Implementing a CNN architecture to classify the MNIST handwritten dataset

Hasan, Sakib (Id: 19-40013-1)

Abstract

Image classification is a major area in the field of computer vision. There are different image classification models that have been introduced to improve the recognition accuracy. In recent years, Convolutional Neural Networks (CNN) has become dominant in the field of computer vision. The CNN architecture has performed admirably in a variety of computer vision and machine learning challenges. Because of its continuous record-breaking efficiency, CNN model is widely applied in current Machine Learning applications. In this paper I have discussed how to develop a convolutional Neural Network for MNIST handwritten digit classification and evaluate the performance of the model using different optimizer.

Introduction

CNN is a type of deep learning model for processing data that has a grid pattern, such as images, which is inspired by the organization of animal visual cortex and designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns.[1] Convolutional neural networks (CNNs) are made up of three types of layers which are convolution, pooling, and fully connected layers. CNN architecture uses multiple convolution filters or kernels that run over the image and compute a dot product. Each filter extracts different features from the image.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 64)                102464
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 122,442
Trainable params: 122,442
Non-trainable params: 0
```

Figure 1: Information about the model

Information about the dataset: The MNIST dataset contains 60000 small square 28 X 28 pixel images of handwritten single digits between 0 and 9. The task is to build a CNN architecture to classify the MNIST handwritten dataset and try to achieve accuracy over 98% by modifying the model architecture and test with different optimizer.

Results:

Using Adam optimizer:

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

```
h = model.fit(x=x_train, y=y_train, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0392 - accuracy: 0.9883 - val_loss: 0.0519 - val_accuracy: 0.9846
Epoch 2/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0364 - accuracy: 0.9893 - val_loss: 0.0494 - val_accuracy: 0.9859
Epoch 3/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0350 - accuracy: 0.9893 - val_loss: 0.0491 - val_accuracy: 0.9860
Epoch 4/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0324 - accuracy: 0.9899 - val_loss: 0.0521 - val_accuracy: 0.9855
Epoch 5/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0310 - accuracy: 0.9905 - val_loss: 0.0466 - val_accuracy: 0.9862
Epoch 6/10
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0284 - accuracy: 0.9914 - val_loss: 0.0528 - val_accuracy: 0.9847
Epoch 7/10
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0268 - accuracy: 0.9920 - val_loss: 0.0472 - val_accuracy: 0.9861
Epoch 8/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0257 - accuracy: 0.9923 - val_loss: 0.0453 - val_accuracy: 0.9877
Epoch 9/10
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0236 - accuracy: 0.9929 - val_loss: 0.0491 - val_accuracy: 0.9859
Epoch 10/10
1500/1500 [==============================] - 37s 25ms/step - loss: 0.0230 - accuracy: 0.9929 - val_loss: 0.0434 - val_accuracy: 0.9879
```

Figure 2: Result of the CNN model using Adam

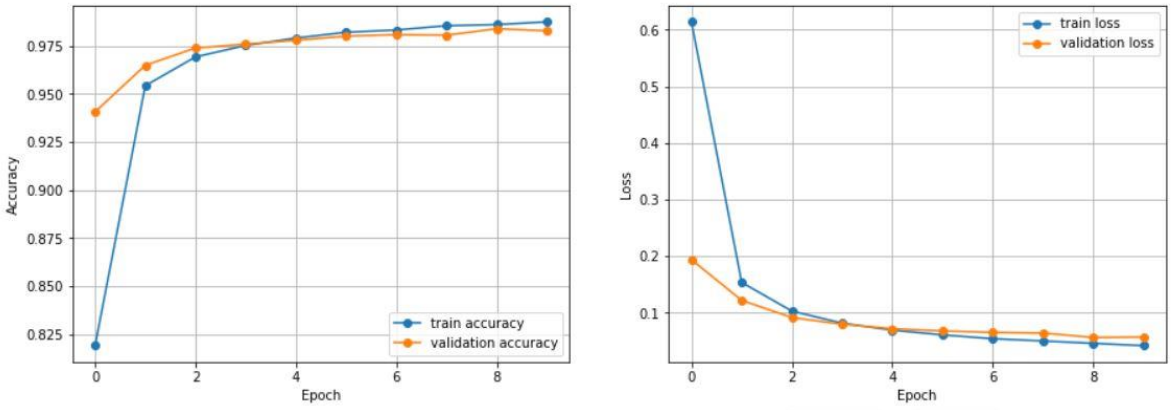

Figure 3: Accuracy and loss of the CNN model using Adam

Using SGD optimizer:

```
model.compile(
    optimizer='sgd',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

```
h = model.fit(x=x_train, y=y_train, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.6142 - accuracy: 0.8192 - val_loss: 0.1939 - val_accuracy: 0.9407
Epoch 2/10
1500/1500 [==============================] - 36s 24ms/step - loss: 0.1537 - accuracy: 0.9542 - val_loss: 0.1220 - val_accuracy: 0.9648
Epoch 3/10
1500/1500 [==============================] - 37s 25ms/step - loss: 0.1031 - accuracy: 0.9692 - val_loss: 0.0917 - val_accuracy: 0.9738
Epoch 4/10
1500/1500 [==============================] - 37s 25ms/step - loss: 0.0817 - accuracy: 0.9752 - val_loss: 0.0799 - val_accuracy: 0.9758
Epoch 5/10
1500/1500 [==============================] - 37s 25ms/step - loss: 0.0694 - accuracy: 0.9790 - val_loss: 0.0719 - val_accuracy: 0.9778
Epoch 6/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0610 - accuracy: 0.9820 - val_loss: 0.0680 - val_accuracy: 0.9800
Epoch 7/10
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0543 - accuracy: 0.9832 - val_loss: 0.0654 - val_accuracy: 0.9808
Epoch 8/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0502 - accuracy: 0.9854 - val_loss: 0.0642 - val_accuracy: 0.9805
Epoch 9/10
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0461 - accuracy: 0.9860 - val_loss: 0.0565 - val_accuracy: 0.9838
Epoch 10/10
1500/1500 [==============================] - 37s 25ms/step - loss: 0.0420 - accuracy: 0.9874 - val_loss: 0.0573 - val_accuracy: 0.9827
```

Figure 4: Result of the CNN model using SGD

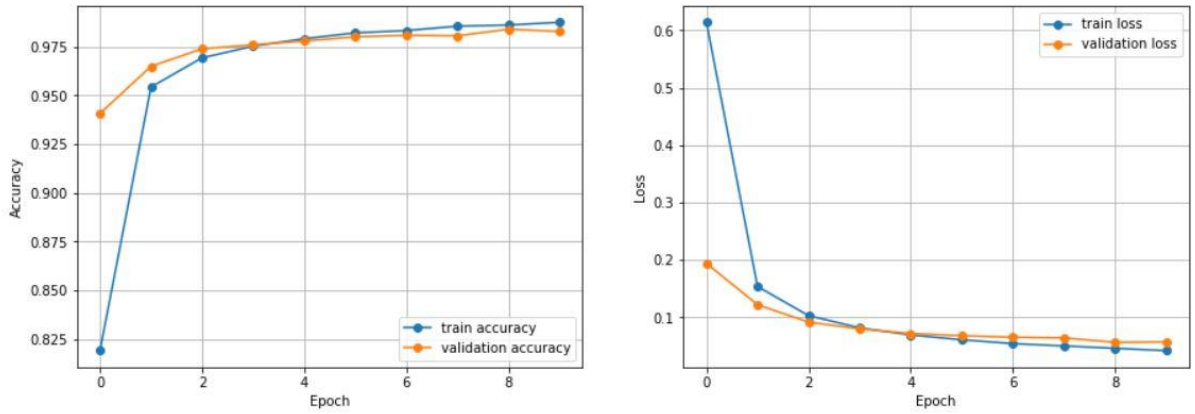

Figure 5: Accuracy and loss of the CNN model using SGD

Using RMSPROP:

```
model.compile(
    optimizer='rmsprop',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
```

```
h = model.fit(x=x_train, y=y_train, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0215 - accuracy: 0.9930 - val_loss: 0.0505 - val_accuracy: 0.9848
Epoch 2/10
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0206 - accuracy: 0.9934 - val_loss: 0.0453 - val_accuracy: 0.9872
Epoch 3/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0189 - accuracy: 0.9943 - val_loss: 0.0488 - val_accuracy: 0.9862
Epoch 4/10
1500/1500 [==============================] - 37s 25ms/step - loss: 0.0180 - accuracy: 0.9945 - val_loss: 0.0456 - val_accuracy: 0.9882
Epoch 5/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0167 - accuracy: 0.9952 - val_loss: 0.0428 - val_accuracy: 0.9877
Epoch 6/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0158 - accuracy: 0.9954 - val_loss: 0.0456 - val_accuracy: 0.9877
Epoch 7/10
1500/1500 [==============================] - 36s 24ms/step - loss: 0.0150 - accuracy: 0.9958 - val_loss: 0.0468 - val_accuracy: 0.9870
Epoch 8/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0142 - accuracy: 0.9959 - val_loss: 0.0409 - val_accuracy: 0.9888
Epoch 9/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0136 - accuracy: 0.9960 - val_loss: 0.0432 - val_accuracy: 0.9876
Epoch 10/10
1500/1500 [==============================] - 37s 24ms/step - loss: 0.0124 - accuracy: 0.9966 - val_loss: 0.0446 - val_accuracy: 0.9880
```

Figure 6: Result of the CNN model using RMSPROP

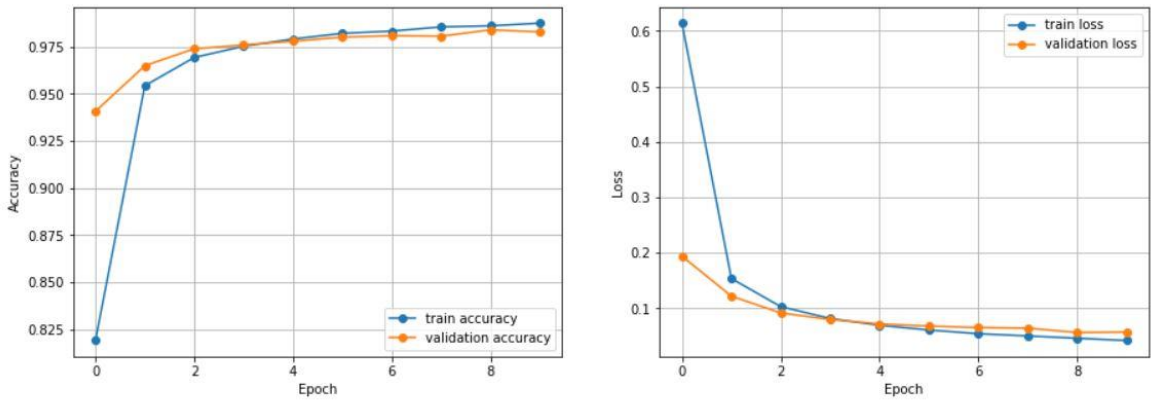

Figure 7: Result of the CNN model using RMSPROP

Summary of the result:

Optimizer	Train Accuracy	Validation Accuracy
Adam	99.29 %	98.79 %
SGD	98.74 %	98.27 %
RMSPROP	99.66 %	98.80 %

Discussion

Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate in order to reduce the losses. Optimization algorithms are responsible for reducing the losses and to provide the most accurate results as much as possible. In this report I have used Stochastic Gradient Descent (SGD), Adam and RMSPROP optimizer. Adam or Adaptive Moment Estimation works with momentums of first and second order. Stochastic Gradient Descent (SGD) tries to alter the model parameters after computation of loss on each training example.

After analyzing the result, it can be seen that all of the optimizer have performed well. First of all, Adam has been used where in last epoch the validation accuracy was 98.79% and loss was 4%. Next after using SGD optimizer, the accuracy was 98.27% and loss was 5.73%. Finally, after using RMSPROP, the accuracy was 98.80% and the loss was 4%. Here, it can be said that all three optimizers have performed effectively.

Reference:

1. Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. Insights Imaging 9, 611–629 (2018). https://doi.org/10.1007/s13244-018-0639-9