Design Simple Convolution Neural Network for Digit Handwrite Detection on Mnist Dataset

Pooya Danandeh

Pooya144@gmail.com

Department of Computer Science, University of Tabriz, Tabriz, Iran

Abstract

We are always dealing with another people's handwriting. In offices, schools, shops and etc. Nowadays most of works in this world done by machines. But how can a machine deal with people handwritings? In this paper, we try to find a way to detect digits from handwritings.

Keywords: Convolution, Neural Network, Deep Learning, Handwriting detection

1 Introduction

Everyone has their own handwriting, so we cannot solve this problem by creating a simple program. We need something that it can learn, and neural networks are the best choice for this. On the other hand, we have to do processing on the images. So, Convolution Neural Networks (CNN) can help us efficiently. In the following, we will implement this neural network in Keras library on Mnist dataset.

2 Implementation

2.1 Need Tools

To implement a CNN, we need some tools we import them.

```
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.regularizers import l1, l2, l1_l2
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import numpy as np
```

2.2 Load and Preprocessing Data

In this section, we load Mnist dataset contains 60000 grayscales 28×28 images in 10 classes and its labels as training data and 10000 grayscales 28×28 images in 10 classes and its labels as test data. We load it and convert labels into categorical labels. (convert a scalar into a vector with a length of 10)

```
(images_train, labels_train), (images_test, labels_test) = mnist.load_data()
train_labels = to_categorical(labels_train)
test_labels = to_categorical(labels_test)
```

2.3 Tuning Parameters

We need to tuning parameters such as number of epochs, training batch size, learning rate and optimizer. We set

number of epochs to 6 dues to large number of images in training data, batch size to 20 and learning rate to 0.0001 for prevent overfitting and optimizer to Adam.

```
input_shape = (28, 28, 1)
number_classes = 10
batch_size = 20
epochs = 6
optimizer = Adam(learning_rate = 0.0001)
```

2.4 Designing CNN

In CNNs, every Convolution layer applies some Filters (Kernels) on an image and send them to next layer. So, we need to determine number of filters, filter (kernel) size and strides (stride of moving filter on the input image), we define 2 Convolution layers with Relu activation function and add padding. Also, we add max pooling layers after every Convolution layer to extract important features from images, and then we define Flatten layer to flatting the image's matrix and preparing it for giving it to Dense layer. And finally, we put a Dense layer with Relu activation function to learn labels and for output layer we define a Dense layer with 10 neurons and Softmax activation function. For prevent overfitting we apply L1 Regularization and Dropout on this network.

```
model = Sequential()
model.add(Conv2D(filters = 4, kernel_size = (3, 3), strides = (1, 1), padding = 'same',
activation = 'relu',input_shape = input_shape, kernel_regularizer= l1(0.001)))
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))
model.add(Conv2D(filters = 8, kernel_size = (3, 3), padding = 'same', activation = 'relu',
kernel_regularizer= l1(0.001)))
model.add(MaxPooling2D(pool_size = (2, 2), strides = (2, 2)))
model.add(Flatten())
model.add(Dense(16, activation = 'relu'))
Dropout(0.2)
model.add(Dense(number_classes, activation = 'softmax'))
```

2.5 Compile and Summary

At this point, we compile the network with Categorical Cross Entropy loss function and Adam optimizer.

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

Summary of the network:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 4)         40

 max_pooling2d (MaxPooling2D  (None, 14, 14, 4)        0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 8)         296

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 8)          0
 2D)

 flatten (Flatten)           (None, 392)               0

 dense (Dense)               (None, 16)                6288

 dense_1 (Dense)             (None, 10)                170

=================================================================
Total params: 6,794
Trainable params: 6,794
Non-trainable params: 0
_________________________________________________________________
None
```

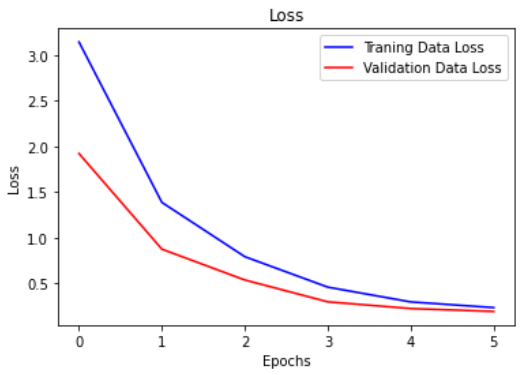
2.6 Training Network

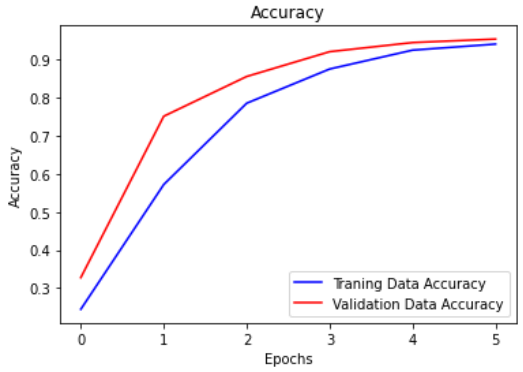
We train the network with 6 epochs, 10% validation and batch size of 20.

```
ModelHistory = model.fit(images_train, train_labels,
                         validation_split = 0.10,
                         epochs = epochs,
                         batch_size = batch_size)
```

```
Epoch 1/6
2700/2700 [==============================] - 25s 9ms/step - loss: 3.1419 - accuracy: 0.2445 - val_loss: 1.9211 - val_accuracy: 0.3277
Epoch 2/6
2700/2700 [==============================] - 25s 9ms/step - loss: 1.3858 - accuracy: 0.5722 - val_loss: 0.8743 - val_accuracy: 0.7517
Epoch 3/6
2700/2700 [==============================] - 25s 9ms/step - loss: 0.7918 - accuracy: 0.7859 - val_loss: 0.5368 - val_accuracy: 0.8562
Epoch 4/6
2700/2700 [==============================] - 25s 9ms/step - loss: 0.4583 - accuracy: 0.8758 - val_loss: 0.2977 - val_accuracy: 0.9213
Epoch 5/6
2700/2700 [==============================] - 25s 9ms/step - loss: 0.2968 - accuracy: 0.9254 - val_loss: 0.2243 - val_accuracy: 0.9453
Epoch 6/6
2700/2700 [==============================] - 26s 10ms/step - loss: 0.2346 - accuracy: 0.9413 - val_loss: 0.1926 - val_accuracy: 0.9545
```

Loss and accuracy plots during training:





2.7 Evaluate the Network

At this point, we should predict on test dataset and see the network accuracy on test data.

```
test_loss, test_acc = model.evaluate(images_test, test_labels)
print('test_acc:', test_acc)
```

Output:

```
313/313 [==============================] - 2s 7ms/step - loss: 0.2134 - accuracy: 0.9463
test_acc: 0.9463000297546387
```

3 Conclusion

We designed a CNN with almost 95% accuracy on test dataset and train dataset. You may get better results by manipulate and experiment any parameter of the network.

4 References

[1] Mnist Digits Classification Dataset

https://keras.io/api/datasets/mnist/

[2] Keras Conv2D and Convolutional Layer

https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

[3] Simple Neural Network on Mnist

https://www.kaggle.com/pooyadanandeh/keras-practice