CSC411: Assignment $\#\sqrt{1}$

Due on Sunday, January 29, 2017

Naireen Hussain

January 28, 2018

Problem 1

Part 1: Describing the Data

The collection of data is quite poorly maintained as there were several URL links that were no longer valid. For some cases, the image had been removed, and replaced with another image, as shown below. These images were not removed. Afterwards, I cropped the images, but there were quite a few where the bounding box extended outside of the actual image, so there were only 1273 left for the 6 actors from the original 1397. The way the invalid bounds were handled were that those images were completely ignored. There were enough images that this was a viable option. Furthermore, while some bounding boxes were accurate in capturing the face, there were some that were not. Some were so inaccurate that one photo had two people, and the bounding box bounded the wrong face, as seen in Figure 1. Other times, part of the face would be cut off, like the chin or the forehead, and in other cases, the bounding box missed the face entirely. No poorly bounded data was removed. All images shown below are after they have been cropped by the bounding box, and before they have been re-sized and recolored.

Here are some that were not correctly bounded, or had other issues:

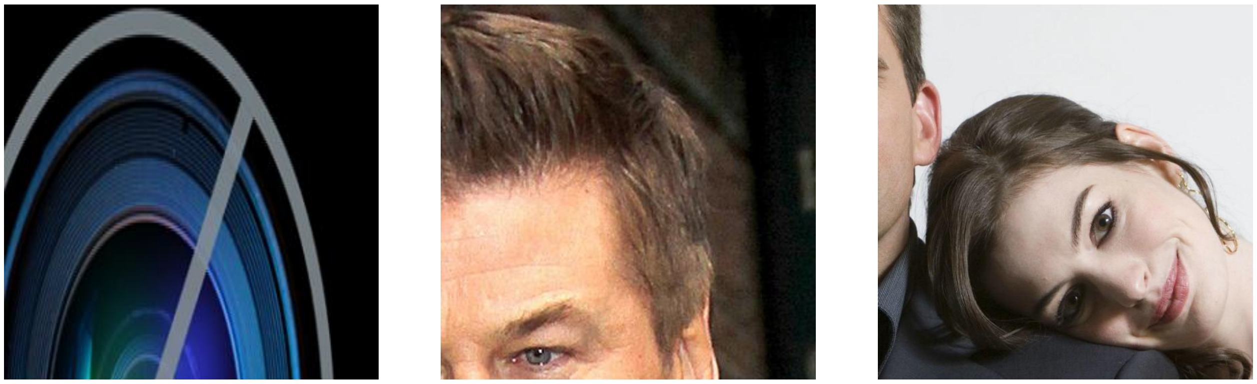


Figure 1: The first image to the left had its image replaced. The middle picture had an incorrect bounding box, so only part of the face was captured. The last one did have a good bounding box, but got the wrong face, as this was supposed to be Steve Carell.

Here are some that were bounded correctly:

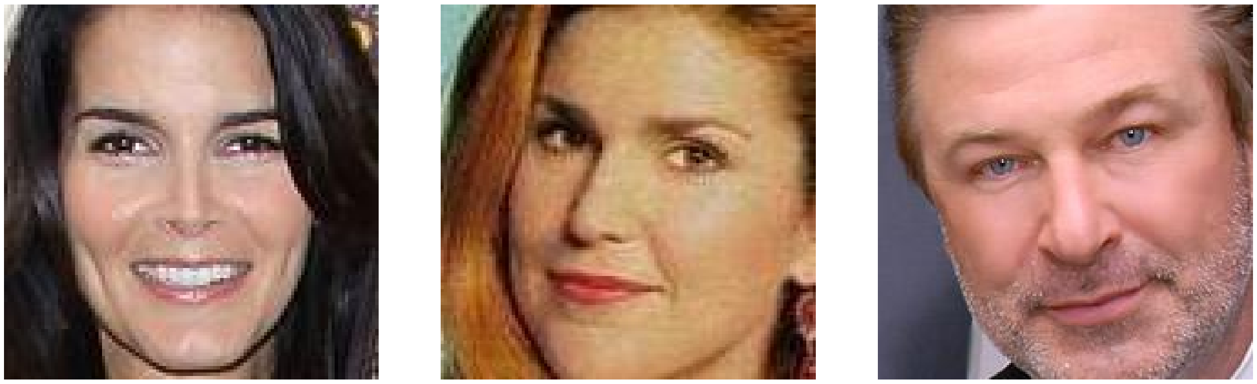


Figure 2: These images were bounded correctly. Even though they all depict a face, they aren't all alighed, since the actors a in a variety of different poses. These pictures also have a tiny bit of either the forehead or the chin cut off.

Many of these images don't have their faces aligned, since the actors weren't all in the same pose when these

photos were taken. In Figure 2, the middle actress has here face slightly turned away from the camera, and the right actor has his head slightly tilted.
Here are some pictures that haven't been cropped yet:

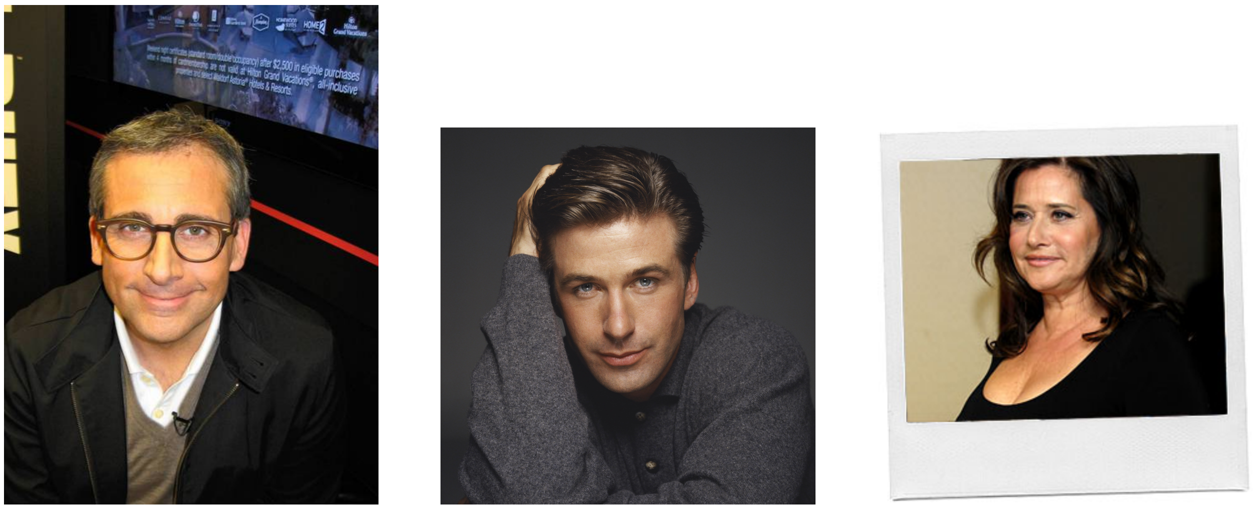


Figure 3: These are some images that haven't been bounded yet.

Problem 2

Selecting The data
Working with the cropped data, I collected the first 110 valid images, and designated that as the training set, and then selected the next 10 for validation, and then the next 10 made up the test set. This means that there were a total of 130 valid images per actor. I did not feel the need to shuffle the data, as it seemed like the listing of picture URLS were already random, so there would be no added value in reshuffling the data. Valid images are images that could be successfully opened to ensure that they were not corrupted. Images at this stage were already checked to make sure they were properly cropped, resized and recolored.
Here is code below with the appropriate error handling. This is in file sort_data2.py

```
import os
import matplotlib.image as mpimg
import numpy as np
from shutil import copyfile
from matplotlib import pyplot as plt

act = list(set([a.split("\t")[0].replace("\n", "")
     for a in open("subset_actors.txt").readlines()]))

files = os.listdir("processed/")

print(len(files))
for name in act:
    counter = 0
    for file_name in files:
        if name.split(" ")[1].lower() in file_name:
            #if the actor is in the file name of pic sort the images in approriate spot accordingly
            if counter > 130:
                breaks
            try:
                if counter <110:
                    copyfile("processed/" + file_name, "Train/" + file_name)
                elif counter<120:
                    copyfile("processed/" + file_name, "Validation/" + file_name)
                elif counter<130:
                    copyfile("processed/" + file_name, "Test/" + file_name)
                counter +=1
                print(counter)
            except Exception as e:
                print(e)
            #once while loop concludes,
            if counter > 130:
                break
```

Problem 3

The Model - Gradient Descent
I used a linear regression classifier model to classify the images. The images of Alec Baldwin were compared against those of Steve Carell. Baldwin was labelled 0, and Carell was labelled 1. There were 220 images in the training set, and then 20 in the validation set. The model got 214 (97.2%) correct on the training set, and 13 (65%) on the validation set. It makes sense that the model does well on the training set, since its predicting on images that were used to construct the model itself. These are results generated when Θ is initialized as a zero vector for the gradient descent.
Since the linear regression model returns the dot product of $\Theta^T X$, which is a continuous value, it needs to be mapped to a binary value of either 0 or 1 for binary classification. For any data point that had a score of less than 0.5, it was mapped to 0, and everything above 0.5 was mapped to 1.
Here is the code to generate the predictions for the validation set after the Θ vector has been determined through gradient descent.
The cost function minimized was

$$J(\Theta_0, ...\Theta_{1025}) = \sum_{i=1}^{m}(h_\Theta(x^{(i)}) - y^{(i)})^2 \tag{1}$$

The constants $\frac{1}{2m}$ was dropped as m is constant, so that missing factor is just scales the answer, but the direction of the gradient searched in gradient descent remains unchanged.

```
#predictions on the validation set,
preds = np.zeros_like(valid_y)
valid_data_ones = np.concatenate((np.ones((1, valid_data.shape[0])).T, valid_data),
     axis = 1)
preds = np.dot(valid_data_ones, theta)

#map preds to binary outputs
preds_binary= np.zeros_like(preds)
preds_binary[preds<0.5] = 0
preds_binary[preds>0.5] = 1
#compute how many the model got right
print len(np.where(valid_y ==preds_binary)[0])
```

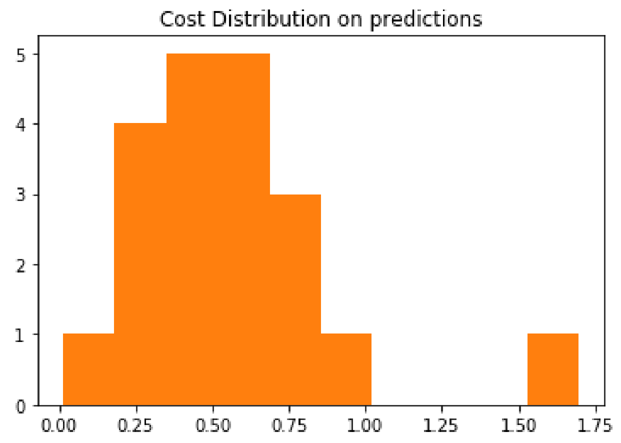
To get the model to work, some fine tuning was required. For example, the model would not work if the learning rate α was too large. Trial and error was used to find a value that worked, and then further tuning was done to find a value that would give a higher score. This was determined to be 1 x10^{-5}. Another value that I tuned was ϵ, which determines when the model stops. If the change in Θ is less than ϵ during one iteration, then gradient descent stops. The larger ϵ is, the quicker the model stops. This was also manually tuned to an optimal value of 1 x10^{-4}. Lastly, a common error I was running into was arithmetic overflow. This occurred when running gradient descent with the original pixel values. To solve this, I scaled down all pixel values by 256.
Here are the first 10 cost values for the training set:

```
[0.28289115  0.11183247  0.20492276 -0.00654187  0.10977967  0.13322678
 0.29099036  0.0232449  -0.00654187  0.28450416]
```

Here are the first 10 cost values for the training set:

```
[0.39899181 0.29958775 0.24561934 0.67736738 0.26899213 0.5821788
0.57735219 0.48045102 0.51039915 0.43317688]
```

Here are two distribution plots of the scores



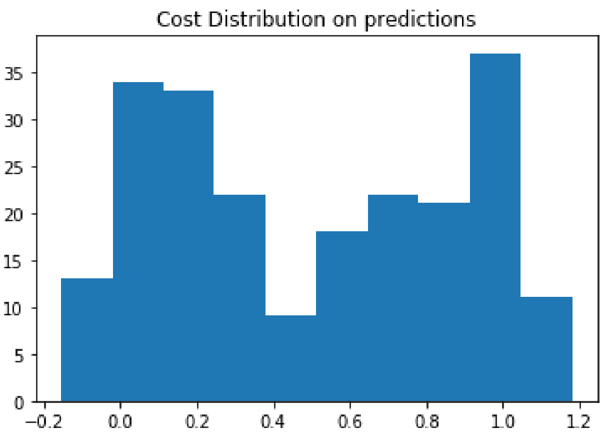
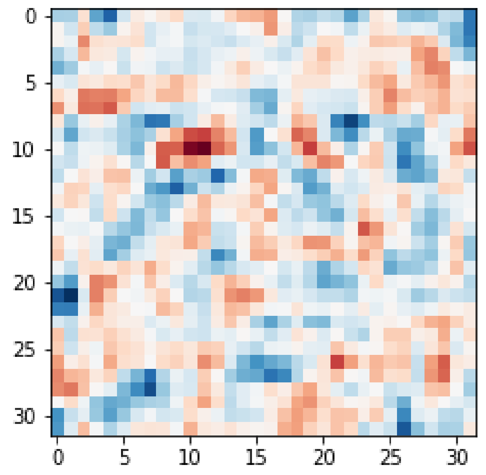


Figure 4: These are some images that haven't been bounded yet.

Problem 4

A

Representing Theta
The model J(Θ) can also be visualized. Here is how it looks trained on the full set of 220 images.



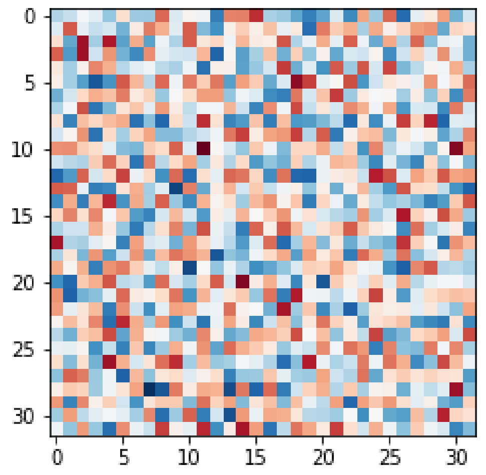


Figure 5: Both images are produced by training on the entire training data set of 220. The left image is produced by initializing Θ as all zeros, and the second one is done with Θ initialized with np.random.random, with the random seed set to 0.

B

Different images can be produced by stopping the gradient descent earlier.

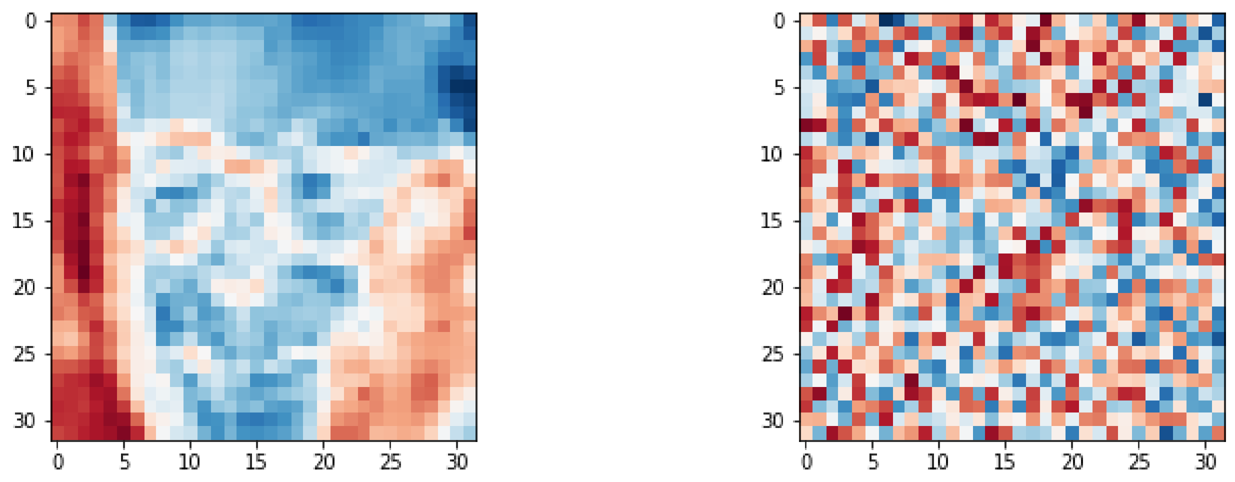


Figure 6: Both images are produced by training on the entire training data set of 220. The left image is produced by initializing Θ as all zeros, and the second one is done with Θ initialized with np.random.random, with the random seed set to 0.

Problem 5

Gender Classification
Since there were 110 images collected for each actor, I has 330 female and male pictures for training, and then 30 male and 30 female pictures for validation amd then another set of 30 male and 30 female pictures for as testing. However, for testing, the 60 faces were used were of actors that were not included in the training or validation stage. The male actors used were Daniel Radcliffe, Gerard Butler and Micheal Vartan, and the female actors were Kristen Chenoweth, America Ferrera, and Fran Drescher. The scores noticeably decrease as the training size increases. This is because the model is beginning to learn the noise in the data, which should not translate well to the validation data set. The score does not vary on the test set or the validation set. The model performs slightly better on the test set than the validation set. Sizes of 110, 220, 330, 440, 550 and 660 were used for the training set to determine how the performance would vary.

Problem 6

Multi-Classification with One Hot Key Encoding

A

$$J(\Theta) = \sum_{i}^{k} (\sum_{j}^{n} (\Theta_{ij}^{T} x^{(i)} - y^{i})_{j}^{2}) \tag{2}$$

$$\frac{\partial J}{\partial \Theta_{pq}} \begin{cases} 2x^{i}(\Theta_{pq}^{T} x^{(i)} - y^{i})_{j}, i = p, j = q \\ 0, i \neq p, j \neq q \end{cases}$$

B

$$\frac{\partial J}{\partial \Theta_{10}} = 2x^{1}(\Theta_{10}^{T} x^{(1)} - y^{(1)})_{0} \tag{3}$$

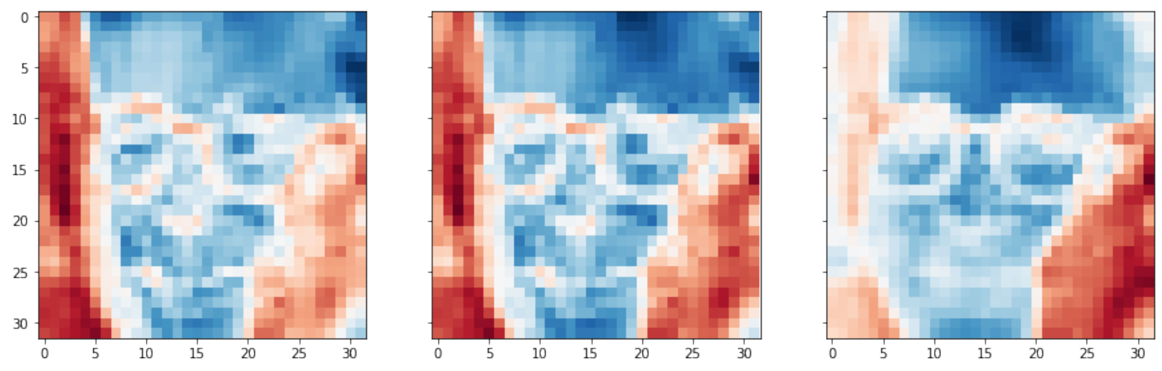


Figure 7: These were generated by tuning the epsilon parameter to stop the gradient descent early. Learning rate values of $10^{-4}, 10^{-5}, 10^{-6}$, gave 370, 183, and 1 iterations for gradient descent respectively. The images are in that order from left to right. In these runs, ϵ was kept constant at 1 x10^{-4} The longer the algorithm runs, the more different the visualization of theta looks from an image. This is done on a dataset of 4, with 2 images of Carell and Baldwin each.

$$\frac{\partial J}{\partial \Theta_{01}} = 2x^0(\Theta_{01}^T x^{(0)} - y^0)_1 \tag{4}$$

The other derivatives in each of the components will be similar, and are each a scalar. A factor of 2 can be pulled out, and can be rearranged to get the following:

$$\frac{\partial J}{\partial \Theta} = 2X(\Theta^T X - Y)^T \tag{5}$$

Here Θ is an n by k matrix. k is the number of features each sample has, which in this case is 1025. X is an n by m matrix, where m represents the number of samples. Thus, the result of $\Theta^T X$ would be k by m matrix, which matches the dimensions of Y. The final gradient matrix has the dimensions n by k, which corresponds to each row being the derivative of the image's performance, and each column corresponds to each individual θ component.

C

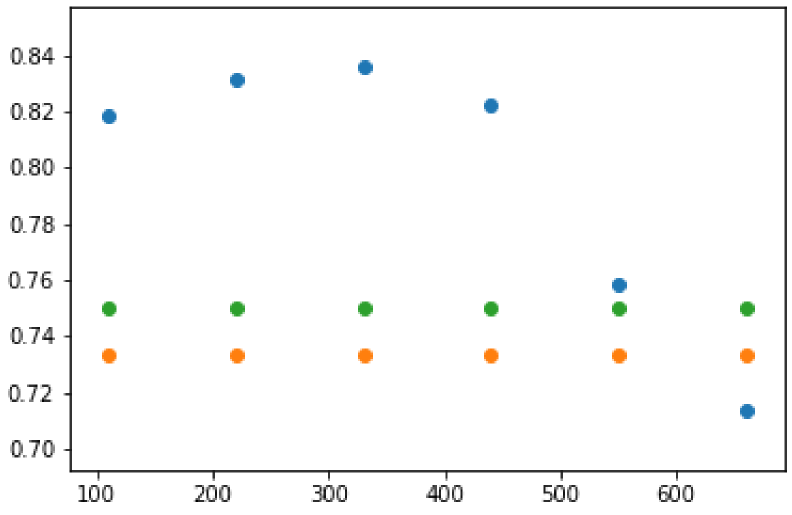
Here is the vectorized Cost Function

```
def df(x, y, theta):
    bias = np.ones( (1, x.shape[0]))
    x = np.hstack( (bias.T, x))
    return  2*np.dot(x.T,(np.dot(theta.T, x.T) - y.T).T )
```

D

To test the gradient, I used the method of finite differences. I used an h value of 1^{-5}. This is because this was enough to have the first 6 significant digits match for the derivatives computed with this method and with gradient descent for 4 of the 5 tested components. The first test component differed quite a bit. I used the Θ vector computed during gradient descent.

```
# Finite Difference checks, 5 elements
h = 0.00001
```



```
for i in range(5):
    test_theta_h =  theta.copy()
    test_theta_h[i, 0] += h
    print ((f(train_data, train_y, test_theta_h ) - f(train_data, train_y, theta))/(h))
    print df(train_data, train_y, theta)[i, 0], "\n"

OUTPUT:

0.00694579966875608
0.00034580094514069515

0.0352128495251236
0.035212827966209406

0.019375812598809716
0.01937578899546767

-0.008749850621825317
-0.008749873670697489

-0.01720764544188569
-0.017207661958343168
```

Problem 7

Performance of the Multi-Classifier
To determine the class of the data in the validation set, I determine the product of $X\Theta^T$, as X has the dimensions (60, 1025), and Θ has the dimensions (6, 1025). This yields an array of dimension (60, 6). I then take the maximum in each row, and set it to 1, and all other elements in that row as zero. This array is then compared to the truth values to see how much the model got right. The best model that I could get could only get 24 out of 60 right on the validation, so a performance of 40% . This score is not surprising,

given that the model could only get 317 right out of 660 on the training set, giving it a score of 48.03% The parameters that I used for the gradient descent are 0.001 for the learning rate, and an ϵ of 0.01. The initial theta was all zeros. These parameters make sense to me because the high ϵ value and learning rate prevent the model from over-fitting to the data. The learning rate could not be further increased, as then it would run into over flow error, and could not be lower as then it would not reach a minimum and would perform very poorly.
Here is the code to determine the actor labels and determine the score.

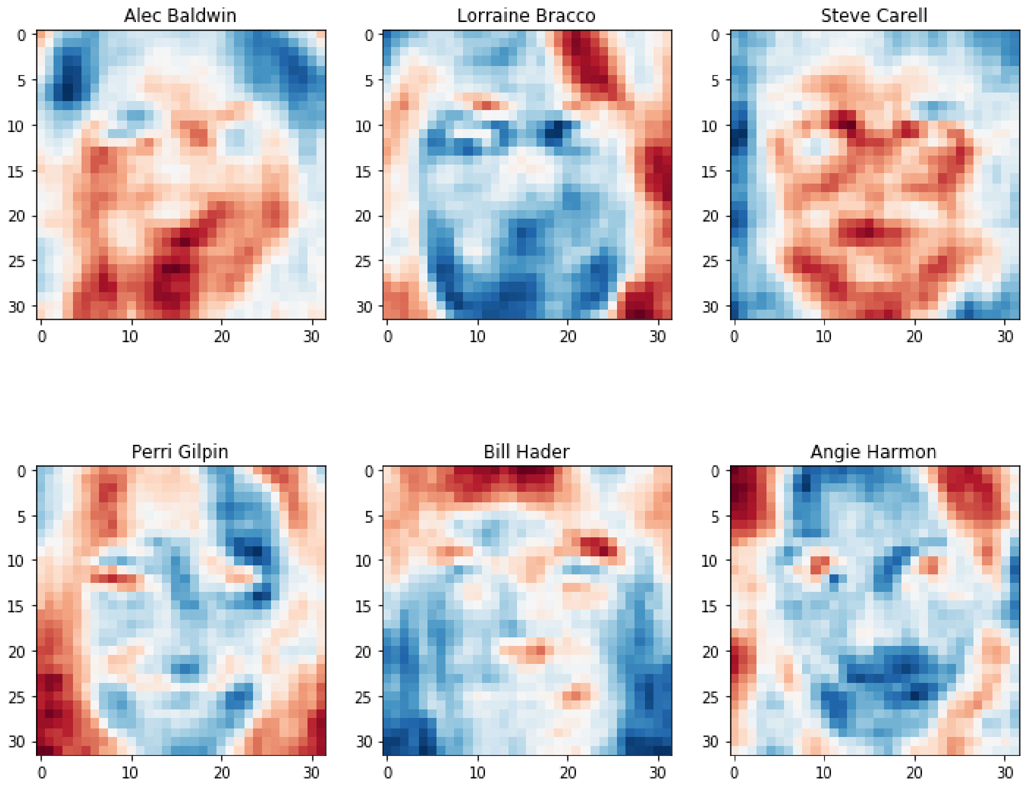
```python
valid_data_ones = np.concatenate((np.ones((1, valid_data.shape[0])).T, valid_data),
     axis = 1)
print valid_data_ones.shape, theta.shape
preds = np.dot( valid_data_ones, theta)

preds_max =  preds.max(axis = 1)
final_preds = np.isin(preds, preds_max).astype(int)

#how many of the hot keys match
len(np.where((test_y == final_preds).all(axis =1)==True)[0])
```

Problem 8

Visualizing the Multi-Class Model


Alec Baldwin
Lorraine Bracco
Steve Carell
Perri Gilpin
Bill Hader
Angie Harmon