Bangla Language OCR using CNN

Sanjiv Roy
October, 2017

I. Definition

Project Overview

The purpose of this project is to apply Convoluted Neural Network(CNN) architecture to the problem of handwritten Bengali character recognition (HBCR) and achieve accuracy of over 95%. To my knowledge, the highest accuracy on the CMTRADB 3.2.1 is 95.84% and the objective is to surpass the same.

Bangla is one of the most spoken languages, ranked fifth in the world. It is also a significant language with a rich heritage; February 21st is announced as the International Mother Language day by UNESCO to respect the language martyrs for the language in Bangladesh in 1952. Bangla is the second most spoken language in India and it's the first language of Bangladesh.

CNNs have been extensively and successfully used for English character recognition however there exist immense opportunity and potential to apply the same in the Bengali language space.

The main challenge in handwritten character classification is to deal with the enormous variety of handwriting styles by different writers in different languages. Furthermore, some of the complex handwriting scripts comprise different styles for writing words. Depending on languages, characters are written isolated from each other in some cases, (e.g., Thai, Laos and Japanese). In some other cases, they are cursive and sometimes the characters are connected with each other (e.g., English, Bengali and Arabic). Bengali handwritten character recognition is particularly challenging owing to the similarities in different character shapes with distinct sounds.

Problem Statement

The Bengali character recognition is a classification problem where the aim is to learn the features of the 50 basic characters - 11 vowels and 39 consonants so that when a handwritten alphabet is input the system should classify the same as the right alphabet. The complexity Bengali character recognition stems from the fact that there exist a number of character pairs that are almost identical, differing only by a dash or a dot (for instance য, ষ and ব are three entirely different alphabet that sounds like Ja, Sha and Ba respectively).

Input: Image file of a Bengali handwritten character and Output will be the correct classification of the character. The table below shows the Bengali characters in printed form along with the phonetics in English.

Please note that the 50 basic characters can be further combined to create several complex alphabets with mixed sound but this is out of scope of the current project.

Vowels:

1	2	3	4	5	
অ	আ	ই	ঈ	উ	
A	AA	I	II	U	
6	7	8	9	10	11
ঊ	ঋ	এ	ঐ	ও	ঔ
UU	R	E	AI	O	AU

Consonants:

12	13	14	15	16
ক	খ	গ	ঘ	ঙ
KA	KHA	GA	GHA	NGA
17	18	19	20	21
চ	ছ	জ	ঝ	ঞ
CHA	CHHA	JA	JHA	NYA
22	23	24	25	26
ট	ঠ	ড	ঢ	ণ
TTA	TTHA	DA	DDHA	NNA
27	28	29	30	31
ত	থ	দ	ধ	ন
TA	THA	THE	DHA	NA
32	33	34	35	36
প	ফ	ব	ভ	ম
PA	PHA	BA	BHA	MA
37	38	39	40	41
য	র	ল	শ	ষ
YY	RA	LA	SHA	SSA
42	43	44	45	46
স	হ	ড	ঢ়	য়
SA	HA	DDHA	DHRDA	YYA
47	48	49	50	
ৎ	◌়	:	ঁ	

KHAND	ANUSWARA	BISARGA	BINDU

Metrics

The model will be evaluated using F1 score.

F1 score is the harmonic mean of precision and recall. Precision is the ratio of true positives to all positive predictions. Recall is the ratio of true positives to all positive examples.

F1 Score = (2*Precision*Recall)/(Precision + Recall)

Where:

Precision = true positives/(true positives + false positives)

and

Recall = true positives/ true positives + false positives)

II. Analysis

Data Exploration

In this project, I have used the publicly available CMATERdb pattern recognition database repository for training and testing the model. The dataset consists of 15000 unconstrained handwritten isolated Bangla characters split into training and test sets. The train and test sets consists of 12,000 and 3,000 greyscale images of varying intensity and orientations. The 50 alphabets are evenly distributed in both the sets. A sample image from each alphabet class is shown in Fig – 2. There is no significant noise that can be seen on visual inspection. However, variability in writing style due to user dependency is quite high. I could see some characters in different classed which are hard to distinguish even by human standard (see Fig - 3 for illustration). In the training set we have 240 images for each character and the test set has 60 images for each (Fig-1).

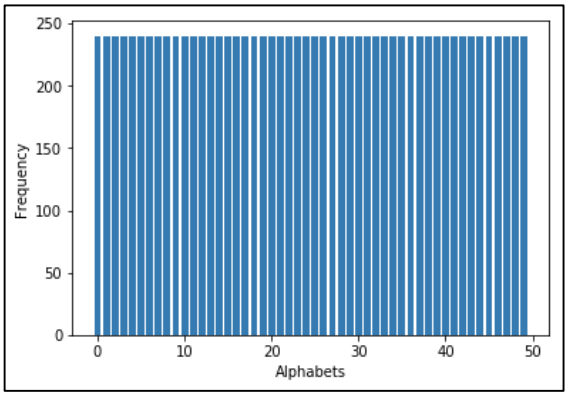
Data Analysis:

There are 50 total alphabets
There are 15000 total alphabet images.

There are 12000 training alphabet images.
There are 3000 test alphabet images.

Distribution of Training and Test Data:

Training Set



Test Set

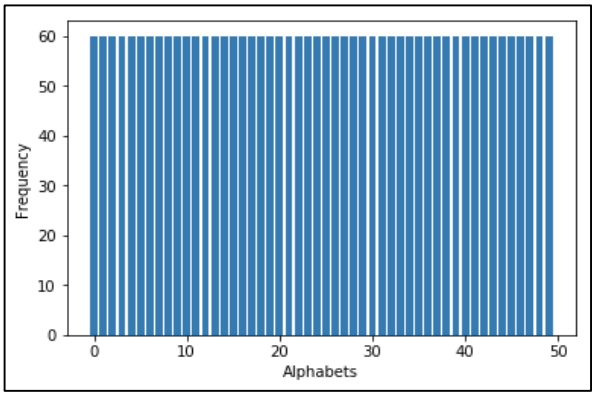


Figure 1

Alphabets from the dataset:

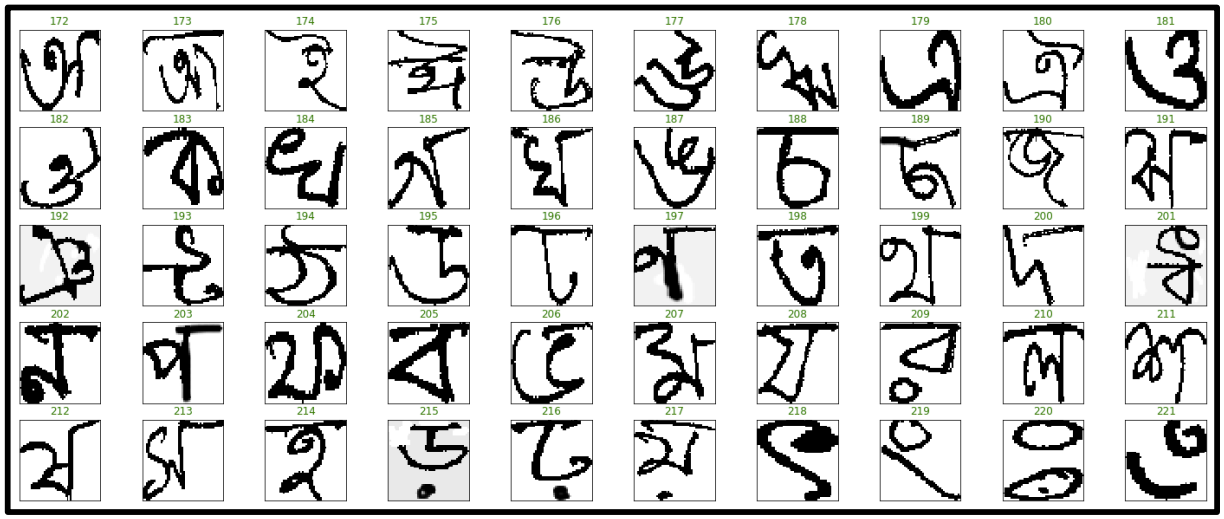


Figure 2

The first set of 10 characters are ৭ and(chr#25 in folder 197) the second set of 10 characters are ২ (chr#30 in folder 202)

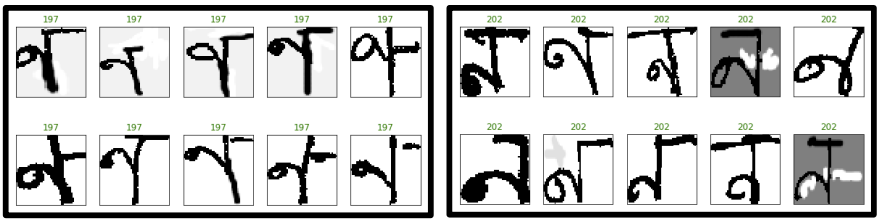


Figure 3

Algorithms and Techniques

In the last decade, **deep learning** has proved its outstanding performance in the field of machine learning and pattern recognition. Deep Neural networks (DNN)generally consists of multiple layers of neurons with one input layer and an output layer having several hidden layers in between. Due to the composition of many layer, DNNs are more capable for representing the highly varying nonlinear function compared to shallow learning approaches.

When it comes to vision, a special category of Deep Learning called Convoluted Neural Network (CNN) has been shown to outperform all other machine learning technique.

CNN is influenced by the way visual perception works in human brains. When we see something with our eyes we see features and the brain combines these features to define a class.

The CNN structure was first time proposed by Fukushima in 1980 (Fukushima, 1980). However, training it was not easy and hence it was not popular till Yann LeCun applied a gradient-based learning algorithm to CNN and obtained successful results in 1990. After that, researchers further improved CNN and reported excellent results and surpassed conventional best records on many benchmark databases, including MNIST handwritten digits database and CIFAR-10 (Krizhevsky & Hinton, 2009). Recently, deep CNN was applied for Hangul handwritten character recognition and achieved the best recognition accuracy (Kim & Xie, 2014).

Here are the reasons behind the suitability of CNN architecture for image classification - It retains the spatial data, it's a parametric model so has a short prediction time and does not need the data once its trained making it suitable for usage in mobile devices, sparsely connected layers means less parameters which helps avoid overfitting and also takes lesser time. With advanced regularization techniques like dropout, it can address the curse of dimensionality and enforces Occam's razor.

Hence I will be using the CNN architecture for the recognition of Bengali characters.

Below is a representative architecture diagram from Yann Lecun's paper (http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf)

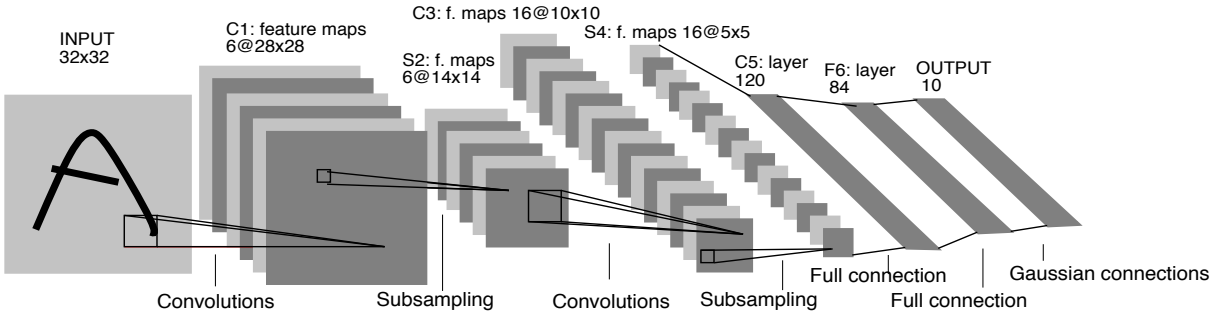


Figure 4

Benchmark

The highest accuracy for the given problem is recorded as 95.84% (see the table below) however it does not use a CNN architecture and employs a separate feature selection process. So it would be prudent to establish a benchmark using a simple CNN architecture. Subsequent improvements could be measured over the benchmark and the goal would be to improve on the highest recorded accuracy.

I have used a simple CNN model with 2 convolutional layers using relu activation and 1 fully connected layer terminating in 50 softmax classifiers. The F1 score thus recorded will be used as the benchmark against which subsequent performance would be measured and as stated above the goal would be to surpass the highest recorded accuracy.

Work Reference	Total Classes	Feature Selection	Classification	Accuracy
Bhowmik et al.	45	Wavelet Transformation	MLP	84.33%
Basu et al.	36	Longest run, Modified Shadow, Octant-centroid	MLP	80.58%
Bhattacharya et al	50	Regular and Irregular Gridbased selection	MQDF, MLP	95.84%
Benchmark – Simple CNN	50	None	CNN	32.57%

Table – Recognition Performance for some contemporary methods of handwritten Bangla character recognition.

III. Methodology

Data Pre-processing

We have pre-processed the images to improve the quality of the CNN algorithm:

(1) Validation Set - split the training data to have a validation set similar in size and distribution to the test set.
There are 2400 validation alphabet images with uniform distribution.

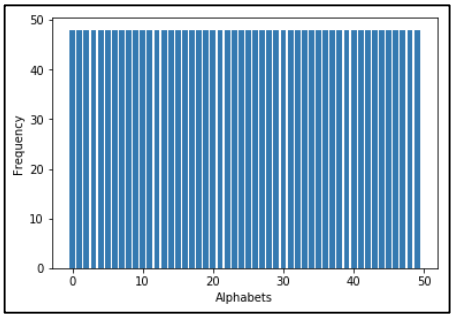


Figure 5

(2) Rescaling: We rescale the images by dividing every pixel in every image by 255. So the scale is now 0-1 instead of 0-255.

(3) We have randomly cropped the input images into 64x64 greyscale images.

(4) One-hot encoding for the categories to compare in the classification stage. For instance, we convert label=3 to a vector as [0 0 0 1 0 0 0 0 0 0 …. 0] (labels starts from 0 to 49 for the 50 classes)

Model Architecture

Handwritten character recognition is a complex task that requires processing of 2 dimensional images. These class of problems are well addressed by CNN architecture for reasons stated below –

- CNN ensures that spatial information is maintained and hence is ideal for image processing
- CNN automatically does feature extraction which is otherwise a separate process in other traditional models
- Since it's a sparsely connected parametric model it is well suited for mobile devices
- Guards against overfitting since it's sparsely connected
- Subsampling/Max pooling ensures certain degree of invariance to scaling, rotation and other distortions which are expected in handwritten characters

The pre-processed greyscale image data in 64x64 size is the input to the convolution layer.

Step1 - Convolution Layer:

A convolution is basically a combined integration of two functions and represents how one modifies the shape of the other. This is popular and extensively used in signal processing.

$$(f * g)(t) \stackrel{\text{def}}{=} \int_{-\infty}^{+\infty} f(\tau) g(t-\tau) d\tau$$

In the context of neural network, convolution is applied by the use of Feature Detectors (also known as Filters or Kernels) on the input image resulting in a feature map (also known as an Activation Map or a Convolved Map).

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

$\otimes$

0	0	1
1	0	0
0	1	1

$=$

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Figure 6 – Representative convolution operation using a 3x3 filter on a toy input of a smiley face with 7x7 dimensions and stride of 1. The resulting feature map is of dimension 5x5

We slide 2D matrices (here we shall call these Feature Extractors or Filters) over the image data which too is a 2D matrix and element-wise multiplication is performed as part of the convolution process. A representative simplified convolution operation is shown above. The resultant output is a Feature Map which is very often reduced in size. In the above case it is reduced from 7x7 to 5x5 using a Feature Extractor of size 3x3 and a stride of 1. We can alter the output Feature map by altering the size of the filter or the stride or padding.

So are we losing information? Yes indeed we are but the purpose is to detect the feature and eliminate the unnecessary information. We humans don't look at every pixel but identify the important features. That's what the convolution operation is preserving; the important features are thereby identified which helps in the classification process.

We usually create multiple feature maps by using many feature detectors; there are as many feature maps as feature detectors. And through training using back

propagated gradient descent the feature detectors are automatically set as part of the supervised learning algorithm. In each feature map we detect certain features and there relative positions in the image.

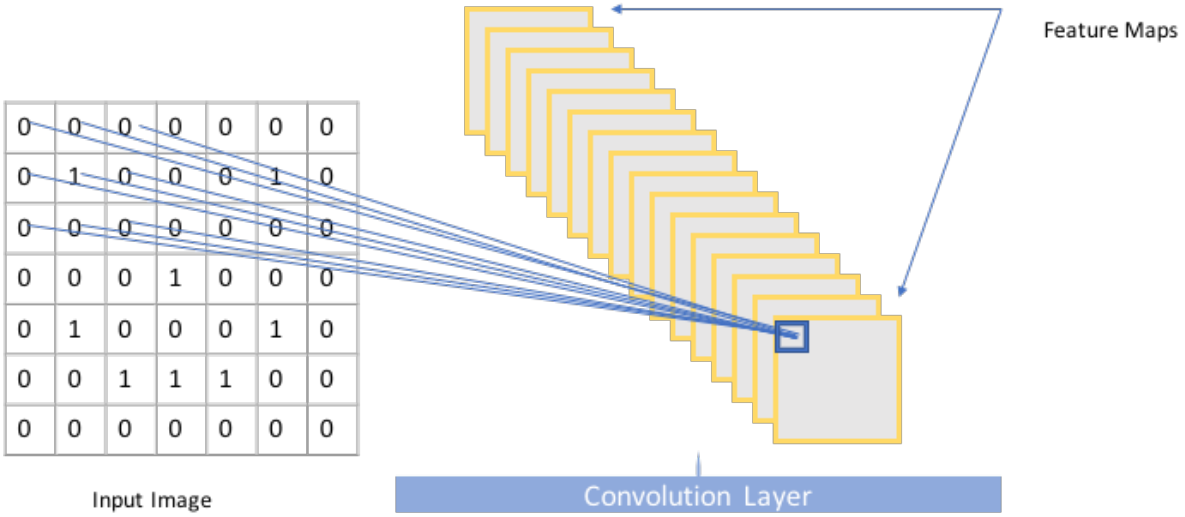


Figure 7 -: Illustrates feature detection in a convolution layer. Each feature map detects certain features and also preserves their relative position with respect to the input

Step1.1 - Convolution Layer – ReLU Activation

We need an activation layer like Relu Activation to add non-linearity to the output. When we apply mathematical operations like convolution we risk making outputs linear whereas the image features are distinguishable because they are non linear. There are borders, different colours, etc. The ReLU activation unit is the preferred method for introducing non-linearity. There are other rectifiers that does a similar function like sigmoid, hyperbolic tangent and leaky ReLU. Sigmoid is rarely used these days due to the vanishing gradient issue. I have used ReLU activation.

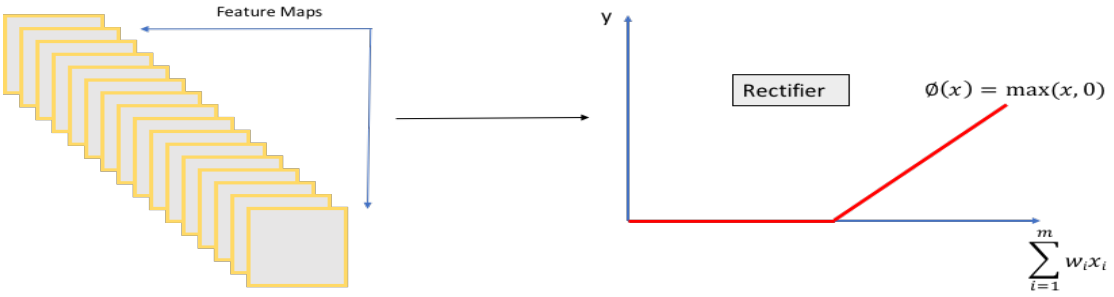


Figure 8 – The ReLU rectifier to enforce non-linearity for sharp feature boundaries

Pooling using max pooling

Pooling ensures that our neural network has spatial invariance that is to say, it accounts for some distortions because it captures the feature within a defined matrix as shown in the example below with max pooling. There are other types of pooling like Mean Pooling, Sum Pooling etc but I will be using max pooling in the project.

Figure 9 – Illustration of max pooling where the max number is picked in every 2x2 coloured matrix. Here a stride of 2 is used.

Apart from preserving the features it also gets rid of unimportant data thereby reducing number of parameters and therefore preventing overfitting. Since we are taking max we are accounting for distortions in the illustration above (Fig 9).

Flattening

At the end of the convolution layers we take the final pooled feature maps and flatten them into a column vector to be processed by a conventional Artificial Neural Network (ANN) with one or more densely connected hidden layers and a final output layer which is a softmax classifier.

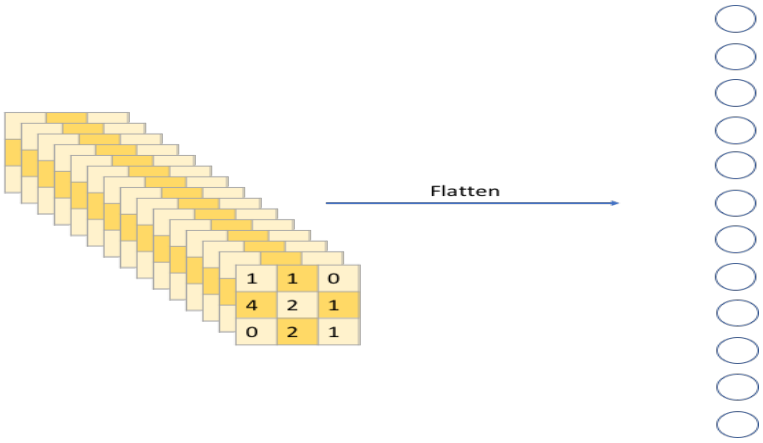


Figure 10 – The pooled featured maps are flattened at the end of the convolution operation into a column vector

Classification using FC layer

Finally we add a fully connected (dense) ANN on the convoluted layer output. The column of vector of outputs are added to the dense layer.

So the column vector is passed through a conventional densely connected ANN for the final classification. A very simplified network is shown below. More hidden layers could be added for higher accuracy and the final layer will pass through a softmax classifier for probability distribution across the different classes. I have used two dense hidden layers for classification in the final model.

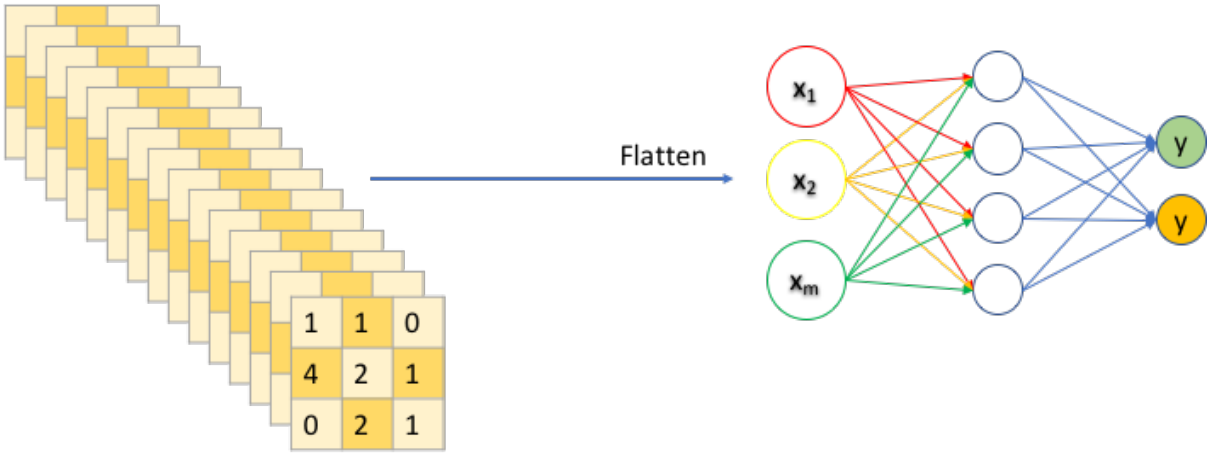


Figure 11 – Illustration of Dense layers after the convolution operations.

Training the network

The network is trained by forward propagating the input images in batches and using gradient based learning through back propagation algorithm to update the feature selectors (filters) with appropriate weights and all the hidden layers. I have selected rmsprop as the optimizer which is a preferred optimiser for image classification tasks. Finally since this is a multiclass classification problem have used categorical crossentropy as the loss function. The number of epochs and batch size have been parameterized for optimization during iteration.

Testing

Once the network is trained, we will run the test images through the network and record the F1 scores for each set of configurations. The network will be optimized by running multiple iterations with different parameters.

Transfer Learning

A very powerful idea in deep learning is that we can apply the knowledge learned by a neural network from one task onto another task in preferably a related domain. So, if we have a model trained on image classification then the same can be used for our problem domain as well. This is essentially what is known as Transfer Learning.
As a final step I have used Transfer Learning methods to compare the results. Please check the results section for more on this.

Refinement

In traditional machine learning projects, one strives to get a balance between Bias and Variance and its essentially a trade-off. However with deep learning, bias and variance are orthogonal and can be tuned separately. I have used the following workflow for the refinement:

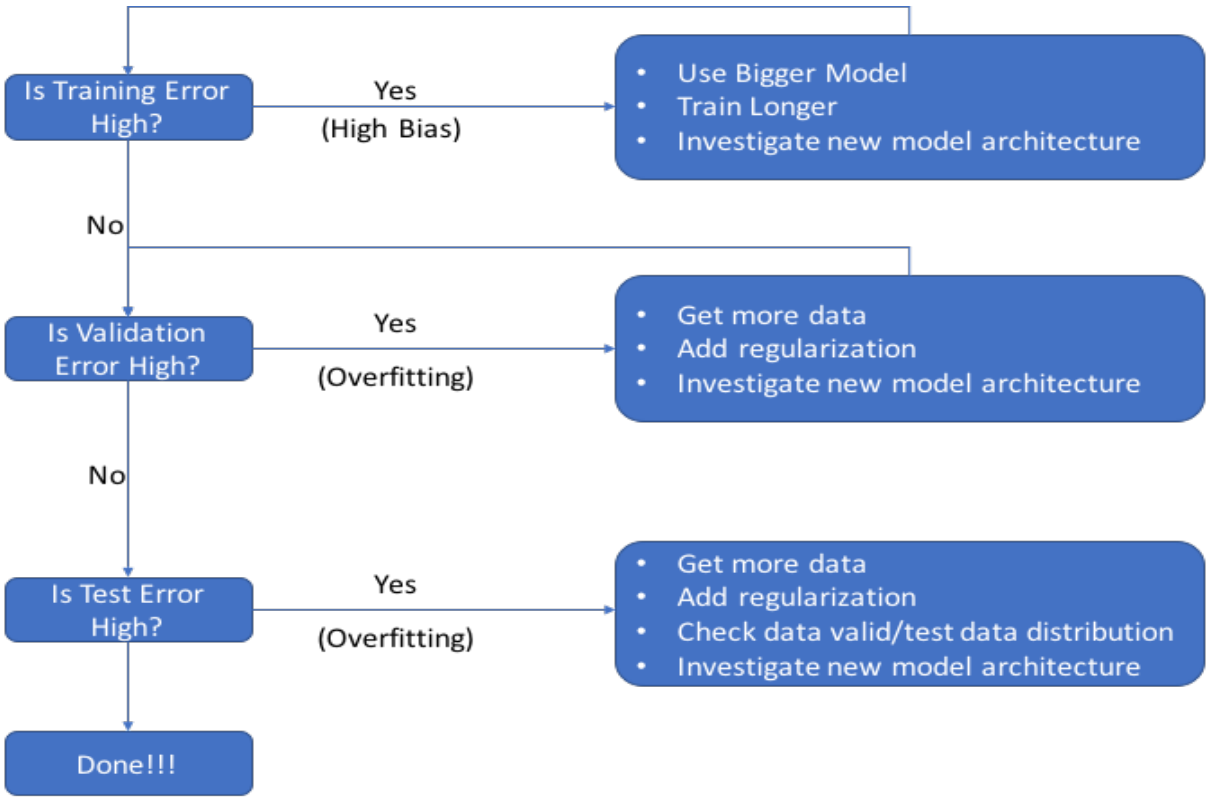


Figure 12 – Workflow used for refinement.

Tuning the bias:

If the training error is high (higher than the human level) then there are lot of avoidable errors and can be addressed by

1) Bigger Model – Adding more convolution layers to increase the depth of the neural net will ensure that more features are extracted which would improve the accuracy
2) Longer Training – Increasing the number of epochs would ensure that the model explore more weights to find the global minimum
3) New model architecture – Different optimization techniques and varying learning rates, activation functions like leaky ReLU, changing batch size to change the stochasticity, decreasing regularization to allow better fitment, etc

Tuning the variance:

If the validation error is much higher than the training error then that means the model has been overfitted for the training data. To address this we should essentially look at

1) Sourcing more data – Usually more data means more features and the accuracy of prediction improves since the model can generalize better
2) Regularization techniques – Dropout technique that randomly drops neurons in the layers and thereby distributes the weights better is an effective way to address overfitting and thereby ensures that the network is able to generalize better on the validation data
3) Transfer learning whereby the knowledge from another network that addresses a similar problem could be an option

If the validation error is low but the testing error is high then there is a possibility that the validation data and testing data are from different distributions. I have ensured that this is ruled out for my project by ensuring same distribution in the two datasets. Apart from the methods listed above Data Augmentation techniques could be used to increase the volume of data. In data augmentation new data is generated from the existing datasets by introducing distortions (rotation, shifting, blurring, zooming etc) in the same image.

Error Analysis:

Finally error analysis also provides insights into the data that may be valuable for tuning the network. There may be mislabelled data or damaged data which could be addressed on a case by case basis to tune the neural net.

IV. Results and Discussion

Table With Accuracy Report

	Benchmark Model	Deep CNN	Deep CNN with Dropout	Deep CNN with Augmentation	Final Model
Training Accuracy	33.06%	99.4%	92.82%	82.3%	85.11%
Validation Accuracy	30.88%	90.96%	93.21%	87.07%	88.71%
Test Accuracy	32.57%	90.63%	92.93%	95.57%	96.33%

Model Evaluation and Validation

Going by the methodology defined at the onset, I set up a quick and simple CNN network with the following configuration:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 63, 63, 16)        80
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 31, 31, 16)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 30, 30, 32)        2080
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
global_average_pooling2d_2 ( (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 50)                1650
=================================================================
Total params: 3,810
Trainable params: 3,810
Non-trainable params: 0
_________________________________________________________________
```

This would be the benchmark model. After training the model for 100 epochs, I got the accuracy of just over 33% on the training data which suggests a very high bias. So I decided to add more layers to the network and finally settled with 5 convolution layers and 2 dense layers. With this model, the training accuracy reached beyond 99% while the validation and test accuracies were just over 90% with same batch

size and number of epochs. This showed high variance so I decided to alter the model by adding some regularization with dropouts to address the overfitting issue.

The below snapshot is from the best recorded loss weights which shows that the training, validation and testing accuracy are around the 93% mark. This is definitely an improvement over the non-regularized model.

```
Epoch 35/100
9504/9600 [============================>.] - ETA: 0s - loss: 0.2530 - a
cc: 0.9285Epoch 00034: val_loss improved from 0.23873 to 0.23520, savin
g model to saved_models/weights.best.from_deepcnnwithDO.hdf5
9600/9600 [==============================] - 5s - loss: 0.2537 - acc: 0
.9282 - val_loss: 0.2352 - val_acc: 0.9321

Test F1 accuracy: 92.9357%
```

Next I decided to experiment by increasing the dense layers and reduced the regularization to improve the accuracy over the training data. However that did not help. I also tried with different optimisers. So as a next step I decided to train with augmented data and increased the train. Below is the snapshot from the experiment. The f1 accuracy on the test data shot up to 95.57%

```
Epoch 196/200
Epoch 00195: val_loss improved from 0.41847 to 0.40018, saving model to
saved_models/weights.best.with_augmentation_new.hdf5
3s - loss: 0.5419 - acc: 0.8236 - val_loss: 0.4002 - val_acc: 0.8707
Test accuracy: 95.5681%
```

Finally with another few hours of training the final results are as below. So we have achieved our goal and surpassed the highest accuracy achieved before.

```
Epoch 00298: val_loss improved from 0.34900 to 0.33933, saving model to
saved_models/weights.best.with_augmentation_new.hdf5
3s - loss: 0.4824 - acc: 0.8511 - val_loss: 0.3393 - val_acc: 0.8871
Test F1 accuracy: 96.3333%
```

I also experimented with other optimisers (adam and adadelta) as well as leaky ReLU activation however the results did not show any improvements beyond what is recorded above.

Error Analysis

As a first step towards error analysis in order to understand the root cause behind issues with the classification, I visualised the misclassified images. A quick run through the images shows that quite a few ণ and(chr#25 in folder 197) and ন (chr#30 in folder 202) have been misclassified. Also there are issues with other similar shaped characters.

Figure 13 – The 110 misclassified alphabets out of 3000 test images.

For further analysis of the errors a confusion matrix is given below with the major err ors highlighted in red.

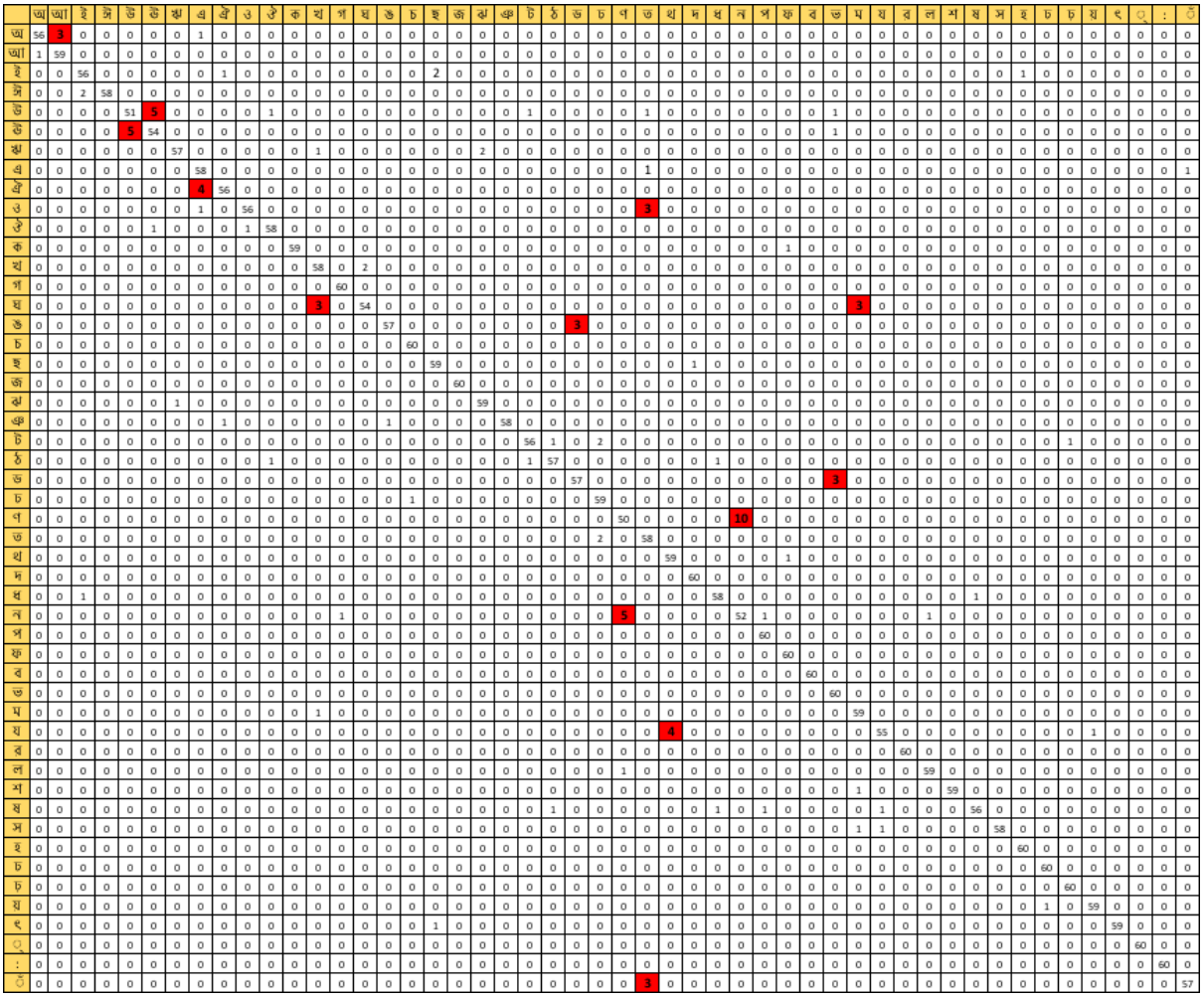


Figure 14 – Confusion matrix for all the classification errors

Here are the key observations:

- 15 errors are in classifying the alphabets ণ and ন. This was expected because the characters are similar and handwritten characters are very hard to distinguish. Analysis of the 15 errors shows that some of the characters are mislabelled in the database.
- 10 errors are in classification of characters ঊ and ঊ. Again as can be seen from the similarity of the characters it's a difficult classification problem for isolated handwritten characters even for humans.
- Avoidable errors are very few

Considering the above observations and the accuracy scores, the network performance is excellent by all means.

Transfer Learning

Transfer learning is a very popular methodology used in deep learning. So I wanted to check out some of the popular models for the given problem set. I used VGG16 and Inception models with the ***imagenet*** weights. I added two hidden dense layers on top. First I generated the bottleneck files with the available data and used these to train the model. I could get an accuracy of 80% with the VGG16 model and around 75% for the Inception model. Since my problem domain area is different from the imagenet classification problem and also I obtained a very high accuracy with the CNN model I built from scratch I did not spend any further effort in this. From literature I found that inception model primarily for more complex color images and is not suitable for the greyscale character recognition. I had to preprocess the images to higher dimensions (150x150) in order to use inception. I would later explore transfer learning with my model and apply the same to other Indian languages derived from Sanskrit.

V. Conclusion

Convolutional neural network (CNN) has ability to recognize visual patterns directly from pixel images and is the preferred method for image classification problems. Therefore, a CNN structure is investigated without any feature selection for Bangla handwritten pattern classification in this study. The method has been tested on a publicly available handwritten character dataset and outcome compared with existing prominent methods for Bangla. The proposed method is shown to have higher accuracy than any other model to the best of my knowledge.

Further Improvements and Future Research:

Removing noise from the dataset may further improve the performance. However I think that recognizing isolated handwritten Bengali characters may be harder compared to recognizing the same given the context of a word. Case in point is the errors with the two N sounding characters (ন and ণ). Given a word it would be easier to identify the characters than from isolated hand-written images. As a next step I would like to explore the word recognition problem in Bengali language.

OCR in itself may not be as useful however subsequent research on the ability to digitize handwritten Bengali text would be of immense academic as well as commercial value. Bengali being one of the most spoken and written languages in the world deserves further research on this. A kindle like rendering of bangla books and works would be much desired.