CSC411: Assignment #1

Due on Monday, January 29, 2018

Xin Jie Lee

January 29, 2018

Prefix

The code for this project is written in Python 3.6 using Anaconda 3 interpreter.

Problem 1

Dataset description

This project involved the building of a system for face recognition and gender classification. The images used in building this system were obtained from FaceScrub, and a total of 1675 images for 12 actors and actresses were obtained. The actors and actresses whose images we would be working with are as follows: Alec Baldwin, America Ferrera, Angie Harmon, Bill Hader, Daniel Radcliffe, Fran Drescher, Gerard Butler, Kristin Chenoweth, Lorraine Bracco, Michael Vartan, Peri Gilpin and Steve Carell. The URLs used to download the images are contained in the files **facescrub_actors.txt** and **facescrub_actresses.txt**. Figure 1 depicts samples of the original images for each of the actors and actresses.

(a) Alec Baldwin.

(b) America Ferrera.

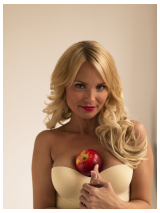
(c) Angie Harmon.

(d) Bill Hader.

(e) Daniel Radcliffe.

(f) Fran Drescher.

(g) Gerard Butler.



(h) Kristin Chenoweth.

(i) Lorraine Bracco.

(j) Michael Vartan.

(k) Peri Gilpin.

(l) Steve Carell.

Figure 1

These images in Figure 1 were obtained by running **get_data.py**.

These images have to be processed before they could be used for training and testing. The first step of processing involved cropping out the faces of the images. Each line in the **facescrub_actors.txt** and **facescrub_actresses.txt** files contains the bounding box information for the location of the person's face

in the format of $x1$,$y1$,$x2$,$y2$, where ($x1$,$y1$) is the coordinate of the top-left corner of the bounding box and ($x2$,$y2$) is that of the bottom-right corner. Assuming the image is represented as a Python NumPy array I, a face in I can be obtained as $I[y1 : y2, x1 : x2]$. For example, the following line in **facescrub_actors.txt** contains the bounding box 463, 450, 1785, 1772, and the face in this image can be obtained as $I[450 : 1772, 463 : 1785]$

```
Alec Baldwin 3209 1862 http://sarcastic-news.com/wp-content/uploads/2013/11/
Alec_Baldwin_PETA_Shankbone_2008.jpg 463,450,1785,1772
```

These cropped images were converted to greyscale (**rgb2gray.py** and **get_data.py**) and resized to 32 by 32 (**get_data.py**). Figure 2 depicts the final processed version of the images from Figure 1.

(a) Alec Baldwin.

(b) America Ferrera.

(c) Angie Harmon.

(d) Bill Hader.

(e) Daniel Radcliffe.

(f) Fran Drescher.

(g) Gerard Butler.

(h) Kristin Chenoweth.

(i) Lorraine Bracco.

(j) Michael Vartan.

(k) Peri Gilpin.

(l) Steve Carell.

Figure 2

However, there were some processed images that do not depict any person's face or the entirety of a person's face. Examples of these are shown in Figure 3. Since the elimination of inaccurate images would help to improve the performance of the program, steps were taken to remove these bad images. These badly processed images are usually the result of inaccurate bounding boxes provided in the **facescrub_actors.txt** and **facescrub_actresses.txt** files. Most of these bad bounding boxes seem to be present in duplicate images in the original dataset. Hence, many of these badly processed images could be removed by checking for duplicate images. Duplicate images were detected by comparing their SHA-256 hash codes (lines 47-52 and lines 101-115 of **get_data.py**). When duplicate images are detected, only the latest copy was kept while earlier copies were deleted. This approach helped eliminated a vast majority of the badly processed images, however the downside to this method was that it greatly reduced the number of images available for training and testing. For example, there were only a total of 86 images available for the actress Peri Gilpin after the elimination of all duplicate images. There were over 100 images available for each of the other actors and

actresses. Finally, the processed images are stored in two seperate folders, one for each gender.

(a) An image that does not capture Carell's face.

(b) An image that does not capture Bracco's face.

(c) An image containing part of Drescher's face.

(d) An image containing part of Hader's face.

Figure 3

Problem 2

Separate the dataset into three non-overlapping parts: the training set, the validation set, and the test set

The training, validation and training inputs are generated seperatey from the labels. The function `preprocess_images` creates the matrices containing the training, validation and test images for an individual. The processed images for the corresponding individual were loaded into an array `images` from the corresponding folder, `cropped/male` for actors and `cropped/female` for actresses. The array `images` was then shuffled and each image was converted into a numpy matrix and flattened into a vector. The image vector would be normalized by dividing every number in the vector by 255 since the pixel intensity of a greyscale image ranges from 0 to 255. This procedure ensured that every input number in the vector ranges from 0 to 1. A bias constant of 1 is then added to the front of every image vector. In total, an image vector will contain 1025 float variables, 1 variable for the bias unit and 1024 variables for the pixels in the image.

For parts 3 and 4 of this project, the training set would hold 100 images, while both the validation and test sets contained 10 images each. The shuffling of the array `images` ensured that different images would be loaded into the various input sets each time the function `preprocess_images` was ran. For parts 5 to 8 of the project, insufficient images avaliable for the actress Peri Gilpin (86 images in total) meant that only 66 images were used for the training set, while the validation and test sets still kept 10 images each. The function `preprocess_input_sets` would create the aggregate training, validation and test input matrices for all the actors and actresses needed for each part of the project.

For parts 3 and 4 of the project, the function `preprocess_labels_two` was used to create the output training, validation and test labels. Images for Alec Baldwin would be assigned the output value of $y = 1$, while images for Steve Carell would be assigned the value of $y = -1$. For part 5 of the project, images for female actresses were assigned the output value of $y = 1$, while images for male actors were assigned the output value of $y = -1$. For parts 7 and 8, the function `preprocess_labels_multi` would be used to generate the output labels in the following format:

Lorraine Bracco = [1, 0, 0, 0, 0, 0]
Peri Gilpin = [0, 1, 0, 0, 0, 0]
Angie Harmon = [0, 0, 1, 0, 0, 0]
Alec Baldwin= [0, 0, 0, 1, 0, 0]
Bill Hader = [0, 0, 0, 0, 1, 0]
Steve Carell = [0, 0, 0, 0, 0, 1]

Problem 3

Build a classifier to distinguish pictures of Alec Baldwin from pictures of Steve Carell

Linear Regression will be used to build the classifier to distinguish pictures between Alec Baldwin and Stevel Carell. The parameters for the linear regression model will be computed iteratively using gradient descent, and the goal will be to find the parameters Θ that best minimize the cost function:

$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\Theta(X^{(i)}) - Y^{(i)})^2$
where

$h_\Theta(X^{(i)}) = \sum_{j=0}^{n} X_j^{(i)} \theta_j$

The cost function measures the average error between the predicted and actual outputs, and the lower the cost function, the lower the average error will be. A lower average error will ensure that the predictions are more accurate. Every iteration of gradient descent will update the parameters using the formula:

$\Theta_k = \Theta_{k-1} - \alpha \nabla J(\Theta_{k-1})$

where

$\nabla J(\Theta_{k-1}) = \frac{1}{m} \sum_{i=1}^{m} (h_{\Theta_{k-1}}(X^{(i)}) - Y^{(i)}) X^{(i)}$

and α is the learning rate to be optimized, X is the input matrix of the images, Y is the matrix containing the output labels identifying the actors/actress in the images, m refers to the total number of images and Θ is the vector of the parameters. For this part of the project, images for Alec Baldwin were assigned labels of 1 while images for Steve Carell were assigned labels of -1.

Before running the gradient descent algorithm, the parameters Θ have to be initialized. The initial paramters were initialized as 0 since 0 is the midpoint between -1 and 1, which is the range of most optimized θ. A grid search was carried out to find the optimal learning rate α. It was noticed that if the chosen α was too large, the system would fail as a result of encountering an overflow in computing the cost function. This was likely the result of cost function growing too large. The gradient descent would update the paramters Θ untill the difference between the previous parameters and the current parameters is less than e^{-5} or the algorithm reached the predefined maxium number of iterations. The grid search for the optimal α was carried out with 4 different number of maximum iterations: $1,000$, $5,000$, $10,000$ and $20,000$. The figues shown below highlight the results of the grid search.

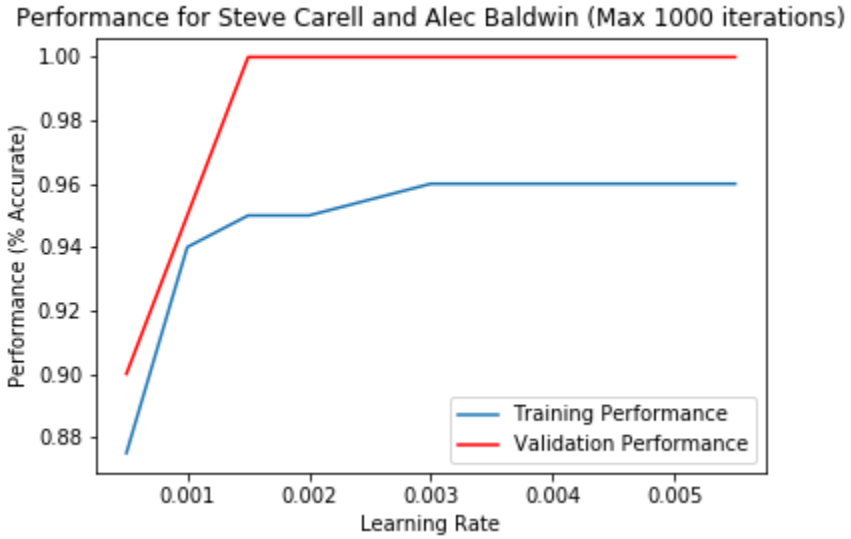


Figure 4: Plot of training and validation performance vs learning rate for maximum of 1,000 iterations

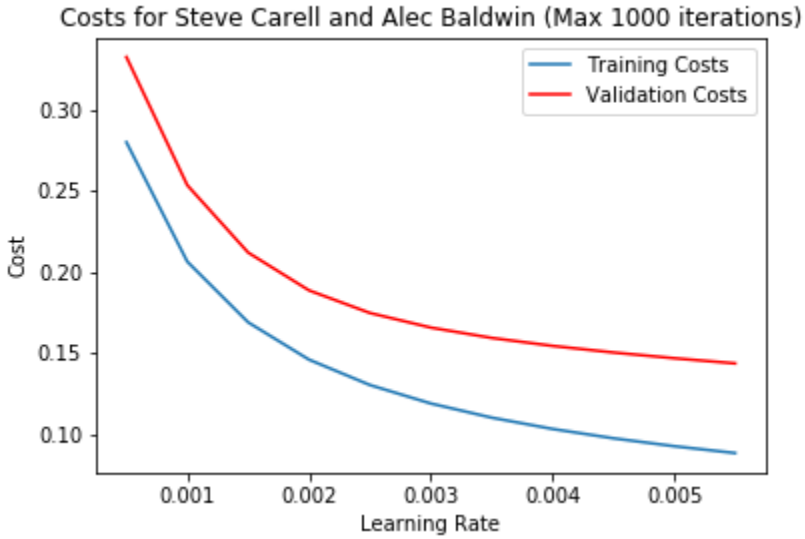


Figure 5: Plot of training and validation cost vs learning rate for maximum of 1,000 iterations

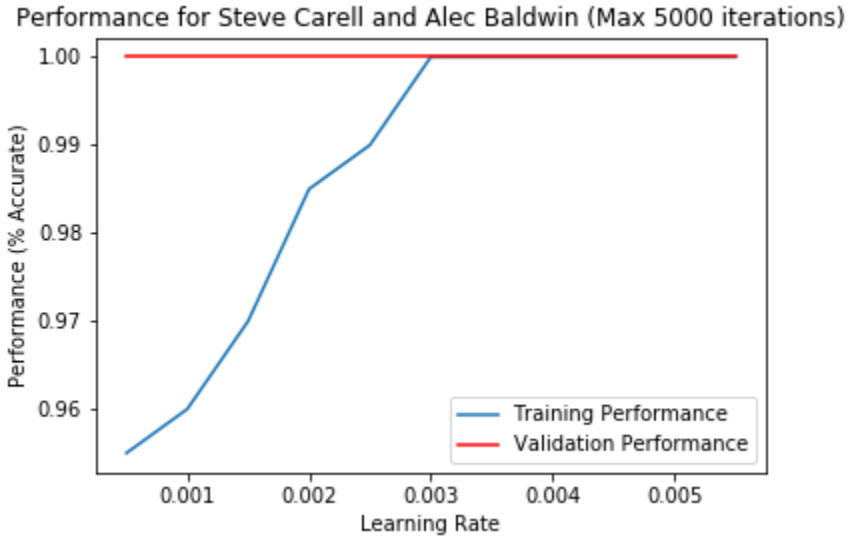


Figure 6: Plot of training and validation performance vs learning rate for maximum of 5,000 iterations

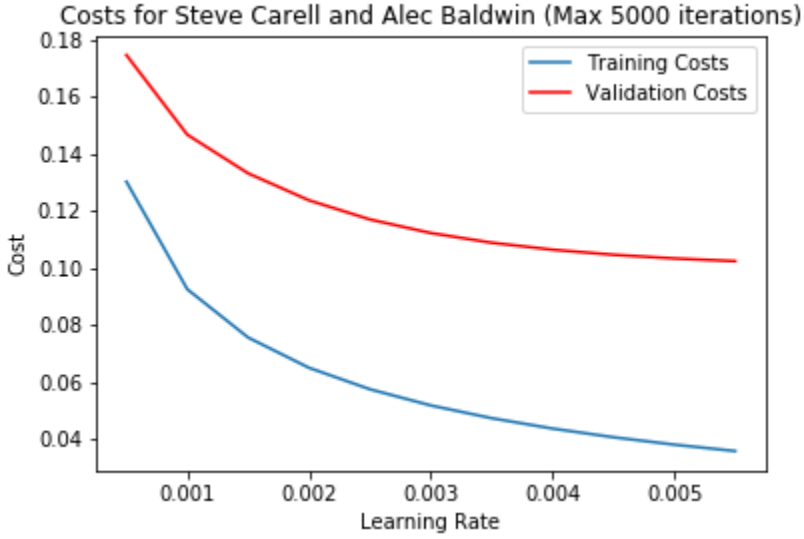


Figure 7: Plot of training and validation cost vs learning rate for maximum of 5,000 iterations

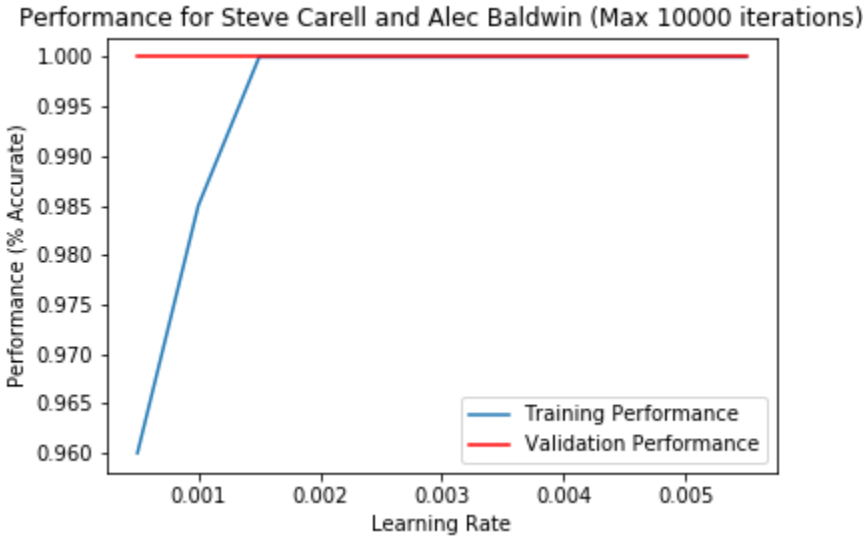


Figure 8: Plot of training and validation performance vs learning rate for maximum of 10,000 iterations

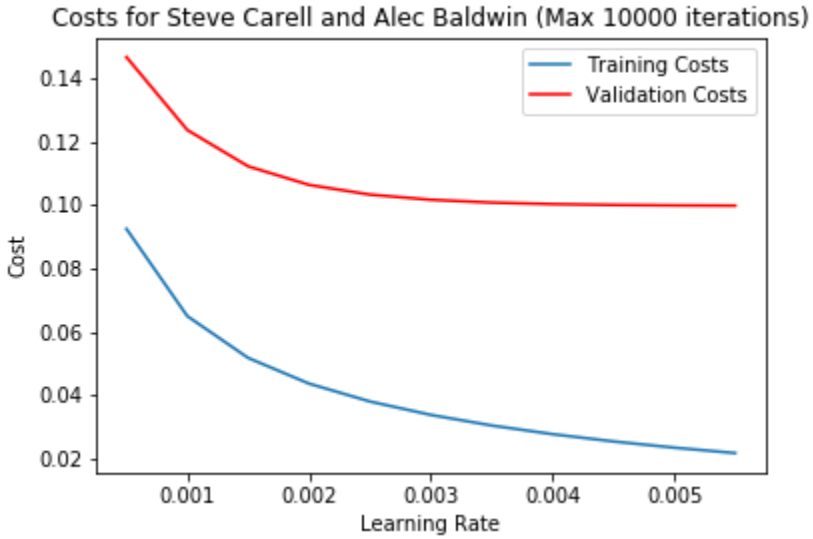


Figure 9: Plot of training and validation cost vs learning rate for maximum of 10,000 iterations

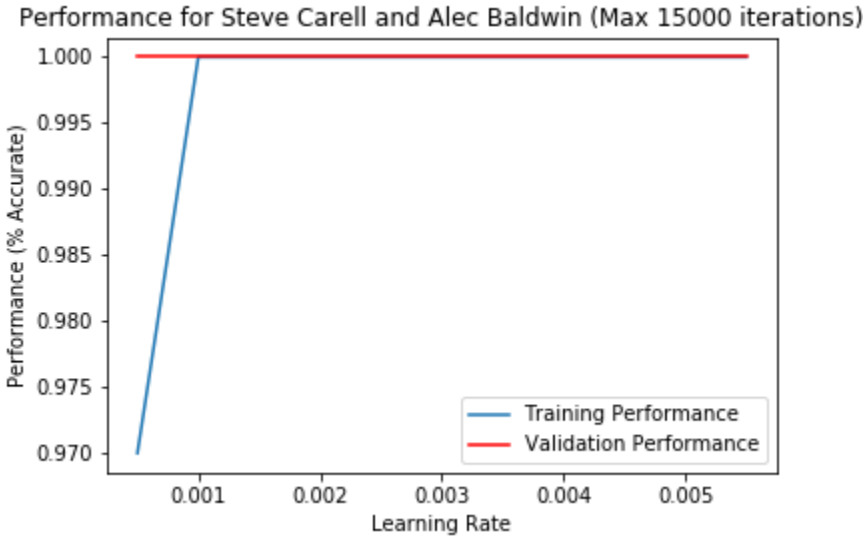


Figure 10: Plot of training and validation performance vs learning rate for maximum of 15,000 iterations

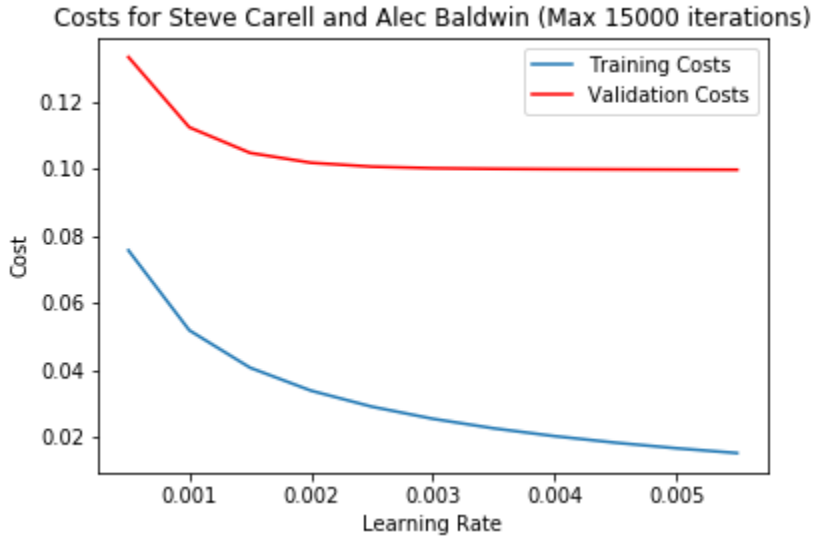


Figure 11: Plot of training and validation cost vs learning rate for maximum of 15,000 iterations

The code for computing the output of the classifier and measuring the accuracy of the predicted output against the actual output label is shown below:

```
def performance(thetas, inputs, labels):
    ''' Returns accuracy of predicted output vs actual output
    for 2 class classification '''
    accuracy = 0
    predictions = np.dot(inputs, thetas)
    for i in range(len(predictions)):
        if predictions[i] > 0:
            # Prediction is closer to 1
            accuracy += (labels[i] == 1)
        elif predictions[i] < 0:
            # Prediction is closer to -1
            accuracy += (labels[i] == -1)
    return accuracy/float(len(labels))
```

In the code, thetas refer to the Θ matrix, while inputs refer to transpose of the X matrix (matrix of images) and labels refer to the transpose of the Y matrix (matrix of labels). The predicted output would be the result obtained from the multiplication of $\Theta^T X$, where Θ is the final optimized parameters and X is the matrix containing the images. In the code, the prediction is computed in line 5. Images in the validation set will be classified as belonging to Alec Baldwin if their predicted output label is closer to 1, or belonging to Steve Carell if their predicted output label is closer to -1.

Based on the results from the grid search, a learning rate of 0.005 was selected. In addition. the maximum number of iterations for the gradient descent algorithm was set to $10,000$. These selections generated relatively good performance and low cost (average error) for the training and validation sets.

Results for chosen learning rate of 0.005 and maximum number of iterations of 10,000
Training Performance: 100.0 %
Validation Performance: 95.0 %
Training Cost: 0.0223267253767
Validation Cost: 0.102806329233

Note: Performance of the model will vary slighlty each time the model is run

Problem 4

***Part 4 (a)**: Visualizing parameters Θ using full training dataset vs training dataset of 2 images*

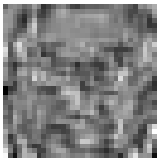
The following greyscale images of the paramaters Θ were obtained by training the model with 2 training images. The faces of the actors are discernabile in these images, which is likely the result of the images capturing all of the details and noise of the 2 training images used to generate the parameters. Hence, the use of a small number of training images will result in overfiting of the model, as the model will fit all of the details and noise of the images relatively well

(a) Parameters for Alec Baldwin using 2 training images.

(b) Parameters for Steve Carell using 2 training images.

Running the model on the full training dataset of 100 images helped reduced the amount of overfitting in the parameters Θ. As expected, the actors' faces were less discernable in the images of the parameters.

(a) Parameters for Alec Baldwin using 100 training images.



(b) Parameters for Steve Carell using 100 training images.

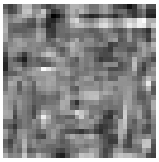
***Part 4 (b)**: Visualizing parameters Θ after 20,000 iterations of gradient descent vs 50 iterations*

The following images below are visualizations of the parameters Θ using the full training dataset and running the gradient descent algorithm for a maximum of 50 iterations. These images closely resemble the faces of their respective actor, and it is apparent that overfitting is present in the model.

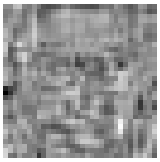
(a) Parameters for Alec Baldwin after 50 iterations.

(b) Parameters for Steve Carell after 50 iterations.

Running the gradient descent algorithm for 20,000 iterations certainly decreased the amount of overfitting in the paramters Θ, and the actors' faces were far less discernable in the images.



(a) Parameters for Alec Baldwin after 50 iterations.



(b) Parameters for Steve Carell after 50 iterations.

Problem 5

Gender classification and overfitting

For this part of the project, the model will be trained to classify the gender of the person depicted in the images. The training of the model will be carried out on images from these 6 actors/actresses: Lorraine Bracco, Peri Gilpin, Angie Harmon, Alec Baldwin, Bill Hader and Steve Carell. These actors/actresses will be referred to as known actors/actresses for the purpose of this report. The trained model will then be tested on images from 6 other actors/actresses: Fran Drescher, America Ferrera, Kristin Chenoweth, Gerard Butler, Daniel Radcliffe and Michael Vartan. These actors/actresses will be refered to as unknown actors/actresses, and 30 images of each unknown actor/actress will be used to test the model. In addition, the model will be trained on datasets of 5, 10, 15, ... , 60, 65 images from each known actor/actress to demonstrate the effects of overfitting. Images for female actesses will be given a label of $y = 1$ while images for male actors will be given a label of $y = -1$. Hence, if the predicted output for a test image is closer to 1, the person in the image will be classified as female. On the other hand, if the predicted output of a test image is closer to -1, the person in the image will be classified as male. Shown below are the results of the models:

Figure 16: Plot of training and validation performance vs training set size for the 6 known actors/actresses

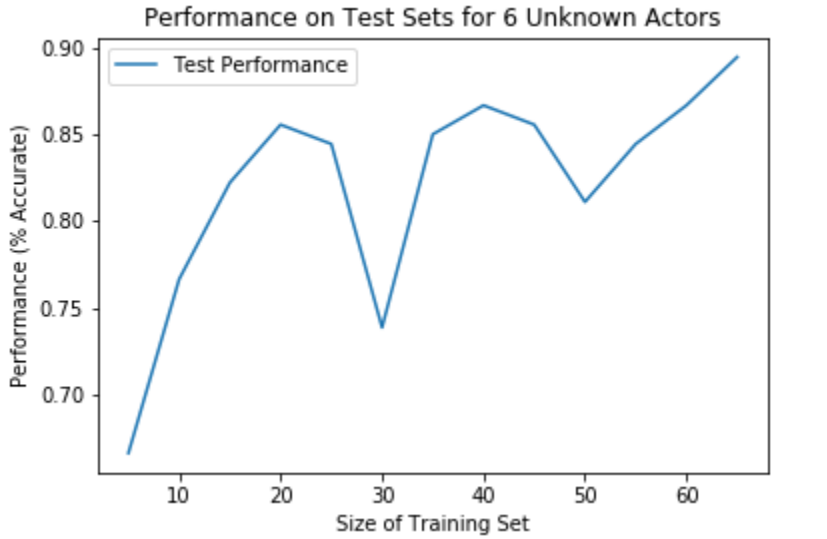


Figure 17: Plot of validation performance vs training set size for the 6 unknown actors/actresses

Running the model on smaller training dataset certainly resulted in a greater amount of overfitting which led to poorer performance. The test performance on images from the 6 unknown actors/actress is relatively poor ($< 80\%$) when using training sets of 10 images or less. Performance generally improved as the size of the training set increased, since overfiiting became less of an issue when more training data were used.

Using a training dataset of 65 images from each known actor/actress produced the following results when tested against test images from the 6 known actors/actresses.

```
Final results of gender classification for known actors/actresses
Training Performance: 98.9743589744 %
Validation Performance: 91.6666666667 %
```

The result of testing the same model on 30 images from each unknown actor/actress produced the following result:

```
Final result of gender classification for unknown actors/actresses
Classification Performance: 85.5555555556 %
```

Note: Performance of the model will vary slighlty each time the model is run

Problem 6

***Part 6 (a)**: Partial derivative of cost function with respect to θ_{pq}*

The cost function is:
$J(\Theta) = \sum_{i=1}^{m}(\sum_{j=1}^{k}(\Theta^T x^{(i)} - y^{(i)})_j^2)$

where m is the total number of images and k is the total number of labels.

The partial derivative of the cost function with respect to θ_{pq} (where p refers to the $(p-1)^{th}$ pixel if $p > 1$ or the bias unit if $p = 1$ and q refers to the q^{th} label) is:

$$\begin{aligned}\frac{\partial J(\Theta)}{\partial \theta_{pq}} &= 2\sum_{i=1}^{m}(\Theta_q^T x^{(i)} - y^{(i)})_j \frac{\partial J(\Theta)}{\partial \theta_{pq}}(\Theta^T x^{(i)} - y^{(i)})_j \\ &= 2\sum_{i=1}^{m}(\Theta_q^T x^{(i)} - y^{(i)})_j x_p^{(i)}\end{aligned}$$

Note:

$$\begin{aligned}\frac{\partial J(\Theta)}{\partial \theta_{pq}}(\Theta^T x^{(i)} - y^{(i)})_j &= \frac{\partial J(\Theta)}{\partial \theta_{pq}}(\theta_{1j}x_1^{(i)} + \theta_{2j}x_2^{(i)} + ... + \theta_{nj}x_n^{(i)}) \\ &= x_p^{(i)}\end{aligned}$$

and
Θ_q refers to the q^{th} column of Θ matrix or the q^{th} row of the Θ^T matrix.

***Part 6 (b)**: Gradient of cost function for classification of mutiple labels*

From 6 (a):
$\Theta_q^T x^{(i)} =$

$$\begin{bmatrix}\theta_{1q} & \theta_{2q} & \dots & \theta_{nq}\end{bmatrix}\begin{bmatrix}x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)}\end{bmatrix}$$

$$\begin{aligned}&= \theta_{1q}x_1^{(i)} + \theta_{2q}x_2^{(i)} + \dots + \theta_{nq}x_n^{(i)} \\ &= \sum_{l=1}^{n}\theta_{lq}x_l^{(i)}\end{aligned}$$

Hence,

$$\begin{aligned}\frac{\partial J(\Theta)}{\partial \theta_{pq}} &= 2\sum_{i=1}^{m}(\Theta_q^T x^{(i)} - y^{(i)})_j x_p^{(i)} \\ &= 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{lq}x_l^{(i)} - y^{(i)})_j x_p^{(i)}\end{aligned}$$

And,

$$\frac{\partial J(\Theta)}{\partial \theta} =$$

$$\begin{bmatrix}2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{l1}x_l^{(i)} - y_1^{(i)})x_1^{(i)} & 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{l2}x_l^{(i)} - y_2^{(i)})x_1^{(i)} & \dots & 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{lk}x_l^{(i)} - y_k^{(i)})x_1^{(i)} \\ \vdots & \vdots & \vdots & \\ 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{l1}x_l^{(i)} - y_1^{(i)})x_n^{(i)} & 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{l2}x_l^{(i)} - y_2^{(i)})x_n^{(i)} & \dots & 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{lk}x_l^{(i)} - y_k^{(i)})x_n^{(i)}\end{bmatrix}$$

To show that this is equivalent to $2X(\Theta^T X - Y)^T$, let us find the matrix representation of $2X(\Theta^T X - Y)^T$:

$\Theta =$

$$\begin{bmatrix} \theta_{11} & \dots & \theta_{1k} \\ \vdots & \vdots & \vdots \\ \theta_{n1} & \dots & \theta_{nk} \end{bmatrix}$$

$\Theta^T X =$

$$\begin{bmatrix} \theta_{11} & \dots & \theta_{n1} \\ \vdots & \vdots & \vdots \\ \theta_{1k} & \dots & \theta_{nk} \end{bmatrix} \begin{bmatrix} x_1^{(1)} & \dots & x_1^{(m)} \\ \vdots & \vdots & \vdots \\ x_n^{(1)} & \dots & x_n^{(m)} \end{bmatrix}$$

$=$

$$\begin{bmatrix} \theta_{11}x_1^{(1)} + \theta_{21}x_2^{(1)} + \cdots + \theta_{n1}x_n^{(1)} & \dots & \theta_{11}x_1^{(m)} + \theta_{21}x_2^{(m)} + \cdots + \theta_{n1}x_n^{(m)} \\ \vdots & \vdots & \vdots \\ \theta_{1k}x_1^{(1)} + \theta_{2k}x_2^{(1)} + \cdots + \theta_{nk}x_n^{(1)} & \dots & \theta_{1k}x_1^{(m)} + \theta_{2k}x_2^{(m)} + \cdots + \theta_{nk}x_n^{(m)} \end{bmatrix}$$

$=$

$$\begin{bmatrix} \sum_{l=1}^n \theta_{l1}x_l^{(1)} & \dots & \sum_{l=1}^n \theta_{l1}x_l^{(m)} \\ \vdots & \vdots & \vdots \\ \sum_{l=1}^n \theta_{lk}x_l^{(1)} & \dots & \sum_{l=1}^n \theta_{lk}x_l^{(m)} \end{bmatrix}$$

$Y =$

$$\begin{bmatrix} Y_1^{(1)} & \dots & Y_1^{(m)} \\ \vdots & \vdots & \vdots \\ Y_k^{(1)} & \dots & Y_k^{(m)} \end{bmatrix}$$

$\Theta^T X - Y =$

$$\begin{bmatrix} \sum_{l=1}^n \theta_{l1}x_l^{(1)} - Y_1^{(1)} & \dots & \sum_{l=1}^n \theta_{l1}x_l^{(m)} - Y_1^{(m)} \\ \vdots & \vdots & \vdots \\ \sum_{l=1}^n \theta_{lk}x_l^{(1)} - Y_k^{(1)} & \dots & \sum_{l=1}^n \theta_{lk}x_l^{(m)} - Y_k^{(m)} \end{bmatrix}$$

$X(\Theta^T X - Y)^T =$

$$\begin{bmatrix} x_1^{(1)} & \dots & x_1^{(m)} \\ \vdots & \vdots & \vdots \\ x_n^{(1)} & \dots & x_n^{(m)} \end{bmatrix} \begin{bmatrix} \sum_{l=1}^n \theta_{l1}x_l^{(1)} - Y_1^{(1)} & \dots & \sum_{l=1}^n \theta_{lk}x_l^{(1)} - Y_k^{(1)} \\ \vdots & \vdots & \vdots \\ \sum_{l=1}^n \theta_{l1}x_l^{(m)} - Y_1^{(m)} & \dots & \sum_{l=1}^n \theta_{lk}x_l^{(m)} - Y_k^{(m)} \end{bmatrix}$$

$=$

$$\begin{bmatrix} x_1^{(1)}(\sum_{l=1}^n \theta_{l1}x_l^{(1)} - Y_1^{(1)}) + \cdots + x_1^{(m)}(\sum_{l=1}^n \theta_{l1}x_l^{(m)} - Y_1^{(m)}) & \dots & x_1^{(1)}(\sum_{l=1}^n \theta_{lk}x_l^{(1)} - Y_k^{(1)}) + \cdots + x_1^{(m)}(\sum_{l=1}^n \theta_{lk}x_l^{(m)} - Y_k^{(m)}) \\ \vdots & \vdots & \vdots \\ x_n^{(1)}(\sum_{l=1}^n \theta_{l1}x_l^{(1)} - Y_1^{(1)}) + \cdots + x_n^{(m)}(\sum_{l=1}^n \theta_{l1}x_l^{(m)} - Y_1^{(m)}) & \dots & x_n^{(1)}(\sum_{l=1}^n \theta_{lk}x_l^{(1)} - Y_k^{(1)}) + \cdots + x_n^{(m)}(\sum_{l=1}^n \theta_{lk}x_l^{(m)} - Y_k^{(m)}) \end{bmatrix}$$

$=$

$$\begin{bmatrix} \sum_{i=1}^m x_1^{(i)}(\sum_{l=1}^n \theta_{l1}x_l^{(i)} - Y_1^{(i)}) & \dots & \sum_{i=1}^m x_1^{(i)}(\sum_{l=1}^n \theta_{lk}x_l^{(i)} - Y_k^{(i)}) \\ \vdots & \vdots & \vdots \\ \sum_{i=1}^m x_n^{(i)}(\sum_{l=1}^n \theta_{l1}x_l^{(i)} - Y_1^{(i)}) & \dots & \sum_{i=1}^m x_n^{(i)}(\sum_{l=1}^n \theta_{lk}x_l^{(i)} - Y_k^{(i)}) \end{bmatrix}$$

Hence:

$$2X(\Theta^T X - Y)^T =$$
$$\begin{bmatrix} 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{l1}x_l^{(i)} - y_1^{(i)})x_1^{(i)} & 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{l2}x_l^{(i)} - y_2^{(i)})x_1^{(i)} & \dots & 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{lk}x_l^{(i)} - y_k^{(i)})x_1^{(i)} \\ \vdots & \vdots & \vdots & \\ 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{l1}x_l^{(i)} - y_1^{(i)})x_n^{(i)} & 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{l2}x_l^{(i)} - y_2^{(i)})x_n^{(i)} & \dots & 2\sum_{i=1}^{m}(\sum_{l=1}^{n}\theta_{lk}x_l^{(i)} - y_k^{(i)})x_n^{(i)} \end{bmatrix}$$
Q.E.D

***Part 6 (c)**: Implement vectorized cost function and gradient*

The code for the vectorized cost function is:

```
def cost_function(thetas, X, Y):
    return (1.0/(2.0*float(len(Y))))*np.sum((np.dot(X, thetas) - Y)**2)
```

The code for the vectorized gradient function is:

```
def gradient(thetas, X, Y):
    return (1.0/(float(len(Y))))*np.dot(X.T, (np.dot(X, thetas) - Y))
```

Note: X in this code is actually equivalent to the transpose of X matrix from Part 6 (a) and 6 (b). Y in this code is actually equivalent to the transpose of Y matrix from Part 6 (a) and 6 (b)

***Part 6 (d)**: Verification of Gradient Descent*

To ensure that the vectorized gradient function is accurate, the partial derivative of the cost function with respect to θ_{pq} can be estimated using the finite difference approach according to this equation:

$$\frac{\partial J(\Theta)}{\partial \theta_{pq}} = \frac{f(\theta_{pq}+h, X, Y) - f(\theta_{pq}, X, Y)}{h}$$

for a small h. The estimated partial derivative can be compared to the result from the gradient function and both computations should generate approximately equivalent results. The average error between the estimated partial derivatives of the cost function with respect to every θ_{pq} and the gradient function can be analyzed to verify the accuracy of the vectorized gradient function. The code for this implementation is shown below:

```
def finite_difference(inputs, thetas, labels, h):
    ''' Returns average error per theta '''
    total_error = 0
    for i in range(thetas.shape[0]):
        for j in range(thetas.shape[1]):
            grad = gradient(thetas, inputs, labels)
            old_cost = cost_function(thetas, inputs, labels)
            new_thetas = thetas
            # Change one theta by h, hold all others constant
            new_thetas[i][j] += h
            new_cost = cost_function(new_thetas, inputs, labels)
            fin_diff = (new_cost - old_cost)/float(h)
            total_error += (fin_diff - grad[i][j])
    average_error = total_error/float(thetas.shape[0]*thetas.shape[1])
    return average_error
```

Using h of 0.0001 and running the code with 66 training images from each of the following actors/actresses: Lorraine Bracco, Peri Gilpin, Angie Harmon, Alec Baldwin, Bill Hader and Steve Carell, produced the following result:

```
Average error between finite difference and gradient function using h = 0.0001 is: 1.55717971133e-05
```

The relatively small error suggests that the implemented vectorized gradient function is correct.

Problem 7

Face Recognition for multiple person

The model will be used to perform face recognition for the following actors/actresses: Lorraine Bracco, Peri Gilpin, Angie Harmon, Alec Baldwin, Bill Hader and Steve Carell. The initial parameters Θ were set to 0 to ensure faster convergence since the final parameters often lie between 1 and -1. As mentioned in part 2 of the report, the output labels are generated in the following format:

Lorraine Bracco = [1, 0, 0, 0, 0, 0]
Peri Gilpin = [0, 1, 0, 0, 0, 0]
Angie Harmon = [0, 0, 1, 0, 0, 0]
Alec Baldwin= [0, 0, 0, 1, 0, 0]
Bill Hader = [0, 0, 0, 0, 1, 0]
Steve Carell = [0, 0, 0, 0, 0, 1]

To determine the predicted output of the i^{th} image, the results of the mutiplication of the Θ^T matrix and the i^{th} column of the X matrix will produce a 6 by 1 vector. The index of the biggest element in the vector will determine the person whom the image belongs to. For example, if the position of the biggest element is at index 2, then the image will be classified as belonging to Angie Harmon since vectors for Angie Harmon's images have a label of 1 in their 2^{nd} index. (Using index numbering that starts at 0). The code below computes the accuracy of the face recognition by comparing the position of the biggest element of the predicted output vector with the index of the label 1 in the actual label vector for the image.

```
def performance_multi(thetas, inputs, labels):
    ''' Returns accuracy of predicted output vs actual output
    for multiple class classification '''
    accuracy = 0
    predictions = np.dot(inputs, thetas)
    for i in range(len(predictions)):
        # If index of maximium element in each row (image) or predictions is equal to
        # the index of 1 in the corresponding row of the label matrix
        if np.argmax(predictions[i]) == np.argmax(labels[i]):
            accuracy += 1
    return accuracy/float(len(labels))
```

A grid search was implemented to find the optimal learning rate α and the maximum number of iterations for running the gradient descent algorithm. The results of the gird serach are shown below:

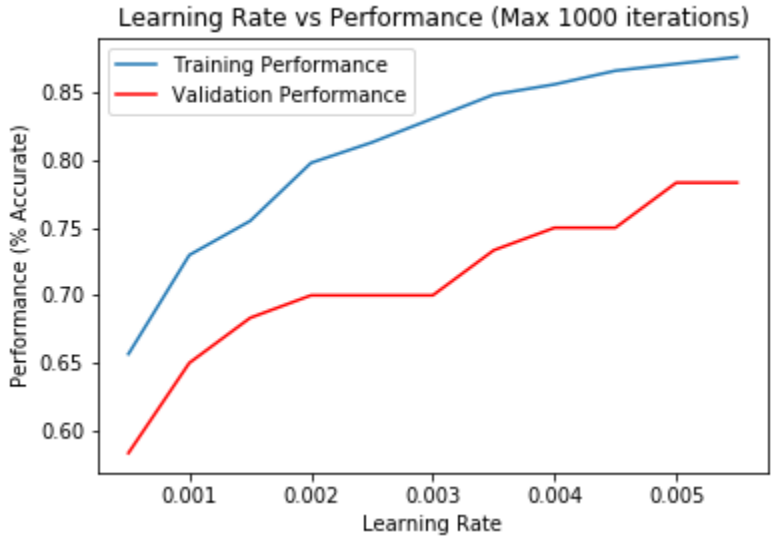


Figure 18: Plot of performance vs learning rate for maximum of 1000 iterations

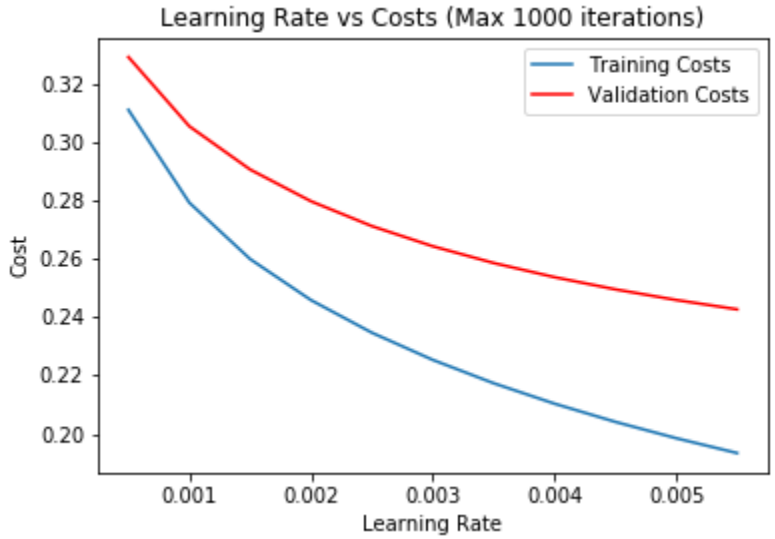


Figure 19: Plot of cost vs learning rate for maximum of 1000 iterations

Figure 20: Plot of performance vs learning rate for maximum of 5000 iterations

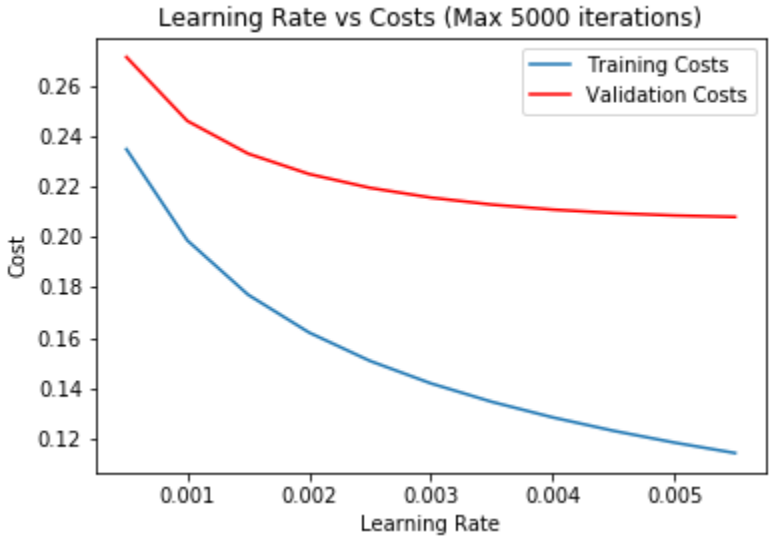


Figure 21: Plot of cost vs learning rate for maximum of 5000 iterations

Figure 22: Plot of performance vs learning rate for maximum of 10,000 iterations

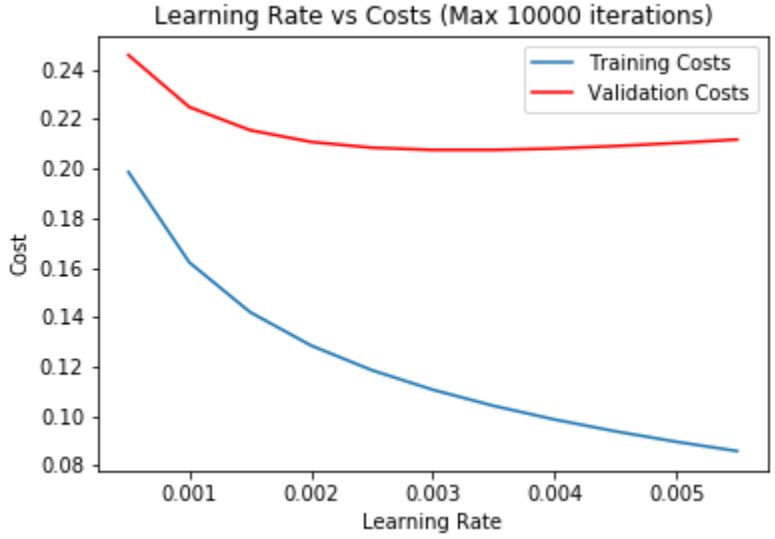


Figure 23: Plot of cost vs learning rate for maximum of 10,000 iterations

Figure 24: Plot of performance vs learning rate for maximum of 15,000 iterations

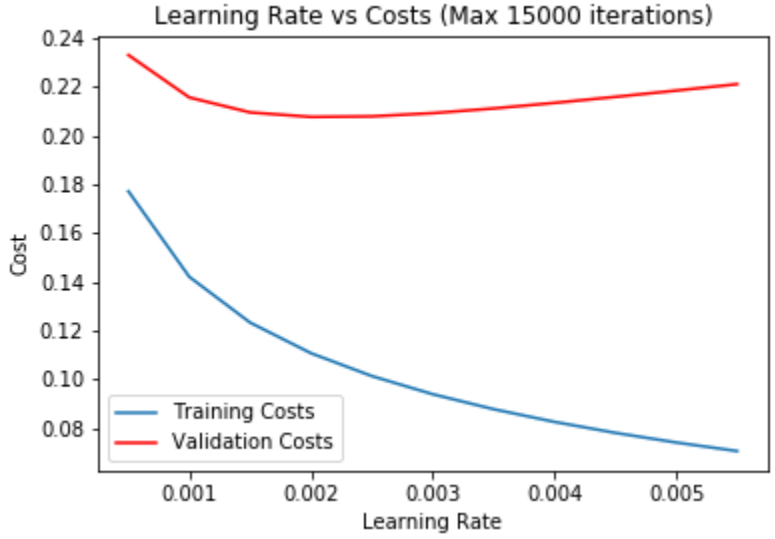


Figure 25: Plot of cost vs learning rate for maximum of 15,000 iterations

It was observed that choosing a learning rate of 0.005 and setting the maximim number of iterations to 10,000 generally produced the best results (higher performance and lower cost). In addition, it was observed that selecting learning rates above 0.006 would often result in an overflow in computing the cost function, likely the result of the cost function growing too big. Shown below is the performance of the classifier with the selected optimizations.

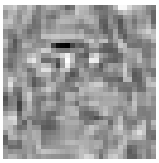
```
Results for chosen learning rate of 0.005 and maximum number of iterations of 10000
Final training set performance: 98.48484848484848 %
Final validation set performance: 80.0 %
Final training cost: 0.0846144034294
Final validation cost: 0.242543483824
```

Note: Performance of the model will vary slighlty each time the model is run

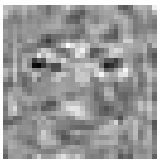
Problem 8

Visualizing parameters Θ

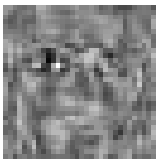
Shown below are the visualizations of the parameters Θ for the 6 actors and actresses whose images were analyzed in Part 7: Lorraine Bracco, Peri Gilpin, Angie Harmon, Alec Baldwin, Bill Hader and Steve Carell. The parameters were optimized by running gradient descent for a maximum number of iterations of 10,000 with the selected learning rate α of 0.005. There were 6 visualizations produced in total, one for each actor/actress.



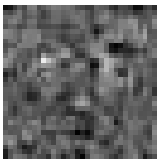
(a) Parameters for Lorraine Bracco.



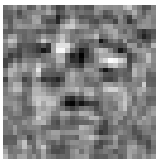
(b) Parameters for Peri Gilpin.



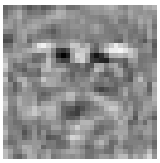
(c) Parameters for Angie Hammon.



(d) Parameters for Alec Baldwin.



(e) Parameters for Bill Hader.



(f) Parameters for Steve Carell.