CSC411 Project 1

Shichen Lu

January 29, 2018

Part 1

The images were taken from the online "FaceScrub" database, which can be found at http://vintage.winklerbros.net/facescrub.html. It contains images of people as well as information for where on the image the person's face is located. For this project, we have taken a set of images from certain actors from the website to use in an image classification project. To evaluate the quality of the images collected, some images from the dataset are produced below:

(a) Alec Baldwin

(b) Lorraine Bracco

(c) Daniel Radcliffe

Figure 1: Uncropped Images

Based on the data given by the database, the images are automatically cropped, resized, and converted to grayscale to produce the following:

(a) Alec Baldwin

(b) Lorraine Bracco

(c) Daniel Radcliffe

Figure 2: Cropped Images

As we can see, there is varying data in the quality of the images. While most images are head-on images of the actor which produce good quality faces when cropped, such as the Daniel Radcliffe example given above, there are a small amount of images in the database where the actors are shot from the side, or are not directly facing the camera, as seen in the Alec Baldwin example. Additionally, there are also a small amount of pictures in which the provided information to crop out the actor's face is completely wrong, as seen in the Lorraine Bracco example. However, the amount of images that fall into the third category is very small (estimated $< 1\%$). Overall, the quality of cropped images are fairly good.

Part 2

Initially, we will analyise six total actors: Lorraine Bracco Peri Gilpin Angie Harmon Alec Baldwin Bill Hader Steve Carell.

The following algorithm was used to separate the cropped face images of each actor:

```
# Generates and returns training, validation, and test sets
    ↪ for each actor
def generate_sets(actors):
    extensions = [".jpg", ".JPG", ".png", ".PNG", ".jpeg", ".
        ↪ JPEG"]
    image_counts = image_count("./cropped")
    training_sets = {key: [] for key in actors}
    validation_sets = {key: [] for key in actors}
    test_sets = {key: [] for key in actors}
    for actor in actors:
        for i in range(image_counts[actor] - 20):
            for extension in extensions:
                if (os.path.isfile("./cropped/" + actor.split()
                    ↪ [1].lower() + str(i) + extension)):
                    training_sets[actor].append((actor, actor.
                        ↪ split()[1].lower() + str(i) +
                        ↪ extension))
        for i in range(image_counts[actor] - 20, image_counts[
            ↪ actor] - 10):
            for extension in extensions:
                if (os.path.isfile("./cropped/" + actor.split()
                    ↪ [1].lower() + str(i) + extension)):
                    validation_sets[actor].append((actor, actor.
                        ↪ split()[1].lower() + str(i) +
                        ↪ extension))
        for i in range(image_counts[actor] - 10, image_counts[
            ↪ actor]):
            for extension in extensions:
                if (os.path.isfile("./cropped/" + actor.split()
                    ↪ [1].lower() + str(i) + extension)):
                    test_sets[actor].append((actor, actor.split
                        ↪ ()[1].lower() + str(i) + extension))
    return (training_sets, validation_sets, test_sets)
```

After getting the images of each actor from the database, this function is ran with an argument that dictates the list of actors to generate test, validation, and training sets for. The function uses helper function "image_count" to

count how many images for each actor we were able to collect from the database (as some links on the database are dead). This "image_count" is reproduced below:

```
# Returns a dictionary that lists how many images of each
    ↪ actor are present in a directory
def image_count(path):
    res = {key: 0 for key in actors}
    for file in os.listdir(path):
        for actor in actors:
            if file.startswith(actor.split()[1].lower()):
                res[actor] += 1
    return res
```

Overall, for each actor, the function simply puts the last ten collected images into the test set, the 20th to 11th last collected images into the validation set, and the rest of the images into the training set. Note that although the images are split the same way for each time the program is run, due to how the datacollection parses the database images, the images are not collected in the same order each time the program is run. So the generated sets are not necessarily the same for each time the program is run.

Part 6

Subsection A

We have

$$J(\theta) = \sum_i (\sum_j (\theta^T x^{(i)} - y^{(i)})_j^2) \tag{1}$$

We can use the chain rule compute the partial derivative with respect to θ_{pq} (ie. to each individual theta element in the vector) as

$$\frac{\partial J}{\partial \theta_{pq}} = 2 \sum_i (\sum_j ((\theta^T x^{(i)} - y^{(i)}) * \frac{\partial \theta^T x^{(i)}}{\partial \theta_{pq}}))_j \tag{2}$$

Additionally, we know that every term in $\frac{\partial \theta^T x^{(i)}}{\partial \theta_{pq}}$ other than