Machine Learning Engineer Nanodegree

Capstone Project

Rashid Kamran
May 16, 2018

Proposal https://github.com/rkamran/MLE/blob/master/capstone/proposal.pdf

---

I. Definition

Project Overview

I recently came across an interesting study "A Diverse Benchmark Dataset for Multi-Paradigm Facial Beauty Prediction (FBP)" [1]. The study mainly assessed the facial attractiveness that is consistent with human perception.

In my project I want to replicate the results of the study and take it fun step further to deploy the trained model to an iOS app that would help people capture best of their faces (aka selfies) using an indicator on the screen.

Problem Statement

This is a classic image analysis problem but rather than classifying we are assigning a numerical score between 0-5 to measure attractiveness based on the faces in the dataset. Higher score perceived to be more attractive based on the score provided in the dataset. The utility of the original regression seems limited as it would be unwise to put a score on a face but by using the technique I will find a way to help people capture their best-looking faces.

Metrics

For a simple regression albeit on image data we will be relying on MSE evaluation metrics. I will try to compare against study's results using 90/10 train/test split and 90/10 train/validation split during training.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

For anecdotal proofs I have also visualized the results of the model using some celebrities, personal, family and friend's photos.

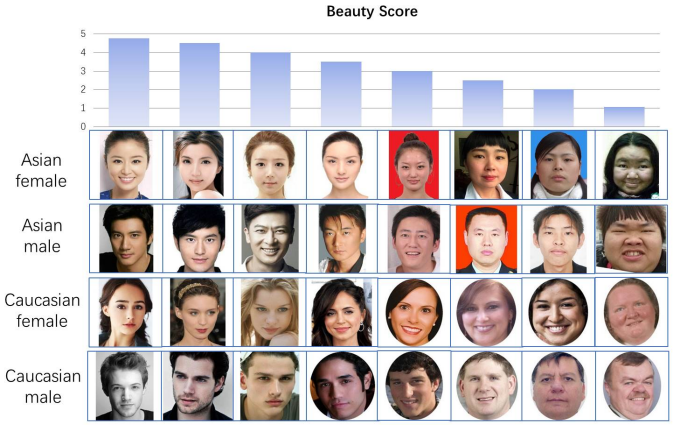
II. Analysis

Data Exploration

For this project I am relying on the original dataset [3] of 5500 face images (Asians and Caucasian male/female). Every face has given a score between 0-5 of perceived attractiveness. All images are 350x350 in shape. Out of 5500 images 4000 faces are Asian(male/female) so it would be interesting at least anecdotally to see the effect on other faces not represented in the dataset.

Here in Fig 1. I am showing an example from the original paper that shows how the dataset is actually distributed.

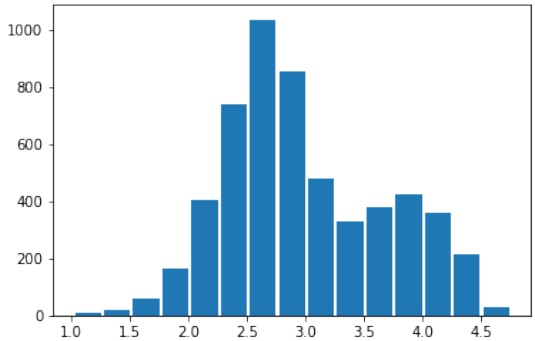
Fig 1. Scoring in the dataset



Exploratory Visualization

Below is a simple histogram of the scores in the original data. This gives us a glimpse of what kind of dataset we have in our hand. It makes as majority scores are regressing to the middle when it comes to attractiveness. It is important to note that there are some celebrity faces in the dataset and that might explain some medium towers around 4.0.

Fig 2. chart represents the score distribution in the data.



Algorithms and Techniques

For this particular problem I am going to split it into two problems. First feature extraction and for that I am planning to use fully trained ResNet50. This particular deep residual network is really good in extracting features from image data.

In the second part I am treating this problem as a simple regression problem. That will make it much easier to train for benchmark and custom solution.

Benchmark

I am going to establish a benchmark by using out of the box Support Vector Regressor and use it on top of extracted features. This will give me a baseline to meet or beat.

Please Note: Actual calculation is being done below after data preprocessing.

III. Methodology

Preprocessing

For dataset involving images it's always a good idea to preprocess, extract feature and serialize. This step will result in faster training time later. It would be much easier to create custom generator later.

`dataprocessor.py` in the source folder provide these utility methods. I am loading a batch of image, extracting features using ResNet50 and storing them into HDF5 file along with associated features.

Once completed all `5500x350x350x3`tensors will be saved as `5500x2048` features map in `./output/FaceDB.hd5f` file. These features are the output of the RestNet50 ImageNet dataset.

Here's a block of code that run the image data through ResNet50. It's a prety straight forward implementation and used Keras image processing utilities to load and convert images into 4D tensors.

```
    def extract_feature(image_files=[], input_shape=(350, 350, 3)):

        images = []

        for image_file in image_files:

            img = preprocess_input(img_to_array(load_img(image_file)))

            images.append(img)

        images = np.stack(images)

        model = ResNet50(weights="imagenet",
```

```
                    include_top=False,
                    input_shape=input_shape,
                    pooling="avg")
    return model.predict(images)
```

Implementation

The first thing I am doing is to establish a baseline result. The way I thought it's better to process is not involve feature extraction as part of the main solution and put the baseline and actual implementation on top of the extracted features.

Baseline

For establishing baseline, I am using `sklearn.svm.SVR`. Since we have already extracted features, so we can run it pretty quickly on a relative small dataset.

The loss function as I mentioned earlier will be Mean Squared Error for our baseline and proposed solution. It's the same old trusted function and can be described as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Dense Layer implementation

I am just adding a dense layer on top of the extracted features for simple regression. After running the ResNet50 feature extractor this problem has become very simple.

I am also using 90/10 split for taring and testing. The only difference is the 10% data is used as validation date during training. Notice how fast the training is due to the fact that the hard part has already been taking care of. I am using Titan XP for training.

Refinement

For optimization I have decided to use **`Adam`** optimization with a learning rate or **`0.0001`**. The results converge rather quickly probably because of the data size so to keep it going for at least **`20`** epochs I have also added a learning rate decay. Here's a complete compilation of the model

```
    model.compile(loss=loss,
                  optimizer=Adam(lr=lr, decay=lr/epochs),
                   metrics=["acc"])
```

IV. Results

Model Evaluation and Validation

For training and evaluating the model I initially did a 90/10 split to set aside 10% data. And during training further split the data into 90/10 to have epoch level validation. Both splits shuffle the dataset as well.

```
#1. train/test split

trainX, testX, trainY, testY = train_test_split(features, labels, test_si
ze=0.1, shuffle=True)

#2. Validation Split on remaining data. 80% then used to train

H = model.fit(trainX, trainY,

                  epochs=epochs,

                  batch_size=batch_size,

                  validation_split=0.1,

                  callbacks=[ModelCheckpoint(model_file,

                            monitor=loss", save_best_only=True)])
```

Support Vector classifier results -
`[INFO] Baseline Score (MSE) 0.03305848691971774`

Dense layer on top of ResNet50
`[INFO] Regression score (MSE) on test data 0.029269013553857803`

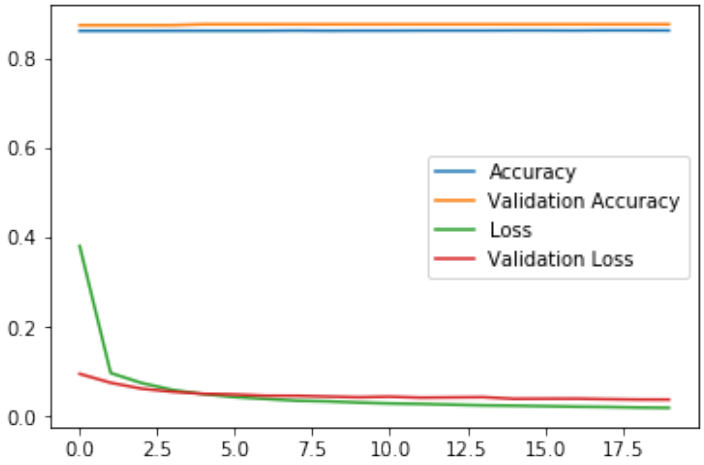
It's not an extraordinary improvement but might be slightly faster when used on edge devices.

Justification

The below graph provides a visualization of how 20 epoch training immediately converged and didn't improve much. I could easily stop the training after 5 epochs.

I believe well-structured data, a good feature extractor like ResNet50 and tilt towards mean immediately get to maximum accuracy which is better than the original study itself [6].

Fig 3. Chart showing the 20 epochs training and validation scores.



V. Conclusion

It's very hard to get similar dataset or create a large split for testing while the actual size is only 5500 so I am resorting to some anecdotal proofs by using some faces which I (eye of the beholder) personally think should get high/low score.

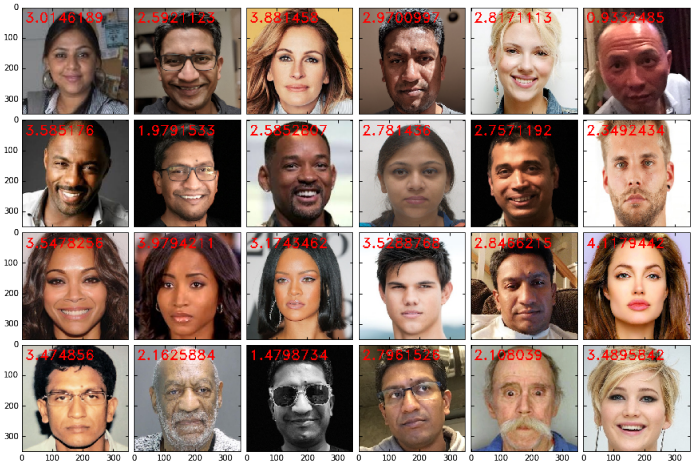
Free-Form Visualization

Anecdotal proof and visual reassurance

Fig 4 shows a grid of some personal, friends, random folks and celebrity photos. Model seems to be assigning high scores to celebrities and regressing back to mean when it comes to normal faces. In some cases, I tried to stylize and was able to get some good score.

Note that how **Angelina Jolie**, **Julia Roberts** and **Idris Elba**'s face got high score although there were no black faces in the dataset.

Fig 4. Scores generated on random faces



iOS App to also validate how model is working

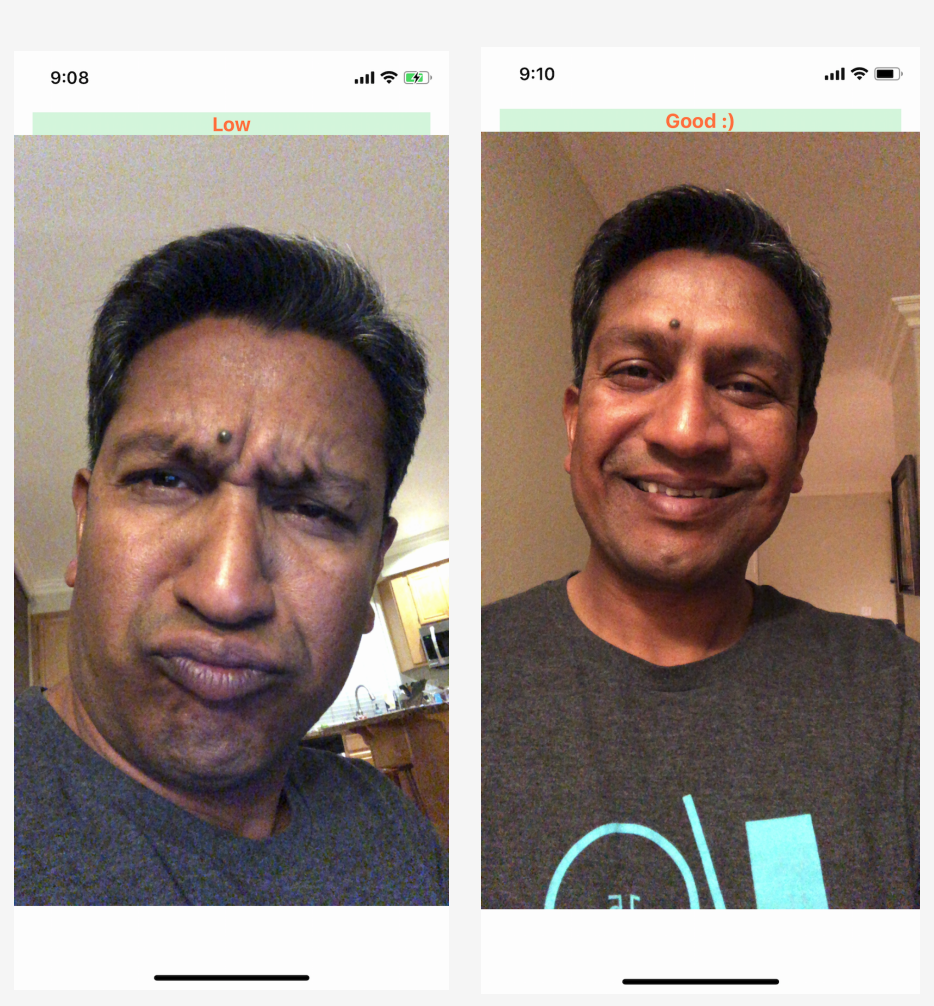
Note: The iPhone application source was also uploaded in the project repo [4]

Rather than providing a score I am classifying a selfie pose as `Low`, `GOOD` and `Awesome`. The idea is to help people capture best faces when the beholder agent things it looks attractive.

Here's a breakdown of categories
`Low < 2.5 2.5 > GOOD < 4.0 Awesome >= 4.0`

Here I am pretending to be frowny and happy and getting the treatment from the app accordingly. I run the model every 10 frames, so it updates pretty quickly as you adjust faces.

Fig 5. iOS screenshot in real time scoring



Reflection

This is a tough problem, first of all assigning a numeric score to attractiveness looks pretty random but then we can all agree that majority of the world is not Miss Universe and then it kind of make sense.

The second aspect which I found really interesting is that how good ResNet50 is in detecting features from an image. At least in my testing which I ran quite a few times to make sure that the's not a fluke it helped converge the model pretty quickly. I still see it as too good to be true scenario and looking for something I missed because my scores are way better than the study itself. That might be because of the upfront feature extraction.

The third thing which I learnt from Dr. Adrian Rosebrock of pyimagesearch that serializing activations after running through a complex model is way efficient and faster compared to running it in the main pipeline. It enabled me to experiment and play with other SKLearn elements without ever worrying about run time or memory issues.

Improvements

I believe the dataset is way too small. I will take it to next level by doing the following

1. Use similar data to create a more elaborate profile of what a healthy good-looking face should be. Of course, there was absolutely no diversity in the dataset and it was very biased towards what it knew
2. Combine the app with object detection technique and run it on multiple faces at the same time