Machine Learning Project Report

Loading the required Libraries and the Dataset

As an initial step to complete the tasks, all the required libraries are imported.

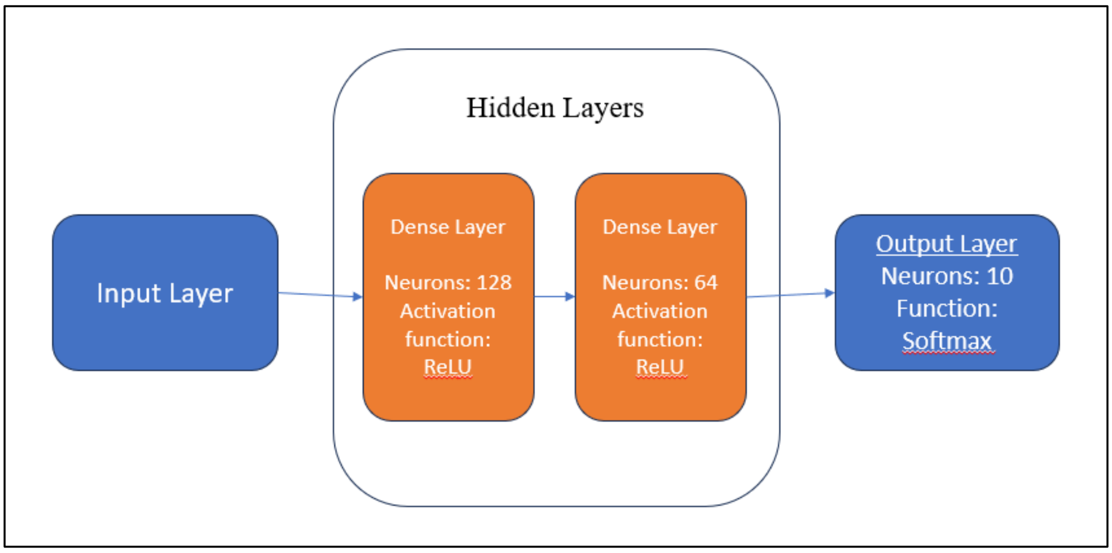
After importing the libraries and the dataset, the pixel values are normalized by dividing them by 255, to be in a range [0,1]. This makes the weights with small values, which doesn't slow down the learning process in Neural Networks.

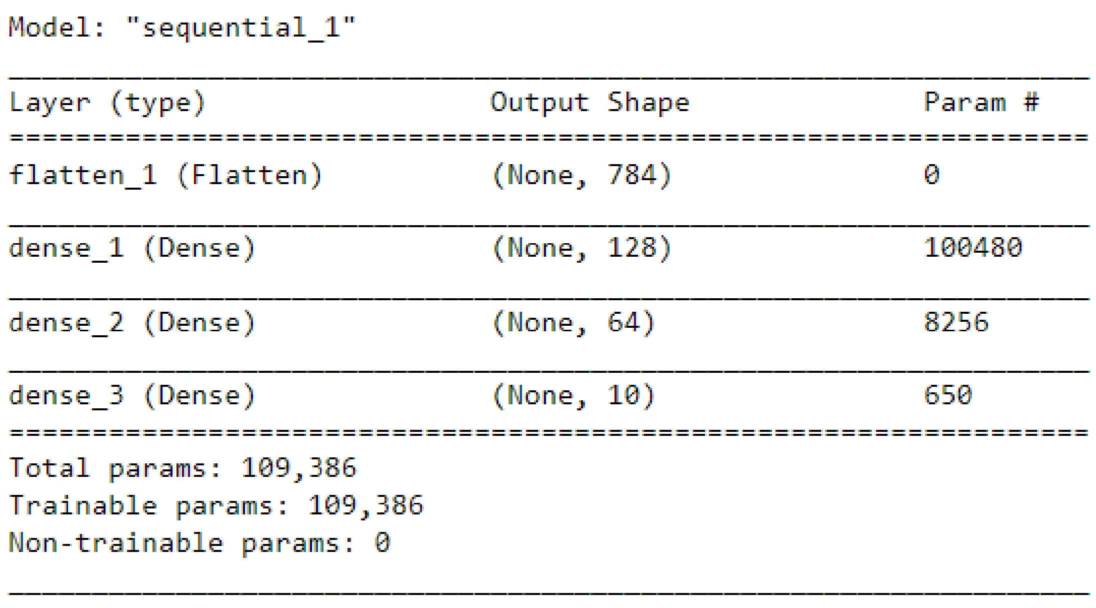
Encoding the labels (output) into one-hot vector, by converting the values to categorical values.

Task 1: Build a neural network without convolutional layers to do the classification task

Step 1: Defining a Model Architecture

- We start by defining a sequential model. A Sequential model is a linear stack of layers in which the layers are added one by one, through which the data flows sequentially.
```
model_T1 = Sequential()
```
- Input Layer: Then we add our first layer with input data, where a 2D image of 28*28 pixels is flattened into a 1D array of 784 values.
```
model_T1.add(Flatten(input_shape=(28, 28, 1)))
```
- Two hidden Layers: Then we add two dense layers one by one. The first layer has 128 neurons with 'ReLU' as an activation function. In simple words, 'ReLU' function outputs the input value if positive and zero otherwise. The second layer has 64 neurons with 'ReLU' as an activation function.
```
model_T1.add(Dense(128, activation='relu'))
model_T1.add(Dense(64, activation='relu'))
```
- Output Layer: The final layer has the output with 10 neurons with the probability of the input image belonging to each of the 10 possible classes. The activation function 'softmax' is used which converts the raw output values into probabilities. We used 'softmax' because it is a multi-class classification problem.
```
model_T1.add(Dense(10, activation='softmax'))
```





```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               100480
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 10)                650
=================================================================
Total params: 109,386
Trainable params: 109,386
Non-trainable params: 0
_________________________________________________________________
```

<u>Step 2:</u> Compiling the model

In this step, we define the Optimizer, Loss function and evaluation metrics.

For this, ‘categorical_crossentropy’ is taken as the loss function, which measures the dissimilarity between the predicted and actual distribution. The optimizer is used for updating the parameters of the model i.e. weights and biases so that the loss function is minimized. ‘Adam’ is used as an optimizer with the default settings.

<u>Step 3:</u> Training the model

Random state is set to 1, so that comparing evaluation metrics is possible

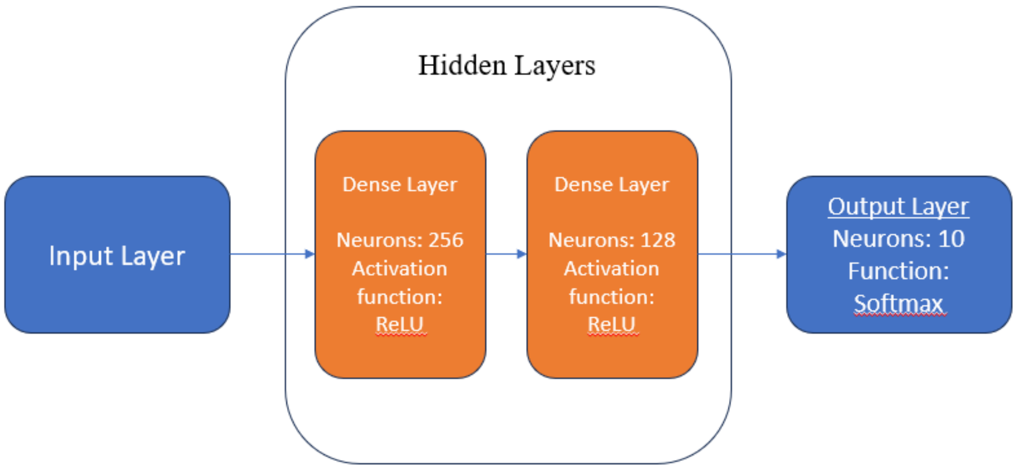
In this step, we train the model with 10 epochs initially. An epoch can described as one complete pass through the entire training dataset. As suggested by the assignment requirements, the model is trained without any separate validation set, but test set is used to monitor the training process.

<u>Step 4:</u> Evaluating the model with Testing Dataset

This shows that our model has an accuracy of 88.54%.

Step 5: Changing the model structure to get an improved performance

Changing the number of neurons: Changed the number of neurons of hidden layers to 256 and 128 respectively, with no increase in number of layers.


Hidden Layers
Input Layer
Dense Layer Neurons: 256 Activation function: ReLU
Dense Layer Neurons: 128 Activation function: ReLU
Output Layer Neurons: 10 Function: Softmax

This change showed an improved model accuracy of 88.70%.

Task 2: Build a neural network with the use of convolutional layers

Step 1: Defining the Model Architecture

- We start by defining a sequential model. A Sequential model is a linear stack of layers in which the layers are added one by one, through which the data flows sequentially.

```
model_T2 = Sequential()
```

- Convolutional Layer 1: Convolutional layers close to input learn low-level features. As this is the first Convolutional layer it will have low level features. I have taken this layer to have 32 filters (3*3 kernal), which means a total of 32*3*3 = 288 parameters. The activation function used is ‘ReLU’, which introduces non-linearity.

```
model_T2.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
```

- Pooling Layer 1: This layer is used to achieve down sampling, which means to create a lower resolution of an input while retaining the important features. This reduces the size of each feature map by a factor of 2. Max pooling is used, which takes the maximum value for each patch of the feature map.

```
model_T2.add(MaxPooling2D((2, 2)))
```

- Convolutional Layer 2: This layer has 64 filters with ‘ReLU’ as activation function.

```
model_T2.add(Conv2D(64, (3, 3), activation='relu'))
```

- Pooling Layer 2: This layer is again using Max pooling to down sample by a factor of 2.

```
model_T2.add(MaxPooling2D((2, 2)))
```

- Flatten Layer: After the above layers a flatten layer is added, which converts feature maps into a 1D vector, which is required before transitioning to the fully connected layers.

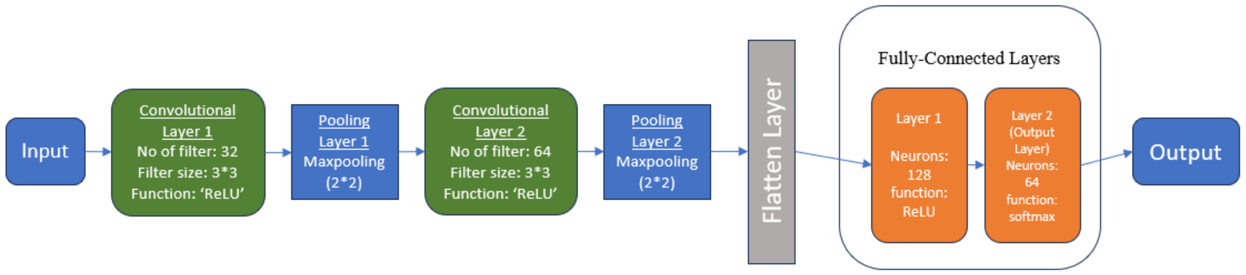
```
model_T2.add(Flatten())
```

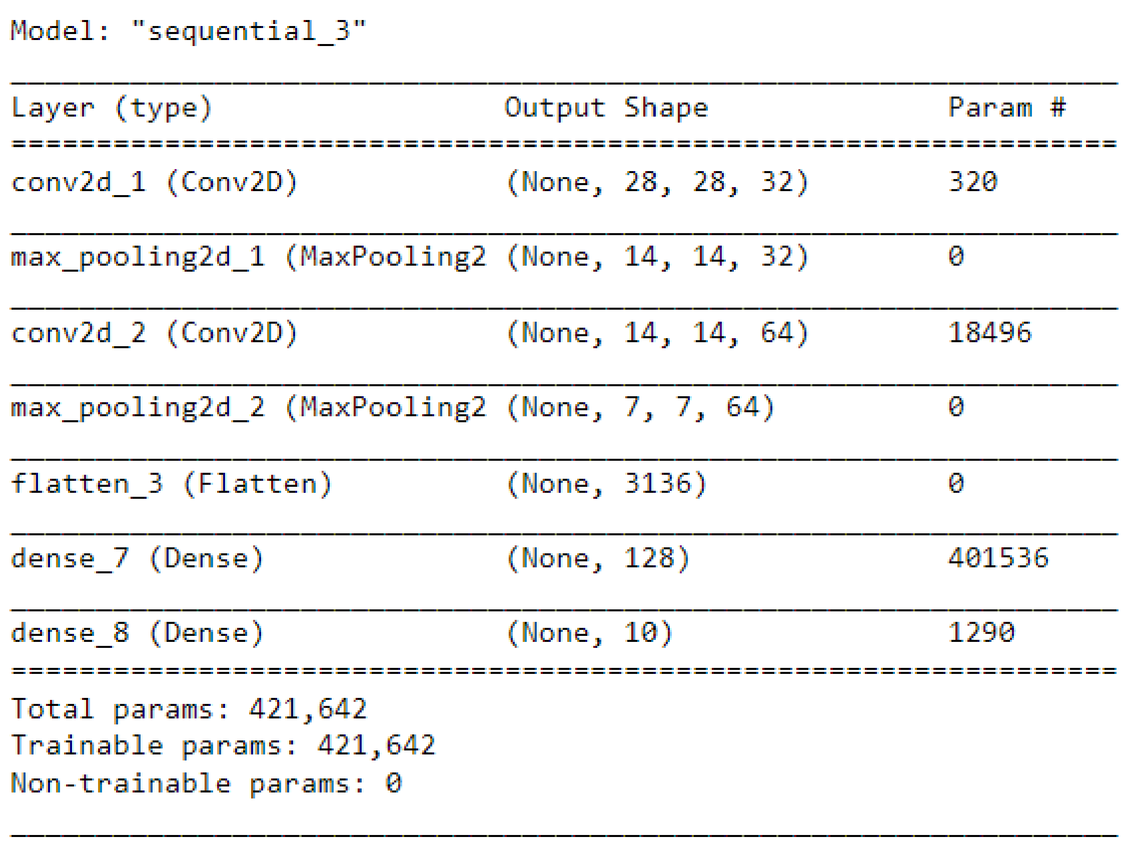
- Fully Connected Layer 1: A dense layer with ‘ReLU’ as a nonlinear activation function to create combinations of features for final predictions. This is made of 128 neurons.

```
model_T2.add(Dense(128, activation='relu'))
```

- Fully Connected Layer 2 (Output Layer): This is the final layer with 10 neurons corresponding to the 10 classes for prediction. The activation function used is ‘softmax’ which assigns probability to each class.

```
model_T2.add(Dense(10, activation='softmax'))
```





```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten_3 (Flatten)          (None, 3136)              0
_________________________________________________________________
dense_7 (Dense)              (None, 128)               401536
_________________________________________________________________
dense_8 (Dense)              (None, 10)                1290
=================================================================
Total params: 421,642
Trainable params: 421,642
Non-trainable params: 0
_________________________________________________________________
```

<u>Step 2:</u> Compiling the model

In this step, we define the Optimizer, Loss function and evaluation metrics.

For this, 'categorical_crossentropy' is taken as the loss function, which measures the dissimilarity between the predicted and actual distribution. The optimizer is used for updating the parameters of the model i.e. weights and biases so that the loss function is minimized.

'Adam' is used as an optimizer. Generally, Adam adapts the learning rates of each parameter individually, which can lead to faster convergence. But here we use it with a custom learning rate of 0.002.

<u>Step 3:</u> Training the model

Random state is set to 1, so that comparing evaluation metrics is possible

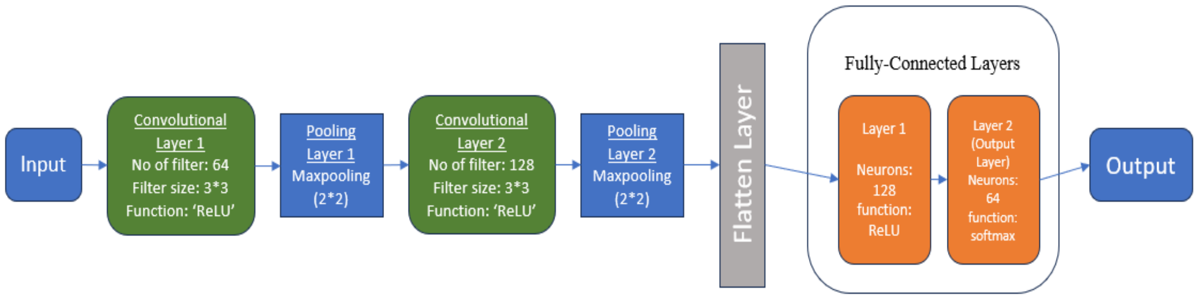
In this step, we train the model with 10 epochs initially. An epoch can described as one complete pass through the entire training dataset. As suggested by the assignment requirements, the model is trained without any separate validation set, but test set is used to monitor the training process.

<u>Step 4:</u> Evaluating the model with Testing Dataset

This shows that our model has an accuracy of 91.14%.

Step 5: Changing the model structure to get an improved performance

Changing the model structure by increasing the number of filters in the convolutional layer to 64 and 128 respectively.



This change showed an improved model accuracy of 91.58%.

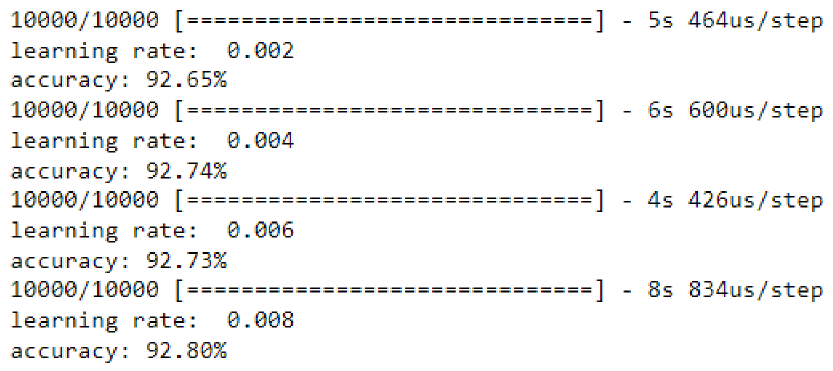
Task 3: Change the type of optimizer or learning rate that you applied in the previous tasks, and see how these changes can influence model performance

To perform this task, the optimizer is set to 'sgd' with a custom setting of changing learning rate of 0.002, 0.003, 0.004 and 0.005.

The performance results are the following:

Learning Rate	Accuracy
0.002	92.65%
0.004	92.74%
0.006	92.73%
0.008	92.80%

Based on this we can say that Learning rate of 0.008 provides the best accuracy.



```
10000/10000 [==============================] - 5s 464us/step
learning rate:  0.002
accuracy: 92.65%
10000/10000 [==============================] - 6s 600us/step
learning rate:  0.004
accuracy: 92.74%
10000/10000 [==============================] - 4s 426us/step
learning rate:  0.006
accuracy: 92.73%
10000/10000 [==============================] - 8s 834us/step
learning rate:  0.008
accuracy: 92.80%
```

Comparing the results

The following are ranked in descending order of their accuracy performance.

Task	Brief Model Description	Optimizer	Accuracy	Rank
Task 3	With 2 Convolutional Layers (64 and 128 filters)	Custom SGD	92.80% (lr = 0.002)	1
Task 2	With 2 Convolutional Layers (64 and 128 filters)	Custom Adam	91.58%	2
Task 2	With 2 Convolutional Layers (32 and 64 filters)	Custom Adam	91.14%	3
Task 1	Without Convolutional Layers (256 and 128 neurons)	Adam	88.70%	4
Task 1	Without Convolutional Layers (128 and 64 neurons)	Adam	88.54%	5

In the above table, only the best learning rate from Task 3 is considered.

Based on the whole analysis, the use of Convolutional Networks give a better performance results than without them. Also, the 'sgd' optimizer seems to be better fitted for this dataset.