Capstone Project

Machine Learning Engineer Nanodegree

Anish Pillay

August 3, 2017

I. Definition

Project Overview

While watching a movie, TV show or even an interview, many a times we come across a situation where we are unable to identify a celebrity or a person in the interview. A facial recognition software to identify celebrities or even a sports person during a live sporting event would be of great help. It would be amazing to see the name of the person on a flick of a button either from your TV remote or a mobile app.

In this project, I have created a facial recognition software using tensorflow to recognize celebrities in a movie clip. The application uses a Facial detection software library from dlib as well as Google's TensorFlow software to recognize faces with incredible accuracy for limited number of input facial images.

The primary dataset used for training is a subset of Public Figures Database and for testing is a custom set of raw images. I have also created a video clip of various movie scenes or interviews of celebrities to test the model. Training Data:

Public Figures - http://www.cs.columbia.edu/CAVE/databases/pubfig/download/

Problem Statement

The goal is to create a facial recognition model that could identify celebrities in images as well as video clips. The steps involved in this process are:

1. Download a celebrity dataset
2. Normalize and Preprocess the data to get only the facial content of the celebrity
3. Integrate a facial detection algorithm to efficiently identify faces in an image
4. Train a Tensorflow model to learn various facial features accurately

5. Build an image pipeline to test the model on images as well as video

This model is expected to be efficient so that it could run on a live stream with minimal lag.

Metrics

If we consider accuracy as our metric, it would be a measure of number of correct predictions from the total number. This could mean that if our 80% of the dataset consisted pictures of Denise Richards, our model would be accurate to 80 % which would not be a correct measure.

On the other hand, F1 score considers both precision and recall in its calculation and thus could give us more reliable results for multi class classification.

$$\text{F1\_score} = \frac{2 * P \ * R}{P \ + R}$$

P= Precision R = Recall

This metric was used in the project as it is a balanced accuracy score considering false positives as well as false negatives. Precision lets you know how many of the predicted value is correct and recall lets you know how many of the actual labels were predicted.

Precision = $\frac{True\ Positive}{True\ Positive + False\ Positive}$; Recall = $\frac{True\ Positive}{True\ Positive + False\ Negative}$

By considering false positives and false negatives in our equation, we make sure to account for instances where we predict a class incorrectly. For instance, if we predict Daniel Craig to be Denise Richards. If either precision or recall is low, it will drive the f1 score down by a factor of 2.

For the project, I have considered a weighted F1 score which basically calculates metrics for each label and finds their average, weighted by support (number of true instances for each label). Thus, accounting for label imbalance in the dataset. Due to variability in weighting support for each label, it serves as an unbiased measure for multi class classification

II. Analysis

Data Exploration

The PubFig database is a large, real-world face dataset consisting of 58797 images of 200 people collected from the internet. There is a large variation in pose, lighting, scene, camera, imaging conditions and parameters. A lot of the images have 2 or more faces in them making it difficult to preprocess through code for training

Some links from the dataset text file don’t exist anymore as well as these images have different resolutions. Also, the face dimensions mentioned for many of the images are no longer valid and might give you irrelevant data. All these needs to be considered while preprocessing the data. For computing ease as well as reduced preprocessing efforts, only a subset of this database is taken into consideration.

From the 200 people, 17 celebrities have been chosen to train and test the model. Each celebrity has 10 – 200 images to train on. After deleting irrelevant files and preprocessing the dataset, a total of 1283 images of 17 classes were available for training.

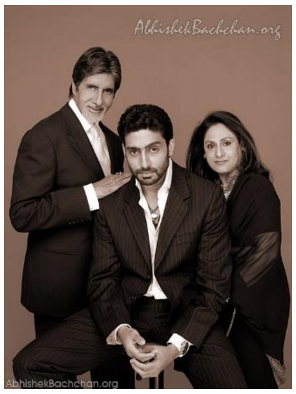
The dev_urls.txt file obtained for the PubFig database contains

1) Person name
2) Image Number
3) Web Link
4) Facial image coordinates in the picture
5) Md5sum link

Due to the variation in the number of images available per celebrity, we might see a bias in predicting one celebrity as compared to the other. This issue is not a major hurdle in classification and can be easily resolved by training on more images of that celebrity.

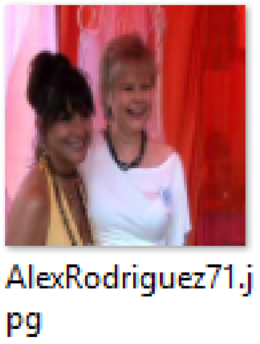
One more important characteristic about this dataset is that the faces in the images are not aligned. Celebrity faces cold be tilted, looking down, half faces etc.

Some abnormalities or characteristics:



3 Faces

Tilted Face



Wrong face in Alex Rodrigues Folder

Weird Faces

Side Profile

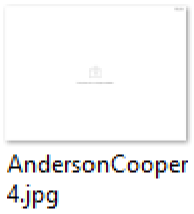
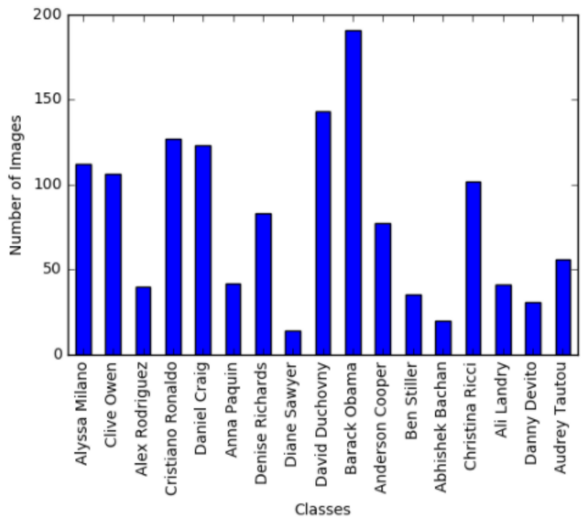
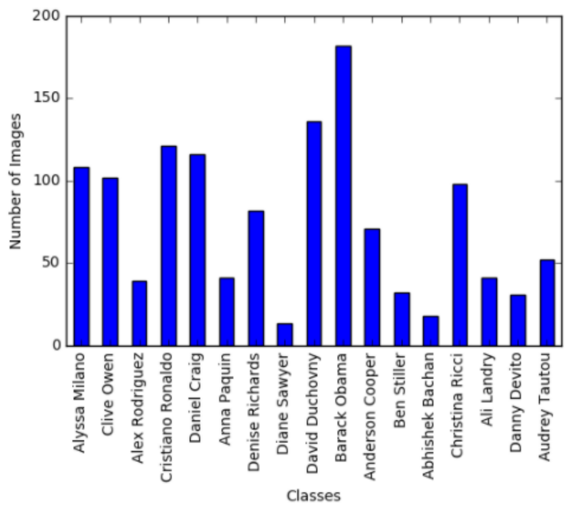


Image not available

Exploratory Visualization

The plot to the left shows the number of images available per class before filtering the data and getting rid of irrelevant pictures and right plot shows dataset filtered for training the model





As can be seen from the plot above, the number of images available per class seem to be unbalanced. This can bring about some challenges. The most visible problem is that if our metrics is accuracy, it will be biased towards the class which has higher number of images.

For this, I have taken 2 measures. After training the model with the available dataset, I have downloaded 200 test images for 5 classes and tested the model on this dataset. As this dataset is separate than the one I trained on, it would give us an unbiased score of the two models. Also, the metric in use is F1 score which is a weighted average of precision and recall

Facial features like eyes, nose, shape of the face, distance between the eyes are some of the features that the model will look at. The problem with the dataset is that the faces are not aligned. It would be beneficial to align the faces such that the eyes are relatively at the same distance from the center line for every image. The same applies for the nose and mouth. This could be considered as a normalization step that could help the algorithm to converge faster.

Algorithms and Techniques

The classifier selected for this type of multilabel classification is a Convolutional Neural Network. CNN is a very powerful algorithm for image classification as it learns various features from the dataset automatically as well as requires less pre-processing for the images. The algorithm provides a probability for each class while classifying images. A threshold on this probability could help filter out results on which the model has a certain amount of confidence. This threshold has been applied while recognizing celebrities in the video.

For this report, I have attempted to build the model from scratch.

With the CNN model, the training parameters were:

- Number of Epochs
- Batch Size of images
- Optimizer Model
- Loss model
- Number of Layers

- Classification threshold
- Layer Types
- Layer Parameters
- Data Preprocessing

Characteristics of CNN model:

CNN's are very similar to Neural Networks but they make the assumption that the inputs are images which allows us to encode certain properties into the architecture.

They comprise of different layers like Convolutional Layer, Activation Layer, Fully connected layer, Max-pooling layer etc.

Convolutional Layer:

Convolutional Layer consists of filters of specific size but same depth as the input image. These filters traverse through the input image and add weights to each region of the filter.

Max Pooling Layer:

Max Pooling layer looks at a specified filter size and stride and takes a max of the weights in that region. This reduces the spatial size and thus reduces the number of parameters and computation in the network.

Activation Layer:

Activation layer defines the output of the node depending on the input. It can give Boolean output or a sigmoid function as output depending on the input

Dense Layer:

Dense layer is basically fully connected layer and implements the function of Activation + Bias. It basically chooses N number of classes from given input number of classes.

Benchmark

Sklearn has an inbuilt dummy classifier for multi-class classification. As this classifier does not apply to images, I used a CNN model from Keras website as my benchmark

The benchmark CNN model architecture is present on Keras website below:

https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

I implemented the architecture mentioned in the website and trained my model to achieve an accuracy of 84% with 50 Epochs and 2000 samples per epoch.

Facial images had a resolution of 150 x 150 pixel images and was able to process a batch of 30 images at a time through the fit_generator function.

With this model set as my benchmark, I wanted to develop a model which learns on a smaller training dataset as well as achieves a better accuracy score. For instance, instead of collecting and training a model with 100 pictures of a celebrity, it would be efficient to train a model with just 10 pictures of the celebrity and successfully recognize an image of him/her with greater accuracy.

III. Methodology

Data Preprocessing

The downloading of images as well as cropping images is done in 'Preprocessing_Images.ipynb'. Instead of using the facial dimension from the dev_url.txt, I used HaarCascade to detect faces in an image and crop them using the dimensions provided by OpenCV CascadeClassifier. These images were written to its respective class folder within 'dataset_cropped' folder.

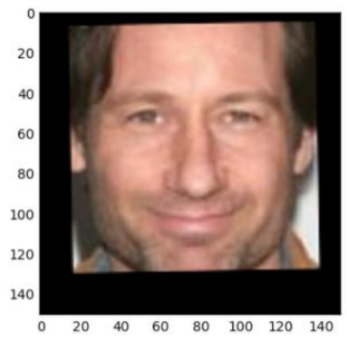
Other preprocessing steps include:

1) Algining Faces in the image
2) Converting to grayscale
3) Flipping the images to not let the model overfit
4) Adding Gaussian Blur to generalize class images appropriately
5) Random Brightness to images

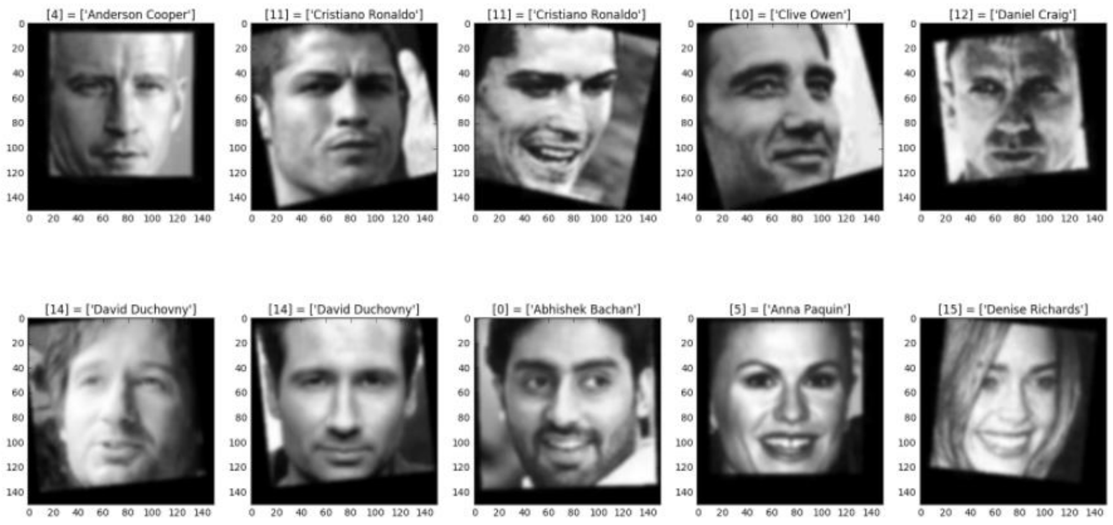
I quickly realized that aligning images while reading them and sending them as an input to the model would increase the training time. So I created a function to align faces and save them in a separate folder 'dataset_aligned'. This helped tremendously decrease my computing time and cost.

Multiple faces in an image caused multiple faces to be saved in the folder. These images had to be manually deleted to eliminate the model from learning wrong images.

Before Preprocessing

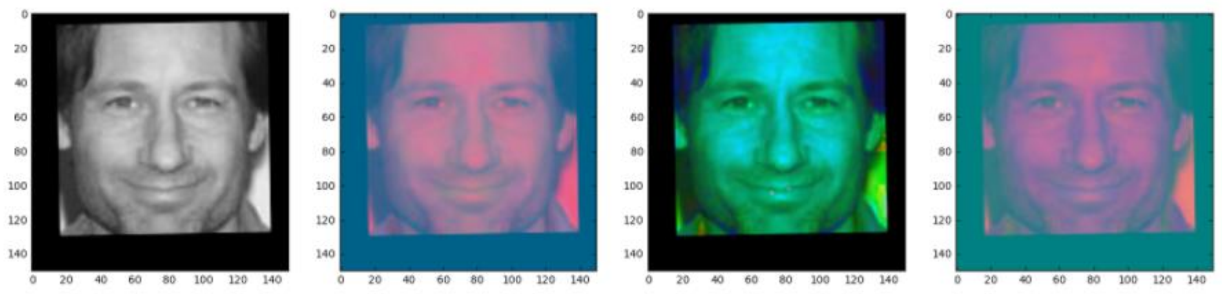


After Preprocessing:

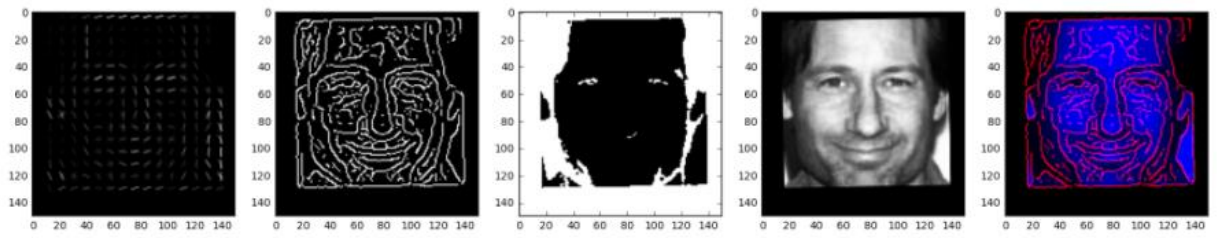

[4] = ['Anderson Cooper']
[11] = ['Cristiano Ronaldo']
[11] = ['Cristiano Ronaldo']
[10] = ['Clive Owen']
[12] = ['Daniel Craig']
[14] = ['David Duchovny']
[14] = ['David Duchovny']
[0] = ['Abhishek Bachan']
[5] = ['Anna Paquin']
[15] = ['Denise Richards']

Other Feature exploration methods were also tested to see if the model learns better with them, but it did not result in any significant improvement

Color Exploration:



Canny Edge, Hog Features, Emphasizing Grayscale as well as a combined result



Implementation

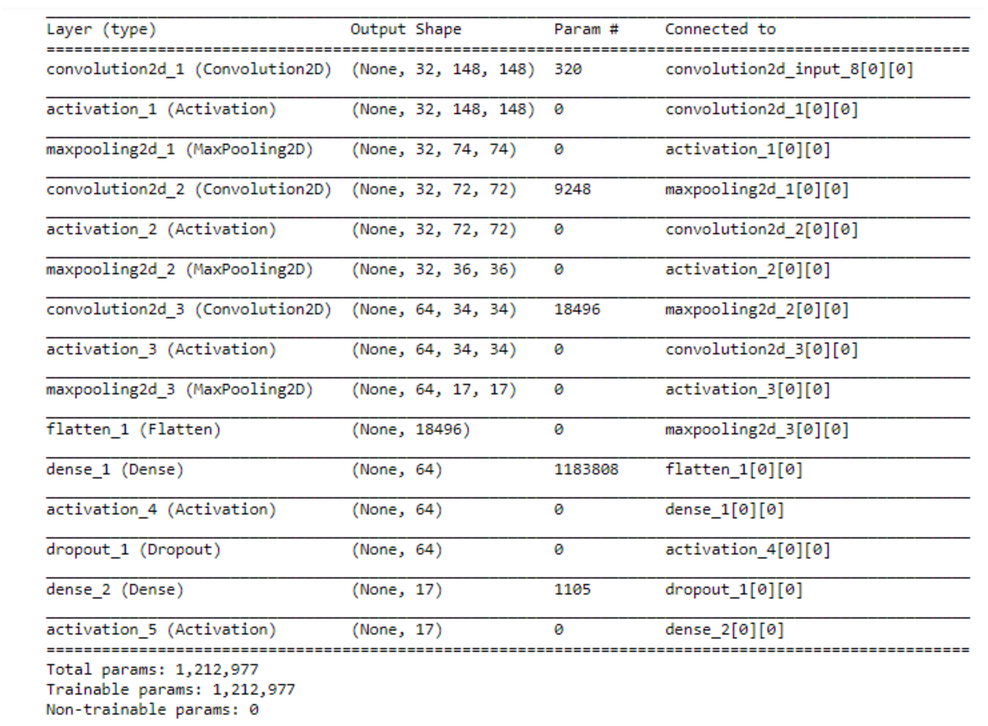
To implement the technique of facial recognition, it was important to get the preprocessing of images correctly and to generalize images across the same class of data. This was taken care of with the preprocessing step.

Further, a baseline model needed to be developed so as to compare future results with. As the models selected were Tensorflow models which require computationally higher RAM as well as speed, I created a data generator function that could feed the model with required number of samples per batch. The samples provided had to be preprocessed as well.

The baseline CNN model consisted of 3 convolutional layers with ReLu Activation and MaxPooling followed by 2 Dense layers and a sigmoid activation to classify the images. The Loss model chosen was 'sparse_categorical_crossentropy' keeping in mind that it's a multiclass classification problem. Adam Optimizer was used for this purpose as it can be directly used without changing its parameters for initial assessment.

A heavy dropout layer was used to randomly remove 50% of features before the last layer and thus avoid overfitting

The layout of the algorithm is as shown below with an input image size of 150 x 150 pixels



```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
convolution2d_1 (Convolution2D)  (None, 32, 148, 148)  320         convolution2d_input_8[0][0]
____________________________________________________________________________________________________
activation_1 (Activation)        (None, 32, 148, 148)  0           convolution2d_1[0][0]
____________________________________________________________________________________________________
maxpooling2d_1 (MaxPooling2D)    (None, 32, 74, 74)    0           activation_1[0][0]
____________________________________________________________________________________________________
convolution2d_2 (Convolution2D)  (None, 32, 72, 72)    9248        maxpooling2d_1[0][0]
____________________________________________________________________________________________________
activation_2 (Activation)        (None, 32, 72, 72)    0           convolution2d_2[0][0]
____________________________________________________________________________________________________
maxpooling2d_2 (MaxPooling2D)    (None, 32, 36, 36)    0           activation_2[0][0]
____________________________________________________________________________________________________
convolution2d_3 (Convolution2D)  (None, 64, 34, 34)    18496       maxpooling2d_2[0][0]
____________________________________________________________________________________________________
activation_3 (Activation)        (None, 64, 34, 34)    0           convolution2d_3[0][0]
____________________________________________________________________________________________________
maxpooling2d_3 (MaxPooling2D)    (None, 64, 17, 17)    0           activation_3[0][0]
____________________________________________________________________________________________________
flatten_1 (Flatten)              (None, 18496)         0           maxpooling2d_3[0][0]
____________________________________________________________________________________________________
dense_1 (Dense)                  (None, 64)            1183808     flatten_1[0][0]
____________________________________________________________________________________________________
activation_4 (Activation)        (None, 64)            0           dense_1[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 64)            0           activation_4[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 17)            1105        dropout_1[0][0]
____________________________________________________________________________________________________
activation_5 (Activation)        (None, 17)            0           dense_2[0][0]
====================================================================================================
Total params: 1,212,977
Trainable params: 1,212,977
Non-trainable params: 0
```

Initially the model was tuned based on the parameters mentioned above for 4 epochs to record if the accuracy had an upward trend. As soon as I saw accuracy increasing, I increased the number of epochs and trained my model. The model weights were saved in a h5 file. After each epoch, the model also gives out a validation accuracy.

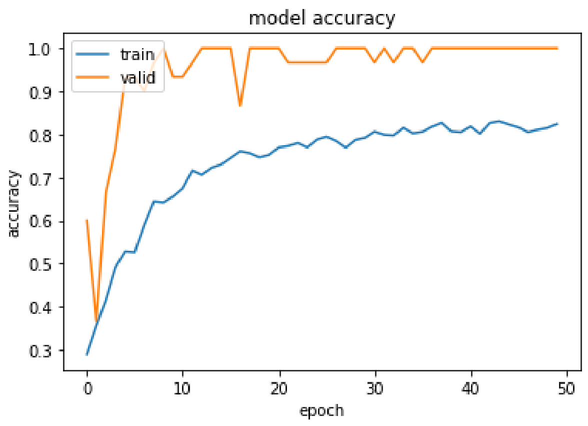
Some observations made while tuning was, normalizing of image pixels helped train the model faster, Gaussian blur worked as a good generalization technique, flipping images as well as randomly increasing/decreasing brightness helped in reducing overfitting.

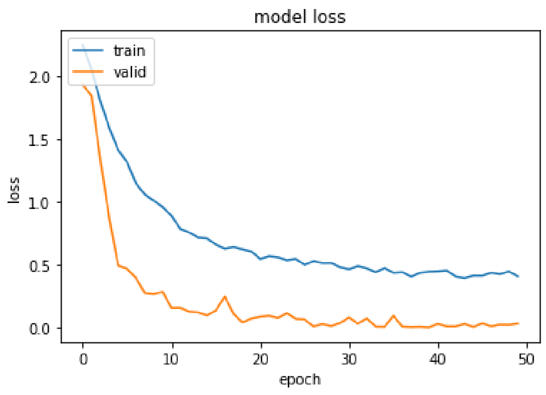
I created an image pipeline to serve the video demonstration part of the project. This pipeline took image as an input, detected faces as well as aligned and cropped them to feed it to the model predictor.

Files:

1) ‘facial_recognition_final_code.ipynb’ – Project code
2) ‘Preprocessing_Images.ipynb’ – Preprocessing database code
3) ‘dev_urls.txt’ – Links to images for the training and validation dataset
4) ‘modelV3.h5’ – Weights for the base CNN model
5) ‘model_CNNModV1.h5’ – Weights for the modified CNN model

Results from Base model are shown below:





Note: The training accuracy is less than validation accuracy due to the use of dropouts in the layer. Usually dropout is activated when training but deactivated when evaluating on the validation set. Besides, the training loss is the average of the losses over each batch of training data. Because your model is changing over time, the loss over the first batches of an epoch is generally higher than over the last batches. On the other hand, the testing loss for an epoch is computed using the model as it is at the end of the epoch, resulting in a lower loss.

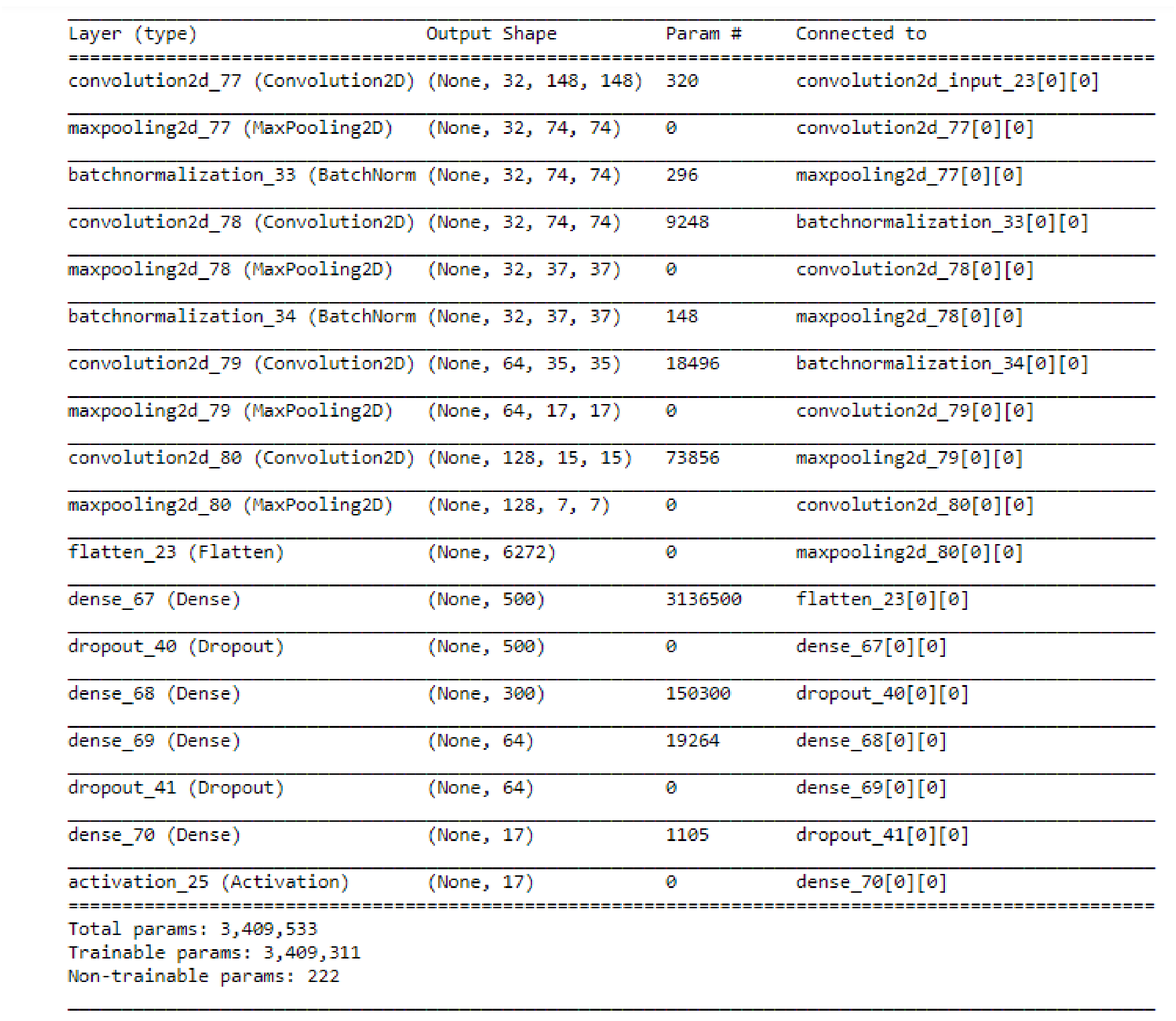
Refinement

Upon setting the baseline and achieving an accuracy of 83 % on the training set. I modified the CNN model to get better accuracy with the limited dataset that I had.

Inspired from Alexnet, I added Batch Normalization layers to my CNN model as well as an additional Convolutional layer. I also added an additional Dense Layer with increased number of features in each Dense layer. Additional Dropout layers heavily throw out features in the final layers thus reducing Overfitting.

I made an additional change in the image pipeline to at least detect faces in the video even though the faces have not been trained using the model. I drew a bounding box around each face and a label if the face was identified with a confidence score of greater than 0.3.

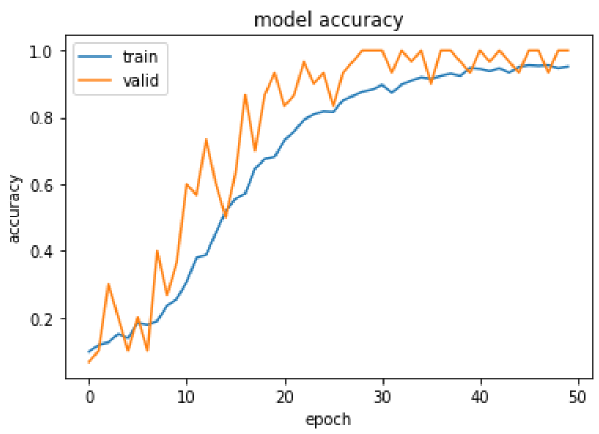
The architecture for the modified CNN model is as below:

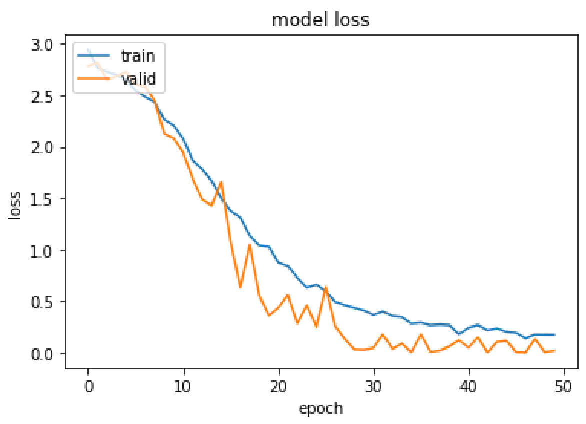


```
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
convolution2d_77 (Convolution2D) (None, 32, 148, 148)  320         convolution2d_input_23[0][0]
____________________________________________________________________________________________________
maxpooling2d_77 (MaxPooling2D)   (None, 32, 74, 74)    0           convolution2d_77[0][0]
____________________________________________________________________________________________________
batchnormalization_33 (BatchNorm (None, 32, 74, 74)    296         maxpooling2d_77[0][0]
____________________________________________________________________________________________________
convolution2d_78 (Convolution2D) (None, 32, 74, 74)    9248        batchnormalization_33[0][0]
____________________________________________________________________________________________________
maxpooling2d_78 (MaxPooling2D)   (None, 32, 37, 37)    0           convolution2d_78[0][0]
____________________________________________________________________________________________________
batchnormalization_34 (BatchNorm (None, 32, 37, 37)    148         maxpooling2d_78[0][0]
____________________________________________________________________________________________________
convolution2d_79 (Convolution2D) (None, 64, 35, 35)    18496       batchnormalization_34[0][0]
____________________________________________________________________________________________________
maxpooling2d_79 (MaxPooling2D)   (None, 64, 17, 17)    0           convolution2d_79[0][0]
____________________________________________________________________________________________________
convolution2d_80 (Convolution2D) (None, 128, 15, 15)   73856       maxpooling2d_79[0][0]
____________________________________________________________________________________________________
maxpooling2d_80 (MaxPooling2D)   (None, 128, 7, 7)     0           convolution2d_80[0][0]
____________________________________________________________________________________________________
flatten_23 (Flatten)             (None, 6272)          0           maxpooling2d_80[0][0]
____________________________________________________________________________________________________
dense_67 (Dense)                 (None, 500)           3136500     flatten_23[0][0]
____________________________________________________________________________________________________
dropout_40 (Dropout)             (None, 500)           0           dense_67[0][0]
____________________________________________________________________________________________________
dense_68 (Dense)                 (None, 300)           150300      dropout_40[0][0]
____________________________________________________________________________________________________
dense_69 (Dense)                 (None, 64)            19264       dense_68[0][0]
____________________________________________________________________________________________________
dropout_41 (Dropout)             (None, 64)            0           dense_69[0][0]
____________________________________________________________________________________________________
dense_70 (Dense)                 (None, 17)            1105        dropout_41[0][0]
____________________________________________________________________________________________________
activation_25 (Activation)       (None, 17)            0           dense_70[0][0]
====================================================================================================
Total params: 3,409,533
Trainable params: 3,409,311
Non-trainable params: 222
____________________________________________________________________________________________________
```

With the modified algorithm, I achieved a 96% accuracy on training data and near 90% on validation data.

A plot of the losses is shown below:





Note: The training accuracy is less than validation accuracy due to the use of dropouts in the layer. Usually dropout is activated when training but deactivated when evaluating on the validation set. Besides, the training loss is the average of the losses over each batch of training data. Because your model is changing over time, the loss over the first batches of an epoch is generally higher than over the last batches. On the other hand, the testing loss for an epoch is computed using the model as it is at the end of the epoch, resulting in a lower loss.

IV. Results

Model Evaluation and Validation

The modified CNN model was chosen as a facial recognition algorithm as it surpassed the base model in accuracy.

To verify the robustness of the model, an additional test was performed with the two models to check the improvement gain. These models were tested on a dataset downloaded from the internet as discussed previously. This dataset consists of 200 images over 5 classes and approximately equal number of images per class. As these images are separate than the ones used to train and validate, it is an unbiased dataset which has been used to test the model.

The metric chosen to compare the Base CNN model and Modified CNN model was F1 score.

<u>F1 SCORE:</u>

Base Model(Benchmark Model): 89%

Modified CNN Model: 91%

Justification

I used F1 score as my metric while comparing the two algorithms over this test data.

The F1 score achieved by modified CNN model was 91 % and by the base CNN model was 89%.

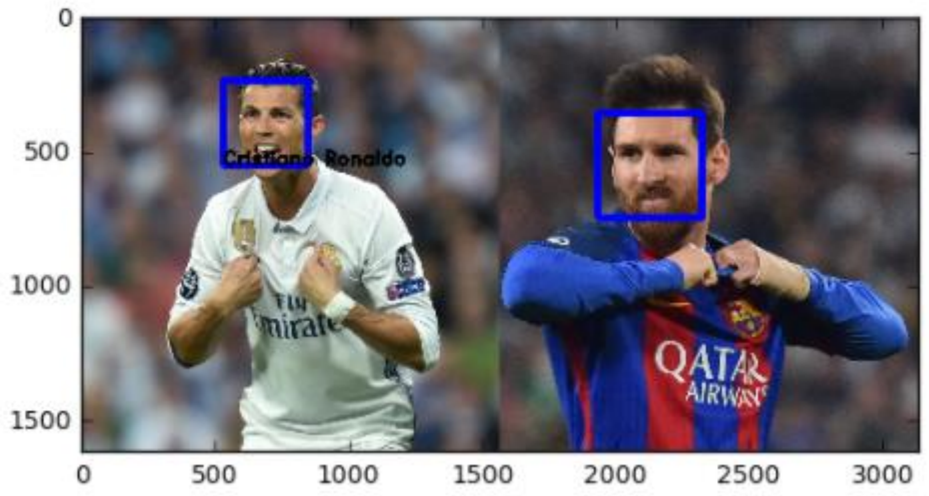
In the video, the algorithm recognizes most of the celebrities successfully and put out their name right beneath their faces. It is also capable of multiple face recognition. The code has been optimized so that it doesn't take time to process each image in the video and thus can be used over a live feed.

There are some cases where the model finds it difficult to identify but it can easily be overcome with some more data possibly with more side faces.

V. Conclusion

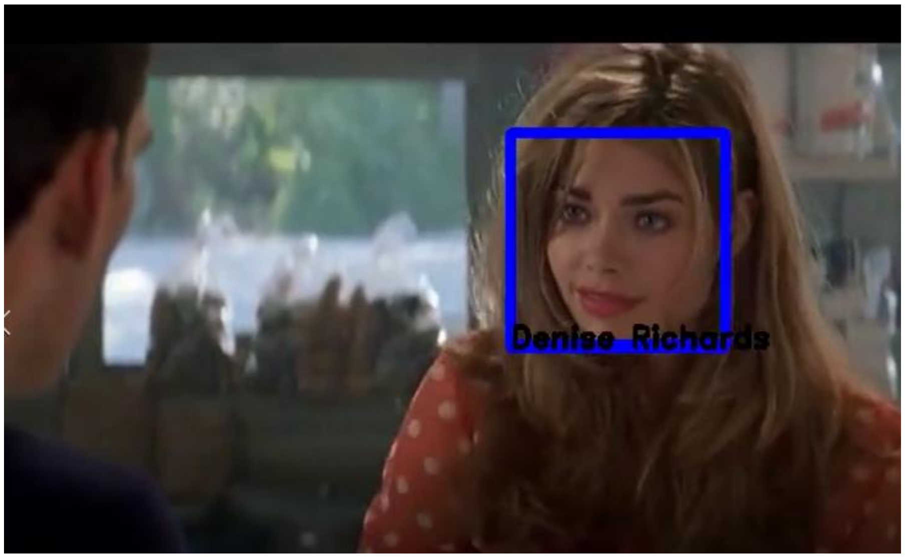
Free-Form Visualization

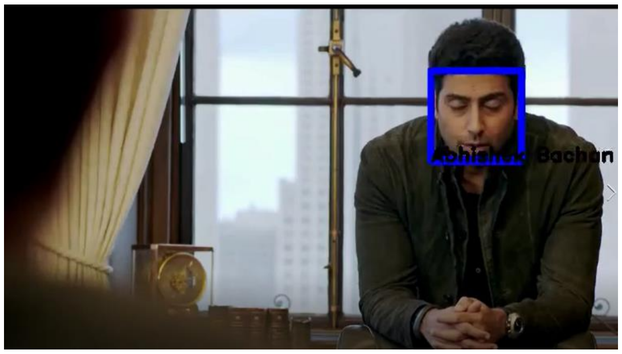
The algorithm works form any resolution of image and performs reliably and consistenly across images or videos. There is an image pipeline as well as a video pipeline which can be used depending on the purpose.

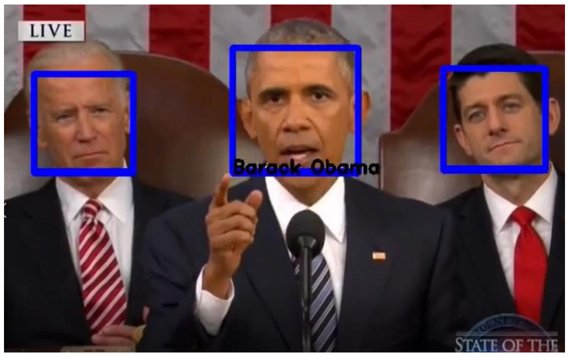


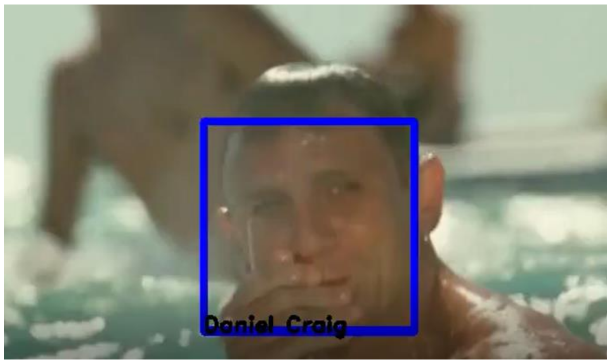
From this image above, the model works efficiently over this 3000 x 1600 pixel image with 2 celebrities with different face sizes. It labels the celebrity who was trained using the algorithm and at least detects the other celebrity face even though it does not label it.

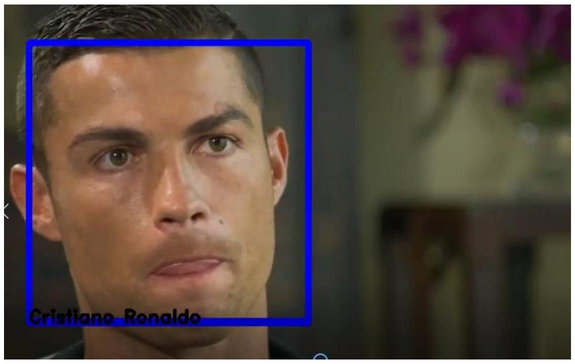
Some grabs from the movie clip:

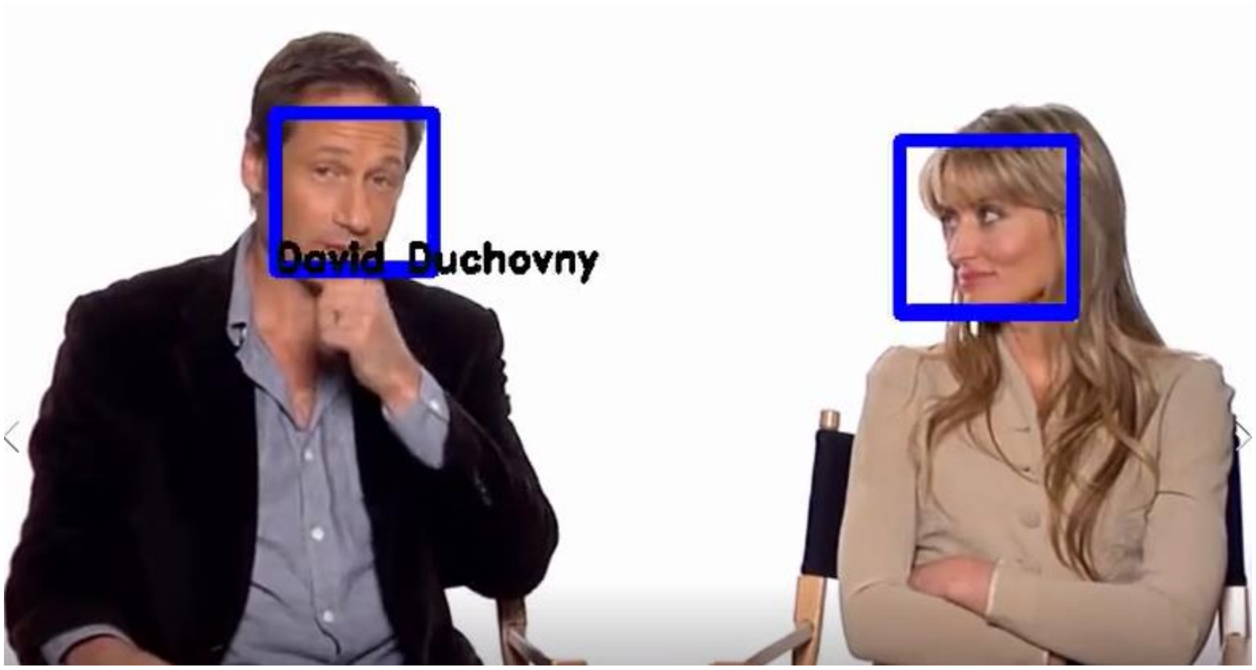












Failure cases where the model has some difficulty

1) Side Face Profile
2) Some obstruction in front of the face
3) Very Blurry images which makes it difficult for the model to classify

Reflection

The steps taken to complete this project can be summarized in the following steps:

1) Public dataset of celebrity images was found

2) Data was downloaded, cropped, aligned and preprocessed

3) Baseline CNN model was created as a baseline

4) The model was modified to give better accuracy with less data

5) Neural test data was collected from the internet to test both model and justify the improvement

6) Image and Video Pipeline was created to test the algorithm over a video stream

The most challenging as well as interesting steps were the 3rd and 4th step. This included getting familiar with TensorFlow, Keras, building CNN models as well as modifying them and tuning them. Video Pipeline was also a challenging task to detect multiple faces and recognize high confidence faces.

One of the initial challenges was the implementation of Alexnet. I quickly realized Alexnet was computationally expensive and required additional resources. While testing this architecture, I had to reduce the resolution of images to 75 x 75 pixels and could only feed in 1 image at a time to the fit generator. Reducing the resolution caused many of the visible facial features to disappear. So I resorted to train a CNN model.

Improvement

Some of the issues with the model can be resolved with training on more data as well as train on data with side faces or limited number of features like just the eyes and nose or nose or mouth.

Besides this, the model can also be used in mobile applications to identify celebrities by pointing your camera towards the TV.

Another major improvement could be brought about by using Alexnet which is a very powerful algorithm for image classification.