Categorical Classification and Deletion of Spam Images on Smartphones using Image Processing and Machine Learning

Arjit Sachdeva
IIM Indore
Indore, India
P14arjits@iimidr.ac.in

Rishabh Kapoor
IT Dept.,
BVP, Delhi, India
rishabh.kapoor2409@gmail.com

Amit Sharma
Zero1, Inc
Delhi, India
amit@zero1.io

Akshit Mishra
Zero1, Inc
Delhi, India
akshit@zero1.io

***Abstract*— We regularly use communication apps like Facebook and WhatsApp on our smartphones, and the exchange of media, particularly images, has grown at an exponential rate. There are over 3 billion images shared every day on Whatsapp alone. In such a scenario, the management of images on a mobile device has become highly inefficient, and this leads to problems like low storage, manual deletion of images, disorganization etc. In this paper, we present a solution to tackle these issues by automatically classifying every image on a smartphone into a set of predefined categories, thereby segregating spam images from them, allowing the user to delete them seamlessly.**

Keywords — image processing; face recognition; whatsapp; classification; machine learning.

I. INTRODUCTION

Messaging apps have grown tremendously in recent years as the cost of mobile data has decreased, making it a much more cost-effective mode of communication. Among the most popular apps in this category, WhatsApp has a strong footprint with a market share of over 55% worldwide. It was reported[1] that on New Year's Eve 2016 alone, over 14 billion messages were sent in India; this included 3.1 billion images! Furthermore, an average of 3.3 billion [2] photos are shared on WhatsApp every day. With the rise in sharing of images, the problem of proper storage and arrangement of this media has turned into a major problem on mobile devices. This has led to a typical phone gallery to be cluttered with various images that a user receives across messaging apps, which further leads to issues like high storage usage, filtering out relevant images, deletion of spam images, and classification/arrangement of images into well-defined categories.

In this paper, we present an approach through which we could automatically scan and analyze every image on a mobile device and classify it based on its type and relevancy to the user. The system we developed also incorporates a self-learning process that improvises upon its decisions based on user feedback and third-party analysis in the modules we developed.

II. RELATED WORKS

Some work has been done on filtering spam images on emails. G. Fumera in their paper titled 'Image spam filtering using textual and visual information' [3] proposed to use OCR to extract text of the image along with text from email's body, which were first indexed, and then the SVM(Support Vector Machine) classifier classified that as a Legitimate or Spam email. Also, they tried to detect some content obscuring techniques (like noise) which could make OCR ineffective, for which they tried to get content by binarization pre-processing step performed by OCR algorithms.

Zhe Wang in their paper titled 'Filtering Image Spam with Near-Duplicate Detection' [4] proposed to filter spam images in emails. It takes advantage of nature of spam messages - that they are sent in large quantity and that machine generated spam images will look similar to each other. They first tried to detect spam message senders by using classical approaches like honeypots (dummy email accounts to attract spam senders), they trained their system on a non-spam image dataset so that their classifier learns how non-spam images looks like and then get threshold value for false-positive data. All emails first went through a traditional spam filter and then it went through their spam filter and all images were processed to detect whether they were actually spam or not and then emails marked as spam was stored in their *feature DB*, and its distance was calculated from other features so that it could be used in future.

Also, face detection and recognition are important aspects of our system. An image containing face and text might be a picture quote or photo of a person. These images might or might not be

978-1-5386-3446-2/17 $31.00 © 2017 IEEE
DOI 10.1109/MLDS.2017.10

useful to the user, for e.g. a person may be interested in quotes by a famous personality, or even some photos of their friends might contain text as well. So if the system is able to identify faces in a picture and check if those faces are relevant to users or not, it would help the user in separating important pictures containing both text and faces from unimportant ones of the same type. We used *Histogram of Oriented Gradients* (HOG) to detect and check whether images have known faces or not. N. Dalal and B. Triggs in their paper titled 'Histogram of Object Gradients for Human Detections' [5] explained how the HOG approach significantly outperformed other methods for human detection. They first took a dataset of positive training examples and trained their classifier and repeated it on a dataset of negative examples, and then it was tested on a random set. HOG proved to be an effective method for object detection and it outperforms other object detection method so it can be used to detect humans very precisely. Also L. R. Cerna et. al. in their paper titled 'Face Detection: Histogram of Oriented Gradients and Bag of Feature Method' [6] analyzed the performance of HOG in face detection. They posed HOG as reminiscent of edge orientation histogram, SIFT descriptor and shape context. As HOG is computed on dense grid of cells that overlap local contrast histogram normalizations of image gradient orientations to improve the detector performance. They first extracted HOG descriptors of each image and then apply the clustering process to those features and obtained clusters of different sizes and every cluster center is regarded as codeword that was used to construct histograms based on the frequency of their appearance in the image. They chose class of each feature using the minimum distance to the cluster center, and those histograms were used to train SVM classifier to detect face in an input image. They showed that this method was very accurate in detecting faces and that to improve results Elastic Bunch Graph Matching Method could be used to extract facial features like eyes, nose etc. and then HOG descriptors could be obtained without using the entire image to enhance performance. Although a method proposed by P.Viola and M.Jones in their paper titled "Robust Real-time Object Detection"[19] has been widely used as it is an efficient moving person detector, using AdaBoost to build a progressive but complex region rejection rules based on Haar-like wavelets and space-time differences, this method was fast enough to run on economical devices but HOG is a more faster and reliable solution.

III. Broad System Outline

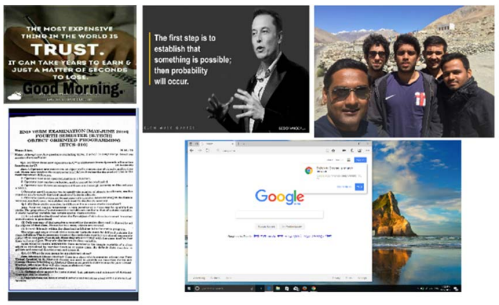


Fig 3.1: Types of Images

To arrive at the classification of an image as spam(unimportant) or important, the system should be able to understand the contents of every image. Moreover, 'importance' is a subjective term that would vary based on user preferences. To achieve a broad understanding of images, we analyzed a set of over 5000+ random images collected from smartphones of different categories and found that a majority of those images could be categorized in five sets, as follows:

1. Camera Images.
2. Screenshots or scanned Documents.
3. Quotes
4. Quotes by famous people
5. Computer Screen shots

An example image of each category is shown in Fig 3.1. A user might not need every kind of these images. For e.g. Camera images which generally contains photos of people like friends or family might be of greater importance to a person rather than a quote which consists of greetings. Also, if the user likes certain personalities and likes to collect their quotes, it might also be relevant to the user. Also, scanned images and documents are important for people and so they might want to keep them for future use.

Our system is based on two modules. We first need to know what kind of images will be important for a user and what kind of images they wish to discard.

The first module checks whether a photo contains a known face. For this we implemented the Facebook Login SDK, and fetched 5 profile pictures of each Facebook friends and, further fetched the contact images residing on the device itself too, to improve our accuracy.

All images which consist of known faces are marked important and hence are not processed further.

Other images are then transferred to the second module which processes them further to check whether it contains text, the context of that text and whether that image could be relevant for the user by taking the textual analysis into account.

IV. Crawling and Model Training for Known Faces

In the first module, we separate those images which contain pictures that have faces that could be of the user's friends and family as these images would be relevant for him/her.

We used face_recognition module [7] in python to detect faces from the image corpus we fetched via the user's Facebook friend list and contact images on the device. The snippet is shown in Fig 4.1. This module first finds all faces with the help of a method called Histogram of Oriented Gradients (HOG) [5], and isolates all faces present in the photograph and then it applies another algorithm called face landmark estimation invented by Vahid Kazemi and Josephine Sullivan [8], so that we could detect faces that are pointing in different directions. Then we encode those faces so that our model saves the face of that person with all its feature. We trained our module on that set and ran it against all of the images present in the user's phone gallery to recognize relevant faces as explained above.

```
import face_recognition as fr
from PIL import Image
import os

faces=[]
known = []
un = []
for filename in os.listdir("train"):
    fac=fr.load_image_file("train//"+filename)
    face=fr.face_encodings(fac)[0]
    faces.append(face)

for filename in os.listdir("test"):
    fac=fr.load_image_file("test//"+filename)
    print(filename)
    enco = fr.face_encodings(fac)[0]
    result=fr.compare_faces(faces,enco)
    rset=list(set(result))
    if True in rset:
        print("Contains Known Face")
        known.append(filename)
    if len(rset)==1 and rset[0]==False:
        print("Unknown Face(s)")
        un.append(filename)
```

Fig 4.1: Face Recognition Snippet

Finally, to check whether an unknown image consists of a known face, we encode faces in that image and check them with our database. If facial features match and our model returns true, we can even tag the name of that person and mark the image important and save it without running any further tests. However, if the model returns false, we transfer the image to the second module for further analysis as no known face was discovered in the image. Fig 4.2 shows working of this module, first the profile picture was taken from Facebook account and then second image was fed to system to know whether the image contains any known image or not, the system detected a known face and to show the result we marked it.

Fig 4.2: Training Image and Result

V. Duplicate Image Detection

A user might receive the same image from many sources like friends, chat groups or even chat applications, and it makes sense to keep one copy and delete the others.

Our system also performs this check i.e. if there are duplicate images, our system can detect them in a user's device.

Mean Squared Error Method(MSE) and Structure Similarity Index(SSIM) can be used to find out similarity between two images. In our system, we used a combination of these methods to improve our accuracy.

In Fig 5.1 we showed if there are 3 images, how will our system compute results. First two images are almost same with only difference in pixel size, while third image is similar to them but not exactly same.

Fig 5.1: Similar Images Output

Mean Squared Error Method [9] is very easy to implement. It's a signal fidelity measure that compares two signals by providing a

quantitative score that measures degree of similarity or conversely, level of difference between them. Usually, the first signal is assumed to be important and free from errors while the other is assumed to be distorted or contaminated by errors.

Equation we used to calculate MSE-

$$MSE = \frac{1}{ab}\sum_{i=0}^{a-1}\sum_{j=0}^{b-1}[A(i,j) - B(i,j)]^2 \quad (1)$$

Here, A and B is matrix representation of images. We converted them into a numpy matrix and then their values were converted to float. And *a* and *b* are their sizes respectively.

A MSE score of 0 means that two images are perfectly similar and as absolute value increases it signifies less similarity in those images.

MSE seems very easy to implement but one of its major drawback are the difference in pixel intensity might not mean that image's context is dramatically different. Hence two exactly same images with distinct colour intensity would not yield desired results.

Mean Square: 0.00, SSIM: 1.00

Fig 5.2: First vs Second

Zhou Wang et. al. proposed a method called Structural Similarity Index(SSIM) in "Image Quality Assessment: From Error Visibility to Structural Similarity" [10], which tries to model the perceived changes in the structural information of an image. In SSIM measurement, the prediction of image quality is based on uncompressed or noise free images as reference.

SSIM provides better results than earlier approaches such as peak signal-to-noise ratio(PSNR) and mean squared error(MSE).

SSIM uses the following equation-

$$SSIM(x,y) = \frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x{}^2+\mu_y{}^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)} \quad (2)$$

Here,

x and y are two images of common size N×N,

μ_x is average of x,

μ_y is average of y,

σ_{xy} is covariance of x and y,

σ_x^2 is variance of x,

σ_y^2 is variance of y,

$c_1 = (k_1L)^2$ and $c_2 = (k_2L)^2$ two variables that stabilizes the division with weak denominator,

L is dynamic range of pixel values, and,

$k_1 = 0.01$ and $k_2 = 0.03$ by default

Mean Square: -12.73, SSIM: 0.25

Fig 5.3: First vs Third

SSIM methods have more parameters than MSE, as it tries to model the interpreted change in structure, while MSE is estimating errors. The differences are very less but results are dramatic in many cases. MSE method is substantially faster but one major drawback is it only estimates the perceived error of the image, SSIM is a slower method but it is able to perceive change in structural information.

To improve our results, we used a combination of SSIM and MSE methods. First MSE is calculated, if the difference is large (greater than 1000), We make a check that whether two images

are structurally similar. If SSIM index is greater than 0.75(i.e. they have 75% similarity), the two images are structurally similar and hence a user might not want to save both of the images. But if MSE is small we can directly predict that two images are similar and hence one of them could be removed.

In Fig 5.2 we see that first two images were same and hence the SSIM index was 1 and MSE was 0 signifying that they are structurally same and doesn't have any difference between them, while in Fig 5.3 we see MSE calculated was -12.73 signifying that images have some identical features, so we calculated SSIM index which was 0.25 signifying 25% similarity, in this case user shall be allowed to make decision whether they would like to keep both images or discard any one of them.

VI. TEXTUAL ANALYSIS VIA TESSERACT-OCR

Image that contain text can typically be a quote, scanned document or even a screenshot. We can detect this by using an OCR (Optical character Reader). For this we have used Pytesseract [11] which is a Python wrapper for Google's Tesseract-OCR [12] Engine and provides practically best results till date. Also, it can read all image types supported by Python's imaging libraries like jpeg, png, gif, bmp etc.

We first open an image using PIL (Python Imaging Library) [13] module as it can support many image formats and it provides powerful image processing and graphical capabilities.

We then process the image using Pytesseract module which can extract all the text from image. It also identifies symbols and digits easily and hence we could read most of the characters present in an image.

Tesseract OCR first stores the outlines of the components, which might be difficult and a costly approach but due to this it is simple to detect inverse text. It can also handle *white-on-black* text easily. First text outlines are grouped into Blobs and then those are in turn organized into text lines and so, those regions are analyzed for fixed pitch or proportional text. Text lines are broken into words differently according to their character spacing, and fixed pitch is separated immediately by character cells. Proportional text is then broken into words by definite and fuzzy spaces, and then recognition is carried out as a two-step process [14].

In the first step, the processor tries to recognize each word if it is satisfactory, and then it is passed to an adaptive classifier as training data. Then adaptive classifier gets a chance to recognize text more precisely in the remaining document. Then system transfers to a second step in cases where words that weren't recognized are tried again. In the final phase, fuzzy spaces are resolved and checks are made to locate small and capitalized text.

Based on our experiments on a diverse image corpus, we found that a quote image generally contains 25-80% text, while a scanned document contains more than 80% text in it.

We used the following steps to classify images based on the OCR analysis:

a) Step 1: Text Area Computation

To find percentage of area occupied by text. To find this we use OpenCV [15] with Pytesseract module to find area that contains text. We first opened the image with the help of OpenCV and then converted it to grayscale, so that we can easily find contours that bounds text. After that, we converted the image into a grayscale image and we applied a threshold function of OpenCV that allows us to separate pixels above the threshold value.

This could help us separate text from rest of the image as text would lie above threshold value and so we can easily get a contour bounding text.

b) Step 2: Cropping Desired Area

As we use threshold function, some portions that might not contain text could also get contoured with text, as they might lie above the threshold value.

Fig 6.1: Representation of contoured position

So, to solve this problem we send that contoured area to the OCR module and detect whether it contains text or not. If it contains text, that area is added to total area bounded by text but if OCR returns null i.e. there is no text present then we can discard that area.

In Fig 6.1 we see that a picture might contains some pixels that may be processed as text by the system, so we have to remove this discrepancy to exactly calculate how much text a picture contains. Also by multiple observations, we found that if length of text is one, then it was already considered with some other part. Then

also we can discard that area, and hence, only relevant area would be considered, which contains text.

To overcome problem shown in Fig 6.1, we check that whether the contoured area contains text or not. To achieve this task we send the cropped area to an OCR and if OCR detects presence of text, we add that area to total text area of image or discard that area otherwise. Fig 6.2 shows the code how we could achieve this task.

```
        # Rectangles that bound contours
        [x, y, w, h] = cv2.boundingRect(contour)

        #Rectangle on original Image
        cv2.rectangle(img, (x, y), (x + w, y + h), (255, 0, 255), 2)
        #Cropping the image and sending it to OCR to filter images that have text
        cropped = img_2 [ y :y +  h , x : x + w]
        cv2.imwrite('results\\resultant_{}.png'.format(index),cropped)
        ab=Image.open('results\\resultant_{}.png'.format(index))
        if getImgText(ab) == 0 :
            pass
            # If there is no text present we should not include that area
        else:
            t=getImgText(ab)
            if len(t)>2:
                arr=w*h
                area+=arr
            else:
                pass
                #Generally this is included with other contours.
                #So we can skip this
        index = index+1
    remove_files(index)
    return area
```

Fig 6.2: Cropping the contoured portion and then sending it to OCR to check whether they contain text

c) Step 3: Face in the Image

There might be instances where percentage of text in the image is below 25% but it still could be a quote which doesn't have relevance to a user, as shown in Fig 6.3.

These are generally quotes that are given by eminent individuals and they contain photos of them. Hence if we detect a face in the image which does not belong to the corpus of known faces that we extracted (part IV), and then, we can mark those images as quotes.

Fig 6.3: Detecting Face(s) in an image

To find faces in images we used Open CV face classifier (Haarcascade Frontal Face Classifier).

Generally, an image with a quote is composed of different shades of the same color(ref: Fig 6.3). We used Haar features to detect faces in an image. Haar like features are good for detecting frontal faces as they have features like nose bridge which is brighter than surrounding faces. Similarly, detecting eyes is easier in Haar based classifiers for the same reason. E.C. Cruz et. al. in their paper titled "A comparison of Haar-like, LBP and HOG approaches to concrete and asphalt runway detection in high resolution imagery" [18], showed how different classifiers perform in different conditions, and we found that Haar features could easily differentiate between shades in an image, and so, images with quotes, which had frontal face in most situations could be easily detected by Haar classifier. Also, P. Negri in the paper titled "Benchmarking haar and histograms of oriented gradients features applied to vehicle detection," [17] has shown that HOG features are best in capturing shapes and outlines in an image which would be better when we need to detect faces in profile photos as they contain faces from different angles, and not specifically for the usecase in this section.

OpenCV has a Cascade Classifier which is quite accurate in detecting faces in an image. To detect whether an image has faces we create a function called *checkFaces* and pass file in it. OpenCV requires an image to be converted to grayscale as it can differentiate those pixels easily. The function returns 0 if there are no faces found and 1 if there are faces.

The user might be interested in saving a quote from a certain personality, so at the beginning, we could ask the user if they want to save that quote and even other quotes from the same person. Based on the user response, our system will learn the preference and accordingly take actions on such images.

d) Step 4: The Relevance Decision

After we have completed all the steps as described above, we should separate important images from the one a user might not be interested to store in his device.

We ran our system on a large corpus of various images from actual mobile devices and arrived at the following conclusions that should be used as thresholds:

- If an image has 25-80% plain text, it would most likely be a quote.
- If it contains text less than 25% but has unknown face(s), then it can be a quote from a famous personality.
- If it contains more than 70% of plain text, but there are more than 7 sentences, then it contains a scanned

document or screenshot. For this, we used the Stanford NLP parser, which is described below.

- If it contains some text and digits but percentage of that text is below 40%, then it could be a screenshot.
- If there is little or no text in an image, it could be a camera image of some static object.

```
for comments in contents:
    try:
        comments=comments.lower()
        res = nlp.annotate(comments,
                   properties={
                       'annotators': 'sentiment',
                       'outputFormat': 'json',
                       'timeout': 1000,
                       })
        '''Extracting all Nouns from the comment'''
        sinNoun= nlp.semgrex(comment, pattern='{tag: /NN|NNP|NNS/}', filter=False)['sentences']
        nouns=[]
        for n in range(len(sinNoun)):
            for t in range(len(sinNoun[n])-1):
                nouns.append(sinNoun[n][str(t)]['text'])
```

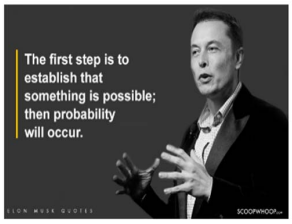
Fig 6.4: Stanford Parser to check image is a document or a quote

We use Stanford's NLP [16] parser to check content of text extracted from images. We can get count of sentences and also their sentiments - quote messages tend to have *very positive* sentiment associated with them and don't have more than 5 sentences. As shown in Fig 6.4 how we used Stanford NLP parser in our system to detect nouns in a screenshot or an image.

At the end of the complete processing, the final classification is shown to the user for review. If the user marks a 'spam' image as important, the system confirms the type of the image. If there is a known face in it, our system learns this information and updates the training dataset as described in Section IV.

VII. Experimental Results

After running our system on a set of different images, following are the outputs that were generated.




```
17mu1KA7c24gMxp.jpg
0.2162175
17mu1KA7c24gMxp.jpg is a
picture quote
```

```
screenshot.jpg
0.9567827130852341
screenshot.jpg is a screenshot or
scanned document
```


```
YmbSH0sUUlcxU4!.jpg
0.3b09475bb5b5b5b6
YmbSH0sUUlcxU4!.jpg is a quote
```


```
2017-08-22
2017-08-22.png might be a
screenshot
```

Fig 7.1: Some Experimental results

We processed the corpus of images that we had extracted from different smartphones as mentioned earlier. Dataset was collected from image folders of popular instant messaging services like WhatsApp, Telegram, Line, WeChat etc. Our system could classify plain quotes and photo quotes with a 90% accuracy. Our first module to detect known faces achieved an accuracy of almost 83% in its classification. Screenshots which contained text or scanned documents were classified with 85% accuracy, and computer screenshots and screenshots which had minimal text were classified with an accuracy of 70%.

Fig 7.1 shows some of our experimental results, the name of image along with percentage text and category has been shown. From here we could transfer control to user to verify whether or not they would like to delete these kind of images.

VIII. Challenges And Future Work

Most of the problems that occurred while classification were due to some images having very small pixel size, or documents that were blurry. We tried to enhance those images but the results were stagnant.

1) Blurry Images

Documents and quotes sometimes are very noisy or blurry. Hence detecting text becomes very painful and inaccurate. Text from a blurry image might return some unrecognised characters and hence any Natural Language Processor might not able to process it. That document could be marked as spam or might not be processed as context won't be parsed.

2) Images with similar pixels-

In some images, the background and text pixels are not much different. Hence, we would not be able to extract complete text.

To overcome this situation, we first convert image to grayscale, then dilate and erode it. In erosion, the boundaries of foreground text are eroded away, so thickness of text is eroded away and so are small noises. In dilation, the foreground text or objects becomes brighter than background. Hence, we can separate foreground text from background. As erosion will remove noises but shrink text, so we dilate it to make it thicker. Despite this, some quotes which had white text on pale yellow background were not detected. We could identify that it had text, but we were not able to extract that text from these images. So, we might not be able to classify it correctly as a screenshot or document.

IX. REFERENCES

[1] J. Schwartz, "The Most Popular Messaging App in Every Country," SimilarWeb, 24 May 2016. [Online]. Available: https://www.similarweb.com/blog/worldwide-messaging-apps. [Accessed 8 August 2017].

[2] Gadgets 360 Staff, "WhatsApp Turns 8: Here Are 8 Incredible WhatsApp Stats You Don't Know," NDTV, 24 February 2017. [Online]. Available: http://gadgets.ndtv.com/apps/features/whatsapp-turns-8-here-are-8-incredible-whatsapp-stats-you-didnt-know-1663073. [Accessed 8 August 2017].

[3] G. Fumera, I. Pillai, F. Roli and B. Biggio, "Image spam filtering using textual and visual information," in *MIT Spam Conference 2007*, Cambridge, 2007.

[4] Z. Wang, W. Josephson, Q. Lv, M. Charikar and K. Li, "Filtering Image Spam with Near-Duplicate Detection.," in *Council of the European Aerospace Societies (CEAS)*, 2007.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)* , San Diego, 2005.

[6] L. R. Cerna, G. Cámara-Chávez and D. Me, "Face Detection: Histogram of Oriented Gradients and Bag of Feature Method," in *International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV)*, Athens, 2013.

[7] A. Geitgey, "Face Recognition - Face Recognition 0.10 documentation," [Online]. Available: https://face-recognition.readthedocs.io/en/latest/readme.html#. [Accessed 14 August 2017].

[8] V. Kazemi and J. Sullivan, "One Millisecond Face Alignment with an Ensemble of Regression Trees," in *Computer Vision and Pattern Recognition (CVPR)*, Columbus, 2014.

[9] Z. Wang and A. C. Bovik, "Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures," *IEEE Signal Processing Magazine* , vol. 26, no. 1, pp. 98-117, 2009.

[10] Z. Wang, A. Bovik, H. Sheikh and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing* , vol. 13, no. 4, pp. 600-612, 2004.

[11] M. Lee. [Online]. Available: https://github.com/madmaze/python-tesseract. [Accessed 10 August 2017].

[12] R. Smith and Hewlett-Packard, "tesseract-ocr," [Online]. Available: https://github.com/tesseract-ocr/tesseract.

[13] S. L. A. (PythonWare), "PIL 1.1.16," [Online]. Available: http://www.pythonware.com/products/pil. [Accessed 10 August 2017].

[14] R. Smith, "tesseract-ocr documentation," [Online]. Available: https://github.com/tesseract-ocr/docs/blob/master/tesseracticdar2007.pdf. [Accessed 10 August 2017].

[15] G. Bradski, "opencv_library," *Dr. Dobb's Journal of Software Tools,* 2000.

[16] Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP Natural Language Processing Toolkit In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60. [pdf] [bib]

[17] P. Negri, X. Clady and L. Prevost, "BENCHMARKING HAAR AND HISTOGRAMS OF ORIENTED GRADIENTS FEATURES APPLIED TO VEHICLE DETECTION," in *ICINCO 2007, Proceedings of the Fourth International Conference on Informatics in Control, Automation and Robotics, Robotics and Automation 1*, Angers, 2007.

[18] u. E. C. Cruz, E. H. Shiguemori and L. N. F. Guimarães, "A comparison of Haar-like, LBP and HOG approaches to concrete and asphalt runway detection in high resolution imagery," in *BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, 2013, 2015.

[19] P. Viola and M. Jones, "Robust Real-time Object Detection," in *SECOND INTERNATIONAL WORKSHOP ON STATISTICAL AND COMPUTATIONAL THEORIES OF VISION – MODELING, LEARNING, COMPUTING, AND SAMPLING* , VANCOUVER, 2001.