# Student Number: 234591

## 1. Abstract

In recent years, deep learning has made remarkable progress, with Convolutional Neural Networks (CNNs) playing a pivotal role in various image classification tasks. Despite their success, the performance of these networks is highly dependent upon the careful selection of hyperparameters. A common misconception prevails that certain hyperparameter values are universally superior to others. However, designing and tuning Neural Networks is more of an art than a science, often necessitating the exploration of a diverse range of values to identify the most suitable configuration for a given problem. Therefore, We created the hypothesis below to demonstrate this:

”***The optimal configuration of learning rates, weight initializations, data transformations, and batch sizes is task-specific and depends on the underlying data-set characteristics and complexity of the image classification problem***”

In this report, I will describe the methodology taken in this experiment, the results and then end with a discussion examining the results and the implications of this relating to our hypothesis.

## 2. Methodology

In this section I will methodology taken in examining this hypothesis. I will start by explaining the data-set used then I will describe the four custom neural network architectures used, explaining each model in detail and close off with a discussion of the hyperparameters and data augmentations tested.

**Fashion MNIST data-set** We used the FashionMNIST dataset, a widely-known dataset for image classification tasks. The dataset consists of 60,000 training images and 10,000 test images, each of size 28x28 and depicting 10 different classes of clothing items [7]. We divided the dataset into training and test sets and loaded them using DataLoader with a batch size of 128. Later, we also experimented with batch sizes of 64, 256, and 512.

**Custom Neural Network Architectures** In order to discover the best model to use moving forward we will test multiple architectures and use the most successful one. As we are limited to 3 hidden layers, that leaves me with 4 choices on the architecture to use; two convolutional layers with one fully connected layer, two fully connected layers with one convolutional layer, three fully connected hidden layers and three convolutional hidden layers. After training and testing each model architecture on the Fashion MNIST data-set, I will use this architecture as the model to be tested when examine the chosen hyperparameters.

- **Two convolutional layers and one fully connected layer**: his model will consist of two convolutional layers, each with 32 and 64 filters of size 3x3 and padding of 1, respectively. A max-pooling layer with kernel size 2x2 follows the second convolutional layer. The output will then be flattened and passed through a fully connected layer with dimensions (64 * 14 * 14, 128) using the ReLU activation function. The final output will be produced by a second fully connected layer with dimensions (128, 10).
- **Two fully connected layers and One convolutional layer**: This architecture will comprise of one convolutional layer with 32 filters of size 3x3 and padding of 1, followed by a max-pooling layer with kernel size 2x2. The output will then be flattened and passed through two fully connected layers with dimensions (32 * 14 * 14, 64) and (64, 128), both using ReLU activation functions. Finally, a third fully connected layer with dimensions (128, 10) generates the output.
- **Three fully connected hidden layers**: This model will consist of four fully connected layers with dimensions (28 * 28, 64), (64, 128), (128, 256), and (256, 10). The ReLU activation function will be applied to the output of the first three layers, while the last layer produces the final output.
- **Three convolutional hidden layers**: This model will consist of three convolutional layers, each with 32, 64, and 128 filters of size 3x3 and padding of 1, respectively. A max-pooling layer with kernel size 2x2 follows the third convolutional layer. The output is then flattened and passed through a fully connected layer with dimensions (128 * 7 * 7, 10) to produce the final output.

[4]

in addition to these architectures, we will explore various weight initialization techniques, including Xavier uniform, Xavier normal, Kaiming uniform, and Kaiming normal. The chosen weight initialization method will applied to model with the highest accuracy.

**Hyperparameters** We will explore various hyperparameters to understand their impact on model performance. For learning rates, we will test the values of 0.0001, 0.001, 0.01, 0.1, and 1. For batch sizes will will test values of 64,128,256,512. The number of training epochs will be set to 10. A momentum of 0.9 will be used for the optimizer. We will also perform data augmentation of the Fashion MNIST images to see if it affect the loss and accuracy result. We will apply four different sets of transformations:

- **Standard normalization**: Conversion to tensors followed by normalization with mean 0.5 and standard deviation 0.5.
- **Random horizontal flip**: Images are randomly flipped horizontally before being converted to tensors and normalized.
- **Color jitter**: Random adjustments in brightness, contrast, saturation, and hue are applied before converting images to tensors and normalizing.
- **Random rotation**: Images are randomly rotated up to 10 degrees before being converted to tensors and normalized.

[3]

For cross-validation, we will employ a 5-fold strategy. The training set will be divided into five equal-sized folds, and each fold will be used as a validation set once. This allowed us to better understand the model's performance, generalization across different data splits and

Lastly we will compare the correlation matrices of neuron activities for a pre-trained ResNet-18 model and the chosen neural network. The comparison will be done on a subset of the FashionMNIST dataset.

## 3. Results

In this section I will display the results which will then be evaluated in the discussion section

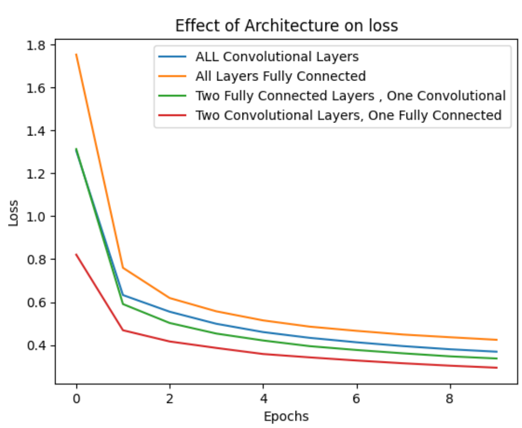


Figure 1. A graph demonstrating how the model architecture affects the loss value against the number of epochs

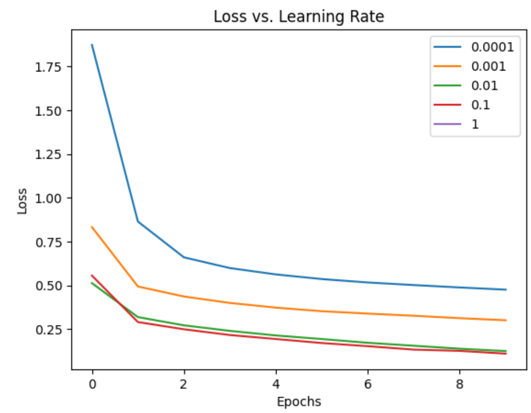


Figure 2. A graph demonstrating how the learning rate affects the loss value against the number of epochs

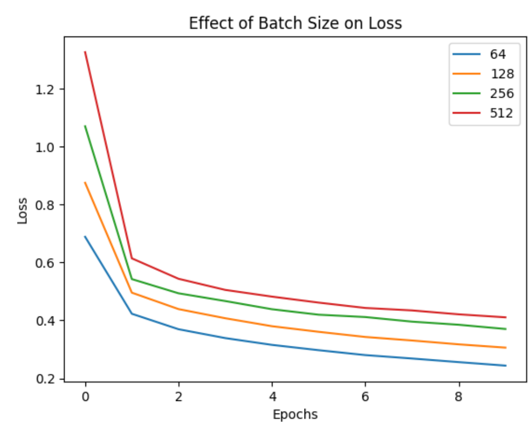


Figure 3. A graph demonstrating how the batch size affects the loss value against the number of epochs

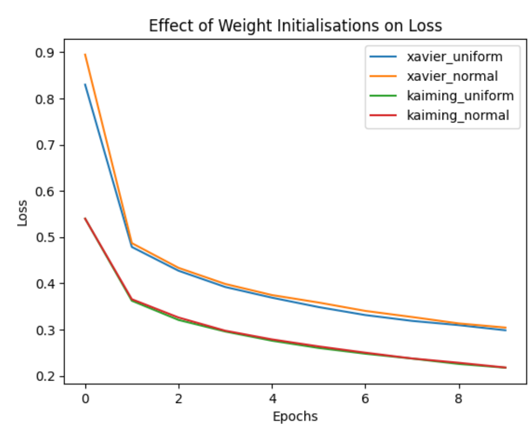


Figure 4. A graph demonstrating how the different types of weight initialisation affects the loss value against the number of epochs

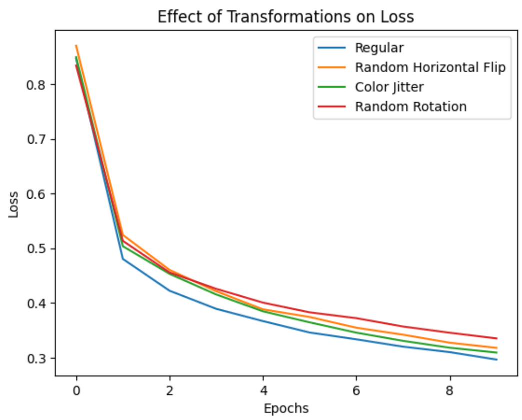


Figure 5. A graph demonstrating how the transformations affects the loss value against the number of epochs

## 4. Discussion

**Model Architecture** Based on these results, it can be concluded that the model with two convolutional layers and one fully connected layer performed the best in terms of both loss and accuracy scoring 88% in accuracy and loss of 0.2942. This could be due to the fact that convolutional layers are better at extracting features from the image input data which can then be efficiently used by the fully connected layer for classification. The combination of theses two types of layers seems to provide a balance between feature extraction and classification. However in hindsight, experimenting with different types of layers, such as dropout or batch normalization could have potentially improved the results. [1]

**Learning Rate** The learning rate is crucial in the optimization process of neural networks. It determines the step size at each learning iteration while moving towards the lowest point in a loss landscape. The best scoring learning rate was 0.01 which scored an accuracy of 91.61% and a loss of 0.0484 over 10 epochs. A surprising revelation was discovering that the learning rate of 0.1 scored 87.43% which could be due to this specific combination of architecture, optimizer and weight initialization. Referring back to the hypothesis, this demonstrates that the success of the hyperparameter value depends on the task at hand. Despite 0.1 being a large learning rate, there is either some over fitting or the value works effectively but not optimally.[6]

**Weight initialization** Weight initialization is the process of defining the initial values of the weights in a neural network before training starts. The Kaiming normal initialization performed the best in both loss and accuracy scoring a 90% accuracy and loss of 0.2208. This could be attributed to the fact that Kaiming initialization are designed specifically for ReLU activation functions which is used in this deep learning model. They aim to preserve the variance of the activation throughout the layers, which can lead to better learning dynamics. In retrospect, I could have experimented with other types of weight initialization and use different activation functions in combination with these initialization. [8]

**Batch Size** Batch size is the total number of training examples used in a single batch updates of the model parameters. The selection of batch size is significant for various reasons such as an improvement in computational efficiency, generalization and memory usage. The best batch was the batch of 64 which scored 89% accuracy and a loss of 0.2440. This is likely due to that fact that a smaller batch size provides more accurate estimates of the gradient yet still being noisy. As a result of the larger number of updates, the batch 64 model was more robust. However, this larger number of updates decreases the computational efficiency which asserts that there is a trade-off between the accuracy of the model and the computational efficiency during training.

**Data Augmentation** Data Augmentation is strategy that increases the diversity of the training data by applying various transformations which may improve the model's ability to generalize and extract features from the image.[2] The best model, which used basic normalization with no additional transformation , scored an accuracy of 87.99% after 10 epochs. However, the differences between the results were very low which might indicate that the specific data augmentation techniques used may have a minor impact on the model in this context which is reflected in the standard deviation of all the accuracies being 0.003357. If I was to improve this experiment then no only would I combined different data augmentation techniques, but I would also examine how the varying degrees of the intensity of augmentations affect the model's performance.

**ResNet Comparison** ResNet is a type of convolutional network that differs from other deep learning models in the fact that it can 'skip connections' which allow the output of one layer to be added to the output of another layer further down the line. This means that the vanishing gradient problem is mitigated. [5] When comparing the matricies of my architecture and the ResNet architecture using standard deviation and the mean, my model's mean was higher which mean that the neurons in the final layer activate more similarly. This means that my models neurons are less diversified and my model may be over-fitting to certain features. Furthermore, my model's standard deviation was higher which means that the correlations between neurons varied widely which makes sense due to the large number of classes this model was trained to classify into.

**Conclusion** To conclude this experiment, we created a deep learning architecture to classify images from the FashionMNIST data-set into 10 classes. I discovered that the best model consisted of two convolutional layers and one fully connected layer. For the hyperparameters examined, this was the best values for each:

- Learning Rate - 0.01 - I was surprised as this is quite a large learning rate
- Weight Initialization - Kaming normal initialization
- Batch Size - 64
- Data Augmentation - basic normalisation

The experiment was a medium-success in proving that each model requires experimentation to investigate which hyperparmeters work best for the task as a learning rate of 0.01 was the most successful despite its largeness, which people would not have predicted. However, I cannot completely assert this success as the ResNet comparison does imply there is some overfitting as well as strange values like a learning rate of 0.1 producing an 87% success rate.

## 5. Appendix

| Model Type | Epoch | Loss | Accuracy |
|---|---|---|---|
| ALL Convolutional Layers | 1 | 1.3058 | - |
| ALL Convolutional Layers | 10 | 0.3688 | 86.09% |
| All Layers Fully Connected | 1 | 1.7539 | - |
| All Layers Fully Connected | 10 | 0.4242 | 83.73% |
| Two Fully Connected Layers, One Convolutional | 1 | 1.3133 | - |
| Two Fully Connected Layers, One Convolutional | 10 | 0.3371 | 86.90% |
| Two Convolutional Layers, One Fully Connected | 1 | 0.8212 | - |
| Two Convolutional Layers, One Fully Connected | 10 | 0.2942 | 88.87% |

Figure 6. Table of model architecture results

| Learning Rate | Epoch | Loss | Accuracy |
|---|---|---|---|
| 0.0001 | 1 | 0.6427 | - |
| 0.0001 | 10 | 0.3072 | 87.98% |
| 0.001 | 1 | 0.4882 | - |
| 0.001 | 10 | 0.1780 | 90.67% |
| 0.01 | 1 | 0.4471 | - |
| 0.01 | 10 | 0.0484 | 91.61% |
| 0.1 | 1 | 0.6139 | - |
| 0.1 | 10 | 0.2383 | 87.43% |

Figure 7. Table of learning rate results

| Batch Size | Epoch | Loss | Accuracy |
|---|---|---|---|
| 64 | 1 | 0.6887 | - |
| 64 | 10 | 0.2440 | 89.20% |
| 128 | 1 | 0.8751 | - |
| 128 | 10 | 0.3061 | 88.00% |
| 256 | 1 | 1.0701 | - |
| 256 | 10 | 0.3705 | 85.70% |
| 512 | 1 | 1.3259 | - |
| 512 | 10 | 0.4107 | 83.06% |

Figure 8. Table of batch size results

| Weight Initialization | Epoch | Loss | Accuracy |
|---|---|---|---|
| Xavier Uniform | 1 | 0.8524 | - |
| Xavier Uniform | 10 | 0.3006 | 87.91% |
| Xavier Normal | 1 | 0.8042 | - |
| Xavier Normal | 10 | 0.2950 | 88.65% |
| Kaiming Uniform | 1 | 0.5379 | - |
| Kaiming Uniform | 10 | 0.2184 | 89.89% |
| Kaiming Normal | 1 | 0.5312 | - |
| Kaiming Normal | 10 | 0.2208 | 90.07% |

Figure 9. Table of weight initialization results

| Data Augmentation | Epoch 1 Loss | Epoch 10 Loss | Final Accuracy (%) |
|---|---|---|---|
| ToTensor, Normalize | 0.8469 | 0.2964 | 87.99 |
| RandomHorizontalFlip, ToTensor, Normalize | 0.8704 | 0.3178 | 87.50 |
| ColorJitter, ToTensor, Normalize | 0.8496 | 0.3092 | 87.92 |
| RandomRotation, ToTensor, Normalize | 0.8341 | 0.3351 | 87.16 |

Figure 10. Table of data augmentation results

| Fold | Epoch 1 Loss | Epoch 10 Loss | Fold Accuracy (%) |
|---|---|---|---|
| 1 | 0.5017 | 0.1844 | 91.33 |
| 2 | 0.1928 | 0.1075 | 93.88 |
| 3 | 0.1251 | 0.0583 | 96.12 |
| 4 | 0.0727 | 0.0291 | 97.45 |
| 5 | 0.0414 | 0.0140 | 99.07 |

Figure 11. Table of cross validation results on a learning rates of 0.01

| Model | Correlation Matrix Shape | Correlation Mean | Correlation Standard Deviation |
|---|---|---|---|
| ResNet-18 | (512, 512) | 0.0221 | 0.1641 |
| Your Model | (10, 10) | 0.1635 | 0.5271 |

Figure 12. Table of Resnet model comparison results

## References


[1] Convolutional neural networks (cnns / convnets). 3

[2] Data augmentation — how to use deep learning when you have limited data, 2018. 3

[3] P. developers. torchvision.transforms — torchvision master documentation, 2019. Accessed: 2023-05-11. 2

[4] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016. 1

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 3

[6] P. Pandey. Understanding learning rates and how it improves performance in deep learning. 2018. 3

[7] Z. Research. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. 1

[8] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147. PMLR, 2013. 3