Optical Character Recognition Mechanics

First A. Lo Pak Ki Kapakki[27]
[1]*CCC Kei Yuen College*
**Contact: lokapakki@gmail.com*

Abstract— **Optical Character Recognition (OCR) is the process that converts an image with text into a machine-readable text format. This project is mainly for educational purposes, and is to recreate existing OCR models using neural network frameworks such as TensorFlow or Pytorch.**

I. Introduction

OCR (Optical Character Recognition) is the process that converts an image with text into a machine-readable text format, such as a word document, or a txt file. OCR has been around since 1914, and can read characters by using selenium photosensors to detect black print and then convert them into audible output. This technology has been used to convert analog characters into digital signals, thus allowing technological advancements such as picture translation, physical e-readers, etc.

II. Importance of OCR

Data preservation is one of the major implications of Optical Character Recognition. Recorded human history spans over 5500 years. It is until the recent 2-3 decades are when data is preserved in the forms of 1s and 0s. As such, it is impossible to hand-document human history by copying data from stone walls to the keyboard by hand.

With the help of OCR, machines can automate the process of converting handwritten characters into digital copies, omitting this repetitive task of storing data in binary form. Storing data in such manner can bring benefits to our society, such as allowing us to search for data and citations, increase data efficiency, and more. This signifies the importance of the ability to convert handwritten characters into digital copies.

III. Objective of this Project

The objective of the project is to explore how modern OCR works, and how we can implement it via different neural network libraries, such as TensorFlow. In this project, we will be going over how we can use TensorFlow to create a basic handwritten digits OCR, handwritten characters, and eventually Chinese characters as well. This project will be divided into 3 stages, which are numerical OCR, English OCR, and Chinese OCR. They will be discussed in the following paragraph.

IV. Methodologies

A. Numerical OCR

In this stage of the project, I have used TensorFlow alongside their MNIST dataset to perform this. The MNIST dataset is a part of the Keras library, which includes a large sample of handwritten numerical digits of 0-9s. This dataset can be used to train neural networks through frameworks such as TensorFlow or PyTorch. In this document, I'll be using the TensorFlow framework as an example.

i. Importing Libraries

For starters, we would need to import the necessary libraries into the program. This can be done through a simple import function. `import tensorflow as tf` `import numpy as np`. The as keywords is used as an alias and can be omitted. It is recommended to keep it as changes in the tensorflow library keyword would not affect the whole program in the future.

ii. Loading MNIST Dataset

We can then start loading the MNIST Dataset. We can download the dataset from the TensorFlow Keras library, through the `tf.keras.datasets.mnist.load_data()` function. This returns two tuples of NumPy arrays. `(x_train, y_train), (x_test, y_test)`.

x_train: uint8 NumPy array of grayscale image data with shapes (50000, 32, 32, 3), containing the training data. Pixel values range from 0 to 255.

y_train: uint8 NumPy array of labels (integers in range 0-9) with shape (50000, 1) for the training data.

x_test: uint8 NumPy array of grayscale image data with shapes (10000, 32, 32, 3), containing the test data. Pixel values range from 0 to 255.

y_test: uint8 NumPy array of labels (integers in range 0-9) with shape (10000, 1) for the test data.

iii. Creating the Labels

Afterwards, create a list of labels for the expected results. In this case, we can simply create a list of strings from 0-9. We will name this variable `mnist_labels` in this example.

iv. Feature Scaling

Before we start the training, we will need to perform feature scaling, which is a method used to normalize the range of independent variables or features of data. Or is simply terms, to map a [0, 255] domain into a [0, 1] for all the pixels. This can be done by a NumPy array manipulation. `x_train = x_train / x_train.max()` By taking the maximum value of the training data (255), we can divide each value from the training data, and convert them into a float between 0 and 1. Do the same with the x_test dataset, and this step would be complete.

v. One-Hot Vector

After performing feature scaling, we can create the one-hot vector required for the neural network. A one-hot is a group of bits among which the legal combinations of values are only those with a single high (1) bit and all the others low (0). Through the use of the built-in function `tf.keras.utils.to_categorical(y_train, num_classes)`, where num_classes is the no. of classes, or simply `len(mnist_labels)` (defined on part iii). Do the same with the y_test testing set.

vi. Reshaping Data

Before we input the data into the neural network, we would need to reshape both the training and testing set. In this case, the input shape is 28x28, we can save that as `INPUT_SHAPE = [28*28]`. We can reshape the model via this NumPy function `x_train = x_train.reshape([-1]+INPUT_SHAPE)`. Do the same with x_test.

vii. Settings up the architecture

Here comes the fun part. Before we start working, we would need to import the keras layers library from TensorFlow via `from tensorflow.keras.layers import *`. Before we start defining and compiling the model, it is recommended that we clear the previous session before defining the network. This can be done via `tf.keras.backend.clear_session()`.

viii. Inputs

To create an input for the neural network, we can define it by `tf.keras.Input(shape=INPUT_SHAPE)`. Then we can start feeding the input into other layers.

ix. Dense Layer

There are different types of layers that you can choose in your model. The first being the Dense layer. A Dense layer is the most basic layer of a neural network, which includes hidden units and an activation function. The weights of the neural networks will be automatically assigned by TensorFlow, so there is no need to assign them manually.

In TensorFlow, you can apply the Dense layer to the inputs or any other variable by using the Dense() function. For example, `x = Dense(units=100, activation='relu')(inputs)` will link the parameter of this function (inputs), into a Dense layer with 100 units and the activation function of relu. The output of the Dense layer is stored in x.

This step can be then stacked by `x2 = Dense(units=50, activation='relu')(x)` to link the output of the previous Dense layer to a new Dense layer.

x. Output layer

The output layer is basically a Dense layer with the number of hidden units equal to the number of classes. This can be defined as so: `outputs = Dense(units=num_classes, activation='softmax')(x)`, where softmax is a function which normalizes the outputs scores into probabilities, such that the sum of all output elements would be 1.

xi. Defining the Model

To tell the program which variable to treat as the inputs and outputs, we can use `tf.keras.Model(inputs=inputs, outputs=outputs, name="model_name_here")`.

xii. Adam Optimizer

Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. The hyperparameters can be set as such:

```
optimizer = tf.keras.optimizers.Adam(
        learning_rate=0.003,
        beta_1=0.9,
        beta_2=0.999,
        epsilon=1e-07
)
```
You may modify the hyperparameters to provide varying results suitable for your project.

xiii. Compiling the model

Finally we can compile the model. This can be done through the `model.compile()` function. It requires the parameters of the loss function, the optimizer, and the metrics. More parameters can be found on the TensorFlow documentations online.

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

To view and check whether your architecture is correct, you can use this function to view an overview image.

` tf.keras.utils.plot_model(model, "dnn_model.png", show_shapes=True)`

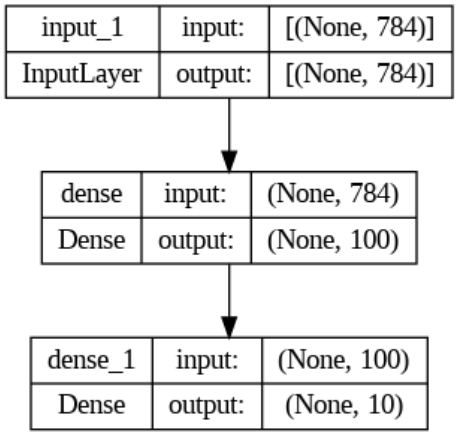

Fig. 1 A sample of how my neural network architecture is designed

xiv. Early Stopping

Before we start the training, we should define an early stopping callback function for the training, or it would run until the loss function returns an absolutely stable result, which takes a very long time.

```
es = EarlyStopping(
      monitor='val_accuracy',
      min_delta = 0,
      patience = 5,
      verbose = 1,
      mode = 'max',
      restore_best_weights = True
)
```

Details of the parameters of the function can be found here:
https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping

xv. Start Training

To start the training, we can use the model.fit() function.

```
history = model.fit(
        x_train,
        train_one_hot,
        epochs=epochs,
        batch_size=batch_size,
        validation_split=0.3,
        callbacks=[es]
)
```

xvi. Viewing the Results

To evaluate the results with the testing dataset, we can use the model.evaluate() function.

`scores = model.evaluate(x_test, test_one_hot, verbose=1)`

In my case, the model has an accuracy of 97.57%, which is not bad since it's a basic 3 layer neural network. Accuracy can be further boosted via Convolutions and Dropout layers, which will not be discussed in this paper due to time limitations. More information on this topic can be found on the TensorFlow official documentations.

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0907 - accuracy: 0.9757
```

Fig. 2 Accuracy of the project

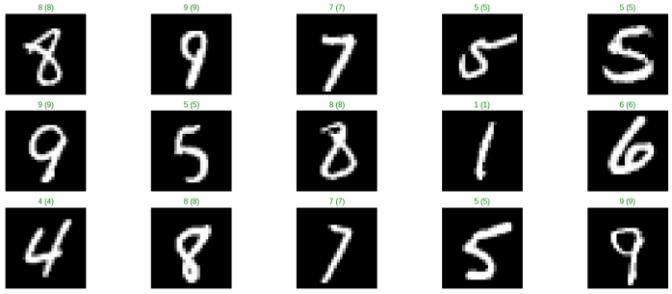

Fig. 3 Performance of the neural network

B. English OCR Models

The same methodologies applied on Part A can be used in this part, and libraries can be found on TensorFlow model zoos. This is left as an exercise for the readers. Lol.

C. Chinese OCR Models

As for Chinese OCR Models, more advanced measures are required to successfully perform this. In this paper, I'll be discussing two main methods I have at the moment.

I. ChangJie Fragments Recognition Method

i. Introduction

Dividing Chinese Characters into their ChangJie construction, and detect the presence and location of each part, and looking them up through a dictionary.

ii. Benefits

Can reduce the samples required for Chinese Characters.

iii. Disadvantages

There are conflicting characters, and some does not have a ChangJie representation.

II. Stroke Detection Method

i. Introduction

Further dividing the characters into each stroke, and caching each stroke into an array, which we can then search for on dictionaries.

ii. Benefits

This method has been proven to work by numerous papers

iii. Disadvantages

Poorly written characters may have strokes linked up at times.

Youtube Link: https://youtu.be/iD1lofecgH4
Github Link: http://github.com/Gameboy612/gef_project