3rd Project: Final Report

CFIAR-10 Image Classification

Dharamendra Kumar
Pranjay Patil
Shubhi Jain
Yash Shrivastava

Masters in Computer Science

Instructor: Dr. Shannon Quinn

Department of Computer Science

University of Georgia, Athens

Project Specifications:

Language

- Java 1.8
- Python 2.7

Techniques

- Convolution Neural Networks (CNN)
- Keras with TensorFlow
- MxNet (Multiple GPU)
- TensorFlow (Multiple GPU)

Experimentation

- Deeplearning4j with Spark

Introduction
Image processing has been a widely researched topic for the last decade. Similarly, image classification has picked up the pace and is being talked about more frequently. In this project we got the opportunity to perform image classification on a large dataset in a distributed fashion.

Problem Statement
We are given the dataset: **CIFAR-10**, which consists of 60,000 images each of 32×32 dimentions, RGB channel. All of the images fall into one of the 10 labels. The aim is to classify the images into one of the 10 labels provided by the dataset.

CIFAR-10 State Of The Art
Currently, the best accuracy on CIFAR-10 dataset is 96.53%. This, however, is very difficult to beat.

Approach
When it comes to image processing, our group decided that choosing to use Neural Networks would result in an easier task. After a lot of web research, we narrowed our focus to Deep Learning. Within that field, **Convolutional Neural Networks (CNN)** came of great use to us.

Background
Initially, people commonly used Neural Networks when it came to images and their processing. They were faced with the tough task of training them since the image dimensions were so large and the training process itself is so time consuming. This process was later switched to Deep Learning as it not only included more than one hidden layer, it also seemed to do a better job overall.

Convolutional Neural Networks is a type of Deep Learning technique. Its input consists of images which constrains the architecture in a more sensible way, in this way it has an advantage. CNN is easily set apart from regular Neutal Networks.

CNN is made up of many layers stacked together. These layers have neurons connected in three dimensions: width, height, and depth. In this way it is able to retain the shape of the image itself.

Layers in CNN:

1) Convolutional Layer

The INPUT [32x32x3] holds the raw pixel values of the image. In this case it is an image of width 32, height 32, and with three colour channels R, G, B. The convolution layer will compute the output of neurons that are connected to local regions in the input. Each computs a dot product between their weights and a small region from which they are connected to in the input volume.

2) RELU Layer

The RELU layer applies an elementwise activation function. This leaves the size of the volume unchanged.

3) Pooling Layer

The POOL layer performs a down sampling operation along the spatial dimensions (width, height).

4) Full Connected Layer

The FC layer computes the class scores, resulting in a volume of the size [1x1x10]. Each of the 10 numbers correspond to a class score among the 10 categories of CIFAR-10. As with ordinary Neural Networks, each neuron in this layer will be connected to all the numbers in the previous volume.

Through the stacking of these layers, CNN transforms the original image layer by layer from the original pixel values to the final class scores. The arrangement of these layers against each other is one of the Hyper-parameters which has to be adjusted by the programmer.

Why CNN?

We had many other options which we could have used to tackle the image classification problem. For example, SVM, Random Forest classifier, etc. We chose CNN to be the classifier for the given problem. CNN seemed to be the best technique for image classification for many reasons. The first and most important reason for our group to choose CNN was because it **retains the initial shape of the image** while it processes it. Secondly, one has to worry less about the feature engineering part since CNN does that by itself. Instead of trying different classifiers, we stuck with CNN and tried working on many libraries available for Deep Learning like TensorFlow and MxNet to improve the accuracy.

Implementation

After selecting the classifier, the next important step is implementation. We discovered many different Deep Learning packages available to us. As a result, we were not confined to only one technique. As we implemented many packages for Deep Learning, we were able to compare them w.r.t many criterions and improve the accuracy. The following are the various implementations we used, which will be explained separately below:

a) Keras with TensorFlow
b) TensorFlow (Multi GPU)
c) MxNet
d) Deeplearning4j (experimentation)

Keras with TensorFlow

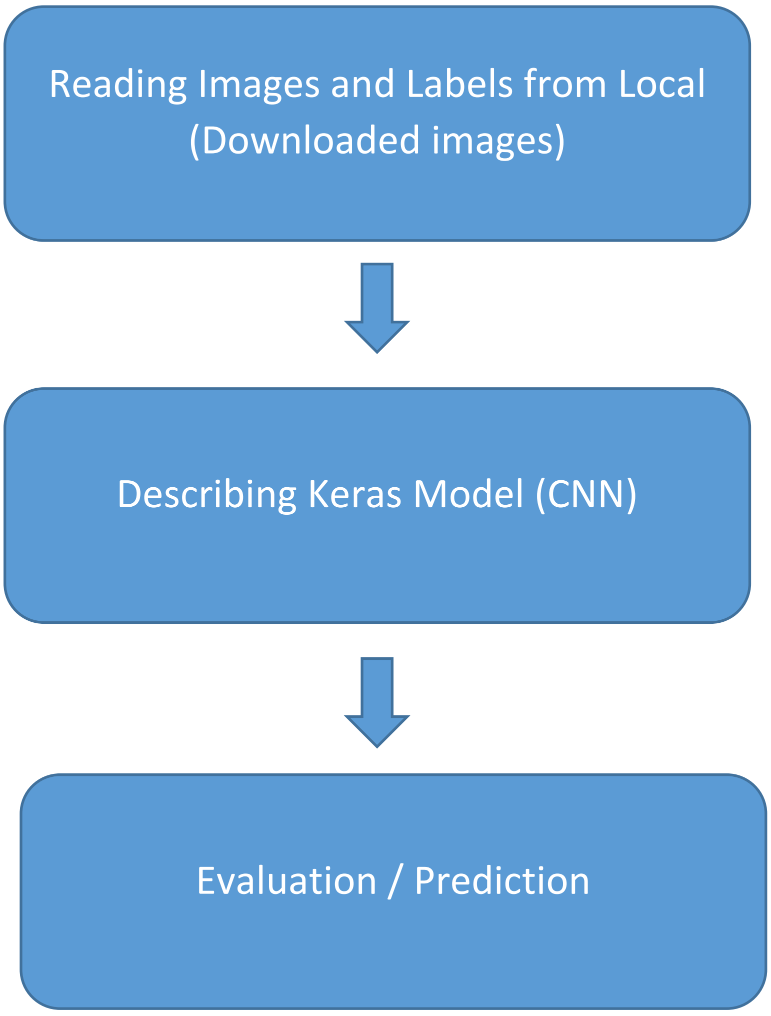
TensorFlow is an open source library developed by Google for numerical computation using data flow graphs. It is a powerful library that allows users to work with large tensors (data arrays) and thus, provides an excellent platform for CNN implementation.

Keras on the other hand, is a model-level that provide high-level building blocks for Deep Learning models. Keras does not take care of the low-level tensor manipulations. It relies on the tensor-manipulation libraries that act as

Keras's backend. TensorFlow and Theano are well-known backend implementations for Keras.

The reason that one of our implementations is in Keras is because it efficiently leverages TensorFlow and it makes the implementation smooth.

The Flow


Reading Images and Labels from Local (Downloaded images)
Describing Keras Model (CNN)
Evaluation / Prediction

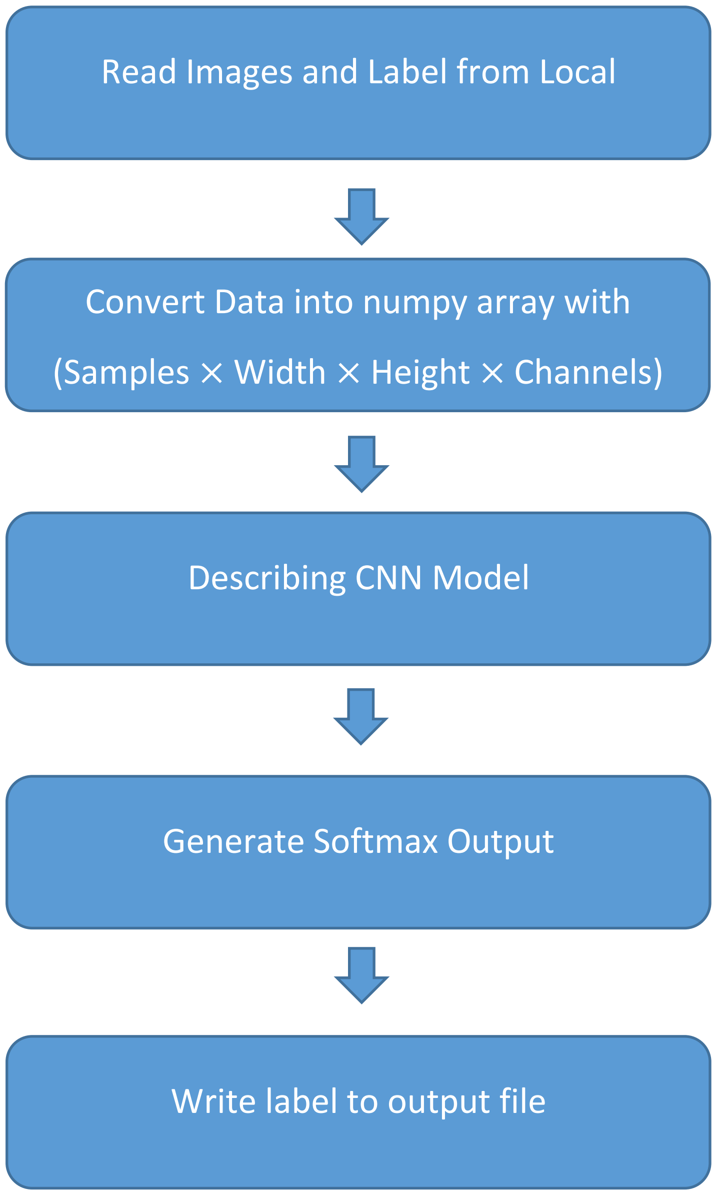
Keras CNN Model: 1000 Epoch

Convolution $(7 \times 7 \times 32)$, Relu
Convolution $(3 \times 3 \times 32)$, Relu
Max Pooling (3×3)
Convolution $(3 \times 3 \times 64)$, Relu
Max Pooling (2×2)
Convolution $(3 \times 3 \times 64)$, Relu
Max Pooling (2×2)
Convolution $(2 \times 2 \times 32)$, Relu
Max Pooling (2×2)
Fully Connected Layer 1024
Fully Connected Layer 512
Fully Connected Layer 10

MxNet Implementation

MxNet is another efficient library for Deep Learning that works with multiple GPU or multiple computers. It can interface with various languages: we implemented MxNet with Python 2.7.

The Flow:


Read Images and Label from Local
Convert Data into numpy array with
(Samples × Width × Height × Channels)
Describing CNN Model
Generate Softmax Output
Write label to output file

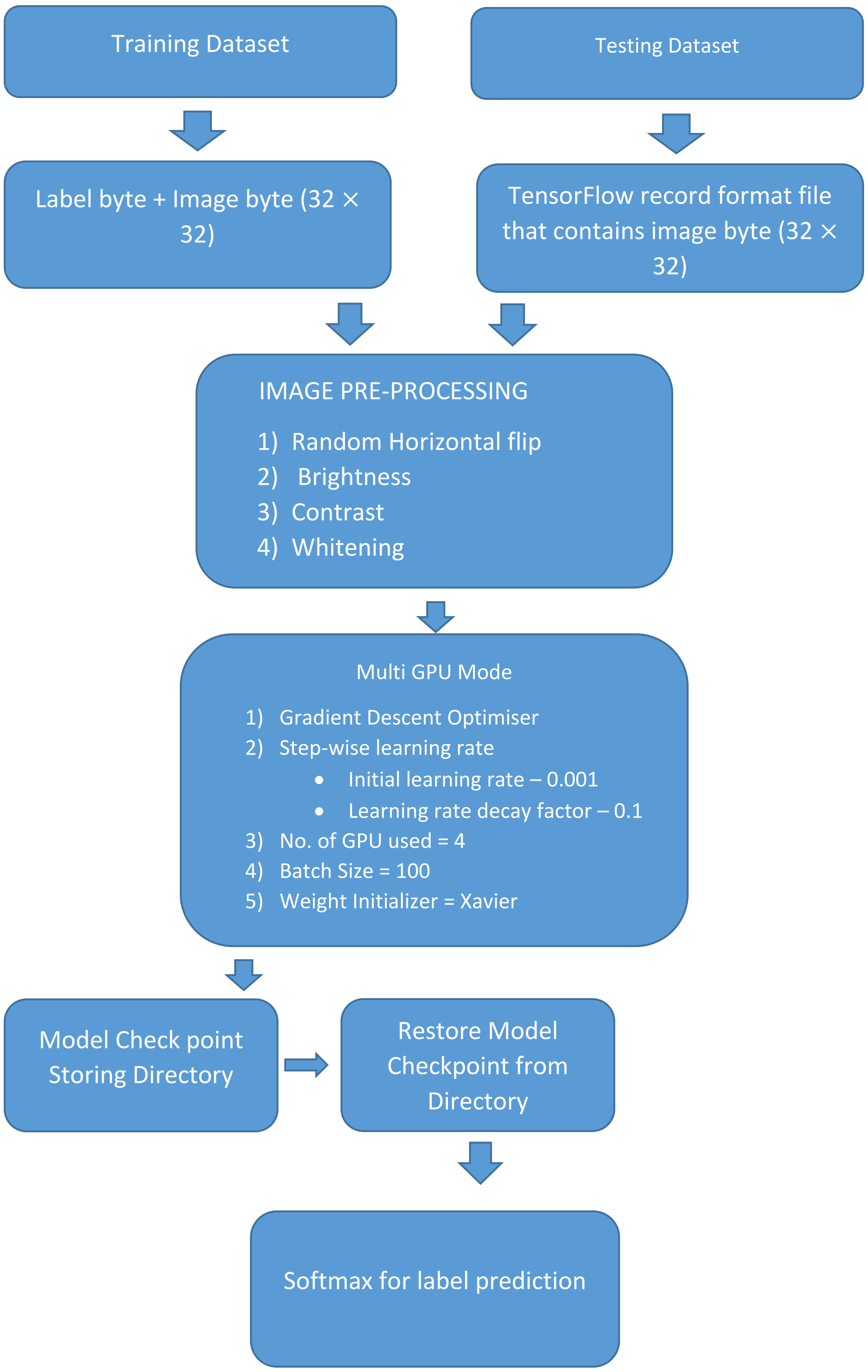
CNN Model: 1000 Epoch

Convolution $(3 \times 3 \times 32)$, Relu
Convolution $(3 \times 3 \times 32)$, Relu
Pooling Max (2×2)
Convolution $(3 \times 3 \times 64)$, Relu
Convolution $(3 \times 3 \times 64)$, Relu
Pooling Max (2×2)
Convolution $(3 \times 3 \times 64)$, Relu
Pooling Max (2×2)
Dense 512
Dense 64
Dense 10
Softmax

TensorFlow (Multi GPU)

TensorFlow library was used for distributed implementation of Convolution Neural Networks. Training a model in a parallel and distributed fashion requires coordinating training processes. For what follows we use the term model replica to mean one copy of model training on a subset of data.

The Flow: 50,000 Epoch


Training Dataset
Testing Dataset
Label byte + Image byte (32 × 32)
TensorFlow record format file that contains image byte (32 × 32)
IMAGE PRE-PROCESSING
1) Random Horizontal flip
2) Brightness
3) Contrast
4) Whitening
Multi GPU Mode
1) Gradient Descent Optimiser
2) Step-wise learning rate
• Initial learning rate – 0.001
• Learning rate decay factor – 0.1
3) No. of GPU used = 4
4) Batch Size = 100
5) Weight Initializer = Xavier
Model Check point Storing Directory
Restore Model Checkpoint from Directory
Softmax for label prediction

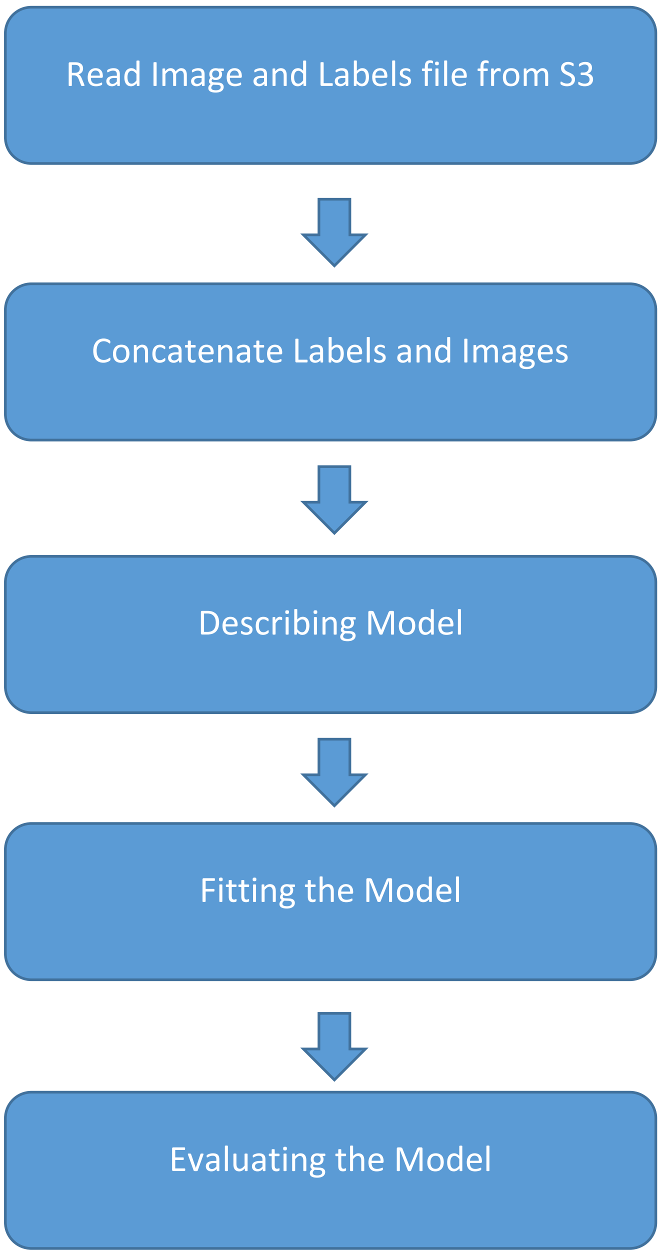
Experimentation

Deeplearning4j Implementation

It is an open-source library that works in a distributed fashion which is written for Java, and Scala. Deeplearning4j is also integrated with Spark and Hadoop. We used Spark for this implementation.

Although the accuracy was not good enough, it was a good experience to learn a new Java based Deep Learning library.

The Flow:


Read Image and Labels file from S3
Concatenate Labels and Images
Describing Model
Fitting the Model
Evaluating the Model

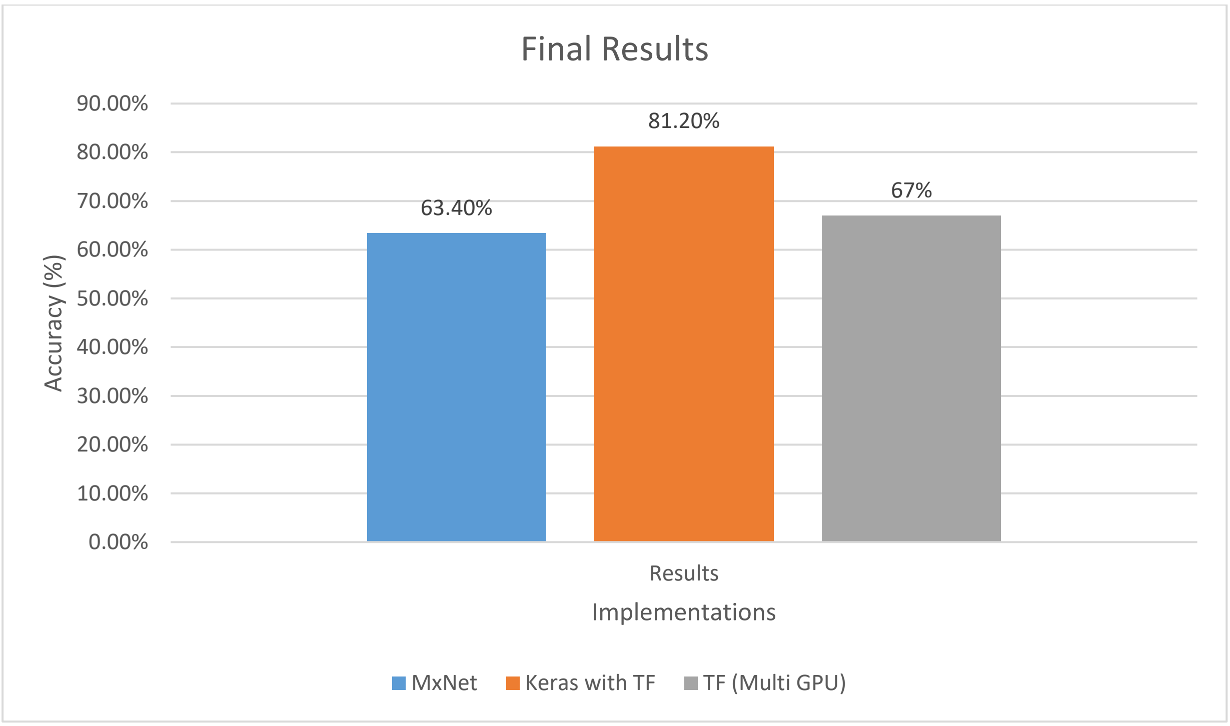
What we tried and didn’t work

1) The biggest disappointment was DeepLearning4J, it was difficult to setup the dependencies on Spark; a lot of dev time went fruitless after this attempt.
2) Many different network architectures were tried that gave sub-par accuracies.
3) Implemented parallel convolution layers in mxnet, no significant accuracy improvement obtained, maybe we did it wrong, or maybe it doesn’t work.

Results

We achieved the best accuracy with **Keras** implementation, i.e. **81.2 %.**

Following is the graphical representation of the various results we achieved.



References

1. CNN Basics - http://cs231n.github.io/convolutional-networks/
2. Book for Keras implementation - Deep Learning with Python by Jason Brownlee
3. http://www.kdnuggets.com/2016/03/must-know-tips-deep-learning-part-1.html
4. https://www.tensorflow.org/versions/r0.11/tutorials/deep_cnn/index.html