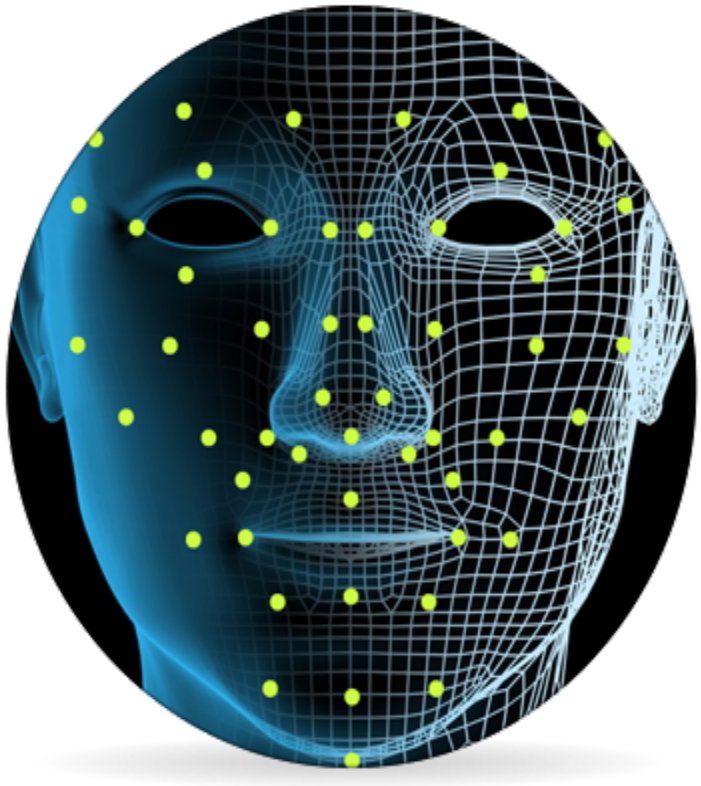
CSE547 Discrete Mathematics Fall 2017

FACIAL FEATURE EXTRACTION AND FACE RECOGNITION

Submitted by: Rahul Doshi(111452163), Santhwana Santhwana(111464609), Shreyas Harisha(111464609)



Introduction

A picture can say what thousand words fail to express. In today's world of ever growing digital image count, we can extract more information from these data than we possibly can imagine. In this project, we have tried to use images already captured, to extract 4096 facial facial extracts and trained the system to recognize images of 15 celebrities.

APPROACH

In this project, we have used CNN network which uses VGG16 weights for creating facial feature extracts and KNN algorithm for face recognition.

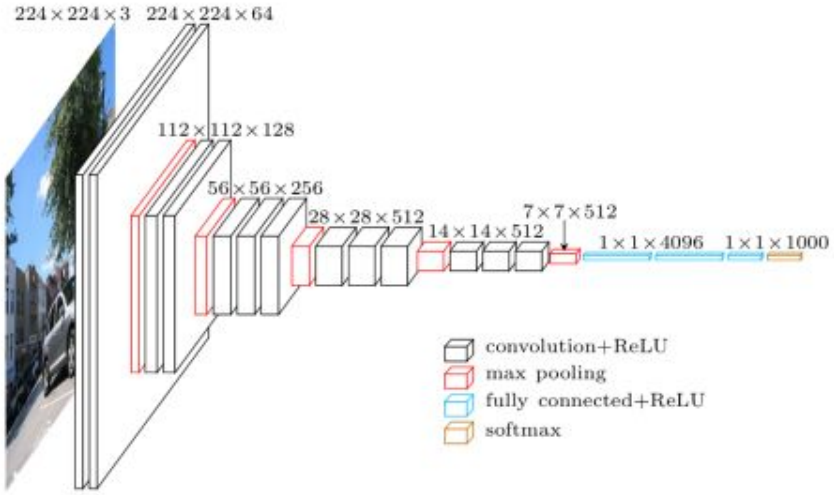
CNN NETWORK USING VGG16 WEIGHTS

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 have used the weights developed by the VGG16 team.

Our model employs Stochastic Gradient Descent for convex optimization.

VGG16 ARCHITECTURE



FACIAL FEATURES EXTRACTION

We have used the dataset from Recognize One Million Celebrities challenge by Microsoft. We have used the reference code from keras team for forming 4096 facial extracts.

- We extracted these features to csv file, which is being used in the next step for training our system and for face prediction.

AFTER EXTRACTION OF 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

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
82	adam_rickitt	30-FaceId-0.j	0.49207	0.20342	0.45661	0.34	0.35685	0	0	0.10742	0.66767	0.0474	0	0.69415
83	adam_rickitt	101-FaceId-0	0.4605	0.48439	0.14081	0	0.54694	0.45481	0	0	0.8781	0.65314	0	0.76929
84	adam_rickitt	64-FaceId-0.j	0.6554	0.22245	0.52296	0.49903	0.30343	0.03332	0	0	0.49355	0.08147	0.07555	0.75508
85	adam_rickitt	95-FaceId-0.j	0.70467	0.46333	0.35003	0.26736	0.71451	0.33641	0.13885	0.14439	0.81271	0.17077	0	0.82701
86	adam_rickitt	76-FaceId-0.j	0.62848	0.40706	0.66075	0.18904	0.47228	0.14138	0.0409	0.04234	0.53553	0.07538	0.10686	0.77204
87	adam_rickitt	106-FaceId-0	0.46402	0.52487	0.50525	0.23227	0.50682	0.2959	0	0	0.65456	0.02394	0.04166	0.79555
88	adam_rickitt	45-FaceId-0.j	0.54322	0.32669	0.87565	0.24811	0.53996	0.30258	0	0	0.65095	0.38816	0	1.10417
89	adam_rickitt	107-FaceId-0	0.51943	0.1314	0.75198	0.44102	0.34682	0.29635	0.01054	0.00791	0.43139	0.50715	0.09731	0.58774
90	adam_rickitt	74-FaceId-0.j	0.26488	0.12322	0.487	0.17716	0.63683	0.21368	0.06501	0.1108	0.81618	0.15773	0	0.65446
91	adam_rickitt	35-FaceId-0.j	0.62823	0.26551	0.43661	0.35554	0.44296	0	0	0.12764	0.66423	0.04583	0	0.6889
92	adam_rickitt	100-FaceId-0	0.66939	0.41787	0.52566	0.31854	0.58558	0.03709	0.1722	0	0.66907	0.17556	0	0.53329
93	adam_rickitt	111-FaceId-0	0	0	0.53695	0	0.40342	0.12008	0.02705	0	0.83332	0.4543	0.14413	0.87312
94	adam_rickitt	79-FaceId-0.j	0.52188	0.34131	0.45356	0	0.76302	0.08785	0	0	0.67902	0.35905	0	0.76314
95	adam_rickitt	61-FaceId-0.j	0.55821	0.30116	0.59576	0.18905	0.43597	0.15683	0.02625	0.15116	0.76058	0.15844	0	0.69408
96	breck_eisner	22-FaceId-0.j	0.36098	0.31913	0.50552	0.40929	0.29748	0.35397	0	0.14171	1.05466	0.54008	0	0.94178
97	breck_eisner	53-FaceId-0.j	0.52204	0.62817	0.22577	0.20817	0.16284	0.18428	0	0	0.44626	0.72116	0	0.50004
98	breck_eisner	0-FaceId-0.jp	0.81188	0.27308	0.74821	0.1934	0.508	0.4052	0	0.15269	0.3696	0.24131	0	0.44262
99	breck_eisner	70-FaceId-0.j	0.34851	0.25548	0.50421	0.41418	0.19127	0.25298	0.09782	0	0.75827	0.41159	0	0.42601
100	breck_eisner	71-FaceId-0.j	0.63151	0.3379	0.26115	0.32713	0.56091	0.14377	0.11112	0	0.72552	0.25888	0.02348	0.55624
101	breck_eisner	52-FaceId-0.j	0.64553	0.43576	0.51348	0.34185	0.54361	0.29852	0.23145	0.10728	0.57565	0.39138	0.08616	0.60592
102	breck_eisner	37-FaceId-1.j	0.62801	0.39666	0.60596	0.30667	0.4553	0.42357	0.15687	0.16994	0.69319	0.24159	0.14144	0.74268
103	breck_eisner	4-FaceId-0.jp	0.53791	0.49478	0.47294	0.29066	0.34912	0.37855	0.02068	0	0.40132	0.44016	0.06961	0.34915
104	breck_eisner	49-FaceId-0.j	0.48091	0.39439	0.40819	0.45016	0.50685	0.21297	0	0.03	0.74059	0.41493	0	0.7304
105	breck_eisner	36-FaceId-0.j	0.8144	0.45432	0.55358	0.32061	0.61205	0.21835	0	0	0.57012	0.19493	0.04282	0.51621
106	breck_eisner	17-FaceId-0.j	0.55489	0.34234	0.47816	0.21875	0.39014	0.38668	0.12869	0	0.54597	0.22574	0.04268	0.62192
107	breck_eisner	60-FaceId-0.j	0.8482	0.47631	0.33382	0.44496	0.53873	0.32491	0.11742	0	0.60908	0.20214	0.06027	0.73308
108	breck_eisner	47-FaceId-1.j	0.64023	0.47852	0.5938	0.37596	0.57293	0.3528	0.09405	0.16673	0.54591	0.36952	0.14278	0.61368
109	breck_eisner	43-FaceId-0.j	0.46786	0.3141	0.5706	0	0.64757	0	0	0.00555	0.9077	0.35803	0	0.45742
110	breck_eisner	23-FaceId-0.j	0.77052	0.41818	0.68374	0.33064	0.45234	0.18233	0	0.19009	0.49983	0.32989	0	0.65267
111	breck_eisner	65-FaceId-0.j	0.63304	0.36949	0.46109	0.24993	0.55066	0.24363	0.12192	0.15662	0.72308	0.43279	0.15617	0.61618

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,

$$d(x, x') = \sqrt{(x_1 - x_1')^2 + (x_2 - x_2')^2 + \ldots + (x_n - x_n')^2}$$

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:

$$P(y = j|X = x) = \frac{1}{K} \sum_{i \in \mathcal{A}} I(y^{(i)} = j)$$

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).

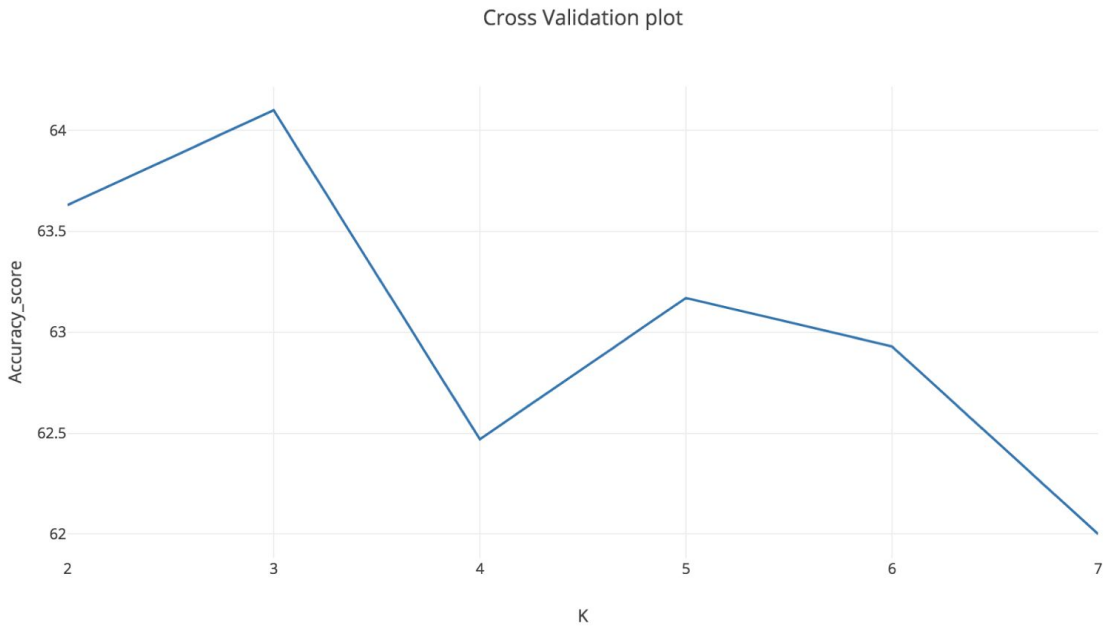
· Feed these arrays containing training data to the KNN library function fit, to fit the model and train the system.

· 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.



RESULT

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

```
TAYYUs-MacBook-Pro:test tayyu$ python3 face_recognition.py
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-packag
es/sklearn/cross_validation.py:41: DeprecationWarning: This module was depr
ecated in version 0.18 in favor of the model_selection module into which al
l the refactored classes and functions are moved. Also note that the interf
ace of the new CV iterators are different from that of this module. This mo
dule will be removed in 0.20.
  "This module will be removed in 0.20.", DeprecationWarning)
**************************

accuracy_score:

64.1025641026

**************************
TAYYUs-MacBook-Pro:test tayyu$
```

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.

INDIVIDUAL CONTRIBUTION

1. Research papers review: Santwana Santwana
2. Facial Features Extraction: Rahul Doshi
3. KNN implementation : Rahul Doshi, Shreyas Harisha
4. Training System and Accuracy: Shreyas Harisha
5. Report: Santwana Santwana, Rahul Doshi, Shreyas Harisha

REFERENCES

- https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/
- http://cs231n.github.io/convolutional-networks/
- https://github.com/keras-team/keras
- http://www.msceleb.org/celeb1m/1m