ECE:657A
Project Question 1

Mohamed Sadok Gastli
20853612
ms2gastl@uwaterloo.ca

Yekta Demirci
20837581
ydemirci@uwaterloo.ca

1 1.1 Explanation of Design and Implementation Choices of your Model

After evaluation of several different algorithms, the highest accuracy is obtained by using principal component (PCA) and support vector machine(SVM). The other used techniques, their accuracy and performances are explained in section 1.4, not here, to follow the report format. After observing clear accuracy difference between PCA+SVM and the other techniques, PCA+SVM model is chosen and further tests were conducted to find the optimum parameters, the number of used PCs and SVM values are explained in section 1.4.

Principal component analysis is a way to map data to orthogonal axis by maximizing the variation. It is already well analysed in assignment 2. Yet, briefly, n dimensional data is mapped to another n dimensional space by maximizing the variance of the data in the light of co-variance matrix. It can be thought as rotating the n dimensional raw data and looking it to from a different perspective. Since variation is a key aspect for machine learning models to distinguish differences and predict, the dimensions which has low variance can be omitted and the data-set dimension can be reduced just using the principal components with high variation.

For the PCA implementation, scikit-learn library of Python is used. Assuming the data set is X and nxd dimensional where n is the number of samples and d is the number of features, the main computational cost arises from X^TxX while computing the covariance matrix, depending on the number of samples or the features(in other words using normal PCA or dual-PCA), the cost of computing the eigen-values will be either $O(p^3)$ or $O(n^3)$ in time[1].

The other used algorithm, support vector machines is a supervised learning algorithm for classification and regression analysis. Briefly, a hyper plane is obtained between different class samples by adding a margin between the hyper-plane and the samples. Then the different samples are classified based on which side of the hyper-plane they fall. Also the margin in the model is controlled by a regularization parameter which determines lenience of the model. In the most basic way, linear hyper plane can be used for classification, however by changing the loss function and penalties the shape of the hyper plane can be changed. Several different kernel is used in our approach and the highest accuracy is obtained by using radial basis function. The formula to calculate the distance by *rbf* is given below. x and xáre the points where the distance between is calculated. The sigma is a free parameters.

Intuitively, the gamma parameter defines how far the influence of a single training example reaches, with low values meaning 'far' and high values meaning 'close'.[2] In some case, gamma plays a similar role to the margin. When rbf is compared with the euclidean distance, it is clear

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\sigma^2}\right)$$

that, as the points get separated the distance value is more affected.

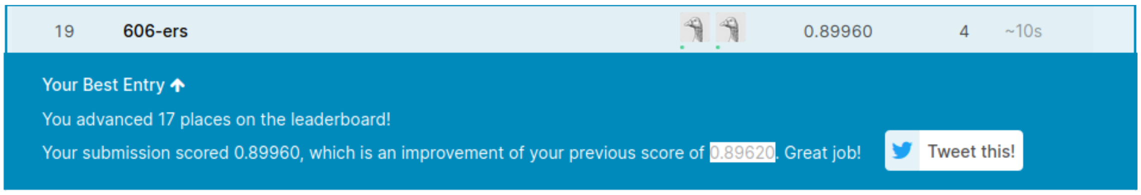
For SVM implementation with 'rbf' scikit-learn library of Python is used. SVM is a quadratic programming problem. It is more expensive than PCA. Given a dataset with *nxd* dimensions where n is the number of samples and d is the number of features, it is $O(n^3 * d)$ in time.[3] However, caching tricks are used as approximation to increase the efficiency, assuming the data-set is not sparse. In our case the data-set is not sparse which is given in section 1.4.

2 1.2 Implementation of your Design Choices

The implemented code blocks are uploaded to LEARN. There are 2 python scripts, *part1.py* and *part2.ipynb*. The first one was the part where several different implementations are used to select a model (PCA or LDA or Isomap or LLE)+(Decision Tree or SVM or KNN). After observing the accuracy, PCA+SVM model is selected and optimum parameters are found in the second python script. 10-fold cross validation is used to check the overall accuracy of the model. It is wise to not run the first script because your computer may crash due to insufficient RAM, however you can check the code in any Python IDE.

3 1.3 Kaggle Competition Score

The highest score is obtained on 19th of April as 19th with the accuracy of 89.62% on 4th submission. The kaggle leaderboard is quite dynamic and rankings frequently change. The people on the top 5 were having accuracy values around 90.1%. Assuming, all of the test samples(10k samples) are used for the leader-board evaluation; 0.4% difference corresponds to 40 more truly classified samples. Considering 10k samples classifying 40 more samples is clearly better yet not that different. So I am assuming, they use similar models but maybe with better parameters. If the accuracy difference was more than 3%, I would suspect the PCA+SVM model and look for a different approach. 0.4% difference may quickly change with using more test samples. Of course, the accuracy may slightly fall as well as it may slightly increase.



19	606-ers	0.89960	4	~10s

Your Best Entry

You advanced 17 places on the leaderboard!

Your submission scored 0.89960, which is an improvement of your previous score of 0.89620. Great job!

Figure 1: Kaggle leaderboard ranking

4 1.4 Results Analysis

First of all, the data-set consists of gray-scale images, yet it is scaled anyways before applying PCA. Even though the data-set had 784 features(28*28 images), most of the variance is covered by only first 50 PCs. This is actually not that surprising since the outer pixels of the images are just plain and cover no information at all and very redundant. The explained variance per PC is given in figure 2. After 15th PC, the explained variance falls below 1%. It shows us that 784 dimensions are very redundant.

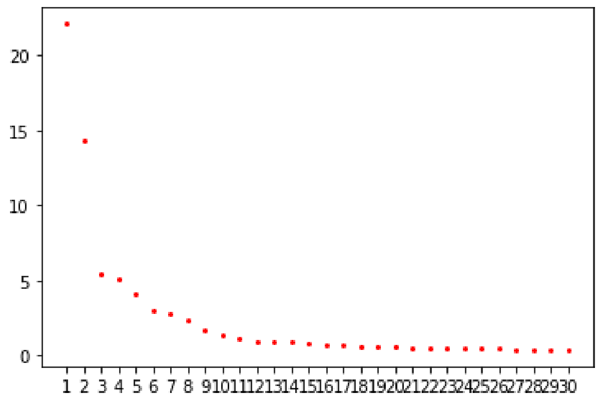


Figure 2: Explained variance by PCs

The distribution of the data in 3D is given in figure 3, by using first 3 PCs which explain the total variance of 41%. As it can be seen in figure 3, label 0 and 4(light blue and navy blue) they are quite distinct than others. However the other 3 labels are not heterogeneously scattered. Label 2(green) one is scattered also within navy blue. Label 1(brown) and 3 are likewise label2, mixed with others.

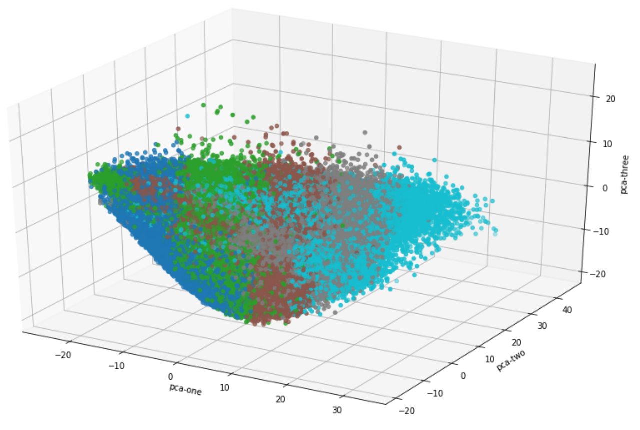


Figure 3: The first three PCs

Also the scatter plot of first 20PCs are given in figure 4 in a (PC(x) vs PC(x+1)) format. After the 8th PC, the variance falls and the data is tufted together. Distinguishable characteristics are not as high as the first 5 PCs at the first glance. After observing these plots, it can be said that, using PCA as a data-reduction is a good way to get rid of redundancy for this data-set. It is clear the data does not exist in a linear space. Therefore PCA still may not the best method to map this data. However, PCA requires less computation power compared to its counter-parts.

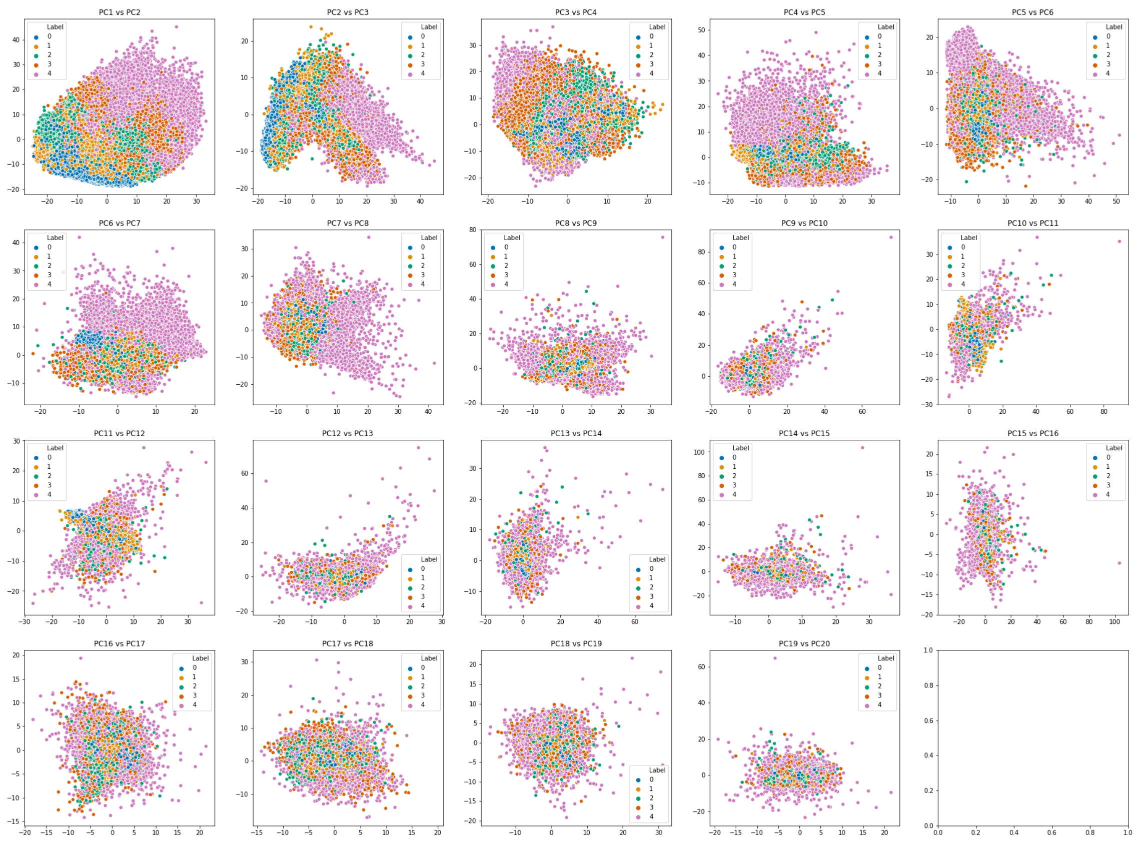


Figure 4: (PC(x) vs PC(x+1)) plots

Just to benchmark, LDA is also applied. Eventually, applying decision Tree, KNN or SVM performed less accuracy than PCA+SVM. However it is good to show the data distribution after LDA. At the first glance it seems more separable than PCA. It is good to note, relying on eye does not necessarily work to get better accuracy at the end. LDA is applied to 60k data points. Since there are 5 labels, the data is reduced to 4 dimensions. The scatter plot of the axis are given in figure 5. All 4 features or only first 2 are tried to be used with SVM and KNN models however the accuracy was around %79 with 10 fold cross-validation.

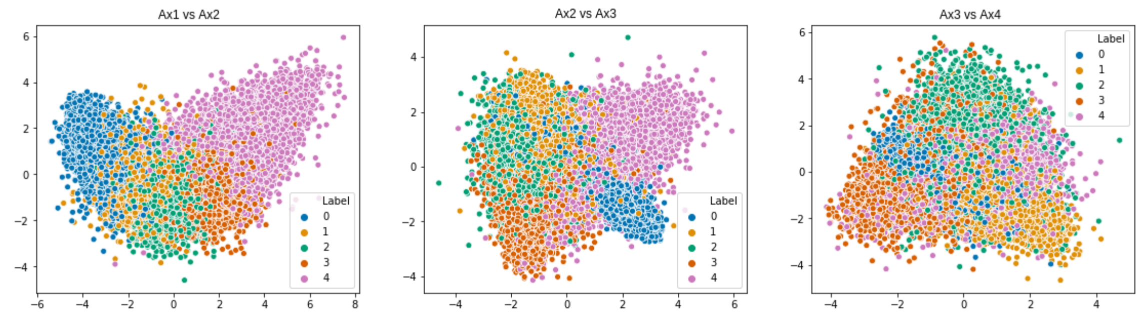


Figure 5: Scatter plots after LDA

Beside PCA and LDA models, Isomap,LLE and t-SNE models were also used. They are are computationally expensive in terms of both time and space. Through each operation, several other algorithms are used to construct graphs and finding similarities between samples. Since they are complex methods and have several steps, each step brings the question of optimization, like how many neighbors should be used while computing the graphs and how many dimensions should exist in the final mapping. Beside that using more than 10k samples were locking up the computer due to insufficient RAM. Yet, just in order to see their performance the training data is divided into 10k groups and first PCA was applied. Then LLE, Isomap and t-SNE methods were applied just to see if the feature reduction could be further improved after PCA. Also it is worth to mentioned that, LLE and t-SNE does not have transform method in scikit-learn. In other words, due to their nature; a model can not be trained by a training data-set and be applied to a test data-set. Each mapping is unique for its data-set. So, these methods might be useful to visualize the data but may not be suitable for training and testing purposes. Also, Isomap requires the same number of sample for both fitting and transforming. In other words, if the model is trained with 60k training samples, also 60k test samples are needed for the transforming. This problem could be solved by dividing training set into 6 groups and using the most occured predictions among 6.

10k random samples are chosen among the data-set by considering equal label distribution. Then PCA is applied, and for 10,20,50,80 PCs Isomap,LLE and t-SNE were used. Different number of neighbours are tried for isomap and LLE. Finally, decision trees with different depths and SVM with *rbf* are used however the accuracy was low so both the distributions and the accuracy plots are skipped. The highest accuracy was obtained around %80 with 10 PCs+t-SNE+decision tree with depth 10. Yet, it was still low than PCA+SVM combination. In addition t-SNE is computationally expensive and it is hard to transform the real test data with it, so it is not really applicable(at least t-SNE offered by scikit-learn).

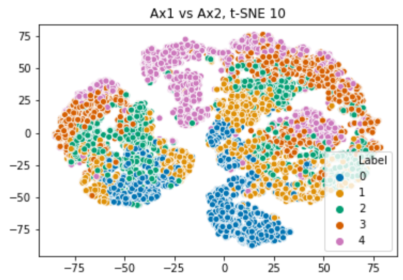


Figure 6: Scatter plot of t-SNE, using 10 PCs

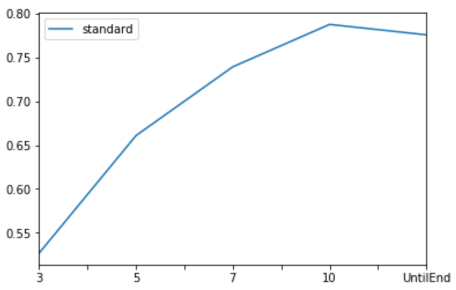
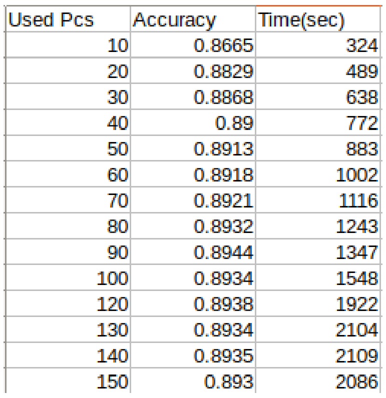


Figure 7: Decision tree with different depths over t-SNE data using 10 PCs

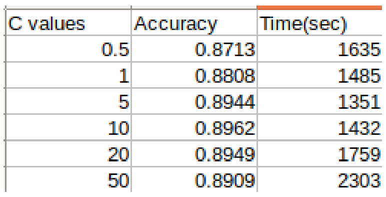
After realizing, the highest accuracy was obtained with PCA+SVM model, several runs were conducted with different PC and C values to get the best accuracy. Through these test, all of 60k samples were used with 10-fold cross validation. It definitely takes indefinite run time applying SVM directly onto the raw data. Even with PC values of 150, the calculation took around 35 minutes with Intel i7-6700HQ CPU at 2.60GHz. The run times and accuracy values for different PCs can be seen in figure 8. SVM with *rbf* with regularization parameter 5 and 10 fold cross validation are used. As it can be seen the best accuracy was obtained with 90PCs, even though the accuracy difference is very small with others. Accuracy values are given in the range of [0,1]. The important thing here is, the only changing variable is number of PCs. However the run time does not linearly increase with the used PC number. As it was given in part 1.1 the run time for SVM is $O(n^3 * d)$, it is linear in the size of features. This might be due to optimization techniques used through the implementation of SVM by scikit-learn. Also, the explained variance was less than %1 after first 20 PCs, however using more PCs still slightly increased the accuracy until PC-90.



Used Pcs	Accuracy	Time(sec)
10	0.8665	324
20	0.8829	489
30	0.8868	638
40	0.89	772
50	0.8913	883
60	0.8918	1002
70	0.8921	1116
80	0.8932	1243
90	0.8944	1347
100	0.8934	1548
120	0.8938	1922
130	0.8934	2104
140	0.8935	2109
150	0.893	2086

Figure 8: Run times and accuracy values for different PCs

After choosing PC value as 90, several different C values were used to find the optimum C value. The only changing variable is C for the results in figure 9 and 10-fold cross validation is used with all 60k samples. Lower C values encourage higher margin, therefore the decision function gets simpler. This affect can also be seen in figure 9 with the run times. Accuracy values are given in the range of [0,1]. For high C values as 50, the margin is smaller and the hyper-plane is shaped according to the test data, more samples are taken into the account and it affects the running time. It takes longer time to train and the accuracy slightly decreases due to over-fit. The best accuracy was obtained with C value 10.



C values	Accuracy	Time(sec)
0.5	0.8713	1635
1	0.8808	1485
5	0.8944	1351
10	0.8962	1432
20	0.8949	1759
50	0.8909	2303

Figure 9: Run times and accuracy values for different C values

After all these tests, the final model is trained by using all 60k samples, choosing only 90PCs, training a SVM model with *rbf* kernel and C value as 10.

The ROC curve for the final chosen metrics can be seen in figure 10. In order to obtain this curve the 60k set is divided into training and test set with the ratio of 0.8. ROC curve shows the output quality. As it can be seen, the points are closer to the top left where true positive is close to 1 and false positive is close to zero. This is ideal and desired. Here; true positive means, a sample is predicted as label x and its true value is also x. False positive represents it is predicted as x however the true value is not x. Also another thing to mention; before applying any predictive model it was observed label 0 and 4 were more separable than others in figure 3 and figure 4. Indeed the area under these classes are higher than others and they are classified slightly better than the other three labels.

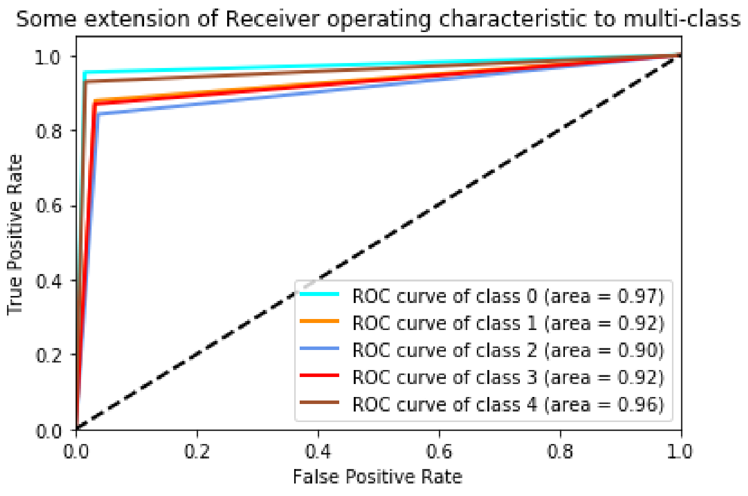


Figure 10: ROC curve for the final model

References

[1] Yousef Saad, Numerical Methods for Large Eigen Value Problems, available at: https://www-users.cs.umn.edu/ saad/eig$_{b}ook_{2}ndEd.pdf$

[2] RBF SVM parameters, Available at: https://scikit-learn.org/stable/auto$_{e}xamples/svm/plot_{r}bf_{p}arameters.html$

[3] SVM Complexity, Available at: https://scikit-learn.org/stable/modules/svm.htmlcomplexity

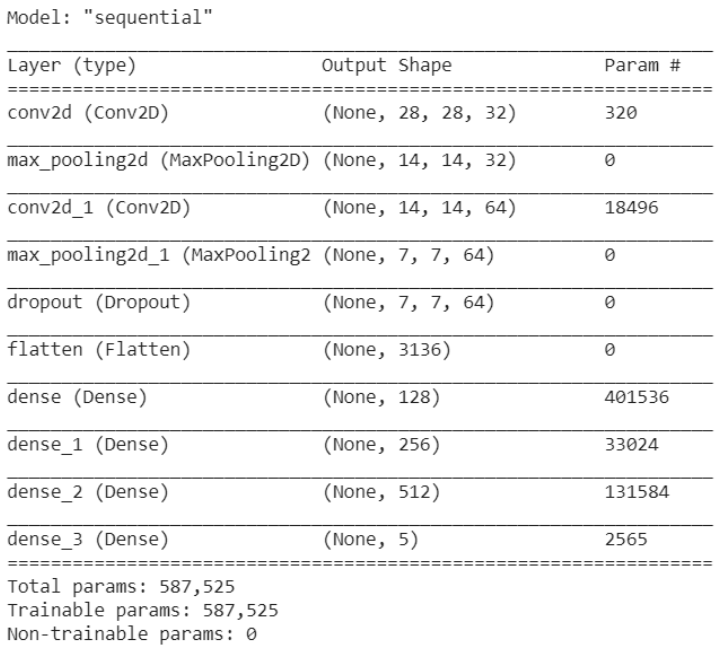
1. Deep CNN Model

Data description

The fashion MNIST dataset contain images of 5 different categories of clothing labelled accordingly. The acquired data had each image flatted into an array of 784 pixels, which had to be normalized and reshaped into an image of size 28x28 pixels before feeding it into the CNN model.

Model description

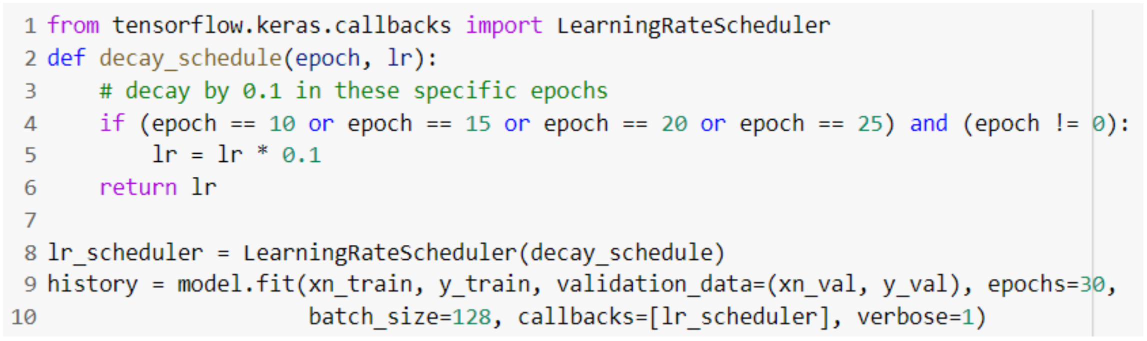
Figure 1 below describes the architecture of the deep CNN model chosen. The main library used is Tensorflow, which is commonly used for machine learning models. It is made up of 2 convolutional layers of incrementing size (32x32, 64x64 respectfully) separated by max pooling layers of stride 2 which would reduce the number of required parameters; which would, in turn, help reduce the computational complexity and help generalize. Those are then fed into 3 dense layers of incrementing size (128, 256, 512 respectfully) that follow the Rectified Linear Unit (ReLu) activation function. Finally, the output layer is made up of 5 nodes corresponding to the 5 classes, and uses the softmax function for classification. The loss function uses categorical cross-entropy, while the optimizer chosen was Adam with a starting learning rate of 0.001. For regularization, a dropout layer of 0.4 dropout is inserted before flattening into the dense layers; this would help reduce the overfitting of the model.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
dropout (Dropout)            (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 128)               401536
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_2 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 2565
=================================================================
Total params: 587,525
Trainable params: 587,525
Non-trainable params: 0
```

Figure 1 – CNN model architecture

When running the model, the learning rate is decayed at specific checkpoint epochs to allow for the model to fine-tune its learning with smaller increments as it trains longer. Figure 2 below shows the approach taken to implement this learning rate decay, which decays the learning rate by 0.1 (in others words, 1% of the rate is used) every 5 epochs starting from epoch 10. The epoch checkpoints and decay were chosen based on extensive testing of different values. Moreover, the model trains on the data in batches of 128 points every time, and runs for 30 epochs; both of which were chosen based on comparing the accuracy of different values and picking the best.



```python
from tensorflow.keras.callbacks import LearningRateScheduler
def decay_schedule(epoch, lr):
    # decay by 0.1 in these specific epochs
    if (epoch == 10 or epoch == 15 or epoch == 20 or epoch == 25) and (epoch != 0):
        lr = lr * 0.1
    return lr

lr_scheduler = LearningRateScheduler(decay_schedule)
history = model.fit(xn_train, y_train, validation_data=(xn_val, y_val), epochs=30,
                    batch_size=128, callbacks=[lr_scheduler], verbose=1)
```

Figure 2 – Learning decay and running model block

The model took about 2 seconds per epoch to train, with 30 epochs totalling to around a minute. Analysing the loss reduction, Figure 3 shows the loss function for both the training and validation sets of the model across the 30 epochs. It can be noted that both the training and validation losses drop significantly up until the 10th epoch, after which the improvement saturates. Moreover, the learning rate decay is reflected on the graph by a reduction of fluctuations after epoch 10. The training loss continues to drop lower than the validation loss, which indicates overfitting of the model since what is being learned by the model beyond that is not generalizable to all data.

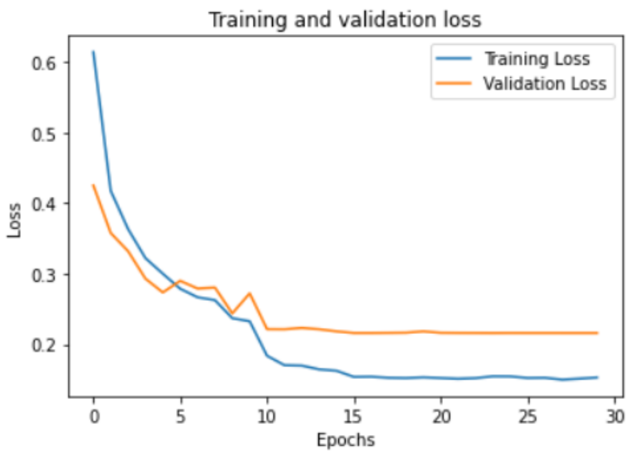


Figure 3 – Loss function of CNN model over different epochs

Figure 4 below shows the accuracy of the CNN model over 30 epochs. It can be noted that the accuracy grows greatly for both the training and validation sets until the 10th epoch, after which the rate of improvement slows down. The learning rate decay and the saturation in minimizing the loss function would explains the dwindling fluctuations beyond the 10th epoch. The validation accuracy plateaus at around 91%, while the training accuracy grows to around 94.5%. A 60/20/20 split was also done to investigate the performance on a test set, which also performed with an accuracy near to that of the validation set, around 91%.

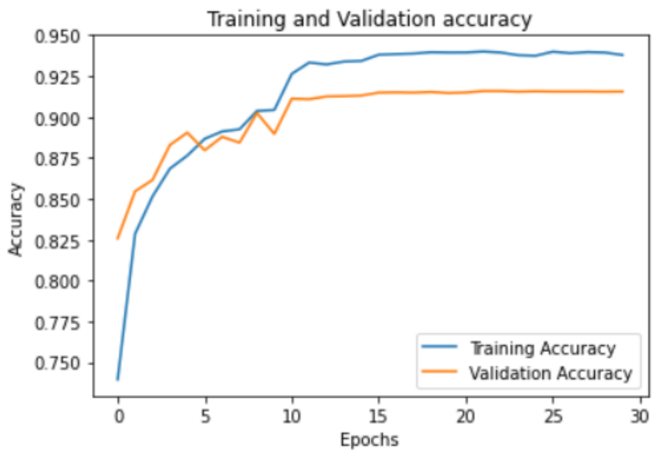
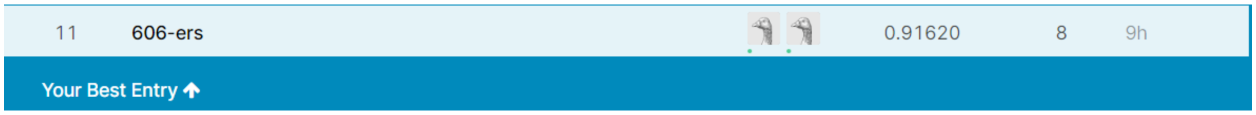


Figure 4 – Accuracy of CNN model over different epochs

On the Kaggle test set, this model managed to achieve 91.6% accuracy as shown in Figure 5, with a runtime of around 0.5 seconds, which is lower than the validation accuracy due to overfitting. The variation in performance between Kaggle submission can be due to a variety of reasons, mainly the different architectures of neural networks which result in different learning patterns and hence performance. Moreover, differences in training epochs, learning rates and regularization methods contribute to variation in outcome.



11	606-ers	0.91620	8	9h

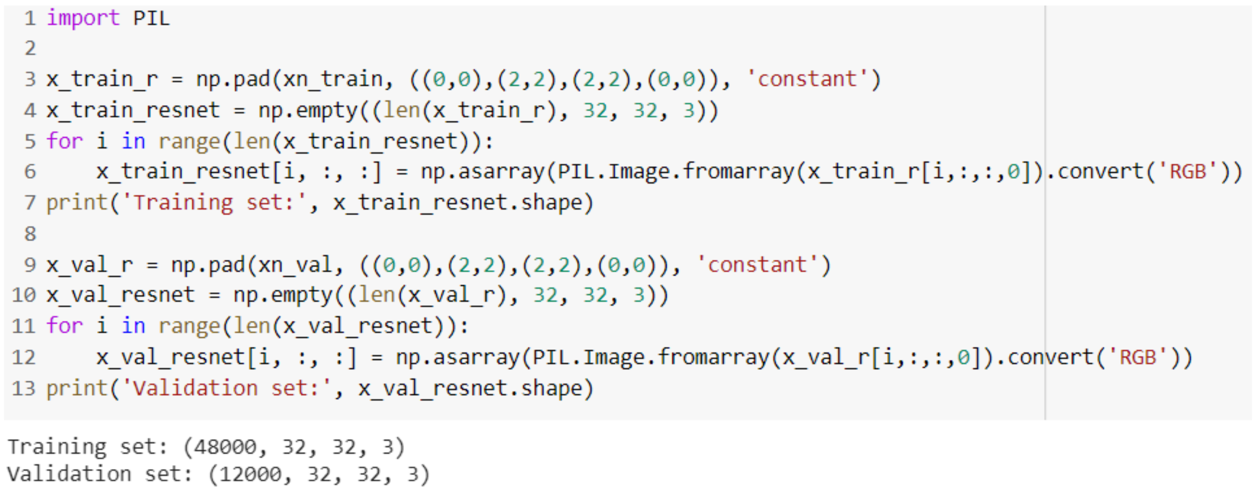
Your Best Entry

Figure 5 – Best Kaggle score

2. ResNet Model

Data description

The fashion MNIST dataset contain images of 5 different categories of clothing labelled accordingly. First, since acquired data had each image flatted into an array of 784 pixels, it had to be reshaped into an image of size 28x28 pixels. The Residual Neural Network (ResNet) model accepts a minimum of 32x32 size images, therefore the 28x28 sized images have to be padded to fit the input. In addition, the model requires a 3-channel input; and since the data is a single channel which denotes a greyscale image. The greyscale input is converted into 3 channels using the Python Imaging Library (PIL), providing the same inputs under different channels. This approach is done as presented in Figure 6, with the results showing both the training and validation sets of size (No. of images, 32, 32, 3), which are now able to be fed to the ResNet model.



```python
import PIL

x_train_r = np.pad(xn_train, ((0,0),(2,2),(2,2),(0,0)), 'constant')
x_train_resnet = np.empty((len(x_train_r), 32, 32, 3))
for i in range(len(x_train_resnet)):
    x_train_resnet[i, :, :] = np.asarray(PIL.Image.fromarray(x_train_r[i,:,:,0]).convert('RGB'))
print('Training set:', x_train_resnet.shape)

x_val_r = np.pad(xn_val, ((0,0),(2,2),(2,2),(0,0)), 'constant')
x_val_resnet = np.empty((len(x_val_r), 32, 32, 3))
for i in range(len(x_val_resnet)):
    x_val_resnet[i, :, :] = np.asarray(PIL.Image.fromarray(x_val_r[i,:,:,0]).convert('RGB'))
print('Validation set:', x_val_resnet.shape)
```

```
Training set: (48000, 32, 32, 3)
Validation set: (12000, 32, 32, 3)
```

Figure 6 – Code block to pre-process the data for ResNet

Model description

The Residual Neural Network (ResNet) model was introduced by Kaiming He et al. in [1]. Its architecture is presented in Figure 7 and involves skipping certain layers and heavy batch normalization. The benefit of skipping layers is providing shortcuts for backpropagation which could help reduce the problem of vanishing gradient. Moreover, it helps the model learn an identity function which, in turn, ensures the deeper layers perform at least as good as the previous layers according to [2]. As shown in the figure, there exists a 50-layer model, a 101-layer model and a 152-layer model, for this experiment the 50-layer model was chosen since it has the lowest number of parameters thus would speed up computational costs.

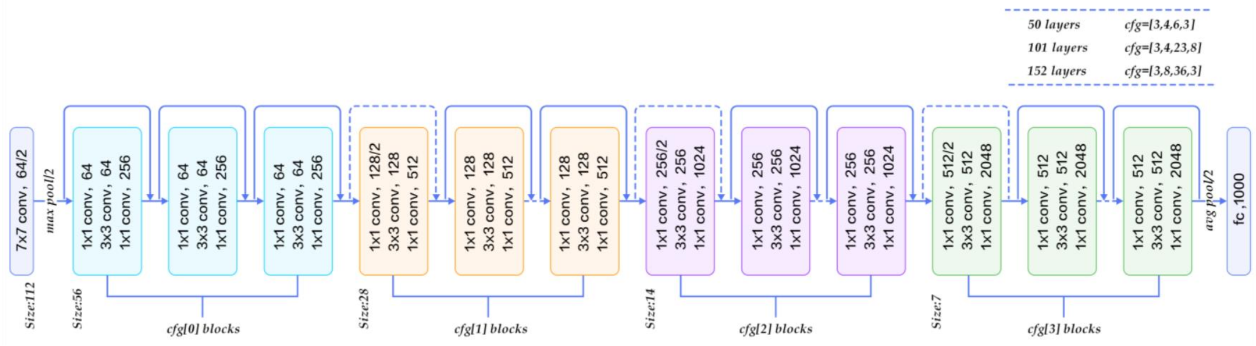
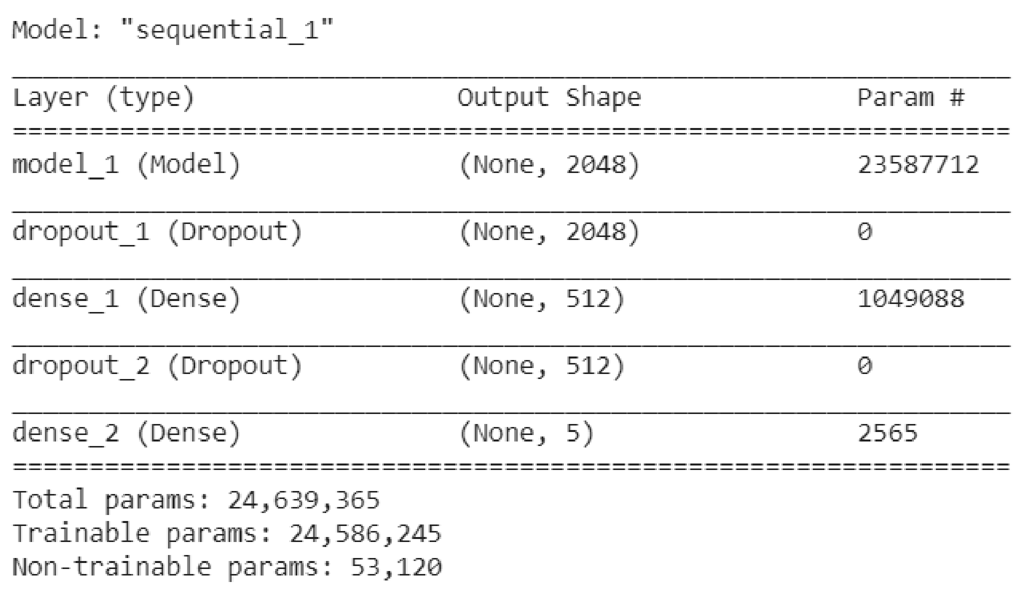


Figure 7 –ResNet architecture [3]

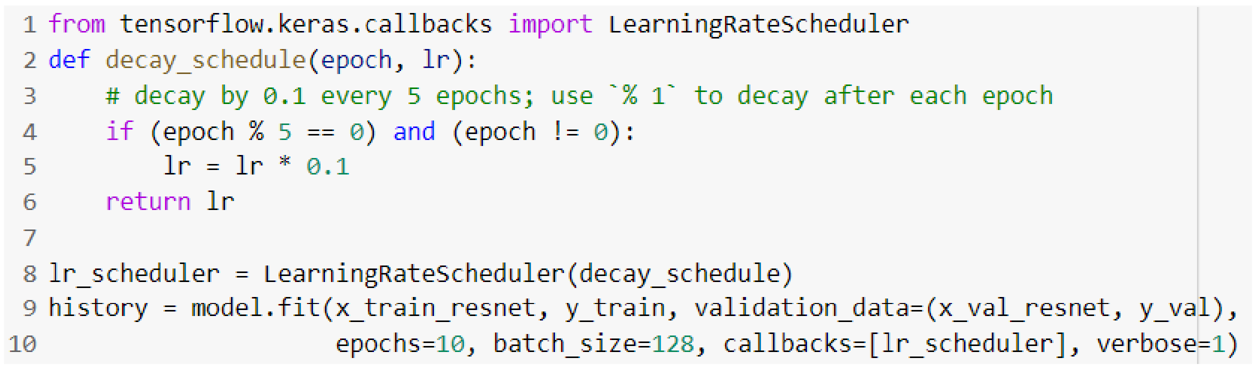
Figure 8 below describes the architecture of the fitted ResNet model. According to [4], to fit the model for the output of the 5 class categories, the output of the pre-trained ResNet model was replaced with a dense layer of size 512 cells that uses a ReLu activation function, which is fed into an output layer of 5 nodes (one for each class) that uses the sigmoid function for classification. Moreover, the loss function uses binary cross-entropy, while the optimizer chosen was Adam with a starting learning rate of 0.0001. Both of those hyper-parameters were chosen based on comparison of different values. For regularization, 2 dropout layers of 0.5 dropout are inserted one before each dense layer; this would help reduce the overfitting of the model.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_1 (Model)              (None, 2048)              23587712
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1049088
_________________________________________________________________
dropout_2 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 2565
=================================================================
Total params: 24,639,365
Trainable params: 24,586,245
Non-trainable params: 53,120
```

Figure 8 – Fitted ResNet model architecture

When running the model, the learning rate is decayed at by 0.1 at the 5th epoch to allow for the model to fine-tune its learning with smaller increments as it trains longer. Figure 9 below shows the approach taken to implement this learning rate decay, and the epoch checkpoint and decay were chosen based on extensive testing of different values. Moreover, the model trains on the data in batches of 128 datapoints at a time, and runs for 10 epochs; both of which were chosen based on comparing the accuracy of different values and picking the best to avoid both underfitting and overfitting.



```python
from tensorflow.keras.callbacks import LearningRateScheduler
def decay_schedule(epoch, lr):
    # decay by 0.1 every 5 epochs; use `% 1` to decay after each epoch
    if (epoch % 5 == 0) and (epoch != 0):
        lr = lr * 0.1
    return lr

lr_scheduler = LearningRateScheduler(decay_schedule)
history = model.fit(x_train_resnet, y_train, validation_data=(x_val_resnet, y_val),
                    epochs=10, batch_size=128, callbacks=[lr_scheduler], verbose=1)
```

Figure 9 – Code block for learning decay and running model

The model took about 50 seconds per epoch to train, with 10 epochs totalling to around 5 minutes. Analysing the loss reduction, Figure 10 shows the loss function for both the training and validation sets of the ResNet model across the 10 epochs. It can be noted that both the training and validation losses drop significantly up until the 5th epoch, after which the validation loss starts growing while the training loss continues to decrease, which indicates potential overfitting to the training set. Moreover, the learning rate decay is reflected on the graph by a reduction of fluctuations after epoch 5.

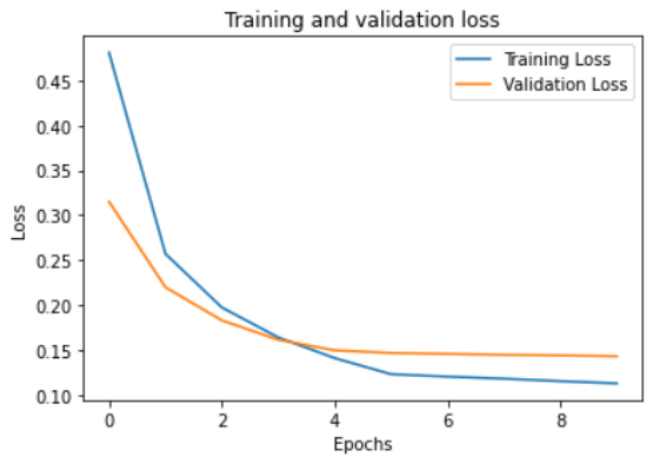


Figure 10 – Loss function of ResNet model over different epochs

Figure 11 below shows the accuracy of the ResNet model over 10 epochs. It can be noted that the accuracy grows greatly for both the training and validation sets until the 5th epoch, after which the rate of improvement slows down. The learning rate decay and the saturation in minimizing the loss function would explains the dwindling fluctuations beyond the 5th epoch for the validation set, while the training accuracy grows hinting to overfitting. The validation accuracy plateaus at around 93%, while the training accuracy grows to around 96%.

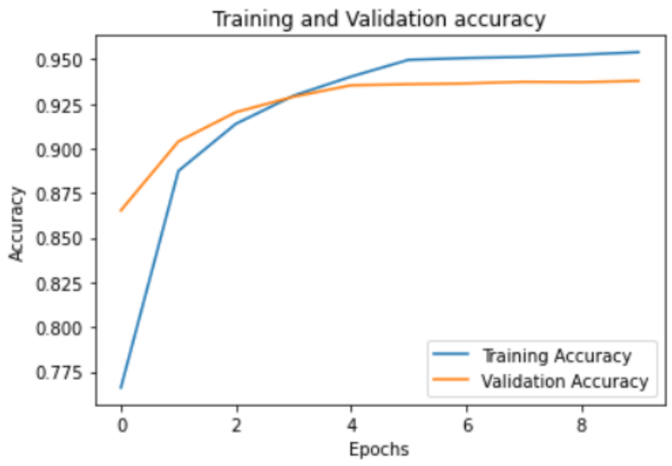


Figure 11 – Accuracy of ResNet model over different epochs

It is as expected that ResNet outperforms the CNN model described previously thanks to its skipping connections approach, as well as containing a lot more trainable parameters. On the other hand, thanks to the CNN's lower complexity, its computational costs are much lower; requiring less time and memory to train compared to ResNet. In addition to that, the CNN model works better with the test set on Kaggle than ResNet, which had a runtime of 7 seconds, and that goes back to the simplicity that provides more generalisability over multiple sets.

3. References

[1] K. He, X. Zhang, S. Ren and J. Sun, “Deep Residual Learning for Image Recognition,” in *Computer Vision and Pattern Recognition (CVPR),* Las Vegas, 2016.

[2] P. Dwivedi, “Understanding and Coding a ResNet in Keras,” 9 January 2019. [Online]. Available: https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33. [Accessed 16 April 2020].

[3] S. Das, “CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more…,” 16 November 2017. [Online]. Available: https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5. [Accessed 15 April 2020].

[4] R. Khandelwal, “Deep Learning using Transfer Learning - Python Code for ResNet50,” 29 August 2019. [Online]. Available: https://towardsdatascience.com/deep-learning-using-transfer-learning-python-code-for-resnet50-8acdfb3a2d38. [Accessed 15 April 2020].