Hand Gesture Recognition

November 2019

By
Rajdeep Singh Lather
Mohammadjavad Esfandiari
Mostafa Tajic H.

Problem Definition

Hand Gesture Recognition

We are increasingly seeing Hand Gestures being used in Human-Computer Interaction. They allow an alternative way to interface with our computers and smartphones. Since most computers and smartphones already have a camera, it's convenient to identify gestures using just that. We are creating a model that can recognize hand gestures using just images.

Data Source

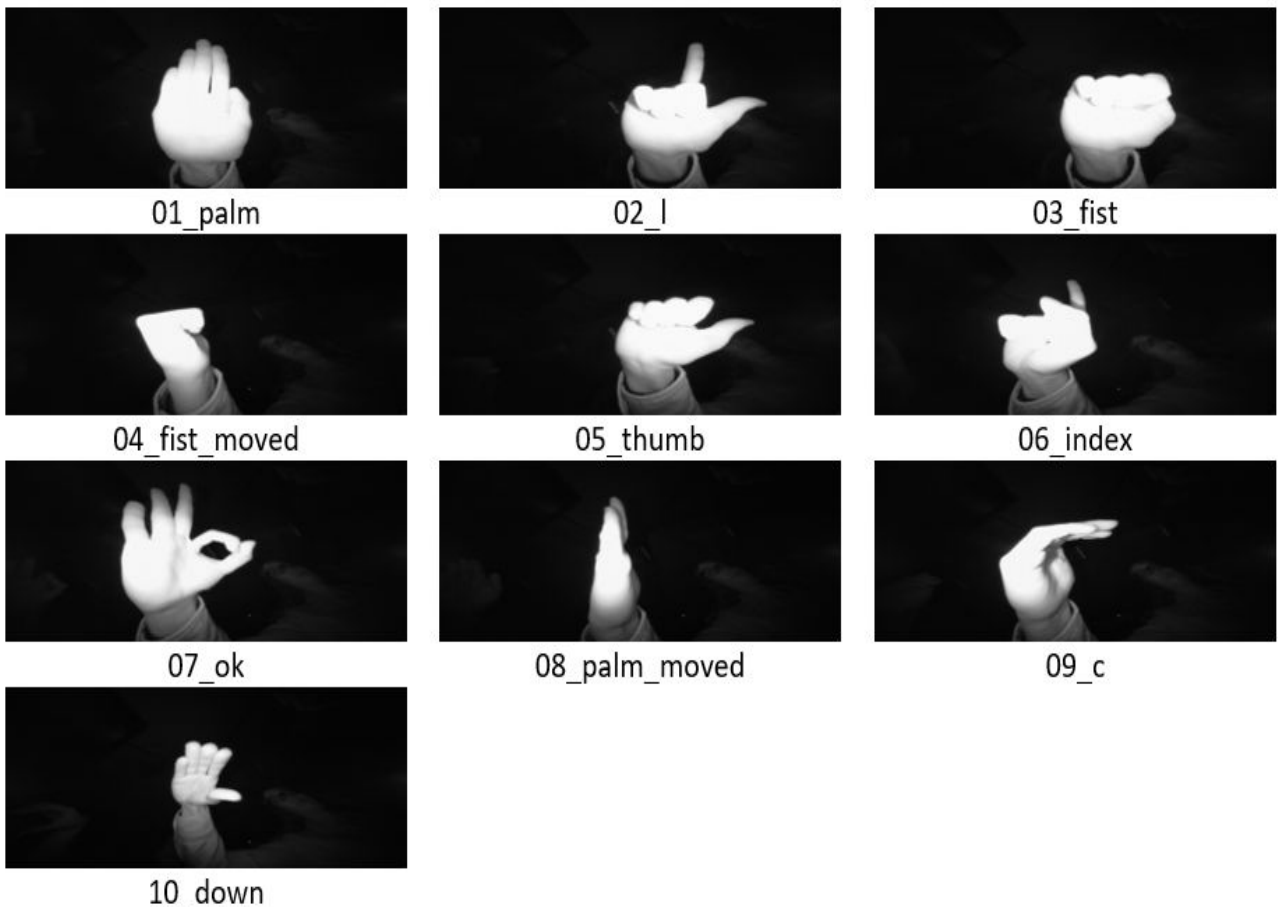
We got the "Hand Gesture Recognition Database" from the Kaggle web site. Hand Gesture Recognition Database, acquired by Leap Motion (kaggle.com/gti-upm/leapgestrecog). It is a set of near-infrared images of 1 GB in size.

System Design and Architecture

Data Capture

We are using a hand gesture recognition database, composed by a set of near-infrared images acquired by the Leap Motion sensor. The database is composed of 10 different hand-gestures that were performed by 10 different subjects (5 men and 5 women). One example of each of the gestures is presented below.

Visualization



Data Summary

Classes - 10

Instances - 20,000

Features - Image of size 640*240 with greyscale value 0-255 for each pixel

Pre-processing

For Preprocessing the image we need to reduce noise and also reduce the dimensions to make the training stage more tractable. Thus, we are using Gaussian blur to smooth out the image and then we are scaling it down to a more manageable size.

Feature Extraction

We have completed the feature extraction part of our project. Are feature extraction steps differ based on what techniques are we using. The 2 techniques that we are using are Decision Trees and Convolutional Neural Networks (CNN).

For decision trees and random forests, we used the scikit-image library to convert the png image into MxN ndarray and then used it for preprocessing. Next, we flatten the array, that is because decision trees in Scikit-Learn only accept 1-dimensional data.

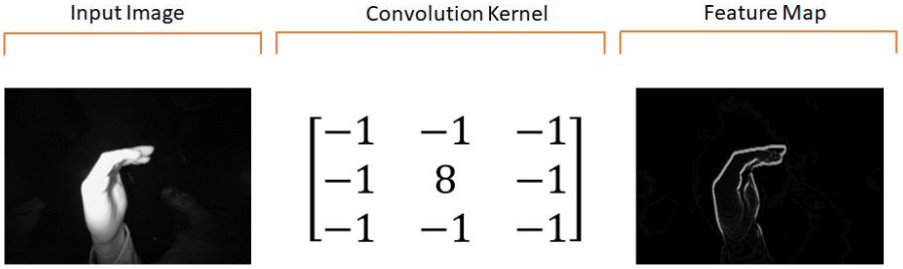
In CNN, we extract features directly from the images, so we do not need to flatten them. To extract the features we use convolutions, it reduces the number of free parameters, allowing the network to be deeper with fewer parameters. Convolution2D in Keras library gets the parameter filters, which is an integer and defines the dimensionality of the output space (i.e. the number of output filters in the convolution). In the first try, we considered 32 for this parameter to reduce the computation cost. Later in the model selection part, we can optimize this parameter. The other parameter, kernel_size, is an integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. It is considered as (3, 3) in this project.

Moreover, based on different filters that the model used for feature detection, there might be negative values in the model. We used Rectified Linear Unit (ReLU), which is defined as the positive part of its argument, as the activation function of our model to make sure there is no negative value in the pixels.

To reduce the dimensions of the data, pooling layers are used. In this project, 2 x 2 clusters, which are typical in CNN, are used. By keeping the maximum value of each cluster, the model keeps the most important feature of that cluster while reducing the dimension of the model.

Finally, flattening is employed to convert the output of the convolutional part of the CNN into a 1D feature vector. This vector can be used as the input layer of the ANN.

Now the features are ready and can be used as training data with different algorithms.



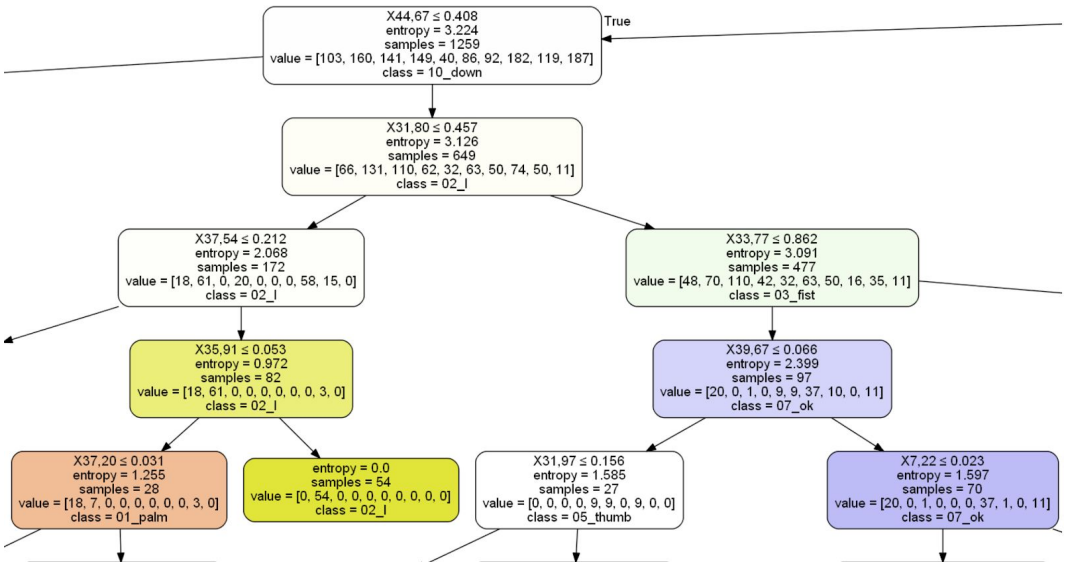
Data Mining

We are comparing between three different data mining techniques, namely Decision Tree, Random Forest and CNN.

Decision Trees

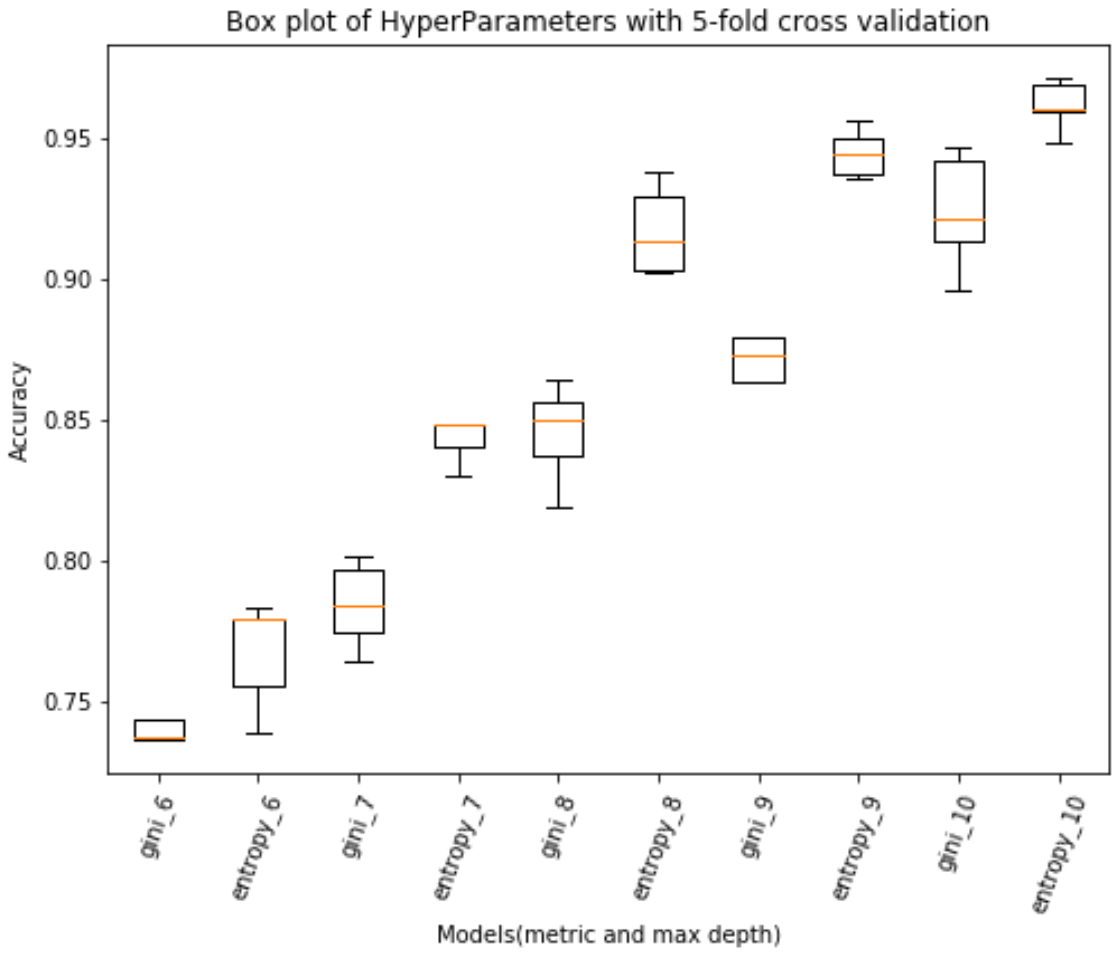
The decision tree model is created using the Scikit_learn package which uses the CART algorithm to grow a binary decision tree. During running "Trial and Error Iterations" on decision trees, we found that not limiting the size of decision trees let to our machines running out of available memory. Thus, we use an early stopping criterion of max-depth to ensure that the tree training step completes in a reasonable time on our machines.

Moreover, to find the best decision tree model a complete model selection approach has been employed. 10 different models varying impurity metrics (Entropy or Gini) and different depth (from 6 to 10) has been constructed. Furthermore, each tree model has been evaluated in a 5-fold cross validation process resulted in a box plot analysis to select the best decision tree model. The graph below shows part of the trained Decision Tree Architecture where Xij means the pixel in row i, column j of the image has been chosen as the splitting feature for the step.



The Best accuracy score for each model is listed below and also plotted using a boxplot.

```
Gini with depth=6 and Score:  0.7636898080928295
Entropy with depth=6  Score:  0.7830246832553952
 Gini with depth=7  Score:  0.8011483778606827
Entropy with depth=7  Score:  0.8832578802407103
 Gini with depth=8  Score:  0.8642661252815765
Entropy with depth=8  Score:  0.9375124995705587
 Gini with depth=9  Score:  0.9076931574439409
Entropy with depth=9  Score:  0.9561438504262793
 Gini with depth=10  Score:  0.946592423854263
Entropy with depth=10  Score:  0.9711257790569702
```



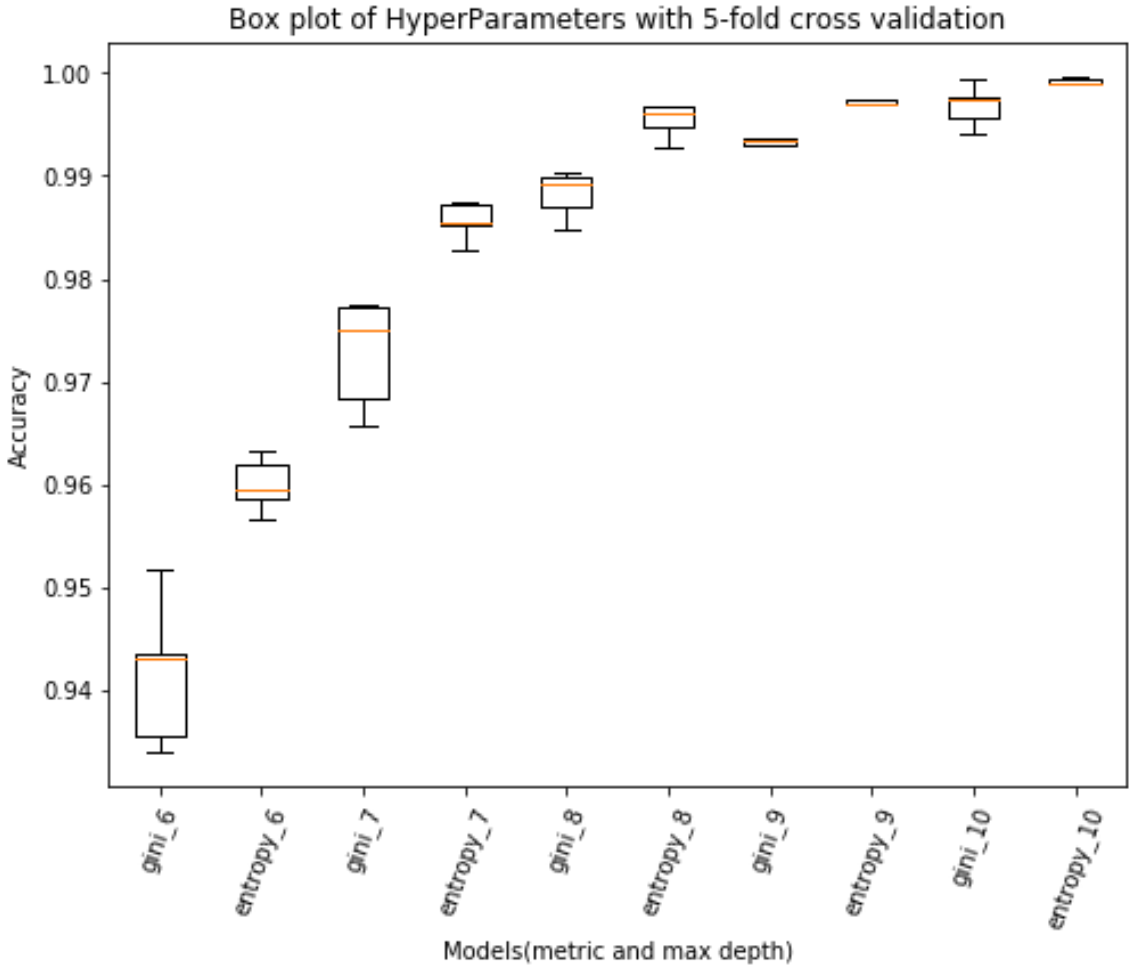
Random Forest

The Random Forest model is created using the Scikit_learn Random Forest Module. Here again we realized that not limiting the size of decision trees makes our machines running out of available memory. Thus, we use an early stopping criterion of max-depth to ensure that the tree training step completes in a reasonable time on our machines. Besides, the number of trees used as estimator in each Random Forest classifier cannot exceed a threshold or otherwise the time complexity of the programs makes running it impossible. The reason here is that we create 10 different tree classifier with different hyperparameters and then for each classifier we perform a 5-fold cross validation which in overall sums up to 50 classifier times the number of estimators in each forest. To make running the program possible we decided the number of estimator to be equal to 10. Therefore, at the end we have 10 random forest model where, each model has been evaluated in a 5-fold cross validation process resulted in a box plot analysis to select the best model for testing.

The Best accuracy score for each model is listed below and also plotted using a boxplot.

```
Gini with depth=6 and Score:  0.9517820860277604
Entropy with depth=6  Score:  0.9632952931747966
Gini with depth=7  Score: 0.9774635044615287
Entropy with depth=7  Score:  0.9874905645541135
Gini with depth=8  Score:  0.9902550645392484
Entropy with depth=8  Score:  0.9966864754676064
```

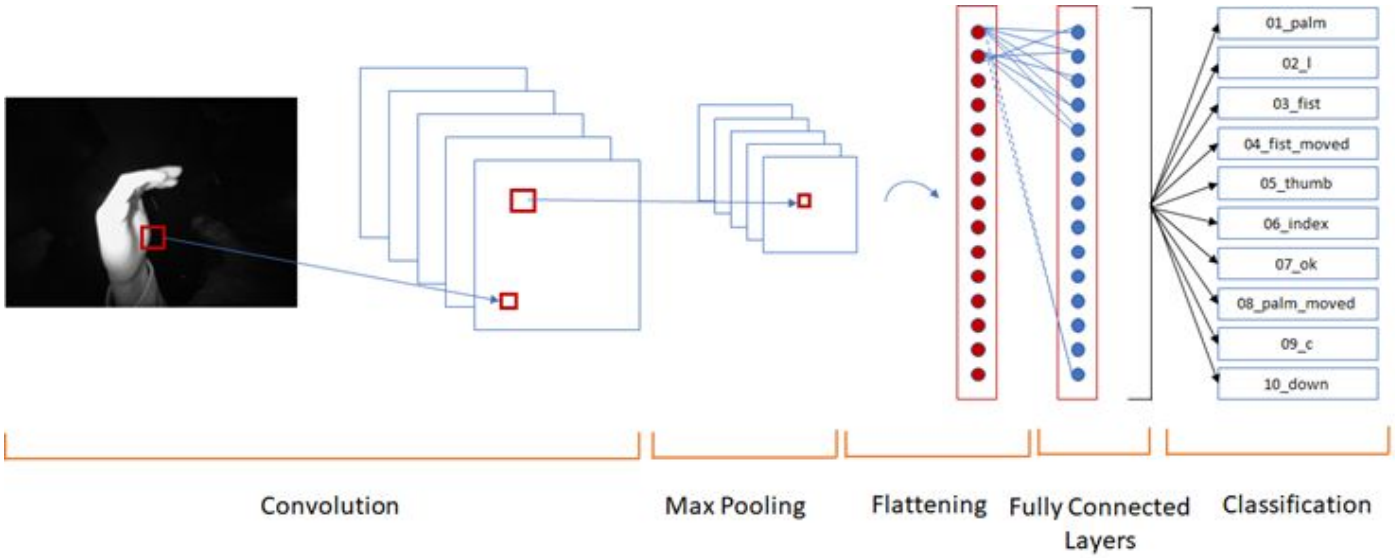
```
Gini with depth=9  Score:  0.996347552883992
Entropy with depth=9  Score:  0.9983366555924695
Gini with depth=10  Score:   0.9993355481727575
Entropy with depth=10  Score:  0.9996677740863789
```



CNN

Convolutional Neural Networks (CNN), a Deep Learning technique is the state of the art technique for image recognition. CNNs are a type of Deep Artificial Neural Network, i.e. they have multiple layers between the input and output layers. They use convolution in place of general matrix multiplication in at least one of their layers. There may be multiple convolution layers including layers such as pooling layers, fully connected layers, and normalization layers. We used the library Keras for Python, which uses TensorFlow, to build the model. CNN includes Convolution, max pooling, and flattening. Each step and the corresponding parameters are briefly described in the following section.

CNN Architecture



Convolution2D in Keras library gets the parameter filters, which is an integer and defines the dimensionality of the output space (i.e. the number of output filters in the convolution). We tried two values of 32 and 16 for this parameter. The other parameter, kernel_size, which is an integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window, is considered (3, 3) in this study. Based on different filters that the model used for feature detection, there might be negative values in the model. We used Rectified Linear Unit (ReLU), which is defined as the positive part of its argument, as the activation function of our model to make sure there is no negative value in the pixels.

To reduce the dimensions of the data, pooling layers are used. In this study, 2 x 2 clusters, which are typical in CNN, are used. By keeping the maximum value of each cluster, the model keeps the most important feature of that cluster while reducing the dimension of the model.

Finally, flattening is employed to convert the output of the convolutional part of the CNN into a 1D feature vector. This vector can be used as the input layer of the ANN.

Model Selection

The validation accuracy is better because regularization mechanisms, such as Dropout and L1/L2 weight regularization, are turned off at that time.

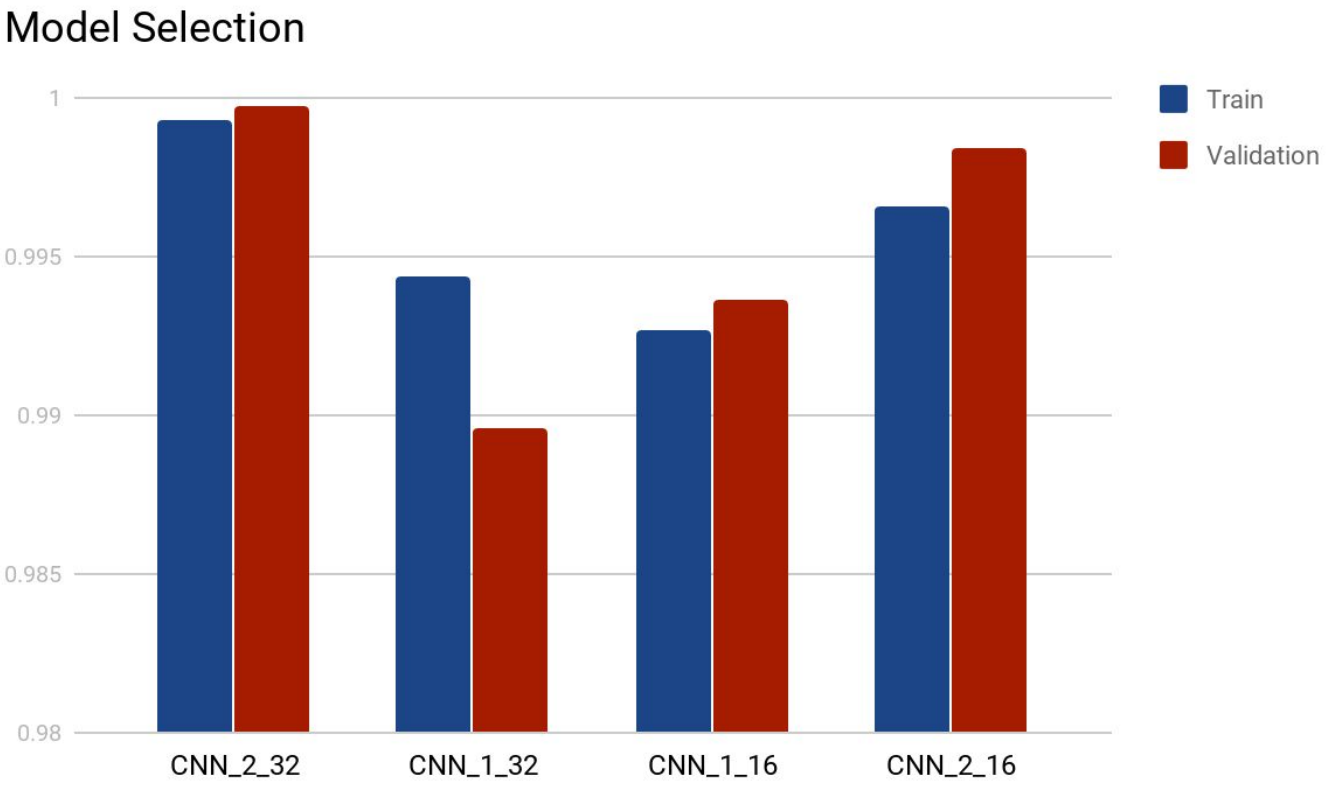
CNN with two layers and 32 filters
```
loss: 0.0023 - accuracy: 0.9993 - val_loss: 0.0013 - val_accuracy: 0.9997
```

CNN with only one layer and 32 filters:
```
loss: 0.0285 - accuracy: 0.9944 - val_loss: 0.0478 - val_accuracy: 0.9896
```

CNN with one layer and 16 filters:
```
loss: 0.0386 - accuracy: 0.9927 - val_loss: 0.0366 - val_accuracy: 0.9936
```

CNN with two layers and 16 filters:
```
loss: 0.0135 - accuracy: 0.9966 - val_loss: 0.0039 - val_accuracy: 0.9984
```

Best Model: CNN with two layers and 32 filters; validation accuracy of 0.9997



Metrics and Performance Evaluation

We have divided the dataset into 2 different groups with 75% records being used for development and the remaining 25% for testing. The development dataset is used for validation, model selection and training of our models.

The results of testing are reported below using Accuracy, Precision, and Recall. Additionally, we are attaching learning curves of our models.

Decision Tree

we have trained and tested a Decision Tree model using entropy as the splitting criterion, and a max depth of 10.

Classification Report

	precision	recall	f1-score	support
01_palm	0.92	0.82	0.87	561
02_l	0.88	0.85	0.86	517
03_fist	0.74	0.79	0.77	467
04_fist_moved	0.90	0.92	0.91	488
05_thumb	0.87	0.91	0.89	478
06_index	0.86	0.90	0.88	480
07_ok	0.92	0.96	0.94	480
08_palm_moved	0.83	0.89	0.86	471
09_c	0.92	0.84	0.88	550
10_down	0.94	0.92	0.93	508
accuracy			0.88	5000
macro avg	0.88	0.88	0.88	5000
weighted avg	0.88	0.88	0.88	5000

Confusion Matrix

```
[[459   6   8   1  49   3  19   0   6  10]
 [  2 439  49   5   9   2   0   0   7   4]
 [  5  26 370  29   2  17   0   1  13   4]
 [  1   9  10 451   0   0   0   1   3  13]
 [ 20   0  13   0 437   2   3   0   3   0]
 [  0  10  31   0   2 432   0   5   0   0]
 [ 13   3   0   4   0   0 459   1   0   0]
 [  0   2   3   0   1  30  11 417   7   0]
 [  0   5  15   9   0  14   0  46 461   0]
 [  0   0   1   1   0   0   8  29   0 469]]
```

The Random Forest performance evaluation

Classification Report:

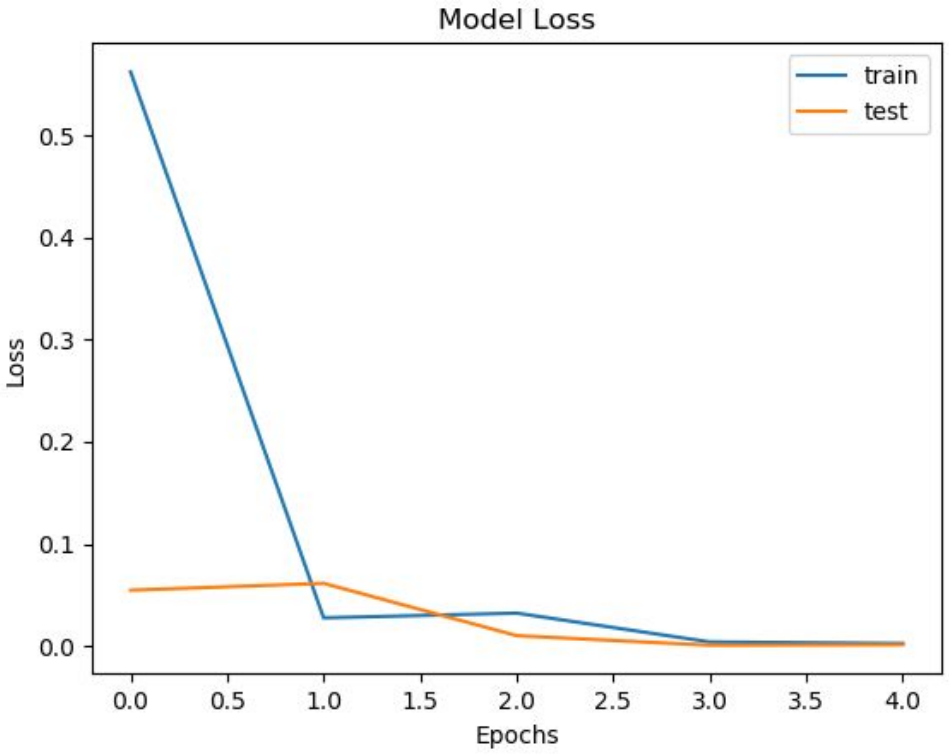
	precision	recall	f1-score	support
01_palm	0.95	0.98	0.96	500
02_l	0.99	0.89	0.93	500
03_fist	0.93	0.94	0.93	500
04_fist_moved	0.97	0.99	0.98	500
05_thumb	0.97	0.98	0.98	500
06_index	0.97	1.00	0.99	500
07_ok	0.99	1.00	0.99	500
08_palm_moved	0.99	0.96	0.98	500
09_c	0.97	0.98	0.97	500
10_down	1.00	1.00	1.00	500
accuracy			0.97	5000
macro avg	0.97	0.97	0.97	5000
weighted avg	0.97	0.97	0.97	5000

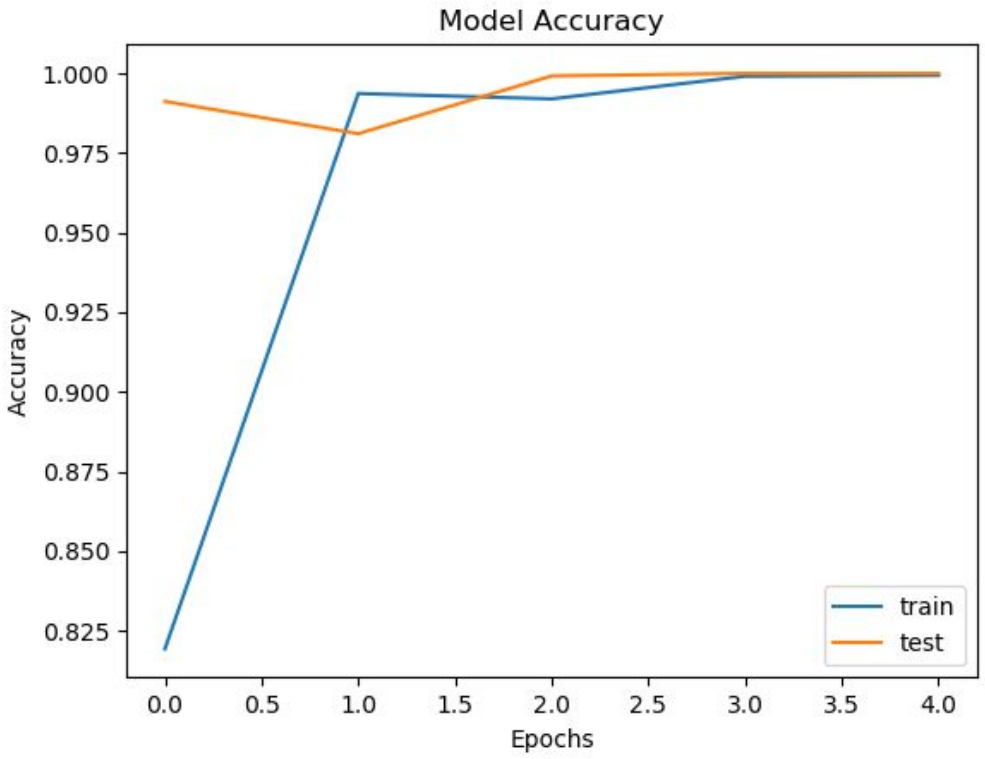
Confusion Matrix:

```
[[490   3   5   0   0   0   2   0   0   0]
 [  5 443  30   8   2   4   4   1   1   2]
 [  7   0 471   2  12   8   0   0   0   0]
 [  0   2   0 497   0   0   0   1   0   0]
 [  8   0   0   0 491   1   0   0   0   0]
 [  0   1   1   0   0 498   0   0   0   0]
 [  0   0   0   0   0   0 500   0   0   0]
 [  0   0   0   5   0   0   0 481  14   0]
 [  7   0   1   0   0   0   1   3 488   0]
 [  0   0   0   0   0   0   0   0   0 500]]
```

The Convolutional Neural Network

As it can be seen, the accuracy on the test set is 99%. This model is based on 2 layers and 32 filters.





```
Epoch 1/5
11250/11250 [==============================] - 83s 7ms/step - loss: 0.5651 -
   accuracy: 0.8238 - val_loss: 0.0773 - val_accuracy: 0.9760
Epoch 2/5
11250/11250 [==============================] - 87s 8ms/step - loss: 0.0412 -
   accuracy: 0.9902 - val_loss: 0.0154 - val_accuracy: 0.9973
Epoch 3/5
11250/11250 [==============================] - 90s 8ms/step - loss: 0.0151 -
   accuracy: 0.9964 - val_loss: 0.0046 - val_accuracy: 0.9992
Epoch 4/5
11250/11250 [==============================] - 99s 9ms/step - loss: 0.0090 -
   accuracy: 0.9979 - val_loss: 0.0035 - val_accuracy: 0.9992
Epoch 5/5
11250/11250 [==============================] - 93s 8ms/step - loss: 0.0023 -
   accuracy: 0.9993 - val_loss: 0.0013 - val_accuracy: 0.9997
```

Classification Report:

	precision	recall	f1-score	support
0	0.98	1.00	0.99	500
1	1.00	0.98	0.99	500
2	1.00	0.99	0.99	500
3	0.99	0.96	0.97	500
4	0.98	1.00	0.99	500
5	0.98	1.00	0.99	500
6	1.00	1.00	1.00	500
7	0.97	1.00	0.99	500
8	1.00	1.00	1.00	500
9	1.00	0.99	0.99	500
accuracy			0.99	5000
macro avg	0.99	0.99	0.99	5000
weighted avg	0.99	0.99	0.99	5000

Confusion Matrix:

```
[[500   0   0   0   0   0   0   0   0   0]
 [  0 488   0   0   3   0   0   9   0   0]
 [  0   0 494   0   0   0   2   4   0   0]
 [ 12   0   0 478   2   8   0   0   0   0]
 [  0   0   0   0 500   0   0   0   0   0]
 [  0   0   0   1   0 499   0   0   0   0]
 [  0   0   0   1   0   0 499   0   0   0]
 [  0   0   0   0   0   0   0 500   0   0]
 [  0   0   0   0   0   0   0   0 500   0]
 [  0   0   0   1   4   0   0   0   0 495]]
```

```
Model Details:
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 78, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 39, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 12, 37, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 18, 32)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 3456)              0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                221248
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
=================================================================
Total params: 231,466
Trainable params: 231,466
Non-trainable params: 0
```

Software & Hardware

Language: Python
Libraries: Pandas, Scikit-learn, Scikit-Image, MatPlotLib, PyDotPlus, Numpy, Keras, TensorFlow
Processor: x86 CPU - Intel i7
Operating System: macOs 10.14

Experience on How Trial and Error Iteration Helps

To sum up the experience gained from the trial and error we mention some of the top lessons learned during the process.

1. Knowing the basics of the what the package is doing under the hood shown to be crucial to the success of our model. As an example, understanding the fact that the Decision Tree implemented by Scikit-learn is a binary tree helps to be able to better interpret the outcome. In this project this understanding helped us to know that the only factor that can limit the growth of the tree is to limit the depth. There were also a number of other times where we had to dig into what the specific implementation of the classifier is doing the be able to achieve the desired outcome, for example in the CNN one of the issues was to convert our 2D images into a 4D array since that was what the 2D convolutional neural network expected as input.
2. The vital role of shuffling the data was another lessons taken from the trial and error iterations. It was found that the best practice is to shuffle the data before separating the training, validation and testing sets. This is going to have more impact on the result if the dataset is already organized, which was the

case for the dataset used for this report. The problem encountered before shuffling was that the train and validate set had an unbalanced number of instances from each class. This makes the model trained to be strong on some of the classes while the validation is trying to test it more on the classes it hasn't been trained well on.

3. Using the concept of box plots and learning curves is a great tool to help the programmer with making a lot of the hard decisions during the training process. A known issue with most of the classifier is the high number of hyperparameters and determining those hyperparameters is a milestone to develop a good model and using plotting tools such as box plots and learning curves makes justifying any decision much more feasible.

Conclusions

Based on the previous sections it is concluded that the CNN has the best performance on our database followed by Random Forest and last comes the Decision Tree model. The intuition behind this result is mainly based on remarkable features of CNNs, mainly the sophisticated convolution and feature selection and how it's been embedded in the training process which makes is the best choice for datasets with a large number of features which is the case for images. In a grayscale image each pixel represent a feature and in our case with a 60×160 image the number of features for each data point adds up to 9600. This fact makes the feature selection process have a high impact on the final outcome of the model and that's why greedy approach of a decision tree cannot do as well as the CNN. However, in the case of Random Forest, distributing the probability of an error in classification between 10 trees is changing the behaviour of the decision trees as expected and it boosts the performance of the model to 99% in validation and 97% in testing which is very close to the result given by CNN which is 99% in both validation and testing.. To sum up, decision tree shows to have detrimental limitations when it comes to datasets with high number of features but it is the fastest among the three methods and CNN shows to be the sweet spot to balance between speed and accuracy in our case since it was both faster and more accurate than the Random Forest classifier.

CNN; Accuracy: 99%
Random Forest; Accuracy: 97%
Decision Tree; Accuracy: 88%