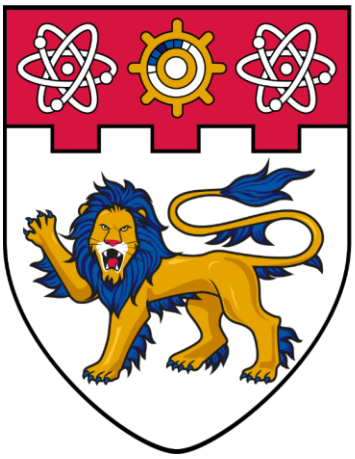

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ4042 Neural Networks and Deep Learning

AY 2023/24 Semester 1

Assignment Report

Project By:

Name	Matriculation Number
Amanda Ling Zhi Qi	U2022213G
Joey Quah (Ke Jieyi)	U2020067E
Jensen Lim	U2021364D

1 Introduction

The Fashion MNIST dataset is a freely available fashion dataset that contains 60,000 training images and 10,000 test images. These grayscale images have size 28x28 and are classified into 10 different categories. A common existing method used to classify this dataset is by building a Convolutional Neural Network (CNN) model. In [1], a CNN model was built from scratch and further improved by adding padding convolutions and increasing the number of filters. Their final model eventually achieved an accuracy of approximately 90.9%.

Although the accuracy attained by their model is considerably high, the research failed to consider many other hyperparameters that could be tuned beyond the convolutional layer. Furthermore, CNN is only one of the many neural networks available in deep learning, thus there could be other methods that are more suitable for image classification tasks and may achieve better results.

In this project, we investigated how to further improve on the CNN model, as well as explored various other models, such as transfer learning with ResNet and Vision Transformers. The main objective of this is to determine the most efficient and effective model for the Fashion MNIST dataset. This was done by comparing both the accuracy and the time taken to run.

2 Neural Network Models

2.1 Convolutional Neural Network (CNN)

CNN is a sequence of layers, and every layer of CNN transforms one set of activations to another through a differentiable function. The three main types of layers include: the Convolutional Layer, Pooling Layer, and Fully-Connected Layer.

The Convolutional Layer does most of the computational heavy lifting. It also includes the RELU activation function, which applies elementwise non-linearity. The Pooling Layer is usually in between successive Convolutional Layers, and this progressively decreases the spatial size of inputs to each layer to reduce the amount of parameters and computation, and control overfitting. The last Fully-Connected Layer is called the “output layer” and represents the class scores.

While [1] explored the effects of changing filter size and number of layers, we look to find out how the type of optimizer used, learning rates and dropout will further affect the CNN’s performance. Optimally, we want see how we can reach a higher accuracy score.

A. Filter Size

Increasing Filter Size:

In general, increasing the filter size can improve accuracy up to a certain point, after which it can start to decrease (Figure 1).This is because larger filters can capture more context from the input image, which can help the network to extract more complex features and make more accurate predictions. Larger filters can also capture more detailed information from the input image, which can lead to better feature extraction and improved accuracy. However, increasing filter size can also decrease accuracy because larger filters can increase the number of parameters in the network, which can lead to overfitting. Additionally, larger filters can discard some fine-grained information from the input image, which can be important for some tasks.

B. Number of Layers

Increasing Number of Convolutional layers:

Increasing the number of convolutional layers in a CNN can improve model accuracy by allowing the network to extract more complex and abstract features from the input data. Each convolutional layer learns a new set of features based on the features learned by the previous layer.

Adding Pooling Layer:

Adding pooling layer helps to reduce dimensionality of feature maps, hence reducing computational cost. It also makes the network more invariant to small translations, increasing accuracy.

However, as shown by our model, adding a pooling layer in between the first convolutional layer and second convolutional layer decreases the model accuracy instead. This could be due to the fact that the pooling layer has discarded some information from the feature maps, and if this information is important for the task at hand, then losing it can decrease accuracy. This is especially likely since the pooling layer is used too early in the network, before the convolutional layers have had a chance to extract all of the relevant features from the input image.

Increasing Number of Filters for Second Layer:

We increased the number of filters from the first convolutional layer to the second convolutional layer. This helps to improve performance of the network and increase accuracy as shown in Figure 1 and 2.

The higher the number of filters, the higher the number of abstractions that the model is able to extract from image data. At the input layer, the network receives raw data, and raw data is always noisy. Thus, at the first layer, we let the model extract some relevant information from the noisy raw pixel data first, and once the useful features have been extracted, we make the CNN extract more complex patterns from it.

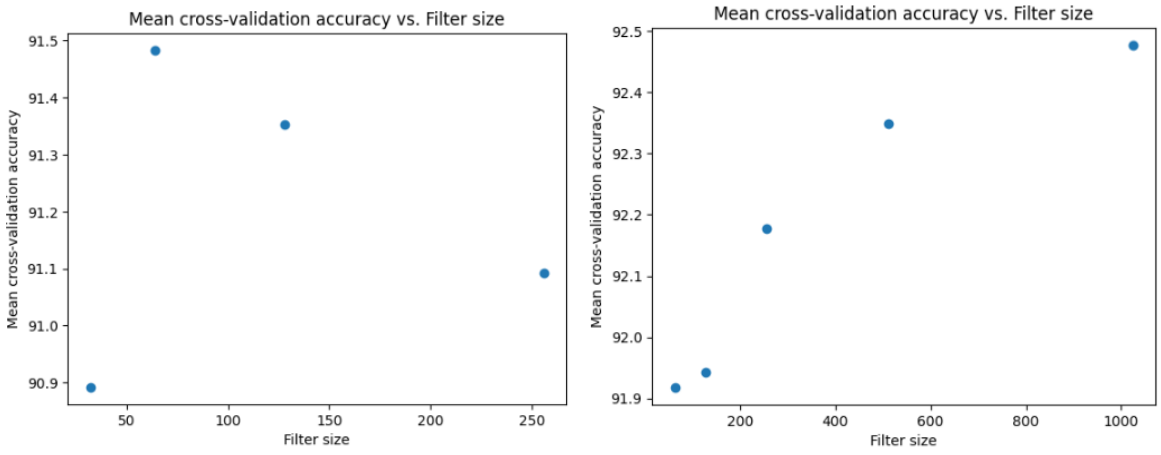

Figure 1 (left): Plot of accuracy over filter size for first layer
Figure 2 (right): Plot of accuracy over filter size for second layer

Optimal filter size for first layer = 64
Optimal filter size for second layer = 1024

C. Choosing Best Optimizer

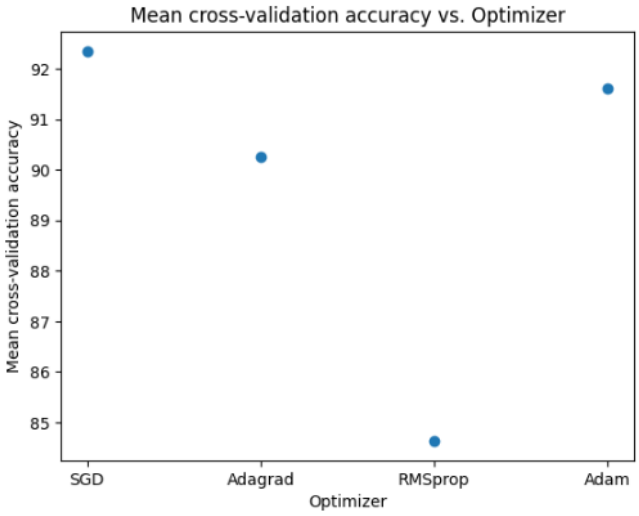

Figure 3: Plot of accuracy over Optimizer

Best Optimizer: SGD with momentum

SGD with momentum:

In deep learning architecture, the SGD algorithm tends to oscillate near the optimum, leading to very slow converging rates. Momentum helps to speed up the convergence.In SGD with momentum, we have added momentum in a gradient function, whereby the present gradient is dependent on its previous gradient and so on. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction. This accelerates SGD to converge faster and reduce the oscillation.

SGD vs Adagrad:

SGD performed better than Adagrad. Adagrad adapts the learning rate of all model parameters, and each parameter has its own learning rate. However, the reason why it performed worse than SGD could be due to the fact that at some point the learning rate is so small that the system stops learning [2]. Due to the accumulation of the squared gradients in the denominator, the accumulated sum keeps increasing during training, causing the learning rate to shrink and becoming infinitesimally small.

SGD vs RMSprop:

SGD performed better than RMSprop. RMSprop is an improved version of Adagrad so it also uses adaptive learning rate. Therefore, similar to Adagrad, SGD performs better for the same reasons above.

SGD vs Adam:

SGD performed better than Adam. This might be because although Adam converges faster, SGD generalises better, thus leading to improved accuracy. This is because SGD minimises the training time, decreasing generalisation error. The model will not see the data several times, and the model would not be able to memorise the data.

The generalisation error is the difference between training and validation error of models learned through SGD. One paper has proven mathematically that SGD is uniformly stable for strongly convex loss functions [3], thus might have optimal generalisation error.

SGD performing better than Adam may also be due to the fact that there are multiple global minima, and adaptive gradient methods like Adam find a solution that is worse than SGD because SGD will converge towards a minimum norm solution while Adam will diverge.

D. Learning Rate

Parameter	Learning Rate = 0.01	Learning Rate = 0.001	Learning Rate = 0.001 with EMA momentum	Learning Rate Scheduler
Accuracy	92.48%	92.03%	92.34%	92.68%

Table 1: Accuracies of different learning rate changes

Decreasing Learning Rate:
As shown, decreasing learning rate from 0.01 to 0.001 did not improve accuracy. This may be due to the fact that the model is already well-optimised when the learning rate = 0.01.

Adding EMA Momentum:
EMA momentum helps to smooth out gradients by computing the exponential moving average of the weights of the model and periodically overwriting the weights with their moving average, which makes the training process more stable and helps the model converge to a better solution.

Adding Learning Rate Scheduler:
After adding the learning rate scheduler, we can see that the accuracy is the same as when we have 3 convolutional layers of 92.68%, hence we have achieved good performance. Using a learning rate scheduler increases accuracy by allowing the model to learn at different speeds during different stages of training. In the early stages of training, the model is learning the basic features of the data. A high learning rate is useful at this stage because it allows the model to learn quickly. However, in the later stages of training, the model is fine-tuning its parameters and learning more complex features. A lower learning rate is useful at this stage because it prevents the model from overshooting the optimal solution.

E. Dropout

Parameter	Without Dropout = 0.25	With Dropout = 0.25
Accuracy	92.48%	90.70%

Table 2: Accuracies of model with and without dropout

Adding Dropout:
Adding a dropout of rate 0.25 decreases accuracy likely because when we dropout neurons during training, the model is not able to use all of the available information to make predictions. Furthermore, the model is constantly changing due to the neurons being dropped out, causing the model to oscillate and not converging to a good solution. This makes the training process more unstable.

F. Optimal Solution

Overall, the most optimal model built has 2 layers, with the first having a filter size of 64 and the second having a filter size of 1024. During training, the optimizer used is SGD while the learning rate is set to 0.01 with a learning rate scheduler to progressively decrease the learning rate. With this optimal model, an accuracy of 92.68% was obtained (Figure 4).

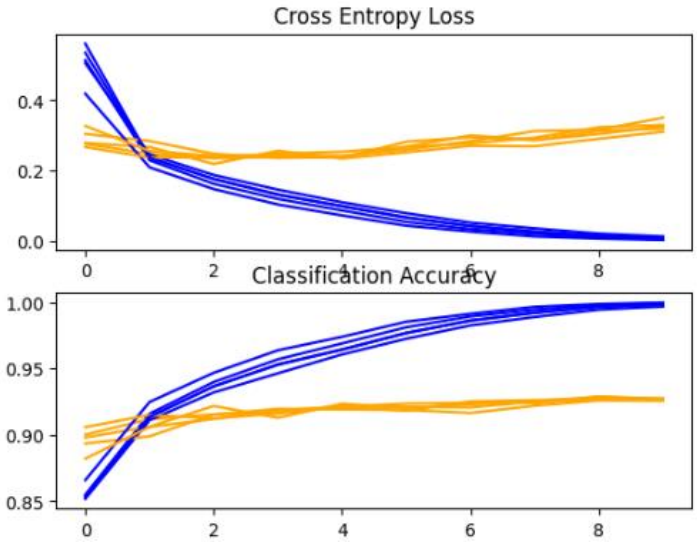

Figure 4: Loss and accuracy curve for optimal model

2.2 Transfer learning

In an attempt to achieve a higher accuracy than the CNN model, we went in a different direction that utilised methods beyond hyperparameter tuning and layer addition. We decided to explore the area of transfer learning, a powerful technique that enables us to draw on the knowledge encapsulated in pre-trained models.

Given that the Fashion MNIST dataset is relatively small, consisting of only 60,000 train images, it is more challenging to build a highly accurate model from scratch because deep learning models often thrive on vast volumes of data. As such, transfer learning takes advantage of pre-trained models that were developed on more extensive datasets, circumventing the constraints of dataset size and expediting the training process.

Furthermore, the clothing images in the Fashion MNIST dataset have unique textures, patterns, and styles. With pre-trained models having already learned intricate hierarchical features from a broad range of images, it becomes well-equipped to recognize patterns and textures in clothing items. This feature extraction capability streamlines the model's learning process and enhances its ability to comprehend the subtle nuances of fashion images.

Additionally, transfer learning mitigates the risk of overfitting, a common challenge when dealing with smaller datasets. By initialising our model with pre-trained weights, we provide it with a head start in learning essential features, reducing the chances of fitting the training data too closely and ensuring better generalisation to unseen data.

For our transfer learning model, we opted for the Residual Network (ResNet) model, specifically ResNet-18. ResNet-18 has a relatively simple architecture as compared to complex alternatives like ResNet-50. Given that the Fashion MNIST dataset has simpler patterns and features, ResNet-18 is most ideal because its shallower architecture prevents overfitting and lightens the computational load while achieving high accuracy rates.

For our initial experiment, we only trained the final fully connected layer of the ResNet-18 model with the Fashion MNIST dataset. However, this yielded an accuracy of 85%, which fell short of the CNN model's performance in earlier experiments. Hence, we tried increasing the number of layers that are being retuned to see if there is an improvement in overall accuracy. As the ResNet–18 model has four layers of ConvNets that are identical [4], we decided to increase the number of tuned layers by one residual block at a time. Finally, we also tried to tune the whole ResNet-18 model with the Fashion MNIST dataset. The results from our experiment were then plotted in the line graph shown in Figure 5 below.

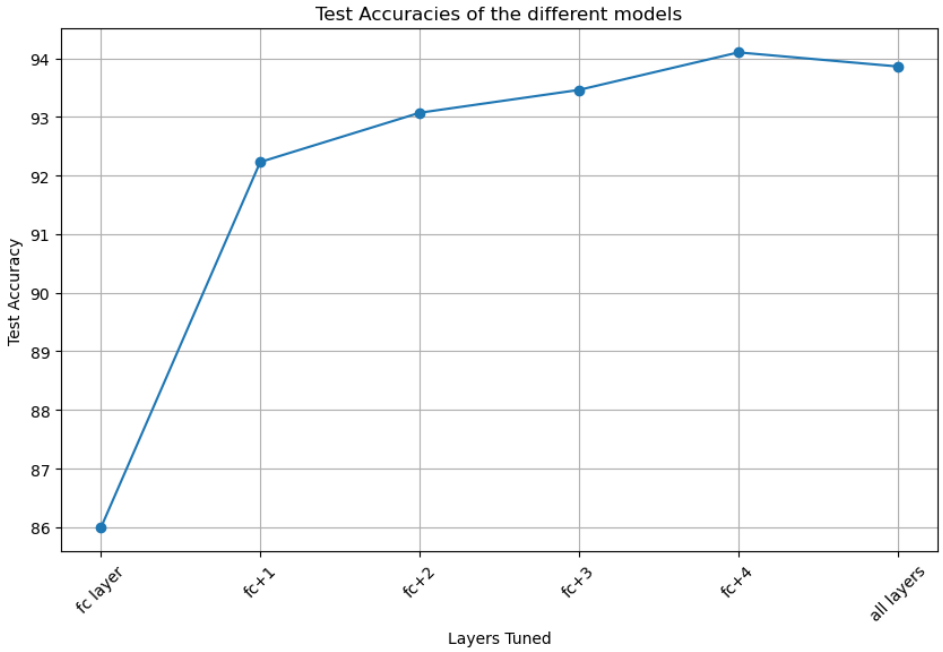

Figure 5: Test accuracies of the ResNet-18 model with different number of tuned layers

From the above results, it is generally indicative that the more the number of tuned layers, the higher the accuracy. However, there is a slight stagnation when it comes to the further stages where more layers are being tuned in the ResNet-18 model. One of the reasons for this is that ResNet-18 is a relatively deep architecture with a certain level of complexity. Thus, since the Fashion MNIST dataset is not as complex as the original dataset the ResNet-18 was trained on, adding complexity by tuning more layers does not improve our performance and leads to overfitting instead.

2.3 Vision Transformer

Vision Transformers (ViT) make use of the Transformer architecture on images by processing on sequence of image patches. Unlike CNN, ViT makes use of a self-attention mechanism and can model long-term global dependencies between image patches. As such, we want to compare the performance of ViT against CNN and transfer learning.

The implementation of our ViT was largely based off [5]. In the base model, we used 2 encoding layers, 2 attention heads and a hidden dimension of 16. The input image was reduced to a dimension of 64 by 64, and then split into 256 patches, each of size 4 by 4. With a learning rate of 0.005 and batch size of 512, the base ViT model achieved a test accuracy of 79.89% after 20 epochs of training.

In order to find the optimal performance of ViT on our dataset, we explored how 3 hyperparameters – *hidden_dim*, *n_encodelayers*, and *n_heads* – affected the accuracy of our model. These parameters were changed individually and compared against the base model. However, in the interest of time, the hyperparameter tuning was done on a smaller dataset, where only 60% of the data was used. Furthermore, the model was trained in 10 epochs, with a learning rate of 0.01 so as to compensate for the lower number of epochs. The base model was also retrained under these conditions for a more fair comparison.

The results of the hyperparameter tuning are as shown in Table 3 below. The leftmost column indicates the parameters that are being changed, as well as the corresponding values stated in the square brackets. The bolded values are that of the base model.

Changed Parameters	Test Accuracies		
Hidden_dim [8, **16**, 32]	74.70%	67.82%	69.32%
N_encodelayer [**2**, 4, 8]	67.82%	70.18%	73.75%
N_heads [**2**, 4, 8]	67.82%	71.03%	63.57%

Table 3: Test accuracies of each tuned model

First, the *hidden_dim* parameter, representing the dimension of the hidden layers, was initially set to 16. As such, we tested for *hidden_dim* of 8 and 32 and found that the optimal parameter was 8, which is the lowest of the three. With a larger hidden layer dimension, the model would have higher capacity to memorise the training data, potentially leading to overfitting. This is possible for our case as the test accuracy is lower for bigger hidden dimensions, especially since our training data size is small. Moreover, classifying the Fashion MNIST dataset is considered a less complex task, hence having a smaller hidden dimension size is sufficient enough.

Second, the number of encoder layers, *n_encodelayer*, was set at 2, and tested for an increase to 4 and 8 layers. Our results showed that a higher number of encoder layers would result in better accuracy. In a transformer, each layer captures different levels of abstraction in the input data. Lower layers tend to capture local and in-depth features while higher layers capture more global and abstract features. Hence, when the number of encoding layers increase, the model can better learn the hierarchical representations and relationships between the visual data during training. As a result, having a deeper network would increase the accuracy during testing.

Thirdly, the number of attention heads, given by *n_heads*, was also set at 2 and tested for 4 and 8 heads. The optimal number seems to be 4 attention heads. The accuracy increased as number of heads increased from 2 to 4, but dipped as it increased to 8. Having multi-head attention encourages the model to learn a greater variety of patterns in the input data, giving rise to the increase in accuracy as the number of heads changed from 2 to 4. However, having too many attention heads may lead to the problem of overfitting, particularly for small datasets which might not have enough diverse examples to

effectively train each head. Therefore, the model becomes poor in generalisation, leading to a decrease in accuracy.

Finally, we retrained the model using the optimal parameters and on the full dataset, for 20 epochs. The new model achieved a test accuracy of 80.44%, higher than that of the base model, which was at 79.89%.

3 Performance Comparison

3.1 Accuracy

Table 4 shows the test accuracies obtained by each optimal model, after hyperparameter tuning.

Model	CNN	Transfer Learning	Transformer
Test Accuracy	92.68%	93.86%	80.44%

Table 4: Test accuracies of each model

From the results, we can see that the transfer learning model has the highest accuracy out of the 3 models. The most likely reason for this was because the transfer learning was able to incorporate the weights that were trained from the ImageNet database which is much larger than the weights learnt from the Fashion MNIST dataset itself. Hence, with a better trained model, the accuracy will be the highest.

While CNN and transfer learning were able to achieve similar accuracy results of more than 90%, the Vision Transformer model attained a much lower accuracy of 80.44%. This difference is likely attributed to transformers requiring much larger amounts of training data in order to achieve comparable performance to CNNs. As mentioned earlier, the transformer performance was greatly affected by the dataset size, as it led to problems of overfitting and poor generalisation. Transformers have a larger number of parameters than CNN, making it a larger, more complex model that would benefit more from having a more varied dataset.

3.2 Time Taken

Table 5 presents the average training time taken per epoch for each of the models.

Model	CNN	Transfer Learning	Transformer
Average Time Taken (per epoch)	42.78 seconds	1.66 minutes	30.70 minutes

Table 5: Average training time for each model per epoch

The Vision Transformer model took the longest amount of time per epoch our of the three, indicating that it is the least computationally efficient. One reason for this is because Vision Transformers have larger model sizes and higher memory requirements than CNNs. Since transformers heavily utilises the self-attention mechanism to capture global dependencies, it becomes more computationally intensive as compared to CNN which uses local receptive fields [6]. Therefore, the training time taken for transformers would be the highest.

On the other hand, transfer learning used the least amount of time. The most likely reason for this is that transfer learning only requires the tuning of specific layers in the model instead of training the whole model like CNN and Transformer. Furthermore, as the ResNet-18 model leverages on features from a larger dataset, this knowledge enables quicker convergence resulting in the shortest training time.

4 Data augmentation

As seen in the analysis of our results, it is found that one problem of the Fashion MNIST dataset is that there is an insufficient number of training data points. This led to stagnation for transfer learning and a much lower performance for the Vision Transformer. As such, we decided to examine the use of data augmentation to artificially increase the volume of data used to train the models. More specifically, we looked into image augmentation methods CutMix and MixUp.

CutMix involves mixing two images together to create a new image. CutMix works by first randomly selecting two images from the training set. Next, it randomly selects a rectangular region from one of the images and replaces it with the corresponding region from the other image. The resulting image is then used as a new training sample. [7]

MixUp creates new training samples by linearly interpolating pairs of existing examples, using their input features and corresponding labels. It generates a weighted combination of random image pairs from the training data [8].

In our implementation, we applied a CutMix alpha of 1.0 and a MixUp alpha of 0.2. These alpha variables control the strength of the pasting and mixing operations respectively. Table 6 below displays the test accuracies of each model with data augmentation.

Model	CNN	Transfer Learning	Transformer
Test Accuracy	92.22%	91.54%	71.25%

Table 6: Test accuracy for each model after data augmentation

4.1 CNN

Data augmentation was carried out on the CNN model by implementing CutMix. Although CutMix is proven to be effective for improving the performance of deep learning models on a variety of tasks, it did not improve the accuracy of the model as the accuracy dropped from 92.68% to 92.22%. A possible reason could be that the CNN model is not complex enough. If the model does not have enough capacity to learn the complex relationships in the data, then data augmentation may not be able to provide enough information for the model to improve.

4.2 Transfer Learning

CutMix and MixUp were implemented on the ResNet-18 model with 4 tuned layers. In the first run, the model achieved an accuracy of 91.06% which is lower than the model without data augmentation. Upon closer look, we realised is that this could be due to the insufficient number of epochs being used. Although the training of the previous few models without data augmentation also used 10 epochs, the validation accuracy of those models converged towards to end. In contrast, the validation accuracy of the

ResNet-18 model with data augmentation can be still be seen increasing even towards the 10th epoch. Thus, we decided to increase the number of epochs for this particular model.

However, despite increasing the number of epochs to 24 with the validation accuracy stabilising, the test accuracy is at 91.54%, which is still lower than the original model. This is most likely due the introduced data augmentation being too intense or unsuitable for the characteristics of the Fashion MNIST dataset, leading to overfitting during training which was counterintuitive to what we originally wanted data augmentation to help us with.

4.3 Transformer

CutMix and MixUp were also implemented on the final ViT model. After training for 20 epochs, the model reached an accuracy of 70.82%, which is lower than the the model without augmentation. It is likely that the number of training epochs used was not sufficient enough for the model to adapt to the augmented data. The learning rate used may also not be the optimal value for this. As such, further training and hyperparameter tuning might be needed for a higher accuracy to be attained. Hence, we tried training for another 5 epochs, which resulted in an accuracy of 71.25%, still lower than the model without augmentation.

Similar to transfer learning, a reason for this could be that the augmentation might have disrupted the self-attention mechanisms used by the model to capture long-range dependencies. Therefore, such augmentation is unsuitable for this dataset, leading to low accuracy scores.

5 Conclusion

Overall, our project managed to improve on the existing CNN model and also discover other models to be used for an image classification task. We also tried out more advanced regularisation techniques like data augmentation, which may not prove to be so useful in our case.

Amongst the models we tested out, transfer learning gave the best performance both in terms of achieving the highest accuracy and relatively lower training times. Contrastingly, the Vision Transformer gave disappointing results, having the lowest accuracy scores and significantly long training times. However, this does not mean that Vision Transformers are ineffective for image classification. Rather, it is less effective for the Fashion MNIST dataset due to the features and size of this dataset.

Moreover, given the constraints of time and computational resources, it is possible that we were unable to fully explore the optimality of the Vision Transformer, especially in terms of number of training epochs. We could also have tried out different learning rates, optimizers and other advanced methods that could bring up the accuracy of the model.

6 References

[1] J. Brownlee, “Deep Learning CNN for Fashion-MNIST Clothing Classification,” *MachineLearningMastery.com*, May 09, 2019. Available: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-fashion-mnist-clothing-classification/. [Accessed: Nov. 10, 2023]

[2] G. Mayanglambam, “Deep Learning Optimizers,” *Towards Data Science*, Nov. 18, 2020. Available: https://towardsdatascience.com/deep-learning-optimizers-436171c9e23f. [Accessed: Nov. 11, 2023]

[3] M. Hardt, B. Recht, and Y. Singer, “Train faster, generalize better: Stability of stochastic gradient descent,” Sep. 03, 2015. Available: http://arxiv.org/abs/1509.01240. [Accessed: Nov. 11, 2023]

[4] P. Ruiz, “Understanding and visualizing ResNets,” *Towards Data Science*, Oct. 08, 2018. Available: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8. [Accessed: Nov. 11, 2023]

[5] B. Pulfer, “Vision Transformers from Scratch (PyTorch): A step-by-step guide,” *MLearning.ai*, Feb. 03, 2022. Available: https://medium.com/mlearning-ai/vision-transformers-from-scratch-pytorch-a-step-by-step-guide-96c3313c2e0c. [Accessed: Nov. 10, 2023]

[6] F. Rustamy, “Vision Transformers vs. Convolutional Neural Networks,” *Medium*, Jun. 04, 2023. Available: https://medium.com/@faheemrustamy/vision-transformers-vs-convolutional-neural-networks-5fe8f9e18efc. [Accessed: Nov. 10, 2023]

[7] “CutMix.” Available: https://paperswithcode.com/method/cutmix. [Accessed: Nov. 11, 2023]

[8] “Mixup.” Available: https://paperswithcode.com/method/mixup. [Accessed: Nov. 11, 2023]