# Neural Networks for Image Classification

**Saumyaa Verma, Alex Wang, Julia Sokolov**

## Abstract

In our report we implement a neural network from scratch. We construct a multilayer perceptron or MLP to classify pictures of clothes from the Fashion-MNIST Dataset. For this, we use mini-batch SDG methods. We conduct multiple experiments and also compare our model with the accuracies achieved by a CNN model. In our experiments, we try different learning rates, activation functions, model depths, batch size, and dropouts.

Conducting our experiments we found that increasing the number of hidden layers led to a general trend of increase in testing accuracy but also led to overfitting. When varying the activation functions, all three (ReLU, leaky ReLU and Tanh) performed in a similar manner, however tanh gave marginally lower accuracies. When varying the dropout, it was found that the more number of nodes dropped, the lesser the accuracy. Finally, we decided to use the MLP with ReLU activation with no dropout, 500 batch size, learning rate of 0.001, with 1 hidden layer consisting of 128 units as our best MLP architecture with an accuracy of almost 89%.

We found that after varying some parameters in our CNN model, we were eventually able to achieve a higher test accuracy than with our MLP model.

## Introduction

In this report we describe the steps that we took to implement a multilayer perceptron and use it to predict on the Fashion-MNIST Dataset. We compare the effectiveness of our model by varying parameters like the depth of the model, dropout/regularization, number of epochs and much more. We also conduct preprocessing on our image data and analyze if that helps us achieve a better model.

The Fashion-MNIST dataset consists of 60,000 training images and 10,000 images in a test set. The images are grayscale and belong to one of 10 classes. Hence, we conducted the classification task of identifying the images as one of the 10 classes.

Based on our analysis we found some interesting results. Firstly, it was very clear that increasing the number of hidden layers led to an increase in accuracy. Model complexity increased and so did the accuracy on unseen data. Testing accuracy for no hidden layers always stayed under 85% however it reached up to 90% for 2 hidden layers. We concluded that 1 hidden layer with ReLU was the best as it also gave up to 90% testing accuracy but also avoided the overfitting and computation costs from the 2 hidden layer model.

Next we varied our activation functions. Since Leaky-ReLU required an extra $\gamma$ parameter, as an extra **creativity** step, we tuned this. Lower $\gamma$ values gave higher accuracies. Hence, we chose the $\gamma$ value of 0.001, which gave us a testing accuracy of up to 89%. ReLU, Leaky-ReLU and Tanh all gave very similar accuracies on unseen data. Moreover, as $\gamma \rightarrow 0$, Leaky-ReLU converges to ReLU. Hence, Leaky-ReLU and ReLU both performed similarly and quite well giving accuracies around 89%.

After this, we played around with the dropout on a 2 hidden layer ReLU-128 model. As we increased the number of nodes that we dropped, we saw a decrease in the accuracy we achieved. This is understandable as the model doesn't fit properly with too few nodes. Without dropping any nodes in our 2 hidden layer, ReLU-128 model, we achieved an accuracy of about 88% for 20 epochs. However, dropping 50% of the nodes made the accuracy drop to less than 70%.

Throughout our experiments, we utilized normalized images, however, using unnormalised images instead, dropped the accuracy to 10%. This is justified as our activation function becomes not non-linear.

Based on all the experiments we did, it was concluded that 1 layer MLP gave good results without overfitting, with ReLU Activation function.

Going an **extra step** we performed hyperparameter tuning on these models, with different batch sizes, learning rates and dropouts. We fixed the number of epochs to be 20 because through our previous experiments we found that accuracies plateaued after this. The 2 layer ReLU model with a batch size of 500, learning rate of 0.001 and dropping no nodes, gave us the best model with an accuracy reaching 89% over 20 epochs. Lastly, we ran all our experiments over 100 epochs. In the few starting passes, we saw a sharp increase in accuracy; however,

as the number of epochs increases so does overfitting. Hence, 20 was a good number to stop at to avoid overfitting.

Lastly, we created two different CNN models and observed the result of learning rate and optimizer used on the test accuracy. In doing so, we observed many different accuracies ranging from 78% to 91%. After varying the learning rate and optimizer used in CNN, we were able to achieve a test accuracy of about 90.16%, which is higher than the maximum test accuracy observed with MLP after tuning the hyperparameters.

Now, with regards to **related works**, this dataset has been used many times. We found the paper on classification of these images using convolutional neural networks particularly helpful as it gave us an insight into the best practices in image classification using neural networks[1].

## Datasets

The Fashion-MNIST dataset is a benchmark dataset that has been used to achieve groundbreaking results in image classification tasks. It is the database of Zalandos articles and is made up of 60,000 training and 10,000 testing images divided amongst 10 classes. The class distribution is shown in Figures 1 and 2.

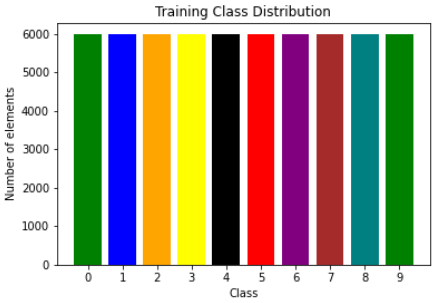


**Figure 1:** Training Set Class Distribution

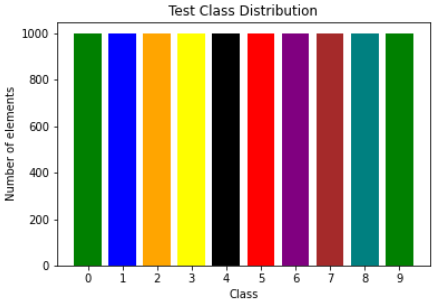


**Figure 2:** Test Set Class Distribution

The labels signify 1 of 10 classes (0 T-shirt/top, 1 Trouser, 2 Pullover, 3 Dress, 4 Coat, 5 Sandal, 6 Shirt, 7 Sneaker, 8 Bag, 9 Ankle boot). Each image is made up of 28×28 pixels. Since the pixels were values between 0 to 255, we normalized the values of the pixels using 0-1 normalization. We also flattened our array, by reshaping the images from 28×28 to an input size of 784. We finally used this input vector of size 784 to conduct our classification tasks. Hence, we vectorised and normalized our data. Vectorisation aids in allowing the data to better fit the model and normalization will aid the speed of convergence in gradient descent. Finally, we one-hot encoded our data.

## Results

### 3.1 Network Depth and Accuracy:

For this part, our group conducted 3 experiments. The three were as follows: model with 0 hidden layers, model with 1 hidden layer and model with 2 hidden layers. For the hidden layers, we used the ReLU activation function with 128 units each. The outer layer was always Softmax. After training the three models with ideal hyper parameters (we conducted hyperparameter tuning in section 3.6), we went ahead and compared the testing accuracies. We plotted the training and testing accuracies for these models in Figure 3 and Figure 4; as we varied the number of epochs in the models.

It is very clear that model with no hidden layers performed the worst on seen as well as unseen data. As for the the models with 1 and 2 hidden layers respectively, the one with 2 hidden layers tended to have a higher training accuracy but testing accuracy was very similar for the two. Overall however, 1 hidden layer seems to be the best as it avoids the overfitting that is brought along with 2 hidden layers. Hence, non-linearity and increasing depth leads to an increase in testing accuracy in general but it also leads to overfitting.


**Figure 3:** Training Accuracy for Different Depths and Epochs

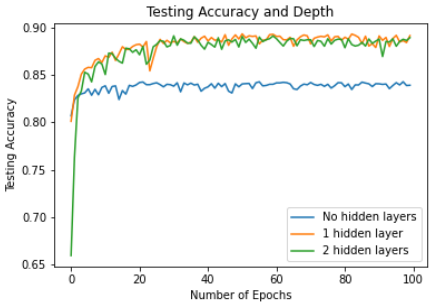


**Figure 4:** Test Accuracy for Different Depths and Epochs

### 3.2 Activation Function and Accuracy:

We conducted 3 experiments in this part. Firstly we considered the model from section 3.1 which consisted of 2 hidden layers, each with 128 units and ReLU activation function. Secondly, we considered a model with 2 hidden layers, each with 128 units and tanh activation function and lastly we considered the exact same model but with a leaky-ReLU activation function. In particular, the leaky-ReLU also required a $\gamma$ value to be used. For this, as an **extra creativity** activity, we conducted tuning on the $\gamma$ value. The $\gamma$ values of 0.001, 0.01 and 0.1 had very similar performances, hence, we ended up conducting all our experiments with $\gamma$ = 0.001 (can be seen in Figure 5 to give high testing accuracy across all epochs).

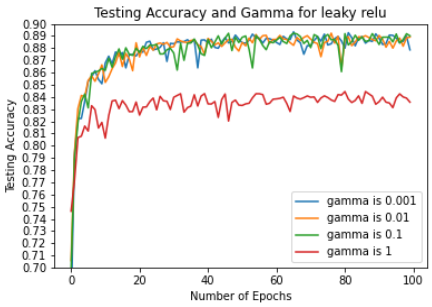


**Figure 5:** Test Accuracy for Different Leaky-Relu Parameters and Epochs

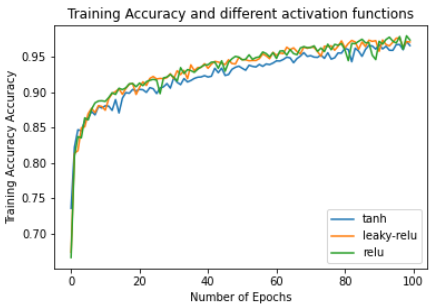


**Figure 6:** Training Accuracy for Different Activation Functions and Epochs

We trained the 3 models and plotted our results in Figure 6 (training accuracy) and Figure 7 (testing accuracy).

Looking at the graphs across all epochs, clearly tanh underperforms. However, the performance of leaky ReLU ($\gamma$ = 0.001) and ReLU are very similar. This is understandable as the smaller the $\gamma$, the more likely it is that the performance of leaky-ReLU converges to that of ReLU.

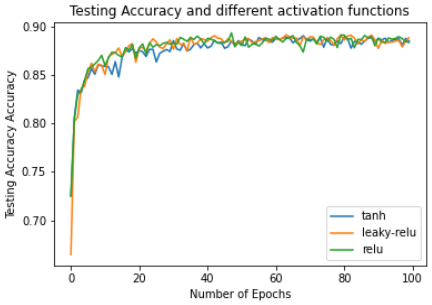


**Figure 7:** Test Accuracy for Different Activation Functions and Epochs

### 3.3 Variation in model accuracy when varying dropout

With 2 hidden layers and ReLU activation, dropping more nodes at each layer consistently lowered both the test and training accuracies (Figures 8 and 9). Our training set is rather small in machine learning terms at fewer than 100k samples, so dropout was likely counterproductive because our model wasn't overfitted to begin with.

50% dropout was an interesting case; the test accuracies flatlined at 10% when the number of epochs increased past 25. The model weights are updated too many times and overfit the reduced training data, leading to poor performance on both the test set and full training set.

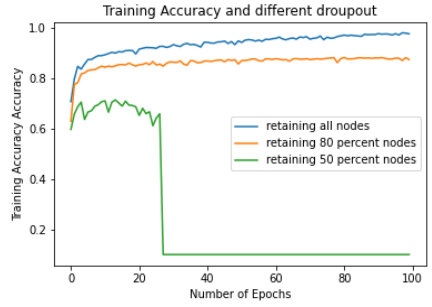


**Figure 8:** Train Accuracy for Different Dropout and Epochs

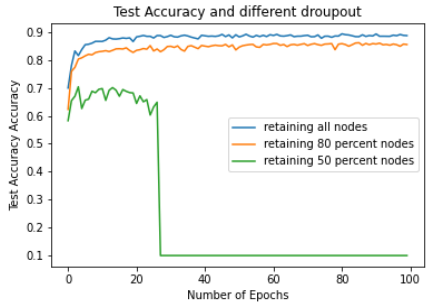


**Figure 9:** Test Accuracy for Different Dropout and Epochs

### 3.4 Variation in accuracy when using un-normalized images

Without normalization, our MLP with 2 hidden layers and ReLU activation achieved test and train accuracies of 10%. Normalization was crucial to our model's performance. This is understandable as weights are now summing up to not be zero. Hence, our activation function is not non-linear anymore.

### 3.5 Comparison of multilayer perceptron model with CNN

When creating the CNN's, we first constructed a CNN that resembled the MLP, with 2 convolution layers, 2 fully connected layers, ReLU as the activation function in each layer, Softmax as the activation function for the final layer, and using SGD (Model 1 in Table 1). This CNN gave an accuracy of about 89.45%, which is slightly higher than the highest accuracy we received with the MLP. We then constructed a few more CNN's, varying the architecture (as an **extra creativity** step) and hyperparameters, to see how it affects accuracy. The architectures can be seen in Table 1.

| | |
|---|---|
| Model 1 | `Conv(filters=8, kernel_size=(3,3),padding='same')-> MaxPooling(pool_size=(2, 2),strides=2) -> Conv(16, kernel_size=(5, 5)) -> MaxPooling(pool_size=(2, 2),strides=2)` |
| Model 2 | `Conv(filters = 32, kernel_size=(3, 3)) -> MaxPooling(pool_size=(2, 2),strides=2) -> Conv(64, kernel_size=(3, 3)) -> MaxPooling(pool_size=(2, 2),strides=2) -> Conv(128, kernel_size=(3, 3)) -> MaxPooling(pool_size=(2, 2),strides=2)` |

**Table 1 :** CNN models where Activation functions are all ReLU and the final dense layers have been omitted

Model 1 contains 2 convolution layers with 2 max pooling layers, while Model 2 contains 3 convolution layers with 3 max pooling layers. On each model, as an **extra creativity** step, we then utilized both SGD and Adam optimizers and observed their performance given various learning rates. The results of the testing accuracy can be seen in Table 2.

| Model | Learning Rate | Optimizer | Accuracy |
|---|---|---|---|
| Model 1 | 0.01 | SGD | 0.89450001716613 |
| Model 1 | 0.001 | SGD | 0.84039998054504 |
| Model 1 | 0.1 | SGD | 0.90069997310638 |
| Model 1 | 0.001 | Adam | 0.91079998016357 |
| Model 1 | 0.1 | Adam | 0.10000000149011 |
| Model 1 | 0.01 | Adam | 0.89529997110366 |
| Model 1 | 0.0001 | Adam | 0.89810001850128 |
| Model 1 | 0.0005 | Adam | 0.90350002050399 |
| Model 2 | 0.01 | SGD | 0.85610002279281 |
| Model 2 | 0.001 | SGD | 0.78070002794265 |
| Model 2 | 0.1 | SGD | 0.88690000772476 |
| Model 2 | 0.001 | Adam | 0.89520001411437 |
| Model 2 | 0.0001 | Adam | 0.89179998636245 |
| Model 2 | 0.0005 | Adam | 0.88459998369216 |

**Table 2 :** Test accuracies given learning rate and optimizer

We see that for Model 1, overall, the test accuracies are almost equal or higher when using Adam rather than SGD. The same occurs for Model 2, but we don't see a significant increase in accuracy from Model 1 (which has less layers), and in some cases the accuracy of Model 2 was actually significantly lower than in Model 1. Thus, we will use Model 1 and the Adam optimizer in attempt to find the optimal learning rate value.

For Model 1 using Adam optimizer (Figure 10), we see that many learning rates achieve a high and similar accuracy. But when looking at the loss (Figure 11) we see that a learning rate of 0.0001 is the only learning rate that is consistently decreasing. So, we will use CNN Model 1, with a learning rate of 0.0001, Adam optimizer, and 90 epochs to train our model. After training this model, we received a test accuracy of about 90.16%

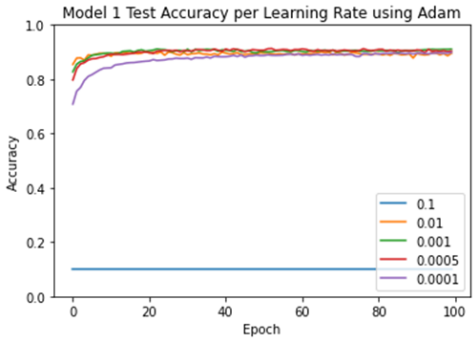


**Figure 10:** Model 1 Test Accuracy per Learning Rate with Adam Optimizer


**Figure 11:** Model 1 Test Loss per Learning Rate with Adam Optimizer

This updated model now has a higher accuracy than the highest we achieved with MLP. This result seems appropriate since with the MLP we used SGD and for the CNN we used Adam, which utilizes both momentum and an adaptive learning rate to improve performance. Also, we know that CNN's have incredible performance for image classification, explaining the boost in accuracy we observed.

### 3.6 Describing the best MLP Architecture and reasoning

We chose to use 1 hidden layer when tuning the hyperparameters because, as Figure 4 shows, the difference in performance between 1 and 2 hidden layers is negligible but using 1 hidden layer requires much less computation power and resources.

While hyperparameter tuning, we found that large learning rates gave really bad accuracies. This is reasonable as large rates cause fast conversion and thus not great results.

Adding dropout caused a slight decrease in test accuracy in all but 2 cases. However, it had a profound effect when the batch size was large (e.g. 500). Without dropout, our MLP using Leaky ReLU and a batch size of 500 achieved an accuracy of only 10%. Configuring the dropout to retain 80% of the nodes at each layer salvaged the accuracy and boosted it to 86%. The opposite effect occurred when the MLP used ReLU and a batch size of 500, where adding dropout cut the accuracy from 88% to 10%. Setting the dropout to 50% salvaged the accuracy and boosted it to 83%. There was little difference between retaining 80% and 50% of the nodes otherwise.

Our tuning led us to decide on ReLU activation with no dropout, 500 batch size, learning rate of 0.001, with 1 hidden layer consisting of 128 units as the best MLP architecture.

### 3.7 Model accuracy when varying the number of Epochs:

For this part, our group has included in every section the graphs of accuracies varied over the number of epochs. From Figures 4 and 7 it can be seen that in general, accuracy increases as we increase the number of epochs; however on average after about 20-30 epochs, the accuracy becomes constant. Hence, we can consider using 20-30 epochs to avoid any overfitting in the data.

## Discussion and Conclusion

Through all our experiments, we came to some important conclusions such as increasing the number of nodes that get dropped decreases accuracy and too many epochs leads to overfitting. Using this knowledge and best hyperparameter search, we found our best MLP model to give an accuracy touching 89% on unseen data. This model had 1 hidden layer with ReLU activation and 128 units and a learning rate of 0.001 with no dropout. Although we were able to improve the performance of our MLP model, overall, our CNN models still performed at least slightly better when using the Adam optimizer. As noted previously, this is to be expected due to CNN's ability to perform image classification and Adam's improved performance compared to SGD. Regularization also proved ineffective overall because our training set was small enough such that our model was never overfitted.

Possible further investigation may include testing more hyperparameters for both MLP and CNN. Such as increasing the depth and changing the number of hidden units for MLP, and testing more architectures and activation functions for CNN. We could also try different activation functions at each layer; our model can only use one activation function at a time and applies it to all layers.

## References

S. Bhatnagar, D. Ghosal and M. H. Kolekar, "Classification of fashion article images using convolutional neural networks," 2017 Fourth International Conference on Image Information Processing (ICIIP), 2017, pp. 1-6, doi: 10.1109/ICIIP.2017.8313740.

## Statement of Contributions

Alex Wang: Task 1 and Task 3
Julia Sokolov: Task 2 and Task 3
Saumyaa Verma: Task 2 and Task 3