**HANDWRITING RECOGNITION AND WRITER IDENTIFICATION**

**ON TEXT INDEPENDENT DATA**

*Report submitted to the SASTRA Deemed to be University as the requirement for the course*

BCSCCS708**: MINI PROJECT**

*Submitted by*

**NAG ASHISH S V**

**(Reg. No.: 221003064, B. Tech, CSE B)**

**RANGA SATYA VISWA PAVAN**

**(Reg. No.: 221003080, B. Tech, CSE B)**

**PRAMODH SAIRAM P V**

**(Reg. No.: 221003117, B. Tech, CSE B)**

# December 2020



SRINIVASA RAMANUJAN CENTRE

## KUMBAKONAM, TAMIL NADU, INDIA – 612 001



## SRINIVASA RAMANUJAN CENTRE

## KUMBAKONAM – 612 001

**Bonafide Certificate**

This is to certify that the report titled “**Handwriting Recognition and Writer Identification on Text Independent Data**” submitted as a requirement for the course, BCSCCS708**: MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. Nag Ashish S V (**Reg. No.: **221003064,** B. Tech CSE B**), Mr. Ranga Satya Viswa Pavan(**Reg. No.: **221003080,** B. Tech CSE B**), Mr. Pramodh Sairam P V(**Reg. No.: **221003117,** B. Tech CSE B**)** during the academic year 2020-20, in the Srinivasa Ramanujan Centre, under my supervision.

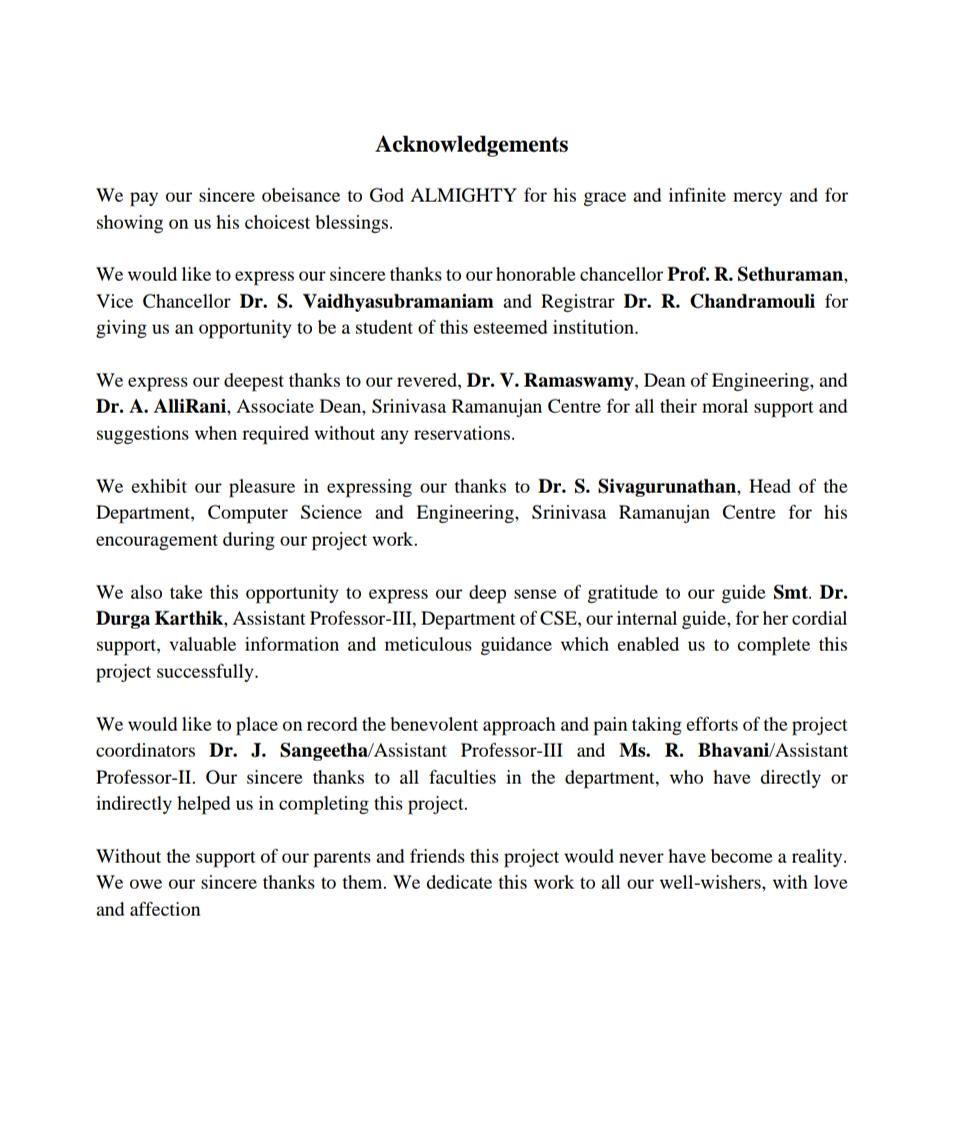
**Signature of Project Supervisor : **

**Name with Affiliation :** Dr.Durga Karthik/AP III/CSE/SRC/SASTRA

**Date :** 28.12.2020

Mini Project *Viva voc*e held on \_\_\_\_\_31.12.2020\_\_\_\_\_\_\_\_\_\_\_

**Examiner 1 Examiner 2**



**Table of Contents**

**Title** **Page No**.

Bonafide Certificate ii

Acknowledgement iii

List of Figures v

List of Tables v

Abbreviations v

Abstract vi

1. Summary of the base paper 1
2. Merits and Demerits of the base paper 8
3. Source Code 10
4. Snapshots 24
5. Conclusion and Future Plans 28
6. References 29
7. Appendex 30

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No** | **Title** | **Page No.** |
| 1.1 | Architecture Diagram for the project | 2 |
| 1.2 | AlexNet Architecture | 5 |
| 2.1 | Existing works on writer identification | 8 |
| 4.1 | Input image from writer | 24 |
| 4.2 | Grayscale image | 24 |
| 4.3 | Edged Image | 24 |
| 4.4 | Anticlockwise rotated image | 24 |
| 4.5 | Clockwise rotated image | 24 |
| 4.6 | Randomly Distorted Image | 24 |
| 4.7 | Randomly Distorted Image | 24 |
| 4.8 | Random crops of 224\*224 | 25 |
| 4.9 | Training and Testing dataset | 25 |
| 4.10 | Training process | 26 |
| 4.11 | Server output | 26 |
| 4.12 | Result from Backend API | 27 |
| 4.13 | Frontend UI | 27 |

**List of Tables**

|  |  |  |
| --- | --- | --- |
| **Table No** | **Title** | **Page No.** |
| 1.1 | Architecture Diagram of CNN followed in project | 3 |

## Abbreviations

|  |  |
| --- | --- |
| CNN | Convolutional Neural Network |
| ANN | Artificial Neural Network |
| SIFT | Scale Invariant Feature Transform |
| LBP | Local Border Patterns |
| SGD | Stochastic Gradient Descent |
| API | Application Programming Interface |
| HTTP | Hypertext Transfer Protocol |

**ABSTRACT**

Identification of individuals using their handwriting is one of the stimulating problems. It can be used in various number of fields like security and criminological analysis, antique documents and literature analysis, forgery and signature analysis etc. Handwriting has a crucial character in demonstration of identity and cultured traits of an individual. It is the key distinctiveness of an individual. Methodologies which are based on deep learning are verified as the versatile techniques for extracting features from huge volumes of diverse data and gives solid and accurate predictions of the underlying patterns than traditional methodologies. Automatic handwriting recognition system aids in identifying whether the stated handwriting is accurately matched and allocated to the original writer of that handwriting.

There are two modes of data capturing in any kind of writer identification, they are online and offline. Spatial coordinate values are considered mainly in the former case whereas the temporal details are considered in latter case. With respect to the textual content, offline writer identification can be categorized as text dependent and text independent. Text-dependent procedures highly emphasize on context and semantics in the handwritten image, so it needs an image having fixed text content and evaluates the resemblance of the input image with already listed prototypes for this purpose. Contrarily, text independent handwriting recognition emphasizes on the images that do not hang on the text content which is fixed.

Since the convolutional neural networks and their state of art architectures are very powerful in extracting deep features and their huge success in the fields of computer vision made us to implement the proposed model of writer identification system. It is done in three stages. The first stage is Image pre-processing where various filters are applied on images to remove noise. The second stage is feature extraction using AlexNet based CNN architecture where the patterns are learnt and deep features are extracted in order to accurately classify different handwriting styles and the final stage is writer classification using Artificial Neural Networks (ANN) followed by evaluation of the model.

**KEYWORDS**: Handwriting Recognition, Writer Identification, AlexNet, Convolutional Neural Networks, Artificial Neural Networks, Image Augmentation.

## CHAPTER 1

## SUMMARY OF BASE PAPER

Base paper title: Automatic Visual Features for Writer Identification: A Deep Learning Approach by: Arshia Rehman- Higher Education Department, GGPGC, Pakistan, Saeeda Naz-The University of Sydney, Sydney, NSW 2006, Australia, Muhammad Imran Razzak and Ibrahim A Hameed-Department of ICT and Natural Sciences, Norwegian University of Science and Technology, 6009 Alesund, Norway

Journal Name: Deep learning,

Published on: 21 January 2019.

Published in: IEEE Access (Volume 7)

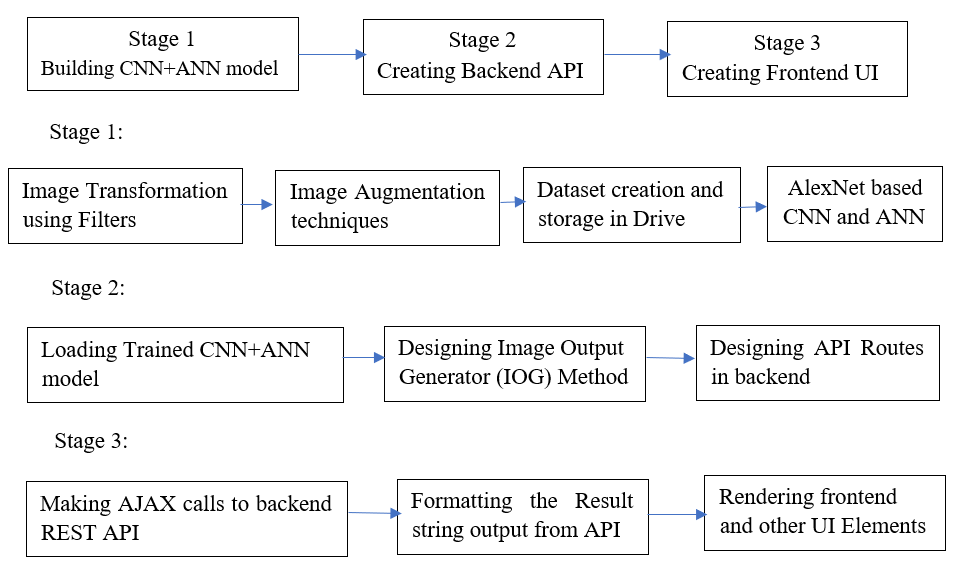
**1.1 Introduction:**

Automatic handwriting recognition and writer identification system helps in determining the valid writer of a particular handwritten text image among a large number of writers whose handwriting has been already trained. Many research scholars had shown interest in this domain because of its large number of applications like security and criminological analysis, antique documents and literature analysis, etc. The traditional approaches extract features from handwritings using hand designed and manual calculations. They used techniques like scale invariant feature transforms, local binary patterns, wavelet transforms etc, where the final feature vectors are extracted from local patches of handwriting images and constructed models like bag of words etc. With the advent of deep learning algorithms like CNNs, ANNs etc, the feature extraction work has become much easier. Since the convolutional neural networks and their state of art architectures like AlexNet, VGG, GoogleNet etc are very powerful in extracting deep features and their huge success in the fields of computer vision made several research scholars to employ CNN techniques for writer identification.

This paper presents a deep learning based automatic feature extraction model using AlexNet CNN architecture, which is a state of art architecture model trained on ImageNet database. In this project the pre-processed handwriting images of 10 different writers are trained on CNN architecture inspired from AlexNet and classification is done using Artificial Neural Networks.

**Convolutional Neural Networks:** A Convolutional Neural Network is a popular deep learning algorithm that takes an input image, process pixel data, undergoes learning by assigning weights and biases through optimizers and large number of back propagations, to extract features from the image and becomes capable of distinguishing each another. The CNN architecture is very similar to the arrangement pattern of neurons present in the brains of human beings and got inspired from organization of the visual cortex. Receptive field is the visual area where discrete neurons are restricted to receive stimuli and responds to them. This visual area is a collection of such fields that gets overlapped.

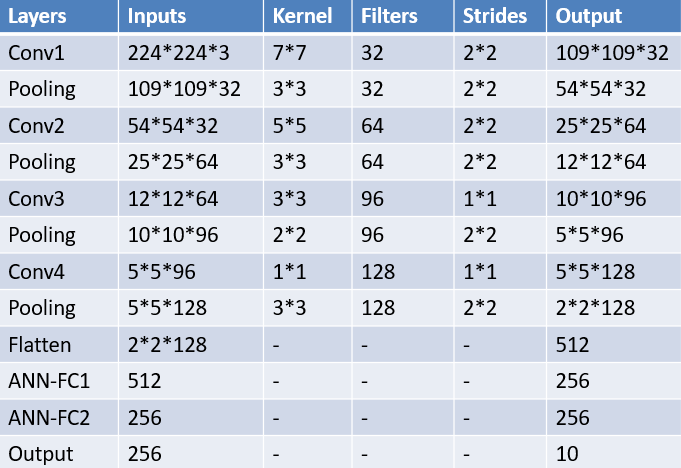
**1.2 Architecture Diagram:**



*Fig 1.1 Architecture of the project*

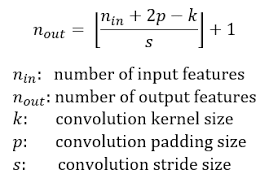
The above image fig 1.1 describes about the architecture of the project. The entire architecture is divided into three stages. Each stage is implemented by a sequence of substages referred as pipelines. Stage 1 is all about constructing the classification model, where the images are pre-processed, augmented and trained on AlexNet based CNN architecture. It comprises of 4 pipelines. The next two stages are for constructing a web application using this trained model. Stage 2 is for designing the backend API of the web application, which is a sequence of 3 pipelines and Stage 3 is for designing the frontend UI of the web application. It is executed as a sequence of 3 pipelines

**AlexNet Based CNN architecture implemented for training the Handwriting Images**

****

*Table 1.1 CNN Architecture*

The above table 1.1 shows the architecture of the convolutional neural network. There are 4 convolution layers, each layer followed by a pooling layer. A flatten layer is used to extract the feature vector and it is sent to two sequential fully connected layers. Finally, there is an output layer having 10 nodes. Each layer has its own configurations of various parameters like kernel size, no of filters, strides size, padding used etc. The formula for the output dimension is shown below.



*Formula for calculating output dimension*

In every convolution layer, the kernel is convolved over the input matrix, where the matrix values get multiplied with the values of filter and gets added. There can be any number of such filters. Stride is the shift length that is made by the kernel. In the entire architecture, we kept valid padding i.e. we didn’t add extra padding to any matrix in the network. Hence p=0 in the entire network. The handwriting image is the input for the network, it is a matrix of size 224\*224\*3. As per the above formula the output matrix dimension should be 109\*109\*32 i.e. ((224+0-7)/2) +1= (217/2) +1= floor (108.5) +1= 108+1 =109. There are 32 filters used, hence output is 109\*109\*32. Pooling layer is used to downscale the image, by extracting the maximum value from each submatrix of kernel size. Flatten layer converts 2d matrix into a 1d feature vector which is sent to a dense network called as Fully connected layers. Finally, the last FC layer is connected to output layer having 10 nodes which represents 10 different writers.

**1.3 PROPOSED METHODOLOGY:**

This paper proposed a model for writer identification, using Convolutional Neural Networks. CNNs are mainly used for their abilities of features extraction. The main feature that the CNN algorithm uses in this writer identification system is ‘Raw Pixel’. Apart from the pixel value, the features like i) Spacing between letters, words, sentences and lines. For ex: cursive handwriting letter space is zero ii) Thickness of letter edges, since pressure applied by every writer while writing will be different, this leads to different thickness in letter edges. iii) Curvature along with depth and height of the letters etc.

The extracted feature vector is sent to Artificial Neural Networks or Fully Connected layers for performing handwriting classification. The obtained training and testing accuracy are considered as the evaluation parameter for the model. The entire workflow can be briefly divided into 3 stages, each stage has 3-4 sub stages called as pipelines.

**Stage 1: Development of the Classification Model**

This stage is mainly for building the classification model and its evaluation. It is broadly divided into 4 pipelines. Each pipeline has its own well-defined task.

**Pipeline 1**: In this stage the main objective is to remove the noise and variations in background lighting, since the images are differently captured by every writer. This is done using Gaussian blur filter and finally extracting the edges alone from the handwriting image. This is done using Canny Edge Detector.

Gaussian Blur filtering is an operation in which a Gaussian filter instead of a box filter is convolved over an image. It is a low-pass filter which eliminates the high-frequency components and reduces the unwanted noise in the image.

The Canny edge detector is an edge detection operation that uses multi-stage algorithm to detect a extensive range of edges in images.

**Pipeline 2**: The main objective of this stage is Image Augmentation. Image augmentation is a procedure which is used to theatrically enlarge the size of a training and testing dataset by creating altered variants of images in the dataset. There are various augmentation techniques like i) Rotation ii) Scaling iii) Random distortion iv) Flipping v) Zooming vi) Horizontal and Vertical shifting, etc

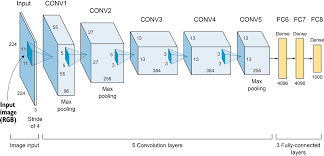
To increase the image samples, for training the model efficiently, the paper proposed two augmentation techniques, rotation and random distortion followed by random patching of images to obtain a size of 224\*224. The following are the operations performed on data.

1)50 images by rotation with 12 degrees clockwise and 14 degrees anti clockwise.  
2)50 images by rotation with 10 degrees clockwise and 8 degrees anti clockwise.  
3)50 images by distortion with grid width 10 units, grid height 10 units and magnitude 12 units.  
4)50 images by distortion with grid width 4 units, grid height 4 units and magnitude 8 units.  
5)random slices of 224\*224 are extracted for every image.

**Pipeline 3:** Creation of appropriate training dataset and testing datasets is the main objective.

In this project, we used 10 writers handwriting, i.e. a total of 10000 images, 1000 images for every writer, of size 224\*224. For every writer 800 images are used for training the model and 200 images are used for testing or validating. So, a total of 8000 images are used for training and a total of 2000 images are used for testing. The created datasets are uploaded in Google Drive, which is considered as a database for this project. Uploading in google drive is efficient for accessing data while working in google colab.

**Pipeline 4:** Constructing a Convolutional Neural Network model based on AlexNet architecture is essential for training the handwriting images for classification. The entire neural network is designed using TensorFlow and keras library. The network architecture is shown in the figure 1.2 The network is constructed by referencing AlexNet State of Art CNN architecture. It has 4 convolution layers followed by 4 pooling layers, each after every convolution layer. Feature vector is obtained from flatten layer and sent to two fully connected layers. Other configurations of the network are described below.



*Fig 1.2 AlexNet CNN architecture*

Activation Functions used:-

In convolution layer:- Relu

In Output layer:- Softmax

Optimizer used: - Stochastic Gradient Descent (SGD)

Loss Function used: - categorical\_crossentropy

No of Epochs: - 20

Evaluation Metric used: - accuracy

To reduce overfitting, Batch Normalization and Dropout of 40% are used in CNN and ANN layers respectively.

**Stage 2:** **Creating backend REST API of web application**

The main objective of this stage is to build a REST API for the developed CNN model using flask. It consists of 3 pipelines.

**Pipeline 1:** - The trained AlexNet CNN model is stored in disk and is loaded whenever necessary. So, the main objective of this stage is to extract the Handwriting classification model from the disk and gets loaded with all the pretrained weights and can be used for further testing.

**Pipeline 2:** - Designing an Image Output Generator (IOG) method, which gives the classification results in the form of 20 votes, which were distributed among the possible writers, for an uploaded handwriting image of any writer. For ex: for any image if the model is 100% confident on a particular writer’s handwriting, all the 20 votes are given to that writer, else it gets divided.

**Pipeline 3:** - Designing a Flask based API having 2 routes

i).‘/’which is a GETroute for displaying home page of the app

ii).‘/predict’which is a POSTroute, that calls predict function and sends the result string to the frontend.

**Stage 3**: **Creating frontend user interface of web application**

The main objective of this stage is to design a simple user interface for testing the model by uploading handwriting images. It consists of three pipelines.

**Pipeline 1:** - Creating UI elements for uploading handwriting and making AJAX calls to the REST API to obtain the result for the uploaded handwriting.

**Pipeline 2:** - Formatting of the result string is done in this pipeline i.e. the writer name which got a greater number of votes out of 20 are considered as 1st priority and displayed appropriately.

**Pipeline 3: -** Designing other UI elements of the front end and rendering them appropriately for a better user experience.

**1.4 CORRECTNESS:**

To prove correctness of any deep learning algorithm, there are certain metrics like accuracy, precision, recall, loss, validation loss, validation accuracy etc to evaluate the model and which in turn prove its correctness. Various metrics should be considered in various situations. In image classification using CNN mainly in every epoch we will be considering parameters like accuracy, validation accuracy, loss and validation loss.

**Accuracy:** - Accuracy is considered as a significant metric in any classification project. It compares the classified image with another data source that is considered to be accurate or ground truth data. Through field work and experimentation ground truth can be collected; however, this is highly expensive and more time taking. Briefly it can be considered as how much correct a classified result is, with respect to correct result. Higher the accuracy score, better the learning and performance of the model

**Validation Accuracy:** - It is nothing but accuracy score obtained on validation dataset. It shows how well the model is behaving on the unknown data after getting trained. Higher the validation accuracy, better the classification on unknown images.

**Loss:** - In order to optimize any algorithm, the function that is used to figure out a solution, is often considered as an objective function. The main aim is to maximize or minimize this objective function, which means that the solution which we are finding should have either highest score or lowest score respectively. Naturally, in neural networks, the error must be as minimum as possible. This objective function is considered as loss function or cost function and the value obtained by this loss function is simply called as loss.

**Validation Loss:** - Loss obtained on validation data is considered as validation loss. For a better generalized model, the loss and validation loss should be as minimum as possible. Higher validation loss and lower validation accuracy is a result of overfitting.

While training the CNN model on certain epochs, the following trends can be seen on these values.

1. Validation loss slowly starts to grow and validation accuracy slowly starts falling. This means model is stuffing values and not learning
2. Validation loss slowly starts to grow and validation accuracy also slowly raises. This must be a situation of overfitting or miscellaneous probability values in scenarios where SoftMax activation function is being used in the output layer
3. Validation loss starts decreasing and validation accuracy starts increasing. This is also acceptable which means that the model built is learning well and working fine.

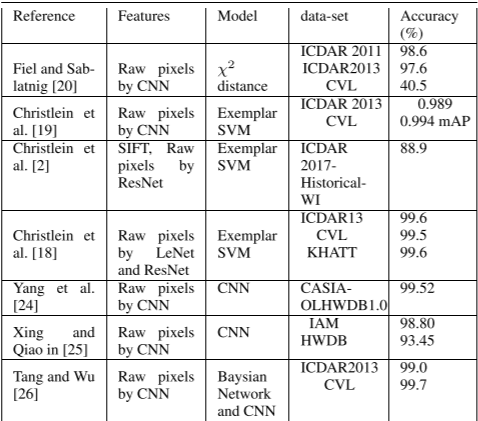
While training our model we observed the 3rd trend, i.e. validation loss is decreasing and validation accuracy is increasing. This shows that the CNN model is generalized and not overfitting. In the 1st epoch, the accuracy is 0.3341, validation accuracy is 0.1870, loss is 2.0037 and validation loss is 3.4596, while in the final epoch i.e. 20th epoch, the accuracy is 0.9919, validation accuracy is 0.9900, loss is 0.0487 and validation loss is 0.0421. With these metrics we can say that the model is not overfitting and it is a highly generalized model. This proves the correctness of our CNN based Handwriting classification model.

**CHAPTER 2**

## MERITS AND DEMERITS OF THE BASE PAPER

**2.1 EXISTING METHODOLOGIES:**

The traditional methodologies of offline text-independent handwriting identification have been classified into two types based on the extracted features. The first approach is based on manually designed features and the second approach is based on deep learning-based feature extraction approaches [1][8][6]. Some hand-designed features such as Scale-Invariant-Feature-Transforms [7], Local Binary Patterns [3], etc were extracted from the local patches of the handwriting images and models like bag of words [4] has been constructed to represent the obtained final feature vectors. To extract features from handwriting images for writer identification, filters like Gabor filters and local binary patterns were used. Other than this Most of the works are based on traditional methods like finding chi^2 distance between letters, using wavelet transformation which is computationally intensive, and other methods like grapheme generation which is time consuming. Many other researchers like Christlein et al. used CNN algorithms [2][6] but followed a technique of extracting features from local structures [5] which resulted in high feature loss and low accurate models. Tang and Wu used Bayesian networks and CNN algorithm on ICDAR 2013 dataset [9].



*Fig 2.1 Existing Works on Writer Identification*

These methodologies didn’t consider, custom (real time) writers’ handwritings rather used benchmark datasets handwritings as shown above. They also didn’t used any modern State of art CNN architecture models like ResNet, VGG, AlexNet, GoogleNet etc which are highly accurate and powerful.

**2.2 PROPOSED METHOD: MERITS &DEMERITS**

In this paper, we follow a technique of extracting features from Global structuresthan local structures, due to which the feature loss is minimized and performance gets improved. Apart from it we used a CNN architecture inspired from the famous State of Art CNN architecture, AlexNet.

The entire project is carried out on custom writers than using the existing old benchmark datasets like QUWI, ICDAR-13 etc., on which more research has already been done. The proposed model performs better than existing models in the areas of Image Augmentation, efficient ways of storing the images, better CNN architecture with a very high accuracy etc.

**Analysis on Image Augmentation techniques:**

The proposed scheme used basic as well as latest image augmentation techniques like rotation and random distortions respectively. With this, the final datasets have high variant images which is very useful for avoiding overfitting and feature loss. With the technique of random distortion in the line alignments, we can easily relate it with the daily life scenario where writers always won’t write uniformly. We followed random cropping technique over sliding window technique in order to ensure the capturing of as many features as possible. With sliding window technique there is loss in features like word gap, curvature and line height etc. It also avoids overfitting of the model.

**Architecture Analysis:**

The architecture is built by taking reference of AlexNet CNN architecture. Since the handwriting images that we are using in this paper are not a part of the ImageNet dataset, the actual weights used in AlexNet architecture didn’t gave a valid accuracy. Hence Transfer learning techniques cannot be applied directly. Instead, tweaking the configurations like number of filters, strides etc in each layer by keeping the same architecture worked well for us. Usage of Stochastic Gradient Descent optimizer over other optimizers like Adam which was used in other existing works gave a relatively less loss. The number of convolution layers and pooling layers can still be optimized. The total trainable parameters can still be reduced.

**CHAPTER 3**

## SOURCE CODE

**3.1 HandwritingClassifierModel.py**

Code for creating a handwriting classifier model on 10 writers handwriting

**Pipeline 1:**

from google.colab import drive

drive.mount('/content/gdrive')

!pip install mapper

import cv2

import numpy as np

import mapper

import matplotlib.pyplot as plt

from PIL import Image

def show\_image(image\_object,name):

  plt.figure(figsize=(14, 7))

  plt.title(name)

  plt.imshow(image\_object)

edged\_image=""

def preprocess1(image\_path):

  image=cv2.imread(image\_path)   #read in the image

  #image=cv2.resize(image,(1300,800))

 #resizing because opencv does not work well with bigger images

  show\_image(image,"Input\_Image")

  gray=cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY)  #RGB To Gray Scale

  show\_image(gray,"Grayscale\_Image")

  blurred=cv2.GaussianBlur(gray,(5,5),0)

#(5,5) is the kernel size and 0 is sigma that determines the amount of blur

  show\_image(blurred,"Blurred\_Image")

  edged=cv2.Canny(blurred,30,50)  #30 MinThreshold and 50 is the MaxThreshold

  show\_image(edged,"Edged\_Image")

  new\_image\_name=input("Enter name for the processed image to get saved..")

  cv2.imwrite(new\_image\_name,edged)

  global edged\_image

  edged\_image=new\_image\_name

preprocess1("/content/ashishhw.jpeg")

**Pipeline 2:**

!pip install Augmentor

#ZIPPING FOLDERS AND DOWNLOAD THE ZIP FILE AUTOMATICALLY

import shutil

from google.colab import files

def make\_zipfolder(zipfoldername,folder\_path):

  shutil.make\_archive(zipfoldername, 'zip', folder\_path)

  files.download(zipfoldername+".zip")

# DELETING A FOLDER WITH CONTENT DIRECTLY

def delete\_folder(path):

  shutil.rmtree(path)

#ZIPPER CELL

#make\_zipfolder("nags\_augmented\_imgs",output\_directory)

make\_zipfolder("ashishcroppedimgs","/content/ashishcroppedimgs")  #give anyname that u like

#delete\_folder('/content/origimgs')

#FOR DISPLAYING SINGLE IMAGE

def view\_image(path):

  print("path=",path)

  img = cv2.imread(path)

  img\_cvt=cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

  plt.imshow(img\_cvt)

  plt.show()

view\_image("/content/ashishedged.jpeg")

#FOR SHOWING ALL THE IMAGES IN A FOLDER

import os

directory = "/content/origimgs"

output\_directory="/content/origimgs/output"

def display\_images(directory,cnt=50):

no=0

for filename in os.listdir(directory):

print(filename)

if(filename==".ipynb\_checkpoints"):

continue

view\_image(directory+"/"+filename)

no=no+1

if(no==cnt):

break

#RENAMING FILES IN A FOLDER

def rename\_files(dir\_path,operation,execs):

count=1

for filename in os.listdir(dir\_path):

if(filename[:8]=='origimgs'):

dest=operation+"-"+str(execs)+"-"+str(count)+".jpeg"

os.rename(dir\_path+"/"+filename,dir\_path+"/"+dest)

count+=1

import Augmentor

def init\_augmentor(flag):

if(flag):

os.makedirs('/content/origimgs')

shutil.move(edged\_image, '/content/origimgs/')

p = Augmentor.Pipeline("/content/origimgs/")

return p

def rotate(no\_of\_images=10,lr=10,rr=10,execs=1):

p=init\_augmentor(0)

p.rotate(probability=0.9, max\_left\_rotation=lr, max\_right\_rotation=rr)

p.sample(no\_of\_images)

rename\_files(output\_directory,"rotate",execs)

def random\_distortion(no\_of\_images=10,gridwdth=4,gridht=4,mag=15,execs=1):

p=init\_augmentor(0)

p.random\_distortion(probability=1, grid\_width=gridwdth, grid\_height=gridht, magnitude=mag)

p.sample(no\_of\_images)

rename\_files(output\_directory,"distortion",execs)

def zoom(no\_of\_images=10,execs=1):

p=init\_augmentor(0)

p.zoom\_random(1, percentage\_area=0.5)

p.sample(no\_of\_images)

rename\_files(output\_directory,"zoomed",execs)

init\_augmentor(1)

rotate(50,14,12,1)

rotate(50,8,10,2)

random\_distortion(50,10,10,12,1)

random\_distortion(50,4,4,18,2)

#RANDOM SILCES OF 224\*224

def get\_random\_crop(image, crop\_height, crop\_width):

max\_x = image.shape[1] - crop\_width

max\_y = image.shape[0] - crop\_height

x = np.random.randint(0, max\_x)

y = np.random.randint(0, max\_y)

crop = image[y: y + crop\_height, x: x + crop\_width]

return crop

path3="/content/origimgs/output/" #path of folders having 200 images

def sliding\_window(no\_of\_crops,foldername,htsize=224,wdsize=224):

cnt=1

for imagename in os.listdir(path3):

if(imagename=='.ipynb\_checkpoints'):

continue

example\_img=path3+imagename

iimg = cv2.imread(example\_img)

for i in range(no\_of\_crops):

iimg2=get\_random\_crop(iimg,htsize,wdsize)

cv2.imwrite('/content/'+foldername+'/cropimg'+str(cnt)+'.jpeg', iimg2)

#cv2.imwrite('/content/croppedimgs68/cropimg'+str(cnt)+'.jpeg', iimg2)

cnt=cnt+1

cropped\_folder=input("Enter a name for the empty folder to save 224\*224 slices")

os.makedirs("/content/"+cropped\_folder)

sliding\_window(5,cropped\_folder)

#check folder then go above upto zipper cell..zip this folder and download make\_zipfolder(cropped\_folder,"/content/"+cropped\_folder)

#UNZIPPING A ZIPPED FILE INTO A FOLDER

!unzip augimgs1.zip -d augmented\_imgs1

**Pipeline 3:**

#os.makedirs("/content/gdrive/My Drive/datasetedgedhws/datasetedged/train\_set/aswadHW")

#os.makedirs("/content/gdrive/My Drive/datasetedgedhws/datasetedged/test\_set/aswadHW")

def upload\_to\_drive(foldername):

testdir="/content/gdrive/My Drive/datasetedgedhws/datasetedged/test\_set/"+foldername

traindir="/content/gdrive/My Drive/datasetedgedhws/datasetedged/train\_set/"+foldername

os.makedirs(traindir)

os.makedirs(testdir)

for cnt in range(1,1001):

if(cnt<=800):

#print("/content/"+cropped\_folder+"/cropimg"+str(cnt)+".jpeg")

shutil.move("/content/"+cropped\_folder+"/cropimg"+str(cnt)+".jpeg", traindir)

else:

shutil.move("/content/"+cropped\_folder+"/cropimg"+str(cnt)+".jpeg", testdir)

foldernameindrive=input("Enter the name of the folder to be created in drive")

upload\_to\_drive(foldernameindrive)

delete\_folder('/content/'+cropped\_folder)

delete\_folder('/content/origimgs')

**Pipeline 4:**

import tensorflow as tf

import tensorflow.keras

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Activation, Dropout, Flatten, Conv2D, MaxPooling2D,BatchNormalization

from keras.preprocessing.image import ImageDataGenerator

test\_datagen = ImageDataGenerator()

test\_set = test\_datagen.flow\_from\_directory('/content/local\_datasetedged/datasetedged/test\_set',class\_mode='categorical',target\_size = (224, 224), batch\_size=96)

train\_datagen=ImageDataGenerator()

train\_set=train\_datagen.flow\_from\_directory('/content/local\_datasetedged/datasetedged/train\_set',class\_mode='categorical',target\_size=(224,224), batch\_size=96)

model4 = Sequential()

# 1st Convolutional Layer

model4.add(Conv2D(filters=32, input\_shape=(224,224,3), kernel\_size=(7,7),strides=(2,2), padding='valid'))

model4.add(Activation('relu'))

# Pooling

model4.add(MaxPooling2D(pool\_size=(3,3), strides=(2,2), padding='valid'))

# Batch Normalisation before passing it to the next layer

model4.add(BatchNormalization())

# 2nd Convolutional Layer

model4.add(Conv2D(filters=64, kernel\_size=(5,5), strides=(2,2), padding='valid'))

model4.add(Activation('relu'))

# Pooling

model4.add(MaxPooling2D(pool\_size=(3,3), strides=(2,2), padding='valid'))

# Batch Normalisation

model4.add(BatchNormalization())

# 3rd Convolutional Layer

model4.add(Conv2D(filters=96, kernel\_size=(3,3), strides=(1,1), padding='valid'))

model4.add(Activation('relu'))

# Batch Normalisation

model4.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2), padding='valid'))

model4.add(BatchNormalization())

# 4th Convolutional Layer

model4.add(Conv2D(filters=128, kernel\_size=(1,1), strides=(1,1), padding='valid'))

model4.add(Activation('relu'))

# Batch Normalisation

model4.add(MaxPooling2D(pool\_size=(3,3), strides=(2,2), padding='valid'))

model4.add(BatchNormalization())

# Passing it to a dense layer

model4.add(Flatten())

# 1st Dense Layer

model4.add(Dense(256, input\_shape=(224\*224\*3,)))

model4.add(Activation('relu'))

# Add Dropout to prevent overfitting

model4.add(Dropout(0.4))

# Batch Normalisation

model4.add(BatchNormalization())

# 2nd Dense Layer

model4.add(Dense(256))

model4.add(Activation('relu'))

# Add Dropout

model4.add(Dropout(0.4))

# Batch Normalisation

model4.add(BatchNormalization())

# Output Layer

model4.add(Dense(10))

model4.add(Activation('softmax'))

model4.summary()

model4.compile(optimizer = 'sgd' , loss = 'categorical\_crossentropy' , metrics=['accuracy'])

model4.fit(x = train\_set, validation\_data = test\_set, epochs = 20)

result=model4.evaluate(test\_set)

print("loss=",result[0])

#print("training accuracy= 0.9919")

print("testing accuracy=",result[1])

import numpy as np

from keras.preprocessing import image

classes=['ammaHW','ashishHW','aswadHW','DKmamHW','nagsHW','nandanaHW','pavanHW','pruthviHW','sravyaHW','yeshwanthHW']

def test\_on\_a\_image(path="/content/datasetedgedv1/datasetedged/test\_set/64/cropimg1000.jpeg"):

test\_image = image.load\_img(path, target\_size = (224, 224))

test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis = 0)

result = model4.predict(test\_image)

for i in result:

count=0

for j in i:

z=float(format(j,'.5f'))

print(classes[count]+"="+str(z\*100)+'%',end="\n")

count+=1

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/dkmamHW/cropimg888.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/dkmamHW/cropimg999.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/aswadHW/cropimg802.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/ammaHW/cropimg952.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/nagsHW/cropimg802.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/ashishHW/cropimg888.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/nandanaHW/cropimg1000.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/pavanHW/cropimg900.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/pruthviHW/cropimg888.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/sravyaHW/cropimg815.jpeg")

test\_on\_a\_image("/content/local\_datasetedged/datasetedged/test\_set/yeshwanthHW/cropimg976.jpeg")

model4.save('alexnetcustom10writers')

"""#Utility Functions for colab"""

make\_zipfolder("alexnetcustom10writers","/content/alexnetcustom10writers")

#make\_zipfolder("local\_datasetedged","/content/gdrive/My Drive/datasetedgedhws") #commenting download code to save time

!unzip /content/local\_datasetedged.zip -d local\_datasetedged

#Linux commands for copying a folder in drive to colab

!cp "/content/gdrive/My Drive/dataset" -r "/content"

#Linux commands for creating a sym link for folder in drive

!ln -s "/content/gdrive/My Drive/datasetedgedhws" "/content/edgeddatasetcopy"

**3.2 Web application Backend.py**

Code for creating a REST API for handwriting classification model using flask

**Pipeline 1:**

!pip install flask-ngrok

!pip install gevent

from flask\_ngrok import run\_with\_ngrok

from flask import Flask, redirect, url\_for, request, render\_template

from werkzeug.utils import secure\_filename

from gevent.pywsgi import WSGIServer

from flask import Flask, redirect, url\_for

from \_\_future\_\_ import division, print\_function

import sys

import os

import glob

import re

import json

#UPLOAD THIS MODEL ZIP FILE BEFORE RUNNING THIS AND THEN RUN

!unzip alexnetcustom10writers.zip -d alexnetcustom10writes

import tensorflow as tf

from tensorflow import keras

model=keras.models.load\_model("/content/alexnetcustom10writes")

**Pipeline 2:**

import cv2

import numpy as np

from numpy import random

import matplotlib.pyplot as plt

from keras.preprocessing import image

from PIL import Image

edged\_image=""

iimg2=""

lst=[]

x=0

edged=""

classes=['ammaHW','ashishHW','aswadHW','DKmamHW','nagsHW','nandanaHW','pavanHW','pruthviHW','sravyaHW','yeshwanthHW']

def test\_on\_a\_image(path="/content/datasetedgedv1/datasetedged/test\_set/64/cropimg1000.jpeg"):

test\_image = image.load\_img(path, target\_size = (224, 224))

test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis = 0)

result = model.predict(test\_image)

max\_person\_score=-10

max\_name="default"

for i in result:

count=0

print("------------------------------------------------")

for j in i:

if(j>max\_person\_score):

max\_person\_score=j

max\_name=classes[count]

z=float(format(j,'.5f'))

print(classes[count]+"="+str(z\*100)+'%',end="\n")

count+=1

print("------------------------------------------------")

return max\_name

def get\_random\_crop(image, crop\_height, crop\_width):

max\_x = image.shape[1] - crop\_width

max\_y = image.shape[0] - crop\_height

x = np.random.randint(0, max\_x)

y = np.random.randint(0, max\_y)

crop = image[y: y + crop\_height, x: x + crop\_width]

return crop

def sliding\_window(no\_of\_crops,htsize=224,wdsize=224):

iimg = edged

writers1=[]

for cnt in range(1,no\_of\_crops+1):

global iimg2

iimg2=get\_random\_crop(iimg,htsize,wdsize)

cv2.imwrite('/content/croppedimg'+str(x)+'.jpeg', iimg2)

cnt=cnt+1

show\_image(iimg2,"final"+str(cnt))

ans=test\_on\_a\_image('/content/croppedimg'+str(x)+'.jpeg')

#print("returned\_ans=",ans)

writers1.append(ans)

#print(writers1)

dct={}

for i in writers1:

if i in dct:

dct[i]=dct[i]+1

else:

dct[i]=1

print(dct)

max=-2222

max\_voted=""

for key,value in dct.items():

if(value>max):

max=value

max\_voted=key

value=(value/20)\*100

print(max\_voted,max)

return str(dct)

def show\_image(image\_object,name):

plt.figure(figsize=(14, 7))

plt.title(name)

if(name!='Grayscale\_Image'):

plt.imshow(image\_object)

else:

plt.imshow(image\_object, cmap='Greys\_r')

def preprocess1(image\_path):

image=cv2.imread(image\_path) #read in the image

gray=cv2.cvtColor(image,cv2.COLOR\_BGR2GRAY) #RGB To Gray Scale

blurred=cv2.GaussianBlur(gray,(5,5),0) #(5,5) is the kernel size and 0 is sigma that determines the amount of blur

global edged

edged=cv2.Canny(blurred,30,50) #30 MinThreshold and 50 is the MaxThreshold

global x

x=random.randint(100)

new\_image\_name='edgedimg'+str(x)+'.jpeg'

while x in lst:

x=random.randint(100)

new\_image\_name='edgedimg'+str(x)+'.jpeg'

lst.append(x)

global edged\_image

edged\_image=new\_image\_name

return sliding\_window(20)

**Pipeline 3:**

#Before running the below code, create a empty folder ips and create a folder templates and move front end files into it i.e index.html and base.html

app=Flask(\_\_name\_\_,template\_folder='/content/templates')

UPLOAD\_FOLDER='/content/ips'

app.config['UPLOAD\_FOLDER']=UPLOAD\_FOLDER

run\_with\_ngrok(app)

@app.route("/")

def index():

return render\_template('index.html')

@app.route('/predict', methods=['GET', 'POST'])

def upload():

if request.method == 'POST':

print("Image recieved on server")

f = request.files['file'] # Get the file from post request

file\_path = os.path.join(app.config['UPLOAD\_FOLDER'] ,f.filename)

f.save(file\_path)

preds = preprocess1(file\_path)

print("preds=",preds)

return preds

return None

app.run()

**3.3 Web Application Frontend.py**

**Index.html**

{% extends "base.html" %} {% block content %}

<h2>Image Classifier</h2>

<div>

<form id="upload-file" method="post" enctype="multipart/form-data">

<label for="imageUpload" class="upload-label">

Choose...

</label>

<input type="file" name="file" id="imageUpload" accept=".png, .jpg, .jpeg">

</form>

<div class="image-section" style="display:none;">

<div class="img-preview">

<div id="imagePreview">

</div>

</div>

<div>

<button type="button" class="btn btn-primary btn-lg " id="btn-predict">Predict!</button>

</div>

</div>

<div class="loader" style="display:none;"></div>

<h3 id="result">

<span> </span>

</h3>

</div>

{% endblock %}

**Base.html**

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>AI Demo</title>

<link href="https://cdn.bootcss.com/bootstrap/4.0.0/css/bootstrap.min.css" rel="stylesheet">

<script src="https://cdn.bootcss.com/popper.js/1.12.9/umd/popper.min.js"></script>

<script src="https://cdn.bootcss.com/jquery/3.3.1/jquery.min.js"></script>

<script src="https://cdn.bootcss.com/bootstrap/4.0.0/js/bootstrap.min.js"></script>

<style>

.img-preview

{

width: 256px;

height: 256px;

position: relative;

border: 5px solid #F8F8F8;

box-shadow: 0px 2px 4px 0px rgba(0, 0, 0, 0.1);

margin-top: 1em;

margin-bottom: 1em;

}

.img-preview>div {

width: 100%;

height: 100%;

background-size: 256px 256px;

background-repeat: no-repeat;

background-position: center;

}

input[type="file"] {

display: none;

}

.upload-label{

display: inline-block;

padding: 12px 30px;

background: #39D2B4;

color: #fff;

font-size: 1em;

transition: all .4s;

cursor: pointer;

}

.upload-label:hover{

background: #34495E;

color: #39D2B4;

}

.loader {

border: 8px solid #f3f3f3; /\* Light grey \*/

border-top: 8px solid #3498db; /\* Blue \*/

border-radius: 50%;

width: 50px;

height: 50px;

animation: spin 1s linear infinite;

}

@keyframes spin {

0% { transform: rotate(0deg); }

100% { transform: rotate(360deg); }

}

</style>

</head>

<body>

<nav class="navbar navbar-dark bg-dark">

<div class="container">

<a class="navbar-brand" href="#">AI Demo</a>

<button class="btn btn-outline-secondary my-2 my-sm-0" type="submit">Help</button>

</div>

</nav>

<div class="container">

<div id="content" style="margin-top:2em">{% block content %}{% endblock %}</div>

</div>

</body>

<footer>

<script src="{{ url\_for('static', filename='/content/static/js/main.js') }}" type="text/javascript"></script>

<script type="text/javascript">

$(document).ready(function () {

// Init

$('.image-section').hide();

$('.loader').hide();

$('#result').hide();

// Upload Preview

function readURL(input) {

if (input.files && input.files[0]) {

var reader = new FileReader();

reader.onload = function (e) {

$('#imagePreview').css('background-image', 'url(' + e.target.result + ')');

$('#imagePreview').hide();

$('#imagePreview').fadeIn(650);

}

reader.readAsDataURL(input.files[0]);

}

}

$("#imageUpload").change(function () {

$('.image-section').show();

$('#btn-predict').show();

$('#result').text('');

$('#result').hide();

readURL(this);

});

// Predict

$('#btn-predict').click(function () {

var form\_data = new FormData($('#upload-file')[0]);

// Show loading animation

$(this).hide();

$('.loader').show();

// Make prediction by calling api /predict

$.ajax({

type: 'POST',

url: '/predict',

data: form\_data,

contentType: false,

cache: false,

processData: false,

async: true,

success: function (data) {

// Get and display the result

$('.loader').hide();

$('#result').fadeIn(600);

$('#result').text(' Result: ' + data);

console.log('Success!');

},

});

});

});

</script>

</footer>

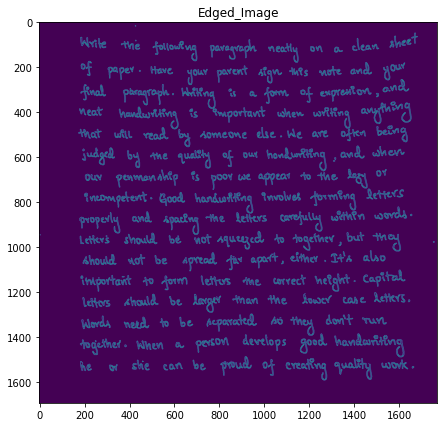
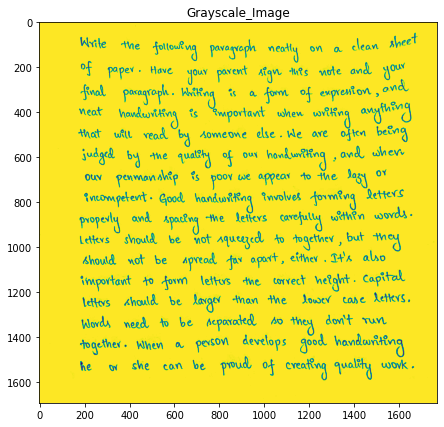
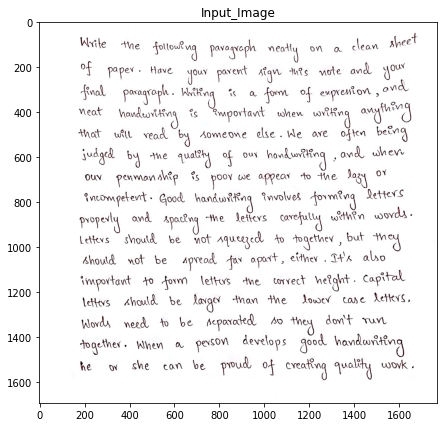
</html>

**CHAPTER 4**

## SNAPSHOTS

**4.1 classification model:**

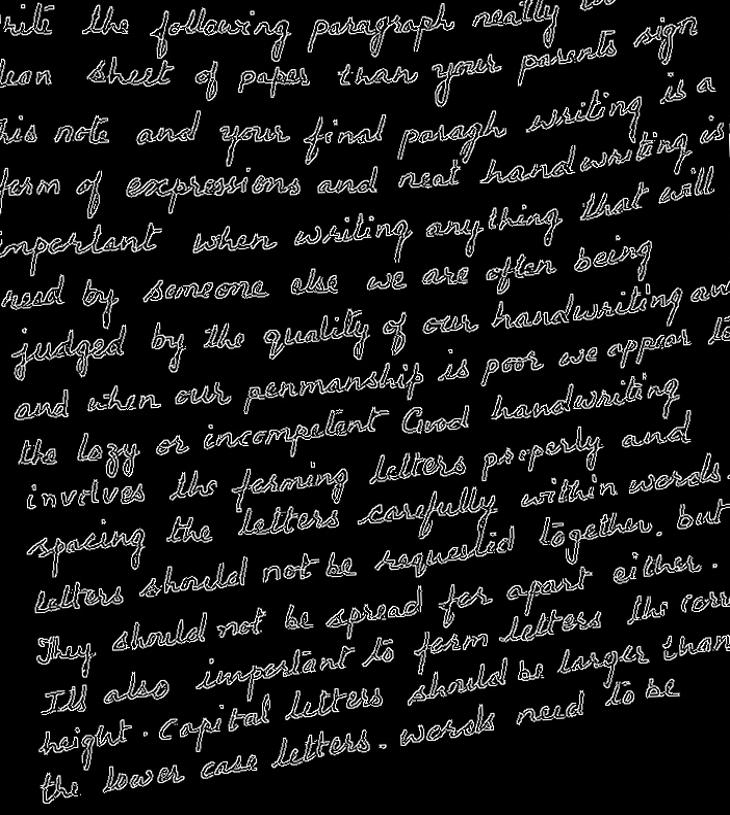
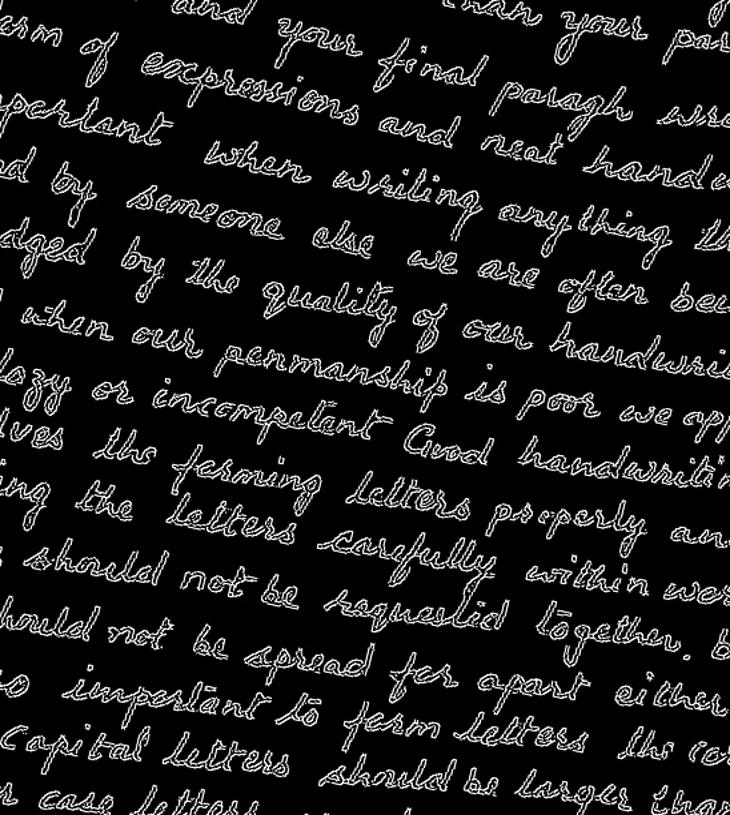
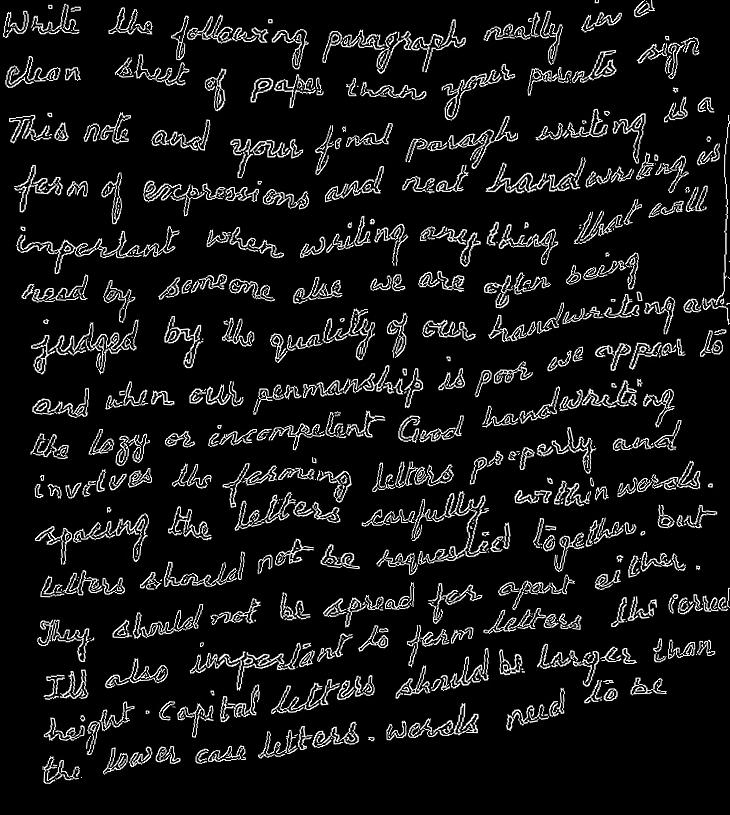
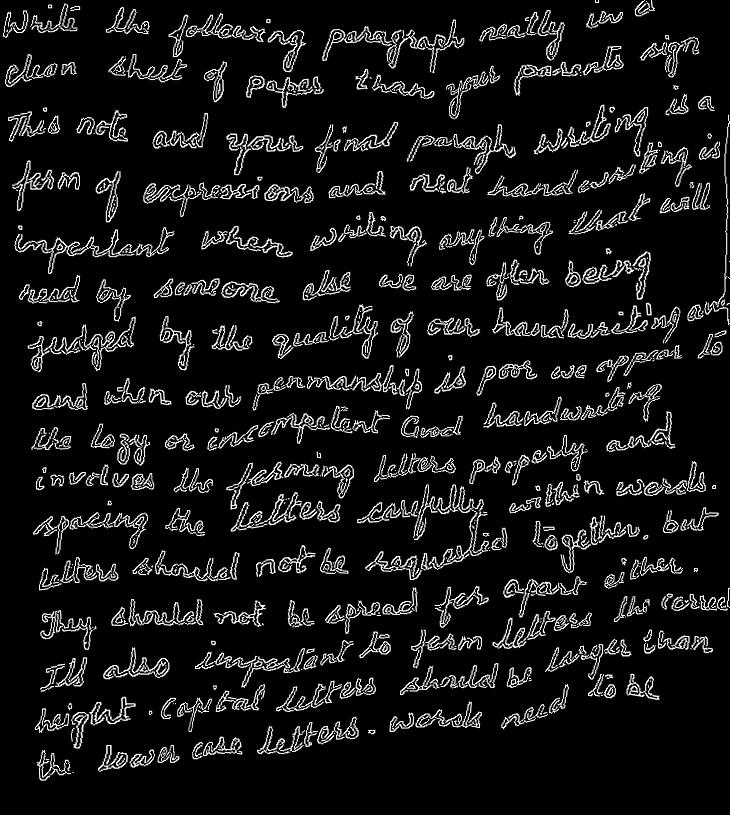
**Pipeline 1**



*Fig 4.1 Input Handwriting Image Fig 4.2 Grayscale image Fig 4.3 Final Edged Image 1*

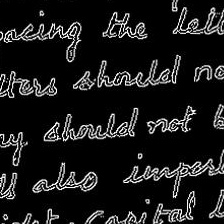
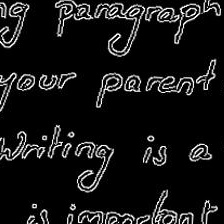
The above images Fig 4.1, Fig 4.2 and Fig 4.3 are outputs after applying filters like grayscale/ gaussian blur and canny edge detectors on the input handwriting image of a writer.

**Pipeline 2**

**   **

*Fig 4.4 Anticlockwise Rotation Fig 4.5 Clockwise Rotation Fig 4.6 Random Distortion 1 Fig 4.7 Random Distortion 2*

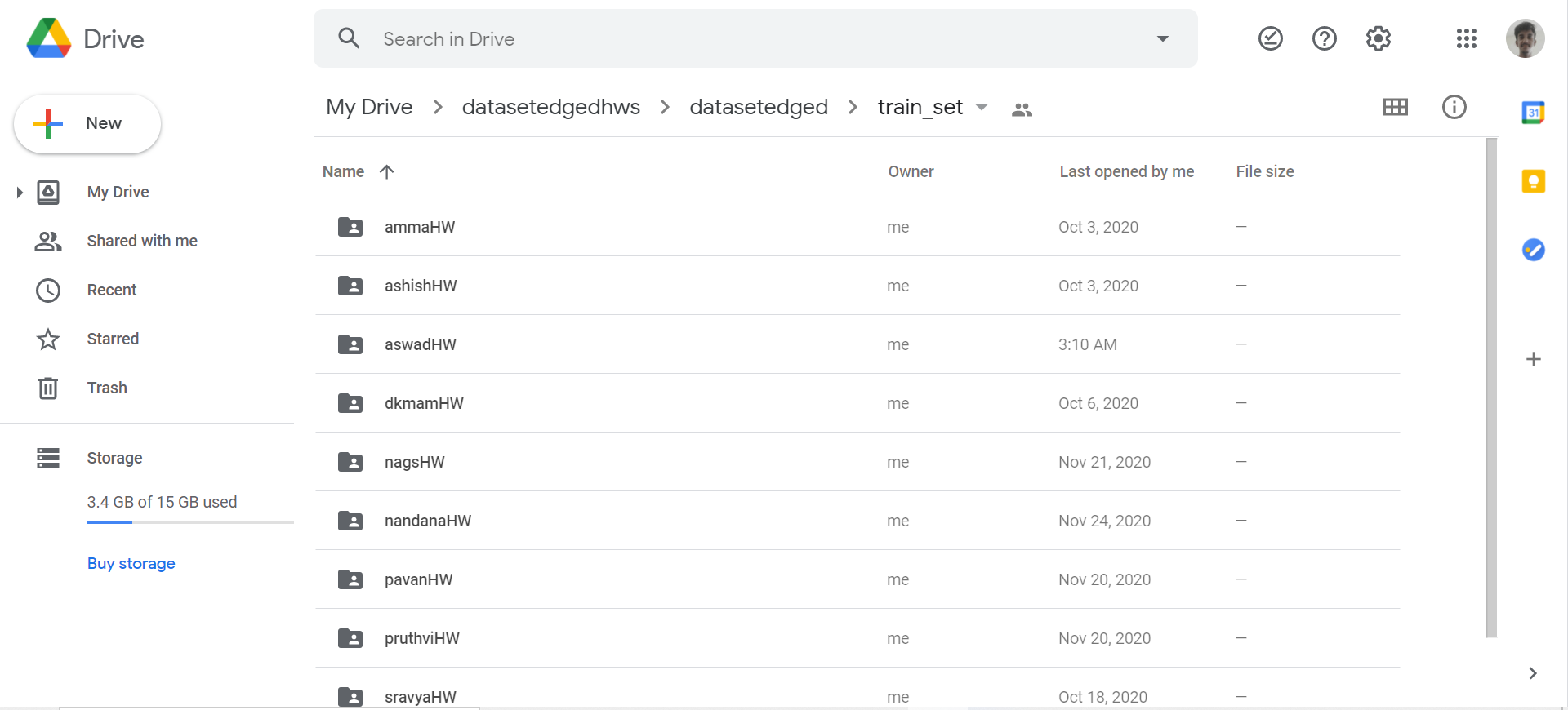
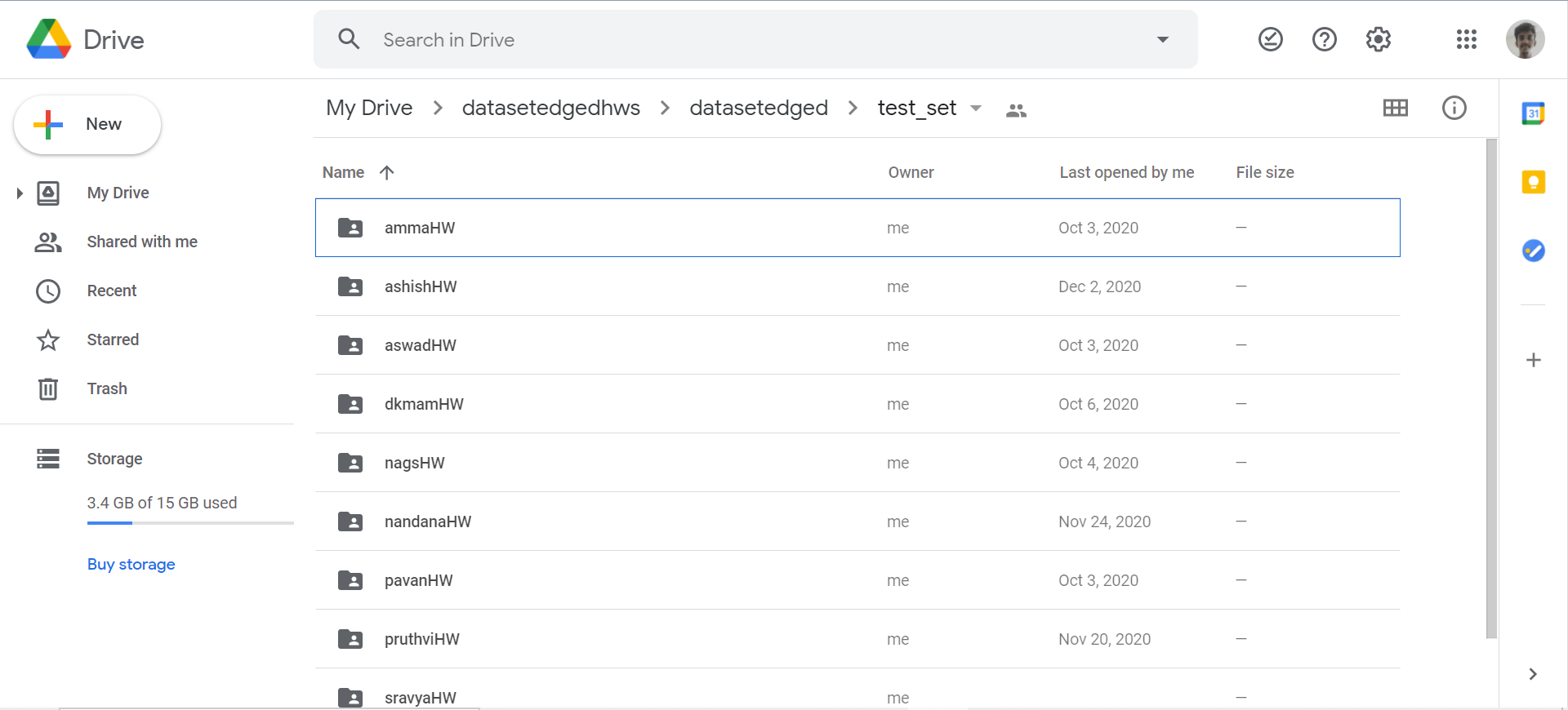
The above images Fig 4.4 is the image after applying a rotation of 8 degree anticlockwise, Fig 4.5 is image after doing 12 degrees rotation in clockwise. Fig 4.6 and Fig 4.7 are the outputs after applying random distortion to the line alignment by grid width if 10 units, grid height of 10 units and grid width of 4 units and grid height of 4 units respectively.

*Fig 4.8 Random crops of size 224\*224*

The above shown image Fig 4.8 is a random cropped snapshot from handwriting image of a writer. The size of these snapshots is 224\*224. These are the images that were sent to the CNN to get trained. For every testing image, the same sized snapshot is generated and tested on the trained model for evaluating accuracy.

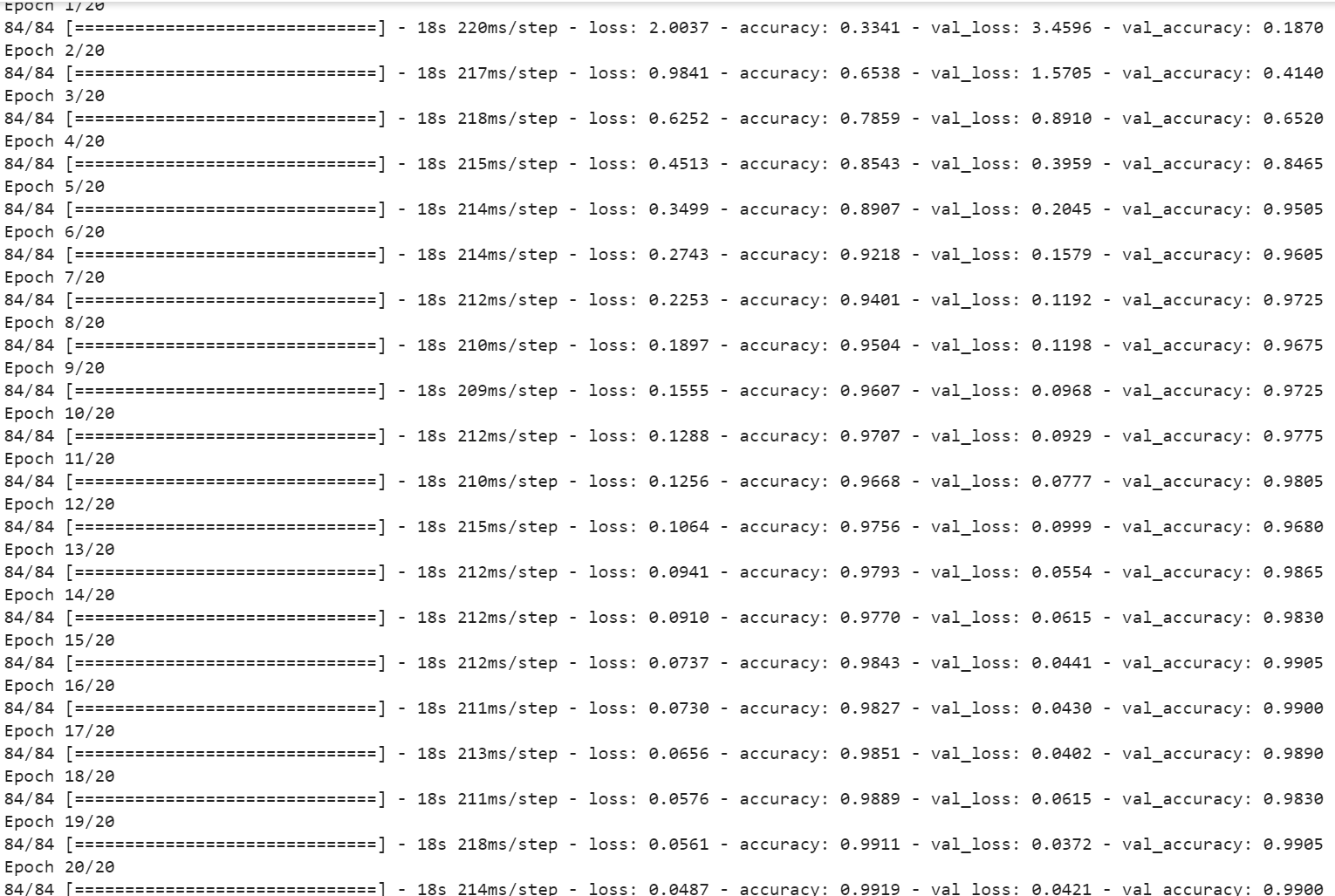
**Pipeline 3:**

*Fig 4.9 Training and Testing datasets stored in Drive*

The above image Fig 4.9 are snapshots of training image dataset and testing image dataset which are stored in google drive. Loading these datasets into working environment (i.e. google colab) is very easy from google drive. There are 2 folders in the folder datasetedgedhw, which are trainset and testset. There are 10 sub folders in each of those 2 folders. Each folder is for one writer. For every writer in trainset there are 800 images. So total 8000 images and 200 images for every writer in testset. So, a total of 2000 images in testset.

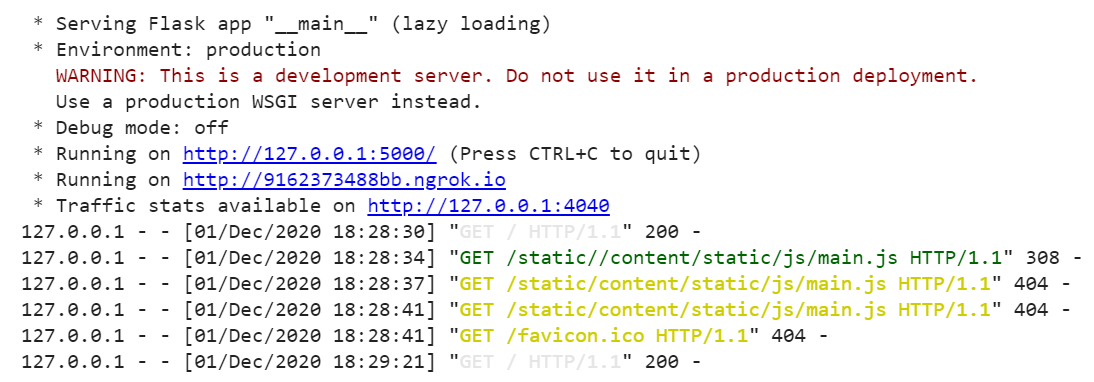
**Pipeline 4:**

****

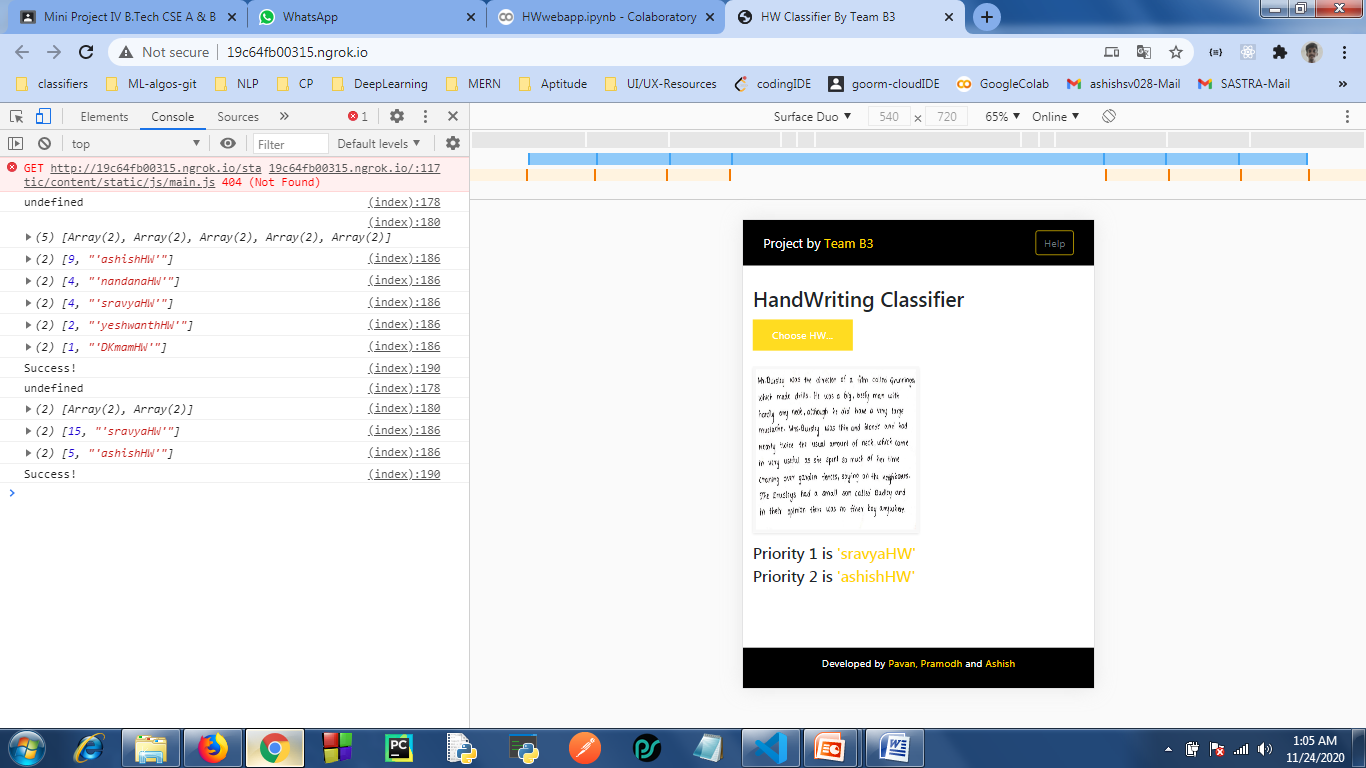
*Fig 4.10 Training process of classification model*

The above image Fig 4.10 shows the evaluation parameters in every epoch. From the image we can see that the loss constantly getting decreased and accuracy constantly increasing. Similar trend can be seen in validation loss and validation accuracy.

**4.2 Web Application Backend:**

****

*Fig 4.11 Server Processing*

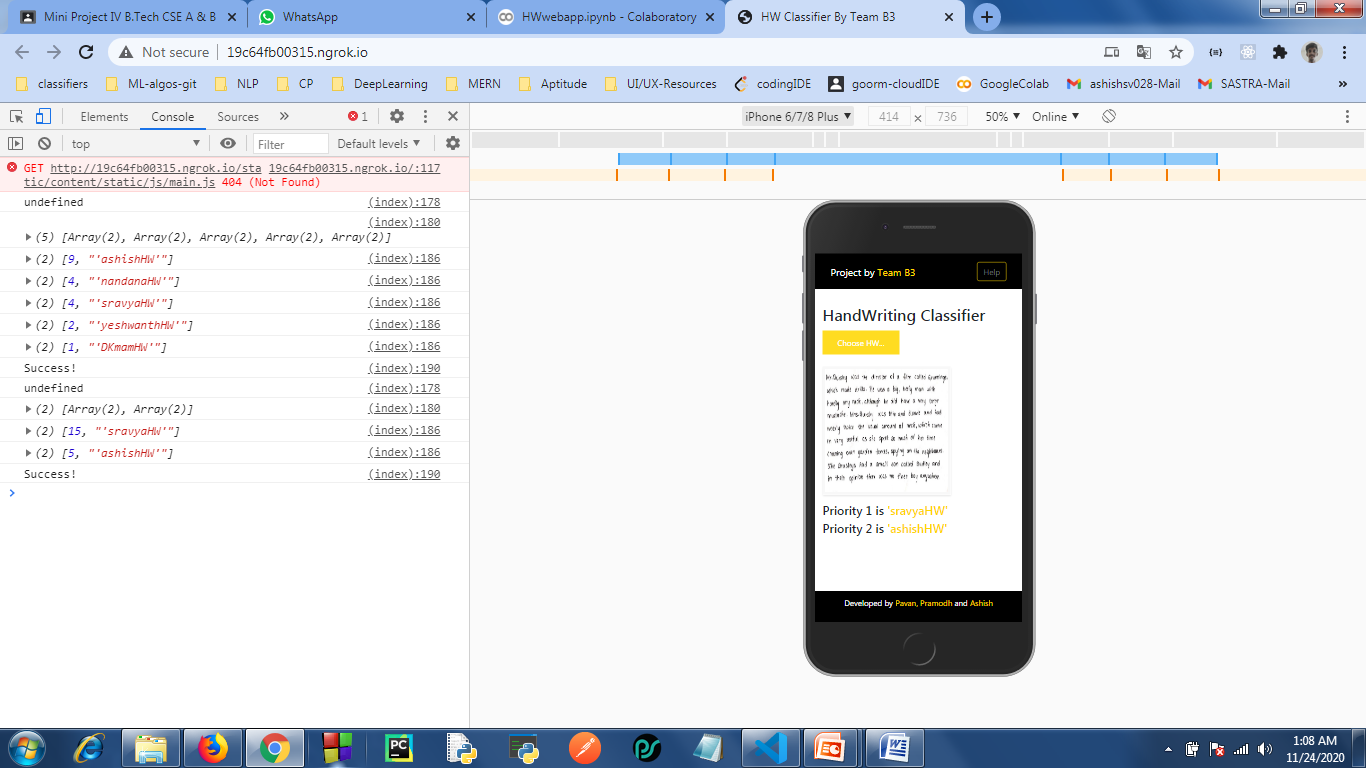
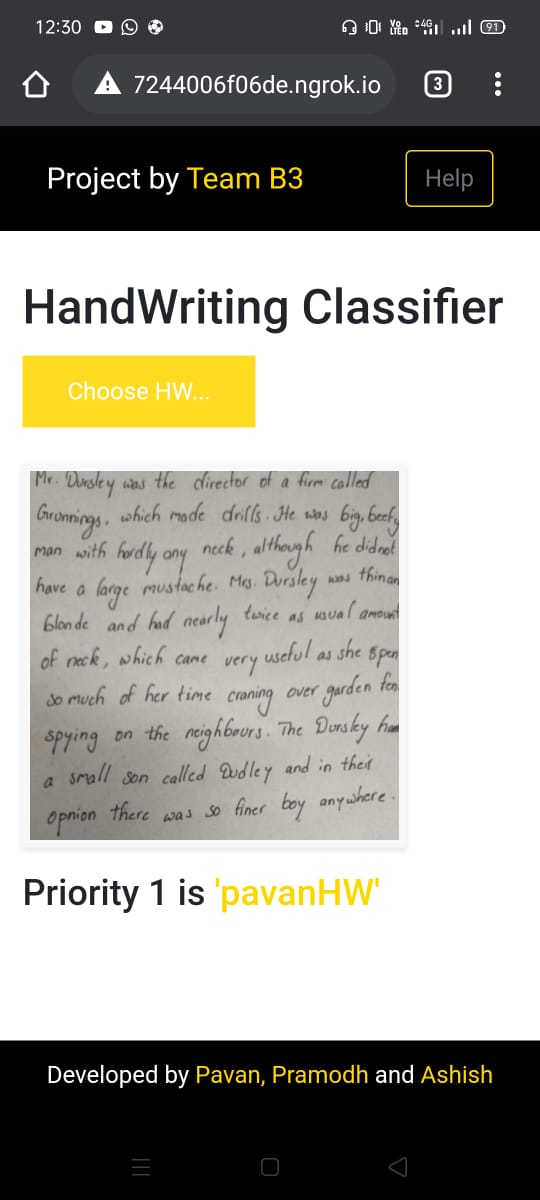
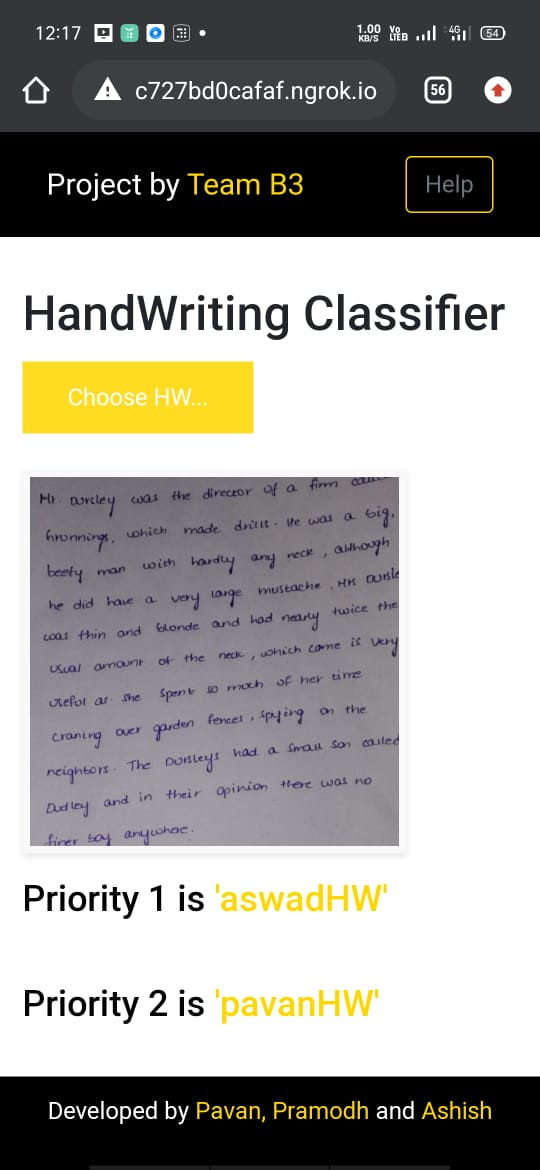


*Fig 4.12 Output sent from API to frontend*

The image 4.11 shows various URLs that were generated as soon as running the server. The web app can be seen live on clicking the link <http://9162373488bb.ngrok.io>. Various logs can also be viewed below.

The image 4.12 shows the result that was sent to frontend from API. From the image it is clearly visible that the API sent 75% chance that the input handwriting is of Sravya and 25% chance that the input image handwriting is of Ashish.

**4.3 Web Application Frontend**

*Fig 4.13 UI images of the webapp frontend*

The above image fig 4.13 show the UI of webapp in various devices. We can also see the result string formatting and result displayed in terms of priority.

**CHAPTER 5**

## CONCLUSION AND FUTURE PLANS

**5.1 CONCLUSION:**

Automatic Handwriting Identification is one of the fascinating research problems in the fields of document analysis and criminological analysis etc. The effective execution of handwriting identification systems can be applicable in banks, check processing, historical and forensic analysis, signature identification, graphology, legal documents and ancient manuscripts etc. In this paper we have implemented a classification model based on AlexNet state of art CNN architecture model on ten different writers and got a decent accuracy and validation accuracy without any overfitting issues. These results and analyses show the correctness of the implemented model and shows the productiveness of the proposed implementation for writer identification.

**5.2 FUTURE PLANS:**

Apart from AlexNet, there are many state of art CNN architecture models like VGG, GoogleNet, ResNet etc. Implementing the same model using any other such architecture can be worked out. Emphasis can be further increased in optimization of architecture by using techniques like Hyper Parameter Tuning, Normalization, Effective dropout ratios etc. Constrains on uploading user input images can be improved, so that unnecessary variations that occur in line spacing while capturing the photo can be removed. Dataset size and number of writers can be further increased. The performance of the model can also be tweaked using other optimizers and loss functions with different number of epochs and can be compared with the implemented model’s configurations.

**CHAPTER 6**

## REFERENCES

[1] A. Rehman, S. Naz, and M. I. Razzak, ‘‘Writer identification using machine learning approaches: A comprehensive review,’’ Multimedia Tools Appl., pp. 1–43, Sep. 2018. doi: 10.1007/s11042-018-6577-1.

[2] V. Christlein, M. Gropp, S. Fiel, and A. Maier. (2017). ‘‘Unsupervised feature learning for writer identification and writer retrieval.’’ [Online]. Available: <https://arxiv.org/abs/1705.09369>.

[3] P. Pandey and K. R. Seeja, ‘‘Forensic writer identification with projection profile representation of graphemes,’’ in Proc. 1st Int. Conf. Smart Syst., Innov. Comput. Singapore: Springer, 2018, pp. 1

[4] A. Durou, I. Aref, S. Al-Maadeed, A. Bouridane, and E. Benkhelifa, ‘‘Writer identification approach based on bag of words with OBI features,’’ Inf. Process. Manage., vol. 56, no. 2, pp. 354–366, 2019.

[5] I. Siddiqi and N. Vincent, ‘‘Combining contour-based orientation and curvature features for writer recognition,’’ in Proc. Int. Conf. Comput. Anal. Images Patterns. Berlin, Germany: Springer, 2009, pp. 245–252.

[6] V. Christlein, D. Bernecker, A. Maier, and E. Angelopoulou, ‘‘Offline writer identification using convolutional neural network activation features,’’ in Proc. German Conf. Pattern Recognit. Cham, Switzerland: Springer, 2015, pp. 540–552.

[7] S. Fiel and R. Sablatnig, ‘‘Writer identification and retrieval using a convolutional neural network,’’ in Proc. Int. Conf. Comput. Anal. Images Patterns. Cham, Switzerland: Springer, 2015, pp.

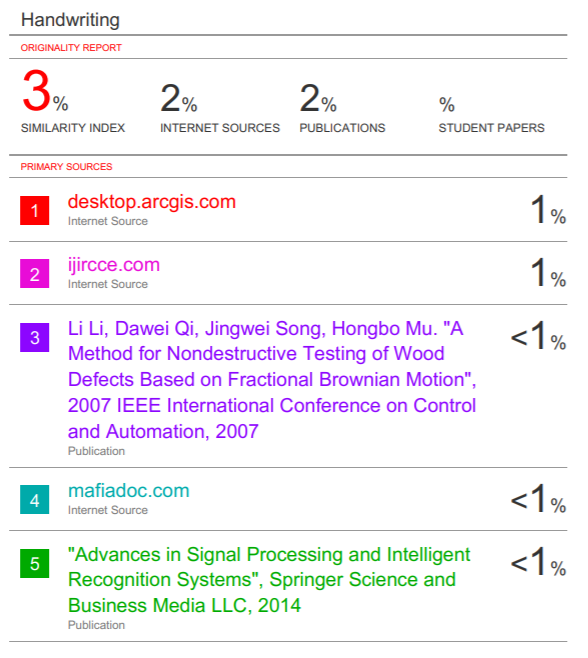
[8] S. Naz et al., ‘‘Urdu Nastaliq recognition using convolutional–recursive deep learning,’’ Neurocomputing, vol. 243, pp. 80–87, Jun. 2017.

[9] Y. Tang and X. Wu, ‘‘Text-independent writer identification via CNN features and joint Bayesian,’’ in Proc. 15th Int. Conf. Frontiers Handwriting Recognit. (ICFHR), Oct. 2016, pp. 566–571.

**CHAPTER 7**

**APPENDIX**

**7.1 Similarity Check Report**

****