HANDWRITTEN CHARACTER RECOGNITION USING NEURAL NETWORKS

A PROJECT REPORT

Submitted by

Varun Shukla (20BAI10292)

Deepika S. Srivastava (20BAI10319)

Gargi Choudhary (20BAI10358)

Janvi Bhalani (20BAI10368)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

VIT BHOPAL UNIVERSITY

KOTHRIKALAN, SEHORE

MADHYA PRADESH - 466114

NOV 2021

VIT BHOPAL UNIVERSITY, KOTHRIKALAN, SEHORE

MADHYA PRADESH – 466114

BONAFIDE CERTIFICATE

Certified that this project report titled **“HANDWRITTEN CHARACTER RECOGNITION USING NEURAL NETWORKS”** is the bonafide work of

“**Varun Shukla (20BAI10292), Deepika S. Srivastava (20BAI10319), Gargi Choudhary (20BAI10358), Janvi Bhalani (20BAI10368)”** who carried out the project work under my supervision. Certified further that to the best of my knowledge the work reported at this time does not form part of any other project/research work based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

PROGRAM CHAIR	**PROJECT GUIDE**
Dr S Sountharrajan	Dr. D. Lakshmi
Program Chair, B.Tech CSE Spl in AIML	Sr. Associate Professor, SCSE
VIT BHOPAL UNIVERSITY	VIT BHOPAL UNIVERSITY

The Project Exhibition I Examination is held on _______________

ACKNOWLEDGEMENT

First and foremost, I would like to thank the Lord Almighty for His presence and immense blessings throughout the project work.

I wish to express my heartfelt gratitude to Dr. S. Poonkuntran, Professor and Dean, School of Computer Science and Engineering for much of his valuable support encouragement in carrying out this work.

I would like to thank my internal guide Dr. D Lakshmi, for continually guiding and actively participating in my project, giving valuable suggestions to complete the project work.

I would like to thank all the technical and teaching staff of the School of Computer Science and Engineering, who extended directly or indirectly all support.

Last, but not least, I am deeply indebted to my parents who have been the greatest support while I worked day and night for the project to make it a success.

LIST OF FIGURES AND GRAPHS

LIST OF FIGURES

LIST OF GRAPHS

ABSTRACT

Handwriting recognition has been under research for the past many decades. Improvements have been done in this field ranging from IBM 1287 optical reader, hand-printed numeric data for computers to using Deep Learning a field of Artificial Intelligence.

Handwriting recognition is a technique used to interpret handwritten input and convert itinto digital text using machine learning tools. This project seeks to classify an individual handwritten text so that handwritten text can be translated to a digital form. To accomplish this task, we classify words directly using Convolutional Neural Network (CNN) with various architectures to train a model that can accurately classify words. The process of Image Contouring has been used to extract every letter from the image and then passit into our model to get a prediction of the letters.

TABLE OF CONTENT

CHAPTER 1

PROJECT DESCRIPTION AND OUTLINE

1.1 INTRODUCTION

Optical Character Recognition or OCR as it is widely known is a system that converts input text into machine-encoded format and uses the principle of pattern recognition. People still prefer taking notes traditionally i.e., using pen and paper. However sometimes it becomes difficult to understand the handwriting because every individual has a different way of writing the alphabet and numbers. When the notes are pass onto someone else maybebecause of the lack of understanding of the handwriting, some important information get lost, or the entire message might remain undelivered.

Handwritten OCR is a sub-field of OCR. It is probably the most interesting and demanding application in the area of pattern recognition. Some of the major factors as to why this is an important research field is its numerous applications in the field of health care, education, legal and finance. Automated Number plate recognition, postal mail sorting and automatic check processing for banks are some of real-life examples where image recognition is used. The handwritten recognition systems can be inspired by biological neural networks. This helps humans to learn and model complex and non-linear relationships. They can be developed from the artificial neural networks which also allows the human brain to recognize the different handwritten objects such as digits, letters and characters.

In this project, we shall use different machine learning techniques such as Neural Networks (Convolutional Neural Networks) to successfully recognize handwritten characters of English language and digits using their scanned images. This project is an attempt to design an OCR model which identifies both alphabets and digits together.

1.2 MOTIVATION FOR THE WORK

Daily, we have seen how medical companies retrieve text from it. Doing thismanually can take a lot of time and can be a cumbersome process as many discrepancies may exist in different handwritings. In order to overcome difficulties like this we would like tocreate a model that would automate the task and reduce the manual labor.

A similar model can also work in banks, thus reducing the manual labor on check processing.

1.3 INTRODUCTION TO THE MODEL

- **What are neural networks?**

  Handwriting Recognition Systems are more efficient when they are based on neural networks. The neural network architecture refers to the elements that are connected to make a network that is used for handwriting recognition. The inspiration behind the working of neural networks is the human brain. Its work is based on the idea of how neurons pass signals around the human brain to process input into an output.

- **The use of CNN:**

  Convolutional Neural Network is a part of deep learning and is a very efficient and effective way of recognizing handwriting.

  The underlying architecture of a CNN model includes several layers, the first being the input layer and the last being the output layer. In between lies the convolutional layer followed by the pooling layers and convolutional layers.

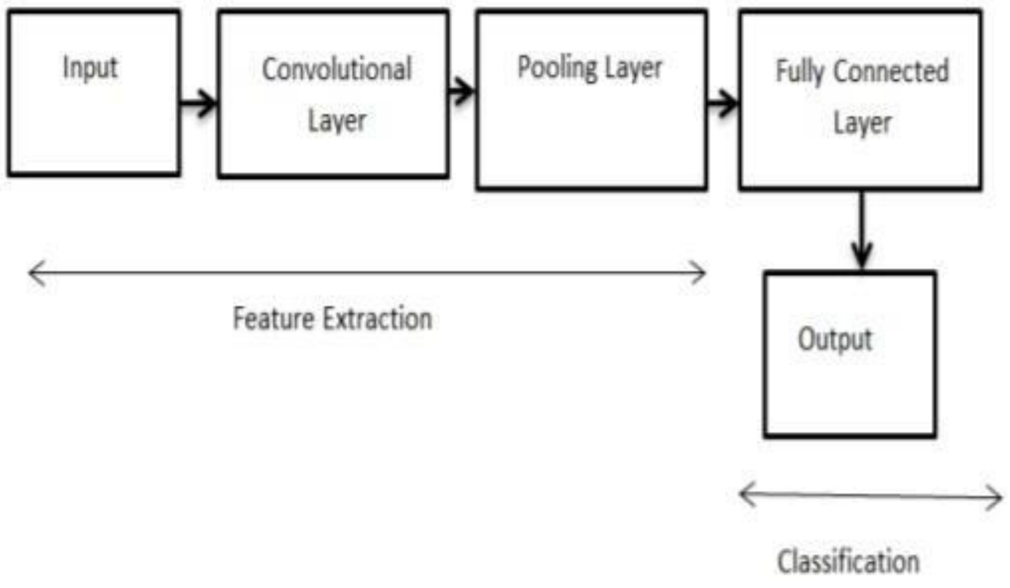


Fig. 1 (a)

- **The use of ANN**

  ANN is a model in machine learning which consists of many artificial neurons connected to each other. The structure resembles that of axons in the human brain. This type of architecture consists of input, hidden and output layers. Thenetworks are composed of many interconnected neurons working in unison to achieve a specific goal.

- **CapsNet or Capsule Net**

  A capsule is a nested set of neural layers. In a regular neural network, we keep onadding more layers. In CapsNet one would add more layers inside a single layer or in other words nest a neural layer inside another neural layer.

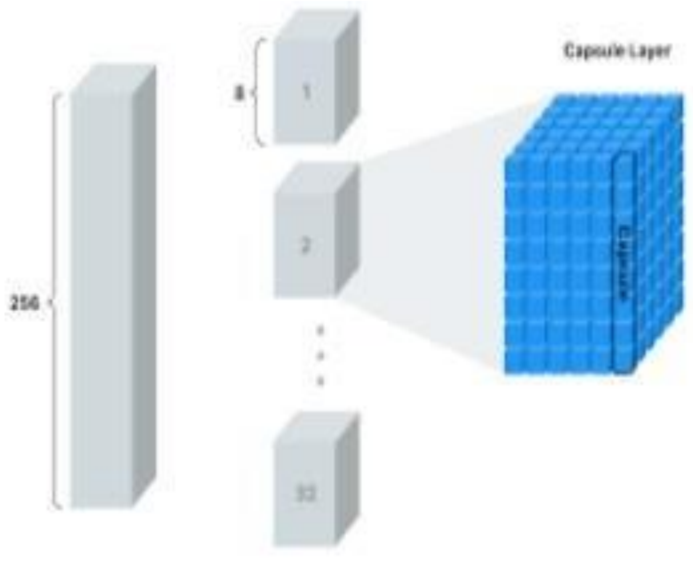


Fig 1(b)

1.4 PROBLEM STATEMENT

To use machine learning algorithms to accurately predict text from scanned handwritten text images. To do this, we need to assume that all images containing the same characters have the same characteristics, and we can conclude that images with these characteristics contain this alphabet. However, this hypothesis is ideal and does not always hold true in practice.

1.5 OBJECTIVE

- To design a system that recognizes handwritten characters using neural networks.
- To overcome the issues of inaccuracy presented in many handwritten recognition systems by developing a more efficient system that would use efficient technology for recognizing characters and words from the images.
- To honor the usefulness of neural network technology in creating such a model that uses the EMNIST dataset in order to extract a set of words/digits/characters from an image given as the input.
- Introducing the use of CapsNet and comparing the accuracy produced by the two different models.

CHAPTER - 2

RELATED WORK INVESTIGATION

2.1 OUTLINE

One of the earliest OCR systems were developed in the 1940s. With the advancement of technology, the system became capable of handling both printed and handwritten texts.

In 1965, IBM launched its “IBM 1287” . It was the first ever optical reader. It was able detect handwritten numbers. In the 1970s, the researchers worked upon improving this technology. The during 1980s to 2000s, the OCR system was deployed in the educational institutes in order to read the characters stamped on the metallic bars. In 2000s, binarization was introduced which was used to preserve the historical texts. The major breakthrough in the field of OCR was in the current decade when Machine learning approaches were used for the system. Several Machine Learning algorithms such as Support Vector Machines (SVM) , k-Nearest Neighbor (kNN), Random Forests (RF), Decision Trees (DT) , Neural Networks (NN) , etc came into picture.

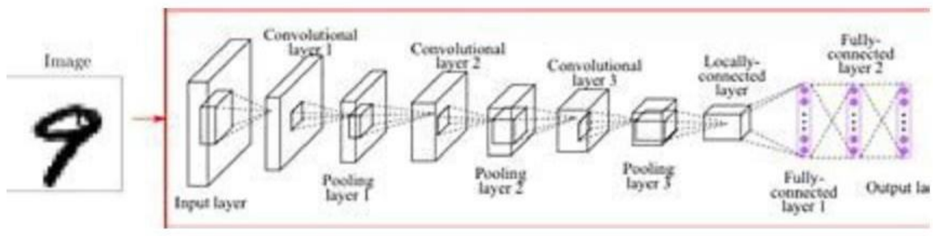
2.2 CORE AREA OF THE PROJECT

This project revolves around the optical character recognition using Convolutional Neural Networks (CNN). We have also tried to use Capsule Network Algorithm in the project.

CONVULATIONAL NEURAL NETWORK

CNN is an artificial neural network mostly used for image recognition. It is a class of deep neural networks.The Convolutional neural networks are regularized versions of multilayer perceptron (MLP). They were developed based on the working of the neurons of the animal visual cortex.

CNN are composed of multiple layers of artificial neurons. Each layer extracts features and pass it on to the next layer which detects more complex features. The final layer will give us an output which would have extracted all the features from the the input image or data.



CNN Architecture

Fig. 2 (a)

CNN consists of a lot of layers. These layers when used repeatedly, lead to a formation of a Deep Neural Network. The fundamental types of layers used to build a CNN are:

1. Input

This layer holds the uncooked pixel values of photograph and convert it to grayscale pics using 28x28 matrix of pixels.

2. Convolutional Layer

This layer gets the effects of the neuron layer that is linked to the enter regions. The wide variety of filters to be used in this layer is described here. Each filter may additionally be a 5x5 window that slider over the input records and receives the pixel with the most intensity as the output.

3. Rectified Linear Unit (ReLU) Layer

This layer applies an thing smart activation function on the picture records and makes use of again propagation techniques. ReLU function is utilized in order to preserve the equal values of the pixels and not being changed by means of the returned propagation.

4. Pooling Layer

Down-sampling operation along the spatial dimensions (width, height), resulting in volume is utilized in this layer.

5. Fully Connected Layer

This layer is used to compute the score instructions that potential which class has the maximum score corresponding to the enter digits. The category label with the largest likelihood is chosen as the ultimate classification from the network and proven in the output.

CAPSULE NETWORK

Capsule Networks (CapsNet) are the networks that are able to fetch spatial information and more important features so as to overcome the loss of information that is seen in pooling operations. Let us see what is the difference between a capsule and a neuron. Capsule gives us a vector as an output that has a direction. For example, if you are changing the orientation of the image then the vector will also get moved in that same direction whereas the output of a neuron is a scalar quantity that does not tell anything about the direction.

There are 4 main components that are present in the CapsNet that are listed below:

1. Matrix Multiplication

It is applied to the image that is given as an input to the network to convert into vector values to understand the spatial part.

2. Scalar Weighting of the Input

It computes which higher-level capsule should receive the current capsule output.

3. Dynamic routing algorithm

It permits these different components to transfer information amongst each other. Higher-level capsules get the input from the lower level. This is a repetitive process.

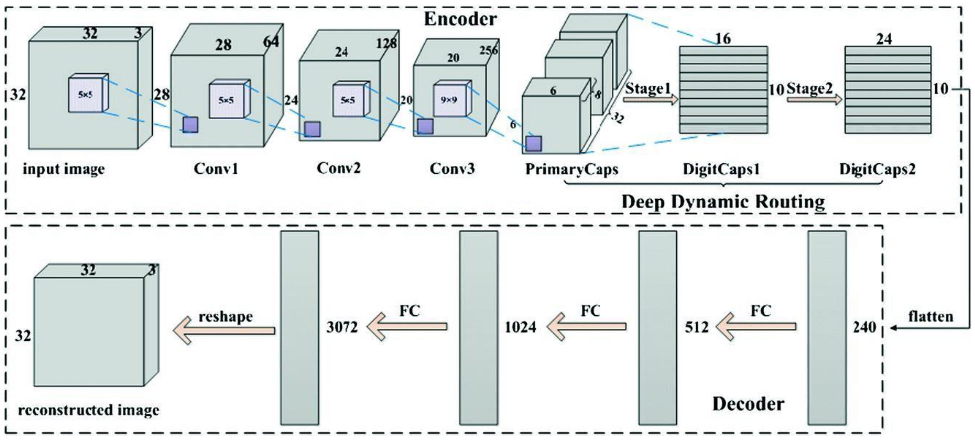


Fig 2(b)

4. Squashing Function

It is the last component that condenses the information. The squashing function takes all the information and converts it into a vector that is less than or equal to 1 also maintaining the direction of the vector.

The architecture consists of 6 layers, first 3 layers are considered to be encoders where the task is to convert the input image into a vector and after that, the last 3 layers are called decoders are used to reconstruct the image using that.

2.3 EXISTING METHODS FOR OCR

KERNEL METHODS

In Machine Learning, a “kernel” is usually used to refer to the kernel trick, a method of using a linear classifier to solve a non-linear problem. Kernel based algorithms such as Support Vector Machines (SVMs) , Kernel Fischer Discriminant Analysis (KFDA) , etc. have shown practical relevance for classification problems. SVM is a supervised learning algorithm. In SVM, kernel performs mapping of feature vectors into higher dimensions in order to find a hyperplane. Some researchers still believe that SVM is better than many other recognition techniques.

STATISTICAL METHODS

There are two types of Statistical methods, parametric and non-parametric. Parametric methods can work even on a very small sample. Parametric classifiers have fixed number of parameters. The training data can be modeled by a probability distribution function as it has fixed parameters. Logistic Regression , Linear Discriminant Analysis (LDA) , Hidden Markov Model (HMM) are few types of parametric methods. HMM was used for speech recognition even before it was used for character recognition and it is believed that HMM provides better results even when the number of lexicons are limited.

On the other hand, non-parametric methods include k-Nearest Neighbour (KNN) and Decision Trees (DT) and the non-parametric classifiers are more flexible in learning concepts.

KNN is the most used non-parametric method and is an unsupervised learning algorithm . It assumes the similarity between the new cases and available cases and put the new case in the category that is most similar to the available categories. According to researchers, kNN achieves a pretty good result in the optical recognition.

TEMPLATE MATCHING TECHNIQUES

This refers to a method where images (small part of it) is matched with a pre-existing template. It is done by comparing each of the pixels of the source image one at a time to the template image. It works on a sliding window approach. Not enough work has been done using this technique hence it is as certain to conclude the accuracy given by it.

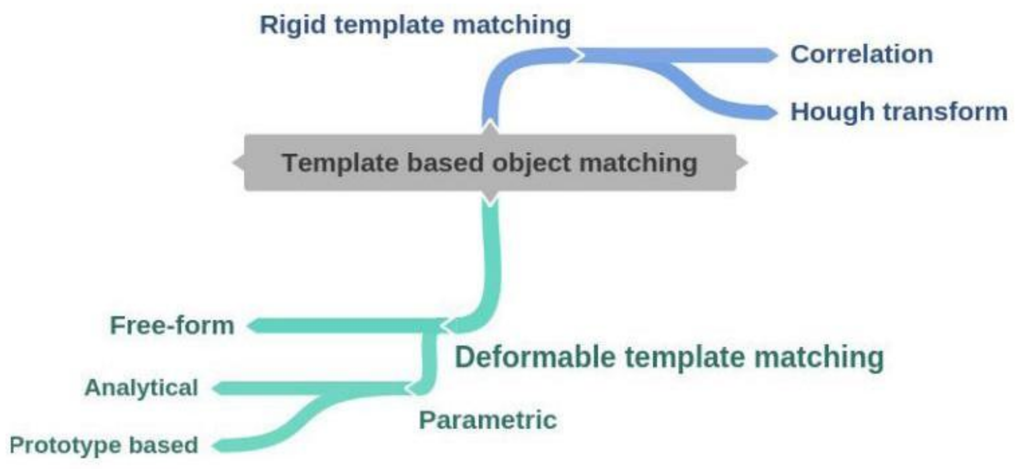
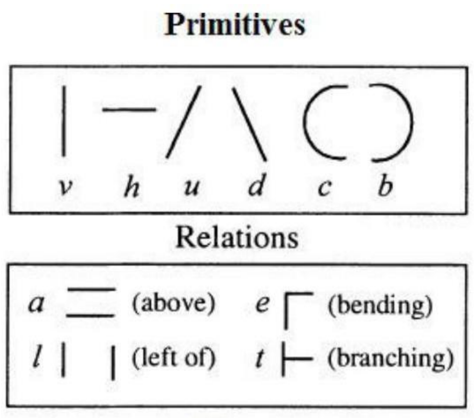


Fig. 2 (c)

STRUCTURAL PATTERN RECOGNITION

This method was used before the popularization of kernel methods and neural networks. Structural pattern recognition aims to classify objects based on a relationship between its pattern structures which are edges, contours, connected component geometry, etc.



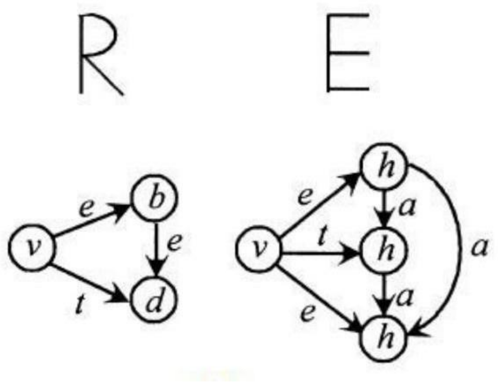


Fig. 2.(d) - (a) Fig. 2 . (d) - (b)

(a) - Primitive and relations

(b) - Directed Graph for capital letter R and E

2.4 PROS AND CONS OF THE METHODS

PROS -

Since many techniques were used for handwritten character recognition we have got a wider idea about the accuracies given by those different techniques. Due to this the debate on which technique is the most suitable seems to have come to an end. Researchers all over the world who have been working on OCR systems have unanimously chosen CNN as the most suitable technique for OCR.

CONS -

Although CNN ought to give better results than all the other techniques for the CNN system, still on some datasets some other techniques provide better accuracy than CNN. Such as Linear Regression provides a better accuracy when we have a small sample whereas CNN might show a great deviation if used on a small sample.

2.5 OBSERVATIONS

With the advancement of technology, the OCR system has seen manifolds. A large number of algorithms and techniques have been used in order to achieve the maximum accuracy. Howsoever, the size of datasets play an important role in determining the accuracy of the models. But if we have to conclude with one technique then a unanimously selected one is the CNN. The CNN model extracts the most complex features from the input and gives the highest accuracy in the outputs. Hence CNN is mostly used in OCR these days. Although a new technique which is Capsule Network is into the picture which is supposed to yield better results than the CNN.

CHAPTER 3

REQUIREMENT ARTIFACTS

3.1 INTRODUCTION

To develop a handwritten character recognition model, we require certain hardware and software components. These components help is acquiring the textual image and convert them into a machine encoded language.

The code for this particular model can be executed on different IDE's like Python 3.7, Anaconda 3, or Google Collab. These IDE's have in-built libraries and requires no prior set-up, which helps the users.

The CNN model has proved to be efficient conversion of image text to machine encoded language. This is because of the different functions it is associated with. In this model, activation functions, optimization functions , loss functions and CapsNet has proved to be effective.

Activation functions are used to learn and approximate any kind of continuous and complex relationship between variables of the network. The most commonly used activation functions are ReLU, Softmax, tanH and sigmoid functions.

Optimizers are algorithms or methods used to change the attributes of the neural network in order to reduce the losses. Attributes may include learning rates and weights.

CapsNet is a machine learning system that closely mimics the biological neural organization. It is used to improve image classification, object detection and object segmentation.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

SOFTWARE:

- **python 3.7**
  Python is broadly utilized universally and is a high-level programming language. It enables one to express ideas in fewer lines of code. Python is a programming language that gives you a chance to work rapidly and coordinate frameworks more effectively.

- **Anaconda3**
  Anaconda is used as IDE all through the implementation of this project. Anaconda is a free and open-source appropriation of the Python and R programming for logical figuring such as statistics science, AI applications, instruction of large-scale information, prescient investigation, etc.

- **Google Collab**
  Google collab allows developers to write and execute Python codes through their browsers. It is a hosted Jupyter notebook that requires no setup.
  There are many advantages of using google collab in this model.
  - ✓ Sharing
  - ✓ Versioning
  - ✓ Code snippets
  - ✓ Price
  - ✓ Forms for non-technical users
  - ✓ Pre-installed libraries
  - ✓ Free GPU and TPU use

HARDWARE:

- Graphical Processor Unit (GPU)
- Memory
- Storage
- Field Programmable Gate Array (FPGA)
- Scanning tools

3.3 DATA REQUIREMENT

In order to train our model, we use the MNIST and EMNIST datasets.

The MNIST dataset is one of the most common datasets used for image classification and accessible from many different sources. It provides an easy-to-use CSV format.

Fig. 3.(a)

The dataset consists of two files:

1. mnist_train.csv
2. mnist_test.csv

The mnist_train.csv file contains about 60000 training examples and labels while the mnist_test.csv contains 10000 test examples and labels.

Each row has 785 values, where the first value is the label (0 to 9) and the remaining 784 values are the pixel values (0 to 255).

The EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19. It is converted to a 28x28 pixel image format and dataset structure that directly matches the MNIST dataset.

3.4 FUNCTIONS REQUIREMENT

ACTIVATION FUNCTION

Activation functions are an important part of any neural network. They are fundamentally used for determining the output of deep learning models, its accuracy and performance efficiency of the training model that can design or divide a huge scale neural network.

Activation function must be efficient, and it should reduce the computation time because the neural network sometimes trained on millions of data points.

OPTIMIZATION FUNCTIONS

Optimization functions are used during backward propagation to update the attributes of neural networks. It also ensures that only that amount of data is provided to the network as much necessary.

The different types of optimization functions are: Adagrad, Adadelta, RmsProp, Adam, etc.

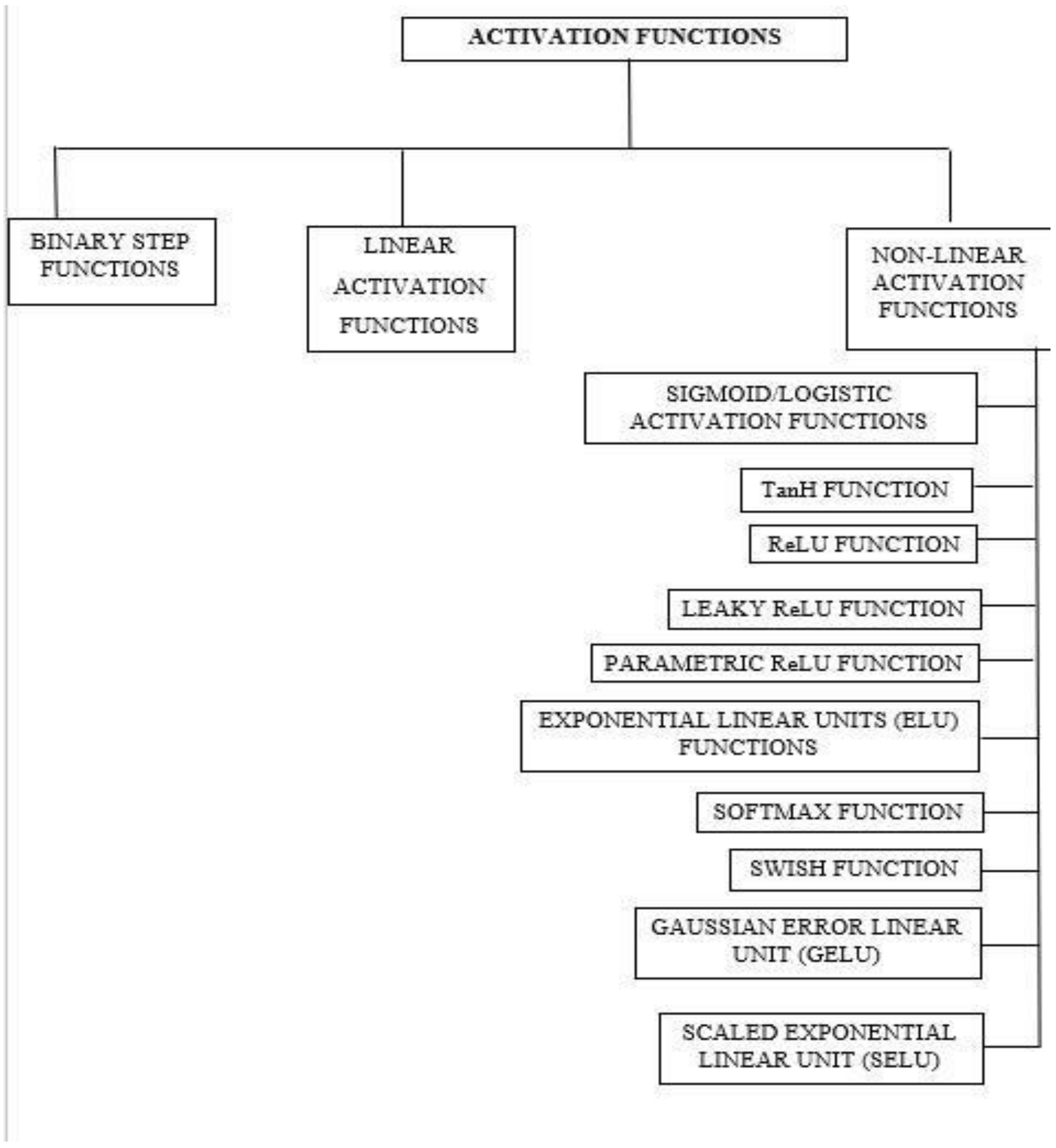

ACTIVATION FUNCTIONS
BINARY STEP FUNCTIONS
LINEAR ACTIVATION FUNCTIONS
NON-LINEAR ACTIVATION FUNCTIONS
SIGMOID/LOGISTIC ACTIVATION FUNCTIONS
TanH FUNCTION
ReLU FUNCTION
LEAKY ReLU FUNCTION
PARAMETRIC ReLU FUNCTION
EXPONENTIAL LINEAR UNITS (ELU) FUNCTIONS
SOFTMAX FUNCTION
SWISH FUNCTION
GAUSSIAN ERROR LINEAR UNIT (GELU)
SCALED EXPONENTIAL LINEAR UNIT (SELU)

Fig. 3.(b)

CAPSNET

CapsNet has more layers inside a single layer i.e., nest of neural networks inside another layer. It performs inverse graphics. CapsNet Is composed of capsules that in turn comprises of a group of neurons in a layer which performs internal computations. Internal computations are used to predict the presence and the instantiation parameters of a particular feature at a given location. The architecture of CapsNet is made up of two points: the encoder and the decoder.

LOSS FUNCTION

The loss function in a neural network quantifies the difference between the expected outcome and the outcome produced by our model.

3.5 PERFORMANCE REQUIREMENT

The performance of the model can be tested based on certain indications of correctness and functionality.

- To test individual units of the system, **unit testing** is used. It focusses on image acquisition, segmentation, feature extraction, digitization, etc. It helps in identifying bugs in the code, thus making it easy to fix the code.
- When individual units of the system are combined and tested as a group, it leads to **integration testing**. The main aim of such testing is to expose faults in the interaction of the integrated units.
- **Validation testing** determines whether the developmental process meets the specified requirements. It helps identifying defects in the system.
- In order to ensure that the systems graphical user interface meets the specified specifications, and it is user friendly, we perform **GUI testing.**

3.6 SUMMARY

For any model to run smoothly it requires certain software's and hardware's. For this particular model, we require Python 3.7, Anaconda 3 and Google Collab as the software's and Graphical Processor Unit (GPU), Memory, Storage, Field Programmable Gate Array (FPGA) and scanning tools as the main hardware components.

In order to train any model, we require data. This can be obtained from the MNIST and EMNIST datasets.

Our model works based on Convolutional Neural Networks and CapsNet, thus, using certain activation, optimization, and loss functions.

To test the performance of the model, we can perform a series of test known as unit testing, integration testing, validation testing and GUI testing. These tests have a common motive of exposing faults, so that users can rectify them accordingly.

CHAPTER - 4

DESIGN METHODOLOGY AND ITS NOVELTY

4.1 METHODOLOGY AND GOAL

This project revolves around different algorithms under Neural Networks.

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.

The algorithms used are Convolutional Neural Networks(CNN) and Capsule Network(CapsNet).

CNN is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data.

CapsNet are the networks that are able to fetch spatial information and more important features so as to overcome the loss of information that is seen in pooling operations. Capsule gives us a vector as an output that has a direction. For example, if you are changing the orientation of the image then the vector will also get moved in that same direction whereas the output of a neuron is a scalar quantity that does not tell anything about the direction.

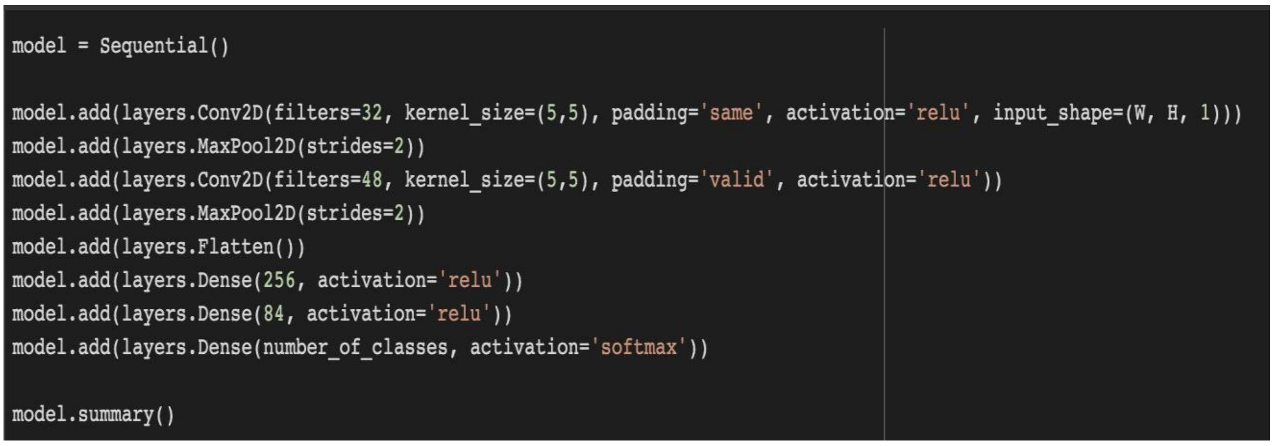
The primary goal of this project is to create a model which recognizes both digits and characters simultaneously. We have trained the model using both Convolutional Neural Network and Capsule Network and calculated the accuracy of the model in both the cases.

4.2 NOVELTY OF THE PROJECT

Based on our research on the topic image recognition specifically character and digits recognition, we found out that though there are models for both characters and digits recognition, but they exist individually i.e., if we feed some data which has both characters and numbers it would probably generate an error, or we'll have to run two different models. So, to ease this thing we plan to combine these two models and make one Character and Digits recognition ML Model.

4.3 FUNTIONAL MODULES DESIGN AND ANALYSIS

4.3.1. CNN Model



```
model = Sequential()

model.add(layers.Conv2D(filters=32, kernel_size=(5,5), padding='same', activation='relu', input_shape=(W, H, 1)))
model.add(layers.MaxPool2D(strides=2))
model.add(layers.Conv2D(filters=48, kernel_size=(5,5), padding='valid', activation='relu'))
model.add(layers.MaxPool2D(strides=2))
model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(84, activation='relu'))
model.add(layers.Dense(number_of_classes, activation='softmax'))

model.summary()
```

Fig. 4.(a)

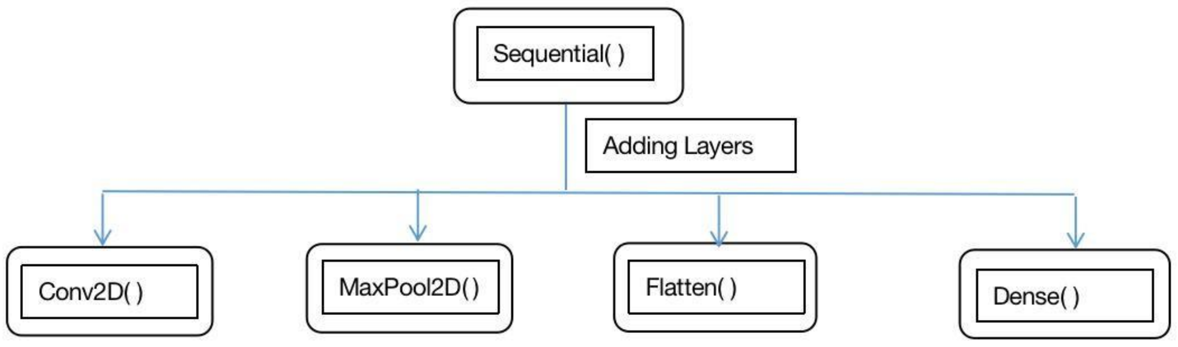


Fig. 4.(b)

A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.

We add layers like Conv2d, MaxPool2D, Flatten and Dense.

Let us understand what these layers do :

1. **Conv2D( )** - this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

Mandatory Conv2D parameter is the numbers of filters that convolutional layers will learn from.

It is an integer value and also determines the number of output filters in the convolution.

kernel_size

This parameter determines the dimensions of the kernel. Common dimensions include 1×1, 3×3, 5×5, and 7×7 which can be passed as (1, 1), (3, 3), (5, 5), or (7, 7) tuples.

It is an integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window.

This parameter must be an odd integer.

padding

The padding parameter of the Keras Conv2D class can take one of two values: 'valid' or 'same'.

Setting the value to "valid" parameter means that the input volume is not zero-padded and the spatial dimensions are allowed to reduce via the natural application of convolution.

Activation

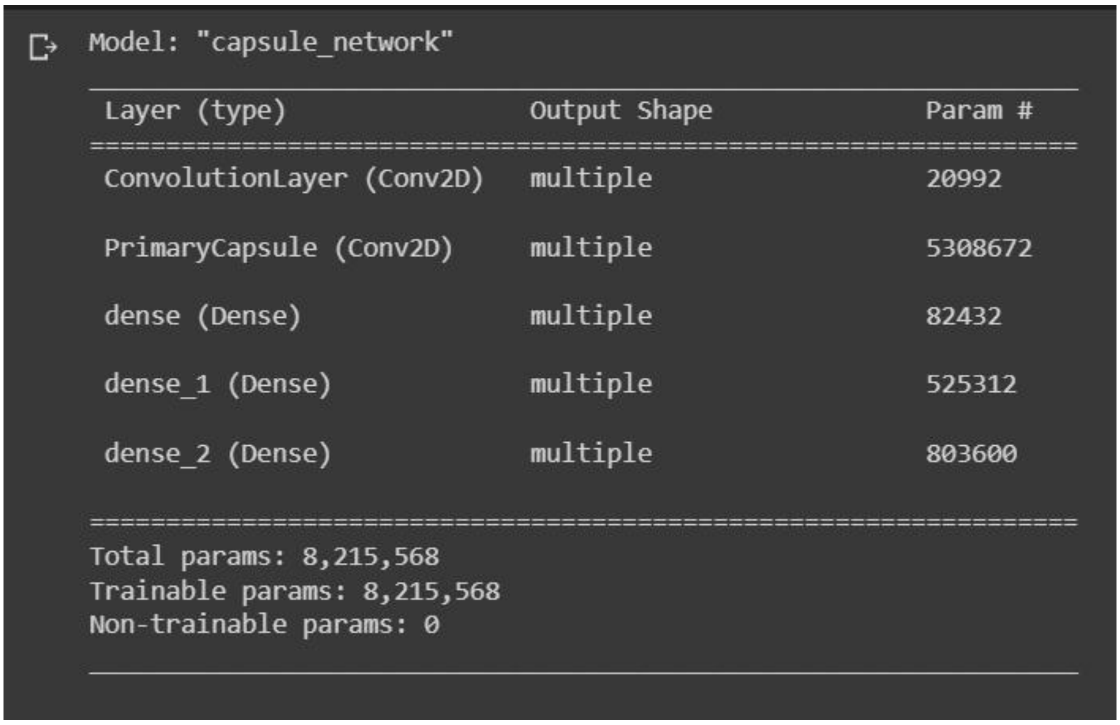
The activation parameter to the Conv2D class is simply a convenience parameter which allows you to supply a string, which specifies the name of the activation function you want to apply after performing the convolution.

2. **MaxPool2D( )** - Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

3. **Flatten( )** - flatten function flattens the multi-dimensional input tensors into a single dimension, so you can model your input layer and build your neural network model, then pass those data into every single neuron of the model effectively.

4. **Dense( )** - A Dense layer feeds all outputs from the previous layer to all its neurons, each neuron providing one output to the next layer. It's the most basic layer in neural networks.

4.3.2. CapsNet Model



```
Model: "capsule_network"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 ConvolutionLayer (Conv2D)   multiple                  20992

 PrimaryCapsule (Conv2D)     multiple                  5308672

 dense (Dense)               multiple                  82432

 dense_1 (Dense)             multiple                  525312

 dense_2 (Dense)             multiple                  803600

=================================================================
Total params: 8,215,568
Trainable params: 8,215,568
Non-trainable params: 0
_________________________________________________________________
```

Fig. 4.(c)

Capsule Network architecture is somewhat similar to convolutional neural network except capsule layers. We can break the implementation of capsule network into following steps:

Primary Capsule Layer: The output from the previous layer is being passed to 256 filters each of size 9*9 with a stride of 2 which will produce an output of size 6*6*256. This output is then reshaped into 8-dimensional vector. So shape will be 6*6*32 capsules each of which will be 8-dimensional. Then it will pass through a non-linear function(squash) so that length of output vector can be maintained between 0 and 1.

Digit Capsule Layer: Logic and algorithm used for this layer is explained in the previous blog. Here we will see what we need to do in code to implement it. We need to write a custom layer in keras. It will take 1152*8 as its input and produces output of size 10*16, where 10 capsules each represents an output class with 16-dimensional vector. Then each of these 10 capsules are converted into single value to predict the output class using a lambda layer.

Decoder Network: To further boost the pose parameters learned by the digit capsule layer, we can add decoder network to reconstruct the input image. In this part, decoder network will be fed with an input of size 10*16 (digit capsule layer output) and will reconstruct back the original image of size 28*28. Decoder will consist of 3 dense layers having 512, 1024 and 784 nodes.

During training time input to the decoder is the output from digit capsule layer which is masked with original labels. It means that other vectors except the vector corresponding to correct label will be multiplied with zero. So that decoder can only be trained with correct digit capsule. In test time input to decoder will be the same output from digit capsule layer but masked with highest length vector in that layer.

Loss Functions: It uses two loss function one is probabilistic loss function used for classifying digits image and another is reconstruction loss which is mean squared error.

CHAPTER 5

TECHNICAL IMPLEMENTATION & ANALYSIS

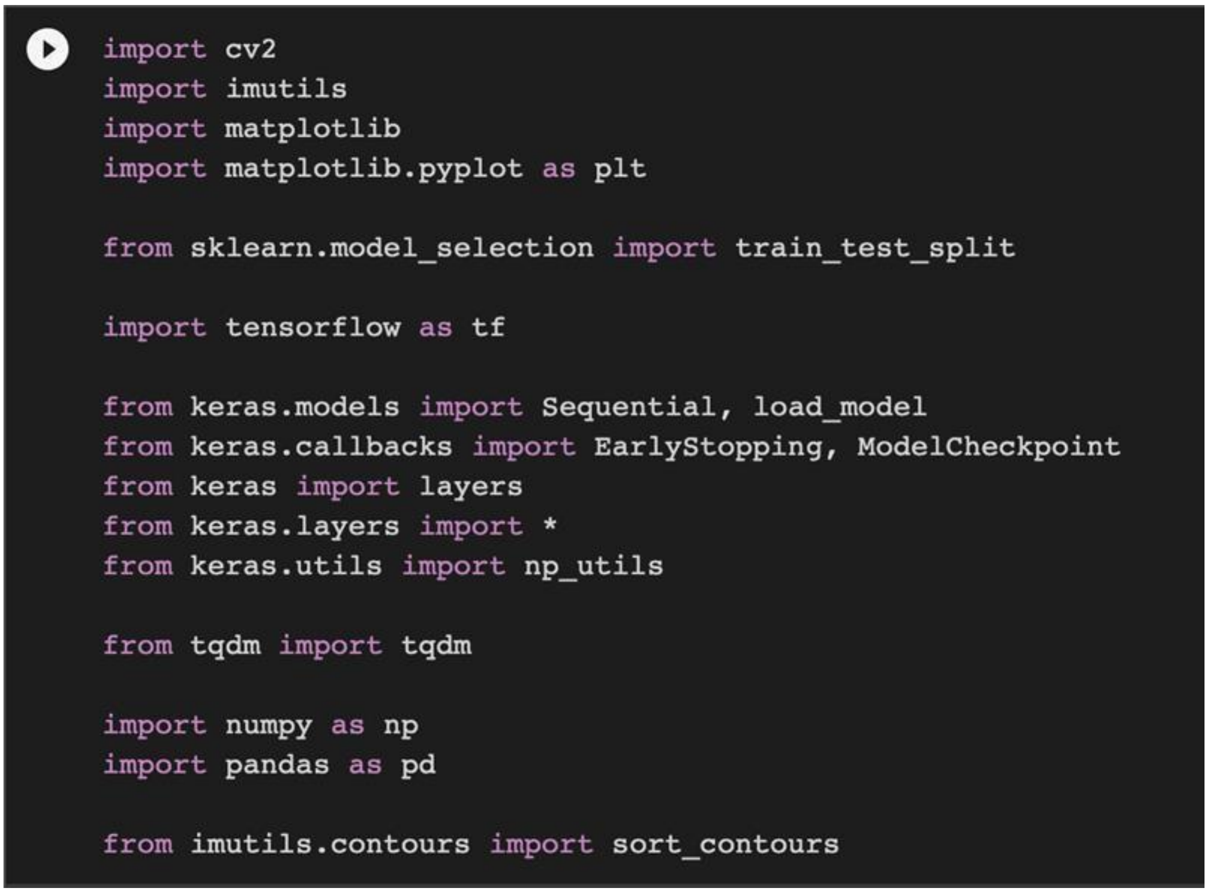
5.1 OUTLINE

In this section of the report we will discuss about the coding aspect of the project. We have implemented the CNN and CapsNet codes in Python Programming Language. We have also analyzed the accuracy and loss of the trained data using graphical representations.

5.2 TECHNICAL CODING AND CODE SOLUTIONS

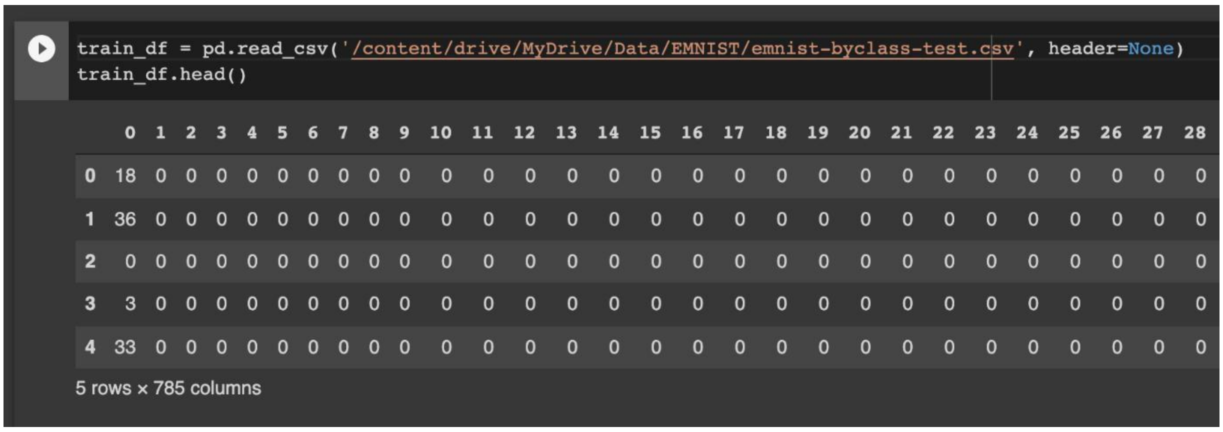
5.2.1 CNN MODEL

STEP 1: Importing the libraries and reading the dataset



```python
import cv2
import imutils
import matplotlib
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import tensorflow as tf

from keras.models import Sequential, load_model
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import layers
from keras.layers import *
from keras.utils import np_utils

from tqdm import tqdm

import numpy as np
import pandas as pd

from imutils.contours import sort_contours
```

Fig. 5.(a)



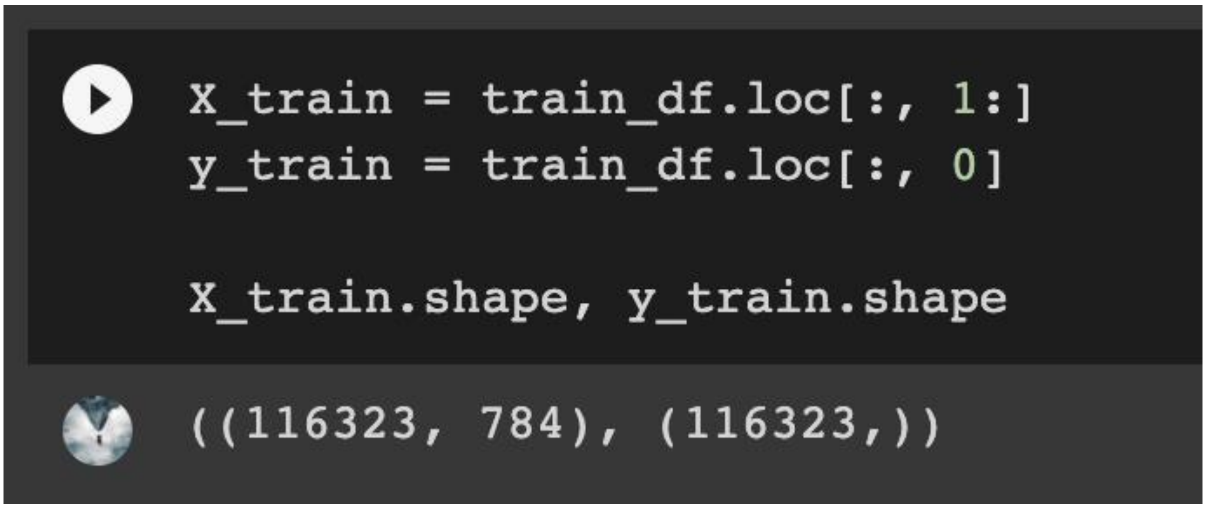
```python
train_df = pd.read_csv('/content/drive/MyDrive/Data/EMNIST/emnist-byclass-test.csv', header=None)
train_df.head()
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
0	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 785 columns

Fig. 5.(b)

STEP 2: Dividing the data into 'X_train' and 'y_train' and getting their shape



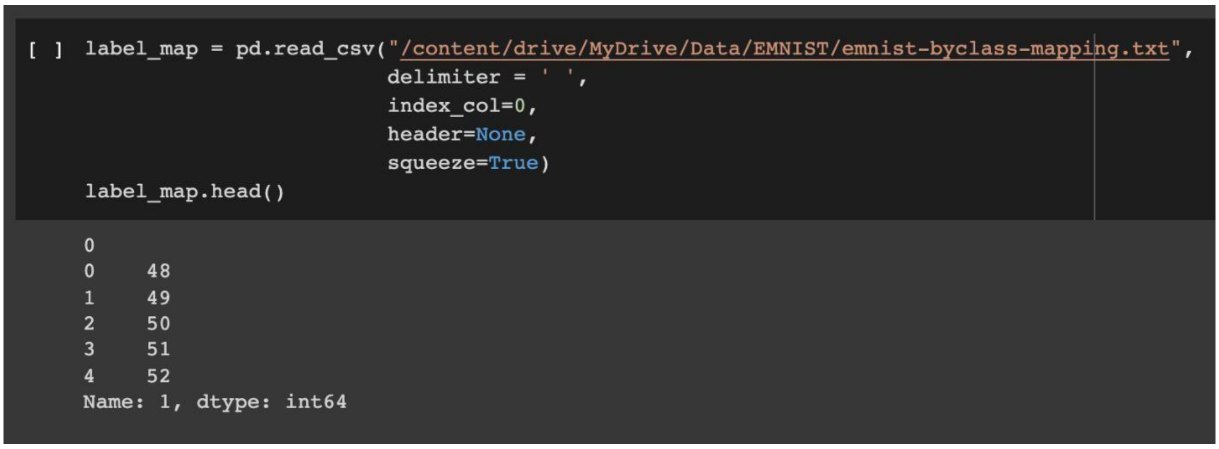
```python
X_train = train_df.loc[:, 1:]
y_train = train_df.loc[:, 0]

X_train.shape, y_train.shape
```

```
((116323, 784), (116323,))
```

Fig. 5.(c)

STEP 3: In a computer vision dataset, it is common to have annotations referring to class labels. In order to annotate an image, an image annotation file will often define the annotations specific to a particular image. This annotation file may or may not contain the class labels specific to the annotation in question.

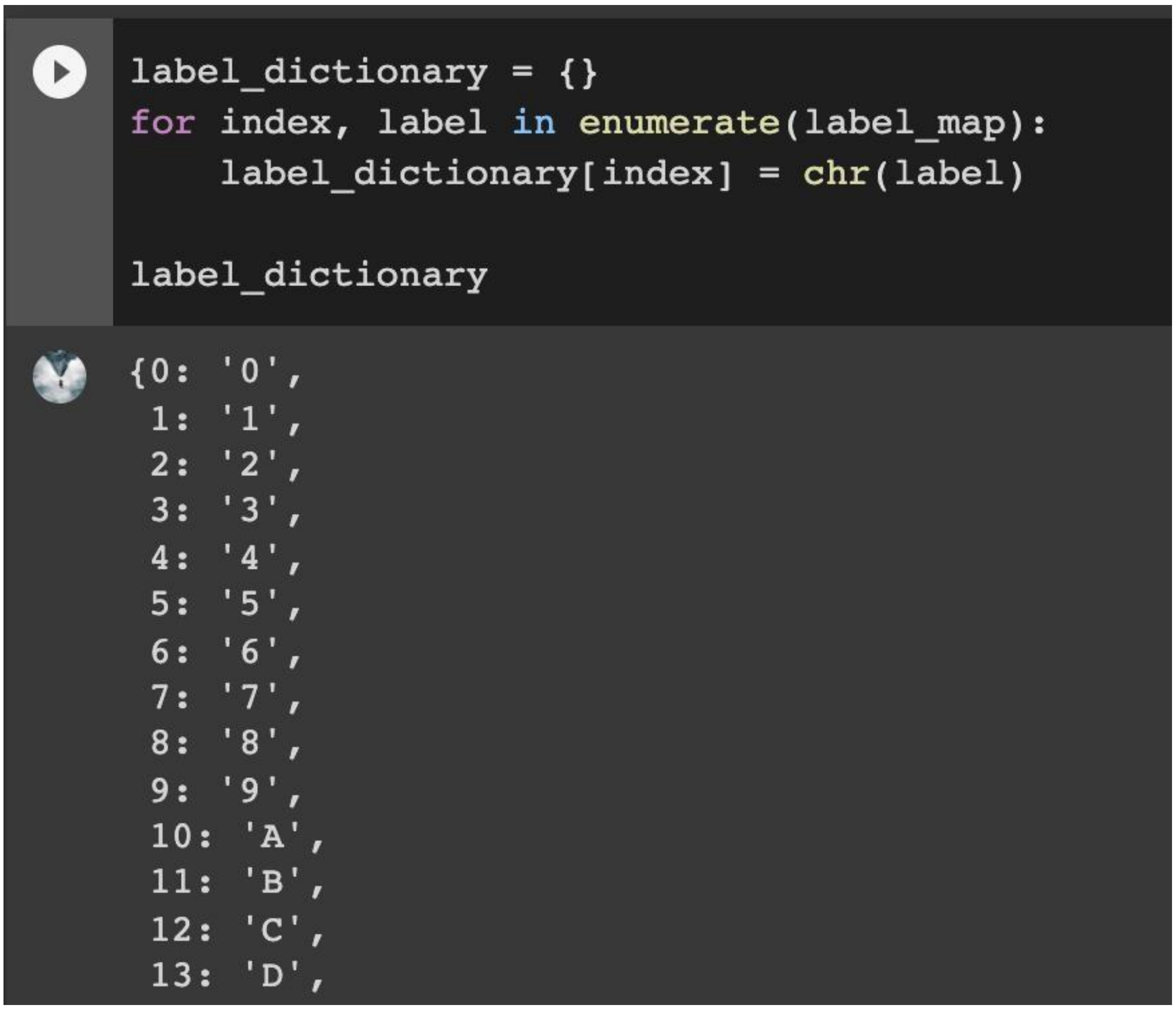
In the case where the annotation file does not specify class labels, a label map is referenced to look up the class name. The label map is the separate source of record for class annotations.



```
[ ]  label_map = pd.read_csv("/content/drive/MyDrive/Data/EMNIST/emnist-byclass-mapping.txt",
                              delimiter = ' ',
                              index_col=0,
                              header=None,
                              squeeze=True)
     label_map.head()

     0
     0    48
     1    49
     2    50
     3    51
     4    52
     Name: 1, dtype: int64
```

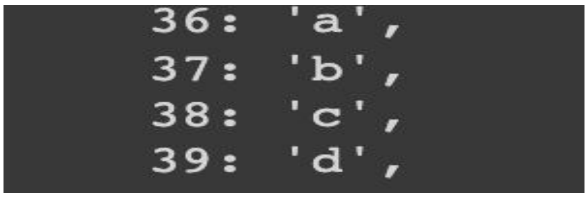
Fig. 5.(d)

STEP 4: Creating a dictionary in order to label the digits(0-9), upper case letters (A-Z) and lower-case letters(a-z) from 0 to 61.



```
label_dictionary = {}
for index, label in enumerate(label_map):
    label_dictionary[index] = chr(label)

label_dictionary

{0: '0',
 1: '1',
 2: '2',
 3: '3',
 4: '4',
 5: '5',
 6: '6',
 7: '7',
 8: '8',
 9: '9',
 10: 'A',
 11: 'B',
 12: 'C',
 13: 'D',
```

Fig. 5.(e)

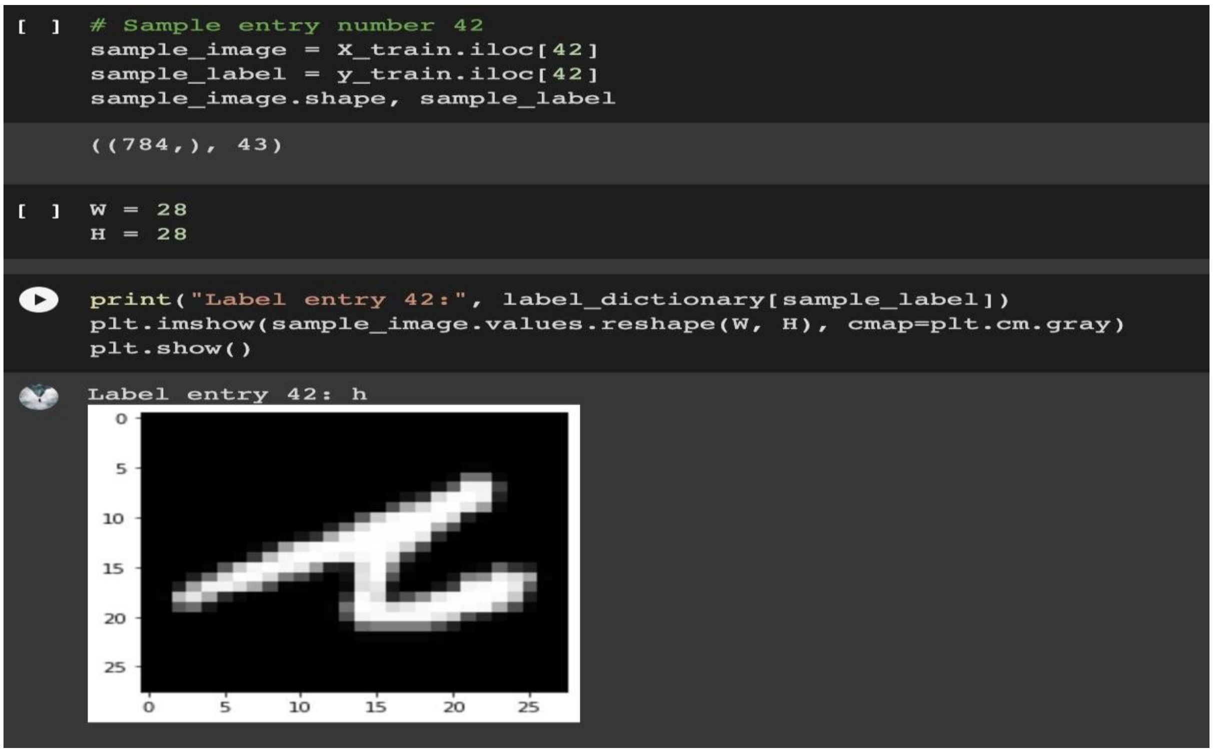


```
36: 'a',
37: 'b',
38: 'c',
39: 'd',
```

Fig. 5.(f)

STEP 5: Extracting a sample image

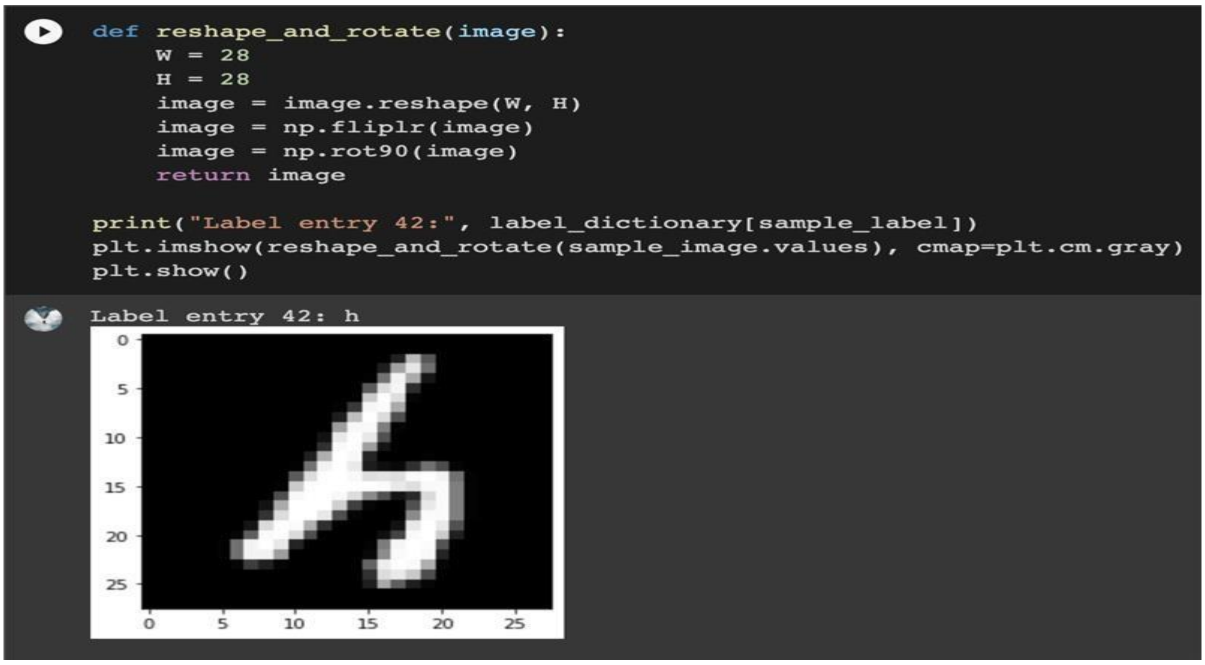
Defining height and width of the sample images as 28 pixels and plotting it.



```
[ ]  # Sample entry number 42
     sample_image = X_train.iloc[42]
     sample_label = y_train.iloc[42]
     sample_image.shape, sample_label

     ((784,), 43)

[ ]  W = 28
     H = 28

     print("Label entry 42:", label_dictionary[sample_label])
     plt.imshow(sample_image.values.reshape(W, H), cmap=plt.cm.gray)
     plt.show()

     Label entry 42: h
```

Fig. 5.(g)

STEP 6: Rotating and reshaping the sample image

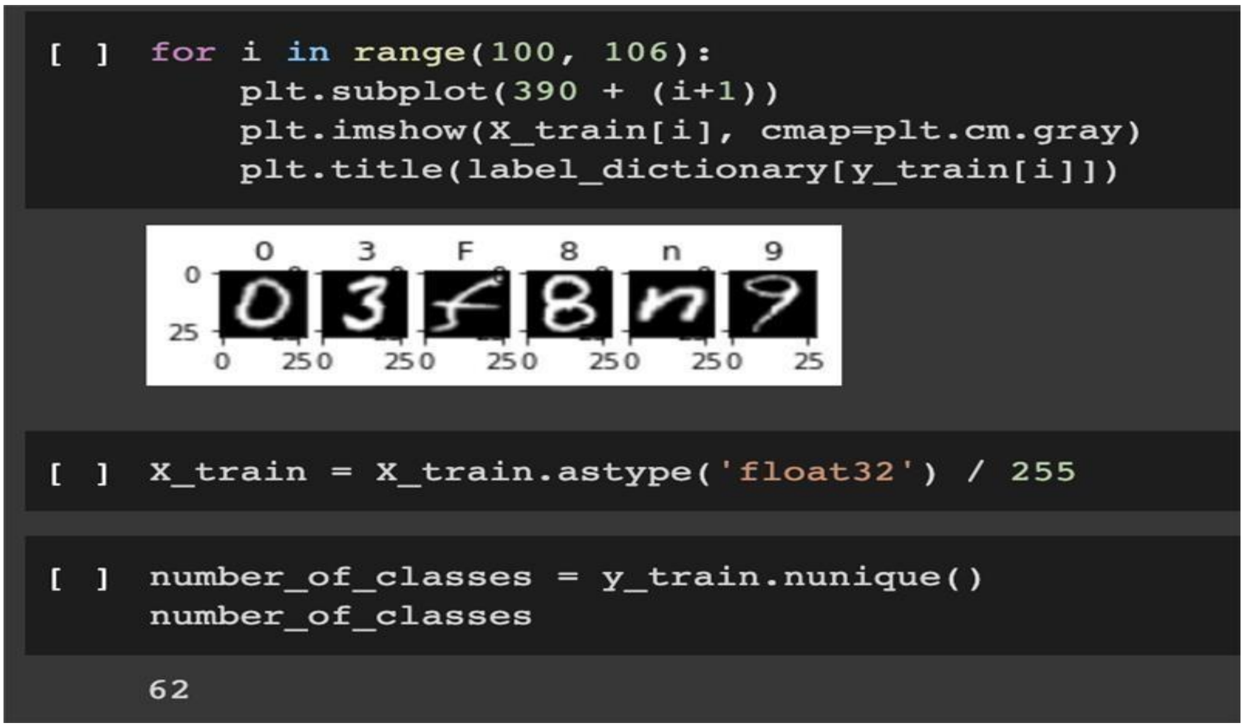


```
def reshape_and_rotate(image):
    W = 28
    H = 28
    image = image.reshape(W, H)
    image = np.fliplr(image)
    image = np.rot90(image)
    return image

print("Label entry 42:", label_dictionary[sample_label])
plt.imshow(reshape_and_rotate(sample_image.values), cmap=plt.cm.gray)
plt.show()
```

```
Label entry 42: h
```

Fig. 5.(h)

STEP 7: Taking the sample images in a range (100,106) and detecting the text.



```
[ ] for i in range(100, 106):
        plt.subplot(390 + (i+1))
        plt.imshow(X_train[i], cmap=plt.cm.gray)
        plt.title(label_dictionary[y_train[i]])
```

```
[ ] X_train = X_train.astype('float32') / 255
```

```
[ ] number_of_classes = y_train.nunique()
    number_of_classes

    62
```

Fig. 5.(i)

Range (10,16)

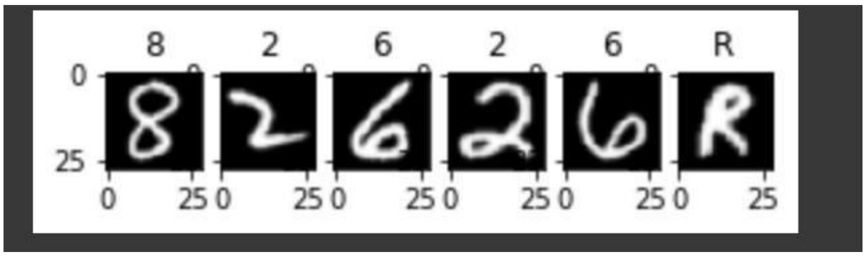


Fig. 5.(j)

Range(42,48)

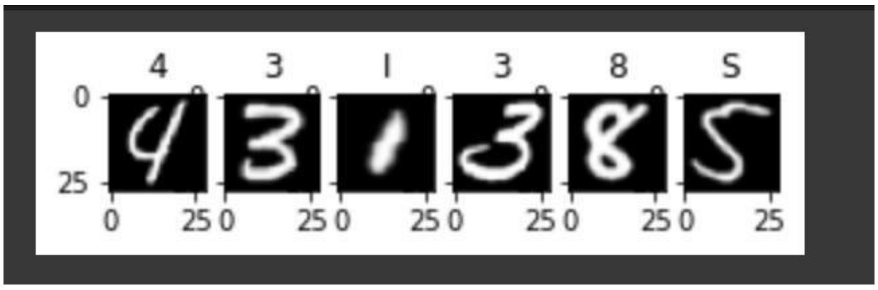
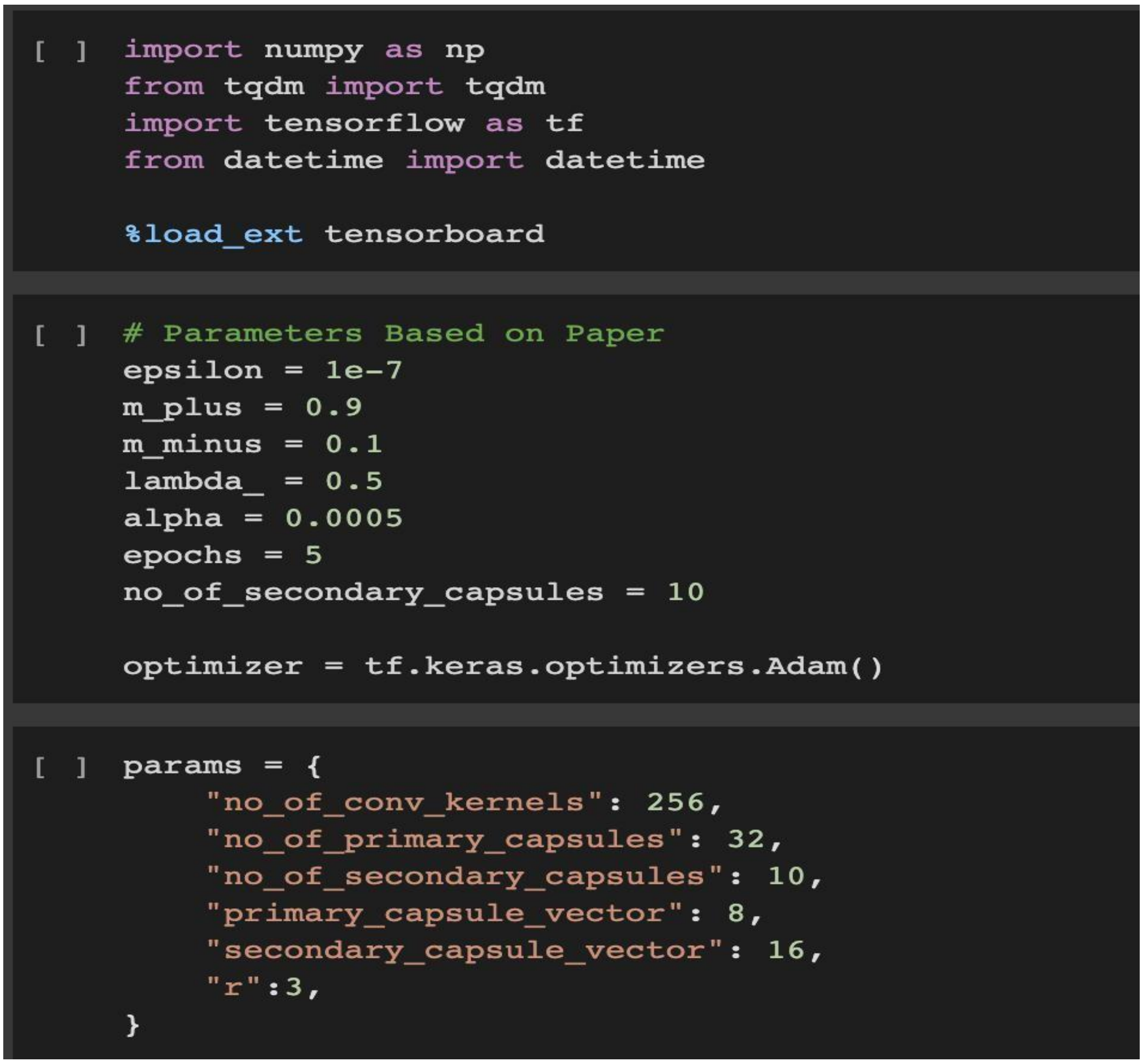


Fig. 5.(k)

5.2.2 CAPSNET MODEL

STEP 1: Importing libraries which are used in capsule network model.



```
[ ]  import numpy as np
     from tqdm import tqdm
     import tensorflow as tf
     from datetime import datetime

     %load_ext tensorboard
```

```
[ ]  # Parameters Based on Paper
     epsilon = 1e-7
     m_plus = 0.9
     m_minus = 0.1
     lambda_ = 0.5
     alpha = 0.0005
     epochs = 5
     no_of_secondary_capsules = 10

     optimizer = tf.keras.optimizers.Adam()
```

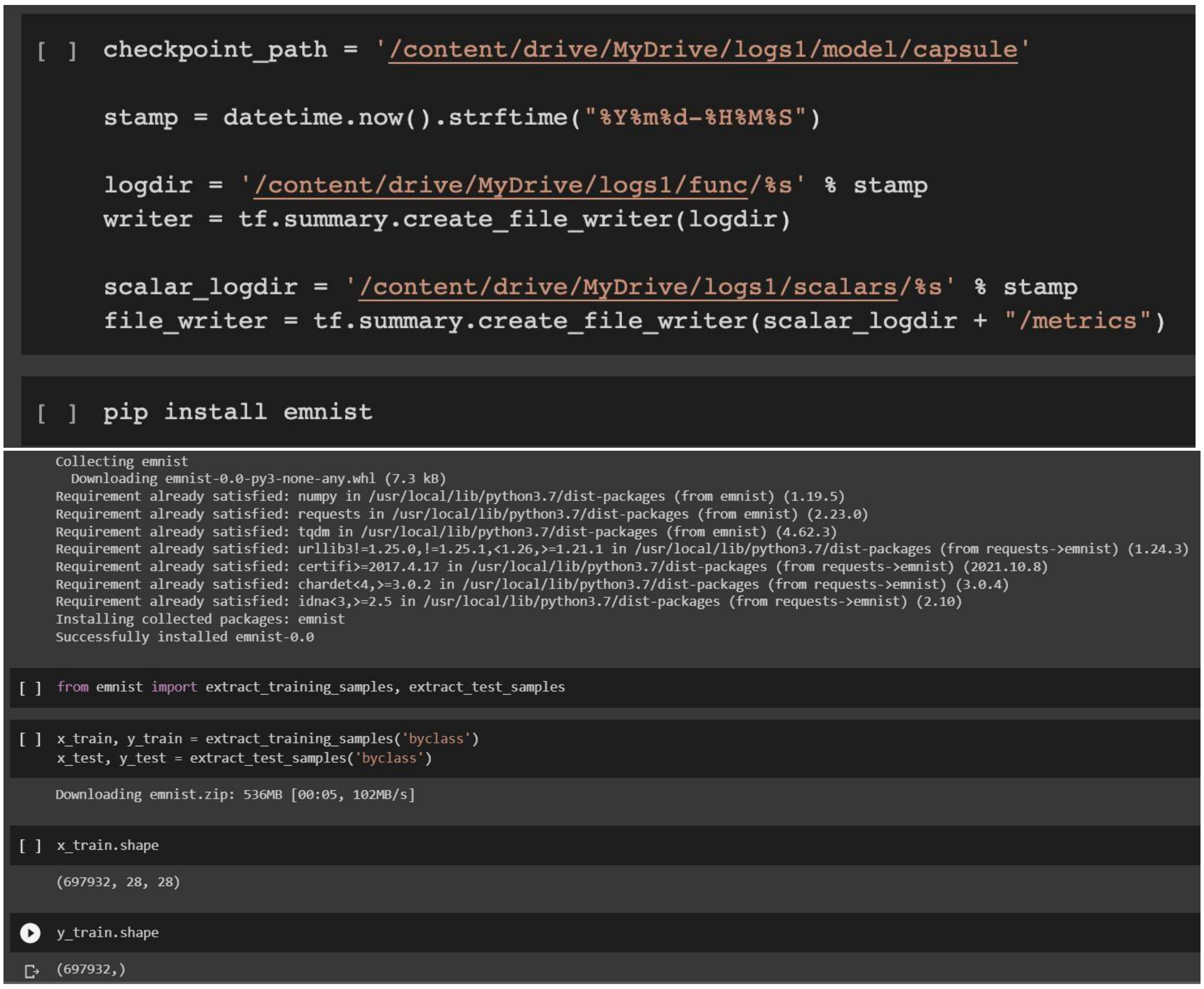
```
[ ]  params = {
         "no_of_conv_kernels": 256,
         "no_of_primary_capsules": 32,
         "no_of_secondary_capsules": 10,
         "primary_capsule_vector": 8,
         "secondary_capsule_vector": 16,
         "r":3,
     }
```

Fig. 5.(l)

STEP 2: Log directory is a directory where the application creates log files. The location and content of log files is defined by the configuration of the Logback framework provided in the logback. xml file.

Log directories are the primary data source for network observability. A log file is a computer-generated data file that contains information about usage patterns, activities, and operations within an operating system, application, server or another device.

Installing and studying the pre-existing dataset of the tensorflow module - EMNIST.



```
[ ]  checkpoint_path = '/content/drive/MyDrive/logs1/model/capsule'

     stamp = datetime.now().strftime("%Y%m%d-%H%M%S")

     logdir = '/content/drive/MyDrive/logs1/func/%s' % stamp
     writer = tf.summary.create_file_writer(logdir)

     scalar_logdir = '/content/drive/MyDrive/logs1/scalars/%s' % stamp
     file_writer = tf.summary.create_file_writer(scalar_logdir + "/metrics")
```

```
[ ]  pip install emnist
```

```
Collecting emnist
  Downloading emnist-0.0-py3-none-any.whl (7.3 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from emnist) (1.19.5)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from emnist) (2.23.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from emnist) (4.62.3)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests->emnist) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->emnist) (2021.10.8)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->emnist) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->emnist) (2.10)
Installing collected packages: emnist
Successfully installed emnist-0.0
```

```
[ ] from emnist import extract_training_samples, extract_test_samples
```

```
[ ] x_train, y_train = extract_training_samples('byclass')
    x_test, y_test = extract_test_samples('byclass')

    Downloading emnist.zip: 536MB [00:05, 102MB/s]
```

```
[ ] x_train.shape

    (697932, 28, 28)
```

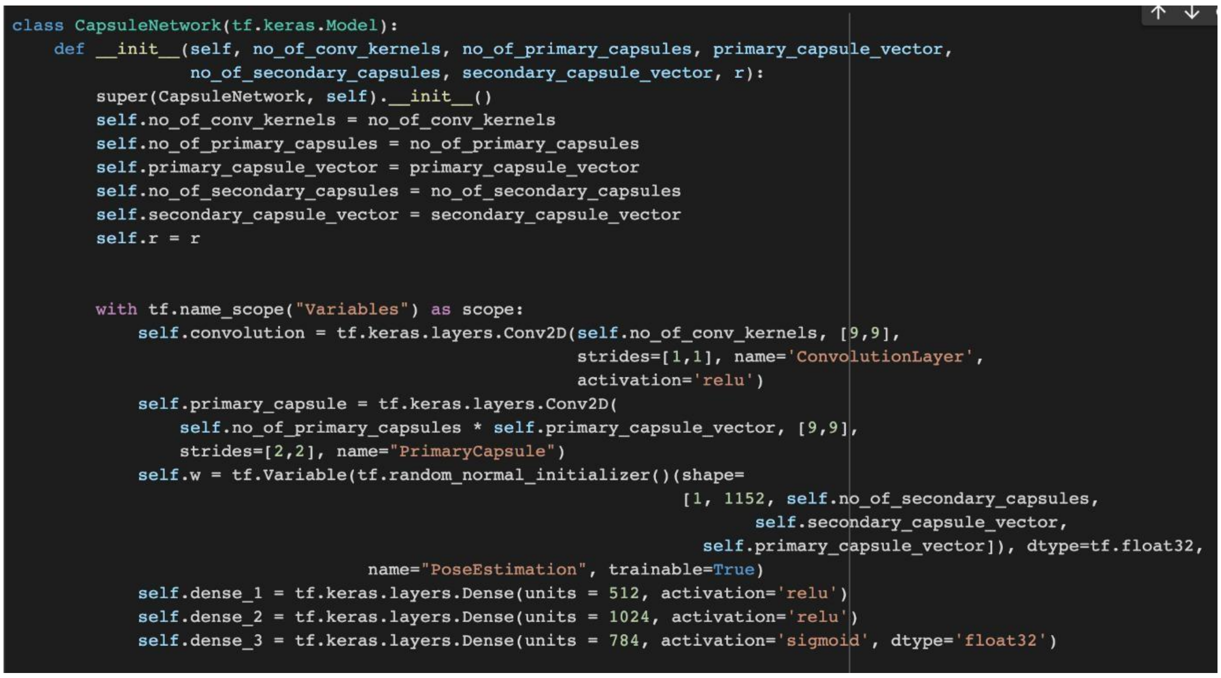
```
y_train.shape

(697932,)
```

Fig. 5.(m)

STEP 3:

Defining a class named ‘CapsuleNetwork’.

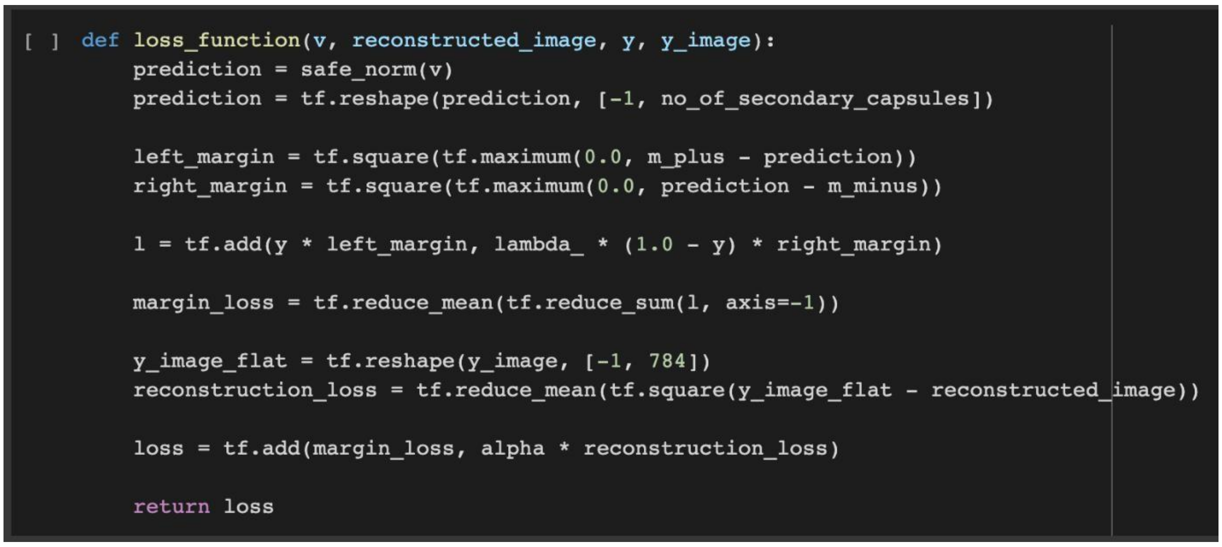
Defining the model using different parameters such as number of primary capules, number of primary capsule vectors, number of secondary capules, number of secondary capsule vectors, number of convolutional kernels, etc.



```
class CapsuleNetwork(tf.keras.Model):
    def __init__(self, no_of_conv_kernels, no_of_primary_capsules, primary_capsule_vector,
                 no_of_secondary_capsules, secondary_capsule_vector, r):
        super(CapsuleNetwork, self).__init__()
        self.no_of_conv_kernels = no_of_conv_kernels
        self.no_of_primary_capsules = no_of_primary_capsules
        self.primary_capsule_vector = primary_capsule_vector
        self.no_of_secondary_capsules = no_of_secondary_capsules
        self.secondary_capsule_vector = secondary_capsule_vector
        self.r = r


        with tf.name_scope("Variables") as scope:
            self.convolution = tf.keras.layers.Conv2D(self.no_of_conv_kernels, [9,9],
                                                      strides=[1,1], name='ConvolutionLayer',
                                                      activation='relu')
            self.primary_capsule = tf.keras.layers.Conv2D(
                self.no_of_primary_capsules * self.primary_capsule_vector, [9,9],
                strides=[2,2], name="PrimaryCapsule")
            self.w = tf.Variable(tf.random_normal_initializer()(shape=
                                                                 [1, 1152, self.no_of_secondary_capsules,
                                                                        self.secondary_capsule_vector,
                                                                    self.primary_capsule_vector]), dtype=tf.float32,
                                 name="PoseEstimation", trainable=True)
            self.dense_1 = tf.keras.layers.Dense(units = 512, activation='relu')
            self.dense_2 = tf.keras.layers.Dense(units = 1024, activation='relu')
            self.dense_3 = tf.keras.layers.Dense(units = 784, activation='sigmoid', dtype='float32')
```

Fig. 5.(n)

Defining the loss function

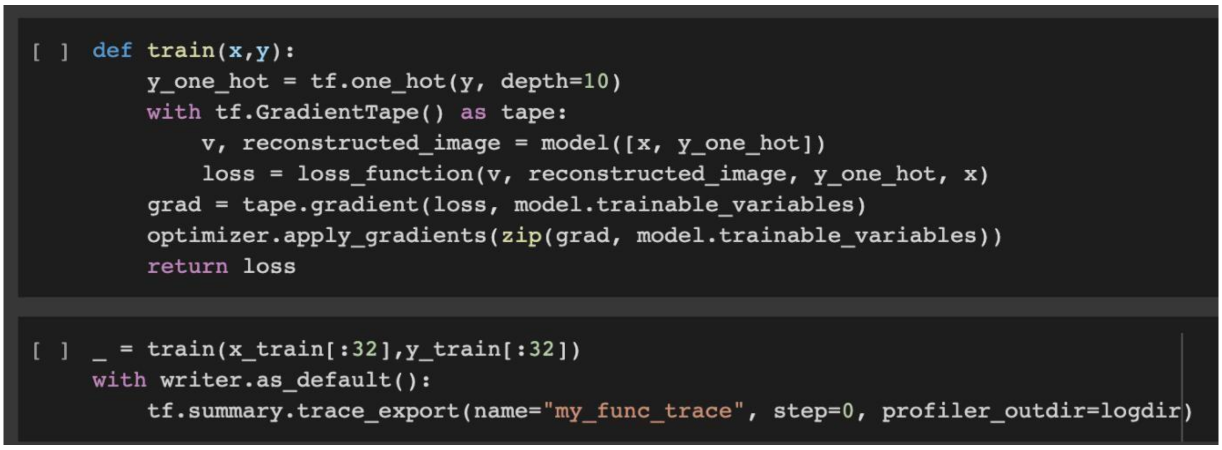


```
[ ] def loss_function(v, reconstructed_image, y, y_image):
        prediction = safe_norm(v)
        prediction = tf.reshape(prediction, [-1, no_of_secondary_capsules])

        left_margin = tf.square(tf.maximum(0.0, m_plus - prediction))
        right_margin = tf.square(tf.maximum(0.0, prediction - m_minus))

        l = tf.add(y * left_margin, lambda_ * (1.0 - y) * right_margin)

        margin_loss = tf.reduce_mean(tf.reduce_sum(l, axis=-1))

        y_image_flat = tf.reshape(y_image, [-1, 784])
        reconstruction_loss = tf.reduce_mean(tf.square(y_image_flat - reconstructed_image))

        loss = tf.add(margin_loss, alpha * reconstruction_loss)

        return loss
```

Fig. 5.(o)

Loss functions measure how far an estimated value is from its true value. A loss function maps decisions to their associated costs. Loss functions are not fixed, they change depending on the task in hand and the goal to be met.

STEP 4:

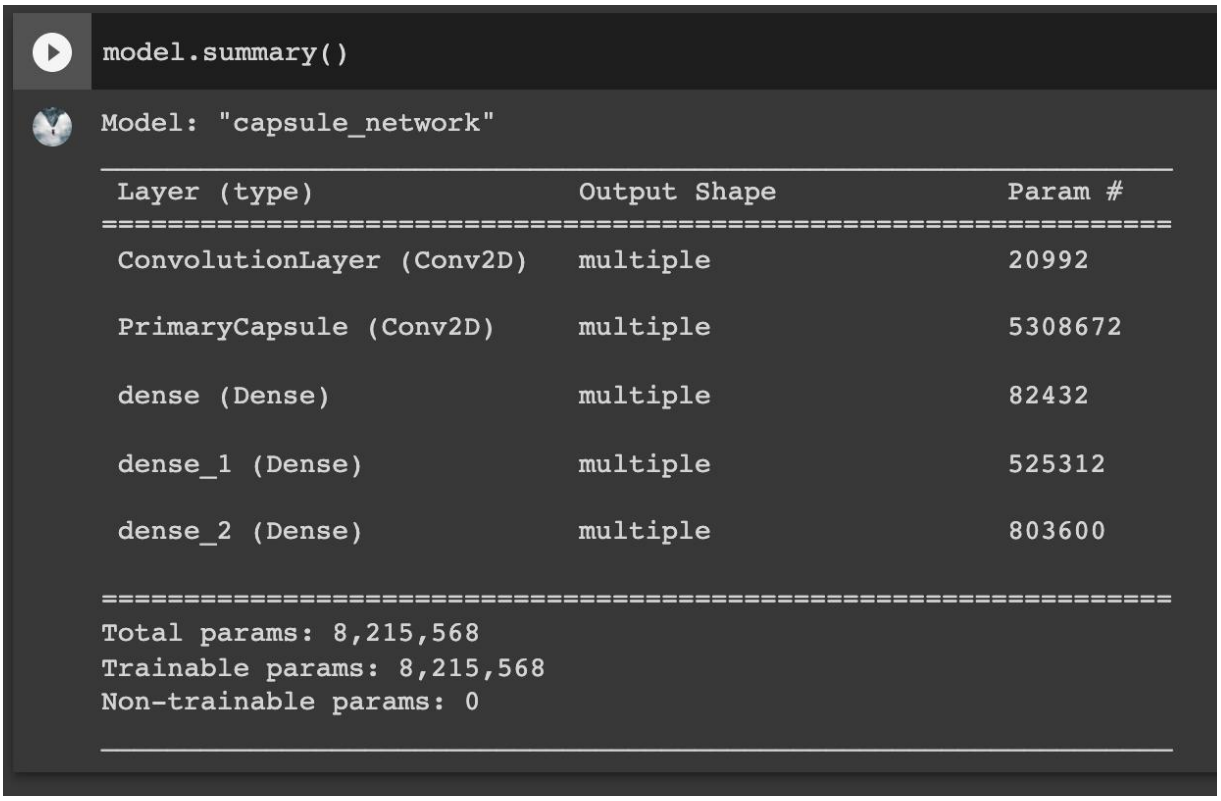


```
[ ] def train(x,y):
        y_one_hot = tf.one_hot(y, depth=10)
        with tf.GradientTape() as tape:
            v, reconstructed_image = model([x, y_one_hot])
            loss = loss_function(v, reconstructed_image, y_one_hot, x)
        grad = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(grad, model.trainable_variables))
        return loss
```

```
[ ] _ = train(x_train[:32],y_train[:32])
    with writer.as_default():
        tf.summary.trace_export(name="my_func_trace", step=0, profiler_outdir=logdir)
```

Fig. 5.(p)

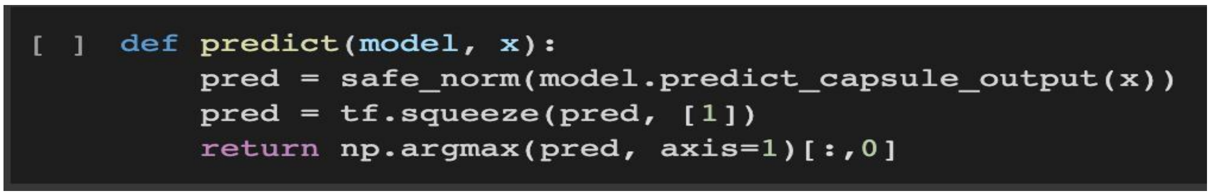
STEP 5: Summarizing the model



```
model.summary()

Model: "capsule_network"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 ConvolutionLayer (Conv2D)   multiple                  20992

 PrimaryCapsule (Conv2D)     multiple                  5308672

 dense (Dense)               multiple                  82432

 dense_1 (Dense)             multiple                  525312

 dense_2 (Dense)             multiple                  803600

=================================================================
Total params: 8,215,568
Trainable params: 8,215,568
Non-trainable params: 0
_________________________________________________________________
```

Fig. 5.(q)

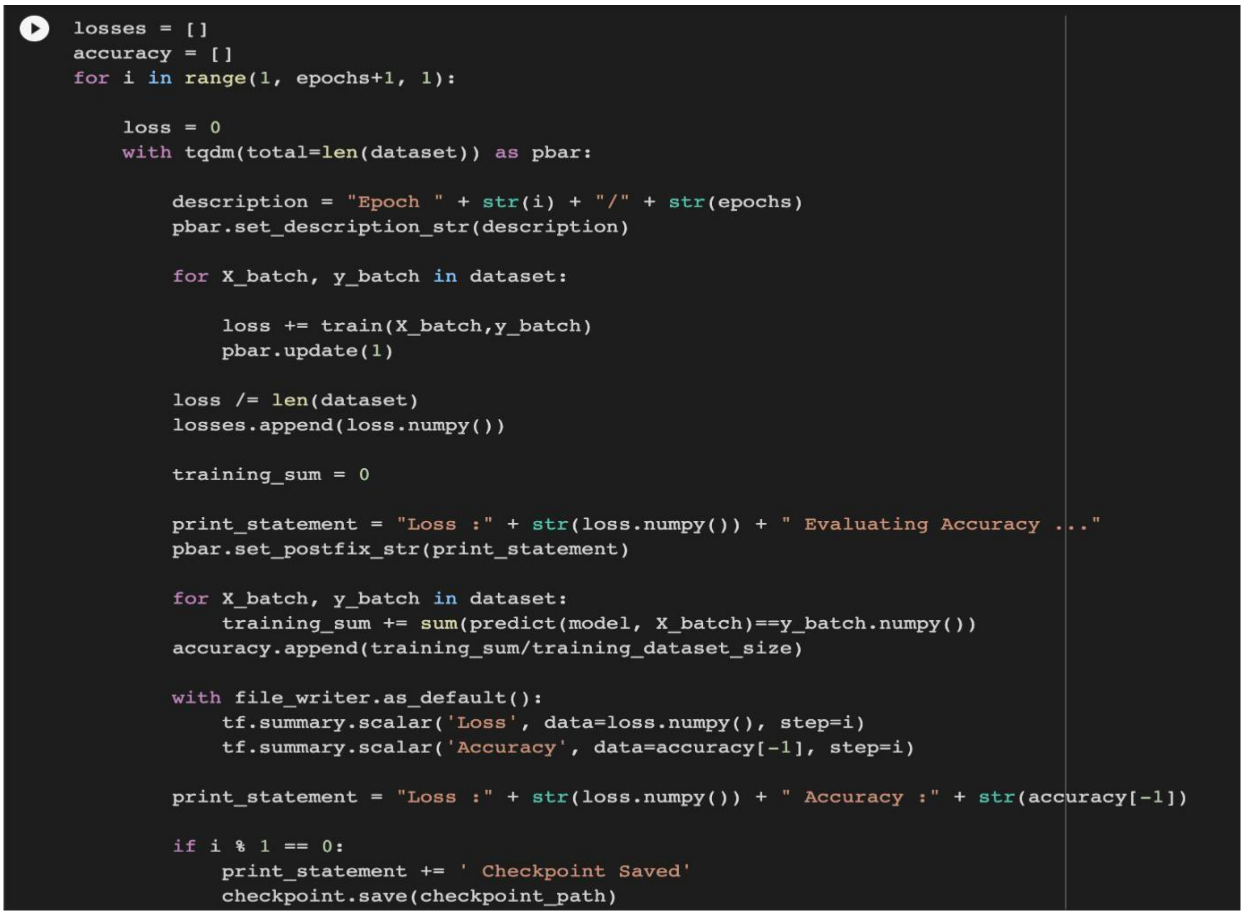
STEP 6: Predicting the model



```
[ ]  def predict(model, x):
         pred = safe_norm(model.predict_capsule_output(x))
         pred = tf.squeeze(pred, [1])
         return np.argmax(pred, axis=1)[:,0]
```

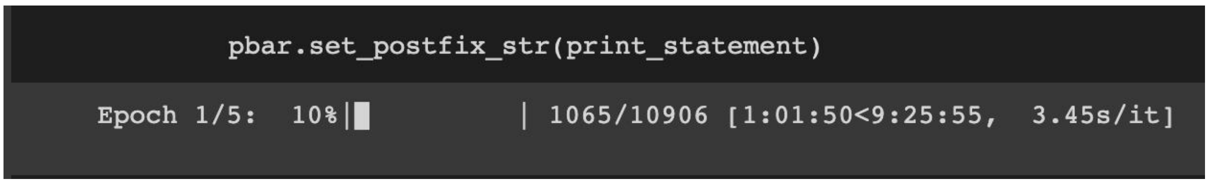
Fig. 5.(r)

STEP 7: Calculating loss and accuracy of the model



```
losses = []
accuracy = []
for i in range(1, epochs+1, 1):

    loss = 0
    with tqdm(total=len(dataset)) as pbar:

        description = "Epoch " + str(i) + "/" + str(epochs)
        pbar.set_description_str(description)

        for X_batch, y_batch in dataset:

            loss += train(X_batch,y_batch)
            pbar.update(1)

        loss /= len(dataset)
        losses.append(loss.numpy())

        training_sum = 0

        print_statement = "Loss :" + str(loss.numpy()) + " Evaluating Accuracy ..."
        pbar.set_postfix_str(print_statement)

        for X_batch, y_batch in dataset:
            training_sum += sum(predict(model, X_batch)==y_batch.numpy())
        accuracy.append(training_sum/training_dataset_size)

        with file_writer.as_default():
            tf.summary.scalar('Loss', data=loss.numpy(), step=i)
            tf.summary.scalar('Accuracy', data=accuracy[-1], step=i)

        print_statement = "Loss :" + str(loss.numpy()) + " Accuracy :" + str(accuracy[-1])

        if i % 1 == 0:
            print_statement += ' Checkpoint Saved'
            checkpoint.save(checkpoint_path)
```

Fig. 5.(s)



```
        pbar.set_postfix_str(print_statement)
Epoch 1/5:  10%|█          | 1065/10906 [1:01:50<9:25:55,  3.45s/it]
```

Fig. 5.(t)

Time taken for 1 epoch – CPU = 10 hrs

GPU= 1.5 hrs

5.3 PROTOTYPE

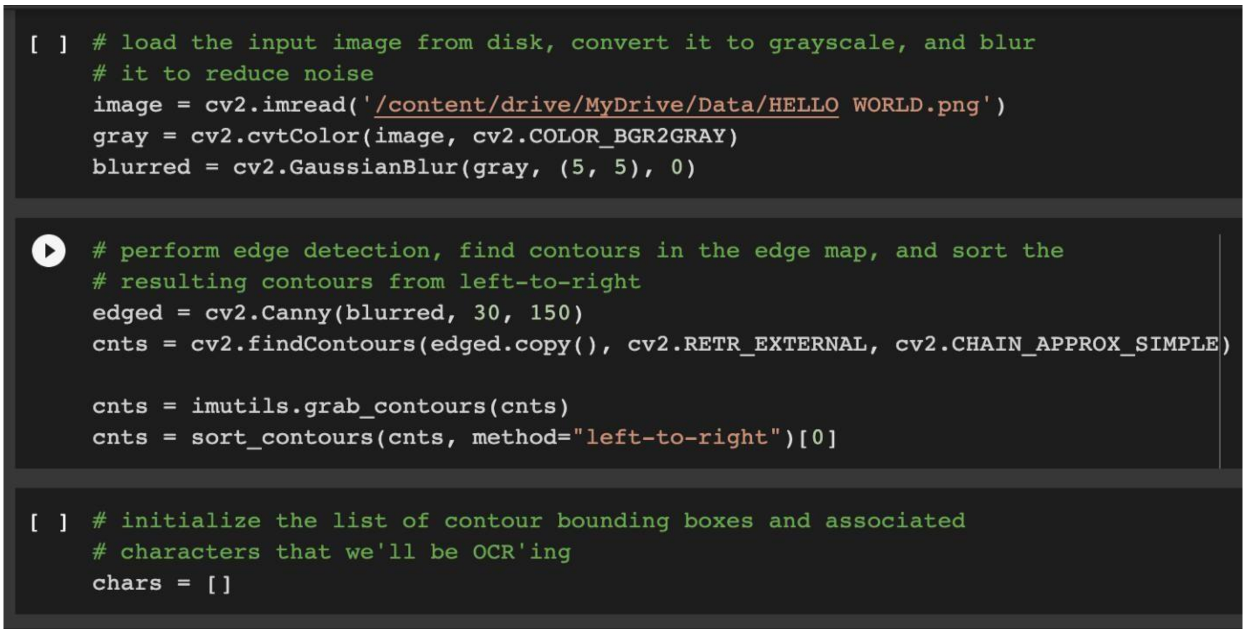
CNN MODEL-

https://colab.research.google.com/drive/1Vu5h8OvWecNZd4KZolcHhikF4i59FJFj#scrollTo=Ql-_dr4zJkME

CAPSNET MODEL-

https://colab.research.google.com/drive/1Veo7M_5FxEb-RNVlDyU 1-eGrMrAlmW#scrollTo=1lwrYnr8yHjF

5.4 TEST AND VALIDATION



```
[ ] # load the input image from disk, convert it to grayscale, and blur
    # it to reduce noise
    image = cv2.imread('/content/drive/MyDrive/Data/HELLO WORLD.png')
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    blurred = cv2.GaussianBlur(gray, (5, 5), 0)
```

```
    # perform edge detection, find contours in the edge map, and sort the
    # resulting contours from left-to-right
    edged = cv2.Canny(blurred, 30, 150)
    cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    cnts = imutils.grab_contours(cnts)
    cnts = sort_contours(cnts, method="left-to-right")[0]
```

```
[ ] # initialize the list of contour bounding boxes and associated
    # characters that we'll be OCR'ing
    chars = []
```

Fig. 5.(u)

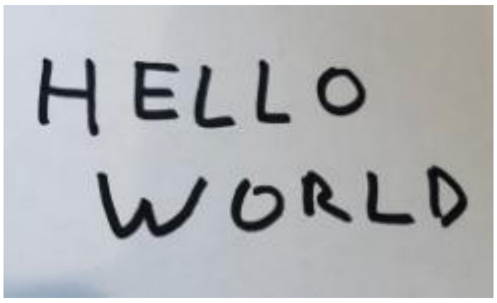
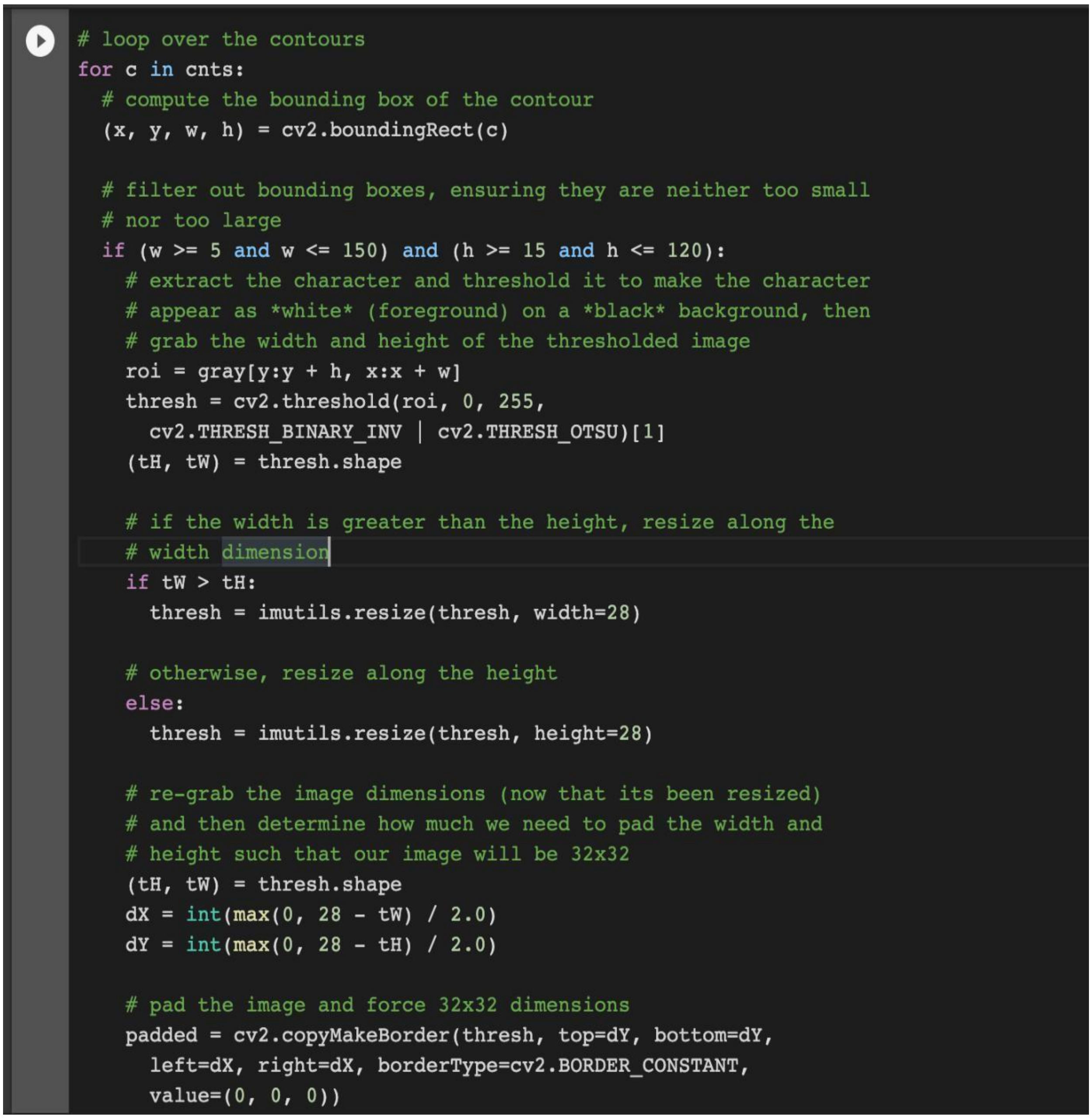
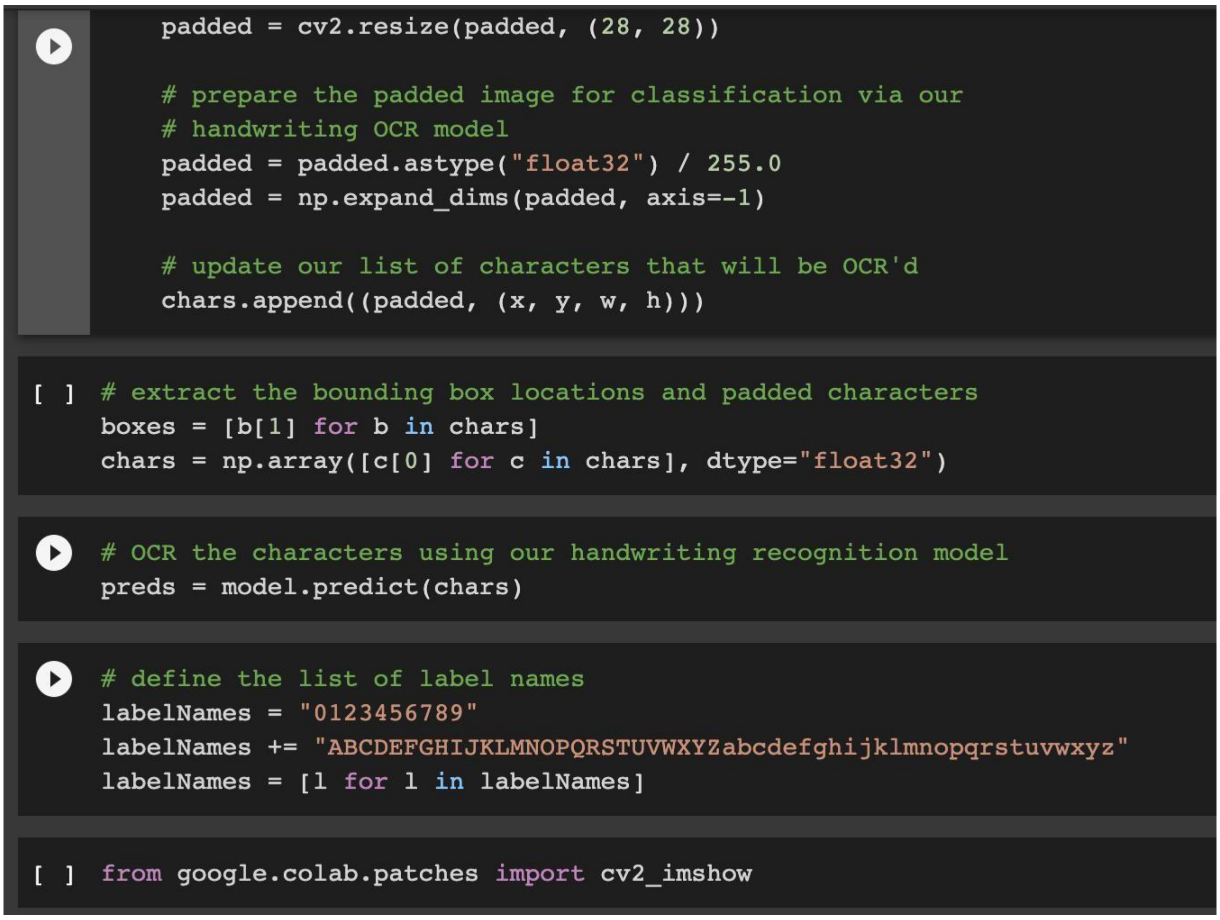


Fig. 5.(v)



```python
# loop over the contours
for c in cnts:
  # compute the bounding box of the contour
  (x, y, w, h) = cv2.boundingRect(c)

  # filter out bounding boxes, ensuring they are neither too small
  # nor too large
  if (w >= 5 and w <= 150) and (h >= 15 and h <= 120):
    # extract the character and threshold it to make the character
    # appear as *white* (foreground) on a *black* background, then
    # grab the width and height of the thresholded image
    roi = gray[y:y + h, x:x + w]
    thresh = cv2.threshold(roi, 0, 255,
      cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
    (tH, tW) = thresh.shape

    # if the width is greater than the height, resize along the
    # width dimension
    if tW > tH:
      thresh = imutils.resize(thresh, width=28)

    # otherwise, resize along the height
    else:
      thresh = imutils.resize(thresh, height=28)

    # re-grab the image dimensions (now that its been resized)
    # and then determine how much we need to pad the width and
    # height such that our image will be 32x32
    (tH, tW) = thresh.shape
    dX = int(max(0, 28 - tW) / 2.0)
    dY = int(max(0, 28 - tH) / 2.0)

    # pad the image and force 32x32 dimensions
    padded = cv2.copyMakeBorder(thresh, top=dY, bottom=dY,
      left=dX, right=dX, borderType=cv2.BORDER_CONSTANT,
      value=(0, 0, 0))
```

Fig. 5.(w)



```
    padded = cv2.resize(padded, (28, 28))

    # prepare the padded image for classification via our
    # handwriting OCR model
    padded = padded.astype("float32") / 255.0
    padded = np.expand_dims(padded, axis=-1)

    # update our list of characters that will be OCR'd
    chars.append((padded, (x, y, w, h)))
```

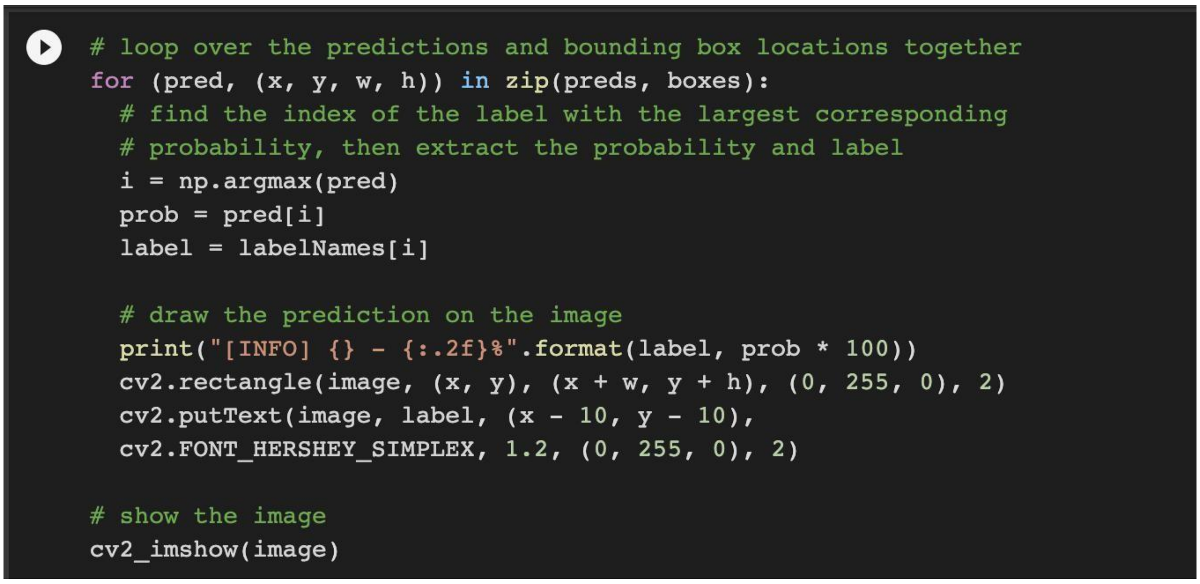
```
[ ] # extract the bounding box locations and padded characters
    boxes = [b[1] for b in chars]
    chars = np.array([c[0] for c in chars], dtype="float32")
```

```
# OCR the characters using our handwriting recognition model
preds = model.predict(chars)
```

```
# define the list of label names
labelNames = "0123456789"
labelNames += "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
labelNames = [l for l in labelNames]
```

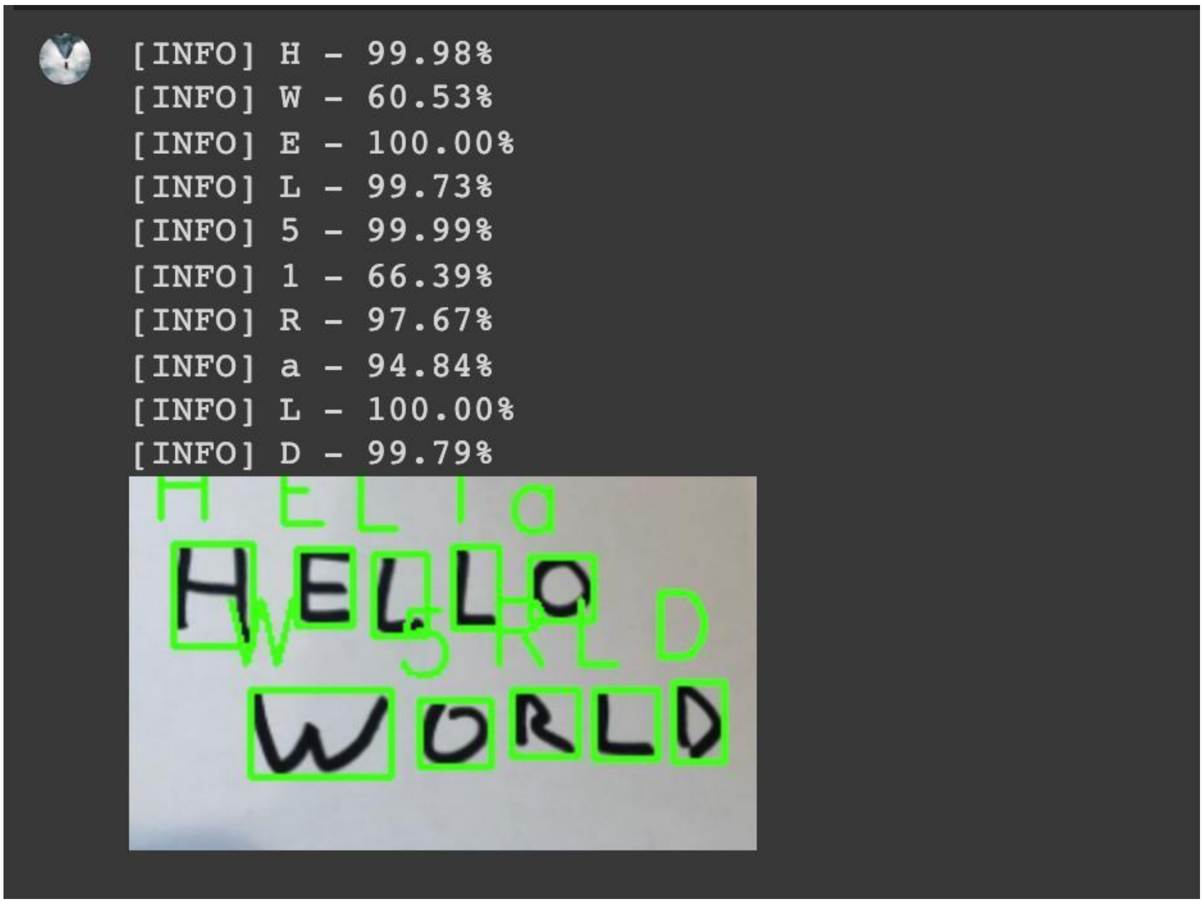
```
[ ] from google.colab.patches import cv2_imshow
```

Fig. 5.(x)



```
# loop over the predictions and bounding box locations together
for (pred, (x, y, w, h)) in zip(preds, boxes):
  # find the index of the label with the largest corresponding
  # probability, then extract the probability and label
  i = np.argmax(pred)
  prob = pred[i]
  label = labelNames[i]

  # draw the prediction on the image
  print("[INFO] {} - {:.2f}%".format(label, prob * 100))
  cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
  cv2.putText(image, label, (x - 10, y - 10),
  cv2.FONT_HERSHEY_SIMPLEX, 1.2, (0, 255, 0), 2)

# show the image
cv2_imshow(image)
```

Fig. 5.(y)



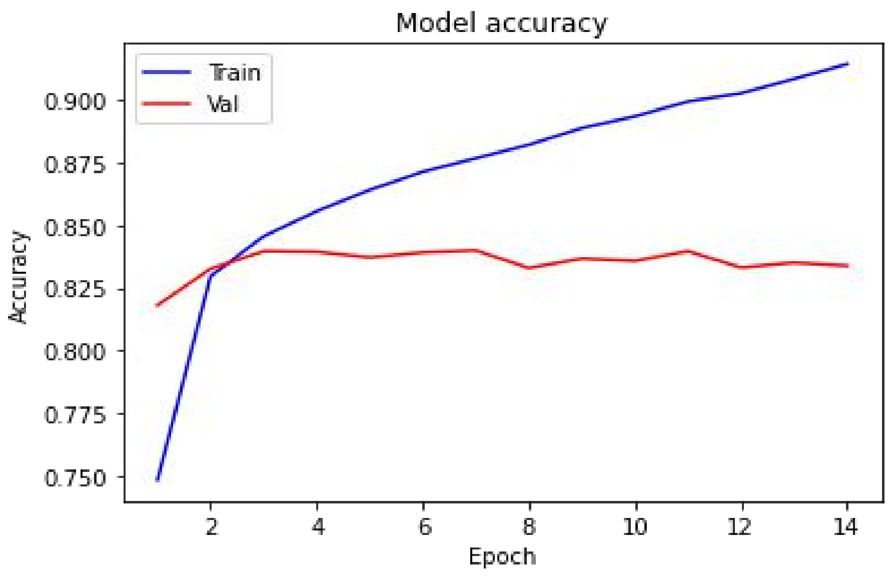
```
[INFO] H - 99.98%
[INFO] W - 60.53%
[INFO] E - 100.00%
[INFO] L - 99.73%
[INFO] 5 - 99.99%
[INFO] 1 - 66.39%
[INFO] R - 97.67%
[INFO] a - 94.84%
[INFO] L - 100.00%
[INFO] D - 99.79%
```

Fig. 5.(z)

5.5 PERFORMANCE ANALYSIS

ACCURACY CURVE

The gap between training and validation accuracy is a clear indication of overfitting. The larger the gap, the higher the overfitting.

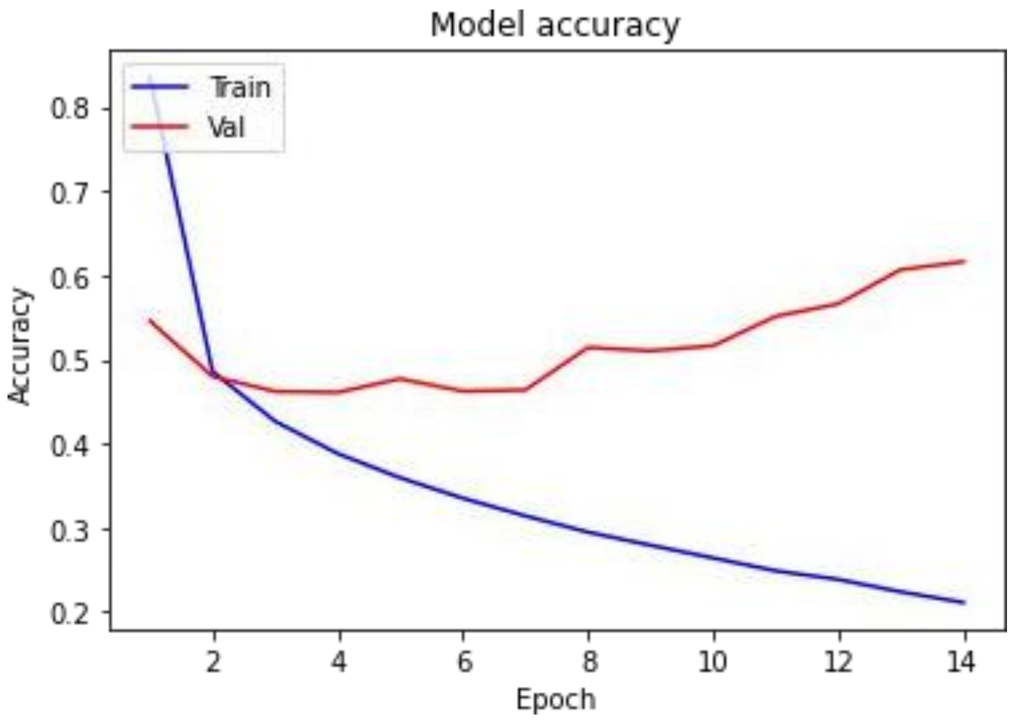


Graph 5.(a)

<u>LOSS CURVE</u>

It gives us a snapshot of the training process and the direction in which the network learns.

During an epoch, the loss function is calculated across every data item, and it is guaranteed to give the quantitative loss measure at the given epoch.



Graph 5.(b)

CHAPTER 6

PROJECT OUTCOME AND APPLICABILITY

6.1 OUTLINE

A handwriting recognition model has a large number of practical applications, in different fields like healthcare, education, banks, digital libraries, etc. The use of CNN and CapsNet helps in enhancing the text on the written documents which can then be compiled and understood easily. This helps in reducing manual labour.

6.2 PROJECT APPLICABILITY ON REAL WORLD APPLICATIONS

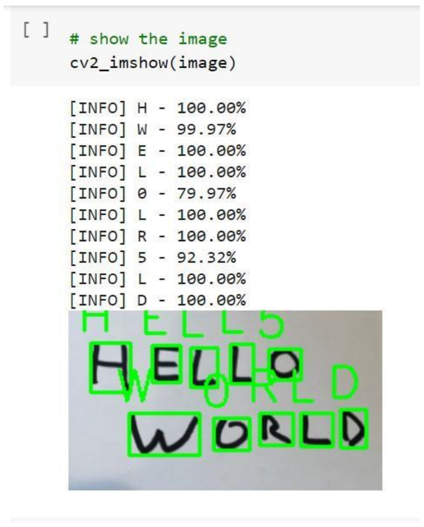
- In banks for automatic check processing
- Automatic number plate recognition
- National ID number recognition system
- To process a large number of documents in industries like education, health, legal and finance.
- In digital libraries, allowing the entry of image textual information into computers.
- Reading of forms
- To help students, lecturers, staff, instructors maintain notes, which can be easily misplaced or lost.
- Saving the trouble of storing numerous books and to relieve the task of maintaining the book's condition.
- In reading medical prescriptions in a medical store.

6.3 SIGNIFICANT PROJECT OUTCOMES

We create an accurate handwriting recognition model that can interpret alphabets, words, digits, and characters effectively.

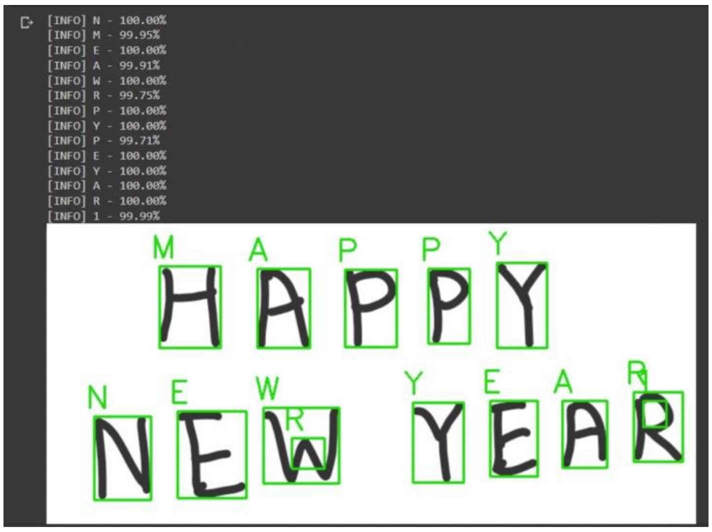
The use of CapsNet, increases the level of accuracy as there are multiple layers inside a single layer.

For example,

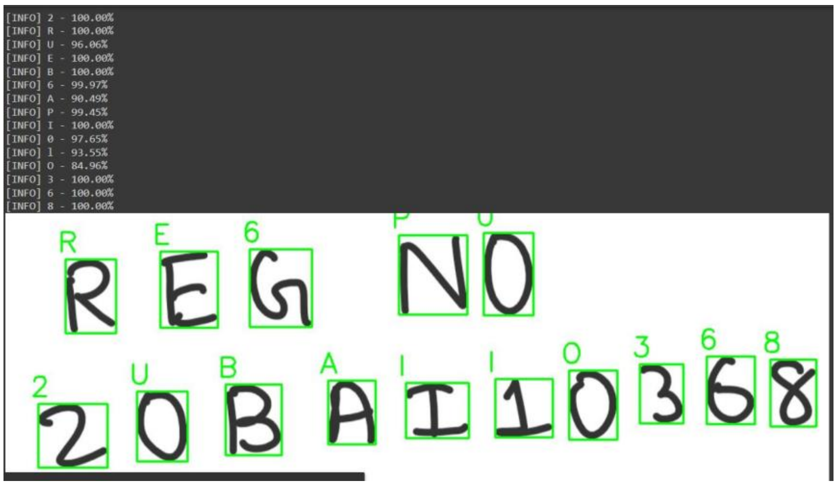


```
# show the image
cv2_imshow(image)
```

```
[INFO] H - 100.00%
[INFO] W - 99.97%
[INFO] E - 100.00%
[INFO] L - 100.00%
[INFO] 0 - 79.97%
[INFO] L - 100.00%
[INFO] R - 100.00%
[INFO] 5 - 92.32%
[INFO] L - 100.00%
[INFO] D - 100.00%
```



```
[INFO] N - 100.00%
[INFO] M - 99.95%
[INFO] E - 100.00%
[INFO] A - 99.91%
[INFO] W - 100.00%
[INFO] R - 99.75%
[INFO] P - 100.00%
[INFO] Y - 100.00%
[INFO] P - 99.71%
[INFO] E - 100.00%
[INFO] Y - 100.00%
[INFO] A - 100.00%
[INFO] R - 100.00%
[INFO] 1 - 99.99%
```



```
[INFO] 2 - 100.00%
[INFO] R - 100.00%
[INFO] U - 96.06%
[INFO] E - 100.00%
[INFO] B - 100.00%
[INFO] 6 - 99.97%
[INFO] A - 90.49%
[INFO] P - 99.45%
[INFO] I - 100.00%
[INFO] 0 - 97.65%
[INFO] 1 - 93.55%
[INFO] O - 84.96%
[INFO] 3 - 100.00%
[INFO] 6 - 100.00%
[INFO] 8 - 100.00%
```

Fig. 6. (a)

6.4 KEY IMPLEMENTATIONS OUTLINES OF THE SYSTEM

- IN BANKS:

  The main aim of using a handwriting recognition system is to make banking operations easier and error free. The particular model aims to recognize the written character on cash deposit, withdrawal, or other transactions. It also helps with automatic banking deposit number recognition, which recognizes the handwritten account number and the amount number on the cash deposit slips, thus automating the cash deposit process at the bank counter.

- IN HEALTHCARE SECTORS

  Medical companies take handwritten prescriptions and retrieve text from it. In order to automate this task, we can use a handwriting recognition model, which reduces manual labour. This model can take images as input, read the text image, and convert it into a digital test, using CNN.

- IN AUTOMATIC NUMBER – PLATE RECOGNITION

  Automatic number plate recognition is used to read vehicle registration plates and create vehicle location data. In this case, the image is first detected using a series of image manipulation, then normalized and enhanced. Using the model, the alphanumerics on the image can then be extracted.

- IN READING OF FORMS

  Since the model is efficient in grasping different styles of handwritings, it can be used in the reading of forms. The textual image can be efficiently converted to a machine encoded language that can then be compiled and read easily.

CHAPTER 7

CONCLUSIONS AND RECOMMENDATION

7.1 CONSTRAINTS OF THE SYSTEM

1. This system does not give the perfect accuracy under any of the models.

2. Sometimes what might look legible to us might not be understood by the system and so it may give some garbage value as the output.

3. The accuracy for upper case alphabets is particularly greater than that for the lower case alphabets.

4. Unavailability of proper datasets imposed problems during the project and will also raise issues if we wish to use a dataset of some other Language.

5. All kinds of handwriting cannot be detected.

6. The research on CapsNet is still going on, so it is difficult to implement it perfectly.

7. Our systems cannot support the processing of the CapsNet model

7.2 FUTURE ENHANCEMENTS

1. Create a model with accuracy greater than 98%.

2. Train the model for more data.

3. Firstly work on recognizing a sentence and then recognizing a whole document.

4. We can deploy the model and create a website for the bigger use.

5. We can create a similar model for vernacular languages.

7.3 INFERENCE

This project gives us a fair idea about the processes involved in handwriting recognition. Although the project still fails to give a 100% accuracy. There is not just one model that can give us an accurate result. Different models have different pros and cons.

REFERENCES

1.https://www.academia.edu/62493423/Contour_Vs_Non_Contour_based_Word_Segmentation_from_Handwritten_Text_Lines_an_Experimental_Analysis

2.https://www.ijert.org/research/recognition-of-handwritten-digits-using-machine-learning-techniques-IJERTV6IS050456.pdf

3.https://www.ijedr.org/papers/IJEDR1704192.pdf

4.https://www.researchgate.net/publication/326408524_Handwritten_Digit_Recognition_Using_Machine_Learning_Algorithms

5.https://thesai.org/Downloads/Volume11No7/Paper_19-Handwriting_Recognition_using_Artificial_Intelligence.pdf

6.https://www.researchgate.net/publication/2286971_An_Overview_of_Handwriting_Recognition

7.https://www.irjet.net/archives/V8/i4/IRJET-V8I4249.pdf

8.https://data-flair.training/blogs/handwritten-character-recognition-neural-network/

9.https://www.sciencedirect.com/science/article/pii/S1877050920307596

10. http://ceur-ws.org/Vol-2870/paper98.pdf#:~:text=The%20handwritten%20text%20recognitio

n%20software%20system%20is%20developed,text%20recognition%20system%20will%20be%20artificial%20neural%20networks.

11. https://www.pyimagesearch.com/2020/08/24/ocr-handwriting-recognition-with-opencv-keras-and-tensorflow/

12. https://exeley.com/international_journal_advanced_network_monitoring_controls/pdf/10.21307/ijanmc-2020-031

13. https://www.ijert.org/handwritten-digit-recognition-for-banking-system#:~:text=Abstract%20%E2%80%93%20The%20aim%20of%20a%20handwriting%20digit,and%20make%20banking%20operations%20easier%20and%20error%20free.

14. https://www.sciencedirect.com/topics/computer-science/handwriting-recognition

15. https://medium.com/@himanshubeniwal/handwritten-digit-recognition-using-machine-learning-ad30562a9b64

16. https://www.geeksforgeeks.org/handwritten-digit-recognition-using-neural-network/#:~:text=Handwritten%20digit%20recognition%20using%20MNIST%20dataset%20is%20a,basically%20detects%20the%20scanned%20images%20of%20handwritten%20digits.

17. https://media.springernature.com/original/springer-static/image/chp%3A10.1007%2F978-3-030-30487-4_15/MediaObjects/489415_1_En_15_Fig1_HTML.png

18. https://www.researchgate.net/profile/Dulani-Meedeniya/publication/348542023_Capsule_Networks_for_Character_Recognition_in_Low_Resource_Languages/links/601d03a0a6fdcc37a802da96/Capsule-Networks-for-Character-Recognition-in-Low-Resource-Languages.pdf?origin=publication_detail

19. https://towardsdatascience.com/understanding-cnn-convolutional-neural-network-69fd626ee7d4#:~:text=CNN%20is%20a%20type%20of,features%20automatically%20for%20better%20classification.

20. https://www.analyticsvidhya.com/blog/2018/04/essentials-of-deep-learning-getting-to-know-capsulenets/