# Only one good face per picture Project

Table of Contents

[Only one good face per picture Project 1](#_Toc26263555)

[1. Project outline 2](#_Toc26263556)

[1.1. Aim of the project 2](#_Toc26263557)

[1.2. Steps towards completing the project 3](#_Toc26263558)

[1.3. Tools used 3](#_Toc26263559)

[1.4. Issues 3](#_Toc26263560)

[1.4.1. Face locating issues 3](#_Toc26263561)

[1.4.2. Memory and performance issues 4](#_Toc26263562)

[2. Results and conclusions 5](#_Toc26263563)

[3. Sources 6](#_Toc26263564)

[3.1. Main sources 6](#_Toc26263565)

[3.2. Pages with useful info: 6](#_Toc26263566)

## Project outline

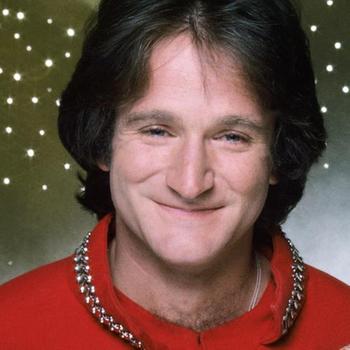
### Aim of the project

Make a neural network that will classify photographs into “good” and “bad” pictures, so the “good” pictures can be fed to my instructor’s (Joni Korpihalkola) age estimation program. The pictures are from an Internet Movie DataBase datadump. They are all supposed to be pictures of faces, but there are also a large number of unsuitable pictures, which don’t contain faces at all, or are taken from so far away as to not be useful for our purposes, just to name a few characteristics.

The “good” pictures or “keeps” as they are called here

* only contain one face
* the face of that one person is clearly visible (face landmarks, eyes and nose, included)

Example of “keeps”:

The “bad” pictures or “deletes” as they are called here

* contain no face or
* contain multiple faces

Examples of “deletes”:

As we’ll show in the section 2.4.1. Face locating issues, profile views of a face are sometimes problematic for face detection.

Characteristics of the model to avoid:

* Underfitting
  + if both training and testing accuracies are poor
  + if the testing accuracy is higher than the training one
* Overfitting (the likelier option)
  + if the training accuracy is higher than the testing accuracy

### Steps towards completing the project

1. get path to a directory
2. find all pictures where either the width or height is less than 200 and trash them
3. rest of the pics are resized into a suitable size and converted to RGB, save them
4. batch\_locations\_from\_pics will go through the pictures and determine keeps and deletes and produce a DataFrame
5. Since the previous step takes forever, let’s move the keeps and deletes at this point to their appropriate folders (temporary keep and (permanent) delete), so we don’t have to do this again in step 7
6. once that’s done, the pictures in the keep (temp) directory are run through “face\_landmarks\_check”
7. keeps are moved to the permanent keep directory and those that failed the face\_landmark check, are moved off the dataset completely (**this is important, because the failed ones are nearly identical to actual “keeps”, so to avoid confusing the model, they are removed from consideration**).
8. Once we have enough pictures in both classes (at least 1000 per class) and we have pictures to spare for testing, we can begin the actual model tinkering.
9. Create the model, tinker with the hyperparameters, run the model and adjust the hyperparameters accordingly.
10. Consider improving the dataset, if the model is performing poorly.
11. Once the model is producing desirable results, save it. Test it with new data.

### Tools used

* Tensorflow 2.0
  + Image classification
  + Neural networks
* Python’s face\_recognition library
  + face\_locations
  + batch\_face\_locations
  + face\_landmarks
* Python’s Pandas library
  + DataFrame
* Python’s PIL library
* Python’s Matplotlib.pyplot

### Issues

#### Face locating issues

Initially I intended to sort the pictures manually, and came up with a number of ways to sort the pictures. However, after realizing it was counter-intuitive to sort the pictures manually, I decided to use face\_recognition to help. At first I used face\_locations with its default model, hog, which didn’t find a face in the below picture. As the face in the picture is clearly visible, and I wanted the function to pick up a face in it, I then switched face\_locations’ model to use CNN model, which did pick up face in it.

After that, we came to the conclusion that sending thousands of photos to face\_locations one at a time was probably not the wisest use of resources. So, I moved on to use batch\_face\_locations, where I could send batches of pictures at a time.

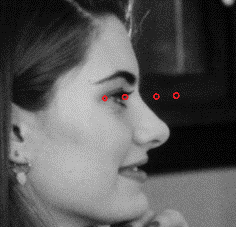
Soon I realized that batch\_face\_locations, and face\_locations in general, functions in a peculiar way. For instance, it does not find any faces in the picture on the left, but does find two faces in the picture on the right:



Similarly, batch\_face\_locations finds one face in both of the pictures below, but if my aim is to feed these pictures to Joni’s age estimation algorithm, these pictures are not very helpful.



I found a way to circumvent batch\_face\_locations’ undesirable way of functioning by, after running batch\_face\_locations, checking the pictures it had labeled “keep” for face\_landmarks: If it does find a nose and two eyes, we’ll “keep” the picture, if it does not, I’ll trash the picture, not to be confused with the label “delete”. I initially ran into trouble as I simply turned the failed “keeps” into “deletes”, but this understandably confused the model, as these pictures were nearly perfect “keeps” but still labeled as “deletes”. After removing the failed “keeps” completely from the dataset, the model improved considerably.

Face\_landmarks is not perfect, as it also sometimes finds landmarks that are not even there (check out the picture of Mädchen Amick for instance, where her left eye is supposedly just hanging there), but it is a nice way to trim “keeps” from the worst offenders. It is also a good opportunity to get rid of some of the profile view pictures, like the one below on the right, from which face\_landmarks can’t find a nose or eyes.

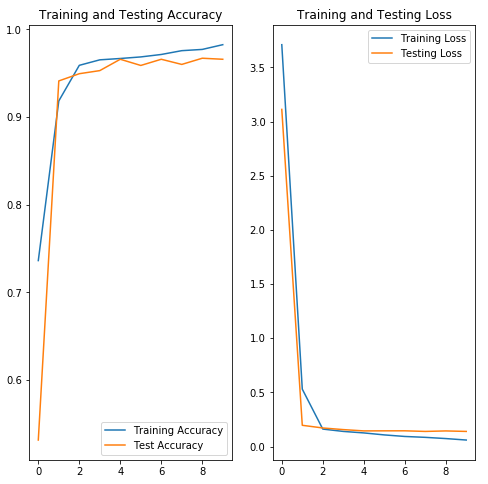
Despite its imperfections, face\_landmarks is definitely an improvement.

#### Memory and performance issues

The function I created for batch\_face\_location originally included too many unnecessary loops, mostly to create an easy-to-read DataFrame of the pictures. Due to the perceived memory drain, I couldn’t run the function on my work computer, so I sent it to my instructor so he could run it on his own computer. Stripping the function of the unnecessary loops helped, after which I could run the function on my work computer, but it would still use CPU instead of GPU which Tensorflow is supposed to use. Running the function would have been faster had I been able to use GPU.

## Results and conclusions

After concluding that two 0.2 dropout layers resulted in underfitting, and removing the dropout layers completely resulted in overfitting, I realized that the best result could be found between the two. So, I introduced two 0.1 dropout layers, which resulted in a rather good fit, illustrated by the two graphs below:



Given the time limit of 6 weeks, I believe this is the best result I can achieve with a model like this, and I’m quite pleased with it. When running model.evaluate on the saved model, its accuracy is 88.57%, with a loss of 0.6292. It is worse than with the original data, but that is to be expected.

Overall, improving and purifying the dataset took the most time compared to everything else during the project. The project lasted six weeks, from October 28th to December 5th in 2019. The first week consisted entirely of researching Tensorflow 2.0 and face\_recognition, then 4 weeks of completing the project and the remaining week was mostly spent testing, reporting and preparing a presentation about the project.

If I had more time, I would finetune the pictures even more before running them, maybe even test if the tip of the nose is between the two eyes, and only accept pictures where the face was looking directly ahead. With more time, I would also try out transfer learning and different types of neural networks, perhaps attacking the problem from a different angle.

## Sources

### Main sources

* Source of the images <https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/>
* Tensorflow How to prepare a directory: <https://www.tensorflow.org/tutorials/load_data/images>
* Tensorflow How to classify images <https://www.tensorflow.org/tutorials/images/classification>

### Pages with useful info:

* How to detect faces:
  + How to detect faces and draw rectangles on said faces <https://israelg99.github.io/2018-11-18-Best-Face-Recognition-in-Python-in-20-Minutes/>
  + Face\_recognition package on Python <https://face-recognition.readthedocs.io/en/latest/face_recognition.html>
  + OpenCV if it’s better than Face\_recognition <https://realpython.com/face-recognition-with-python/>
  + Face detect if It’s better than face\_recognition <https://github.com/shantnu/FaceDetect/blob/master/face_detect.py>
* TF 2.0 Image classification, username = theComputerScientist <https://www.youtube.com/watch?v=bNntsCOdFxg>
* How to fight overfitting <https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d>
* Moving files with shutil
  + <https://automatetheboringstuff.com/chapter9/>
* Building a dataset of our pictures
  + <https://www.pyimagesearch.com/2018/04/09/how-to-quickly-build-a-deep-learning-image-dataset/>
  + This part is deprecated, but I mainly take note of what to do with the directories of the pictures. <https://www.pyimagesearch.com/2018/04/16/keras-and-convolutional-neural-networks-cnns/>