2021-2022, Fall

Report Title: CNN architecture to classify the MNIST handwritten dataset.

Course Name: COMPUTER VISION AND PATTERN RECOGNITION

Section: B

Submitted to: DEBAJYOTI KARMAKER

Written by:
AHMED, FAISAL 18-36639-1

CNN Architecture to Classify the MNIST Handwritten Dataset

AHMED, FAISAL 18-36639-1

Department of Computer Sciences, American International University-Bangladesh

Abstract

Image processing and Deep learning are two zones of excessive awareness to researchers and scientists around the world. It is having multiple applications fields such as robotics, medicine, and security and surveillance. Deep Learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound, and text. MNIST data set is having a huge number of handwritten text data set and it is frequently used for training, testing, and validation of CNN deep model. In this article, we have created an efficient model with multiple convolutions, relu and pooling layers. Which is tested on MNIST data set with 98.45% accuracy? Further, this model is tested on similar kind of random image data set which gives significant results in terms of accuracy.

Keywords: Deep learning, Computer Vision.

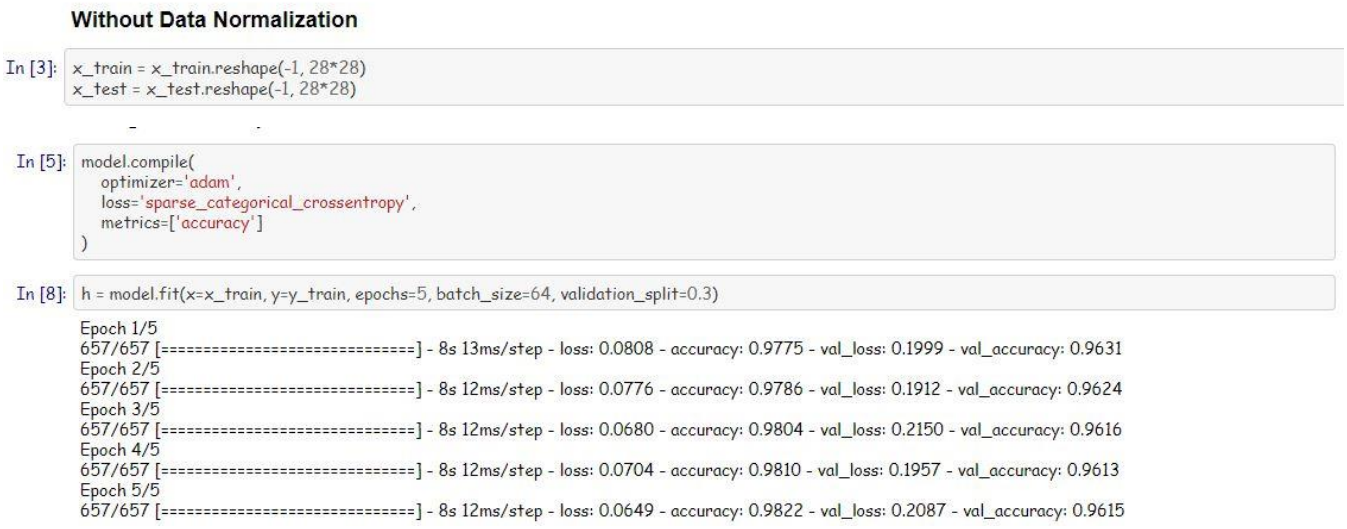
1. Introduction

Recognizing handwritten digits has been of great importance and has various uses in online handwriting recognition on 21st century smartphones and tablets, to extract postal zip-codes on mail or letterheads, processing bank check amounts, numeric entries in forms (like - tax forms) filled up by hand, or to automatically identify license plates etc. There are different challenges faced while attempting to solve this problem. The handwritten digits are generally different in strokes, size, thickness, orientation, and distance from the margins thereby increasing the complexity for recognition. Some attempts on recognizing the handwritten digits have been made using ANN , by combining SVM using rule-based reasoning , and by applying multi-column Deep Neural Networks

for Image Classification. Furthermore, Lottery Digit Recognition Based Multi - feature, is an alternate implementation of digit recognition compared to NN and SVM, where the problem is in the implementation of the number of algorithms needed to implement. User Independent Online Handwritten Digit Recognition is a locally based recognition system that characterized a digit by its stroke, where the disadvantage is the problem of classifying the strokes. Our goal was to train a model that could classify a digit based on its pattern using CNN to recognize a similar pattern of the handwritten digit. In this paper, we have proposed a novel CNN model to achieve the best performance on the handwritten digit recognition task from random images and the MNIST dataset. Images are of English handwritten digits taken from a variety of resources, normalized to 28×28 size to fit into the model. We designed our model using four convolutional layers in which after every two convolutional layers there are two max-pooling layers to extract the features of the digits in the image.

2. Result

Without Data Normalization



In [9]:
```
model.compile(
    optimizer='SGD',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

In [10]:
```
h2 = model.fit(x=x_train, y=y_train, epochs=5, batch_size=64, validation_split=0.3)
```

```
Epoch 1/5
657/657 [==============================] - 12s 14ms/step - loss: 0.5594 - accuracy: 0.9327 - val_loss: 0.3382 - val_accuracy: 0.9602
Epoch 2/5
657/657 [==============================] - 7s 11ms/step - loss: 0.0483 - accuracy: 0.9897 - val_loss: 0.2728 - val_accuracy: 0.9682
Epoch 3/5
657/657 [==============================] - 8s 12ms/step - loss: 0.0275 - accuracy: 0.9922 - val_loss: 0.7351 - val_accuracy: 0.9370
Epoch 4/5
657/657 [==============================] - 8s 12ms/step - loss: 0.0270 - accuracy: 0.9939 - val_loss: 0.3010 - val_accuracy: 0.9712
Epoch 5/5
657/657 [==============================] - 8s 12ms/step - loss: 0.0120 - accuracy: 0.9963 - val_loss: 0.2917 - val_accuracy: 0.9709
```

In [11]:
```
model.compile(
    optimizer='RMSProp',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

In [12]:
```
h3 = model.fit(x=x_train, y=y_train, epochs=5, batch_size=64, validation_split=0.3)
```

```
Epoch 1/5
657/657 [==============================] - 15s 18ms/step - loss: 0.0521 - accuracy: 0.9907 - val_loss: 0.4547 - val_accuracy: 0.9660
Epoch 2/5
657/657 [==============================] - 12s 18ms/step - loss: 0.0504 - accuracy: 0.9903 - val_loss: 0.4250 - val_accuracy: 0.9687
Epoch 3/5
657/657 [==============================] - 10s 15ms/step - loss: 0.0504 - accuracy: 0.9920 - val_loss: 0.4778 - val_accuracy: 0.9678curacy: 0.
Epoch 4/5
657/657 [==============================] - 10s 15ms/step - loss: 0.0513 - accuracy: 0.9909 - val_loss: 0.4647 - val_accuracy: 0.9664
Epoch 5/5
657/657 [==============================] - 10s 15ms/step - loss: 0.0488 - accuracy: 0.9918 - val_loss: 0.5147 - val_accuracy: 0.9639: 1s - loss: 0.0473 - accur
acy - ETA: 1s - loss: 0.0 - ETA
```

With Data Normalization

In [13]:
```
x_train = x_train.reshape(-1, 28*28).astype("float32") / 255.0
x_test = x_test.reshape(-1, 28*28).astype("float32") / 255.0
```

In [15]:
```
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

In [16]:
```
hn = model.fit(x=x_train, y=y_train, epochs=5, batch_size=64, validation_split=0.3)
```

```
Epoch 1/5
657/657 [==============================] - 11s 14ms/step - loss: 0.4730 - accuracy: 0.8635 - val_loss: 0.1487 - val_accuracy: 0.9541
Epoch 2/5
657/657 [==============================] - 8s 13ms/step - loss: 0.1111 - accuracy: 0.9671 - val_loss: 0.1299 - val_accuracy: 0.9609
Epoch 3/5
657/657 [==============================] - 9s 13ms/step - loss: 0.0624 - accuracy: 0.9803 - val_loss: 0.1141 - val_accuracy: 0.9691
Epoch 4/5
657/657 [==============================] - 9s 13ms/step - loss: 0.0440 - accuracy: 0.9862 - val_loss: 0.0969 - val_accuracy: 0.9709
Epoch 5/5
657/657 [==============================] - 9s 13ms/step - loss: 0.0336 - accuracy: 0.9896 - val_loss: 0.1053 - val_accuracy: 0.9728
```

In [17]:
```
model.compile(
   optimizer='SGD',
   loss='sparse_categorical_crossentropy',
   metrics=['accuracy']
)
```

In [18]:
```
hn2 = model.fit(x=x_train, y=y_train, epochs=5, batch_size=64, validation_split=0.3)
```

```
Epoch 1/5
657/657 [==============================] - 10s 13ms/step - loss: 0.0191 - accuracy: 0.9943 - val_loss: 0.0853 - val_accuracy: 0.9772
Epoch 2/5
657/657 [==============================] - 8s 12ms/step - loss: 0.0128 - accuracy: 0.9970 - val_loss: 0.0839 - val_accuracy: 0.9779
Epoch 3/5
657/657 [==============================] - 8s 12ms/step - loss: 0.0112 - accuracy: 0.9977 - val_loss: 0.0835 - val_accuracy: 0.9778
Epoch 4/5
657/657 [==============================] - 8s 12ms/step - loss: 0.0102 - accuracy: 0.9977 - val_loss: 0.0832 - val_accuracy: 0.9786
Epoch 5/5
657/657 [==============================] - 8s 12ms/step - loss: 0.0097 - accuracy: 0.9980 - val_loss: 0.0828 - val_accuracy: 0.9787
```

In [19]:
```
model.compile(
   optimizer='RMSProp',
   loss='sparse_categorical_crossentropy',
   metrics=['accuracy']
)
```

In [20]:
```
hn2 = model.fit(x=x_train, y=y_train, epochs=5, batch_size=64, validation_split=0.3)
```

```
Epoch 1/5
657/657 [==============================] - 14s 17ms/step - loss: 0.0255 - accuracy: 0.9918 - val_loss: 0.1141 - val_accuracy: 0.9760
Epoch 2/5
657/657 [==============================] - 11s 17ms/step - loss: 0.0185 - accuracy: 0.9940 - val_loss: 0.1169 - val_accuracy: 0.9749
Epoch 3/5
657/657 [==============================] - 11s 16ms/step - loss: 0.0135 - accuracy: 0.9962 - val_loss: 0.1267 - val_accuracy: 0.9753
Epoch 4/5
657/657 [==============================] - 10s 15ms/step - loss: 0.0099 - accuracy: 0.9967 - val_loss: 0.1414 - val_accuracy: 0.9732
Epoch 5/5
657/657 [==============================] - 12s 18ms/step - loss: 0.0093 - accuracy: 0.9966 - val_loss: 0.1733 - val_accuracy: 0.9704
```

3. Discussion

In this work, with the aim of improving the performance of handwritten digit recognition, we evaluated variants of a convolutional neural network to avoid complex pre-processing, costly feature extraction and a complex ensemble (classifier combination) approach of a traditional recognition system. Through extensive evaluation using a MNIST dataset, the present work suggests the role of various hyper-parameters. We also verified that fine tuning of hyper-parameters is essential in improving the performance of CNN architecture. We achieved a recognition rate of 99.89% with the Adam optimizer for the MNIST database, which is better than all previously reported results. The effect of increasing the number of convolutional layers in CNN architecture on the performance of handwritten digit recognition is clearly presented through the experiments.

4. References

[1] Z Selmi, M Ben Halima and A M. Alimi, "Deep Learning System for Automatic License Plate Detection and Recognition", 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), pp. 1132-1138, 2017.

Show in Context View Article Full Text: PDF (390KB) Google Scholar

[2] J Schmidhuber, "Deep Learning in Neural Networks: An Overview", Neural Networks, vol. 61, pp. 85-117, 2015.

Show in Context CrossRef Google Scholar

[3] https://github.com/jwwthu/MNIST-MIX.