Machine Learning Engineer Nanodegree

Capstone Project

Rachit Kumar Agrawal
September 25th, 2017

I. Definition

(approx. 1-2 pages)

1. Project Overview

I have picked up the domain of Computer Vision for my capstone project. In last decade, the amount of data has exploded exponentially. These data are in different forms - text, image and video. With each of these forms of data exploding, it becomes extremely important to categorize them or classify them and classification of these images is not humanly possible. So, it becomes important to let machine do it for us. I am personally motivated for this problem because, I feel this is first step to my ultimate goal to create a prototype for the Autonomous Driving.

1.1 Dataset

In this work, I am using the publicly available CIFAR-10 dataset. This dataset was collected ty students at MIT and NYU over the span of six months; it is described in detail in [1]. They assembled it by searching the web for images of every non-abstract English noun in the lexical database WordNet [2][3]. Two popular subsets were created out of this large dataset of tiny images called – CIFAR-10 and CIFAR-100. I am using CIFAR-10 dataset for this work. This dataset has 6000 examples of each of 10 classes. Out of these 6000 examples from each class, 5000 examples are kept for training and the remaining 1000 are kept for the evaluation of the model.

2. Problem Statement

The problem I am trying to a multi-class classification problem in one of the very popular image dataset CIFAR-10. In short, "Object Classification on CIFAR-10 dataset". The goal is to classify given RGB 32x32 pixel images across 10 object categories. The performance of this prediction is then compared with the label provided along with the dataset.

Based on the literature survey, it has been identified that Convolution Neural Network is the best approach to solve any Image Classification problem. I plan to use AlexNet architecture, one of the very popular architecture for CIFAR-10 dataset. Even though, this architecture is quite famous in itself, it’s known to still take considerable time. I plan to give less weightage to the edge pixels by using Conv Layers at the beginning of my ConvNet so as to reduce the working shape to 28x28 after couple of layers.

3. Metrics

There are many evaluation metrics – precision, recall, f1-score, accuracy etc. Among all, accuracy is one of the most intuitive metric but it has its own shortcomings. Accuracy may depict an incorrect picture where there is an imbalance in the number of data belonging to different classes. But in our project, we have equal number of data to be predicted from each of the classes (1000). Hence, we don't expect to suffer from the same problem and we plan to use *accuracy* as our performance evaluation metric. The accuracy is defined as follows:

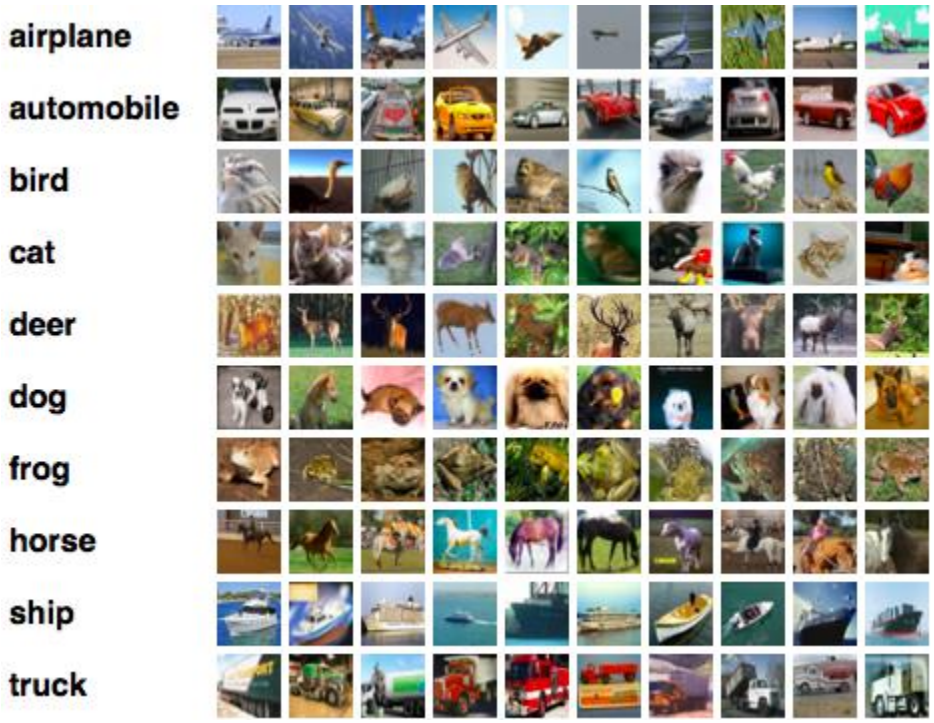
$$Accuracy = \frac{\text{Number of images correctly predicted}}{\text{Total images in the dataset}}$$

II. Analysis

(approx. 2-4 pages)

1. Data Exploration

The dataset for this work has been obtained from the *keras.datasets* module. There are many ways to obtain this dataset. We chose to use *keras* module as it's one of the easiest way to get this dataset and saves us from all the data fetching part of the work. The dataset has total 60000 images – 50000 for training and 10000 for testing. These data points are well distributed among all the classes and hence we have 5000 training data for each label and 1000 testing data for each label. Some of the example images from each of these classes are shown below:



The input images provided are 32x32 RGB images. Even though there are substantial number of images per class but they are not enough to get a good score on a CNN.

2. Exploratory Visualization

We try to explore the images from the training dataset and see how they look like. In the image below, I have plotted the first 10 images from each class:

The classes of data points that are given is very interesting. We have very varied classes except two class – dog and cat. I personally feel these two classes have a lot of common features. Rest all the classes will have their own unique features but **cat** and **dog** are two classes which will have a lot of common features and specially with such low-resolution images, there are high chances that they might be predicted incorrectly of the other class (dog as cat and cat as dog).

3. Algorithms and Techniques

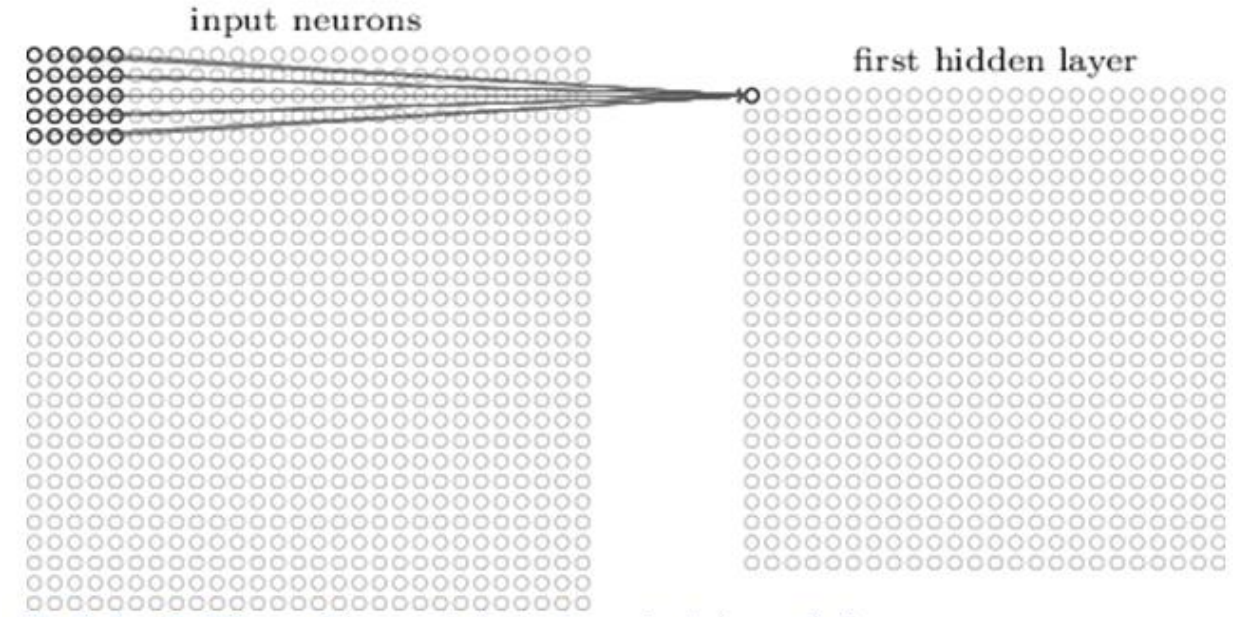
3.1 Convolution Neural Network

I plan to use Convolution Neural Network (CNN) for this project. In machine learning, a convolution neural network is a class of deep, feed-forward artificial neural networks that has successfully been applied to analyzing visual imagery. CNNs use a variation of multilayer perceptron designed to require minimal preprocessing [4]. Convolution networks were inspired by biological processes [5].

A CNN consists of an input layer, a output layer and multiple hidden layers. The hidden layers can be either convolution(CNV), pooling(PL) or fully connected(FC). We will consider each of these layers in detail in the following section.

3.1.1 Convolution Layer (CNV):

The Convolution Layer is the core building block of a Convolution Network that does most of the computation heavy lifting. The CNV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a CNV might have size 3(W)x3(H)xN(D). N, represent the number of features we want to extract in this layer from the images. For our model in this study, N is chosen as 32. During the forward pass, we slide(convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation. Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume. An example of convolution operation is shown in figure below:



Visualization of 5 x 5 filter convolving around an input volume and producing an activation map

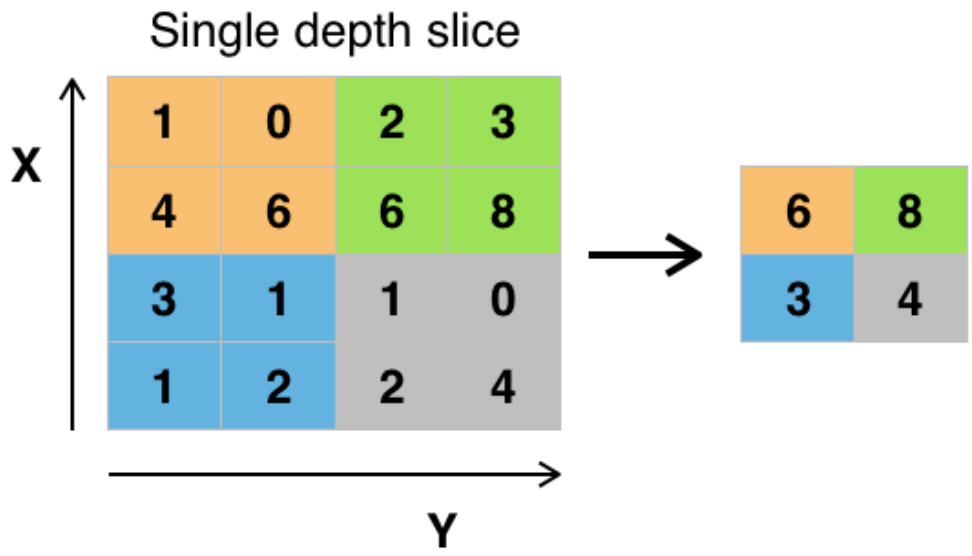
3.1.2 Activation Layer:

After each conv layer, it is convention to apply non-linear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations). There are many type of activation functions like *sigmoid, tanh and ReLU* but researchers found out that ReLU layers work far better because the network can train a lot faster without making a significant difference to the accuracy. The *tanh* and *sigmoid* function suffers from vanishing gradient problem because of the slope being too low for high numbers. ReLU

doesn't suffer from this problem and hence it trains faster. The ReLU [6] layer applies the function f(x) = max(0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0. This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the CNV layer. For these benefits, we have used *ReLU* activation function with all the CNV layer for this study.

3.1.3 Pooling Layer (PL):

After some ReLU layers, programmers may choose to apply a pooling layer. It is also referred to as a down-sampling layer. In this category, there are also several layer options, with *MaxPooling* being the most popular. This basically takes a filter (normally of size 2x2) and a stride of the same length. It then applies it to the input volume and outputs the maximum number in every sub-region that the filter convolves around.



Example of Maxpool with a 2x2 filter and a stride of 2

Other options for pooling layers are average pooling and L2-norm pooling. The intuitive reasoning behind this layer is that once we know that a specific feature is in the original input volume (there will be a high activation value), its exact location is not as important as its relative location to the other features. We have used a *MaxPooling* Layer of size 2x2 with a stride of 2.

3.1.4 Dropout Layer (Dropout):

Droput is a regularization technique to reduce over-fitting in neural networks and we perform this regularization functionality in this layer. It is very easy to lead to over-fitting with any neural network due to the number of parameters going highly complex even with a small neural network. Dropout is a very efficient way of performing model averaging with neural networks. This layer expects probability as the fraction of inputs to be dropped randomly. We use three such layers in our model.

3.1.5 Fully-connected layer (FC):

After several convolution and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. We have used one FC layer in our model with 512 neurons, details of which has been explained in the latter section.

3.1.6 Classification Layer (CL):

This is typically final layer of a CNN that transforms all the net activations in the final output layer to a series of values that can be interpreted as probabilities. One of the most common function is *softmax* function. The property of the *softmax* function is that it makes sure that the total probability of all the possible classes is 1.

3.2 Neural Network Framework

Neural Network Framework has become popular off late because these frameworks are more efficient than implementing this Neural Network using NumPy because these frameworks knows the entire computation graph that must be executed, while NumPy only knows the computation of a single mathematical operation at a time. These makes all the frameworks easy to use and much faster than standard python libraries.

We had multiple options to choose from among the various Neural Network framework available. I did my literature survey to compare these different frameworks and came up with the following summary:

Property	Caffe	Neon	TensorFlow	Theano	Torch
Core	C++	Python	C++	Python	Lua
CPU	Yes	Yes	Yes	Yes	Yes
Multi-threaded CPU	Blas	Only Data Loader	Eigen	Blas, conv2D, limited OpenMP	Widely Used
GPU	Yes	Customized nVidia backend	Yes	Yes	Yes
Multi-GPU	Only Data Parallel	Yes	Most Flexible	No	Yes
Nvidia cuDNN	Yes	No	Yes	Yes	Yes
Quick Deploy, on standard Models	Easiest	Yes	Yes	No	Yes
Auto Gradient Compu	Yes	Yes	Yes	Most flexible	Yes

Looking at the above comparison metric, *Torch* and *TensorFlow* are the top two choices. But

TensorFlow has a better support for Multi-GPU. This is a very important metric because, GPUs are heavily used for the training and having better support for multi-GPU means huge time savings. Due to this, I chose TensorFlow with Keras as the framework to learn and use for my project.

4. Benchmark

There have been many solutions for CIFAR-10 dataset. Among these, the state-of-the-art solution has been provided by Graham, Benjamin called Fractional Max-Pooling[7] where he achieved a result of **96.53%** accuracy on the test dataset. The model that he uses in his work uses a much more complex model than what we have done in this study. His model also needs days of training on very high-end GPUs. For us, in the absence of such high-end hardware, we make a simpler model and compare to see how much accuracy can be achieved with a reasonable amount of training in a standard hardware.

III. Methodology

(approx. 3-5 pages)

1. Data Preprocessing

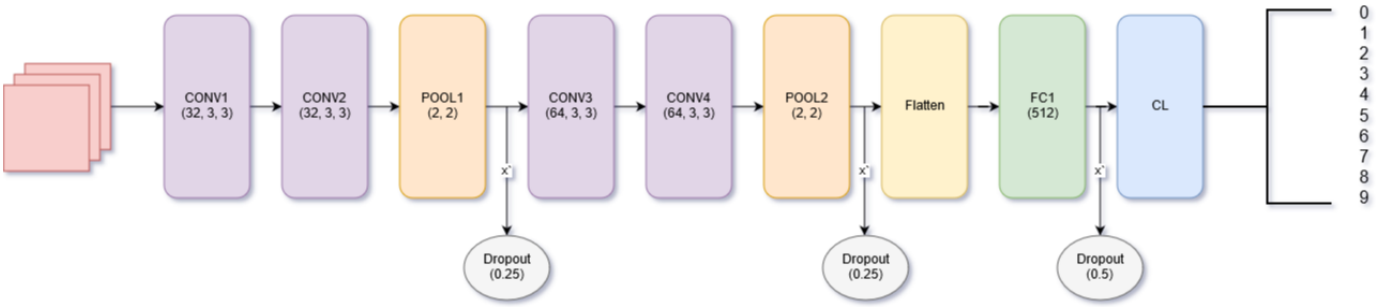
Several works in the past has shown that edge pixels contribute very less to the actual features of an image [8]. These edge pixels have a lower weightage in our model as we use two CONV layers that reduces the size of input space to 28x28 after 2 layers. Also, we normalize the image pixels from the range of (0-255) to (0-1).

Apart from normalizing the input images, we also convert the output integer labels of the images into one-hot encoding labels. Integer labels enforce order, but our classes don't have any order. To remove this incorrect ordering information for the output labels we convert the output integer labels into one-hot encoding labels. For example: integer label 4 will be represented as:

[0 0 0 0 1 0 0 0 0 0]

2. Implementation

The modified AlexNet architecture model is used for this study which is shown as below:



After the pre-processing, the images are passed through different layers with following parameters:

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 30, 30, 32)	896
conv2d_2 (Conv2D)	(None, 28, 28, 32)	9248
max_pooling2d_1 (MaxPooling2	(None, 14, 14, 32)	0
dropout_1 (Dropout)	(None, 14, 14, 32)	0
conv2d_3 (Conv2D)	(None, 12, 12, 64)	18496
conv2d_4 (Conv2D)	(None, 10, 10, 64)	36928
max_pooling2d_2 (MaxPooling2	(None, 5, 5, 64)	0
dropout_2 (Dropout)	(None, 5, 5, 64)	0
flatten_1 (Flatten)	(None, 1600)	0
dense_1 (Dense)	(None, 512)	819712
dropout_3 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 10)	5130

There are total 890410 tunable parameters.

3. Refinement

The initial training was done only for 3 epochs. The initial result was ~60% with only 6 minutes of training. This result was improving with every increase in the training iteration. To get better accuracy we increased the number of epochs to 40. This increased the validation accuracy to ~79%. These 40 epochs were achieved in close to 80 mins. To achieve an even better accuracy we chose an even higher number of epochs to a point where the training time is still reasonable and we probably see higher accuracy. So, for this we increased the number of epochs further 1.5

times to 60 epochs. But with increase in another 20 epochs the accuracy didn't improved much. So, this model can be seen as giving 79% accuracy in just 80 mins and ~80% with total time of 2 hours. Based on the time, this is a highly efficient model that gives decent enough accuracy in very quick time.

Apart from this another refinement used was to use Batch Normalization Layer. Batch Normalization typically helps in learning the features faster. Also, it is typically applied after the activation layer and before the regularization layer(Dropout). We created another model which had this BN layer before every Drop Out Layer in the previous model. We did observe that the model learned the features faster and achieve the peak accuracy in just 27 epochs in comparison to 41 epochs taken by the previous model. Details of both are shown below:

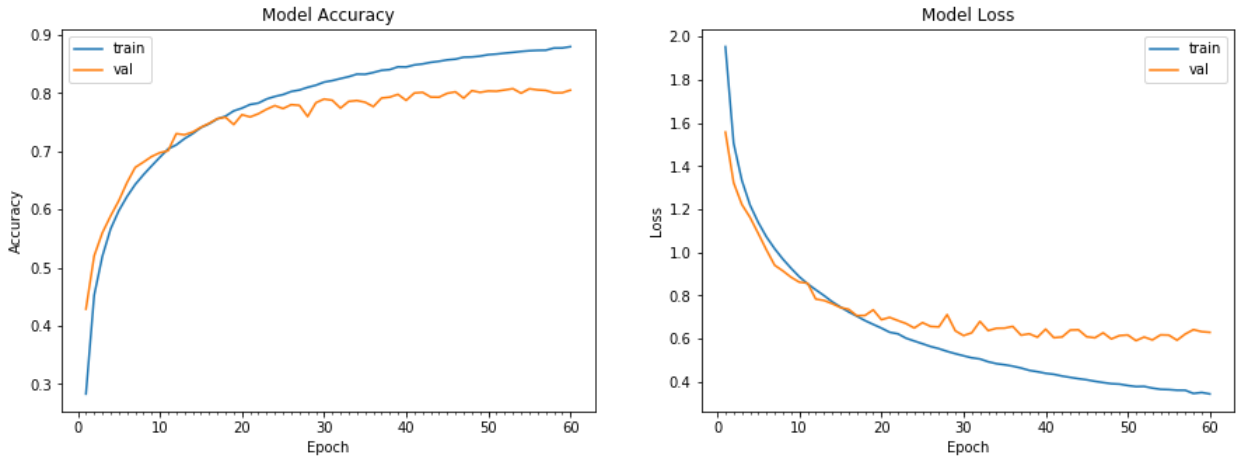


Figure 1: Model1: AlexNet Modelling without Batch Normalization Layers

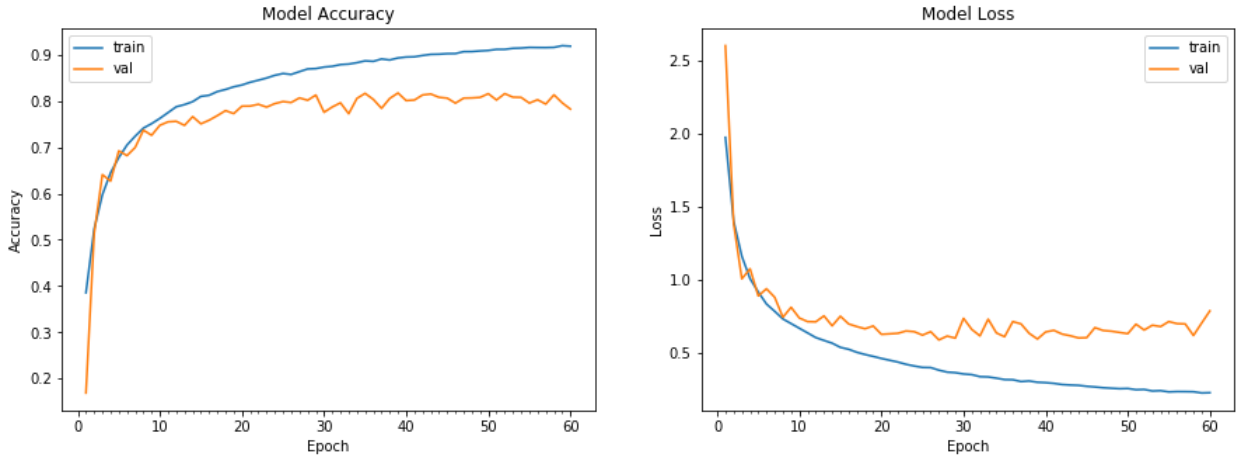


Figure 2: Model2: AlexNet Modelling with Batch Normalization Layers

IV. Results

(approx. 2-3 pages)

1. Model Evaluation and Validation

For the model evaluation, we use all the 10,000 images in the evaluation set of CIFAR-10. It calculates the accuracy described in the Evaluation Metrics section.

System Information:

- Intel Core i5 – 7300 @ 2.6 GHz.
- Memory: 16 GB RAM

Total 60 epoch optimization iterations are used for this work. After 60 epochs, the classification accuracy is about 80% on the test-set.

Our model that gives an accuracy of ~80% is much lower in comparison to the benchmark model but when the training time and hardware at disposal is considered, it is a respectable result. The benchmark result is a highly complex model with many days of training while our model is relatively simpler and we could achieve the result of 80% accuracy after ~2 hours of training.

We also saw the value of Batch Normalization Layer in the modified model that gave us faster learning. Model with Batch Normalization Layer achieves the peak accuracy at a faster rate than the one without it.

V. Conclusion

(approx. 1-2 pages)

In this study, we achieved a decent classification accuracy with a relatively simpler CNN architecture and in very quick time with the use of TensorFlow. This lays a strong foundation for various problems in the domain of Computer Vision. Also, even though our result in this study is promising, there are many latest techniques that can be used to improve the results even further.

We used two variants of AlexNet and achieved 80% accuracy with little over 2 hours of training. This is without even any GPU at our disposal. One of the models used BN layer which increased the learning rate and suggests that we should use BN layer whenever possible in our CNN to have faster learning.

One way in which the above model can be improved in terms of speed is by using depth-wise separable convolutions to build light weight deep neural networks [10]. A new technique recently presented by Andrew G. Howard et. al. A standard convolution both filters and combines inputs into a new set of outputs in one step. The depth-wise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size. There are several other models that shows better results but most of them needs heavy training time and usage of high end GPUs. In the absence of such high-end hardware at my disposal, improving our model to use such depth-wise convolution seems to be next logical step to greatly improve the speed

that can be used on mid-range hardware and yet give improved results in about same training time.

VI. References

1. Torralba, A., Fergus, R., Freeman, W. T., 2008, 80 million tiny images: a large dataset for nonparametric object and scene recognition.
2. WordNet can be found at http://wordnet.princeton.edu/
3. Miller, G. A., 1995, WordNet: a lexical database for English.
4. LeCun, Yann. *LeNet-5, convolution neural networks*.
5. Matusugu, Masakazu; Katshhiko Mori; Yusuke Mitari; Yuji Kaneda (2003). *Subject independent facial expression recognition with robust face detection using a convolution neural network*. Neural Networks. 16 (5): 555-559.
6. Vinod, Nair; Geoffrey E. Hinton. *Rectified linear units improve restricted Boltzmann machines*. ICML' 10 Proceedings of the 27th International Conference on International Conference on Machine Learning, June 21- 24, 2010, USA
7. Graham, Benjamin. *Fractional Max-Pooling.* In *arxiv:cs/arXiv:1412.6071,* 2015.
8. Krizhevsky, Alex. *Convolution Deep Belief Networks on CIFAR-10.* Unpublished manuscript, 2010.
9. Krizhevsky, Alex. *Learning multiple layers of features from tiny images*. Computer Science Department, University of Toronto, Tech. Report, 2009.
10. Howard, Andrew G; Zhu, Menglong; Chen, Bo; Kalenicenko, Dmitry; Wang, Weijun; Weyand, Tobias; Andreetto, Marco; Adam, Hartwig. *MobileNets: Efficient Convolution Neural Networks for Mobile Vision Applications.* arXiv:1704.04861v1, 17 Apr 2017.