Fashion MNIST Neural Network Modelling

Content

Foreword

The Fashion MNIST dataset comprises 70,000 grayscale images of 10 fashion products (tops, trousers, dresses, etc.). The data can be loaded straight from Keras into a training set (60,000 images) and a testing set (10,000 images).

Performance prediction is an important and active research area. In particular, several research efforts have built empirical models using machine learning algorithms for performance prediction.

1. Data Load and Exploring

The Fashion MNIST dataset comprises 70,000 grayscale images of 10 fashion products (tops, trousers, dresses, etc.). The data can be loaded straight from Keras into a training set (60,000 images) and a testing set (10,000 images).

The images are 28*28 arrays with pixel values 0 to 255, and the labels are an array of integersOur9, representing ten clothing classes. Predicting the clothing class is the task here; it's a multi-class classification task.

Before training Neural Network (NN) models, these images were first converted into vectors to enable a fully connected network to classify the images by applying the reshape function in Numpy. The vector values were rescaled between 0 and 1. Also, a built-in function 'to_categorical' was utilised to convert the integer labels (target labels, what we want to predict) to one-hot encoded brands.

2. Training NN models

Training a NN requires 4 steps:

- Build the architecture
- Compile the model
- Train the model
- Evaluate the model.

The NN architecture is decided by the number of layers and activation functions. The model1 initial ANN is a simple 3-layer NN, input layer, hidden layer with sigmoid activation function and has 32 neurons, output layer with softmax activation function and has 10 neurons to classifier the 10 fashion categories of the data. The model summary is as below (Figure 2.1):

```
Model: "sequential_23"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_46 (Dense)            (None, 32)                25120

 dense_47 (Dense)            (None, 10)                330

=================================================================
Total params: 25,450
Trainable params: 25,450
Non-trainable params: 0
_________________________________________________________________
```

Figure 2.1 Model summary information of model1

The Neural Networks model needs to be compiled with three critical things, optimiser, loss function and metric. The model1 initial ANN was collected first with stochastic gradient descent optimiser (SGD) as the optimiser, "categorical cross-entropy" as the loss function and "accuracy" as the metrics. The default learning rate of SGD is 0.01.

Next, train the model with validation to minimise the overfitting. Keras has built-in automatic validation; the validation split has been set at 0.1, which means 10% out of the training set is for validating in each epoch. Epoch = 50 has been applied for all the models in this task. It means the training set has been trained 50 times. The batch size is 128. As the training sample is 60,000, it will take 469 (60,000/128) iterations to complete one epoch.

Lastly, evaluate the model. The below curve graph presents loss and accuracy. The performance is not bad, with a test loss of 0.51 and test accuracy of 0.821. Different learning rates, optimisers, batch sizes, and architectures have been explored to check the model performance improvement.

```
Test loss: 0.51
Test accuracy: 0.821
<Figure size 576x432 with 0 Axes>
```

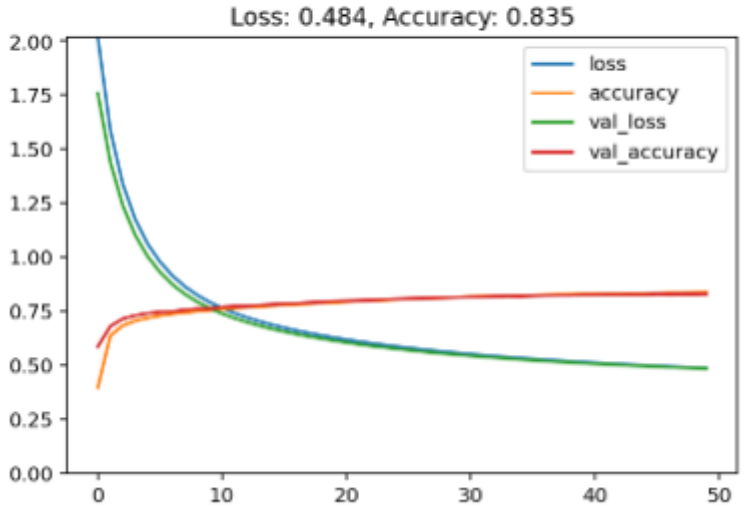


Figure 2.2 Initial ANN Model loss and accuracy graph for training and validation

3. The effect of varying Learning rates

Learning Rate	Val-accuracy
0.00015	0.1053
0.00050	0.2135
0.00080	0.4295
0.00320	0.6085
0.0060	0.7028
0.0219	0.7638
1.620	0.0917
4.617	0.0990

Table 3.1 the result of Optuna tuning learning rates

The learning rate represents the magnitude of each parameter update. For SGD, when the batch size has been fixed at 128, the learning rate close to 0.02 seems to work better here, with less test loss (0.3) and higher test accuracy (0.857) than the previous one. However, if the learning rate is too high, gradient explosion is likely to occur, the loss vibration amplitude is large, and the model is difficult to converge. Table 3.1 shows the test accuracy of the model tends to increase with learning rate but then decrease.

Similarly, the lower the learning rate also raises the concern since the slower the loss function changes and overfitting could happen quickly; i.e. learning rate is too low, the loss function doesn't improve.

From the below plot (Figure 3.2), the model performance of accuracy improves tuning the learning rate from 0.821 to 0.857. However, the vibration of the loss curve was more significant than the previous one (Figure 2.2), which means this learning rate might be higher than ideal.

```
Test loss: 0.439
Test accuracy: 0.857
<Figure size 576x432 with 0 Axes>
```

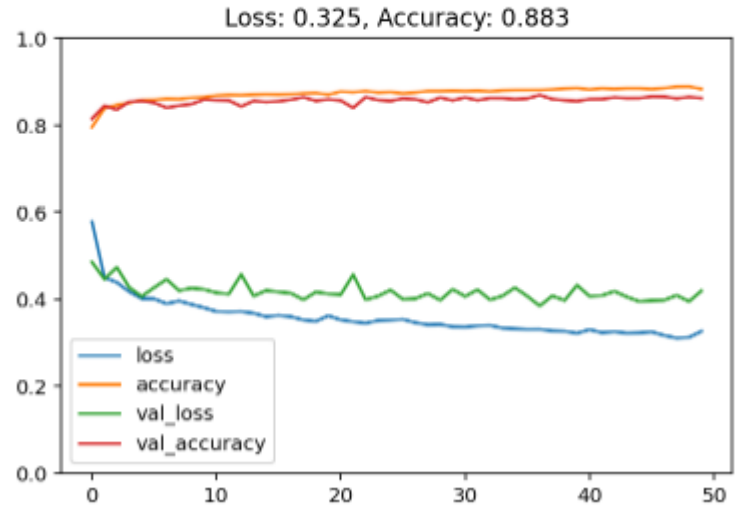


Figure 3.2 Model loss and accuracy graph for training and validation with the best learning rate

4. The effect of different Optimizers

Optimizer	Val-accuracy
Adadelta	0.1528
SGD	0.7212
Adam	0.8568
Nadam	0.8560
Rmsprop	0.8532

Table 4.1 shows the result of Optuna tuning optimisers.

Although the best optimiser might be another for each run, Adam, Nadam and RMSProp have similar validation accuracy values (near 0.85) and test loss (near 0.36). Table 4.1 shows that Adam, Nadam and RMSProp perform much better than SGD and Adadelta. They are better than the Keras default SGD optimiser; the possible reason is that they are relatively less dependent on the learning rate or even automatically adapt the learning rate during the training process than SGD. Also, we need to consider that the other probable reason is that SGD converges to the local optimum and could be trapped in the saddle point (Ruder, S.).

Table 4.1 also shows that Adadelta performs the worst among the five optimisers. The possible explanation is that only fixed items are accumulated in Adadelta. These items only approximated the corresponding average value so that the local minimum was jitter repeatedly during the late training duration.

Optuna is used as well to optimise the process of the hyperparameters.

By changing the optimiser from SGD to Adam, the performance improved to 0.876, and the loss curve was more flattened, which is good. However, over-fitting issues happen again (Figure 4.2). All is going well in the early stage of training, but the model is not generalising well towards the end! The plot shows the widening gap between the validation and training loss.

Could the batch size modification solve the problem? Batch size tuning in the next will show some changes.

```
Test loss: 0.365
Test accuracy: 0.878
<Figure size 576x432 with 0 Axes>
```

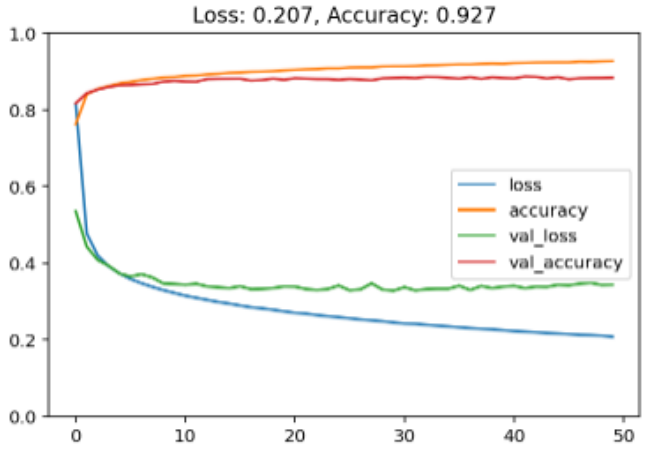


Figure 4.2 Model loss and accuracy graph for training and validation with the best optimiser (adam)

5. The effect of different Batch Sizes

Batch size	Val-accuracy
32	0.8690
64	0.8627
128	0.8568
256	0.8447
1024	0.7868
54000	0.3687

Table 5.1 the result of optuna tuning batch_size trial.

Based on the previous network model, different batch sizes have been suggested by Optuna to evaluate performance, and the number of epochs was fixed at 5 to save time. 32 and 64 batch sizes provide better validation accuracy than others (Table 5.1). Batch size is defined as the number of examples from the training dataset used to estimate the error gradient.
Table 5.1 shows that the validation accuracy decreases while the batch size increases. The large-batch method seems to lack generalisation ability; a study has suggested that it is because the large-batch methods tend to converge to Sharpe minimisers of the training function. The batch size can control the estimate of the error gradient when training neural networks (Dai, X., & Zhu, Y.). The model's performance is also good, with a test loss of 0.411 and test accuracy of 0.855 (Figure 5.2).

The performance improved with the ANN model; however, it did not change Much over the above Adam model.

```
Test loss: 0.411
Test accuracy: 0.855
<Figure size 576x432 with 0 Axes>
```

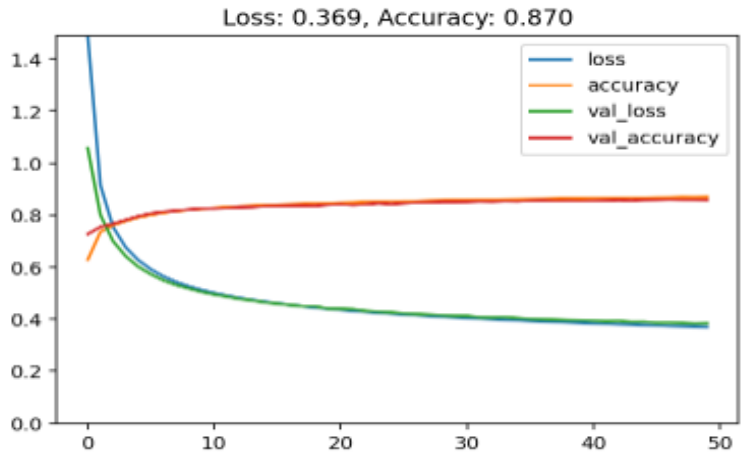


Figure 5.2 Model loss and accuracy graph for training and validation with the best batch size

The smaller or larger batch size cannot guarantee better convergence. Some researchers have studied the effect of batch size on network performance with either the training accuracy or training time to determine which small batches or large batches are better. A small batch can converge faster than the large one, but a large batch will get optimum minima, which the small one cannot reach. Small batches could need to prevent a lower learning rate from overshooting minima and a significant regularisation effect from the high variance. Small batch sizes generalize well on the validation set. The gradient oscillates more apparent in the small batch size than in the larger batch size (Brownlee, J).

6. The effect of different Architectures

6 different architectures have been explored, and the results were collected in Table 6.1 below.

Architectures	Test Loss	Test Accuracy
Initial NN	0.510	0.821
Dropout ANN	0.378	0.871
Deep ANN	0.441	0.878
Simple CNN	0.907	0.885
Dropout CNN	0.269	0.900
Deep CNN with Batch Normalization	0.278	0.931
VGG-16	0.445	0.921

Table 6.1 the result of 6 different NN architectures

Let's get some concepts before getting into the results.

- Dropout, when forward propagation, makes the activation value of a neuron stop working with a certain probability (randomly set some activations to zero), making the model more generalised because it won't depend too much on some local features (Brownlee, J).

- DNN (Deep Neural Networks) mainly represents that we add two hidden layers in this case. This project needs the architecture of NN: 2 Convolutional layers + 2 MaxPooling layers + 3 Dropout layers + 2 Fully-Connected layers (Johnson, J.).

- CNN (Convolutional Neural Networks) as 'feed-forward neural networks' includes filters and pooling layers as convolutional architecture. Compared with Multi-Layers Perceptron(MLP), CNN takes tensors as input and makes it easier to understand spatial relations between pixels of an image (Saha, S).

- The Batch Normalization method is to normalise the activation outputs of the previous layer, basically scaling the outputs so that they have a mean of about 0 and a standard deviation of 1. As the name suggests, this is done over batch intervals, which happen concurrently with each backdrop. Empirically, batch normalisation improves performance and learning speed broadly, and I certainly see gains after applying it. Batch normalisation can be used for all hidden layers in the network, but it is also common to use it only for the input layer.

- VGG uses a kernel size of 3x3 and a max-pooling size of 2x2 in the entire network. For example, the 16 of the more commonly used VGG-16 refers to the total number of layers of conv+fc is 16, which is the number of layers excluding max pool (Wei, J.).

The above table shows the deeper CNN model, BatchNormalization CNN and VGG provide better test accuracy (over 0.9) and the first two present less in test loss. Comparing the initial ANN model and dropout ANN model, the performance of deep ANN improves to 0.878. But the parameters are close to the dropout model, which means it needs more time to train. The overfitting issue starts to appear; this can be seen by the widening gap between the validation and training loss and accuracy lines.

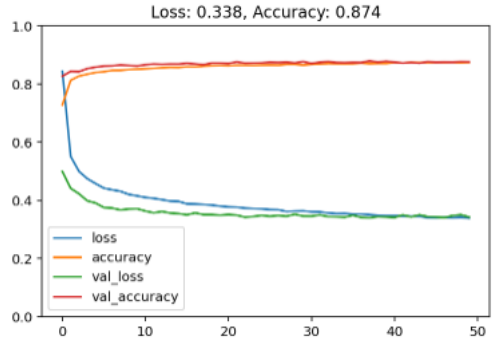
Adding the dropout layer may address the over-fitting issue, add more layers of ANN and at the same time add the dropout layer to see if the performance could be further improved. But that would keep increasing the number of parameters and more time needed for model training. Another common and widely used architecture CNN has been tried to get out.

As the above concepts show, all the vectorised data and fully connected models don't consider any spatial relationships in the image. It can mess up all the surrounding pixels into a vector. CNN related models generally give Good test accuracy. Interestingly, the simple CNN model generates overfitting from the below plots. The dropout method was applied to CNN to get the non-overfitting

result. Then, Batch Normalization was used to optimise the impact and get the best test accuracy without apparent overfitting. VGG is a new method through the research, and its accuracy keeps at approximately 0.92; although the model presents overfitting in the end part, the performance is good.

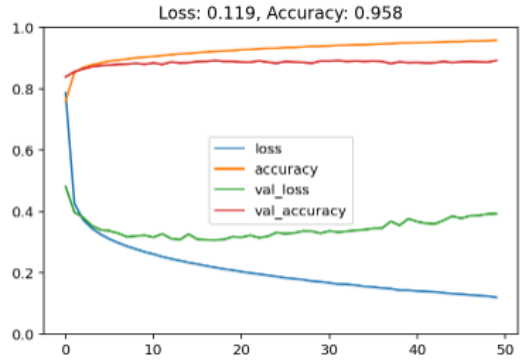
Dropout ANN

```
Test loss: 0.378
Test accuracy: 0.871
<Figure size 576x432 with 0 Axes>
```



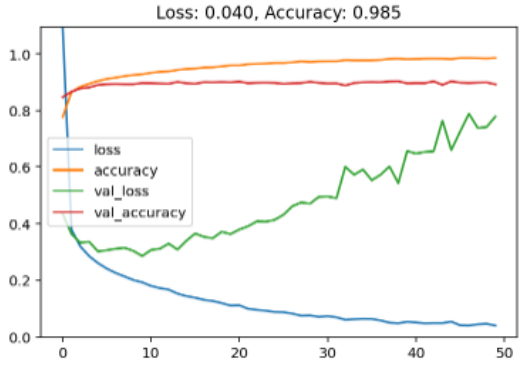
Deep ANN

```
Test loss: 0.441
Test accuracy: 0.878
<Figure size 576x432 with 0 Axes>
```



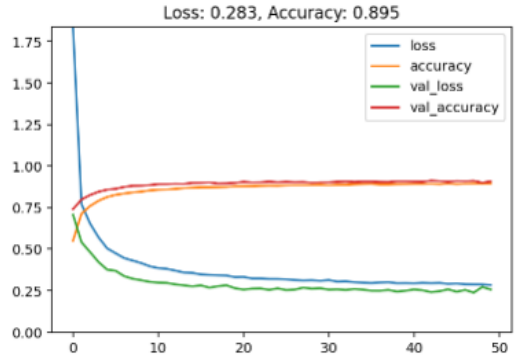
Simple CNN

```
Test loss: 0.907
Test accuracy: 0.885
<Figure size 576x432 with 0 Axes>
```



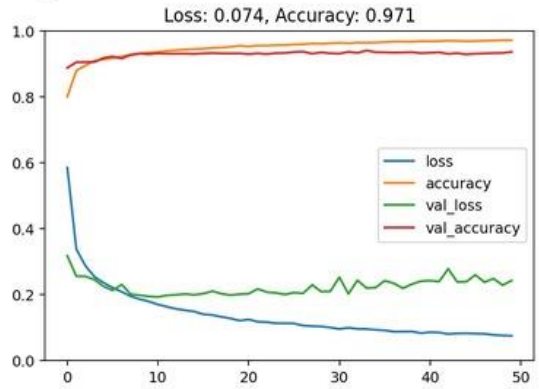
Dropout CNN

```
Test loss: 0.269
Test accuracy: 0.9
<Figure size 576x432 with 0 Axes>
```



Deep CNN with BN

```
Test loss: 0.278
Test accuracy: 0.931
<Figure size 576x432 with 0 Axes>
```



VGG-16

```
Test loss: 0.445
Test accuracy: 0.921
<Figure size 576x432 with 0 Axes>
```

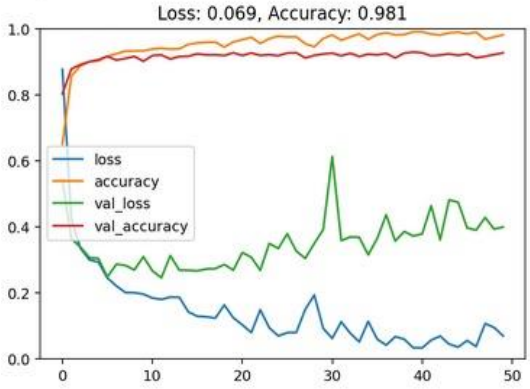


Figure 6.1 Model loss and accuracy graph for training and validation with six different architectures (Left-right and top-bottom: Dropout Model, Deep Model, Simple CNN, Deeper CNN, Batch Normalisation CNN, and VGG)

Conclusion

Neural Networks perform excellent on image classification. While building the NN, many aspects need to be considered, like the architectures, hyperparameters (learning rate, optimisers, batch sizes), as they could affect the model performance. Optuna enables efficient hyperparameter tuning. It's notable that overfitting is easily diagnosed by monitoring the performance of deep NN. In this task, it's found that adjusting the batch size (32 or 64 recommended) and applying the Dropout method can address the over-fitting issues. In terms of architecture, adding hidden layers will help improve the model performance to some extent. CNN would generally be more powerful than fully connecting ANN on image classification task. DeepCNN with Batch Normalisation primarily performs the best with the highest accuracy 0.931 and very low loss 0.278. Since this method can standardise the inputs to a layer for each mini-batch, it can stabilise the learning rate by adding extra layers to the deep neural network.

References

1. Brownlee, J. (2019). A gentle introduction to batch normalisation for deep neural networks. *Machine Learning Mastery; Machine Learning Mastery: San Juan, Puerto Rico*.
2. Brownlee, J. (2019). How to control the stability of training neural networks with the batch size. *Machine Learning Mastery*, *20*.
3. Dai, X., & Zhu, Y. (2020). On Large Batch Training and Sharp Minima: A Fokker–Planck Perspective. Journal of Statistical Theory and Practice, 14(3).

https://doi.org/10.1007/s42519-020-00120-9

4. Ruder, S. (2016). An overview of gradient descent optimisation algorithms. *arXiv preprint arXiv:1609.04747*.

5. Brownlee, J. (2019b, August 6). A Gentle Introduction to Dropout for Regularizing Deep Neural Networks. Machine Learning Mastery. https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

6. Johnson, J. (2020, July 27). What's a Deep Neural Network? Deep Nets Explained. BMC Blogs. https://www.bmc.com/blogs/deep-neural-network/

7. Saha, S. (2018). A comprehensive guide to convolutional neural networks—the ELI5 way. *Towards data science*, *15*.

8. Wei, J. (2019). VGG neural networks: The next step after alexnet. *Towardsdatascience. com*.