CSCI 1430 Final Project Report: Automated Selfie Camera

hapi_asunshine9981: Gabby Asuncion, Sophia Chen, and Kelvin Yang
Brown University
8th May 2020

Abstract

The overall goal of this project was to build an automated selfie camera, that would, at a reasonable accuracy, detect smiles in the subject and capture an imagine accordingly. Several methods of image preprocessing, model architectures, feature extractions, and accuracy validation were used in order to determine the optimal pipeline for smile detection from live input. Key components for the optimized final product included a pre-trained Haar Cascade face detection model from OpenCV and a tuned Support Vector Machine classifier trained on Histogram of Gradient features of training images. Hyperparameters were tuned using k-fold cross validation, resulting in a classifier that achieved reasonable accuracy and performance on live input.

1. Introduction

The two fundamental components of our automated selfie camera are (1) a smile detection model and (2) live video stream input. We sought to tackle each of the components independently, as integration of the two was relatively easy.

The main determinant of the performance of our selfie camera was a well designed and fine-tuned smile detection model. This poses some difficulty, as there are many ways to determine whether a subject should be classified as 'smiling', and the smile detection is highly sensitive to image quality, colors, and a subject's position relative to the camera. Keeping these difficulties in mind, our approach focused on the mouth region of a face to determine 'smiling', while removing all other possible variables by greyscaling images and scaling image sizes appropriately. By focusing on the mouth region, we understood that a smile was classically viewed as a curved appearance, which dictated the way we extracted features in our data.

In regards two the live video stream input, a main difficulty is determining how sensitive the model should be in regards to live image input, and how long the user should be required to hold a smile before the image was captured.

With this pipeline and model architecture, this will allow for easy extensions to different classification tasks from live image input, as well as experimentation into different ways to better determine emotion in images.

2. Related Work

Initial project inspiration came from a paper titled "Developing a Practical Smile Detector" by Whitehill et. al. [4], regarding a potential model architecture with a support vector machine and a boosting algorithm. This also gave us insight into what sort of features to extract, as well as the GENKI4k training dataset. We also drew insight from the paper "Efficient smile detection by Extreme Learning Machine" by An et.al [1] for further model and feature extraction insight.

Most software techniques used were made using built in sklearn library models and imutils video streaming libraries. Data was from the GENKI4k [2] library. In order to further improve classification accuracy, we used OpenCV's CascadeClassifier [3] with a pre-trained Haar Cascade frontal face detection XML file.

3. Method

3.1. Smile Detection Model

We began by building the smile detection model by reading and preprocessing our training data corpus, the GENKI4k dataset. This dataset contained 4000 JPG images of faces with various ethnicities, illumination conditions, and personal identities. Each was labeled as smiling (1) or non-smiling (0), with additional data regarding head pose (yaw, pitch, roll). Each image was read in and reshaped to 100 by 100 pixels and converted to greyscale, in order to remove affects of color of our smile classification. The corresponding label for each image was also saved, and we then performed an 80-20 train test split (with shuffling) on the dataset. With our training dataset, our first attempt at a smile detection model involved simply taking in the raw image pixels, flattening each of the images, and feeding them into a support vector classifier with a linear kernel and a margin of error

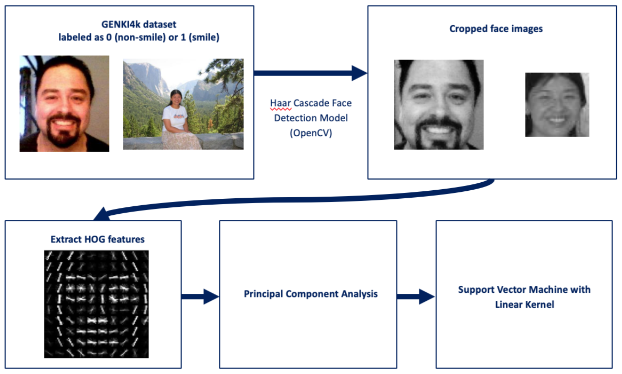

GENKI4k dataset
labeled as 0 (non-smile) or 1 (smile)
Haar Cascade Face Detection Model (OpenCV)
Cropped face images
Extract HOG features
Principal Component Analysis
Support Vector Machine with Linear Kernel

Figure 1: Smile detection model pipeline.

of 5. The trained model was then used to classify our test images, to determine performance.

This simple model architecture proved to be unsuccessful in classifying images on both the test and train set. Therefore we focused our efforts on extracting features from our images, instead of dealing with raw image pixels. Because smiles can be determined by the curved structure of the mouth, we extracted a Histogram of Gradients for each of the images. Then, support vector classifier training was performed on the features of the images, in order to determine patterns in the gradients of the mouth region that would indicate a smile. The HOG features were extracted with 15 orientations, 10 cells per block, and 10 pixels per cell. However, because the resulting features were of very high dimensionality, we applied Principal Component Analysis to transform the data into a more manageable feature space and pinpoint the features that contributed most to effective classification. These were then fed into our support vector machine and trained.

While training on the HOG features significantly improved classification accuracy, the performance was still sub-optimal, regardless of hyperparameter tuning. We realized that our dataset included images of all different zoom levels, therefore the model was most likely learning qualities of images that weren't focused on the subjects themselves. Thus, we wanted to crop our images so that it focused on the subject's faced. This was done by using a pre-trained Haar Cascade face detection model from OpenCV, to determine the bounding box of the face in each image, and then cropped appropriately. This step was done before our original preprocessing step. Adding this level of data processing significantly improved our performance, as seen in the Results section below. The resulting model pipeline is highlighted in Figure 1.

Because we had a limited dataset, some of which were not perfectly oriented, we utilized 5-fold cross-validation in order to generate more train-test sets to determine performance. Once the parameters were tuned appropriately based on the scores in each fold, our resulting train SVM model, along with the tuned PCA transformation, were pickled and saved to be used in our live camera feed.

3.2. Live Camera Feed

The live camera feed was implemented using a VideoStream package, which, once initialized, read in frames that are represented as pixel matrices. Frames are continuously read in and fed into the pre-trained model face detection model, in order to crop the frame/image appropriately. The resulting cropped image went through the same preprocessing steps as our training dataset, and the corresponding HOG features were extracted. Using the pickled PCA transformation, the dimensionality of the data was reduced, and fed into our smile detection model. We made sure that a positive labeling of a smile was detected for at least 10 frames, before the frame was saved as an image as a selfie. The resulting video stream pipeline is highlighted in Figure 2.

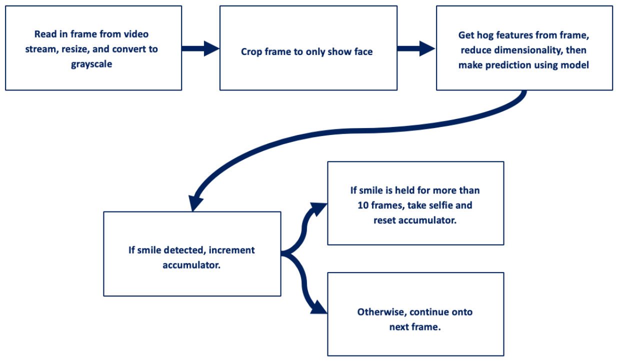

Read in frame from video stream, resize, and convert to grayscale
Crop frame to only show face
Get hog features from frame, reduce dimensionality, then make prediction using model
If smile detected, increment accumulator.
If smile is held for more than 10 frames, take selfie and reset accumulator.
Otherwise, continue onto next frame.

Figure 2: Live camera feed pipeline.

4. Results

Our initial setup of training on raw images pixels, without extracting HOG features, achieved the highest of 53% accuracy. The resulting accuracies was not much better than random guessing. Thus, our eventual approach was to extract HOG features on the cropped images. The resulting scores from cross validation was significantly improved, as seen in Table 1. A specific hyperparameter tuned was the amount of variance explained in our PCA reduction. With a hyperparameter value of 95%, we achieved the highest accuracies. The optimal set of hyperparameters were pinpointed by relatively similar accuracies accross all folds, and a good balance between the training and testing accuracies to prevent overfitting. The resulting images and image features performed well, starting with the initial image inputs in Figure 3.

After cropping in the images based on bounding box returned from the face detection model, we got the images in

Fold	Accuracy
0	0.8250
1	0.8125
2	0.8078
3	0.8078
4	0.8016
Train Set	0.910
Test Set	0.832

Table 1: Accuracies for SVM training on hog features extracted from cropped images.

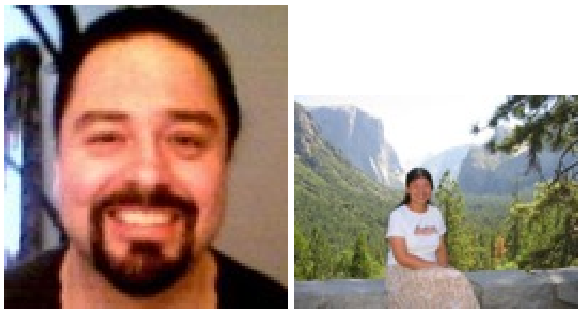
]


Figure 3: Original training images.

Figure 4. The HOG features were then extracted, as seen in Figure 5. As we can see from the gradients, the mouth region is able to clearly detect the curved-shape of the mouth for a smile. Based on live stream image input, the camera captured the following images Figure 6, upon detecting 10 frames of smiles of the subject. The performance was reasonable, which was determined by how long the subject had to smile for, and different subject poses/orientations as well. We did find, however, that for certain head poses where the subject was not entirely facing the camera, the model was not a sensitive to smiles in this scenario.

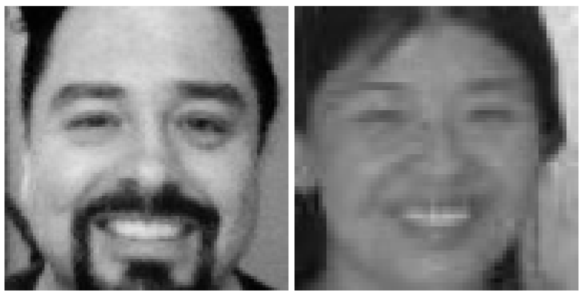


Figure 4: Preprocessed images (cropped and greyscaled).

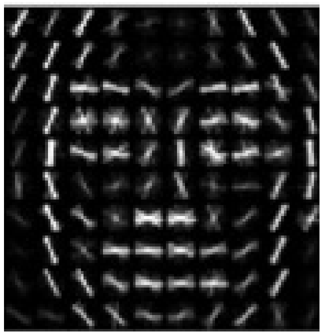


Figure 5: Extracted hog features.

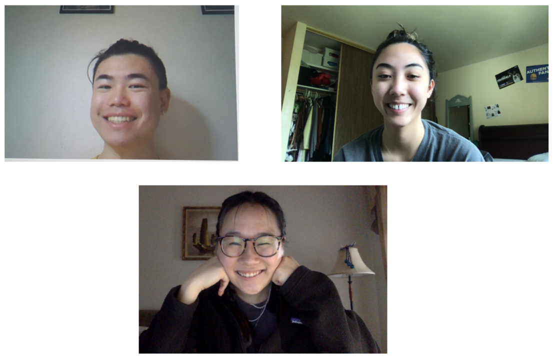


Figure 6: Live testing results.

4.1. Discussion

Based on our results, we achived a reasonably high accuracy in detecting smiles, and interacting directly with the live camera input showed that our model was able to detect smiles at comparable sensitivity and accuracy. In our approach, we made sure to normalize all factors that could effect classification with our model, such as color and additional objects in images. However, it was difficult to control for different positions of the subject's faces. It is interesting to note that our GENKI4k dataset contained information about head pose parameters as well – thus, using these terms to determine and encode variability in head orientation would significantly improve our classification.

After initially creating a weak classifier, we wondered if adding boosting would increase our testing accuracy. Boosting is typically used to strengthen weak classifiers and would likely increase the processing time dramatically. If we were to add boosting to the weak classifier version of our model, we would need to consider if the sacrifice in processing time would be worth the increased accuracy of the model.

The primary consideration for making such trade-offs is the application of the tool itself. While our application detects smiles and captures selfies when it does, the model could also be used to detect smiles in group pictures and only keep the photos in which everyone is smiling. In this case, it would be beneficial to increase the model's accuracy using boosting since processing time would not be as much of a concern. However, in our particular application, we would like to ensure that the latency between detecting and capturing the photo is minimized. Therefore, we decided not to use boosting and instead extracted hog features for classification, yielding a strong classifier and eliminating the need for boosting.

We have also considered using both the eyes and the mouth regions of the face to detect smiles. To add this to our current application, we would likely have to implement a bag of words function to build a vocabulary of local image features (i.e. eyes and mouth) then train separate models for each region of interest in the face by extracting hog features. However, the shapes of eyes when smiling vary much more than the shape of the mouth and would lead to difficulty when building a sufficiently diverse training set and improving the accuracy of the model.

5. Conclusion

Ultimately, our application successfully detected smiles and captured selfies when the subject was smiling. While detecting smiles is an advantageous tool for hard-to-time moments, such as with kids, this model can be adapted to detect other shapes and capture images for those. For example, instead of storing hours of video feed in a CCTV, we could equip security cameras with the ability to capture photographs of theft. The basic structure of the model can be adapted for different needs by changing the training set.

However, it is important to note that selecting the training data is crucial, especially when handling serious issues such as crime. The data chosen should not exhibit discrimination so not to bias the model. While the technology of the model itself may be applied to wide variety of uses, we must be aware of the potential shortcomings of the technology and training data.

References

[1] L. An, S. Yang, and B. Bhanu. Efficient smile detection by extreme learning machine. 05 2013. 1

[2] http://mplab.ucsd.edu. The MPLab GENKI Database, GENKI-4K Subset. 1

[3] P. Viola and M. Jones. The OpenCV Library. 2001. 1

[4] J. Whitehill, G. Littlewort, I. Fasel, and J. Movellan. Developing a practical smile detector. 01 2008. 1

Appendix

Team contributions

Gabby I wrote the functionality for the live video stream aspect of the application. This required reading in a frame from the video stream and running it through the model to detect a smile. I also came up with the idea for this application as well as our team name (a combination of all our social media handles).

Kelvin I pickled to save the trained smile detection model and loaded it into the video stream part of the app.

Sophia I wrote the smile detection model and data processing, in regards to the Haar Cascade classifier outputs. I tuned the different hyperparameters in order to achieve highest performance accuracy.

Data Files

Because of the size our data files, they were not included in the Github repository. A compressed version of all the training images (original and cropped) have been uploaded to Google Drive, at this link. Instructions on how to run the program with these data files can be found in the next section.

Running the Program

First begin by cloning this repository.

Because of the size of the data directory, a compressed version of the files have been uploaded to Google Drive. Download the zipped file. This will give you a directory named 'data', with a directory 'genki4k' and a README file explaining the contents. Simply move all the contents of the 'data' directory into the 'data' directory of the cloned repository. More details as the the contents of the data directory can be found the the downloaded README.

You will need specific dependencies, which can be installed with the following requirements.txt file, as well as the 'imutils' package.

Now, you can run the application! Simply navigate to the 'code' directory and run 'python run.py', and you will see the live camera input begin!