Machine Learning Engineer Nanodegree

Capstone Project

Khalid Hakami

Oct 18th, 2018

I. Definition

Project Overview

Domain Background

Arabic language, one of the top spoken languages in the world. A language that has lots of unique features compared to other languages. It is written form right to left and the characters are so artistic that each writer can have his own style. For all the beautiful features it has, it became so complex for computers to detect. The research in Arabic computing are not enough and there are lots of potential to improve. As an Arabic speaker, I find myself responsible to use my knowledge in machine learning to help the industry to evolve and provide more solutions to Arabic speakers.

Problem Statement

Arabic words are always written in cursive (joined) style and it has deferent shapes and technique which make it harder to read words. On the other hand, recognizing Arabic characters letter by letter is much easier. Arabic characters are used in many identification problems, such as car plates, employees' IDs and others. An auto recognition system for Arabic characters will help creating security systems, traffic monitoring and so on.

Solution Statement

The proposed solution is to use CNN to detect handwritten Arabic charchters, by providing the input as csv array images.

Evaluation Metrics

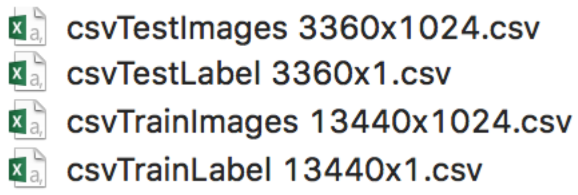
I will evaluate the performance by looking at the number of correct classified characters and calculate the Accuracy of the model and then compare it to the benchmark model.

$$Accuricy = \frac{N\ correct}{N\ predicted}$$

Analysis

Data Exploration

The data set used in this project is a data set from Kaggle. The data-set is composed of 16,800 characters written by 60 participants, the age range is between 19 to 40 years, and 90% of participants are right-hand. Each participant wrote each character (from 'alef' to 'yeh') ten times. The data was scanned at the resolution of 300 dpi. Each block is segmented automatically using Matlab 2016a to determining the coordinates for each block. The database is partitioned into two sets: a training set (13,440 characters to 480 images per class) and a test set (3,360 characters to 120 images per class).
(https://www.kaggle.com/mloey1/ahcd1)


csvTestImages 3360x1024.csv
csvTestLabel 3360x1.csv
csvTrainImages 13440x1024.csv
csvTrainLabel 13440x1.csv

The data are stored as csv files of 1024 column. Each column represents a pixel in an image and each row represents an individual grayscale image. The value of each pixel varies from 0-255.

The data was cleaned and labeled. I normalized the values to optimize the performance of the training.

```
x_train = tf.keras.utils.normalize(x_train, axis=1)
x_test = tf.keras.utils.normalize(x_test, axis=1)
```

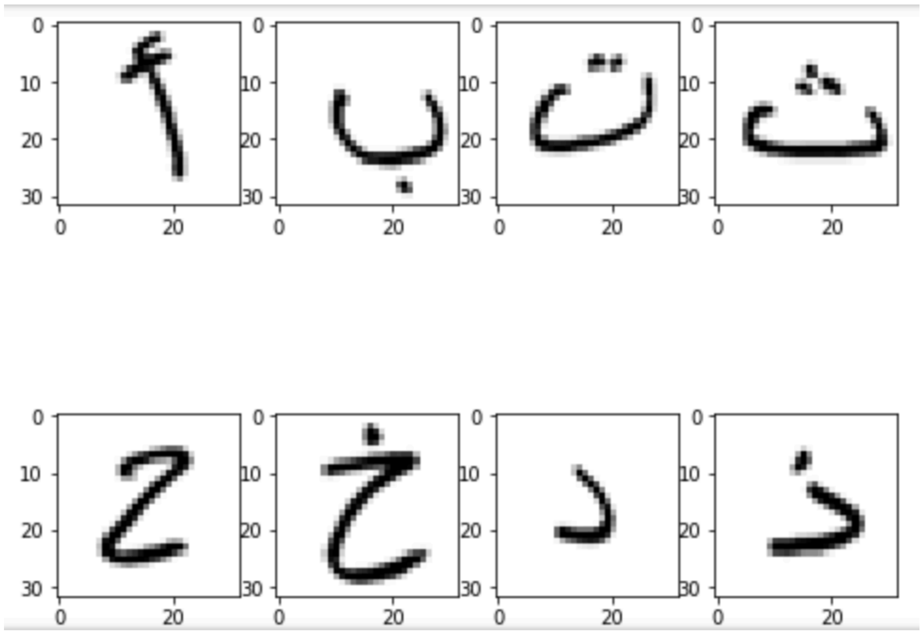
Exploratory Visualization

This problem has 28 feature class (total number of alphabetic characters in Arabic). 600 images per class. To ease the visualization and the processing of the images, it was read into NumPy arrays and then reshaped into a 2D arrays. Swapping the axes was an extra step made for better visualization.

```
x_train = x_train.iloc[:,:].values.astype('float32')
y_train = y_train.iloc[:,:].values.astype('int32')-1
x_test = x_test.iloc[:,:].values.astype('float32')
y_test = y_test.iloc[:,:].values.astype('int32')-1
```

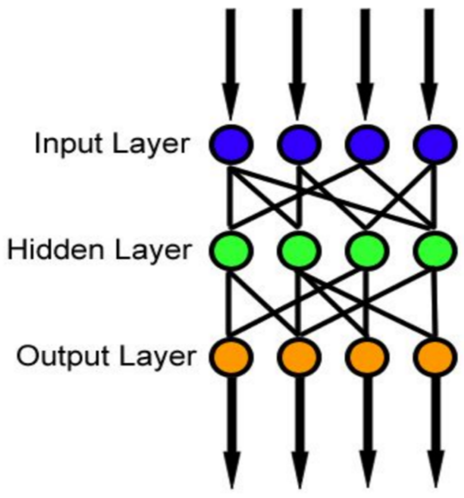
```
x_train = x_train.reshape(x_train.shape[0], 32, 32)
x_train = x_train.swapaxes(1, 2)
x_test = x_test.reshape(x_test.shape[0], 32, 32)
x_test = x_test.swapaxes(1, 2)
```

the final result after plotting the 32x32 pixel images using *matplotlib.pyplot:*



Algorithms and Techniques

As intended in the proposal I will use Convolutional Neural Network to identify alphabetic characters.



I will use a sequential model so everything goes in a one order. The model will have:

- Flatten layer: to make 32x32 image to 1x1024 image
- Dense layer: double Dense layers with 128 units and the activation function is `relu`,
- Output layer : a Dense layer with softmax activation funcation.

Benchmark

The model will be benchmarked against a random benchmarked model. One suggested model is https://github.com/tahaemara/arabic-characters-recognition

Methodology

Data Preprocessing

The data was clean and labeled. No outliers or noise where detected. So the only processing steps where:

1- Normalizing the pixels values, which showed better training result in less training time compared to the non-normalized data.

2- Reshaping the numpy arrays:

   a. To (32,32,1) 4D array for the first model.

```
x_train = x_train.reshape([-1, 32, 32, 1])
x_test = x_test.reshape([-1, 32, 32, 1])
```

   b. To 32x32 2D array for the second model.

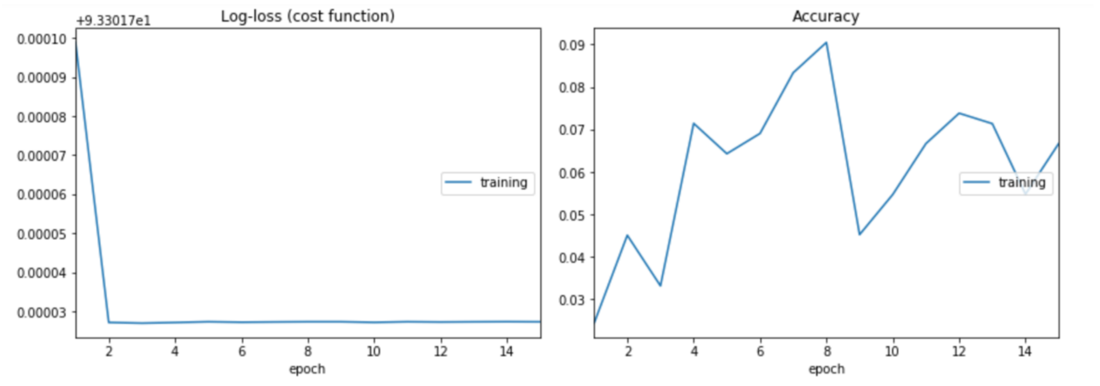
```
x_train = x_train.reshape(x_train.shape[0], 32, 32)
x_train = x_train.swapaxes(1, 2)
x_test = x_test.reshape(x_test.shape[0], 32, 32)
x_test = x_test.swapaxes(1, 2)
```

Implementation

At the beginning, I used the following layers:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_56 (Conv2D)           (None, 32, 32, 32)        832       
_________________________________________________________________
conv2d_57 (Conv2D)           (None, 32, 32, 32)        25632     
_________________________________________________________________
max_pooling2d_29 (MaxPooling (None, 16, 16, 32)        0         
_________________________________________________________________
dropout_44 (Dropout)         (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_58 (Conv2D)           (None, 16, 16, 64)        18496     
_________________________________________________________________
conv2d_59 (Conv2D)           (None, 16, 16, 64)        36928     
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 8, 8, 64)          0         
_________________________________________________________________
dropout_45 (Dropout)         (None, 8, 8, 64)          0         
_________________________________________________________________
flatten_21 (Flatten)         (None, 4096)              0         
_________________________________________________________________
dense_57 (Dense)             (None, 256)               1048832   
_________________________________________________________________
dense_58 (Dense)             (None, 256)               65792     
_________________________________________________________________
dropout_46 (Dropout)         (None, 256)               0         
_________________________________________________________________
dense_59 (Dense)             (None, 28)                7196      
=================================================================
Total params: 1,203,708
Trainable params: 1,203,708
Non-trainable params: 0
```

This architecture was selected based on different sources. However, the result of the model was not acceptable. As you can see from the accuracy graph below, the training accuracy reached 4% and the lose function reached a plateau at 0.00003



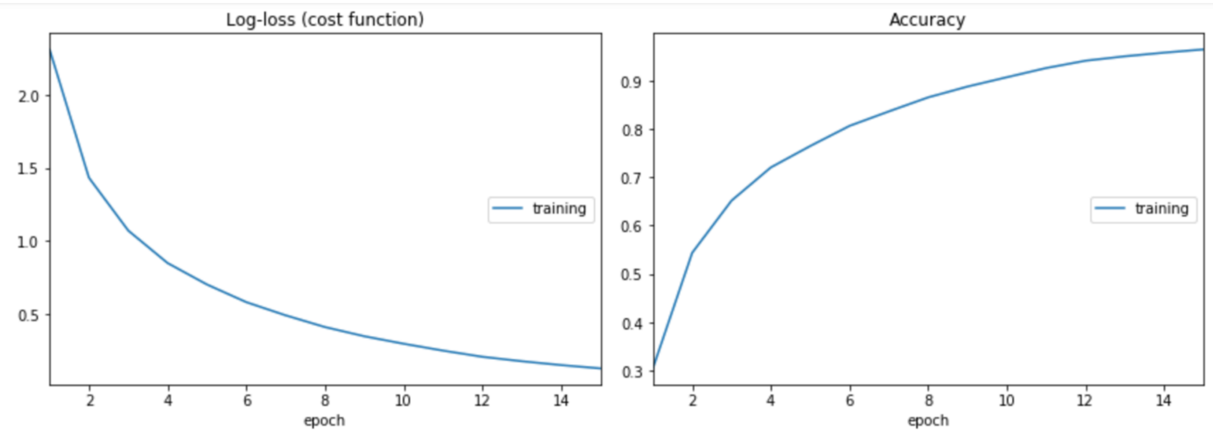
Refinement

To refine the model, I decided to make the layers as simple as I can and then try to improve the accuracy. The layers I chose was:

1- A flatten layer
2- Two Dense layer with 128 units and the activation function is `relu`,
3- A final Dense layer with softmax activation function

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
flatten_3 (Flatten)          (None, 1024)              0         
_________________________________________________________________
dense_9 (Dense)              (None, 128)               131200    
_________________________________________________________________
dense_10 (Dense)             (None, 128)               16512     
_________________________________________________________________
dense_11 (Dense)             (None, 28)                3612      
=================================================================
Total params: 151,324
Trainable params: 151,324
Non-trainable params: 0
_________________________________________________________________
```

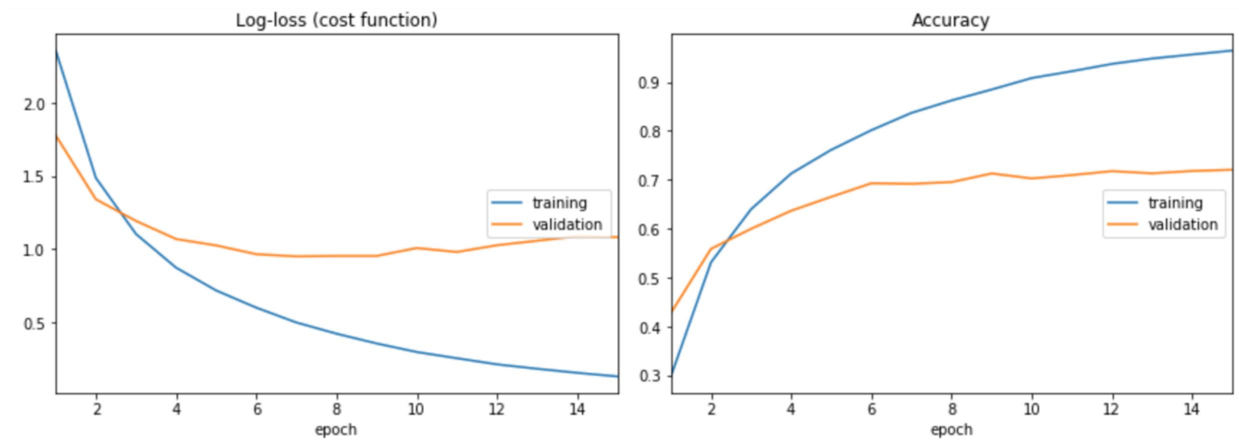
this model has better accuracy with training accuracy of 98% and testing accuracy of 72%



Results

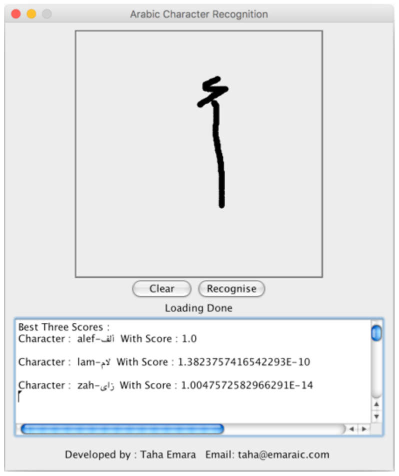
Model Evaluation and Validation

Based on the two accuracy result from the previews section. The second model was the final pick with training accuracy of 98% and testing accuracy of 72%.



Justification

I believe this is a reasonable accuracy but not acceptable for real world solution. There is still big space for improvement specially when this model is compared to the selected bench mark model that have an accuracy of 92.29%

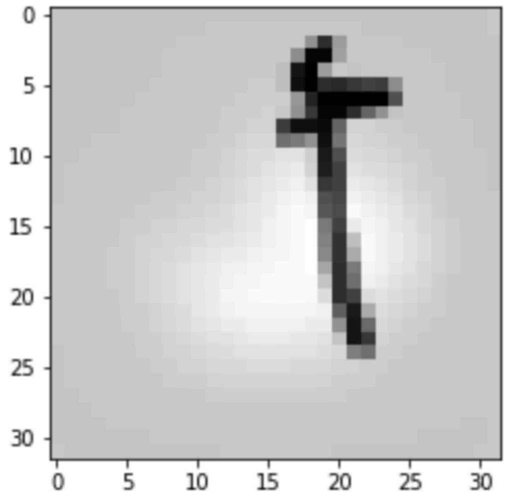


```
Loading Done

Best Three Scores :
Character :  alef-الف  With Score : 1.0

Character :  lam-لام  With Score : 1.3823757416542293E-10

Character :  zah-زای  With Score : 1.0047572582966291E-14

Developed by : Taha Emara   Email: taha@emaraic.com
```

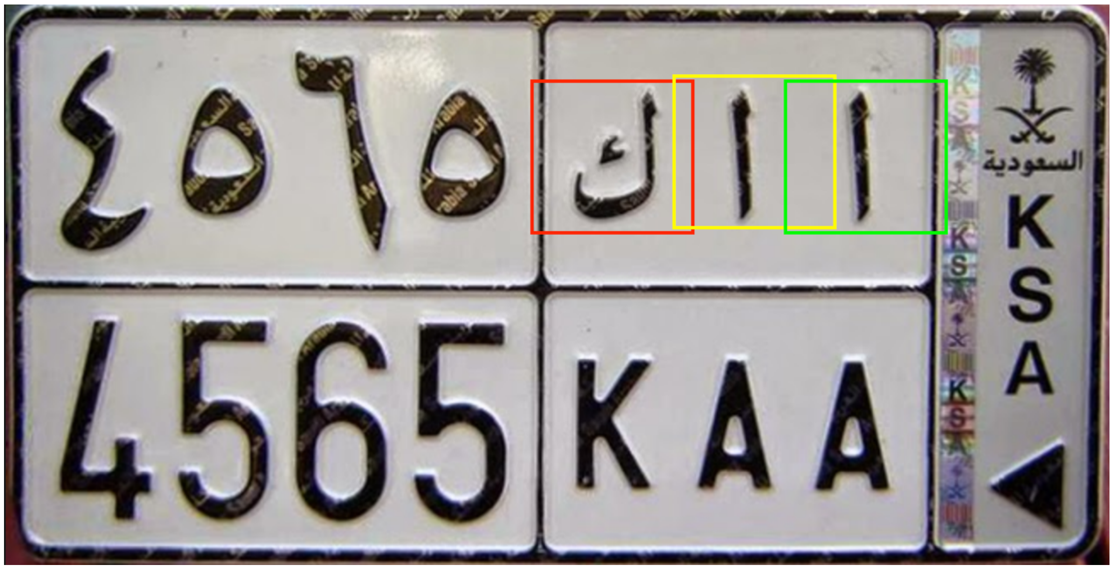
Conclusion

Free-Form Visualization

This model can detect Arabic characters from any gray scale 32x32 pixle image. Which make it a useful model for real case applications (after improving the accuracy).



this model can be applied to recognize Arabic car plates such the following:



Reflection

Arabic language is not like any other language. Its letters have some unique
structure and because of the lack of content in Arabic language I decide to create this
model.
After trying two different CNN models I believed that we can still improve the content of Arabic computing by collecting more Arabic datasets and trying to use more advance machine learning techniques.
This project motivates me to do further research in this area and the next steps for me is to do improve this model and then try to build another model that cover Arabic digits and then Arabic words.

Improvement

My model detects Arabic characters only. A real case application will need to read a whole word and not only the letters.
Arabic is different than English when it comes to word spelling, where the characters will have different shape depending on its positon at the word. An example for one letter would look like this “ح ـحـ”. It would be a very interesting problem to solve using machine learning.