universität innsbruck

igs iis

Visual Computing Project Report
Smile detection using Viola-Jones features

Martin Beyer & Nicolas Marte & Islam Mechtijev

February 14, 2022

Introduction

Finding the best picture of a collection of group-photos is a strenuous task, with potential bias error introduction. Everyone should be smiling and seem happy for the best possible picture. To facilitate and automate this process, smile detection comes in handy. Paul Viola and Michael Jones introduced face-detection in the early 2000s [3]. Further work extended their algorithm to include different classifiers, such as eye- and smile-classifiers. This allows developers to find numerous features in faces.
The main idea of their algorithm is to cascade the classifiers. For facial features, first detect faces. If no face is found, drop the complete picture, otherwise search for the facial features.
The task of this project was to implement smile-detection on a given dataset. The dataset consists of various different faces and expressions. We propose two different solutions for the task. Firstly, we present a method using the Viola-Jones algorithm, but since our result of this provided a bad accuracy we decided to look into a second approach using a convolutional neural network, trained with the dataset. Here the accuracy turned out to about 91%. With this score, the neural network approach would be able to solve the mentioned group picture task.

Martin Beyer implemented the webcam smiling application and used his insights for pair-programming the OpenCV Haar Classification.
Nicolas Marte tried to parallelize the OpenCV Haar Classification approach. Unfortunately, this did not work out, so we had to stick to the serial implementation.
Islam Mechtijev trained a model using OpenCV tools and implemented a different solution using TensorFlow/Keras. Most of the project, however, was done while pair-programming using the Live-Share feature of Visual Studio Code. In addition, the report was finalized by all of us together in two separate sessions.

Background

Before we were immediately looking for smiles, we had to classify whether a face is in a picture or not. We achieved this using the Viola-Jones algorithm. Paul Viola and Michael Jones developed this method in the early 2000s, using the OpenCV library. Viola-Jones isn't just a typical algorithm. It's a learning algorithm, so-called machine-learning. "For the task of face detection, the initial rectangle features selected by AdaBoost are meaningful and easily interpreted. The first feature selected seems to focus on the property that the region of the eyes is often darker than the region of the nose and cheeks." In addition, the eyes are darker than the bridge of the nose. This way, the second feature is obtained [3]. Besides feature detection, Viola-Jones is using the schematic depiction of a so-called detection cascade. This is a series of classifiers, which are applied to every sub-window. As soon as a certain sub-window is rejected, the process terminates. Using this method increases the effectiveness massively, as seen in Figure 1. After this process, the images are classified. They either contain a face or not. Using different cascade-classifiers allows the programmer to determine what features should be detected in the collection of sample pictures. In our case, we were using smile cascade-classifiers to decide whether a person was smiling or not.

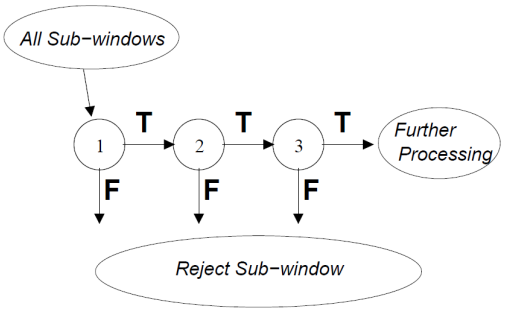

Figure 1: The process of the detection cascade [3]

Method

First we implemented a rather simple version of a Viola-Jones cascade classifier which takes a video-stream from a webcam. For this first experiment, we followed the OpenCV Tutorial on Cascade Classifiers [2], but changed the eye classifier to the Haar-cascade smile classifier provided by OpenCV. The source-file of this program is "`liveSmileDetection.py`".
With our first hands-on experience on how the Viola-Jones classifier works, we started on the actual task to classify images of the given dataset, whether a person smiling or not. Initially, we used the OpenCV provided Haar-cascade smile classifier.

Since all our images were already cut out to the face only, we didn't need to cascade the classifiers. This way we were able to perform a smile classification on every image. Then we were able to compare our classification with the label of the image. Using this method, we counted the True Positives and vice versa.

See following code snippet:

```
smiles = smileCascade.detectMultiScale(frame)
if type == 'positive':
    self.y_true.append(1)
    #True Positives
    if len(smiles) > 0:
        self.y_pred.append(1)
    #False Negative
    else:
        self.y_pred.append(0)
elif type == 'negative':
    self.y_true.append(0)
    #False Positive
    if len(smiles) > 0:
        self.y_pred.append(1)
    #True Negative
    else:
        self.y_pred.append(0)
```

Our assumption was already a correct labeling of the images by their provided directory structure. This means that all images with people smiling were in the "positives" directory, while all images with non-smiling people were in the "negatives" directory. However, for example, the picture "`5.jpg`" in Figure 2 is in the "positives" directory. Even though, the person in the image is clearly not smiling. We noticed this impurity in the dataset later.

Figure 2: `5.jpg` - a False Positive Sample

In addition, tried to train a model using the tools OpenCV provides. While the OpenCV tutorial provides a basic overview of the required steps, this method isn't without its challenges. One such challenge was getting the *opencv_train_cascade* to run properly because the *opencv_train_cascade* function is very particular about the parameters that it needs. The OpenCV forums also had conflicting opinions about what constitutes "correct" parameters. We managed to train a model using *Opencv_train_cascade* using different parameters.

However, the results were less than stellar. Our model recognized every image as a smile. We suspect that the problem lies with the "`bg.txt`" file. Because the model always gives the same results even though we have tried many parameters.

Nevertheless, we wanted to train our own model. We decided to use TensorFlow/Keras for our image classification. We used a tutorial from Keras[1] as a guideline. For this method, it was important to separate the dataset into a test and data section. This was done using the "`datasetHandler.py`". The convolutional neuronal network consists of 4 layers, each activated with the relu function. We achieved 91% validation accuracy after training for 50 epochs on the full dataset.

Results

LiveSmileDetection was our first application. A screenshot of this application can be found in Figure 3. As mentioned in the description of the Viola-Jones algorithm, we first detected the face (green box) and then the smile(s)(blue boxes) in the area of interest. However, as is visible in the picture, a lot of different features were also classified as a smile, which were not smiles. The face detection did a great job, but there were a lot of cases, where multiple smiles were detected in one face. So we noticed a high amount of False Positives occurring.

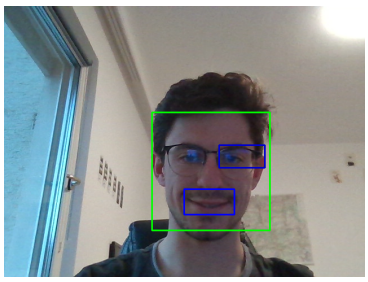

Figure 3: Face and Smile detection from webcam

OpenCV Haar Classification was our next approach, we applied the Haar-classifier provided by OpenCV to the complete dataset. As mentioned, at this point, we were not aware of the dirty dataset. Figure 4 shows our confusion matrix. It is visible that there are a lot of False Positives. 44% of all faces are wrongfully assigned as smiling. These results are in line with the LiveSmileDetection application. These results could also be influenced by the dirty dataset. Another detail worth mentioning is the execution time. We chose a serial implementation which takes roughly 20 seconds to execute. This will be discussed later in the report.

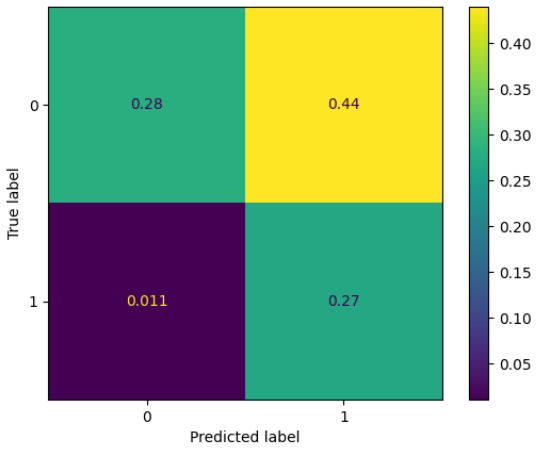

Figure 4: Normalized Confusion Matrix using OpenCV Haar-classifier

Using **Tensorflow/Keras** we achieved 91% validation accuracy. Although the accuracy is only about 91% (see Figure 6), this method provides the best overall classification compared to our other methods. Our model has a confidence of 97.14% in Figure 5 and a 99.08% confidence in Figure 6. Figure 8 shows how the loss of the model is gradually decreasing with the amount of epochs. In addition, Figure 7 displays the accuracy per epoch of the model. The more epochs, the better the accuracy.

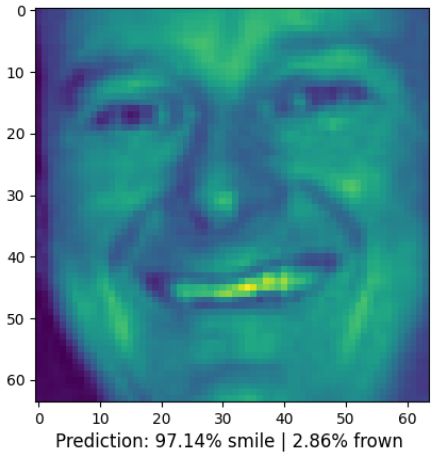

Figure 5: TensorFlow positive image

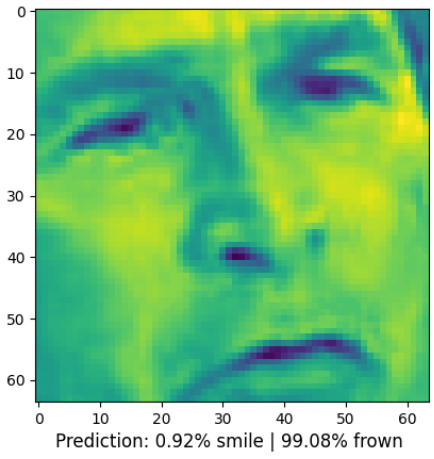

Figure 6: TensorFlow negative image

An advantage of using TensorFlow is that the training can be improved. Currently, the training is using 50 epochs, which could be increased. Furthermore, the dataset could be extended/improved using *data_augmentation* (flipping / rotating).

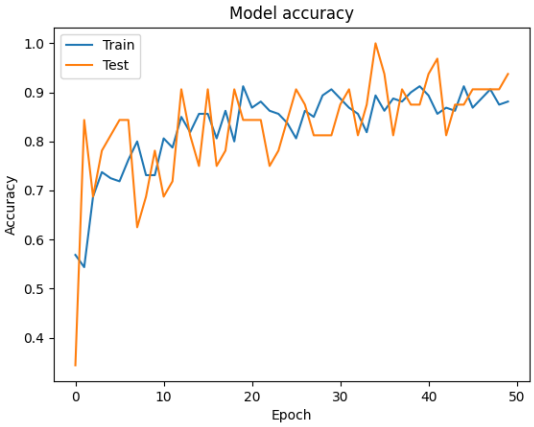

Figure 7: Model accuracy per Epoch

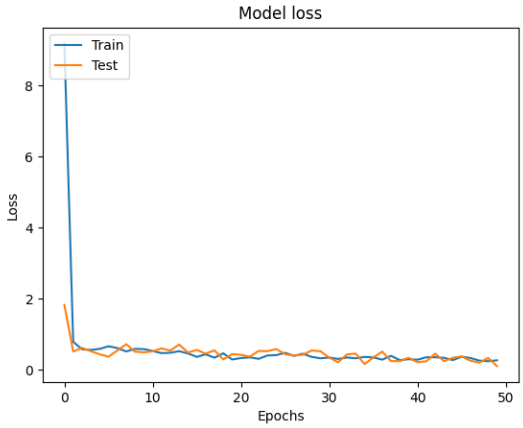

Figure 8: Loss function

Conclusion

A **problem / finding** worth mentioning would be the fact that there are endless approaches for face- / smile-detection. In the beginning, we were not sure whether we had to train our own classifiers or not. Several provided sources showed how to do so. However, the parameters, especially while using OpenCV's *opencv_traincascade*, varied a lot (e.g *-w* and *-h*).

Furthermore, we applied an additional black-and-white filter to every sample image, even though the pictures were monochrome already. This deteriorated our results, so we immediately removed any additional filters. This probably correlates to the feature detection mentioned in the **Background** section.

Since we were not happy with the results of the OpenCV training method, we decided to create our own training using TensorFlow. As presented in section **Results**, the results turned out a lot better than using 'prebuilt' classifiers.

Using TensorFlow, we ran into the problem when trying to increase the number of epochs: `Your input ran out of data;...`. We did not find a good solution for this issue, and we decided to stay on 50 epochs.

Our OpenCV Haar Classification approach is limited to the given Haar-classifiers. Since there is no room for adjustment, the results could not be improved.

We could improve the dataset and use the provided indices to only take pictures from the positives directory, which are also listed in the index file. This will skip some images, similar to `5.jpg2`.

Since the dataset already consists of faces only, we decided to skip the step where the algorithm is looking for faces and directly searched the images for smiles. However, if working with images of unknown origin, it is crucial to not skip the face detection, as this will drastically cut the computational time and resources.

One major part with potential improvement is the time-efficiency of the whole process of classifying via the OpenCV Haar approach. The fact that we are working with a big dataset leads to a rather long execution time, since the algorithm has to processes all the images. Shortening the dataset was not an option for us, so we attempted to parallelize our code using different approaches. The first approach was parallelizing using multiprocessing. However, this lead to terrible accuracy of our smile detection. As a result, we tried a different technique - multithreading. Unfortunately, this did not give us any better results either. After several attempts, we decided that sticking to the sequential code works best, even though it takes roughly 20 seconds to run.

References

[1] François Chollet. Image classification from scratch. https://keras.io/examples/vision/ image_classification_from_scratch/.

[2] Ana Huamán. Tutorial cascade classifier. https://docs.opencv.org/4.x/db/d28/ tutorial_cascade_classifier.html.

[3] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. volume 1, pages I–511, 02 2001.