Hand Gesture Recognition

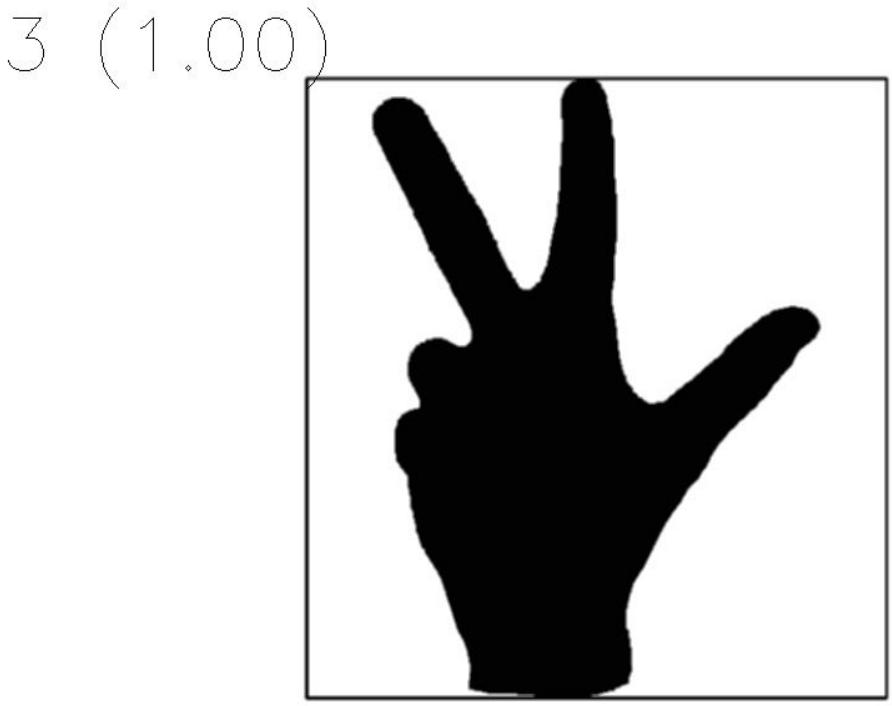
Group 9

Timothee Bayart, ID: 252959 (t.bayart@stud.uis.no)

Teo Bouvard, ID: 252898 (t.bouvard@stud.uis.no)

Emile Hertoghe, ID: 252910 (et.hertoghe@stud.uis.no)

Vincent Nicolas, ID: 252904 (v.nicolas@stud.uis.no)



Introduction

In this project, we train a simple image classifier which predicts how many fingers are raised given the image of a hand. This classifier is able to predict the class for each frame of video in real-time. To complete this task, we must first collect a dataset, implement a preprocessing pipeline and train our model.

For this project we worked in a group. Some simple tasks were performed in pairs while the more complex tasks were performed together.

- Vincent and Téo worked on data extraction to create and class all the images.
- Timothée and Emile worked on the real-time prediction from the webcam, using our trained model.
- For the image preprocessing, we all worked on it to find the different steps and functions to use, and to tune the different parameters to produce the best output image.
- For the creation and training of the model, it's Téo who mostly contributed to it, although we all worked on this step, to find the right model to apply.

Data extraction

Our first task is to capture data from a webcam in order to create a labeled dataset which we could use to train a model.
We start by making six different videos, one for each class, where we show the corresponding number of fingers. For example the first video will be a video with zero fingers held up, the second video with one finger held up, etc. We then loop over these 6 videos, preprocess every frame and store them in the folder associated with the class name. We choose to do this instead of taking hundreds of pictures manually as this saves up a considerable amount of time. By capturing a 60 second video for each class, we end up with approximately 9000 labeled images in our dataset.
This dataset is organized so that the data for each class is contained in a different directory, as requested.

Image preprocessing

This is the most important part. Our main goal in this step is to process the images to reduce as much as possible the complexity of the problem. The starting images are big, in colours, do not only contain the hand and have a lot of noise. At the end of the process we want to have an

image that only contains the hand in white over a black background, resized to 50*50 pixels with the least possible noise.
Our image preprocessing pipeline comprises 9 consecutive steps which are described below.

1. We read the original image as a pixel matrix in RGB.
2. We convert the image from RGB to HSV color space. This greatly simplifies image segmentation because we separate color information from intensity and lightning. If we were to segment the image in RGB color space, we would have to identify the narrow range of colors of a hand, and the segmentation would fail for slight variations around this color range. In HSV color space however, we have much more flexibility concerning the threshold values as we mostly focus on filtering Saturation (S) and Value (V) channels.
3. We use lower and upper threshold values to binarize the image. All pixels in the defined range are replaced by white pixels, while the rest of the image is converted to black. This segmentation phase creates two sets of points : hand pixels and non-hand pixels. Both of these sets are not perfect, so we use the following two operations to get a better segmentation.
4. We apply a morphological opening to get rid of noise in the image. The opening operation is repeated three times to discard most of the pixels wrongly identified as hand pixels during the segmentation phase.
5. We apply a morphological closing to fill holes in the hand. The closing operation is repeated five times to try to fill hand pixels wrongly identified as non-hand pixels in the segmentation phase.
6. Once our image is properly segmented we extract contours in the image, only considering contours which generates a greater area than a threshold value. This threshold value is big enough so that the only remaining contour is the hand. Once identified, we compute the square bounding box of the hand.
7. We crop the image around the previously computed square bounding box. All pixels outside of the bounding box are deleted.
8. We resize our image to 50*50 pixels.
9. We apply a Gaussian filter to the resized image to smooth the irregularities introduced by downsizing the image.

A sample image at each step of this pipeline can be found in Appendix I.

Model training and evaluation

This part is a common machine learning step. First of all, we populate an array of features with the preprocessed images and an array of targets with the corresponding class labels. During development, we split this dataset in training and testing splits, holding out 20% of our data. Once the model achieves a correct performance on the testing set, we train our final model on the whole dataset.

We use Keras's sequential neural network as our classifier. The different layers are described below.

1. The input layer only reshapes the data, flattening our 50*50 images to a one dimensional feature array of size 2500.
2. The first hidden layer is a dense layer of 128 neurons, with a RELU (REctified Linear Unit) activation function.
3. The second hidden layer is a dense layer of 64 neurons, also with a RELU activation function.
4. The output layer is a dense layer with 6 neurons, each corresponding to a class, with a softmax activation function. This allows us to have the normalized probabilities of each class on the output layer.

The model is trained for 3 epochs using sparse categorical cross entropy as the loss function.

Applying our model

Once we achieve a reasonable performance on our test set, we train the model on the whole dataset and save it to a file. We can now use it to do live predictions, using a process which is quite similar to the data extraction part.
We load the model, open a video stream and preprocess each frame. In the data extraction part, we were saving this preprocessed frame to a file in the corresponding class folder. In this part, we use the preprocessed frame as input to the loaded model. This allows us to get the probability of each class for the given frame. We display the most probable class on the image in real time along with its corresponding probability, which can be thought of as the confidence of the model.

Results and analysis

With this setup, we manage to achieve 100% accuracy on our test set in only two or three epochs. This is mostly due to an intensive data preprocessing phase and strong constraints on the initial images.

```
Train on 6960 samples
Epoch 1/3
6960/6960 [==============================] - 1s 84us/sample - loss: 4.2886 - accuracy: 0.9664
Epoch 2/3
6960/6960 [==============================] - 0s 44us/sample - loss: 6.8511e-11 - accuracy: 1.0000
Epoch 3/3
6960/6960 [==============================] - 0s 43us/sample - loss: 5.1383e-11 - accuracy: 1.0000
1741/1 - 0s - loss: 6.1225e-06 - accuracy: 1.0000
Accuracy on 1741 test samples: 1.0
```

Here, we chose to gather our data from a controlled environment, with a uniform background and only the hand in the images. In a real world setup, we would have to acquire more data because we wouldn't be able to use the same assumptions during preprocessing. As an example, we assume the hand is the only segmented object, which is not true as soon as we have the user face in the frame. The way the user raises his fingers may also influence the model's predictions. To build a more robust model, we would need a deeper model as well as a bigger and more diverse dataset.

Conclusion

We implemented an efficient yet narrow scale real-time image classifier using a deep neural network trained on heavily preprocessed images. The main difficulty consisted in producing an adequate preprocessing pipeline by tuning the segmentation parameters to isolate hands in the images correctly.

Appendix I

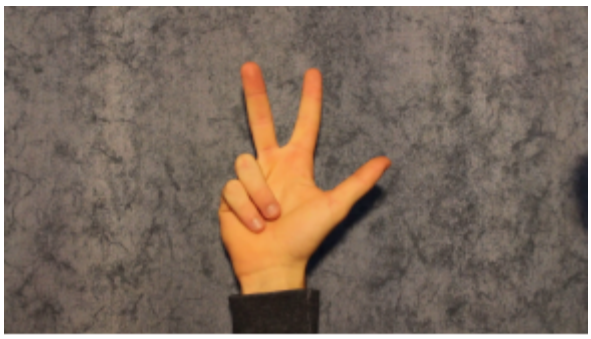


Figure 1 : Original image

Figure 2 : HSV image

Figure 3 : Segmented image

Figure 4 : Segmented image + opening

Figure 5 : Segmented image + closing

Figure 6 : Bounding box identification

Figure 7 : Cropped image

Figure 8 : Resized image

Figure 9 : Filtered image