Springboard Capstone Project 2

Analyze Hand Gestures - Final Report

Bogdan Sburlea

May 3, 2020

The objective of this project is to create a machine learning model that authenticates a real user in front of a camera. The user is required by a software application to raise a certain number of fingers and the model verifies that the required gesture was accomplished or not. This is just a part of a larger real-life project that aims to automate the process of taking good quality pictures for different purposes. The second part of the project (not covered by this capstone 2 project) will be to select the "best" picture after the user is authenticated.

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 Acquisition and Data Cleaning

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

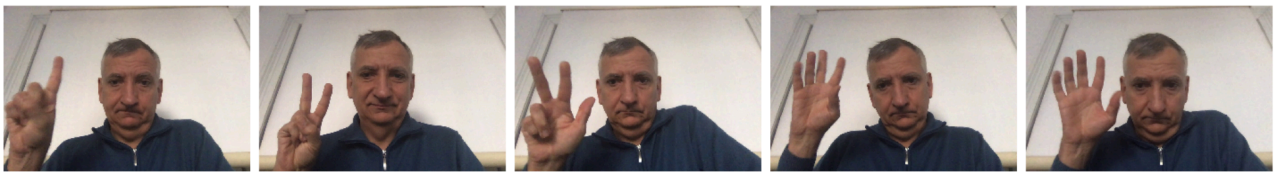
The images were all converted to 255x255x3 numpy arrays. The data was then split in /train and /test with about 3000 samples in each of the 1…5 /train images.

Data Processing

Processing the data was done with Tensorflow 2 Convolutional Neural Networks (CNN). Since the training is 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%
- no horizontal or vertical shift because the gesture is close to one edge in a few images
- normalized the 0…255 values from jpeg to the float interval [0., 1.]
- I did not use greyscale conversion because the color can provide more information
- rescaled the image to 255x255 for faster processing
- returned the image itself and the one-hot encoded class based on the class directory

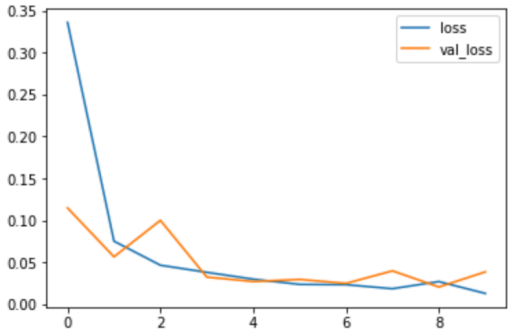
I trained and tested many CNN with different parameters. One idea was to use cv2.Canny() for simplifying the problem and for eliminating the variance introduced by different backgrounds. Unfortunately, the results were not good. The model was good for the /train images that were extracted from the same .mov file (very similar images with the corresponding /train categories) as shown below:



The accuracy for the model was about 98% but the real test was with Internet images.

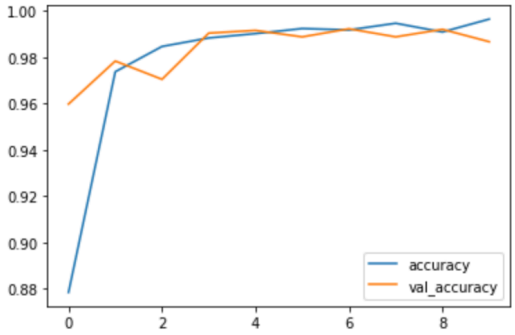
```
losses_model10[['loss','val_loss']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a5c33b790>
```




```
losses_model10[['accuracy','val_accuracy']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a5cadab90>
```



The problem with the Internet images is that it takes a long time to acquire good quality photos from the point of view of resolution and gestures. A few sample Internet images are shown below:

```
plotImages(test1_images[:5], test1_labels[:5])
```

```
Image 1  one-hot encoded class: [0. 1. 0. 0. 0.]
Image 2  one-hot encoded class: [0. 0. 0. 0. 1.]
Image 3  one-hot encoded class: [0. 0. 0. 1. 0.]
Image 4  one-hot encoded class: [1. 0. 0. 0. 0.]
Image 5  one-hot encoded class: [0. 0. 0. 1. 0.]
```

```
plotImages(test1_images[5:10], test1_labels[5:10])
```

```
Image 1  one-hot encoded class: [0. 0. 0. 0. 1.]
Image 2  one-hot encoded class: [0. 1. 0. 0. 0.]
Image 3  one-hot encoded class: [1. 0. 0. 0. 0.]
Image 4  one-hot encoded class: [0. 0. 0. 1. 0.]
Image 5  one-hot encoded class: [0. 0. 0. 0. 1.]
```

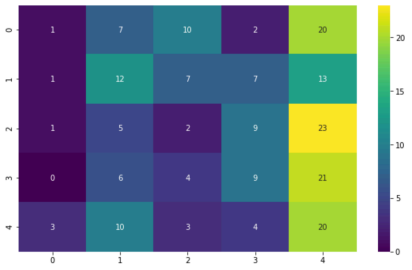
Furthermore, the Internet images do not always have white background, the resolution can be very different from the /train resolution and the size of the fingers can also vary. For these images, the results for the first few CNN models were disappointing:

```
print(classification_report(labels, predictions1))
```

```
              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
```

```
plt.figure(figsize=(10,6))
sns.heatmap(confusion_matrix(labels, predictions1), cmap='viridis', annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc31071b8d0>
```



The 0.22 accuracy is only marginally superior to the 0.2 accuracy for a random choice of one of the five classes. From the heatmap it is obvious that the model is biased towards class 4 and to a lesser extent towards class 2 (that is 5 fingers and 2 fingers).

Therefore, new models must be designed and tested.

This is the final model that can be actually used for Internet images:

```
model7.summary()
```

```
Model: "sequential_10"
_________________________________________________________________
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 hidden CNN layers with higher numbers of filters and different kernel sizes. These are the best parameters that I could find after running tens of models.

The training part went smoothly:

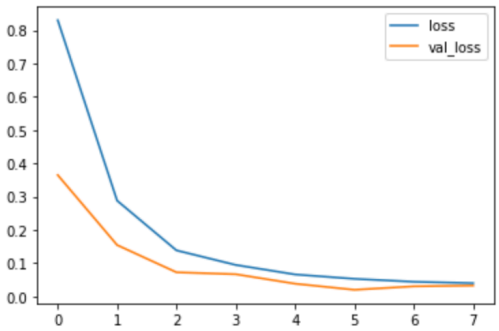
```
history = model7.fit_generator(train_data_gen,
                               epochs=epochs,
                               validation_data=test_data_gen,
                               callbacks=[early_stop])
```

```
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
WARNING:tensorflow:sample_weight modes were coerced from
  ...
    to
  ['...']
Train for 1014 steps, validate for 430 steps
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
```

Now, the 99% validation accuracy is even higher as for the previous models but this was calculated against /test data from the frames from the .mov files that were not used for training. The reason is that there are not enough Internet images even for /test for Tensorflow processing. The accuracy and loss are shown below:

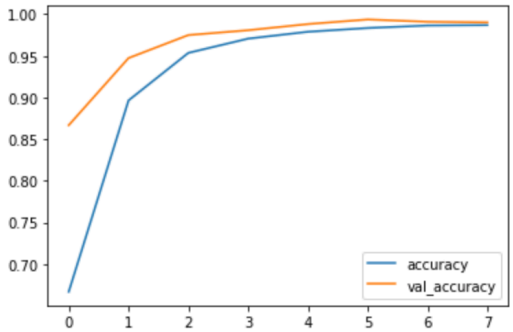
```
losses_model7[['loss','val_loss']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3bc8dea410>
```




```
losses_model7[['accuracy','val_accuracy']].plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3bcafadfd0>
```



For the Internet images, the metrics are now improved:

```
print(classification_report(labels, predictions7))
```

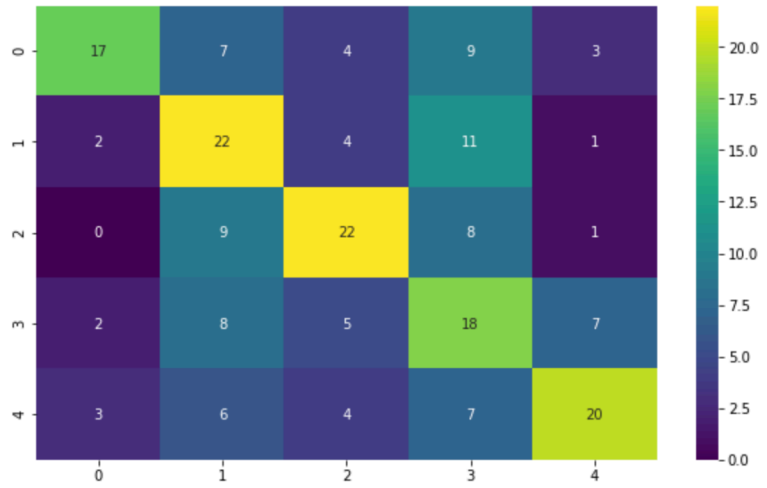
```
              precision    recall  f1-score   support

           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
```

```
confusion_matrix(labels, predictions7)
```

```
array([[17,  7,  4,  9,  3],
       [ 2, 22,  4, 11,  1],
       [ 0,  9, 22,  8,  1],
       [ 2,  8,  5, 18,  7],
       [ 3,  6,  4,  7, 20]])
```

```
plt.figure(figsize=(10,6))
sns.heatmap(confusion_matrix(labels, predictions7), cmap='viridis', annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3bcaf202d0>
```



The final test was done with 200 Internet pictures (40 pictures for each class). The accuracy of about 50% is not perfect but it is the best that was achieved so far. A good sign is that the values on the diagonal are the highest for any line or column.

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 these new images as a preprocessing step and train the CNN with these modified inputs.

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.