Capstone Project

Chiranjeev Ghai

Machine Learning Engineer Nanodegree

August 18 ,2016

Definition

Project Overview

Image Recognitions is of utmost importance in today's world. As computers are getting better in image recognitions due to advances in Deep learning many new concepts have emerged like: -

- Auto tagging of Friends in photograph on social network.
- Recommending similar items from a photograph.
- Translate written text into one's preferred language,
- Recognize text and objects in an image for a visually impaired person.

In this project I am training machine to recognize digits (0-9) in an image.

Problem Statement

The goal is to train a model so that it can recognize digits (0-9) in an image with good accuracy. For this I used SVHN datasets for training and testing purpose:-

The tasks involved are following: -

1. Download and load training and testing datasets (.mat) file in memory
2. This will give be a dictionary of two variables
    - 'X ': - a 4D matrix of image
    - 'y': - labels for corresponding images
3. X and y are extracted and kept in separate variables.
4. Images are in format of [height, width, channels, no of images]
5. Reshape images in format of [no of images, height, width, channels] for simplifications purpose.
6. One hot encoded the labels
7. Create a convolution neural network.
8. Train the network with data
9. Find Accuracy on test data and Optimize for better results.

Metrics

Since this is a multi classification problem, some of the metrics options I have are 1) Accuracy 2) confusion matrix 3) log loss

Accuracy

Accuracy simply measures how often the classifier makes the correct prediction. It's the ratio between the number of correct predictions and the total number of predictions (the number of test data points). It takes into account both true positives and true negatives with equal weight.

$$accuracy = \frac{true\ positive + true\ negative}{dataset\ size}$$

Confusion Metrics

Accuracy does not make distinction between classes. Correct answers for class 1 or other classes are treated equally. To get a more detailed breakdown of a correct and incorrect classification of a particular class we use confusion metrics. Precision and Recall are part of this. A sample confusion matrix is for binary classification Is shown below: -

```
                       Condition: A        Not A

  Test says “A”      True positive   |   False positive
                      ----------------------------------
  Test says “Not A”  False negative  |    True negative
```

The generalization to multi-class problems is to sum over rows / columns of the confusion matrix. Given that the matrix is oriented as above, i.e., that a given row of the matrix corresponds to specific value for the "truth", we have:

$$Precision_i = \frac{M_{ii}}{\sum_j M_{ji}}$$

$$Recall_i = \frac{M_{ii}}{\sum_j M_{ij}}$$

That is precision is the fraction of events where we correctly declared *i* out of all instances where the algorithm declared *i*. Conversely, recall is the fraction of events where we correctly declared *i* out of all of the cases where the true of state of the world is *i*.

Log loss

Its the logarithmic loss. It gets into the finer details of the classifier. If the raw output of the classifier is a numeric probability instead of a class label of 0 or 1, then log-loss can be used. It output probabilities between 0 and 1 and penalizes extreme value. This works well in situations where the classifier should be wary of being overly confident or unable to predict anything -- input data should be pre-processed for outliers in order to minimize log loss.

$$logloss = -\left(\frac{1}{N}\right)\sum_{i=1}^{N}\sum_{J=1}^{M} y_{ij} log(p_{ij})$$

In this project the requirement is to classify digits from 0-9 and there is no need for per class accuracy. So I guess **accuracy** metric which gives me result of overall accuracy on each class is a good metric for this project. The steps for finding the accuracy I used in my projects are: -

1. The output layer consists of 10 neurons which predict classes from 0-9
2. The readout applies softmax to the output making the output range from 0-1
3. The maximum value from the predicted layer is selected and matched with the true value
4. The success is predicted as 1 and 0 for failure and average is taken.

Analysis

Data Exploration

Training and **testing** data are provided in .mat files and can be loaded in python using scipy easily.

Training data can be downloaded from URL

http://ufldl.stanford.edu/housenumbers/train_32x32.mat

Testing data can be downloaded from URL

http://ufldl.stanford.edu/housenumbers/test_32x32.mat

Training data image size [32,32,3,73257]

Training data image size [32,32,3,26032]

All digits have been resized to a fixed resolution of 32-by-32 pixels. The original character bounding boxes are extended in the appropriate dimension to become square windows, so that resizing them to 32-by-32 pixels does not introduce aspect ratio distortions. Nevertheless, this preprocessing introduces some ''distracting'' digits to the sides of the digit of interest. Loading the .mat files creates 2 variables: '''X''' which is a 4-D matrix containing the images, and '''y''' which is a vector of class labels.

No of images per label in training data are

```
{'10': 4948, '1': 13861, '3': 8497, '2': 10585, '5': 6882, '4': 7458, '
7': 5595, '6': 5727, '9': 4659, '8': 5045}
```

No of images per label in testing data are

```
{'10': 1744, '1': 5099, '3': 2882, '2': 4149, '5': 2384, '4': 2523, '7'
: 2019, '6': 1977, '9': 1595, '8': 1660}
```

In this data set we see that image with 0 digit in the image are labeled 10.

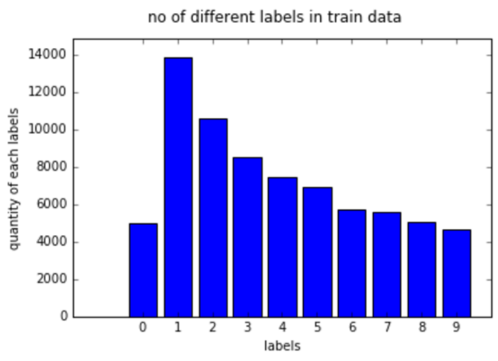
As training and testing data are both 32X32X3 that is 32 height, 32 width and 3 channels for RGB and maximum value of RGB is 255 so I normalize the data in range of -1 to 1 so that a m achine can better understand the images and give better accuracy.

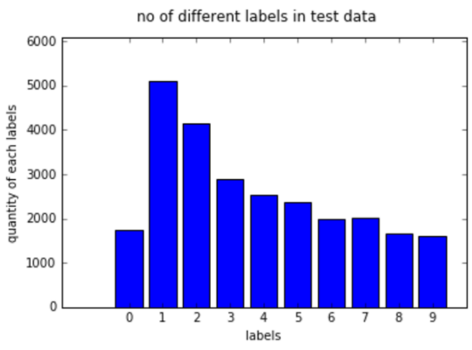
Exploratory Visualization

Below are some random images from both train and test sets

- Each image is in shape of 32X32X3 where first and second 32 are for height and width while 3 is for no of channels
- As there are three channels that is RGB and each image value range from 0-255
- Generally for image classification an edge detection algorithm is a good choice.
- Neural network is a good algorithm for edge detection.
- Images in training and testing dataset are of very low resolution and blurry.
- Compared to other image classification algorithms, convolutional neural networks use relatively little pre-processing. This means that 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 a major advantage for CNNs.

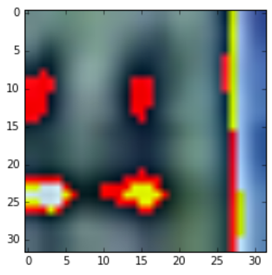
- Added histogram for both training and testing dataset for in depth visualization of each label in dataset: -



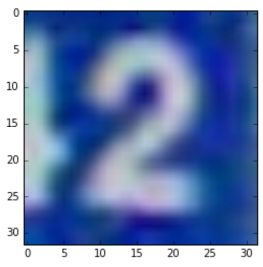


- We can see in both histograms the maximum no of images are for 1 and 2 and even other label are present in almost sane ratio in both dataset.

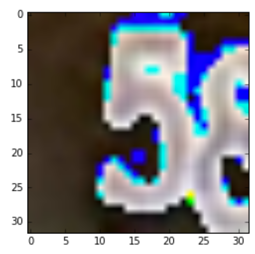
Images from Training set: -



Number 4

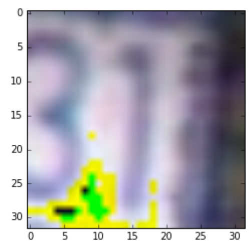


Number 2

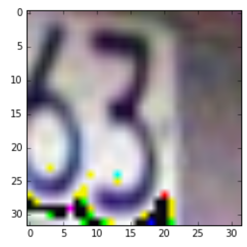


Number 5

Images from Testing set: -



Number 1



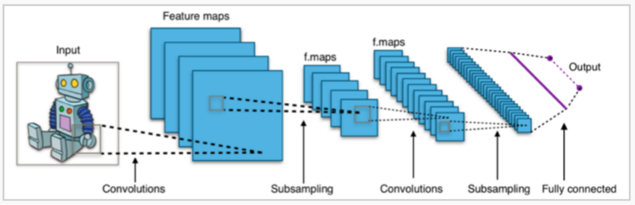
Number 3

Algorithms and Techniques

The algorithm I used is a Convolution Neural Network, which is the state-of-the-art algorithm for most image processing tasks, including classification. A convolutional neural networks (CNNs) consist of multiple layers of small neuron collections which process portions of the input image, called receptive fields. The outputs of these collections are then tiled so that their input regions overlap, to obtain a better representation of the original image; this is repeated for every such layer.

A CNN architecture is formed by a stack of distinct layers such as: -

1. **Convolution Layer**: - It will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x32] if we decided to use 32 depth.
2. **RELU (Rectified Linear Units)**: - It will apply an element wise activation function, such as the max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x32]).
3. **Pooling Layer**: - This layer will perform a down sampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x32].
4. **Fully Connected Layer**: - This layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of SVHN digits (0-9).
5. **Loss layer**: - The loss layer specifies how the network training penalizes the deviation between the predicted and true labels and is normally the last layer in the network. Softmax function is applied in this layer



Typical CNN architecture

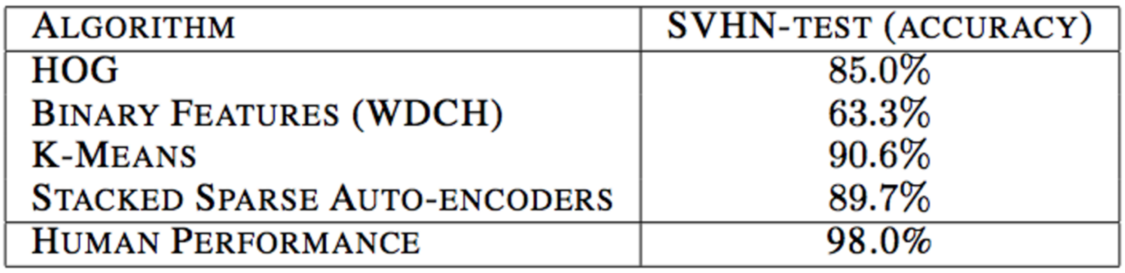
There are certain parameters that can be played around to optimize the performance of the classifier. Some of these parameters are as follows: -

- Kernel Size: - the size of the patch whether 5X5 or 3X3 or any other.
- Stride: - the no of pixels to shift each time the filter is moved.
- Padding: - Same padding (Zero padded) or Valid padding.
- Training length (number of epochs).
- Batch Size: - How many images to be processed per epochs.
- Learning Rate: - How fast to learn this can be dynamic.
- Dropout probability: - At each training stage, individual nodes are either "dropped out "of the net with a probability of 1-*p* or kept with probability of *p.* The dropout method is used to prevent overfitting.

Benchmarking

I look into the accuracy most people are getting on MNSIT data for digit recognition it was around 97%, but images on MNSIT are much more refine than SVHN as we can see from images attached above due to low resolution it's really hard even for humans to recognize the digits. So I kept the benchmark **85%** as good accuracy and around **90%** as great accuracy.

Generally, a machine is well trained if it's able to predict as per human level accuracy, but in svhn it is really hard to match human level accuracy which is ~98%. Let's just see some other algorithms and how well they perform on this dataset



ALGORITHM	SVHN-TEST (ACCURACY)
HOG	85.0%
BINARY FEATURES (WDCH)	63.3%
K-MEANS	90.6%
STACKED SPARSE AUTO-ENCODERS	89.7%
HUMAN PERFORMANCE	98.0%

So, we can see that the maximum accuracy achieved in this case is 90.6% by K-MEANS. And most other achieve a respectable score between 85-90%. So I believe achieving a score between **85-90%** is good enough. The source for the table is mentioned below.
Source: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

Methodology

Data Preprocessing

The preprocessing done on the data consist of following steps.

- Images in both training data as well as testing data are in the shape of [32, 32, 3, no_of_images]. So, reshaped it to a more convenient shape [no_of_images, 32, 32, 3]
- Normalize the data in the range of -1 to 1. Data is in 3 channels and in range of 0-255 So I divide by a factor of 128 and subtract 1 from it to fit it in range of -1 to 1 for better accuracy.
- One hot encoded the labels for better accuracy

Implementation

Let’s start the implementation by defining the variables.

- Image Height 32
- Image Width 32
- Channels 3(RGB)
- Kernel Size used is 5X5
- Stride of 1
- Padding SAME
- Dropout Probability 0.5
- Initial I used depth of 32

Steps involved in implementation are: -

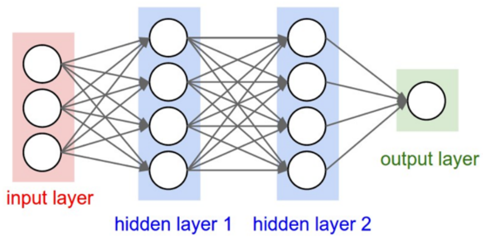
- Normalize and reshape the training and testing data as define in data preprocessing above
- One hot encoded the labels
- Created placeholders for the variable that will be input at the time Tensorflow starts computation

```
x = tf.placeholder(tf.float32, shape=[None, 32,32,3])
y_ = tf.placeholder(tf.float32, shape=[None, 10])
```

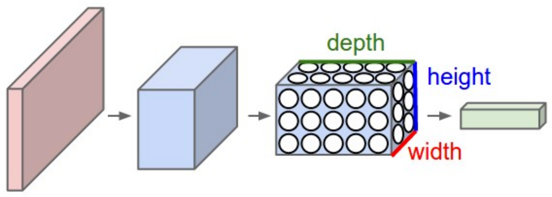
- Weight and biases are initialized. It is good to initialize weights with small amount of noise for symmetry breaking, and to prevent 0 gradients.
- Then input image is applied to **First Convolution Layer** which consist of two process
    1. Convolution- this will compute 32 features for each 5X5 kernel
    2. RELU is applied to make output non-linear
    3. max pooling. –it will apply 2*2 max pooling to the output of RELU
- Then output of First Convolution layer is applied to Second Convolution Layer which is similar to the First one except it will compute 32 features for each 5X5 kernel
- Output of Second Convolution Layer is applied to Densely Connected Layer.
- Densely Connected Layer is a fully connected layer with 1024 neurons. As image size is reduced we are using fully connected layer
- Output of Densely connected layer is passed to Readout (softmax) layer
- I randomize the image set and pass the data set in batches at each training step. if all images in training set are fed to the system I repeat the process till the training completes
- The reason I chose ConvNets over RegularNets is
    1. *Regular Neural Nets don’t scale well to full images*. In SVHN, images are only

of size 32x32x3 (32 wide, 32 height, 3 color channels), so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This amount still seems manageable, but clearly this fully-connected structure does not scale to larger images. For example, an image of more respectible size, e.g. 200x200x3, would lead to neurons that have 200*200*3 = 120,000 weights. Moreover, we would almost certainly want to have several such neurons, so the parameters would add up quickly! Clearly, this full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

2. *3D volumes of neurons*. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular, unlike a regular Neural Network, the layers of a ConvNet have neurons arranged in 3 dimensions: **width, height, depth**. (Note that the word *depth* here refers to the third dimension of an activation volume, not to the depth of a full Neural Network, which can refer to the total number of layers in a network.) For example, the input images in SVHN are an input volume of activations, and the volume has dimensions 32x32x3 (width, height, depth respectively). The neurons in a layer will only be connected to a small region of the layer before it, instead of all of the neurons in a fully-connected manner. Moreover, the final output layer would for SVHN have dimensions 1x1x10, because by the end of the ConvNet architecture we will reduce the full image into a single vector of class scores, arranged along the depth dimension. Here is a visualization:



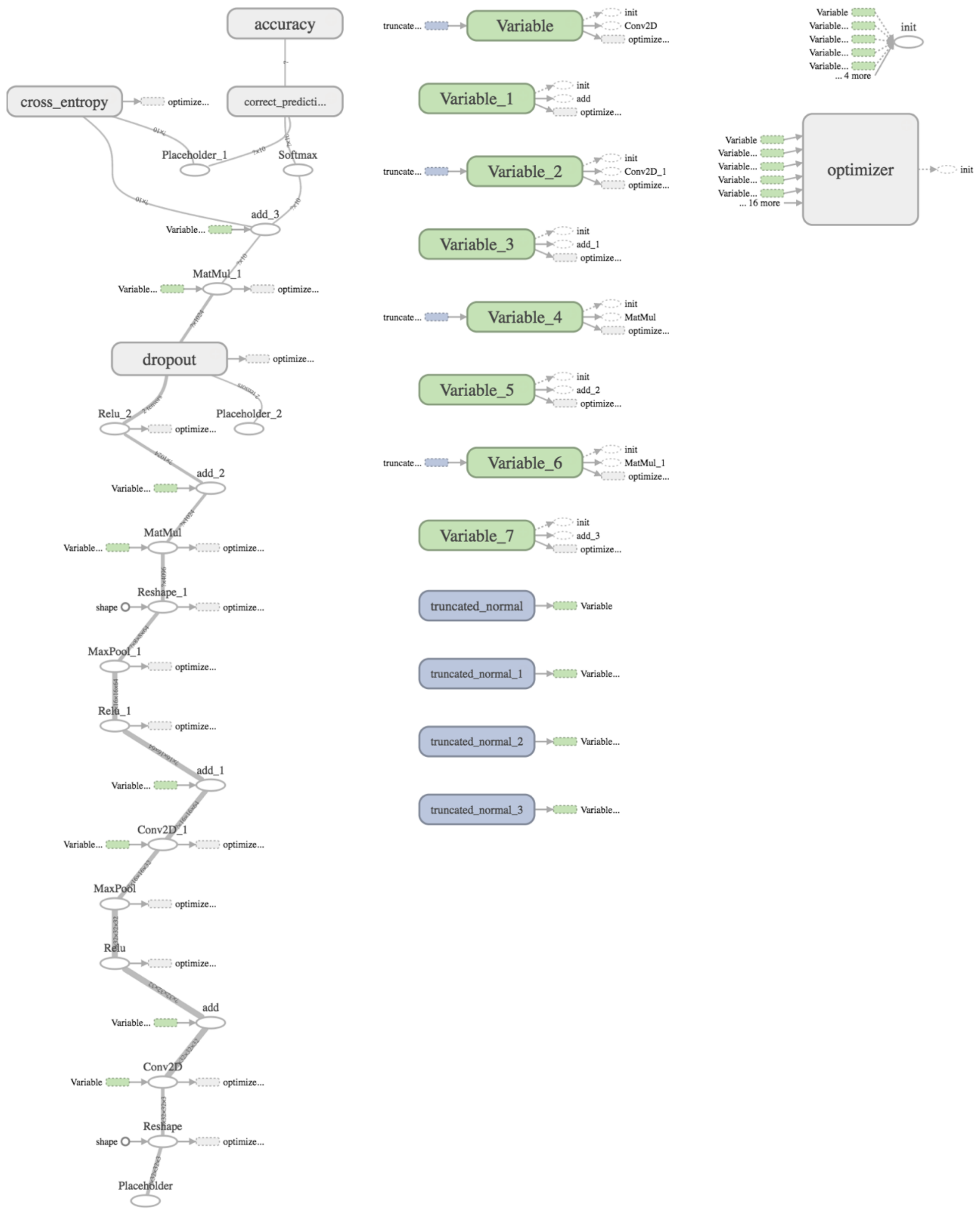
Regular neural network



Convolution network

- The complication that I faced while working on this project were: -
    1. Initially I was getting Nan after some epochs. Then with a little bit of research I found it’s good to normalize image data and keep the learning rate low.
    2. It was taking a lot of time while using normal gradient descent then I switch to stochastic gradient descent which decrease the training time and doesn’t affect the accuracy much.
    3. I was a new to Tensorflow so in beginning it’s a bit tough for me to understand some things but after some time I got the hang of it.

- A brief diagram of my model is shown below: -



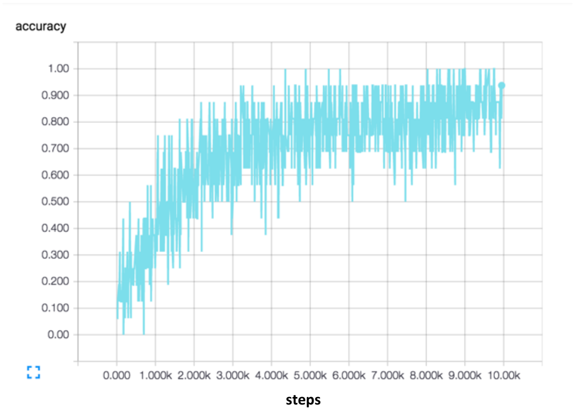
Refinements

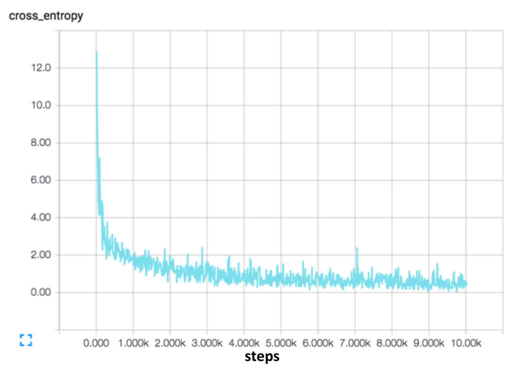
There are some parameters that can be played around to improve the accuracy of the model

1. **Batch Size: -** I used stochasting **gradient** descent instead of gradient descent to make training faster as I am using a machine with only cpu. Stochasting gradient descent make convergence faster. I have added results with different batch sizes
2. **Kernel Size:** - We have different kernel size option 3x3 or 5x5 I have tried both but got a better result with 5X5. Smaller kernel size means there are low level feature and they don't affect neighboring pixels, while bigger kernel size means there is a correlation between features of neighboring pixels.
3. **Epochs:** - I used my model for different epochs 10000 and 20000.
4. **Learning rate:** - I kept my learning rate 1e-4. When I kept my learning rate high I was getting nan and machine was unable to train.
5. **Dropout:** Dropout prevents overfitting. Though the datasize is not big so it only increases accuracy slightly. I kept my dropout 0.**5.**

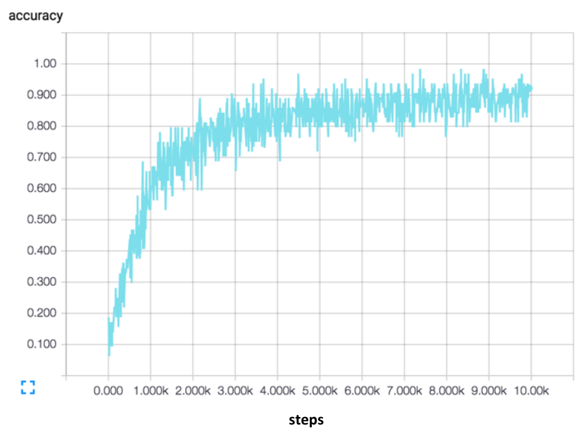
At epochs 10000 and batch size of 16

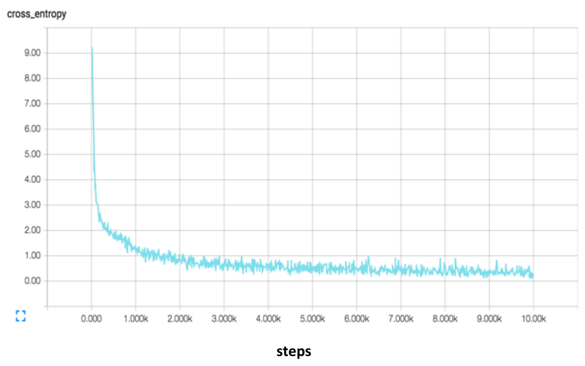
Accuracy on test set: 85.66%



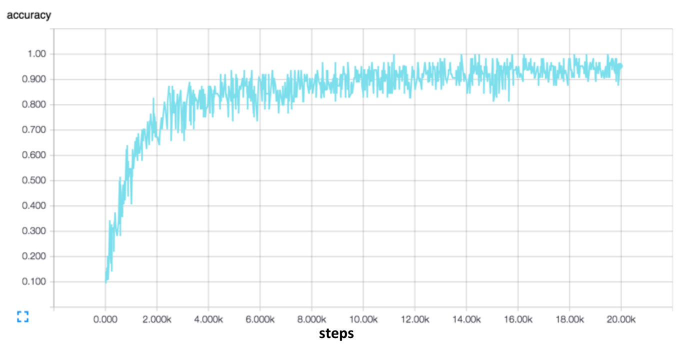


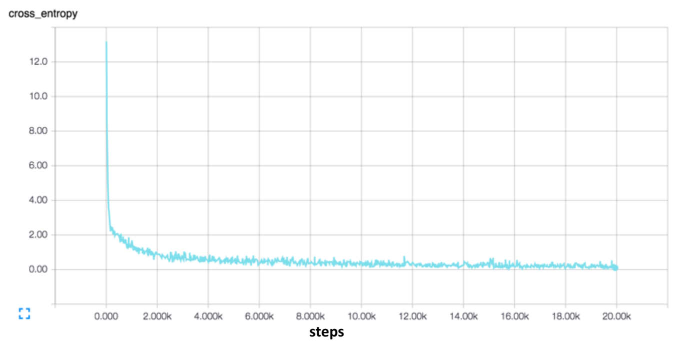
At epochs 10000 and batch size 64
Accuracy at Test set: 88.43%





At epochs 20000 and batch size 64
Accuracy at Test set: 89.43%





Results

Model Evaluation and Validation

Machine used: Dual core i5 and 8 gb Ram

Test Accuracy: ~90%

With parameters **epoch** 20000, **batch size** 64, **learning rate** 1e-4, **kernel** 5X5, **stride** 1, **zero padding**, **dropout probability** 0.5, **POOL** 2X2

With the given parameters I am able to get a satisfactory accuracy.

Justifications

As per my benchmark I was expecting accuracy of more than 85% in every scenario.

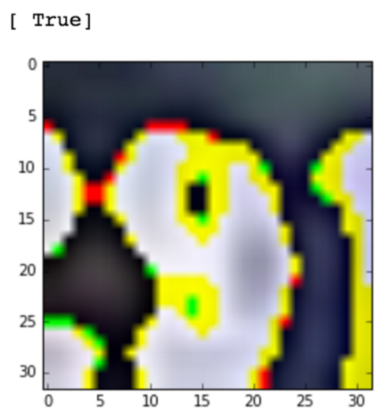
I got minimum accuracy of 85.66 at 10,000 epochs and 16 batch size which is not bad but got improved by lot by increasing epochs and batch size.

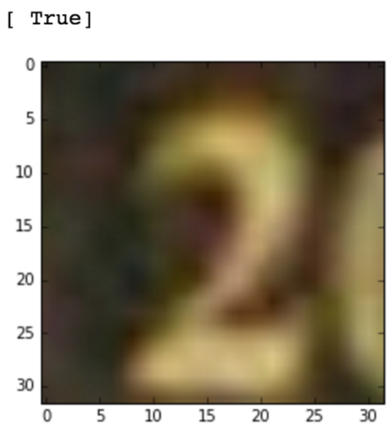
At 20000 epochs and 64 batch size I got an accuracy of ~90% which was great as per my benchmark.

Conclusion

Free form visualization

I have already added some images from different testData that are predicted true





Reflection

This process used for this project can be summarized using the following steps:

1. An initial problem and relevant dataset was found.
2. The dataset was downloaded and preprocessed as per convenience.
3. A benchmark for the problem was set.
4. A classifier was decided and implemented using Tensorflow.
5. The model was trained multiple times with different parameters until a satisfactory accuracy was achieved.

The problems I got in the project was majorly due to my little knowledge of Tensorflow but after some time I got the hang of it.

Thus the problem of identifying images is solved by the model.

This is what I conclude and learned from this project

1) Stochoistic gradient decent give comparable accuracy as batch one and decreases training time by lot
2) More the hidden layers, more training time for the model, but we can make our model deeper instead of wider for better resluts
3) Convolutions are the models to choose when correlation data is there like images, audio clips etc.

Improvements

The possible improvements that can increase the performance of the model are:

1. A better system with a GPU that can train faster and better. This will also enable me to fine tune my parameters more by reducing training time.
2. A bigger and better dataset. Right now images are of low resolution even harder for humans to correctly identify.