CM1: Default Network

The dataset for this assignment is Fashion MNIST with a Twist. MNIST is a dataset of 28 x 28 grayscale fashion apparel images, consisting of two datasets:

Datasets provided	Training & Testing
Training Dataset Instances	60,000
Testing Dataset Instances	10,000
Features in an instance	784 features
Greyscale range	0 – 255
Number of labels	5

Table 1: features of the dataset

Data Preprocessing

After both the original datasets are loaded in the kernel memory the shape of the datasets is the following:

```
print(f"Shape of x_train is {fashion_raw_x_train.shape}")
print(f"Shape of x_test is {fashion_raw_x_test.shape}")
print(f"Shape of y_train is {fashion_raw_y_train.shape}")
print(f"Shape of y_test is {fashion_raw_y_test.shape}")
```

```
Shape of x_train is (60000, 784)
Shape of x_test is (10000, 784)
Shape of y_train is (60000, 1)
Shape of y_test is (10000, 1)
```

Figure 1: Shape of the original training & testing datasets

Since the data of each 28 x 28 pixels is in a single row format of 784 features in both x_train and x_test csv files, the shape of each instance of the training and the testing dataset needs to be altered before it could be fed to the default neural network. Furthermore, the y_train and y_test labels need to be altered as well for the neural network.

Therefore, the reshape features was used to convert the single row of 784 features into a 2D matrix of 28 x 28 for both x_train and x_test. The labels were converted into a categorical column. This is shown in Figure 2.

```
# convert the single array of 784 into 28x28 matrix for the training and testing dataset
fashion_x = fashion_raw_x_train.values.reshape(60000, 28, 28,1)/255
fashion_x_test =fashion_raw_x_test.values.reshape(10000, 28, 28,1)/255

# use one hot encoding to make seperate coloumns for each of the label
fashion_y = keras.utils.to_categorical(fashion_raw_y_train, 5)
fashion_y_test = keras.utils.to_categorical(fashion_raw_y_test, 5)
```

Figure 2: Reshaping of the original dataset

This was followed by the division of the x_train dataset into a training and validation datasets by using the train test split function with training set of size 0.8 and validation set of size 0.2. A random state of 27 was used as shown in Figure 3.

```
[12] fashion_x_train, fashion_x_val, fashion_y_train, fashion_y_val = train_test_split(fashion_x,
                                                                                       fashion_y, test_size=0.2, random_state=27)
```

Figure 3: Splitting of the training dataset

We can confirm the result of the preprocessing of the data by checking the shape of the data as shown in

```
print(f"Shape of training dataset after preprocessing: {fashion_x_train.shape}")
print(f"Shape of validation dataset after preprocessing: {fashion_x_val.shape}")
print(f"Shape of testing dataset after preprocessing: {fashion_x_test.shape}")

Shape of training dataset after preprocessing: (48000, 28, 28, 1)
Shape of validation dataset after preprocessing: (12000, 28, 28, 1)
Shape of testing dataset after preprocessing: (10000, 28, 28, 1)
```

Figure 4: Shape of the datasets after preprocessing

After the dataset is initialized and preprocessed for the neural network, the neural network is defined by using **Tensorflow** as shown in Figure 5.

```
# initialize the default CNN model
cm1 = tf.keras.models.Sequential()

# Add convolution 2D
cm1.add(layers.Conv2D(32, kernel_size=(3, 3), activation='relu',
                      kernel_initializer='he_normal', padding="same",
                      strides=(1,1), input_shape=(28, 28, 1)))

# Add a pooling layer
cm1.add(layers.MaxPooling2D((2, 2)))

# Add convolution 2D
cm1.add(layers.Conv2D(32, kernel_size=(3, 3), padding="same",
                      strides=(1,1), activation='relu'))

# Flatten the layer
cm1.add(layers.Flatten())

# Add the fully connected layer
cm1.add(layers.Dense(128, activation='relu'))

# Add the softmax function for the output layer
cm1.add(layers.Dense(5, activation='softmax'))


cm1.compile(loss=keras.losses.categorical_crossentropy, optimizer='adam',
            metrics=['accuracy'])
```

Figure 5: Structure of the default neural network

For the default neural network, the first layer is a convolution 2D layer with 32 filters, kernel size of 3 x 3, relu activation function, stride of (1,1) and input shape of 28 x 28 x 1. This is followed by the max pooling layer with (2,2). The max pooling layer is followed by another convolution 2D layer with similar settings as the previous convolution 2D layer. After the second convolution 2D layer, a flatten layer is added and then a dense layer is added with 128 nodes and relu activation function. Finally, the softmax layer is added in the end which will conduct the classification into the 5 labels. The visualization of the neural network is shown in Figure 6.

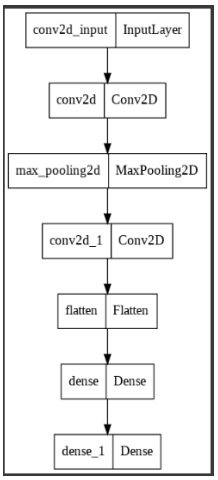

Figure 6: Visualization of cm1

The model will measure losses with cross entropy and the adam optimizer is used for the accuracy. The summary of the neural network is shown in Figure 7.

```
[ ]  cm1.summary()

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        320

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 32)        9248

 flatten (Flatten)           (None, 6272)              0

 dense (Dense)               (None, 128)               802944

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 813,157
Trainable params: 813,157
Non-trainable params: 0
```

Figure 7: Summary of the default network

After the model has been defined, the model was trained using the training and validation dataset with

```
cm1_model = cm1.fit(fashion_x_train, fashion_y_train, batch_size=128, epochs=50, verbose=1, validation_data=(fashion_x_val,fashion_y_val))
```

Figure 8: training of the default model

The results are discussed in CM3.