Springboard Capstone Project 2

Analyze Hand Gestures - Final Report

Bogdan Sburlea

May 3, 2020

Table of Contents

Executive Summary

The objective of this capstone project is to create a machine learning model that authenticates a real user in front of a camera. The model will be integrated into a real project that first authenticates that a real person is in front of a camera and then then captures the "best" image of that person from a video stream. The image can be used for various purposes like badges, IDs, user profiles for different websites, etc.

Introduction

The process of authenticating has the following steps:
- The user runs a software application that includes the model described below
- The application asks the user in front of a camera to raise a number of fingers (1 to 5)
- If this number is the same with the one detected by the model, the user is authenticated
- Otherwise, this sequence is repeated three times before having a failed authentication

Data Acquisition and Data Cleaning

One of the hurdles encountered was the time-consuming process of gathering good quality photos that are labeled. There are websites that can provide this but those photos are not a very good fit for the specific needs of the project. Therefore I decided to record a .mov file of myself with gestures within five classes (raising 1 to 5 fingers). I am using different positions and different distances from the camera, for several minutes for each class. I extracted the frames for the .mov files and converted these into jpg images. In order to increase the variance, I added tens of Internet photos for each class.

Data cleaning is being done by eliminating the frames that are temporally positioned between the two successive signs. Furthermore, I am deleting the images that are cut by one of the edges of the screen and the ones that are hard to understand due to various reasons.

After moving train and test data to the corresponding classes, there are:
- 12460 training jpgs (6 directories, total=12767), 5331 test jpgs (6 directories, total = 5338)
- 1.6 GB used for training jpgs
- 708 MB used for test jpgs

I made several tests with different formats like RGB, HSV, HSL and greyscale. The best results were obtained for RGB format. Greyscale format can be processed faster because there aren't 3 channels as for RBG, but it loses some important information that is associated with color.

The images were all scaled to 255x255x3 numpy arrays to simplify data processing. The data was then split in /train and /test with about 3000 samples in each of the 1…5 /train images.

Data Processing

Processing the data was done with Tensorflow 2 Convolutional Neural Networks (CNN). Since the training is very processor-intensive, I used a modern Nvidia GPU running on Ubuntu 18.04 LTS.

For increasing the variance even further, I applied the following transformations on the ImageDataGenerator for the images (for all train and test classes):

- image rotation by no more than +-20%. It is unlikely to have users that make hand gestures with a hand angle that is outside this interval. I reached this conclusion after visualizing many internet images that represent hand gestures.

- no horizontal or vertical shift because the gesture is close to one edge in a few images

- converted the 0…255 values from jpeg to the float interval [0., 1.] This was needed because TensorFlow works best with images that have the pixel values in this range

- returned the image itself and the one-hot encoded class based on the class directory

I trained and tested many CNN with different parameters. All the models were good for the / train images that were extracted from the same .mov file (very similar images with the corresponding /train categories) as shown below:

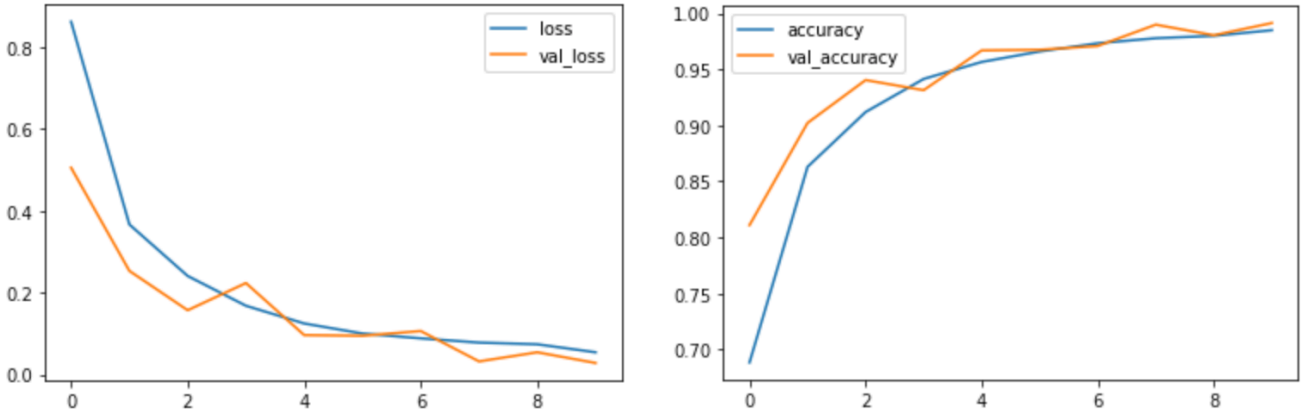
An early model is shown below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 255, 255, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 85, 85, 16)        0
_________________________________________________________________
flatten (Flatten)            (None, 115600)            0
_________________________________________________________________
dense (Dense)                (None, 255)               29478255
_________________________________________________________________
activation (Activation)      (None, 255)               0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 1280
=================================================================
Total params: 29,479,983
Trainable params: 29,479,983
Non-trainable params: 0
_________________________________________________________________
```

```
Epoch 1/10
1005/1005 [==============================] - 383s 381ms/step - loss: 0.8625 - accuracy: 0.6885 - val_loss: 0.5052 -
val_accuracy: 0.8110
Epoch 2/10
1005/1005 [==============================] - 381s 379ms/step - loss: 0.3666 - accuracy: 0.8630 - val_loss: 0.2530 -
val_accuracy: 0.9022
Epoch 3/10
1005/1005 [==============================] - 382s 381ms/step - loss: 0.2407 - accuracy: 0.9119 - val_loss: 0.1568 -
val_accuracy: 0.9404
Epoch 4/10
1005/1005 [==============================] - 382s 380ms/step - loss: 0.1678 - accuracy: 0.9414 - val_loss: 0.2233 -
val_accuracy: 0.9314
Epoch 5/10
1005/1005 [==============================] - 380s 378ms/step - loss: 0.1245 - accuracy: 0.9566 - val_loss: 0.0961 -
val_accuracy: 0.9669
Epoch 6/10
1005/1005 [==============================] - 381s 379ms/step - loss: 0.0999 - accuracy: 0.9659 - val_loss: 0.0944 -
val_accuracy: 0.9674
Epoch 7/10
1005/1005 [==============================] - 381s 379ms/step - loss: 0.0881 - accuracy: 0.9732 - val_loss: 0.1060 -
val_accuracy: 0.9708
Epoch 8/10
1005/1005 [==============================] - 379s 377ms/step - loss: 0.0777 - accuracy: 0.9777 - val_loss: 0.0317 -
val_accuracy: 0.9898
Epoch 9/10
1005/1005 [==============================] - 381s 379ms/step - loss: 0.0735 - accuracy: 0.9798 - val_loss: 0.0540 -
val_accuracy: 0.9804
```

This simple model has only one convolutional + pooling layer (the hidden layer). For the existing /test classes, the model performed well:

The accuracy for this model was around 98%:



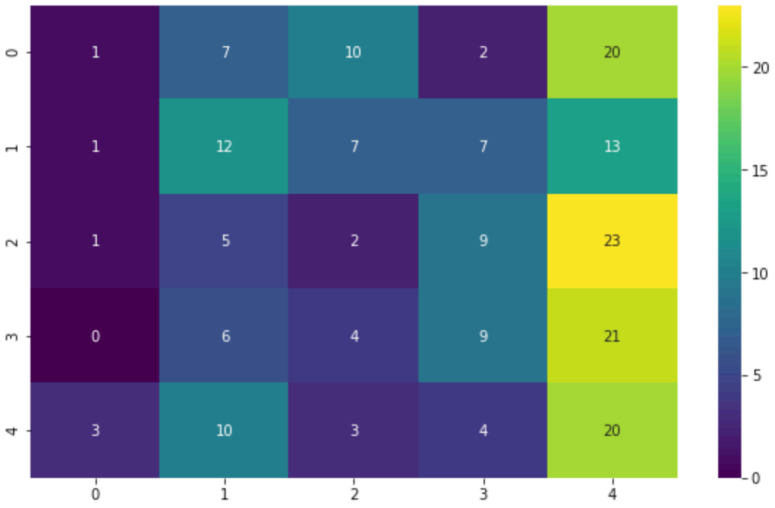
Since the metric was validation loss, it is shown above how the val_loss is not decreasing any more after epoch 7.

However, but the real test was with Internet images. The main problem was that I couldn't acquire a high number of internet images for each class even for /test. A few examples of Internet images are shown below:

There are several issues problems with the Internet images: The background is diverse, the fingers can overlap with the body, the zoom is different, etc. Testing the model accuracy on these images shows disappointing values:

	precision	recall	f1-score	support
0	0.17	0.03	0.04	40
1	0.30	0.30	0.30	40
2	0.08	0.05	0.06	40
3	0.29	0.23	0.25	40
4	0.21	0.50	0.29	40
accuracy			0.22	200
macro avg	0.21	0.22	0.19	200
weighted avg	0.21	0.22	0.19	200

I tested for 200 images and since there are 5 classes, the precision = 0.22 of the model is only marginally superior to a random guess. The heat map shows an interesting aspect:



It looks like the model is highly biased towards class 4 (5 fingers). The model is too simple, with only one convolutional layer a low number of filters (16 filters). The next models were more and more complex in order to catch the desired features.

Model number 7 is much improved and has a better accuracy for Internet images:

Layer (type)	Output Shape	Param #
conv2d_24 (Conv2D)	(None, 255, 255, 32)	2432
max_pooling2d_24 (MaxPooling	(None, 127, 127, 32)	0
conv2d_25 (Conv2D)	(None, 127, 127, 64)	32832
max_pooling2d_25 (MaxPooling	(None, 63, 63, 64)	0
conv2d_26 (Conv2D)	(None, 63, 63, 64)	36928
max_pooling2d_26 (MaxPooling	(None, 31, 31, 64)	0
flatten_10 (Flatten)	(None, 61504)	0
dense_20 (Dense)	(None, 128)	7872640
activation_10 (Activation)	(None, 128)	0
dense_21 (Dense)	(None, 5)	645

Total params: 7,945,477
Trainable params: 7,945,477
Non-trainable params: 0

There are 3 CNN layers with 32, 64, 64 filters. The kernel size was (5, 5) for the first hidden layer, (4, 4) for the second hidden layer and (3, 3) for the third hidden layer. The pooling layers and the dense layer are similar to the previous ones.

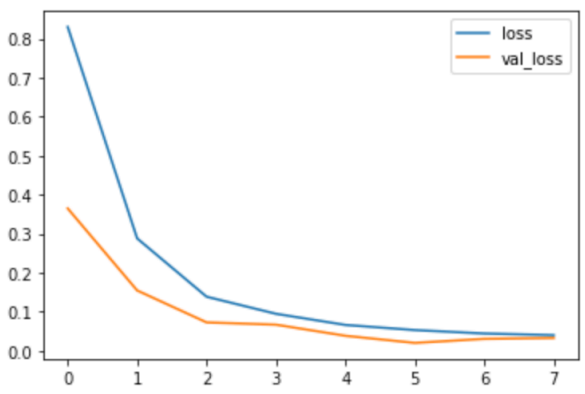
The training part went smoothly:

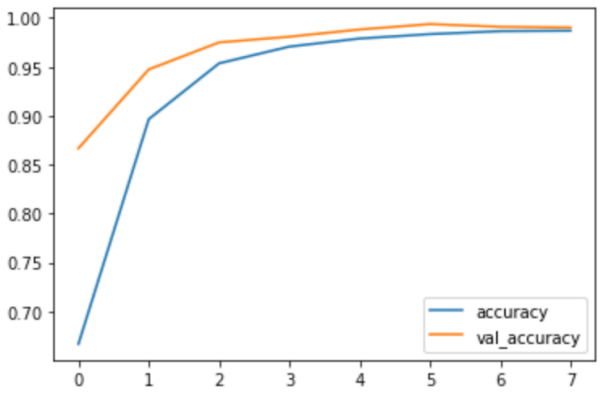
```
Epoch 1/10
1014/1014 [==============================] - 390s 385ms/step - loss: 0.8298 - accuracy: 0.6666 - val_loss: 0.3647 -
val_accuracy: 0.8667
Epoch 2/10
1014/1014 [==============================] - 386s 381ms/step - loss: 0.2880 - accuracy: 0.8965 - val_loss: 0.1546 -
val_accuracy: 0.9474
Epoch 3/10
1014/1014 [==============================] - 387s 381ms/step - loss: 0.1386 - accuracy: 0.9536 - val_loss: 0.0727 -
val_accuracy: 0.9750
Epoch 4/10
1014/1014 [==============================] - 387s 382ms/step - loss: 0.0948 - accuracy: 0.9707 - val_loss: 0.0669 -
val_accuracy: 0.9808
Epoch 5/10
1014/1014 [==============================] - 386s 381ms/step - loss: 0.0662 - accuracy: 0.9790 - val_loss: 0.0384 -
val_accuracy: 0.9882
Epoch 6/10
1014/1014 [==============================] - 386s 381ms/step - loss: 0.0531 - accuracy: 0.9835 - val_loss: 0.0202 -
val_accuracy: 0.9937
Epoch 7/10
1014/1014 [==============================] - 387s 382ms/step - loss: 0.0443 - accuracy: 0.9865 - val_loss: 0.0306 -
val_accuracy: 0.9910
Epoch 8/10
1014/1014 [==============================] - 389s 384ms/step - loss: 0.0401 - accuracy: 0.9870 - val_loss: 0.0328 -
val_accuracy: 0.9903
```

The accuracy is now even higher than before, reaching 99%:

	loss	accuracy	val_loss	val_accuracy
0	0.829805	0.666646	0.364718	0.866715
1	0.287973	0.896505	0.154596	0.947384
2	0.138564	0.953584	0.072693	0.975000
3	0.094769	0.970721	0.066916	0.980814
4	0.066226	0.978980	0.038425	0.988227
5	0.053073	0.983480	0.020180	0.993750
6	0.044289	0.986501	0.030577	0.990988
7	0.040062	0.986994	0.032797	0.990262

The loss function and the accuracy converge faster to the final values:

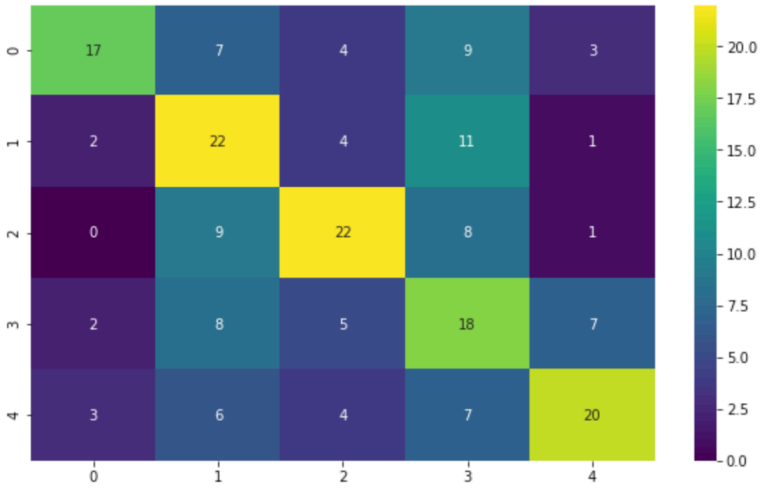




For the Internet images, the results were the following:

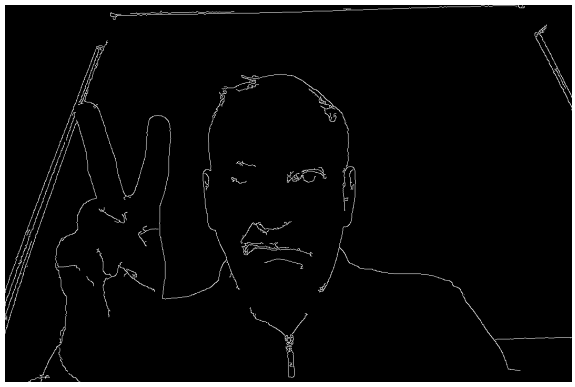
	precision	recall	f1-score	support
0	0.71	0.42	0.53	40
1	0.42	0.55	0.48	40
2	0.56	0.55	0.56	40
3	0.34	0.45	0.39	40
4	0.62	0.50	0.56	40
accuracy			0.49	200
macro avg	0.53	0.49	0.50	200
weighted avg	0.53	0.49	0.50	200

The accuracy is now 49% and all other metrics (precision, recall, F1 score) are improved.



In the heat map, the values on the first diagonal are the highest for each line or column. This means that given a random Internet image, the highest probability for class prediction is for the correct class. There is still a slight bias towards classes 1 and 3 (columns 1 and 3) but this can be handled by adding more Internet images for /train and /test in order to increase the variance.

Another idea was to use cv2.Canny() function especially for eliminating the variance introduced by different backgrounds. After applying cv2.Canny() only the edges are retained:

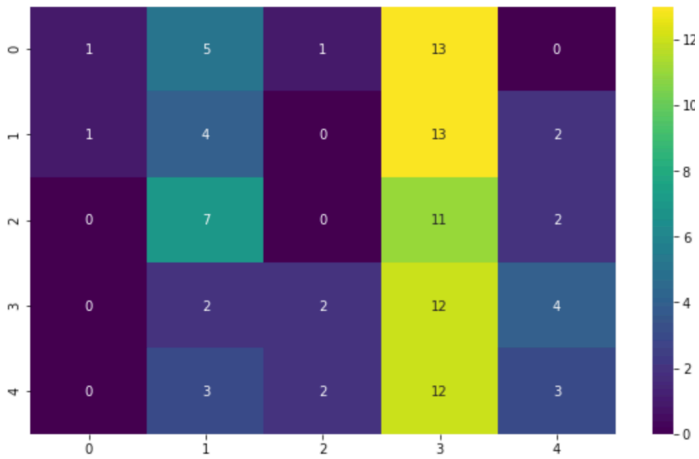


Unfortunately, using Canny() for preprocessing the images before feeding the neural network is tricky. It works well for the .mov file because the image characteristics are relatively constant (contrast, hue, illumination, etc.). The Canny() parameters can be tuned for getting good edge images from the video file. However, for the Internet images each image must have different Canny() parameters because the image characteristics are extremely varied.

The model has a similar behavior for the training part and validation accuracy was 98%. But for the Internet images, the results were the following:

	precision	recall	f1-score	support
0	0.50	0.05	0.09	20
1	0.19	0.20	0.20	20
2	0.00	0.00	0.00	20
3	0.20	0.60	0.30	20
4	0.27	0.15	0.19	20
accuracy			0.20	100
macro avg	0.23	0.20	0.16	100
weighted avg	0.23	0.20	0.16	100

The value of 20% for accuracy is identical to the random choice accuracy.



The bias towards class 3 is extreme. The best model remains model 7 (the previous one).

To improve the accuracy even more there are several ideas:
- Add more Internet images for /train and /test, at least 1000 images for each class.
- Test more CNN structures, maybe with more hidden layers for allowing a better fit for this complex type of data
- Extract only the hand and fingers from each photo, make it of the same size (eventually equalize the aspect ratio as well)
- Apply cv2.Canny() on all images as a preprocessing step. But due of the usage of Canny() on Internet images that was discussed before, there must be an extra step that automatically tunes the Canny() parameters depending on the image characteristics

All these ideas will require a significant effort and a lot of time.

Conclusions and Next Steps

This project showed that it is possible to create a model that can recognize hand gestures for good quality photos of persons. The ideas for improvement showed above would increase the model accuracy to at least 90% that would allow using the model in production.

The model can be deployed using Flask, Docker, Kubernetes or other solutions but this is outside the scope of this capstone project.

Acknowledgements

I would like to express my thanks to thanks to Springboard and especially my mentor for the continuous support and advice for the whole capstone 2 project.