```
import numpy as np # linear algebra
import pandas as pd
```

Objective

The objective of this project is to build and evaluate a Convolutional Neural Network (CNN) model to classify handwritten digits using the MNIST dataset. The aim is to achieve high accuracy in recognizing digits from 0 to 9, thereby demonstrating the effectiveness of CNNs in image recognition tasks.

This project involves several key steps:

Data Loading and Preprocessing:

Load the MNIST dataset, which contains 60,000 training images and 10,000 test images of handwritten digits. Normalize the pixel values to range between 0 and 1. Exploratory Data Analysis:

Visualize random samples from the dataset to understand the input data. Model Building:

Construct a Sequential CNN model with two convolutional layers, each followed by a max-pooling layer. Add a fully connected (dense) layer and an output layer with softmax activation to classify the digits. Model Training:

Compile the model using the Adam optimizer and categorical cross-entropy loss function. Train the model on the training data for 10 epochs, using a batch size of 64, and validate it on the test data. Model Evaluation:

Evaluate the model's performance on the test dataset to determine its accuracy and loss. Visualize the training and validation accuracy and loss over epochs to understand the model's learning curve.

Data Loading and Preprocessing:

```
import tensorflow as tf
from keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np
import os
for dirname, _, filenames in os.walk('/content/Mnist dataset.zip'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

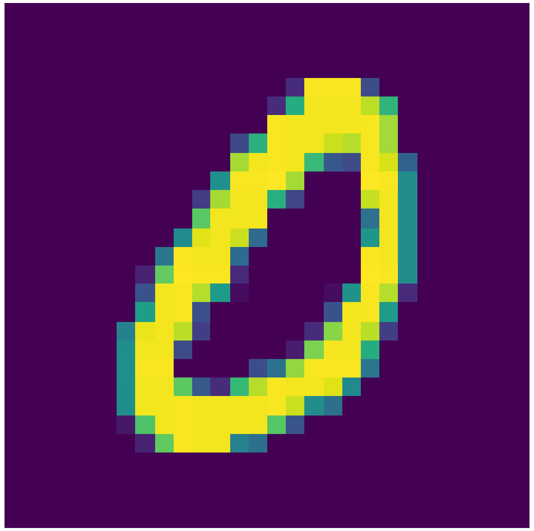
```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

Exploratory Data Analysis:

```
random_index = np.random.randint(0,len(train_images))
random_image = train_images[random_index]
random_label = train_labels[random_index]
```

```
plt.imshow(random_image)
plt.axis('off')
plt.show()
```



```
random_image.shape
```

```
(28, 28)
```

```
labels = str(list(range(1,11)))
```

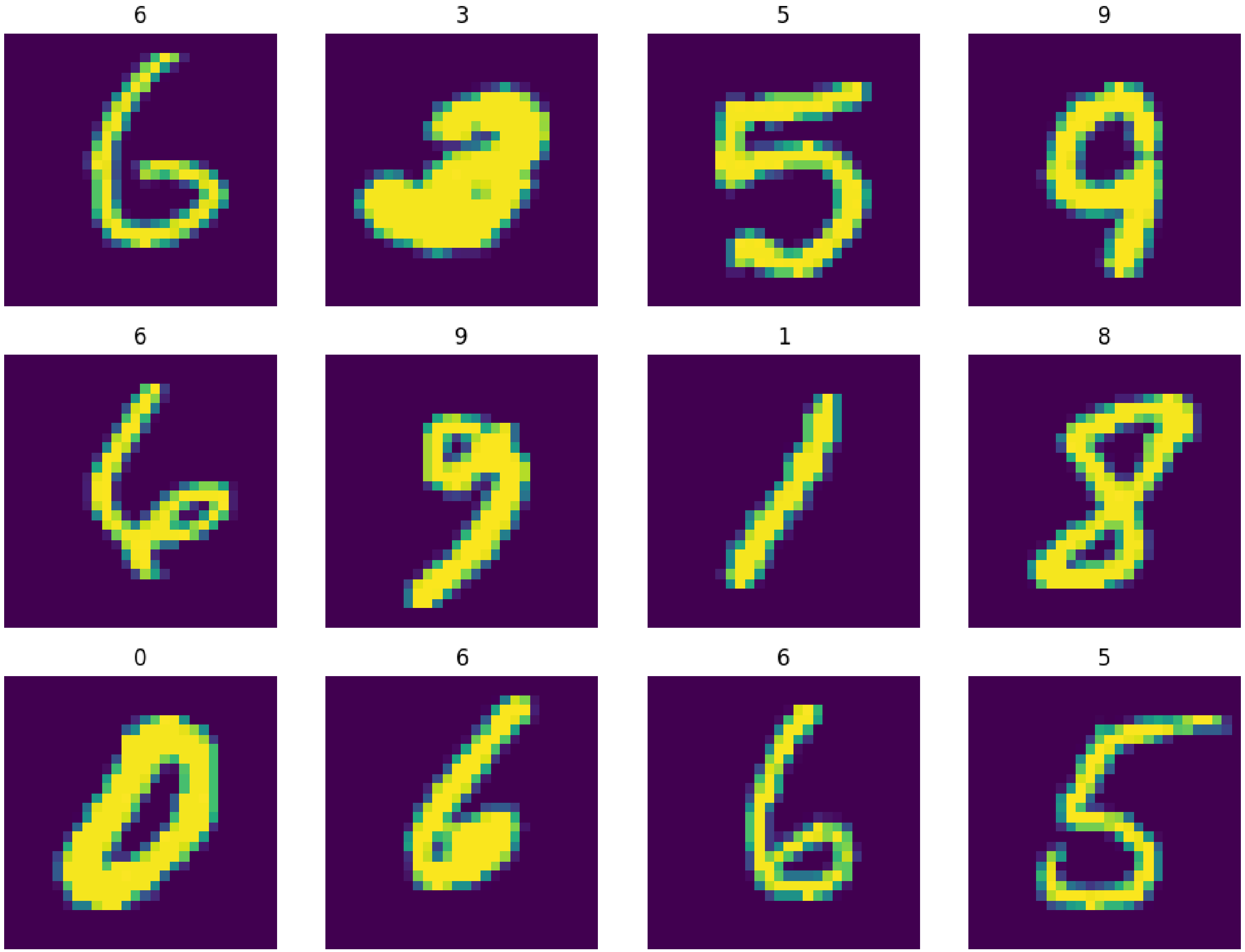
```
# Selecting 12 random images
num_samples = 12
random_indices = np.random.choice(train_images.shape[0], num_samples, replace=False)
sample_images = train_images[random_indices]
sample_labels = train_labels[random_indices]
```

```
sample_labels
```

```
array([6, 3, 5, 9, 6, 9, 1, 8, 0, 6, 6, 5], dtype=uint8)
```

```
# Plotting
plt.figure(figsize=(10, 10))
for i in range(num_samples):
    plt.subplot(4, 4, i + 1)
    plt.imshow(sample_images[i])
    # plt.title(i)


    plt.title(int(sample_labels[i]))  # Ensure sample_labels[i][0] is an index within the range of labels
    plt.axis('off')
plt.tight_layout()
plt.show()
```




```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
train_images.shape
```

```
(60000, 28, 28)
```

```
# Build Convolutional Neural Network Model
model = tf.keras.models.Sequential([
    # Ist Convolutional Layer
    tf.keras.layers.Conv2D(32, (3, 3), strides=(1, 1), activation='relu', input_shape=(28, 28, 1)),
    tf.keras.layers.MaxPooling2D((2, 2)),

    # 2nd convolutional layer
    tf.keras.layers.Conv2D(64, (3, 3), strides=(1, 1), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    # ANN
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(32, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. Wher
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

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

```
history = model.fit(train_images, train_labels, epochs=10, batch_size=64,
                    validation_data=(test_images, test_labels))
```

```
Epoch 1/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 73s 74ms/step - accuracy: 0.8670 - loss: 0.4507 - val_accuracy: 0.9760 - val_loss: 0.0718
Epoch 2/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 63s 67ms/step - accuracy: 0.9800 - loss: 0.0647 - val_accuracy: 0.9820 - val_loss: 0.0568
Epoch 3/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 76s 60ms/step - accuracy: 0.9861 - loss: 0.0447 - val_accuracy: 0.9889 - val_loss: 0.0348
Epoch 4/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 83s 62ms/step - accuracy: 0.9895 - loss: 0.0345 - val_accuracy: 0.9886 - val_loss: 0.0322
Epoch 5/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 80s 60ms/step - accuracy: 0.9925 - loss: 0.0258 - val_accuracy: 0.9883 - val_loss: 0.0374
Epoch 6/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 56s 60ms/step - accuracy: 0.9926 - loss: 0.0232 - val_accuracy: 0.9898 - val_loss: 0.0299
Epoch 7/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 65s 69ms/step - accuracy: 0.9940 - loss: 0.0176 - val_accuracy: 0.9896 - val_loss: 0.0317
Epoch 8/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 73s 60ms/step - accuracy: 0.9961 - loss: 0.0126 - val_accuracy: 0.9906 - val_loss: 0.0314
Epoch 9/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 57s 61ms/step - accuracy: 0.9960 - loss: 0.0118 - val_accuracy: 0.9909 - val_loss: 0.0311
Epoch 10/10
938/938 ━━━━━━━━━━━━━━━━━━━━ 79s 58ms/step - accuracy: 0.9968 - loss: 0.0111 - val_accuracy: 0.9907 - val_loss: 0.0338
```

Displaying the model architecture and the number of parameters.

```
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)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dense (Dense)	(None, 32)	51,232
dense_1 (Dense)	(None, 10)	330

Total params: 211,136 (824.75 KB)
Trainable params: 70,378 (274.91 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 140,758 (549.84 KB)

Model Building:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy

# Load and preprocess the dataset (using MNIST as an example)
(train_images, train_labels), (test_images, test_labels) = tf.keras.datasets.mnist.load_data()

# Normalize the images to the range [0, 1]
train_images = train_images / 255.0
test_images = test_images / 255.0

# Define the model
model = Sequential([
    Flatten(input_shape=(28, 28)),  # Flatten the input data
    Dense(64, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Compile the model
model.compile(optimizer=Adam(),
              loss=SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

# Evaluate the model
loss, accuracy = model.evaluate(train_images, train_labels)

# Print the results
print(f"The model accuracy is: {accuracy}\nThe model loss is: {loss}")
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/reshaping/flatten.py:37: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
  super().__init__(**kwargs)
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1047 - loss: 2.3639
The model accuracy is: 0.10573333501815796
The model loss is: 2.36128830909729
```

Model Evaluation:

```
loss,accuracy = model.evaluate(train_images,train_labels)
print(f"The model accuracy is : {accuracy} \n the model loss : {loss}")
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - accuracy: 0.1047 - loss: 2.3639
The model accuracy is : 0.10573333501815796
```

```
the model loss : 2.36128830909729
```

```
import matplotlib.pyplot as plt
```

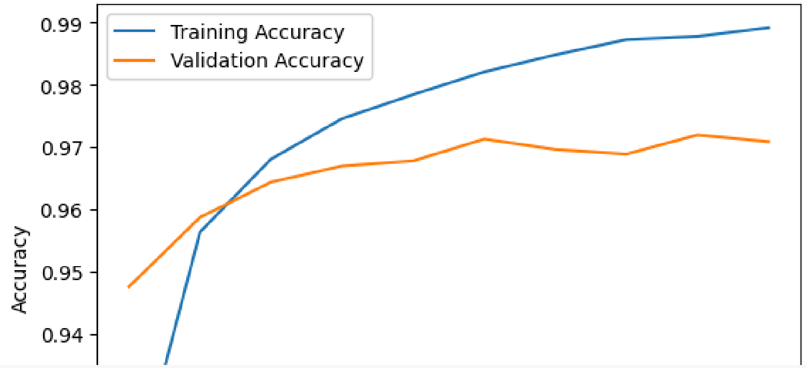
```
history = model.fit(train_images, train_labels, epochs=10, validation_split=0.2)
```

```
Epoch 1/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 7s 3ms/step - accuracy: 0.8520 - loss: 0.5176 - val_accuracy: 0.9475 - val_loss: 0.1757
Epoch 2/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9541 - loss: 0.1483 - val_accuracy: 0.9587 - val_loss: 0.1368
Epoch 3/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9667 - loss: 0.1059 - val_accuracy: 0.9643 - val_loss: 0.1264
Epoch 4/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9750 - loss: 0.0806 - val_accuracy: 0.9669 - val_loss: 0.1137
Epoch 5/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9789 - loss: 0.0671 - val_accuracy: 0.9678 - val_loss: 0.1134
Epoch 6/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9836 - loss: 0.0526 - val_accuracy: 0.9712 - val_loss: 0.1017
Epoch 7/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9864 - loss: 0.0417 - val_accuracy: 0.9696 - val_loss: 0.1077
Epoch 8/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9876 - loss: 0.0382 - val_accuracy: 0.9688 - val_loss: 0.1139
Epoch 9/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9895 - loss: 0.0344 - val_accuracy: 0.9719 - val_loss: 0.1064
Epoch 10/10
1500/1500 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9909 - loss: 0.0283 - val_accuracy: 0.9708 - val_loss: 0.1123
```

Visualization of Training History:

```
# Grapgh

plt.plot(history.history['accuracy'],label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label = 'Validation Accuracy')
plt.xlabel("Epochs")
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
plt.plot(history.history['loss'],label='Training Loss')
plt.plot(history.history['val_loss'],label='Validation Loss')
plt.xlabel("Epoche")
plt.ylabel("Loss")
plt.legend()
```