Pruning the fully connected layers of a convolutional neural network “by distinctiveness”

Nicholas Evans

u3095460@anu.edu.au

Abstract. Neural network pruning has been shown to increase classification performance and reduce network size. Until now, most literature on this subject has focused on simple feed forward networks. In this study we prune the fully connected layers of a convolutional neural network (CNN) using Gedeon and Harris' *pruning by distinctiveness* technique and evaluate the effectiveness of this technique by comparing the results to a baseline model. We first train a very capable classifier of handwritten digits on the EMNIST letters dataset, achieving a baseline accuracy of 94.25%. We then apply Gedeon and Harris' technique and observe a small increase in classification accuracy and a significant reduction in network size. We conclude that *pruning by distinctiveness*, when applied to fully connected layers of a CNN, can lead to a smaller fully connected layers and increased classification performance.

Keywords: Artifical neural network, pruning, OCR, EMNIST

1 Introduction

Convolutional neural networks (CNNs) are considered the state-of-the-art in the field of computer vision, having achieved super human results in many image classification tasks (He et al. 2015). With immense computing power on the desktop and the rise of the GPU in deep learning applications, the temptation to create overly large CNNs presents itself. Yet smaller networks can be quicker to converge, make faster predictions and have greater invariance to noisy data (Siestma and Dow 1998, p.I-1, p. I-331). We can look back to early neural networks research for techniques that help us reduce the size of a trained network while maintaining or even improving performance.

Pruning of neural networks is one such technique that has been shown to be effective at reducing network size and improving generalisation. The idea of pruning hidden neurons *by distinctiveness* was first introduced by Gedeon and Harris in their 1992 paper *Network Reduction Techniques* (Gedeon and Harris 1992). This technique computes the angle between the activation vectors of pairs of units and combines units that are deemed to be non-distinct or deletes units that are deemed opposite or *complimentary* (Gedeon and Harris 1992).

In this study we apply *pruning by distinctiveness* to the fully connected layers of a CNN and evaluate the technique in this context. First, we train a CNN to perform handwritten letter classification on the EMNIST letters dataset and obtain a baseline result. We then apply the pruning technique to the CNN and compare the results to the baseline. We find that pruning by distinctiveness leads to a small increase in classification performance and a significant reduction in network size. We further investigate how and when the pruning occurs and address how pruning by distinctiveness can be applied to batch normalization layers. We note some limitations of the pruning process and conclude our discussion with suggestions for future work.

1.1 The dataset

EMNIST is a suite of datasets of handwritten characters that, like the well known MINST dataset of handwritten digits, are derived from the NIST special database 19. EMNIST provides datasets of the same quality and structure of MINST but the content is extended to include uppercase and lowercase letters (Cohen et al. 2017). EMNIST contains 6 datasets, this paper focuses on the EMNIST Letters dataset which has 26 classes, one for each letter in the alphabet.
The motivation for this choice of dataset is to explore the effect of *pruning by distinctiveness o*n a realistic solution to an interesting and practical classification problem. The authors of the dataset have also published benchmark results that we will use to compare our own results against.

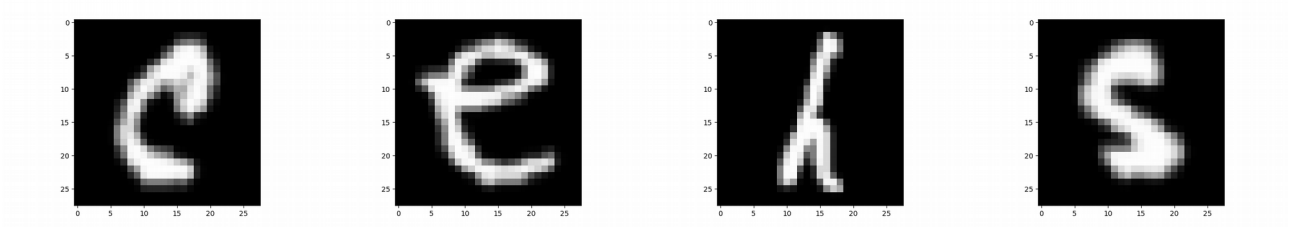


Figure 1: Example images from the EMNIST dataset

The dataset contains one feature per pixel of a 28 x 28 image, flattened into a 784 feature vector. The letters come per-centered on a 28 * 28 grid and a Gaussian filter is also applied to soften the edges (Cohen et al. 2017, p.4). The dataset comes pre-split into a training and testing set (see section 2.5) and the training and testing sets contain an even distribution of all letter classes, illustrated below in Figure 2.

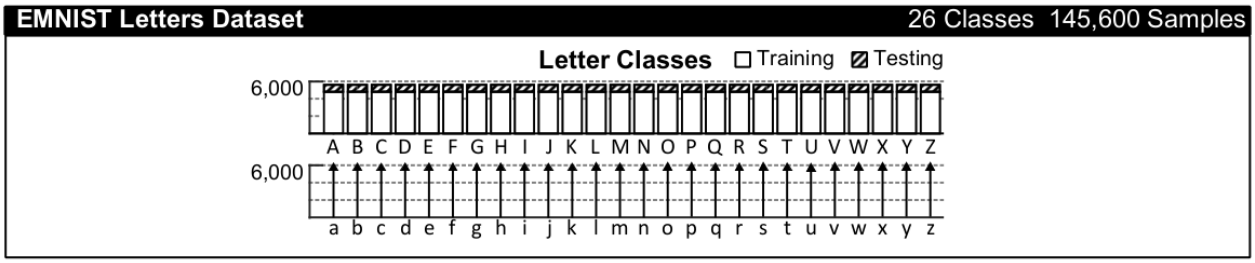


Figure 2: Distribution of classes across the training and testing set of EMNIST (Cohen et al. 2017, p.4)

2 Method

2.1 Network design

The main goal of this paper is to evaluate the effectiveness of pruning by distinctiveness on the fully connected layers of a CNN. Given this fact, we are not concerned about achieving optimal classification accuracy. We do, however, want to conduct our experiments on a realistic network architecture that performs the task well. Therefore our method was to start with an architecture known to perform well on similar classifications tasks and adapt this architecture to perform well on EMNIST. We use a separate validation set for model validation but do not perform an exhaustive hyper-parameter search.

We base our initial network design loosely on LeCun et al's LeNet 5 implementation, that achieved greater than 99% accuracy on MNIST dataset (LeCun et al. 1998, p.12). Given that our input images are of identical size to the MNIST dataset, we keep 2 convolutional layers and a kernel size of 5. Classification of letters is an inherently more complex task than classification of digits, therefore we look to increase the number of features in each convolutional layer and the number of hidden nodes in our output layer. We use a single hidden layer in the fully connected portion of the network as this simplifies the pruning step. Had our goal been to maximize classification performance we would have looked to use additional fully connected layers.

We perform validation, using this initial network design, looking for values for the following parameters:

- Convolutional layer 1 features – min 10, max 50
- Convolutional layer 2 features – min 20, max 100
- fully connected Hidden layer size min 100, max 1000

The results of the hyper parameter search are shown in table 5. Details of the validation set are given in section 2.5

Table 1. Hyper-parameter search for number of features

# features Convolutional layer 1	#features Convolutional layer 2	Fully connected layer hidden neurons	Average % correct
10	20	500	94.07
20	**40**	**500**	**94.21**
40	80	500	94.20
50	100	500	94.20

Table 2. Hyper-parameter search for number of hidden units

# features Convolutional layer 1	#features Convolutional layer 2	Fully connected layer hidden neurons	Average % correct
20	**40**	**250**	**94.3**
20	40	500	94.3
20	40	1000	94.1

Based on the results of our basic hyper parameter search we use 20 and 40 as the number of features in convolutional layers 1 and 2, respectively and 250 for the (starting) hidden layer size.
We optimise our model using mini batch stochastic gradient decent and use negative log likelihood as the loss function.

Rectified linear units are used as the activation function on all layers. We add batch normilsation layers directly after each convolutional layer and directly before the ReLU in the fully connected layer.

Batch normalization works by normalising inputs to each mini-batch. It allows higher learning rates and acts as a regularizer, reducing or eliminating the need for dropout (Ioffe and Szegedy, 2015 p.1). The addition of the batch normalisation in the fully connected part of the network requires us to do some extra work as part of the pruning process. This is discussed further in section 2.2.

2.2 Pruning by distinctiveness

Our technique is based on the paper *Network Reduction Techniques* (Gedeon and Harris 1992) and hence the approach is very similar to what is described in that paper.

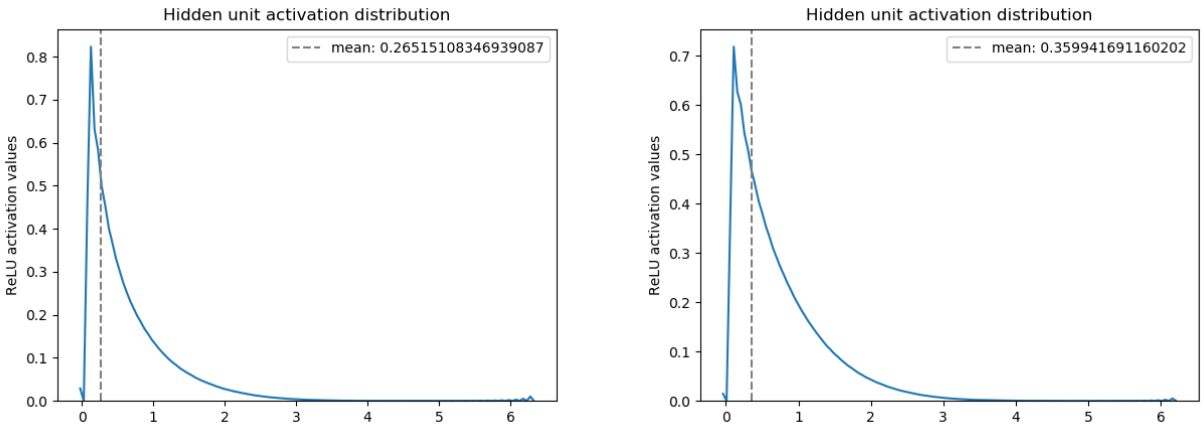
Every 2 epochs we obtain an activation vector for each hidden unit in the fully connected layer. The activation vector contains that unit's activation value for each training example. Table 4 illustrates how these vectors are formed. Note, we store these activations across mini batches so that each calculation is done against vectors of every training example in the epoch. We examine all possible pairs of these vectors, and calculate the angle between each pair.

Table 3. Example of vectors of hidden unit activations as per (Gedeon and Harris 1992)

	Unit 1	...	Unit i
Pattern 1	Hidden unit 1's activation for pattern 1	...	Hidden unit i's activation for pattern 1
:		...	...
Pattern n	Hidden unit 1's activation for pattern n	...	Hidden unit i's activation for pattern n

In Gedeon and Harris' paper the activations are constrained to the range 0 to 1 by the sigmoid activation function and the vector angle calculations are normalised to 0.5, 0.5, the midpoint of this range. This yields an angular range of 0-180° rather than 0-90°(Gedeon and Harris 1992, p.5). If the angle between the two vectors is less than then a defined threshold threshold (initially 15°), one of the units is removed and the weights and bias are added to the other unit in the pair. If the angle is greater than 165° then the units are deemed complimentary and they are both removed (Gedeon and Harris 1992, p.5).

Our implementation differs from Gedeon and Harris since we use ReLUs as the activation function. This leaves us with the problem of finding a suitable mid-point from which to measure the angles. To help us decide which point to measure from, we plot the distribution of the activations at each pruning step (every 2 epochs). Example distributions at epochs 2 and 12 are shown in Figure 3.



Epoch 2 Epoch 12

Figure 3. Distribution of the activations at epochs 2 & 12

The distributions of the activations are skewed heavily to the left, since the ReLU will give zero for any negative input. As training progresses we see the mean increasing and moving closer to the center of mass of the distribution. We use the mean of the activations as the measuring point as this can be calculated efficiently and gives acceptable results. Our approach differs here to Gedeon and Harris' since the location from which we measure the angle changes slightly each time pruning is performed.

Since we are using batch normalization layers in our fully connected layers we extend the concept of pruning by distinctiveness to these layers. We still prune based on the angle between activation vectors of the hidden units but we now also prune the relevant weights and bias from the batch normalization units, adding the weight and bias from item being pruned to the other in the pair.

We experiment with the pruning threshold set to 15°, 20°, 25° & 30° and compare the classification performance and network size at each threshold value. These results are presented in section 3.

2.3 Preprocessing

As noted in section 1.1, the dataset comes preprocessed to a certain extent. Batch normalization is used, so in some ways we move preprocessing inside the structure of the network. No additional preprocessing of the data is performed as part of this study.

2.4 Stopping Criteria

We cease training when the loss on the test set increases. We allow one extra epoch after pruning occurs to allow the network to "recover" from the pruning, however in practice we don not see this condition often triggered.

2.5 Validation, training and test sets

The EMNIST data set comes pre-split into a training and testing set which is described below in Table 2. We create an additional validation set to conduct a hyper parameter search. The validation set is formed by taking a random stratified sample from EMNIST's training portion. We do this to ensure that the reported results are on data that the network has not seen before. The data set is deemed too large to perform K fold cross validation in a reasonable amount of time. The split of the data that was used for model validation is shown below in Table 1.

Table 4. Split of training and validation data for hyper parameter search

Training examples	Validation Examples	Total
104000	20800	124800

For training the final model we retrain using the entire EMNIST training set to ensure that the number of training and testing examples is the same as the Cohen et al. benchmark paper. The final split of the data that is used for training is described in Table 2.

Table 5. Split of training and validation data for training and testing final model

Training examples	Testing Examples	Total
124800	20800	145600

3 Results and Discussion

A final result of 94.25% was achieved after training our baseline model for an average of 21 epochs. This is a respectable result and easily better than the 85.15% published in the EMNIST benchmark paper of Cohen et al. (Cohen et al. 2017, p.7). This is not surprising given that CNNs are known to excel at image classification tasks. We restate here that the primary goal of this paper is not to maximize the classification accuracy but rather to explore the effect of pruning by distinctiveness on a realistic and capable model. We therefore attempt no further optimization of this result and rather use it for comparison against the results of the pruned models at various pruning thresholds.

The results of the experiment, with angles set to 15°, 20°, 25° and 30° are presented in table 6. Each value represents the average of 10 trials with different random seeds used in each trial.

Table 6. Final Results

Prune threshold	Average % correct	Starting size hidden layer	Average size, hidden layer	Average epochs
Baseline – no pruning	94.25	250	250	21
15	94.38	250	234	25
20	94.48	250	225	24
25	**94.52**	**250**	**220**	**27**
30	94.37	250	216	19

The best result of 94.52% accuracy is achieved when the pruning threshold was set to 25 degrees. Here we see an increase in performance compared to the baseline result of 94.25% and a 12% reduction in the size of the fully connected layer. As the pruning threshold increases, we can see more hidden units are pruned from the network. Performance increases up to the 25 degree threshold and then starts to decrease at 30 degrees.

It is possible that this technique simply represents a form of "online hyper-paramter search" for the optimum hidden layer size. We now look to answer the question of whether the pruning simply finds a good size for the hidden layer or whether it is able to find a better minima by including more nodes in the early part of the training process. To explore this further we conduct an additional 10 trials, setting the hidden layer size to 220, the average size of our best pruning result.

Table 7. Results with no pruning and the same size

Prune threshold	Average % correct	Starting size hidden layer	Average epochs
Baseline – no pruning	94.37	220	22

The statically sized network achieves accuracy of 94.37%, which is not as good as the pruned network. It appears there is some utility in having more nodes in the network early on in the training. The pruned network is potentially finding minima that would otherwise be missed. This statically sized result is also slightly better than the baseline result of 94.25% so we can see that the pruning alsovdoes a good job of finding an optimised hidden layer size via it's "online hyper-paramter" search.

We observe that the pruning step takes a long time to run due to the calculation of the vector angles for every possible pair of units. As the number of hidden neurons grows, the time required to compute the angles grows (at best) polynomially, which renders this technique quite slow for large numbers of hidden neurons. In our work we were unable to utilise the GPU for these computations so the calculation time was significant when compare to the overall training process. Previous work has suggested that network pruning can improve training convergence time (Hagiwara 1990, p.1), this is unlikely to be the case using our implementation due to the time complexity of the angle calculation.

Note the calculated angles are sensitive to the point at which we choose to measure. While we do out best to choose a valid middle point, it is difficult to compare the results achieved at these thresholds with other studies unless the angle is measured from the mean of the activations.

4 Conclusion and future work.

In this paper we set out to examine whether or not pruning by distinctiveness is effective when applied to the fully connected layers of a convolutional neural network. In our experiments, we have observed improved classification performance on the EMNIST data set and a significantly smaller network when compared to our baseline model. We can conclude that pruning by distinctiveness is indeed effective when applied to the fully connected layers of a CNN. Furthermore, we have shown that the pruning technique can also be applied to fully connected batch normalisation layers and we have presented a simple method for achieving this.

We briefly address the question of whether or not pruning is effectively an "online hyper parameter search". Our results suggest that including more nodes early on in the training process enables the network to find a better overall minima as well as finding a good value for the hidden layer size.

We observe that the pruning process takes a relatively long time as the number of hidden units grows large. Future work could look to parallelise this calculation on a GPU.

We also observe that much of the pruning takes place early on in the training process, in the first few pruning cycles. One idea to combat this phenomena is to add a gain term to the pruning threshold, effectively increasing the pruning threshold as training progresses. This would be a simple way to encourage more pruning later in the training process.

Further work in the area of network pruning could examine the noise resistance effects of pruning that have been suggested by Siestma and Dow (1988, p. I-331). After randomly adding noise to the dataset images one could compare the classification performance of a pruned vs non-pruned CNN.

References

1. Cohen, G., Afshar, S., Tapson, J. and van Schaik, A., 2017. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*.

2. He, K., Zhang, X., Ren, S. and Sun, J., 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026-1034).

3. Gedeon, T.D., Harris, D.: Progressive Image Compression, Proceedings International Joint Conference on Neural Networks, vol. 4, pp. 403-407, Baltimore, 1992

4. Hagiwara, M., 1990, June. Novel backpropagation algorithm for reduction of hidden units and acceleration of convergence using artificial selection. In *Neural Networks, 1990., 1990 IJCNN International Joint Conference on* (pp. 625-630). IEEE.

5. LeCun, Y., Bottou, L., Bengio, Y. and Haffner, P., 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), pp.2278-2324.

6. Sietsma, J. and Dow, R.J., 1988, July. Neural net pruning-why and how. In *IEEE international conference on neural networks* (Vol. 1, pp. 325-333). IEEE San Diego.

7. Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.