COMP5329: Assignment 2

Daryl McClure (dmcc6359 - 470536146) / Alister Palmer (apal9344 - 470538519)

1. Introduction

In this project, the aim is to achieve the best possible classification by training on a provided training dataset of 37882, 128 pixel by 128 pixel, grayscale images that are transformed to arrays with values ranging between 0-255. Each image contains a seemingly digitally generated single centered character out of context with almost no visual noise. The dataset contains black characters on a white background with varying height and width, belonging to one 62 classes including digits 0-9 and the alphabet in both upper-case and lower-case. Each character has been stripped of context, including other characters of a word or sentence, relative size of neighbouring characters or a plane that may give some hint towards the case of a character. Thus, for this distilled challenge classifiers will be relying primarily on the features of the character itself to perform classification. The characters are sourced from a wide variety of fonts that includes italicised, elaborate and highly stylized examples as exhibited in Figure 1 below. Inspection of the dataset also reveals some instances of noise in the dataset in the form of, for example, upper-case letters being mislabeled as lower-case.

Figure 1: Selection of Images from the dataset

Regular | Italics | Elaborate | Stylized | Mislabeled as lower ‘a’

Recent technological advances in data transfer speeds and memory have allowed for structured and unstructured data to be created and disseminated at a previously unfathomable pace and scale. Thus, it has becoming increasingly important to be able to sort and analyse this constant stream of information which has spawned the development of an array of advanced classification techniques within deep learning aiming to redefine the state of the art. As demonstrated by the dominance of Convolutional Neural Networks (CNNs) on the vast ImageNet dataset, beginning with AlexNet improving on the previous winner’s accuracy metric by over 10%, CNNs are widely considered to be state of the art for computer vision problems. Specifically, the characters provided by this dataset present a character-level classification problem that is preliminary element of many natural language processing applications.

The group will draw on the findings of recently published research to inform the architectures, techniques and hyperparameter tuning decisions to maximise accuracy.

2. Techniques

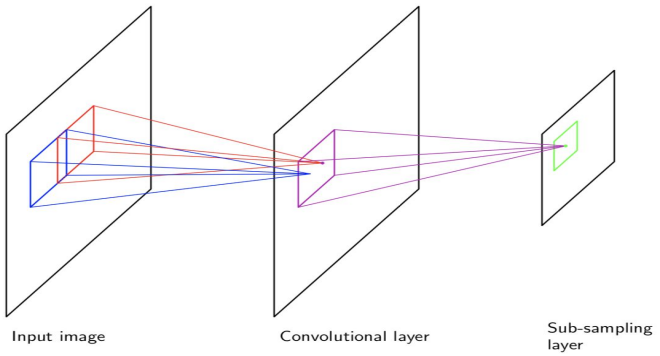
2.1 Cloud Computing for Accelerated Model Training

Attempting to run our CNN experiments on a standard CPU based resource proved to be an unacceptably time consuming process. For example, running a 9 layer benchmark CNN (Model D) on a 2017 Macbook Pro (2.3GHz Core i5 CPU, 16GB DDR RAM) requires a runtime of 1 hour 2 minutes for a single epoch. Without a suitably powerful GPU readily available, the group decided to utilise cloud computing to expedite experimentation. We created an p2-xlarge GPU instance on Amazon EC2 consisting of a GPU with 12GB memory, 4 vCPUs and 61GB of memory. The AWS Deep Learning AMI was adopted and comes pre-installed with Python3.6, Tensorflow and Keras. Other required packages were installed to the instance and experiments conducted using Jupyter notebooks. Within this environment the same benchmark CNN ran in 122 seconds.

2.2 Convolutional Layer

First introduced by LeCun et al in 1989, the addition of convolutional layers to a neural network allow it to recognise the identity of a feature that is invariant to transformations of inputs such as scaling, some rotation and deformation. This in turn leads to a more robust feature recognition capability on unseen examples being attained by the CNN with less training (6). As depicted in Figure 2, by scanning a relatively small filter across an input image the information can be represented with less parameters. This approach is much more successful with images as it harnesses the fact that ‘nearby pixels are more strongly correlated than more distant pixels’ (Bishop) by extracting local features in early layers.

Figure 2: Example of part of a Convolutional Neural Network (7)



These numerous feature maps, each with its own set of weights and bias parameters, are subsequently combined to detect complex features. The convolutional layer, or multiple layers in the

case of VGGNet, is followed by a subsampling layer, effectively reducing dimensions while retaining some information. After this convolutional stage, the output can be fed into a fully connected neural network with a reduced spatial resolution. The resulting CNN can be trained with backpropagation to minimise the chosen loss function, for example cross entropy, while using far fewer independent parameters than directly importing into a fully connected neural network.

2.3 Activation Functions

In recent years the Rectified Linear Unit (ReLU) has become the standard activation function used in large CNNs. Formal studies by Krizhevsky (1) and anecdotal experience of the group have found it to be more performant than Sigmoid or Tanh. The commonly cited reasons are attributed to its linear, non-saturating functional form coupled with its evaluation requiring only inexpensive operations.

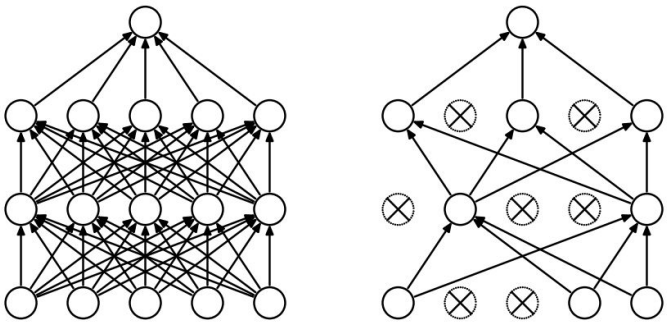
ReLU units are not, however, without drawbacks. They are not strictly zero centred like the Tanh function. More problematically, they suffer from fragility during training in situations where the learning rate is too high or large errors gradients are back propagated leading to weight updates that can deactivate the neuron. Once deactivated, the neuron irreversibly “dies” and effectively falls off the data manifold.

While Leaky ReLU has been posited as a means to fix the problem of dying ReLUand some success has been reported, results in general have not been consistent. In light of this the group chose to apply ReLU as the chosen hidden layer activation function.

2.4 Drop Out

Deep Learning architectures typically contain many hidden layers. On one on hand this admits such networks with the capability to represent extremely complex relationships but also leaves them vulnerable, particularly when data is limited, to over fitting on noise within the data. This is not an uncommon problem in machine learning and many methods have been drawn upon from computational statistics, such as L1 and L2 regularisation, to reduce the problem.

Figure 3: Pictorial Representation of Dropout in A Fully Connected Network



(a) Standard Neural Net (b) After applying dropout.

Another approach to regularisation is using ensembles that average predictions generated across different models involving ideally different datasets and different possible hyperparameter settings. For large networks training on large datasets this is both difficult and time consuming even if sufficient data were available.

Dropout is an approach that seeks to simultaneously address both these challenges at once. It avoids overfit and provides an indirect means of theoretically averaging over 2^n different models. The approach can be summarised as follows:

- During a given batch a set of neurons are selected at random, typically using a sample from a suitably defined binomial distribution, with probability *p* and their values are set to zero.
- The dropped neurons do not contribute to the batch training phase in either the subsequent feed-forward or back propagation phases.

In doing so, training is effectively performed on a new network. This can be thought of as giving rise to a form of stochastic regularisation. In Srivastava's original paper (2) the motivation is inspired from sexual reproduction where minor random mutations are introduced that help to reduce the propensity for complex co-adaptations to occur. This avoids overfit and creates a more robust network with an improved likelihood for better generalisation.

During the training phase a neuron is kept with a probability $q = 1 - p$. The original article proposes this ensemble scaling action is also repeated during the testing phase. This approach is generally referred to as *direct* Dropout. A more common approach has since evolved where the scaling is shifted to the training phase whereby the outcome from a given neuron is scaled by $1/q$ to recover the, on average, outcome had no neurons been dropped and in turn during the test phase no scaling is needed. This approach is referred to as *inverted* Dropout.

Dropout has a number of advantages over more conventional approaches such as L2 regularisation. Two of the most commonly noted are given below:

- Dropout is scale free and does not penalise large weights should they be needed.
- Dropout is scale invariant and is unaffected by different scaling that may result across different layers.

Despite being simple in concept and implementation Dropout has been proven as an effective a robust method of controlling overfit and as such has become a standard inclusion in CNN since the advent of AlexNet.

2.5 Batch Normalization

At a basic level Batch Normalisation could be construed as a simple extension of feature scaling, using mean normalisation, to internal hidden layers of the network. More technically, it seeks to address the problem known as 'internal covariate shift' where the distributional properties of the inputs to a given layer are affected by changes, such as the weight updates, that occur in prior layers. Moreover chaining effect creates a correlated co-dependence between layers. Just as whitening, i.e. zero mean and variance, and decorrelation of input feature data is known to lead to faster network convergence, internal covariate shift has the opposite effect adding complication to the weight learning process during backpropagation.

Batch Normalisation seeks to address this. In its simplest form the output of each neuron, and thus dimension of that layer, is normalised to have a distribution with zero mean and unit variance.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - E\left[x^{(k)}\right]}{\sqrt{Var\left[x^{(k)}\right]}} \qquad (1)$$

Normalising the output however constrains the degree of nonlinearity the subsequent layer can represent through its activation function. For example, a sigmoid would be constrained to effectively operate with linear behaviour. In light of this the the normalised values for each dimension are re-scaled by $\gamma^{(k)}$ and shifted $\beta^{(k)}$**.** Both of these parameters are learned along with the original model parameters thereby restoring the representational power of the network.

Figure 4, taken from Ioffe's original article (3), outlines the final algorithm for mini-batch application. The inclusion of ε in the normalisation step is simply a small constant (typically ~1E-08) to avoid numerical instability.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Figure 4: Training Phase Batch Normalisation

Once the network has been trained the neuron layers are normalised using the population, rather than mini batch, statistics during the inference phase. This is outlined in Algorithm 2 of the original paper but for brevity we simply outline below the normalisation expression used at inference.

$$\widehat{x} = \frac{x - E[x]}{\sqrt{Var[x] + \varepsilon}} \tag{2}$$

$$E[x] \leftarrow E_B\left[\mu_B\right] \tag{3}$$

$$Var[x] \leftarrow \frac{m}{m-1} E_B\left[\sigma_B^2\right] \tag{4}$$

2.6 Cross Entropy Loss

While Mean Square Error, or L2 loss, is commonplace in machine learning it is best suited to regression problems while Cross-entropy loss, or log loss, is better suited to classification problems where probabilities of being in a given class are reported. In light of this Cross Entropy Loss was the chosen loss function for error backpropagation. For multiclass problems such as this project the generalised form of the binary classification, typical in Logistic Regressions, is used and takes the following form:

$$J = -\sum_{i=1}^{K} y_o^{(i)} \, ln\left(p_o^{(i)}\right) \tag{5}$$

Where K represents the number of classes, $y_o^{(i)}$ is a binary indicator (0,1) if class label i is the correct classification for observation o and $p_o^{(i)}$ is the predicted probability observation o is of class i.

2.7 SoftMax

Softmax regression is a generalisation of the binary Logistic Regression case $y^{(i)} \in \{0, 1\}$ to the multiple class case $y^{(i)} \in \{1, \ldots, K\}$ and is paired with the Cross Entropy generalisation outlined above.

Softmax is applied to the activated output of the final hidden layer of the fully connected network and essentially normalises the activated output of each neuron such that they form a discrete probability distribution:

$$y^{(i)} = \frac{exp\left(A^{(i)}\right)}{\sum_{i=i}^{K} exp\left(A^{(i)}\right)} \quad (6)$$

2.8 Adam SGD Optimisation

Adaptive Moment Estimation (Adam) is one of a family of methods that compute adaptive learning rates for each parameter. Its authors describe it as combining the advantages of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp).

AdaGrad modifies the learning rate for a given parameter based on the the accumulated squares of the gradients while RMSProp refines this overly aggressive reduction in learning rate with a decaying average approach. Like AdaGrad and RMSProp, Adam stores an exponentially decaying average of squared gradients v_t but in addition stores the equivalent exponentially decaying average the gradient m_t like Momentum:

$$m_t = \beta_1 m_{t-1}(1-\beta_1) + g_t \quad (7)$$
$$v_t = \beta_2 v_{t-1}(1-\beta_2) + g_t^2$$
$$g_{t,i} = \nabla_\theta J\left(\theta_{t,i}\right)$$

To avoid zero bias issues these estimates are corrected as follows to generate the final update:

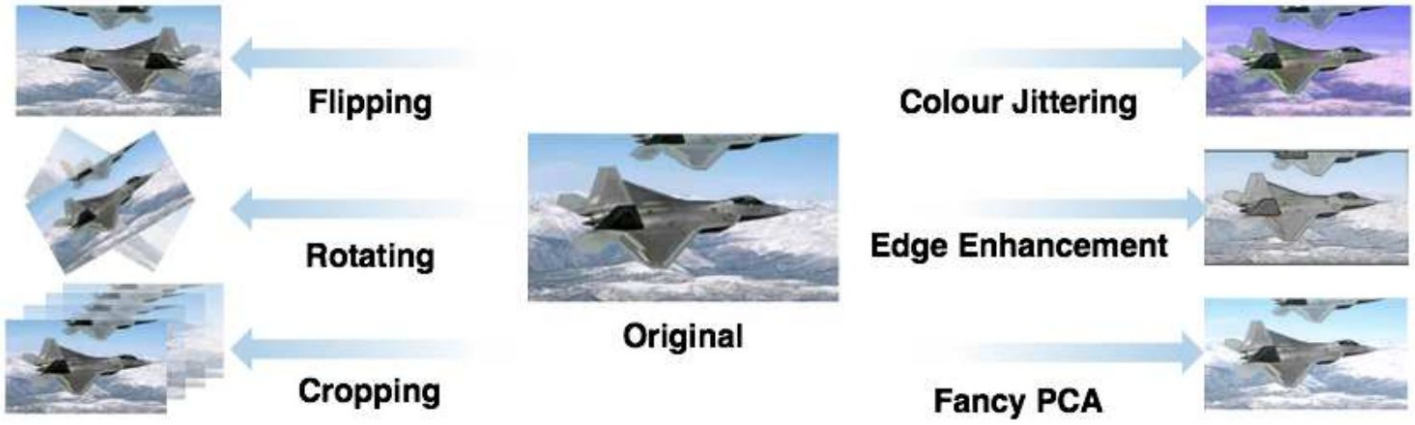
$$\widehat{m}_t = \frac{m_t}{1-\beta_1^t} \quad (8)$$
$$\widehat{v}_t = \frac{v_t}{1-\beta_2^t}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\widehat{v}_t}+\varepsilon}\widehat{m}_t$$

The authors propose default values of 0.9 for β_1 and 0.999 for β_2, while ε is set to 1E-08 to avoid division by zero. A useful summary and comparison of alternative adaptive learning rates can be found in Ruder (4)

2.9 Data Augmentation

It is generally accepted that while traditional machine learning methods will produce a superior result on smaller amounts of data, as the size of labelled datasets grow, the accuracy of deep learning methods such as neural networks will continue to improve and can achieve a level accuracy beyond the capability of alternative methods. Unfortunately, there is genuine cost to produce large amounts of labelled data suitable to train neural networks and in the absence of sufficient training data practitioners will seek to artificially modify existing data to substitute more real data with the aim of achieving higher accuracy.

Figure 5: Examples of Data Augmentation Techniques (5)



In the context of images, widespread techniques to achieve data augmentation by boosting the amount of trainable data include simple transformations such as flipping, rotating and cropping as shown on the left of Figure 5. Further, photometric methods that alter colouring as shown on the right of Figure 5 have, in practice assisted in generating diverse data. Data augmentation also encompasses techniques aimed to aid the training of models by highlight edges to help CNNs detect features and advanced techniques such as Principal Component Analysis and ZCA whitening.

2.10 Transfer Learning

Transfer learning is a design methodology used in machine learning where a model developed on one problem is re-purposed for a second related task. Image classification, along with Natural Language Processing, is particularly well suited to transfer learning where complex convolution networks, such as VGG or Google Inception, pre-trained on millions of images require vast compute and time resources. Once trained these networks can, however, be exploited for problems where only limited data and or modest computational resources are available .

The power of convolutional networks rests in their low to high level feature extraction. Provided the domains are similar, the pre-trained convolutional layers can be directly used to transform the data associated with the new problem to a feature space amenable to a simple classifier trained to the new problem. The original more potentially more complex classifier can be replaced using a simple SVM or a relatively shallow fully connected Neural Network that can be routinely trained on conventional desktop hardware in a reasonable time frame. If needed, and where further resources are available, refinement of the pre-trained network is also possible. Layers of the CNN related to low and medium level features would be left intact while the final layers related to high level features are included for fine tuning as part of the new classifier training phase. Performance in this particular application can be found in Section 3.7 and fell short of a fully-trained bespoke network although runtime was significantly reduced 25 minutes on a local machine.

In summary Transfer Learning represents a rich vein to exploit when domains are similar and computational resources or data are scant.

3. Experiments

Based on consideration of the general advancements in image classification and more specifically research relating to character-level classification, the group expects to define and tune convolutional neural networks to maximise classification accuracy.

As discussed in the introduction, this dataset seems to provide a relatively limited amount of detail to inform a classifier.The group decided to try and improve upon the Feedforward NN’s accuracy by exploring Convolutional Neural Networks (CNNs). In considering modern CNN concepts such as VGGNet, ResNet and GoogLeNet it intuitively appeared this problem may not hold sufficient information to necessitate a deeper or more complex network infrastructure. Thus, the residual block transmission mechanism of ResNet or inception module featured in GoogLeNet were initially avoided in favour of the streamlined VGGNet approach, which would facilitate a reasonable runtime.

Figure 6: Experiment Model Architectures

Model 0 FFNN	**Model A** VGGNet-like	**Model B/C** Less Width	**Model D** Less FC Width	**Model E** Deeper Conv	**Model F** Deeper FC	**Model G** Single Convs
Input (128 x 128 Grayscale Image)						
FC4096 Batchnorm Drpt 0.5 FC256	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25
	Max-Pool (pool size 2 x 2)					
FC256 Batchnorm Drpt 0.5	Conv3-64 Batchnorm Drpt 0.25 Conv3-64 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25
	Max-Pool (pool size 2 x 2)					
FC62 Batchnorm Drpt 0.5	Conv3-128 Batchnorm Conv3-128 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	Conv3-32 Batchnorm Drpt 0.25
	Max-Pool (pool size 2 x 2)					
	FC4096 Batchnorm Drpt 0.5 FC4096 Batchnorm Drpt 0.5	FC1024 Batchnorm Drpt 0.5 FC1024 Batchnorm Drpt 0.5	FC512 Batchnorm Drpt 0.5 FC512 Batchnorm Drpt 0.5	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25	FC512 Batchnorm Drpt 0.5 FC512 Batchnorm Drpt 0.5 FC512 Batchnorm Drpt 0.5 FC62 Batchnorm Drpt 0.5	Conv3-32 Batchnorm Drpt 0.25 Conv3-32 Batchnorm Drpt 0.25
				FC512 Batchnorm Drpt 0.5 FC512 Batchnorm Drpt 0.5		FC512 Batchnorm Drpt 0.5 FC512 Batchnorm Drpt 0.5
FC62 - Batchnorm - Soft-max						
Val Acc: 86.0%	**89.7%**	**B:90.0% C:89.9%**	**90.79%**	**85.0%**	**89.0%**	**89.9%**
20 epochs	8 epochs	**B:**15 **C:**12	22 epochs	20 epochs	30 epochs	84 epochs
97s/epoch	297s/epoch	115s/epoch	110s/epoch	110s/epoch	114s/epoch	53s/epoch
Traintime: 32mins	39mins	**B:**29m **C:**23m	40mins	37mins	57mins	74mins
Params: 71M	120M	7.5M	3.5M	955K	3.8M	600K

3.1 Experiment Constants:

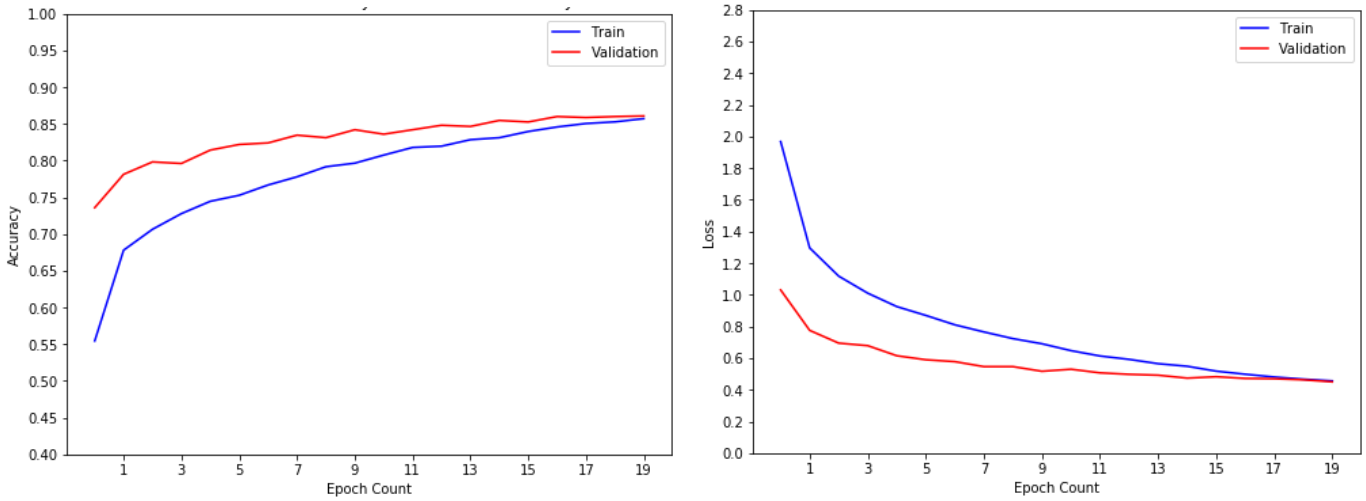
- Supplied training set split into 34,093 (90%) training set and 3,789 (10%) validation set and the provided 6,262 sample validation set used as holdout test set.
- Experiments conducted using Python 3.6, Tensorflow 1.8, Keras 2 on an AWS Deep Learning AMI (technical specifications in section 2.1).
- All experiments used Adam optimizer and categorical Cross Entropy loss, with a batch size of 32.
- Baseline preprocessing was converting values of input arrays to float32 and divided by 255 to result in values between 0 and 1.
- Dropout of 0.25 is applied after each max-pool layer and 0.5 after each fully-connected layer.
- The output layer consisted of a fully-connected 62 neuron layer with Softmax activation.

3.2 Model 0: Feedforward Neural Network (86% validation accuracy at 20 epochs, 97s epochs)

Firstly, to establish a baseline accuracy, the group trained a feedforward neural network (NN) topology. Each fully connected layer contains ¼ neurons of the previous layer (or input image)with 50% dropout and batch normalization (momentum=0.99) applied after each layer. This model managed to achieve validation accuracy of 86% after 20 epochs and saw signs of convergence (Figure 7). The holdout test data also confirmed 86% accuracy.

This result was surprisingly high, but considering the images for classification are centred and devoid of visual noise many of the impediments to a neural network achieving good accuracy are not present in the dataset. Also, the validation accuracy/loss in early epochs appears consistently higher than training metrics. It is thought that given the limited number of samples, the validation set may not be entirely representative of the broader dataset, this could be addressed by applying stratified folds.

Figure 7: Model 0 Accuracy & Loss

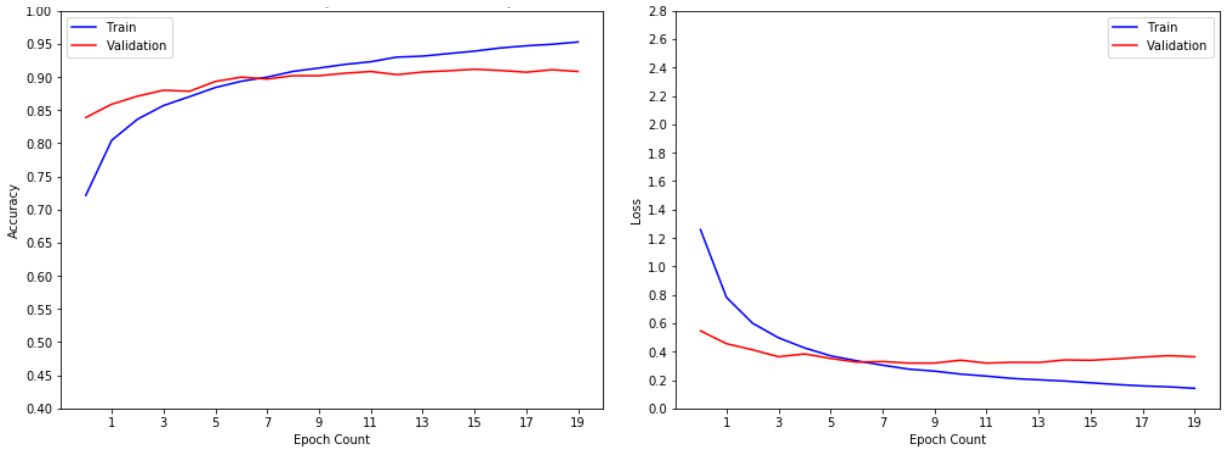


3.3 Model A: VGGNet-like (89.7% validation accuracy at 8 epochs, 299sec per epoch)

It was decided to adapt the general VGGNet structure to take advantage of the multiple 3x3 convolutional layers between pooling layers to achieve desired receptive fields while minimising parameters. The group adapted this framework to smaller dimensions of the given dataset by using less convolutional layers but maintained the 2 fully-connected layers of 4096 neurons. Also, the group added batch normalisation, a technique that was developed after VGGNet but has since been used extensively to good effect. Among practitioners there is some disagreement about whether to include batch normalization before or after activation. The group has chosen to place batch normalization layers before activation in all cases.

Achieved an accuracy of 89.7% at 8 epochs before overfitting, significantly better than Model 0. However, inspection of the accuracy plotted in Figure 8, this model seems to have indications of underfit. The runtime per epoch is rather long at 299 seconds, VGGNet was developed for the imagenet dataset but our simpler dataset appears less complex and perhaps a similar result could be achieved with a less sophisticated model.

Figure 8: Model A Accuracy & Loss



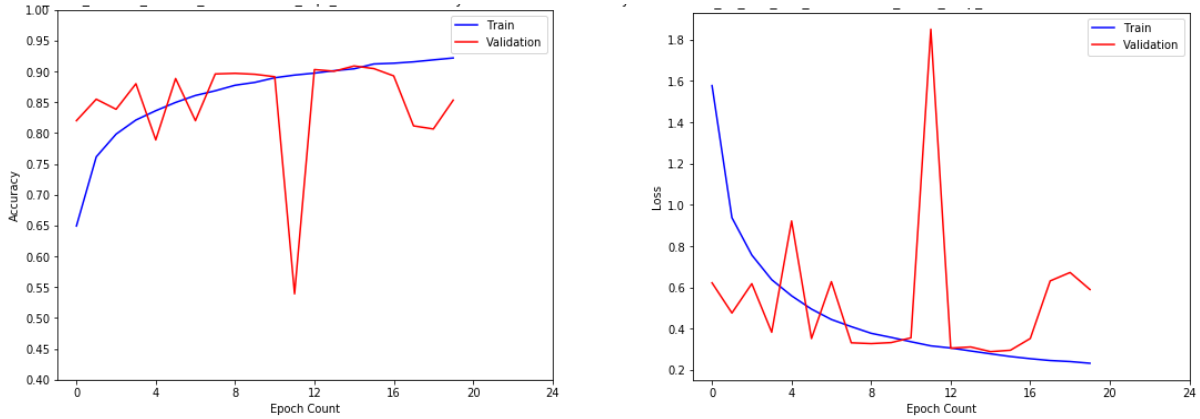
3.4 Model B: Reduced Layer Width (90% validation accuracy at 15 epochs, 115secs per epoch)

The group sought to establish whether a network with less ‘width’, for example less filters in convolutional layers and less neurons in fully-connected layers, would be able to achieve a similar results as Model A. This model retained the depth as Model A but all convolutional layers were reduced to 32 filters and fully-connected layers limited to 1024 neurons.

Figure 9 shows that Model B at 90% validation accuracy after 15 epochs surpassed Model A despite its streamlined architecture. Holdout test set accuracy fell slightly short at 88.3%. Although epochs ran faster at only 115 secs, the increase in epochs before overfit made for a negligible reduction in overall training time. Noticeably, it was difficult to analyse the training because the validation accuracy

and loss were very erratic between epochs and the group aimed to rectify this with subsequent experiments.

Figure 9: Model B Accuracy & Loss

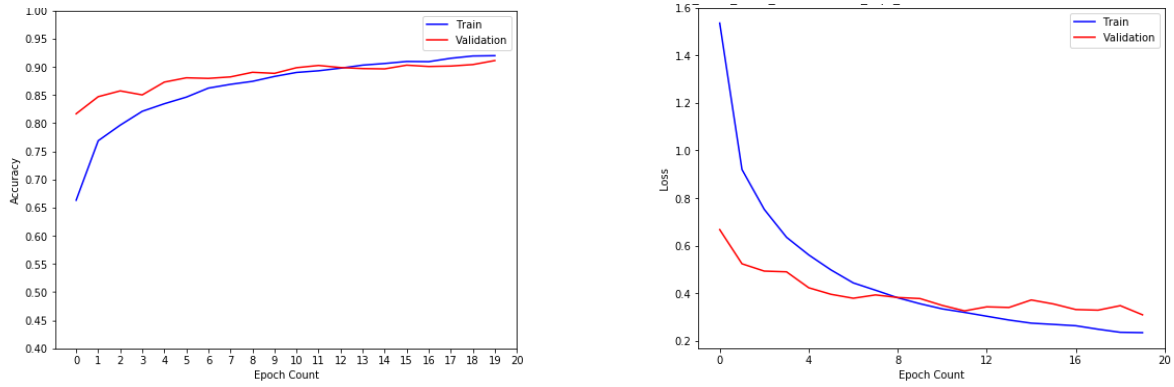


3.5 Model C: Reduce BN Momentum (89.9% validation accuracy at 12 epochs, 115secs per epoch)

By adjusting the momentum hyperparameter within the batch normalization layers, the group was able to generate a smoother progression.

This resulted in 89.9% validation accuracy at 12 epochs before overfitting. The holdout test accuracy was higher still at 90.7%. Thus, given the success of simplifying the network between Model A and Model C. the higher holdout test accuracy seemed to indicate there could be further accuracy gains to achieved, so the group decided to try further simplification and gauge the impacts.

Figure 10: Model C Accuracy & Loss



3.6 Model D: FC Layers only 512 (90.79% validation accuracy at 22 epochs, 110secs per epoch)

To further simplify the model the group reduced the fully connected layers to 512 neurons each, which produced a marginal reduction in epoch runtime. For Model D(a), an additional simple preprocessing step was taken, transforming values to around a zero mean in a range of -1 to 1 as it can positively impact training results (Stanford). Comparison of Model D and D(a) in Figure 11 and Figure 12 respectively suggest that on the size and complexity of our network and dataset this provided no meaningful improvement in efficiency.

This model, despite its reduction in trainable parameters to 3.5million (Appendix 1.4), produced the best validation accuracy of 90.79% at 22 epochs that was confirmed by the holdout testset result and adopted as the group's final prediction model.

Figure 11: Model D 20 epochs

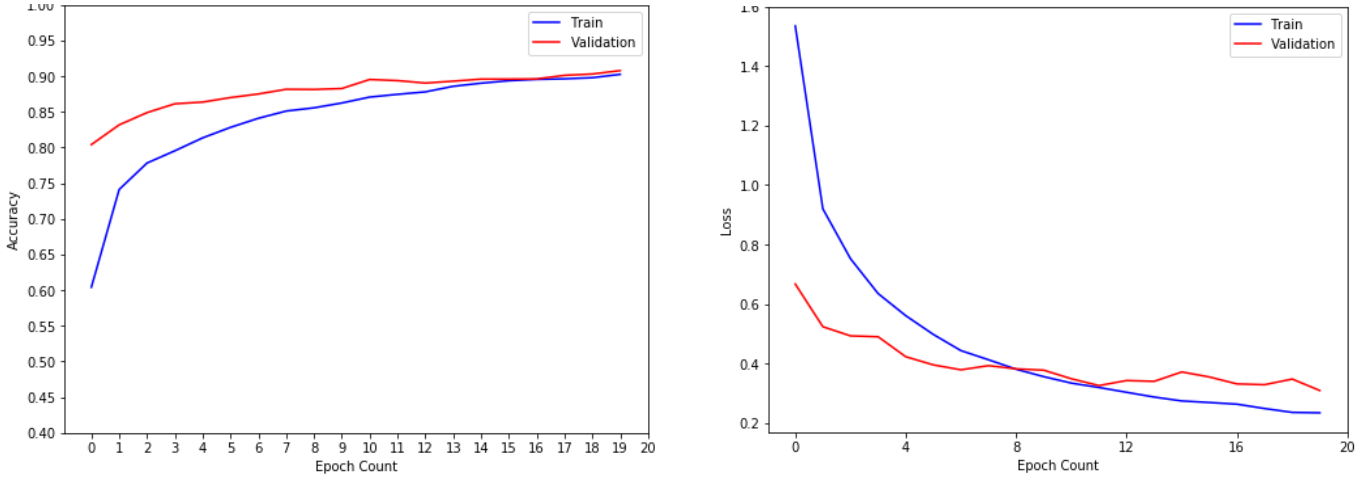
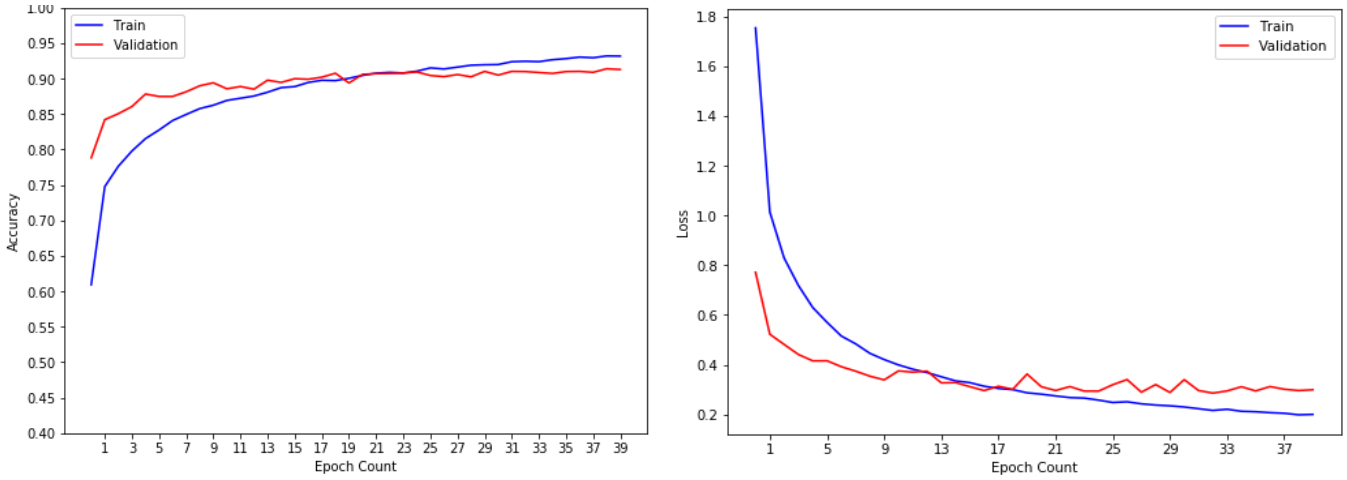
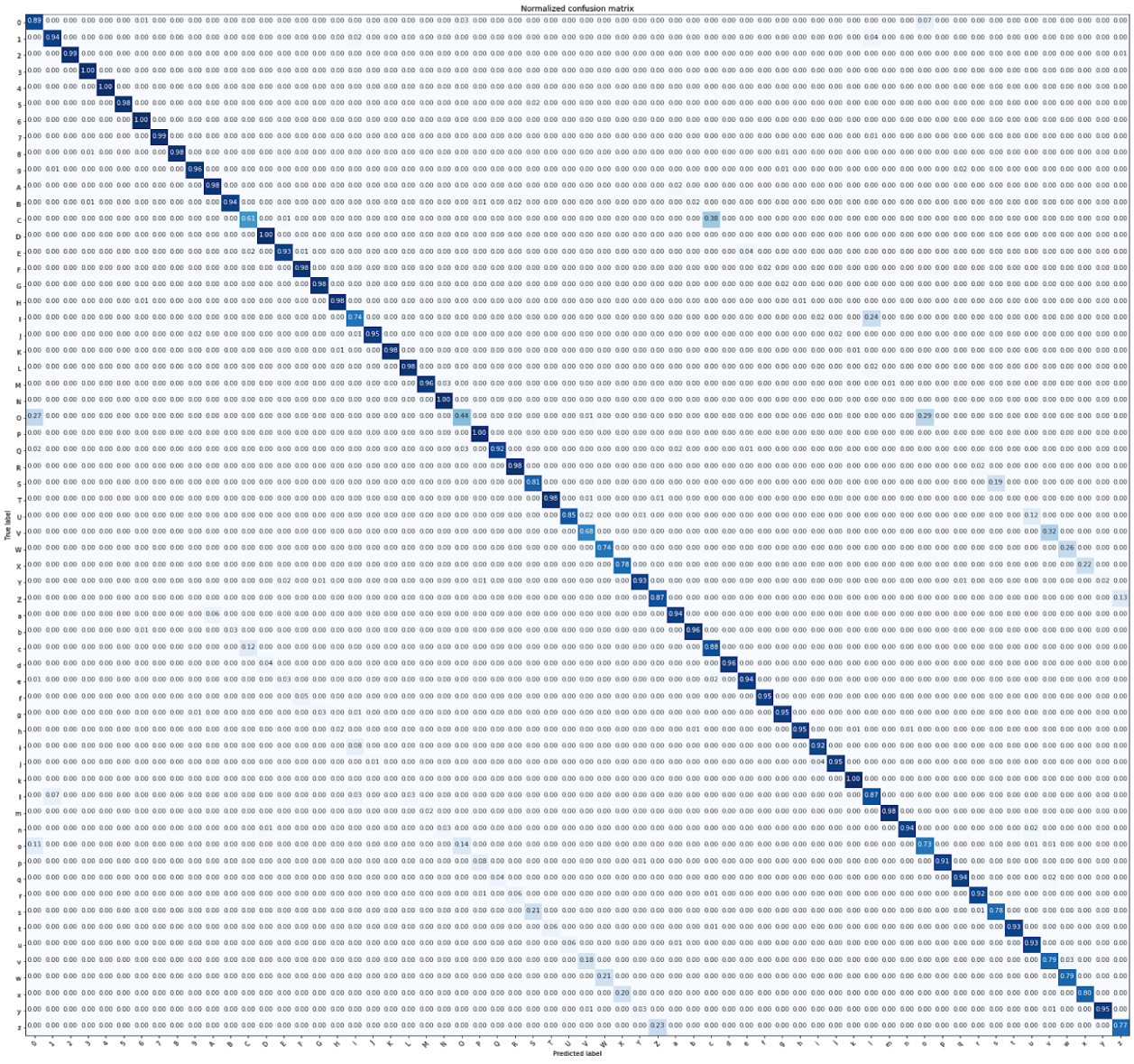


Figure 12: Model D Accuracy & Loss



By inspecting the confusion matrix in Figure 13, 45 of the 62 classes had a prediction accuracy of over 90%. Of the remaining 15, the majority of misclassifications (C-c, O-o, S-s, U-u, V-v, W-w, X-x) resulted in confusion between upper-case and lower-case where to the human eye the shapes are similar as can be seen by parallel lines 26 classes either side of parts of the main diagonal. Other notable misclassifications include capital i and lower-case l, zero and upper-case O. The nature of these misclassifications and the level of dataset mislabelling noted in the introduction, seem to suggest that without additional context a CNN will be able to significantly improve beyond the 91% level.

Figure 13: Model D Confusion Matrix



3.7 Model D: Data Augmentation (90% validation accuracy at 20 epochs, 110secs per epoch)
The group then explored three data augmentation techniques using Model D. The input images contained images of characters that if flipped horizontally, vertically or warped would lose their inherent meaning so only minor scaling could be realistically considered. Using the Keras imagedatagenerator to generate augmented data in real-time was considered a valuable technique as it provided a much broader of differning training sets over the epochs. The validation set was kept as the original data as the group was most interested in the performance on the original data.

1) Model D(b) width_shift_range=0.2,height_shift_range=0.2 (87% at 30 epochs)
The width and height of images was randomly scaled independently by 20% so as to keep the entire character within the input image. This only marginally increased epoch runtime to 122s but the warped training sets resulted in sub-optimal validation accuracy.

2) Model D(c) zoom_range=0.25 (90% at 30 epochs)
Although this technique rescaled the images vertically and horizontally in unison, it did not improve accuracy. Intuitively, the invariant elements of a CNN would be expected to largely unaffected by simply rescaled inputs and the results in this case support that expectation.

Model E: More Conv (89% validation accuracy at 20 epochs, 110secs per epoch)
The group tested a network with an addition convolution module but the results suggested no meaning improvement over the previous best Model D and it was decided this model was unnecessarily deep for the information available in the dataset.

Model F: More FC Layers (85% validation accuracy at 30 epochs, 114secs per epoch)
In order to explore alternative topologies, the group added an additional 512 neuron fully connected hidden layer followed by a 62 neuron fully connected layer. However, even after 30 epochs this model only achieved 85% validation accuracy and did match Model D's accuracy.

Model G: Single Conv Layers (89.9% validation accuracy at 84 epochs, 53secs per epoch)
To explore a simpler but deeper architectures, the group decided to try only one convolutional layer before each max-pooling layer but increase the depth of these units to 5, followed by 2 fully connected layers. The validation accuracy of 89.99% at 84 epochs was near the previous but accuracy improvement had plateaued so the model was not pursued further.

Model I: Transfer Learning VGGNet16 (89.7% validation accuracy at 11 epochs, 115s epochs)
By using the pre-trained VVGNet16 convolutional architecture and applying our own fully connected neural network (basd on Model D) the group achieved 89.7% in 25 minutes of training on a local

machine. Although this accuracy was lower than Model D, it has the significant advantage in runtime as it did not require the use of a GPU-enabled environment to reach a very reasonable accuracy.

3.8 Results Discussion

The initial Feedforward Neural Network's established a baseline accuracy of a surprisingly high 85%, which confirmed the group's expectation that centered alignment and lack of noise in the input images of this dataset presented would be conducive to classification using neural networks. As anticipated a Convolutional Neural Network represented a material improvement over the baseline. The group then sought to maximise accuracy by attempting to apply a variety of network configurations iteratively and adjusting architectures based on inspection of the progressive training and validation accuracy/loss. Beginning with an architecture using concepts informed by the VGGNet architecture, such as multiple 3x3 convolutional layers between pooling layers, we applied subsequent advancements such as Batch Normalisation. Eventually the increasing width of convolutional layers present in VGGNet was abandoned for a simpler series of 32 filter convolutional layers with no negative impact on accuracy.

Data augmentation was explored but because the images depict characters, to retain proper context only limited rescaling and warping were applied with no material improvement to predictive ability. Experimentation revealed that simpler networks with far less parameters and showed a slightly improved accuracy. Varying combinations of convolutional layers and fully connected layers until Model D was found to give the best accuracy at 90.79%.

Inspection of the resultant confusion matrix reveals that over 70% of classes were consistently predicted (over 90%). However where the CNN appeared to struggle was with similarly shaped characters, despite their relative size, such as some uppercase and lowercase pairs of the same alphabetic symbol.

5. Conclusions and Future Work

5.1 Conclusions

The final CNN model achieved over 90% accuracy which seemed to handle the majority of prediction cases. The difficulty in differentiating some uppercase and lowercase pairs appears to be a result of the scale invariant focus on local features by CNNs. Given a larger training dataset or images with greater context, perhaps it could be overcome by these models. Otherwise, the presence of mislabels training dataset, which is to be expected in 'real-life', highly stylized fonts and italics seem to contribute the inaccuracies present and hindered progress towards higher accuracies. In a practical use case, these limitations could be addressed by providing input characters with more context such

as words, sentences and a consistent horizontal plane that may aid the model to grasp greater contextual information to improve accuracy.

5.2 Future Work

To further improve accuracy on this dataset the group considers that exploriation of a wider range of transfer learning models (ResNet etc) could potentially boost predictive power. Research into Regional CNNs reveal that for the simplified input images presented in this dataset, there is unlikely to be a material gain in accuracy because the images only contain a single character that is centred. The occurrence of sometimes unusually high validation accuracies in the conducted experiments could be improved by applying a stratified sampling approach, such as stratified k-folds, to ensure that the validation set is more representative of the broader data set.

The group did implement some data augmentation methods, but alternatives such as PCA or ZCA whitening, which in practice has been used to effect on similar character sets such as MNIST (8), could be applied to measure any potential improvements. Finally, in the absence of more training data, ensembling of preferred models may increase training time considerably but help to increase accuracy.

6. References

(1) Hinton GE, Krizhevsky A, Sutskever I. ImageNet Classification with Deep Convolutional Neural Networks. arXiv:1207.0580 [Internet]. 2012. Available from: http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf

(2) Srivastava N, Hinton GE, Krizhevsky A, Sutskever I. Dropout: A simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014) 1929-1958. Available from: https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

(3) Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:1502.03167 [Internet]. 2015. Available from: https://arxiv.org/pdf/1502.03167v3.pdf

(4) Ruder, S: An overview of gradient descent optimization algorithms (2016) Available from: https://arxiv.org/abs/1609.04747

(5) Taylor L, Nitschke G. Improving Deep Learning using Generic Data Augmentation. arXiv:1708.06020 [Internet]. 2017. Available from: https://arxiv.org/pdf/1708.06020.pdf

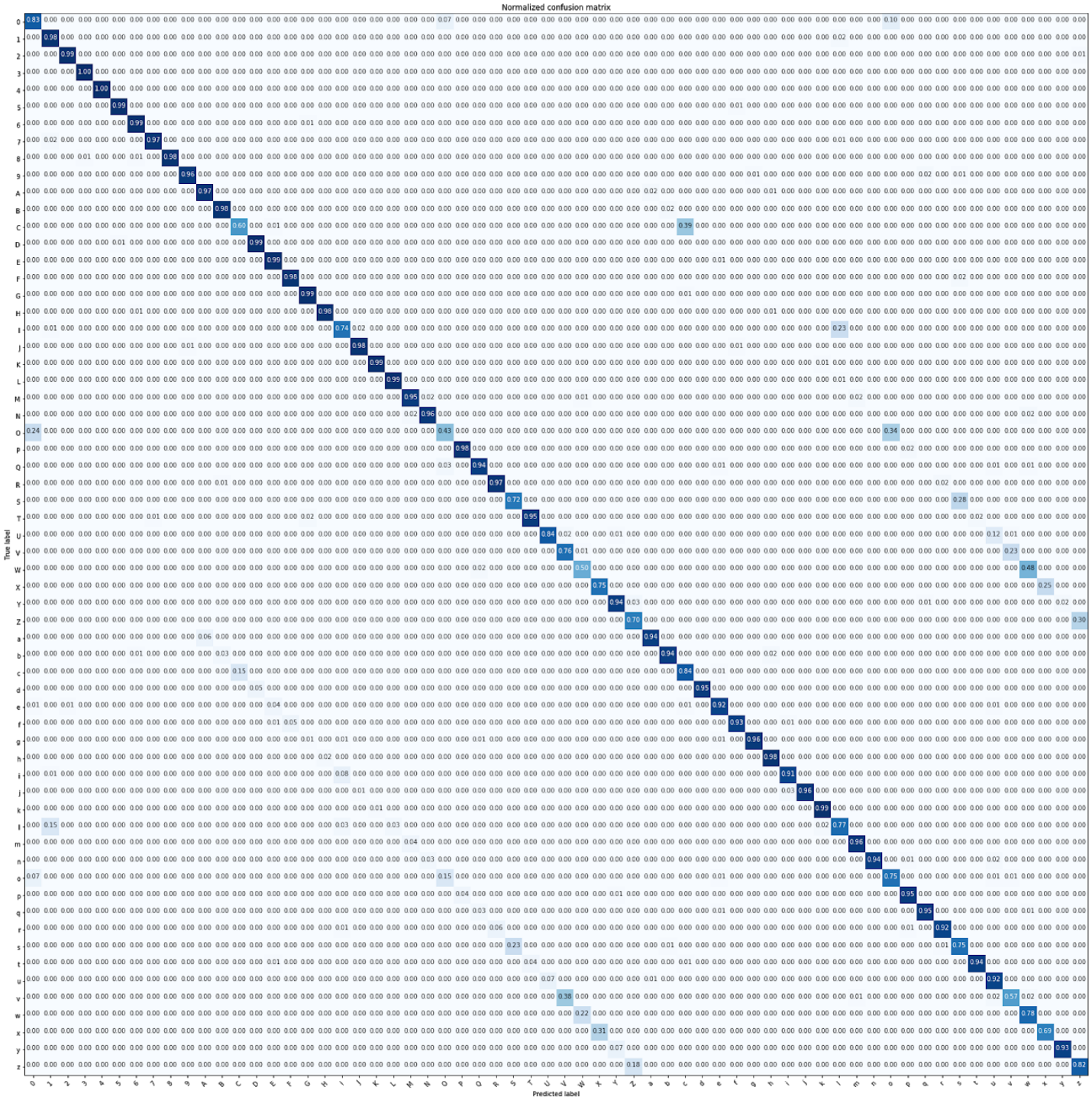
(6) Fergus R, Zeiler MD. Visualizing and Understanding Convolutional Networks [Internet]. 2014. Available from: https://arxiv.org/pdf/1311.2901.pdf

(7) Bishop M. Pattern Recognition and Machine Learning. Springer; 2011.

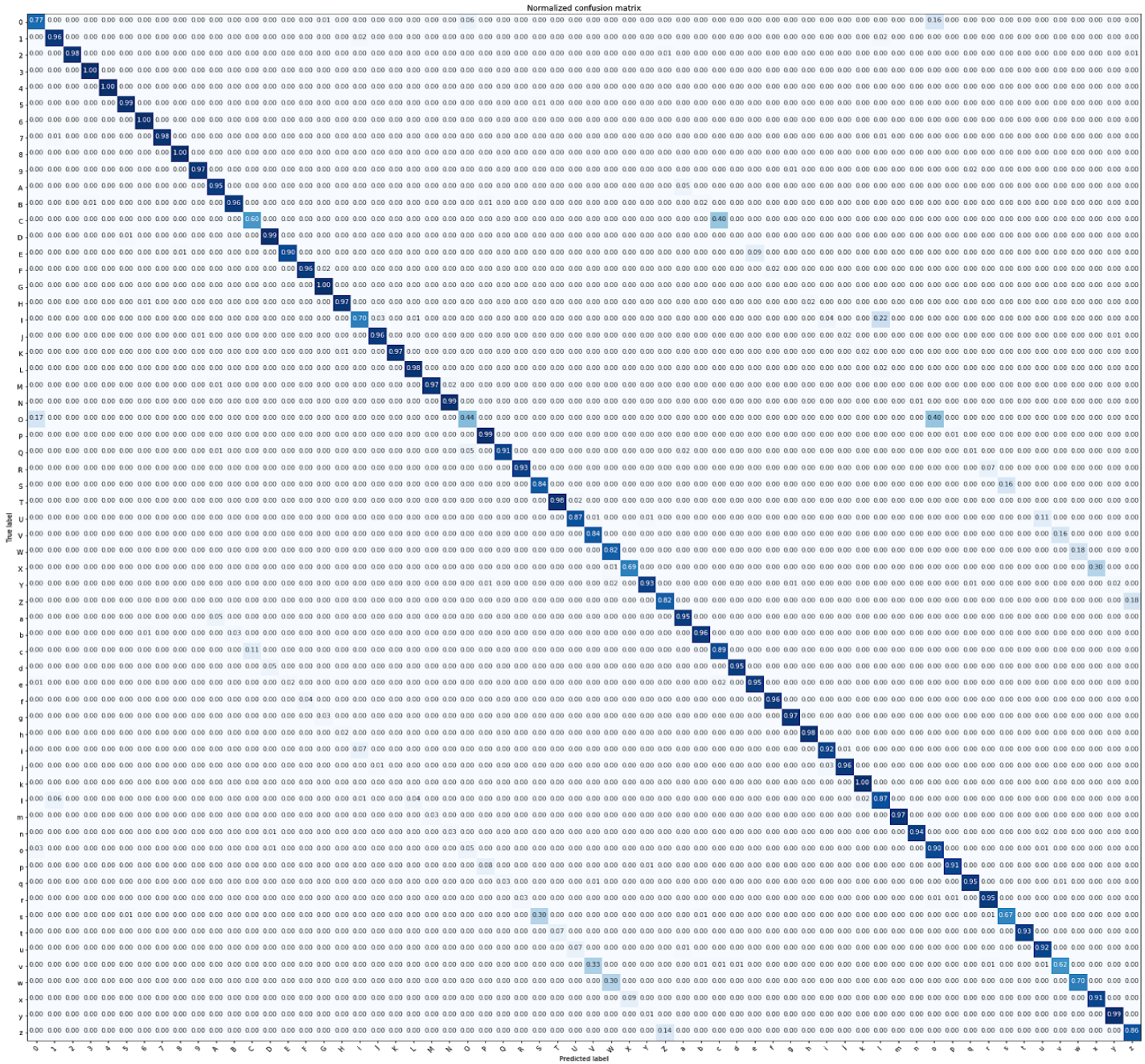
(8) Stanford Deep Learning Wiki [Internet]. Accessed on 4th June 2018. Available from: http://deeplearning.stanford.edu/wiki/index.php/Data_Preprocessing#MNIST_Handwritten_Digits

7. Appendix

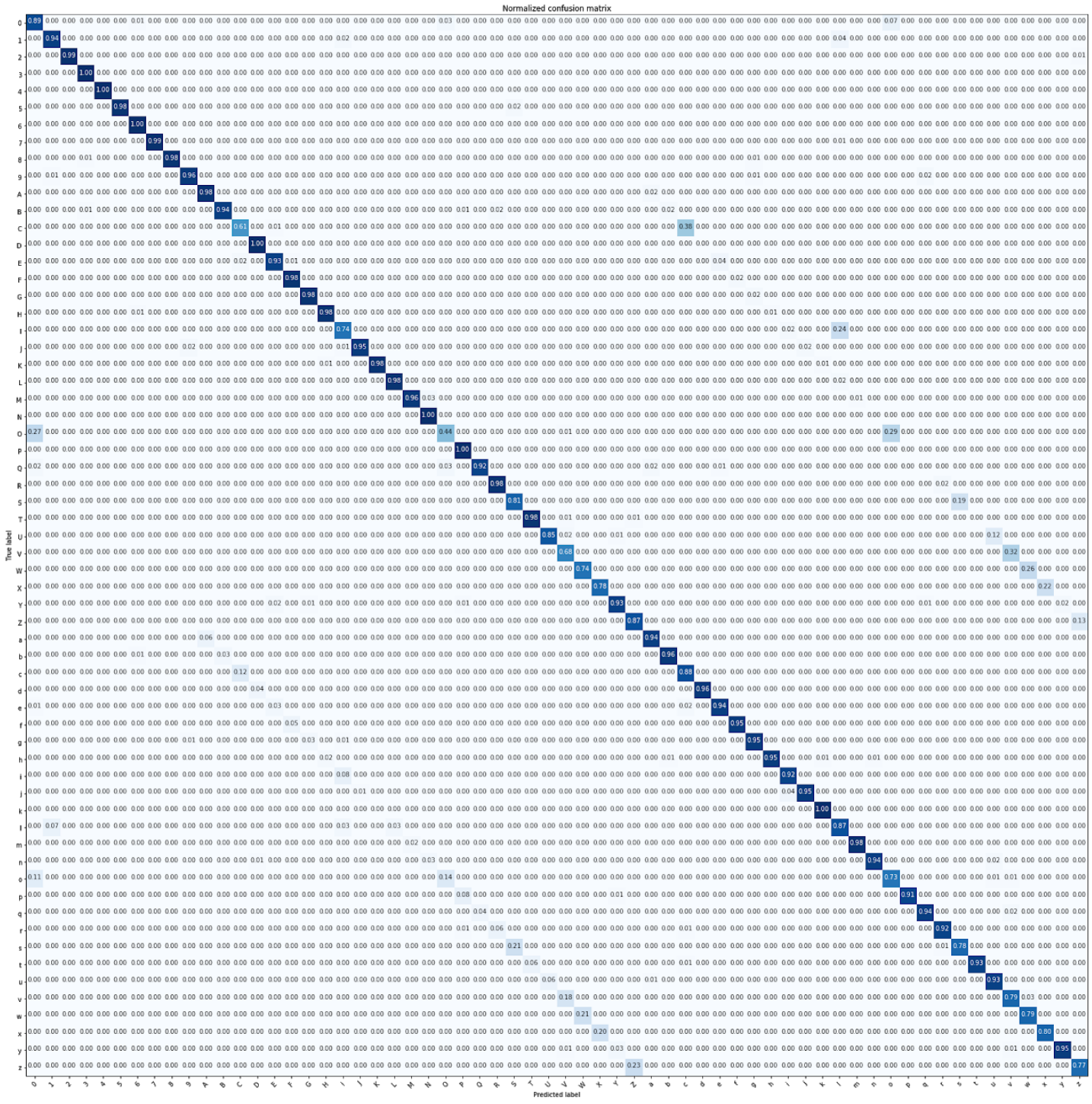
Appendix 7.1: Model A Confusion Matrix



Appendix 7.2: Model C Confusion Matrix



Appendix 7.3: Model D Confusion Matrix



Appendix 7.4: Model D Summary

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 128, 128, 32)	320
batch_normalization_10 (Batc	(None, 128, 128, 32)	128
activation_10 (Activation)	(None, 128, 128, 32)	0
conv2d_8 (Conv2D)	(None, 126, 126, 32)	9248
batch_normalization_11 (Batc	(None, 126, 126, 32)	128
activation_11 (Activation)	(None, 126, 126, 32)	0
max_pooling2d_4 (MaxPooling2	(None, 63, 63, 32)	0
dropout_6 (Dropout)	(None, 63, 63, 32)	0
conv2d_9 (Conv2D)	(None, 63, 63, 32)	9248
batch_normalization_12 (Batc	(None, 63, 63, 32)	128
activation_12 (Activation)	(None, 63, 63, 32)	0
conv2d_10 (Conv2D)	(None, 61, 61, 32)	9248
batch_normalization_13 (Batc	(None, 61, 61, 32)	128
activation_13 (Activation)	(None, 61, 61, 32)	0
max_pooling2d_5 (MaxPooling2	(None, 30, 30, 32)	0
dropout_7 (Dropout)	(None, 30, 30, 32)	0
conv2d_11 (Conv2D)	(None, 30, 30, 32)	9248
batch_normalization_14 (Batc	(None, 30, 30, 32)	128
activation_14 (Activation)	(None, 30, 30, 32)	0
conv2d_12 (Conv2D)	(None, 28, 28, 32)	9248
batch_normalization_15 (Batc	(None, 28, 28, 32)	128
activation_15 (Activation)	(None, 28, 28, 32)	0
max_pooling2d_6 (MaxPooling2	(None, 14, 14, 32)	0
dropout_8 (Dropout)	(None, 14, 14, 32)	0
flatten_2 (Flatten)	(None, 6272)	0
dense_4 (Dense)	(None, 512)	3211776
batch_normalization_16 (Batc	(None, 512)	2048
activation_16 (Activation)	(None, 512)	0
dropout_9 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 512)	262656
batch_normalization_17 (Batc	(None, 512)	2048
activation_17 (Activation)	(None, 512)	0
dropout_10 (Dropout)	(None, 512)	0
dense_6 (Dense)	(None, 62)	31806
batch_normalization_18 (Batc	(None, 62)	248
activation_18 (Activation)	(None, 62)	0

Total params: 3,557,910 Trainable params: 3,555,354