Module - Artificial Neural Networks & NLP

Module Code - CSIP5103

A Neural Network Model for Fashion-MNIST Image Classification

Student ID - P2938154

Student Name - Patel jay Maheshkumar

Executive Summary

This project is concerned with the implementation and comparison of two deep learning networks, “Artificial Neural Network (ANN)” and “Convolutional Neural Network (CNN)” to for classifying Fashion-MNIST dataset items (clothing) out of ten categories, and of 70,000 grayscale images. The main objective was to comparison of the performances of each architecture in classifying fashion items and managing classically similar entities. Normalised and pre-processed data were used to train both models with the help of “TensorFlow Keras”. The ANN consisted of two instructions of hidden dense layers with ReLU activation, and CNN had “convolutional and pooling layers” to identify spatial data. The Adam optimiser and categorical cross-entropy loss were used to train in the form of ten epochs to optimise the training.

ANN got a test accuracy of 88.4% and the CNN got 90.6% which shows a better performance in learning spatial hierarchies and minimizing misclassification. Both models were found to be confused with similar classes that were visually similar, including Shirt, T-shirt/top, Coat, but the CNN minimized the number of errors since it built a convolutional feature extraction. Detailed assessments encompassed confusion matrices and classification reports, and training curves reflecting convergence and stability. The CNN was more accurate, precise and recalled, as well as it took more time to compute, when compared to the ANN. The conclusion of the analysis is that CNNs are better suited to “image-based classification problems” because they are also able to capture both the local patterns and textures. Future work involves augmentation of the data, regularisation of the drop-outs, and learning rate optimization to make both models more robust and less overfitting.

Table of Contents

1. Introduction

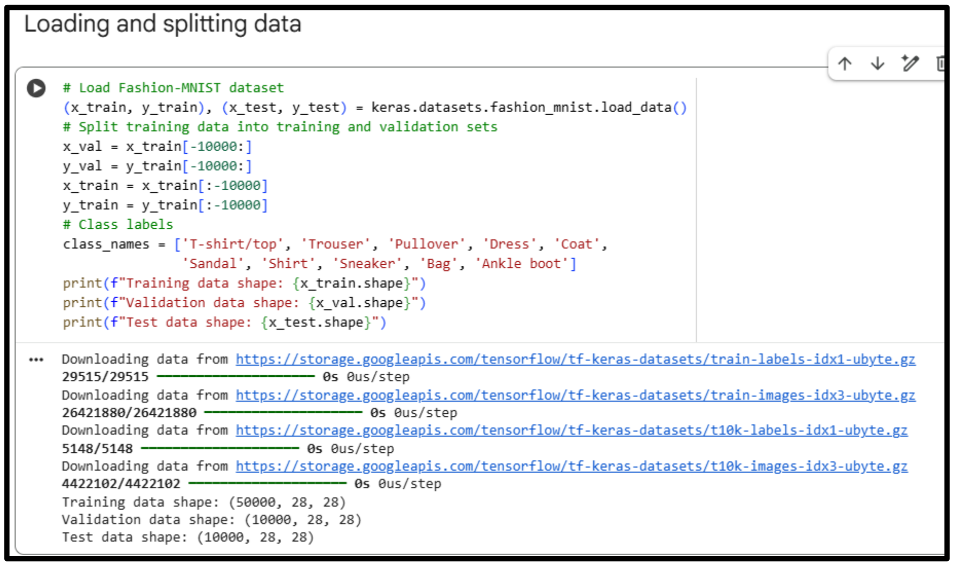
This report provides the technical application of image classification with the help of the "Fashion-MNIST dataset", which provides grayscale images of clothing elements of ten categories. Accurate classification of images by two models of deep learning, an "Artificial Neural Network (ANN)" and a "Convolutional Neural Network (CNN)", are aimed to be designed, trained, and evaluated. The comparison shows the way the various network architectures treat spatial data and what the areas of confusion between similar classes, visually like shirts and coats, share.

2. Data Handling

2.1 Dataset Description

Fashion-MNIST dataset includes 70,000 grayscale 28 by 28 pixel images divided into 10 clothes categories: "T-shirt, trousers, pullover, dress, coat, sandals, shirt, sneaker, bag and ankle boot". It consists of 60,000 training and 10,000 testing samples.

2.2 Data Loading and Splitting



Loading and splitting data

```python
# Load Fashion-MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
# Split training data into training and validation sets
x_val = x_train[-10000:]
y_val = y_train[-10000:]
x_train = x_train[:-10000]
y_train = y_train[:-10000]
# Class labels
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
print(f"Training data shape: {x_train.shape}")
print(f"Validation data shape: {x_val.shape}")
print(f"Test data shape: {x_test.shape}")
```

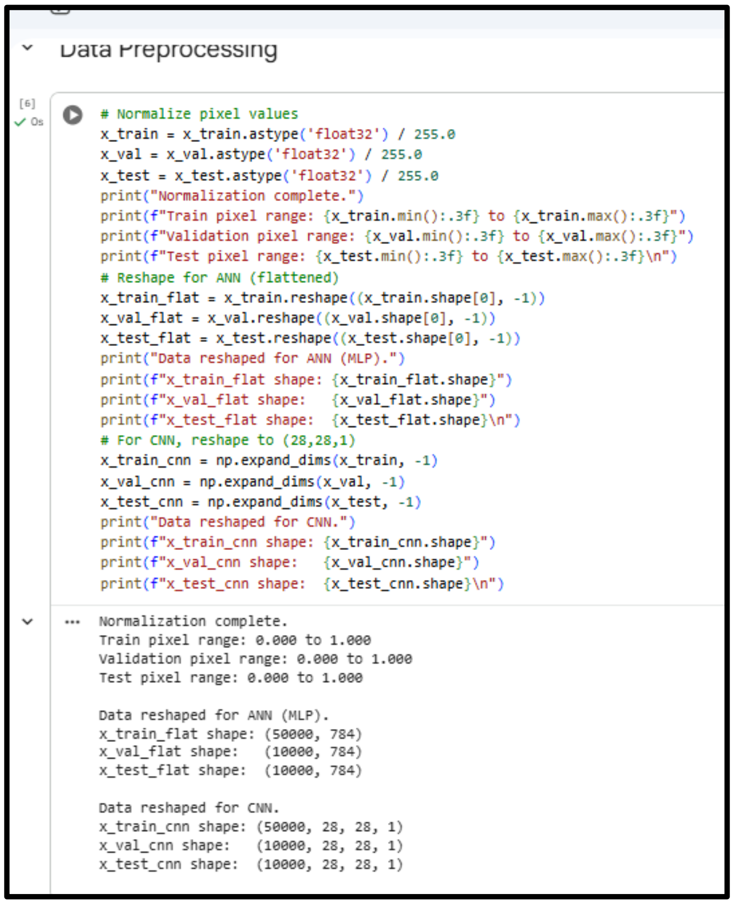
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
29515/29515 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26421880/26421880 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
5148/5148 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Training data shape: (50000, 28, 28)
Validation data shape: (10000, 28, 28)
Test data shape: (10000, 28, 28)
```

Figure 1: Loading the dataset and splitting the data

(Source: Google Colab)

TensorFlow was used to load "Fashion-MNIST" with "keras.datasets". It also has "60,000 training and 10,000 test images". The training set was separated to obtain 10,000 samples, which were used to form validation, thus remaining "50,000 training, 10,000 validation and 10,000 test 28x28 pixel images".

2.3 Data Normalization



```python
# Normalize pixel values
x_train = x_train.astype('float32') / 255.0
x_val = x_val.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
print("Normalization complete.")
print(f"Train pixel range: {x_train.min():.3f} to {x_train.max():.3f}")
print(f"Validation pixel range: {x_val.min():.3f} to {x_val.max():.3f}")
print(f"Test pixel range: {x_test.min():.3f} to {x_test.max():.3f}\n")
# Reshape for ANN (flattened)
x_train_flat = x_train.reshape((x_train.shape[0], -1))
x_val_flat = x_val.reshape((x_val.shape[0], -1))
x_test_flat = x_test.reshape((x_test.shape[0], -1))
print("Data reshaped for ANN (MLP).")
print(f"x_train_flat shape: {x_train_flat.shape}")
print(f"x_val_flat shape:   {x_val_flat.shape}")
print(f"x_test_flat shape:  {x_test_flat.shape}\n")
# For CNN, reshape to (28,28,1)
x_train_cnn = np.expand_dims(x_train, -1)
x_val_cnn = np.expand_dims(x_val, -1)
x_test_cnn = np.expand_dims(x_test, -1)
print("Data reshaped for CNN.")
print(f"x_train_cnn shape: {x_train_cnn.shape}")
print(f"x_val_cnn shape:   {x_val_cnn.shape}")
print(f"x_test_cnn shape:  {x_test_cnn.shape}\n")
```

```
Normalization complete.
Train pixel range: 0.000 to 1.000
Validation pixel range: 0.000 to 1.000
Test pixel range: 0.000 to 1.000

Data reshaped for ANN (MLP).
x_train_flat shape: (50000, 784)
x_val_flat shape:   (10000, 784)
x_test_flat shape:  (10000, 784)

Data reshaped for CNN.
x_train_cnn shape: (50000, 28, 28, 1)
x_val_cnn shape:   (10000, 28, 28, 1)
x_test_cnn shape:  (10000, 28, 28, 1)
```

Figure 2: Data Preprocessing

(Source: Google Colab)

The pixel values were rescaled to 0-1 to guarantee a faster convergence. This was then reformatted in a flattened form of 784 features to be used by ANN or 28 x 28 x 1 dimensions to be used by CNNs, to facilitate convolutional processing.

2.4 Exploratory Data Analysis and Visualizations

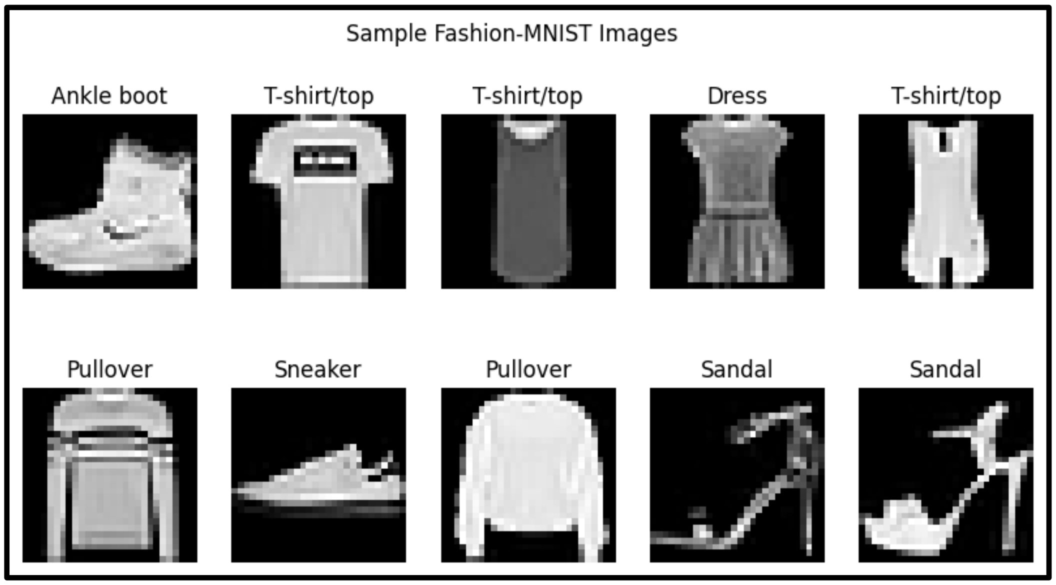


Figure 3: Data Visualisation

(Source: Google Colab)

In order to visually examine the data, a sample set of ten training set images was presented to ensure the appropriate labelling and grayscale of all clothing categories.

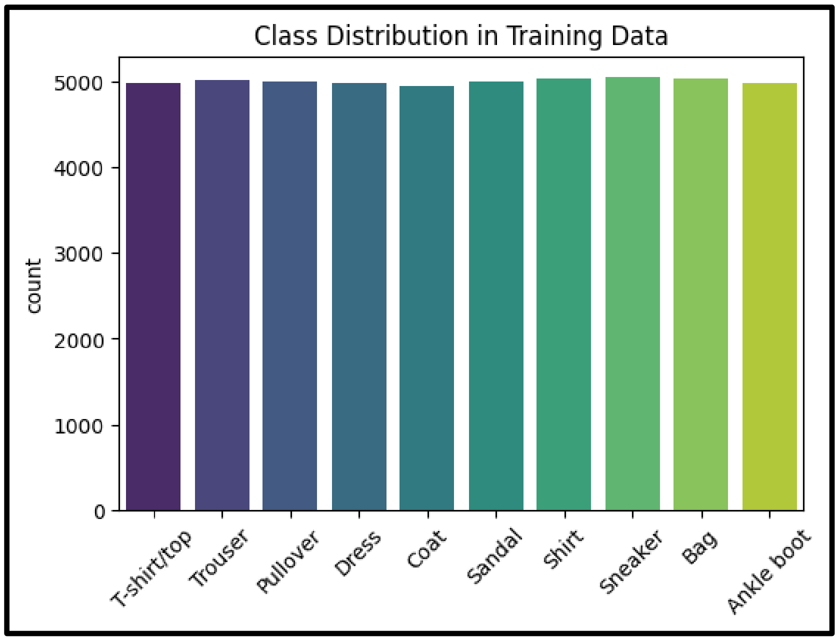


Figure 4: Class distribution

(Source: Google Colab)

The plot of the class distribution reveals that the ten categories of clothes in the Fashion-MNIST training dataset are represented evenly, which would result in a balanced dataset to be used in supervised learning.

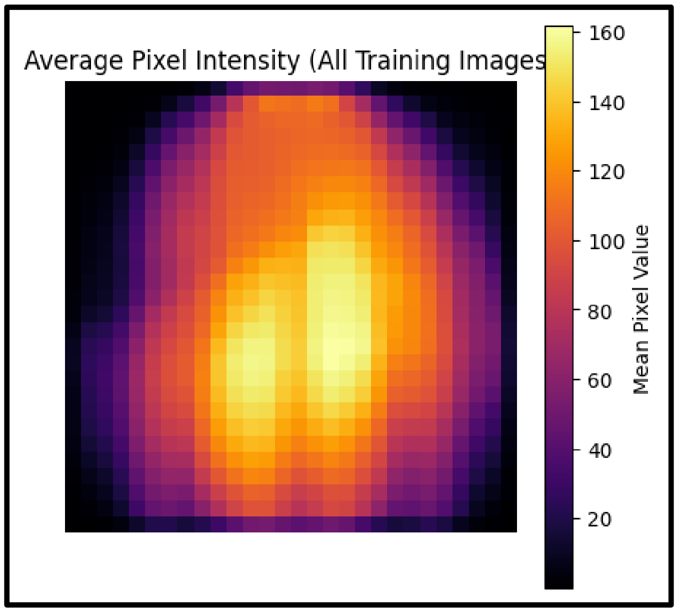


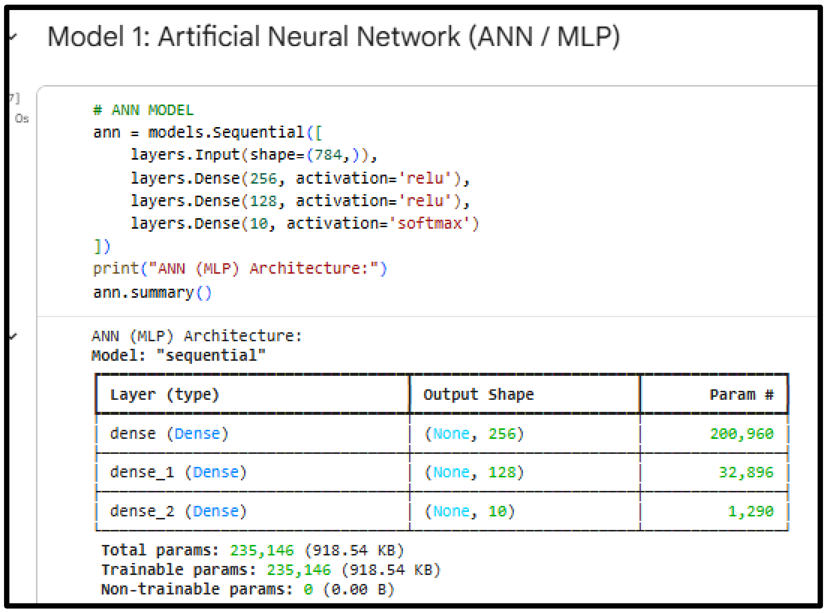
Figure 5: Mean Pixel Intensity Heatmap

(Source: Google Colab)

The heatmap shows the average intensity of pixels in all training images, which show brighter regions in the middle and darker ones in the corners, demonstrating that the majority of clothing pieces will be placed at the centre of the image.

3. Network Topology

3.1 Artificial Neural Network (ANN) Architecture


Model 1: Artificial Neural Network (ANN / MLP)

```
# ANN MODEL
ann = models.Sequential([
    layers.Input(shape=(784,)),
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
print("ANN (MLP) Architecture:")
ann.summary()
```

```
ANN (MLP) Architecture:
Model: "sequential"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                   ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ dense (Dense)                  │ (None, 256)            │       200,960 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                │ (None, 128)            │        32,896 │
├────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_2 (Dense)                │ (None, 10)             │         1,290 │
└────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 235,146 (918.54 KB)
 Trainable params: 235,146 (918.54 KB)
 Non-trainable params: 0 (0.00 B)
```

Figure 6: ANN Model Architecture

(Source: Google Colab)

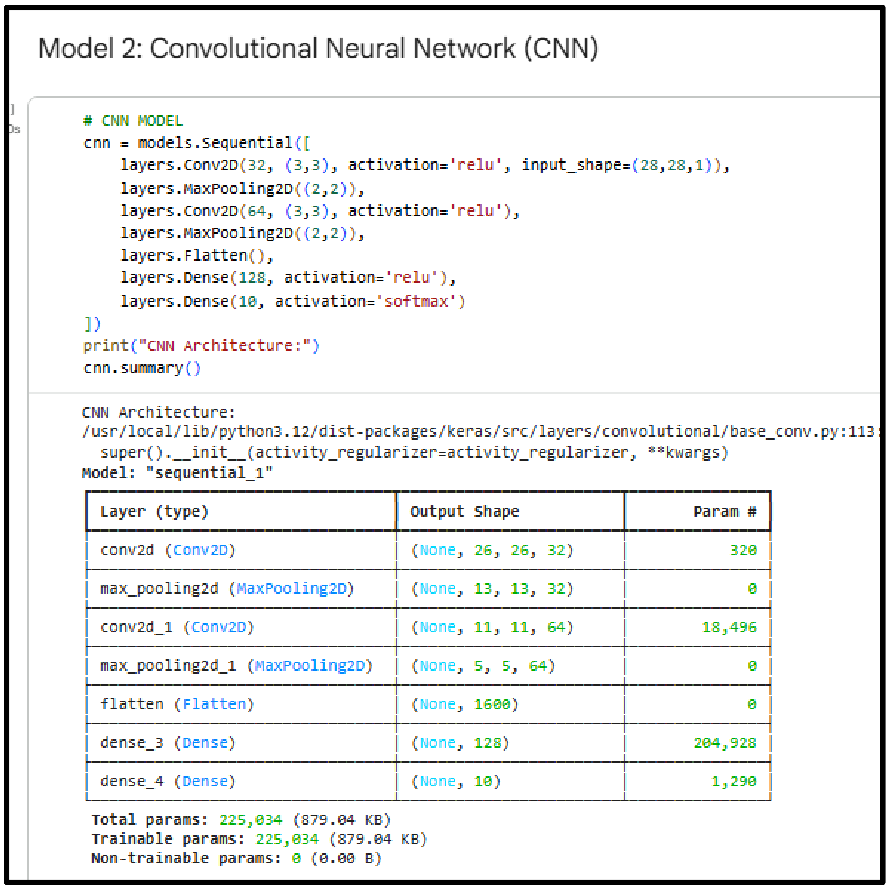
Layer Type	Output Shape	Activation	Parameters
“Dense (Input)”	“(None, 256)”	“ReLU”	200,960
“Dense”	“(None, 128)”	“ReLU”	32,896
“Dense (Output)”	“(None, 10)”	“Softmax”	1,290
Total	—	—	**235,146**

Table 1: ANN (MLP) Architecture

(Source: Self-created)

The ANN is made of three fully connected layers, two hidden layers with 256 and 128 neurons with the help of the ReLU function, and a softmax output layer with 10 classes. The selected architecture has a trade-off between learning capacity and computer efficiency. ReLU thus achieves faster convergence, whereas softmax activation on the output layer allows probabilistic interpretation of fashion categories, which is applicable to multi-class image classification.

3.2 Convolutional Neural Network (CNN) Architecture



Model 2: Convolutional Neural Network (CNN)

```
# CNN MODEL
cnn = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
print("CNN Architecture:")
cnn.summary()
```

```
CNN Architecture:
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_1"
┌────────────────────────────────┬──────────────────────────┬───────────────┐
│ Layer (type)                   │ Output Shape             │       Param # │
├────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d (Conv2D)                │ (None, 26, 26, 32)       │           320 │
├────────────────────────────────┼──────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)   │ (None, 13, 13, 32)       │             0 │
├────────────────────────────────┼──────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)              │ (None, 11, 11, 64)       │        18,496 │
├────────────────────────────────┼──────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D) │ (None, 5, 5, 64)         │             0 │
├────────────────────────────────┼──────────────────────────┼───────────────┤
│ flatten (Flatten)              │ (None, 1600)             │             0 │
├────────────────────────────────┼──────────────────────────┼───────────────┤
│ dense_3 (Dense)                │ (None, 128)              │       204,928 │
├────────────────────────────────┼──────────────────────────┼───────────────┤
│ dense_4 (Dense)                │ (None, 10)               │         1,290 │
└────────────────────────────────┴──────────────────────────┴───────────────┘
 Total params: 225,034 (879.04 KB)
 Trainable params: 225,034 (879.04 KB)
 Non-trainable params: 0 (0.00 B)
```

Figure 7: CNN Model Architecture

(Source: Google Colab)

Layer Type	**“Output Shape”**	**“Activation”**	**“Parameters”**
“Conv2D (32 filters, 3×3)”	(None, 26, 26, 32)	ReLU	320
“MaxPooling2D”	(None, 13, 13, 32)	—	0
“Conv2D (64 filters, 3×3)”	(None, 11, 11, 64)	ReLU	18,496

“MaxPooling2D”	(None, 5, 5, 64)	—	0
“Flatten”	(None, 1600)	—	0
“Dense”	(None, 128)	ReLU	204,928
“Dense (Output)”	(None, 10)	Softmax	1,290
Total	—	—	**225,034**

Table 2: CNN Architecture

(Source: Self-created)

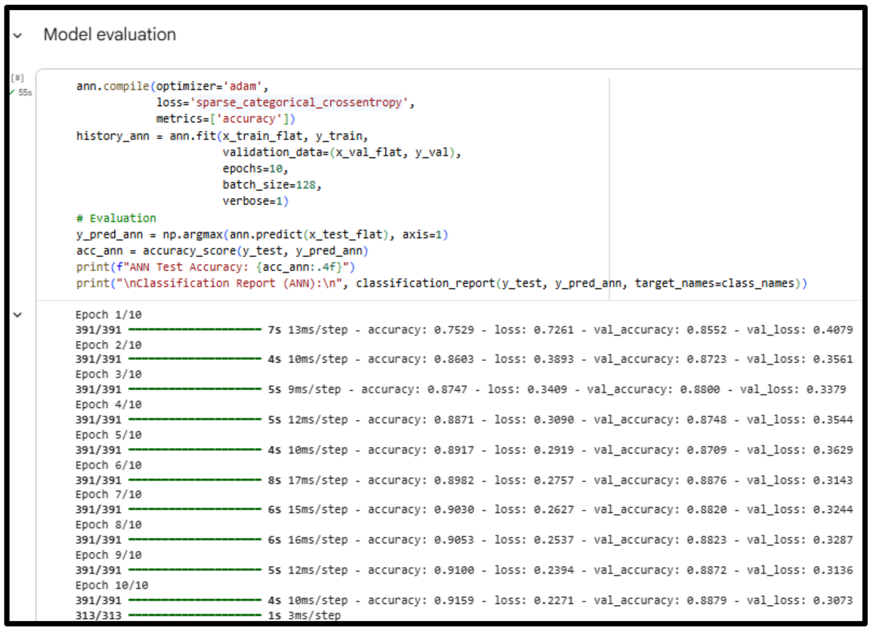
The CNN structure consists of two convolutional layers (32 and 64 filters), which are used to extract the spatial features and thereafter a dimensional reduction is achieved with max-pooling layers (Beohar and Rasool, 2021). In dense layers, feature maps are converted to a 1D vector by a flattening layer. ReLU activation makes the learning process quicker, whereas softmax classifies ten fashion classes. This topology provides hierarchical image properties, thus enhancing recognition accuracy compared to ANN, since the topology maintains spatial relationships.

4. Training Setup

4.1 Optimizer and Loss Function

The two models were coded with the Adam optimiser that adjusts the learning rates in the course of training to achieve efficient convergence. The categorical cross-entropy loss utility was employed because the task required multi-class classification based on ten Fashion-MNIST classes (Xhaferra *et al.* 2022). This combination provides a stable change in gradient and reduces the error in classification.

4.2 Learning Rate and Hyperparameters

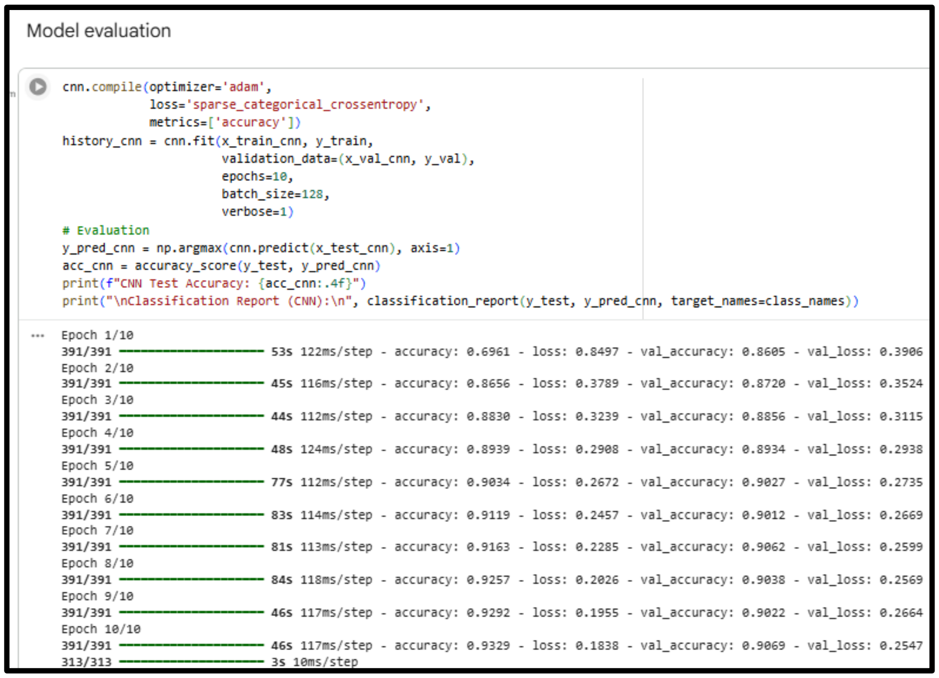

Model evaluation

```python
ann.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
history_ann = ann.fit(x_train_flat, y_train,
                      validation_data=(x_val_flat, y_val),
                      epochs=10,
                      batch_size=128,
                      verbose=1)
# Evaluation
y_pred_ann = np.argmax(ann.predict(x_test_flat), axis=1)
acc_ann = accuracy_score(y_test, y_pred_ann)
print(f"ANN Test Accuracy: {acc_ann:.4f}")
print("\nClassification Report (ANN):\n", classification_report(y_test, y_pred_ann, target_names=class_names))
```

```
Epoch 1/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 7s 13ms/step - accuracy: 0.7529 - loss: 0.7261 - val_accuracy: 0.8552 - val_loss: 0.4079
Epoch 2/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.8603 - loss: 0.3893 - val_accuracy: 0.8723 - val_loss: 0.3561
Epoch 3/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 5s 9ms/step - accuracy: 0.8747 - loss: 0.3409 - val_accuracy: 0.8800 - val_loss: 0.3379
Epoch 4/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.8871 - loss: 0.3090 - val_accuracy: 0.8748 - val_loss: 0.3544
Epoch 5/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.8917 - loss: 0.2919 - val_accuracy: 0.8709 - val_loss: 0.3629
Epoch 6/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 8s 17ms/step - accuracy: 0.8982 - loss: 0.2757 - val_accuracy: 0.8876 - val_loss: 0.3143
Epoch 7/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 6s 15ms/step - accuracy: 0.9030 - loss: 0.2627 - val_accuracy: 0.8820 - val_loss: 0.3244
Epoch 8/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - accuracy: 0.9053 - loss: 0.2537 - val_accuracy: 0.8823 - val_loss: 0.3287
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.9100 - loss: 0.2394 - val_accuracy: 0.8872 - val_loss: 0.3136
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.9159 - loss: 0.2271 - val_accuracy: 0.8879 - val_loss: 0.3073
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```

Figure 8: ANN Model Training

(Source: Google Colab)


Model evaluation

```python
cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
history_cnn = cnn.fit(x_train_cnn, y_train,
                      validation_data=(x_val_cnn, y_val),
                      epochs=10,
                      batch_size=128,
                      verbose=1)
# Evaluation
y_pred_cnn = np.argmax(cnn.predict(x_test_cnn), axis=1)
acc_cnn = accuracy_score(y_test, y_pred_cnn)
print(f"CNN Test Accuracy: {acc_cnn:.4f}")
print("\nClassification Report (CNN):\n", classification_report(y_test, y_pred_cnn, target_names=class_names))
```

```
Epoch 1/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 53s 122ms/step - accuracy: 0.6961 - loss: 0.8497 - val_accuracy: 0.8605 - val_loss: 0.3906
Epoch 2/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 45s 116ms/step - accuracy: 0.8656 - loss: 0.3789 - val_accuracy: 0.8720 - val_loss: 0.3524
Epoch 3/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 44s 112ms/step - accuracy: 0.8830 - loss: 0.3239 - val_accuracy: 0.8856 - val_loss: 0.3115
Epoch 4/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 48s 124ms/step - accuracy: 0.8939 - loss: 0.2908 - val_accuracy: 0.8934 - val_loss: 0.2938
Epoch 5/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 77s 112ms/step - accuracy: 0.9034 - loss: 0.2672 - val_accuracy: 0.9027 - val_loss: 0.2735
Epoch 6/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 83s 114ms/step - accuracy: 0.9119 - loss: 0.2457 - val_accuracy: 0.9012 - val_loss: 0.2669
Epoch 7/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 81s 113ms/step - accuracy: 0.9163 - loss: 0.2285 - val_accuracy: 0.9062 - val_loss: 0.2599
Epoch 8/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 84s 118ms/step - accuracy: 0.9257 - loss: 0.2026 - val_accuracy: 0.9038 - val_loss: 0.2569
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 46s 117ms/step - accuracy: 0.9292 - loss: 0.1955 - val_accuracy: 0.9022 - val_loss: 0.2664
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 46s 117ms/step - accuracy: 0.9329 - loss: 0.1838 - val_accuracy: 0.9069 - val_loss: 0.2547
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
```

Figure 9: CNN Model Training

(Source: Google Colab)

The default Adam learning rate of 0.001 was used on both models. Other hyperparameters were ReLU activation in the hidden layers, and softmax in the outputs and accuracy as the main measure of evaluation. The parameters gave a trade-off between generalisation and the speed of convergence.

4.3 Batch Size and Epochs

A batch size of 128 with 10 epochs was used for every model in training, such that there was adequate one to converge to the model but not to an extent of overfitting (Shaeri *et al.* 2025).

4.4 Training Environment and Framework

All the experiments were carried out in Google Colab with TensorFlow (Keras) backend with GPU acceleration. Figures 8 and 9 illustrate training in the ANN model and CNN training performance, respectively, which demonstrate the improvement in the accuracy and loss with epochs.

5. Evaluation and Results

5.1 Model Accuracy and Loss

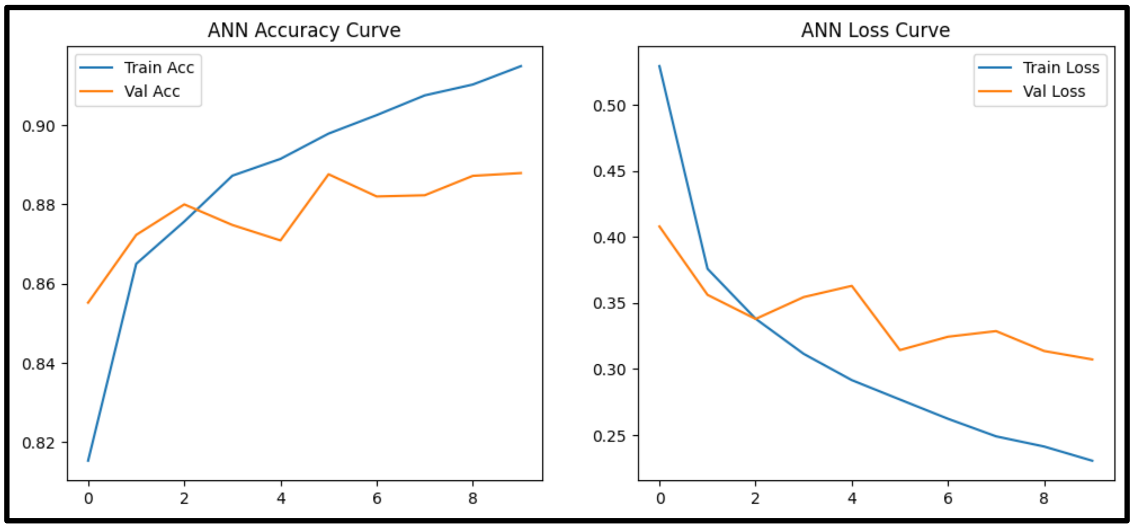


Figure 10: ANN Model Accuracy and Loss Curve

(Source: Google Colab)

The ANN model has a consistent improvement with the increase in the number of epochs and eventually reaches around 91.6 percent of training accuracy and 88 percent of validation accuracy. Both the accuracy and loss curves show stable learning with low overfitting and the convergence point is stable.

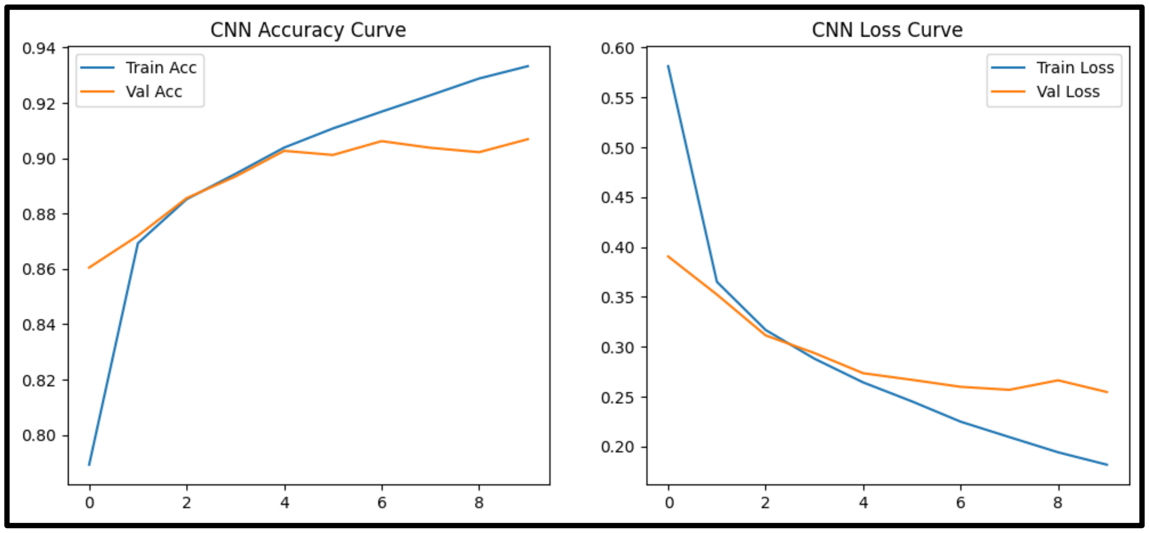
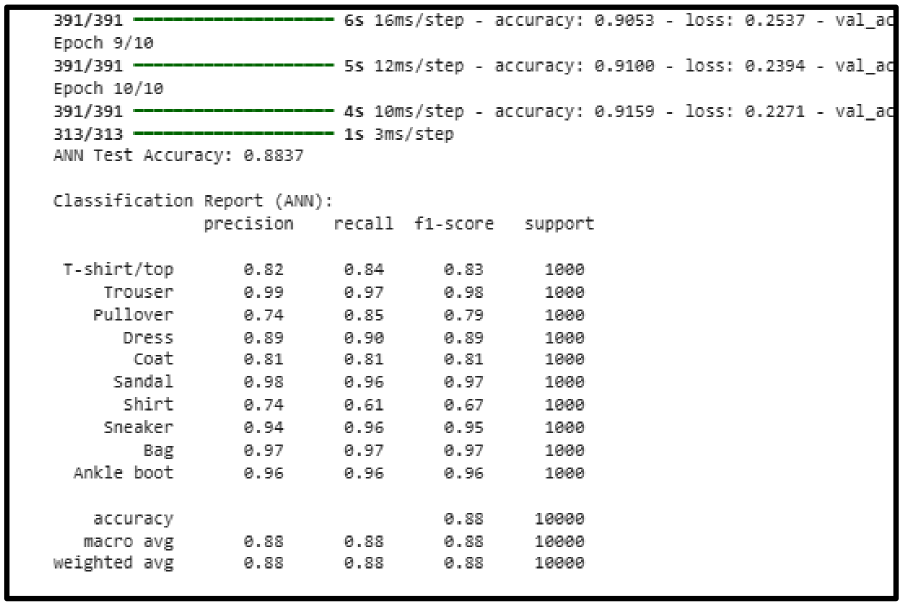


Figure 11: CNN Model Accuracy and Loss Curve

(Source: Google Colab)

The CNN model has the limits of around 93.3% training and 90.7% validation accuracy, which is higher generalization in comparison to the ANN. The smooth convergence demonstrating very limited overfitting is observed by accuracy and loss curves, confirming that approximately, features are learning.

5.2 Class-wise Performance Metrics

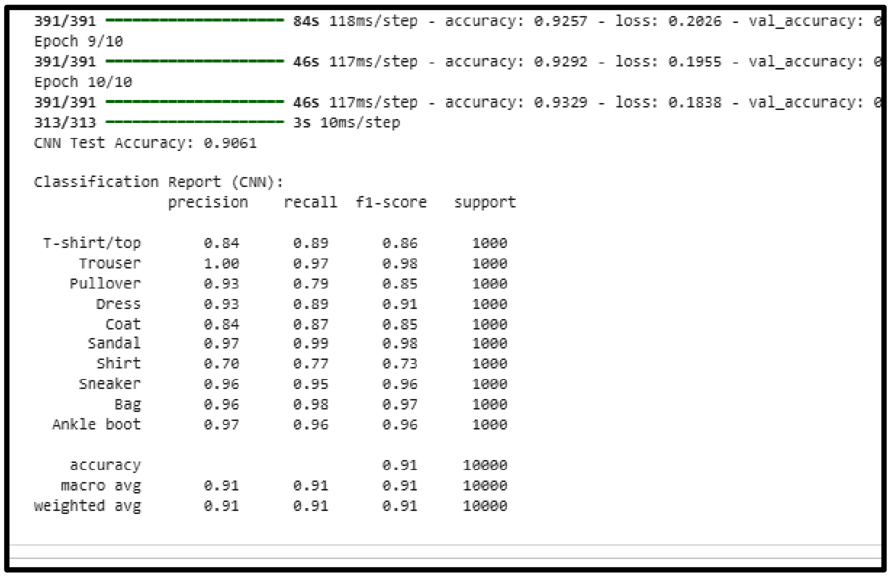


```
391/391 ━━━━━━━━━━━━━━━━━━━━ 6s 16ms/step - accuracy: 0.9053 - loss: 0.2537 - val_ac
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 5s 12ms/step - accuracy: 0.9100 - loss: 0.2394 - val_ac
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 4s 10ms/step - accuracy: 0.9159 - loss: 0.2271 - val_ac
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
ANN Test Accuracy: 0.8837

Classification Report (ANN):
               precision    recall  f1-score   support

 T-shirt/top       0.82      0.84      0.83      1000
     Trouser       0.99      0.97      0.98      1000
    Pullover       0.74      0.85      0.79      1000
       Dress       0.89      0.90      0.89      1000
        Coat       0.81      0.81      0.81      1000
      Sandal       0.98      0.96      0.97      1000
       Shirt       0.74      0.61      0.67      1000
     Sneaker       0.94      0.96      0.95      1000
         Bag       0.97      0.97      0.97      1000
  Ankle boot       0.96      0.96      0.96      1000

    accuracy                           0.88     10000
   macro avg       0.88      0.88      0.88     10000
weighted avg       0.88      0.88      0.88     10000
```

Figure 12: ANN Model Results

(Source: Google Colab)

The ANN scored 88.4 per cent in the test, with good performance in the majority of classes. Nevertheless, there was observed confusion between similar categories that were visually similar, such as shirt and coat, meaning that there are restrictions in the capture of spatial specificities.



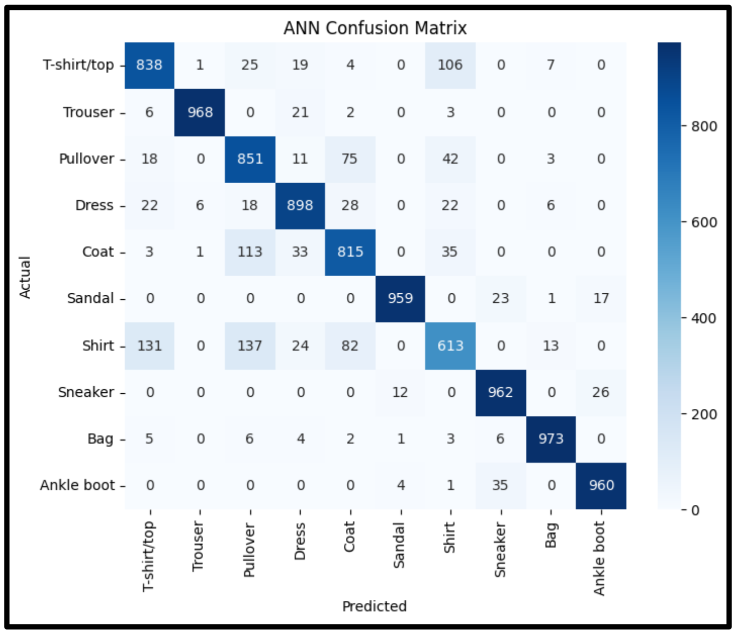
```
391/391 ━━━━━━━━━━━━━━━━━━━━ 84s 118ms/step - accuracy: 0.9257 - loss: 0.2026 - val_accuracy: 0
Epoch 9/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 46s 117ms/step - accuracy: 0.9292 - loss: 0.1955 - val_accuracy: 0
Epoch 10/10
391/391 ━━━━━━━━━━━━━━━━━━━━ 46s 117ms/step - accuracy: 0.9329 - loss: 0.1838 - val_accuracy: 0
313/313 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step
CNN Test Accuracy: 0.9061

Classification Report (CNN):
              precision    recall  f1-score   support

 T-shirt/top       0.84      0.89      0.86      1000
     Trouser       1.00      0.97      0.98      1000
    Pullover       0.93      0.79      0.85      1000
       Dress       0.93      0.89      0.91      1000
        Coat       0.84      0.87      0.85      1000
      Sandal       0.97      0.99      0.98      1000
       Shirt       0.70      0.77      0.73      1000
     Sneaker       0.96      0.95      0.96      1000
         Bag       0.96      0.98      0.97      1000
  Ankle boot       0.97      0.96      0.96      1000

    accuracy                           0.91     10000
   macro avg       0.91      0.91      0.91     10000
weighted avg       0.91      0.91      0.91     10000
```

Figure 13: CNN Model Results

(Source: Google Colab)

The CNN had a test percentage of 90.6, making it better than the ANN. It was able to identify the majority of the classes but had some confusion between the shirt and pullover, owing to their mate visual textures.

5.3 Confusion Matrix Analysis

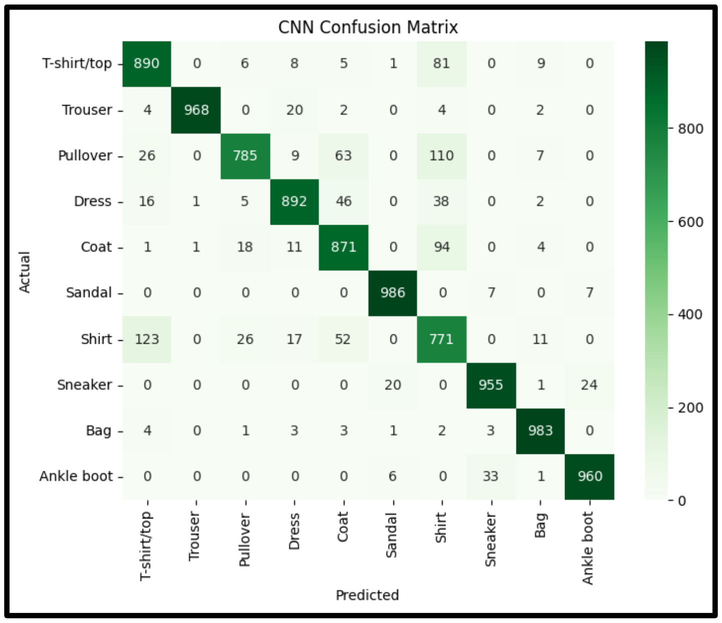


	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	838	1	25	19	4	0	106	0	7	0
Trouser	6	968	0	21	2	0	3	0	0	0
Pullover	18	0	851	11	75	0	42	0	3	0
Dress	22	6	18	898	28	0	22	0	6	0
Coat	3	1	113	33	815	0	35	0	0	0
Sandal	0	0	0	0	0	959	0	23	1	17
Shirt	131	0	137	24	82	0	613	0	13	0
Sneaker	0	0	0	0	0	12	0	962	0	26
Bag	5	0	6	4	2	1	3	6	973	0
Ankle boot	0	0	0	0	0	4	1	35	0	960

Figure 14: ANN Confusion Matrix

(Source: Google Colab)

The “ANN confusion matrix” indicates high accuracy rates in such classes as Trouser, Sneaker and Bag; however, there are large levels of misclassification between Shirt, T-shirt/top and Pullover, suggesting that it is not easy to distinguish between visually similar clothes.



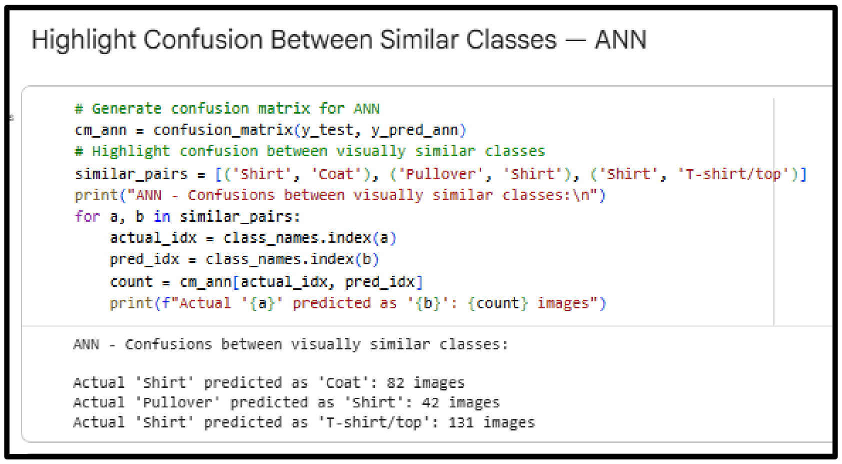
	T-shirt/top	Trouser	Pullover	Dress	Coat	Sandal	Shirt	Sneaker	Bag	Ankle boot
T-shirt/top	890	0	6	8	5	1	81	0	9	0
Trouser	4	968	0	20	2	0	4	0	2	0
Pullover	26	0	785	9	63	0	110	0	7	0
Dress	16	1	5	892	46	0	38	0	2	0
Coat	1	1	18	11	871	0	94	0	4	0
Sandal	0	0	0	0	0	986	0	7	0	7
Shirt	123	0	26	17	52	0	771	0	11	0
Sneaker	0	0	0	0	0	20	0	955	1	24
Bag	4	0	1	3	3	1	2	3	983	0
Ankle boot	0	0	0	0	0	6	0	33	1	960

Figure 15: CNN Confusion Matrix

(Source: Google Colab)

The “CNN confusion matrix” shows good classification accuracy, particularly on Bag, Sandal and Trouser. There is still slight confusion between Shirt, T-shirt/top and Coat, indicating the visual resemblance between the upper-body garment classes.

5.4 Model Confusion



Highlight Confusion Between Similar Classes — ANN

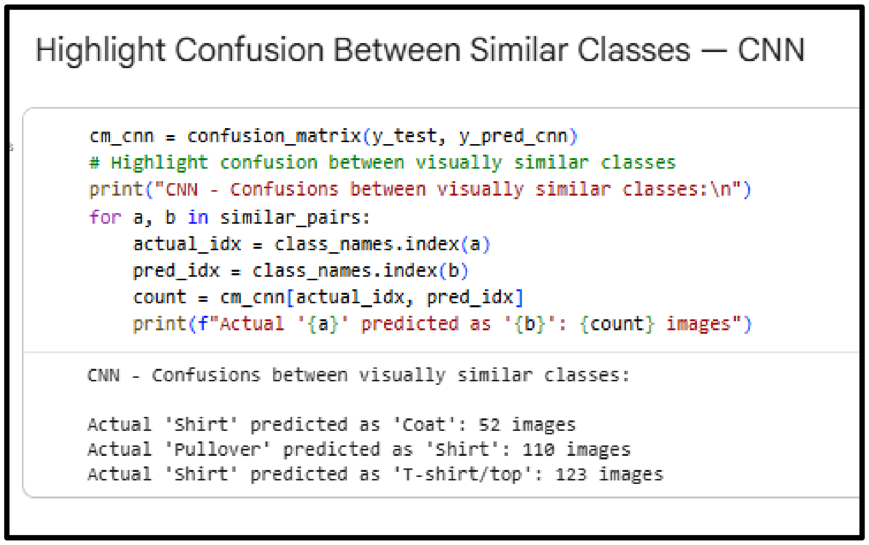
```
# Generate confusion matrix for ANN
cm_ann = confusion_matrix(y_test, y_pred_ann)
# Highlight confusion between visually similar classes
similar_pairs = [('Shirt', 'Coat'), ('Pullover', 'Shirt'), ('Shirt', 'T-shirt/top')]
print("ANN - Confusions between visually similar classes:\n")
for a, b in similar_pairs:
    actual_idx = class_names.index(a)
    pred_idx = class_names.index(b)
    count = cm_ann[actual_idx, pred_idx]
    print(f"Actual '{a}' predicted as '{b}': {count} images")
```

```
ANN - Confusions between visually similar classes:

Actual 'Shirt' predicted as 'Coat': 82 images
Actual 'Pullover' predicted as 'Shirt': 42 images
Actual 'Shirt' predicted as 'T-shirt/top': 131 images
```

Figure 16: ANN Model Confusion

(Source: Google Colab)

The ANN model demonstrates significant classification error between similar classes in appearance, T-shirts, which should have been classified as Shirts (131) and Coats (82), and Pullovers frequently were represented as Shirts (42), which demonstrates that it does not have so much spatial feature recognition.



Highlight Confusion Between Similar Classes — CNN

```
cm_cnn = confusion_matrix(y_test, y_pred_cnn)
# Highlight confusion between visually similar classes
print("CNN - Confusions between visually similar classes:\n")
for a, b in similar_pairs:
    actual_idx = class_names.index(a)
    pred_idx = class_names.index(b)
    count = cm_cnn[actual_idx, pred_idx]
    print(f"Actual '{a}' predicted as '{b}': {count} images")
```

```
CNN - Confusions between visually similar classes:

Actual 'Shirt' predicted as 'Coat': 52 images
Actual 'Pullover' predicted as 'Shirt': 110 images
Actual 'Shirt' predicted as 'T-shirt/top': 123 images
```

Figure 17: Confusion Between Similar Classes — CNN

(Source: Google Colab)

CNN model minimized the confusion significantly, as compared to ANN, but still Shirts were mistaken as T-shirts (123) and Coats (52). Pullovers shared the visual overlap with Shirts, where they were confused with them (110).

6. Critical Analysis

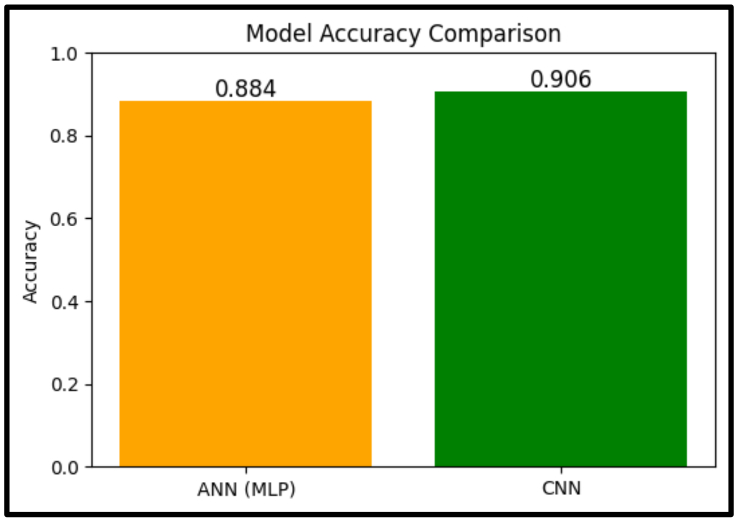


Figure 18: Model Comparison

(Source: Google Colab)

The CNN performed a lot better than the ANN, with 90.6 and 88.4 test accuracy, respectively. The convolutional layers of CNN are used to detect spatial and texture features and enhance the recognition of detailed clothing features. Nevertheless, the two models had poor performance with visually similar categories, such as Shirt, T-shirt and Coat, because of overlapping pixel structure (Nocentini *et al.* 2022). Although ANN was trained quickly and used fewer hard powers, it did not have spatial recognition. CNN was more accurate, but it required more resources and time to train. The two models have good generalization, with CNN being more applicable to image tasks.

7. Reflection and Improvements

7.1 Potential Enhancements

Future directions would be to augment the data (rotation, flipping, and zooming) to make the datasets more diverse and to minimise overfitting and improve training and model stability (Bbouzidi *et al.* 2024). The dropout layers, the use of batch normalization and the learning rate schedule might be the improvements to make the training process more stable and to raise the overall accuracy and the generalization rate of the model produced.

7.2 Future Work and Extensions

Future directions might look to realise more complex CNNs, including ResNet or VGG, to extract features in more depth. It is possible to enhance the accuracy through the incorporation of transfer learning and ensemble techniques. Also, Grad-CAM-generated real-time deployment and interpretability would be more informative for model-based decision-making.

References

Bbouzidi, S., Hcini, G., Jdey, I. and Drira, F., 2024. Convolutional neural networks and vision transformers for fashion mnist classification: A literature review. arXiv preprint arXiv:2406.03478.

Beohar, D. and Rasool, A., 2021, March. Handwritten digit recognition of MNIST dataset using deep learning state-of-the-art artificial neural network (ANN) and convolutional neural network (CNN). In 2021 International conference on emerging smart computing and informatics (ESCI) (pp. 542-548). IEEE.

Nocentini, O., Kim, J., Bashir, M.Z. and Cavallo, F., 2022. Image classification using multiple convolutional neural networks on the fashion-MNIST dataset. Sensors, 22(23), p.9544.

Sumera, S.R., Anjum, N. and Vaidehi, K., 2022. Implementation of CNN and ANN for Fashion-MNIST-Dataset using Different Optimizers. Indian Journal of Science and Technology, 15(47), pp.2639-2645.

Shaeri, P., Karimi, A. and Middel, A., 2025. Mnist-gen: A modular mnist-style dataset generation using hierarchical semantics, reinforcement learning, and category theory. arXiv preprint arXiv:2507.11821.

Xhaferra, E., Cina, E. and Toti, L., 2022, October. Classification of standard fashion MNIST dataset using deep learning based CNN algorithms. In 2022 international symposium on multidisciplinary studies and innovative technologies (ISMSIT) (pp. 494-498). IEEE.

Appendix

Python Code

```
“ “ “  “
# SETUP AND IMPORTS
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from tensorflow import keras
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

"""## Loading and splitting data"""

# Load Fashion-MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.fashion_mnist.load_data()
# Split training data into training and validation sets
x_val = x_train[-10000:]
y_val = y_train[-10000:]
x_train = x_train[:-10000]
y_train = y_train[:-10000]
# Class labels
class_names = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']
print(f"Training data shape: {x_train.shape}")
print(f"Validation data shape: {x_val.shape}")
print(f"Test data shape: {x_test.shape}")

"""## Basic Data Exploration and Visualizations"""

# Display few sample images
plt.figure(figsize=(10, 5))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(class_names[y_train[i]])
    plt.axis('off')
plt.suptitle("Sample Fashion-MNIST Images")
plt.show()

# Class Distribution
plt.figure(figsize=(6,4))
sns.countplot(x=y_train, palette="viridis")
plt.title("Class Distribution in Training Data")
plt.xticks(ticks=np.arange(10), labels=class_names, rotation=45)
plt.show()
```

```
# Mean Pixel Intensity Heatmap
mean_image = np.mean(x_train, axis=0)
plt.figure(figsize=(5,5))
plt.imshow(mean_image, cmap='inferno')
plt.title("Average Pixel Intensity (All Training Images)")
plt.axis('off')
plt.colorbar(label='Mean Pixel Value')
plt.show()

"""## Data Preprocessing"""

# Normalize pixel values
x_train = x_train.astype('float32') / 255.0
x_val = x_val.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
print("Normalization complete.")
print(f"Train pixel range: {x_train.min():.3f} to {x_train.max():.3f}")
print(f"Validation pixel range: {x_val.min():.3f} to {x_val.max():.3f}")
print(f"Test pixel range: {x_test.min():.3f} to {x_test.max():.3f}\n")
# Reshape for ANN (flattened)
x_train_flat = x_train.reshape((x_train.shape[0], -1))
x_val_flat = x_val.reshape((x_val.shape[0], -1))
x_test_flat = x_test.reshape((x_test.shape[0], -1))
print("Data reshaped for ANN (MLP).")
print(f"x_train_flat shape: {x_train_flat.shape}")
print(f"x_val_flat shape:   {x_val_flat.shape}")
print(f"x_test_flat shape:  {x_test_flat.shape}\n")
# For CNN, reshape to (28,28,1)
x_train_cnn = np.expand_dims(x_train, -1)
x_val_cnn = np.expand_dims(x_val, -1)
x_test_cnn = np.expand_dims(x_test, -1)
print("Data reshaped for CNN.")
print(f"x_train_cnn shape: {x_train_cnn.shape}")
print(f"x_val_cnn shape:   {x_val_cnn.shape}")
print(f"x_test_cnn shape:  {x_test_cnn.shape}\n")

"""## Model 1: Artificial Neural Network (ANN / MLP)"""

# ANN MODEL
ann = models.Sequential([
    layers.Input(shape=(784,)),
    layers.Dense(256, activation='relu'),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

```
print("ANN (MLP) Architecture:")
ann.summary()

"""### Model evaluation"""

ann.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
history_ann = ann.fit(x_train_flat, y_train,
                      validation_data=(x_val_flat, y_val),
                      epochs=10,
                      batch_size=128,
                      verbose=1)
# Evaluation
y_pred_ann = np.argmax(ann.predict(x_test_flat), axis=1)
acc_ann = accuracy_score(y_test, y_pred_ann)
print(f"ANN Test Accuracy: {acc_ann:.4f}")
print("\nClassification Report (ANN):\n", classification_report(y_test, y_pred_ann,
target_names=class_names))

# Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, y_pred_ann), annot=True, fmt='d', cmap='Blues',
            xticklabels=class_names, yticklabels=class_names)
plt.title("ANN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

"""## Highlight Confusion Between Similar Classes — ANN"""

# Generate confusion matrix for ANN
cm_ann = confusion_matrix(y_test, y_pred_ann)
# Highlight confusion between visually similar classes
similar_pairs = [('Shirt', 'Coat'), ('Pullover', 'Shirt'), ('Shirt', 'T-shirt/top')]
print("ANN - Confusions between visually similar classes:\n")
for a, b in similar_pairs:
    actual_idx = class_names.index(a)
    pred_idx = class_names.index(b)
    count = cm_ann[actual_idx, pred_idx]
    print(f"Actual '{a}' predicted as '{b}': {count} images")

# Accuracy & Loss Curves
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history_ann.history['accuracy'], label='Train Acc')
plt.plot(history_ann.history['val_accuracy'], label='Val Acc')
plt.title("ANN Accuracy Curve")
plt.legend()
plt.subplot(1,2,2)
plt.plot(history_ann.history['loss'], label='Train Loss')
plt.plot(history_ann.history['val_loss'], label='Val Loss')
plt.title("ANN Loss Curve")
plt.legend()
plt.show()

"""## Model 2: Convolutional Neural Network (CNN)"""

# CNN MODEL
cnn = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.MaxPooling2D((2,2)),
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.MaxPooling2D((2,2)),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dense(10, activation='softmax')
])
print("CNN Architecture:")
cnn.summary()

"""### Model evaluation"""

cnn.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])
history_cnn = cnn.fit(x_train_cnn, y_train,
                      validation_data=(x_val_cnn, y_val),
                      epochs=10,
                      batch_size=128,
                      verbose=1)
# Evaluation
y_pred_cnn = np.argmax(cnn.predict(x_test_cnn), axis=1)
acc_cnn = accuracy_score(y_test, y_pred_cnn)
print(f"CNN Test Accuracy: {acc_cnn:.4f}")
print("\nClassification Report (CNN):\n", classification_report(y_test, y_pred_cnn,
target_names=class_names))

# Confusion Matrix
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, y_pred_cnn), annot=True, fmt='d', cmap='Greens',
            xticklabels=class_names, yticklabels=class_names)
```

```
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# Accuracy & Loss Curves
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history_cnn.history['accuracy'], label='Train Acc')
plt.plot(history_cnn.history['val_accuracy'], label='Val Acc')
plt.title("CNN Accuracy Curve")
plt.legend()
plt.subplot(1,2,2)
plt.plot(history_cnn.history['loss'], label='Train Loss')
plt.plot(history_cnn.history['val_loss'], label='Val Loss')
plt.title("CNN Loss Curve")
plt.legend()
plt.show()

"""## Highlight Confusion Between Similar Classes — CNN"""

cm_cnn = confusion_matrix(y_test, y_pred_cnn)
# Highlight confusion between visually similar classes
print("CNN - Confusions between visually similar classes:\n")
for a, b in similar_pairs:
    actual_idx = class_names.index(a)
    pred_idx = class_names.index(b)
    count = cm_cnn[actual_idx, pred_idx]
    print(f"Actual '{a}' predicted as '{b}': {count} images")

"""## Final Model Comparison"""

# Compare Accuracy
models_list = ['ANN (MLP)', 'CNN']
accuracy_values = [acc_ann, acc_cnn]
plt.figure(figsize=(6,4))
bars = plt.bar(models_list, accuracy_values, color=['orange', 'green'])
plt.title("Model Accuracy Comparison")
plt.ylabel("Accuracy")
plt.ylim(0, 1)
# Add accuracy labels on top
for bar, acc in zip(bars, accuracy_values):
    plt.text(bar.get_x() + bar.get_width()/2, acc + 0.01, f"{acc:.3f}", ha='center', fontsize=12)
plt.show()
”””
```