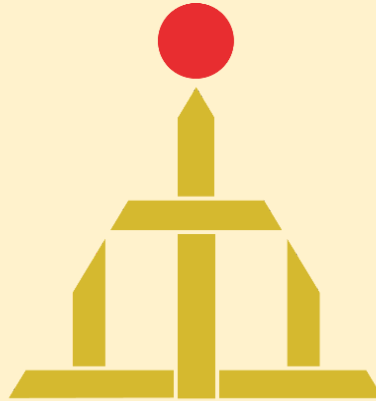


Indian Institute of Information Technology Bhagalpur - 813210



भारतीय सूचना प्रौद्योगिकी संस्थान भागलपुर
Indian Institute of Information Technology
Bhagalpur

Devanagari Character Recognition

Minor Project, 2020.

Submitted by:

Abhishek Kumar (170101003)
Himanshu Ranjan (170101017)
Suraj Kumar (170101053)
Rashi Krishna (170102037)

Guided by :

Dr. Thejaswini

Department of Computer Science & Engineering
IIIT, BHAGALPUR, BIHAR 813210, INDIA
July-Dec 2020

Contents

Introduction	3
Problem Statement	3
Dataset	3
Goals	4
Tools Used	5
System Analysis	5
System Architecture	5
Preprocessing	6
Preprocess Labels	6
Preprocess Images	7
Model	8
Evaluating Model Accuracy	11
Case of overfitting	11
No overfitting :	11
Testing	12
Segmentation	13
Web Application	14
Result	15
Effect of changing activation type	16
Effect of changing learning rate	16
Working	17
Predicting a single character (in jupyter-notebook)	17
Predicting a whole word (in jupyter-notebook)	18
Predicting a whole word (using web application)	20
Applications of Devanagari Character Recognition	21
Future Scope	21
References	22

Introduction

Character Recognition is the identification of printed characters from an image, a book, a handwritten note, cheques, or letters. It is similar to using a handheld scanner to read a barcode, or reading OMR exam sheets, but it can distinguish between different alphabets.

Problem Statement

Given an image of a devanagari character, the goal is to predict the correct label for the image.

Python and tensorflow can be used to build a feed forward neural network for the same.

Dataset

The dataset contains labeled images of handwritten Devanagari characters, [Link here](#).

The dataset contains 92 thousand images (32x32 pixels) of 58 characters, digits 0 to 9, vowels 'a' to 'ah' and consonants "ka" to "gya". The images are in png format.



Goals

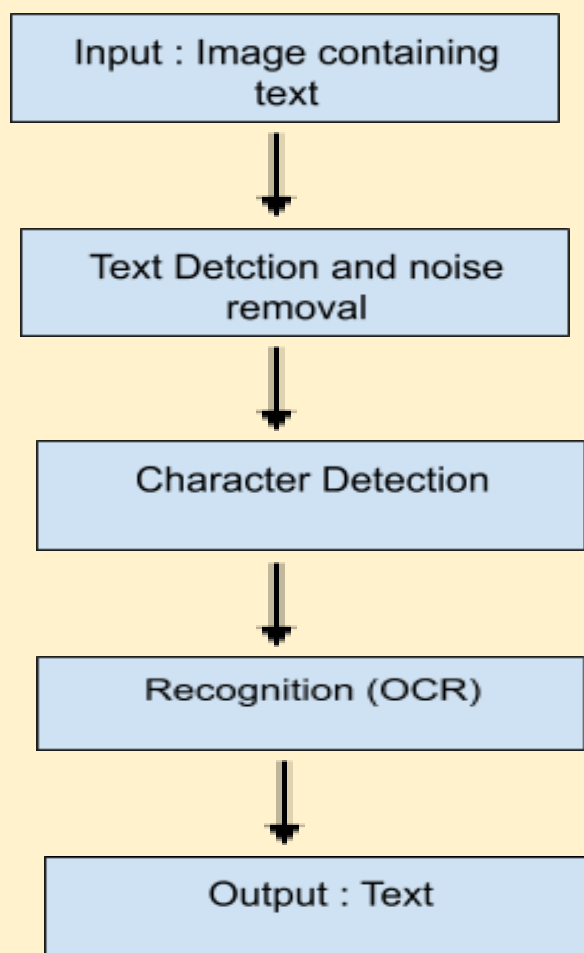
1. The accuracy on the validation dataset for increasing number of levels
2. Effect of changing the hidden unit type from ReLU to Sigmoid to Tanh
3. Accuracy with different learning rates

Tools Used

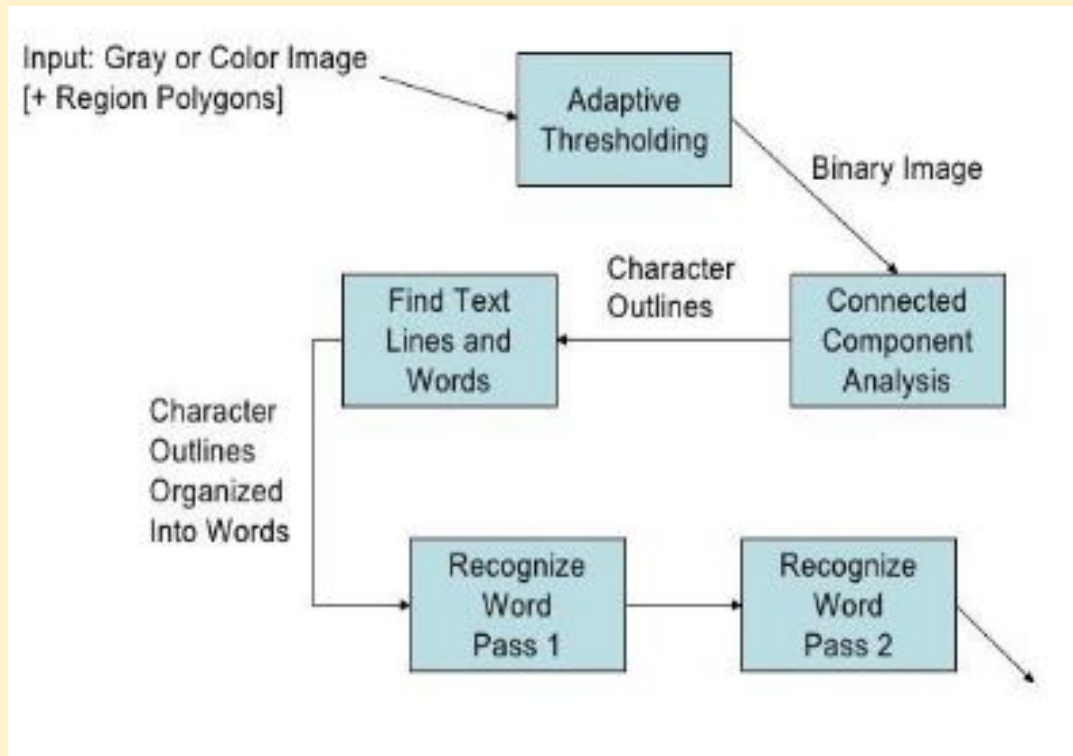
- **Python** - The overall project is coded using Python and its libraries.
- **Tensorflow** - TensorFlow is a free and open-source software library for machine learning. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.
- **Keras** - Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.
- **Flask** - Flask is a micro web framework written in Python.

System Analysis

System Architecture



Detailed architecture of the system :



Preprocessing

Preprocess Labels

The images are stored in a directory named after their character name. We created a dataframe containing pixel value of each image along with their actual devnagri label.

7

	Unnamed: 0	0	1	2	3	4	5	6	7	8	...	775	776	777	778	779	780	781	782	783	labels
0	0	251	250	249	249	252	255	253	244	255	...	255	255	255	255	255	255	255	255	255	aa
1	0	255	253	245	255	251	245	255	245	247	...	248	254	255	246	250	254	255	252	249	aa
2	0	253	250	255	242	255	248	249	249	251	...	244	250	255	245	255	255	255	255	255	aa
3	0	255	255	246	255	255	251	255	250	178	...	254	154	255	249	250	255	255	255	255	aa
4	0	255	244	253	211	138	65	53	136	232	...	255	255	255	255	255	247	255	237	177	aa

5 rows x 786 columns

Preprocess Images

Our dataset already contains images of size 28X28.

But for images of greater size we will need to first preprocess the image.

1. Apply gaussian filter to the image to make text wider.
2. Invert black and white because most of the image is white.
3. Scale down the value of each pixel. (Dividing by 255).

```
char_names = data.character.unique()
rows = 10; columns = 6;
fig, ax = plt.subplots(rows, columns, figsize=(8,16))
for row in range(rows):
    for col in range(columns):
        ax[row,col].set_axis_off()
        if columns*row+col < len(char_names):
            x = data[data.character==char_names[columns*row+col]].iloc[0,:-1].values.reshape(28,28)
            x = x.astype("float64")
            x/=255
            ax[row,col].imshow(x, cmap="binary")
            ax[row,col].set_title(char_names[columns*row+col].split("_")[-1])

plt.subplots_adjust(wspace=1, hspace=1)
plt.show()
```



Model

For our image classifier model we are using CNN. CNN can easily be implemented using keras.

Why CNN ?

One major advantage of using CNNs over NNs is that we do not need to flatten the input images to 1D as they are capable of working with image data in 2D. This helps in retaining the “spatial” properties of images.

Another important advantage of CNN is parameter sharing and reuse. This means instead of using a parameters =

number of nodes on layer, CNN uses a fixed sized small filter. And our task is to find the best filter that can classify our characters with maximum accuracy.

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_4 (Conv2D)	(None, 30, 30, 32)	320
conv2d_5 (Conv2D)	(None, 28, 28, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_6 (Conv2D)	(None, 12, 12, 64)	36928
conv2d_7 (Conv2D)	(None, 10, 10, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	0
dropout_1 (Dropout)	(None, 5, 5, 64)	0
flatten_3 (Flatten)	(None, 1600)	0
dense_3 (Dense)	(None, 128)	204928
dense_4 (Dense)	(None, 64)	8256
dense_5 (Dense)	(None, 46)	2990
Total params: 308,846		
Trainable params: 308,846		
Non-trainable params: 0		

None

We have implemented a CNN with 3 dense layers (including the output layer), 4 convolution, 2 pooling and a dropout layer (with a dropout of 20%). The summary of the model built is shown in the above figure.

Choosing number of hidden layers and neurons in the hidden layers

Deciding the number of neurons in the hidden layers is a very important part of deciding your overall neural network architecture. Though these layers do not directly interact with the external environment, they have a tremendous influence on the final output. Both the number of hidden layers and the

number of neurons in each of these hidden layers must be carefully considered.

Using too few neurons in the hidden layers will result in something called underfitting. Underfitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set.

Using too many neurons in the hidden layers can result in several problems. First, too many neurons in the hidden layers may result in overfitting. Overfitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. A second problem can occur even when the training data is sufficient. An inordinately large number of neurons in the hidden layers can increase the time it takes to train the network. The amount of training time can increase to the point that it is impossible to adequately train the neural network. Obviously, some compromise must be reached between too many and too few neurons in the hidden layers.

There are many rule-of-thumb methods for determining the correct number of neurons to use in the hidden layers, such as the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be $\frac{2}{3}$ the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

These three rules provide a starting point for you to consider. Ultimately, the selection of an architecture for your neural network will come down to trial and error. But what exactly is meant by trial and error? You do not want to start throwing random numbers of layers and neurons at your network. To do so would be very time consuming. Chapter 8, “Pruning a

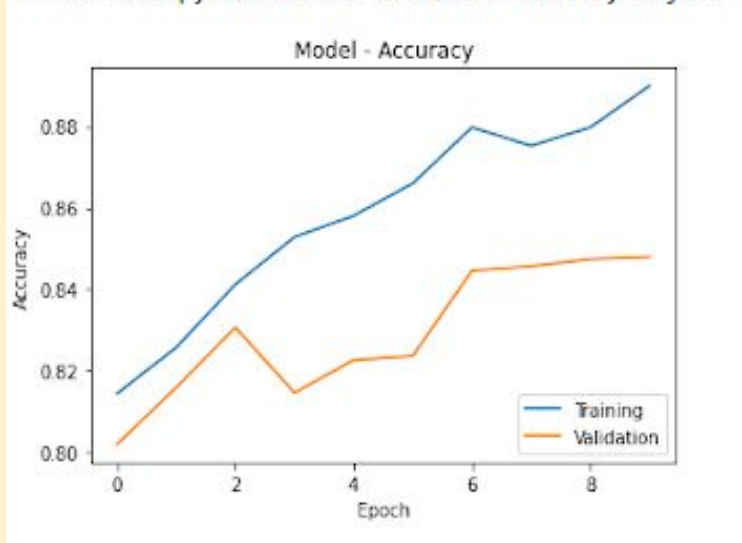
Neural Network” will explore various ways to determine an optimal structure for a neural network.

Evaluating Model Accuracy

Case of overfitting

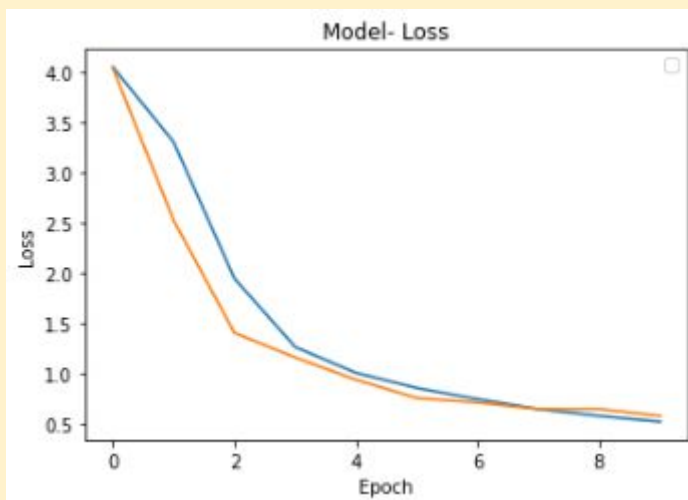
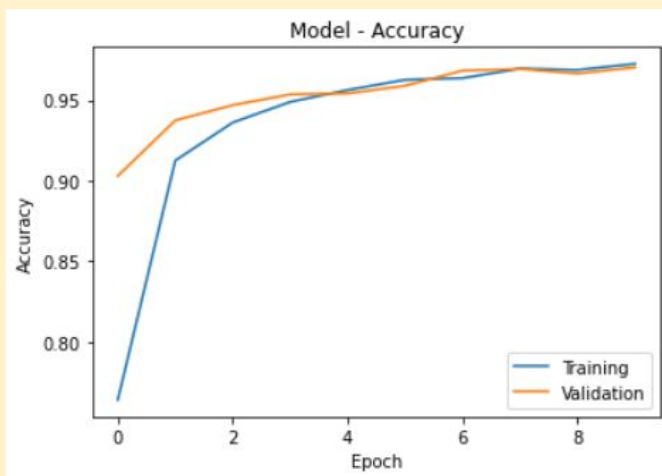
Training our model for 1000 epochs gave good training accuracy but failed to produce good accuracy value for the testing set.

```
<tensorflow.python.keras.callbacks.History object at 0x0000000000000000>
```



No overfitting :

Running the model for 10 epochs with batch size = 32, produced the following results.



Testing

```
In [41]: scores = cnn.evaluate(x_test, y_test, verbose=0)
          print("Accuracy: %.2f%%" % (scores[1]*100))
```

Accuracy: 97.03%

The accuracy on testing dataset is 97.03%.

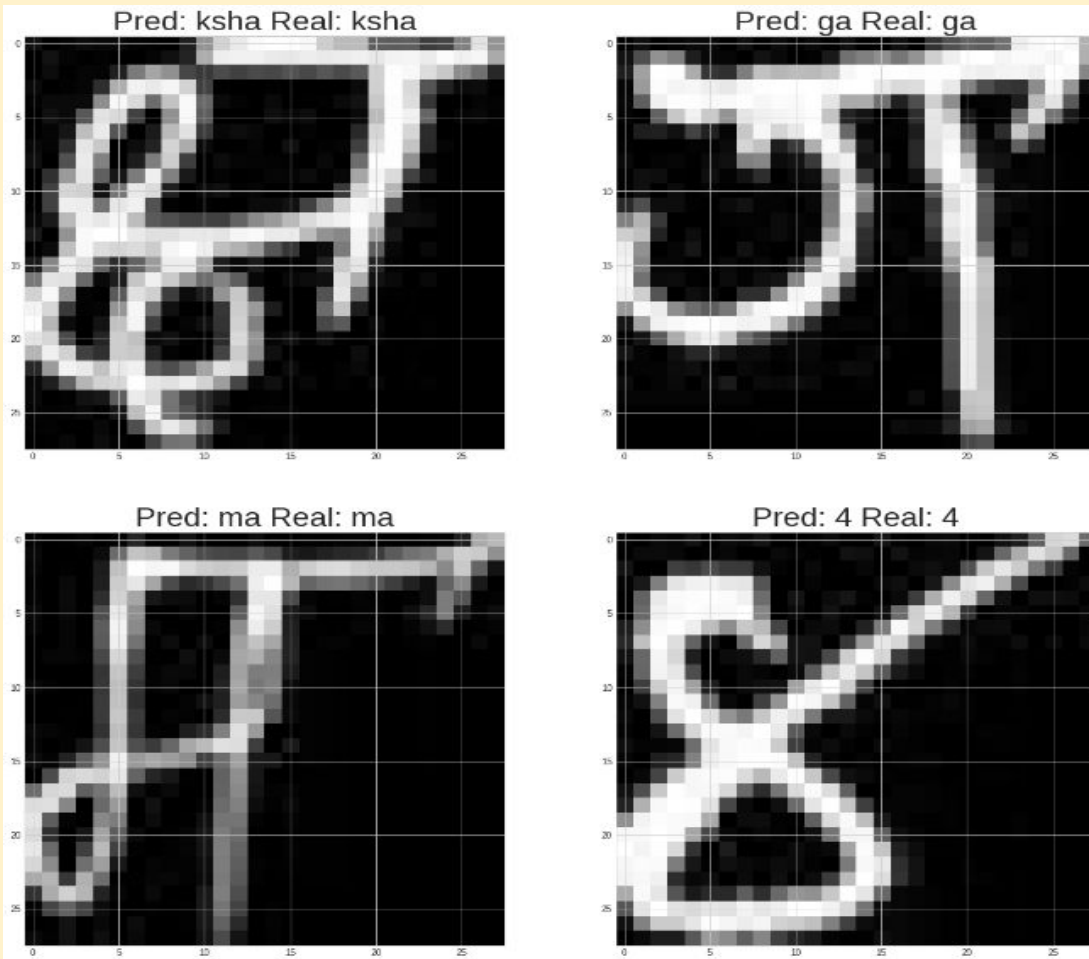
We can view some of the predictions on our dataset.

```

plt.style.use('seaborn-whitegrid')
r=2
c=2
fig = plt.figure(figsize=(20, 20))
for i in range(r*c):
    plt.subplot(r, c, i+1)
    lbl = np.argmax(y_test[i])
    img = x_test[i]
    img = img.reshape(1, 28, 28, 1)
    prediction = cnn.predict(img)
    prediction = np.argmax(prediction)
    title = f"Pred: {classes[prediction]} Real: {classes[lbl]}"
    plt.title(title)
    plt.imshow(img.reshape(img_height_rows, img_width_cols))

plt.show()

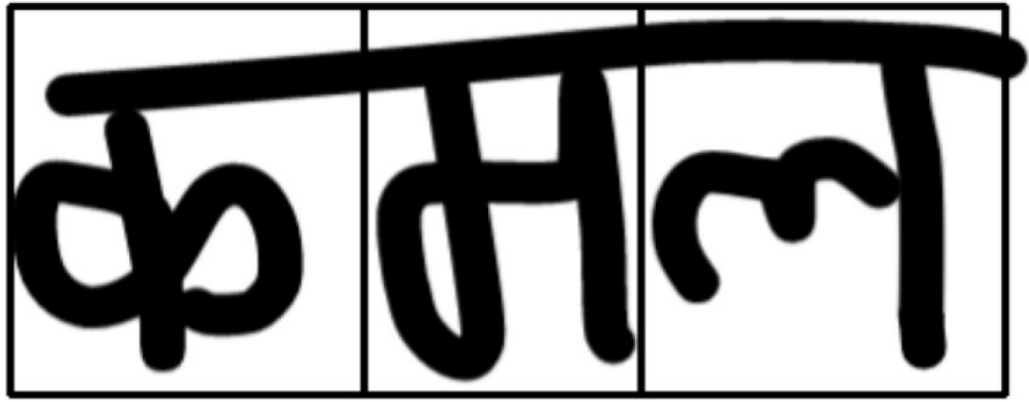
```



Segmentation

To predict **a whole word** we need to slice each character and pass it to our model. This process of slicing a word into its component character is segmentation.

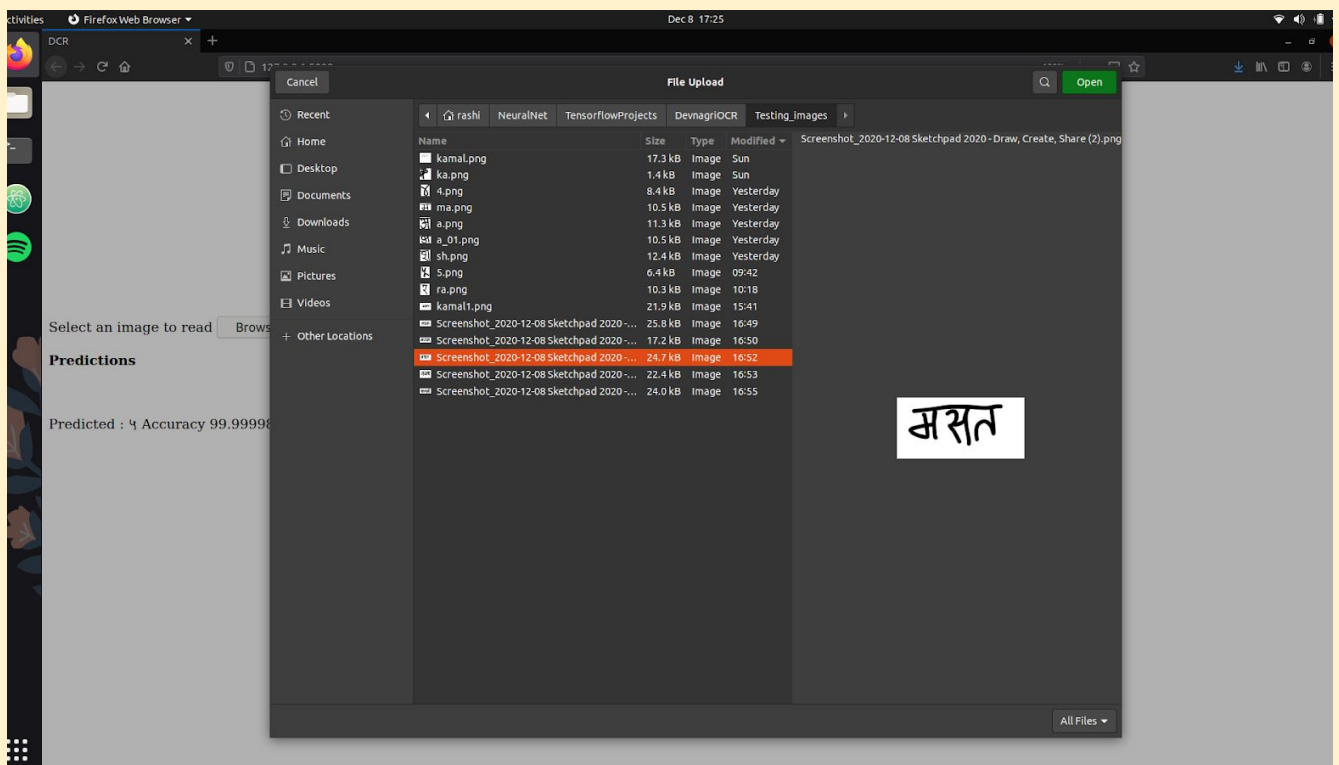
Segmentation can be done simply with the help of white space between two characters. But problems may arise when dealing with characters like 'ग'.



Web Application

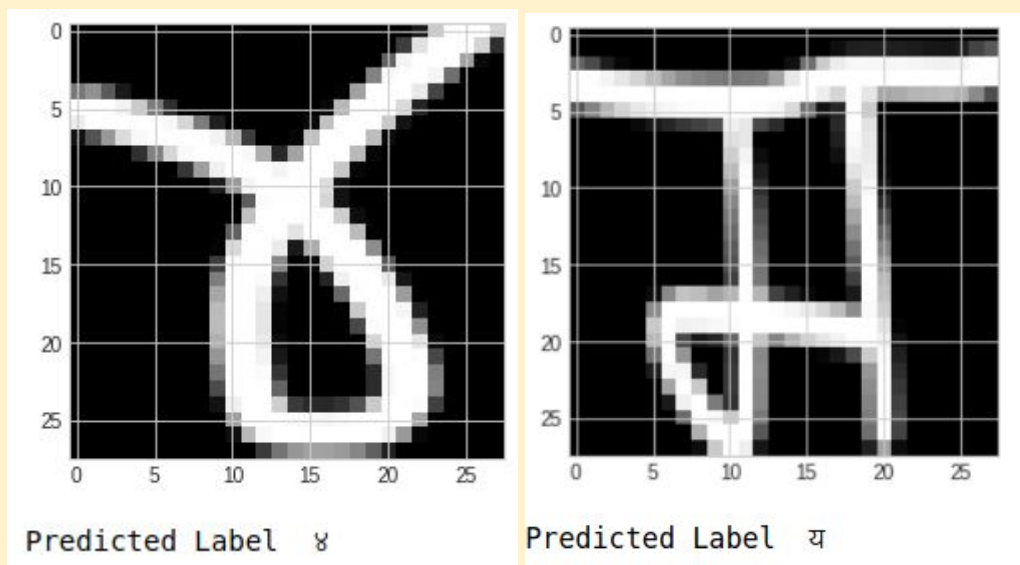
The CNN model is saved in a '.hdf5' file, so we don't have to build our model again and again.

Flask, a python web framework is used to build a web interface to deploy our CNN model.





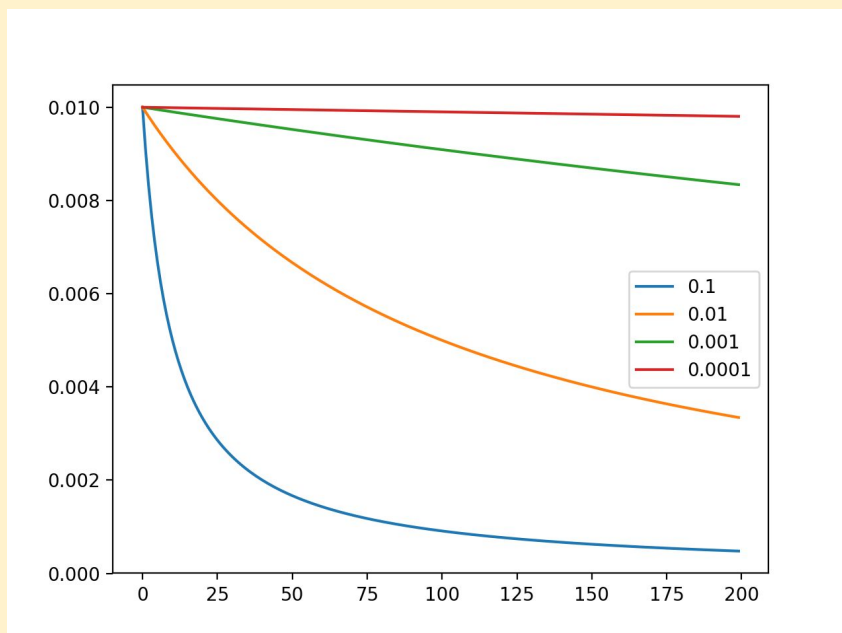
Result



Effect of changing activation type

Activation Type	Accuracy (after 10 epochs)
ReLU	97.03%
Sigmoid	88.24%
Tanh	91.23%


Effect of changing learning rate



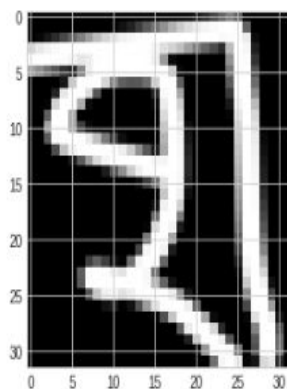
Working

1. Predicting a single character (in jupyter-notebook)

```
In [63]: from PIL import Image, ImageOps
testing_image=Image.open('Testing_images/sh.png')
testing_image.resize((32,32))
```

Out[63]: 

```
In [56]: def pred(testing_image):
testing_image=ImageOps.grayscale(testing_image)
img=np.array(testing_image)
img=img.reshape(1,32,32,1)
img=img/255.0
img=1-img
plt.imshow(testing_image)
lab=classes[np.argmax(new_model.predict(img))]
plt.show()
return lab
print(f"Predicted Label {pred(testing_image)}")
```



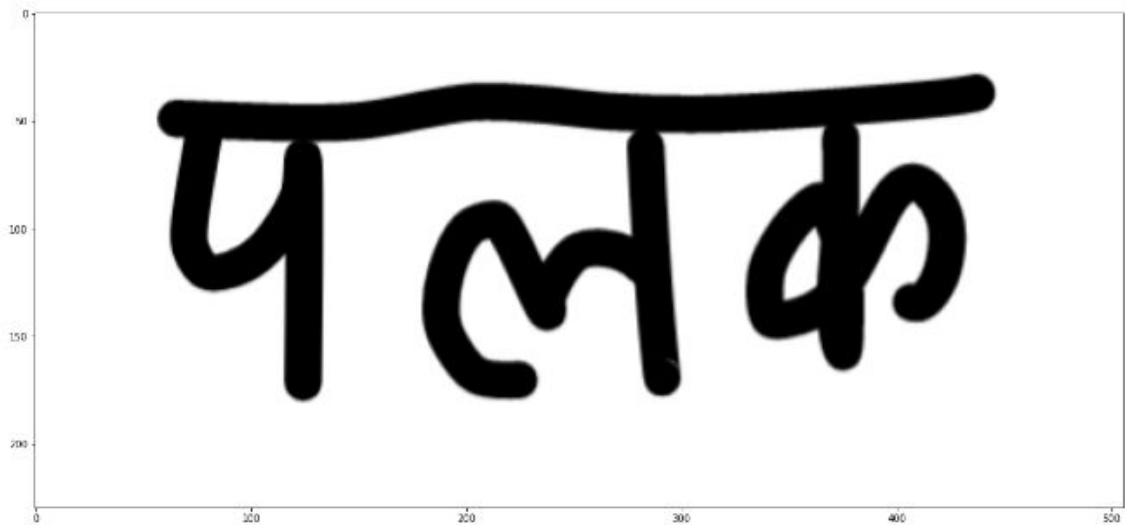
Predicted Label motosaw

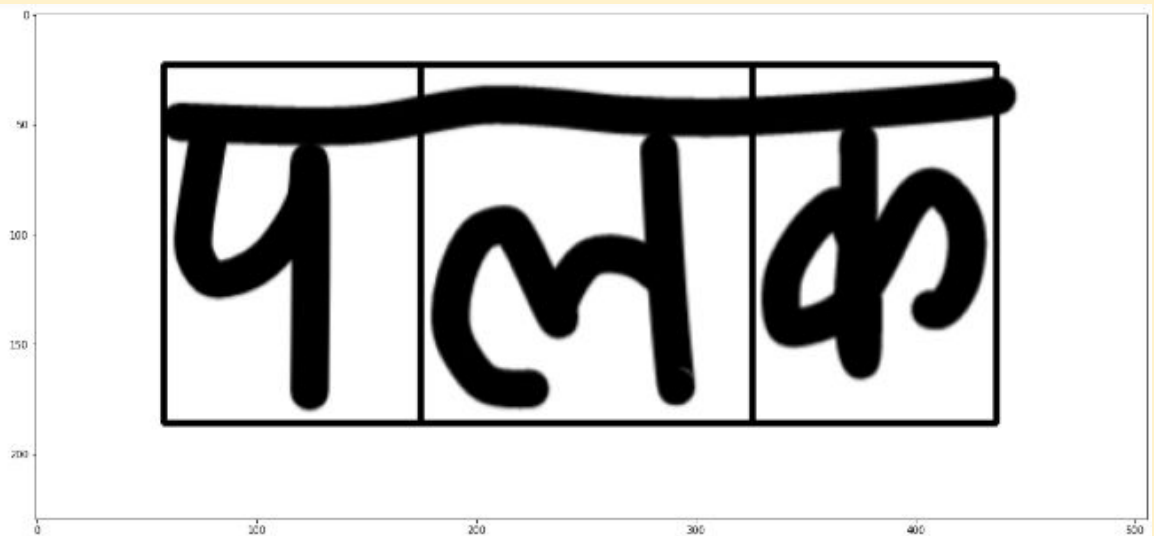
2. Predicting a whole word (in jupyter-notebook)

```
In [30]: from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import cv2
from keras.models import load_model
```

```
In [54]: image_location="Testing_images/Screenshot_2020-12-08 Sketchpad 2020 - Draw, Create, Share (7).png"
```

```
In [55]: img=cv2.imread(image_location,cv2.IMREAD_GRAYSCALE)
def show(img, figsize=(20, 20)):
    fig = plt.figure(figsize=figsize)
    plt.imshow(img, cmap="gray")
    plt.show()
show(img)
```





```
In [59]: def prediction(img):
# load json and create model
loaded_model=load_model("dcr.hdf5")
characters = ['क', 'ख', 'ग', 'घ', 'ङ', 'च', 'छ', 'ज', 'झ', 'झ', 'ट', 'ठ', 'ड', 'ढ', 'ण', 'त', 'थ', 'द', 'ध', 'न', 'प', 'फ', 'ब', 'भ', 'म', 'य', 'र', 'ल', 'व', 'श', 'ष', 'स', 'ह', 'ळ', '०', '१', '२', '३', '४', '५', '६', '७', '८', '९']

x = np.asarray(img, dtype = np.float32).reshape(1, 32, 32, 1) / 255.0
#x=1-x

output = loaded_model.predict(x)
output = output.reshape(46)
predicted = np.argmax(output)
devanagari_label = characters[predicted]
success = output[predicted] * 100

return devanagari_label, success
```

```
In [60]: def classifier(segments):
pred_lbl = ""
acc = []
for segment in segments:
segment = cv2.resize(segment, (32, 32))
segment = cv2.GaussianBlur(segment, (3, 3), 0)
segment = cv2.erode(segment, (3, 3), 1)
#show(segment)

lbl, a = prediction(segment)
pred_lbl+=lbl
acc.append(a)
return pred_lbl, np.array(acc).mean()
classifier(segments[0])
```

WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.

Out[60]: ('फलक', 99.98141527175903)

3. Predicting a whole word (using web application)

```
Terminal
rashi@rashi-Vostro-3578: ~/NeuralNet/TensorflowProjects/DevnagriOCR/DCRwebapp
(base) rashi@rashi-Vostro-3578:~/NeuralNet$ cd TensorFlowProjects
(base) rashi@rashi-Vostro-3578:~/NeuralNet/TensorflowProjects$ ls
data.csv  DevnagriOCR  Pnpfile  templates
dataset   'DevnagriOCR (copy)'  file1.csv
(base) rashi@rashi-Vostro-3578:~/NeuralNet/TensorflowProjects$ cd DevnagriOCR
(base) rashi@rashi-Vostro-3578:~/NeuralNet/TensorflowProjects/DevnagriOCR$ cd DCRwebapp
(base) rashi@rashi-Vostro-3578:~/NeuralNet/TensorflowProjects/DevnagriOCR/DCRwebapp$ ls
app.py  dcr.hdf5  Pnpfile  templates
(base) rashi@rashi-Vostro-3578:~/NeuralNet/TensorflowProjects/DevnagriOCR/DCRwebapp$ python3 app.py
* Loading Keras model.....
2020-12-28 20:47:56.371733: I tensorflow/core/platform/cpu_feature_guard.cc:143] Your CPU supports instructions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2 AVX AVX2 FMA
2020-12-28 20:47:56.396199: I tensorflow/core/platform/profile_utils/cpu_utils.cc:102] CPU Frequency: 1800000000 Hz
2020-12-28 20:47:56.397031: I tensorflow/compiler/xla/service/service.cc:168] XLA service 0x5583e0382000 initialized for platform Host (this does not guarantee that XLA will be used). Devices:
2020-12-28 20:47:56.397072: I tensorflow/compiler/xla/service/service.cc:176] StreamExecutor device (0): Host, Default Version
2020-12-28 20:47:56.397255: I tensorflow/core/common_runtime/process_util.cc:147] Creating new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads for best performance.
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
* Model loaded successfully
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [28/Dec/2020 21:19:35] "GET / HTTP/1.1" 200 -
app.py:168: DeprecationWarning: The binary mode of fromstring is deprecated, as it behaves surprisingly on unicode inputs. Use frombuffer instead
  nparr=np.fromstring(decoded,np.uint8)
127.0.0.1 - - [28/Dec/2020 21:19:44] "POST /predict HTTP/1.1" 200 -
```



Applications of Devanagari Character Recognition

- **Digitizing Books:** It can be used to digitize books written in Devanagari. Digitizing books can provide a large number of advantages such as searching through the book, cost reduction, and easy storage.
- **Indexing of Images** in Search Engines: Most of the sites often use images to represent Devanagari text. The reissue of search engines which can search for keywords provided in Devanagari Script. For the efficient functioning of the search engines it becomes necessary for them to include some software to recognize Devanagari text from websites.
- **Recognizing addresses** on envelopes in post offices: use of such software could be in recognizing Devanagari addresses on envelopes in post offices, thus automating the overall process.
- **Use for those who don't know Hindi:** This software, if added with the capability of transliteration/ translation can prove quite useful for many people who don't understand Hindi but want to read some book written in Hindi. It would allow the books to be easily translated into other languages.

Future Scope

- Online character Recognition
- Printed Text Recognition
- Handwriting Recognition
- Language Recognition
- Graphics Document Recognition
- Document Understanding
- Tables and Forms Processing
- Document Engineering

References

- [Kaggle Database](#)
- [Tensorflow Tutorials](#)
- [Deep Learning and feed forward neural network](#)