

# Sinhala Optical Character Recognition with Machine Learning

## Team Members

120638G	S.W. Sonnadara	ICE
120450X	D.R.J. Pathirana	ICE
120222C	M.L. Hettiarachchi	CSE
120629F	H.S. Siriwardhana	ICE

## Table of Contents

Team Members .....	1
1. Introduction.....	3
<b>Available ML Techniques for classifications applications</b> .....	3
<b>Deep Neural Networks</b> .....	4
<b>Static Classification Techniques</b> .....	5
<b>Other multi class classification techniques</b> .....	6
<b>Comparison of available ML approaches for OCR Application</b> .....	8
<b>Data Preprocessing</b> .....	9
2. ML approaches followed .....	11
<b>Convolutional Neural Networks</b> .....	11
<b>Advantages of using a CNN in character recognition applications</b> .....	12
<b>LeNet</b> .....	12
<b>Our CNN Implementation in Caffe deep learning framework</b> .....	13
3. Results and evaluation .....	17
4. Conclusion .....	26
5. References .....	27

## **1. Introduction**

Today there are many applications which supports optical character recognition. If you search for a mobile app there are many applications, including applications developed by google for optical character recognition. These applications provide optical character recognition for a wide range of languages for both hand written characters and machine-printed characters. But if we check for a mobile or a desktop application to recognize Sinhala font in internet, then you will find only the Optical Character Recognition (OCR) for printed Sinhala documents trained by Software Development Unit of University of Colombo School of Computing (UCSC). So our team decided to build a Sinhala OCR as our project.

As we have purposed in our project proposal, our plan was to develop an OCR to recognize Sinhala characters when images of the characters we given. We did not try to detect images of Sinhala documents because it needs to apply more complex steps such as identifying words segmentation of characters and recognizing characters. But for Sinhala language identifying segmentation of characters from a document is not easy. We have narrowed our project scope only to recognize images with a single printed Sinhala characters. Then we did some research and followed some guidelines of lectures to find the best approach to develop this application. All the steps we followed are described in the following sections separately.

### **Available ML Techniques for classifications applications**

There are several machine learning techniques that can be used in multiclass classification applications such as Optical Character recognition applications. There are even Tesseract like OCR engines available which will use Adaptive classification techniques. Also there are famous Open source projects like Tensorflow maintained by Google's Machine Intelligence research organization which can used as the machine learning engine which has deep learning methods implemented. Other than that, we can conventional ML methods like k-nearest neighbor classification, Support vector Machines and ensemble learning methods like Boosted Trees. Following section describe some of them in more detail.

## **Deep Neural Networks**

Deep Neural Networks (DNN) become popular as a machine learning mechanism to perform recognition. High accuracy DNN has achieved in both spotting text regions. Deep Neural Networks are essentially multi-layered learning and feature processing neural networks. Each neuron (node) in each layer is fed with information passed from nodes connected to it. A processing mechanism (transfer function) then determines how much of the processed information will be passed to the nodes connected to the present one. The architecture of the network, that is, the way neurons and layers are connected, plays a primary role in determining the network's ability to produce meaningful results.

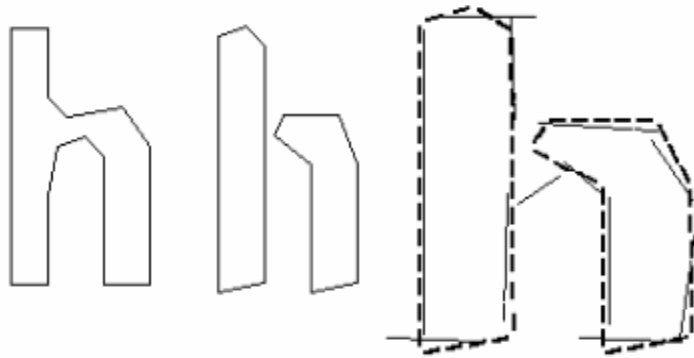
The advantage of DNNs is that architecture can be made heterogeneous. Similar to the human visual system, different neurons and processing layers are more sensitive to different features of objects. Edges of objects are seen more sharply by one set of neurons, while others are more sensitive to color gradients.

## **Deep Neural Networks in Tensorflow**

Tensorflow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research. Tensorflow Neural Network library also supports Deep Neural Networks which can be used in OCR applications.

Therefore, we can use the Tensorflow deep learning library to build the required CNN for our character recognition application as well. Because, it has all the required layers such as convolutional, pooling, normalization, localization and softmax layers.

## Static Classification Techniques



*Figure 1:*

<http://static.googleusercontent.com/media/research.google.com/en/pubs/archive/33418.pdf>

There are OCR readers which used topological features. They are nicely independent of font and size, but not robust to real time images used. Apart from that an intermediate idea is use of segments of the polygonal approximation as feature, but also images may damage with this approach. The breakthrough solution is features in unknown should not be same as features of training data.

During the training of data segments of a polygonal approximation is used. But when recognition letters small, fixed length units are extracted from the outline and matched many-to-one against the clustered prototype features of the training data.

In the first of the above image, letter is in a single piece. second is disconnected and third is short thin lines are the extracted features from the unknown data. Thin, longer lines are the clustered segments of the polygonal approximation and they are used as prototypes. This example shows that this process of small features matching large prototypes is easily able to cope with recognition of damaged image.

## **Tesseract OCR Engine**

Tesseract is an open-source OCR engine that was developed at HP between 1984 and 1994. Early versions of Tesseract did not include layout analysis and so inputting multi-columned text, images, or equations produced a garbled output. Later new image formats were added using the Leptonica library. Tesseract can detect whether text is monospaced or proportional. Now lots of languages can be detected by tesseract. Also Tesseract is suitable for use as a backend. Therefore, Tesseract is not just a ML library. It is an Engine that can be used to Optical Character Recognition. But our ambition was only to build a ML approach to classify machine-printed characters.

Tesseract OCR engine uses the static classification techniques inside the process. Since the static classifier has to perform well in generalizing to any kind of font, its ability to discriminate between different characters or between characters and non-characters is weakened. A more font-sensitive adaptive classifier that is trained by the output of the static classifier is therefore commonly used to obtain greater discrimination within each document, where the number of fonts is limited. Tesseract does not employ a template classifier, but uses the same features and classifier as the static classifier. The only significant difference between the static classifier and the adaptive classifier, apart from the training data, is that the adaptive classifier uses isotropic baseline/x-height normalization, whereas the static classifier normalizes characters by the centroid (first moments) for position and second moments for anisotropic size normalization.

## **Other multi class classification techniques**

### **Support Vector Machines (SVM)**

An SVM is a large-margin classifier, which is a vector space based machine learning method where the goal is to find a decision boundary between two classes that is maximally far from any point in the training data. For the character recognition we can use a Linear SVM or a SVM with a polynomial kernel.

SVM uses a technique called the kernel trick to transform the data and then based on these transformations it finds an optimal boundary between the possible outputs. It is capable to do some extremely complex data transformations, then figures out how to separate the data based on the

labels or outputs defined. To implement a OCR which uses no-linear data we can use a non-linear SVM, or SVM using a non-linear kernel. Non-linear SVM means that the boundary that the algorithm calculates doesn't have to be a straight line. The kernel trick takes the input data and transforms it. The benefit is that it can capture much more complex relationships between the data points without having to perform difficult transformations on our own. The downside is that the training time is much longer as it's much more computationally intensive. But sometimes with SVM complex data transformations and resulting boundary plane are very difficult to interpret. So it is not widely uses when compared to other techniques.

### **k Nearest Neighbor (k-NN)**

K-NN algorithm is one of the simplest classification algorithm. It finds the k closest points from training data and classify data. KNN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems. Even with such simplicity, it can give highly competitive results. KNN algorithm can also be used for regression problems.

This approach finds a group of k objects in the training set that are closest to the test object, and bases the assignment of a label on the predominance of a particular class in this neighborhood. The key elements of this approach are a set of labeled objects, e.g., a set of stored records (data), a distance or similarity metric to compute distance between objects and the value of k, the number of nearest neighbors. In order to classify an unlabeled object/item the distance of this object to the labeled objects is computed, its k-nearest neighbors are identified and the class labels of these nearest neighbors are then used to determine the class label of the object.

#### **Pros of k-NN:**

- Simple to implement
- Flexible to feature / distance choices
- Naturally handles multi-class cases
- Can do well in practice with enough representative data

#### **Cons of k-NN:**

- Large search problem to find nearest neighbors
- Storage of data
- Must know we have a meaningful distance function

## Boosted Trees

Boosting algorithms are generating new improved algorithm using combining weak algorithms. In more generally, boosting algorithm is that one can take a series of learning machines (weak learners) each having a poor error rate (but better than 50%) and combine them to give an ensemble that has very good performance. (strong learner).

The first boosting implementation was in OCR using neural networks as the weak learners. The main variation between many boosting algorithms is their method of weighting training data points and hypotheses. There are boosting algorithms such as AdaBoost, LPBoost, TotalBoost, BrownBoost etc. So using such advanced algorithms, we can get better classification results.

## Comparison of available ML approaches for OCR Application

One of the major problem in Neural Network approach in Machine Learning application is its training time. It is significantly large compared to other approaches like SVM, K-NN and decision tree based algorithms.

Also when comparing prediction accuracy in a generic multi class classification problem, Neural networks tends to offer less prediction accuracy compared to other ML approaches discussed above.

But when we considering computer vision and voice recognition application, the **curse of dimensionality** is heavily affected to such applications. Because images have very-high dimensional inputs. In our application it was 4900 [70x70]. In such case the most of the ML approaches will struggle.

If we compare SVM and K-NN classifiers, K-NN directly classifies directly classify data points with a given distance metric. Also the standard SVM can only be used for binary classification (But in LibSVM toolbox has multiclass classification). When it comes to Computer Vision applications, SVM has performed little more accurately than the K-NN approaches.



Also when considering computer vision applications, it is not practical to connect neurons to all neurons in the previous volume because such a network architecture does not take the **spatial structure** of the data into account. But it very important feature that is need for character recognition applications. Convolutional Neural Networks exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume.

Furthermore, if we consider the well-known handwritten digit database: **Mnist**, the most accurate classification models for that database are based on convolutional neural networks. Therefore, it has already proven that using CNN approach will be much suitable for OCR application. The models built using K-NN and SVM had low prediction accuracy in general for this database's predictions.

Therefore, we can conclude that using a CNN for character recognition application will be more appropriate than using the other ML approaches listed above.

## **Data Preprocessing**

Since there is no publicly available dataset we can use for training, we had to generate a dataset on our own for our project. So first we collected many available Sinhala fonts as possible in all types, in order to generate Sinhala characters in various styles.

Next, we wrote a script to generate 32 Sinhala consonants character images using all the fonts we collected. As old non-Unicode Sinhala fonts didn't have specific agreed mapping for ASCII to Sinhala, we had to change our script to fit to different font types. We generated total number of 2747, 70x70 px images of 32 Sinhala consonants, using more than 50 font styles. A sample of generated images are shown below.

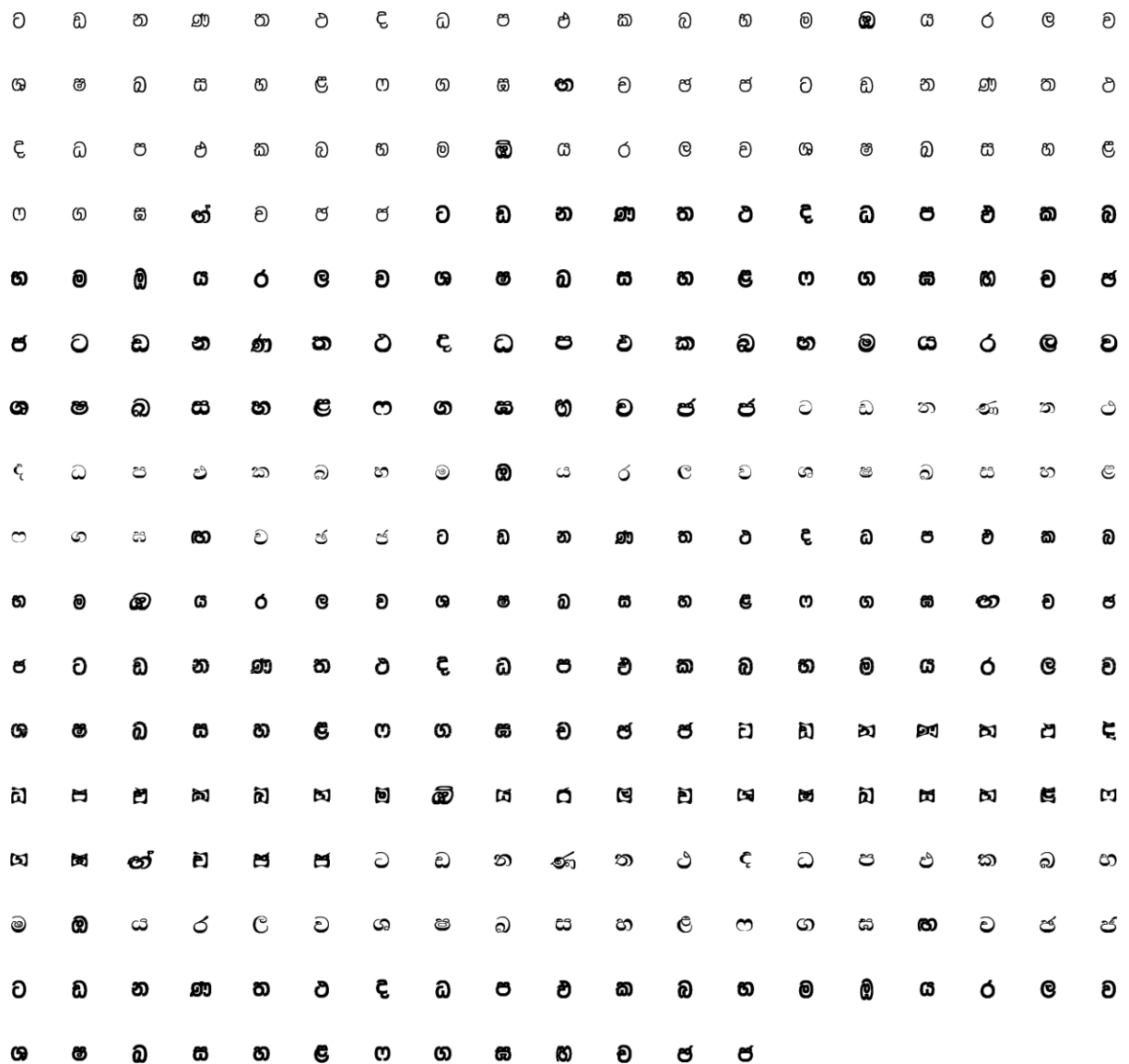


Figure 2: sample of generated Sinhala characters

Then we had to make two datasets for training phase and testing phase. So we split 2430 images as training data and rest of images as test data set. As we decide to use Caffe Deep learning framework for our project (As described in next topics), as the next step we had to create two databases in a Caffe friendly format to feed to the Data layers.

We used LMDB format in Caffe, as it gives good IO performance and suitable for fairly large datasets. To create LMDB database using Caffe tools, we also have to provide it a text file including file name of the file and its label. In our image generation script, we also generated this while generating images. Then we generated two LMDB databases for training and testing using Caffe's convert\_imageset tool.

## 2. ML approaches followed

### Convolutional Neural Networks

After explored some possible approaches for build an OCR reader we found that using an Artificial Neural Network Approach will be more suitable. Especially using **Convolutional Neural Networks (CNN)** can give better classification results in vision applications as the research suggest. Therefore, although the training time is significantly larger in NN approaches, we selected CNN because of its heavy usage and application specific advantages in vision applications in the globe.

CNN is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex, whose individual neurons are arranged in such a way that they respond to overlapping regions tiling the visual field.

CNN have been widely used applications of image and video recognition, recommender systems and natural language processing domains.

We used **LeNet CNN** for our OCR application. But there are more CNNs like **GoogLeNet**, which is composition of multi-scale dimension-reduced modules and **VGG**, which is 16 layers of 3x3 convolution interleaved with max pooling + 3 fully-connected layers are also available.

Mnist database is a well-known handwritten digit database in the globe. Most of the accurate classifiers for that database are based on the CNN was also one of the reason we selected CNN for our character recognition application.

## Advantages of using a CNN in character recognition applications

Our application is dealing high-dimensional inputs 4900 dimensions [70x70 images] and therefore, it not practical to connect neurons to all neurons in the previous volume. Also that approach does not take the **spatial structure of the data** into account. CNN exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers by each neuron is connected to only a small region of the input volume.

convolutional neural networks required relatively **little pre-processing** when comparing to other image classification algorithms. Therefore, the network is responsible for learning the filters that in traditional algorithms were hand-engineered.

The lack of dependence on prior knowledge and human effort in designing features is also an advantage for CNNs.

Another advantage of convolutional networks is the **use of shared weight** in convolutional layers, which means that the same filter (weights bank) is used for each pixel in the layer; this both **reduces memory footprint and improves performance**.

Also if we consider the well-known handwritten digit database: **Mnist**, the most accurate classification models for that database are based on convolutional neural networks. Therefore, it has already proven that using CNN approach will be much suitable for OCR applications.

## LeNet

LeNet is a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten and machine-printed digits. When we consider about Convolutional Neural Networks (CNNs), LeNet-5 is the latest convolutional network designed for handwritten and machine-printed character recognition. In our OCR application we used a LeNet implementation in Caffe Deep Learning Neural Network Library. Following section will demonstrate about what are the layers available in the LeNet.

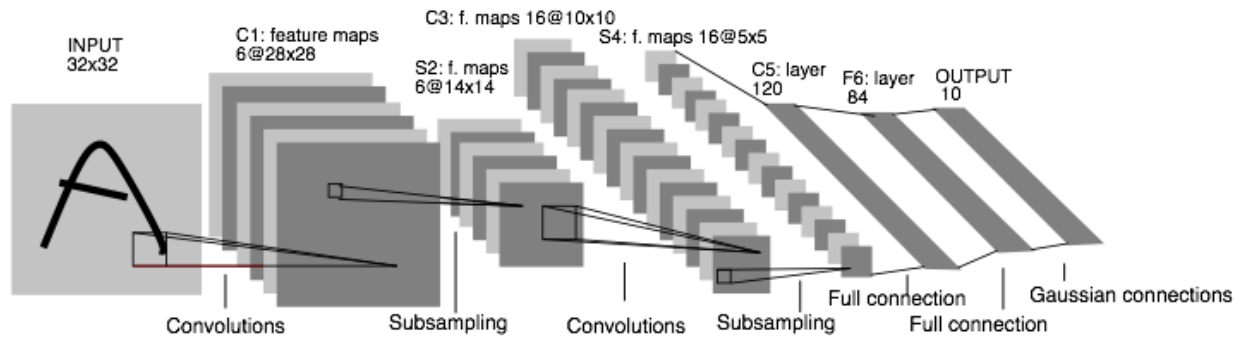


Figure 3: layers in LeNet 1989: <http://caffe.berkeleyvision.org/>

### Our CNN Implementation in Caffe deep learning framework

In our application, we selected to use Caffe deep learning framework. To be more specific a Convolution Neural Network of LeNet implementation. The LeNet we used were slightly different from the standard LeNet implementation. Here we have activation layer of Rectified Linear Unit (ReLU) rather than original sigmoid activation layer.

Our CNN consists of a convolutional layer followed by a pooling layer, another convolution layer followed by a pooling layer, and then two fully connected layers similar to the conventional multilayer perceptions.

In Caffe, the neural network will be defined using protobuf format definitions. Following diagram shows the CNN we created with important parameters for our OCR application.

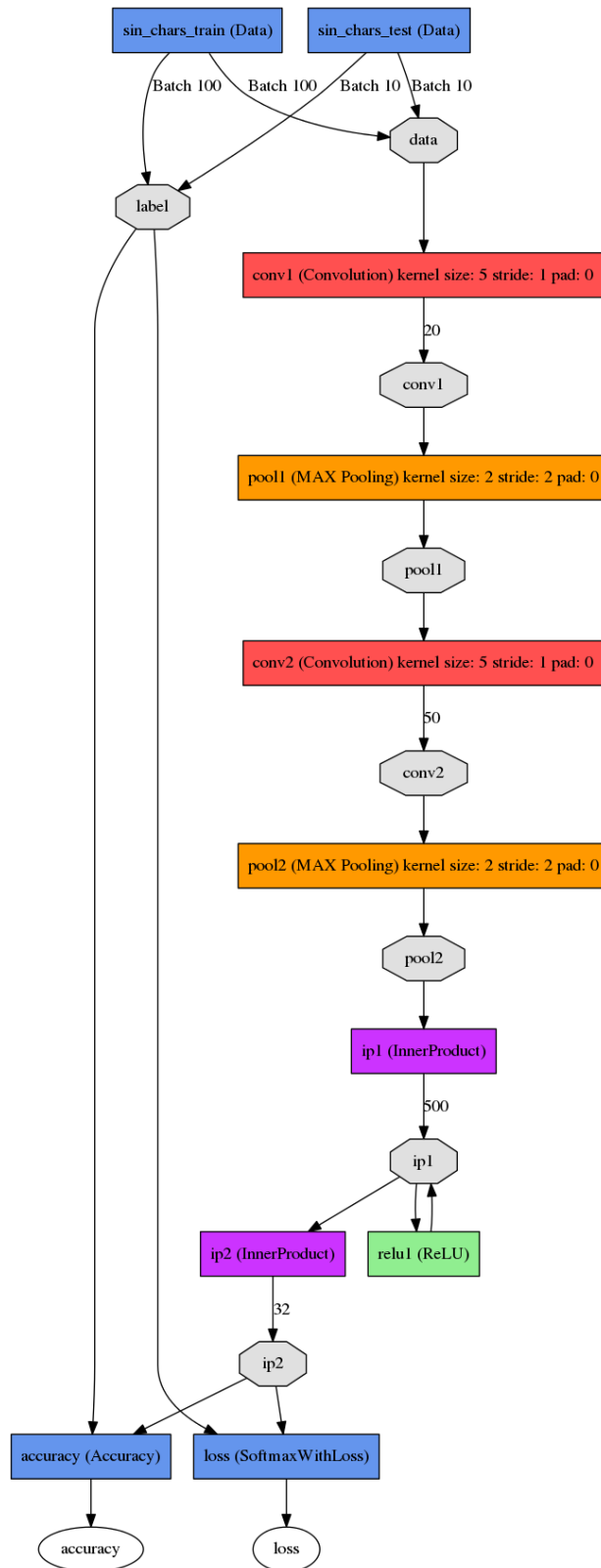


Figure 4: Implement LeNet CNN with Caffe

First layer in the network is the Data layer of the CNN. It consists of two datasets we created earlier for training and testing phases. This layer outputs two blobs, data blob and label blob.

Next there is a Convolutional layer and it is the core building block of a CNN. This enables see some specific type of feature at some spatial position in the input. Our first convolution consists of a layer with 5x5 kernel size with stride of 1 px. We have defined it to learn 20 filters of input data and each filter is initialized by Xavier algorithm. Then Output from the convolution is passed to Max Pooling layer, where polling will be done with 2x2 kernel with 2 px stride.

Again output of the Max Pooling layer is again fetched to another Convolutional layer with 50 filters. This layer also consists of 5x5 kernel with 1 px stride. Output of this layer again fetched to a Max pooling layer with same parameters as before.

Next layer on the network is a fully connected layer with 500 neurons, with ReLU activation function. There is another fully connected layer with 32 neurons, which is same as the number of classes in our dataset.

As in the diagram, after the above layer accuracy and loss of the current model will be calculated against test dataset.

After Defining the CNN, then we have to define the solver. It consists of solver parameters of the NN like test iterations, test interval, learning rate, regularization parameters like gamma, learning rate policy, maximum number of iteration and what is solver mode (running on CPU or GPU). Our solver file with parameters looks like in the following figure.

```

1. # The train/test net protocol buffer definition
2. net: "lenet_train_test.prototxt"
3. # test_iter specifies how many forward passes the test should carry out.
4. # we have test batch size 10 and 30 test iterations,
5. # covering the 300 testing images.
6. test_iter: 30
7. test_initialization: false
8. # carry out testing every 50 training iterations. (accuracy will be calculated)
9. test_interval: 50
10. # base learning rate, momentum and the weight decay of the network.
11. base_lr: 0.001
12. momentum: 0.9
13. weight_decay: 0.0005
14. # the learning rate policy
15. lr_policy: "inv"
16. gamma: 0.0001
17. power: 0.75
18.
19. # display every 10 iterations
20. display: 10
21. # the maximum number of iterations
22. max_iter: 600
23. # snapshot intermediate results
24. snapshot: 100
25. snapshot_prefix: "model/test2/lenet"
26. # solver mode: CPU or GPU
27. solver_mode: CPU

```

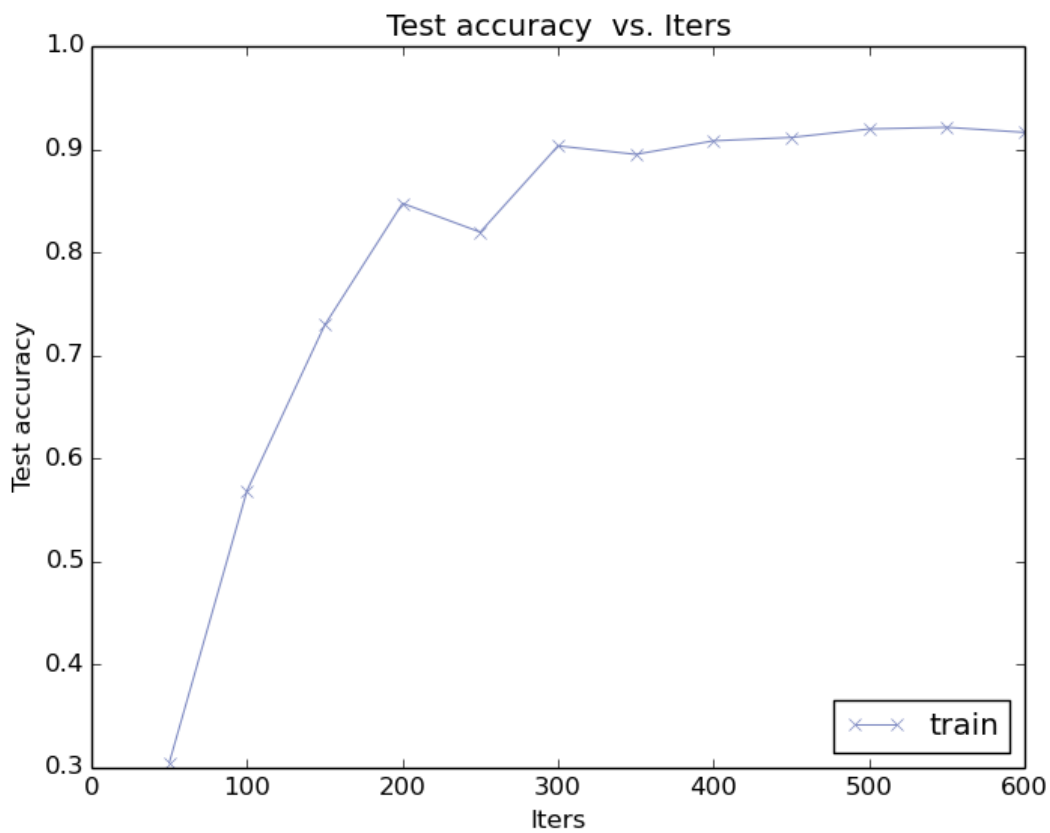
*Figure 5: our solver file (solver.prototxt)*



### 3. Results and evaluation

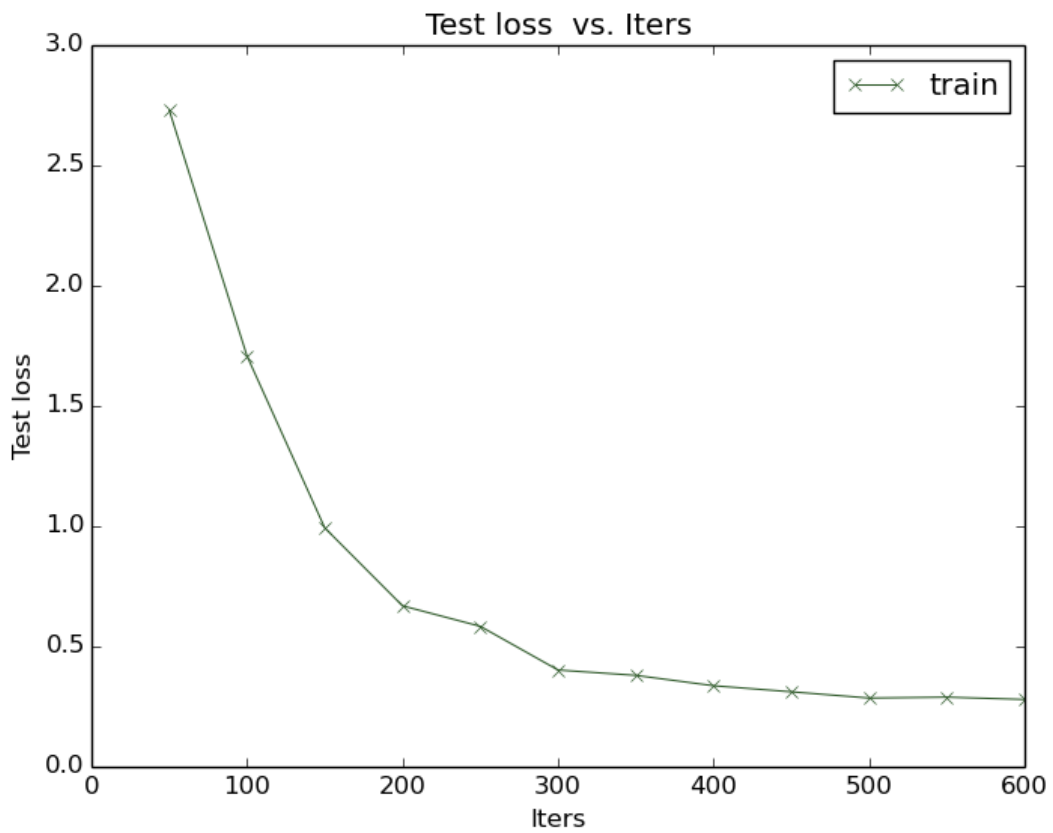
Using our CNN, we trained the dataset using 600 iterations resulting 24 epochs. We tested the current model with the test dataset in every 50 iterations.

We obtained a test accuracy of 0.916394 from our training model. Following is the graph representing the increase of test accuracy over iterations of the model.



*Figure 6: Accuracy of test images over iterations*

Following are the graphs representing the test loss and train loss of the model over iterations.



*Figure 7: Test loss over iterations.*

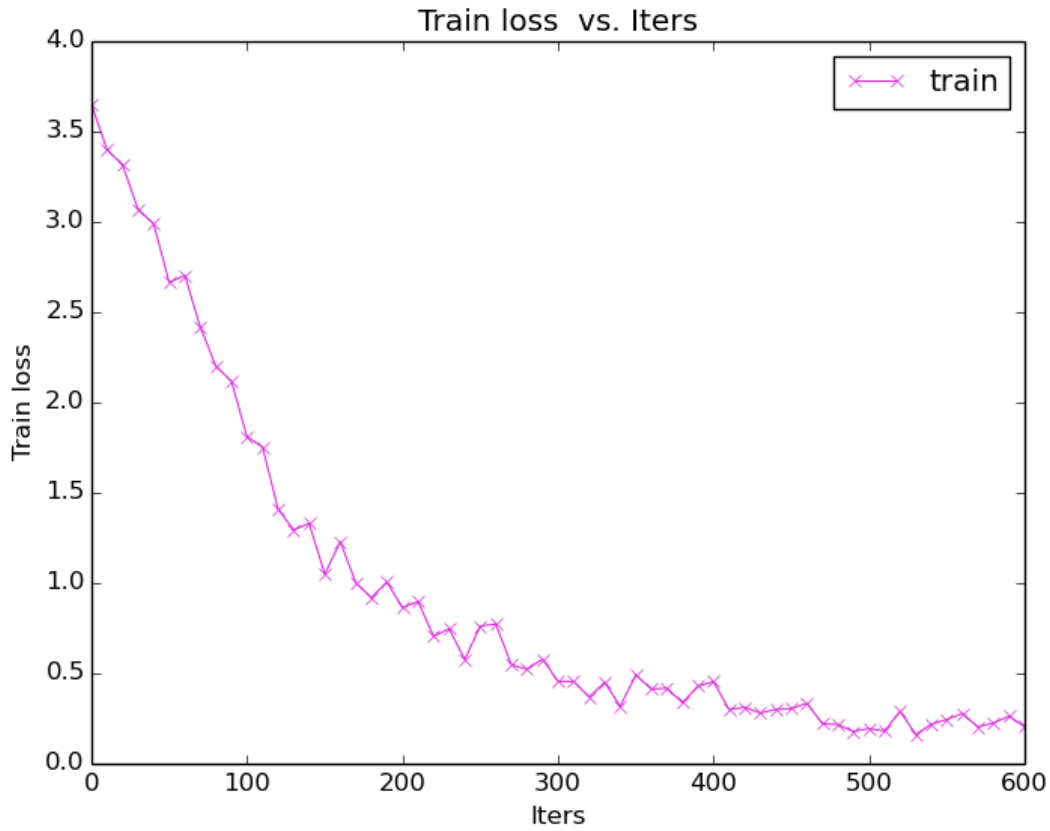



Figure 8: Train loss over iterations

After training of the model, we wrote a script using pyCaffe API to test another set of input images which were not used as train data or test data when training, to do cross validation.

Here are some outputs we obtained using those images and our trained model. Following table consists of 32 Sinhala consonants in six different typefaces and the model output for each character.

Image	Output probabilities
	<p>label is ක with 0.9711 prob</p> <p>label is ට with 0.0184 prob</p> <p>label is ඊ with 0.0031 prob</p>

	<p>label is ස with 0.0030 prob</p> <p>label is ඡ with 0.0017 prob</p>
බ	<p>label is ඩ with 0.4542 prob</p> <p>label is ට with 0.3166 prob</p> <p>label is ණ with 0.1940 prob</p> <p>label is ඞ with 0.0256 prob</p> <p>label is ජ with 0.0073 prob</p>
ඟ	<p>label is ග with 0.7427 prob</p> <p>label is ඞ with 0.2416 prob</p> <p>label is ඞ with 0.0073 prob</p> <p>label is ණ with 0.0041 prob</p> <p>label is න with 0.0015 prob</p>
ඹ	<p>label is ස with 0.9954 prob</p> <p>label is ස with 0.0018 prob</p> <p>label is ක with 0.0010 prob</p> <p>label is ඞ with 0.0008 prob</p> <p>label is ඡ with 0.0004 prob</p>
ඹ	<p>label is ජ with 0.5289 prob</p> <p>label is ො with 0.3022 prob</p> <p>label is ග with 0.0540 prob</p> <p>label is ඞ with 0.0483 prob</p> <p>label is ළ with 0.0254 prob</p>
ඵ	<p>label is ඩ with 0.8547 prob</p> <p>label is ට with 0.0676 prob</p> <p>label is ට with 0.0373 prob</p> <p>label is ඵ with 0.0160 prob</p> <p>label is ජ with 0.0090 prob</p>

ඡ	<p>label is ඡ with 0.5978 prob</p> <p>label is ඡ with 0.3639 prob</p> <p>label is ඡ with 0.0234 prob</p> <p>label is ඡ with 0.0062 prob</p> <p>label is ඡ with 0.0059 prob</p>
ඡ	<p>label is ඡ with 0.7294 prob</p> <p>label is ඡ with 0.2331 prob</p> <p>label is ඡ with 0.0359 prob</p> <p>label is ඡ with 0.0008 prob</p> <p>label is ඡ with 0.0004 prob</p>
ච	<p>label is ච with 0.3282 prob</p> <p>label is ච with 0.2377 prob</p> <p>label is ච with 0.1524 prob</p> <p>label is ච with 0.0969 prob</p> <p>label is ච with 0.0701 prob</p>
ච	<p>label is ච with 0.9995 prob</p> <p>label is ච with 0.0004 prob</p> <p>label is ච with 0.0001 prob</p> <p>label is ච with 0.0000 prob</p> <p>label is ච with 0.0000 prob</p>
ඡ	<p>label is ඡ with 0.5005 prob</p> <p>label is ඡ with 0.4611 prob</p> <p>label is ඡ with 0.0243 prob</p> <p>label is ඡ with 0.0110 prob</p> <p>label is ඡ with 0.0013 prob</p>


ඌ	<p>label is ඌ with 0.3991 prob</p> <p>label is ඹ with 0.2153 prob</p> <p>label is ණ with 0.1525 prob</p> <p>label is ඵ with 0.1282 prob</p> <p>label is ළ with 0.0486 prob</p>
ඹ	<p>label is ඹ with 0.6883 prob</p> <p>label is ක with 0.1676 prob</p> <p>label is න with 0.1400 prob</p> <p>label is ඞ with 0.0039 prob</p> <p>label is ජ with 0.0001 prob</p>
ඳ	<p>label is ඌ with 0.9137 prob</p> <p>label is ඳ with 0.0746 prob</p> <p>label is ඵ with 0.0098 prob</p> <p>label is ට with 0.0006 prob</p> <p>label is ළ with 0.0004 prob</p>
ඳ්	<p>label is ද with 0.9863 prob</p> <p>label is ළ with 0.0137 prob</p> <p>label is ඩ with 0.0000 prob</p> <p>label is ඵ with 0.0000 prob</p> <p>label is ඹ with 0.0000 prob</p>
ඞ	<p>label is ඩ with 0.9989 prob</p> <p>label is ඩ with 0.0007 prob</p> <p>label is ඩ with 0.0002 prob</p> <p>label is ළ with 0.0001 prob</p> <p>label is ඩ with 0.0001 prob</p>

උ	<p>label is උ with 0.9970 prob</p> <p>label is ඔ with 0.0014 prob</p> <p>label is ඔ with 0.0009 prob</p> <p>label is ස with 0.0006 prob</p> <p>label is ස with 0.0001 prob</p>
ඌ	<p>label is ඌ with 0.9996 prob</p> <p>label is ඌ with 0.0001 prob</p> <p>label is ඩ with 0.0001 prob</p> <p>label is ඩ with 0.0001 prob</p> <p>label is න with 0.0000 prob</p>
ඍ	<p>label is ඍ with 0.6530 prob</p> <p>label is ඍ with 0.3436 prob</p> <p>label is ඔ with 0.0014 prob</p> <p>label is ඩ with 0.0013 prob</p> <p>label is ඔ with 0.0002 prob</p>
ඎ	<p>label is ඎ with 0.4783 prob</p> <p>label is ඎ with 0.4597 prob</p> <p>label is ඎ with 0.0483 prob</p> <p>label is ඔ with 0.0073 prob</p> <p>label is න with 0.0044 prob</p>
ඏ	<p>label is ඏ with 0.9999 prob</p> <p>label is ඔ with 0.0001 prob</p> <p>label is ඩ with 0.0000 prob</p> <p>label is ඍ with 0.0000 prob</p> <p>label is ඔ with 0.0000 prob</p>

ඔ	<p>label is ඔ with 0.6886 prob</p> <p>label is ඔ with 0.1814 prob</p> <p>label is ඔ with 0.0945 prob</p> <p>label is ස with 0.0287 prob</p> <p>label is ඔ with 0.0035 prob</p>
ඔ	<p>label is ඔ with 0.9991 prob</p> <p>label is ස with 0.0008 prob</p> <p>label is ස with 0.0001 prob</p> <p>label is ඔ with 0.0000 prob</p> <p>label is ඔ with 0.0000 prob</p>
ඔ	<p>label is ඔ with 0.9934 prob</p> <p>label is ඔ with 0.0034 prob</p> <p>label is ඔ with 0.0020 prob</p> <p>label is ඔ with 0.0006 prob</p> <p>label is ඔ with 0.0003 prob</p>
ඔ	<p>label is ඔ with 0.7700 prob</p> <p>label is ඔ with 0.2180 prob</p> <p>label is ඔ with 0.0037 prob</p> <p>label is ඔ with 0.0033 prob</p> <p>label is ඔ with 0.0021 prob</p>
ඔ	<p>label is ඔ with 0.9961 prob</p> <p>label is ඔ with 0.0018 prob</p> <p>label is ඔ with 0.0014 prob</p> <p>label is ඔ with 0.0003 prob</p> <p>label is ඔ with 0.0002 prob</p>



ඌ	<p>label is ඌ with 0.3765 prob</p> <p>label is ඍ with 0.3288 prob</p> <p>label is උ with 0.2622 prob</p> <p>label is ෙ with 0.0142 prob</p> <p>label is ෑ with 0.0100 prob</p>
ඍ	<p>label is ඍ with 0.9478 prob</p> <p>label is ස with 0.0474 prob</p> <p>label is ම with 0.0032 prob</p> <p>label is ඞ with 0.0007 prob</p> <p>label is ඣ with 0.0004 prob</p>
ඎ	<p>label is ස with 0.9979 prob</p> <p>label is ස with 0.0015 prob</p> <p>label is ඞ with 0.0004 prob</p> <p>label is ෙ with 0.0001 prob</p> <p>label is ඞ with 0.0000 prob</p>
ඏ	<p>label is ඞ with 0.8732 prob</p> <p>label is ඞ with 0.1087 prob</p> <p>label is ඞ with 0.0162 prob</p> <p>label is ඌ with 0.0007 prob</p> <p>label is ඡ with 0.0004 prob</p>
ඐ	<p>label is ඥ with 0.9981 prob</p> <p>label is ඥ with 0.0011 prob</p> <p>label is ෙ with 0.0007 prob</p> <p>label is ඌ with 0.0000 prob</p> <p>label is ඵ with 0.0000 prob</p>

	<p>label is ම with 0.9990 prob</p> <p>label is ම with 0.0009 prob</p> <p>label is ම with 0.0000 prob</p> <p>label is ම with 0.0000 prob</p> <p>label is ම with 0.0000 prob</p>
---	--

We also publish our project on GitHub with the generated dataset and the scripts we used to evaluate the trained model. [<https://github.com/wathmal/sinhala-ocr>]

#### 4. Conclusion

As we have discussed above for OCR applications using Convolutional Neural Network can have categorized as a suitable ML approach when compared to other multi class classification approaches like SVM, K-NN and Tree based methods. But we can have lots of optimizations for this approach. Because the LeNet is one of the basic NN available and there are more complex networks GoogLeNet currently available. Therefore, if we can implement such network, we may be able to get much more accurate results.

Also there are complex ensemble learning methods like, boosting trees available, we can use such method to combine NN and a decision tree algorithm which may lead to better results.

Also in our dataset we only used generated perfect images without any kind of noise. In Order to have more accurate trained model which can identify natural machine-printed characters, we should have introduced some Gaussian and pepper noise to our generated image set. Also we can use publicly available data sets also to improve the training data set.

In this approach we only used 32 Sinhala consonants. But for an improved model, we have to use all the Sinhala consonants and vowels with vowel modifiers to have a complete Sinhala character recognition model.

As a further improvement we can add computer vision layer to tokenize the sentences to letters, and then we can have complete OCR application which will be able to convert complete image and identify it as words and sentences. Then we can further improve the predicted words by using another ML layer to predict incomplete words. Which means we can improve this application to the production level as well.

## 5. References

- <http://www.cvisiontech.com/resources/ocr-primer/ocr-neural-networks-and-other-machine-learning-techniques.html>
- <http://www.rsipvision.com/deep-learning-for-ocr/>
- [http://www12.tuiasi.ro/users/103/F2\\_2012\\_2\\_Tautu.pdf](http://www12.tuiasi.ro/users/103/F2_2012_2_Tautu.pdf)
- <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.6729&rep=rep1&type=pdf>
- <http://professordrucker.com/Publications/BoostingDecisionTrees.pdf>
- <http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/33418.pdf>
- [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)
- <http://yann.lecun.com/exdb/mnist/>
- [https://www.tensorflow.org/versions/r0.10/tutorials/deep\\_cnn/index.html](https://www.tensorflow.org/versions/r0.10/tutorials/deep_cnn/index.html)
- <http://caffe.berkeleyvision.org/gathered/examples/mnist.html>
- <http://yann.lecun.com/exdb/lenet/>
- <https://developers.google.com/protocol-buffers/docs/overview>
- <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.21.6729&rep=rep1&type=pdf>