CSC411: Assignment1

Due on Friday, February 3, 2016

Xiaodi Lu

February 4, 2017

Part 1

Dataset description
The data-set consists of 1918 various size of colored uncropped images of a few actors and actresses, and grayscale images of the same data set but resized and cropped out the faces to 32 × 32-pixel. The cropped images of actors and actresses are random and the faces of theirs can be a side face, or there are some hair covering it, or the face is not completely in the image and there is considerable variation in the appearance of the faces. However, examining the dataset, it appears that there are more nicely cropped faces than other images, and most of the face are facing the front. Most faces do not align with each other, and the faces can have various emotions on the faces and facing all sort of directions. A random sample of 3 "cropped out faces"s is shown in Figure 1, 2, and 3.

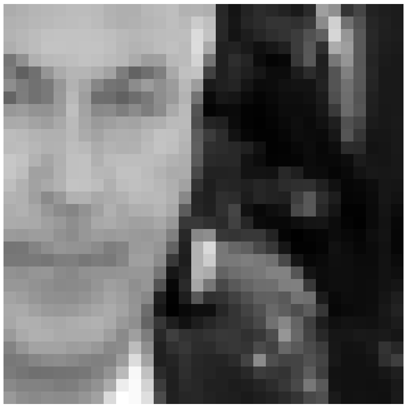


Figure 1: This face is not cropped properly leaving part of the face out of the picture

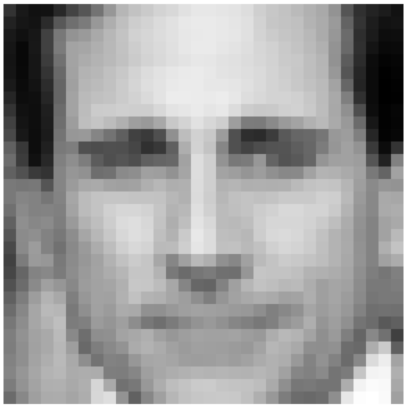


Figure 2: This face is cropped perfectly, no unnecessary background in the picture

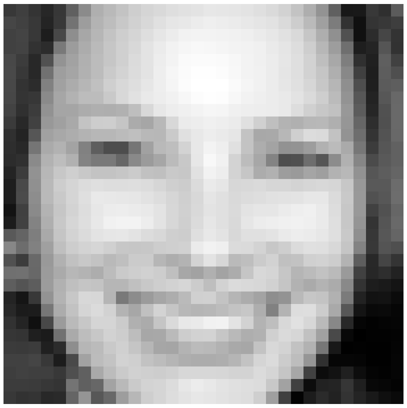


Figure 3: This face is also cropped perfectly and the person is smiling in the picture, unlike the others.

The uncropped dataset images might be taken when people are not looking in the camera, or smilling at the camera, or can be taken for a cover photo of a magazine, which there could be letters and other stuff covering some part of the face. And the faces usually do not align with each other. A random sample of 3 "cropped out faces"s is shown in Figure 4, 5, and 6.

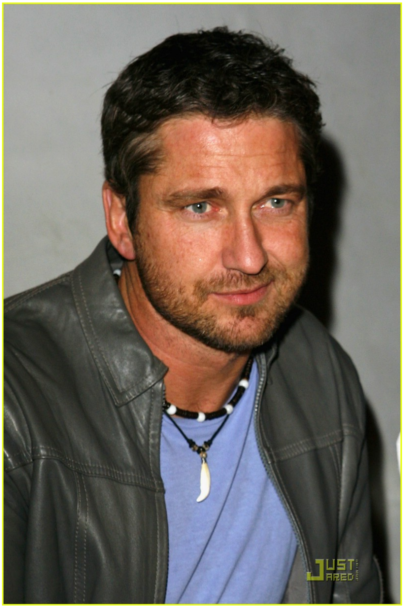


Figure 4: This face is not cropped properly leaving part of the face out of the picture

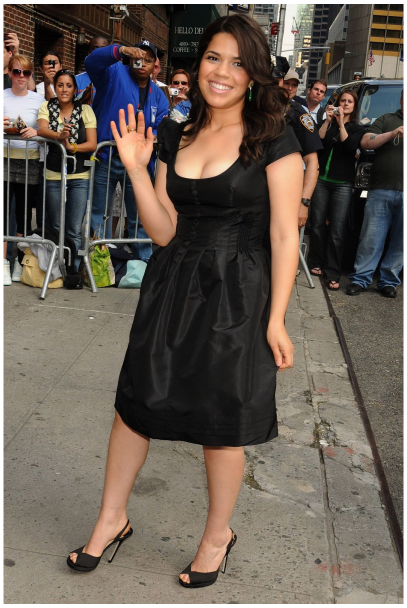


Figure 5: This face is cropped perfectly, no unnecessary background in the picture

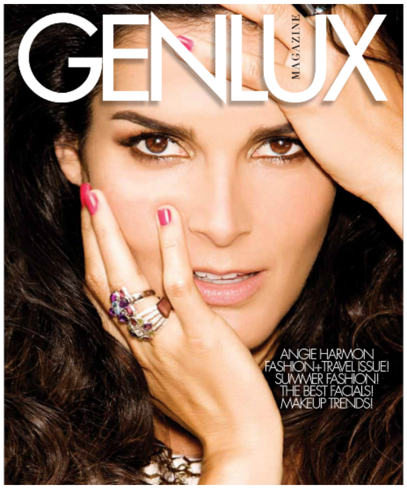


Figure 6: This face is also cropped perfectly and the person is smiling in the picture, unlike the others.

Part 2

Separate the dataset into three non-overlapping parts

First, I created a folder for each actor, then I take all images from the cropped folder and separate them by actors using the image's file name and put it in the actor's list in the dictionary, and random each list. Then I loop over each actor's image list in the dictionary to copy the first 100 images of each list into the actor's training set subfolder, the next 10 images into the test set subfolder, and the next 10 images to the validation set subfolder, and the folders are all created in the actor's folder.

Part 3

Build a classifier to distinguish pictures of Bill Hader form pictures of Steve Carell
The cost function I used is the following function f

```
def f(x, y, theta):
    return 0.0025*sum( (y - np.dot(x, theta)) ** 2)
```

which is this function put into code $\frac{1}{2m}\sum_i(\theta^T x^{(i)} - y^{(i)})^2$

On the training set:
Value of the cost function: 0.047
Percentage of images that were correctly classified: 0.910
On the validation set:
Value of the cost function: 0.003
Percentage of images that were correctly classified: 0.900

We used the function *grad_descent* to compute the classifier and the function $class_or_correct$ to construct the data.

```
def df(x, y, theta):
    return -0.005 * np.dot(x.T, (y - np.dot(x, theta)))

def grad_descent(f, df, x, y, init_t, alpha):
    t = init_t.copy()
    max_iter = 30000
    iter  = 0
    while iter < max_iter:
        t -= alpha*df(x, y, t)
        iter += 1
    return

def class_or_correct(size, set, flag):
    '''flag = 1 :classifiy
       flag = 0 :correction
    '''
    x = np.empty(shape=[0, 1024])
    y = np.array([[1 for v in range(size/2)]])
    y1 = np.array([[0 for p in range(size/2)]])
    y = np.concatenate((y, y1), 1)
    y = np.reshape(y, (size,1))
    one = np.array([[1 for q in range(size)]])
    one = np.reshape(one, (size,1))

    hader = os.listdir("hader/" + set)
    carell = os.listdir("carell/" + set)

    for i in range(size):
        if (i < size/2):
            im = imread("hader/" + set + hader[i])[:,:,0]
        else :
            im = imread("carell/"+ set + carell[i-size/2])[:,:,0]
        im = np.reshape(im, (1, 1024))
        x = np.concatenate((x, im), 0)

    x = np.concatenate((one, x), 1)
    if flag:
        t = np.zeros([1025, 1])
        return grad_descent(f, df, x, y, t, 5*1e-10)
    else:
        correction = 0
        expect = np.dot(x, new_t)

        for i in range(size):
            if i < size/2:
                if expect[i] >= 0.5:
                    correction += 1
            else:
                if expect[i] < 0.5:
                    correction += 1
        print"cost: %.3f\n" %(f(x, y, new_t))
        print"Percentage: %.3f\n" % (correction/float(size))

def part3(size):
    global new_t
    new_t = class_or_correct(size, "trainning_set/", 1)
    class_or_correct(size, "trainning_set/", 0)
    class_or_correct(10, "validation_set/", 0)
```

I made sure that the image pixel array for Hader would be labeled as 1 and Carell would be labeled as 0. And the two actors are in their half of the matrix in x. And the alpha shouldn't be too large. When I set the alpha too large, the t in the *grad_descent* function would jump further and further away from the minimum point that we want. When that happens I kept on trying to descrease alpha and stop at the point where the cost is considerably small. The iteration being too large could result in overfitting, so I kept the *max_iteration* not too large and not too small either, otherwise the performance would be bad.

Part 4

Display the θs that you obtain by training using the full training dataset, and by training using a training set that contains only two images of each actor

Hypothesis Function: $h_\theta(x) = \theta_0 + \theta_1 x_1 + ... + \theta_n x_n$

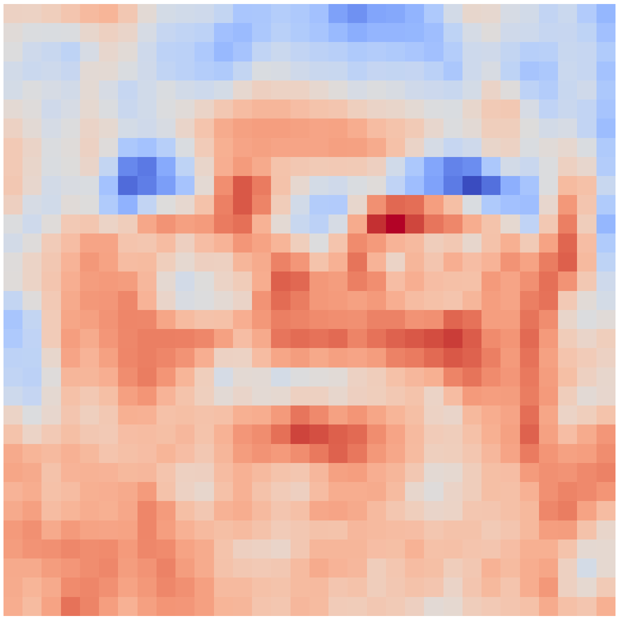


Figure 7: The theta image using full training dataset

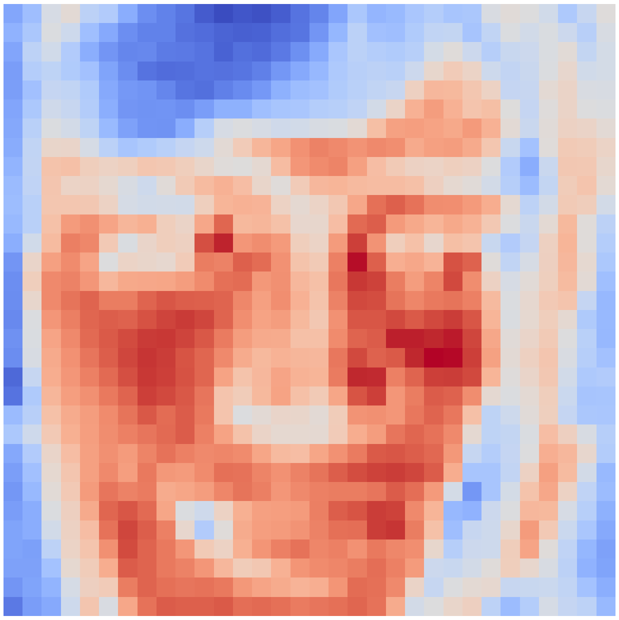


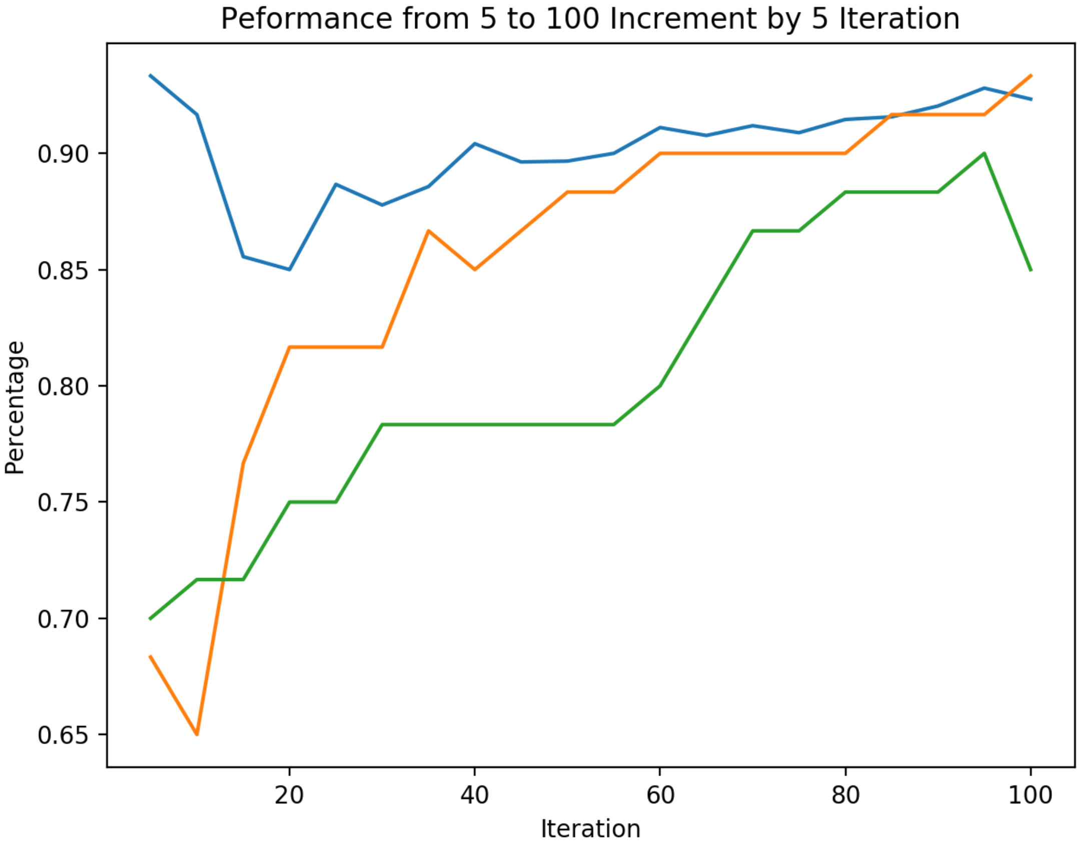
Figure 8: The theta image using a training subset of two images each actor

Part 5

Demonstrate Overfitting

```
act =[’Fran Drescher’, ’America Ferrera’, ’Kristin Chenoweth’, ’Alec Baldwin’,
                                                    ’Bill Hader’, ’Steve Carell’]

act_test = [’Gerard Butler’, ’Daniel Radcliffe’, ’Michael Vartan’, ’Lorraine Bracco’,
                                                    ’Peri Gilpin’, ’Angie Harmon’]
```

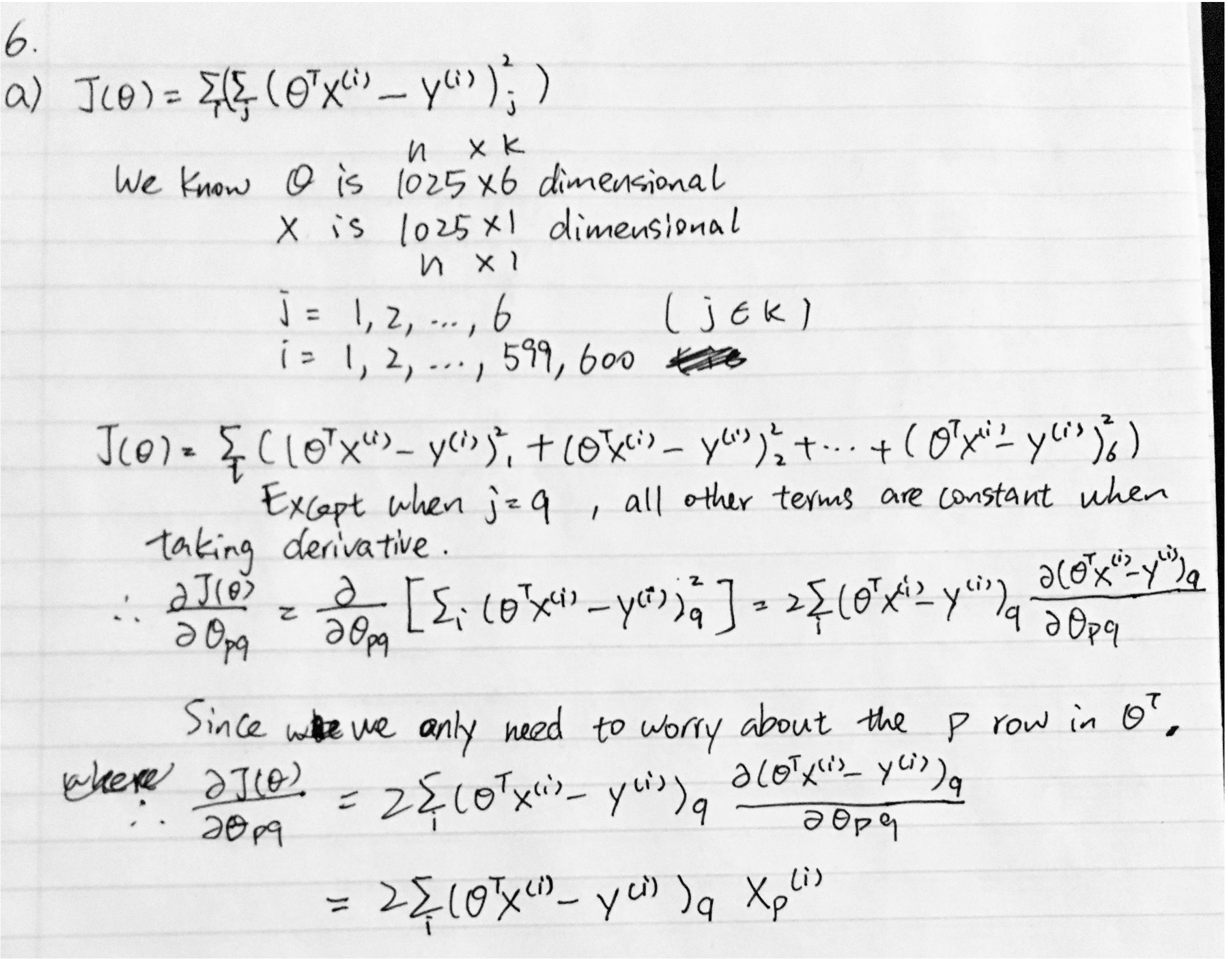


Note: the $x - axis$’s label is wrong it should be the training example size. The blue line is the performance percentage of the classifier on the training set of the actors in *act*. The yellow line is the performance percentage of the classifier on the validation set of the actors in *act*. The green line is the performance percentage of the classifier on the test set of the actors in *act_test*. With the classifier built by each iteration.

The performance of the *act_test* is increasing as iteration increase except at the last point where it went down which is a result of overfitting.

Part 6

A different way of classifying inputs



6.

a) $J(\theta) = \sum_i (\sum_j (\theta^T x^{(i)} - y^{(i)})_j^2)$

We know θ is $\overset{n}{1025} \times \overset{k}{6}$ dimensional

x is $\underset{n}{1025} \times \underset{1}{1}$ dimensional

$j = 1, 2, \ldots, 6$ $\quad (j \in k)$

$i = 1, 2, \ldots, 599, 600$

$J(\theta) = \sum_i ((\theta^T x^{(i)} - y^{(i)})_1^2 + (\theta^T x^{(i)} - y^{(i)})_2^2 + \cdots + (\theta^T x^{(i)} - y^{(i)})_6^2)$

Except when $j = q$, all other terms are constant when taking derivative.

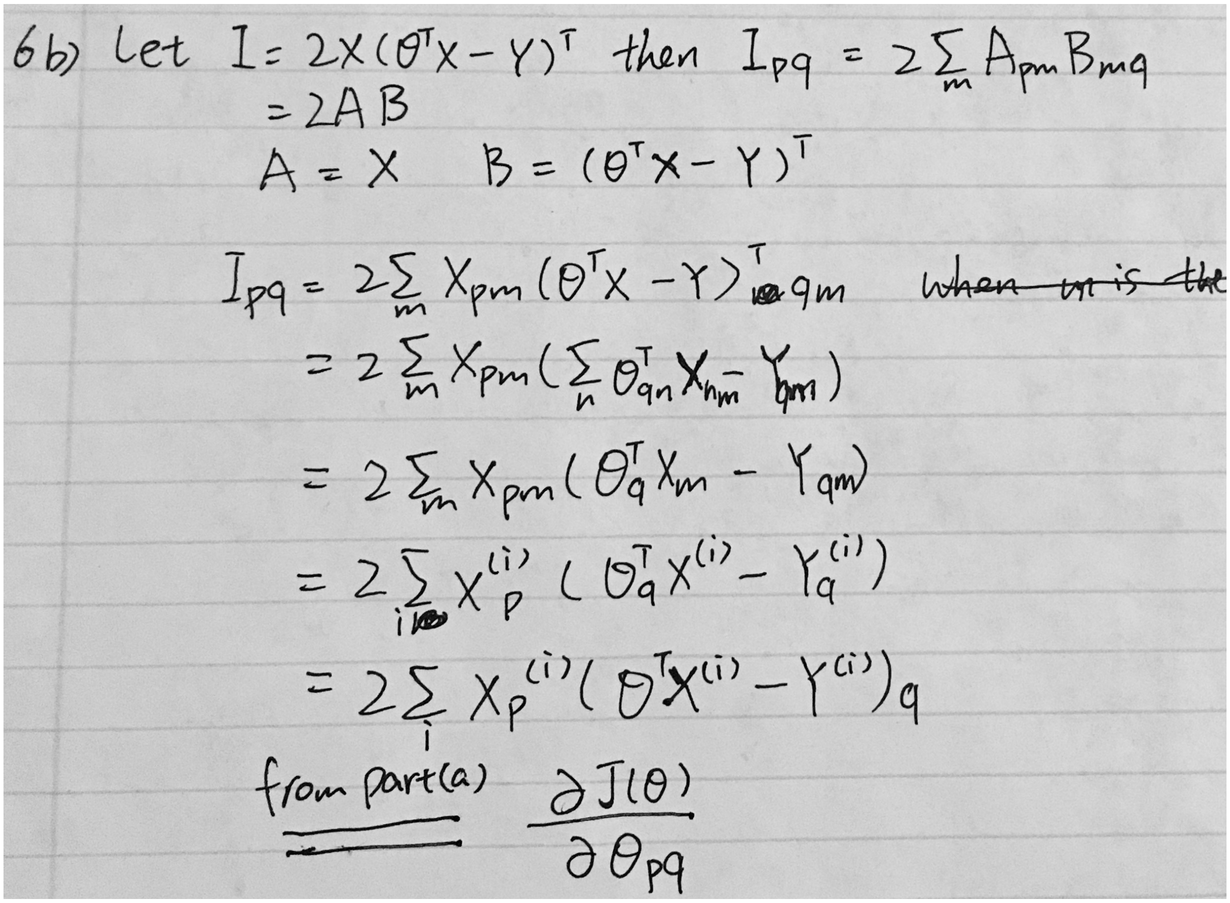
$$\therefore \frac{\partial J(\theta)}{\partial \theta_{pq}} = \frac{\partial}{\partial \theta_{pq}} \left[\sum_i (\theta^T x^{(i)} - y^{(i)})_q^2 \right] = 2 \sum_i (\theta^T x^{(i)} - y^{(i)})_q \frac{\partial (\theta^T x^{(i)} - y^{(i)})_q}{\partial \theta_{pq}}$$

Since ~~whe~~ we anly need to worry about the p row in θ^T,

~~where~~

$$\therefore \frac{\partial J(\theta)}{\partial \theta_{pq}} = 2 \sum_i (\theta^T x^{(i)} - y^{(i)})_q \frac{\partial (\theta^T x^{(i)} - y^{(i)})_q}{\partial \theta_{pq}}$$

$$= 2 \sum_i (\theta^T x^{(i)} - y^{(i)})_q \, x_p^{(i)}$$



6b) Let $I = 2X(\theta^T X - Y)^T$ then $I_{pq} = 2\sum_m A_{pm} B_{mq}$

$= 2AB$

$A = X \quad B = (\theta^T X - Y)^T$

$I_{pq} = 2\sum_m X_{pm} (\theta^T X - Y)^T_{qm}$ ~~when m is the~~

$= 2\sum_m X_{pm} (\sum_n \theta^T_{qn} X_{nm} - Y_{qm})$

$= 2\sum_m X_{pm} (\theta^T_q X_m - Y_{qm})$

$= 2\sum_i X^{(i)}_p (\theta^T_q X^{(i)} - Y^{(i)}_q)$

$= 2\sum_i X_p^{(i)} (\theta^T X^{(i)} - Y^{(i)})_q$

from part(a) $= \dfrac{\partial J(\theta)}{\partial \theta_{pq}}$

X is $n \times m$ dimension
θ is $n \times k$ dimension
Y is $k \times m$ dimension
θ^T is $k \times n$ dimension
n is the number of pixels
m is the number of training examples
k is the number of possible labels

6.c
The cost function from Part 6 and the vectorized gradient function in Python.

```
def cost(x, y, theta):
    #x = vstack( (ones((1, x.shape[1])), x))
    return sum( (np.dot(theta.T,x) - y) ** 2)

def derive(x, y, theta):
    #x = vstack( (ones((1, x.shape[1])), x))
    return 2 * np.dot(x, (np.dot(theta.T, x) - y).T)

def vectorized_grad_descent(f, df, x, y, init_t, alpha):
    t = init_t.copy()
    max_iter = 30000
    iter  = 0
    while iter < max_iter:
        t -= alpha*df(x, y, t)
        # if iter % 500 == 0:
        #     print "Iter", iter
        #     print "x = (%.2f, %.2f, %.2f), f(x) = %.2f" % (t[0], t[1], t[2], f(x, y, t))
        #     print "Gradient: ", df(x, y, t), "\n"
        iter += 1
    return t
```

6.d
The cost function from Part 6 and the vectorized gradient function in Python.

```
def part6(p, q):
    h = 5*1e-11
    x = np.ones([1025, 600])
    y = np.ones([6, 600])
    theta = np.ones([1025, 6])
    theta_h = np.ones([1025, 6])
    theta_h[p, q] += h
    print (cost(x, y, theta_h) - cost(x, y, theta))/h
    print derive(x, y, theta)[p, q]
```

I printed out the derivative calculated using the finite difference and the dervie function on several function call $(p, q) = (3, 5)$ finite difference output is 1230239.86816 derive function output is 1228800.0 $(p, q) = (6, 1)$ finite difference output is 1230239.86816 derive function output is 1228800.0 $(p, q) = (16, 4)$ finite difference output is 1230239.86816 derive function output is 1228800.0

Part 7

Perform face recognition

Alpha is $5*1e-11$. I first tried setting alpha to $5*1e-10$, but the t in the function kept jumping further and further away from the minimum point. On second try, I used a smaller alpha $5*1e-11$, and it worked out fine.

The *max_iteration* was not changed from part 5, since the performance percentage were high enough on both set, which implies overfitting did not occur.

X is our dataset the first entry on every column is the bias and the rest is its pixel array we got from the image. Every column is a training example therefore the dimension is 1025×600.

Y is the expected output from the function using our classifier, in other words, we are hoping the theta we get can help us recognize every person, therefore it is $[1,0,0,0,0,0]$ for the first person, and $[0,1,0,0,0,0]$ would be the label for the second person, and $[0,0,1,0,0,0]$ for the third, $[0,0,0,1,0,0]$ for the fourth person, and $[0,0,0,0,1,0]$ for the fifth person, $[0,0,0,0,0,1]$ for the last person, with each person having 100 training examples, the dimension of Y is 6×600

On the training set:
Percentage of images that were correctly classified: 0.998
On the validation set:
Percentage of images that were correctly classified: 0.850

Part 8

Reconstructing images where salt-and-pepper noise was applied

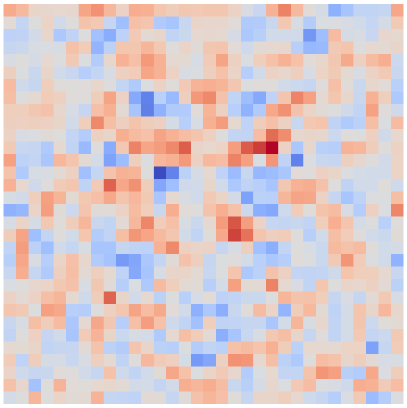


Figure 9: Fran Drescher

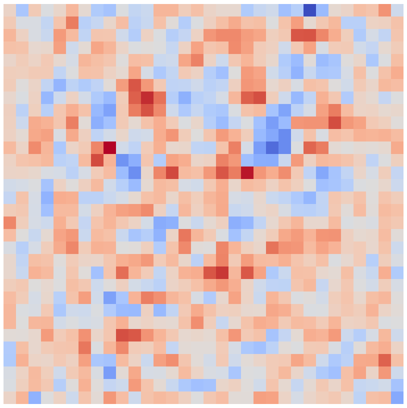


Figure 10: America Ferrera

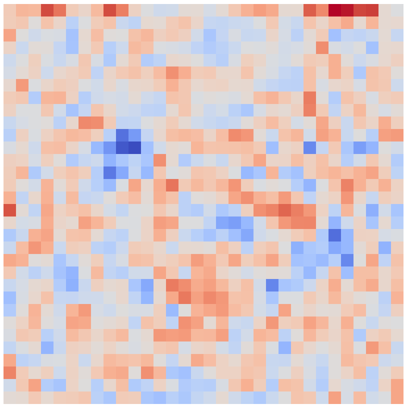


Figure 11: Kristin Chenoweth

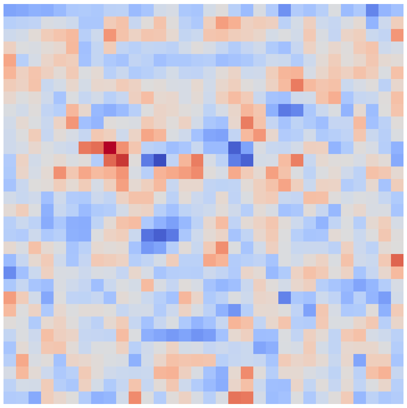


Figure 12: Alec Baldwin

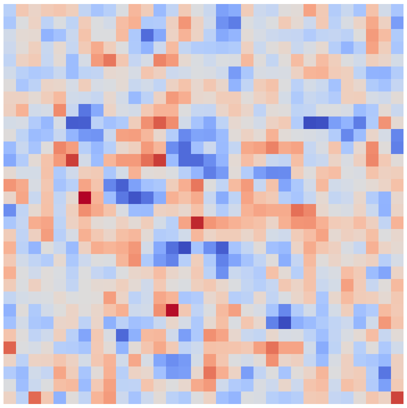


Figure 13: Bill Hader

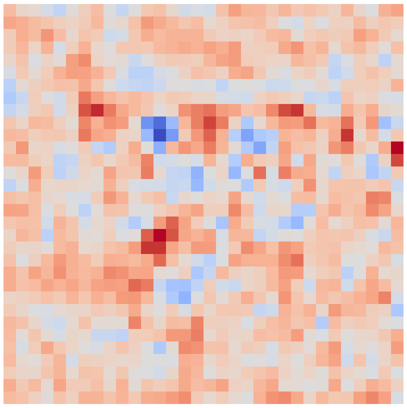


Figure 14: Steve Carell