In designing our face recognizer, we’ve used Very Deep Convolutional Networks for Large-Scale Image Recognition with an aim to achieve high accuracy. Because of the absence of large enough training data set and the processing power, we decided to rely on transfer learning and have used weights that had been extracted by the VGG team at Oxford in their VGG16 model. Our model works on the principles developed in the making of AlexNet.

Our CNN model accepts colored images of dimensions 224 X 224.

In each stage of the network, we have used 2 convolution layers convolving with more and more kernels in each stage. We have used the kernel of fixed size 3X3.

To prevent image shrinkage and in turn, loss of details because of convolutions, we also add a Zero Padding at each layer.

For adding the non-linearity to the model, we have decided to use rectified linear unit (**ReLU**) because of its advantages over other non-linearity functions such as the sigmoid function and the tanh function. Because of being linear, it does not suffer from some of the limitations of sigmoid and tanh functions such as vanishing and exploding gradients, which could in turn lead to dead neurons.

At the end of each stage, we use pooling (MaxPooling in our case) to combine the outputs of neuron clusters at one layer into a single neuron in the next layer for developing more complex features.

The last stage of our training network uses a fully connected layer, in which we create a dense layer followed by 50% dropout to reduce overfitting.

To prevent training the convolution layers, we use the weights developed by the VGG16 team.

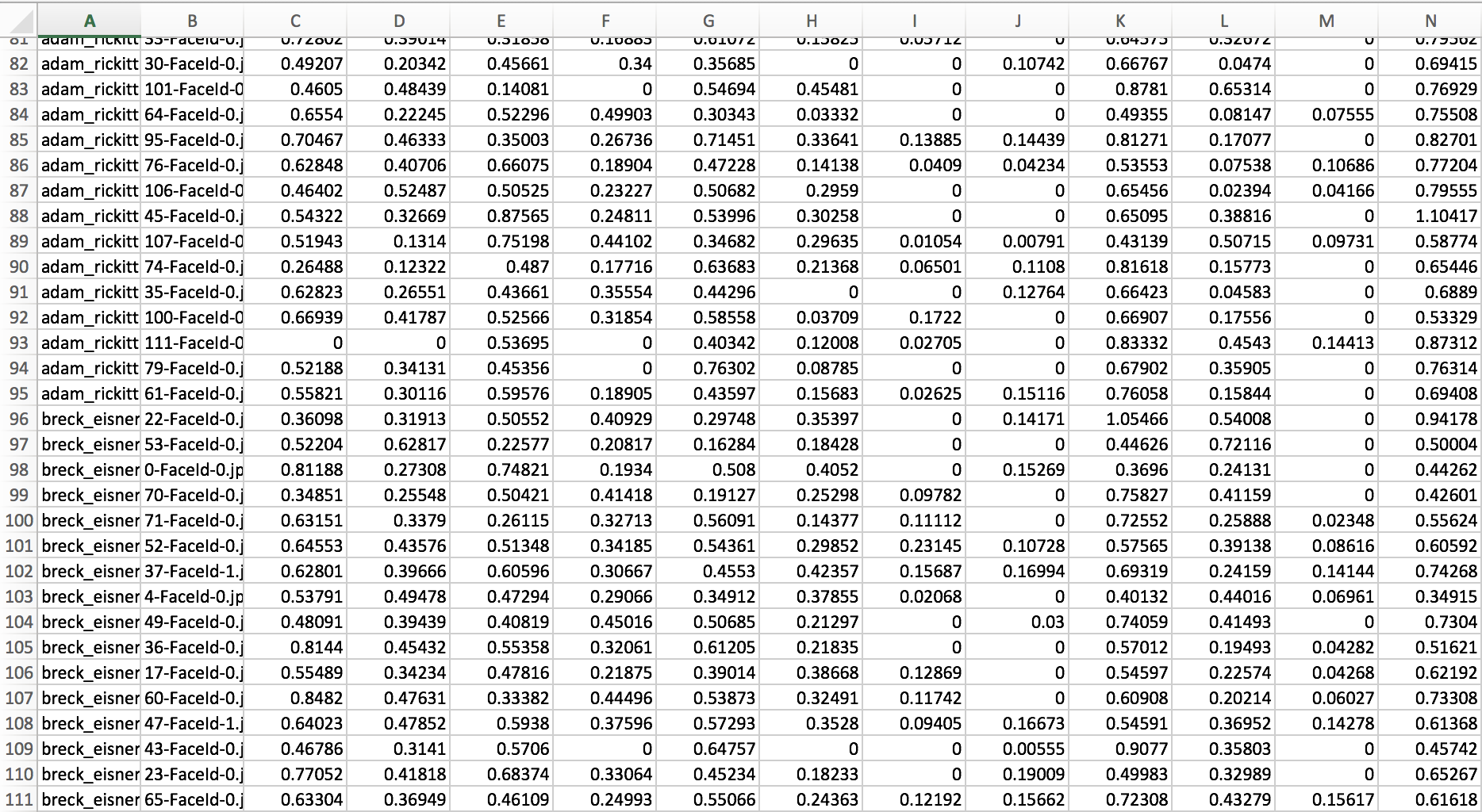
Our model employs Stochastic Gradient Descent for convex optimization.

**What do we do after extracting 4096 facial features?**

Once we have the base features to feed as training data, we use this data on a model of our choice(based the best possible model for the given variables) and train the system with our training data.

In our project, we have used K nearest neighbor model to predict the accuracy of the test data against the training data we already have.

Training Dataset containing facial extractions of 13 different celebrities



**What is KNN?**

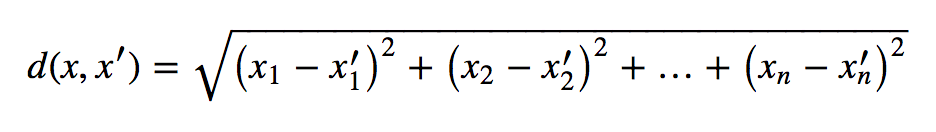
K nearest neighbor falls under supervised learning family. In this, we give a labelled dataset with training observations(x,y) and we can capture the relationship between x and y.

The goal is to learn a function h : X -> Y, so that given an unseen observation x, h(x) can confidently predict the corresponding output y.

**More about KNN:**

The working of KNN algorithm is based on forming majority vote between K most similar instances to a test observation.

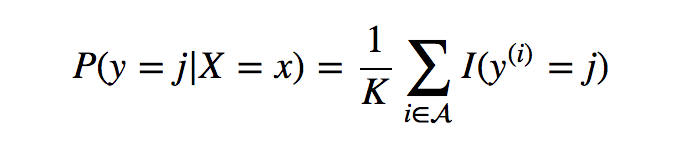
This similarity is calculated based on the distance metric between the 2 data points. The Euclidean distance is given by,



Other measures like Hamming, Chebyshev and Manhattan distance can also be used based on suitable settings.

KNN performs the following 2 steps:

* It computes distance d between x (for all values of the test) and each training data and store those K points in the training data that are closest to x in a set U.
* Then, for each points in U, it estimates the conditional probability given by:



Note: We usually select K as an odd integer to avoid tie.

In the end, the test input which is being queried (x) gets assigned to a class based on the largest probability.

The main criterion in KNN that decides the accuracy of prediction is the K value. Selection of K to be small can lead to forcing the classifier to be restricted and blind to overall distribution. Whereas larger K value will make the decision boundaries smooth, thereby making lower variance and increased bias.

**Use of KNN on our Dataset:**

We used python implementation of KNN from the library sklearn.neighbors. In our implementation, we used numpy for data extraction from the csv file and to convert it into array as required by the KNN functions.

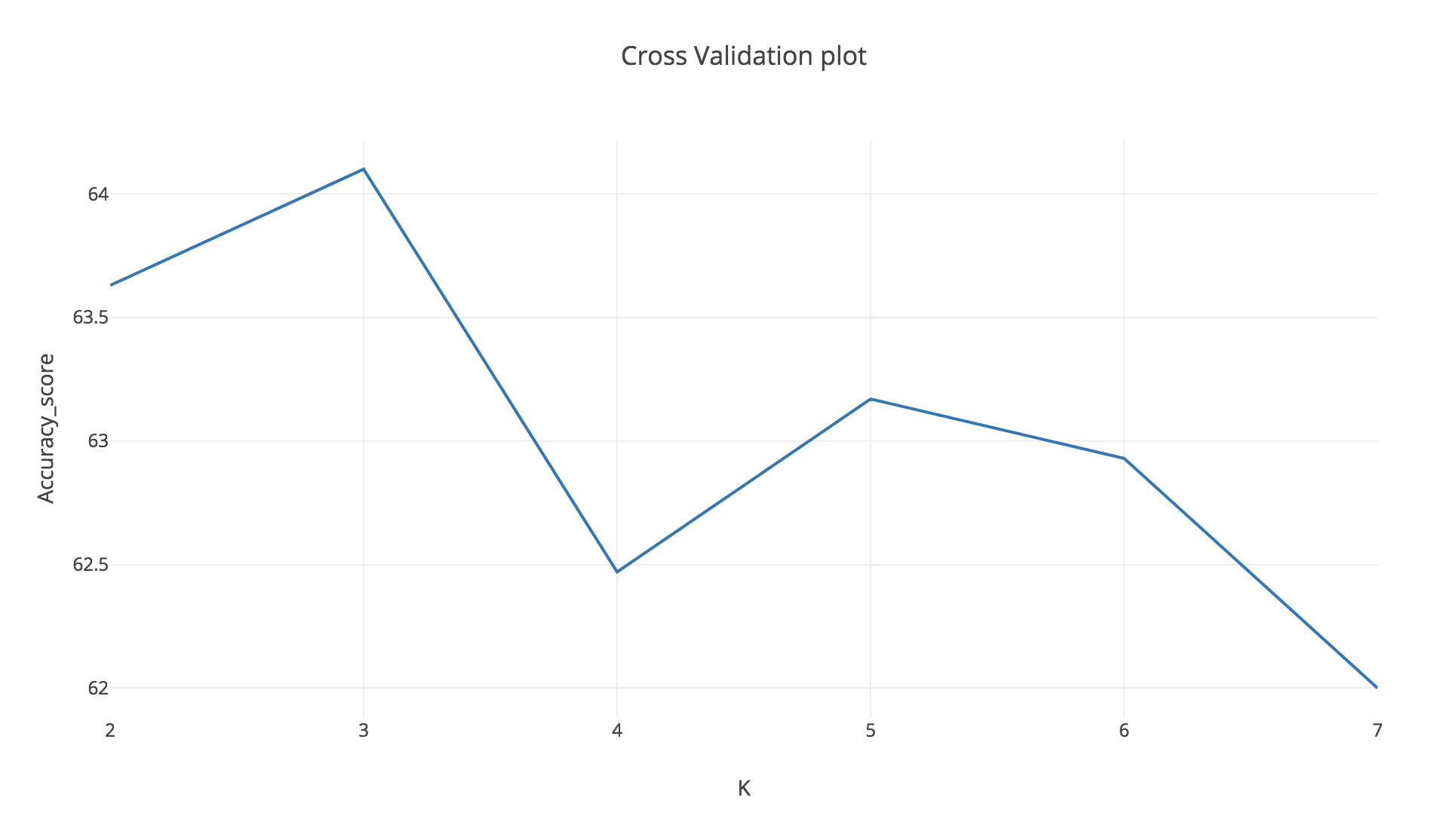
**Steps in implementation:**

* Extract the training data from the csv file containing the 4096 features extracted before and create a 2D array (X\_train) and 1D array containing the respective classes (Y\_train).
* Feed these arrays containing training data to the KNN library function fit, to fit the model and train the system.
* Create a 2D array of the test data (X\_test) with the corresponding features and feed it to the predictor to predict the accuracy of our prediction.

After these steps, we calculate the accuracy of our prediction. We used the accuracy calculator given by the skylearn library to calculate the accuracy, which came out to be 63% with the use of KNN.

**Optimization:**

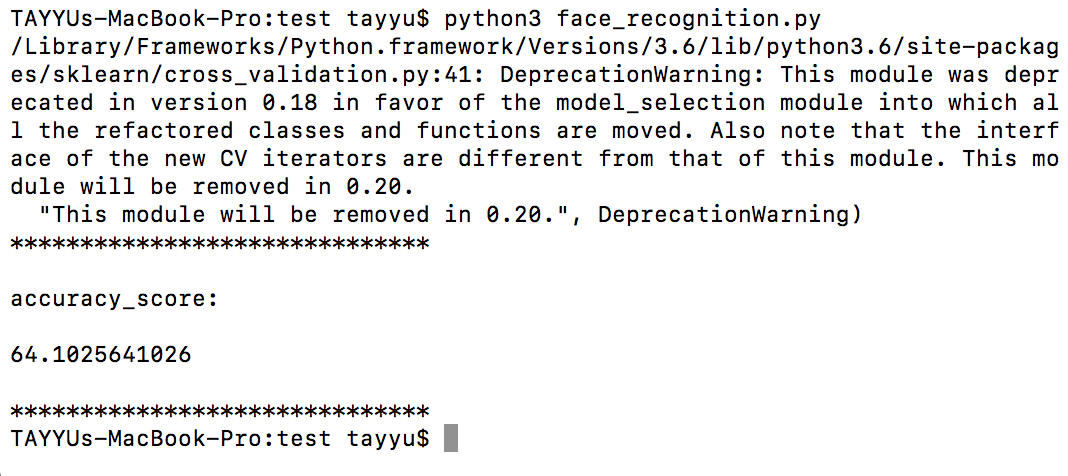
The accuracy scores we get may not be accurate. We may get better prediction based on the value of K. By parameter tuning with cross validation, we observed different prediction scores based the varying K value.



**Results:**

Using the training data from the dataset by Microsoft from their 1 million celebrity challenge, KNN model with K value equal to 3 gave us the best accuracy score of 64.1%.

Our Result screenshot



**Conclusion:**

There are numerous models and algorithms for facial recognition out there and KNN is one among them. With the Supervised learning, the prediction accuracy depends on how well the system is trained, which in term depends on the training dataset availability. More the number of data available for training, better the accuracy of prediction.