Hand gesture recognizer

by

Dominik Bober, Szymon Duda, Adam Klekowski, Przemysław Ziaja

AGH University of Science and Technology

Cracow

2021

TABLE OF CONTENTS

Page

CHAPTER 1. OVERVIEW

1.1 Dataset

Dataset which was used to train the model is available at the following link:

https://www.kaggle.com/gti-upm/leapgestrecog

The dataset contains 10 directories of photos grouped by class,

Directories:

- Palm
- L
- Fist
- Fist_moved
- Thumb
- Index
- Ok
- Palm_moved
- C
- Down

Each directory contains 2000 photos.

1.2 Technology

Model was created with Python3. We used Tensorflow, Matplotlib and OpenCV library. The training stage was accelerated using nvidia gpu and software from nvidia like cuDNN.

1.3 Final application

User can obtain predictions using `hand_gesture_recognition/app.py`. There are two patterns which are accepted by the script:

- `python3 app.py path_to_photo`
- `python3 app.py path_to_directory_containing_photos`

To change these patterns it is necessary to modify `get()` method in `app.py`

CHAPTER 2. DATA PREPROCESSING

Data which we have used to train our model is very clear. There was no need for large preprocessing, however, we wanted to test if we can increase the performance of the model. We have selected a few algorithms for image processing. Photos depicture a hand of a person whose complexion is pale and on a black background. Naturally, we have chosen algorithms equalizing histograms of photographs so we can differentiate areas where a hand is located. `hand_gesture_recognition/dataset_preprocessing/Preprocessing.ipynb` notebook contains all results of the preprocessing stage.

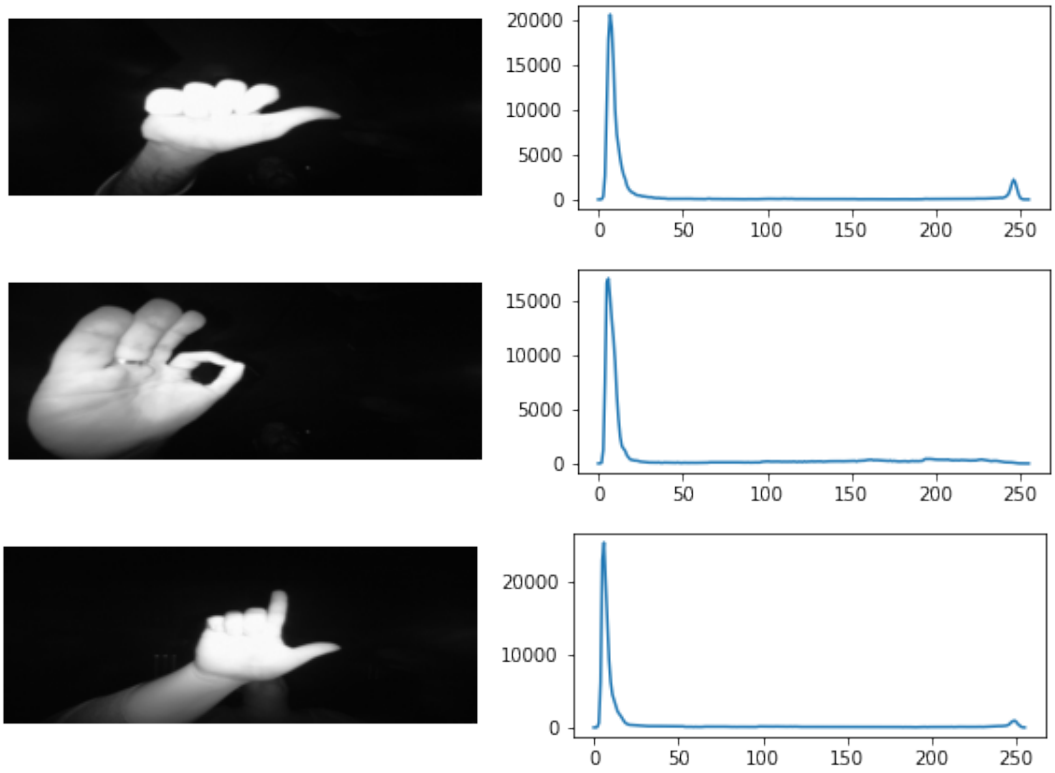

Figure 2.1 Examples of photos and graphs showing their histograms.

2.1 Histogram equalization

First we have tried simple histogram equalization using `cv2.equalizeHist()`. Method equalized histogram using pixels from the whole picture. Thanks to this method we can see the face of a person making a hand gesture, but the hand gesture is not sharp. The surrounding of the hand that mildly glows would be of no avail to the recognition of the hand's gesture[1].

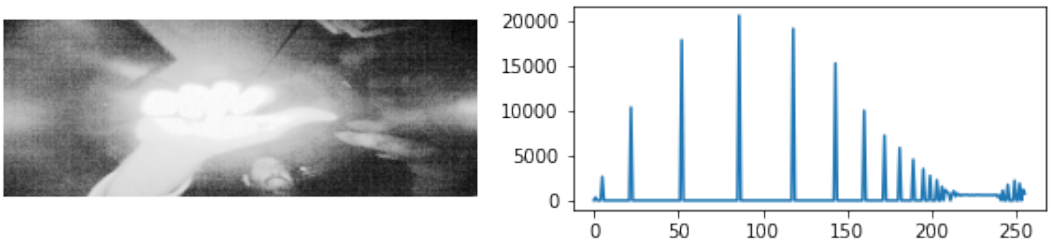

Figure 2.2 Effect of histogram equalization.

2.2 Clahe algorithm

Using pixels from a full photo is superfluous and cumbersome, so we decided to use the adaptive algorithm. Clahe algorithm uses pixels from the surrounding area of a certain pixel to equalize histogram of the photo. As we can see, histograms of photographs are much more continuous as well as boundaries between fingers are much more visible. At the same time, the glow around the hand, as well as the face, is nearly non-existent thus the hand is visible. It is an advantage because in the real world it is virtually impossible to get photography of a hand with a consistent background[2].

[1] Documentation of histogram equalization
[2] Clahe algorithm explenation

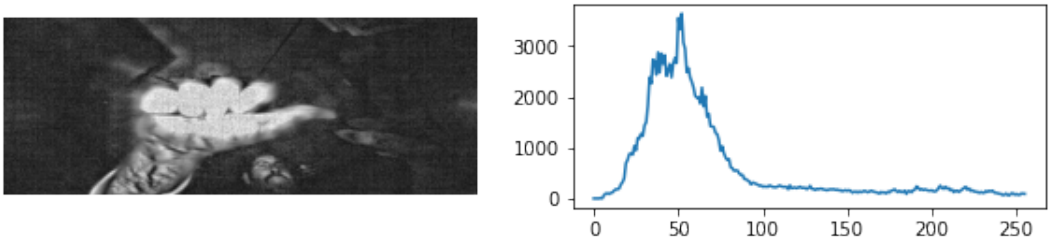

Figure 2.3 Effect of using the Clahe algorithm.

2.3 BBHE algorithm

Next, we have tried Bi-histogram equalization. It is a little more advanced algorithm than standard histogram equalization from the previous section and much faster than Clahe algorithm. How does it work? Pixels from photography are divided into two sets. The criterion is the brightness of the pixel. Next on the two sets is performed standard histogram equalization and pixels are merging back into photography. As we can see the hand is visible, borders between fingers are clear, however, it is because the photographs have black background[3].

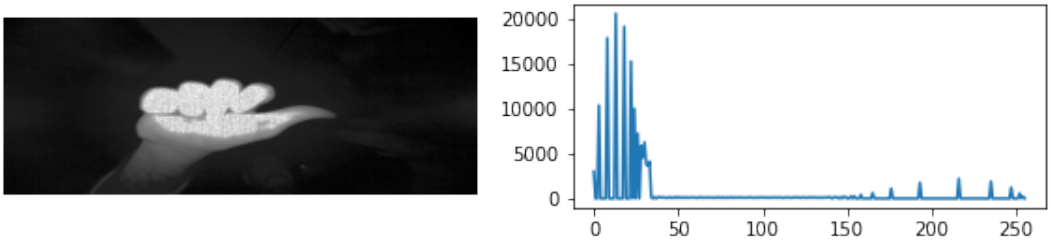

Figure 2.4 Effect of using the BBHE algorithm.

2.4 DSIHE algorithm

DISHE algorithm is a variation of BBHE algorithm. The main difference is the criterion of pixels division. We will not describe the algorithm only advantages and disadvantages. The output photographs use much more channels than output photographs from standard histogram

[3] BBHE explanation - site 15.

equalization, however, there is a bright glow around a hand and algorithm is more expensive than standard histogram equalization[4].

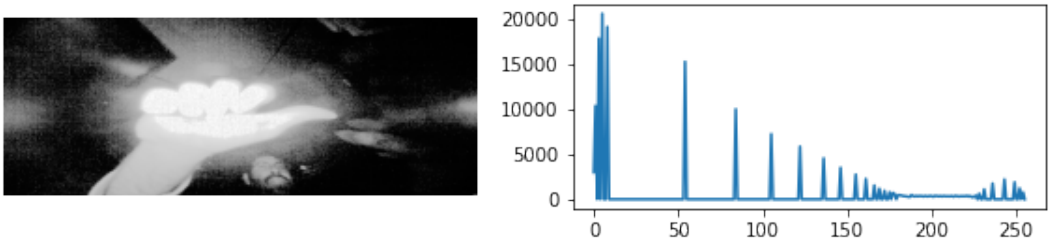

Figure 2.5 Effect of using the DSIHE algorithm.

2.5 Summary

After analyzing the pros and cons of individual algorithms and checking how they deal with preprocessing, we decided to use Clahe algorithm. The output photographs contain clear boundaries between fingers and background. It is computationally expensive but real-world data shows other algorithms will perform less efficiently. The preprocessing stage has shown that the dataset despite time invested in making photographs is weak. In the real world, nobody would take photography of hand in perfect conditions. The main goal of building a machine learning model is to perform well in the real world, not on a test set.

[4] DSIHE explanation - site 19.

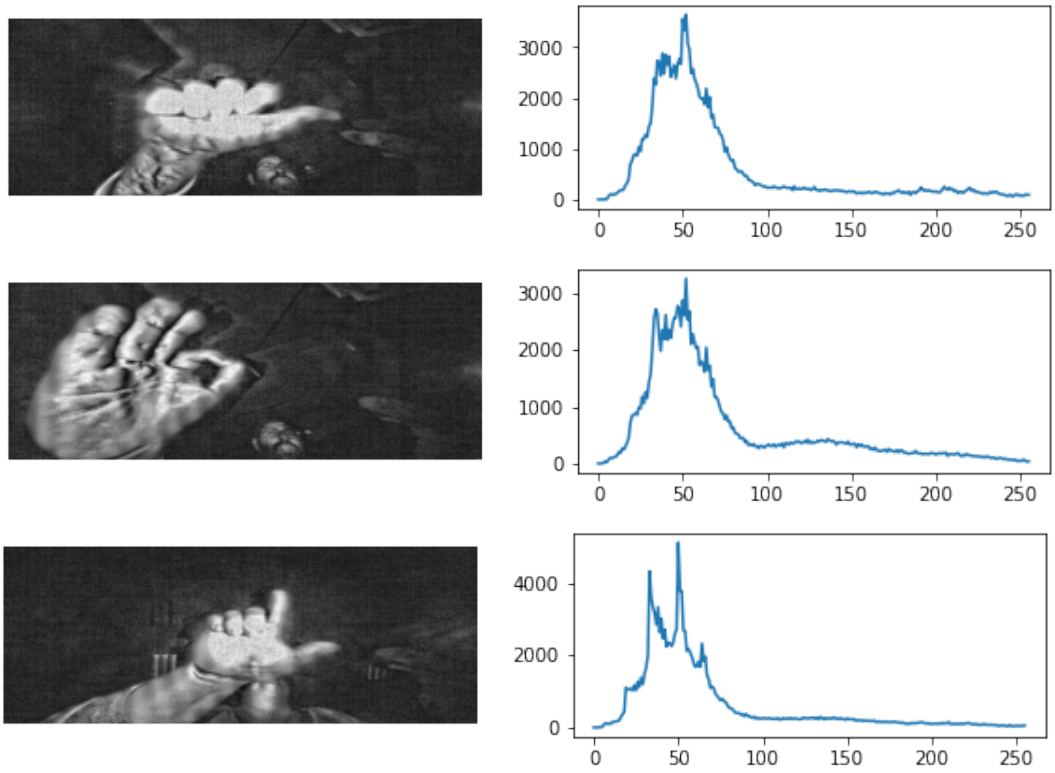

Figure 2.6 Effect of using the algorithm we chose.

CHAPTER 3. RAW DATA MODEL

First, we have created the model using data from the original dataset.

We decided to use a convolutional neural network so we have tried many combinations of convolutional layers.

3.1 1st iteration

First, we have set up GPU acceleration and we have written a function to get data from the hard drive. Setting up a GPU unit is an individual thing, but getting data was tricky. We could not read all data at once. In the real world case scenario, we will not be able to read all data. We made use of `tensorflow.data.Dataset.from_generator()` method and we have created a generator that read photographs from the directory. Photographs are divided into 10 series and each series is divided into classes.

Generator at the same time returns photo and label, the label is extracted using directory name. 1st model was trained to recognize 10 classes. The performance was bad. We have obtained less than 40% accuracy. In every model, we made use of Adam optimizer.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 240, 640, 64)      640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 120, 160, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 120, 160, 128)     73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 60, 80, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 80, 256)       295168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 40, 256)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 30, 40, 512)       1180160
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 15, 20, 512)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 20, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 10, 512)        0
_________________________________________________________________
flatten (Flatten)            (None, 35840)             0
_________________________________________________________________
dense (Dense)                (None, 64)                2293824
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 6,204,106
Trainable params: 6,204,106
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.1 model.summary()

3.2 2nd iteration

We came up with an idea that creating good architecture by us might be difficult so we decided to use VGG-16 architecture. Figure 3.1 presents the original VGG-16 architecture.

There are pre-trained models on the Internet but VGG-16 has square input shape and we needed a rectangular one. Below we included our implementation of the algorithm Despite fact that we decreased the number of convolutional layers and number of learning parameters was 20 times smaller accuracy jump to 75%.

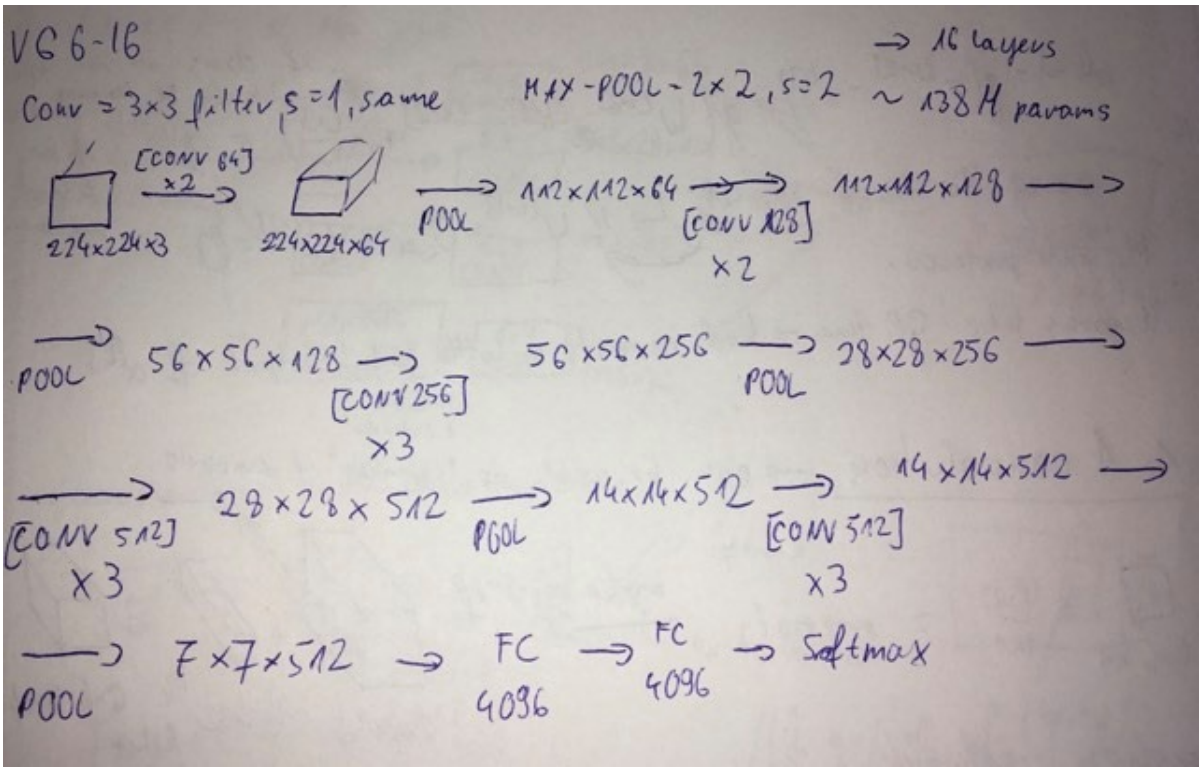

Figure 3.2 VGG-16 architecture

It is worth to notice that we have no information about the learning process. After learning with the fit method it is possible to get information about parameters like loss, accuracy in each epoch, however, our pipeline does not support setting up epoch number in `fit()` method. Number of epochs is set up in method `tf.data.Dataset.from_generator()`. To prevent overfitting we stop learning in the random moment and evaluate it to obtain maximum accuracy at the test set.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 240, 640, 64)      640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 120, 320, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 120, 320, 128)     73856
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 60, 160, 128)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 60, 160, 256)      295168
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 80, 256)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 30, 80, 512)       1180160
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 15, 40, 512)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 40, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 20, 512)        0
_________________________________________________________________
flatten (Flatten)            (None, 71680)             0
_________________________________________________________________
dense (Dense)                (None, 64)                4587584
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 520
=================================================================
Total params: 8,497,736
Trainable params: 8,497,736
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.3 model.summary()

CHAPTER 4. CLAHE MODEL

4.1 Assumptions and preparation

In the beginning, we assumed that the instance of class fist and fist_moved is the same class. there is no point to separate moved hand gestures from not moving.

To speed up computations we prepared script that applies Clahe algorithm on photographs, moves moved hand gestures to not move class and saves it on a hard drive. The script is stored in `hand_gesture_recognition/archive/leapgestrecog/to_clahe.py`.

```
def image_getter(base_dir='./leapGestRecog'):
    for part in os.listdir(base_dir):
        for img_class in os.listdir(os.path.join(base_dir,part)):
            if '_moved' in img_class:
                if 'fist' in img_class:
                    os.system(f'mv {os.path.join(base_dir,part,img_class)}/*
                    {os.path.join(base_dir,part,"03_fist")}')
                else:
                    os.system(f'mv {os.path.join(base_dir,part,img_class)}/*
                    {os.path.join(base_dir,part,"01_palm")}')

                os.system(f'rm -rf {os.path.join(base_dir,part,img_class)}')

    for part in os.listdir(base_dir):
        for img_class in os.listdir(os.path.join(base_dir,part)):
            for i in os.listdir(os.path.join(base_dir,part,img_class)):
                image = cv2.imread(os.path.join(base_dir,part,img_class,i),cv2.IMREAD_GRAYSCALE)
                clahe = cv2.createCLAHE(16,(24,64))
                new_img = clahe.apply(image)
                cv2.imwrite(os.path.join(base_dir,part,img_class,i),new_img)
```

Figure 4.1 to_clahe.py

4.2 Model

The best change comparing to the raw data model is input data. The structure of directories is the same as in the original dataset, however, the number of categories was decreased by 2. In the previous model, the number of classes was encoded in the directory name. We have made a simple workaround. Depending on directory name data generator scales class number so classes are encoded from 0 to 7.

To prevent overfitting we considered the use of method `tf.Keras.preprocessing.image.ImageDataGenerator()`. The method takes a picture and can transform it by applying rotation, zoom etc. The transformation retains the original size of the picture. During training, it turned out that regularization L2 on the dense layer and breaking training is enough.

After the application of changes, the accuracy of our model was increased to 90%. Below we present a summary of our most efficient model.

```
1125/1125 [==============================]
- 270s 240ms/step
- loss: 1.7459 - sparse_categorical_accuracy: 0.8902
- val_loss: 1.6874
- val_sparse_categorical_accuracy: 0.9085

125/125 [==============================]
- 9s 73ms/step
- loss: 1.6874
- sparse_categorical_accuracy: 0.9085
```

CHAPTER 5. SUMMARY

As we describe the model performs very wel. Model have similar accuracy on the training set and test set so there is no overfitting.

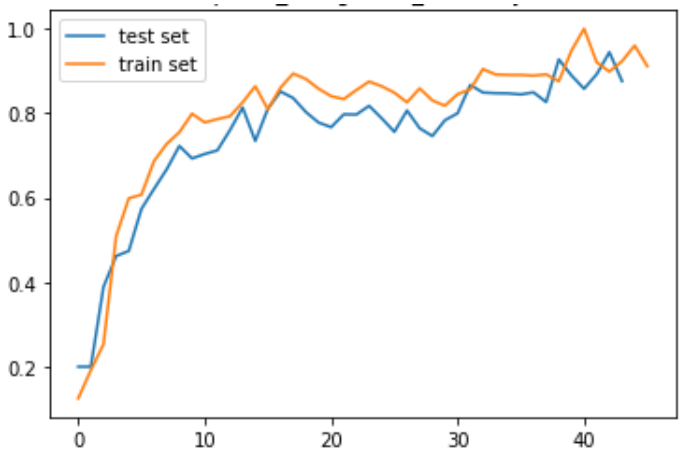

Figure 5.1 Value of accuracy in function of training time

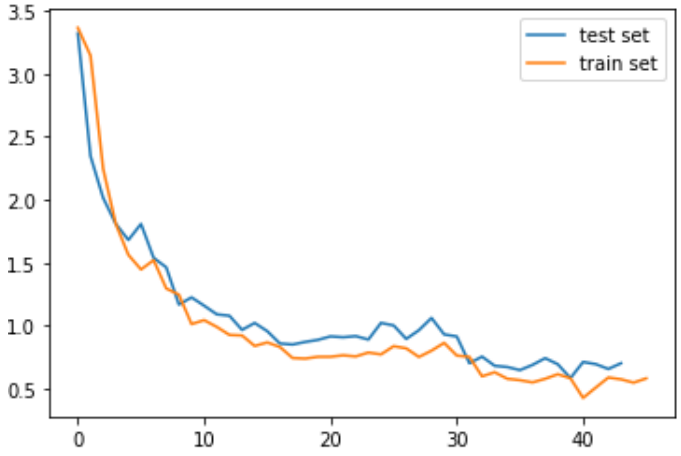

Figure 5.2 Value of cost in function of training time

Thanks to GPU speed up we could create and train many models and choose the best one.

We have to mention that model work well on data that we got from Kaggle, however, in real life conditions, we are disappointed about performance of model.

Photos were taken in laboratory conditions and to improve we should take photos using a webcam. The main disadvantage of the model is that we do not know to what stimuli the model is sensitive. One possible method to solve the problem is to blur random parts of an image and see how does it affect the prediction. A better solution is to rebuild the model and use YOLO algorithm so we could obtain category and frame around the hand.

As we mentioned, photos were taken in the laboratory. We have taken pictures by ourself in a dormitory room. The results are listed below.

Figure 5.3 Pictures used in real world test

Photos we have taken have other proportions than photos in the training set and the hand obscures more of the picture. All photos were predicted as **ok** sign what is worrying. It means that model has overfitted training data under unknown circumstances.

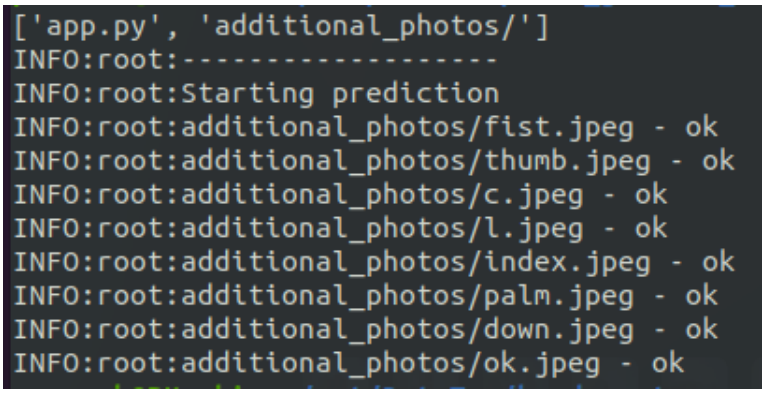

Figure 5.4 Results of real world test

REFERENCES

[1] Histogram Equalization

[2] Histograms - 2: Histogram Equalization

[3] HISTOGRAM EQUALIZATION BASED METHODS FOR BRIGHTNESS PRESERVATION AND LOCAL CONTENT EMPHASIS