Snapchat Filters with Python and OpenCV

Jair Aguirre
jair.aguirre.1@gmail.com

William Gottschalk
will.gottschalk@gmail.com

ABSTRACT

Facial Recognition and computer vision are increasingly used in production applications. Self-driving cars use computer vision to detect and interpret a scene to make realtime decisions. Amazon's brick and mortar stores use facial recognition to gather data about a user's decision making process. There will also be an estimated 1.3 trillion photos taken in 2017 with a large percentage of those with one or more faces in it. As computer power becomes more advanced, we are able to bring complex facial recognition technology to more and more people to help improve their lives. In this paper, we attempt to outline the steps necessary to reproduce Snapchat's facial filters using Python and computer vision libraries such as OpenCV. We feel this is problem gives a good survey of a number of technologies involved in understanding computer vision from both an academic and production standpoint.

KEYWORDS
Facial Recognition; Computer Vision; Viola-Jones; Haar Cascades; Landmark Detection

1. INTRODUCTION

Computer vision is the field of study of how computers can gain high level information from images and videos. Some applications of computer vision are: scene construction, event and object tracking, motion estimation and image restoration.

Snap Inc, has taken advantage of these applications of computer vision by creating the popular Snapchat filters. At a high level, Snapchat filters recognize your face through your video camera and apply a facial "mask" which can render different cartoon characters onto your face in real-time. Usually, these renderings are accompanied by animations triggered by facial expressions.

We aim to outline the steps necessary to implement your own version of Snapchat filters by diving into how the underlying technologies work. In part two, we cover the theoretical underpinnings of the algorithms that power our implementation of Snapchat filters as well as some of the state-of-the-art algorithms involved that didn't make it into our implementation. In Part Three, we explore some of the current open source facial recognition libraries, how we used them for training a model, the methods used in the implementation, and possible improvements to be made by looking ahead at some of the state-of-the-art research. Part Four address our results and presents the challenges that we had during the project as well as optimizations for the future.

2. RELATED WORK

Face Detection is can be thought of as a special case of object detection in computer vision. Face detection algorithms determine the locations, size, additional landmarks and even gaze direction of human faces. This has a wide range of applications ranging from entertainment to surveillance.

2.1 Viola-Jones

The Viola-Jones framework is the first facial detection system to offer real-time object detection rates. The Viola-Jones framework can be used for more than just facial detection, but, the discovery of the algorithm was largely motivated but it. The framework is made up of a series of steps. First, is the feature types and evaluation of those feature types. Second the algorithm is trained using an Adaboost classifier. The third step involves mapping over the image and applying Haar-like with the Adaboost classifier to classify whether an image was an object or not.

Haar-like features are digital image features used in object recognition. Their name comes from a similar mathematical construct called a Haar wavelet [quote]. A rectangular Haar-like feature can be thought of as the difference of the sum of pixels of areas inside the of a rectangle. These features can possess any position or scale inside of the original image. There are three types of features in the feature set: 2, 3, and 4-rectangular features. These Haar-like features can be computed very quickly by using integral images. For 2-rectangle features, only six lookups are needed to calculate the integral image,

3-rectangle features need eight and 4-rectangle features only need nine lookups.

The model is trained using a series of weak learners. These weak learners are then combined to form a much stronger classifier [9]. The boosting algorithm that is used in this case is the AdaBoost algorithm.

During the classification process, the system detects objects by moving a sliding window across the image and applying these smaller classifiers to a region defined by the current location of the window. If any of the weaker classifier results comes out negative, the labeling stops and the window slides to the next sub-image. However if the classification continues with positive results up to the strongest classifier, then an object is detected. In the case of facial recognition, the coordinates of a face are returned to the user.

Because of it's use of Haar-like features, the Viola Jones algorithm is an extremely efficient algorithm of object recognition. It's also a very effective because it can be used to detect many different kinds of objects if trained on them. However, the Viola-Jones algorithm does have some shortcomings. It's poor at handling different lighting conditions, it cannot detect rotated images, it's poor at detecting facial side profiles, and ultimately to implement Snapchat filters, we need more fine-grain control over the facial landmarks in order to more accurately map images onto faces.

2.2 Local Binary Patterns

Local Binary Patterns (LBP) improve upon the Viola-Jones algorithm by including texture classification. In the case of facial recognition, each face can is comprised of subfeatures that can effectively taken into account. This is similar to how convolutional neural networks are learned at each hidden layer.

The first step is to create a LBP texture description by converting the image to grayscale and select pixels within a neighborhood *r*. The LBP value for the center pixel is then calculated and stored in a 2D output array for the same size as the neighborhood sample. The values in the resulting 2D array are binary representations of whether or not the center pixel's value is greater than or less than a threshold value. Then the 2D array is transformed into a 1D binary representation and the aggregated into a final value.

In order to detect a texture, you must measure a collection of LBPs over an image patch and form a histogram for each window. Once each of the histograms are obtained, they should all be concatenated to form a feature vector of the entire window. These features can then be processed to determine. Overall using LBPs for the Viola Jones framework will result in greater speed but yield less accurate results. In addition, LBPs use integers as opposed to Haar with uses floats which contributes to the accuracy/speed tradeoff.

2.3 Regression Based Landmark Detection

For many applications, just detecting the presence of a face is not enough. In order to manipulate facial images or to detect emotions, we need to landmark specific locations on a face in order to gain more information.

The shape regression approach is able to predict facial shape in a cascade manner. Beginning with an initial shape S^0, S is progressively refined by estimating a shape increment ΔS stage by stage following the equation:

$$\Delta S^t = \mathrm{W}^t \Phi^t(\mathrm{I}, \mathrm{S}^{t-1})$$

where I is the input image, S^{t-1} is the shape from the previous stage. Φ^t is a feature mapping function and W^t is a linear regression matrix [11]. The W matrix attempts to minimize the sum of square errors [12]. This can also be done using a tree based regressor.

Tree based regressors fit to the residual targets using a gradient boosting algorithm [12]. At each split node in the regression tree, a decision is made depending on the intensities of two neighboring pixels.

2.4 Neural Net Based Landmark Detection

More recently neural networks and more specifically, convolutional neural networks (CNNs) have been used in research in order to do facial landmark detection. The problem with a standard CNN is that they are usually used for classification and the problem of landmark detection is a regression problem.

Tweaked Convolutional Neural Networks (TCNNs) have been proposed to fine-tune the final layers for particular head poses [13]. The proposed network architecture consists of four convolutional layers followed by a fully connected layer. The final output layer is a $2 \times n$ layer where n is the number of facial landmarks to detect. In order to avoid overfitting, L2 normalization is used and the model is trained with an ADAM optimizer [13].

These types of networks are fast becoming the popular state of the art in terms of accuracy of representing facial landmarks. This is because CNNs naturally learn a hierarchy of facial representations. However, for a model to succeed in a production environment like Snapchat, the model must be accurate and fast. There haven't been many performance benchmarks for speed of CNNs. In order to operate at 60 fps, all operations including facial location, landmark detection, and image rendering must be done in less than 16 milliseconds per frame (and even less than that due things like garbage collection).

3. IMPLEMENTATION

We experimented with multiple model training techniques and parameters. To facilitate expedient research and efficient computing, we restricted our model training to three different LBP models and two Haar models, all trained on the same data.

3.1 Data Collection

We used the Labeled Faces in the Wild dataset [8] to conduct our experiments. The data set contains more than 13,000 250x250 images of faces collected from the web. Each face has been labeled with the name of the person pictured. 1680 of the people pictured have two or more distinct photos in the data set.

3.2 Data Processing

OpenCV offers a manual annotation tool, `opencv_annotation` to facilitate target object detection in model training. Running the tool presents the user with an image in which a rectangle can be drawn around the target object (or objects) in the image. Because it is a time consuming task to manually annotate hundreds or even thousands of images, some of which can contain multiple target objects, we only trained one model using this technique, and only on 150 randomly chosen images from the dataset.

OpenCV also offers a sample creation tool, `opencv_createsamples` with various optional parameters by which to create a vector of image binary files by which to train models. The parameters we used to train our models are explained in model training section that follows.

3.3 Facial Detection Model Training

We trained several models with varying parameter inputs to better understand how these models perform. The summary below outlines the parameters we used on the models we trained.

Where the technique is noted as "manually annotated", we manually annotated the total positive images with rectangles identifying the face(s) in the images. Where the technique is noted as "auto annotated" we used a command line utility to create an annotation file where the face detection rectangle dimensions are the same as the target image dimensions.

For the sake of expedient research, we used mostly default parameters, and only varied the number of positive and negative samples, noted as "*N*" below. We used a heuristic of 85% of the total training positives (noted as "Total Pos/Neg *N* below) for training the models, based on advice from users in several public posts on answers.opencv.org.

Larger dimension images take longer to train than small dimension images, and generally, Haar models take longer to train than LBP models. We trained only one model on most of the data we had available, but we restricted these images to 24x24 to save on training time.

All model training was completed on a 2016 Macbook Pro with 4-core 2.5 Ghz and 16 GB RAM.

Technique	**Total Pos/Neg *N***	**Pos *N***	**Neg N**	**Dims**	**Train Time**
LBP (manually annotated)	150/1000	127	500	50x50	**9 min**
LBP (auto annotated)	12392/3019	8000	4000	24x24	**45 min**
LBP (auto annotated)	8000/1000	5000	1500	50x50	**4h, 24m**
Haar (auto annotated)	12932/3019	10000	4000	24x24	**5h, 1m**
Haar (auto annotated)	8000/1000	5000	1500	50x50	**12h, 5m**

There is a clear advantage in terms of speed to training models with smaller dimensions. But in our research, we found this comes with a cost. The LBP model with lower dimensions performed much worse than the others (see "Testing Results").

3.4 Landmark Detection

We used one landmark detection technique to test and calibrate our filter implementation. The OpenCV library comes with a Haar nose classifier that we used to detect a nose, but only once a face was detected. We did not further investigate landmark detection in this research.

3.4 Mask Implementation

Once a face is detected and the OpenCV model "draws" a rectangle around the face (the rectangle is based on the annotation file in the model training), we were then able to use arithmetic to draw and place images elsewhere in the image or live frame, based on the geometry of the target object rectangle. The geometry of the target object rectangle in OpenCV is as follows, where the top left of the rectangle is the first point of reference and the bottom right of the rectangle is the second point of reference:

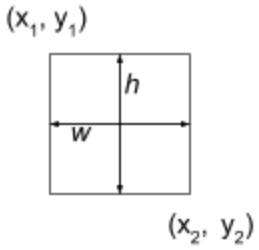

fig 1. OpenCV represents face detection using x_1, y_1, w, and h whereas image positions need to be computed and rendered as x_1, y_1, x_2, and y_2

4. TESTING

To test our models, we implemented several tests to measure the performance of the trained models. The tests only involved the detection of a face, as the implementation of the mask filter depends critically on the detection of a face within an image. If a face is not detected, the mask filters cannot be placed within the image.

4.1 Testing Framework

For the sake of time and code simplicity, we implemented a heuristic accuracy measurement to test how accurate the trained models work on test data. The heuristic nature of the accuracy metric is explained below.

The test data for each model was the data that was explicitly not used for training. For example, if there a total of 200 images in the initial data set, and the model was trained on 150 of them, then the remaining 50 were used for testing.

We ran a test on each of the images in the test data set, where the test returned a '1' if there was a face detected, and '0' otherwise. The accuracy rate was calculated as such:

$$\Sigma\,(\textit{all test results}) \,/\, \Sigma\,(\textit{tests})$$

Because some of the images in the data set have multiple faces, it's possible the accuracy rate can be above 1.0 for excellent classifiers and/or data sets where the majority of the images have multiple faces.

4.2 Testing on Still Images

We include a benchmark metric for the frontal face classifier that ships with the OpenCV library. The frontal face classifier was trained on 130 grayscale pictures with 530 unique faces, where 5000 positive samples were used with 3000 negative samples [10] and uses the Haar features described in the section above.

Technique	Positive Samples	Negative Samples	Dims	Accuracy
Haar (benchmark	5000	3000	?	**1.076**
Haar (auto annotated)	5000	1500	50x50	**.7476**
LBP (auto annotated)	5000	1500	50x50	**.7281**
LBP (manually annotated)	127	500	50x50	**.6514**
LBP (auto annotated)	10000	4000	24x24	**.2355**
Haar (auto annotated)	10000	4000	24x24	**.1599**

4.3 Testing in Real-Time Live Camera Stream

Our trained models worked successfully in live camera stream tests. The models were able to detect a face as well as properly place masks. The models with 50x50 dimension training produced the most stable real-time results.

5. CONCLUSION

Computer Vision is an exciting technology that holds many promises for the future, including driverless cars and many other applications in retail and security. We succeeded in implementing an open-source, real-time version of a very popular production level application for image filtering. Although our trained models were not as accurate in still-image testing, they still were successful in producing a real-time filter. Although Haar training took 3x as long to train than LBP, it performed better. Perhaps increasing the number of negatives to the same as the benchmark and increasing the dimensions would yield similar results.

Much work lies ahead to overcome challenges inherent in this type of research.

5.1 Challenges

OpenCV is an open-source product and documentation is not robust. Therefore, it is imperative to start work early on any experimentation with the library. We encountered many challenges installing the library and its dependencies and Python bindings in various environments, including in Amazon EC2.

Computer Vision requires a significant amount of time to train some models and a significant amount of data. In the case of face data, this can present a challenge in terms

of privacy and the diversity of the face data (skin color, features, angles). As for the time requirement to train models, tbb (Intel Threading Blocks) can facilitate running multiple cores simultaneously (which we used in this research) but even still, models such as Haar are challenging to train quickly at larger dimensions than 24x24.

5.2 Future Work

Future work in this area may include improving OpenCV documentation and installation materials, collecting more data for model training, and running many more model variations with many more parameter changes. Future work may also include creating an Amazon EC2 Community AMI with all of the updated OpenCV software as well as the Python bindings with multi-threading support to take advantage of the EC2 computing infrastructure. A community instance such as this would make it much easier and efficient for researchers starting in this field.

In addition, we would like to explore facial landmark detection in 3D for 2D images. This would help to produce more life-like filters and allow things like hats to be rotated when the position of the head rotates.

8. REFERENCES

[1] http://mylio.com/true-stories/next/how-many-digital-photos-will-be-taken-2017-repost

[2] P. Dollar, P. Welinder, and P. Perona. Cascaded pose regression. In Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.

[3] Y. Sun, X. Wang, and X. Tang. Deep convolutional network cascade for facial point detection. In Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013.

[4] X. Zhu and D. Ramanan. Face detection, pose estimation, and landmark localization in the wild. In Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. IEEE, 2012.

[5] C. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In International Conference on Computer Vision, 1998.

[6] H. Rowley, S. Baluja, and T. Kanade. Neural network-based face detection. In IEEE Patt. Anal. Mach. Intell., volume 20, pages 22–38, 1998.

[7] P. Viola, M. Jones, Robust Real-Time Face Detection, International Journal of Computer Vision 57(2), 2004.

[8] G. Huang, M. Ramesh, T. Berg, E. Learned-Miller, Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments, University of Massachusetts, Amherst, Technical Report 07-49, October 2007. http://vis-www.cs.umass.edu/lfw/index.html

[9] Freund, Y. Boosting a weak learning algorithm by majority. Information and Computation 121(2), 256–285 (1995)

[10] R. Lienhart, A. Kuranov, V. Pisarevsky, Empirical Anlysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection (2002) http://www.lienhart.de/Prof._Dr._Rainer_Lienhart/Publications_files/MRL-TR-May02-revised-Dec02_1.pdf

[11] S. Ren, X. Cao, Y. Wei, and J. Sun. Face alignment at 3000 fps via regressing local binary features. In Proc. Conf. Comput. Vision Pattern Recognition. IEEE, 2014.

[12] V. Kazemi and J. Sullivan. One millisecond face alignment with an ensemble of regression trees. In Proc. Conf. Comput. Vision Pattern Recognition. IEEE, 2014.

[13] Y. Wu, T. Hassner, K. Kim, G Medioni, P. Natarajan. Facial Landmark Detection with Tweaked Convolutional Neural Networks. https://arxiv.org/pdf/1511.04031v2.pdf

[14] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Facial landmark detection by deep multi-task learning. In European Conf. Comput. Vision, pages 94–108. Springer, 2014.