Capstone Report

Handwritten Devnagri Character & Numeral Recognition using Deep Learning

Nisha Gadhe
May 2018

I. Definition

Project Overview

Handwriting recognition (or HWR) is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. The image of the written text may be sensed "offline" from a piece of paper by optical scanning (optical character recognition) or intelligent word recognition. Alternatively, the movements of the pen tip may be sensed "online", for example by a pen-based computer screen surface, a generally easier task as there are more clues available.

Devanagari is part of the Brahmic family of scripts of Nepal, India, Tibet, and South-East Asia. The script is used to write Nepali, Hindi, Marathi and similar other languages of South and East Asia. The Nepalese writing system adopted from Devanagari script consists of 12 vowels, 36 base forms of a consonant, 10 numeral characters and some special characters. Devanagari is used to write in India and Nepal.

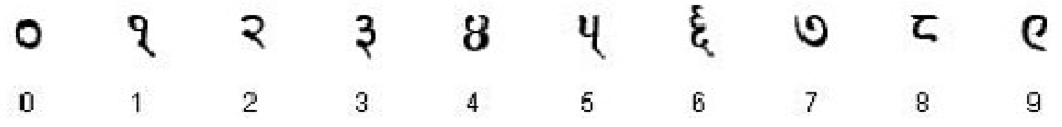


Fig 1. Devanagari – Numerals

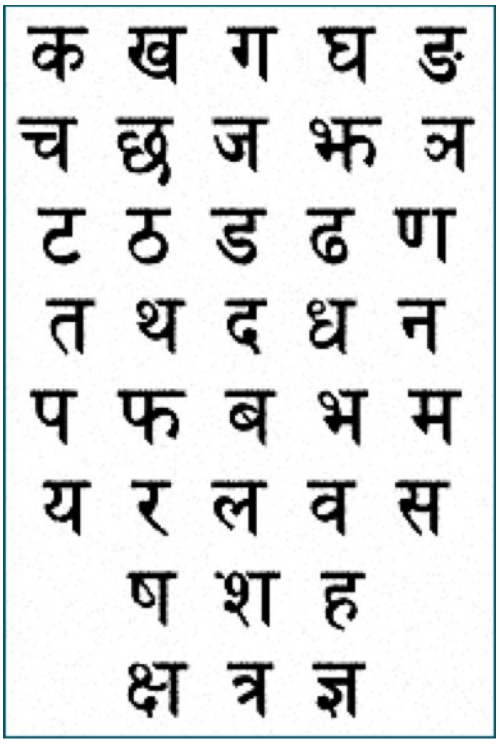


Fig 2. Devanagari Consonants

Nowadays, techniques such as deep neural networks are successfully used on Devanagari characters & digits. We propose here to apply some of these techniques to the Devanagari characters, used to write in India and Nepal.

A deep convolutional neural network(CNN) has multiple convolutional layers to extract the features automatically. The features are extracted only once in most of the shallow learning models, but in the case of deep learning models, multiple convolutional layers have been adopted to extract discriminating features multiple times. This is one of the reasons that deep learning models are generally successful.

Character recognition is a field of image processing where the image is recognized and converted into a machine-readable format. As discussed above, the deep learning approach and especially deep convolutional neural networks have been used for image detection and recognition.

In this project, we are going to apply Deep Convolutional Neural Network(CNN) techniques for the correct recognition of handwritten Devanagari characters & numeral in an image.

Citation

This dataset was originally created by Computer Vision Research Group, Nepal. [website archive] (https://web.archive.org/web/20160105230017/http://cvresearchnepal.com/wordpress/dhcd/)

Problem Statement

Here Our goal is to predict the handwritten Devanagari characters & numeral characters in an image. The dataset contains Devanagari Characters. It comprises of 92000 images [32x32 px] corresponding to 46 characters, consonants "ka" to "gya", and the digits 0 to 9. The vowels are missing. The CSV file is of the dimension 92000 * 1025. There are 1024 input features of pixel values in grayscale (0 to 255). The column "character" represents the Devanagari Character Name corresponding to each image. There are 46 classes present out of which 10 classes for digits & 36 classes for characters & each class have 2000 images which can be used to train and test the model.

This is the supervised learning problem more specifically a classification problem. The model should be able to classify which Devanagari number (0 – 9) & consonants is handwritten in the image. The model can be scored for its ability to predict the number correctly over large different test data and real data. In this work, a deep convolutional neural network is applied for handwritten Devanagari characters recognition.

The main contributions of my work can be summarized in the following points:

1) To load necessary Python libraries and read/load the Kaggle Downloaded csv data through pandas as discussed in the data exploration.

2) The loaded data will be first explored and visualized using numpy and matplotlib library to understand the nature of the data. Exploring the data will help us in deciding how to approach and whether any preprocessing of the data is needed. Preprocessing of the data is done as required.

3) Once the data is ready, the Deep convolutional neural network(CNN) will be built based on the architecture (ConvNet) as discussed in the Implementation Section. Once the model is built, it will be compiled to check if the architecture has any error.

4) Then the compiled model will be trained on the training data and evaluated using accuracy score against the testing data. Then the results can be analyzed and compared with respect to the benchmark model to know the overall performance of the model.

5) Now we have a trained model, a test is conducted against the model by loading images of Devanagari characters which are not from kaggle downloaded dataset to evaluate the performance of the final model.

6) The images are loaded and then preprocessed to match the Kaggle downloaded dataset format so that we can test it. The modelis then made to predict the Devanagari consonant& numeral characters in the preprocessed image.

Thus, now we can evaluate our model’s performance over real-time data.

Metrics

Like any other n-class image classification problem, the metric of central importance for us is accuracy. The final goal is to improve accuracy on the validation set.

For our refined DNN model, we compiled model with 'categorical_crossentropy' as loss funtion, 'adam'(Adaptive Moment Estimation) as optimizer and 'accuracy' as metrics. We achieved accuracy on testing data set is 97.38%.

Accuracy (fomula defined below) is the most relevent metric for us because, it tells us how accurately the model performs on the unseen data.
Accuracy is the proportion of samples predicted correctly among the total number of samples examined. For Example, in our problem, it is the ratio of the handwritten Devanagari characters predicted correctly to the total number of handwritten Devanagari characters evaluated. It measures the correctness of a model. The final model has predicted the Devanagari characters of 13 out 14 images correctly

$$\textbf{Accuracy} = \frac{\textbf{Number of samples predicted correctly}}{\textbf{Total number of samples examined}}$$

II. Analysis

Data Exploration

Devanagari handwritten character dataset is created by collecting the variety of handwritten Devanagari characters from different individuals from diverse fields. Handwritten documents are then scanned and cropped manually for individual characters. Each character sample is 32x32 pixels and the actual character is centered within 28x28 pixels.

Padding of 0 valued 2 pixels is done on all four side to make this increment in image size. The images were applied grayscale conversion. After this, the intensity of the images was inverted making the character white on the dark background. To make uniformity in the background for all the images, we suppressed the background to 0 value pixel. Each image is a gray-scale image having background value as 0.

Devanagari Handwritten Character Dataset contains a total of 92,000 images with 72,000 images in consonant datasets and 20,000 images in the numeral dataset. Handwritten Devanagari consonant character dataset statistics are shown in Table I and handwritten Devanagari numeral character dataset statistics is shown in Table II.

Table I - Consonants Character Dataset

Class	ka	kha	ga	gha	kna	cha	chha	ja	jha	yna	taamatar	thaa
#	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000
Class	daa	dhaa	adna	tabala	tha	da	dha	na	pa	pha	ba	bha
#	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000
Class	ma	yaw	ra	la	waw	motosaw	petchiryakha	patalosaw	ha	chhya	tra	gya
#	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000
Total	72000											

Table II - Numeral Dataset

Class	digit_0	digit_1	digit_2	digit_3	digit_4	digit_5	digit_6	digit_7	digit_8	digit_9
#	2000	2000	2000	2000	2000	2000	2000	2000	2000	2000
Total	20000									

The dataset can be read by using pandas. Also we can re-check again whether in the dataset imbalance class is present or not.

```
import pandas as pd
dataset = pd.read_csv("/home/nisha/dlnd/Hand-writing-recog/input/data.csv")
dataset.groupby("character").count()
```

Also this dataset not having any unusual properties like outliers, blurry & out of focus images. It would be more clear if we check Exploratory Visualization section.

Exploratory Visualization

Let’s visualize all image of the Devanagari consonants & numeral characters in the training set of downloaded kaggle dataset.
Each character sample is 32x32 pixels and the actual character is centered within 28x28 pixels.

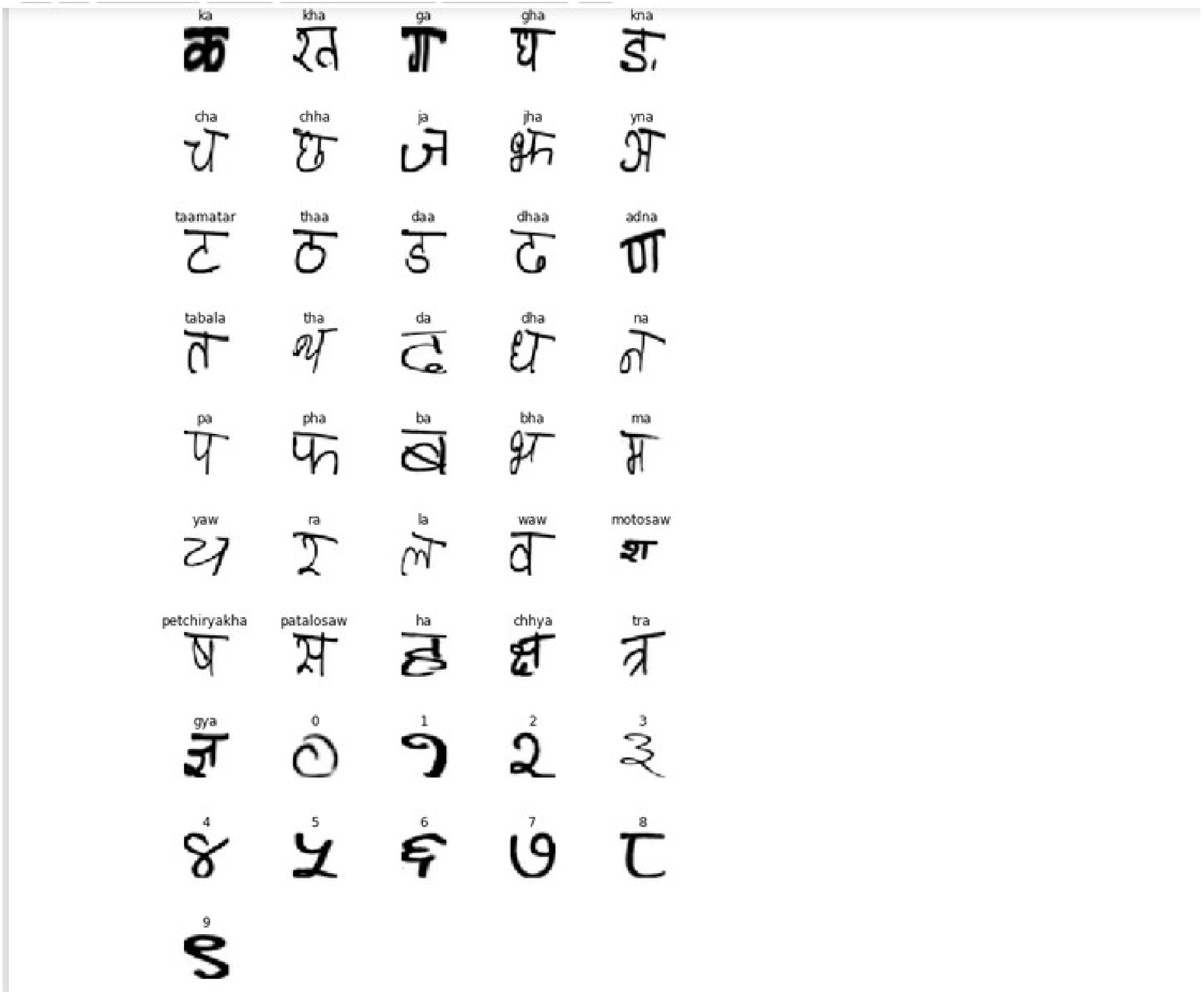


Fig 3. Visualization of all Devanagari Consonants & Numerals Character in the dataset

Let us verify the pixel distribution of any random character using matplotlib.

Plotting a histogram can give you a quick visualization of your data distribution. It is important to select the correct 'bin' size (groups of data) to get the best curve approximation.

This plot will show you if your data values are centered (normally distributed), skewed to oneside or the other, or have more than one 'mode' - localized distribution concentrations.

They can also be rearranged as a Pareto Plot from highest frequency to lowest, allowing you to focus on the most important factors to address in problem solutions.

One of the more common is to decide what value of threshold to use when converting a grayscale image to a binary one by thresholding. If the image is suitable for thresholding then the histogram will be bi-modal --- i.e. the pixel intensities will be clustered around two well-separated values.

This histogram is a graph showing the number of pixels in an image at each different intensity value found in that image. For an 8-bit grayscale image there are 256 different possible intensities, and so the histogram will graphically display 256 numbers showing the distribution of pixels amongst those grayscale values.

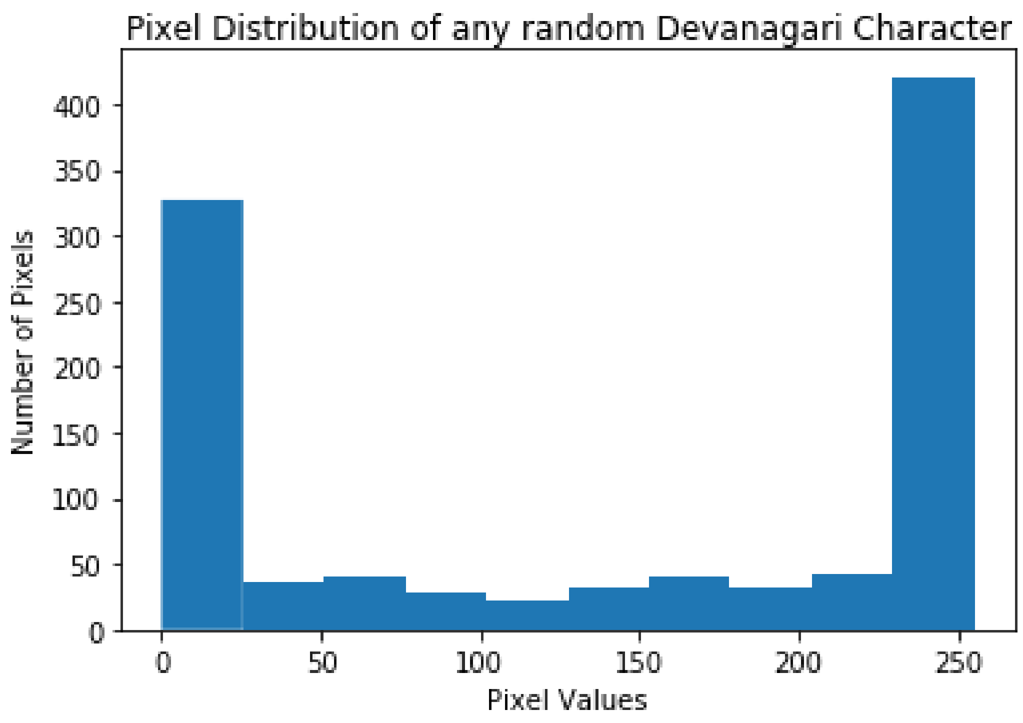


Fig. 4 Pixel Distribution

The above histogram shows the number of pixels for every pixel value, ie from 0 to 255.
In this way this visualization will help us indata preprocessing

Algorithms and Techniques

Given that the problem is a supervised learning problem and more specifically a classification problem, there are lot of algorithms available for training a classifier to learn from the data. The algorithm chosen for this project is Deep Neural Network(DNN) using Convolutional Neural Network(ConvNet).

Convolutional Neural Networks
Convolutional Neural Network (CNN or ConvNet) is a biologically-inspired trainable machine leaning architecture that can learn from experiences like standard multilayer neural networks. ConvNets consist of multiple layers of overlapped tiling collections of small neurons to achieve better representation of the original image. ConvNets are widely used for image recognition. A CNN consists of a lot of layers.These layers when used repeatedly, leading to a formation of a Deep Neural Network. Three main types of layers used to build a CNN are:

1) First Convolution Layer:
The convolution layer is the core building block of a convolutional neural network. It convolves the input image with a set of learnable filters or weights, each producing one feature map in the output image.

2) ReLU Function:
The Rectified Linear Unit apply an elementwise activation function, such as the max (0, x) thresholding at zero.

3) Pooling Layer:
The pooling layer is used to progres-sively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network.The pooling layer takes small rectangular blocks from the convolution layer and subsamples it to produce a single output from that block. There are several ways to do this pooling, such as taking the average or the maximum, or a learned linear combination of the neurons in the block.

4) Fully Connected Layer:
The fully-connected layer is used for the high-level reasoning in the neural network. It takes all neurons in the previous layer and connects it to every single neuron it has. Their activations can be computed with a matrix multiplication followed by a bias offset as a standard neural networks.
An example of a Convolutional Neural Network is given below

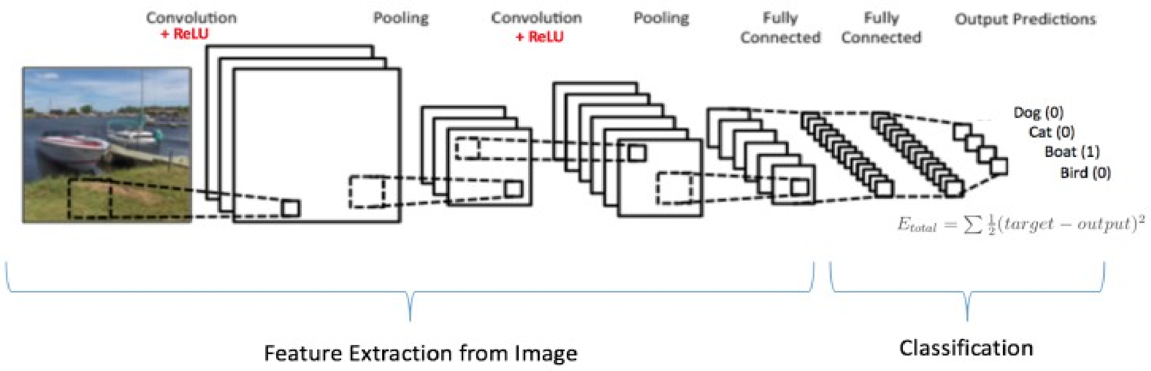


Fig 5. CNN Example

Benchmark

The benchmark model solves the same handwritten devnagari Consonants / Numeral characters recognition problem mentioned in this project by using CNN model.

The CNN has complexity in computations, it offers several advantages in pattern recognition and classification in the similar manner close to human intelligence to a small extent.

In this case, we compiled our initial CNN model with 'categorical_crossentropy' as loss funtion, 'sgd' (Stochastic Gradient Descent) as optimizer and 'accuracy' as metrics & achieved a test set accuracy of 80%. After that we added 2 dropout layer in our redefined model to achive more accuracy on test set data than initial model. We have described each model architect clearly in implementation section. For Redeined CNN model, we compiled it with 'categorical_crossentropy' as loss funtion, 'adam'(Adaptive Moment Estimation) as optimizer and 'accuracy' as metrics. We achieved a test set accuracy of 97.38%. We used, Adam Optimizer because it achieves good results fast on large models and datasets.

III. Methodology

Data Preprocessing

Yes, downloaded dataset needs data preprocessing in this project.

In previous sections, I have shown all different classes in dataset, character 'ka' to 'gya' & digit '0' to '9'. All these categorical features have string values.

Sklearn provides a very efficient tool for encoding the levels of a categorical features into numeric values. LabelEncoder encode labels with value between 0 and n_classes-1.

We can encode all the categorical features by using below code

le = LabelEncoder()

After all categorical features are get encoded, I have reshaped trained & test data images into 32x32.

Implementation

In this project implementation process divided into two steps

1. Build Model, Model training and evaluating stage
2. Model testing on real data

Let us check on the first step

1. Build Model, Model training and evaluating stage

In the first stage, the model was trained on the preprocessed training data and evaluated against the test data. The following steps are done during the first stage:
1. Buid Deep Neural Network (DNN) model with Keras.
2. Load both the training and testing data into memory and preprocess them as described in the above section .
3. Define the initial network architecture and training parameters as shown in the code and block diagram below in Fig. 6 & Fig. 7.
4. Define the loss function with metrics as Accuracy.
5. The network is traied on the trained on the training data and evaluated against the test data.
6. Note down the loss & accuracy score as well as accuracy curves
7. If the accuracy is not high enough, repeat from step 2 by architecture of the network &by changing the hy hyper parameters value and typing different loss function.

In the first stage, step 2,3 was difficult implementing since finding the optimum architecture, hyper-paramenters and loss function was time consuming becuase a single run takes 30-40 mins.

Initial CNN model code & block diagram

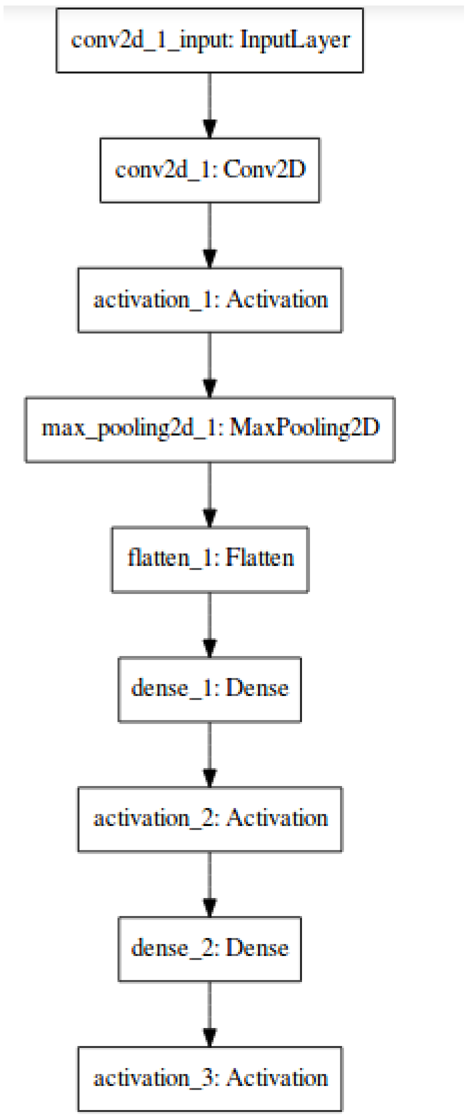
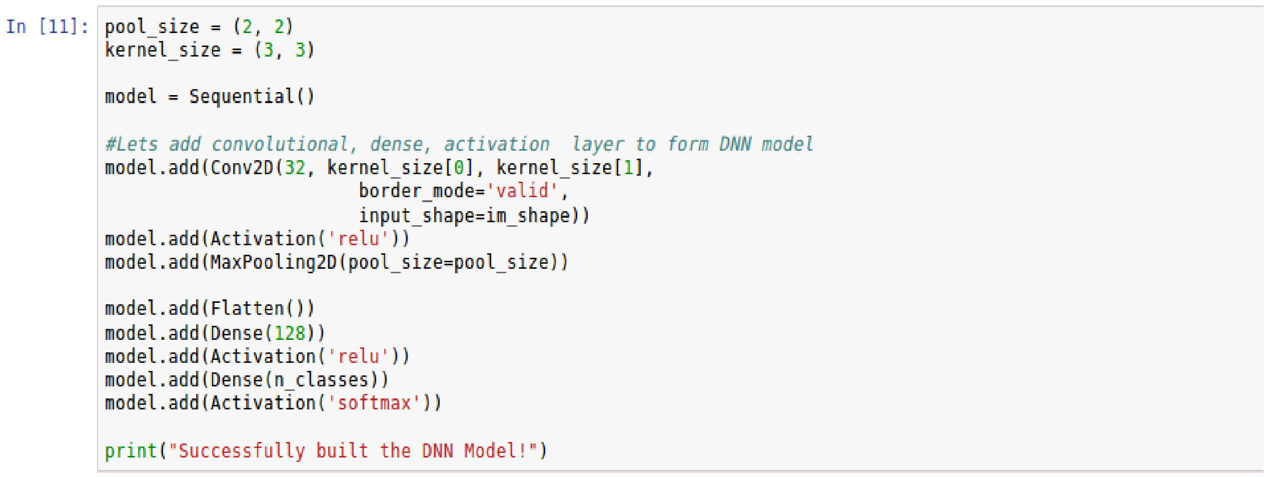


Fig. 6 Initial CNN Model Block Diagram



```
In [11]: pool_size = (2, 2)
         kernel_size = (3, 3)

         model = Sequential()

         #Lets add convolutional, dense, activation  layer to form DNN model
         model.add(Conv2D(32, kernel_size[0], kernel_size[1],
                                 border_mode='valid',
                                 input_shape=im_shape))
         model.add(Activation('relu'))
         model.add(MaxPooling2D(pool_size=pool_size))

         model.add(Flatten())
         model.add(Dense(128))
         model.add(Activation('relu'))
         model.add(Dense(n_classes))
         model.add(Activation('softmax'))

         print("Successfully built the DNN Model!")
```

Fig. 7 Initial CNN Model Built Code

Initial CNN Model Description

The first layer of the model is the Input Layer which gets the input data as arrays during the training phase. The next layer is the convolutional layer with 32 filters and 3,3 kernel size which convolutes the input image. The next layer is an activation Layer made of Rectified Linear Unit (ReLU) which acts as an elementwise activation function. The nextlayer is the max pooling layer where down sampling is carried out. Then a flatten layer is used to flatten the data. Then A dense or fully connected layer of size 128 is applied followed by an activation layer.

The next layer is fully connected layer of size 10 i.e. it is the output size followed by a softmax activation layer. The output of the final activation layer is the final output. Once the model is built using the above architecture, it is compiled by specifying the loss function as categorical cross entropy, optimizer as SGD(Stochastic Gradient Descent), and metrics as accuracy. Once the modelis complied with no errors, it is trained over the training dataset x_train and y_train. The model is then evaluated against thetest dataset x_test and y_test. The accuracy score of the initial model is measured to be 80%.

2. Model testing on real data

During the second stage, the final model is chosen and is tested against real data rather than the downloaded kaggle dataset itself.

The following steps are involved in this stage:
1. Download or scan or take photos of handwritten Devanagari consonants & numerals character.
2. Load the images into the program from a folder using OpenCV python library
3. Format the images in order to match with Dataset i.e. 32x32 pixel grayscale image
4. Make the model predict the new images and verify the performance and robustness of the model.

In the second stage, step 3 was a bit tough as OpenCV (advanced image processing) library is downloaded and setup for just two operations(resizing and converting to grayscale format)

Refinement

The initial model is improved by using the following technique
1) It is made further deep by adding two more convolutional layer and one more extra dense (fully connected) layer.
2) Two dropout layer is added to ensure that overfitting does not happen
3) Using Adam optimizer instead of SGD(Stochastic Gradient Descent) to update the weights efficiently.
4) Fig. 8 & 9 illustrate the whole refine cnn model

Once the model is built using the above architecture, it is compiled by specifying the loss function as categorical cross entropy, optimizer as Adam, and metrics as accuracy. Once the model is complied with noerrors, it is trained over the training dataset x_train and y_train. The model is then evaluated against the test dataset x_test and y_test. The accuracy score for refined model is measured to be 97.38%.

Refined CNN model block diagram & code

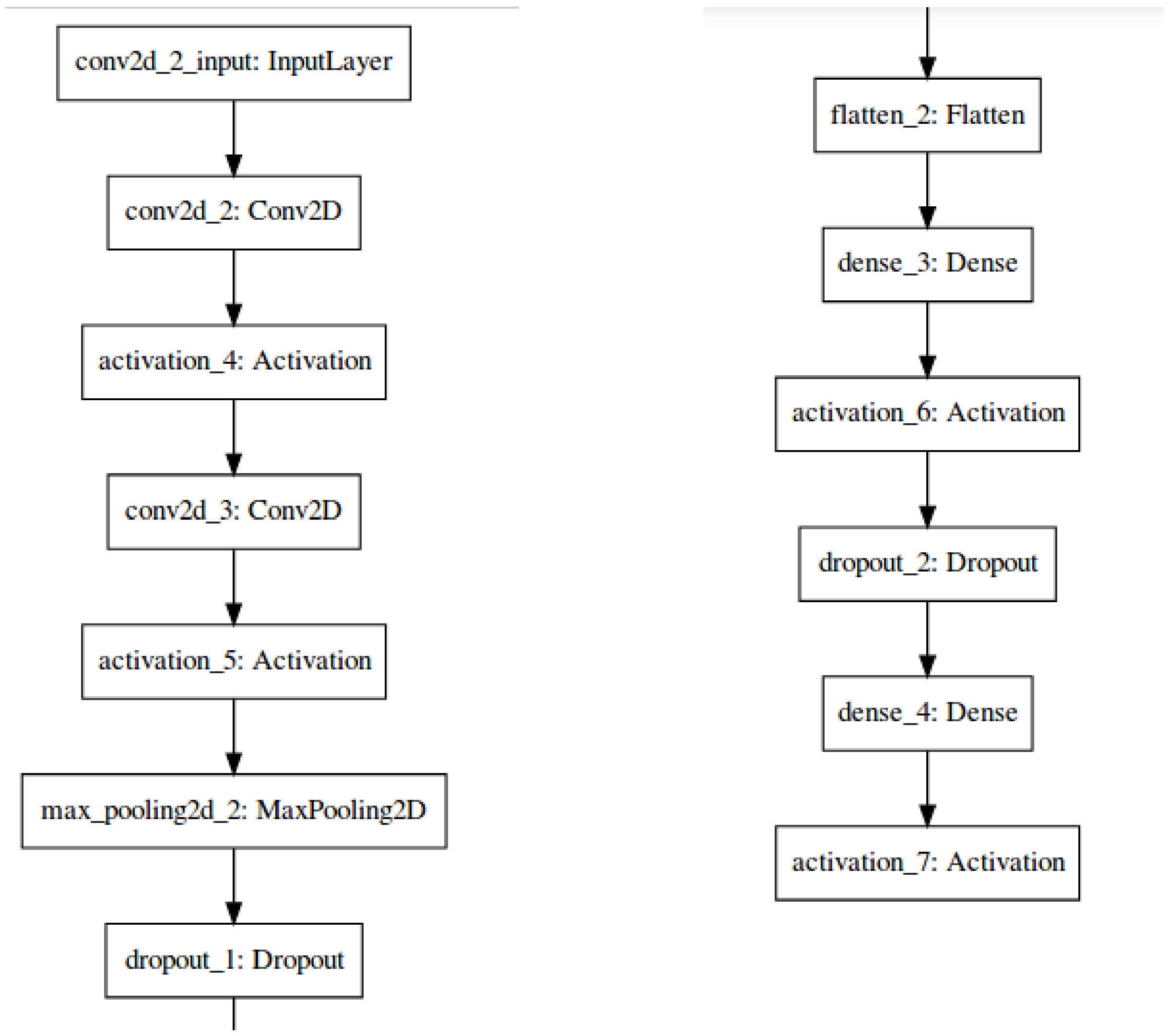
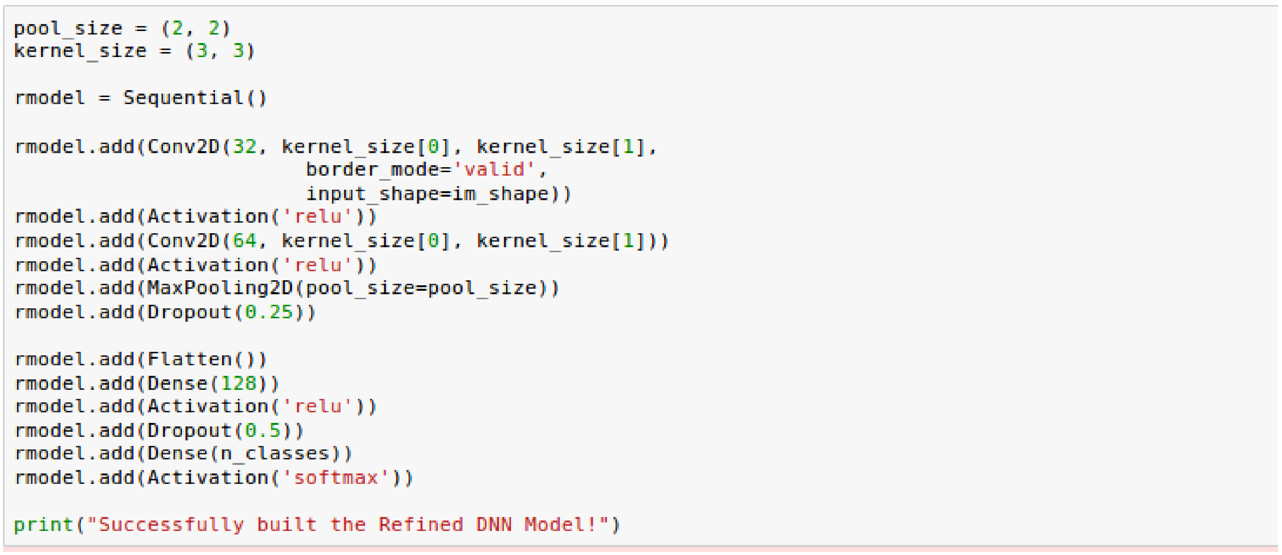


Fig. 8 Refined CNN model block diagram



```
pool_size = (2, 2)
kernel_size = (3, 3)

rmodel = Sequential()

rmodel.add(Conv2D(32, kernel_size[0], kernel_size[1],
                        border_mode='valid',
                        input_shape=im_shape))
rmodel.add(Activation('relu'))
rmodel.add(Conv2D(64, kernel_size[0], kernel_size[1]))
rmodel.add(Activation('relu'))
rmodel.add(MaxPooling2D(pool_size=pool_size))
rmodel.add(Dropout(0.25))

rmodel.add(Flatten())
rmodel.add(Dense(128))
rmodel.add(Activation('relu'))
rmodel.add(Dropout(0.5))
rmodel.add(Dense(n_classes))
rmodel.add(Activation('softmax'))

print("Successfully built the Refined DNN Model!")
```

Fig. 9 Refined CNN model code

IV. Results

Model Evaluation and Validation

The model is evaluated against the test set (x_test,y_Test). The final architecture and hyper-parameter are chosen because they performed best out of the other models tried. The robustness of the final model is verified by conducting a test against the final model using different images of Devanagari Consonants & Numerals character other than Downloaded kaggle dataset itself as shown in Fig. 10.

The following observations are based on the results of the test:
1) The final model has predicted the Devanagari characters of 13 out 14 images correctly.

2) Our model had a problem in predicting Devanagari Consonants & Numerals that have thin stokes like the consonant character 'yna' with thin stroke in the test cases

The observations of test confirms that our model is reliable and robust enough to perform well on real data.
Use of dropout layer in the final model ensures that small changes in training data or the input space do not affect the results greatly.

Justification

In order to justify that our model has potentially solved the problem, we need to look at the results of the real-time test conducted against model as shown in Fig. 10. Our model can correctly predict 13 out of 14 tests. Thus, our model solved the problem with a good accuracy score and reliability in real-time applications.

Lets revisit our performance summary.

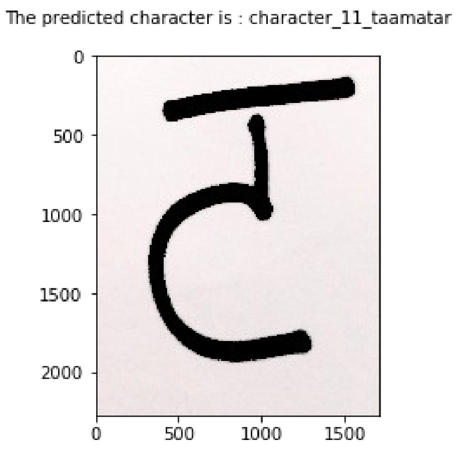
- Test data set Accuracy of 80% of initial model
- Test data set Accuracy of 97.38% of refined model
- The final model has predicted the Devanagari characters(not from the dataset) of 13 out 14 images correctly. Thus our model has performed well enough.

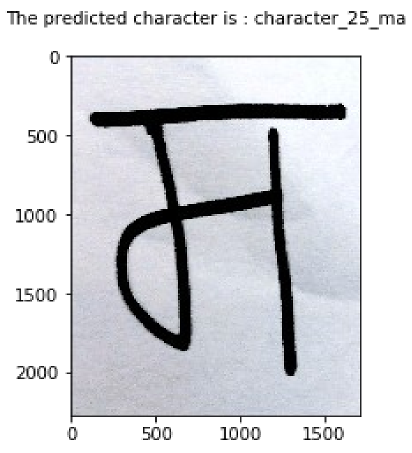
We have achieved an accuracy of 97.38% on test set. While it is ~6% away from human performance, it should be considered good because, It performed extremely well on unseen images (not a part of original dataset).

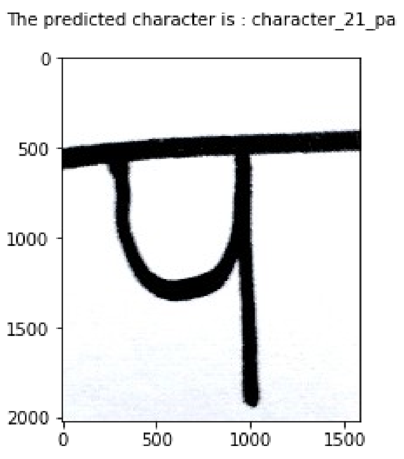
V. Conclusion

Free-Form Visualization

Devanagari Consonants Character :



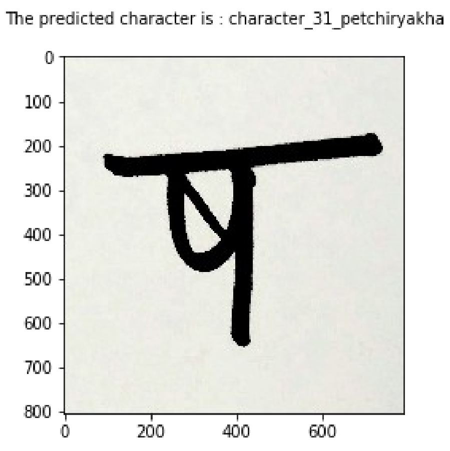


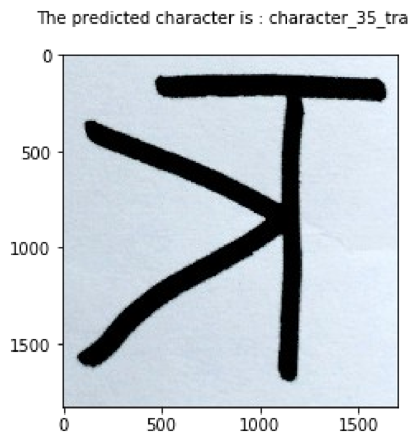


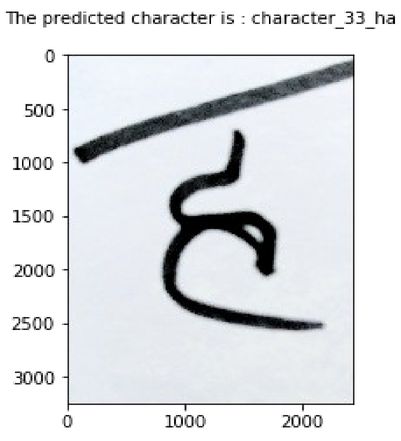
1. Character= taamatar

2. Character= ma

3. Character = pa



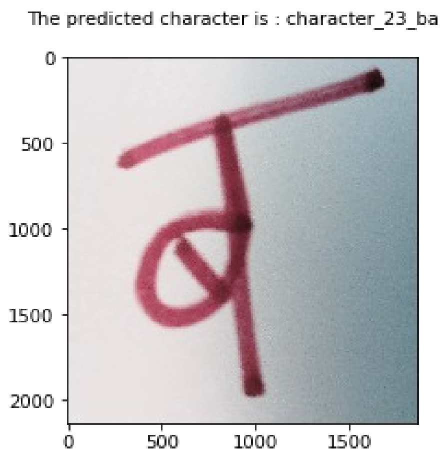


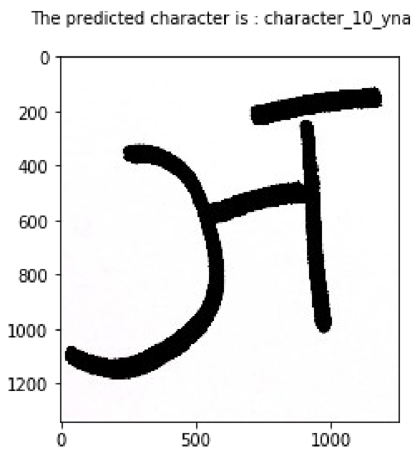


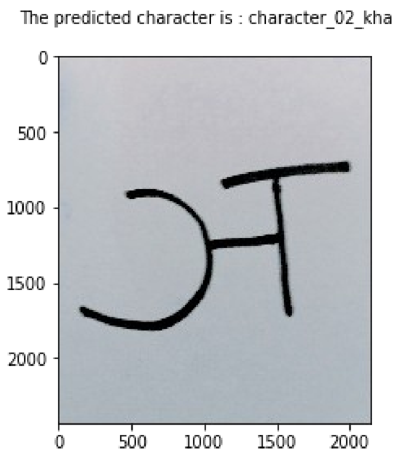
4. Character= petchiryakha

2. Character= tra

3. Character = ha





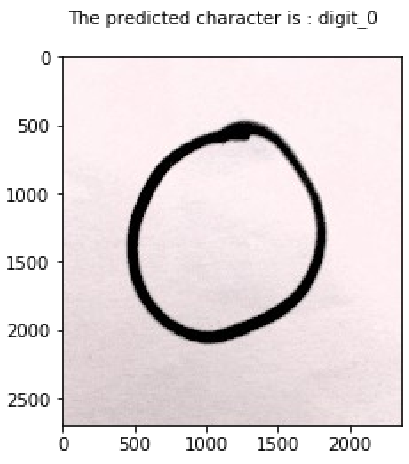


7. Character= ba

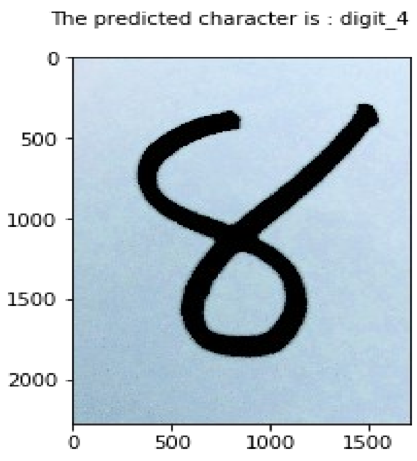
8. Character= yna

9. Character = yna(Wrongly predicted)

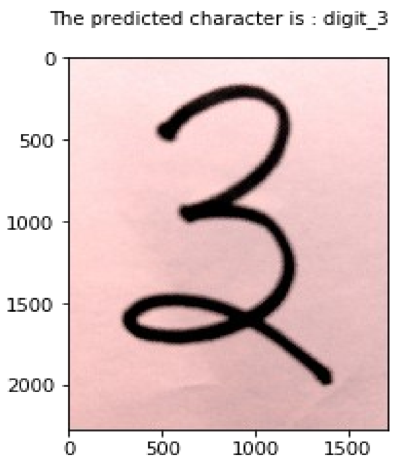
Devanagari Numerals Character :



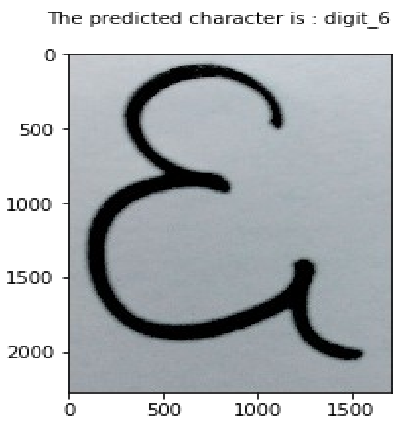
1. Numeral= 0



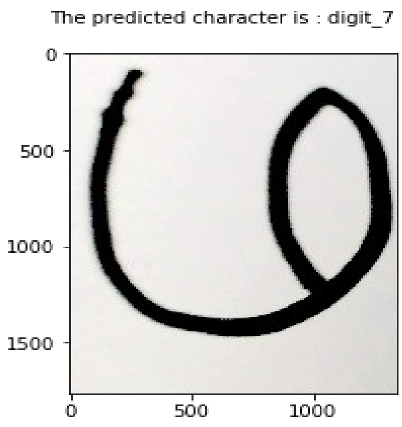
2. Numeral= 4



3. Numeral = 3



4. Numeral= 6



5. Numeral= 7

Fig. 10

Fig. 10 Results of a test conducted by loading images of Devanagari consonant & numeral characters which are not from kaggle dataset to evaluate the performance of the finalmodel. From the above results, we can see that our final model has predicted the Devanagari consonant & numeral characters of 13 out 14 images correctly. Thus our model has a good accuracy score of 97.38% and good reliability as seen in the above test.

Reflection

The process involved in this project can be summarized using the below steps:

1. Prepared a problem statement & relevant database found on kaggle.
2. The kaggle dataset was downloaded and loaded in the code
3. The data is preprocessed as required
4. A deep neural network model using the convolutional neural network is built

5. The model is trained using the training dataset and evaluated against the test dataset multiple times in order to find an optimum model architecture with good hyper-parameter
6. A test is conducted against the refined model by loading images of Devanagari consonant & numeral characters which are not from kaggle dataset to evaluate the performance of the final model.

I found steps 4 and 5 to be tough as I was new to Deep Learning and it took a while for me to grasp the concept. However usingthe Kaggle dataset and keras library simplified my process a lot.

There was lots of tutorial & guide availble to understad deep convlolutional network easily.
Also Keras helped me in kick-starting into deep learning without writing complex code which was really helpful in quick prototyping and experimenting in order to build a better model.

Improvement

The final model in this project uses only simple Convolutional Neural Network architecture.
There are more advanced architecture like the Deep Residual Neural Network(ResNet) and VGG-16 which could be used to improve the accuracy a little. Also we could hyperas library to fine tune the hyper-parameters and architecture of the model.

References

1. https://en.wikipedia.org/wiki/Handwriting_recognition
2. https://en.wikipedia.org/wiki/Cross_entropy
3. https://en.wikipedia.org/wiki/Convolutional_neural_network
4. https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/
5. https://www.quora.com/What-is-dropout-in-deep-learning
6. https://keras.io/layers/core/