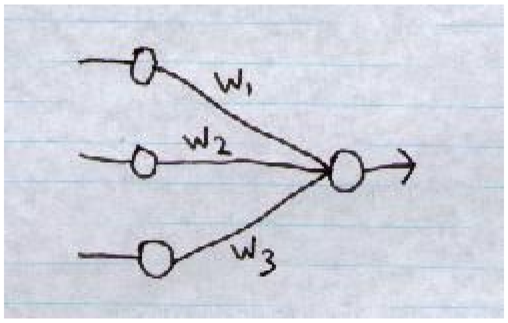
Introduction:
In this assignment, I worked on training a deep network for a classification task on the CIFAR-10 dataset. The report that follows will be detailing my work and my results regarding this same.

Dataset:
CIFAR – 10 is a dataset consisting of 60000 images each of size 32x32. This dataset is used for object classification. "10" in the "CIFAR-10" indicates the number of classes/labels that are present in the datasets. Thus, there are 10 classes which are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. Out of the 60000 images, 50000 are used for training (five training batches) and 10000 are used for testing (1 test batch). The test batch will contain 1000 images that are randomly chosen from each class while there will 5000 images from each class for training. This dataset was collected by Alex, Nair and Geoffrey.

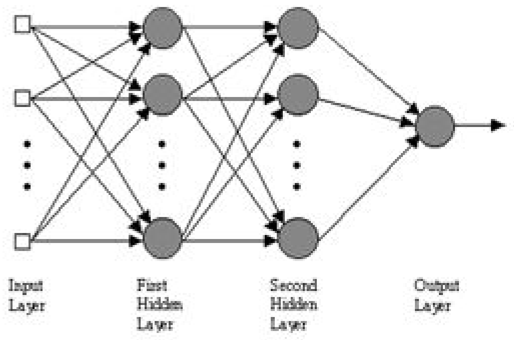
The dataset can be downloaded from:
https://www.cs.toronto.edu/~kriz/cifar.html

Some terms which need to be known:

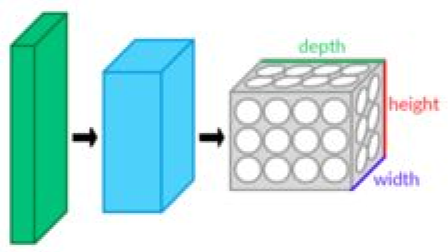
Perceptron: It defines a hyperplane in a multi-dimensional space. Basically, it takes in n-inputs. It then takes a dot product with the weight vector it has. To this, a bias is added to produce a result. This result is sent through another function, in some cases, and the output is got. They form the fundamental units in a classification task. They can be used to detect a single class or in multi-class detection.



Multi-Layered Perceptron: This is the term given to the setup which contains several of the perceptrons lined up in different layers. Generally, an MLP has an input layer where the input can be features or images or speech, depending on the task, and then there are several layers of perceptrons and then an output. This is also used in learning a function (classification in our case). This is also known as an artificial neural network.



Convolutional Neural Networks: They are similar to the multi-layered perceptron described earlier. Each neuron has a receptive field. Let’s describe this in terms of images as classification based on images is the task at hand. A receptive field is the number of neurons it takes input from if the neuron is in a hidden layer or the number of pixels it sees if the neuron is in the input layer. In case of an input layer, the intuition becomes clearer. In a CNN the neurons are placed in 3d blocks. Please look at the image below to get more idea.



Neurons are packed in a 3d volume. The green block is one layer of CNN, the blue block is another and similarly, the white box. Each layer of CNN has slices of neuron matrices. When an image is given as an input to a CNN layer, each neuron will interact with some number (parts of the image) of pixels based on the receptive field value of the neuron. This interaction can seen as a dot product with the weight vectors associated with the neurons and this dot product is a convolution operator. The receptive field is the window size of the convolution operator. Packing such slices of neurons in 3d blocks and having several layers of such blocks form a CNN.
The layers where a convolution is done with a filter, it is called a “conv” layer. There will be some layers where “pooling” is done. Pooling can also be looked at as convolution and working on the basis of receptive fields. There are 3 kinds of pooling:

- Max_pooling: Here, the neuron selects the maximum pixel value from the pixels that fall into its receptive field. (the dot product with the filter gives you the max)
- Mean pooling: Here, the neuron takes the mean of all the pixels that fall into its receptive field. (the dot product here, gives you the mean)

- Stochastic Pooling: Probabilities of each element in the block falling in the receptive field are computed. These are sorted. A random selection is made. (the dot product here gives a random selection)

Fully connected network: This is a network where every neuron is connected to every other neuron in its successive layer.

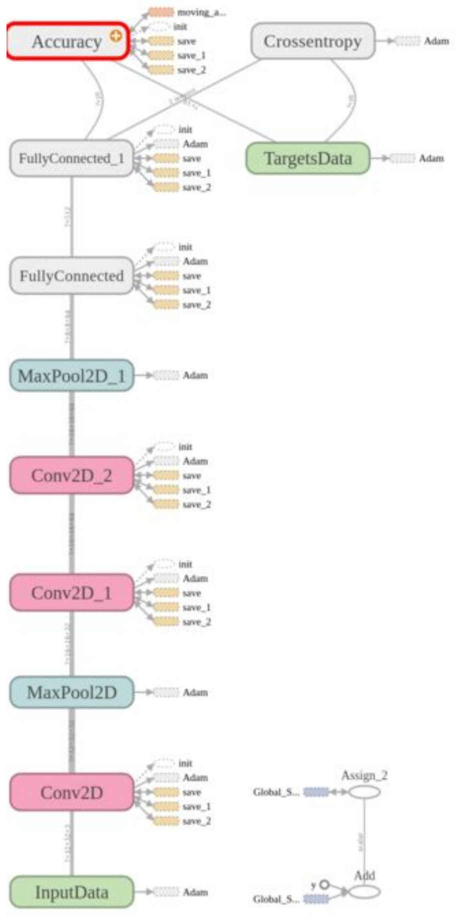
Methods used for fitting: essentially, a CNN or any neural network is to find a multidimensional curve for the given data. To fit the curve, one goes through an iterative procedure minimizing a cost function. Some parameters involved in fitting the curve are:

- Regression: The regression are the methods used to solve the problem at hand (to estimate a function). There are several methods like “adam”, “stochastic gradient descent”, etc.
- Loss functions: In any regression, a cost function is minimized. That cost function is the loss function. This loss function can be mean square error (MSE can be asked to be reduced) or cross entropy.

Activation functions: These are the functions through which the dot product (a bias may be added here sometimes) is passed. Some activation functions and their implications are:

- Rectified Linear Unit (ReLU): this computes the function, $f(x) = max(0, x)$. This activation function supposedly accelerated the convergence of stochastic gradient descent method.
- Sigmoid: computes the function, $\sigma(x) = 1/(1 + exp(-x))$. This takes in a real value and squashes it between 0 and 1.
- Tanh: it is the regular hyperbolic tangent function

Network deployed in the assignment:
There will be an input layer, convolution layer, a max pooling layer, two convolution layers followed by a max pooling layer. This is followed by two fully_connected layers. This block diagram for the network from tensorboard is as follows.



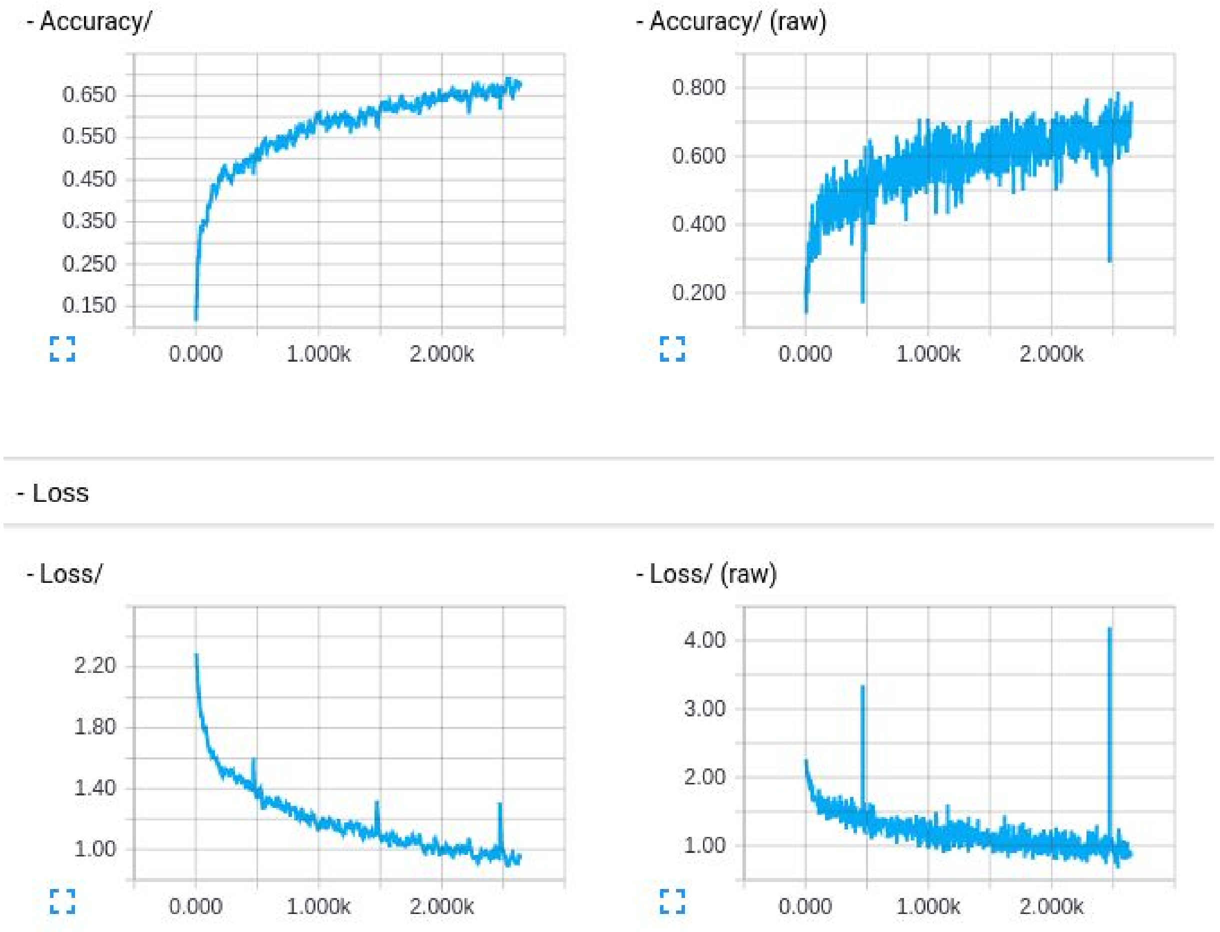
The classification is done, as mentioned above, on cifar-10 dataset. This network is deployed on tflearn which provides a simple interface for theano and tensorflow. I followed a tutorial in implementing this network.
The tutorial is located at:
https://www.tensorflow.org/versions/r0.8/tutorials/deep_cnn/index.html

The sample code which I referred to is at:
https://github.com/tflearn/tflearn/blob/master/examples/images/convnet_cifar10.py

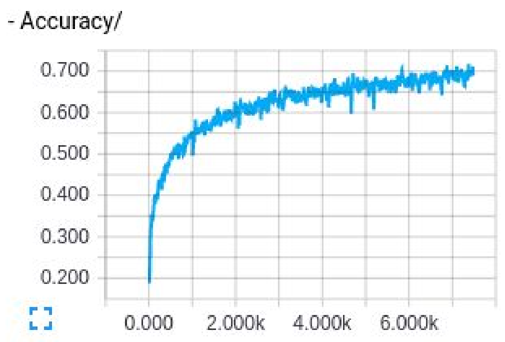
The loss functions I tried comparing are Mean Square Error and Cross Entropy. The Optimizers I tried comparing are Stochastic Gradient Descent and ADAM. Also, n-fold validation is done. The dataset is divided into 10 parts and shuffled and passed to the network.

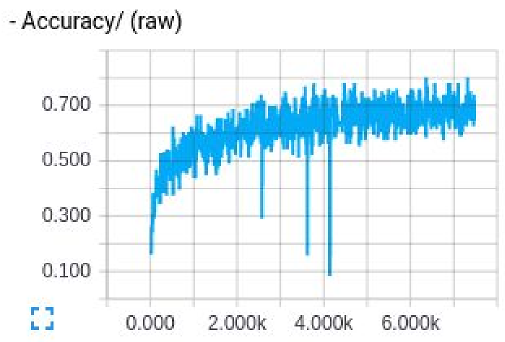
The network with loss function as MSE and optimizer as SGD was getting trained faster and it got converged quicker compared to the netowks which used Cross Entropy as the loss function and SGD as the optimizer and also the network with Cross Entropy as the loss function and ADAM as the optimizer.

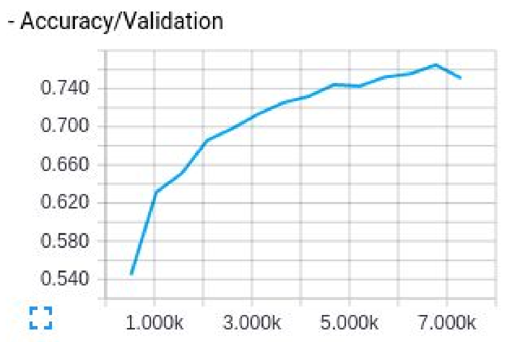
The accuracy with “cross_entropy” as the loss function and the optimiser as “ADAM” is 68%. The loss is around 0.92. The corresponding graphs are:

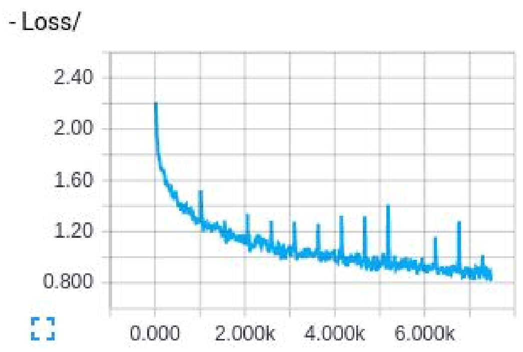


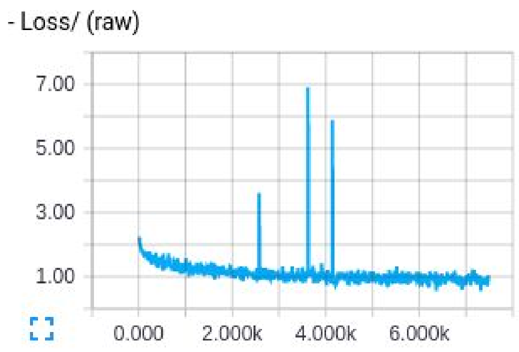
I included a dropout layer between the two fully connected layers and with the loss functions and the optimizers intact. I got an accuracy of 75% and a loss of 0.74. The graphs are as follows:

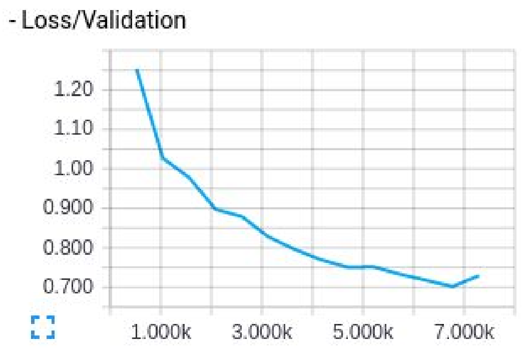




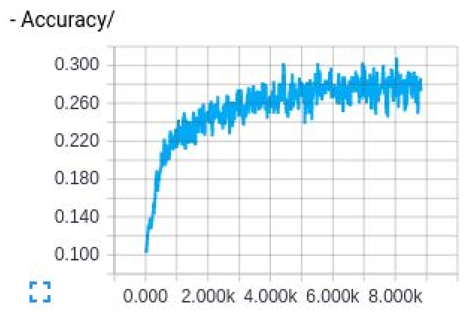


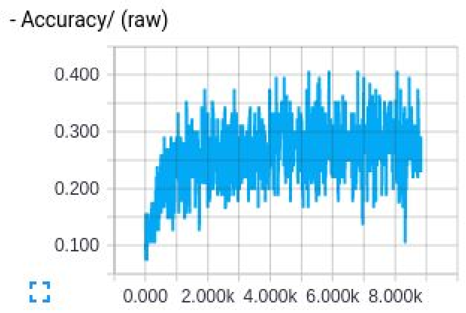


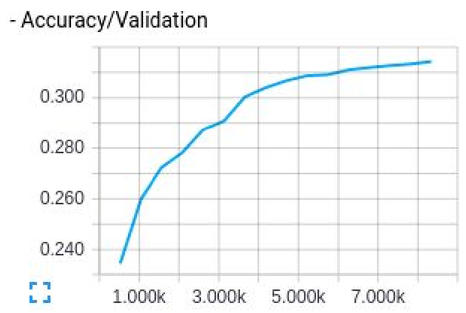


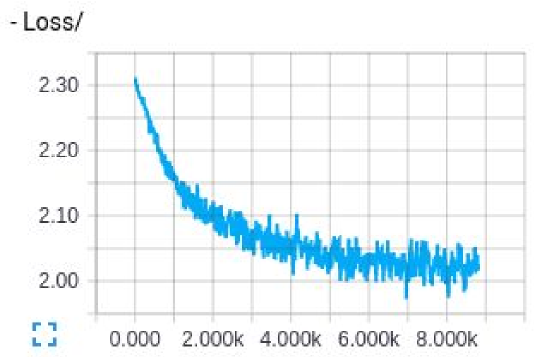


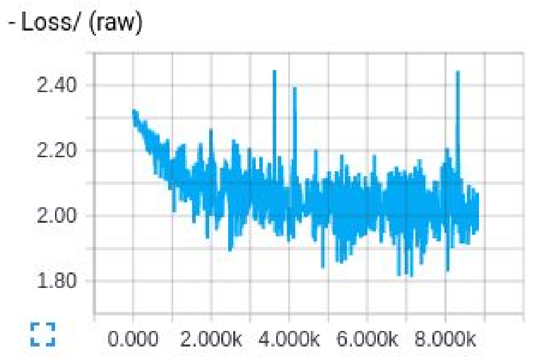
Then, I changed the optimizer to SGD with the, “cross_entropy” as the loss function. I got an accuracy of 31% and loss of 1.94. The graphs are as follows:

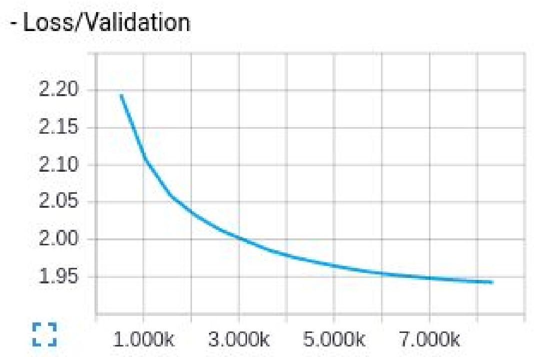












I then changed the loss function as MSE with SGD as the optimizer. The accuracy was just 10%. But the convergence rate was faster.

Conclusions:

- Implemented a deep net which could classify the cifar10 dataset into 10 classes.
- The optimiser, “adam” with the loss function of “cross_entropy” provided the best results.
- The Dropout layer seems to be affecting the results positively since I was able to see better accuracy when a dropout layer was included as opposed to when the dropout layer wasn’t there.
- The optimiser, “sgd” with the loss function of “mse” did not perform well.