1. **Implement a simple neural network using tensorflow to classify the handwritten digits from MNIST dataset**

```python
# Step 1: Import Libraries
import tensorflow as tf                      # TensorFlow for machine learning
from tensorflow.keras import layers, models  # Import Keras layers and model APIs
import matplotlib.pyplot as plt              # For visualizing predictions

# Step 2: Load and Preprocess MNIST Dataset
mnist = tf.keras.datasets.mnist              # Load the built-in MNIST dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()  # Split into training and test sets

# Normalize pixel values to [0, 1] for faster training and better performance
x_train, x_test = x_train / 255.0, x_test / 255.0

# Step 3: Build the Neural Network Model
model = models.Sequential([                           # Create a sequential model
    layers.Flatten(input_shape=(28, 28)),             # Flatten 28x28 images to 784-element vectors
    layers.Dense(128, activation='relu'),             # Fully connected hidden layer with 128 units and ReLU activation
    layers.Dropout(0.2),                              # Dropout layer to prevent overfitting (drops 20% of nodes randomly)
    layers.Dense(10, activation='softmax')            # Output layer: 10 neurons (one per digit), softmax for probabilities
])

# Step 4: Compile the Model
model.compile(
    optimizer='adam',                                 # Use Adam optimizer (adaptive learning rate)
    loss='sparse_categorical_crossentropy',           # Suitable for integer labels (0-9)
    metrics=['accuracy']                              # Track accuracy during training and testing
)

# Step 5: Train the Model
model.fit(
    x_train, y_train,                                 # Input and labels
    epochs=5,                                         # Number of training passes over the data
    validation_split=0.1                              # 10% of training data is used for validation
)

# Step 6: Evaluate on Test Data
test_loss, test_acc = model.evaluate(x_test, y_test)   # Evaluate on unseen test data
print('\nTest accuracy:', test_acc)                   # Print the final test accuracy

# Optional: Step 7 - Plot Predictions
predictions = model.predict(x_test)                   # Predict class probabilities for each test image

# Define a function to display a prediction
def plot_image(i, predictions_array, true_label, img):
    plt.grid(False)                                   # Remove grid lines
    plt.xticks([])                                    # Remove x-axis ticks
    plt.yticks([])                                    # Remove y-axis ticks
    plt.imshow(img, cmap=plt.cm.binary)               # Show image in grayscale

    predicted_label = tf.argmax(predictions_array)    # Get predicted digit (highest probability)
    color = 'blue' if predicted_label == true_label else 'red'  # Blue if correct, red if wrong

    plt.xlabel(f"Pred: {predicted_label} (True: {true_label})", color=color)  # Add label with prediction

# Display the first 10 test images with predictions
plt.figure(figsize=(10, 5))                           # Set figure size
for i in range(10):
    plt.subplot(2, 5, i + 1)                          # 2 rows × 5 columns of subplots
    plot_image(i, predictions[i], y_test[i], x_test[i])  # Show image and prediction
plt.tight_layout()                                    # Adjust layout to avoid overlap
plt.show()                                            # Display the full plot
```

2.Train a convolutional neural network to classify dogs and cats using kaggle dataset

```python
# Step 1: Import required libraries
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Step 2: Preprocess the images (rescale pixel values to [0, 1])
train_gen = ImageDataGenerator(rescale=1./255)
val_gen = ImageDataGenerator(rescale=1./255)

# Step 3: Load the training and validation images
train_data = train_gen.flow_from_directory(
    'data/train',              # Folder should contain 'cats' and 'dogs' subfolders
    target_size=(150, 150),    # Resize all images to 150x150
    batch_size=20,             # Process 20 images at a time
    class_mode='binary'        # 0 = cat, 1 = dog
)

val_data = val_gen.flow_from_directory(
    'data/validation',         # Validation folder with same structure
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)

# Step 4: Build the CNN model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2, 2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Binary output
])

# Step 5: Compile the model
model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=['accuracy']
)

# Step 6: Train the model
model.fit(
    train_data,
    epochs=5,
    validation_data=val_data
)
```

3.fine tune a pre trained CNN model Model V66 RESNET or mobile net on small dataset for a specific classification tasks

```python
# Import required libraries
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Step 1: Preprocess image data
# Rescale pixel values to [0, 1]
train_gen = ImageDataGenerator(rescale=1./255)
val_gen = ImageDataGenerator(rescale=1./255)

# Load training images from folders
train_data = train_gen.flow_from_directory(
    'data/train',              # Folder with class subfolders (e.g., cats, dogs)
    target_size=(160, 160),    # Resize images
    batch_size=32,
    class_mode='binary'        # For 2-class classification
)

# Load validation images
val_data = val_gen.flow_from_directory(
    'data/validation',
    target_size=(160, 160),
    batch_size=32,
    class_mode='binary'
)

# Step 2: Load pre-trained MobileNetV2 (without top layer)
base_model = tf.keras.applications.MobileNetV2(
    input_shape=(160, 160, 3),
    include_top=False,
    weights='imagenet'
)
base_model.trainable = False  # Freeze the base model layers

# Step 3: Add custom classification layers on top
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),     # Reduces tensor shape
    tf.keras.layers.Dense(1, activation='sigmoid') # Output: 0 or 1
])

# Step 4: Compile the model
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Step 5: Train top layers (feature extractor phase)
model.fit(
    train_data,
    epochs=5,
    validation_data=val_data
)

# Step 6: Unfreeze some base model layers for fine-tuning
base_model.trainable = True
fine_tune_at = 100  # Freeze all layers before this

for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False

# Step 7: Recompile with a lower learning rate
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Step 8: Continue training (fine-tuning phase)
model.fit(
    train_data,
    epochs=10,
    initial_epoch=5,
```

```
    validation_data=val_data
)
```

1. **Create RNN or a transformer to classify text documents into different catogories such as sentiment anlysis or topic selection**

```python
# Import required libraries
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.datasets import imdb
from tensorflow.keras.preprocessing.sequence import pad_sequences

# Step 1: Load IMDB sentiment dataset
# Keep only the 10,000 most frequent words
vocab_size = 10000
max_length = 200

(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=vocab_size)

# Step 2: Pad all sequences to the same length
x_train = pad_sequences(x_train, maxlen=max_length)
x_test = pad_sequences(x_test, maxlen=max_length)

# Step 3: Build the RNN model
model = tf.keras.Sequential([
    layers.Embedding(input_dim=vocab_size, output_dim=32, input_length=max_length),  # Word embedding
    layers.SimpleRNN(32),                                                            # RNN layer
    layers.Dense(1, activation='sigmoid')                                            # Output: 0 (neg) or 1 (pos)
])

# Step 4: Compile the model
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Step 5: Train the model
model.fit(x_train, y_train, epochs=5, validation_split=0.2)

# Step 6: Evaluate on test data
loss, accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy:", accuracy)
```

5.Generate Synthetic images or data that resemble real data using GAN

```python
# Simple GAN to Generate Handwritten Digits

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

# Load and preprocess MNIST data
(x_train, _), _ = tf.keras.datasets.mnist.load_data()
x_train = (x_train - 127.5) / 127.5  # Normalize to [-1, 1]
x_train = x_train.reshape(-1, 28, 28, 1).astype('float32')
train_ds = tf.data.Dataset.from_tensor_slices(x_train).shuffle(60000).batch(256)

# Generator model: turns noise into an image
def build_generator():
    model = tf.keras.Sequential([
        layers.Dense(7*7*256, use_bias=False, input_shape=(100,)),
        layers.BatchNormalization(), layers.LeakyReLU(),
        layers.Reshape((7, 7, 256)),
        layers.Conv2DTranspose(128, 5, strides=1, padding='same', use_bias=False),
        layers.BatchNormalization(), layers.LeakyReLU(),
        layers.Conv2DTranspose(64, 5, strides=2, padding='same', use_bias=False),
        layers.BatchNormalization(), layers.LeakyReLU(),
        layers.Conv2DTranspose(1, 5, strides=2, padding='same', use_bias=False, activation='tanh')
    ])
    return model

# Discriminator model: tells real from fake images
def build_discriminator():
    model = tf.keras.Sequential([
        layers.Conv2D(64, 5, strides=2, padding='same', input_shape=[28,28,1]),
        layers.LeakyReLU(), layers.Dropout(0.3),
        layers.Conv2D(128, 5, strides=2, padding='same'),
        layers.LeakyReLU(), layers.Dropout(0.3),
        layers.Flatten(), layers.Dense(1)
    ])
    return model

# Losses and optimizers
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)
generator = build_generator()
discriminator = build_discriminator()
gen_opt = tf.keras.optimizers.Adam(1e-4)
disc_opt = tf.keras.optimizers.Adam(1e-4)

# Training step
@tf.function
def train_step(images):
    noise = tf.random.normal([256, 100])
    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        gen_imgs = generator(noise, training=True)
        real_out = discriminator(images, training=True)
        fake_out = discriminator(gen_imgs, training=True)
        gen_loss = cross_entropy(tf.ones_like(fake_out), fake_out)
        disc_loss = cross_entropy(tf.ones_like(real_out), real_out) + \
                    cross_entropy(tf.zeros_like(fake_out), fake_out)
    gen_grad = gen_tape.gradient(gen_loss, generator.trainable_variables)
    disc_grad = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
```

```python
    gen_opt.apply_gradients(zip(gen_grad, generator.trainable_variables))
    disc_opt.apply_gradients(zip(disc_grad, discriminator.trainable_variables))

# Generate and show images
def generate_images(model, test_input):
    preds = model(test_input, training=False)
    plt.figure(figsize=(4,4))
    for i in range(preds.shape[0]):
        plt.subplot(4, 4, i+1)
        plt.imshow((preds[i, :, :, 0] + 1) / 2.0, cmap='gray')
        plt.axis('off')
    plt.tight_layout()
    plt.show()

# Train loop
def train(dataset, epochs):
    seed = tf.random.normal([16, 100])
    for epoch in range(epochs):
        for batch in dataset:
            train_step(batch)
        print(f'Epoch {epoch+1} done')
        generate_images(generator, seed)

# Run training
train(train_ds, epochs=3)  # You can increase epochs for better results
```

6.implementation object detection model using frameworks like tensorflow or pytorch and evaluate it performance on dataset such as cow or pascal

```python
# Step 1: Install required packages (run this in your terminal or Jupyter notebook)
# !pip install tensorflow tensorflow-hub opencv-python

# Step 2: Import libraries
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
import cv2
import matplotlib.pyplot as plt

# Step 3: Load and preprocess one image
def load_image(img_path):
    img = tf.io.read_file(img_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, (384, 384)) / 255.0  # Resize and normalize
    return img

img_path = 'cow.jpg'  # Replace with your image file
image = load_image(img_path)
input_tensor = tf.expand_dims(image, 0)  # Add batch dimension

# Step 4: Load pre-trained object detection model
model = hub.load("https://tfhub.dev/tensorflow/efficientdet/lite2/detection/1")

# Step 5: Run detection
output = model(input_tensor)
boxes = output["detection_boxes"][0].numpy()
classes = output["detection_classes"][0].numpy().astype(np.int32)
scores = output["detection_scores"][0].numpy()

# Step 6: Draw boxes on the image
labels = {1: 'person', 17: 'cat', 18: 'dog', 20: 'cow'}  # Add more as needed

img_np = np.array(image * 255, dtype=np.uint8)
h, w = img_np.shape[:2]

for box, cls, score in zip(boxes, classes, scores):
    if score < 0.3:
        continue
    y1, x1, y2, x2 = box
    start = (int(x1 * w), int(y1 * h))
    end = (int(x2 * w), int(y2 * h))
    cv2.rectangle(img_np, start, end, (0, 255, 0), 2)
    label = f"{labels.get(cls, 'ID:'+str(cls))} ({score:.2f})"
    cv2.putText(img_np, label, (start[0], start[1]-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 1)

# Step 7: Show result
plt.imshow(img_np)
plt.axis("off")
plt.show()
```