A Deep Neural Network and Convolutional Neural Network solution for MNIST dataset

Course Name: Big Data

Course Code: PM-ASDS19

Submitted to:

Dr. Md. Rezaul Karim

Associate Professor

Department of Statistics, JU

Submitted by:

Mohammad Saiduzzaman Sayed

ID: 20215063

Batch: 5th

Sec: A

Professional Masters in

Applied Statistics and Data Science (PM-ASDS)

JAHANGIRNAGAR UNIVERSITY

A Deep Neural Network and Convolutional Neural Network solution for MNIST dataset

(With different Loss Functions and different Optimization Methods)

CONTENTS

LIST OF FIGURES

Abstract

We build a classification model using the MNIST dataset to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9.

The MNIST dataset is a suitable dataset to compare the performance and accuracy of handwriting recognition methods. We use Deep Neural Network and Convolutional Neural Network to classify handwritten digit images using the MNIST dataset with 10 different labels. We achieved an accuracy of 99.34% on the test data with CNN and 97.31% with DNN.

Our neural network has hidden layers, each followed by a nonlinear ReLU activation function. A recent regularization technique, Dropout, is used to reduce overfitting. We compare the performance of different loss functions and different optimizers to train the network and prove that Adam outperforms the others.

This procedure helps us to gather knowledge about a dataset, Deep learning method, Convolutional Neural network, and steps to reach how to build a model and use it to predict.

Keyword: MNIST, DNN, CNN, tensorflow, keras

Chapter 1
Introduction

The MNIST (Modified National Institute of Standards and Technology) database is a large collection of handwritten digits. It has a training set of 60,000 examples, and a test set of 10,000 examples with grayscale images of handwritten digits between 0 to 9 each of 28x 28 pixels. It is a subset of a larger NIST Special Database.

There have been so many previous works on dealing with MNIST dataset. There are a lot of paper concerning the performance of Regression Model, CNN, DNN and other machine learning models.

Most of the work related to MNIST involves the neural network and its improvement. I rarely find someone using DNN and CNN with different loss functions and different optimization methods to select the best classification model for this dataset. So, in this report, I will try to use different loss functions and different optimization functions applied to MNIST datasets and compare their performance.

Chapter 2

Methodology

Problem Statement:

We have to build a classification model using DNN and CNN with different loss functions and different optimization methods to classify a given image of a handwritten digit into one of 10 classes representing integer values from 0 to 9 using the MNIST dataset.

Dataset Details:

Description: The MNIST dataset contains 60,000 training images and 10,000 testing images. Images format is 28x28 pixels.

Usage: MNIST

Artificial Neural Network:

The Artificial Neural Network (ANN) is a subfield of Machine Learning. It is made up of computational models inspired by the human brain as well as biological neural networks. The goal is to solve forecasting, pattern recognition, and classification problems by simulating human intelligence, reasoning, and memory.

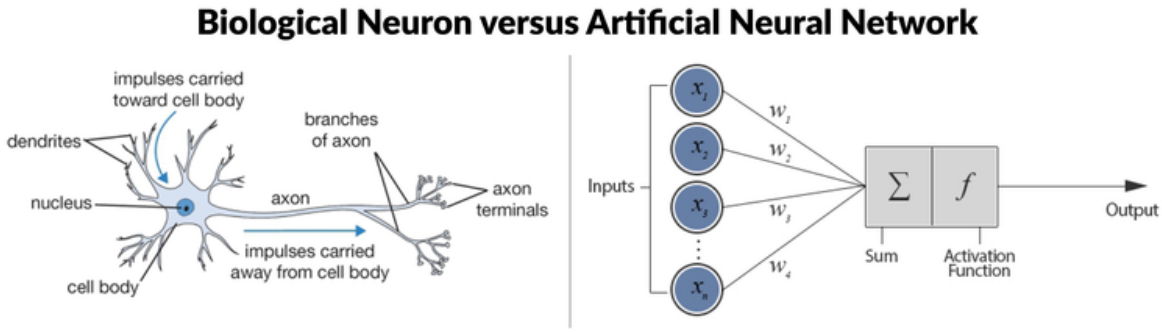


Fig 1: Biological Vs Artificial Neural Network

Network Design and Training: Deep Neural Network

A Deep Neural Network (DNN) is an Artificial Neural Network (ANN) with multiple layers between the input and output layers. There are different types of neural networks, but they always consist of the same components: neurons, synapses, weights, biases, and functions. These components functioning like the human brains and can be trained like any other ML algorithm.

Components of DNN:

~ Perceptron

~ Dense layer

~ Activation Function

~ Optimizing Weights: Loss Function, Stochastic Gradient Descent

~ Overfitting and Underfitting

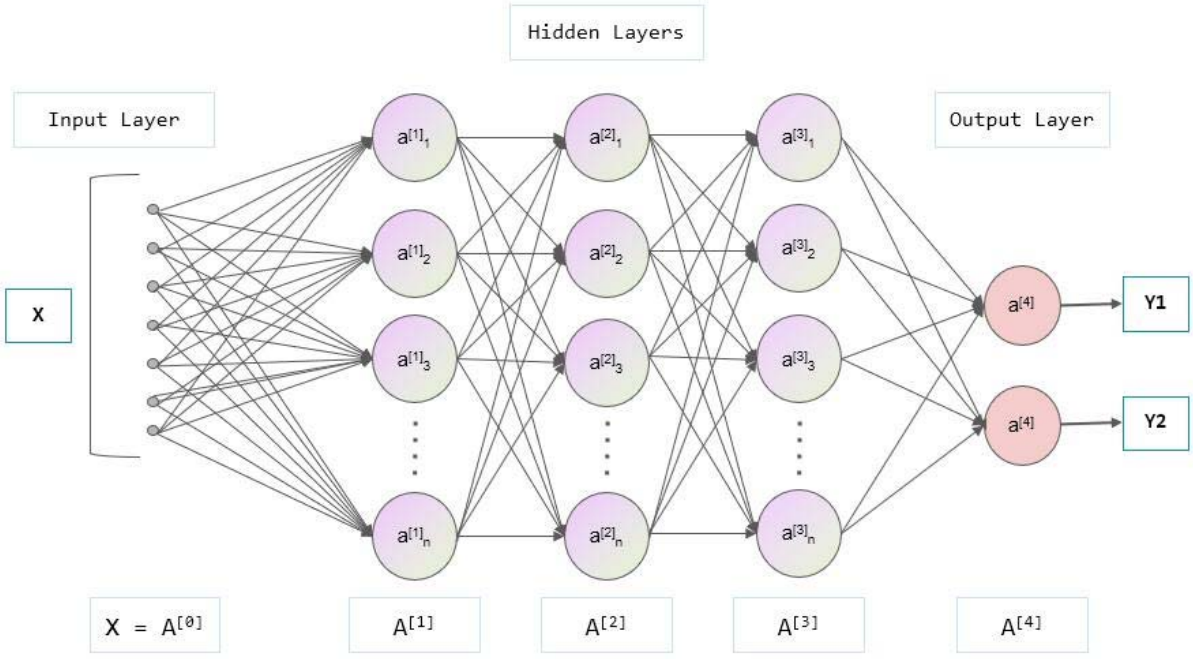


Fig 2: Overview of Deep Neural Network

Characteristics of DNN:

~ a supervised learning and based on observed functioning of human brain.
~ The NN is structured as a directed graph with many nodes (processing elements) and arcs (interconnections) between them. The nodes in the graph are like individual neurons, while the arcs are their interconnections.

Network Design and Training: Convolutional Neural Network

CNNs are based on the biological visual cortex. Small sections of cells in the cortex are responsive to certain areas of the visual field. The studies showed that some individual neurons in the brain only activated or fired in the presence of edges of a specific orientation.

Components of CNN:

~ Convolutional layers

~ ReLu layers

~ Pooling layers

~ Fully connected layers

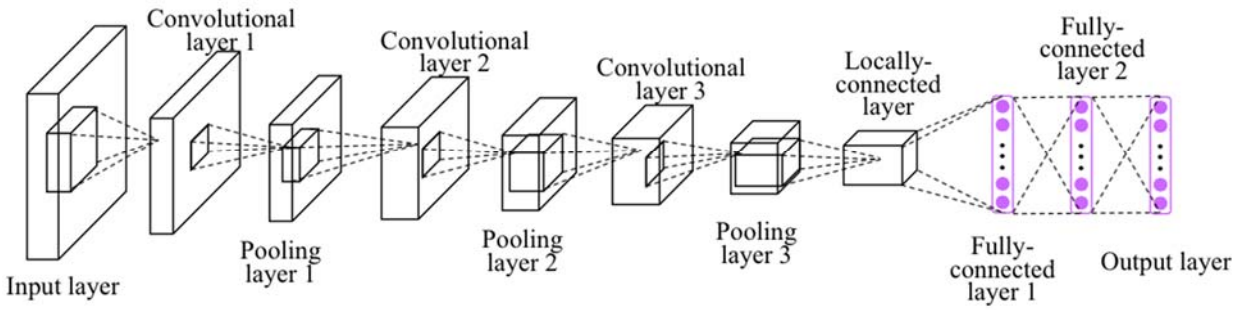


Fig 3: Overview of Convolutional Neural Network

Characteristics of CNN:

Aside from the convolution process, CNN has several distinguishing features that set it apart from other neural network systems.

~ Sparse interactions: This means that, unlike standard NNs, not every input interacts with every output, and CNN can be achieved by making the kernel much smaller than the input.

- ~ Parameter Sharing: For the full input image, parameters have the same number as the kernel(m*m).
- ~ Pooling: Basically, reduces dimensions, computation, and overfitting. Also makes the model tolerant towards small distortion and variations.
- ~ Weight: Weight of the hidden layers should be identical to each other's neighbors and except the small receptive field, the weight of others is zero.

Loss functions in Neural Network:

In a neural network, the loss function measures the difference between the expected outcome and the outcome obtained by the machine learning model. The gradients used to update the weights can be derived from the loss function. The cost is calculated by taking the average of all losses.

Different types of loss functions:

- ~ Regression Loss Functions
  1) Mean Squared Error Loss: $L = (y_i - \hat{y}_i)^2$
  2) Mean Squared Logarithmic Error Loss
  3) Mean Absolute Error Loss
- ~ Binary Classification Loss Functions
  1) Binary Cross-Entropy: $L = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)$
  2) Hinge Loss
  3) Squared Hinge Loss
- ~ Multi-Class Classification Loss Functions
  1) Sparse Multiclass Cross-Entropy Loss: $L = \sum_{j=1}^{M} y_j \log(\hat{y}_j)$
  2) Multi-Class Cross-Entropy Loss
  3) Kullback Leibler Divergence Loss

Optimization Methods in Deep Learning Algorithm:

- ~ Mini-batch GD
- ~ SGD with Momentum
- ~ Adaptive Gradient Algorithm (Adagrad)
- ~ Adaptive delta Learning Rate Method (Adadelta)
- ~ Root Mean Square Propagation (RMSprop)
- ~ Adaptive Moment Estimation (Adam)

Compactional Tools:

This whole project computing is done in Google Colaboratory and using python 3.9. The packages we used in the project are TensorFlow, Keras, Malplot, and Numpy.

Chapter 3

Data Analysis and Result

Deep Neural Network Approach:

The MNIST dataset has a training set of 60,000 examples, and a test set of 10,000 examples with grayscale images of handwritten digits between 0 to 9 each of 28x 28 pixels. After load the dataset normalizing the train and test data. To be flattened, we reshape the data.

After fit the model, the below image show that input layer shape is 784 and a hidden layer with 100 perceptron where we used ReLu activation function. The output layer has 10 perceptron, and we use SoftMax function.

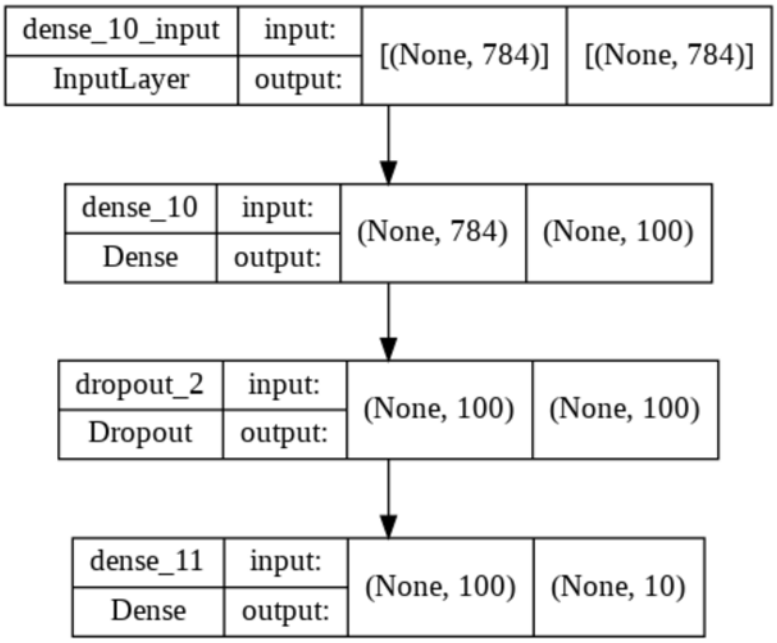


Fig 4: Layer details for DNN Model

Now, we applied different optimization method and sparse categorical loss function to compile the model.

Model Summary:

Model: "sequential_5"

_____

Layer (type) Output Shape Param #

==========

dense_10 (Dense) (None, 100) 78500
dropout_2 (Dropout) 0
dense_11 (Dense)
(None, 100)
(None, 10) 1010

==========

Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0

_____

Comparison Between Optimization Methods:

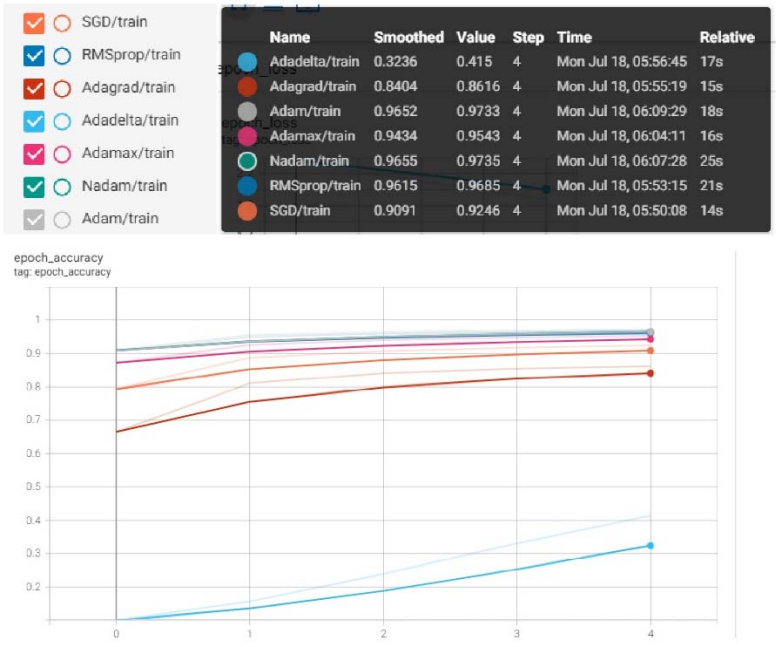


Fig 5: Accuracy on different optimization method

The above screenshot shows that, the Nadam optimization method is the best method for this data.

After 5 training epochs (1875 training iterations), the accuracy of Nadam optimizer is 97.32%.

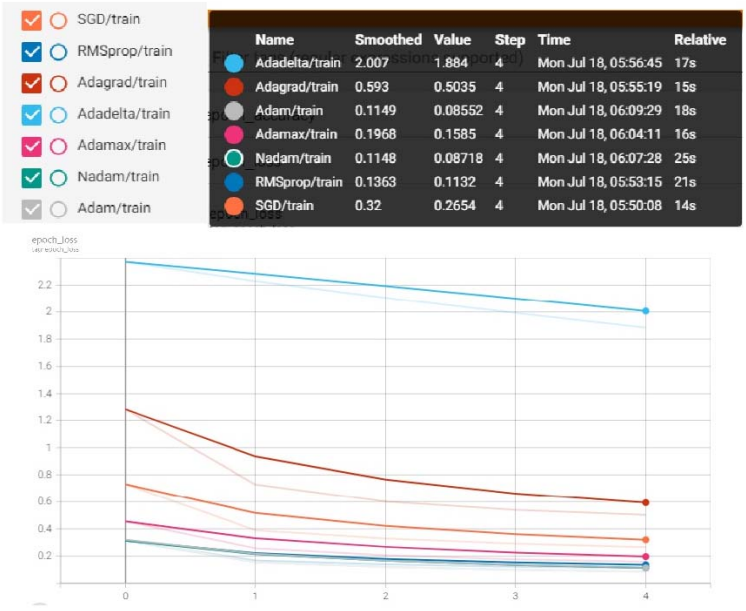


Fig 6: Loss on different optimization method

The above figure shows that after 5 training epochs (1875 training iterations), the lowest loss is 8%. The lowest loss is Nadam optimizer.

Comparison Between Loss Functions:

There are a lot of types of loss functions like regression type, binary type, and multiclass type. As our dataset has multiclass, we compared only multiclass type loss functions. Other types of loss functions are not suitable for our dataset. Using the DNN algorithm with 5 epochs and the Nadam optimization method, the accuracy of the training set and test set are given below:

Loss function	Train accuracy	Test accuracy
Mean Squared Error	10.08%	10.01%
Mean Squared Logarithmic Error	11.21%	10.84%
Mean Absolute Error	8.99%	8.5%
Sparse Multiclass Cross-Entropy	97.32%	97.31%
Kullback Leibler Divergence	9.73%	8.33%

Confusion Matrix:

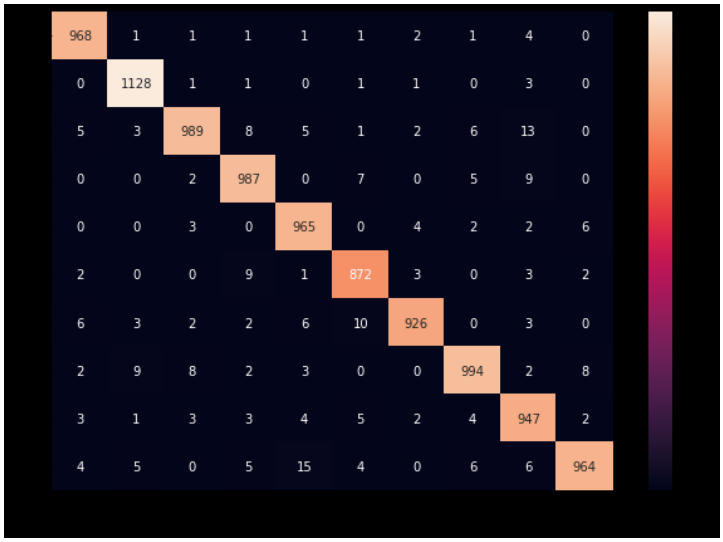


Fig 7: Confusion Matrix

Convolution Neural Network Approach:

The below image show that implements of shape of different layers.

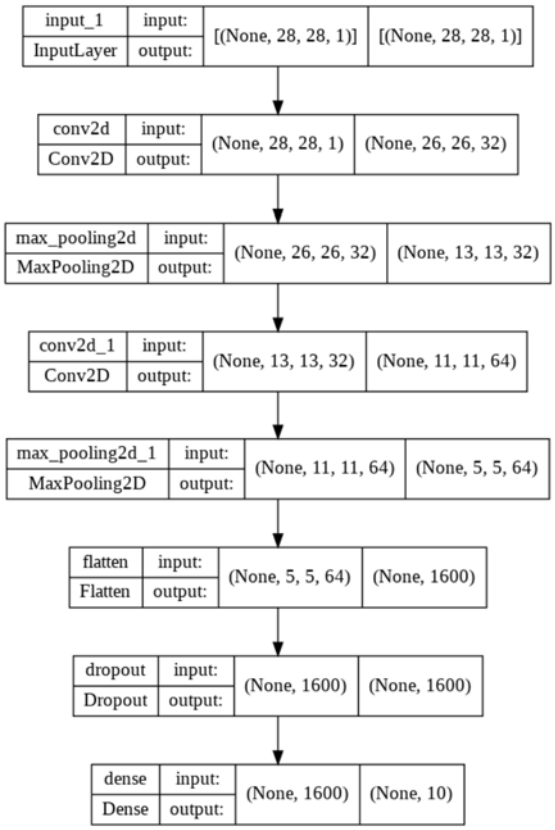


Fig 8: Layer details for CNN Model

Now, we applied different optimization method and categorical loss function to compile the model.

Model Summary:

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling 2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense (Dense)	(None, 10)	16010

Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0

Comparison Between Optimization Methods:

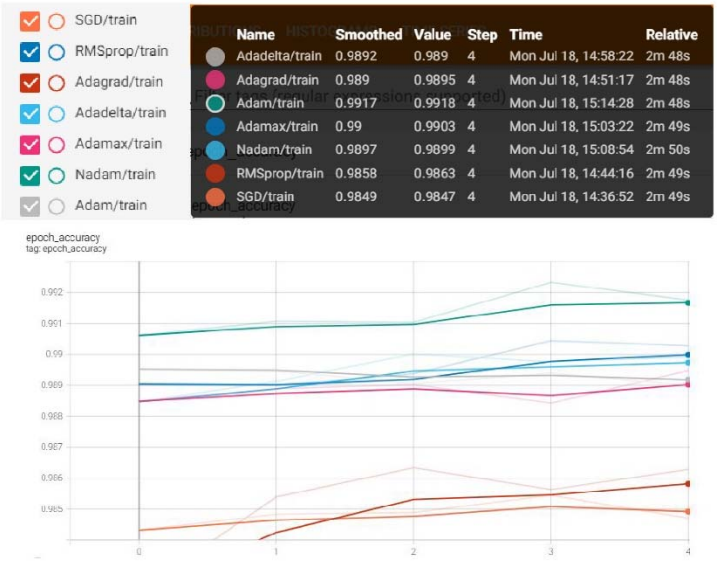


Fig 9: Accuracy on different optimization method

The above screenshot shows that, the Adam optimization method is the best method for this data. After 5 training epochs (422 training iterations), the accuracy of Adam optimizer is 99.18%.

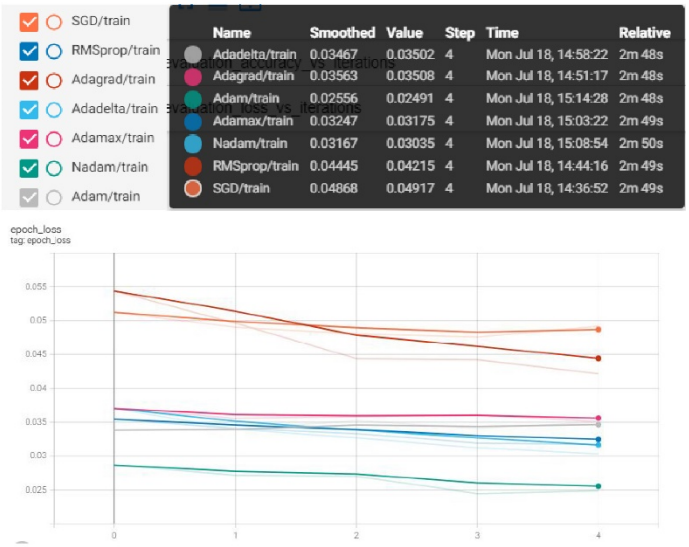


Fig 10: Loss on different optimization method

The above figure shows that after 5 training epochs (422 training iterations), the lowest loss is 2%. The lowest loss is Adam optimizer.

Comparison Between Loss Functions:

There are a lot of types of loss functions like regression type, binary type, and multiclass type. With CNN, we obtain train and test accuracy measure using Adam optimizer.

Loss function	Train accuracy	Test accuracy
Mean Squared Error	99.32%	99.26%
Mean Squared Logarithmic Error	99.37%	99.33%
Mean Absolute Error	99.37%	99.32%
Hinge Loss	99.47%	99.34%
Binary Cross-Entropy Loss	97.43%	99.32%
Kullback Leibler Divergence	99.33%	99.23%

Chapter 4

Conclusions

If we ask how accurate humans are at manual recognition, the answer is 98.3 percent since some numbers are difficult to see or misleading enough to be misread. However, DNN, CNN and some other approaches are working better than that.

Here, for DNN we obtain 97.32% on training set and 97.31% on test set accuracy using Nadam optimization method and Sparse Categorical Loss function. On the other hand, for CNN we get 99.47% on training set and 99.34% on test set accuracy using Adam optimization method and Hinge Loss Function. Hence, we can say that CNN with Adam optimizer and Hinge Loss function is best approach for MNIST data classification for given image.

However, to be sure of choosing the right approach and right model, we can try other approaches and comparison with these models. Now, we can conclude that CNN is a good approach to reducing dimensions and computation. And DNN is the based theory of CNN.

References

Nielsen, M. (2019). Neural Networks and Deep Learning. Retrieved from http://neuralnetworksanddeeplearning.com/index.html

Zhang,A. Lipton,Z.C., Li,M., Smola,J.A. (2019). Dive into Deep Learning. Retrieved from https://d2l.ai/index.html

Vafader,M. (2018). A Convolutional Neural Network solution for MNIST dataset. Retrieved from https://www.researchgate.net/publication/339439927_A_Convolutional_Neural_Network_solution_for_MNIST_dataset

Appendix

Python Code for DNN Approch

Source Code:

[1]:
```
#Library
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import random
from tensorflow import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.utils import np_utils
```

[2]:
```
#Load the dataset
(X_train, y_train), (X_test,y_test) = mnist.load_data()
```

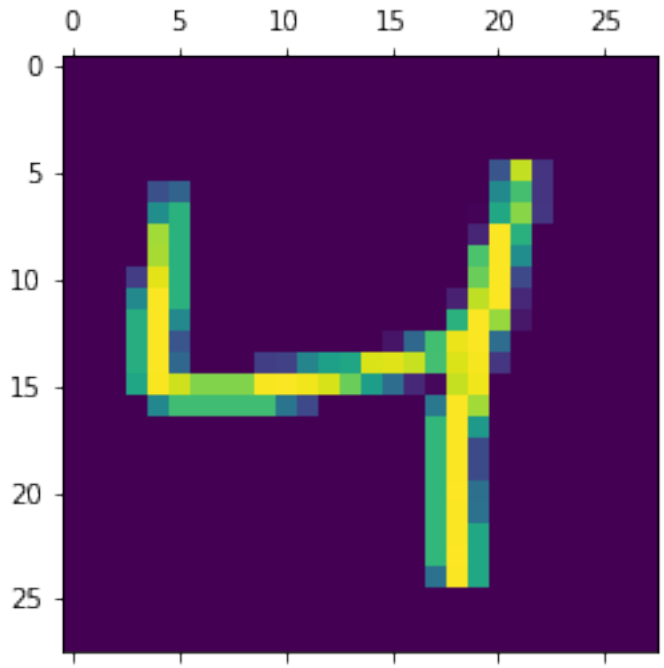
```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
```

[3]:
```
# See train and test data shape
print("Dimension of Train Data", X_train.shape)
print("Dimension of Train Data", X_test.shape)
```

```
Dimension of Train Data (60000, 28, 28)
Dimension of Train Data (10000, 28, 28)
```

[4]:
```
# Check plot
plt.matshow(X_train[2])
```

[4]:
```
<matplotlib.image.AxesImage at 0x7f842a8a80d0>
```



[5]:
```
# Normalizing the training data
X_train = X_train / 255.0
X_test = X_test / 255.0
```

[6]:
```
# Reshape data
X_train_flattened=X_train.reshape(len(X_train),28*28)
X_test_flattened=X_test.reshape(len(X_test),28*28)
```

[22]:
```
# Build model
model = Sequential()
model.add(Dense(100,input_shape=(784,),activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10,activation='softmax'))
tb = tf.keras.callbacks.TensorBoard(log_dir="logs/Nadam/", histogram_freq=1)
model.compile(optimizer='Nadam',
loss='sparse_categorical_crossentropy',
metrics=['accuracy']) #other loss functions and optimization methods applied here
model.fit(X_train_flattened,y_train,epochs=5, callbacks=[tb])
```

```
Epoch 1/5
1875/1875 [==============================] - 7s 3ms/step - loss: 0.3189 -
accuracy: 0.9081
Epoch 2/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1580 -
accuracy: 0.9528
```

```
Epoch 3/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1210 -
accuracy: 0.9632
Epoch 4/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1015 -
accuracy: 0.9689
Epoch 5/5
1875/1875 [==============================] - 6s 3ms/step - loss: 0.0876 -
accuracy: 0.9732
```

[22]:
```
<keras.callbacks.History at 0x7f84252fb950>
```

[21]:
```
%reload_ext tensorboard
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 138), started 0:19:10 ago. (Use '!kill 138' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

[23]:
```
# Evaluate the model
model.evaluate(X_test_flattened,y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0839 -
accuracy: 0.9731
```

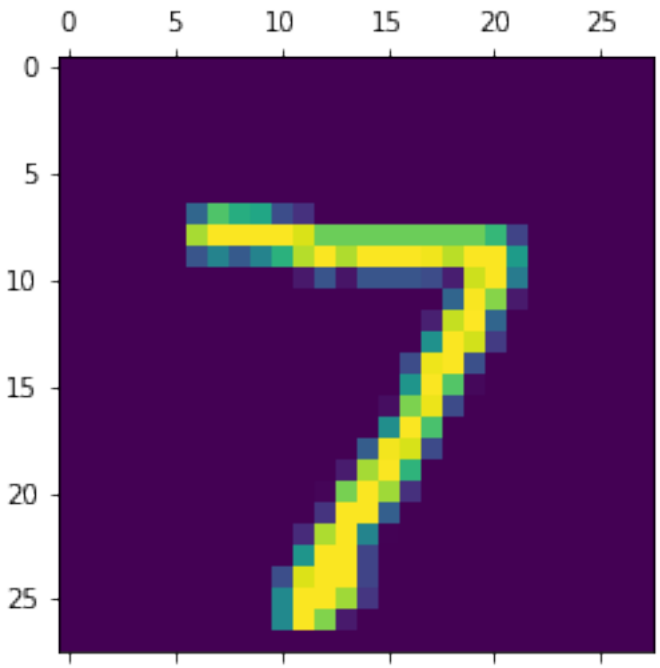
[23]:
```
[0.0839034765958786, 0.9739999771118164]
```

[24]:
```
# Model prediction
model.predict(X_test_flattened)
```

[24]:
```
array([[3.13286733e-07, 4.87760676e-09, 1.65458152e-06, …,
        9.99925256e-01, 8.14820964e-07, 3.21961397e-06],
       [3.45543039e-09, 3.01627279e-05, 9.99923229e-01, …,
        5.66496719e-13, 4.28002664e-07, 5.55026294e-12],
       [1.27446583e-07, 9.98263776e-01, 3.80544778e-04, …,
        1.09132566e-03, 1.29893408e-04, 1.23545249e-07],
       …,
       [2.34022697e-11, 1.94231764e-09, 2.68488176e-10, …,
        2.94336041e-06, 1.55640064e-05, 1.64647645e-04],
       [5.27833444e-10, 2.41321931e-08, 2.17804494e-13, …,
        1.25803326e-10, 4.43863864e-05, 1.76101689e-10],
       [3.87750880e-08, 4.23013430e-10, 1.26360055e-05, …,
        1.03887851e-12, 3.88208754e-09, 2.36749921e-12]], dtype=float32)
```

[25]:
```
# the actual test data
plt.matshow(X_test[0])
```

[25]: <matplotlib.image.AxesImage at 0x7f841f166950>



[26]:
```
# the predicted test data
y_predicted=model.predict(X_test_flattened)
```

[27]:
```
# the predicted value
np.argmax(y_predicted[0])
```

[27]: 7

[28]:
```
# predicted value
y_predicted_labels=[np.argmax(i) for i in y_predicted]
y_predicted_labels[:5]
```

[28]: [7, 2, 1, 0, 4]

[29]:
```
# actual value
y_test[:5]
```

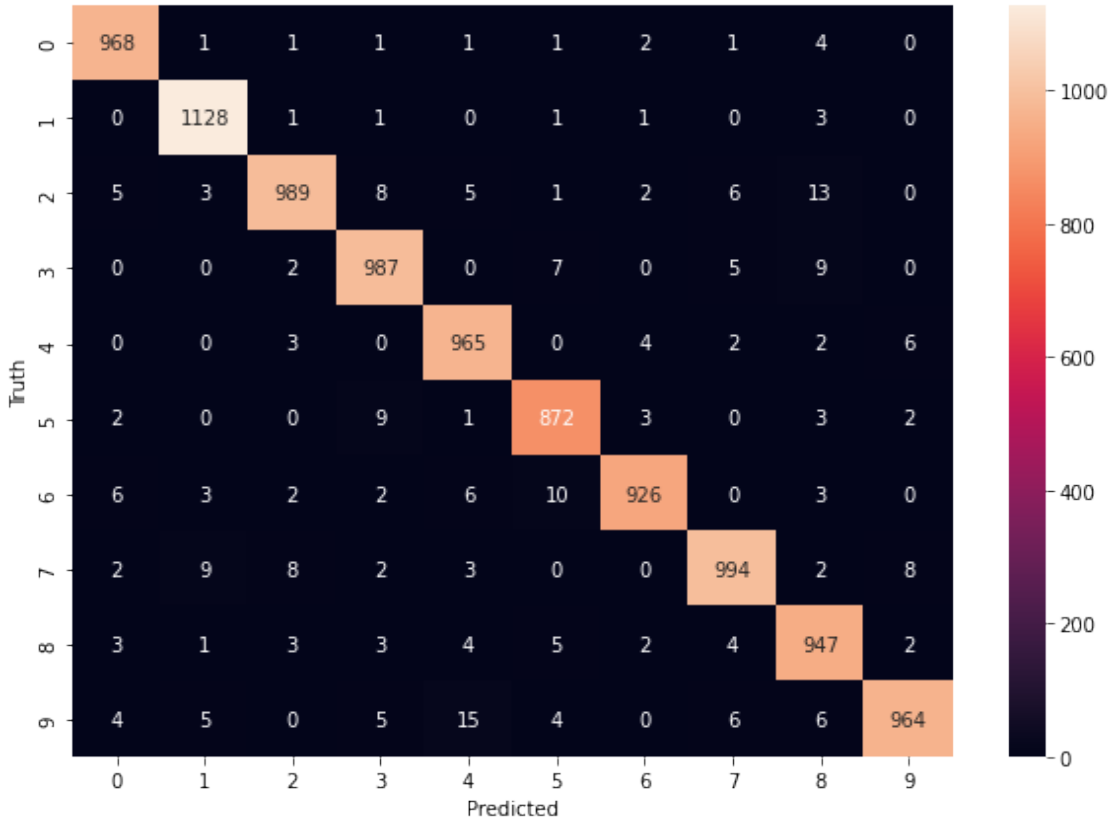
[29]: array([7, 2, 1, 0, 4], dtype=uint8)

[30]:
```
# Confusion Matrix
cm=tf.math.confusion_matrix(labels=y_test,predictions=y_predicted_labels)
cm
```

[30]:
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[ 968,    1,    1,    1,    1,    1,    2,    1,    4,    0],
       [   0, 1128,    1,    1,    0,    1,    1,    0,    3,    0],
       [   5,    3,  989,    8,    5,    1,    2,    6,   13,    0],
       [   0,    0,    2,  987,    0,    7,    0,    5,    9,    0],
       [   0,    0,    3,    0,  965,    0,    4,    2,    2,    6],
       [   2,    0,    0,    9,    1,  872,    3,    0,    3,    2],
       [   6,    3,    2,    2,    6,   10,  926,    0,    3,    0],
       [   2,    9,    8,    2,    3,    0,    0,  994,    2,    8],
       [   3,    1,    3,    3,    4,    5,    2,    4,  947,    2],
       [   4,    5,    0,    5,   15,    4,    0,    6,    6,  964]],
      dtype=int32)>
```

[31]:
```
# the heat plot
import seaborn as sn
plt.figure(figsize=(10,7))
sn.heatmap(cm,annot=True, fmt='d')
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

[31]:
```
Text(69.0, 0.5, 'Truth')
```



[32]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix,accuracy_score
y_predict=y_predicted_labels[:len(y_test)]
print('Classification Report:\n',classification_report(y_test,y_predict))
print('Confusion Matrix:\n',confusion_matrix(y_test, y_predict))
print('Accuracy Score:',accuracy_score(y_test,y_predict))
```

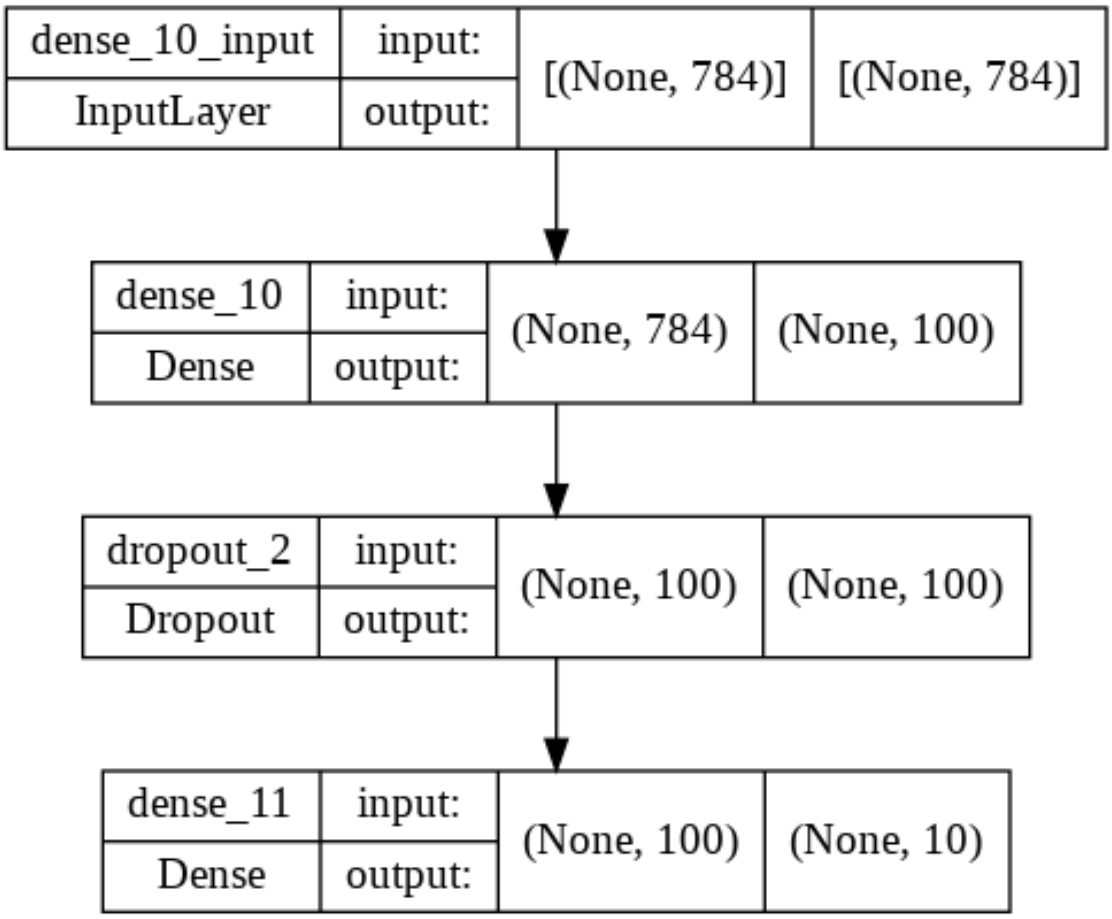
```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.98       980
           1       0.98      0.99      0.99      1135
           2       0.98      0.96      0.97      1032
           3       0.97      0.98      0.97      1010
           4       0.96      0.98      0.97       982
           5       0.97      0.98      0.97       892
           6       0.99      0.97      0.98       958
           7       0.98      0.97      0.97      1028
           8       0.95      0.97      0.96       974
           9       0.98      0.96      0.97      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000

Confusion Matrix:
 [[ 968    1    1    1    1    1    2    1    4    0]
 [   0 1128    1    1    0    1    1    0    3    0]
 [   5    3  989    8    5    1    2    6   13    0]
 [   0    0    2  987    0    7    0    5    9    0]
 [   0    0    3    0  965    0    4    2    2    6]
 [   2    0    0    9    1  872    3    0    3    2]
 [   6    3    2    2    6   10  926    0    3    0]
 [   2    9    8    2    3    0    0  994    2    8]
 [   3    1    3    3    4    5    2    4  947    2]
 [   4    5    0    5   15    4    0    6    6  964]]
Accuracy Score: 0.9731
```

[33]:
```
model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 dense_10 (Dense)             (None, 100)               78500

 dropout_2 (Dropout)          (None, 100)               0

 dense_11 (Dense)             (None, 10)                1010

=================================================================
Total params: 79,510
Trainable params: 79,510
Non-trainable params: 0
_________________________________________________________________
```

[34]:
```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

[34]:



Python Code for CNN Approch

[4]:
```
# Library
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

[2]:
```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# Load the data and split it between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")


# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

[3]:
```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dropout (Dropout)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

[19]:
```
tb_cnn= tf.keras.callbacks.TensorBoard(log_dir="logs/Adam/", histogram_freq=1)

model.compile(loss="categorical_crossentropy", optimizer="Adam",␣
↪metrics=["accuracy"]) #other loss functions and optimization methods applied here
```

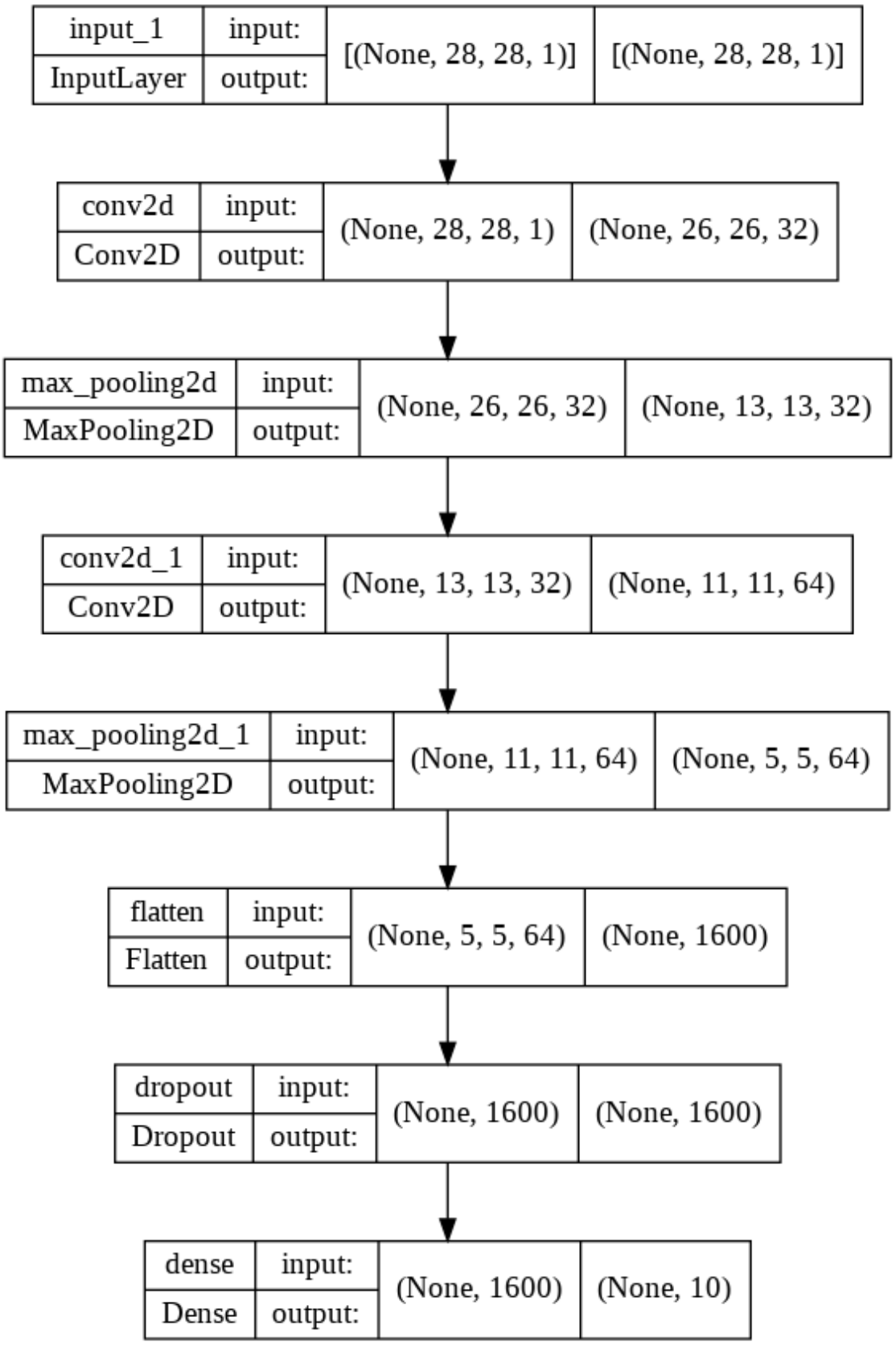
```
model.fit(x_train, y_train, batch_size=128, epochs=5, validation_split=0.1,␣
 ↪callbacks=[tb_cnn])
```

```
Epoch 1/5
422/422 [==============================] - 44s 103ms/step - loss: 0.0287 -
accuracy: 0.9906 - val_loss: 0.0288 - val_accuracy: 0.9923
Epoch 2/5
422/422 [==============================] - 42s 99ms/step - loss: 0.0272 -
accuracy: 0.9911 - val_loss: 0.0275 - val_accuracy: 0.9918
Epoch 3/5
422/422 [==============================] - 42s 99ms/step - loss: 0.0270 -
accuracy: 0.9910 - val_loss: 0.0279 - val_accuracy: 0.9913
Epoch 4/5
422/422 [==============================] - 43s 102ms/step - loss: 0.0245 -
accuracy: 0.9923 - val_loss: 0.0287 - val_accuracy: 0.9933
Epoch 5/5
422/422 [==============================] - 42s 99ms/step - loss: 0.0249 -
accuracy: 0.9918 - val_loss: 0.0274 - val_accuracy: 0.9932
```

[19]:
```
<keras.callbacks.History at 0x7f71e2652290>
```

[23]:
```
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='model_plot.png', show_shapes=True,␣
 ↪show_layer_names=True)
```

[23]:


input_1
InputLayer
input:
output:
[(None, 28, 28, 1)]
[(None, 28, 28, 1)]
conv2d
Conv2D
input:
output:
(None, 28, 28, 1)
(None, 26, 26, 32)
max_pooling2d
MaxPooling2D
input:
output:
(None, 26, 26, 32)
(None, 13, 13, 32)
conv2d_1
Conv2D
input:
output:
(None, 13, 13, 32)
(None, 11, 11, 64)
max_pooling2d_1
MaxPooling2D
input:
output:
(None, 11, 11, 64)
(None, 5, 5, 64)
flatten
Flatten
input:
output:
(None, 5, 5, 64)
(None, 1600)
dropout
Dropout
input:
output:
(None, 1600)
(None, 1600)
dense
Dense
input:
output:
(None, 1600)
(None, 10)

[22]:
```
%reload_ext tensorboard
%tensorboard --logdir logs
```

```
Reusing TensorBoard on port 6006 (pid 520), started 0:57:19 ago. (Use '!kill 520' to kill it.)
```

```
<IPython.core.display.Javascript object>
```

[45]:
```
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.021228782832622528
Test accuracy: 0.9930999875068665
```

THE END