# CSC411: Assignment #1

Due on Monday, January 30, 2018

**Cameron Alizadeh**

January 31, 2018

# Part 1: Description of Dataset

The dataset is comprised of photos of famous actors' faces taken from the FaceScrub dataset. The subset of actors used is as follows: Alec Baldwin, Bill Hader, Daniel Radcliffe, Gerard Butler, Michael Vartan, Steve Carell, Lorraine Bracco, Kristin Chenoweth, Fran Drescher, America Ferrera, Peri Gilpin, Angie Harmon. Each actor has approximately 90-130 photos in the dataset, with some duplicates. In the interest of saving time, duplicate photos were not detected and removed.

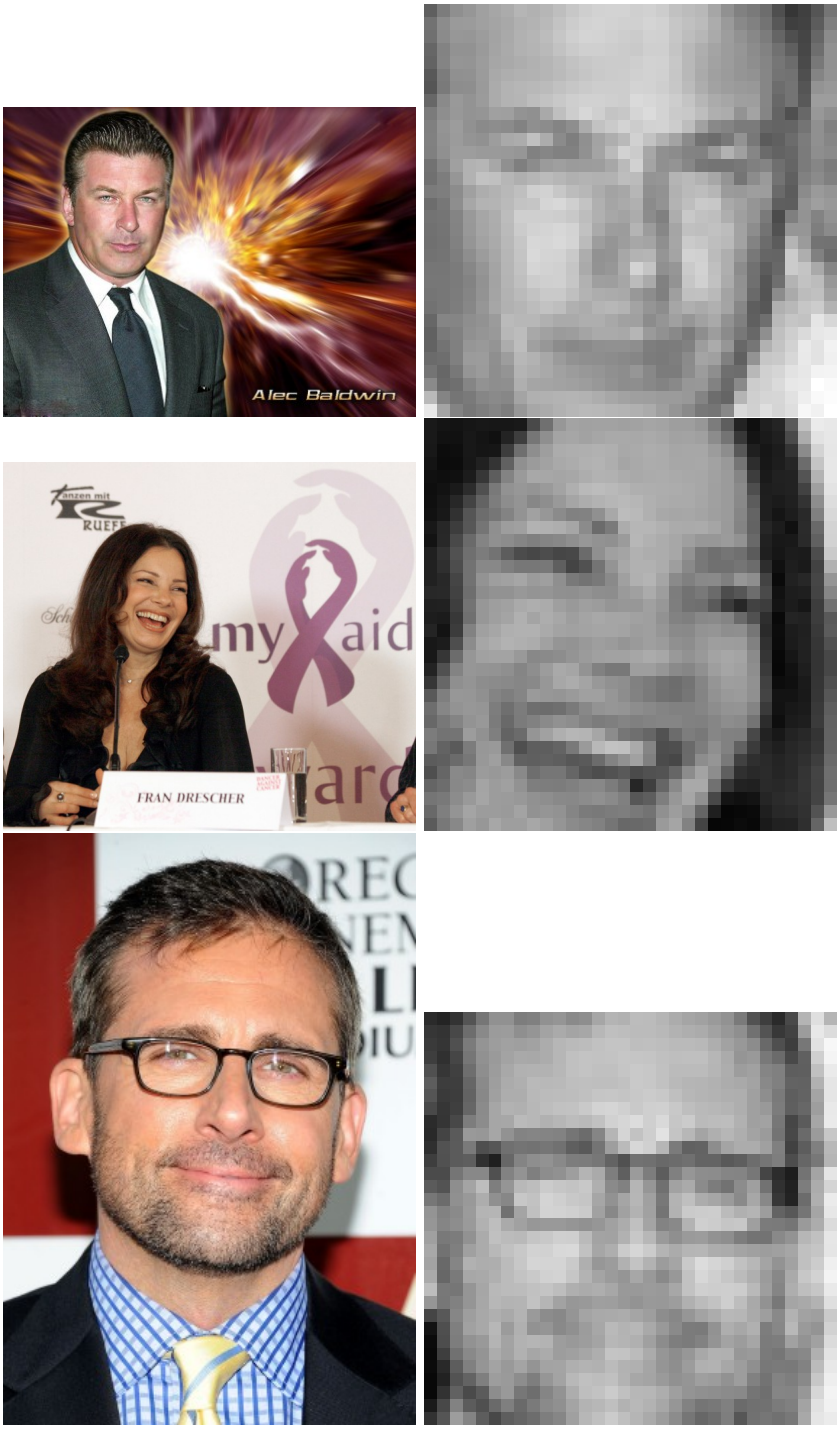
Figure 1: Sample Raw Images Downloaded from the FaceScrub Dataset. Note the variety in photo style and type. Left: Alec Baldwin, Centre-left: Peri Gilpin, Centre-right: Kristin Chenoweth, Right: Michael Vartan


The data was downloaded with script provided in download_images.py. The script computes the hashcode of the downloaded file and compares it to the hash given in the faceScrub dataset to ensure data integrity. This process resulted in over 90 photos for each actor except for Peri Gilpin, of whom only 87 photos were available. Thus, the dataset was augmented with 3 hand-picked photos of Peri Gilpin, available at the following URLS: https://www.famousbirthdays.com/headshots/peri-gilpin-4.jpg, https://www.famousbirthdays.com/headshots/peri-gilpin-7.jpg, https://res.cloudinary.com/allamerican/image/fetch/t_face_s270/https://speakerdata2.s3.amazonaws.com/photo gilpin.jpg

Figure 2: The Three Extra Gilpin Photos Used.

Upon download and hash confirmation, the data was processed by clean_images.py. The images were cropped according to the bounding box specified in the FaceScrub dataset, resized to 32 x 32 pixels, and converted to greyscale. The extra three Gilpin photos specified above were resized and converted to greyscale, but not cropped, as they are already square. Examples of raw photo downloads and their cleaned, converted equivalents are shown below.

Figure 3: Dataset Photos Before and After the Cleaning Process. Raw images are shown on the left, while clean images are shown on the right. Top: Alec Baldwin, Middle: Fran Drescher, Bottom: Steve Carell



## Part 2: Dataset Separation

The dataset was separated into three non-overlapping segments: the training set (70 images per actor), the validation set (10 images per actor), and the test set (10 images per actor). The algorithm used is as follows: image file names associated with an actor were stored as strings in a python list. These lists were "shuffled" into a random permutation using random.shuffle() with a seed of 42 for reproducibility. Then, for each actor, filenames with indices 0-69 had their associated image copied into a new folder "/{actor}/{train}/", indices 70-79 into "/{actor}/{validation}/", and indices 80-89 into "/{actor}/{test}/". If the actor had more than 90 clean images available, the rest were discarded. The random.shuffle() function ensures pseudorandom separation of the photos into sets, as desired.

Figure 4: The Core For Loop in split_datasets.py, Used to Create Train, Validation, and Test Sets. Note the use of random.shuffle(), as specified.

```
    for actor in files_list:
        actors_list = files_list[actor]
        random.shuffle(actors_list)
        for i in range(70):
            copyfile("./data/clean/" + actors_list[i], "./data/organized/" + actor + "/train/" +
                                                                actors_list[i])
        for i in range(70, 80):
            copyfile("./data/clean/" + actors_list[i], "./data/organized/" + actor + "/validation
                                                                /" + actors_list[i])
        for i in range(80, 90):
            copyfile("./data/clean/" + actors_list[i], "./data/organized/" + actor + "/test/" +
                                                                actors_list[i])
```

## Part 3: Linear Regression Classifier

A classifier using linear regression was built to distinguish images of Alec Baldwin from Steve Carell. The cost function, shown below, is the quadratic cost function. A hypothesis is computed as the matrix product of the feature vector and the training set. This hypothesis is compared with the training set labels, and the squared error is taken for each example in the training set. Finally, the average squared error is returned.

Figure 5: The Cost Function for Linear Regression

```
def linear_regression_cost_function(x, theta, y):
    m = x.shape[0]
    h = np.matmul(x, theta)
    cost = 0.5 / m * np.sum(np.square(h - y))
    return cost
```

Images were imported into Python and converted to numpy arrays using matplotlib.image.imread(). These numpy arrays were reshaped from 32 x 32 to 1 x 1024. An additional [1] was added to each array as a bias feature, changing the overall dimensions to 1 x 1025. These image arrays were then stacked to produce a 140 x 1025 training set matrix. The gradient descent algorithm was then run, with an experimentally derived learning rate of alpha = 0.005. Cost histories of both the training set and validation set were recorded, and gradient descent was stopped when validation errors began to increase in order to avoid overfitting. To compute the gradient, a function that returns the gradient of the cost function was used.

As mentioned above, the learning rate was experimentally determined to be alpha=0.005. However, performance does not significantly change for smaller learning rates, even learning rates as low as alpha=0.0001. This is because the max_iterations parameter in the gradient descent code is quite high, allowing convergence for even small learning rates. However, small learning rates come at the detriment of requiring more time to converge. Large learning rates produce a more significant error: divergence. For learning rates of 0.006 and greater, gradient descent failed to converge. Costs increased with each successive iteration, and as such, the overfitting clause causes gradient descent to exit immediately. For more information, see the gradient_descent() function in linear_regression.py

The final performance of the classifier on the test set was an accuracy of 85%. The performance on the validation set and training set were 95% and 99.3% respectively. The final cost of the training set was 0.0157, and the cost of the validation set was 0.1779. Sample code to compute the output of the classifier is shown below, as well as the code used to generate performance metrics.

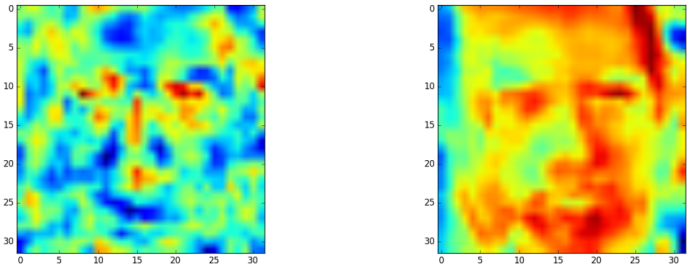
Figure 6: Code Used to Compute the Output of the Classifier. calculate_performance() calculates the accuracy of the classifier on a set of images, while make_prediction() returns a prediction made on a single photo.

```
def calculate_performance(x, theta, y):
    m = x.shape[0]
    h = np.matmul(x, theta)
    results = h * y
    num_success = (results > 0).astype(int).sum()
    return float(num_success) / m

def make_prediction(x, theta):
    h = np.matmul(x, theta)
    if h > 0:
        return "Alec Baldwin"
    else:
        return "Steve Carell"
```

## Part 4: Visualizing Gradient Descent

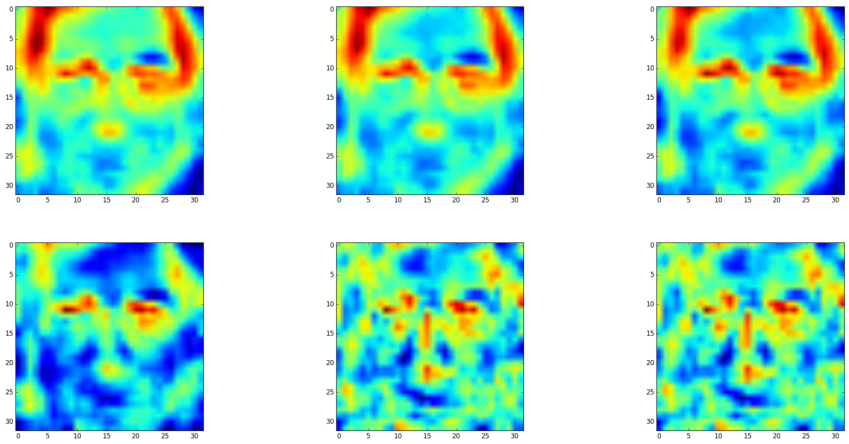
The output of the logistic regression classifier can be visualized as an image. By reshaping the output theta array from 1x1024 (without the bias feature) to 32 x 32, the array can be plotted as an image, as shown below.

Figure 7: Output Theta Array Visualization for Different Sized Training Sets. Left: visualization with full training set, Right: visualization using two images of each actor



Stopping the gradient descent process early yields results similar to those seen using the reduced training set. This is visualized in the images below.

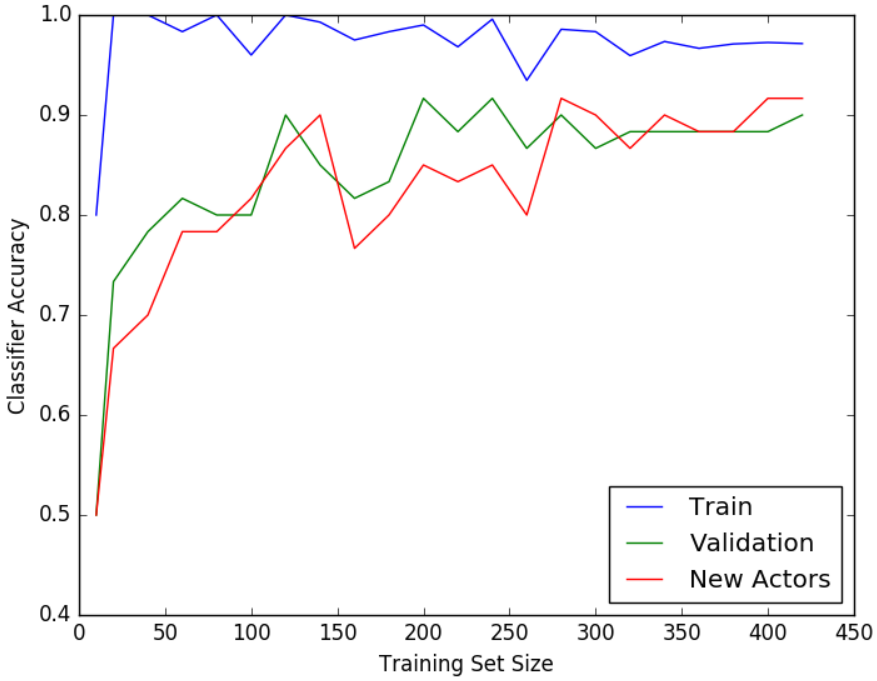
Figure 8: Output Theta Array Visualization Showing the Effect of Stopping Early. From left to right, top to bottom, photos indicate results after 1 iteration, 10 iterations, 100 iterations, 1000 iterations, 10000 iterations, and the full 11470 iterations



## Part 5: Demonstrating Overfitting

By restricting the training set to specific sizes, we can demonstrate the effect of overfitting on our classifier. As the training set size increases, we note that performance on the validation set and test set increases, as demonstrated by the increasing accuracy. However, we see a slight decrease in training set accuracy, due to the large number of images being trained on. Note that the test set described in this section is the union of test sets for the 6 actors whose images aren't being used for training. Whereas the training and validation sets are composed of images of Lorraine Bracco, Peri Gilpin, Angie Harmon, Alec Baldwin, Bill Hader, and Steve Carell, the test uses images of America Ferrera, Fran Drescher, Kristin Chenoweth, Daniel Radcliffe, Micharl Vartan, and Gerard Butler.

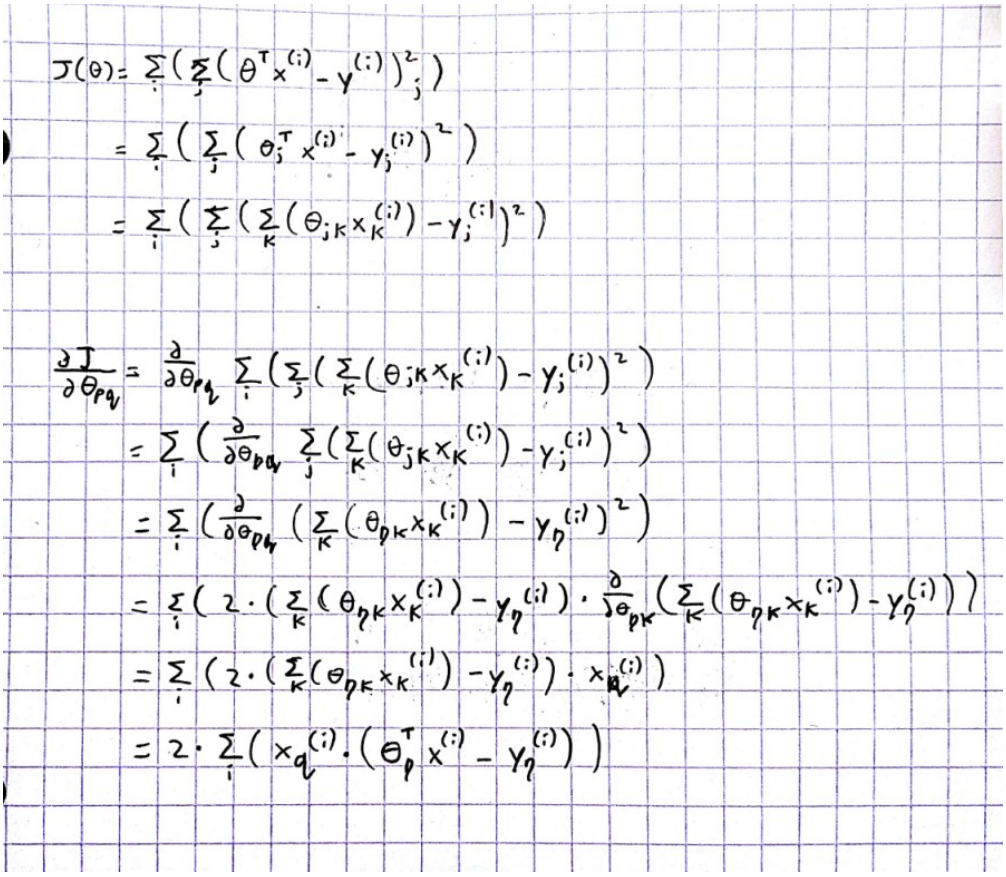
Figure 9: Classifier Performance Plotted as a function of Training Set Size



## Part 6: One-hot Encoding

One-hot encoding refers to the process of assigning label vectors instead of scalars, in which each vector index is 0 except for a single index that's 1. Using this encoding, we can build a classifier that classifies multiple actors, instead of just two. Let's begin by examining the cost function used. The derivative is computed as follows:

Figure 10: Derivation of Cost Function Derivative



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

$$= \sum_i \left( \sum_j \left( \theta_j^T x^{(i)} - y_j^{(i)} \right)^2 \right)$$

$$= \sum_i \left( \sum_j \left( \sum_k \left( \theta_{jk} x_k^{(i)} \right) - y_j^{(i)} \right)^2 \right)$$

$$\frac{\partial J}{\partial \theta_{pq}} = \frac{\partial}{\partial \theta_{pq}} \sum_i \left( \sum_j \left( \sum_k \left( \theta_{jk} x_k^{(i)} \right) - y_j^{(i)} \right)^2 \right)$$

$$= \sum_i \left( \frac{\partial}{\partial \theta_{pq}} \sum_j \left( \sum_k \left( \theta_{jk} x_k^{(i)} \right) - y_j^{(i)} \right)^2 \right)$$

$$= \sum_i \left( \frac{\partial}{\partial \theta_{pq}} \left( \sum_k \left( \theta_{pk} x_k^{(i)} \right) - y_\eta^{(i)} \right)^2 \right)$$

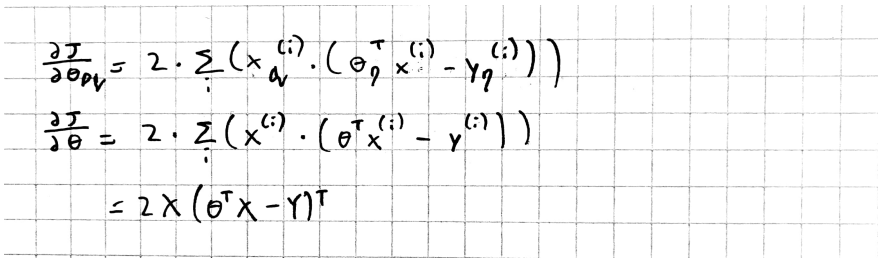
$$= \sum_i \left( 2 \cdot \left( \sum_k \left( \theta_{\eta k} x_k^{(i)} \right) - y_\eta^{(i)} \right) \cdot \frac{\partial}{\partial \theta_{pk}} \left( \sum_k \left( \theta_{\eta k} x_k^{(i)} \right) - y_\eta^{(i)} \right) \right)$$

$$= \sum_i \left( 2 \cdot \left( \sum_k \left( \theta_{\eta k} x_k^{(i)} \right) - y_\eta^{(i)} \right) \cdot x_q^{(i)} \right)$$

$$= 2 \cdot \sum_i \left( x_q^{(i)} \cdot \left( \theta_\eta^T x^{(i)} - y_\eta^{(i)} \right) \right)$$

Now, let $m$ be the number of training examples, $n$ be the number of features, and $k$ be the number of possible labels. Then, X is of dimension $n$ x $m$, $\theta$ is of dimension $n$ x $k$, and Y is of dimension $k$ x $m$. In examining the derivative computed above, we see that each derivative $\frac{\partial J}{\partial \theta_{pq}}$ is an element of the gradient $\frac{\partial J}{\partial \theta}$ lying at column $p$ and row $q$. The derivation is shown in the notes below:

Figure 11: Derivation of Cost Function Gradient



$$\frac{\partial J}{\partial \theta_{pq}} = 2 \cdot \sum_i \left( x_q^{(i)} \cdot \left( \theta_\eta^T x^{(i)} - y_\eta^{(i)} \right) \right)$$

$$\frac{\partial J}{\partial \theta} = 2 \cdot \sum_i \left( x^{(i)} \cdot \left( \theta^T x^{(i)} - y^{(i)} \right) \right)$$

$$= 2X \left( \theta^T X - Y \right)^T$$

The cost function and its vectorized gradient function were implemented in python, and the code is included below. Note that the functions have been modified from the derived equations; X is of dimension $m$ x $n$, $\theta$ is of dimension $n$ x $k$, and Y is of dimension $m$ x $k$.

Figure 12: The Cost Function and Gradient Function for One-hot Linear Regression

```
def one_hot_cost_function(x, y, theta):
    m = x.shape[0]
    h = np.matmul(x, theta)
    cost = 0.5 / m * np.sum(np.square(h - y))
    return cost

def one_hot_gradient(x, y, theta):
    m = x.shape[0]
    h = np.matmul(x, theta)
    grad = 1.0 / m * np.matmul(x.T, h - y)
    return grad
```

To ensure that the gradient function is working correctly, we compute the gradient using the finite difference approximation and compare it to our function results. The code used is shown below:

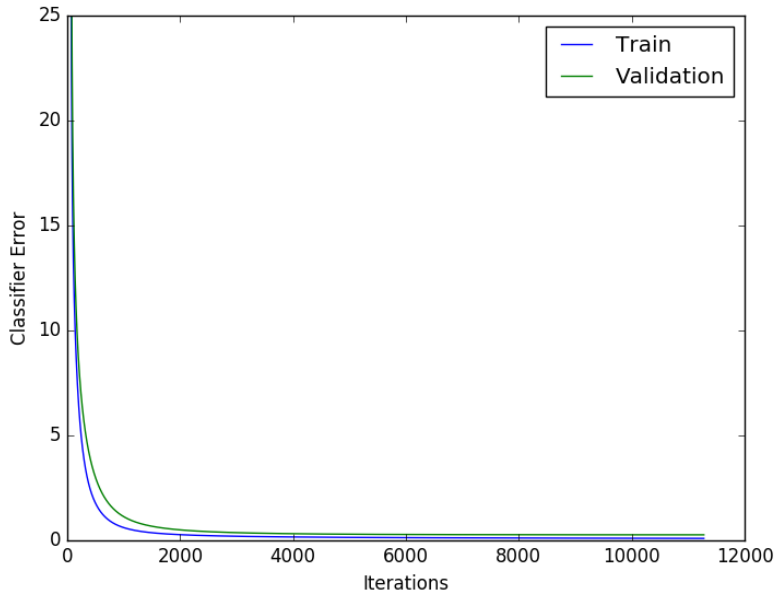
Figure 13: Comparing Finite-Difference Approximation with Gradient

```
def gradient_check(x, y, theta):
    h_values = [0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001, 0.00000001, 0.000000001, 0.
                                                    0000000001, 0.00000000001]
    grad_history = np.zeros((len(h_values), 1))
    grad = one_hot_gradient(x, y, theta)

    for i in range(len(h_values)):
        h = h_values[i] * np.ones((theta.shape[0], theta.shape[1]))
        f_prime = float(one_hot_cost_function(x, y, theta + h) - one_hot_cost_function(x, y,
                                                        theta)) / h

        grad_history[i, 0] = np.sum(np.square(f_prime - grad))

    print grad[0,0]
    print f_prime[0, 0]
    plt.plot(grad_history)
    plt.show()
```

# Part 7: Face Recognition

A face classifier was built using the one-hot encoding gradient and cost functions. The classifier was trained on a dataset of 420 32x32 pixel images of 6 actors (m = 420, k = 6, n = 1025). The accuracy of the classifier on the training set, validation set, and test set was 0.976, 0.80, and 0.85 respectively, indicating good performance. The parameters used for gradient descent were alpha = 0.005 and max_iterations = 50000. However, early stopping was implemented to prevent overfitting, forcing gradient descent to stop after approximately 11000 iterations. The value of alpha was experimentally derived; the highest possible value of alpha that converges was chosen. The code used to build the classifier can be found in faces.py

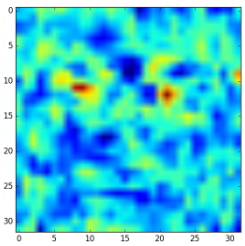
Figure 14: Face Classifier Training Performance. Classifier accuracy is plotted as a function of the number of training iterations.



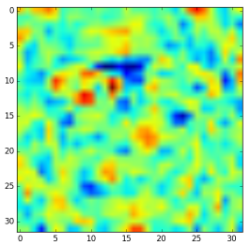
# Part 8: Theta Matrix Visualization

As we did with linear regression, the theta matrix can be visualized by plotting the array corresponding to each category as in image. The result is shown below:

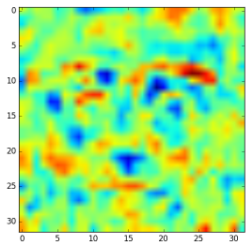
Figure 15: Theta Matrix Visualization



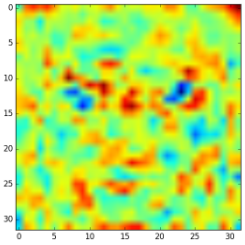
(a) Alec Baldwin



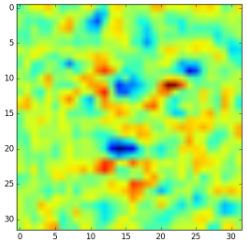
(b) Lorraine Bracco



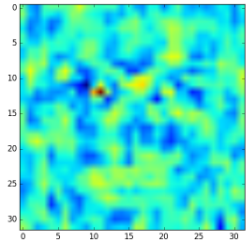
(c) Steve Carell



(d) Peri Gilpin



(e) Bill Hader



(f) Angie Harmon