Report

Case studies 2019

na_omit group

Motivation

Our project is used to see which actor you look alike. To do that we translated a dataset of actors' images to 128-dimensional vectors using deep learning. The application itself uses the image from the user's camera which is also translated to a vector (the same way the actors' photos were). When we have this vector we can find actor's vector that is closest to the user's one. Then the photo, that this actor's vector was made from, is returned as an output. We decided to conduct this experiment to find out if this solution works and by work we mean that it really returns the most look alike photo to the user's one. \ We wanted to test if the solution really works (returns most look alike actor) and if so then how stable it is (how often is the same, most similar actor returned for one person). To do that we decided to test it by sending images of actors that are in our databse - if it works then the same actor should be returned most of the time. Other part of the experiment is we measured how the amount of actors in database affects the outcome accuracy. We would expect that changing the amount of actors that are considered by our application could change the accuracy. Other part of this experiment is we took a photo of a person and put it in the databse, then let this person use the application and see how often his photo will be returned. \

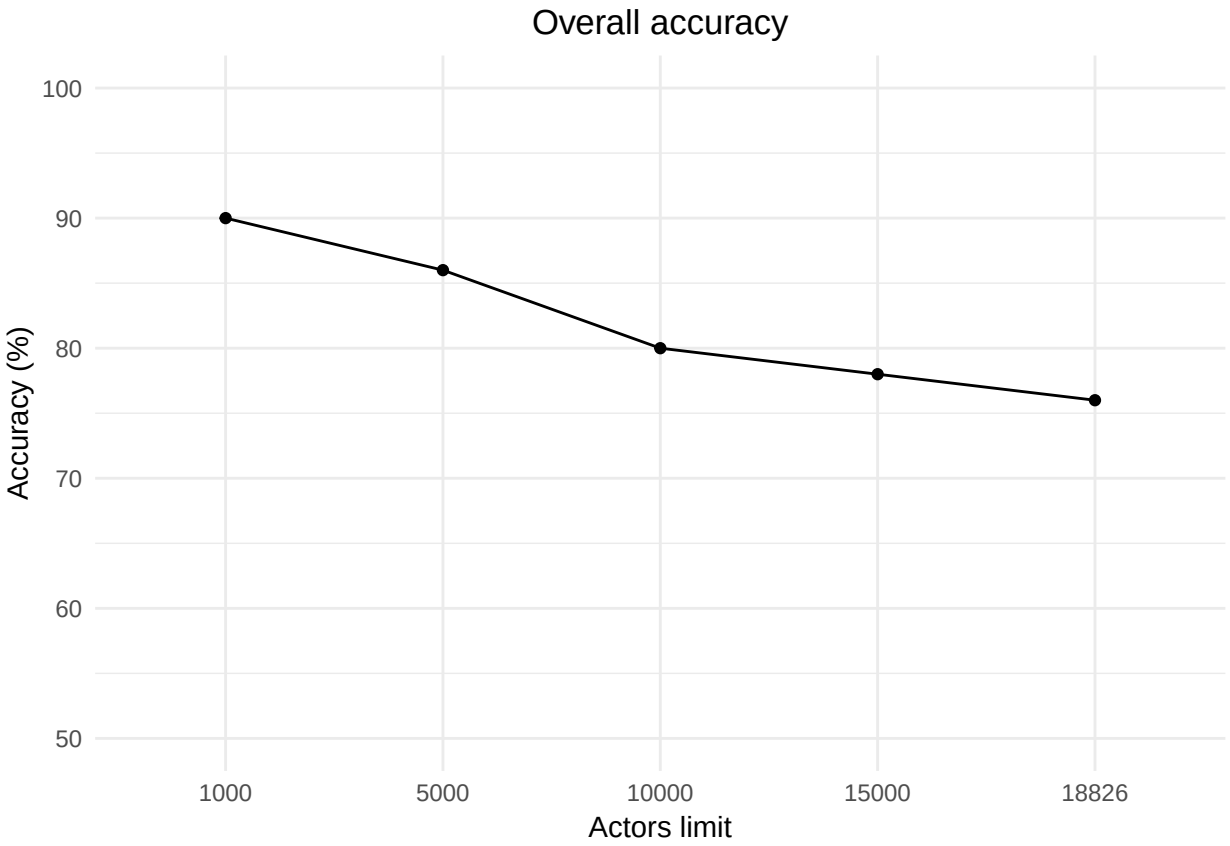
Experiment details

First we need to define what we mean by accuracy - we will send a photo of an actor, that we know that he/she is in our database and see if this actor is returned. The accuracy is a percent of times that the proper actor was returned to the amount of all tries. We tested the accuracy change by performing an experiment - we chose 4 actors and got 5 photos of each actor. We selected the photos that we knew didn't appear in the database. Each photo has been changed to make it look like it has been received from the webcam (image from webcam is not as sharp as a photo taken with phone/camera and our application will be using webcams, so we need to stimulate the inpure images it creates) - we had to lower the quality of the photos. Each photo was transformed 5 independent times by pipeline:

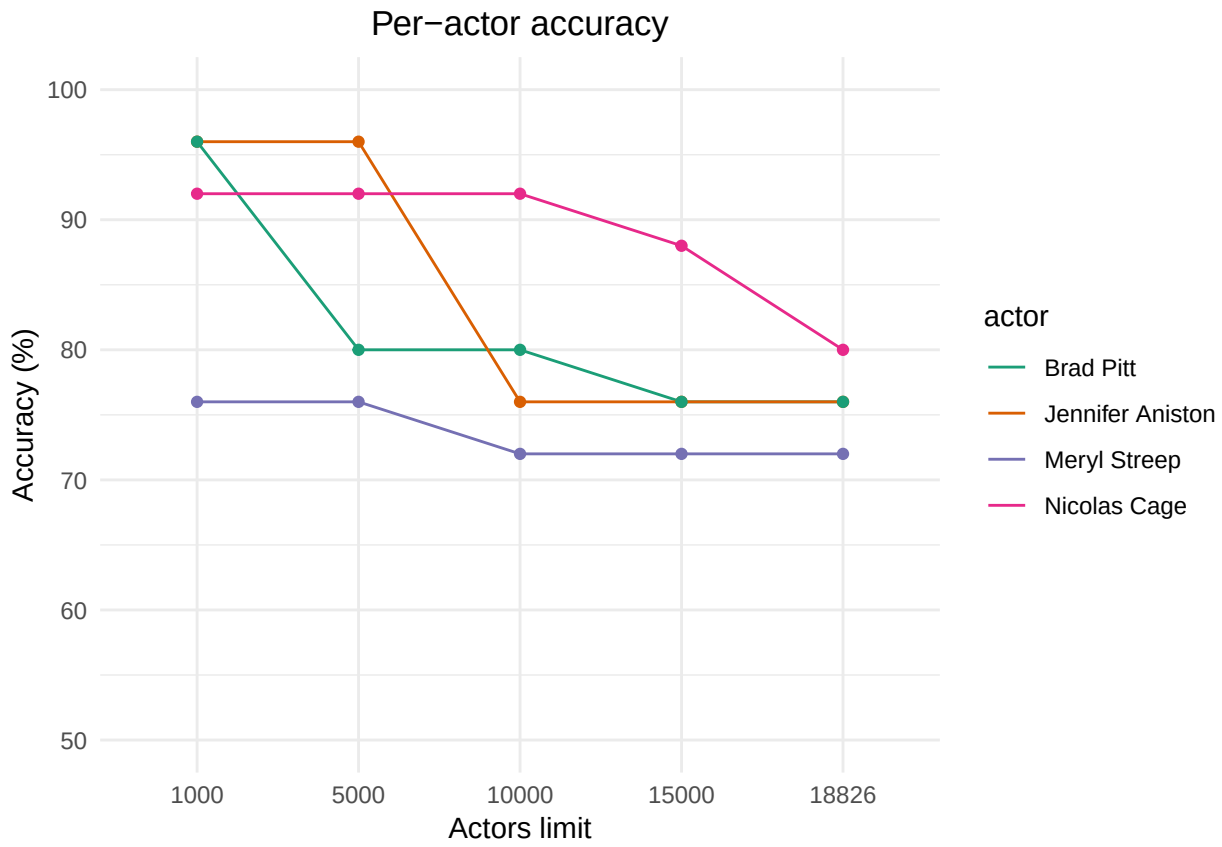
- Add random number form [-30, 30] to color of each pixel.
- Change contrast by scaling each pixel value to $255 * (I_{ij}/255)^{gamma}$, where gamma is random number form [0.7, 1.3].
- Degrade image quality by applying JPEG compression with random degree from [60, 85].

To achieve this we used python library *imgaug* and finally got a collection of 100 photos. Whole collection was processed 5 times, each time with different amount of actors (the values being 1000, 5000, 10000, 15000, all). A special version of the user interface has been prepared, allowing us to upload photos and read results. This code is on the branch test. The results of this experiment are below.//

Results



The chart abowe shows the overall accuracy (mean of accuracies for actors chosen to the experiment). The trend, as we expected, is downward. The more actors in the database, the lower accuracy is. However, the drop is not drastic - accuracy vary from 90 to 75, so the difference is 15% only, while the amount of actors taken in consideration grew more than 18 times - from 1000 in the beginning to 18826 at the end.



The chart above shows the accuracy of recognizing particular actors in the pictures. The general trend is downward, although the increase of the actors limit does not always lead to a decrease in accuracy.

Conclusions

As we expect changing the amount of actors that are considered by our application changes the accuracy. The more actors in the database, the lower accuracy is. Overall accuracy vary from 90 to 75, so its drop is rather low.