Computer Vision Final Project - Naren Suri, nsuri@iu.edu

Facial Expression Recognition

The goal of the project is to detect the human expressions given an image or video feed. This Kaggle data set challenge aims to detect the expressions of humans from 7 categories of the expressions. The data set is labeled with the expressions of the each face. All the face images are processed for only gray-scale images and centered with a size of 48 X 48. The data set is taken from the Kaggle competition.

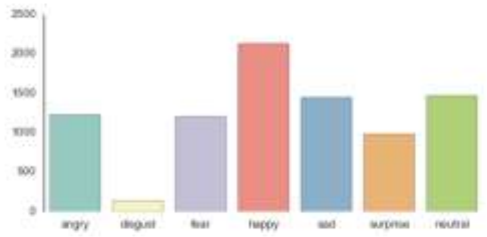
I have solved the problem in the following ways,

1. Created a Deep Neural network with just python by writing the both forward and backward propagations code manually. The algorithm is trained and tested on private test data. I have included the results below.
2. Also, Implemented the Deep Neural network in TensorFlow with softmax regression.
3. Finally, Implemented the Convolutional Neural network with Keras and Theano, as Keras give higher flexibility and short code in terms of writing neural networks.
4. I have analyzed the filters those are being generated by the CNN in the above model
5. Also, I have used the HAAR cascade features to detect human's faces real time through the web-cam or video or images.
6. The detected face through one of the means discussed above is then sent to the CNN model that is already trained above for prediction of the expression.

Motivation:

Human facial expressions can be categorized into 7 emotions which are sad, surprise, fear, anger, disgust, and neutral. These signals are not so easy to understand by machines, however machines might be more helpful at a restaurant as they can understand the expression besides the words it hear. There are many such applications where a machine having sense for human expressions would help them. Also, we may track the expressions of a thief or some anti-social element to take actions in advance. Though this problem is simple for human cognition, it's not really an easy task for the machines for various reasons. With the advent of Neural Networks, the process of parameter tuning with high computing power is helping humans to solve some problems and face expression recognition is one among them.

Data Set: As mentioned above the data set has facial expressions of humans and there are 7 expressions listed.



We have more samples for the happy. But the samples for the disgust are very limited. Since there is a class imbalance in the data, I have followed the two approaches. 1. Duplicate sampling – generate the duplicates for the class with less number of records. The ideal way is to generate the data samples from its distribution, since there is no information on the distribution parameters of the disgust, I have generated duplicate samples using the python built-in functions.

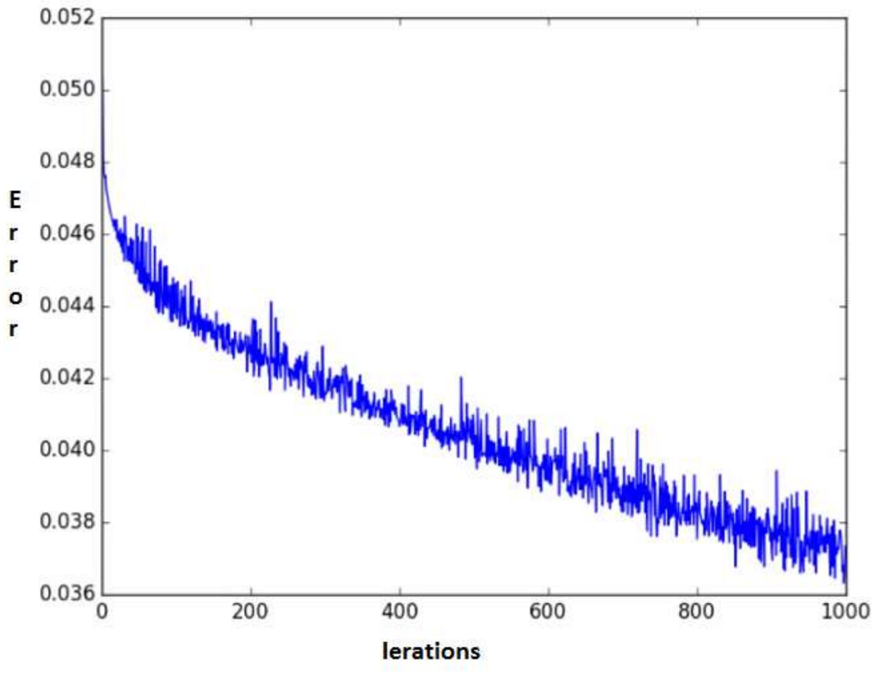
2. The second approach is, as the disgust is too less and it is similar to anger I have merged both the disgust and the anger.

Model:

1. Deep neural networks for Face Expression Recognition:
   The deep neural net try to learn the weights by itself. We have to specify the number of layers and the number of units in the each layer. A DNN keeps going forward over a forward propagation and calculates the error it made and try to correct them by adjusting weights. The DNN propagates the error back in the network that we every layer try to help reduce the error and this helps in reducing the overall error. This communication of forward and backward helps the network to minimize the error and converge with the weights.

   I have implemented the DNN on the face expression data: FaceRecogniton-DeepNeuralNet.py
   The cost function that I have calculated started reducing over the each iteration.

1 Cost Function over Each Iteration of DNN



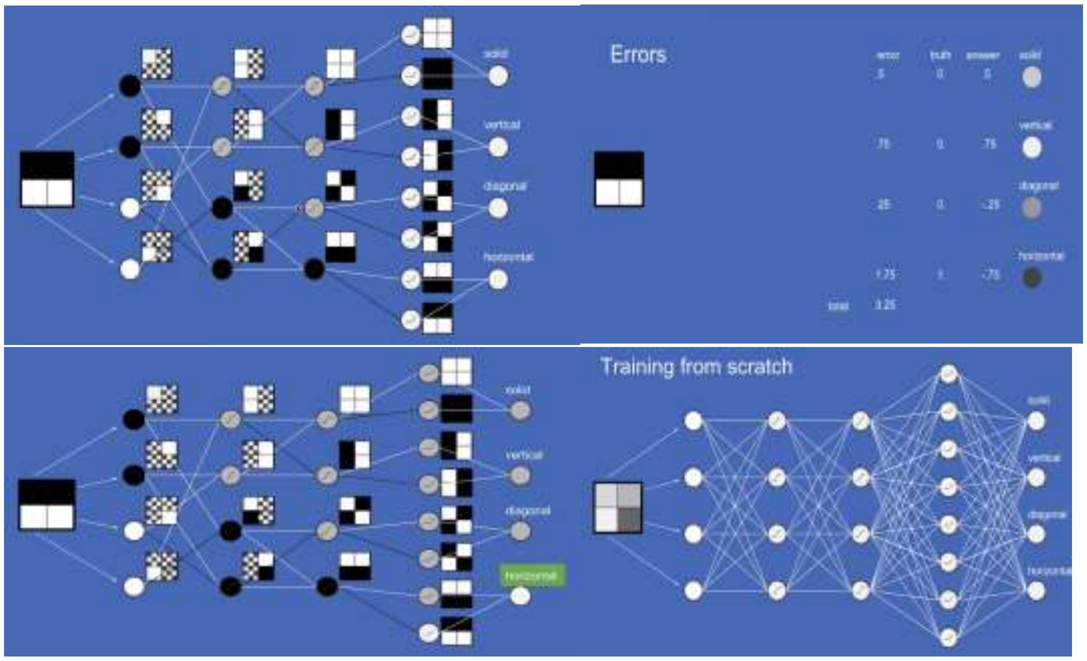
Iterations on X-Axis and the error rate on Y-Axis over Each Iteration of DNN

The cost of error is decreasing with the each iteration of the DNN while updating it weights. The code clearly explains the each step that's being done. There is around 64% accuracy in predicting the face expression correct. I have used the validation data sets and averaged the predicted results.

Deep Neural Net with the TensorFlow:
I have implemented the same functionality using the TensorFlow to avoid wiring the optimization functions of back ward propagation and the error reduction. The TensorFlow default optimization functions have done that for me. The code is attached as: FaceRecognition-DeepNeuralNet-TensorFlow.py

This deep network has the capability to scale compared to the above DNN. In the above DNN I have predefined the layers, where as in TensorFlow version I made it more scalable that is we can enter as many layers we want in between. I have run for the same configuration as above and got the same results.

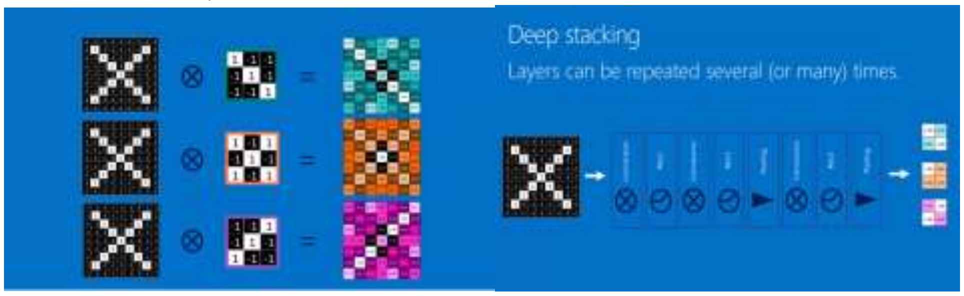
Few Simple illustrations of the DNN model:

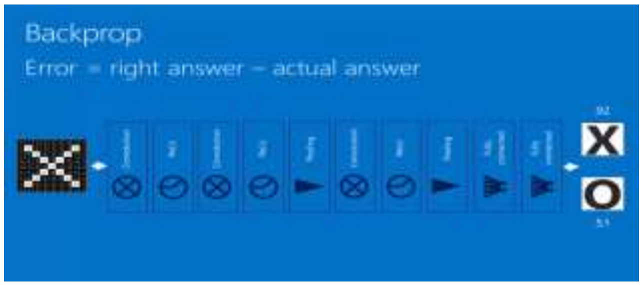


Slides credit: Brandon Rohrer. The last picture indicates the model can be even more complex.

Convolutional Neural Networks:

CNN can see the spatial relationship in the model while it get trained. This understanding for spatial information makes the CNN a good choice for the images. Because eyes and nose and every piece in image has some story and spatial relationship. The filters those get learned by the end try to learn this behavior of the problem. Like DNN the CNN too has both the forward and backward propagation.
But we use the convolution and pooling in the CNN. I have used the Theano and Keras to implement CNN and saved both the weights and model through Keras to use them for evaluations later. Please look into the attached HTML page for more work on CNN.
Here are some simple illustrations of a CNN. Credits : Brandon Rohrer

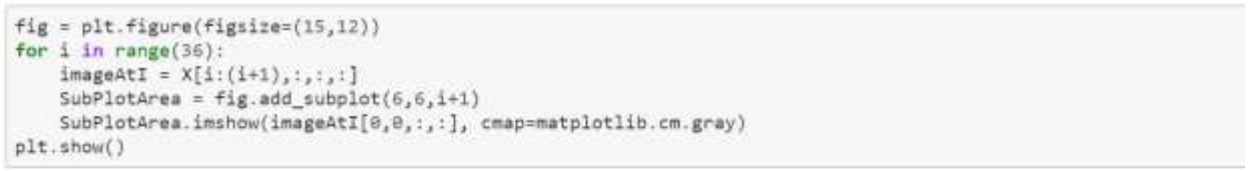




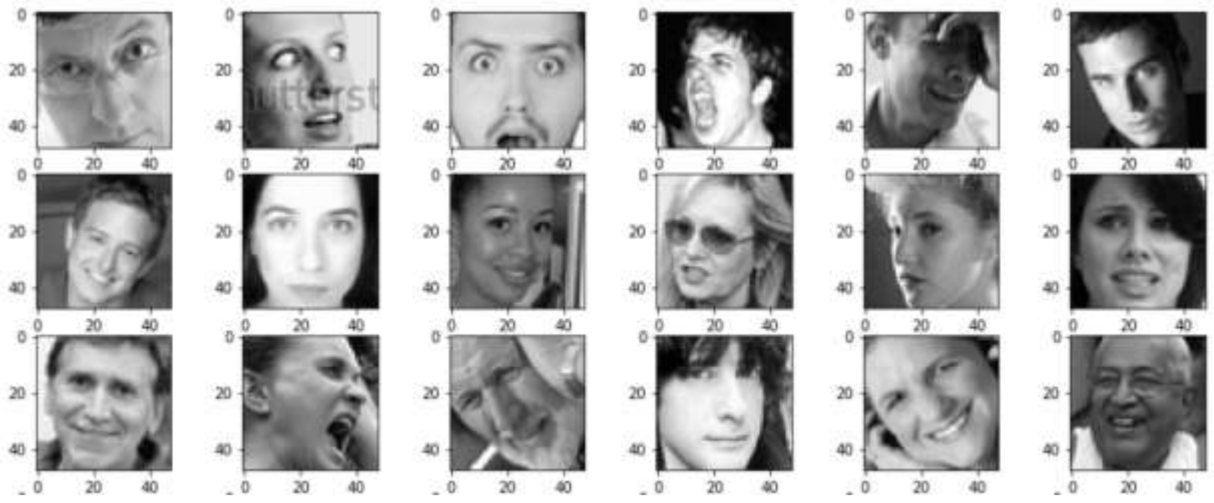
The File DataGeneration.py generates the training and testing data and saved as Numpy objects.
The Facerecognition-CNN-Keras.py helps in creating the model.

*** The details of the CNN and its results are discussed in the html file attached.

Few images from the data set:



```
fig = plt.figure(figsize=(15,12))
for i in range(36):
    imageAtI = X[i:(i+1),:,:,:]
    SubPlotArea = fig.add_subplot(6,6,i+1)
    SubPlotArea.imshow(imageAtI[0,0,:,:], cmap=matplotlib.cm.gray)
plt.show()
```

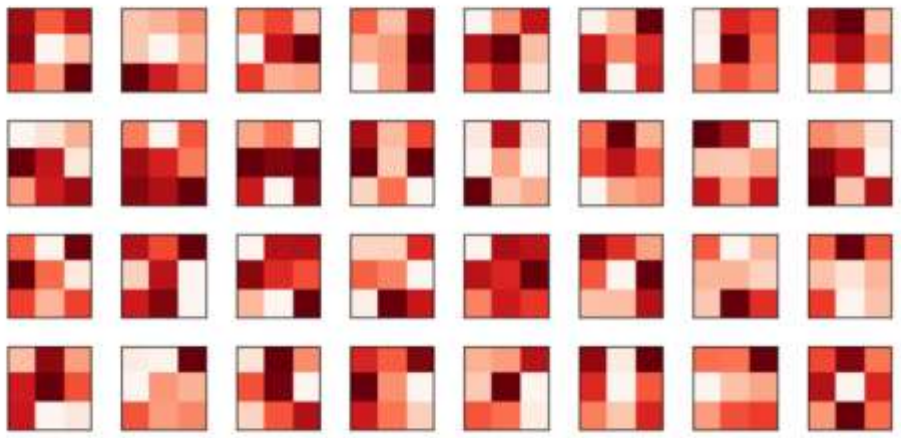


Filters at different layers of the trained model: - Layer 4

Exploring Fiters of the CNN modeled :

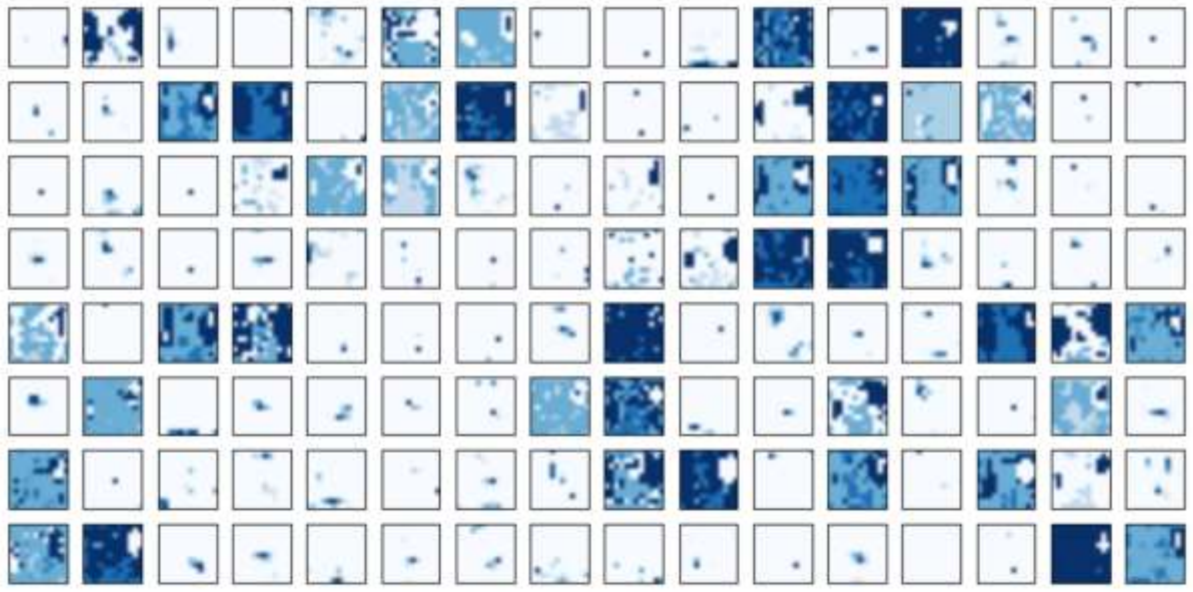
since we have loaded the model, now lets look into the filters

```
FilterValuesAtLayer = CNNmodelForFaceExpresions.layers[0].W.get_value()
fig = plt.figure(figsize=(8, 4))
for IthNeuronUnit in range(len(FilterValuesAtLayer)):
    SubPlotAreaProps = fig.add_subplot(4, 8, IthNeuronUnit+1)
    SubPlotAreaProps.matshow(FilterValuesAtLayer[IthNeuronUnit][0], cmap = matplotlib.cm.Reds)
    plt.xticks(np.array([]))
    plt.yticks(np.array([]))
    plt.tight_layout()
```



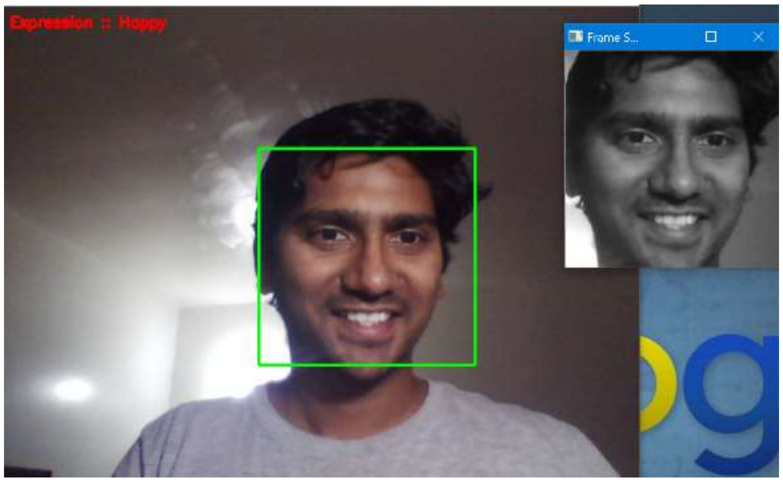
Transformation of the image in network at layer 11 for some sample example is:

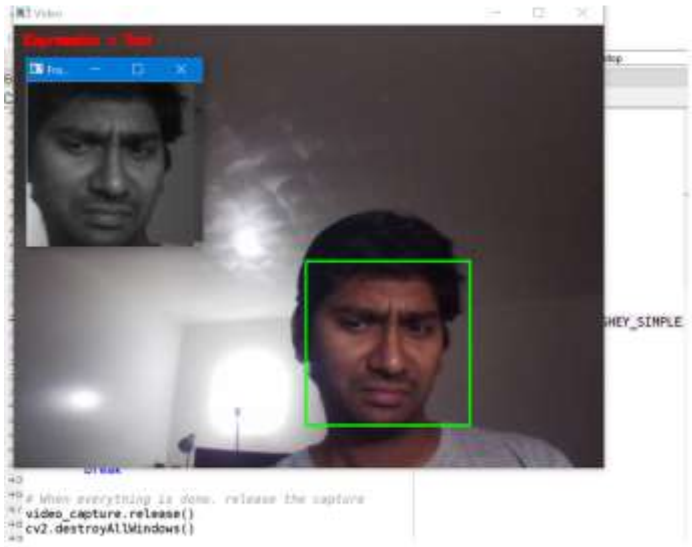
```
LearningAsReachedALayer(SampleFaceImage,11)
```



*** The CNN model has given an accuracy of 68% on the private test data.

Real time analysis of the expressions of the humans:
I have used the OpenCV for the webcam and the cascade of HAAR features. The webcam keeps loading images and the HAAR cascade helps to filter the image and decide whether it’s a face or not using fist few cascades. The HAAR helps in the detection of the face, I have selected that articular face and then resized it and sent to the model that I have already trained above. The model is able to detect the human faces and the expressions too.





```
        break

# When everything is done, release the capture
video_capture.release()
cv2.destroyAllWindows()
```

The prediction is correct and the model is able to detect things in real time.

The CNN is helping in detecting the expressions.