Report for Assignment 3

by XINGLONG LI – 21115109

The Fashion MNIST dataset consists of grayscale images of clothing items, each represented by 784 feature columns (i.e., 28×28 pixels), along with one label column indicating the class, with values ranging from 0 to 4.

1. Feature Engineering

Neural networks are notoriously sensitive to the scaling of their inputs [1]. We examined each feature column and observed that the feature values range from 0 to 255 (as shown in the red rectangles Figure 1). If some feature values are significantly large, the gradients of their corresponding weights tend to be large as well. This may cause the weights to fail to converge and oscillate around the optimal values. This issue can be mitigated by using a smaller learning rate. However, a small learning rate may slow down convergence for weights corresponding to smaller feature values, causing them to fail to reach optimal values.

To address this, we normalize the features using min-max normalization. Specifically, we divide each value by 255 to ensure that all features are scaled within the same range, thereby facilitating convergence during training.

	min_value	max_value
count	784.0	784.000000
mean	0.0	252.420918
std	0.0	15.956882
min	0.0	16.000000
25%	0.0	255.000000
50%	0.0	255.000000
75%	0.0	255.000000
max	0.0	255.000000

Figure 1. Statistics of minimum and maximum values of all feature columns.

2. Validation Dataset Construction

In deep learning model training, it is important to strike a balance between maximizing training data and ensuring proper evaluation of the training process. While having more training data generally improves model performance, it is also essential to reserve a portion of the data for validation in order to monitor the training progress and make timely adjustments if necessary. Thus, we allocate 20% of the training data as a validation set.

3. Model Experimental Design and Result

Models are trained by GPU on Google Colab, and all runtimes are compared on the platform.

3.1. Experiment 1: Base Model

The base model is the default network in Q1 from the assignment requirement. Its layers and the number of its different parameters are shown in Figure 2.

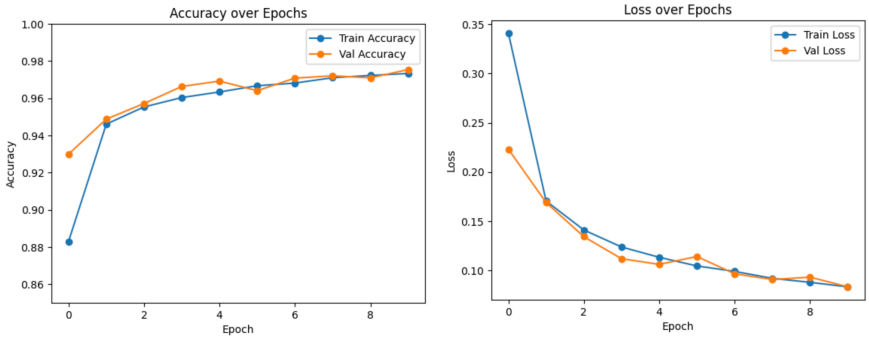
Model: "sequential_3"

Layer (type)	Output Shape	Param #
conv2d_6 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_7 (Conv2D)	(None, 14, 14, 32)	9,248
flatten_3 (Flatten)	(None, 6272)	0
dense_6 (Dense)	(None, 512)	3,211,776
dense_7 (Dense)	(None, 5)	2,565

Total params: 3,223,909 (12.30 MB)
Trainable params: 3,223,909 (12.30 MB)
Non-trainable params: 0 (0.00 B)

Figure 2. Summary of the base model

Figure 3 presents the training and validation accuracy and loss curves over epochs, accompanied by detailed epoch-wise performance metrics. From the accuracy plot, we observe a steady increase in both training and validation accuracy throughout the training process, indicating that the model is learning effectively without significant overfitting. The loss plot shows a consistent decline in both training and validation loss values, which further confirms the convergence of the model. The validation loss closely follows the training loss, suggesting good generalization performance on unseen data. The final test accuracy reached 97.32%.




```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.7917 - loss: 0.5665 - val_accuracy: 0.9299 - val_loss: 0.2231
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9421 - loss: 0.1841 - val_accuracy: 0.9488 - val_loss: 0.1689
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9534 - loss: 0.1472 - val_accuracy: 0.9572 - val_loss: 0.1342
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9582 - loss: 0.1283 - val_accuracy: 0.9663 - val_loss: 0.1118
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9639 - loss: 0.1138 - val_accuracy: 0.9693 - val_loss: 0.1060
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 5ms/step - accuracy: 0.9662 - loss: 0.1085 - val_accuracy: 0.9641 - val_loss: 0.1140
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9663 - loss: 0.1039 - val_accuracy: 0.9708 - val_loss: 0.0964
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9705 - loss: 0.0918 - val_accuracy: 0.9721 - val_loss: 0.0906
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9731 - loss: 0.0853 - val_accuracy: 0.9710 - val_loss: 0.0930
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9722 - loss: 0.0849 - val_accuracy: 0.9754 - val_loss: 0.0832

⏱ Training time: 35.27 seconds

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9750 - loss: 0.0823
Test Accuracy: 0.9732
```

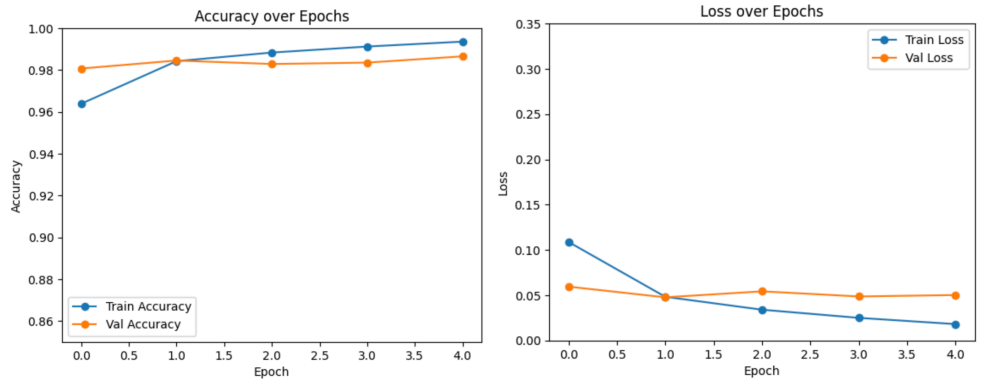
Figure 3. Training and validation accuracy/loss curves and epoch-wise metrics of the base model.

3.2. Experiment 2: Base Model with Adam Optimizer and Early Stopping

In this model, we replace the SGD optimizer in the base model with the Adam optimizer and introduce early stopping to prevent overfitting, which does not change model layers, thus its model structure is still the same as in Figure 2. Compared to SGD, Adam adaptively adjusts the update magnitude of each parameter individually during backpropagation, based on the first and second moments of the gradients. This allows for faster and more stable convergence, making it a more effective choice for deep learning tasks.

Early stopping automatically terminates the training process when validation performance stops improving, helping to avoid unnecessary training and improve generalization. Specifically, we set the patience parameter to 3, which means that training will stop if the validation loss does not decrease for 3 consecutive epochs. This not only helps avoid overfitting but also reduces training time by stopping at an optimal point.

As shown in Figure 4, both training and validation accuracy increase rapidly and reach a high level within a few epochs. At the same time, the validation loss remains stable. The model achieves high performance with significantly reduced training time. In our experiment, the model achieved a test accuracy of 0.9823 with only about 23.6 seconds of training time. This shows that the combination of Adam optimizer and early stopping not only improves the model performance, but also significantly increases the training efficiency.




```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9342 - loss: 0.1823 - val_accuracy: 0.9808 - val_loss: 0.0594
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9838 - loss: 0.0498 - val_accuracy: 0.9846 - val_loss: 0.0476
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9884 - loss: 0.0340 - val_accuracy: 0.9829 - val_loss: 0.0542
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9915 - loss: 0.0235 - val_accuracy: 0.9836 - val_loss: 0.0486
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9937 - loss: 0.0170 - val_accuracy: 0.9866 - val_loss: 0.0502

⏱ Training time: 23.57 seconds

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9824 - loss: 0.0543
Test Accuracy: 0.9823
```

Figure 4. Training and validation accuracy/loss curves and epoch-wise metrics of Experiment 2.

3.3. Experiment 3: Deeper Fully Connected Network

In this experiment based on experiment 2, we extend the base model by adding an additional fully connected (FC) layer with 256 units after the original 512-unit FC layer. This design follows a pyramid-like structure, which is commonly used in deep neural networks (DNNs) to progressively reduce the dimensionality and enhance hierarchical feature learning. As shown in Figure 6, the addition of this extra FC layer increases the model complexity and the total number of parameters significantly—from 3.22 million in the base model to over 3.35 million in this model. While the training time is slightly longer, the model achieves better performance, with the test accuracy improving from 0.9823 to 0.9848.

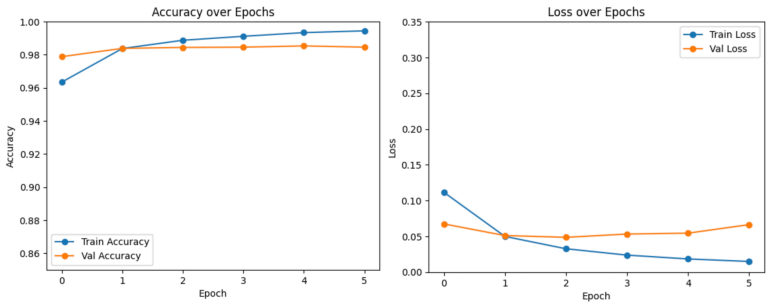
Model: "sequential_14"

Layer (type)	Output Shape	Param #
conv2d_28 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_14 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_29 (Conv2D)	(None, 14, 14, 32)	9,248
flatten_14 (Flatten)	(None, 6272)	0
dense_29 (Dense)	(None, 512)	3,211,776
dense_30 (Dense)	(None, 256)	131,328
dense_31 (Dense)	(None, 5)	1,285

Total params: 3,353,957 (12.79 MB)
Trainable params: 3,353,957 (12.79 MB)
Non-trainable params: 0 (0.00 B)

Figure 5. Summary of the model in Experiment 3.

From the training and validation curves in Figure 6, we can observe that both accuracies rise rapidly and remain high, and the validation loss stays low and stable, despite its increased depth and complexity. This experiment demonstrates that adopting a deeper FC structure in a pyramid-shaped architecture can enhance performance, at the cost of increased training time and model size. With more parameters introduced, the model exhibits a slightly higher overfitting.




```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9317 - loss: 0.1941 - val_accuracy: 0.9788 - val_loss: 0.0673
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9831 - loss: 0.0507 - val_accuracy: 0.9838 - val_loss: 0.0510
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9897 - loss: 0.0310 - val_accuracy: 0.9844 - val_loss: 0.0486
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.9912 - loss: 0.0233 - val_accuracy: 0.9846 - val_loss: 0.0532
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9942 - loss: 0.0163 - val_accuracy: 0.9853 - val_loss: 0.0544
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9950 - loss: 0.0129 - val_accuracy: 0.9846 - val_loss: 0.0663

⏱ Training time: 29.79 seconds

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9851 - loss: 0.0517
Test Accuracy: 0.9848
```

Figure 6. Training and validation accuracy/loss curves and epoch-wise metrics of Experiment 3.

3.4. Experiment 4: More Convolution Layers

In this experiment, we further increase the depth of the network by adding two additional convolutional layers and two max pooling layers compared to Experiment 3. As shown in Figure 7, this modification results in a deeper network architecture that captures more complex spatial patterns.

Although the total number of parameters is slightly reduced due to the smaller flatten layer connected to the first fully connected layers, the increased depth leads to higher computational complexity per forward and backward pass. As a result, the training time increases (from 29.79s to 40.47s), as reflected in the epoch-wise training logs.

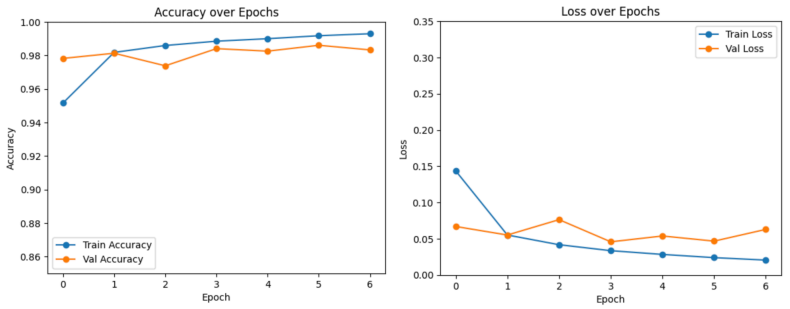
Model: "sequential_21"

Layer (type)	Output Shape	Param #
conv2d_54 (Conv2D)	(None, 28, 28, 32)	320
max_pooling2d_33 (MaxPooling2D)	(None, 14, 14, 32)	0
conv2d_55 (Conv2D)	(None, 14, 14, 64)	18,496
max_pooling2d_34 (MaxPooling2D)	(None, 7, 7, 64)	0
conv2d_56 (Conv2D)	(None, 7, 7, 128)	73,856
conv2d_57 (Conv2D)	(None, 7, 7, 128)	147,584
max_pooling2d_35 (MaxPooling2D)	(None, 3, 3, 128)	0
flatten_21 (Flatten)	(None, 1152)	0
dense_50 (Dense)	(None, 512)	590,336
dense_51 (Dense)	(None, 256)	131,328
dense_52 (Dense)	(None, 5)	1,285

Total params: 963,205 (3.67 MB)
Trainable params: 963,205 (3.67 MB)
Non-trainable params: 0 (0.00 B)

Figure 7. Summary of the model in Experiment 4.

Nevertheless, the deeper architecture enables better feature extraction and improves the model performance: the test accuracy increases from 0.9848 to 0.9858.(see Figure 8)



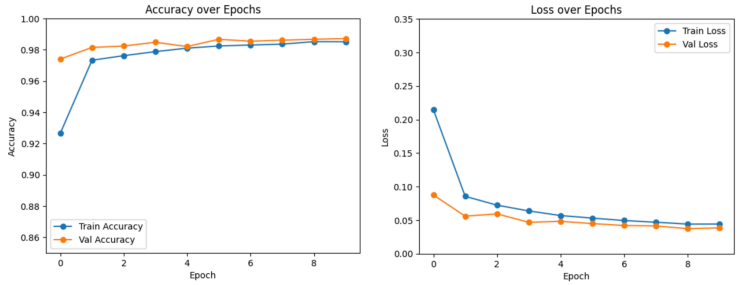

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 10s 8ms/step - accuracy: 0.8958 - loss: 0.2893 - val_accuracy: 0.9783 - val_loss: 0.0669
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 7s 6ms/step - accuracy: 0.9820 - loss: 0.0558 - val_accuracy: 0.9813 - val_loss: 0.0551
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9856 - loss: 0.0424 - val_accuracy: 0.9738 - val_loss: 0.0763
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9879 - loss: 0.0337 - val_accuracy: 0.9841 - val_loss: 0.0457
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9916 - loss: 0.0240 - val_accuracy: 0.9826 - val_loss: 0.0538
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9922 - loss: 0.0220 - val_accuracy: 0.9861 - val_loss: 0.0467
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - accuracy: 0.9937 - loss: 0.0186 - val_accuracy: 0.9833 - val_loss: 0.0629

⏱ Training time: 40.47 seconds

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9867 - loss: 0.0476
Test Accuracy: 0.9858
```

Figure 8. Training and validation accuracy/loss curves and epoch-wise metrics of Experiment 4.

3.5. Experiment 5: Add Dropout Layers

In this experiment, we introduce dropout layers to address the overfitting issue observed in Experiments 3 and 4 (see Figures 3, 6, and 8 for comparison). The overfitting patterns are particularly interesting: the base model did not exhibit overfitting, while Experiment 3 showed clear overfitting due to a significant increase in model parameters compared to the base model — which aligns with the common understanding that larger models tend to overfit more easily. However, Experiment 4 presents a more intriguing observation. Despite having fewer parameters (963,205) than the base model (3,223,909), the model still suffered from overfitting. This suggests that deeper convolutional layers alone can also lead to overfitting, even with a reduced parameter count. Although some people have found this phenomenon in practice [2], they have not offered a convincing explanation. Initially, we planned to explore dropout, L1, and L2 regularization techniques to mitigate this problem. Eventually, we only use dropout, as it proved highly effective in practice and resolved the overfitting issue without the need for L1, and L2 regularization. Dropout randomly sets a fraction of neurons to zero during training. This prevents the model from relying too much on specific neurons and promotes more robust, distributed feature learning, thereby reducing overfitting.

As shown in Figure 9, applying dropout after convolutional and dense layers narrows the gap between training and validation accuracy compared to previous experiments. Validation accuracy improves and remains stable, and validation loss stays low throughout training. The final test accuracy reaches 0.9863, the highest among all experiments, clearly demonstrating the effectiveness of dropout in improving model performance.




```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 12s 7ms/step - accuracy: 0.8522 - loss: 0.4091 - val_accuracy: 0.9741 - val_loss: 0.0875
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9720 - loss: 0.0898 - val_accuracy: 0.9815 - val_loss: 0.0560
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.9757 - loss: 0.0743 - val_accuracy: 0.9824 - val_loss: 0.0594
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9790 - loss: 0.0615 - val_accuracy: 0.9848 - val_loss: 0.0468
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 5ms/step - accuracy: 0.9816 - loss: 0.0550 - val_accuracy: 0.9821 - val_loss: 0.0483
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9818 - loss: 0.0515 - val_accuracy: 0.9868 - val_loss: 0.0449
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9835 - loss: 0.0489 - val_accuracy: 0.9855 - val_loss: 0.0420
Epoch 8/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9843 - loss: 0.0449 - val_accuracy: 0.9862 - val_loss: 0.0414
Epoch 9/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9857 - loss: 0.0432 - val_accuracy: 0.9868 - val_loss: 0.0373
Epoch 10/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9853 - loss: 0.0442 - val_accuracy: 0.9872 - val_loss: 0.0385

⏲ Training time: 54.88 seconds

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9854 - loss: 0.0487
Test Accuracy: 0.9863
```

Figure 9. Training and validation accuracy/loss curves and epoch-wise metrics of Experiment 5.

The model summary is shown in Figure 10.

Model: "functional_205"

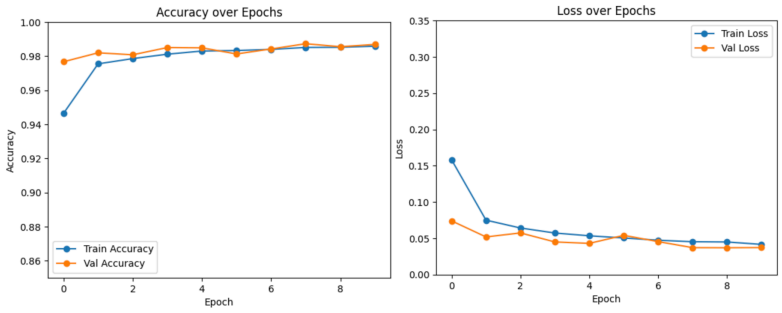
Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28, 1)	0
conv1 (Conv2D)	(None, 28, 28, 32)	320
pool1 (MaxPooling2D)	(None, 14, 14, 32)	0
dropout1 (Dropout)	(None, 14, 14, 32)	0
conv2 (Conv2D)	(None, 14, 14, 64)	18,496
pool2 (MaxPooling2D)	(None, 7, 7, 64)	0
dropout2 (Dropout)	(None, 7, 7, 64)	0
conv3 (Conv2D)	(None, 7, 7, 128)	73,856
dropout3 (Dropout)	(None, 7, 7, 128)	0
conv4 (Conv2D)	(None, 7, 7, 128)	147,584
pool4 (MaxPooling2D)	(None, 3, 3, 128)	0
dropout4 (Dropout)	(None, 3, 3, 128)	0
flatten (Flatten)	(None, 1152)	0
dense_512 (Dense)	(None, 512)	590,336
dropout5 (Dropout)	(None, 512)	0
encoding_layer (Dense)	(None, 256)	131,328
output_layer (Dense)	(None, 5)	1,285

Total params: 963,205 (3.67 MB)
Trainable params: 963,205 (3.67 MB)
Non-trainable params: 0 (0.00 B)

Figure 10. Summary of the model in Experiment 5.

3.6. Experiment 6: Data Augmentation

In this experiment, we apply data augmentation based on the model from Experiment 5 to increase data diversity and improve performance. The augmentation includes slight rotations, width and height shifts, and zooming, which simulate real-world variations and enhance model robustness. However, as shown in Figure 11, the test accuracy does not improve. Moreover, since the augmented samples are added into the original training set, the dataset size doubles, leading to significantly longer training time. Thus, we do not adopt data augmentation in the final model.




```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 15s 6ms/step - accuracy: 0.8947 - loss: 0.2897 - val_accuracy: 0.9768 - val_loss: 0.0740
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 16s 5ms/step - accuracy: 0.9743 - loss: 0.0783 - val_accuracy: 0.9820 - val_loss: 0.0519
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9780 - loss: 0.0661 - val_accuracy: 0.9808 - val_loss: 0.0574
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9812 - loss: 0.0580 - val_accuracy: 0.9851 - val_loss: 0.0450
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9835 - loss: 0.0517 - val_accuracy: 0.9849 - val_loss: 0.0430
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9839 - loss: 0.0480 - val_accuracy: 0.9813 - val_loss: 0.0541
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9829 - loss: 0.0488 - val_accuracy: 0.9843 - val_loss: 0.0454
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9850 - loss: 0.0449 - val_accuracy: 0.9873 - val_loss: 0.0371
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 8s 5ms/step - accuracy: 0.9858 - loss: 0.0428 - val_accuracy: 0.9856 - val_loss: 0.0370
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9864 - loss: 0.0394 - val_accuracy: 0.9869 - val_loss: 0.0372

⏲ Training time: 104.04 seconds

313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.9860 - loss: 0.0436
Test Accuracy: 0.9861
```

Figure 11. Training and validation accuracy/loss curves and epoch-wise metrics of Experiment 6.

3.7. Experiment 7: Residual Network

In this experiment, we explore the residual network structure, which is originally proposed to address the vanishing gradient problem in deep neural networks. Although our model is not very deep, we are still interested in investigating this structure. Our inspiration comes from the idea of Wide & Deep models [3] used in recommender systems: removing the "wide" part that inputs x directly leads to significantly degraded performance; similarly, using only the "wide" part without the deep part also performs poorly.

Here, we treat the residual block as the "wide" part (i.e. x) and combine it with the "deep" part (i.e. F(x)) to enhance model learning, the model structure is shown in Figure 12.

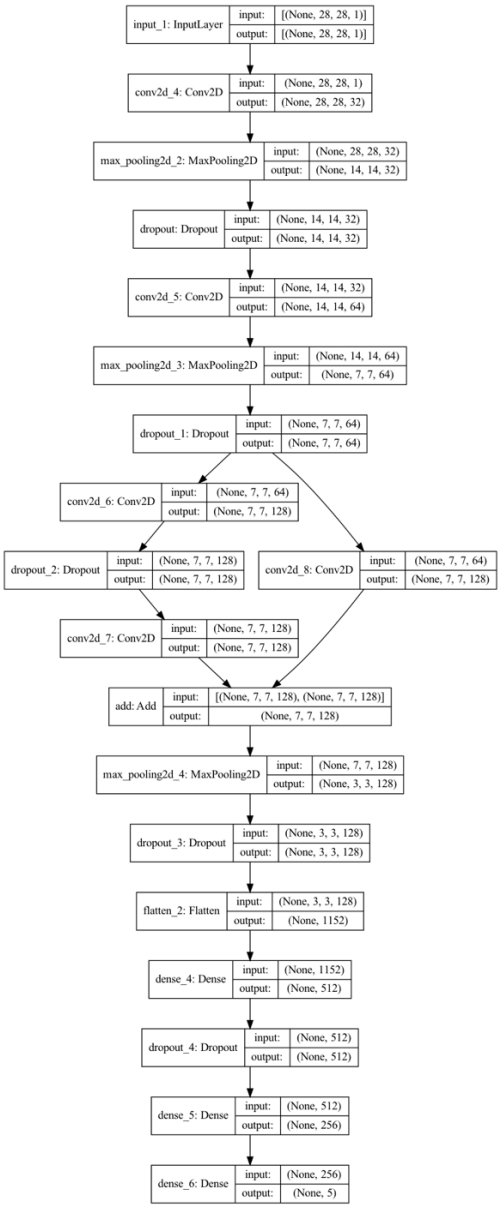


Figure 12. CNN with residual block.

As shown in Figure 13, the model achieves a test accuracy of 0.9835, which is worse than Experiment 5. To save space, we omit the accuracy/loss curves in the report, as they show similar trends to previous experiments.

```
Epoch 1/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 13s 9ms/step - accuracy: 0.8523 - loss: 0.3922 - val_accuracy: 0.9755 - val_loss: 0.0792
Epoch 2/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 14s 6ms/step - accuracy: 0.9715 - loss: 0.0886 - val_accuracy: 0.9787 - val_loss: 0.0680
Epoch 3/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9773 - loss: 0.0693 - val_accuracy: 0.9826 - val_loss: 0.0508
Epoch 4/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9794 - loss: 0.0617 - val_accuracy: 0.9848 - val_loss: 0.0469
Epoch 5/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9818 - loss: 0.0529 - val_accuracy: 0.9852 - val_loss: 0.0473
Epoch 6/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9831 - loss: 0.0502 - val_accuracy: 0.9846 - val_loss: 0.0532
Epoch 7/10
750/750 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9825 - loss: 0.0500 - val_accuracy: 0.9845 - val_loss: 0.0479

⏱ Training time: 51.81 seconds

313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9835 - loss: 0.0503
Test Accuracy: 0.9839
```

Figure 13. Training and validation epoch-wise metrics of Experiment 7.

3.8. Experiment 9: Classical Deep CNNs

For comparison, we also train two classical CNNs: GoogleNet and ResNet. These models are included to evaluate the performance gap between our models and widely used excellent open-source architectures. We do not use them in the subsequent experiments, since we want to more focus on the models we construct. Their initial test accuracies are 0.9813 and 0.9846, respectively. Notably, these models tend to overfit easily, so we apply data augmentation to enrich the training data. As a result, their test accuracies improve to 0.9861 and 0.9865, which are comparable to the best results obtained from our earlier experiments, but GoogleNet and ResNet have the cost of significantly deeper architectures.

4. Latent Feature Extraction, Clustering Visualization and Label analysis

We finally choose the model in Experiment 5 as the best model. Then we take its last fully connected layer and use that as a compact representation of the data. We use KMeans and DBScan to reduce dimensions and PCA and t-SNE to visualize. Although we could alternatively use other intermediate layers or concatenate multiple layers to obtain more complex feature representations, we do not explore these options here. As shown in the following analysis, the features extracted from the final fully connected layer already provide a clear separation between different labels.

4.1. KMeans Clustering

Figures 14 and 15 show the PCA and t-SNE visualization of the encoded features extracted from the CNN, respectively. In the left plot, the points are colored by the true class labels, while the right plot shows the clustering results from KMeans (with 5 clusters). Each color represents a different cluster.

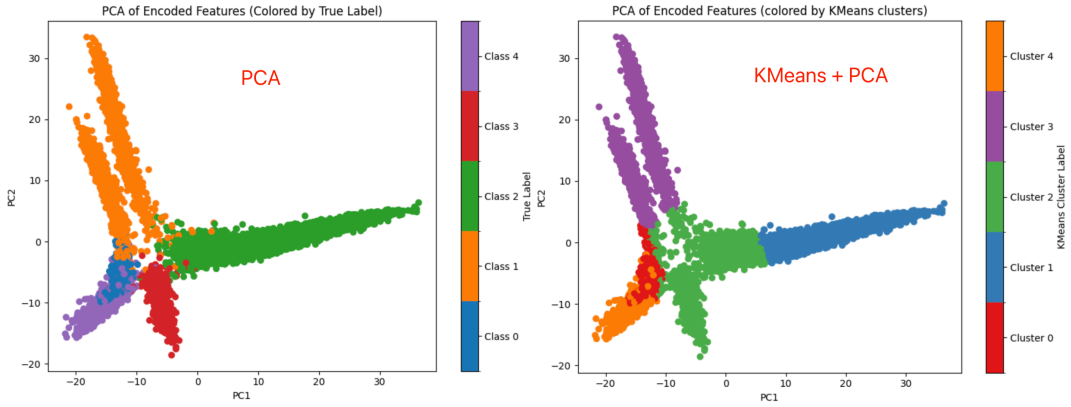


Figure 14. PCA visualization and KMeans clustering.

From the figure 14, we can observe several interesting patterns. For instance, the samples from Class 1 (mostly grouped in Cluster 3) appear to be further divided into two subgroups, suggesting possible subcategories. Class 0 (mainly in Cluster 0) and Class 4 (mainly in Cluster 4) seem quite similar, as their PCA projections overlap significantly—possibly indicating similar items. Cluster 2 contains a mix of Classes 1, 2, and 3, while Cluster 1 is mostly composed of Class 2 samples.

From the figure 15, we observe very similar distribution patterns as in t-SNE visualization. Class 0 (mostly in Cluster 0) and Class 4 (mostly in Cluster 4) still appear quite similar, as their t-SNE projections are very close, although not overlapping as in the PCA plot. Similarly, Class 2 is again split into two clusters (Cluster 1 and Cluster 2), and its left part is grouped with Class 3 into Cluster 2, even though the spatial distance between Class 3 and the left part of Class 2 is significantly larger than that between the left and right parts of Class 2.

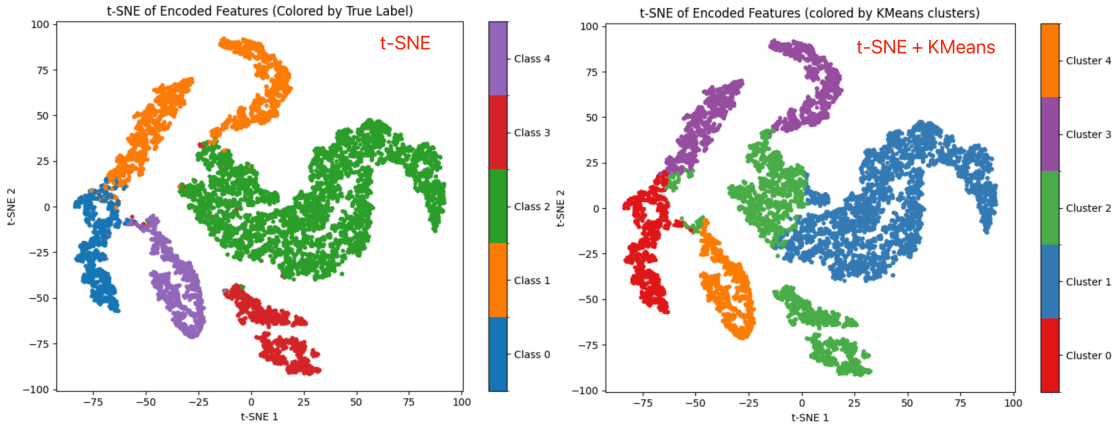


Figure 15. t-SNE visualization and KMeans clustering.

To further interpret these clusters, we randomly select five samples from each cluster and visualize them along with their true labels in Figure 16. The sample inspection confirms our earlier observations: in 4th row, we can see that class 1 includes two types of clothing, ankle

boots and trousers; label 0 consists of thin closed shoes including sneakers, low-cut closed shoes, or casual shoes. In contrast, label 4 contains open-toe shoes, which are often worn in summer. Class 0, 1, and 4 form a continuous region. This is because they include footwear (despite they are different types of footwears). Class 2 mainly consists of upper-body clothes such as shirts, blouses, or dresses, which are typically worn on the upper part of the body. Interestingly, we observe that class 2 items are divided into two clusters (cluster 1 and cluster 2). By images shown, items in cluster 1 are mostly men's upper-body clothes, while cluster 2 mainly contains women's upper-body clothes. We notice that class 3 items also appear in cluster 2. In Figure 16, the only class 3 item in cluster 2 is a women's handbag and the class 3 items are women's upper-body clothes, which suggests that cluster 2 may primarily represent women-related items.

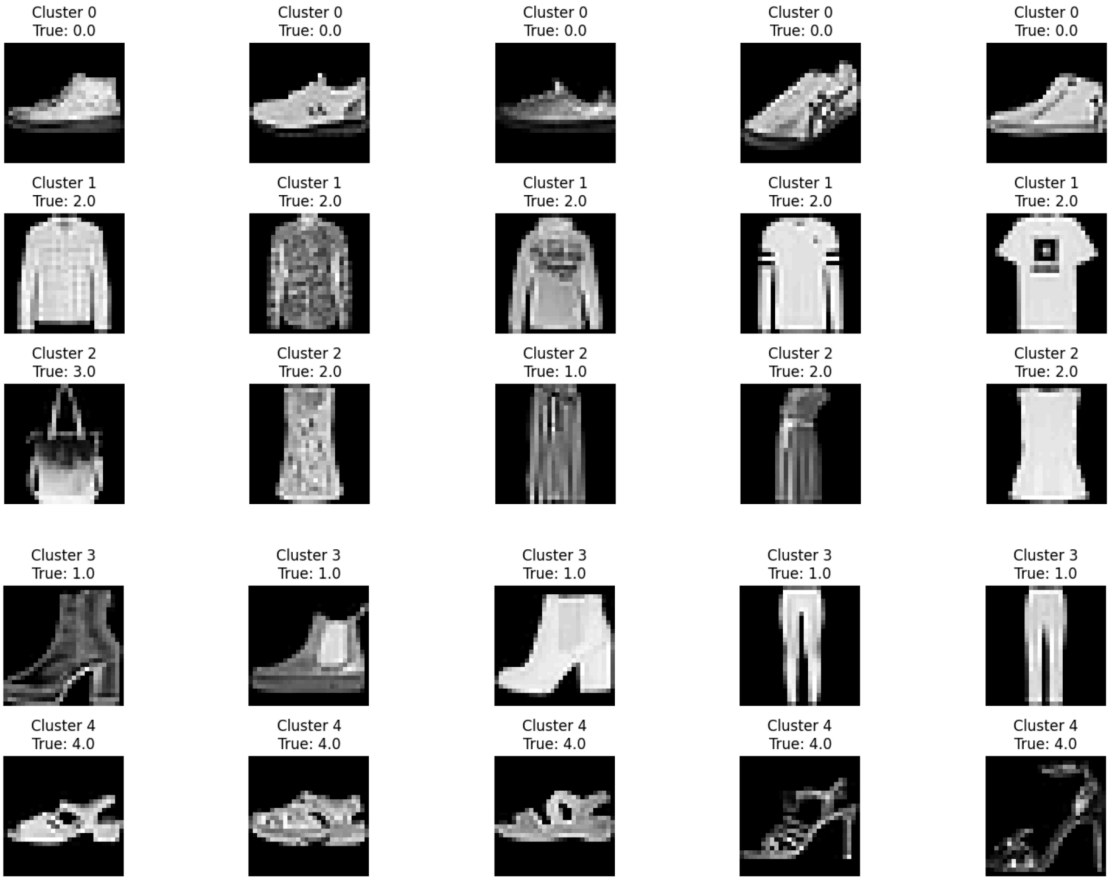


Figure 16. Original image in each cluster from KMeans, 'True' represents real label (class).

4.2. DBSCAN Clustering

We perform the same operations using DBSCAN, and the corresponding PCA and t-SNE visualization results are shown in Figure 17. DBSCAN mainly identifies five large groups (clusters 0–4, highlighted by red rectangles in Figure 17). Unlike KMeans, it splits class 1 into two clusters (cluster 0 and cluster 2). From our previous analysis, we know that class 1

contains both trousers and ankle boots, but we are not sure which part corresponds to trousers and which to ankle boots. We expect that DBSCAN clustering can help distinguish between these two categories.

Cluster 0 includes class 0, class 1, and possibly part of class 4. According to the KMeans analysis, class 0 corresponds to thin closed shoes, while class 1 includes ankle boots (which are often worn in winter), and class 4 represents open-toe shoes (which are often worn in summer). Since all of them are shoes, it is reasonable that they are grouped together in DBScan.

We also observe that class 3 forms an independent cluster. Thus, when we randomly visualize samples from each cluster, we expect to see more class 3 items compared to KMeans clustering where class 3 along with other two classes form a cluster. Based on earlier analysis, class 3 items are likely to be women's handbags.

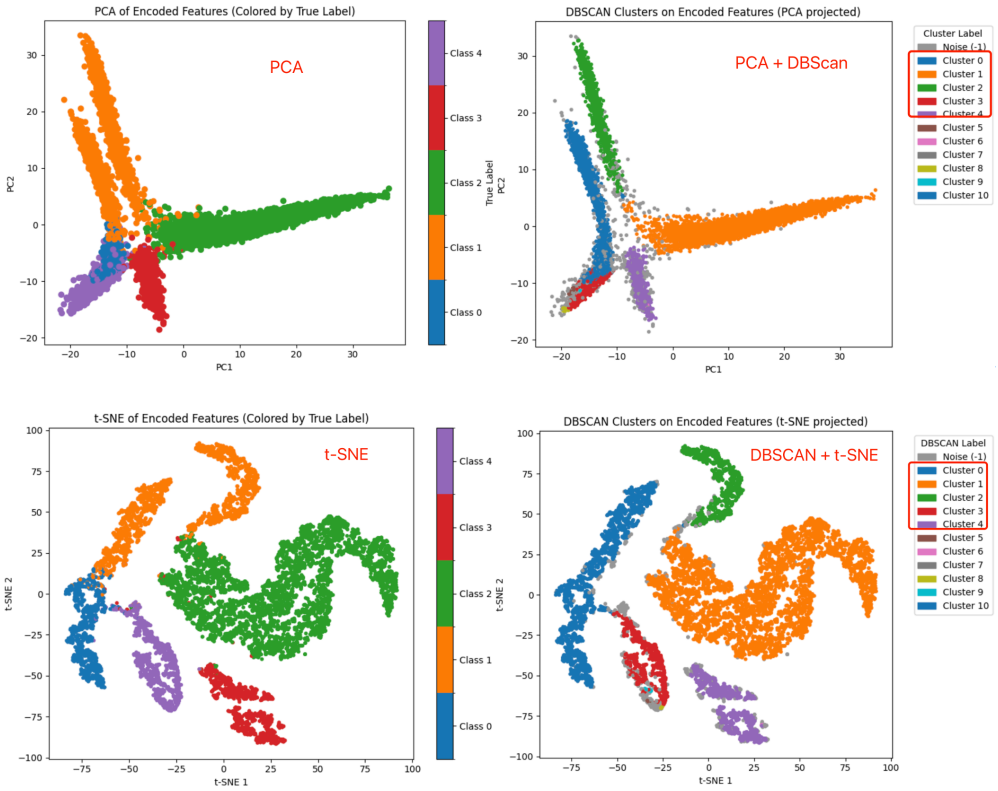


Figure 17. PCA visualization and DBScan clustering.

As in the KMeans analysis, we randomly visualize five samples from each DBSCAN cluster with their true labels (see Figure 18). Our above speculation from Figure 17 is confirmed: cluster 0 indeed contains items from class 0 and class 1 (class 4 does not appear, likely due to its small proportion in cluster 0). Both class 0 and class 1 belong to the shoe category. The left part of class 1 corresponds to ankle boots, while the right part of class 1 (in cluster 2) corresponds to trousers.

Class 3 corresponds to cluster 4, which contains exactly women's handbags (see Figure 18). In the previous KMeans analysis, class 3 and the left part of class 2 were grouped into one cluster (cluster 2), the former being women's handbags (only ONE class 3 item is shown in the images in Figure 16) and the latter being women's upper body garments, so we inferred that cluster 2 contains women's items. More class 3 items are shown here to support this inference. In addition, like the KMeans result, class 4 items (open-toe shoes) are grouped together again (cluster 3).

One outlier is worth mentioning: it is a true class 1 item found in cluster 1(3rd item in row 2). The item appears to be a woman's skirt (a lower-body garment worn from the waist down, like trousers), which ideally should belong to cluster 2. However, it could also be interpreted as a dress (a one-piece garment worn from the upper body, like upper-body clothes), which would correspond to cluster 1. This ambiguity makes it difficult to classify even for human observers, and the model ultimately assigns it to cluster 1.

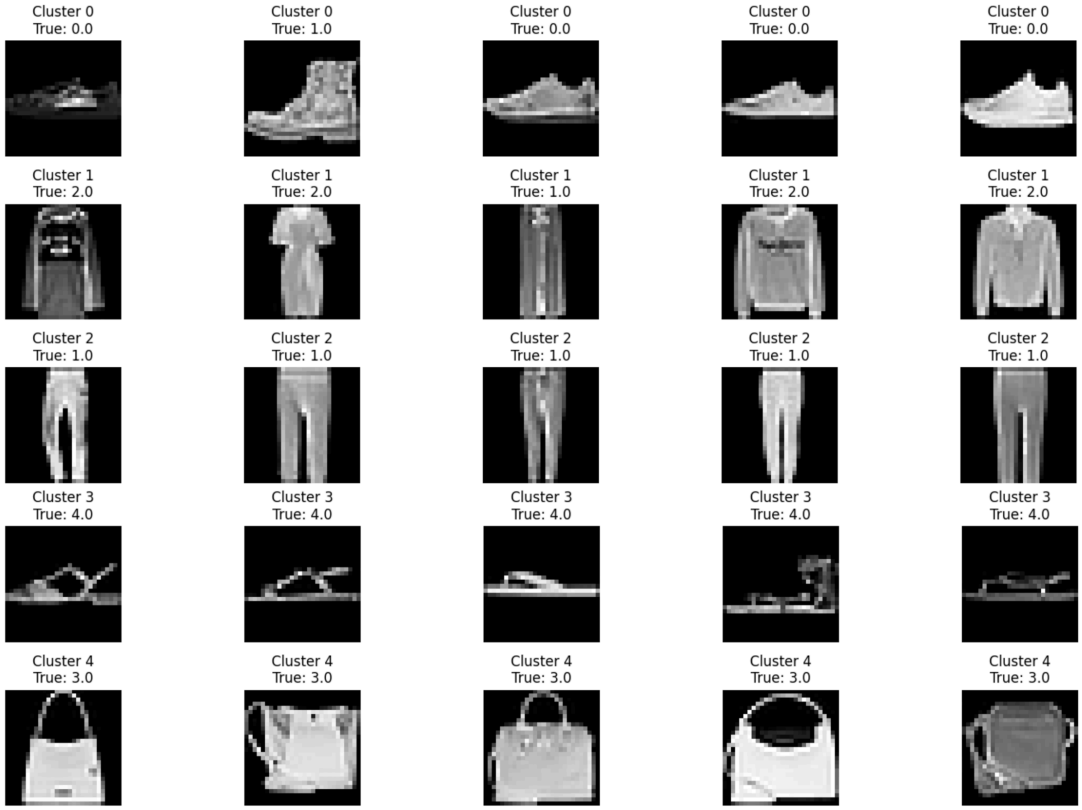


Figure 18. Original image in each cluster from DBScan, 'True' represents real label (class).

5. Summary

We summarize the results of all experiments in Table 1. While training time is not strictly comparable due to early stopping, the testing time can be fairly compared across models, although there is normal time variation each time the same data is predicted on the same model. The first three models (Experiments 1–3) share almost identical test time since their model parameter sizes are very close. Adam optimizer and early stop do not change parameter size, the Models 1 and 2 thus have same model size. Model 3 differs from Model 2 by adding only one fully connected (FCN) layer. This increase in model parameters is minimal and does not significantly affect the inference time, hence the testing time remains similar.

In Model 4, several convolutional layers and max pooling layers are added and significantly reduces the number of model parameters, which speeds up inference. Model 5 adds several dropout layers to Model 4 but does not slow down the test as all model parameters are used in dropout layers during the prediction, resulting in almost identical test time for both Model 4 and Model 5.

Model 6 incorporates data augmentation during training, but its model remains unchanged. Thus, its test time is again similar to Models 4 and 5.

Model 7 introduces a residual block. Due to the nature of the residual connections, where the input x and output F(x) shapes need to be matched and added, an additional shape transformation and addition step is required. This increases the forward pass time during inference, making the test time significantly longer than that of Experiments 4, 5 and 6.

Table 1. Experiments summary

Experiment	1	2	3	4	5	6	7
test accuracy	0.9732	0.9823	0.9848	0.9858	0.9863	0.9861	0.9835
training time	35	24	30	40	55	104.04	52
testing time	1.4	1.48	1.43	1.3	1.28	1.38	2.7

model 1 in Experiment 1: base model = default network
model 2 in Experiment 2: model 1 + adam optimizer + early stop
model 3 in Experiment 3: model 2 + a layer of FCN with 256 output-unit after final FCN
model 4 in Experiment 4: model 3 + two layers of conv layers & two max pooling layers before the flatten layer
model 5 (final model that will be plot) in Experiment 5: model 4 + multiple dropout layers
model 6 in Experiment 6: model 5 + data augmentation
model 7 (in Figure 12) in Experiment 7: model 5 + residual block

Training Parameters: epoch = 10, others not in above experiments are default.

The “model 5” in Experiment 5 is the best model with best accuracy on test dataset, and we plot its model structure in Figure 19.

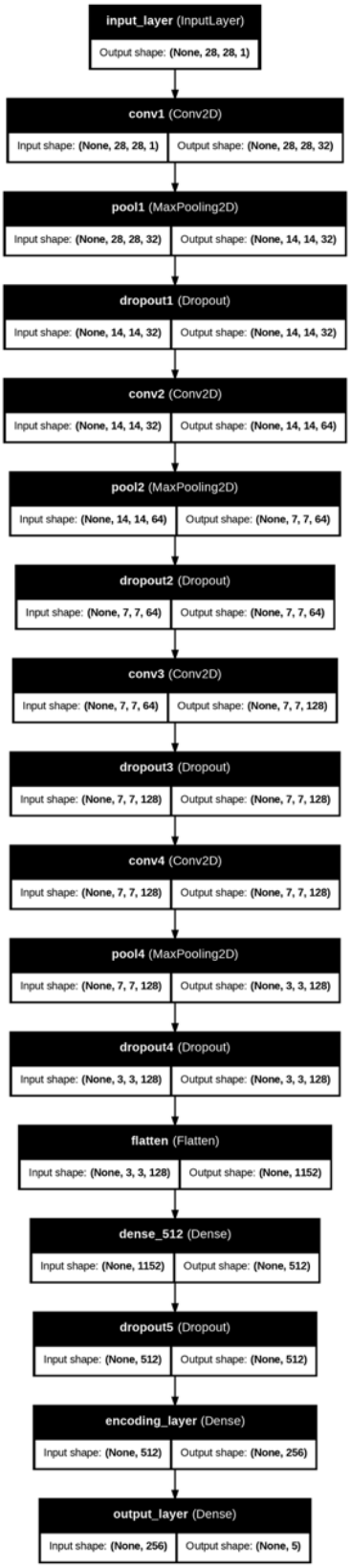


Figure 19． The structure of the best model “model 5”

Detailed item meanings the five labels represent are listed in the Table 2. Based on our previous analysis, we further annotate the potential subcategories in the feature space to better interpret the clustering results. As shown in Figure 20, label (class) 1 consists of two subcategories: the left part corresponds to high-cut ankle boots, while the right part represents trousers. Label (class) 2 also exhibits an internal structure without a clear boundary—its left part mainly contains upper-body clothes for women, while the right part corresponds to upper-body clothes for men.

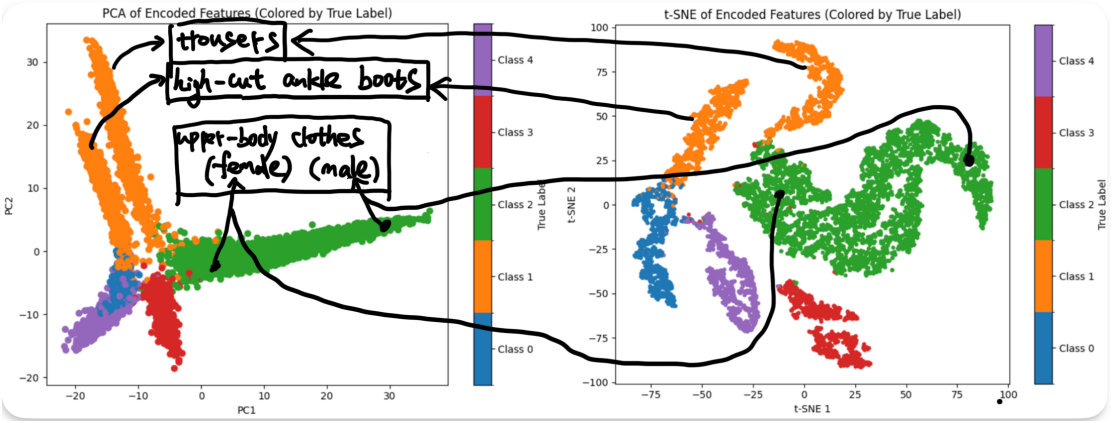


Figure 20 . Annotated PCA and t-SNE visualizations showing specific Class 1 and Class 2.

Table 2. The meaning of all labels

Label	corresponding items
0	thin closed shoes including sneakers, low-cut closed shoes, or casual shoes
1	hight-cut ankle boots(left) and trousers(right)
2	upper-body clothes such as shirts, blouses, or dresses, which are typically worn on the upper part of the body.
3	women's handbags
4	open-toe shoes, which are often worn in summer.

References:

[1]. S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. CoRR, abs/1502.03167, 2015.

[2]. https://www.quora.com/Do-convolution-layers-reduce-overfitting

[3]. https://arxiv.org/abs/1606.07792