Mobile Application for Passive Stroke Detection

Aimas Lund
Student ID: 20196534
Technical University of Denmark
s174435@student.dtu.dk

Joshua Kim
Student ID: 20196549
University of Illinois
joshuakim0724@yahoo.com

***Abstract*—In the United States, someone is having a stroke every 40 seconds. Every four minutes, someone dies of stroke. As of 2017, 140,000 Americans die from strokes annually [5], resulting in the third leading cause of death in the US. While applications have been created for detecting stroke, it deals with a limitation of only being able to confirm that a stroke has occurred [1]. Dealing with the problem of detection in the moment and timely medical assistance [6], we explore the possibility of a passive stroke detector application.**
In order to develop this detector, we researched detectable facial features that commonly occur during a stroke: the left eye, right eye, upper lip area, and the lower lip area [4]. Then, by extracting these features using Google's Firebase Machine Learning kit, we were then able to develop a detector of stroke by feeding a WEKA based naive Bayes classifier with the extracted features. As a result, the passive application will take one photo in any specified time-interval and determine if the user is having a stroke.

***Index Terms*—Stroke, Machine Learning, Background Service, Mobile Application.**

I. Introduction

In a world full of sudden and punishing dangers, strokes are not an exception. Strokes are currently the leading cause of serious, long-term disability, and the third leading cause of death in the United States. For treatments done within an hour of the stroke, patients have a much greater chance of survival and avoiding long-term brain damage.
With every minute a stroke is untreated, the average patient loses 1.9 million neurons and 13.8 billion synapses [6]. Each untreated minute is equivalent to 3.6 years loses of neurons. With every minute being extremely valuable, we wanted to investigate whether passively detecting stroke with a smart device was feasible.
We found that a common and clear giveaway of ongoing stroke is facial drooping. In essence, the one half of the face becomes paralyzed, and collapses in a fashion that is hard - if not impossible - for a person to replicate deliberately. And example of this is demonstrated in Figure 1.

II. Related Works

When researching related works, we found that other developers/researchers have already created a passive stroke detector. However, these applications are limited by requiring an external sensor, or smart watch, in order to track heart rate and alert users of a possible stroke. One purely mobile application was created by scientists in Valencia [7].
This app would ask users to perform three tasks of smiling, repeating a basic sentence, and raising both arms above their head while holding the phone. After computing the result, the app would notify the user if a stroke could have possibly occurred. While this application is similar to ours, it deals with the limitation of passively detecting the application. However, if used in conjunction with our application, it could serve as confirmation for the users.

Fig. 1: Simulated example of a stroke

III. The Solution

When developing this app, we had to consider three main factors - computation time, classification accuracy and power consumption. Given the app is supposed to run passively in the background, while the user performs other tasks on the device.
Therefore, we wanted our application to require the least amount of computation power and eliminate power consumption. This came down to minimizing the sample data, to perform computations on the least amount of pixels, and restricting camera use, as this is a unit that requires a lot of power to operate.
Furthermore, we want our application to be as accurate as possible, such that we eliminate the majority (if not all) false positives/negatives. False positives would lead to a decrease in the user experience, as the app will interfere with the user. Yet, we still want the application to correctly classify any occurrence of stroke in the user.

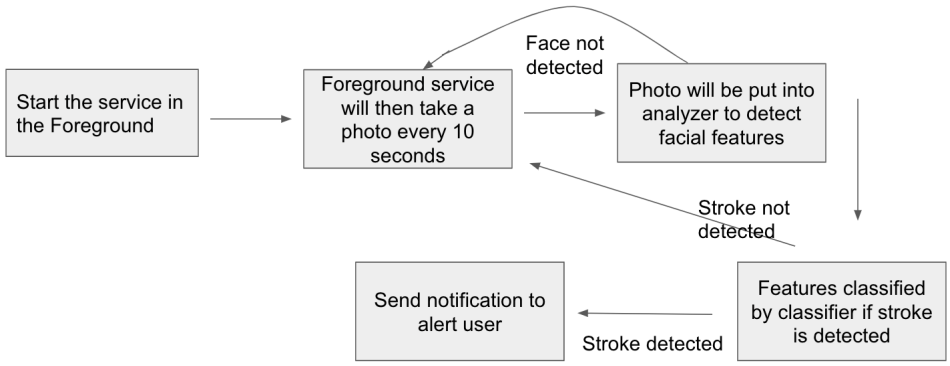

Fig. 2: Program flow

A. Implementation Summary

- Operating System: **Android**
- Language: **Java**
- APIs
  1) Google FireBase
  2) WEKA ML Kit

Algorithm 1: Stroke Detector Algorithm

```
Result: Prediction: String → "Normal" | "Drooping"
camera = Camera(res=480x640);
faceDetector = FireBaseDetector();
featureExtractor = FeatureExtractor();
classifier = NaiveBayes();
while buttonsDisabled do
    image = camera.takeImage();
    faceDetector.detect(image);
    if faceDetector.faces != null then
        predictions = [];
        for face : faces do
            instance = featureExtractor.extract(face);
            append(predictions,
              classifier.predict(instance));
        end
    else
    end
    return predictions
end
```

Note: *buttonsDisabled* is a boolean variable indicating whether the buttons in the application has been disabled. Using this, implies that the "start"-button has been hit, and therefore only the "stop"-button is left activated.

B. Implementation

In order to develop this application, we first needed a way to take photos, which we could analyze. Since this application should passively detect, while the user is using the phone, we decided to use the front-facing camera to create images of 480x640 resolution. We chose this resolution specifically, given this was the lowest preset resolution for the Samsung camera app, which contained more than 400x400 pixels, which was required for accurate contour extraction for Google FireBase. By using a smaller resolution we eliminate computation power and thereby also eliminating power consumption.

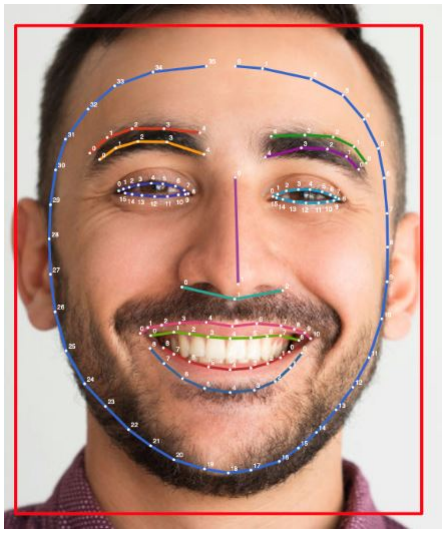

Fig. 3: Example of contours extracted from a face

After receiving the photo, the facial features points (referred to as 'contours') are extracted. These are a list of points which benchmark where certain parts of certain facial features are located in the image (see Figure 3).
By utilizing Firebase ML Kit, we are able to detect the face in real-time and generate contours for any face contained in the image. If a face is not detected, the application will ignore the photo and wait for the next photo to be taken. Facial contours created by the Firebase ML Kit have a lot of available datapoints such as ears, eyebrows, nose, etc., as shown in figure 2. We however primarily focused on areas best suited for facial drooping, eyes and mouth. While eyebrows

Prefix meaning		Suffix Meaning	
re	Right eye	0	Minimum x-value
le	Left eye	1	Maximum x-value
ul	Upper lip	2	Variance of set of x-values
ll	Lower lip	3	Average of set of x-values
		4	Minimum y-value
		5	Maximum y-value
		6	Variance of set of y-values
		7	Average of set of y-values
		8	Slope of pre-selected points

TABLE I: Overview of generated features for Naive Bayes classifier

were initially included, it was found to causing a lot of false positives, so they were excluded in our end result.

C. Detecting Facial Drooping

Using the extracted contours of a face, the minimum, maximum, variance, and average, for each X, Y value were calculated. In addition to these values, two key-points signifying the left- and right extremity of that specific feature was selected, and then the slope of the line through both points were calculated. This was done to detect whether or not the face would be drooping, based on the fact that one side of the face would hang lower than the opposite side.
These values would eventually be inputted into a classifier, and the classifier would determine whether a stroke or not was detected.

D. Classifying the image

In order to build the classifier, we trained a Naive Bayes classifier with a set of images with a neutral facial expression, and a set of images with drooping- or otherwise paralyzed faces. We had to expand the training data to contain other, similar types of facial paralysis, in order to have sufficient training data. With the 30 images we were able to get, we created the classification for drooping. Since we could not discover an online "regular" facial database, we decided to reach out and ask friends for "regular" photos, and manually find some more photos online. Utilizing these datasets, the classifier was trained. This model is automatically loaded in on start of the application.

E. Passive Detection

In order to create an application capable of passively detecting a stroke, running the entire process in the background would be required. By utilizing the ForegroundService library found in Java, we are able to create a passive monitoring service. This service will take a photo every 10 seconds, and then run an analysis on the face which will eventually be classified as "drooping" or "normal" by the classifier. If drooping is detected, then a notification will be sent to the user urging them to receive immediate assistance.
In table I, the features created for the classifier is specified.

IV. Results

Since we faced the limitation of testing how accurate stroke detection is, we decided to research more about the how much energy the application consumes. For the following data, a LG Nexus 5X phone was initially charged to 100%. A LG Nexus 5X uses about 2% battery over the course of a hour in idle mode. With the app running in the background, 13% battery was used resulting in around 11% battery being consumed by the application. Over the course of 2.5 hours, the phone was brought down to 70% with a computed power use of 85 mAh.

V. Challenges

We have faced quite a few challenges while developing this project.
The FireBase API seems to be designed for use in real time, and not to extract data from a lot of images in rapid succession. This became evident for us, when we wanted to train the model using the images we acquired. The FireBase API will automatically open a separate thread using the Task class, when it detects a face in an image. This posed many issues in how to control that separate thread, in order for the API to manage to compute the contours in a given image, before we fed the data to the classifier in another thread. We ultimately found a way to "hack" this, by using strategic break points in the debugger, in order to at least have a working model. However, we cannot provide the possibility to train the model using the "train model"-button, as initially intended.

We have also encountered problems with the limited amount of data that we have. Using the data provided, the model is stable, if it is used by a person who provided his/her face for the training data. The model can however trigger false positives for people who did not provide an image of their face, before the model was trained. The model is especially unstable with for female users, as the training data consisted mostly of male faces. A solution for this could be to ask the user to provide a photo of themselves, before using the app.

VI. Future Work

Using the two different APIs in order to classify whether the face is drooping or not is somewhat inefficient. Moreover, the Bayes Classifier might not be the bet classifier for such a problem. We suggest for future improvement on the classifier that a neural network can be introduced that works as a very niche object classifier, which classifies an image as either a normal face, a "non-face" or a drooping face. However, having a model that can accurately detect such small details in faces and also not give false positives, must require a lot of training data - which, to our knowledge, is not publicly available currently.

Given the simple way to use this app, we also discussed making a "click widget" for the app would be ideal. At its core, the app should just have a "click-and-go"-design. Therefore, there should not be a reason to open an app in order to active/deactivate the functionality that our app provides.

VII. What we have learned

If there is one lesson that we have learned, while doing this project, then it will be how much work and data it takes for Machine Learning to work properly. Due to the nature of human faces, it's extremely difficult to accurately pinpoint different without large data sets. Moreover, it gives good insight into why companies like Google and Facebook are so interested in getting big amounts of data, in order to perfect their models. However even with our lack of extensive data, we proved that passively detecting facial drooping is possible and will become more accurate if more resources and data are given.

VIII. Individual contributions

Tasks	Aimas	Joshua
Implement Camera	x	
Implement Service	x	x
Implement WEKA	x	
Implement FireBase	x	x
Test Model		x
Implement Contour Analysis	x	x
Acquire Training Data		x
Add Training Environment for WEKA		x
Energy Consumption Test		x

References

[1] J. Jaakkola, S. Jaakkola, O. Lahdenoja, T. Hurnanen, T. Koivisto, M. Pänkäälä, T. Knuutila, T.O. Kiviniemi, T. Vasankari, K. Airaksinen. Mobile Phone Detection of Atrial Fibrillation With Mechanocardiography. Available: https://ahajournals.org/doi/pdf/10.1161/CIRCULATIONAHA.117.032804

[2] FireBase App Development Platform https://firebase.google.com/use-cases

[3] WEKA Machine Learning Platform https://www.cs.waikato.ac.nz/ml/weka

[4] Better Health - Effects of Stroke. https://www.betterhealth.vic.gov.au/health/conditionsandtreatments/effects-of-stroke?viewAsPdf=true

[5] Stroke Center statistics on stroke in the United States http://www.strokecenter.org/patients/about-stroke/stroke-statistics/

[6] Integris - The Golden Hour of a Stroke https://integrisok.com/resources/on-your-health/2019/may/why-is-the-golden-hour-so-important-when-it-comes-to-stroke

[7] Related stroke detection app from Valencia's Polytechnic University https://newatlas.com/stroke-detection-app/55871/ (Paper unavailable).

Appendix

"overview_uml.jpeg" - Class UML file attached in deliverables.