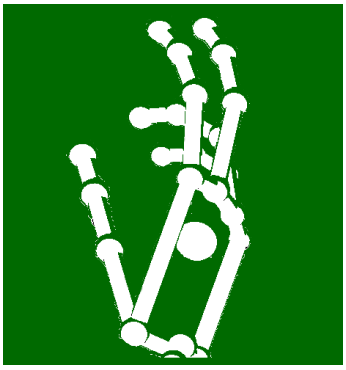
Gesture Recognition App
"Handy Gestures"
for Mobile Applications with Androids (700.380, 19S)

Michał Gilski, Cihan Yatbaz

Summer term 2019 at Alpen-Adria-Universität Klagenfurt



1 Motivation

1.1 Idea

Modern smartphones are becoming more and more powerful and with that come many new possibilities of how they can improve your life, by applications that could only run on more powerful devices like PCs. We decided to use that by creating a mobile image processing app that uses uses neural networks to classify real time images.

Next came the target group. We have decided to bring forth a social group that can uniquely benefit from this kind of technology, the Deaf Community. For years they have used hand shapes, positions, and movements to communicate and express ideas. But unfortunately not everybody can "hear" them. Applications like ours bring closer to the rest of the society. Our aim became to reduce the difficulties experienced by people with disabilities while communicating. Using defined hand gestures, our app can detect them, translate them into texts and then vocalize these texts. In the end, our goal is to provide theses people with a more comfortable form communication with non sign language speaking people, by using gestures that are natural to them.

1.2 Potential users

Even though the app is dedicated to the Deaf people, anyone can find it useful or just fun to play around with. It could be useful for both the Deaf person and the person trying to communicate with them. By providing a hand-less typing system that can be used without touching the screen, we hope it can be also used in different creative ways to make peoples life easier.

2 Similar applications

Looking at the Google Play Store, we found that there are not many applications related to hand gesture recognition like ours. Existing apps were usually made to teach what hand signs mean. In addition to this, there are also applications that make some functions on the phone with hand gestures. However, our app can perform both the meaning of hand gestures and some functions through hand gestures. Our application includes functions such as creating a text using hand gestures and vocalizing this text, and all of it possible with just hand gestures and not touching the screen (in auto detect mode). With these functions we provide, our application is much more useful and suitable for its purpose.

3 System Description

3.1 Key functionalities

Our app includes the following main functions:

- Detection of sign from a real time camera preview, which can be done in two ways:
  - Using the dedicated button to capture the sign
  - Using automatic detection that works without the user even touching the screen
- Detection of sign from a gallery images
- Writing to the screen by using specific sign language combinations
- Vocalizing the displayed text (can be activated using a sign)

Our app also contains some additional functions:

- Viewing the help guide that describes both the usage and hand gesture combinations
- Switching the cameras used (front or back) in preview mode
- Add special characters or run commands on the created text (space, backspace, clear and previously mentioned vocalize)

3.2 Used technologies, sources and settings

In order to make the app more easily we relied on some ready components and libraries. Also some changes in the project gradle were required or useful.

Used Libraries:

- Tensorflow for js (used for training the model)
- Tensoflow lite (used to deploy the model)
- OpenCV (used to handle camera stream)

Used components:

- MobileNet training code from Tensorlow examples
  `https://github.com/tensorflow/examples/tree/master/lite/examples/gesture_classification/web`

- Code to translate the model to a TF lite model
  https://github.com/tensorflow/examples/blob/master/lite/examples/gesture_classification/ml/tensorflowjs_to_tflite_colab_notebook.ipynb
- Code to run the model on android (ImageClassifier and ImageClassifierFloatInception)
  https://github.com/tensorflow/examples/blob/master/lite/examples/gesture_classification/android/app/src/main/java/org/tensorflow/lite/examples/gesture/ImageClassifier.java
  https://github.com/tensorflow/examples/blob/master/lite/examples/gesture_classification/android/app/src/main/java/org/tensorflow/lite/examples/gesture/ImageClassifierFloatInception.java
- Code to force an OpenCV stream into portait mode
  https://heartbeat.fritz.ai/working-with-the-opencv-camera-for-android-rotating-orienting-and-s

Changes to gradle:

- Disable the compression of the tflite model
- Enable APK splicing on OpenCV to reduce app size

3.3 Preparation of model

3.3.1 Classifier training

The training of the model uses a pretrained model called MobileNet and then uses the training script to fine-tune the model to the recorded gestures. We used 6 different classes of signs, each corresponding to a American sign language sign. We chose the ones that resemble each other the least and using the js camera script trained it with 100 samples per class. We made sure to include both hands (mirrored image) and different backgrounds with variant lighting conditions. For the training parameters we used a learning rate of 0.0001, batch size of 0.4 and 100 hidden units, running this for 20 epochs. After the training we achieved a log loss of 0.01 and while testing we concluded that the accuracy is about 85%, which seem pretty good results.

Figure 1 This is the JS script UI used for training (on webcam).

3.3.2 Model conversion

Model conversion was done in a Google Colaboratory environment and returned a converted model tflite that contain the network weights.

3.4 Processing Pipeline

3.4.1 Deploying the model

The model is loaded by ImageClassifer, which runs the model on a image and returns a set of probabilities. The speed of the operation is dependant on the phone, but reaches values of about 100ms per frame. Then the result is passed on to the Recognition class that handles the extracting information from a pair of signs and then translating them to a specified command accompanied with setting the images to a image view, vibrating to signify a completed prediction and using Text-to-speech software to vocalize the text.

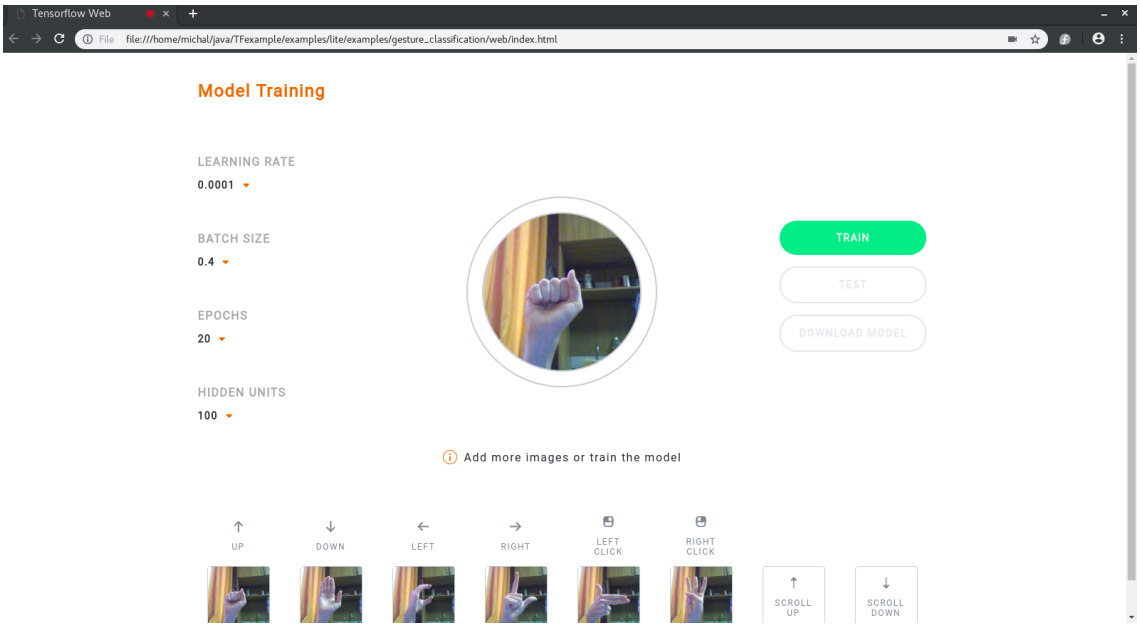


Figure 1: The training script

3.4.2 Gallery option

The source of the images can be the image gallery. Two images are loaded into image views and then are classified. The command is executed when the user presses the button and an extra speech button is available.

3.4.3 Preview option

Another way to run the recognition is to use the preview. There are two modes to the preview recognition.

Manual trigger
While using the manual button for detection the first frame is captured and classified immediately, while the second one after 2s to give the user time to change the sign. This delayed calling of a function is handled through a thread called by a handler after 2 s).

Auto Detection
The automatic detection is done ran in a separate thread, because it does not classify every frame (with classification taking 100 ms it would be impossible), but instead every 500 ms, thus requiring the thread to sleep for some time. When it is this mode is activated by a button it starts the process of detection.

Figure 2 See for the algorithm for automatic detection.

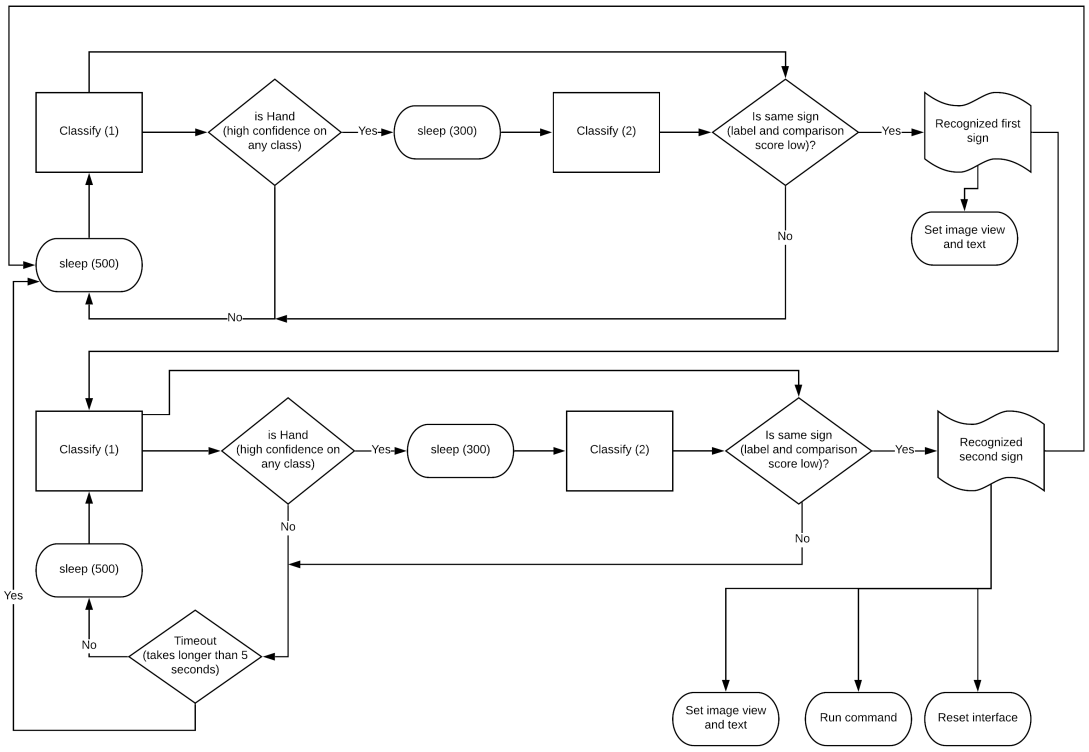


Figure 2: A representation of the auto detect process

3.5 Classes - activities

3.5.1 Menu

The menu activity handles the start menu with buttons to other activities and the quit option. This is the starting activity.

Figure 3 See the screenshot of the menu.

Components:

- Preview button - launches the preview activity
- Gallery button - launches the gallery activity
- Help button - launches the help activity
- Quit button - Exits the app

3.5.2 Gallery

This is the gallery activity that handles recognition of symbols based on loaded images from the gallery.

Figure 6 See the screenshot of the gallery.

Methods:

- onCreate - Gets links to all the required components and classes
- pickImage - Allows the user pick an image from the gallery using the onActivityResult override. Uses the button tag and request codes to determine where to load the image
- speakButton - Runs the speak method from the Recognition class
- classify - Gets the predictions for the images and runs the command through the Recognition class

3.5.3 Preview

This is the preview activity that handles recognition of symbols based on a camera stream.

Figure 4 See the screenshot of the preview with auto detect off.

Figure 5 See the screenshot of the preview with auto detect on.

Methods:

- BaseLoaderCallback - Loads OpenCV
- onCreate - Sets up the preview and links to other objects
- startAuto - Handles button to toggle automatic detection (renames the button to stop and hides the manual detection button)
- onCameraFrame - Returns the frame image
- setToBitmap - Sets the Mat image to a bitmap (correctly rotated)
- showFrame - Manual start of the detection (2s between signs)
- classifyFirst - Classify the first sign (use Recognition class) and set the first image view
- classifySecond - Classify the second sign, set the second image view and run the command
- swapCamera - Swaps the camera between front and back

3.5.4 Help

This activity describes the usage of the app and shows the hand sign combinations.

Figure 7 See the screenshot of the help.

3.6 Helper classes

3.6.1 Recognition

This class handles the recognition and execution of the commands based on a pair of images.

Methods:

- vibrate - Vibrates for a specific am mount of time
- putLetter - Adds a letter to the text view
- speak - Vocalizes the text in the text view
- getPredictionID - Maps letters to IDs
- makeCommandList - Maps two prediction IDs to char command
- getCommand - Returns the char command from prediction nIDs
- executeCommand - Executes the command based on the char command
- runCommand - Public start method of the command execution
- classifyFrame - Extracts a thumbnail from the bitmap and runs the classifier on it

3.6.2 Autodetection

This class is runs the automatic detection in a separate thread after being activated by a button.

Methods:

- AutoDetection - Gets references to objects from Preview
- begin - Starts the thread
- stop - Stops the thread
- oneSign - Handles detection of one sign, by running the detection twice (probing and verification) and comparing the results. It uses the detect method to find frames with potential signs and compareTo to verify that sign is stable. Also sets the interface objects of the detected frame.
- run - Runs the detection of the two signs and executes the commands (using Recognition), handles timeout for second sign
- resetInterface - Resets the interface objects
- sleep - Sleeps the thread for some time
- detect - Checks if the highest confidence is above a threshold (hand in view)
- compareLabels - Returns a (heuristic) comparison value of two sets of predictions, higher usually means less similar frames, use for verification of sign
- getBitmapFromMat - Returns a bitmap from Mat
- updateSortedLabels - Updates the sorted array of labels with probabilities (from the latest prediction of ImageClassifer) and splits them into labels and probabilities arrays
- classify - Classifes the image and updates the labels

4 Results

4.1 Overview

As mentioned before the training of the process was successful and yielded satisfying results with around 85 % accuracy and a log loss of 0.01. Testing on the actual phone could only be done subjectively as the camera is not a good tool for stable tests. The recognition it works very well, recognizing signs made by both hands.

The automatic detection is dependant on two thresholds, making it hard to force detection when a hand is in view and stop the detection from false positives. Nonetheless, the probing and verification method worked well, with rare false defections and a lot of good ones in different lighting situations.

4.2 Screenshots

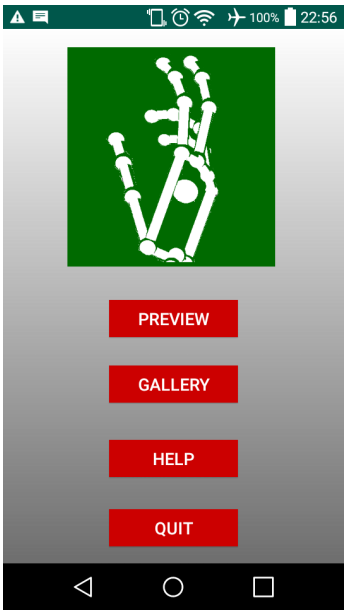


Figure 3: The menu screen

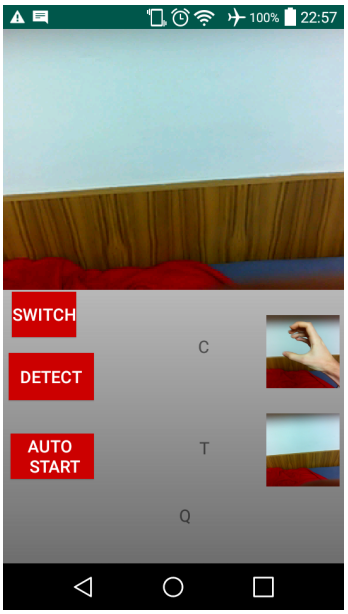


Figure 4: The preview screen (with auto detect off)

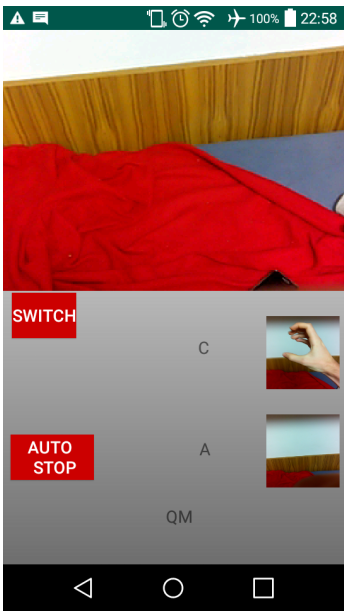


Figure 5: The preview screen (with auto detect on)

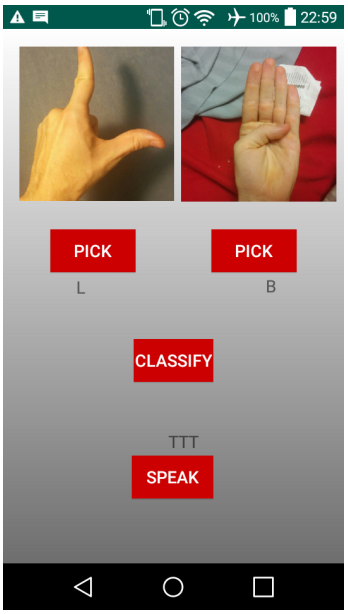


Figure 6: The gallery screen

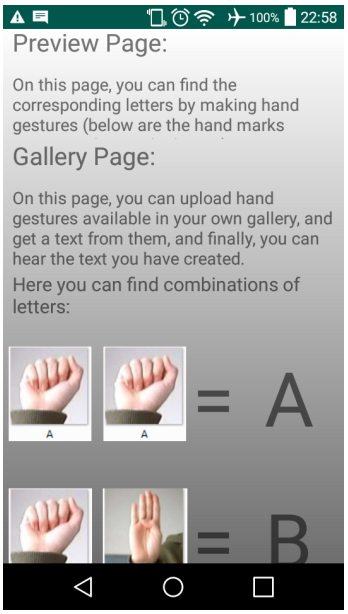


Figure 7: The help screen

5 Discussion

5.1 Observations

An interesting effect our training method had was side invariance. The switching of cameras from front to back meant that the hand was seen from the other side. Because of the CNN architecture, the most important factor seems to be the silhouette of the hand. This resulted in our model easily handling both sides of the hand, making it easily usable by the holder of the phone as well as another person.

Also the size variance worked pretty well, detecting the symbol in different scales. Due to the thumbnail cropping from the center, translation does not work that well, so the hand should be in center of the view.

5.2 Advantages

The unique part of this approach is surely the automatic detection, that allows the user to write to screen entirely without touching the screen. In case o mistakes the backspace and clear options are available through signs, as well as timeout that allows the user to reset the fist detected sign by waiting.

5.3 Limitations

Being based on just a RGB image, which comes from a phone camera, one cannot expect miraculous results. The classification was confused the most when the background has similar colors to a hand or if any other body parts were seen in the frame. Also the classifier was trained without access to a plethora of test subjects, which means that some skin colors may not work as well as others.

Another thing that may be a downside of the app is the required processing power. Even older phones can run the app (tested on some), but sometime the weaker phones may not deliver a perfectly smooth experience, but any new recent Android phone should not have any problems with the app.

5.4 The biggest challenges

The biggest issues were for sure handling the threads in the Preview class. In the auto detection mode, the code is ran in 3 threads (main UI thread, OpenCV preview and the auto detection thread) which was a challenge to synchronize. Also setting the OpenCV preview to fit in its frame in portrait mode was challenging.

5.5 Future possible improvements

There is a always room for improvement:

- Experiment more with the threshold values
- Train the model on more variant data (for example more skin colors)
- Add more interactive interface
- Integrate the help within the detection screens

- Adding more signs for the classifier to recognize, maybe whole alphabet (for usage with one sign per command)

6 Technical requirements

The requirements for the app are relatively low and are as follows:

- Android with minimum API 21 (Lollipop 5.0)
- Camera in phone (at least one, front or back)
- 25 MB of storage for app installation
- Reasonable phone speed (anything from the last 4 years should be fine)

7 User manual

First of all, you need to be sure about the Technical requirements.

Buttons that we have :

- PREVIEW : On this page you can find the letters that correspond to them by making live gestures (You can find combinations of letters on the Help Page). There are three buttons, SWITCH, DETECT and AUTO START/STOP. You can switch the camera to click switch. And you can learn the letter that corresponds to your hand gesture by pressing the show button. The last one turn on and off the auto detection when you only need to hold your hand in view until the phone vibrates and shows your sign. Then show the second sign and hold until the command is executed.
- GALLERY : On this page, you can upload hand gestures available in your own gallery, and get a text from them, and finally, you can hear the text you have created. There are four buttons. Two buttons are picking the image from the gallery and transferring to the screen. The classify button is classifying to the images which are you uploaded and matches them with the corresponding letters. The speak button is vocalizing the text which is you created by hand gesture images.
- HELP : Here is some information we think you might need. You can also find all combinations of letters and characters here.
- QUIT : You can use to quit the application.