DL_Practical_3

November 12, 2025

Assignment No: 03

Aim: To Implement Image classification model using CNN Deep Learning Architecture.

Problem Statement: Build the Image classification model using CNN Deep Learning Architecture by dividing the model into following 4 stages: a. Loading and preprocessing the image data b. Defining the model's architecture c. Training the model d. Estimating the model's performance

[1]:

```python
# Assignment 3: Image Classification using CNN on MNIST

# a) Loading and preprocessing the image data

import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

print("Imports success!\n")

# Load the MNIST dataset
with np.load(r"C:\Users\kusha\Desktop\mnist_dataset.npz") as data:
    x_train = data["X_train"]
    y_train = data["y_train"]
    x_test  = data["X_test"]
    y_test  = data["y_test"]

print(
    f"x_train:\n"
    f"  Data type: {x_train.dtype}\n"
    f"  Shape    : {x_train.shape}\n"
    f"  Pixel value range: {x_train.min()} to {x_train.max()}\n"
)
print(
    f"y_train:\n"
    f"  Data type: {y_train.dtype}\n"
    f"  Shape    : {y_train.shape}\n"
    f"  First 10 labels: {y_train[:10]}\n"
)
```

```
print(
    f"x_test:\n"
    f"  Data type: {x_test.dtype}\n"
    f"  Shape    : {x_test.shape}\n"
    f"  Pixel value range: {x_test.min()} to {x_test.max()}\n"
)
print(
    f"y_test:\n"
    f"  Data type: {y_test.dtype}\n"
    f"  Shape    : {y_test.shape}\n"
    f"  First 10 labels: {y_test[:10]}\n"
)
print(f"First image sample of x_train (pixel values):\n{x_train[0]}\n")

# --- Preprocessing ---
# Normalize images to [0, 1] float and reshape for CNN input
x_train = x_train.astype("float32") / 255.0
x_test  = x_test.astype("float32")  / 255.0

x_train = x_train[..., None]  # (samples, 28, 28, 1)
x_test  = x_test[..., None]

print(
    f"\nAfter normalization and reshaping for CNN:\n"
    f"x_train:\n"
    f"  Data type: {x_train.dtype}\n"
    f"  Shape    : {x_train.shape}\n"
    f"  Pixel value range: {x_train.min()} to {x_train.max()}\n"
    f"y_train:\n"
    f"  Data type: {y_train.dtype}\n"
    f"  Shape    : {y_train.shape}\n"
)

print(f"  Sample normalized image of x_train (first image pixels):\n{x_train[0].
 ↪squeeze()}\n")
```

```
Imports success!

x_train:
  Data type: uint8
  Shape    : (60000, 28, 28)
  Pixel value range: 0 to 255

y_train:
  Data type: uint8
  Shape    : (60000,)
  First 10 labels: [5 0 4 1 9 2 1 3 1 4]
```

```
x_test:
  Data type: uint8
  Shape    : (10000, 28, 28)
  Pixel value range: 0 to 255

y_test:
  Data type: uint8
  Shape    : (10000,)
  First 10 labels: [7 2 1 0 4 1 4 9 5 9]

First image sample of x_train (pixel values):
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3  18  18  18 126 136
  175  26 166 255 247 127   0   0   0   0]
 [  0   0   0   0   0   0   0   0  30  36  94 154 170 253 253 253 253 253
  225 172 253 242 195  64   0   0   0   0]
 [  0   0   0   0   0   0   0  49 238 253 253 253 253 253 253 253 253 251
   93  82  82  56  39   0   0   0   0   0]
 [  0   0   0   0   0   0   0  18 219 253 253 253 253 253 198 182 247 241
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  80 156 107 253 253 205  11   0  43 154
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0  14   1 154 253  90   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0 139 253 190   2   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  11 190 253  70   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  35 241 225 160 108   1
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0  81 240 253 253 119
   25   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  45 186 253 253
  150  27   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  16  93 252
  253 187   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 249
  253 249  64   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0  46 130 183 253
```

```
  253 207   2   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0  39 148 229 253 253 253
  250 182   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  24 114 221 253 253 253 253 201
   78   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  23  66 213 253 253 253 253 198  81   2
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  18 171 219 253 253 253 253 195  80   9   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0  55 172 226 253 253 253 253 244 133  11   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0 136 253 253 253 212 135 132  16   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]


After normalization and reshaping for CNN:
x_train:
  Data type: float32
  Shape    : (60000, 28, 28, 1)
  Pixel value range: 0.0 to 1.0
y_train:
  Data type: uint8
  Shape    : (60000,)

  Sample normalized image of x_train (first image pixels):
[[0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.01176471 0.07058824 0.07058824 0.07058824 0.49411765 0.53333336
  0.6862745  0.10196079 0.6509804  1.         0.96862745 0.49803922
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.11764706 0.14117648 0.36862746 0.6039216
  0.6666667  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686
  0.88235295 0.6745098  0.99215686 0.9490196  0.7647059  0.2509804
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.19215687 0.93333334 0.99215686 0.99215686 0.99215686
  0.99215686 0.99215686 0.99215686 0.99215686 0.99215686 0.9843137
  0.3647059  0.32156864 0.32156864 0.21960784 0.15294118 0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.07058824 0.85882354 0.99215686 0.99215686 0.99215686
  0.99215686 0.99215686 0.7764706  0.7137255  0.96862745 0.94509804
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.3137255  0.6117647  0.41960785 0.99215686
  0.99215686 0.8039216  0.04313726 0.         0.16862746 0.6039216
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.05490196 0.00392157 0.6039216
  0.99215686 0.3529412  0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.54509807
  0.99215686 0.74509805 0.00784314 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.         0.         0.04313726
  0.74509805 0.99215686 0.27450982 0.         0.         0.
  0.         0.         0.         0.         0.         0.
  0.         0.         0.         0.        ]
```

```
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.13725491 0.94509804 0.88235295 0.627451   0.42352942 0.00392157
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.31764707 0.9411765  0.99215686 0.99215686 0.46666667
 0.09803922 0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.1764706  0.7294118  0.99215686 0.99215686
 0.5882353  0.10588235 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.0627451  0.3647059  0.9882353
 0.99215686 0.73333335 0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.9764706
 0.99215686 0.9764706  0.2509804  0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.18039216 0.50980395 0.7176471  0.99215686
 0.99215686 0.8117647  0.00784314 0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.15294118 0.5803922  0.8980392  0.99215686 0.99215686 0.99215686
 0.98039216 0.7137255  0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.09411765 0.44705883
 0.8666667  0.99215686 0.99215686 0.99215686 0.99215686 0.7882353
 0.30588236 0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.09019608 0.25882354 0.8352941  0.99215686
 0.99215686 0.99215686 0.99215686 0.7764706  0.31764707 0.00784314
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.07058824 0.67058825 0.85882354 0.99215686 0.99215686 0.99215686
 0.99215686 0.7647059  0.3137255  0.03529412 0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.21568628 0.6745098
 0.8862745  0.99215686 0.99215686 0.99215686 0.99215686 0.95686275
 0.52156866 0.04313726 0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.53333336 0.99215686
 0.99215686 0.99215686 0.83137256 0.5294118  0.5176471  0.0627451
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]
[0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.         0.         0.
 0.         0.         0.         0.        ]]
```

[2]:
```
# b) Defining the model's architecture (CNN)

model = keras.Sequential([
    layers.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, kernel_size=3, activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=2),
    layers.Conv2D(64, kernel_size=3, activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=2),
    layers.Flatten(),
    layers.Dense(128, activation="relu"),
    layers.Dense(10, activation="softmax"),
])

model.summary()
```

```
Model defined. Summary:
```

Model: "sequential"

```
 Layer (type)                          Output Shape                       ␣
↪Param #

 conv2d (Conv2D)                       (None, 28, 28, 32)                     ␣
↪320

 max_pooling2d (MaxPooling2D)          (None, 14, 14, 32)                     ␣
↪  0

 conv2d_1 (Conv2D)                     (None, 14, 14, 64)                  ␣
↪18,496

 max_pooling2d_1 (MaxPooling2D)        (None, 7, 7, 64)                       ␣
↪  0

 flatten (Flatten)                     (None, 3136)                           ␣
↪  0

 dense (Dense)                         (None, 128)                        ␣
↪401,536

 dense_1 (Dense)                       (None, 10)                           ␣
↪1,290


Total params: 421,642 (1.61 MB)

Trainable params: 421,642 (1.61 MB)

Non-trainable params: 0 (0.00 B)
```

[3]:

```
# c) Training the model

model.compile(
    optimizer=keras.optimizers.SGD(learning_rate=0.01, momentum=0.9),
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"]
)

print("Training started...\n")
history = model.fit(
    x_train, y_train,
    validation_data=(x_test, y_test),
    epochs=10,
    batch_size=64,
    verbose=2
)
print("\nTraining completed!")
```

```
Training started…

Epoch 1/10
938/938 - 11s - 12ms/step - accuracy: 0.9259 - loss: 0.2468 - val_accuracy:
0.9794 - val_loss: 0.0638
Epoch 2/10
938/938 - 10s - 11ms/step - accuracy: 0.9794 - loss: 0.0671 - val_accuracy:
0.9860 - val_loss: 0.0438
Epoch 3/10
938/938 - 10s - 11ms/step - accuracy: 0.9857 - loss: 0.0464 - val_accuracy:
0.9868 - val_loss: 0.0387
Epoch 4/10
938/938 - 10s - 11ms/step - accuracy: 0.9889 - loss: 0.0350 - val_accuracy:
0.9887 - val_loss: 0.0333
Epoch 5/10
938/938 - 10s - 11ms/step - accuracy: 0.9907 - loss: 0.0291 - val_accuracy:
0.9880 - val_loss: 0.0367
Epoch 6/10
938/938 - 11s - 12ms/step - accuracy: 0.9925 - loss: 0.0235 - val_accuracy:
0.9899 - val_loss: 0.0325
Epoch 7/10
938/938 - 11s - 12ms/step - accuracy: 0.9941 - loss: 0.0187 - val_accuracy:
0.9906 - val_loss: 0.0289
Epoch 8/10
938/938 - 10s - 11ms/step - accuracy: 0.9954 - loss: 0.0157 - val_accuracy:
0.9907 - val_loss: 0.0282
Epoch 9/10
938/938 - 10s - 11ms/step - accuracy: 0.9957 - loss: 0.0133 - val_accuracy:
0.9914 - val_loss: 0.0292
Epoch 10/10
938/938 - 11s - 12ms/step - accuracy: 0.9966 - loss: 0.0110 - val_accuracy:
0.9898 - val_loss: 0.0330

Training completed!
```

[4]:

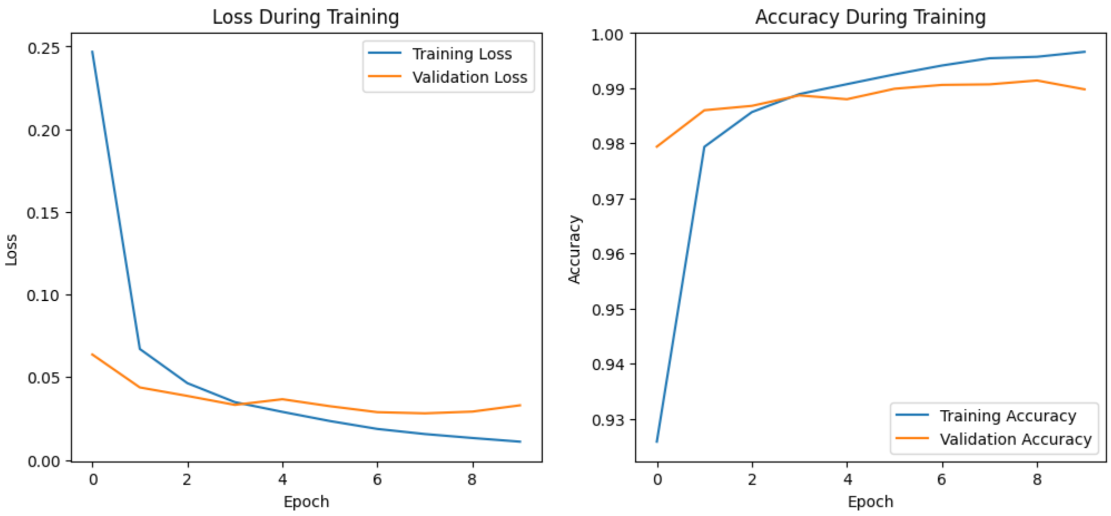
```
# d) Estimating the model's performance

test_loss, test_acc = model.evaluate(x_test, y_test, verbose=1)
print(f"\nTest accuracy: {test_acc:.4f}")
print(f"Test loss: {test_loss:.4f}")
```

```
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss During Training')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy During Training')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

```
313/313              1s 3ms/step -
accuracy: 0.9881 - loss: 0.0408

Test accuracy: 0.9898
Test loss: 0.0330
```



[5]:

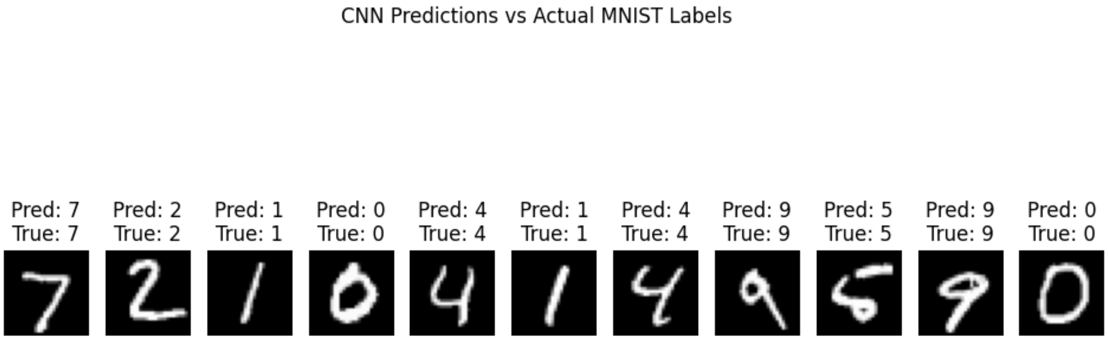
```
# Visualize predictions on test samples (first 11 digits)

count = 11
predictions = model.predict(x_test[:count])
predicted_labels = np.argmax(predictions, axis=1)

plt.figure(figsize=(12, 5))
```

```python
for i in range(count):
    plt.subplot(1, count, i + 1)
    plt.imshow(x_test[i].squeeze(), cmap='gray')
    plt.title(f"Pred: {predicted_labels[i]}\nTrue: {y_test[i]}")
    plt.axis('off')
plt.suptitle("CNN Predictions vs Actual MNIST Labels")
plt.show()
```

```
1/1               0s 82ms/step
```


CNN Predictions vs Actual MNIST Labels
Pred: 7
True: 7
Pred: 2
True: 2
Pred: 1
True: 1
Pred: 0
True: 0
Pred: 4
True: 4
Pred: 1
True: 1
Pred: 4
True: 4
Pred: 9
True: 9
Pred: 5
True: 5
Pred: 9
True: 9
Pred: 0
True: 0