CSC321 Assignment 3

g2lujinn

Due: 10 pm, March 21th, 2016

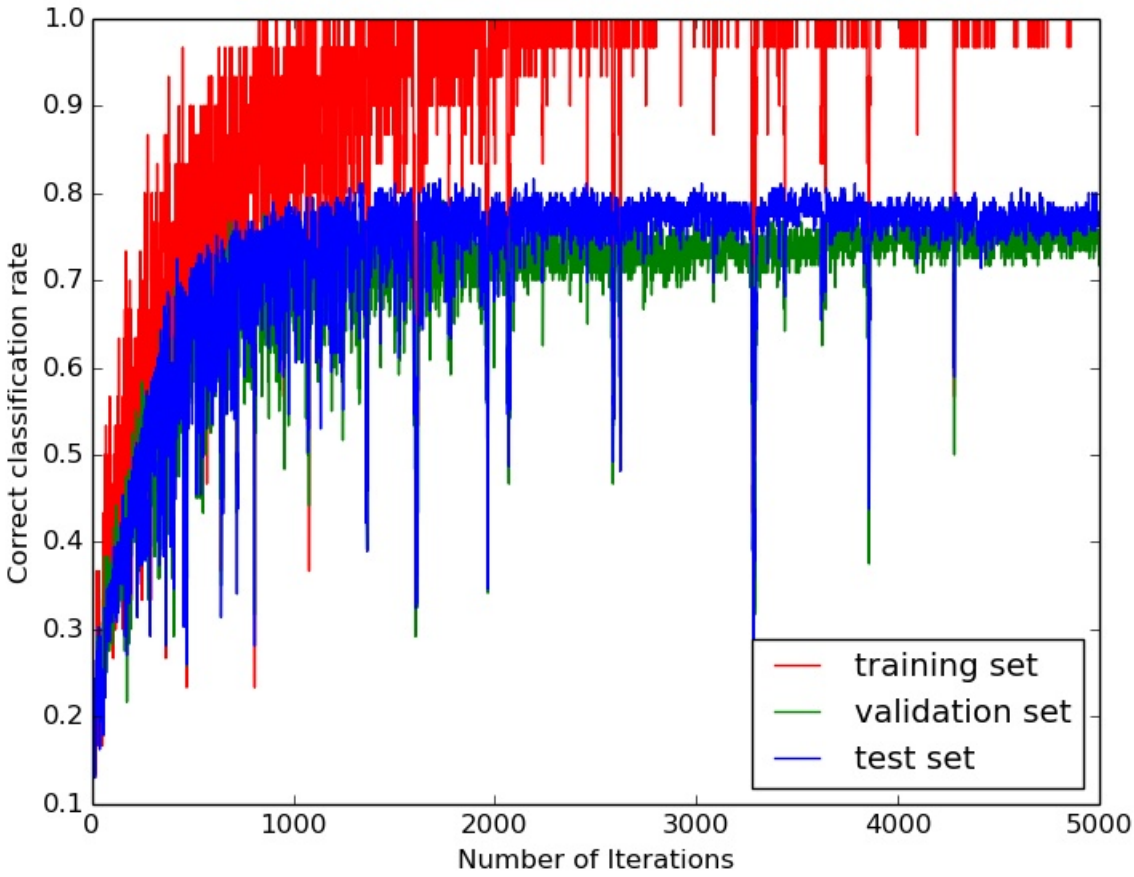
Part 1

I downloaded the faces and checked the hash value before crop them out, then I picked the first 70 faces for each actor/actress and stored them in a training set directory, similarly, I picked 20 faces for the validation set and the rest of them for test set. The faces are resized to 64x64.

Then I used a function to convert all of the faces into nx4096(where n is the number of faces in each set) image matrices, then I stored them in a dictionary and put the dictionary into a .mat file so it's easy to read.

The activation function I used was tanh for the first layer, and a linear activation function for the top layer. The network is a fully connected feed-forward network with one hidden layer of 300 units. The learning rate is 0.005 and the weights are initialized with given code in handout.

The final performance classification on the test set is 0.778378 for 1400 iterations, and below is the learning curve:


1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
Correct classification rate
0
1000
2000
3000
4000
5000
Number of Iterations
training set
validation set
test set

Part 2

Steps I took for part 2:

1. I first redownloaded the images and cropped them out to 227x227 faces.

2. Then like part 1, I resorted them into 3 directories: training set, validation set, test set with the same size as part 1.

3. Then I modified the code of AlexNet and put them into a loop to vectorize the data for each actor. The data is returned at conv4 layer and then converted back to numpy array. Thus to put all data inside the .mat file.

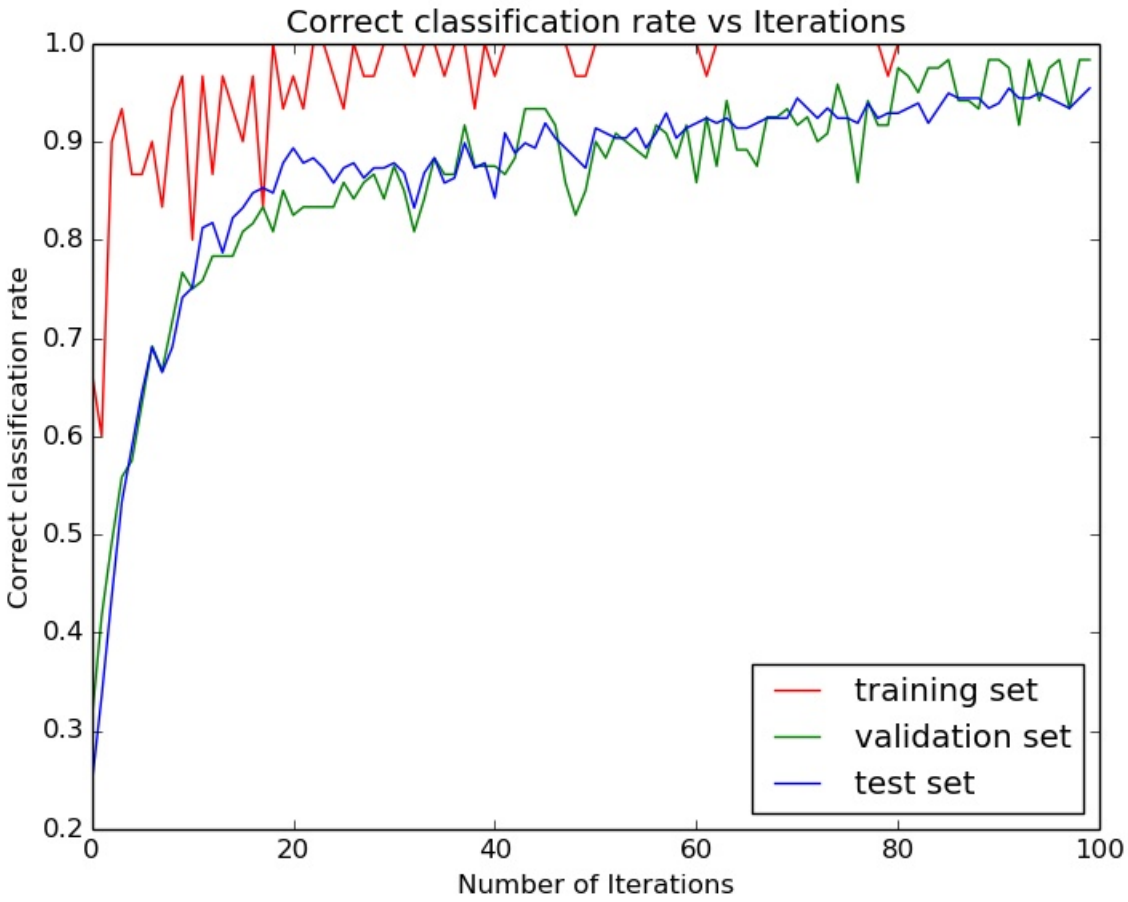
4. And finally done by modifying the dimensions and loops of the code from part 1 to get the performance rate.

To run the program, the "*create_M*" function will create the .mat file which contains training set, validation set, test set data for all actors, at the bottom of the code is the place where you can test the output names of actors by setting the actor's name as testing images, the specific test .mat file will be created for that specific actor, the output will print a list actor's names with each assigned with the probability of likelihood.

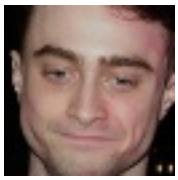
Note that I created a actor_classes.py which contains all the names of the actors/actresses similar to caffeclasses.py.

The final performance classification on the test set is 0.954315 for just 100 iterations which is much improved comparing to part 1.

Below is the learning curve:


Correct classification rate vs Iterations
Correct classification rate
Number of Iterations
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0
20
40
60
80
100
training set
validation set
test set

Below are sample outputs for Daniel Radcliffe after 100 iterations of learning:

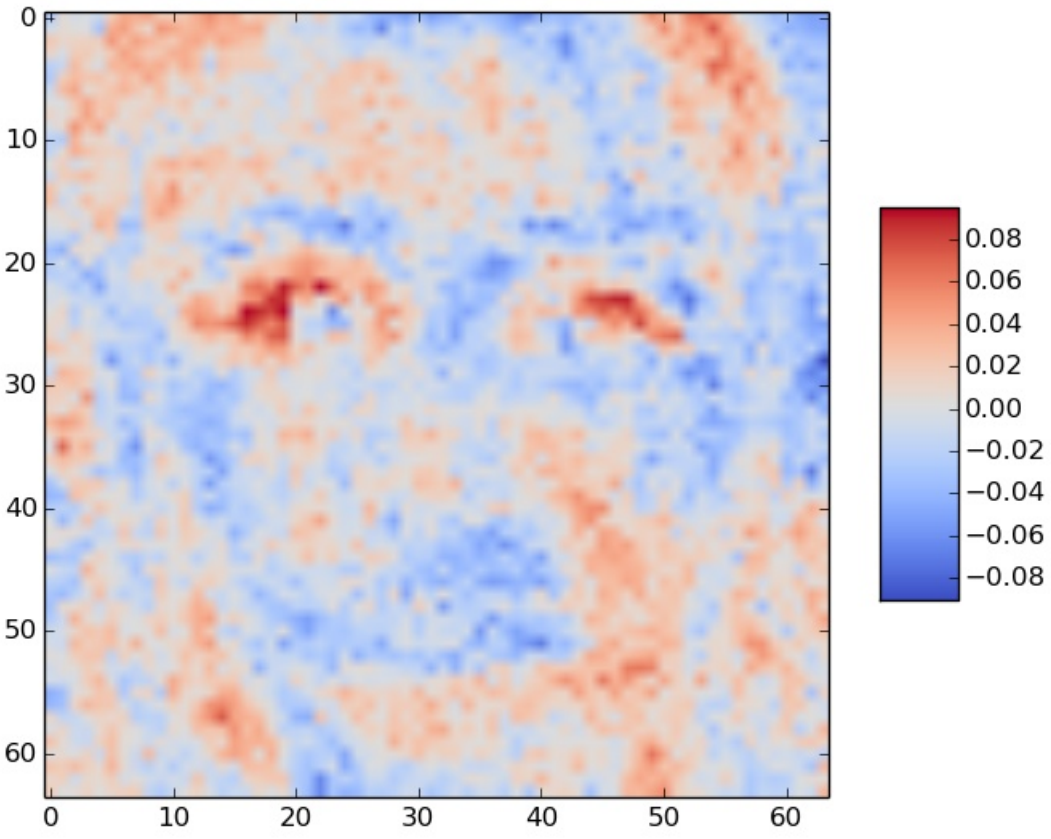


```
Daniel Radcliffe 0.949428
Angie Harmon 0.0401714
Michael Vartan 0.00363285
Peri Gilpin 0.00361683
Gerard Butler 0.00162882
Lorraine Bracco 0.00152235
```

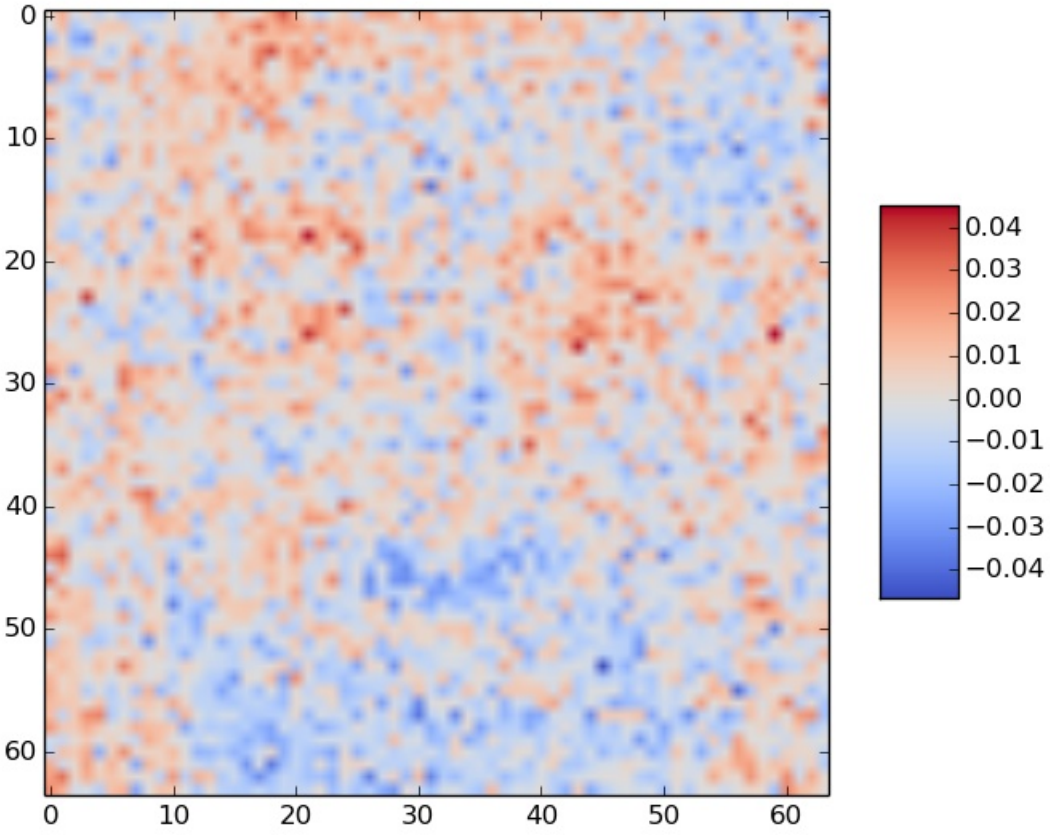
Part 3

To get the heatmaps, first run faces_tf.py twice to create 2 .pkl files for 300 and 800 hidden units respectively, then run part3.py to load the .pkl files and visualize the heatmaps. Note the lam was set to 0.00001.

The hidden feature with 300 hidden units are showed below:



The hidden feature with 800 hidden units are showed below:



They are interesting because they look like a face, the features of a face(eyes, mouth, nose etc.) have visually higher weights than the rest of the face. Note the learning rate is lowered to 0.001 for 800 hidden units, while the number of epochs is lowered to 3000.

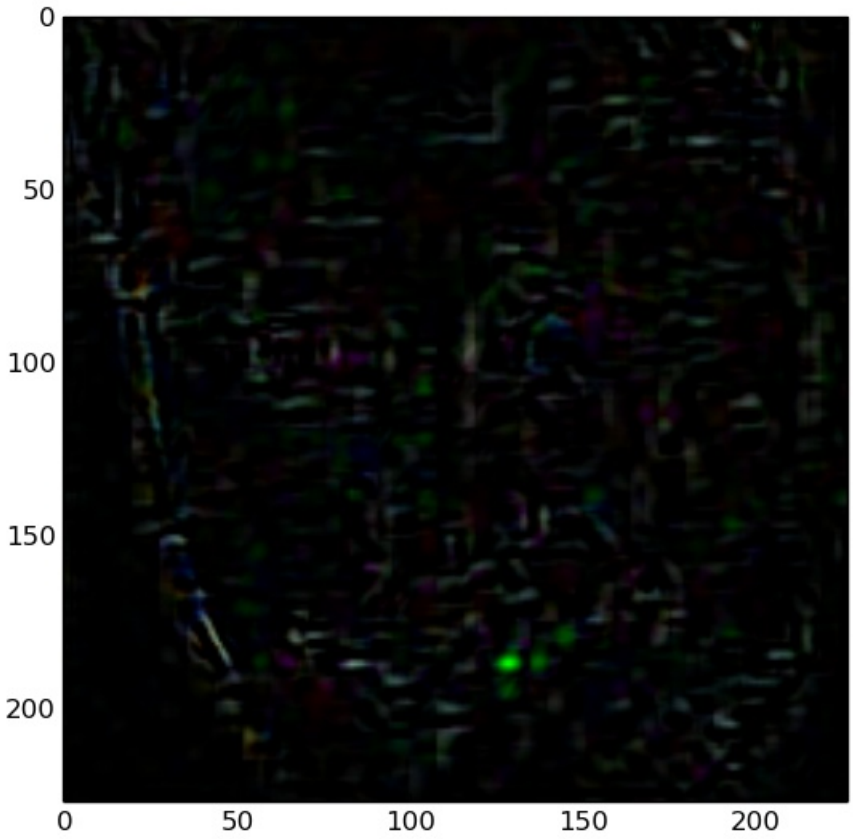
Part 4

I simply saved the trained weights and biases into a .pkl file named "*new_snapshot*99.*pkl*" from part 2, then I load it from part 4, in part 4, I did no training, only testing on the actors/actresses, for each actor/actress, I created a .mat file which contains the test images for that actor/actress, so the .mat file is loaded for test each time. Below are examples and their running outputs for Peri gilpin and Michael Vartan:

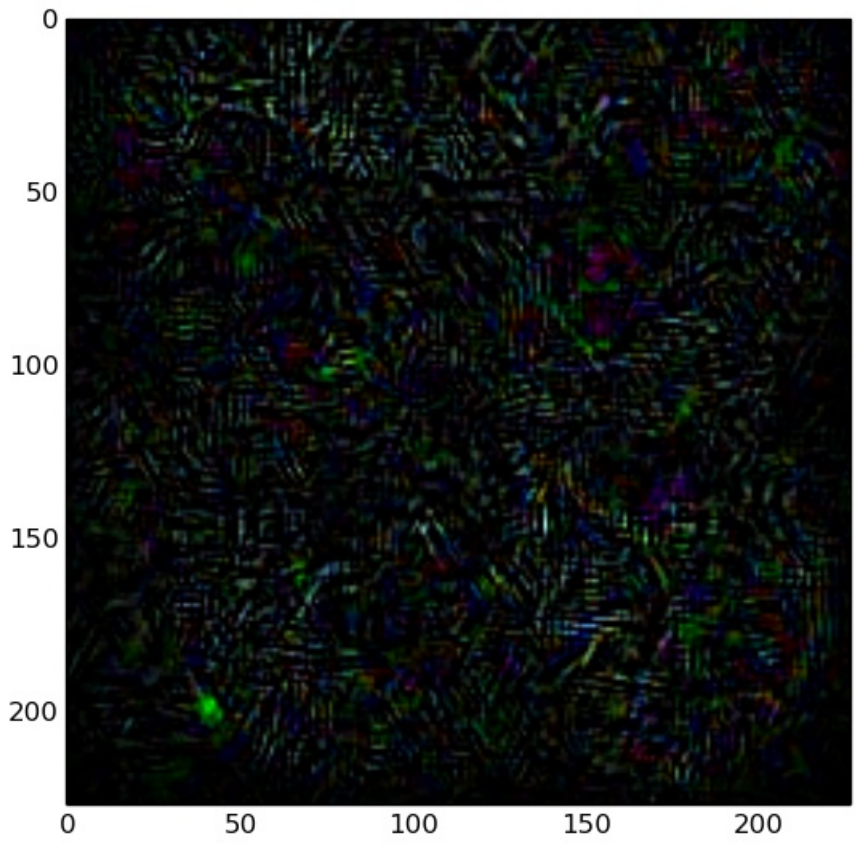
```
Peri Gilpin 0.96817
Angie Harmon 0.0274914
Lorraine Bracco 0.00326442
Daniel Radcliffe 0.000527523
Michael Vartan 0.000337669
Gerard Butler 0.000208584

Michael Vartan 0.888481
Gerard Butler 0.0362361
Lorraine Bracco 0.00392354
Peri Gilpin 0.0540983
Daniel Radcliffe 0.0107492
Angie Harmon 0.0065121

Gerard Butler 0.999996
Michael Vartan 3.76825e-06
Daniel Radcliffe 7.35815e-08
Lorraine Bracco 2.33333e-11img
Angie Harmon 9.6079e-12
Peri Gilpin 4.01875e-12
```

Part 5

I got two resulting images with and without dividing x by 255.0:



This is the gradient with dividing x by 255.0



This is the gradient without dividing x by 255.0

Code used is showed below:

```
# Load network from part 2
snapshot = cPickle.load(open("new_snapshot99.pkl"))
init_W0 = snapshot["W0"]
init_b0 = snapshot["b0"]
init_W1 = snapshot["W1"]
init_b1 = snapshot["b1"]

nhid = 300
W0 = tf.Variable(init_W0)
b0 = tf.Variable(init_b0)
```

```
W1 = tf.Variable(init_W1)
b1 = tf.Variable(init_b1)

layer1 = tf.nn.tanh(tf.matmul(conv4flatten, W0)+b0)
layer2 = tf.matmul(layer1, W1)+b1


y = tf.nn.softmax(layer2)
y_ = tf.placeholder(tf.float32, [None, 6])
gradients = tf.gradients(layer2, x)

init = tf.initialize_all_variables()
sess = tf.Session()
sess.run(init)
##########################################################################################################

#Output:
# Feed the session with test actor x and its targets
output = sess.run(y)
inds = argsort(output)[0,:]
for i in range(6):
    print class_names[inds[-1-i]], output[0, inds[-1-i]]

# part 5 starts here
gradient = sess.run(gradients)
gradient = np.asarray(gradient)
img = gradient[0][0].flatten()
img[img < 0] = 0
img *= 1/max(img)
plt.imshow(img.reshape((227,227, 3)))
show()
```