

Digit Recognizer

Big Data & Artificial Intelligence for Business

(BUDT737)

Team Number: Group 32



UNIVERSITY OF
MARYLAND

Team Members:





Anshika Kapoor, George Puthean,
Kazi Tanveer Rehaman, Sai Vaishnav Vinjamuri

Table of Contents

I.	Account names, team name on Kaggle.com	3
II.	Methods & Concepts In Use	3
	Libraries used	3
1.	Reshape Train Data	4
2.	Normalisation	4
3.	Callbacks from Keras	4
4.	Model Building	5
5.	Model Architecture	7
6.	Model Fitting	7
7.	Reshaping Validation Data	8
III.	Results & Reports	8
1.	Rank on Kaggle	8
2.	Plots	9
3.	Final Accuracy and Loss (Validation Data)	11
IV.	What We Did	12
V.	What We Learnt	13
VI.	List of References	13

I. Account names, team name on Kaggle.com

Kaggle.com team name: **BUDT737_Team_32**

UID	Student Name	Kaggle.com URL	Signature
117444285	Anshika Kapoor	https://www.kaggle.com/anshikakapoor	
117447941	George Puthean	https://www.kaggle.com/georgeputhean	
118127002	Sai Vaishnav Vinjamuri	https://www.kaggle.com/saivaishnavvinjamuri	
118327082	Kazi Tanveer Rehman	https://www.kaggle.com/kazitanveer	

II. Methods & Concepts In Use

Libraries used

```
# import the necessary libraries
import math
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
import numpy as np
np.random.seed(2)
```

1. Reshape Train Data

The built-in `reshape()` function in Tensorflow allows tensors to be reshaped to match the input requirements and maintain consistency. In our code, we reshaped the training data into 28 x 28 matrices.

```
in_train = in_train.values.reshape(-1,28,28)
```

2. Normalisation

Normalisation is a scaling technique in which values are shifted and rescaled so that they end up ranging between 0 and 1. It is also known as Min-Max scaling. The expression for normalising a data is given below:

$$x_{scaled} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

$\min(x)$ and $\max(x)$ are the minimum and the maximum values of a feature respectively.

In our code, for both training and validation data, the **minimum value is 0** and the **maximum value is 255**.

```
in_train = in_train / 255.0  
in_valid = in_valid / 255.0
```

3. Callbacks from Keras

Callbacks are an important type of object in Keras that are designed to be able to monitor the performance in metrics at certain points in the training run and perform some action that might depend on those performance in metric values. We use a callback when we want to automate some tasks after every training/epoch that help us have controls over the training process.

```
# set call backs for the fit functions  
from keras.callbacks import LearningRateScheduler  
callback = LearningRateScheduler(lambda x: 1e-3 * 0.9 ** x)
```

4. Model Building

Keras LayersAPI : Layers are the basic building blocks of neural networks in Keras. Each layer receives input information, does some computation and finally outputs the transformed information. The output of one layer will flow into the next layer as its input.

Convolution Layers: Convolutional layers are the major building blocks used in convolutional neural networks. A convolution is the application of a filter to an input that results in an activation. Repeated application of the same filter to an input results in a map of activations called a feature map, indicating the locations and strength of a detected feature in an input. All convolution layers have certain properties listed below:

- Filters - refers to the number of filters to be applied in the convolution. It affects the dimension of the output shape.
- Kernel size - refers to the length of the convolution window.
- Strides - refers to the stride length of the convolution.
- Padding - refers to how padding needs to be done on the output of the convolution.

In our model we used the following convolution layers:

- Conv2D Layer - This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.
- MaxPooling2D Layer - This layer downsamples the input along its height and width by taking the maximum value over an input window for each channel of the input. The window is shifted by strides along each dimension.

Besides Convolution Layers, we also used the below 2 layers:

- Flatten - Flatten is used to flatten the input. For example, if flatten is applied to layer having input shape as (batch_size, 2, 2), then the output shape of the layer will be (batch_size, 4)
- Dense - Dense layer does the below operation on the input and returns the output.

$$\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$$

where,

input = the input data

kernel = weight of the data

dot = numpy dot product of all input and its corresponding weights

bias = a biased value used in machine learning to optimise the model

activation = the activation function.

Activation Function: An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network. The activation function is primarily used to modify the data in a non-linear way. This non-linearity allows us to spatially modify the representation of the data. The activation function allows us to change the way we see the data.

In our Model we used the following activation function:

- Relu - Applies the rectified linear unit activation function. The function returns 0 if it receives any negative input, but for any positive value x it returns that value back. So it can be written as $f(x)=\max(0,x)$.
- Softmax - Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. The elements of the output vector are in range (0, 1) and sum to 1.

```
# setup model and layers

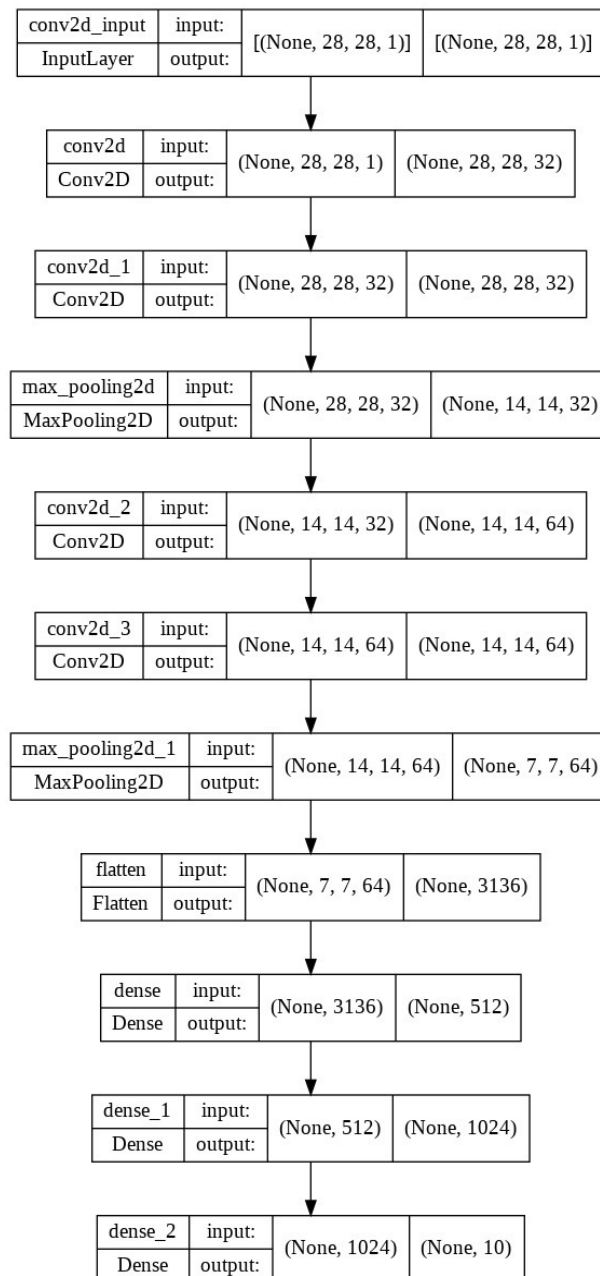
model = keras.Sequential(layers=[
    keras.layers.Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same', activation = 'relu', input_shape = (28,28,1)),
    keras.layers.Conv2D(filters = 32, kernel_size = (3,3),padding = 'Same', activation = 'relu', input_shape = (28,28,1)),
    keras.layers.MaxPool2D(strides=(2,2)),

    keras.layers.Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same', activation = 'relu', input_shape = (28,28,1)), # hidden size filters increase
    keras.layers.Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same', activation = 'relu', input_shape = (28,28,1)),
    keras.layers.MaxPool2D(strides=(2,2)),

    keras.layers.Flatten(),
    keras.layers.Dense(512, activation = "relu"),
    keras.layers.Dense(1024, activation = "relu"),
    keras.layers.Dense(10, activation="softmax")
])
adam = tf.keras.optimizers.Adam(learning_rate = 1e-4)
model.compile(optimizer=adam, loss='sparse_categorical_crossentropy', metrics=["accuracy"])
```

5. Model Architecture

The image of the Model architecture is given below which gives us an idea about what the inputs and outputs are at each layer.



6. Model Fitting

When we fit the model we change 3 parameters listed below:

- **batch_size** - The batch size is the number of samples that are passed to the network at once. If unspecified then batch size is set to 32.
- **epochs** - The number of epochs defines the number of times that the learning algorithm will work through the entire training dataset. One epoch means that

each sample in the training dataset has had an opportunity to update the internal model parameters. An epoch consists of one or more batches.

- **validation_split** - It defines the portion of the training set that we allot to the validation dataset and evaluate the performance of our model on that validation dataset each epoch.

```
# history function to store the epoch results
history = model.fit(in_train, out_train, batch_size=64, epochs=28, validation_split=0.15, callbacks=[annealer])
```

7. Reshaping Validation Data

We then reshaped the Validation dataset so that we can fit it to the model to evaluate the loss and accuracy.

```
# shape the validation data
in_valid = in_valid.values.reshape(-1,28,28)
```

III. Results & Reports

1. Rank on Kaggle

(Note: Rank as of 19:30 hrs, May 13 2022)

419	BUDT737_Team_32		0.99221	5	1s
-----	-----------------	---	---------	---	----



Your Best Entry!

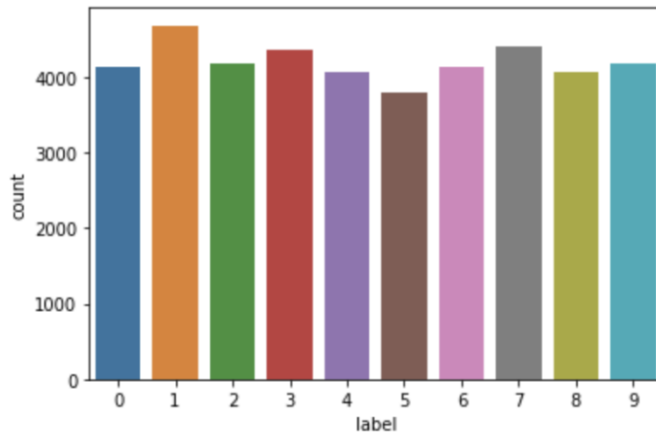
Your most recent submission scored 0.99221, which is the same as your previous score. Keep trying!

Thus, our test accuracy is **99.221%**.

2. Plots

- a. Bar Chart showing the count in each label (Training Data)

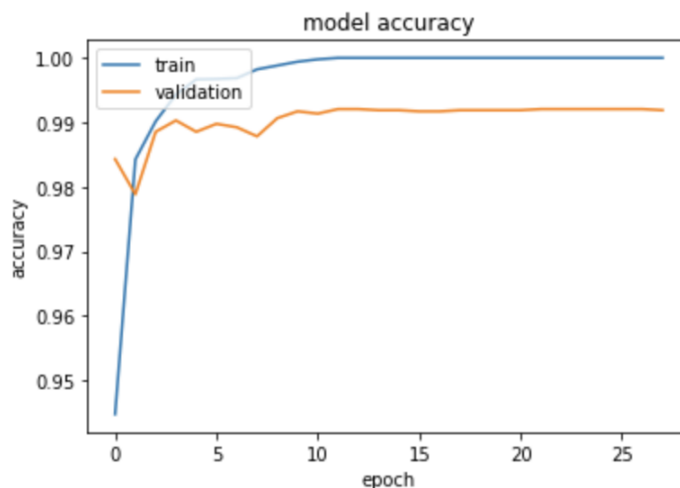
```
# count plot which shows the number of each label in training data
sns.countplot(x=train['label'],)
plt.show()
```



Label 1 has the highest count and label 5 has the least count.

- b. Accuracy vs Epoch graph

```
# Accuracy vs Epoch
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

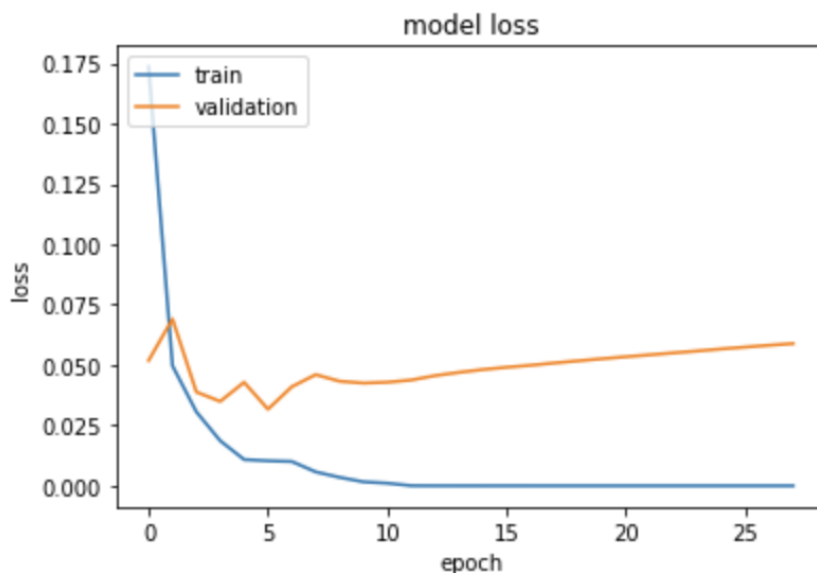


Points to note here:

- As expected the training accuracy is greater than the validation accuracy.
- The starting accuracy at 0th epoch is greater in case of validation than training.
- Validation accuracy varies a little until the 9th epoch, after that it behaves in a constant manner and takes the value of 99%.
- Training accuracy varies a little until the 11th epoch, after that it behaves in a constant manner and takes the value of 100%.

c. Loss vs Epoch graph

```
# Loss vs Epoch
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'validation'], loc='upper left')
plt.show()
```

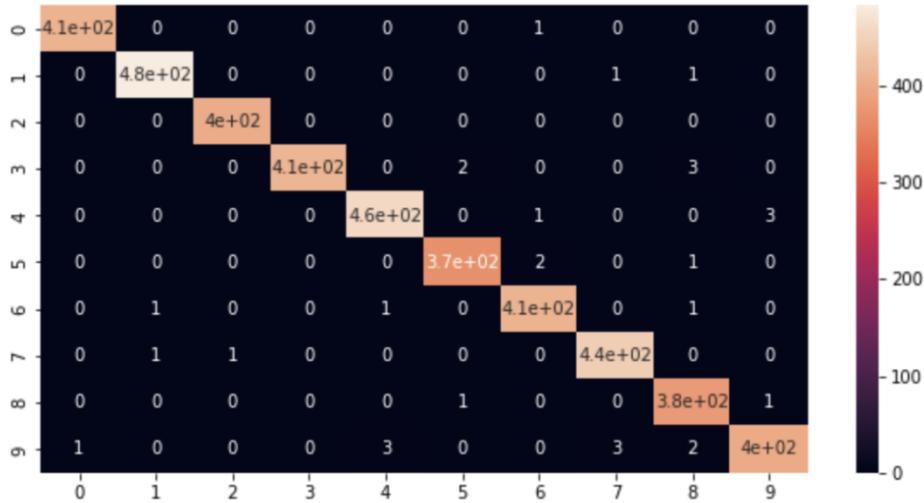


Points to note here:

- As expected the training loss is lower than the validation loss.
- The starting loss at 0th epoch is lower in case of validation than training.
- Validation loss varies a little until the 10th epoch, after that it increases throughout until the 28th epoch.
- Training loss varies a little until the 11th epoch, after that it decreases throughout until the 28th epoch.

d. Confusion Matrix for Validation data

```
# creating confusion matrix and then heatmap
conf_matrix = tf.math.confusion_matrix(labels=out_valid, predictions=preds_valid)
plt.rcParams["figure.figsize"] = (10,5)
sns.heatmap(conf_matrix,annot=True)
plt.show()
```



e. Training and Validation results sample

```
Epoch 10/28
503/503 [=====] - 3s 6ms/step - loss: 0.0016 - accuracy: 0.9994 - val_loss: 0.0425 - val_accuracy: 0.9917 - lr: 3.8742e-04
Epoch 11/28
503/503 [=====] - 3s 6ms/step - loss: 0.0011 - accuracy: 0.9998 - val_loss: 0.0429 - val_accuracy: 0.9914 - lr: 3.4868e-04
Epoch 12/28
503/503 [=====] - 3s 6ms/step - loss: 5.4304e-05 - accuracy: 1.0000 - val_loss: 0.0438 - val_accuracy: 0.9921 - lr: 3.1381e-04
Epoch 13/28
503/503 [=====] - 3s 6ms/step - loss: 1.4285e-05 - accuracy: 1.0000 - val_loss: 0.0456 - val_accuracy: 0.9921 - lr: 2.8243e-04
Epoch 14/28
503/503 [=====] - 3s 6ms/step - loss: 9.0601e-06 - accuracy: 1.0000 - val_loss: 0.0470 - val_accuracy: 0.9919 - lr: 2.5419e-04
Epoch 15/28
503/503 [=====] - 3s 6ms/step - loss: 6.2494e-06 - accuracy: 1.0000 - val_loss: 0.0481 - val_accuracy: 0.9919 - lr: 2.2877e-04
Epoch 16/28
503/503 [=====] - 3s 6ms/step - loss: 4.6519e-06 - accuracy: 1.0000 - val_loss: 0.0491 - val_accuracy: 0.9917 - lr: 2.0589e-04
Epoch 17/28
503/503 [=====] - 3s 6ms/step - loss: 3.5698e-06 - accuracy: 1.0000 - val_loss: 0.0499 - val_accuracy: 0.9917 - lr: 1.8530e-04
Epoch 18/28
503/503 [=====] - 3s 6ms/step - loss: 2.7755e-06 - accuracy: 1.0000 - val_loss: 0.0509 - val_accuracy: 0.9919 - lr: 1.6677e-04
Epoch 19/28
503/503 [=====] - 3s 6ms/step - loss: 2.2389e-06 - accuracy: 1.0000 - val_loss: 0.0517 - val_accuracy: 0.9919 - lr: 1.5009e-04
Epoch 20/28
503/503 [=====] - 3s 6ms/step - loss: 1.8067e-06 - accuracy: 1.0000 - val_loss: 0.0526 - val_accuracy: 0.9919 - lr: 1.3509e-04
Epoch 21/28
503/503 [=====] - 3s 6ms/step - loss: 1.4594e-06 - accuracy: 1.0000 - val_loss: 0.0534 - val_accuracy: 0.9919 - lr: 1.2158e-04
Epoch 22/28
503/503 [=====] - 3s 6ms/step - loss: 1.2048e-06 - accuracy: 1.0000 - val_loss: 0.0542 - val_accuracy: 0.9921 - lr: 1.0942e-04
Epoch 23/28
503/503 [=====] - 3s 6ms/step - loss: 9.9814e-07 - accuracy: 1.0000 - val_loss: 0.0550 - val_accuracy: 0.9921 - lr: 9.8477e-05
Epoch 24/28
503/503 [=====] - 3s 6ms/step - loss: 8.2934e-07 - accuracy: 1.0000 - val_loss: 0.0558 - val_accuracy: 0.9921 - lr: 8.8629e-05
Epoch 25/28
503/503 [=====] - 3s 6ms/step - loss: 6.8719e-07 - accuracy: 1.0000 - val_loss: 0.0567 - val_accuracy: 0.9921 - lr: 7.9766e-05
Epoch 26/28
503/503 [=====] - 3s 6ms/step - loss: 5.7291e-07 - accuracy: 1.0000 - val_loss: 0.0574 - val_accuracy: 0.9921 - lr: 7.1790e-05
Epoch 27/28
503/503 [=====] - 3s 6ms/step - loss: 4.7989e-07 - accuracy: 1.0000 - val_loss: 0.0582 - val_accuracy: 0.9921 - lr: 6.4611e-05
Epoch 28/28
503/503 [=====] - 3s 6ms/step - loss: 4.0227e-07 - accuracy: 1.0000 - val_loss: 0.0589 - val_accuracy: 0.9919 - lr: 5.8150e-05
```

3. Final Accuracy and Loss (Validation Data)

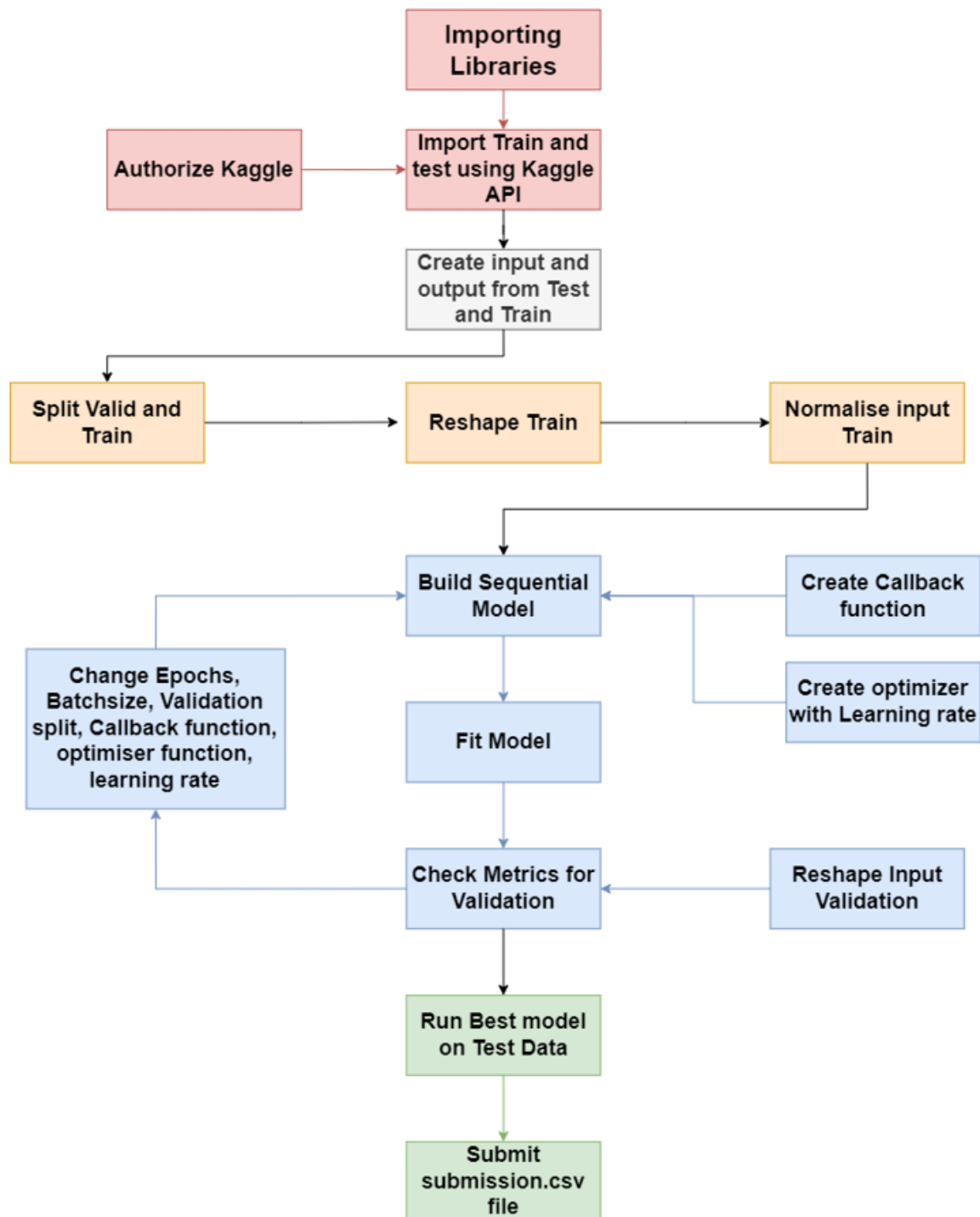
```
final_loss,final_accuracy = model.evaluate(in_valid,out_valid)
```

```
132/132 [=====] - 0s 3ms/step - loss: 0.0627 - accuracy: 0.9926
```

Accuracy is **99.26%** and Loss is **6.27%**.

IV. What We Did

Flowchart:



V. What We Learnt

- The dataset MNIST is effectively solved, from the dataset, we demonstrated it on the basis for learning and practised ways to develop the model, evaluated it, and finally implemented convolutional deep learning neural networks for image classification from scratch
- We also explored many improvements for the model, as well as how to save the model and then load it in order to make prediction forecasting on new data.
- We also learnt ways to develop the harness of the test in order to develop a robust and evaluate accurate prediction for a model and exhibit a baseline performance for digit recognizer which is our classification task.
- We learnt how to change optimizer functions and add new layers like Maxpool2D, Batch Normalisation etc. We understood the use of Dropout layers as well.
- We also learnt to explore extensions to a baseline convolutional neural network model to improve and increase the model capacity like the Callback and Learning Rate
- Moreover, we also developed a finalised model that could evaluate the performance of our final model, & implement it to make predictions on new images.
- Implemented 28 epochs of training for which we managed to achieve 99.226% accuracy on the validation set.
- Generally Test Accuracy is lower than validation. Here it is 99.221% less than validation
- Hence, as a conclusion we were able to develop a Baseline Model and also an Improved Model and then we went on to Finalise the Model and Make Predictions.

VI. List of References

- www.tensorflow.org
- keras.io
- www.kaggle.com
- www.analyticsvidhya.com
- <https://towardsdatascience.com/>
- www.tutorialspoint.com
- stackoverflow.com
- machinelearningmastery.com