Problem Statement

The question is this: given a twitter profile picture, can we build a model that determines whether they are wearing eyeglasses in the profile picture? My client would be any sort of eyeglass company (e.g., Warby Parker), that wants a more automated way of targeting potential customers. The company would search through different Twitter users and, if the model determines they wear eyeglasses, the eyeglass company would target these users with advertising or speciality deals. The company would thus be able to more effectively target potential customers (those that wear glasses) and improve sales. Additionally, the modeling dataset contains a variety of other attributes, e.g., wearing a hat, having a moustache, having bangs, etc., with which we could use to retrain the CNN and target different users on Twitter. A company that sells hats or facial care products for men would thus also be able to use this model to focus their advertising efforts on Twitter users that wear hats, have moustaches/beards, etc.

Data Cleaning and Wrangling

The primary dataset for this project is the Celebrity Faces Attribute (CelebA) dataset, which "is a large-scale face attributes data with more than 200K celebrity images, each with 40 attribute annotations." Much of the preprocessing and data wrangling steps for this dataset were done following the code in this GitHub repo, where a convolutional neural network (CNN) was trained on the same dataset (CelebA) to determine whether the celebrity in the image is wearing eyeglasses or not. The images in the CelebA dataset are of size 178 x 218 and are ≲10 KB each. The first step in this process is gathering all the images in the dataset where someone is wearing eyeglasses--a total of 13,193 images from the total 202,599 image dataset, or ~6.5%. We would like our total, modeling dataset to consist of 80,000 images, so we gather the remaining 66,807 non-eyeglasses images. After gathering all the images, we shuffle their order and use a custom face aligner to realign the celebrity images such that the alignment process between the CelebA dataset and the later Twitter-profile-picture dataset is consistent. In some instances, the face-aligner fails to capture any faces in the celebrity image and thus fails to align them--these instances are skipped over and not included in the modeling dataset (removes ~3.85% of the 80,000 images, leaving us with 76,914 images for modeling). The images are then converted to grayscale and resized to 28 x 28. The images along with their corresponding labels (1 for eyeglasses, 0 for no eyeglasses) and image file names are saved together in a numpy file.

To construct the Twitter-profile-picture dataset, the avatars.io service along with Python's `requests` module was used to first obtain the profile pictures from Twitter. I gathered 20 Twitter profile pictures and hand labeled them 1 if they wear glasses and 0 if they do not. These images were then put through the same pipeline as the CelebA images--first align the face and resize to the same size as the images in the CelebA dataset (178 x 218), then convert to grayscale, and finally resize to 28 x 28. This process is shown for a few examples in **Figure 1**.

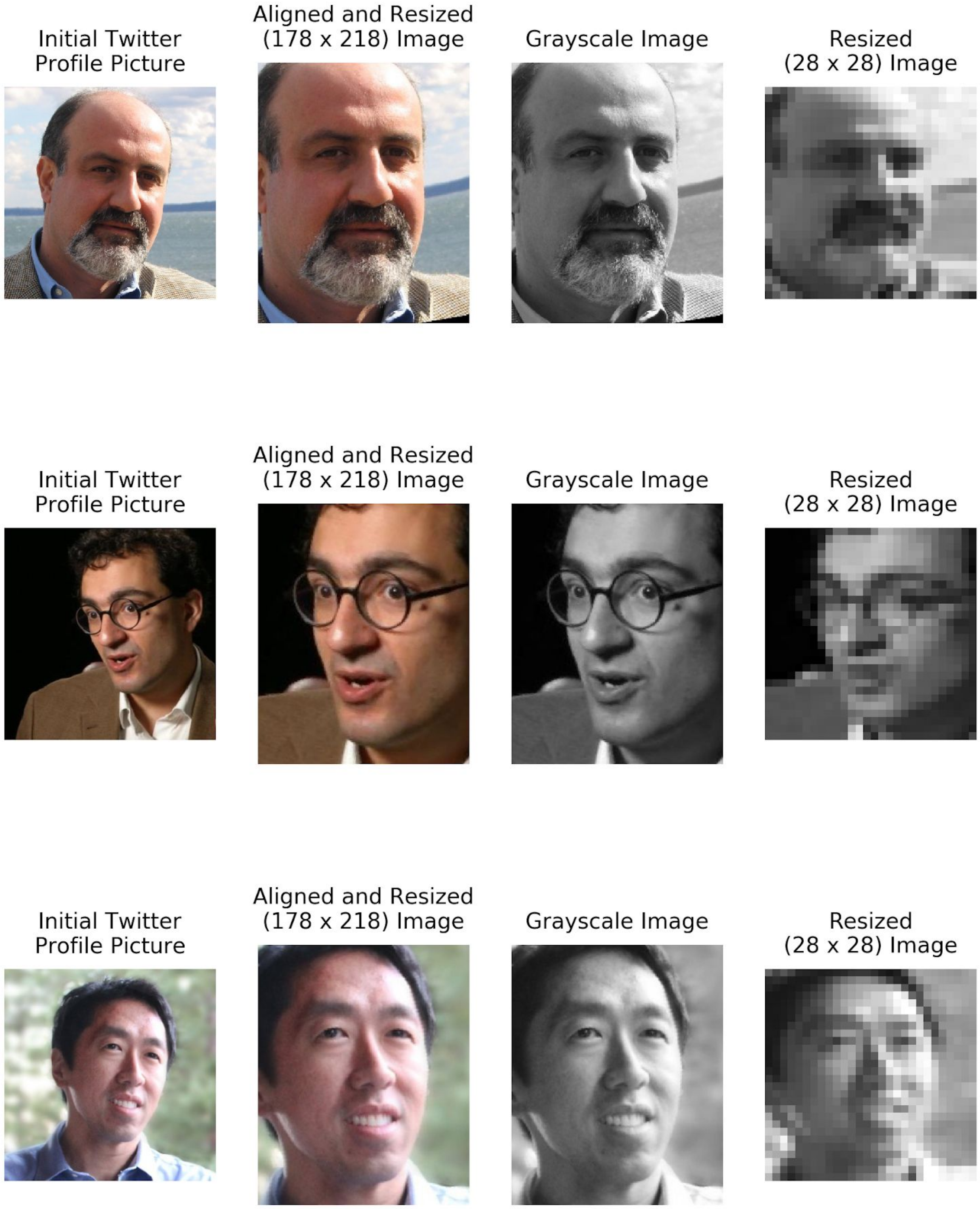

Initial Twitter Profile Picture
Aligned and Resized (178 x 218) Image
Grayscale Image
Resized (28 x 28) Image
Initial Twitter Profile Picture
Aligned and Resized (178 x 218) Image
Grayscale Image
Resized (28 x 28) Image
Initial Twitter Profile Picture
Aligned and Resized (178 x 218) Image
Grayscale Image
Resized (28 x 28) Image

Figure 1. Three example of the image preprocessing for twitter profile pictures. From left to right: first we have the initial Twitter profile picture, then the alignment and resizing to the CelebA

dataset, then converted to a grayscale image, and finally resized to 28 x 28, which will be used for modeling. From top to bottom, the Twitter users are @nntaleb, @SimonDeDeo, and @AndrewYNg.

The dataset is now ready for modeling.

Machine Learning Modeling

Convolutional Neural Networks

To approach this problem, we used a convolutional neural network (CNN) that, similarly to ordinary neural networks (NN), receive an input (in this case images), and transforms it through a series of hidden layers. Each hidden layer consists of a set of neurons and in a traditional NN, each neuron is fully connected to all the neurons in the previous layer. While this scheme works well for a variety of different data sets, the scheme struggles to scale to image data. In CIFAR-10 for example, images are only of size 32x32x3, so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 32*32*3 = 3072 weights. This is manageable, but this fully-connected structure obviously doesn't scale well to larger images; images of size 200x200x3 would lead to neurons that have 120K weights. Additionally we would like to have several layers of neurons, which would add up to quite a large number of parameters. Indeed, this full-connectivity is wasteful and the huge number of parameters would lead to overfitting.

CNNs take into consideration that the input is images and constrain the architecture in a more sensible way. CNNs arrange neurons in three dimensions: width, height, and depth. In CIFAR-10, those number would be 32x32x3 respectively. In particular, neurons in CNNs are connected only to a small region of the layer before it, instead of all the neurons in the previous layer like a typical NN. The final output layer of a CNN is a vector of class scores along the depth dimension; in out case, that would be 1x1x2 because we are concerned only with two classes (wearing eyeglasses: 1, or not wearing eyeglasses: 0).

CNNs transform input images through a series of layers. As mentioned, each layer accepts a 3D input volume and transforms it to a 3D output volume. The three main types of layers are the convolution layer, pooling layer, and the fully-connected layer, each of which is briefly described below.

Convolutional (CONV) Layer

“The convolution layer’s parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels). During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.”

RELU Layer

RELU layer will apply an elementwise activation function, such as the max(0,x) thresholding at zero.

Pooling Layer

“It is common to periodically insert a Pooling layer in-between successive convolutional layers. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The

Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations."

Fully-connected Layer

"Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular NNs. Their activations can hence be computed with a matrix multiplication followed by a bias offset."

CNN Architecture

The most common form of a CNN architecture stacks a few CONV-RELU layers, follows them with pool layers, and repeats this pattern until the image has been merged spatially to a small size. At some point, it is common to transition to fully-connected layers. The last fully-connected layer holds the output, such as the class scores. In other words, the most common ConvNet architecture follows the pattern:

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

where typically:

- `N <= 3`
- `M >= 0`
- `K < 3`

More details on CNNs and their inner workings can be found in this comprehensive review.

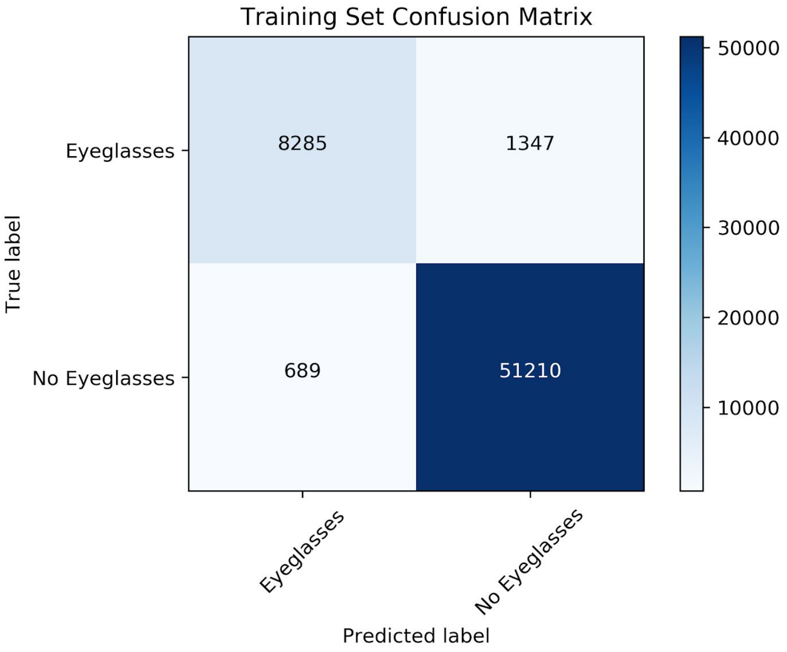
For this project, we used a CNN architecture based on a previous project that had the exact same goal: to determine whether the celebrity in an image is wearing eyeglasses or not. This CNN was built using TensorFlow (TF). In general, and especially with NNs, we don't want to reinvent the wheel, so using a previously tested CNN architecture for this problem saves a lot of work. In this architecture, the flow of data is as follows:

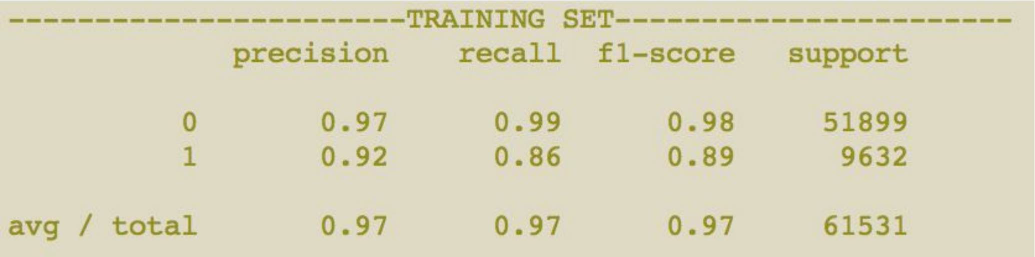
1. Take as input CelebA images of size 28x28 and reshape to a `(batch_size x 28 x 28 x 1)` sized tensor, as is required by the TF algorithm.
2. Pass the input tensor to the first convolution layer, which contains 32 filters of size 5x5 with RELU activation. This transforms input tensor of size `(batch_size x 28 x 28 x 1)` to size `(batch_size x 28 x 28 x 32)`
3. Pass the input tensor from the first convolution layer to the first pooling layer with a filter of size 2x2 and stride of 2. This transforms the input tensor of size `(batch_size x 28 x 28 x 32)` to size `(batch_size x 14 x 14 x 32)`
4. Pass the input tensor from the first pooling layer to the second convolution layer with 64 filters. This transforms the input tensor of size `(batch_size x 14 x 14 x 32)` to size `(batch_size x 14 x 14 x 64)`
5. Pass the input tensor from the second convolution layer to the second pooling layer with a filter of size 2x2 and stride of 2. This transforms the input tensor of size `(batch_size x 14 x 14 x 64)` to size `(batch_size x 7 x 7 x 64)`
6. Next we flatten the input tensor from the second pooling layer, transforming the input tensor of size `(batch_size x 7 x 7 x 64)` to size `(batch_size x 7*7*64)`
7. Then we add a fully-connected layer with 1024 neurons and RELU activation, changing the input tensor of size `(batch_size x 7*7*64)` to size `(batch_size x 1024)`
8. Next we add a dropout layer with a rate of 0.4. This operation randomly sets 40% of the input units to zero and help prevent overfitting.
9. Finally we add a fully-connected layer with that will perform the class predictions. It takes as input the tensor from the dropout operation of size `(batch_size x`

`1024)` and transforms it to a tensor of size `(batch_size x 2),` thus giving both the probability that the celebrity in the image is wearing glasses and the probability they are not wearing glasses.

This architecture was chosen because it performed best during the testing/training process in the aforementioned project similar to this one.

We are now ready to train/test the CNN model. Our input 76,914 are split into a training and testing set (80% for training, 20% for testing) and performing training with a batch size of 100 and 2000 epochs. We use the softmax cross entropy for the loss function and the Adam optimizer to perform the training. **Figure 2** shows the confusion matrices and corresponding statistics for the testing and training set. Overall, the model performs very well, although the model suffers primarily in recall--i.e., it has the most trouble attempting to correctly classify all the instances where a celebrity is indeed wearing eyeglasses. **Figure 3** shows the ROC curves and associated AUC for both the training and testing set. The model achieved an AUC of ~0.98 and ~0.99 for the training and testing sets respectively.

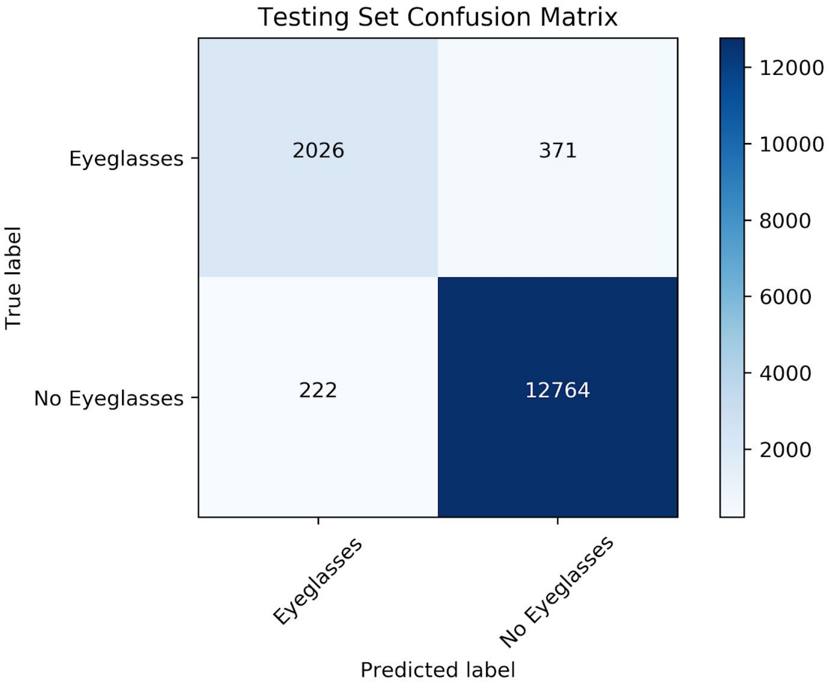


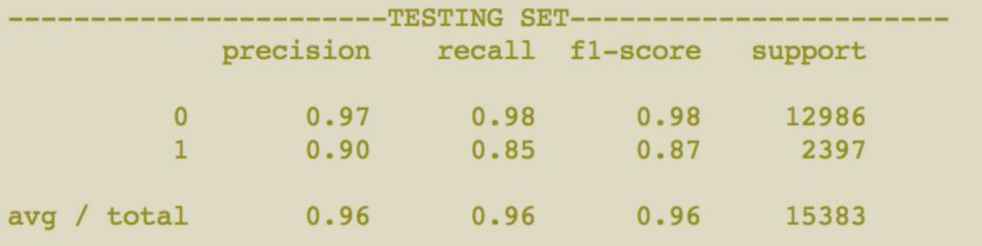


```
-----------------------TRAINING SET-----------------------
             precision    recall  f1-score   support

          0       0.97      0.99      0.98     51899
          1       0.92      0.86      0.89      9632

avg / total       0.97      0.97      0.97     61531
```





```
-----------------------TESTING SET-----------------------
             precision    recall  f1-score   support

          0       0.97      0.98      0.98     12986
          1       0.90      0.85      0.87      2397

avg / total       0.96      0.96      0.96     15383
```

Figure 2. Confusion matrices and statistics for the training (top) and testing (bottom) sets. While the statistics for the negative class (denoted with zero: no eyeglasses) are nearly optimal, the positive class (denoted with one: wearing eyeglasses) suffers slightly from lower recall, indicating the model's difficulty in correctly identifying all instances where someone is indeed wearing eyeglasses

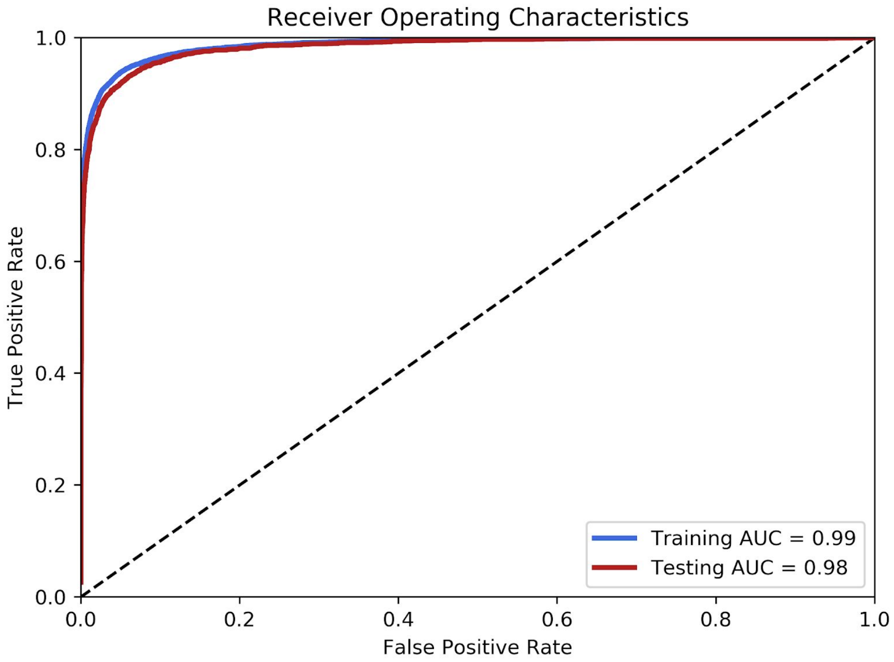
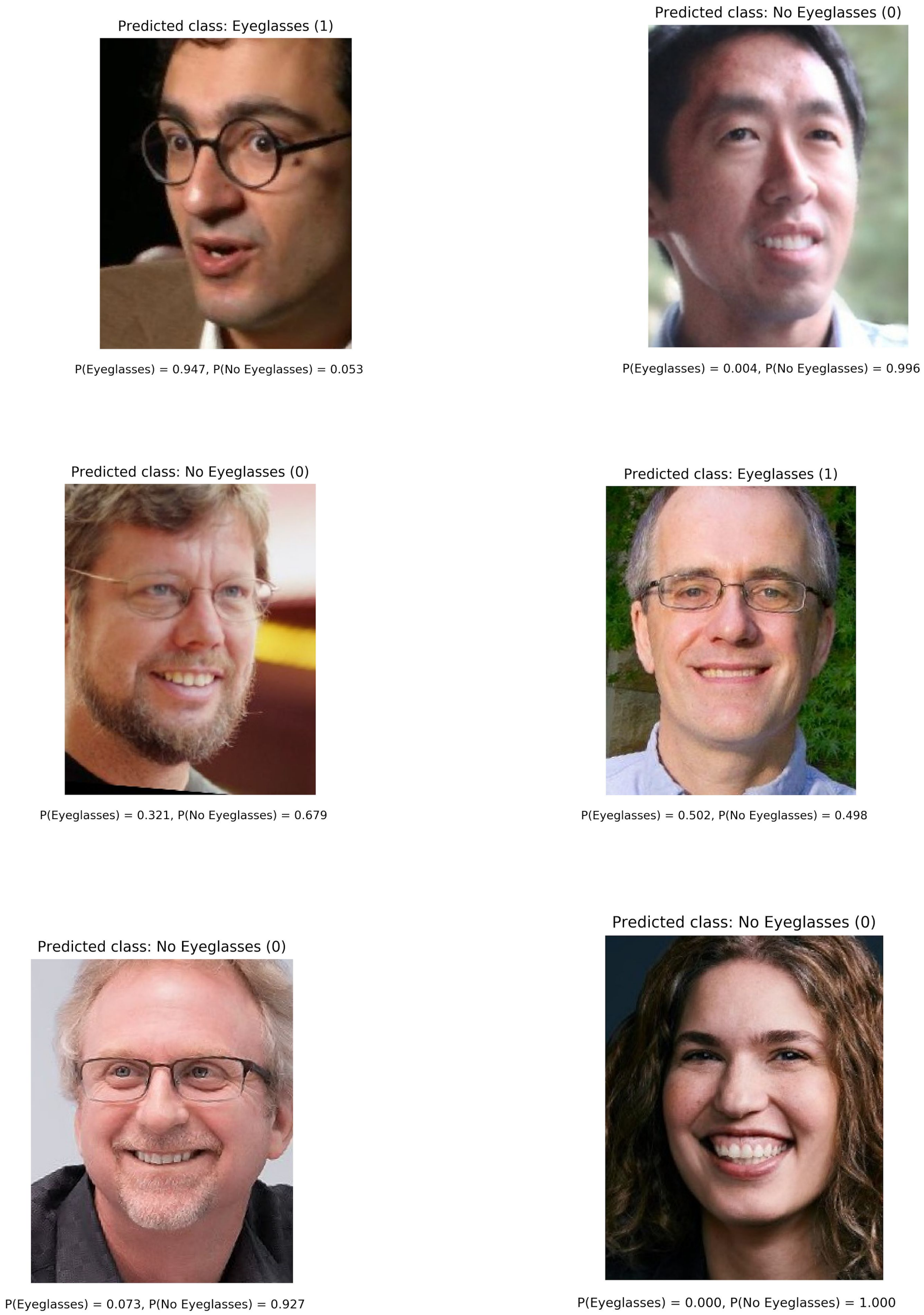


Figure 3. ROC curves for both the training and testing sets. High AUC values indicate that our model is performing well.

Applying to Social Media Profile Pictures

Now we apply our trained CNN to our set of Twitter profile pictures. Our model performs quite well: out of the 20 Twitter profile pictures, the correct class was predicted for 17 of the images giving an accuracy of 85%. As we saw in the training/testing section, the most model struggles most with recall, i.e., detecting eyeglasses when someone is actually wearing them. **Figure 4** shows a few examples of twitter profile pictures and their associated class probabilities. Even in the case where the model predicts someone is not wearing eyeglasses when they are (i.e., a type II error), we observe some probability (>10%) that they are wearing glasses, indicating that using a lower threshold could increase accuracy on Twitter images. All the Twitter images used for evaluation and their associated class probabilities can be found here.



Predicted class: Eyeglasses (1)

P(Eyeglasses) = 0.947, P(No Eyeglasses) = 0.053

Predicted class: No Eyeglasses (0)

P(Eyeglasses) = 0.004, P(No Eyeglasses) = 0.996

Predicted class: No Eyeglasses (0)

P(Eyeglasses) = 0.321, P(No Eyeglasses) = 0.679

Predicted class: Eyeglasses (1)

P(Eyeglasses) = 0.502, P(No Eyeglasses) = 0.498

Predicted class: No Eyeglasses (0)

P(Eyeglasses) = 0.073, P(No Eyeglasses) = 0.927

Predicted class: No Eyeglasses (0)

P(Eyeglasses) = 0.000, P(No Eyeglasses) = 1.000

Figure 4. Class probabilities for a few Twitter images. Whereas the model always does well at detecting the negative class (no eyeglasses) it sometimes struggles with detecting eyeglasses when the person is actually wearing them. The two profile pictures in the second row demonstrate this

difficulty; although both users wear glasses, only one image was correctly labeled with a probability just over 50%. The other image of Guido van Rossum showed a non-significant probability of wearing eyeglasses, yet the probability of not wearing them was higher, indicating that a lower threshold might capture more positive class cases. In other cases (bottom left) the model fails to significantly detect eyeglasses at all.

It should be noted that the Twitter profile pictures chosen for evaluation are in a similar format to the modeling dataset (CelebA); they are mostly front-on images of the user's face. There are a plethora of bizarre Twitter profile pictures that the model would surely not do as well on. An eyeglasses company using this model would be best served by targeting Twitter users that have their real names and pictures on their account, not anonymous users.

Concluding Remarks

As shown, this CNN model works quite well on social media profile pictures. An accuracy even close to 85% is impressive and certainly helpful for a targeted advertising campaign. Additionally, one could adapt this pre-trained CNN for other business applications. For example, if a men's facial care company (like Gilette) was trying to target social media users with beards or facial hair, the model could be retrained on images of people with beards from the CelebA dataset and subsequently applied to social media images. Considering all the attributes in the CelebA dataset, there are a variety of different businesses that could benefit from running targeted advertising campaigns using this model on social media.