Machine Learning Engineer Nanodegree

Capstone Project

Vidit Shah
September 8,2017

I. Definition

- The following is project report for the Udacity Machine Learning Nanodegree Capstone Project
- The Project is based on the classification of images.
- The following project comes under the domain of Image Processing and analysis.
- Previously, one need to hardcode all the features of an image to classify them but as hardware advancement took place Implementation of Convolutional Neural Network(CNN) became more approachable and trustworthy way for the image classification purpose.

Project Overview and Statement

- In this project we will classify Flags(images) of India and United States of America(USA).
- We will train a network in which the CNN will learn how to classify the images itself provide new Image is fed into it.

- This is a binary image classification problem as we need to classify an image which will have one of the 2 classes as output.

Metrics

- I Have used Mean squared error as a metrics(Cross entropy)
- A loss function can be defined in many different ways but a common one is MSE (mean squared error), which is ½ times (actual - predicted) squared.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

II. Analysis

Data Exploration & Visualization.

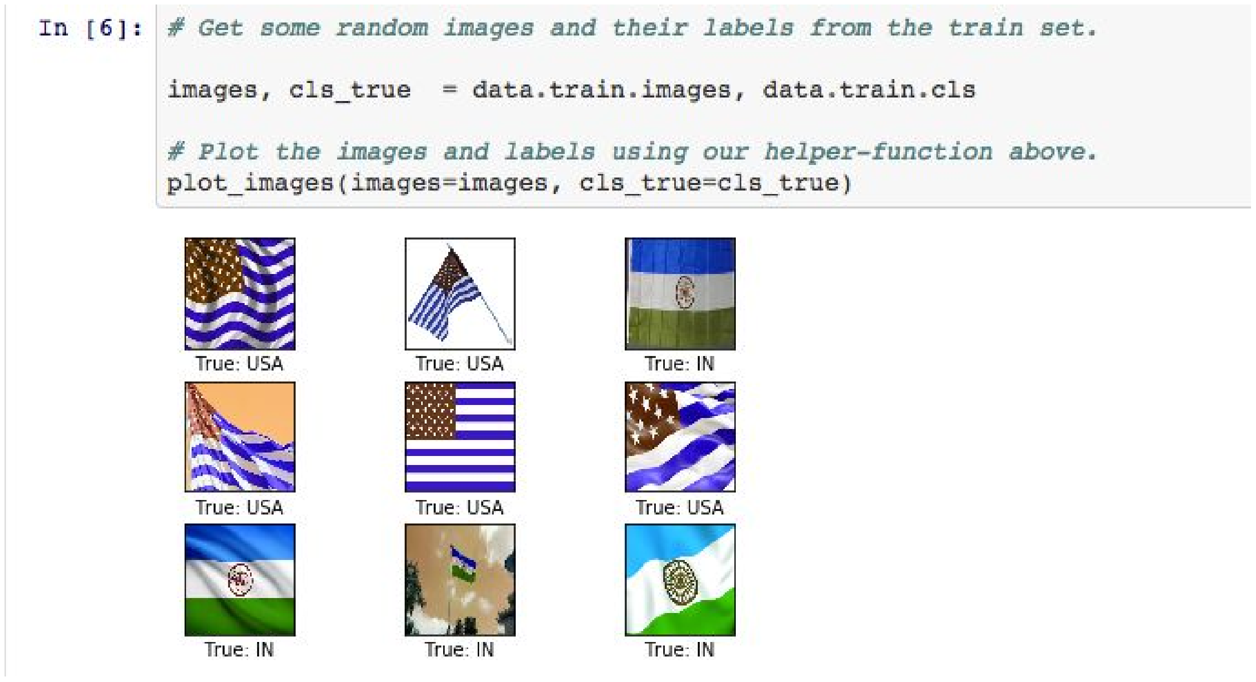
The Data consists of flag images of India and USA.
Images in dataset :

India-Class(IN)

USA -class(USA)

We divide the data into training and testing phase in order our network and learn and optimize:

Some plotted training data with its classes from the jupyter notebook.(Screenshot):



```
In [6]: # Get some random images and their labels from the train set.

        images, cls_true  = data.train.images, data.train.cls

        # Plot the images and labels using our helper-function above.
        plot_images(images=images, cls_true=cls_true)
```

Class for flags of India : IN
Class for flags of USA : USA

Algorithms and Techniques

- Algorithm used is Convolutional Neural Networks.
- Imagine one holding a torch and going through image row by row.
- Consider the output from the torch as patch which moving over the image from left to right and shifting by certain amount of space.
- Each patch stores some data and moves to the next patch .This happen till the patch has received from the top left of the image to bottom right of the image.
- The features which the patch gathers are stored in a stack over the other(Increasing the depth)
- One can run how many patches he/she wishes to run on the network.
- we describe our parameters and Hyperparameters.
- To get lay man idea difference between parameter and hyperparameter
- Hyper parameter affects performance of our model and often tuned(changed).For example- Batch size, epoch and learning rate. On the other hand Parameters are fixed image sizes and few constants we need.
- I have described a separate python scripts for handling the input data,we set the input size of image (Height and width) to 55 we resize every incoming image to size specified with help of openCV library.

Benchmark

We compare SVM and CNN. (hypothesis)

1. CNNs are designed to work with image data, while SVM is a more generic classifier;
2. CNNs extract features while SVM simply maps its input to some high dimensional space where (hopefully) the differences between the classes can be revealed;
3. Similar to 2., CNNs are deep architectures while SVMs are shallow;

4. Learning objectives are different: SVMs look to maximize the margin, while CNNs are not

Methodology

Data Preprocessing

Only Data preprocessing we require is resizing images which is described above.

Implementation(Building the Model)

Describe Weights and Biases for our model

```
def new_weights(shape):
        return tf.Variable(tf.truncated_normal(shape, stddev=0.05))
```

Above, is an example of describing weights. Shape takes 4 dimensional tensor having values of following parameters

```
shape = [filter_size, filter_size, num_input_channels, num_filters]
```

I have described bias similarly.

```
def new_biases(length):
        return tf.Variable(tf.constant(0.05, shape=[length]))
```

Now we describe our main Convolutional Layer where all the calculation takes place.

```
layer = tf.nn.conv2d(input=input,filter=weights,strides=[1, 1, 1, 1], padding='SAME')
```

tf.nn.conv2d takes input image,weights,number of pixel it needs to move(stride) and padding as parameter and handles all the complex calculation behind it.

Later Max pooling comes in the picture

tf.nn.max_pool handles all the operation for max pooling.

After consecutive convolutional ,pooling and activation we need to flatten our dimension from 4d to 2d for prediction purposes.

Flattening Layer

num_features = layer_shape[1:4].num_elements() layer_flat = tf.reshape(layer, [-1, num_features])

With help of tf.reshape we reshape our image to desired dimension and forward it to fully connected layer

FULLY CONNECTED LAYER

The output from Fully Connected Layer is fed into softmax function and predictions is made with help of tf.argmax() function and thus, now we need to Calculate

Error

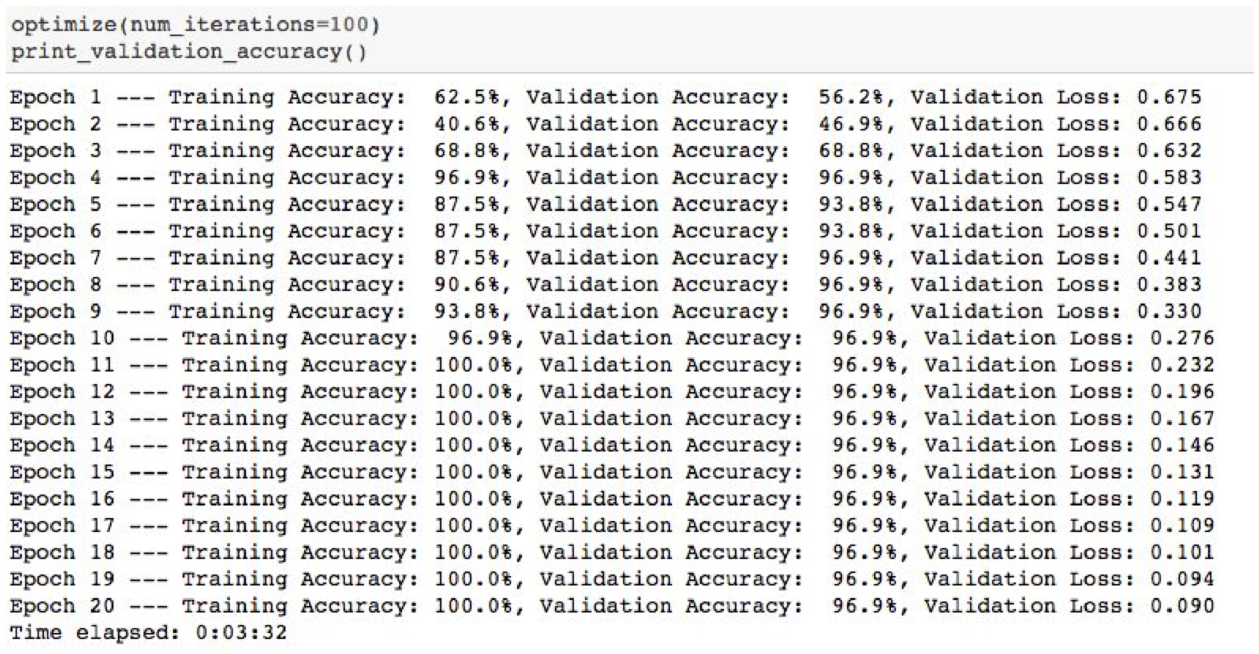
Error Is Calculated With help of Cross Entropy and Gradient Descent Optimizer taking in consideration of Learning Rate between Predicted output and True output.

IV. Results

Model Evaluation and Validation

We can Check how our model is performing by iterating over the dataset with a fixed batch size. After each epoch our goal is to minimize the loss and increase the accuracy our model

I have attached the screenshot of my model for clear and statistical understanding of the terms.



```
optimize(num_iterations=100)
print_validation_accuracy()
```

```
Epoch 1 --- Training Accuracy:  62.5%, Validation Accuracy:  56.2%, Validation Loss: 0.675
Epoch 2 --- Training Accuracy:  40.6%, Validation Accuracy:  46.9%, Validation Loss: 0.666
Epoch 3 --- Training Accuracy:  68.8%, Validation Accuracy:  68.8%, Validation Loss: 0.632
Epoch 4 --- Training Accuracy:  96.9%, Validation Accuracy:  96.9%, Validation Loss: 0.583
Epoch 5 --- Training Accuracy:  87.5%, Validation Accuracy:  93.8%, Validation Loss: 0.547
Epoch 6 --- Training Accuracy:  87.5%, Validation Accuracy:  93.8%, Validation Loss: 0.501
Epoch 7 --- Training Accuracy:  87.5%, Validation Accuracy:  96.9%, Validation Loss: 0.441
Epoch 8 --- Training Accuracy:  90.6%, Validation Accuracy:  96.9%, Validation Loss: 0.383
Epoch 9 --- Training Accuracy:  93.8%, Validation Accuracy:  96.9%, Validation Loss: 0.330
Epoch 10 --- Training Accuracy:  96.9%, Validation Accuracy:  96.9%, Validation Loss: 0.276
Epoch 11 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.232
Epoch 12 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.196
Epoch 13 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.167
Epoch 14 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.146
Epoch 15 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.131
Epoch 16 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.119
Epoch 17 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.109
Epoch 18 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.101
Epoch 19 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.094
Epoch 20 --- Training Accuracy: 100.0%, Validation Accuracy:  96.9%, Validation Loss: 0.090
Time elapsed: 0:03:32
```

Thus,with such loss result we can conclude parameters are tuned well.

Justification

To Check whether the model train can predict well on new Data .
I added 2 images of Flags of India and USA and observed the predictions of the model.
Screenshots of approach provided.

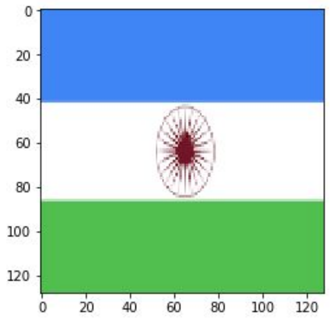
Image of USA and India Flags(Provided)

```
In [26]: plt.axis('off')

         test_us = cv2.imread('usflag.jpg')
         test_us = cv2.resize(test_us, (img_size, img_size), cv2.INTER_LINEAR) / 255

         preview_us = plt.imshow(test_us.reshape(img_size, img_size, num_channels))
```

```
In [27]: test_in = cv2.imread('IFLAG.png')
         test_in = cv2.resize(test_in, (img_size, img_size), cv2.INTER_LINEAR) / 255
         preview_in = plt.imshow(test_in.reshape(img_size, img_size, num_channels))
```



Result:

```
In [28]: def sample_prediction(test_im):

             feed_dict_test = {
                 x: test_im.reshape(1, img_size_flat),
                 y_true: np.array([[1, 0]])
             }

             test_pred = session.run(y_pred_cls, feed_dict=feed_dict_test)
             return classes[test_pred[0]]

         print("Predicted class for United States of America flag: {}".format(sample_prediction(test_us)))
         print("Predicted class for India Flag: {}".format(sample_prediction(test_in
                                                                             )))

         Predicted class for United States of America flag: USA
         Predicted class for India Flag: IN
```

As we can see model predicted correctly.

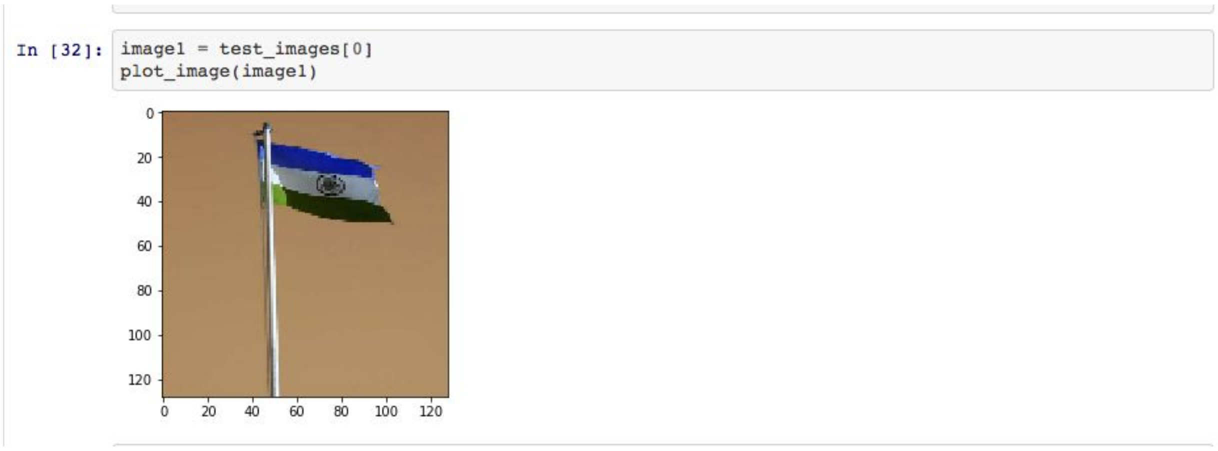
V. Conclusion

Free-Form Visualization

We will plot the weights ,how the model learned the parameters to classify the images.
This plotting makes us the behaviour how the model learns

For the following image we will see how each recorded its weight.

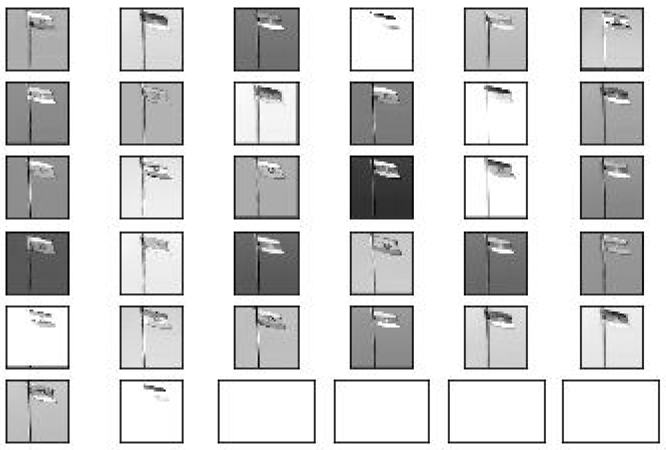
Image:



```
In [32]: image1 = test_images[0]
         plot_image(image1)
```

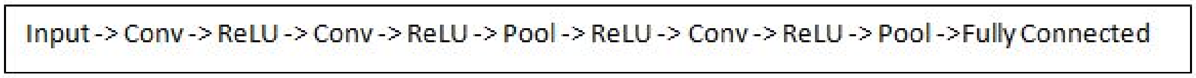
Weights:

```
In [35]: plot_conv_layer(layer=layer_conv1, image=image1)
```



Reflection

PipeLine for CNN is as follows:



Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool ->Fully Connected

The most interesting thing I like is the power of matrices which can lead to classify such complex tasks,like it's mind boggling and overwhelming to digest

One difficulty encountered was as I was creating my own I had to create my own batch function which can resize and can be fed into the network.
Like traditionally when one start to learn about CNN,one uses MNIST,CIFAR dataset which one can feed into network directly but in this I had to create my own function.I’m satisfied with my performance Model as with very small dataset it could learn really well

Improvement

The model Can be improved by providing more Training Data to it.

More the data more well model will learn :)