CSCI 1430 Final Project Report: Emotion-based Snapchat Dog Filter

Conrad's Cheerleaders: Griffin Beels, Brandon Li, Giuse Nguyen, Conrad Zborowski.
Brown University
11th May 2020

Abstract

Snapchat filters currently lack a level of interactivity beyond generating static overlays for identified facial regions of interest (ROIs). We present a method of expanding on this interactivity by applying facial expression detection to the ROI, which in turn displays a corresponding overlay to match the identified emotion. Our model is effective and notably compact, providing an attractive option in its application to live camera feeds, which require continuous computations at a rapid framerate.

1. Introduction

When we think about computer vision, we commonly associate the field with the cutting edge in commercial industry; perhaps this means one of its many applications to autonomous operation, e.g. self-driving cars. However, computer vision also plays an increasingly integral role in entertainment, specifically its use in social media. A prime example of this is Snapchat. Snapchat is a popular app that experienced an explosion in usage in 2012. Utilizing convolutional neural networks (CNNs), Snapchat filters identify facial presence and positioning to apply effects such as makeup or tattoos.

One of the most popular face filters is the dog filter in Snapchat, which overlays a dog's features onto one's face. However, this dog filter lacks a level of dynamism in that beyond the occasional licking animation characteristic of the filter processing an open mouth, the dog filter and most of Snapchats other filters are nonreactive to user input captured from the camera feed beyond the initial facial identification. We wanted to expand on this level of interactivity that can be found in a Snapchat filter beyond the dog filter's licking animation that appears when the user opens their mouth.

Using the data currently captured and processed by Snapchat's facial recognition algorithms is not enough to ascertain the user's current emotional state, as even the slightest changes in a few of a person's 43 facial muscles could indicate a difference in mood. Therefore, we propose to use a convolutional neural network in conjunction with image processing to identify the emotion of a person captured in a camera feed. After detecting emotion, we will apply emotion-specific filters to our dog filter based the user's facial expressions.

By improving emotional intelligence of the AI in social media platforms such as Snapchat, we stand to create a more accurate depiction of user self-expression. With augmentations to the level of empathy between users, we can further expand upon Snapchat's high level of bonding social capital[1].

2. Related Work

Much work has already gone into facial expression recognition and emotion detection. Correa et al. and Atul Balaji, both from which we draw inspiration for our own model architecture, implemented a simplified versions of AlexNet [2][3], both achieving accuracies above 60% .

Some work has achieved high accuracy when incorporating more information into the model. For example, Kim et al. incorporates auditory as well as visual information to identify speaker emotion, achieving average accuracies of up to 93.12% on the Surrey Audio-Visual Expressed Emotion (SAVEE) dataset and average accuracies of up to 56.63% on the Interactive Emotional Dyadic Motion Capture (IEMOCAP) dataset using linear SVMs[4].

We used these resources as guidance for the construction of our own architecture, however, we faced a computational constraint in that we needed quickly compute the desired emotion before the next frame, something that was not factored into the identified related works, which all worked with static images.

3. Method

3.1. Process Pipeline

Our input is a video, or rather a stream of images in which we capture and process each frame separately, and ultimately

output an added emotion specific dog ears filter. The ”happy” emotion will produce perked ears, the ”sad” emotion will produce drooped ears, and the ”neutral” emotion will produce slightly bent ears.

The broad overview of our process is as follows: Pipeline

Process Pipeline

Step 1: Pass the current frame of the live-feed as input.

Step 2: Use Haar-cascades to identify and extract facial information from the frame.

Step 3: Generate a 48x48 feature input from the captured ROI.

Step 4: Pass the feature input into the trained model to determine emotion.

Step 5: Use the associated set of ears according to the identified emotion.

Step 6: Apply the overlay to the corresponding facial regions

Step 7: Display the current filter overlay.

To extract the actual frame from the image, we utilize OpenCV's libraries to process the live video. OpenCV provides video streaming functionality that emulates the Snapchat camera well. OpenCV also provides a diverse set of functionality for for facial detection. Specifically we used the Haar Feature-based Cascade Classifiers, which have the flexibility of selectively applying only the necessary features for our purposes, which in this case is only 10 out of the 6000 total identifiable attributes.

3.2. Model

We trained the neural net using the the Facial Expression Recognition 2013 (FER-2013) dataset, which contains 35,685 examples of 48x48 pixel grayscale images of faces grouped into 7 different emotional classifications. The smaller images were especially useful for our purposes, as performing convolutions and fully connected layers on smaller images is ideal for realtime video processing.

3.2.1 Architecture

We based our architecture off of the CNN mentioned in Correa et al. [5], which implemented a simplified version of AlexNet that takes 48x48 images and is less expensive computationally. Low computational cost is especially important for our task, as we are applying our filter to every single frame of a live video feed and therefore cannot afford any noticeable slowdowns. We used categorical cross entropy loss for our loss function and an Adam optimizer with a learning rate of .0001. Our model trained for 50 epochs using a batch size of 64. Table 1 below shows our architecture.

Layer	Size	Pool/Kernel	activation
Convolution	32	3x3	ReLu
Convolution	64	3x3	ReLu
MaxPool	-	2x2	-
Dropout(25%)	-	-	-
Convolution	128	3x3	ReLu
MaxPool	-	2x2	-
Convolution	128	3x3	ReLu
MaxPool	-	2x2	-
Dropout(25%)	-	-	-
Dense	1024	-	ReLu
Dropout(50%)	-	-	-
Dense	7	-	Softmax

Table 1: The layers in the architecture.

3.2.2 Filter Construction

After training our weights, we applied our model on a frame-by-frame basis to a live video feed input to identify the current emotion of the largest face identified in the frame. For increased stability of the filter, we implemented a ”voting” method that caches the previous four frames of the video feed to contribute to the identification of the current emotion to allow for smoother transitions between different filter states. Although our model detects 7 different emotions, for simplicity's sake and for further stability in between emotion transitions, we grouped emotions together into 3 major categories. Happy and surprised both counted as happy, sad, angry, and disgust counted towards sad, and lastly neutral and fearful were grouped as neutral. For the three major identified as happy, sad, and neutral, we used perked ears, slightly bent, and drooped ears, respectively. Using the OpenCV library's *detectMultiScale* function and open-sourced Haar cascades [6] [7] to identify facial locality and dimension, we positioned the ears at the centers of the upper left and right quadrants of the identified ROI. We initially positioned the nose overlay at the center of the ROI, and later we switched to using a Haar cascade to detect nose position, a decision that will be expanded upon in the Results and Discussion section.

4. Results and Discussion

After 10 epochs, our accuracy for detecting and categorizing images based on the 7 emotions raised above 45% and after 20 epochs, raised above 60%, staying roughly the same for the next 30 epochs after which it finished with a validation accuracy of 63%. This is the same accuracy obtained by the model used by Correa et al. [5], and comes close to the

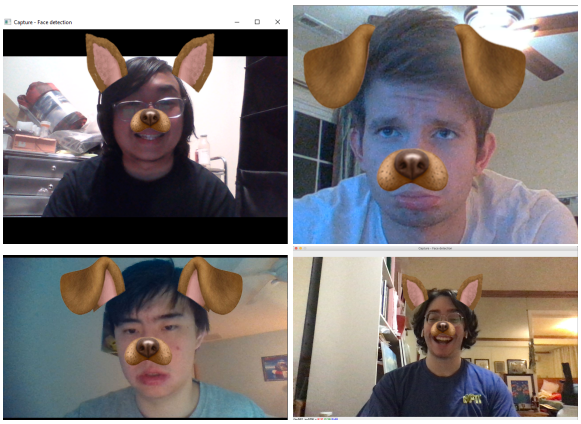

Figure 1. Top left: Happy dog filter, Top right: Sad dog filter, Bottom left: Neutral dog filter, Bottom right: Happy dog filter

state-of-the-art mentioned by the paper, which was 67% at the time of writing.

As demonstrated in Figure 1 above, the filter could generally detect the correct emotion, although some emotions seemed to be harder than others to detect, which is to be expected. This makes sense as happy, neutral, and surprised are the most distinguishable features to both humans and neural nets, while sad, fearful and angry are often mistaken as neutral [5]. The happy ears (perked) were the easiest ears to have appear, while the most difficult was the drooped ears characteristic of sadness.

Many of the issues and tradeoffs we encountered were due to the fact that instead of static pictures, we were dealing with live video feeds, which pose challenges of not only correctly identifying faces for all frames, but also correctly identifying emotions in addition to maintaining consistency of the filter such that the filter does not change too often, which can be jarring. The main issues we encountered, as well as our resolutions, are detailed below.

Emotion Voting Upon completing our training and our preliminary dog filter overlay, one of the biggest problems we encountered was the rapid shifting of detected emotions in the live feed. To resolve this and add consistency to our emotion filter, we implemented a voting system where the emotion detected from the last 4 frames would be used to vote and determine the emotion of the current frame. This way, our filter would reasonably show what essentially would be the most likely emotion over the short period of time.

Nose detection In our original implementation, we initially opted to have the nose positioned based on the facial dimensions extracted by our Haar cascade methods. However, we found that the large diversity of nose shapes, sizes, and positioning on faces disallowed a ”one size fits all approach.” Therefore, we utilized cascade detection to find the position of the nose and add the dog nose picture to the same position. The tradeoff with using nose detection is the finickiness that comes with using detection algorithms on live feeds. Since we already are using detection algorithms for both the facial expressions as well as the presence of the face itself, using detection for yet another aspect of our project added yet another level of complexity to our filter. However, we were able to come to a resolution for this issue which we explain in the element flickering section.

Element flickering: Sometimes the cascade detection was inaccurate by nature of face detection algorithms not being perfect. How, then, do you make sure that the nose or ears don't randomly fly across the screen? Our solution was to implement caching of elements, as well as a validation check for (1) the size of the elements detected and (2) the positioning of the elements. For (1) each Haar cascade has the method *detectMultiScale* which has a flag for only returning the largest element resembling a face or nose. Additionally, we assert that each face's width and height is greater than some minimum bound. With these restrictions in place, the current face is validated further than simply choosing the first result. If these checks fail, then the previously cached successful filter position is used. For (2) we additionally check to confirm that the difference in positioning for a newly detected nose is within a reasoning bounding box of the old nose position. The bounding box check allows for reduction in flickering (i.e., a dog nose briefly flashing on the forehead or chin) because those cases would be flagged as too far away from the old nose.

Obstructions: We found that objects that partially obstructed the face had a tendency to interfere with detection, therefore preventing the dog ears and nose from appearing. The most important case to note for this situation would be the instance where the user is wearing glasses. Unfortunately, we did not have a solution for this particular issue, as this likely is due to our facial cascade detection itself not being built to handle glasses, thereby not recognizing faces with in this case.

5. Conclusion

The key takeaway of our approach is the viability what may perhaps be the start of a new brand of filters, as well as demonstrate a means to expanding the interactivity of communication apps such as Discord, Twitch, Zoom, etc.. As photo-sharing social media platforms such as Snapchat lack such a means of communication, that is, a filter that detects not only the face but also reacts based on the user's emotion, we provide a way for users to further express both themselves and their creativity.

In future work, we are interested in farther investigating

ways in which emotion can be used as an additional input in applications and games.

6. Acknowledgements

We wish to thank Mary Dong for her insight and guidance throughout the project, especially in the midst of our current crisis, as well as for being an all around awesome TA. We also would like to thank James Tompkin for teaching Computer Vision and keeping the class fun and interesting!

References

[1] J Phua, SV Jin, JJ Kim - Computers in Human Behavior, 2017 - Elsevier

[2] https://github.com/isseu/emotion-recognition-neural-networks

[3] https://github.com/atulapra/Emotion-detection

[4] Y. Kim and E. M. Provost, "Emotion recognition during speech using dynamics of multiple regions of the face,' [Online].

[5] E. Correa, A. Jonker, M. Ozo, and R. Stolk, "Emotion recognition using deep convolutional neural networks," https://github.com/isseu/emotion-recognition-neuralnetworks/blob/master/paper, 2016

[6] https://github.com/opencv/opencv/tree/master/data/haarcascades

[7] https://github.com/opencv/opencv_contrib/blob/master/modules/face/data/cascades/haarcascade_mcs_nose.xml

Appendix

Github Repository

Because our complete repository is too large to submit through gradescope, we submitted a compact version of our project with the bare minimum required to run. Training and testing data can be found at the repository: https://github.com/griffinbeels/emotion-based-filter

Team contributions

Please describe in one paragraph per team member what each of you contributed to the project.

Griffin Beels Griffin and Conrad worked on constructing the model architecture. Griffin was responsible for the preliminary set up for the project. Griffin additionally implemented the voting scheme to further allow for stability in between emotional state shifts as well as other optimizations to prevent flickering.

Giuse Nguyen Giuse worked with Brandon on the filter overlay. Giuse was responsible for the initial setup of the filter overlay class, in addition to the calculations and implementations of the ear positioning and changing based on emotional state.

Brandon Li Brandon worked with Giuse on the filter overlay. Brandon added the nose detection and implemented overlay resizing based on the facial distance from the camera in addition to optimizations to prevent flickering. Worked with Conrad on the final report paper.

Conrad Zborowski Conrad constructed the model architecture with Griffin. Conrad identified the neural net and laid out the architecture for the project, and in addition performed optimizations on the model via tweaking the hyperparameters and dropout rates to determine the optimal combination maximize validation accuracy. Worked with Brandon on the final report paper.