universität innsbruck

igs iis

Visual Computing Project Report

Lukas Lechner & Claudio Nardin & Dominic Egger

February 15, 2022

Introduction

This report paper was written as part of the course PS Visual Computing, conducted by Prof. Antonio Rodriguez-Sanchez et al. in the winter term 2021/22 at the University of Innsbruck. The goal of the project was to evaluate a machine learning model for smile detection in pictures using Viola and Jones features on a given dataset. [6] Due to the low performance of OpenCV's default smile detection cascade, the decision was made to train a custom cascade classifier using OpenCV's cascade training functionalities on the given image dataset. The second objective was to compare the performance of the Viola and Jones classifier with the LeNet architecture. [2]
Summarizing the results of the project, it can be stated that the goal of evaluating, retraining and tuning OpenCV's smile detection cascade was successfully achieved.
The subtasks of the project were divided as follows:

- Claudio Nardin: Theoretical background of Viola & Jones feature detection
- Dominic Egger: Description of the LeNet architecture
- Lukas Lechner: Evaluation of the algorithm on the given dataset

Background

As already mentioned in the introduction, the goal of this project was to implement an efficient smile detection algorithm using the mechanism provided by Viola and Jones. In order to better understand the results of this project, it is helpful to take a closer look at the structure of the Viola-Jones algorithm. This algorithm consists of 3 parts:
- Compute Haar like features using integral images
- Using AdaBoost to find efficient classifiers
- Creating Cascade filters

Compute Haar like features using integral images
Haar like features are areas similar to kernels that are characterized by 2 zones. These zones are defined by multiple rectangles and thus describe different characteristics as it is displayed in Figure 1. In the process, all the pixels in an image that are in one zone are added up and the two zones are then subtracted from one. This result can then be used to see if this image section contains this feature. To calculate the sum of the zones efficiently, an integral image is being used. In this case, the pixels contain the sum of all pixels to the left and above (including the row and column up to the pixel itself) of the original image. The sum can then easily be calculated by adding the top left and the bottom right pixel value and then subtracting the top right and bottom left pixel values from it, as it is illustrated in Figure 2.

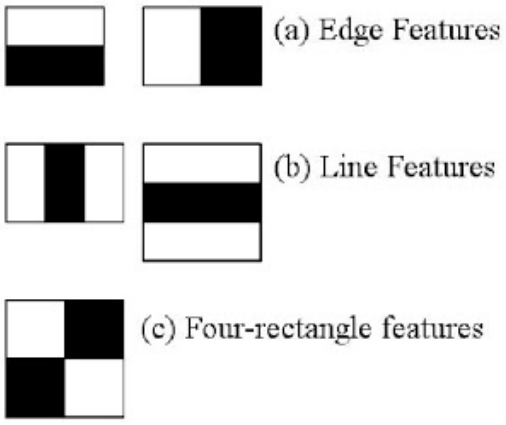

Figure 1: Basic Haar features[1]

Using AdaBoost to find efficient classifiers
Since there are theoretically more than 180000 rectangle features per image-subwindow, the idea is to combine a small number of these features into an effective classifier. [6] To determine this set

[1] https://docs.opencv.org/3.4/d2/d99/tutorial_js_face_detection.html
[1] https://www.mathworks.com/help/images/integral-image.html

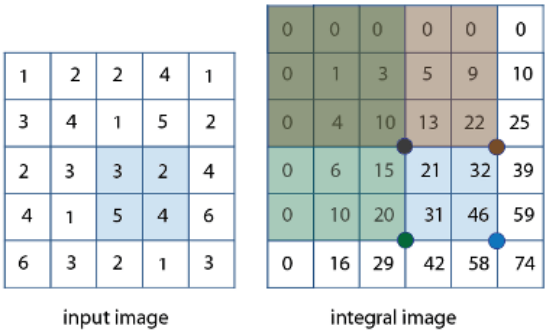

Figure 2: Calculation of an Integral Image[1]

of features and to train the classifier, an adaptive boosting algorithm is used. This then finds the optimal classifier using a set of positive and negative images.

Creating Cascade filters

By using a cascade filter, the computing time can be drastically reduced while the performance is increased. The idea behind cascade filter is that less complex classifiers are applied to subwindows first and later more expensive ones. This creates a process in which subwindows that do not meet simple classifiers are no longer checked later with more complex classifiers. The result of this a decision tree which can be seen in Figure 3. In order to achieve the best possible results, the sections of the cascade are also optimized with AdaBoost.

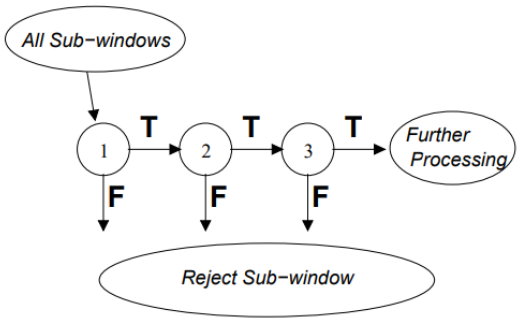

Figure 3: Decision tree of Classifiers forming a Cascade[3]

In this project a cascade filter from Sunny Meng [2] is used as well as a self-trained one. These cascading classifiers are trained to recognize whether a person is smiling or not.

The LeNet architecture

The following paragraph is dedicated to introducing the LeNet architecture, which will be compared against the cascade classifier in the Evaluation section. The LeNet architecture is a multi-layer convolution neural network for image classification firstly mentioned in the research paper "Gradient-Based Learning Applied to Document Recognition"[4]. In this work the network is used for optical character recognition (OCR) on the MNIST dataset. The structure consists of the following layers in chronological order: input, convolution, subsample, convolution, subsample, convolution, 2x fully connection and gaussian connection. As shown in the following figure.

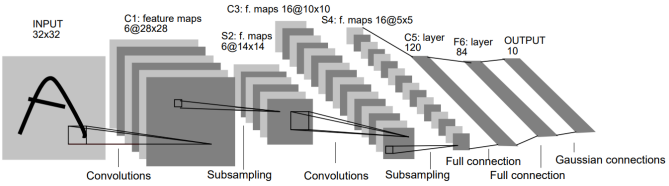

Figure 4: LeNet Structure for optical character recognition

The activation function of all convolution layers and the first fully connected layer is tanh and for the second fully connected layer is softmax. For a deep learning architecture it has a small memory footprint which makes it fast but also has a good classification accuracy. The achieved classification accuracy on the MNIST dataset was 90%, with a error rate of 1% and a rejection rate of 9%. Some adjustments can increase the accuracy of LeNet e.g. adjust the number of convolution layers, fully connected layers or output layers. Also using ReLU as activation function can increase the accuracy.

Method

For the purpose of cascade training and evaluation, OpenCV's implementations of the cascade feature extraction and detection processes were utilized. For simplicity and better visualisation, the classification and evaluation processes were deployed inside of a Jupyter notebook, which is available on GitHub as a Google Colab version and also as a locally executable version. [2] The notebook and cascade files are also provided inside of the report paper directory. Considering the code structure, the classification task was split into two separate loops: The first loop iterates over all negative examples from the dataset and updates the `false_positives` and `true_negatives` variables depending on the correctness of the predictions, while the second loop iterates over the positive examples and increments the `true_positives` and `false_negatives` variables accordingly. As a visual

[3] `http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=E8A24F24C77BD49CFC479C3397D843E0?doi=10.1.1.10.6807&rep=rep1&type=pdf`

[2] `https://github.com/degschta/vc_final_project`

aid, every 100th iteration of the loop displays a representation of the currently processed image, along with rectangle overlays representing the gold label (blue), prediction (green), and truth/prediction overlap (red).

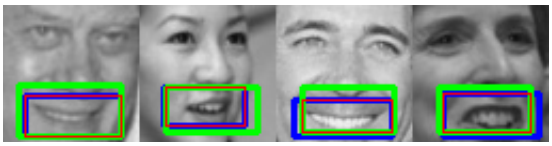

Figure 5: Example visualisations from the detection process. The rectangles represent the prediction (green), ground truth (blue), and prediction/truth overlap (red).

Initially, an attempt was made to evaluate the default smile cascade classifier provided by OpenCV on the given image dataset. This was achieved by loading OpenCV's cascade classifier with the default smile detection cascade inside of a Jupyter Notebook on Google Colab. After running the classification task, it quickly became evident that the default smile cascade performs poorly on the given dataset. Hence, the decision was taken to train a custom cascade classifier using OpenCV's built-in `opencv_traincascade` console functionality. At this point, the first major problem arose, namely OpenCV's abandonment of the cascade sampling and training console commands in recent versions (>3.4) due to legacy C API removal. One OpenCV contributor justified the removal of these functionalities by referring to modern DNN approaches, which apparently perform much higher across all domains. [1] The obstacle of missing legacy functionalities could eventually be overcome by running OpenCV 3.4 in a Docker container with Ubuntu 16.04. [3] After running the training and classification processes with different parameters, a performance comparison of the resulting classifiers based on confusion matrix, precision and recall was made to determine the highest-performing configuration. Finally, OpenCV's built-in visualization console command `opencv_visualisation` was used to graphically represent the Haar features from each cascade.

Results

Before diving into the results section, one caveat with respect to the given image dataset has to be addressed, namely its high number of false annotations. This flaw quickly becomes evident upon visual inspection of the dataset. Specifically, many faces inside the negatives folder actually portray people who are smiling. These incorrect annotations likely impacted the evaluation and also the quality of the custom cascade classifier that was trained based on the image dataset.

For the first trial, OpenCV's default smile cascade

Figure 6: Example faces falsely or controversially annotated as 'not smiling'.

`haarcascade_smile.xml` was used to classify all images from the given dataset. This was done by instantiating OpenCV's `CascadeClassifier` class with the respective cascade XML. Now running the `CascadeClassifier.detectMultiScale` method with default parameters yielded the following performance results:

```
Trial 1: default classifier
scaleFactor: 1.1
minNeighbors: 3
```

		True	
		Positive	Negative
Pred.	Positive	3486	5833
	Negative	120	3697

Precision: 0.374
Recall: 0.967
Accuracy: 0.547

Figure 7: Haar features visualisation of OpenCV's default smile cascade (stage 0).

Assessing these results, it is easy to see that the default classifier performs poorly when applied to the given image dataset with default parameters. Specifically, the exorbitant rate of false positive predictions poses a major performance problem. Therefore, the

[3] `http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=E8A24F24C77BD49CFC479C3397D843E0?doi=10.1.1.10.6807&rep=rep1&type=pdf`

obvious step was taken to tune the parameters of the `CascadeClassifier.detectMultiScale` method. Executing the detection method with an increased `scaleFactor` and `minNeighbors` significantly enhanced the performance of the classification task, as is clearly reflected in the confusion matrix:

```
Trial 2: default classifier
scaleFactor: 1.3
minNeighbors: 10
```

		True	
		Positive	Negative
Pred.	Positive	2134	677
	Negative	1472	8853

Precision: 0.76
Recall: 0.59
Accuracy: 0.84

Despite the obvious improvement of classification performance, it should be noted that training with the modified parameters saw a huge increase in processing time, and recall remained low.
Hence, the decision was made to train a custom Viola Jones features classifier using samples from the given image dataset as training input. To this end, OpenCV's command line utilities `opencv_createsamples` along with `opencv_traincascade` were used. Unfortunately, the initial attempt of a custom classifier performed poorly when applied to the dataset. The training process for this first trial was specified with a sample width/height of 20, a positive sample size of 1000, a negative sample size of 500, and 10 stages. The 20x20 width/height setting of the samples was based on a recommendation of Kuranov et al., who conducted an empirical analysis of various detection cascades. [5] However, after some trial and error it turned out that the width/height setting seemed to be responsible for the poor performance of the classifier. Adjusting the sample measurements to a width of 20 and a height of 10 for the training process resulted in a tremendous performance boost of the resulting cascade classifier. This choice of sample dimensions is based on the logic that 20x10 more accurately corresponds to the natural proportions of the human mouth. Given below are the performance results for this first successful version of a custom classifier:

```
Trial 3: custom classifier

training specs:
```

Figure 8: Haar features visualisation of the custom cascade from trial 3 (stage 0).

```
sample width/height: 20/10
pos/neg sample size: 1000/500
stages: 20

detection specs:
scaleFactor: 1.1
minNeighbors: 3
```

		True	
		Positive	Negative
Pred.	Positive	3606	1215
	Negative	0	8315

Precision: 0.75
Recall: 1.00
Accuracy: 0.91

It is also noteworthy that the classification process for this custom classifier executes faster than the default classifier specification from trial 2 by several orders of magnitude. With that being said, the confusion matrix also reveals one major problem of the custom classifier, namely the relatively high number of false positive predictions, which can be partly attributed to the falsely annotated faces from the original dataset, an issue shortly addressed at the beginning of this chapter.
Nevertheless, a further attempt was made at improving the custom classifier. The first adjustment was to increase the `minNeighbors` detection parameter in order to make the classifier more robust against low-threshold predictions. This resulted in a significant performance boost of the classification task, as can be observed in the confusion matrix:

```
Trial 4: custom classifier

training specs:
sample width/height: 20/10
pos/neg sample size: 1000/500
stages: 20

detection specs:
scaleFactor: 1.1
```

`minNeighbors: 4`

		True	
		Positive	Negative
Pred.	Positive	3606	689
	Negative	0	8841

Precision: 0.84
Recall: 1.00
Accuracy: 0.95

At this stage, the cascade classifier achieved close to state-of-the-art results with respect to the dataset. The only remaining problem at this point was the relatively high number of false positives. However, as was already mentioned initially, many faces in the dataset are controversially or even falsely annotated as 'not smiling', and as a consequence, a great number of the 'false' positives can be attributed to the insufficiency of the dataset. For further visualization of the problem, given below are four example faces which were predicted by the custom cascade as 'smiling', but are actually 'not smiling' according to the gold labels of the dataset.

Figure 9: Example faces which were classified as 'smiling' by the custom cascade, but are 'not smiling' according to the ground truth of the dataset.

The final stage of the evaluation consisted in comparing the best-performing custom classifier with the LeNet architecture. This architecture has already been introduced in the Background section of this paper. The cascade classifier will be compared against the LeNet architecture for smile detection, which is an implementation of LeNet tailored to smile detection. Specifically, the LeNet architecture for smile detection[2] creates 20 instead of 6 channels on the first convolution layer and 50 channels instead of 16 on the second convolution layer. Also, the convolution layer doesn't change the resolution in this implementation. This results in an output size of 500 for the fully connected layer. The final output size is two for either smiling or not smiling. The activation function is ReLU in this implementation for more accurate results. This implementation achieves a classification accuracy of 92% on the SMILES dataset. The following figure shows the layer structure of this LeNet implementation.

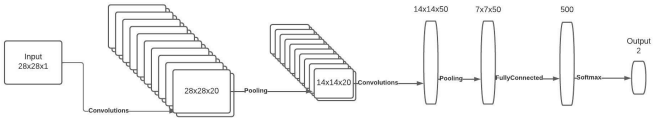

Figure 10: LeNet Structure for smile detection[2]

Given below are the evaluation results of LeNet compared to the best-performing custom cascade:

LeNet performance:

	precision	recall	F1 score	support
not smiling	0.95	0.93	0.94	1895
smiling	0.83	0.88	0.85	738
avg/total	0.92	0.91	0.92	2633

Trial 4 custom cascade performance:

	precision	recall	F1 score	support
not smiling	1.0	0.93	0.96	9475
smiling	0.84	1.00	0.91	3690
avg/total	0.96	0.95	0.95	13165

Comparing the performance metrics of LeNet to the custom cascade reveals superior performance of the cascade classifier when trained and configured as demonstrated in trial 4. However, it should be noted that further testing of both classifiers on novel datasets would be necessary in order to evaluate portability and performance on unseen data.

Conclusion

Summarizing the progression of the project, the goals of evaluating OpenCV's default smile cascade and, based on the results, improving the quality of the classification process were successfully achieved. Specifically, training a custom cascade classifier on the given dataset drastically enhanced the accuracy of the predictions. Comparing the performance of our custom classifier with LeNet revealed that cascade classifiers can compete with modern CNN approaches with respect to the given task.
The main obstacle during the project was training a custom Viola & Jones cascade. More specifically, all OpenCV >3.4 have abandoned the `opencv_createsamples` and `opencv_traincascade` console functionalities, which are needed for training a custom Viola & Jones cascade. This problem was eventually solved by running OpenCV 3.4 in a Docker container with Ubuntu 16.04.
Regarding future research, it is fair to say that there

is still room for improvement of the custom cascade classifier, especially by way of tuning parameters or increasing the sample size during the training process. Also, further investigations into the performance of the custom classifier could be useful in order to determine how well the custom cascade performs on novel or unseen datasets.

References

[1] Opencv4 createsamples and traincascade are missing from build/apps. `https://github.com/meng1994412/Smile_Detection`. Accessed: 2022-02-13.

[2] Smile detection. `https://github.com/meng1994412/Smile_Detection`. Accessed: 2022-02-12.

[3] Ubuntu 16.04 with opencv+modules and python2.7. `https://github.com/dymat/docker-opencv`. Accessed: 2022-02-13.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[5] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In Bernd Michaelis and Gerald Krell, editors, *Pattern Recognition*, pages 297–304, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[6] Paul A. Viola and Michael J. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR (1)*, pages 511–518. IEEE Computer Society, 2001.