Face Recognition and Gender Classification with Regression

Shahin Jafari

January 27, 2018

Contents

How to Use the Code

In order to use the supplied code, the user must specify the part number in the main function of the classifier.py . The program will run that specific part as a result.
Note that in some parts, you may find os.chdir() command. This was used because I did not use Windows command line and instead, I used Pyzo, which is an IDE. Therefore, I had to change the working directory myself in the code. Although they are commented out, please make sure double check them. Thank you!

Part 1

The dataset is a dataset of pictures provided by FaceScrub. It includes pictures of 6 actors and 6 actresses. The readme.txt file that is associated with this dataset has the URL's to download the pictures and boundaries to crop them in order to extract artists' faces.
Some of the pictures are removed from their associated URL's by the host, and hence when they are downloaded, there is no meaningful picture incorporated in them. For example:

Figure 1: A picture download from a URL in the readme file that should have included a picture of Bill Hader.

A few of them have a very bad bounding boxes. As a result, the uncropped version of the picture is very good and reasonable, but since the bounding box is inaccurate, they become unusable. An example can be found below:

Figure 2: A very good picture of Lorraine Bracco.

Figure 3: The cropped version of the picture above. Note that the picture is now unusable due to a inaccurate bounding box.

Some of them are from magazines, and they include text as well as the picture. Nomally, this is fine as long as the text does not cover the face of the artist, but in some cases, the text actually hiddens some of the features of the artist's face. An example is illustrated below:

Figure 4: A cropped picture of Angie Harmon. Note that some of the letters are covering the face and does not lead to an accurate result.

There are also repetitive pictures for some artists which will make the regression biased toward that specific picture. Below is an example of a repetitive picture in the dataset:

Figure 5: A picture download for Fran Drescher and labeled as Drescher43.

Figure 6: A picture download for Fran Drescher and labeled as Drescher121. Note that this one and the above picture are both the same, and downloaded twice.

The rest of the dataset images were fine and reasonable to use. Note that for this project, in order to minimize the error on the dataset, the above cases for irregular pictures except for the repetitive images were removed manually to achieve better results. In practice, it might not be possible to remove them by hand, but since the number of pictures in practice is large, they wouldn't affect the performance greatly.
The cropped faces are mostly aligned with each other, with the exception that some of the picture are from different angles and for example, their eye pixels may get aligned with the forehead of the rest of the dataset. Again, they would not be significant source of error, since they will be averaged out by the rest of the datasets' pixels and attributes. It can also help the algorithm to potentially identify the new pictures from different angles, without any explicit code.
The images were all downloaded using urllib.urlretrieve command since the timeout function did not result in enough number of images, and it could also yield inconsistencies with different internet connections. All the used images are supplied.

Part 2

In this part, I have decided to split the dataset in a very systematic way such that it will always yield the same results around the same dataset with the same picture names. The function receives the set of actors and the number of images per each actor/actress and returns three lists, where they represent the training set, the validation set and the test set in order. The algorithm is as follows:

1. do it for all artists in list of received actors
2. i=0
3. Training Set
   - Repeat until the length of the training set is not equal to the number of actors * number of images per actor that must be in the set
   - read image[i] of specified actor
   - i= i+1
4. Validation Set
   - Repeat until the length of the validation set is not equal to the number of actors * number of images per actor that must be in the set
   - read image[i] of specified actor
   - i= i+1
5. Test Set
   - Repeat until the length of the test set is not equal to the number of actors * number of images per actor that must be in the set
   - read image[i] of specified actor
   - i= i+1

This algorithm puts the first n pictures in the training set, the next m pictures in the validation set and the next l pictures in the test set where:
n = number of actors * images per actor in the training set
m = number of actors * images per actor in the validation set
l = number of actors * images per actor in the test set
Note that this function also labels the pictures of all the sets, with their corresponding name. There are 3 versions of this function on my code where each one have the same dividing algorithm, but the lableing is different amongst them.
Also note that this systematic way of division can introduce bias, where if the first n pictures have a particular attribute and the next m does not have that atribute, the data will be overfitted but since the number of images are large, it would be minimal.

Part 3.

For this part, the difference square cost function was used in order to classify the pictures:

$$J(\theta) = \sum_{i}^{m} \theta^T x - y \tag{1}$$

From the dividing function in part 2, each image was labled as 1 or -1, where 1 represents Alex Baldwin and -1 represents Steve Carell. Below are the results obtained for this part. Note that in order to reproduce the results, please use the images supplied and consult 'How to Use the Code' section of this document.

```
cost on training set per image: 0.353835
cost on validation set per image: 0.361828
performance on training set: 92.8571%
performance on validation set: 95%
```

Figure 7: Performance of linear regression on a training set of 70 images, validation set of 10 images and test set of 10 images.

In order to make the algorithm work, I have tried different αs and reduce them until the algorithm worked. When the α is too large, the gradient descent would not be able to make the $\theta' s$ converge and all the $\theta_i' s$ became nan(not a number). This can be a consequence of the gradient descent where it missed the local minimum. It sometimes also made the system to show a warning message, indicating that the multiplication cause an overflow and again, all the $\theta_i' s$ became nan. Other important variables used in this algorithm are the maximum number of iterations on gradient descent and the accuracy of it which where 30000 and $1e^{-5}$.
The source code is shown on the next page.

```
from pylab import *
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cbook as cbook
import random
import time
from scipy.misc import imread
from scipy.misc import imresize
import matplotlib.image as mpimg
import os
from scipy.ndimage import filters
import urllib


def get_baskets(act,train_size, valid_size, test_size):
    '''
    input: actors list , size of each set per actor
    output: 3 baskets ( list ), training , validation and test
    '''
    training = []
    validation = []
    test = []
    cnt = 0
    label = 0
    for a in act:
        if a == 'Alec Baldwin':
            label = 1
        elif a== 'Steve Carell':
            label = -1
        cnt += 1
        i = 0
        while len(training) !=  cnt*train_size :
            name = a.split()[1]
            filename = name+str(i)
            try:
                image = imread(os.getcwd() + '\\cropped\\'+ a +'\\'+filename+ '.JPG')
                img_gray = rgb2gray(image)
                img_scaled = imresize(img_gray,(32,32)) / 255.0
            except IOError:
                try:
                    image = imread(os.getcwd() + '\\cropped\\'+ a +'\\'+filename+ '.png')
                    img_gray = rgb2gray(image)
                    img_scaled = imresize(img_gray,(32,32)) / 255.0
                except:
                    i += 1
                    continue
            i += 1
            training.append((img_scaled,label))

        while len(validation) !=  cnt*valid_size:
            name = a.split()[1]
            filename = name+str(i)
            try:
```

```
                image = imread(os.getcwd() + '\\cropped\\'+ a +'\\'+filename+ '.JPG')
                img_gray = rgb2gray(image)
                img_scaled = imresize(img_gray,(32,32)) / 255.0
            except IOError:
                try:
                    image = imread(os.getcwd() + '\\cropped\\'+ a +'\\'+filename+ '.png')
                    img_gray = rgb2gray(image)
                    img_scaled = imresize(img_gray,(32,32)) / 255.0
                except:
                    i += 1
                    continue
            i += 1
            validation.append((img_scaled,label))

        while len(test) !=  cnt*test_size:
            name = a.split()[1]
            filename = name+str(i)
            try:
                image = imread(os.getcwd() + '\\cropped\\'+ a +'\\'+filename+ '.JPG')
                img_gray = rgb2gray(image)
                img_scaled = imresize(img_gray,(32,32)) / 255.0
            except IOError:
                try:
                    image = imread(os.getcwd() + '\\cropped\\'+ a +'\\'+filename+ '.png')
                    img_gray = rgb2gray(image)
                    img_scaled = imresize(img_gray,(32,32)) / 255.0
                except:
                    i += 1
                    continue
            i += 1
            test.append((img_scaled,label))

    return training,validation,test

def train(training):
    '''
    input: training basket
    output: a matrix of thetas for linear regression
    '''

    # how to obtain the best learning rate
    '''
    init_theta = np.zeros((1+32*32,1))
    t_min = init_theta
    l_rates = [1,0.1,0.01,0.001,0.0001,0.00001,0.00001,0.000001]
    #learning_rate = 0.00001
    x = np.zeros((1 + 32*32,len(training)))
    y = np.zeros((1,len(training)))
    x[0,:] = 1
    i = 0
    f_min = 0
    for learning_rate in l_rates:
        for i in range(len(training)):
```

```
            x[1:,i] = training[i][0].flatten()
            y[0,i] = training[i][1]
        theta = grad_descent(f,df, x, y,init_theta,learning_rate)
        fun_value = f(x,y,theta)
        if (i == 0 or fun_value < f_min):
            i += 1
            f_min = fun_value
            t_min = theta
        print (fun_value)
    return t_min
    '''

    #using the found learning rate
    init_theta = np.zeros((1+32*32,1))
    learning_rate = 0.0000001
    x = np.zeros((1 + 32*32,len(training)))
    y = np.zeros((1,len(training)))
    x[0,:] = 1
    for i in range(len(training)):
        x[1:,i] = training[i][0].flatten()
        y[0,i] = training[i][1]
    theta = grad_descent(f,df, x, y,init_theta,learning_rate)

    return theta

def f(x, y, theta):
    '''
    cost function
    input: training set and their label(x,y = 0 or 1) and thetas
    output: cost function
    '''
    return sum( (y - dot(theta.T,x)) ** 2)      #J

def df(x,y,theta):
    '''
    gradient function
    input: training set and their label(x,y = 0 or 1) and thetas
    output: derivative of cost function
    '''
    return -2*sum( (y - dot(theta.T,x)) * x, 1).T        #J

def grad_descent(f, df, x, y, init_t, alpha):
    EPS = 1e-5   #EPS = 10**(-5)
    prev_t = init_t-10*EPS
    t = init_t.copy()
    max_iter = 30000
    iter  = 0
    sub = np.zeros(init_t.shape)
    while norm(t - prev_t) >  EPS and iter < max_iter:
        prev_t = t.copy()
        sub = alpha*df(x, y, t).reshape(sub.shape)
        t -= sub
        iter += 1
```

```
    return t         #t is the fitted thetas


def classify(test_set, theta):
    '''
    input: test set, thetas
    output: percentage correct classified
    '''

    correct = 0
    total = 0
    column = np.zeros((1 + 32*32,1))
    for point in test_set:
        column[0,0] = 1
        column[1:,0] = point[0].flatten()
        label_act = point[1]
        if ( dot(theta.T , column) >= 0):
            label_comp = 1
        else:
            label_comp = -1

        if (label_act == label_comp):
            correct += 1
        total += 1.0

    return correct/total * 100

def get_cost(set,theta):
    x = np.zeros((1 + 32*32,len(set)))
    y = np.zeros((1,len(set)))
    x[0,:] = 1
    for i in range(len(set)):
        x[1:,i] = set[i][0].flatten()
        y[0,i] = set[i][1]

    return f(x,y,theta)
```

Part 4

a) In order to display the thetas, one must remove the y-intercept from them since we consider the y-intercept in our regression. Therefore, 1024 values were used forθs instead of 1025. Note that 1024 *theta* comes from the fact that the pictures were 32x32.

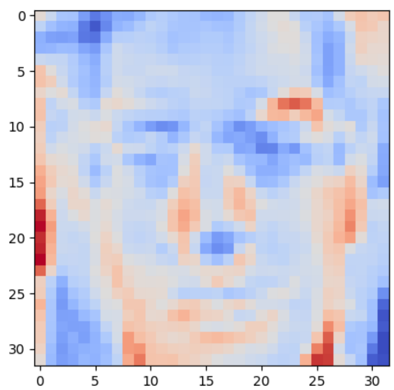

Figure 8: Visualized θs obtained from training on 2 images per actor (Alec Baldwin and Steve Carell)

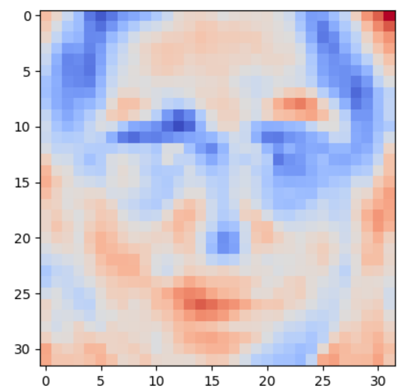

Figure 9: Visualized θs obtained from training on 70 images per actor (Alec Baldwin and Steve Carell)

Note that the picture of the θs coming from the training set of 2 images per actor looks more like a human than the other picture. It makes sense, since if we were only to train based on 1 picture per actor and the number of actors is 1, then we would have an exact picture of that actor as our θs. Therefore, the less number of training images, the more it's biased toward a specific configuration that is close to a picture that it used for trainin.

b)

Initializing θs have a significant effect on the displayed θs. The reason is that the gradient descent does not always give us the true minimum, and it may stop in the middle of the process due to maximum iterations. In those cases, if θs were initialized randomely, there is no facial attribute associated to them and since the gradient descent was not able to find the minimum, the displayed θs would be very disoriented. On the otherhand, if we initialize it as ones for example, since the label for one of the actors is also 1, even if it stops early, it will be toward the actors face rather than random points. By the same logic, increasing the number of iterations of gradient descent could result in more random orientation, since the gradient descent is also prone to miss the local minimum. Overall, θ initialization is the dominant factor in having a face-like final θs.

The following pictures were obtained from initializing the θ matrix as all zeros.

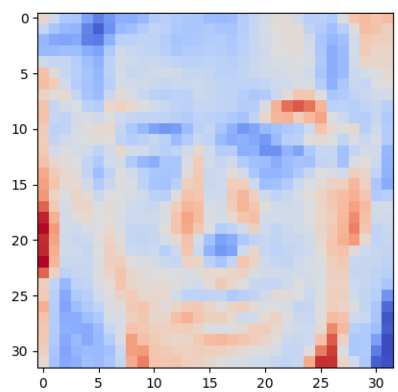

Figure 10: Visualized $\theta' s$ obtained from training on 2 images per actor (Alec Baldwin and Steve Carell)

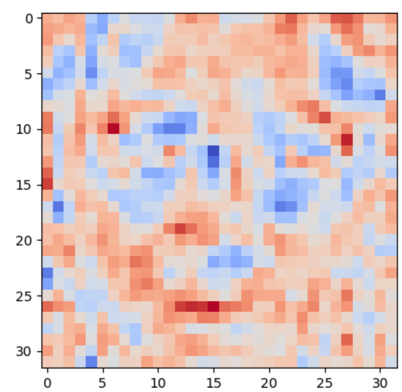

Figure 11: Visualized $\theta' s$ obtained from training on 70 images per actor (Alec Baldwin and Steve Carell)

The following pictures were obtained from initializing the θ matrix as all ones.

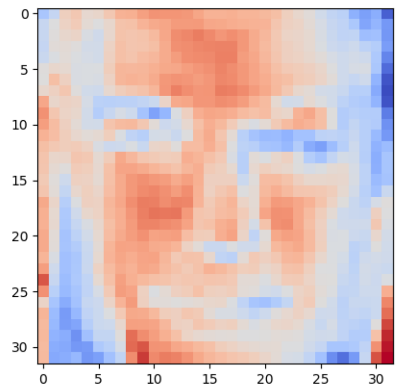

Figure 12: Visualized $\theta's$ obtained from training on 2 images per actor (Alec Baldwin and Steve Carell)

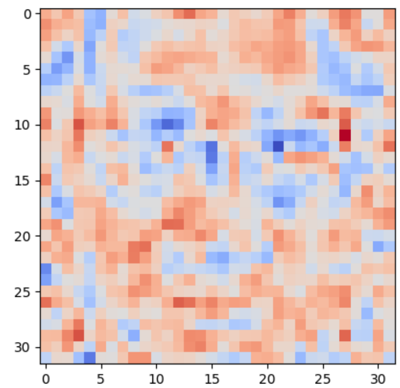

Figure 13: Visualized $\theta's$ obtained from training on 70 images per actor (Alec Baldwin and Steve Carell)

The following pictures were obtained from initializing the θ matrix as random numbers.

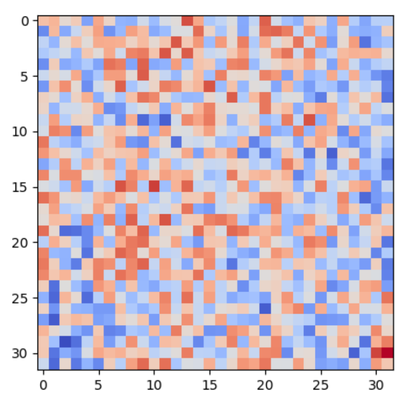

Figure 14: Visualized $\theta's$ obtained from training on 2 images per actor (Alec Baldwin and Steve Carell)

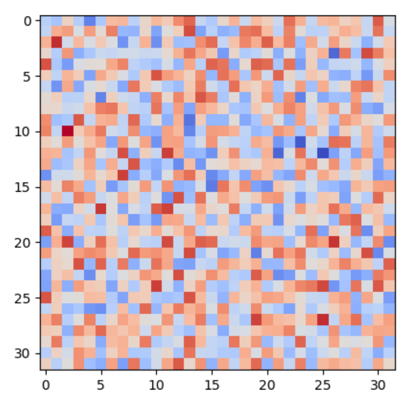

Figure 15: Visualized $\theta's$ obtained from training on 70 images per actor (Alec Baldwin and Steve Carell)

Part 5.

For this part, training sets of various size from
size = [1,5,10,15,20,25,30,35,40,45,50,55,60,65,70,75,80,85,90,95,100]
images per actor had been made. The validation set of the actors that are in act has 10 images per actor and the validation set of the actors that are not in act has 50 images per actor. The reason is that for the actors in act, since many of the pictures are already used in training, the validation set could not have more than 10 images per actor simply because the pictures were not enough to do so. All the performances have been saved into an array and plotted afterwards. Below, are the results.

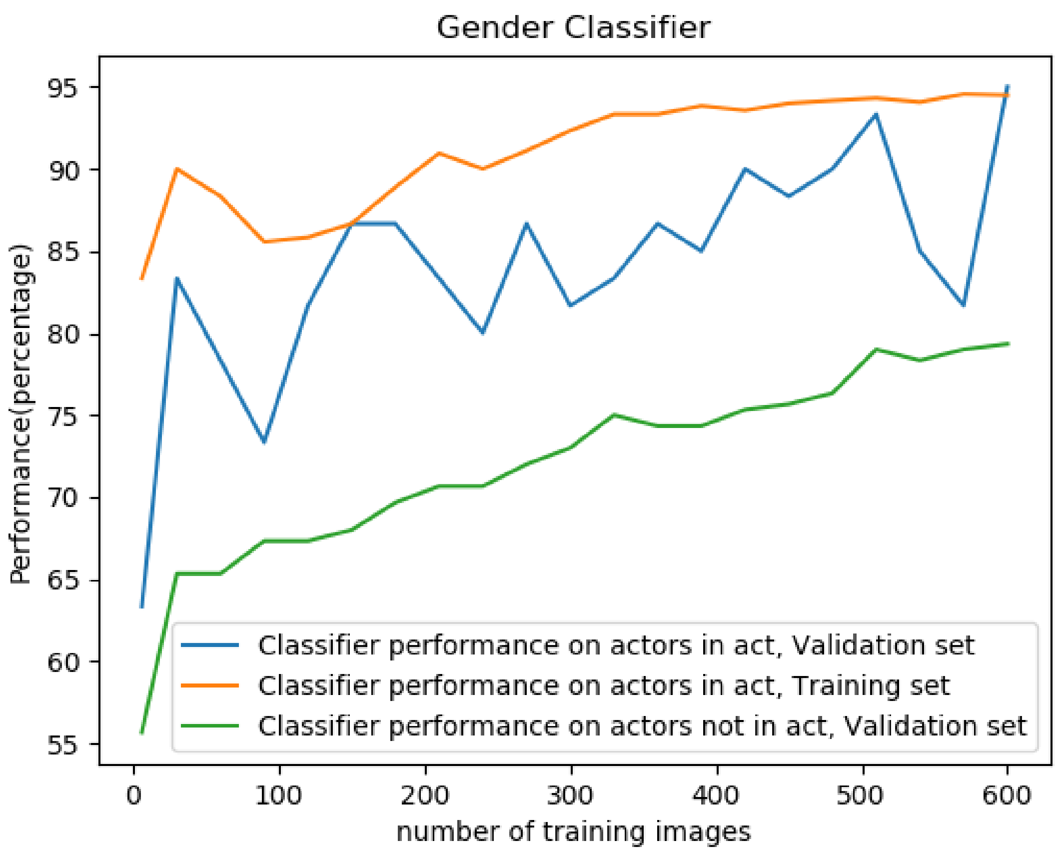

Figure 16: Gender Classifier performance, trained on actors in ’act’.

Note that as the size of the training set increases, the performance of the classifier on the set of actors that are not in act increases. At the beginning, where we are only using 1 to 10 pictures for each actor, total of 6 to 60 images, the classifier does not perform really well in gender classifying on the set of actors that are not in the act set(about 65%) but it does well on the training set(90%). This demonstrates that we are overfitting and our classifier is not trained well.
As the number of images per actor increase in our training set, it performs better on the validation set of actors in act and on the validation set of actors that are not in act.

Another observation is that the classifier's performance is always the best on the training set, followed by the validation set of the same actors in the training set, followed by the validation set of the actors that were not in act and it completely makes sense. The algorithm was trained by 6 actors in act, and obviously it must perform better in classifying them than others that it did not interact with.

Part 6.

a)

let i = p. Therefore, the cost function becomes: $J(\theta) = (\theta^T x^{(p)} - y^{(p)})_q^2)$

Now, $\theta = \begin{bmatrix} \theta_{0-0} & \theta_{0-1} & \dots & \theta_{0-k} \\ \theta_{1-0} & \theta_{1-1} & \dots & \theta_{1-k} \\ \vdots & \vdots & \ddots & \vdots \\ \theta_{1024-0} & \theta_{1024-1} & \dots & \theta_{1024-k} \end{bmatrix}$

$x^{(p)} = \begin{bmatrix} 1 \\ x_{p-1} \\ \vdots \\ x_{p-1024} \end{bmatrix}$

$y^{(p)} = \begin{bmatrix} y_{p-0} \\ y_{p-1} \\ \vdots \\ y_{p-k} \end{bmatrix}$

If we multiply the quantities above, we would obtain: $\begin{bmatrix} ((\theta_{0-0} + \theta_{1-0}x_{p-1} + \ldots + \theta_{1024-0}x_{p-1024}) - y_{p-0})^2 \\ \vdots \\ ((\theta_{0-k} + \theta_{1-k}x_{p-1} + \ldots + \theta_{1024-k}x_{p-1024}) - y_{p-k})^2 \end{bmatrix}$

By selecting the q's element of the result, we would get:

$((\theta_{0-q} + \theta_{1-q}x_{p-1} + \ldots + \theta_{1024-q}x_{p-1024}) - y_{p-q})^2$

b)
First let us define the dimensions of each quantity.
x is $1025 \times m$, θ is $1025 \times k$ and therefore θ^T is $k \times 1025$. y is $k \times m$. Our J(θ) must then be $k \times m$.

We need to take the derivative of all the $\theta p - q$. Using the result of part a, we can say that:

$$\partial J/\partial\theta_{ij} = 2(\theta_{0-q} + \theta_{1-q}x_{p-1} + \ldots + \theta_{1024-q}x_{p-1024} - y_{p-q})(x_{p-q}) \tag{2}$$

Therefore, $\partial J/\partial\theta$ would be:

$$\sum_p(\sum_q 2(\theta_{0-q} + \theta_{1-q}x_{p-1} + \ldots + \theta_{1024-q}x_{p-1024} - y_{p-q})(x_{p-q})) \tag{3}$$

$$= 2\sum_p(\sum_q(\theta^T_{row=p}x_{column=p} - y_{p-q})x_{p-q}) \tag{4}$$

$$= 2\sum_p X_{row=p}(\theta^T_{row=p}X_{column=p} - y_{column=p}) \tag{5}$$

$$= 2X(\theta^T X - Y) \tag{6}$$

c)

```
def f_mat(x,y,theta):
    '''
    cost function
    input: training set and their label (x,y = 0 or 1) and thetas
    output: cost function
    '''
    #return sum(sum((np.matmul(theta.T, x)-y)**2,0))
    #return sum(sum((np.matmul(theta.T, x)-y)**2,1))
    return sum((np.matmul(theta.T, x)-y)**2,1)

def df_mat(x,y,theta):
    '''
    gradient function
    input: training set and their label (x,y = 0 or 1) and thetas
    output: derivative of cost function
    '''
    return 2*(np.matmul(x,(np.matmul(theta.T,x)-y).T))
```

d)

In order to show that the vectorized gradient function works, a finite difference function was written and tested on the samples. Then, another gradient was calculated using the vectorized method. At the end, to compare the two, the following formula was used:

$$\max\left(\frac{\nabla_{vectorized} - \nabla_{finitedifference}}{\nabla_{vectorized}} \times 100\%\right)$$

The result was as follows:

```
vectorized gradient is:
[[11448.19607843 11448.19607843]
 [ 3929.09570165  3930.68001538]
 [ 4090.25986928  4091.88339869]
 ...
 [ 4217.80887351  4214.36573626]
 [ 5232.784406    5227.60009227]
 [ 5900.66965013  5895.26572857]]
gradient using finite difference is:
[[11448.19618203 11448.19622859]
 [ 3929.09571528  3930.68012781]
 [ 4090.2598761   4091.88345075]
 ...
 [ 4217.80883335  4214.36578035]
 [ 5232.78438486  5227.60013118]
 [ 5900.66965669  5895.26575059]]
maximum percentage difference between vectorized gradient and finit
e difference approximation with h of 1e-05 is 4.63814e-06%
```

Figure 17: The calculated gradient using the finite difference method and the vectorized method. Note that they all look similar to each other to 4 decimal places and the maximum % difference is about $4.6 \times 10^{-6}\%$

In this part, I orginally chose h as 1, and all the values very close and the maximum difference was 0.09% but each number was off by factors of 10. To get them to match each other to 3 decimal places, I used trial and error and reduce the value of h each time until they matched. This served as a double check on the vectorized gradient descent function, meaning that we are now able to use it safely.

Part 7.

I used learning rate of 0.0000001 and I initialized all the θs as 0 becuase of its good performance on previous parts. The learning rate was originally high, and as a result, θs did not converge to a value and I used trial and error and in each trial, I reduced it by a factor of 10 until it converged. The training set had 80 images per actor (total of 240 images) and the validation set had 30 images per actor (total of 180) images and the performance is as follows:

```
performance on validation set is 75%
performance on training set is 84.2857%
```

Figure 18: The results obtained from the new labeling algorithm.

Note that in this part, since we have 6 labels, we need to map 5 of them to zero and one of them to 1 so that we can predict the label of the image. I divided all the labels by the maximum of them, to obtain a value of 1 and some decimal values. By type casting them to integers, I was able to get a label, consisted of 5 zeros and 1 one, and hence I was able to match and image with a predefined label.

For this part, the parameters of the gradient descent were kept the same as the ones before, error $= 1^{-5}$ and maximum iteration = 30000. Other combinates were tested and the table below are the results:

Error Bound	Maximum Iterations	Validation Set Performance	Training Set Performance
1×10^{-5}	30000	75%	84.2857%
1×10^{-7}	30000	75%	84.2857%
1×10^{-3}	30000	29.4444%	33.5714%
1×10^{-4}	30000	38.333%	47.381%

As a result of the above trials, the error bound was set to 1×10^{-5} for the good performance and fast computation.

Error Bound	Maximum Iterations	Validation Set Performance	Training Set Performance
1×10^{-5}	30000	75%	84.2857%
1×10^{-5}	300000	75%	86.9048%
1×10^{-5}	3000	54.4444%	68.8095%

As a result of the above trials, the maximum number of iterations was set to 30000 for the good performance and fast computation.

Part 8.

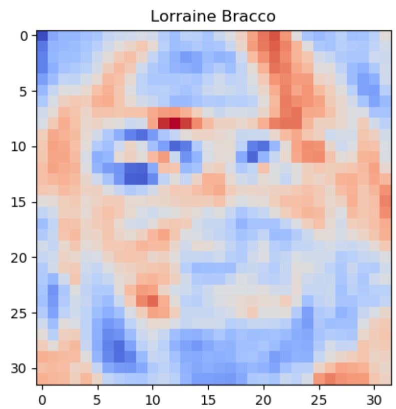

Figure 19: Visualized θs of Bracco

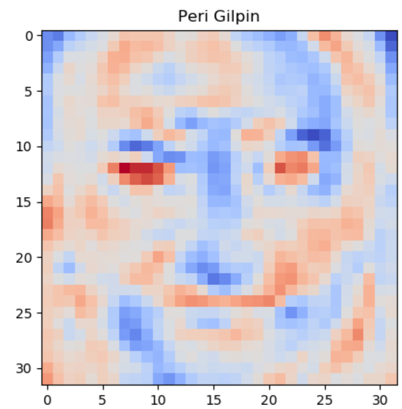

Figure 20: Visualized θs of Gilpin

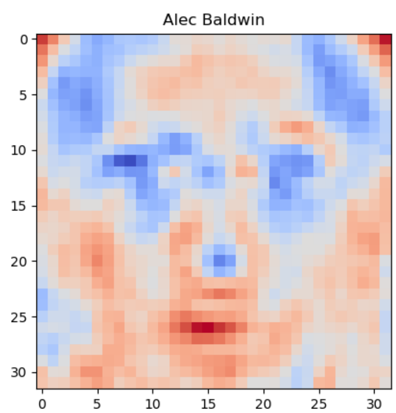

Figure 21: Visualized θs of Baldwin

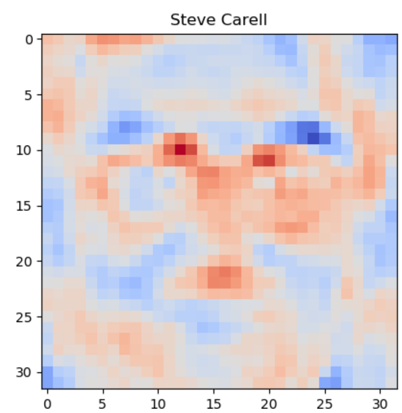

Figure 22: Visualized θs of Carell

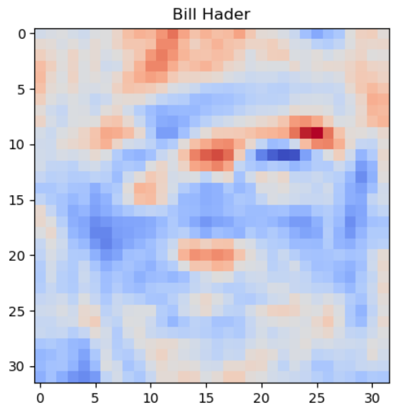

Figure 23: Visualized θs of Hader

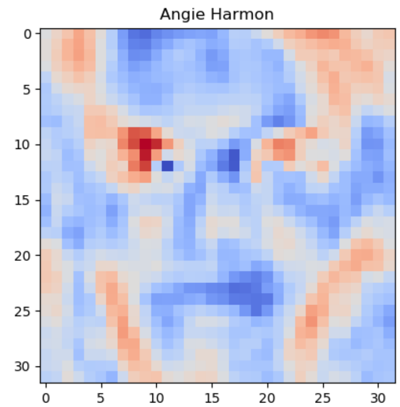

Figure 24: Visualized θs of Harmon

As one can see from the pictures above, they all infer some features of each actors' face. For example, Bill Hader has a straight hair and it's identifiable in his visualized θs. In many of Steve Carell images, he is wearing glasses and it's shown in his visualized θs by showing large and wide eyes which are the results of the computer thinking that the glasses are actually his eyes.