CODE:

```
#Step 1: Import libraries
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
from sklearn.metrics import classification_report, confusion_matrix
import itertools

# Step 2: Load MNIST dataset
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
print("Training data shape:", x_train.shape)  # (60000, 28, 28)
print("Test data shape:", x_test.shape)      # (10000, 28, 28)

# Step 3: Preprocessing
x_train = x_train.reshape(-1, 28*28).astype("float32") / 255.0
x_test = x_test.reshape(-1, 28*28).astype("float32") / 255.0
# Split validation set from training data
x_val = x_train[-10000:]
y_val = y_train[-10000:]
x_train = x_train[:-10000]
y_train = y_train[:-10000]

# Step 4: Build the neural network
model = keras.Sequential([
    layers.Input(shape=(28*28,)),          # input layer (784 features)
    layers.Dense(128, activation='relu'),   # hidden layer 1
    layers.Dropout(0.2),
    layers.Dense(64, activation='relu'),    # hidden layer 2
    layers.Dropout(0.2),
```

```
    layers.Dense(10, activation='softmax')  # output layer (10 classes)
])
# Step 5: Compile the model
model.compile(
    optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy']
)
# Show model summary
model.summary()

# Step 6: Train the model
history = model.fit(
    x_train, y_train,
    validation_data=(x_val, y_val),
    epochs=15,
    batch_size=128,
    verbose=2
)

# Step 7: Evaluate on test data
loss, acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\n✅ Test accuracy: {acc:.4f}")

# Step 8: Make predictions
y_probs = model.predict(x_test)
y_pred = np.argmax(y_probs, axis=1)

# Step 9: Visualize predictions
num_samples = 12
indices = np.random.choice(len(x_test), num_samples, replace=False)
plt.figure(figsize=(12,6))
```

```python
for i, idx in enumerate(indices):
    plt.subplot(3,4,i+1)
    plt.imshow(x_test[idx].reshape(28,28), cmap='gray')
    plt.title(f"T:{y_test[idx]} P:{y_pred[idx]}")
    plt.axis("off")
plt.tight_layout()
plt.show()
```

CODE:

```
# Step 1: Import required libraries

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from sklearn.metrics import classification_report, confusion_matrix


# Step 2: Load the MNIST dataset

(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

print("Raw shapes:", x_train.shape, y_train.shape, x_test.shape, y_test.shape)

# (60000, 28, 28) (60000,) (10000, 28, 28) (10000,)


# Step 3: Normalize and reshape the image data

x_train = x_train.astype("float32") / 255.0

x_test  = x_test.astype("float32")  / 255.0

# Add channel dimension

x_train = np.expand_dims(x_train, -1)   # shape -> (60000, 28, 28, 1)

x_test  = np.expand_dims(x_test, -1)    # shape -> (10000, 28, 28, 1)


# Step 4: Convert labels to one-hot encoded vectors

num_classes = 10

y_train_cat = keras.utils.to_categorical(y_train, num_classes)

y_test_cat  = keras.utils.to_categorical(y_test,  num_classes)

#Create a validation split (pull 10k from train for validation)

val_size = 10000

x_val = x_train[-val_size:]

y_val = y_train_cat[-val_size:]

x_train = x_train[:-val_size]

y_train_cat = y_train_cat[:-val_size]
```

```python
print("Train / Val / Test shapes:", x_train.shape, y_train_cat.shape, x_val.shape, y_val.shape, x_test.shape)
# Step 5: Build CNN model
model = keras.Sequential([
    layers.Conv2D(32, kernel_size=(3,3), activation='relu', input_shape=(28,28,1)),
    layers.Conv2D(64, kernel_size=(3,3), activation='relu'),
    layers.MaxPooling2D(pool_size=(2,2)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])

# Step 6: Compile the model
model.compile(
    optimizer=keras.optimizers.Adam(),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
model.summary()

# Step 7: Train the model
epochs = 12
batch_size = 128
history = model.fit(
    x_train, y_train_cat,
    validation_data=(x_val, y_val),
    epochs=epochs,
    batch_size=batch_size,
    verbose=2
)
```

```
# Step 8: Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test_cat, verbose=2)
print(f"\nTest loss: {test_loss:.4f} — Test accuracy: {test_acc:.4f}")

# Step 9: Display training accuracy graph
plt.figure(figsize=(12,5))
plt.subplot(1,2,1)
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.xlabel('Epoch'); plt.ylabel('Loss'); plt.title('Loss'); plt.legend()
plt.subplot(1,2,2)
plt.plot(history.history['accuracy'], label='train acc')
plt.plot(history.history['val_accuracy'], label='val acc')
plt.xlabel('Epoch'); plt.ylabel('Accuracy'); plt.title('Accuracy'); plt.legend()
plt.show()

#Step 10: Make predictions
y_pred_probs = model.predict(x_test)
y_pred = np.argmax(y_pred_probs, axis=1)
y_true = y_test

# Step 11: Visualize some sample predictions
n_samples = 12
idxs = np.random.choice(len(x_test), n_samples, replace=False)
plt.figure(figsize=(12,6))
for i, idx in enumerate(idxs):
    plt.subplot(3, 4, i+1)
    plt.imshow(x_test[idx].reshape(28,28), cmap='gray')
    plt.title(f"T:{y_true[idx]}  P:{y_pred[idx]}")
    plt.axis('off')
plt.tight_layout()
plt.show()
```

CODE:

```
# Step 1: Import required libraries
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.utils import to_categorical
import matplotlib.pyplot as plt
import numpy as np

# Step 2: Download and load the CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = datasets.cifar10.load_data()

# Step 3: Visualize some sample images
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
               'Dog', 'Frog', 'Horse', 'Ship', 'Truck']
plt.figure(figsize=(10, 4))
for i in range(10):
   plt.subplot(2, 5, i+1)
   plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(x_train[i])
   plt.xlabel(class_names[int(y_train[i])])
plt.show()

# Step 4: Preprocess the data
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
# Convert class labels to one-hot encoded format
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)
```

```
# Step 5: Build CNN with BatchNorm + Dropout
model = models.Sequential([
    layers.Conv2D(32, (3,3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    layers.BatchNormalization(),
    layers.Conv2D(32, (3,3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),
    layers.Conv2D(64, (3,3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(64, (3,3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),
    layers.Conv2D(128, (3,3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.Conv2D(128, (3,3), activation='relu', padding='same'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2,2)),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.BatchNormalization(),
    layers.Dropout(0.5),
    layers.Dense(10, activation='softmax')
])

# Step 6: Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
print("y_train shape:", y_train.shape)
```

```
print("y_test shape:", y_test.shape)

# Step 7: Train the model
history = model.fit(x_train, y_train, epochs=20,
          validation_data=(x_test, y_test),
          batch_size=64)

# Step 8: Evaluate the model
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=2)
print(f"\nTest Accuracy: {test_acc:.4f}")
print(f"Test Loss: {test_loss:.4f}")

# Step 9: Perform predictions on test images
predictions = model.predict(x_test)

# Step 10: Visualize predictions
plt.figure(figsize=(10, 4))
for i in range(10):
  plt.subplot(2, 5, i+1)
  plt.xticks([])
  plt.yticks([])
  plt.grid(False)
  plt.imshow(x_test[i])
  predicted_label = np.argmax(predictions[i])
  true_label = np.argmax(y_test[i])
  color = 'green' if predicted_label == true_label else 'red'
  plt.xlabel(f"{class_names[predicted_label]}\n({class_names[true_label]})", color=color)
plt.show()
```

CODE:

```
# Step 1: Import required libraries

import tensorflow as tf

import tensorflow_datasets as tfds

import matplotlib.pyplot as plt

import numpy as np

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

from tensorflow.keras.preprocessing import image_dataset_from_directory


# Step 2: Load TensorFlow Flowers dataset

dataset, info = tfds.load('tf_flowers', with_info=True, as_supervised=True, shuffle_files=True)

train_dataset = dataset['train']


# Step 3: Split into train, validation, test

train_size = 0.7

val_size = 0.15

test_size = 0.15

train_ds = train_dataset.take(int(info.splits['train'].num_examples * train_size))

val_ds = train_dataset.skip(int(info.splits['train'].num_examples *
train_size)).take(int(info.splits['train'].num_examples * val_size))

test_ds = train_dataset.skip(int(info.splits['train'].num_examples * (train_size + val_size)))


# Step 4: Preprocessing

IMG_SIZE = 160

BATCH_SIZE = 32

def format_example(image, label):

    image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))

    image = image / 255.0

    return image, label
```

```
train_ds = train_ds.map(format_example).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
val_ds = val_ds.map(format_example).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
test_ds = test_ds.map(format_example).batch(BATCH_SIZE).prefetch(tf.data.AUTOTUNE)
class_names = info.features['label'].names
print("Class names:", class_names)

# Step 5: Build Transfer Learning Model
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(IMG_SIZE, IMG_SIZE, 3))
base_model.trainable = False  # freeze base
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dropout(0.3),
    Dense(len(class_names), activation='softmax')
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Step 6: Train the Model
history = model.fit(train_ds,
                    validation_data=val_ds,
                    epochs=5)

# Step 7: Evaluate on Test Set
loss, acc = model.evaluate(test_ds)
print(f"Test Accuracy: {acc*100:.2f}%")

# Step 8: Visualization of Training
plt.plot(history.history['accuracy'], label='Train Acc')
plt.plot(history.history['val_accuracy'], label='Val Acc')
plt.xlabel('Epoch')
```

```
plt.ylabel('Accuracy')

plt.legend()

plt.title('Training vs Validation Accuracy')

plt.show()


# Step 9: Show One Example per Class with Prediction

plt.figure(figsize=(15, 10))

shown_classes = set()

i = 1

for images, labels in test_ds.unbatch().take(500):  # check first 500 test samples

    label = labels.numpy()

    if label not in shown_classes:

        ax = plt.subplot(2, 3, i)

        plt.imshow(images.numpy())


        # Model prediction

        img_array = tf.expand_dims(images, 0)  # add batch dimension

        predictions = model.predict(img_array, verbose=0)

        pred_label = np.argmax(predictions[0])


        plt.title(f"True: {class_names[label]}\nPred: {class_names[pred_label]}")

        plt.axis("off")


        shown_classes.add(label)

        i += 1

        if len(shown_classes) == len(class_names):

            break

plt.show()
```

CODE:

```
# Step 1: Import dependencies
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers, Sequential
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.metrics import classification_report, accuracy_score

# Step 2: Download and Load Dataset (IMDB Reviews)
# Keep only top 10,000 most frequent words
num_words = 10000
maxlen = 200  # max review length
print("Loading dataset...")
(X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=num_words)
print(f"Training samples: {len(X_train)}, Test samples: {len(X_test)}")
# Pad sequences to ensure equal length
X_train = pad_sequences(X_train, maxlen=maxlen)
X_test = pad_sequences(X_test, maxlen=maxlen)
print("Data after padding:", X_train.shape, X_test.shape)

# Step 3: Build RNN Model
model = Sequential([
   layers.Embedding(input_dim=num_words, output_dim=128, input_length=maxlen),
   layers.SimpleRNN(128, activation="tanh", return_sequences=False),
   layers.Dense(1, activation="sigmoid")
])
model.summary()

# Step 4: Compile and Train Model
```

```
model.compile(
    optimizer="adam",
    loss="binary_crossentropy",
    metrics=["accuracy"]
)
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=5,
    batch_size=64,
    verbose=1
)

# Step 5: Perform Predictions
y_pred_probs = model.predict(X_test)
y_pred = (y_pred_probs > 0.5).astype("int32")

# Step 6: Calculate Performance Metrics
acc = accuracy_score(y_test, y_pred)
print("\n✅ Test Accuracy:", acc)
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=["Negative", "Positive"]))

# Step 7: Visualization of Training
plt.figure(figsize=(12, 5))
# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history["accuracy"], label="Train Accuracy")
plt.plot(history.history["val_accuracy"], label="Val Accuracy")
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()
```

```python
# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history["loss"], label="Train Loss")
plt.plot(history.history["val_loss"], label="Val Loss")
plt.title("Model Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

CODE:

```
# Step 1: Import dependencies
import tensorflow as tf
from tensorflow.keras.datasets import imdb
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.sequence import pad_sequences
import matplotlib.pyplot as plt

#Steo 2: Load Dataset
max_features = 10000   # Vocabulary size (top 10,000 words)
maxlen = 200          # Maximum review length
print("Loading IMDb dataset...")
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
print(f"Training samples: {len(x_train)}, Test samples: {len(x_test)}")

#Step 3: Preprocess the Dataset
print("Padding sequences...")
x_train = pad_sequences(x_train, maxlen=maxlen)
x_test = pad_sequences(x_test, maxlen=maxlen)

#Step 4: Build RNN Model
print("Building RNN model...")
model = Sequential()
model.add(Embedding(input_dim=max_features, output_dim=64, input_length=maxlen))
model.add(SimpleRNN(64))   # RNN Layer
model.add(Dense(1, activation='sigmoid'))  # Output Layer

#Step 5: Compile the Model
model.compile(optimizer='adam',
        loss='binary_crossentropy',
```

```
            metrics=['accuracy'])
print(model.summary())

#Step 6: Train the Model
print("Training the model...")
history = model.fit(x_train, y_train,
                epochs=3,
                batch_size=64,
                validation_split=0.2)

#Step 7: Evaluate the Model
print("Evaluating the model...")
loss, accuracy = model.evaluate(x_test, y_test, verbose=0)
print(f"\nTest Accuracy: {accuracy:.4f}")
print(f"Test Loss: {loss:.4f}")

#Step 8: Predict Sentiment for New Inputs
word_index = imdb.get_word_index()
# Function to decode review back to text
reverse_word_index = {value: key for key, value in word_index.items()}
def decode_review(encoded_review):
   return " ".join([reverse_word_index.get(i - 3, "?") for i in encoded_review])
# Pick one review from test set
sample_review = x_test[1]
decoded = decode_review(sample_review)
print("\nSample Review (decoded):")
print(decoded)
prediction = model.predict(sample_review.reshape(1, -1))[0][0]
print(f"\nPredicted Sentiment Score: {prediction:.4f}")
print("Predicted Label:", "Positive 😊" if prediction > 0.5 else "Negative 😞")
```

```
#Step 9: Plot Training History
plt.figure(figsize=(12,5))

# Accuracy plot
plt.subplot(1,2,1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title("Model Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.legend()

# Loss plot
plt.subplot(1,2,2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title("Model Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()

plt.show()
```

CODE:

```
# Step 1: Import dependencies
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.datasets import mnist

#Steo 2: Load Dataset
(x_train, _), (x_test, _) = mnist.load_data()

#Step 3: Preprocess the Dataset
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
# Flatten 28x28 images into vectors of size 784
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
print(f"Training data shape: {x_train.shape}, Test data shape: {x_test.shape}")

#Step 4: Build Autoencoder Model
encoding_dim = 32  # Size of encoded representation (compressed feature size)
# Input placeholder
input_img = Input(shape=(784,))
# Encoder network
encoded = Dense(128, activation='relu')(input_img)
encoded = Dense(64, activation='relu')(encoded)
encoded = Dense(encoding_dim, activation='relu')(encoded)
# Decoder network
decoded = Dense(64, activation='relu')(encoded)
decoded = Dense(128, activation='relu')(decoded)
```

```
decoded = Dense(784, activation='sigmoid')(decoded)

# Autoencoder model

autoencoder = Model(input_img, decoded)

# Encoder model (for extracting compressed features)

encoder = Model(input_img, encoded)


#Step 5: Compile the Model

autoencoder.compile(optimizer='adam', loss='binary_crossentropy')


#Step 6: Train the Autoencoder

history = autoencoder.fit(x_train, x_train,

                    epochs=20,

                    batch_size=256,

                    shuffle=True,

                    validation_data=(x_test, x_test))


#Step 7: Evaluate the Model

loss = autoencoder.evaluate(x_test, x_test, verbose=0)

print(f"\nTest Reconstruction Loss: {loss:.4f}")


#Step 8: Predict (Reconstruct Images)

decoded_imgs = autoencoder.predict(x_test)


#Step 9: Visualize the Results

n = 10  # Number of digits to display

plt.figure(figsize=(20, 4))

for i in range(n):

    # Display original

    ax = plt.subplot(2, n, i + 1)

    plt.imshow(x_test[i].reshape(28, 28), cmap="gray")

    plt.title("Original")

    plt.axis("off")
```

```
    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
    plt.title("Reconstructed")
    plt.axis("off")
plt.show()
```

CODE:

```
# Step 1: Import required libraries

from ultralytics import YOLO

import cv2

import matplotlib.pyplot as plt


# Step 2: Load the pretrained YOLOv8 model (nano version for speed)

# This auto-downloads weights from Ultralytics Hub

model = YOLO("yolov3.pt")

print("✅ YOLOv3 model loaded successfully!")


# Step 3: Run object detection on a sample image

img_path = "https://ultralytics.com/images/bus.jpg"

results = model(img_path)


# Step 4: Display results

for r in results:

    annotated_img = r.plot()   # returns an annotated numpy array (BGR)

    # Save the annotated image

    cv2.imwrite("output.jpg", annotated_img)

    # Show inline (Jupyter / Colab)

    plt.imshow(cv2.cvtColor(annotated_img, cv2.COLOR_BGR2RGB))

    plt.axis("off")

    plt.show()


# Step 5: Print detected objects

for r in results:

    for box, cls, conf in zip(r.boxes.xyxy, r.boxes.cls, r.boxes.conf):

        print(f"Detected {model.names[int(cls)]} with confidence {conf:.2f}")


# Step 6: (Optional) Run on webcam
```

```
cap = cv2.VideoCapture(0)  # open webcam
while True:
    ret, frame = cap.read()
    if not ret:
        break
    results = model(frame)
    annotated_frame = results[0].plot()
    cv2.imshow("YOLOv8 Webcam Detection", annotated_frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

\

CODE:

```
# Step 1: Import dependencies
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.keras import layers

# Step 2: Load and preprocess the dataset (MNIST)
# Load MNIST dataset (28x28 grayscale images)
(X_train, _), (_, _) = tf.keras.datasets.mnist.load_data()
# Normalize images to [-1, 1] for GAN training
X_train = (X_train.astype("float32") - 127.5) / 127.5
X_train = np.expand_dims(X_train, axis=-1)  # Add channel dimension
print("Dataset shape:", X_train.shape)

# Step 3: Build Generator Model
def build_generator(latent_dim):
  model = tf.keras.Sequential([
    layers.Dense(7*7*256, use_bias=False, input_shape=(latent_dim,)),
    layers.BatchNormalization(),
    layers.LeakyReLU(),
    layers.Reshape((7, 7, 256)),
    layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding="same", use_bias=False),
    layers.BatchNormalization(),
    layers.LeakyReLU(),
    layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding="same", use_bias=False),
    layers.BatchNormalization(),
    layers.LeakyReLU(),
    layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding="same", use_bias=False, activation="tanh"),
  ])
  return model
```

```
# Step 4: Build Discriminator Model
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Conv2D(64, (5, 5), strides=(2, 2), padding="same", input_shape=[28, 28, 1]),
        layers.LeakyReLU(),
        layers.Dropout(0.3),
        layers.Conv2D(128, (5, 5), strides=(2, 2), padding="same"),
        layers.LeakyReLU(),
        layers.Dropout(0.3),
        layers.Flatten(),
        layers.Dense(1, activation="sigmoid"),
    ])
    return model

# Step 5: Compile Models
# Latent space dimension
latent_dim = 100
# Build models
generator = build_generator(latent_dim)
discriminator = build_discriminator()
# Compile discriminator
discriminator.compile(
    loss="binary_crossentropy",
    optimizer=tf.keras.optimizers.Adam(1e-4),
    metrics=["accuracy"]
)
# Build GAN (stacked model)
discriminator.trainable = False  # Freeze discriminator in GAN training
gan_input = tf.keras.Input(shape=(latent_dim,))
fake_image = generator(gan_input)
validity = discriminator(fake_image)
gan = tf.keras.Model(gan_input, validity)
```

```python
gan.compile(loss="binary_crossentropy", optimizer=tf.keras.optimizers.Adam(1e-4))

# Step 6: Train GAN
# Training parameters
epochs = 10000
batch_size = 128
sample_interval = 1000
# Labels for real and fake images
real_label = np.ones((batch_size, 1))
fake_label = np.zeros((batch_size, 1))
# Function to save generated images
def save_generated_images(epoch):
    noise = np.random.normal(0, 1, (16, latent_dim))
    gen_imgs = generator.predict(noise)
    gen_imgs = 0.5 * gen_imgs + 0.5  # Rescale to [0, 1]
    fig, axs = plt.subplots(4, 4, figsize=(4, 4))
    count = 0
    for i in range(4):
        for j in range(4):
            axs[i, j].imshow(gen_imgs[count, :, :, 0], cmap="gray")
            axs[i, j].axis("off")
            count += 1
    plt.suptitle(f"Generated Images at Epoch {epoch}")
    plt.show()
# Training loop
for epoch in range(1, epochs + 1):
    # Train Discriminator
    idx = np.random.randint(0, X_train.shape[0], batch_size)
    real_imgs = X_train[idx]
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    fake_imgs = generator.predict(noise)
```

```
    d_loss_real = discriminator.train_on_batch(real_imgs, real_label)
    d_loss_fake = discriminator.train_on_batch(fake_imgs, fake_label)
    d_loss = 0.5 * np.add(d_loss_real, d_loss_fake)
    # Train Generator
    noise = np.random.normal(0, 1, (batch_size, latent_dim))
    g_loss = gan.train_on_batch(noise, real_label)
    # Print progress
    if epoch % 100 == 0:
        print(f"{epoch} [D loss: {d_loss[0]:.4f}, acc.: {100*d_loss[1]:.2f}] [G loss: {g_loss:.4f}]")
    # Save generated images
    if epoch % sample_interval == 0:
        save_generated_images(epoch)

# Step 7: Evaluate GAN
# Generate some test images
test_noise = np.random.normal(0, 1, (10, latent_dim))
generated_images = generator.predict(test_noise)
print("Generated images shape:", generated_images.shape)
# Show one generated image
plt.imshow(generated_images[0, :, :, 0], cmap="gray")
plt.title("Sample Generated Digit")
plt.axis("off")
plt.show()
```