained for.

Then, in addition to performing these sets of paired experiments on MNIST, we performed them on Fashion-MNIST [24], CIFAR-10, and CIFAR-100 [25].

For the MNIST and Fashion-MNIST experiments, we used the same augmentation strategy as described in this paper. For the CIFAR-10 and CIFAR-100 experiments, the augmentation strategy was as described in this paper with the exception of the translation operation (described in section 3), which works by detected the empty space around the MNIST and Fashion-MNIST images. There is no such empty space in the CIFAR-10 and CIFAR-100 images. Additionally, since CIFAR-10 and CIFAR-100 images are 32×32 pixels, are full color, and are comprised of more complex features, we ran an additional pair of experiments for each of these datasets that used additional convolutions. We labeled these two additional models M3 and M4 (see Table 6).

For all four datasets, a model that included the branching and merging and HFCs achieved the highest mean accuracy with statistical significance.

For MNIST, the higher accuracy was statistically significant with a p-value of 0.000757. For Fashion-MNIST, the higher accuracy was statistically significant with a p-value of 0.000003. Given that both of these datasets are monochromatic images with a size of 28×28 and our network was designed with those properties in mind, this is not especially surprising. The fact that the accuracy for Fashion-MNIST was not competitive with current state of the art for that dataset is also not surprising as our network design was optimized for accuracy on classification of the Arabic numerals in the MNIST dataset. This design encompasses the number of parameters used as well as the domain specific augmentation strategy, both of which would quite likely be different for achieving state of the art Fashion-MNIST accuracy.

For CIFAR-10, model M2 (without branching and merging and HFCs) achieved a statistically significant higher accuracy than M1 with a p-value of 0.027646, while model M3 achieved a statistically significant higher accuracy than M4 with a p-value of 0.018672. The higher accuracy of model M3 relative to M2 was statistically significant with a p-value of 0.049882.

For CIFAR-100, model M1 (with branching and merging and HFCs) achieved a statistically significant higher accuracy than M2 with a p-value of 0.000196, and model M3 achieved a statistically significant higher accuracy than M4 with a p-value of 0. The higher accuracy of model M3 relative to M1 was not statistically significant after 5 trials with a p-value of 0.094366.

Table 6: Models Used for Experiments With and Without Branching and Merging and Homogeneous Filter Capsules

Model	Description
M1	The model described in this paper
M2	M1 without branching and merging and HFCs
M3	M1 with 5 additional 3×3 convolutions and branches after convolutions 5, 9, and 13
M4	M2 with 5 additional 3×3 convolutions

5   Conclusion

In this paper, we proposed using a simple convolutional neural network and established design principles as a basis for our architecture. We then presented a design that branched out of the series of stacked convolutions at different points to capture different levels of abstraction and effective receptive fields, and from these branches, rather than flattening to individual scalar neurons, used capsules instead.

Table 7: Experiments With and Without Branching and Merging and Homogeneous Filter Capsules

| | Accuracy | | | |
	Min	Max	Mean	SD
MNIST				
M1	99.74%	**99.76%**	0.997540	0.0000894
M2	99.66%	99.73%	0.996940	0.0002702
Fashion-MNIST				
M1	93.42%	**93.66%**	0.935440	0.0008989
M2	92.85%	93.04%	0.929480	0.0008871
CIFAR-10				
M1	84.57%	85.44%	0.849020	0.0033656
M2	85.01%	85.70%	0.853200	0.0024617
M3	85.36%	**85.71%**	0.855580	0.0014550
M4	85.09%	85.50%	0.853040	0.0017530
CIFAR-100				
M1	50.42%	51.73%	0.512740	0.0051013
M2	49.07%	50.19%	0.495760	0.0040470
M3	51.41%	**51.96%**	0.516320	0.0022410
M4	46.27%	47.59%	0.469400	0.0053245

We conducted 5 trials of each unique type of experiment in order to establish statistical significance. (SD is standard deviation)

We also investigated three different methods of merging the output of the branches back into a single set of logits. Each of the three merge strategies generated models that could be ensembled to create new state of the art results. Although the experiment that initialized the branch weights to ones produced an ensemble with a higher accuracy than the other two experiments, the experiment that initialized the branch weights randomly produced the most ensembles at or exceeding the previous state of the art, as well as having a slightly higher average and lower standard deviation across the trials. This suggests that the random initialization method is preferred.

Beyond the network architecture, we proposed a robust and domain specific data augmentation strategy aimed at simulating a wider variety of renderings of the digits.

In doing this work, we established new MNIST state of the art accuracies for both a single model and an ensemble. In addition to the network design and augmentation strategy, the ability to use an adaptive gradient descent method [10] allowed us to achieve this on consumer hardware (2x NVIDIA GeForce GTX 1080 Tis in an otherwise unremarkable workstation) and was an enabling factor in both initial explorations and the training of the 96 trials.

References

[1] Christian Szegedy et al. "Going Deeper with Convolutions". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 1–9.

[2] Christian Szegedy et al. "Rethinking the Inception Architecture for Computer Vision". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 2818–2826.

[3] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2015), pp. 770–778.

[4] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 2261–2269.

[5] Kamran Kowsari et al. "RMDL: Random Multimodel Deep Learning for Classification". In: *Proceedings of the 2nd International Conference on Information System and Data Mining (ICISDM 2018)*. 2018, pp. 19–28.

[6] Dan C. Ciresan, Ueli Meier, and Jürgen Schmidhuber. "Multi-Column Deep Neural Networks for Image Classification". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2012), pp. 3642–3649.

[7]   Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic Routing Between Capsules". In: *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*. 2017.

[8]   Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database". In: *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[9]   Geoffrey E. Hinton, Sara Sabour, and Nicholas Frosst. "Matrix Capsules with EM Routing". In: *Proceedings of the 6th International Conference on Learning Representations (ICLR 2018)*. 2018.

[10]   Adam Byerly and Tatiana Kalganova. *Homogeneous Vector Capsules Enable Adaptive Gradient Descent in Convolutional Neural Networks*. 2019. arXiv: 1906.08676 [cs.CV].

[11]   Seyyed Hossein Hasanpour et al. *Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures*. 2016. arXiv: 1608.06037 [cs.CV].

[12]   Jia-Ren Chang and Yong-Sheng Chen. *Batch-Normalized Maxout Network in Network*. 2015. arXiv: 1511.02583 [cs.CV].

[13]   Ikuro Sato, Hiroki Nishimura, and Kensuke Yokoi. *APAC: Augmented PAttern Classification with Neural Networks*. 2015. arXiv: 1505.03229 [cs.CV].

[14]   Li Wan et al. "Regularization of Neural Networks using DropConnect". In: *Proceedings of the 30th International Conference on Machine Learning (ICML 2013)*. 2013.

[15]   Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*. 2011.

[16]   Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML 2015)*. 2015.

[17]   Geoffrey E. Hinton. *What's wrong with convolutional nets?* MIT Tech TV. 2018. URL: https://techtv.mit.edu/collections/bcs/videos/30698-what-s-wrong-with-convolutional-nets.

[18]   Geoffrey E Hinton. "Learning translation invariant recognition in a massively parallel networks". In: *PARLE Parallel Architectures and Languages Europe*. Springer Berlin Heidelberg, 1987, pp. 1–13. ISBN: 978-3-540-47144-8.

[19]   Geoffrey E Hinton et al. "Improving Neural Networks by Preventing Co-Adaptation of Feature Detectors". In: (2012). arXiv: 1207.0580v1 [cs.NE].

[20]   Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Vol. 9. 2010, pp. 249–256.

[21]   Zhun Zhong et al. *Random Erasing Data Augmentation*. 2017. arXiv: 1708.04896 [cs.CV].

[22]   Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the 6th International Conference on Learning Representations (ICLR 2014)*. 2014.

[23]   Pavel Izmailov et al. "Averaging Weights Leads to Wider Optima and Better Generalization". In: *Proceedings of the 34th Conference on Uncertainty in Artificial Intelligence (UAI 2018)*. 2018.

[24]   Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms". In: (2017). arXiv: 1708.07747 [cs.LG].

[25]   Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.

The code used for all experiments and summary level data is publicly available on GitHub at: https://github.com/AdamByerly/BMCNNwHFCs

A   Appendix

A.1   Digits Disagreed Upon

What follows is the complete set of 93 digits that were predicted correctly by at least one model and incorrectly by at least one model. These in combination with the digits from Figure 7 represent the complete set of digits that were not predicted correctly by all 96 trials. Each image is captioned first by the class label in the test data set associated with the image, then the number of trials that predicted it correctly, and last the index of the digit in the test data. For example, the first image presented below has a class label of 9, 24 trials predicted that correctly, and it exists at index 193 in the MNIST test data.

9	4	2	9	1	5	6	4	8	6	2	4	1	7	3
24	31	6	77	31	33	85	76	59	74	70	30	31	31	88
193	247	321	359	409	412	445	447	582	625	659	708	716	846	938

8	6	4	4	6	9	4	7	5	1	0	2	5	6	8
62	91	18	91	29	18	31	31	20	82	62	62	53	31	89
947	1014	1112	1147	1182	1232	1242	1260	1393	1403	1438	1459	1737	1822	1878

7	1	5	5	4	2	0	1	9	6	2	5	6	9	4
93	31	31	88	31	31	61	31	61	34	44	1	76	58	9
1903	2018	2035	2040	2053	2098	2326	2355	2414	2454	2462	2597	2654	2720	2771

9	1	7	9	6	4	5	6	7	9	3	2	1	3	8
58	80	80	31	1	31	60	5	65	80	88	58	14	31	62
3005	3073	3225	3369	3422	3534	3558	3762	3808	3821	4018	4176	4201	4443	4497

2	1	6	6	3	9	9	4	7	3	3	0	8	1	1
79	77	31	88	29	72	23	86	43	61	91	31	40	31	30
4504	4507	4571	4699	4740	4761	4823	4860	5654	5955	6371	6597	6625	6783	6883

4	4	8	7	1	8	4	7	7	7	2	4	6	6	5
31	31	61	16	65	76	12	91	63	31	87	31	31	31	24
8081	8095	8279	8316	8376	8408	8527	9015	9505	9637	9664	9669	9679	9693	9729

3	2	0
31	61	93
9750	9839	9850