Capstone Project

Mohamed Shehata

Machine Learning Engineer Nanodegree

16th of July 2018

Definition

Project Overview

Hearing loss is a big problem that affects all people across the globe, hence, the sign language was created to help the deaf communicate with other people, but not all people can understand sign language.
In this project, I created a machine learning project that takes an image of a hand movement and predict the number represented in the image by the sign language using CNN and SVM.

A similar machine learning problem that I found similar to mine is this one, In here he tried to predict the hand gesture and its corresponding letter in sign language, but the main difference is that he uses object detection and video instead of images. This project was inspired by this institute.

Problem Statement

The goal is to create an app that helps people better communicate with the deaf through predicting the number he's holding; The tasks involved are the following:

1. Downloading the sign language dataset
2. Exploring the data and seeing what the images look like.
3. Building a CNN model
4. Fitting and training the CNN model

5. Measuring the accuracy of the CNN model
6. Building an SVM to classify the images
7. Fitting the SVM model on the dataset
8. Measuring the accuracy of the SVM model
9. Determining which model is better

The final product will be useful for helping the deaf.

Metrics

- Accuracy is a common metric for classifiers
- Accuracy = (True positives + true negatives)/Dataset size
- Confusion Matrix can be very helpful to see model drawbacks.

The accuracy metric was used when evaluating the classifier because false negatives and false positives both erode the user experience:

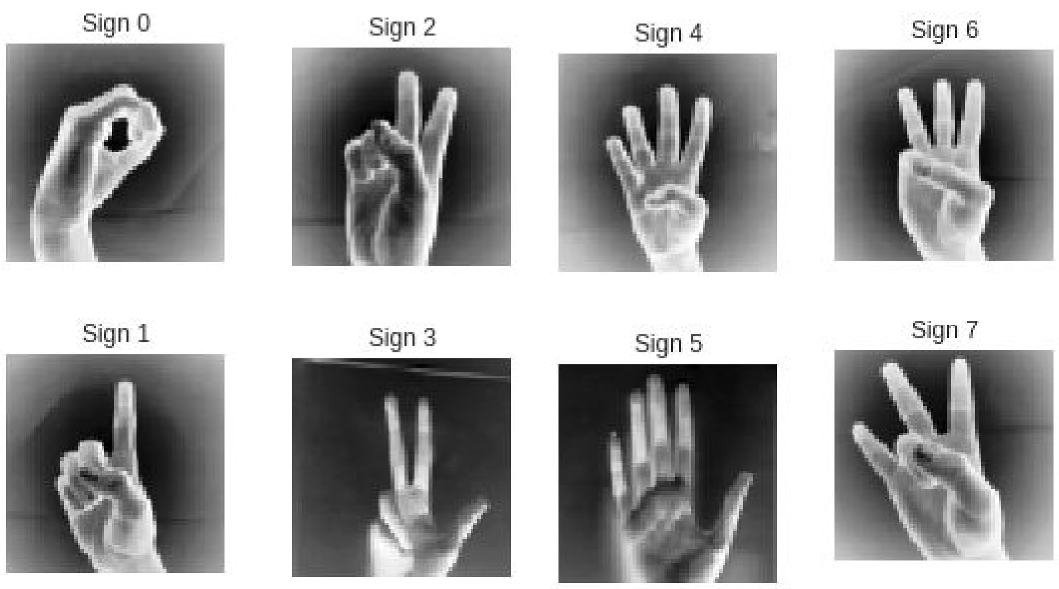
- False negatives result in either a longer delay between the user pointing the camera text and device speaking the text (“ processing delay ”) or in the worst case, completely prevent the application from predicting the digit.
- On the other hand, false positives make the application try to extract hand movements from images that don’t contain any. This results in unnecessary computations on the remote server, which can be both costly and slow the application down.

Analysis

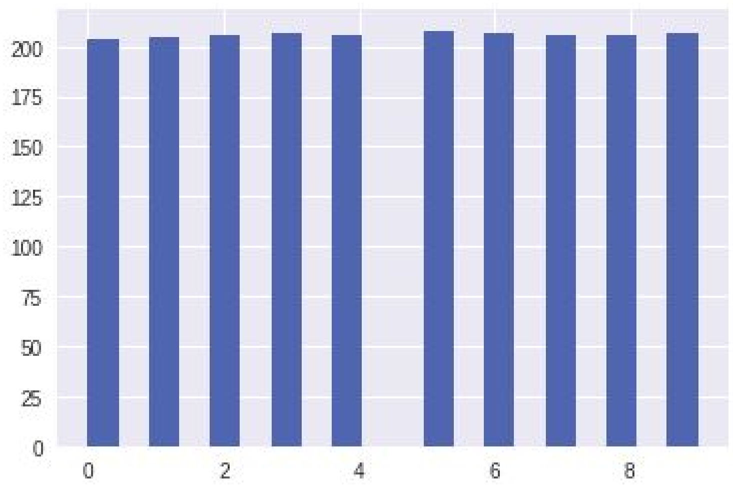
The sign language dataset size is 8 MB before extraction and 32 MB after extraction, divided into 10 classes with each class containing hand movements that represent a certain digit from 0 to 9.

All images are grayscale images with the same size 100*100 and represented in a numpy arrays.

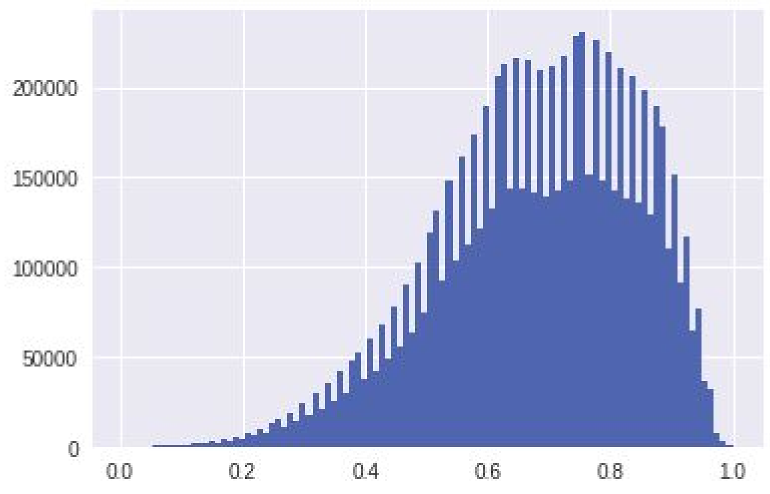
Here’s a sample from the images



The total number of images in the dataset is 2180, divided as 218 images for each digit class.
The dataset was collected by 218 students, 10 images were taken for each student to represent the 10 digits.
The figure below shows the number of images in each class
As you see each class has almost the same number of images
Which is good because very big difference in number of image ber class the model will suffer from biased to the biggest class.



In the figure below, I tried to get a glimpse of how the pixels intensity looks like by creating a histogram to show that.



Algorithms and Techniques

For this project, a wide variety of classification algorithms were used, like Convolutional Neural Networks, Logistic Regression and Support Vector Machine.

- **Convolutional Neural Networks:**

  It's the first model used in the project, CNN is considered to be a state-of-art algorithm for image processing tasks, generally, CNNs work better with training data compared to other approaches. This dataset may not be large in size, but it's big enough to fit this criteria and make the CNN comfortable with it. The output for CNNs is an assigned probability for each class; this is very good because using a classification threshold, the number of false positives can be reduced.

  - The tradeoff is this increases the number of false positives.

  The following parameters can be tuned to optimize the classifier:

  ❏ The classification threshold (as mentioned above)
  ❏ Training parameters
  ❏ Training length (number of epochs)

- ❏ Batch Size (How many images to look at once during a single training step)
- ❏ Solver type (What algorithm to use for learning)
- ❏ Learning rate (How fast to learn; this can be dynamic)
- ❏ Weight decay (prevents the model being dominated by a few 'neurons')
- ❏ Neural Network Architecture
- ❏ Number of Layers
- ❏ Layer Types (Convolutional, fully connected or pooling)
- ❏ Layer Parameters
- ❏ Preprocessing Parameters (It's not needed in this project as the dataset is already preprocessed.)

Benchmark

To create an initial benchmark for this model, I used more than classification algorithm and tested them against each other, the three algorithms are:

- Convolutional Neural Network
- Support Vector Machine
- Logistic Regression

Two of the three models did well but some models did better than other; The support vector machine classifier did very bad and got an accuracy of 49%.

The Logistic Regression did fairly well and got an accuracy of 76.4%; it's a very good accuracy in fact, but it wasn't the best.

The last model I used is the Convolutional Neural Network Model; it got a really high accuracy of 98.35%
This dataset was provided by kaggle, so one way to benchmark this model is to test it against other models.

This model got an accuracy of 98.35 %, and the model I tried to benchmark it for got an accuracy of 96.42%; It's close but my model got better accuracy and higher performance.
You can find the model I benchmarked for here.

Methodology

For this project, there isn't much preprocessing needed on the original dataset itself as the data was neatly made into images with a grayscale and a fixed size of 64*64; having the data already cleaned and preprocessed to some extent the second stage of the preprocessing started as I split the data into training, testing and validation sets; the splitting was made as follows:

- The original dataset was split into training set and testing set with a training set of size 80% of the original data and the testing set was 20% of this data
- The testing set was split into two sets: the testing sets and the validation set of size 50% of the testing and the testing set was 50%
- In total, the splitting was 80% for training, 10% for testing and 10% for validation.

The implementation part is divided into two parts.

- Building several models
- Testing these models against each other to check with which one I'll proceed for the rest of the project.

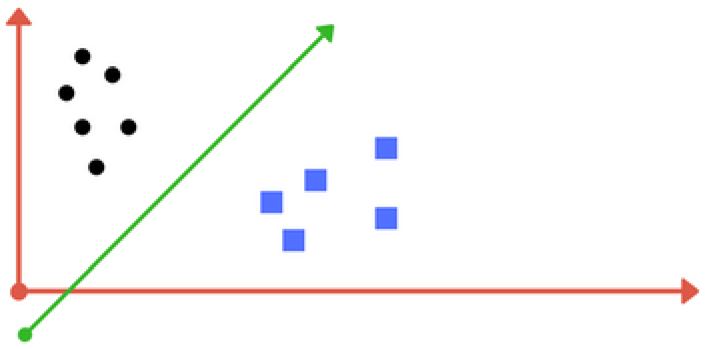
For the first part I built three different models:

- Support Vector Machine
- Logistic Regression
- Convolutional Neural Network

1. Support Vector Machine (SVM)

'A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimensional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side.'

The data is separated by a hyperplane with dimensions = number of dimensions - 1.



So, separating the data by a hyperplane is by trying to maximize the margin. I choose the kernel 'RBF' as it's a general purpose kernel.

After testing the model accuracy, it got an accuracy of 49%

2. Logistic Regression

Logistic regression is 'used where the response variable is *categorical*. The idea of Logistic Regression is to find a relationship between features and probability of particular outcome.

E.g. When we have to predict if a student passes or fails in an exam when the number of hours spent studying is given as a feature, the response variable has two values, pass and fail.

This type of a problem is referred to as Binomial Logistic Regression, where the response variable has two values 0 and 1 or pass and fail or true and false. Multinomial Logistic Regression deals with situations where the response variable can have three or more possible values.'

After testing the model accuracy, it got an accuracy of 76%

3. Convolutional Neural Network

The third and last model I used for this project is a Convolutional Neural Network model. During implementation stage, the classifier was trained on the preprocessed training data. And can be further divided into the following
steps:
1. Load both the training and validation images into memory, preprocessing them as described in
the previous section.
2. Implement helper functions:
   a. get_batch(...): Draws a random sample from the training/validation data.
   b. fill_feed_dict(...): Creates a feed_dict, which is a Python dictionary that contains all of the data required for a single training step (a batch of images, their labels, and the learning rate).
3. Define the network architecture and training parameters

4. Define the loss function, accuracy
5. Train the network, logging the validation/training loss and the validation accuracy
6. Plot the logged values
7. If the accuracy is not high enough, return to step 3
8. Save and freeze the trained network

CNN Architecture:

For this model, I had to reshape the data to be able to feed it to the neural network. And then the architecture was as follows:

Two convolutional layers followed by a pooling layer.
The convolutional layers had a 'relu' activation function.
The first convolutional layer had a 64 filter to extract the obvious features like: generally hands or fingers.
The second convolutional layer had a 32 filter to extract the less detailed features.
After that there's a pooling layer; The pooling layer performs downsampling operation along the spatial dimensions (width, height), outputting a reduced volume than the previous layer.
To reduce the chance of overfitting, a dropout layer was added to regularize the data as in this layer, it 'drops' 'neurons' at random while calculating the forward and back propagation.
The activation function which is a 'relu'; stands for Rectified Linear Units. This layer applies the non-saturating activation function. f (x) = x + = max(0, x)
It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

The last layer before the fully connected layer is the flatten layer, in this layer; the features are mapped into a single 1D vector. The flatten layer is needed so that you

	can make use of fully connected layers after some convolutional/maxpool layers. It combines all the
found	local features of the previous convolutional layers.

Now a fully connected layer is added. This layer will reduce the size of input data to the size of classes that the CNN is trained for by combining output of CNV layer with different weights. Each neuron at the output of the Convolutional layer will be connected to all other neurons.

The coding process went smoothly, except the part with shaping the data to make it fit with the different models, this got me frustrated a lot and I had to do a lot of research about how to reshape the data.

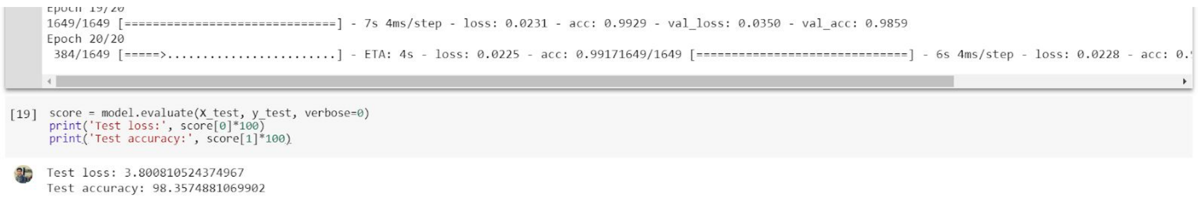
Refinements:

- At first, I made a large kernel size = 32*32 and few number of filters 4,2 in the convolutional layers , this got me an accuracy of 89% but after increasing the number of filters to 32, 64 and a kernel size = 4*4 the accuracy went up to 98.84%
- By increasing the number of epochs, I was able to increase the accuracy to 98.3%.
- Changing the dropout layer from 0.5 to 0.25 helped increase the accuracy from 98.8% to get a final accuracy of 99.0%

Results

Model Evaluation and Validation

The data was split into training, testing, and validation sets, with no overfitting happening. The splitting was as follows a training set of size 80% of the original data, and 20% for the test set. after that a validation set was mad with 50% from the test data and the test data was left with 50%.
After testing on validation set it give the score in the image:

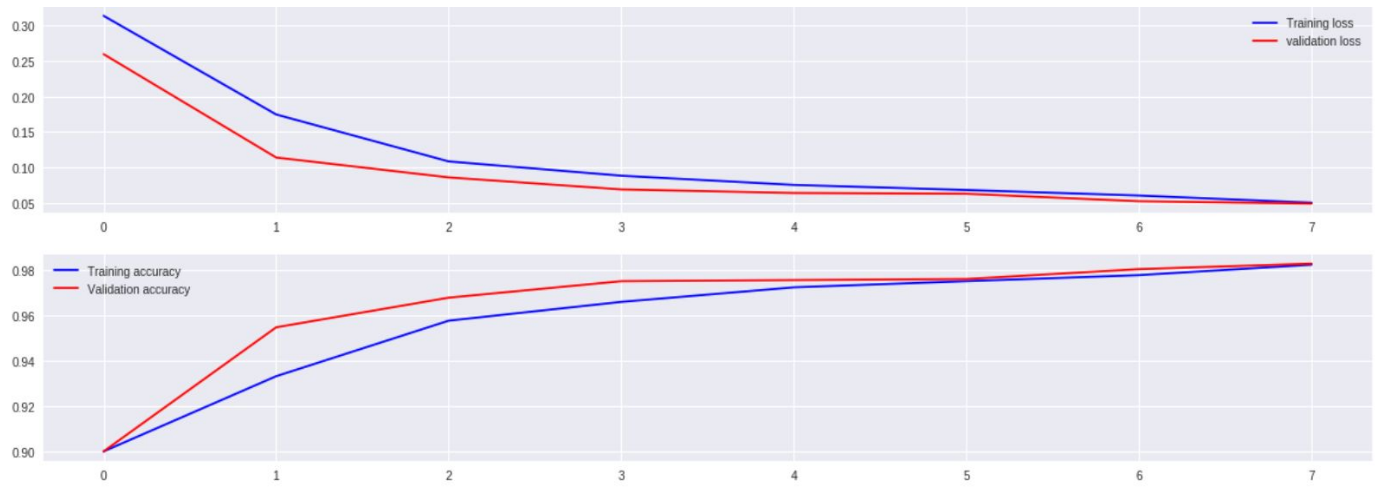


```
Epoch 19/20
1649/1649 [==============================] - 7s 4ms/step - loss: 0.0231 - acc: 0.9929 - val_loss: 0.0350 - val_acc: 0.9859
Epoch 20/20
 384/1649 [=====>........................] - ETA: 4s - loss: 0.0225 - acc: 0.99171649/1649 [==============================] - 6s 4ms/step - loss: 0.0228 - acc: 0.
```

```
[19] score = model.evaluate(X_test, y_test, verbose=0)
     print('Test loss:', score[0]*100)
     print('Test accuracy:', score[1]*100)
```

```
Test loss: 3.800810524374967
Test accuracy: 98.3574881069902
```

As for the evaluation and validation, I have used 2 main values to check: Accuracy and validation loss.



After going through errors and bug fixes, and increasing the number of epochs, the accuracy kept increasing until it reached an accuracy of 98.35% with 20 epochs with no overfitting happening to the model.

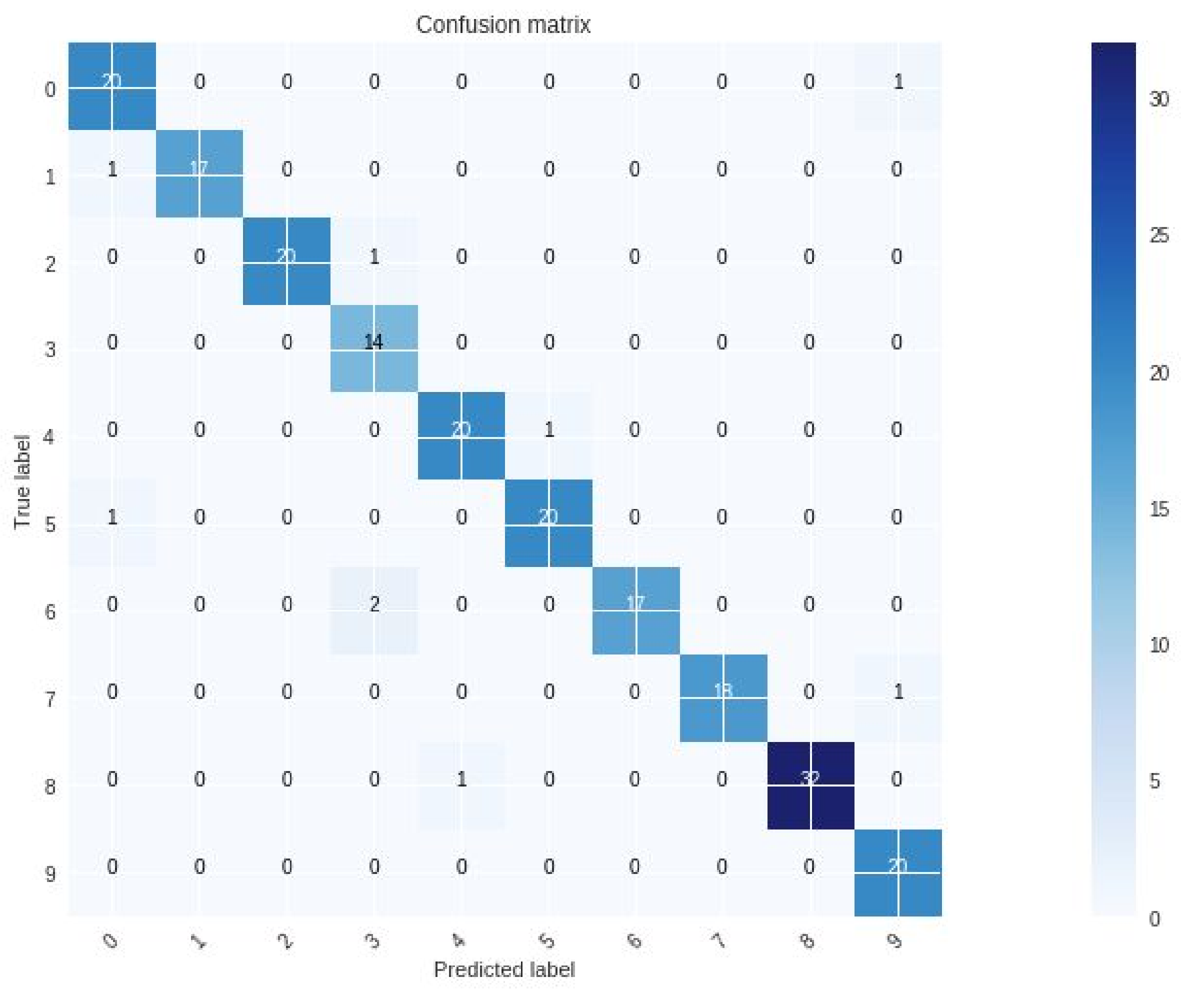
Justification:

The results of the final classifier are much better than that of the benchmark model. It has score of 98.35% is a decent score when compared to 96.42% (benchmark model's performance). I believe the final solution will definitely contribute significantly towards solving the current problem and also with more training data and more preprocessing stages, there are possibilities of improving the model further.

Conclusion:

This model can help people communicate with the sign language users easily in the number related conversations. Having a dataset of hand movements, and building a classification model using CNN to classify the hand movements into their respective digit. The problem was interesting to a great extent having at first being confused about what model to choose; this problem was solved by trying more than one model and testing their performance against each other.

To show the model quality we get the images which classified incorrectly in the test set this means the images that uncropped correctly can be misclassified. This model can help people communicate with the sign language users without the need to learn sign language. I plotted the confusion matrix to observe which category is poorly classified by the classifier and observe its performance visually. We can see in the below confusion matrix plot, that the classification was very good to some extent; It looks like the classifier is didn't have difficult time with classification Hence, the classifier doesn't need lots of improvements. But I believe that improving the dataset more will help boost the F1-score significantly.



True label \ Predicted label	0	1	2	3	4	5	6	7	8	9
0	20	0	0	0	0	0	0	0	0	1
1	1	17	0	0	0	0	0	0	0	0
2	0	0	20	1	0	0	0	0	0	0
3	0	0	0	14	0	0	0	0	0	0
4	0	0	0	0	20	1	0	0	0	0
5	1	0	0	0	0	20	0	0	0	0
6	0	0	0	2	0	0	17	0	0	0
7	0	0	0	0	0	0	0	18	0	1
8	0	0	0	0	1	0	0	0	32	0
9	0	0	0	0	0	0	0	0	0	20

Improvements:

As a future work and improvements, I'd like to try to increase the dataset size to help increase the accuracy, also, another datasets and models can be featured into the model to help translate the sign language into written words or spoken, not just the digits to help them speak better in video calls or in real life with non speakers.

References:

1. https://medium.com/machine-learning-101/chapter-2-svm-support-vector-machine-theory-f0812effc72
2. https://medium.com/data-science-group-iitr/logistic-regression-simplified-9b4efe801389
3. https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-1.pdf
4. https://en.wikipedia.org/wiki/Convolutional_neural_network
5. https://medium.com/machine-learning-world/convolutional-neural-networks-for-all-part-ii-b4cb41d424fd