



COMP6345 - Intelligent System

Interactive Character Recognition Using CNN (Convolutional Neural Network)

Submitted by

Noviani Litriani - 2001594623

Reinaly Edbert Fargo - 2001594573

Santi Amelia Andrini - 2001614871

10/1/2019

Submitted to

Raymondus Raymond Kosala

Table of Contents

Table of Contents	2
Problem Description	3
Solution Features	4
Solution Design Architecture	6
Experiments and Tests	7
Program Manual	9
Project Documentation	10

1. Problem Description

In today's modern world, we can see that every aspect of our lives is surrounded by technology devices, such as laptops and mobile phones. Not only adults, but children aged 5-10 years are also fascinated by the invention of smart devices, especially tablets. They often keep their eyes glued to the screens for entertainment purposes only, hence it can be a huge distraction for their studies.

Another distinguishable problem we have noticed from children is that some of them tend to have a hard time to focus when studying. This happens even from a very young age when they start to learn basic knowledge and skill, such as reading and writing alphabets.

All things considered, we decided to create an Interactive Character Recognition application as our final project for this course. The application is created to provide more interactive and more effective learning styles for young generations. We are also inspired by a project we have discovered on Youtube. It is called Letter Adventure, an educational game which uses a Kinect sensor for user input.



Letter Adventure - Linguistic Educational Game Video. Retrieved from <https://www.youtube.com/watch?v=kkOHGCKhKoc>

2. Solution Features

Algorithm

For this project, Convolutional Neural Network (CNN) is our main algorithm to make a prediction from the drawings. CNN is a class of deep neural networks which most commonly applied to analyzing visual imagery. CNN use a variation of multilayer perceptrons designed to require minimal preprocessing. CNN is one of the most used deep neural networks that is used in analyzing visual imagery because CNN is able to process and detect any feature that is available inside an image.

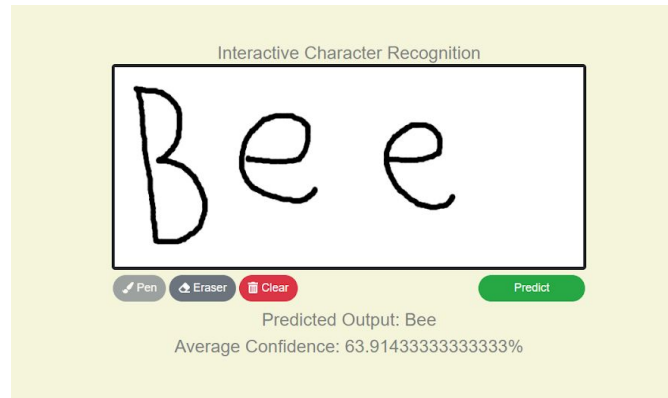
Technologies

Some libraries and technologies are included in this project to help us achieve our objectives, and those are:

- Keras
It is a high-level neural networks API. It is used to create the deep learning library for our model.
- Open Source Computer Vision Library (OpenCV)
It is an open source computer vision and machine learning software library and used for detecting each alphanumerical character drawn in the canvas.
- Flask
It is a Python microframework for web development.
- MNIST Database
It is a database of handwritten digits (0-9), commonly used for training image processing systems. In this case, we use EMNIST or Extended MNIST database for our dataset since it offers digits, small (a-z) and capital (A-Z) letters. The dataset can be downloaded [here](#).

User Interface

We utilized Flask, the Python-based web framework to help us creating the user interface of the application. The UI design is very simple since the targeted user for this application will be children aged 4-6.



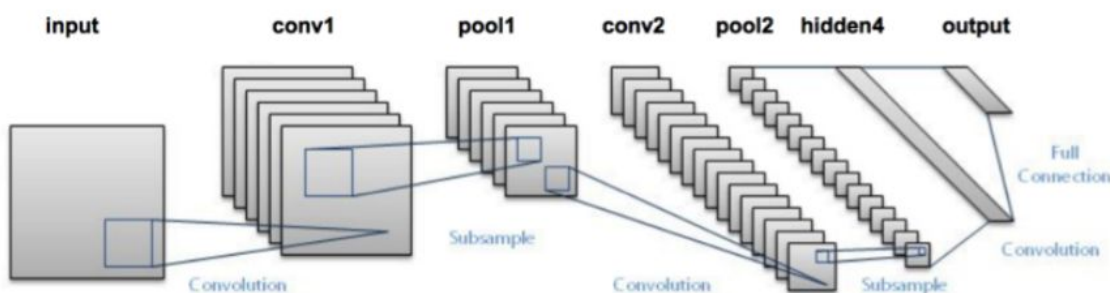
ICR User Interface.

3. Solution Design Architecture

We are using a LeNet architecture as our model when we are training our artificial intelligent. There are 2 convolution layers that are used and 1 Max Pooling layer that is used after the processed images have gone through both of the convolution layers. Convolution layer is the layer to extract features from an input image where the relationships between pixels are preserved by learning image features using small squares of input data. Pooling layers section would reduce the number of parameters when the images are too large. We are using MaxPolling to take the largest element from the rectified feature map.

After the data has passed the pooling layer. It will then be processed by the hidden layers. Those hidden layers are Dropout, Flatten, and Dense (ReLU and SoftMax). Dropout is used to make sure that the artificial intelligence is not over-trained which could result in a worse accuracy when being tested. Flatten is used to Flatten the output and feed into a fully connected layer (FC Layer). Dense layer is used to Take a number of neurons and activation function as arguments.

After the data has passed the hidden layers, the data will then be compiled and used in our application artificial intelligence.



```
model = Sequential()
model.add(Convolution2D(nb_filters, kernel_size, padding='valid', input_shape=input_shape, activation='relu'))
model.add(Convolution2D(nb_filters, kernel_size, activation='relu'))
model.add(MaxPooling2D(pool_size=pool_size))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(nb_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adadelta', metrics=['accuracy'])
```

4. Experiments and Tests

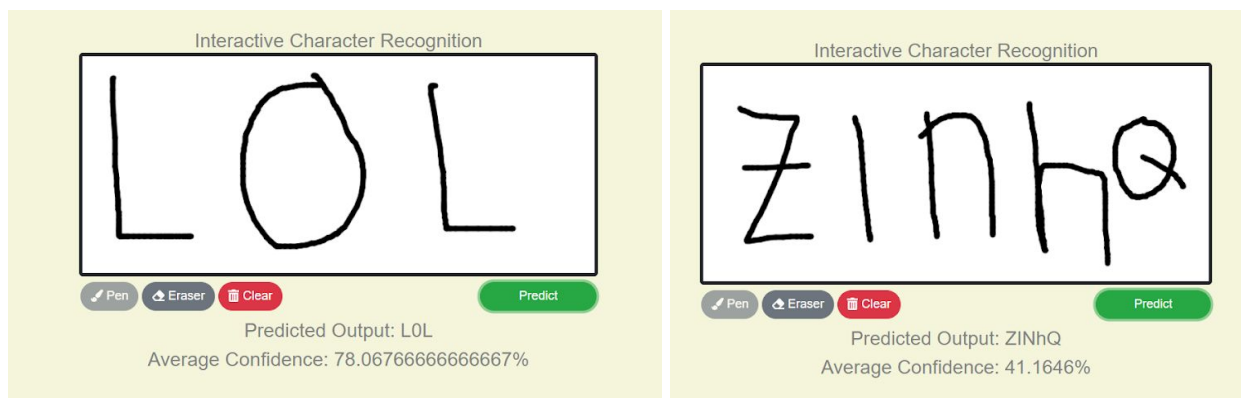
During the development of ICR, we tried training two EMNIST datasets: ByClass and Balanced dataset. At first, we chose running 10 epochs on ByClass dataset which contains more than 650,000 samples. It took around 3 hours to finish the training and testing process and showed an accuracy of 87.7%. However, when we did manual testing by drawing the characters from our application, most predicted outputs were incorrect.

Then, we decided to train the Balanced dataset with the same number of epochs. The dataset has an equal number of samples per class and has 131,600 samples in total. It is believed to be the most applicable dataset among others. Afterward, we split the dataset, 85% for training and 15% for testing, and takes 30 minutes to complete. Surprisingly, the accuracy is slightly greater than the ByClass dataset and it is 88.2% with the test score of 0.347. The predicted outputs were also much more accurate now.

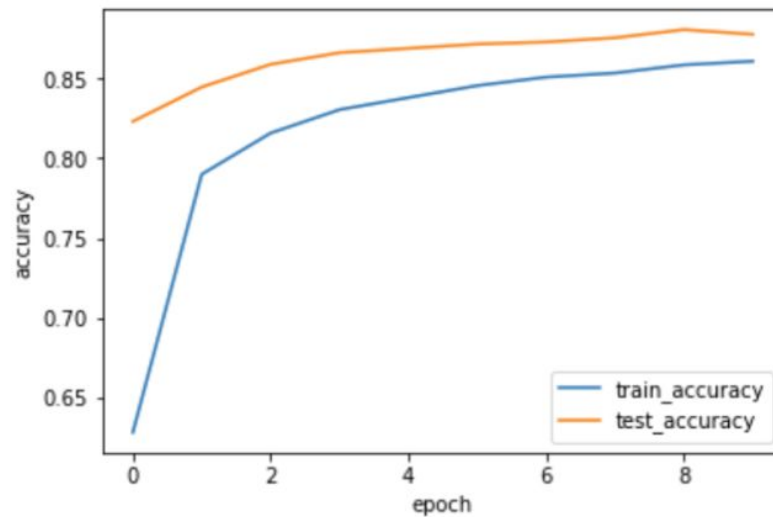
```
112800/112800 [=====] - 272s 2ms/step - loss: 0.3176 - acc: 0.8847 - val_loss: 0.3473 - val_acc: 0.8816
Test score: 0.3472998789460101
Test accuracy: 0.8816489361702128

Process finished with exit code 0
```

However, there is a slight problem with the current model. It still hard to differentiate characters with similar features. Take a look at examples below, O was mistaken to be 0 because they look almost the same and in the right picture, the small z and n were mistaken to be capital Z and N.



We also tried increasing the number of epochs for better accuracy. Unfortunately, there was not much difference in accuracy and based from an article related to EMNIST classification, the writer provides a graph, pointing out that the right tail of the line graph is becoming smooth or will become more stagnant even if the number of epoch increases.

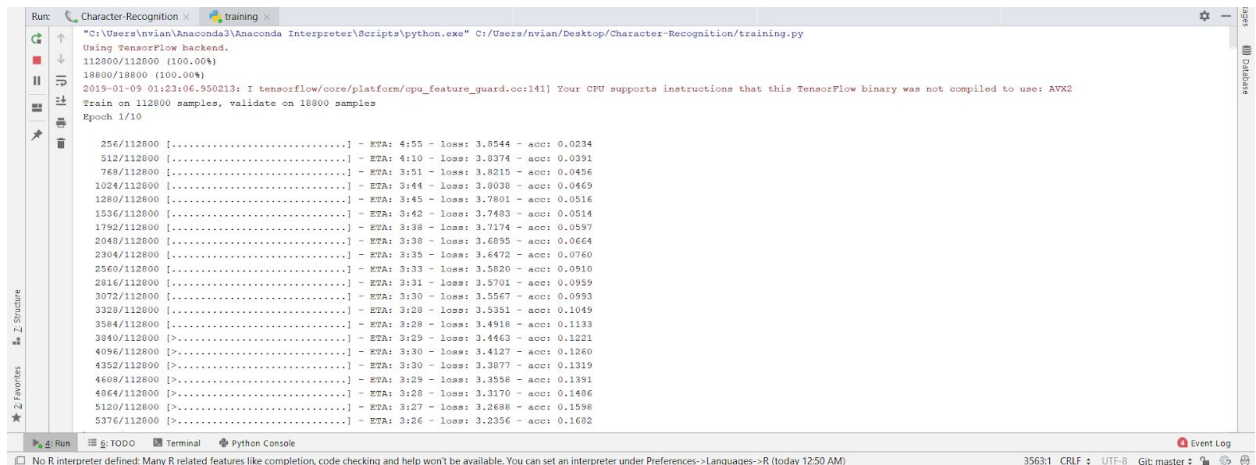


Epoch vs Accuracy Graph. Retrieved from https://medium.com/@siyao_sui/emnist-classification-with-convolutional-neural-network-5f0da8eed3c4

5. Program Manual

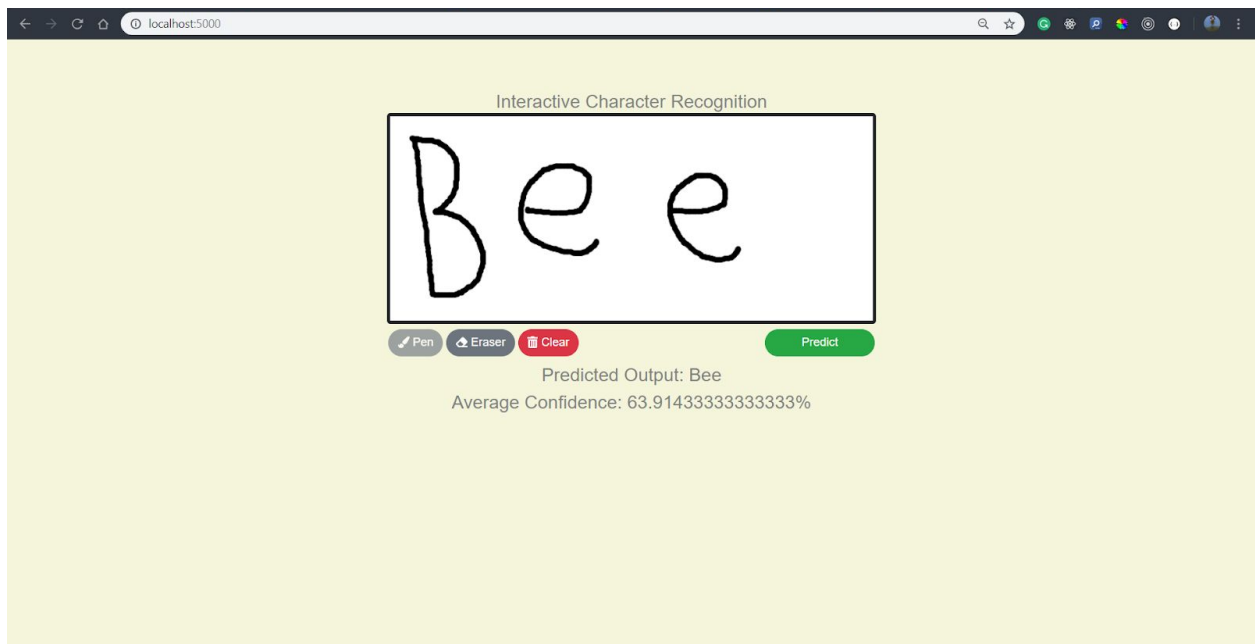
Follow the steps below to use our program:

1. Download the EMNIST dataset first from [here](#).
2. Download all dependencies for this project: Keras, tensorflow-gpu, Flask, OpenCV, etc.
3. Create a folder named *samples* and put the downloaded dataset in this folder.
4. Run *training.py* file to start training the dataset and create the model for image prediction.



```
Run: Character-Recognition x training x
"C:\Users\nvian\Anaconda3\Anaconda Interpreter\Scripts\python.exe" C:\Users\nvian\Desktop\Character-Recognition\training.py
Using TensorFlow backend.
112800/112800 (100.00%)
18800/18800 (100.00%)
2019-01-09 01:23:06.550213: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that this TensorFlow binary was not compiled to use: AVX2
Train on 112800 samples, validate on 18800 samples
Epoch 1/10
256/112800 [.....] - ETA: 4:55 - loss: 3.8544 - acc: 0.0234
512/112800 [.....] - ETA: 4:10 - loss: 3.8374 - acc: 0.0391
768/112800 [.....] - ETA: 3:51 - loss: 3.8215 - acc: 0.0456
1024/112800 [.....] - ETA: 3:44 - loss: 3.8038 - acc: 0.0469
1280/112800 [.....] - ETA: 3:45 - loss: 3.7801 - acc: 0.0516
1536/112800 [.....] - ETA: 3:42 - loss: 3.7483 - acc: 0.0514
1792/112800 [.....] - ETA: 3:38 - loss: 3.7174 - acc: 0.0597
2048/112800 [.....] - ETA: 3:38 - loss: 3.6695 - acc: 0.0664
2304/112800 [.....] - ETA: 3:35 - loss: 3.6472 - acc: 0.0740
2560/112800 [.....] - ETA: 3:33 - loss: 3.5820 - acc: 0.0910
2816/112800 [.....] - ETA: 3:31 - loss: 3.5701 - acc: 0.0955
3072/112800 [.....] - ETA: 3:30 - loss: 3.5567 - acc: 0.0993
3328/112800 [.....] - ETA: 3:28 - loss: 3.5351 - acc: 0.1049
3584/112800 [.....] - ETA: 3:28 - loss: 3.4918 - acc: 0.1133
3840/112800 [.....] - ETA: 3:29 - loss: 3.4463 - acc: 0.1221
4096/112800 [.....] - ETA: 3:30 - loss: 3.4127 - acc: 0.1260
4352/112800 [.....] - ETA: 3:30 - loss: 3.3877 - acc: 0.1319
4608/112800 [.....] - ETA: 3:28 - loss: 3.3558 - acc: 0.1391
4864/112800 [.....] - ETA: 3:28 - loss: 3.3170 - acc: 0.1486
5120/112800 [.....] - ETA: 3:27 - loss: 3.2688 - acc: 0.1598
5376/112800 [.....] - ETA: 3:26 - loss: 3.2356 - acc: 0.1682
```

5. Run *main.py* and start drawing!



6. Project Documentation

Project's repository : <https://github.com/nvianiltr/Character-Recognition> (Character Recognition)

Project's demo video : https://youtu.be/Akr_xPrxak0 (Interactive Character Recognition - Binus Int. Intelligent Systems 2019)