Of Fashion and Neural Networks

Priyanka Chakraborti *University of Nebraska-Lincoln*
Bryan Moore *University of Nebraska-Lincoln*

I. INTRODUCTION

The main goal of this study is to find a suitable fully connected neural network architecture that can classify images from the the Fashion-MNIST dataset with high accuracy without resorting to the usage of any convolutional layers. This dataset is described in detail in section II. For each of the architectures considered we perform a grid-search to determine the optimal hyper-parameters through extensive use of statistical benchmarking. Furthermore, once the best parameters have been decided for each architecture a second benchmarking is performed to isolate the best overall architecture. Finally we perform data augmentation to see if we can further improve the accuracy.

Deciding on the optimal hyper-parameters for a particular architecture is not something that should be attempted just by looking at a single conglomerate number. We instead utilize 95% confidence intervals to decide whether the differences in error between models are statistically significant. To do this we implement 5-Fold Cross-Validation, and assume a student-t distribution, to calculate the 95% confidence limit for the estimated difference in error between all considered combinations. Once the optimal hyperparameters are found the same approach is used to decide between architectures. For this comparison we also added another hyperparameter. This is whether we use early stopping, with a patience parameter of 20, in addition to the other discovered hyperparameters. This process is described in detail in section IV. The findings are discussed in section V and the conclusions in section VI.

II. DATA SUMMARY

The dataset used in this study is the Fashion-MNIST [1]. Every image in this set is a 28x28 grayscale image, and contains 10 unique classes. These classes are shown in Fig. 1. The value for each pixel in these images is an integer between 0 and 255.

We read in 60,000 images from this set. Of these we randomly shuffle the order of the images and set 20% aside to serve as a test set. This leaves us with 48,000 images in our training set and 12,000 images in our test set. It is worth noting that the number of samples in each set puts us far above the threshold where the central limit theorem begins to apply. We can thus safely assume a Gaussian distribution for each dataset. From Fig. 1 we assert that there are in fact an equal number of samples in each class, thus rejecting the need to over-sample or under-sample any particular class.

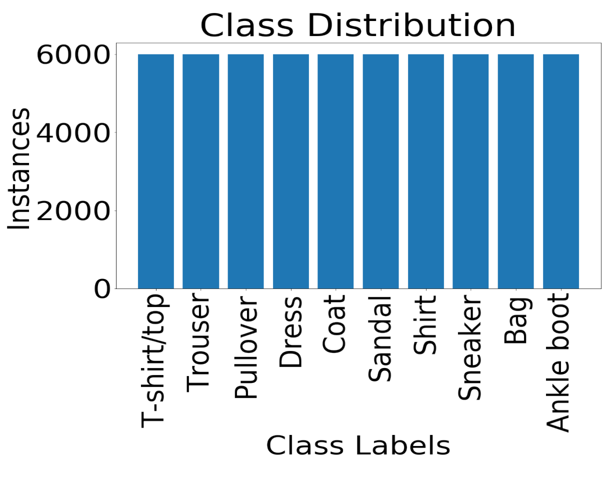

Fig. 1: Histogram of number of samples for each of the 10 classes in the dataset.

III. ARCHITECTURES EMPLOYED

For this study we have chosen to create four unique architectures. The input layer is the same for all architectures. It expects a tensor of dimension (None,784), as the flattened 28x28 images have 784 pixels. The first architecture has two densely connected hidden layers, each utilizing a relu activation function. This was chosen to avoid the vanishing gradient problem commonly encountered with sigmoid functions, thus allowing us to robustly propagate our errors backward during training. It is worth noting that we also explored leaky-relu, but did not see a significant improvement and thus utilized the less complex relu as our activation function. For the output layer we used a softmax activation function. This returns the probability of the input being in any of the 10 possible classes, from which we assign the maximum probability as the predicted label. We explored the parameter space and decided that putting three times as many neurons in the first hidden layer as the second hidden layer provided the best results, so this is what we assigned during cross-validation.

The second considered architecture is a small variant of the first. All details are identical except that dropout layers were inserted between hidden layer 1 and hidden layer 2, and between hidden layer 2 and the output layer. Dropout was chosen as one of our regularizers as it should reduce the co-dependence neurons in different layers have with each other, thus reducing over-fitting. Including this as a hyperparameter will thus show whether co-dependence of neurons is a significant problem with this dataset or not. The dropout percentage is set to 30%. We explored increasing the dropout percentage, but found that although this increased the required number of epochs, it had no significant effect on the accuracy. We thus chose 30% for our grid-search.

The third considered architecture is identical to architecture 1, except that we added two additional hidden layers, totalling four hidden layers. Here we put 3 times as many neurons in the first hidden layer as the fourth hidden layer. The second hidden layer has 2.5 times as many neurons, with rounding, as the fourth hidden layer. The third hidden layer has 1.5 times as many neurons, with rounding, as the fourth. The fourth considered architecture is the same as architecture 3 except that dropout layers were added between hidden layer 1 and hidden layer 2, between hidden layer 2 and hidden layer 3, between hidden layer 3 and hidden layer 4, and finally, between between hidden layer 4 and the output layer, with the same drop-out percentage as architecture two.%.

For all 4 architectures we required 10 neurons in the output layer as there are 10 classes in our data set. As the updates get smaller moving backwards from the output layer we assigned more neurons to the earlier layers and less as we progressed to the output layer. For architectures with dropout layers we used the same number of neurons as the layer they are dropping from with the intuition that the dropout would mimic the same behaviour as the former with the added benefit of regularizing. The number of neurons for the last hidden layer was set as a hyper parameter during grid search cross validation and the final design decisions were made from that outcome.

All architectures are trained by minimizing the cross-entropy loss function. We use this to update the weights in our graph and compare the effects of hyper-parameters and differences in architecture, although our main target is actually the classification accuracy. This is due to the discontinuous derivatives of the raw classification accuracy, which makes updating the weights impossible.

IV. METHODS FOR COMPARING ARCHITECTURES

When read in the class labels are denoted as integers between 0 and 9. To rid any natural ordering amongst the class labels, they are one-hot encoded before feeding into our architectures. We have chosen to use a batch size of 50 for weight updates. This was chosen to hopefully lower the run-time and ensure that we have enough variance in weights to avoid being stuck in a local minima. We utilized 50 epochs for our search for optimal hyper-parameters, as learning curves (such as those shown in Fig. 3) clearly show that the optimal accuracy on the test set occurs well within this region. All images are randomly shuffled, with the seed set to 42, such that although the shuffling is effectively random, it is also repeatable.

For comparing the considered models we use 5-fold cross validation for each. This means that for each model we have a training set of 38,400 images, and a validation set of 9,600. Both are solidly within the regime where the central limit theorem is valid. We do not shuffle any further. This ensures that we can directly compare the same folds across all hyperparameters we are testing. Our initial grid-search is over the number of neurons in the last hidden layer and the exponential decay rate for the first momentum estimate parameter (beta-1) for the Adam Optimizer. We have chosen to test each architecture against 100, 200, and 300 neurons in the last hidden layer to see the effect increasing complexity per layer has on the cross-entropy.

For the Adam Optimizer we left all other hyperparameters at default so as to minimize bias and only modified beta-1. The decay rates were tested in the range from 0.1 to 0.9. The 0.1 hyperparameter was chosen as it causes the window over which previous moments are significant to increase significantly, meaning we effectively keep a longer history of past updates. All combinations of these hyperparameters were tested for each of the 4 architectures.

For a given choice of grid hyperparameters, each fold can be considered an independent investigation as we reset the Tensor-Flow graph before the new fold. At the end of training a fold the accuracy and cross entropy for that fold is computed and saved for statistical evaluation.

We next compare all hyperparameter combinations for each of the 4 architectures. To that end, we calculate the estimate of the error difference between each hyperparameter combination with every other. Note that K-Fold splits will create the same groups every time as we seeded our random permutation before the shuffle. We are therefore comparing the same subset of images every time for each hyperparameter combination within a given architecture. We denote each of these estimated error differences as p_i, where i denotes the fold number. We then average these into a single variable denoted p, which represents the expected difference in error between the two models.

The below formula is used to estimate the maximum value the error difference at the 95% confidence interval, assuming a Student-T distribution of the differences in error.

$$p + t_{c,K-1} s_p \tag{1}$$

where K is the number of folds (5) and s_p is defined below.

$$s_p = \sqrt{\frac{1}{K(K-1)} \sum_{i=1}^{K} (p_i - p)^2} \tag{2}$$

For our desired 95% confidence $t_{c,K-1}$ is 2.132. We use this one-way T-Test by locating the largest positive value for $p + t_{c,K-1} s_p$ from each row, as shown in Table I. This means the model with the largest positive value outperformed the model specific by that row. If the optimal model for that row is found to be the same as the row, it means that it outperformed all models it competed against. Using this approach we will implement logical comparisons to trace back which combination was the best. As an example, if P1>P2 and P2>P3 it means that P1>P3, and P1 is thus the optimal architecture. Any ties will be decided by selecting the combination which exhibits the greater average accuracy on the hold-out test set. This comparison is repeated for each architecture.

After the optimal hyperparameters for each architecture are determined we follow the same approach as above to compare the results from each of the 4 architectures. We also add in the additional hyperparameter of whether to implement Early Stopping. From these results we select our chosen model.

To form a full picture of our optimal architecture we then train it on the train set, with 20% of the train data set aside for evaluating early stopping. The original test set is used to get a reasonable estimate of what the model performance will be on the test-set held by the instructors. During early stopping we save the model any time the accuracy on the validation set is lower than we have seen previously, and restore the last save point after we are convinced the accuracy on the validation set will not improve. We used a patience parameter of 20 epochs. This restores our model to the last epoch before it began to overfit.

In figures 2, 3, and 4 we show respectively the confusion matrix on the test set, the learning curves for our chosen optimal models, and the learning curves for our chosen model after additional optimization is implemented. We elaborate further in the results section below.

V. Results

A. *Choose Best Model*

The results (see Table I) show that for the 2-layer network with no dropout the optimal hyperparameters are to use Beta-1 for the Adam Optimizer as 0.9 and 100 neurons for the second hidden layer. This was the best choice for every comparison in this architecture, meaning no logical comparisons were required.

TABLE I: Confidence Intervals for 2 Hidden Layers and No Dropout Layers

	Model 1	Model 2	Model 3	Model 4	Model 5	Model 6
Model 1	0.0000	0.0205	-0.0572	-0.0090	-0.0816	-0.0575
Model 2	-0.0205	0.0000	-0.0777	-0.0295	-0.1020	-0.0780
Model 3	0.0572	0.0777	0.0000	0.0482	-0.0243	-0.0003
Model 4	0.0090	0.0295	-0.0482	0.0000	-0.0726	-0.0485
Model 5	0.0816	0.1020	0.0243	0.0726	0.0000	0.0240
Model 6	0.0575	0.0780	0.0003	0.0485	-0.0240	0.0000

Each row represents a different set of hyper parameters, and the column which corresponds to the largest value in that row represents the model which outperformed it as judged by the condition imposed by equation (1). Model 1 has 100 neurons in hidden layer 2 and beta-1 as 0.1, Model 2 has 100 neurons and beta-1 as 0.9, Model 3 has 200 neurons and beta-1 as 0.1, Model 4 has 200 neurons and beta-1 as 0.9, Model 5 has 300 neurons and beta-1 as 0.1, and Model 6 has 300 neurons and beta-1 as 0.9.

For the 2-layer network with 30% dropout the optimal hyperparameters for beta-1 and the number of neurons were exactly identical to the one without drop-out. This was found to be the best choice for every comparison in this architecture. For the 4-layer network without dropout the optimal hyperparameters were to use beta-1 as 0.1 and 100 neurons in the last hidden layer. The 4-layer network with 30% dropout had identical optimal hyperparameters as the 4-layer without dropout.

It is interesting to note that both 2-layer networks have beta-1 as 0.1, whereas both 4-layer networks have beta-1 as 0.9. This indicates that for 2-layer networks it is important to retain momentum from the history of updates, perhaps because they are not able to learn as many details about the images in the same number of epochs. Perhaps 4 layer networks may not need this as they are able to extract more detailed features from the input images.

The second one-sided T-test compares the best of all 4 architectures with and without early stopping. Table II and Table III show the calculate 95% confidence limits to the error difference for all of these cases. In these we see that without early stopping the best architecture is 4 hidden layers, no dropout layers, set beta-1 to 0.1, and use 100 neurons in the last hidden layer. With early stopping the best architecture is 2 hidden layers, no dropout layers, a beta-1 of 0.9, and 100 neurons in the last hidden layer. We note here that the optimal number of neurons is 100 for all architectures. However, inspecting the learning curves shown in Fig. 3 shows evidence of significant overfitting. We investigated lowering the number of neurons in the last hidden layer to 50, but this decreased the accuracy on the train set without helping with the overfitting problem. Thus the final chosen model has retained 100 neurons in the last hidden layer.

TABLE II: Confidence Limits for Error Differences with No Early Stopping

	Model 1	Model 2	Model 3	Model 4
Model 1	0.0000	0.0174	0.0580	0.0693
Model 2	-0.0174	0.0000	0.0406	0.0519
Model 3	-0.0580	-0.0406	0.0000	0.0113
Model 4	-0.0693	-0.0519	-0.0113	0.0000

Model 1 has 2 hidden layers, no dropout, 100 neurons in last hidden layer, and beta-1 as 0.9. Model 2 has 2 hidden layers, 30% dropout, 100 neurons in last hidden layer, and beta-1 as 0.9. Model 3 has 4 hidden layers, no dropout, 100 neurons in last hidden layer, and beta-1 as 0.1. Model 4 has 4 hidden layers, 30% dropout, 100 neurons in last hidden layer, and beta-1 as 0.1. Of these Model 3 is optimal.

TABLE III: Confidence Limits for Error Differences with Early Stopping

	Model 1	Model 2	Model 3	Model 4
Model 1	0.0000	-0.0069	-0.0191	-0.0179
Model 2	0.0069	0.0000	-0.0122	-0.0110
Model 3	0.0191	0.0122	0.0000	0.0012
Model 4	0.0179	0.0110	-0.0012	0.0000

Model 1 has 2 hidden layers, no dropout, 100 neurons in last hidden layer, and beta-1 as 0.9. Model 2 has 2 hidden layers, 30% dropout, 100 neurons in last hidden layer, and beta-1 as 0.9. Model 3 has 4 hidden layers, no dropout, 100 neurons in last hidden layer, and beta-1 as 0.1. Model 4 has 4 hidden layers, 30% dropout, 100 neurons in last hidden layer, and beta-1 as 0.1. Of these Model 1 is optimal.

Finally, we are left with the two optimal architectures. To this end, we used accuracy of the hold out test set to make our final decision. These results are shown in Table IV. Although these exhibit similar accuracy, we have decided to use the model with early stopping. It utilized less data in the training set and achieved a slightly higher accuracy on the the hold out set. We believe that using the entire training set might achieve higher accuracy on the actual competition test-set.

TABLE IV: Accuracy On Test Set - Comparing Two Chosen Architectures

	Accuracy On Test Set
No Early Stopping	0.8867
With Early Stopping	0.8870

The model without Early Stopping has 4 hidden layers, no dropout, 100 neurons in last hidden layer, and beta-1 as 0.1. The model with Early Stopping has 2 hidden layers, no dropout, 100 neurons in last hidden layer, and beta-1 as 0.9. Here we can see that the model with early stopping showed the higher accuracy on the our hold out test-set.

We thus conclude that out of our considered models the optimal one has 2 hidden layers, no dropout layers, beta-1 as 0.9, 100 neurons in last hidden layer, and uses early stopping.

B. *Characterize Chosen Model*

We now characterize our chosen model. We create a confusion matrix for our predictions across classes. This is shown in Fig 2.

From this we see clear evidence of significant misclassification between shirt and T-shirt/top. To further study this problem we utilize the triad lens of Precision, Recall, and F1-Score for each class. The results are shown in Table V. Recall and Precision are defined as shown below:

$$\text{Precision} = \frac{t_p}{t_p+f_p} \textit{ and } \text{Recall} = \frac{t_p}{t_p+f_n}$$

Where t_p are the number of true-positive classifications, f_p are the number of false-positive classifications, and f_n are the number of false-negative classifications. We would emphasize high recall if correctly labeling an image as part of our class was the priority, and accidentally labeling other images as part of our class when they are not is acceptable.

Conversely, we would emphasize high precision if incorrectly labeling an image as part of our class would be a more fatal prediction than assigning it a different class. For our purposes there is no clear reason to prefer false positives over false negatives, or vice versa, and we will therefore allow the F1-Score to guide us. This is a combination of the precision and recall, such that, both are equally weighted.

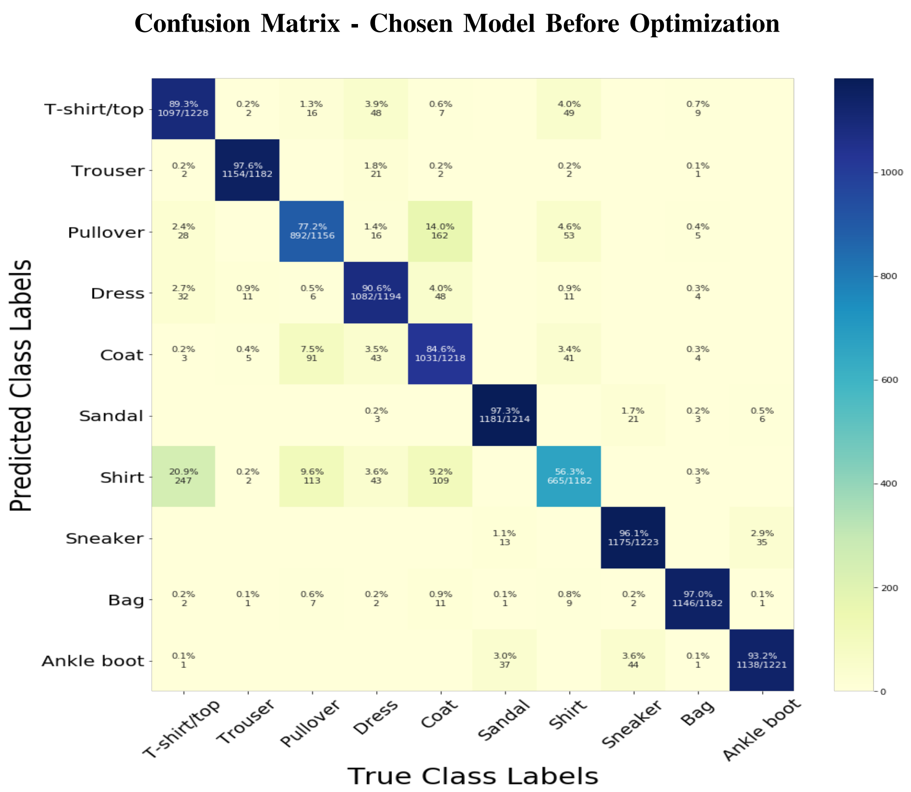

Fig. 2: This shows the percentage of each class correctly predicted by our chosen model. Note that true positive rates for each class lie on the diagonal.

TABLE V: Precision, Recall, and F1 Score On Test-Set

	Precision	Recall	F1-Score
T-shirt/top	0.78	0.89	0.83
Trouser	0.98	0.98	0.98
Pullover	0.79	0.77	0.78
Dress	0.86	0.91	0.88
Coat	0.75	0.85	0.80
Sandal	0.96	0.97	0.97
Shirt	0.80	0.56	0.66
Sneaker	0.95	0.96	0.95
Bag	0.97	0.97	0.97
Ankle boot	0.96	0.93	0.95

Overview of Precision, Recall, and F1-Score for all classes. Note that, using the F1-Score as our guide, the most problematic classes are Shirt, Pullover, Coat, and T-shirt/top.

In Table V we see that our model has difficulties classifying Shirt, Pullover, Coat, and T-shirt/top. Logically this makes sense as these items of clothing are quite similar to each other. In the next subsection we investigate a few tweaks to our model which help to address this. Before we continue to that, we first examine a few Learning Curves for this model. These are shown in Fig 3. We use these to visually understand how the cross-entropy, accuracy, and generalization error change over epochs.

The optimal epoch for early stopping, as identified by our algorithm, is epoch 8. From the learning curves in Fig 3 it is easy to see why. The train set continues to learn more and more from the samples after that point, improving its own accuracy, without benefiting the validation set at all. The cross-entropy plot makes this even more clear, as after epoch 8 the error on the validation set stops decreasing, and actually begins increasing. We will try to address this over-fitting as well in the following subsection.

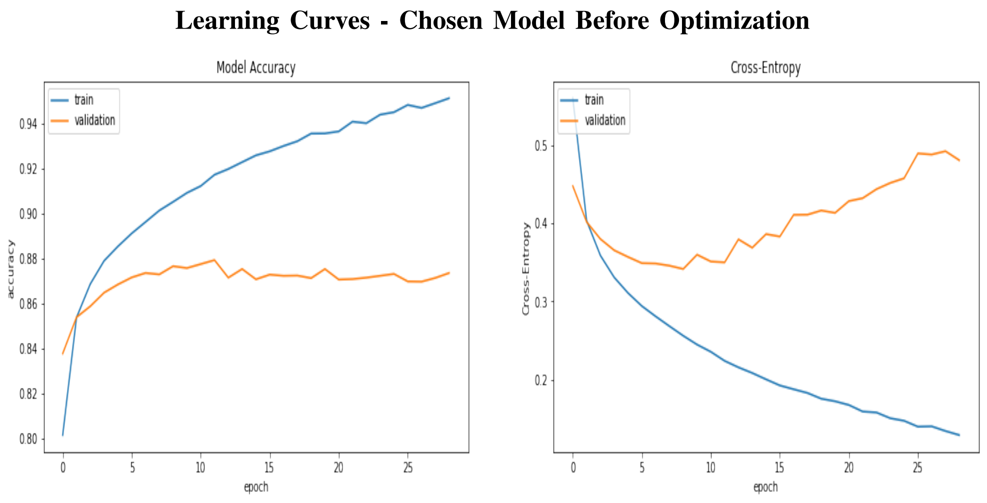

Fig. 3: In both plots there is evidence of significant overfitting after epoch 8.

C. *Optimize Chosen Model*

To address the overfitting problem shown by the learning curve in Fig 3 we first examine the effect of adding L2 regularization to our hidden layers and output layer. By iterating through combinations we found that adding L2 regularization of 0.0013 to each layer, and increasing the batch size to 1000, drastically helped to decrease the generalization error. Adding L2 regularization creates a sparser matrix of the weights, thus combating fitting to the noise in the train set. A larger batch size also helps avoid over fitting by ensuring that weight updates are done using the average of a larger collection of images. The improvement in generalization error can be seen in Fig 4. Combining this with adjusting beta-1 from 0.90 to 0.99 led to significantly better performance on our holdout test set, as well as the instructor's test set.

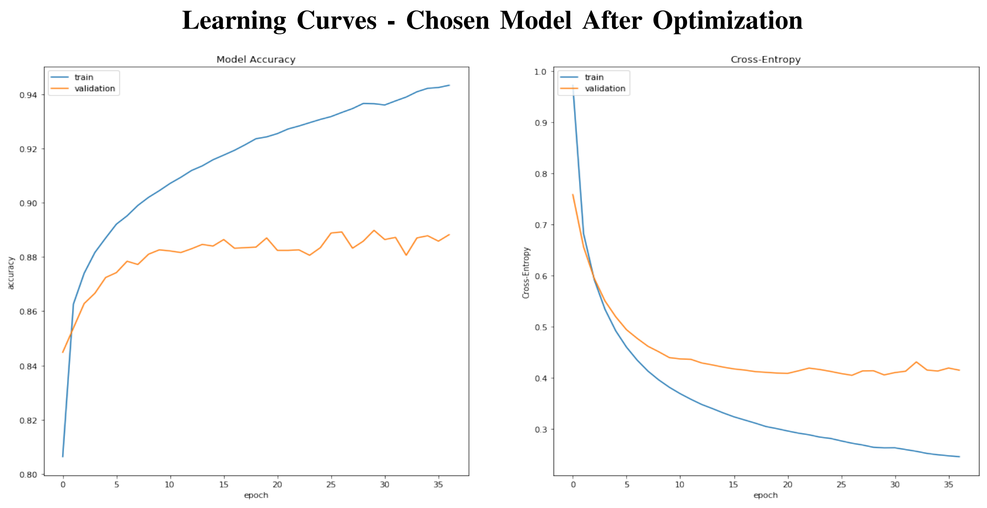

Fig. 4: This shows the learning curves on the optimization and validation sets after adding L2 regularization, increasing the batch size to 1000, adjusting the beta-1 hyperparameter for the Adam Optimizer to 0.99, adding noisy duplicates of images in the train set, setting only 10% of the data aside as the test set, and only 10% of what is left in the training set aside for the validation set. We see significantly less overfitting than previously.

Boot-strapping is one method that is widely used to address misclassifications in a minority or majority class. A more traditional classifier like random forest takes advantage of 'boosting' to turn weak learners into strong learners. Boosting has been implemented in CNN's such as Alex-net [2]. The Fashion-MNIST dataset has an equal number of class samples, hence the misclassification can be attributed to the handicapped response of the fully connected nets to finer details. As a final attempt towards better classification of Shirt, Pullover, Coat, and T-shirt/top in our model we make duplicates of every image in the optimization set, leaving the validation and test set untouched. We add one or more random combination of noise to each image. These types of noise are salt and pepper, speckle, Poisson, and Gaussian. We duplicate each class so that no single class is over sampled. To an extent this could fundamentally alter the weight updates, but we reason that on average it will only perform slightly worse on images it was classifying correctly before. The results seem to agree with this hypothesis, as seen in the confusion matrix in Fig 5.

We also choose to set aside only 10% of the data as our hold-out test set. This leaves 6000 images in the test set, which we believe is sufficient to indicate whether our alterations are providing any benefit. We also reduced the number of images we put in our validation set to 10% of the training set. This leaves 5400 images for determining our Early Stopping epoch. We thus have 48600 images in our test set. Augmenting this with the noisy duplicates with gives us 97200 images to learn from. As can be seen from the confusion matrix, the problems with misclassification of the three classes has been lowered. Evaluating this model against our hold-out test-set gives 90.25% while the instructor's test set shows an accuracy of 92.5%.

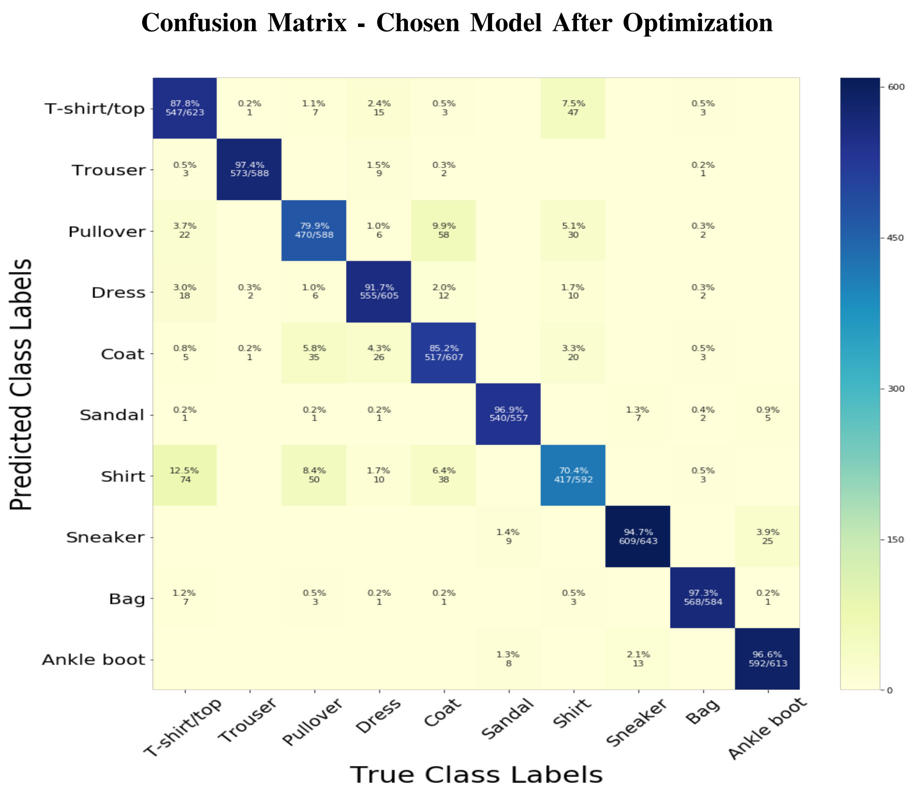

Fig. 5: This shows the percentage of each class correctly predicted by our chosen model. Note that correct predictions for whether an image should be within the class lie on the diagonal. Although we are predicting slightly worse on the less-problematic classes from before, overall we see improvement over the model before this optimization.

VI. Conclusion

In conclusion, we have conducted a statistical investigation of four possible fully connected neural network architectures, and their sub-architectures, and used each to classify the Fashion MNIST dataset with sufficiently good accuracy. No convolutional layers were implemented. We arrive at an optimal model of two layer architecture with no dropout layers, beta-1 as 0.9, 100 neurons in second hidden layer, and using early stopping. This was further fine tuned to include L2 regularization of 0.0013, increasing the batch-size to 1000, and setting beta-1 to 0.99 instead of 0.90. We also restricted the amount of data set aside for the test-set to 10% and only set aside 10% of the training set for early stopping. Finally, we utilized data augmentation to duplicate our train images with noise, achieving a final accuracy of 92.5% on the instructor's test set.

We deduce that any further improvements, both in terms of overfitting and separation of similar clothing, likely require the introduction of convolutional layers. We thus leave this as possible improvements for the future and hope this investigation has helped to shed light on how even a simple neural network can classify low-quality images with reasonably high accuracy.

References

[1] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," 2017.

[2] M. Moghimi, S. J. Belongie, M. J. Saberian, J. Yang, N. Vasconcelos, and L.-J. Li, "Boosted convolutional neural networks," in *BMVC*, 2016.