

```
# -*- coding: utf-8 -*-
"""NEW.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1hoe-J-UjFpwb_81G8ksV5l6TB7NaNifg
"""

!pip install tensorflow keras pillow pyttsx3

!pip install -q kaggle

!pip install Opendatasets

import opendatasets as od
import pandas
data_dir = od.download(
    "https://www.kaggle.com/datasets/gopalbhattraí/pascal-voc-2012-dataset",
    force=True,
    data_dir='pascal-voc-2012-dataset' # Specify a different extraction directory
)

!pip install gTTS

import tensorflow as tf
import numpy as np
import json
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.applications import ResNet50, MobileNet
from tensorflow.keras.applications.resnet50 import preprocess_input as preprocess_input_resnet
from tensorflow.keras.applications.mobilenet import preprocess_input as preprocess_input_mobilenet
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, Add
from gtts import gTTS
import pyttsx3
import IPython.display as ipd
from pycocotools.coco import COCO
import requests
from io import BytesIO
from PIL import Image

data_dir = "/content/pascal-voc-2012-dataset"

import pathlib
data_dir= pathlib.Path(data_dir) #to add dataset path in directory
data_dir

list(data_dir.glob('*/*.jpg'))[:5]

image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)

# Load libraries
import matplotlib.pyplot as plt
import pandas as pd
import pickle
import numpy as np
import os
import cv2
import keras
from keras.applications.xception import Xception
from keras.applications.inception_v3 import InceptionV3
from keras.optimizers import Adam
from keras.layers import Dense, Flatten, Input, Dropout, LSTM, TimeDistributed, Embedding, Bidirectional, Activation, RepeatVector, Concatenate
from keras.models import Sequential, Model
from tensorflow.keras.utils import to_categorical
import random
from keras.preprocessing import image, sequence
from tensorflow.keras.applications.xception import preprocess_input as preprocess_xception
from tensorflow.keras.applications.inception_v3 import preprocess_input as preprocess_inception
from tensorflow.keras.applications import Xception, InceptionV3
from tensorflow.keras.layers import GlobalAveragePooling2D, Concatenate, Dense
from tensorflow.keras.models import Model
import torch
from transformers import BlipProcessor, BlipForConditionalGeneration
from PIL import Image
import requests
from gtts import gTTS
import IPython.display as ipd
```

```

image_count = len(list(data_dir.glob('/*/*.jpg')))
print(image_count)

# Load the BLIP processor and model
processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")
# # Load the BLIP processor and model
# processor = BlipProcessor.from_pretrained("Salesforce/blip-image-captioning-base")
# model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")

def load_and_preprocess_image(img_path, target_size):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)

    # Preprocess for Xception and InceptionV3
    img_xception = preprocess_xception(img_array)
    img_inception = preprocess_inception(img_array)

    # Expand dimensions to match input shape for both models
    img_xception = np.expand_dims(img_xception, axis=0)
    img_inception = np.expand_dims(img_inception, axis=0)

    return img_xception, img_inception

# Function to add prefix to layer names
def add_prefix_to_layers(model, prefix):
    for layer in model.layers:
        layer._name = prefix + layer.name

# Load Pretrained Models
xception_model = Xception(weights='imagenet', include_top=False)
add_prefix_to_layers(xception_model, "xception_")

inception_model = InceptionV3(weights='imagenet', include_top=False)
add_prefix_to_layers(inception_model, "inception_")

# Load Pretrained Models
xception_model = Xception(weights='imagenet', include_top=False)
inception_model = InceptionV3(weights='imagenet', include_top=False)

img_path = "/content/pascal-voc-2012-dataset/pascal-voc-2012-dataset/VOC2012_test/VOC2012_test/JPEGImages/2008_000006.jpg" # Removed ext
img_xception, img_inception = load_and_preprocess_image(img_path, target_size=(299, 299))

# Preprocess for Xception and InceptionV3
img_xception = preprocess_xception(img_xception)
img_inception = preprocess_inception(img_inception)

from tensorflow.keras.applications import Xception, InceptionV3
from tensorflow.keras.layers import Input, GlobalAveragePooling2D, Dense, Dropout, BatchNormalization, Concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

# Load pretrained models without top layers
xception_model = Xception(include_top=False, weights='imagenet', input_shape=(299, 299, 3))
inception_model = InceptionV3(include_top=False, weights='imagenet', input_shape=(299, 299, 3))

# Unfreeze some layers in the pretrained models for fine-tuning
for layer in xception_model.layers[-10:]:
    layer.trainable = True
for layer in inception_model.layers[-10:]:
    layer.trainable = True

def create_hybrid_model():
    # Inputs
    xception_input = Input(shape=(299, 299, 3), name='xception_input')
    inception_input = Input(shape=(299, 299, 3), name='inception_input')

    # Feature extraction
    xception_output = xception_model(xception_input)
    inception_output = inception_model(inception_input)

    # Global average pooling
    x_xception = GlobalAveragePooling2D()(xception_output)
    x_inception = GlobalAveragePooling2D()(inception_output)

    # Concatenate outputs
    concatenated = Concatenate()([x_xception, x_inception])

    # Fully connected layers
    x = Dense(512, activation='relu')(concatenated)
    x = BatchNormalization()(x)
    x = Dropout(0.5)(x)

```

```

x = Dense(256, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)

# Final predictions
predictions = Dense(10, activation='softmax')(x) # Adjust the output size to your task

# Build the model
model = Model(inputs=[xception_input, inception_input], outputs=predictions)
return model

# Create the model
hybrid_model = create_hybrid_model()

# Compile the model
hybrid_model.compile(
    optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

hybrid_model.summary()

def load_and_preprocess_image(img_path, target_size):
    img = image.load_img(img_path, target_size=target_size)
    img_array = image.img_to_array(img)

    # Preprocess for Xception and InceptionV3
    img_xception = preprocess_xception(img_array)
    img_inception = preprocess_inception(img_array)

    # Expand dimensions to match input shape for both models
    img_xception = np.expand_dims(img_xception, axis=0)
    img_inception = np.expand_dims(img_inception, axis=0)

    return img_xception, img_inception

img_path = '/content/pascal-voc-2012-dataset/pascal-voc-2012-dataset/VOC2012_test/VOC2012_test/JPEGImages/2008_000006.jpg' # replace with your own image path
img_xception, img_inception = load_and_preprocess_image(img_path, target_size=(299, 299))

# Make prediction
preds = hybrid_model.predict([img_xception, img_inception])
predicted_class = np.argmax(preds[0])

# Load ImageNet labels
!wget https://storage.googleapis.com/download.tensorflow.org/data/imagenet_class_index.json
import json
with open('imagenet_class_index.json', 'r') as f:
    class_dict = json.load(f)
class_labels = {int(key): value[1] for key, value in class_dict.items()}

predicted_label = class_labels[predicted_class]
print("Predicted label:", predicted_label)

# Extract Features
features = hybrid_model.predict([img_xception, img_inception])

# Ensure transformers library is installed
!pip install -q transformers

from transformers import GPT2Tokenizer, TFGPT2LMHeadModel

# Load the tokenizer and the model
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
captioning_model = TFGPT2LMHeadModel.from_pretrained("gpt2")

# Prepare input for the model
input_ids = tokenizer.encode("Features: ", return_tensors="tf")

# Prepare input for the model with attention mask
input_ids = tokenizer.encode("Features: ", return_tensors="tf")

# Create attention mask (no padding tokens, so all values are 1)
attention_mask = tf.ones_like(input_ids)

# Generate caption with attention mask and pad token ID
output = captioning_model.generate(
    input_ids,
    attention_mask=attention_mask,
    max_length=50,
    num_return_sequences=1,
    pad_token_id=tokenizer.eos_token_id # Use eos_token_id as pad token for GPT-2
)

```

```

caption = tokenizer.decode(output[0], skip_special_tokens=True)
print("Generated caption:", caption)

# Load and preprocess image
img_path = "/content/pascal-voc-2012-dataset/pascal-voc-2012-dataset/VOC2012_test/VOC2012_test/JPEGImages/2008_000022.jpg" # Local file
img = Image.open(img_path) # Open the image directly

# Preprocess the image using the BLIP processor
inputs = processor(img, return_tensors="pt")

!pip install -q transformers
from transformers import BlipForConditionalGeneration

# Assuming 'model' was originally a PretrainedResNetModel, replace it with:
model = BlipForConditionalGeneration.from_pretrained("Salesforce/blip-image-captioning-base")

# Now, the 'generate' method should be available
output = model.generate(**inputs)
caption = processor.decode(output[0], skip_special_tokens=True)

print("Generated caption:", caption)

!pip install deep_translator

!pip install deep_translator # Install deep_translator in the current environment
from deep_translator import GoogleTranslator
from gtts import gTTS
import IPython.display as ipd
# Prompt user for language code
def get_language_code():
    # List of language codes for common languages
    languages = {
        'English': 'en',
        'Spanish': 'es',
        'French': 'fr',
        'German': 'de',
        'Chinese': 'zh',
        'Japanese': 'ja',
        'Korean': 'ko',
        'Italian': 'it',
        'Hindi': 'hi',
        'Tamil': 'ta',
        'Bengali': 'bn',
        'Telugu': 'te',
        'Punjabi': 'pa',
        'Marathi': 'mr',
        'Russian': 'ru',
        'Arabic': 'ar',
        'Portuguese': 'pt',
        'Dutch': 'nl',
        'Swedish': 'sv',
        'Norwegian': 'no',
        'Greek': 'el',
        'Turkish': 'tr',
        'Persian': 'fa',
    }
    print("Available languages:")
    for lang_name, lang_code in languages.items():
        print(f"{lang_name}: {lang_code}")

    lang_choice = input("Enter the language code (e.g., 'en', 'es'): ").strip()
    return lang_choice if lang_choice in languages.values() else 'en' # Default to 'en' if not found

# Translate text to the desired language using deep_translator
def translate_text(text, dest_lang):
    translated = GoogleTranslator(source='auto', target=dest_lang).translate(text)
    return translated

# Convert text to speech
def text_to_speech(text, lang='en'):
    tts = gTTS(text=text, lang=lang)
    tts.save('caption.mp3') # Save the speech to an MP3 file
    return 'caption.mp3'

# Example usage
language_code = get_language_code() # Get language code from user

# Assuming 'caption' is already defined in the context
translated_caption = translate_text(caption, language_code) # Translate caption

```

```

print(f"Translated Caption: {translated_caption}")

voice_output = text_to_speech(translated_caption, lang=language_code) # Generate speech

# Play the audio
ipd.display(ipd.Audio(voice_output)) # This will play the generated audio

!pip install --upgrade torchvision
!pip install -q timm # Install the timm library

from torch.utils.data import Dataset # Import the Dataset class

class CustomDataset(Dataset):
    def __init__(self, data, targets, transform=None):
        self.data = data
        self.targets = targets
        self.transform = transform

    def __len__(self):
        return len(self.data)

    def __getitem__(self, idx):
        sample = self.data[idx]
        target = self.targets[idx]

        if self.transform:
            sample = self.transform(sample)

        return sample, target

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import torchvision.transforms as transforms
import torchvision.models as models # Added this import
import timm # Import timm to access external models like Xception
# Import Compose
from torchvision.transforms import Compose, Resize, Normalize

# Define the model with Pretrained Xception + InceptionV3 Hybrid
class HybridXceptionInceptionModel(nn.Module):
    def __init__(self):
        super(HybridXceptionInceptionModel, self).__init__()
        self.xception = timm.create_model('xception', pretrained=True) # Create Xception using timm

        # Change aux_logits to True to match the pretrained weights
        self.inception_v3 = models.inception_v3(pretrained=True, aux_logits=True)

        # Modify the fully connected layers for binary classification
        self.xception.fc = nn.Linear(self.xception.fc.in_features, 1)
        # Remove the auxiliary classifier after loading the weights
        self.inception_v3.AuxLogits = None
        self.inception_v3.fc = nn.Linear(self.inception_v3.fc.in_features, 1)

        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        xception_out = self.xception(x)
        inception_out = self.inception_v3(x)
        # Combine both model outputs
        x = (xception_out + inception_out) / 2 # Averaging the results from both models
        x = self.sigmoid(x)
        return x

import torch
import numpy as np

# Placeholder - Replace these with your actual data
# Example: 100 samples with 3 channels (RGB) and 224x224 dimensions
x_data = np.random.randn(100, 3, 224, 224).astype(np.float32) # Example data
y_data = np.random.randint(0, 2, (100, 1)).astype(np.float32) # Example binary target values

# Convert x_data and y_data to torch tensors
x_data = torch.tensor(x_data, dtype=torch.float32)
y_data = torch.tensor(y_data, dtype=torch.float32)

# Now x_data and y_data are ready for use

import torch
import torch.nn as nn

```

```

import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset, random_split
import torchvision.transforms as transforms
import torchvision.models as models
import timm
from torchvision.transforms import Compose, Resize, Normalize

# Assuming you have your data in x_data and y_data as NumPy arrays
# Initialize x_data and y_data with your actual data
# For example, if your data is in NumPy arrays:
# x_data = np.array(...) # Replace ... with your actual data
# y_data = np.array(...) # Replace ... with your actual data

# Clone and detach x_data and y_data
x_data = x_data.clone().detach().to(torch.float32) # Ensure it's a proper tensor for further use
y_data = y_data.clone().detach().to(torch.float32)

# Create a TensorDataset
dataset = TensorDataset(x_data, y_data)

# Split into train and test sets
train_size = int(0.8 * len(dataset)) # 80% for training
test_size = len(dataset) - train_size
train_dataset, test_dataset = random_split(dataset, [train_size, test_size])

# Define HybridXceptionInceptionModel
class HybridXceptionInceptionModel(nn.Module):
    def __init__(self):
        super(HybridXceptionInceptionModel, self).__init__()
        # Use timm for Xception model
        self.xception = timm.create_model("xception", pretrained=True)
        # Use default InceptionV3 model without specifying aux_logits=False
        self.inception_v3 = models.inception_v3(weights=models.Inception_V3_Weights.IMAGENET1K_V1)
        self.fc = nn.Linear(2048 + 2048, 1) # Combining both Xception and Inception V3 outputs
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        # Forward pass for both models
        xception_features = self.xception(x)
        inception_features = self.inception_v3(x)[0] # Use the output features from InceptionV3

        # Concatenate both feature vectors and pass through final layer
        combined_features = torch.cat((xception_features, inception_features), dim=1)
        output = self.fc(combined_features)
        return self.sigmoid(output)

# Initialize model, loss function, and optimizer
model = HybridXceptionInceptionModel()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Define loss function and optimizer
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=0.0001) # Lower learning rate for fine-tuning

# Define the transformations
transform = Compose([
    Resize((299, 299)), # Resize images to 299x299 (for InceptionV3)
    Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]) # ImageNet mean and std
])

# Create the DataLoader for the hybrid model
train_loader = DataLoader(CustomDataset(train_dataset.dataset.tensors[0], train_dataset.dataset.tensors[1], transform=transform), batch_size=16)
test_loader = DataLoader(CustomDataset(test_dataset.dataset.tensors[0], test_dataset.dataset.tensors[1], transform=transform), batch_size=16)

# Define the model with Pretrained ResNet
class PretrainedResNetModel(nn.Module):
    def __init__(self):
        super(PretrainedResNetModel, self).__init__()
        self.resnet = models.resnet50(pretrained=True)
        self.resnet.fc = nn.Linear(self.resnet.fc.in_features, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        x = self.resnet(x)
        x = self.sigmoid(x)
        return x

# Initialize model, loss function, and optimizer
model = PretrainedResNetModel().to(torch.float32) # Ensure model parameters are in float32
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

```

```

# Define loss function and optimizer
criterion = nn.BCELoss().to(device) # Ensure loss function is on the correct device
optimizer = optim.Adam(model.parameters(), lr=0.0001)

import torch
import torch.nn as nn
import torch.nn.functional as F

class SelfAttention(nn.Module):
    def __init__(self, input_dim):
        super(SelfAttention, self).__init__()
        self.query = nn.Linear(input_dim, input_dim)
        self.key = nn.Linear(input_dim, input_dim)
        self.value = nn.Linear(input_dim, input_dim)

    def forward(self, x):
        # x.shape = (batch_size, seq_len, input_dim)
        queries = self.query(x) # (batch_size, seq_len, input_dim)
        keys = self.key(x) # (batch_size, seq_len, input_dim)
        values = self.value(x) # (batch_size, seq_len, input_dim)

        # Calculate attention scores (scaled dot product attention)
        scores = torch.matmul(queries, keys.transpose(-2, -1)) / torch.sqrt(torch.tensor(x.size(-1), dtype=torch.float32)) # (batch_size, seq_len, seq_len)

        # Apply softmax to get attention weights
        attention_weights = F.softmax(scores, dim=-1) # (batch_size, seq_len, seq_len)

        # Weighted sum of values (output of attention layer)
        weighted_values = torch.matmul(attention_weights, values) # (batch_size, seq_len, input_dim)

        return weighted_values

!pip install transformers

from transformers import ViTFeatureExtractor, ViTModel

#Loading pre trained ViT model and feature extractor
feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224-in21k')
vit_model = ViTModel.from_pretrained('google/vit-base-patch16-224-in21k')

import torch

# Assuming x_data and y_data are already tensors, use clone().detach() to avoid the warning
x_data = x_data.clone().detach().to(torch.float32)
y_data = y_data.clone().detach().to(torch.float32)

# If you want to verify the conversion
print(x_data.dtype) # Should print: torch.float32
print(y_data.dtype) # Should print: torch.float32

from torchvision import models
import torch.nn as nn
import torch.optim as optim
import torch
from transformers import ViTFeatureExtractor, ViTModel

# Data Augmentation
transform = transforms.Compose([
    transforms.RandomResizedCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

# Regularization (L2)
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=0.0001)

class HybridXceptionInceptionModel(nn.Module):
    def __init__(self):
        super(HybridXceptionInceptionModel, self).__init__()
        self.xception = models.xception(pretrained=True)
        self.inception_v3 = models.inception_v3(pretrained=True, aux_logits=False)
        self.vit_feature_extractor = ViTFeatureExtractor.from_pretrained('google/vit-base-patch16-224-in21k')
        self.vit_model = ViTModel.from_pretrained('google/vit-base-patch16-224-in21k')
        self.dropout = nn.Dropout(0.5) # Adjust dropout probability as needed

        # Replace the fully connected layer with a new one that combines features from Xception and InceptionV3
        self.fc = nn.Linear(self.xception.fc.in_features + self.inception_v3.fc.in_features + self.vit_model.config.hidden_size, 1)
        self.sigmoid = nn.Sigmoid()

    def forward(self, x):
        # Xception feature extraction

```

```

xception_features = self.xception(x)

# InceptionV3 feature extraction
inception_features = self.inception_v3(x)

# Apply dropout
xception_features = self.dropout(xception_features)
inception_features = self.dropout(inception_features)

# Preprocess input for ViT
inputs = self.vit_feature_extractor(images=x, return_tensors="pt").to(x.device)
vit_outputs = self.vit_model(**inputs).last_hidden_state
vit_features = vit_outputs[:, 0, :] # Use the [CLS] token embedding as the ViT feature

# Combine features from Xception, InceptionV3, and ViT
combined_features = torch.cat([xception_features, inception_features, vit_features], dim=1)

# Final output
output = self.fc(combined_features)
output = self.sigmoid(output)
return output

import torch
import torch.nn as nn

# Training loop
epochs = 20
for epoch in range(epochs):
    model.train()
    epoch_loss = 0
    for batch_x, batch_y in train_loader:
        batch_x, batch_y = batch_x.to(device).to(torch.float32), batch_y.to(device).to(torch.float32)

        optimizer.zero_grad() # Zero the gradients
        outputs = model(batch_x) # Forward pass
        loss = criterion(outputs, batch_y) # Compute loss
        loss.backward() # Backward pass
        optimizer.step() # Update weights

        epoch_loss += loss.item()

    epoch_loss /= len(train_loader)
    print(f"Epoch {epoch+1}/{epochs}, Loss: {epoch_loss:.4f}")

# Evaluate the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score # Metrics import
import matplotlib.pyplot as plt
import numpy as np

model.eval()
y_true = []
y_pred = []

# Evaluation Loop
with torch.no_grad():
    for batch_x, batch_y in test_loader:
        # Move data to device and convert to float32
        batch_x, batch_y = batch_x.to(device, dtype=torch.float32), batch_y.to(device, dtype=torch.float32)

        # Get predictions
        outputs = model(batch_x)
        y_true.extend(batch_y.cpu().numpy()) # Append ground truth
        y_pred.extend(outputs.cpu().numpy()) # Append predictions

# Convert predictions to binary (threshold at 0.5)
y_pred = (np.array(y_pred) > 0.5).astype(int)
y_true = np.array(y_true).astype(int)

# Calculate Metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, zero_division=0)
recall = recall_score(y_true, y_pred, zero_division=0)
f1 = f1_score(y_true, y_pred, zero_division=0)

# Print Metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")

# Visualize Metrics
metrics = {
    'Accuracy': accuracy,

```

```

'Precision': precision,
'Recall': recall,
'F1 Score': f1
}

plt.figure(figsize=(10, 6))
plt.bar(metrics.keys(), metrics.values(), color=['b', 'g', 'r', 'c'])
plt.title('Evaluation Metrics')
plt.ylabel('Score')
plt.ylim(0, 1)
plt.grid(True)
plt.show()

def calculate_mcc(tp, tn, fp, fn):
    """
    Calculate the Matthews Correlation Coefficient (MCC).

    Parameters:
    tp (int): True Positives
    tn (int): True Negatives
    fp (int): False Positives
    fn (int): False Negatives

    Returns:
    float: The MCC score
    """

    # Calculate the MCC
    numerator = (tp * tn) - (fp * fn)
    denominator = np.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))

    if denominator == 0:
        return 0.0 # Return 0 if the denominator is zero to avoid division by zero

    return numerator / denominator

# Calculate tp, tn, fp, fn from y_true and y_pred
from sklearn.metrics import confusion_matrix
tn, fp, fn, tp = confusion_matrix(y_true, y_pred).ravel()

mcc_score = calculate_mcc(tp, tn, fp, fn)
print(f'MCC Score: {mcc_score:.4f}')

!pip install matplotlib seaborn

import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

# Assuming y_true and y_pred are available as NumPy arrays
conf_matrix = confusion_matrix(y_true, y_pred)

# Plot the confusion matrix using Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()

y_pred_proba = model(batch_x).detach().cpu().numpy() # Store probabilities

# Evaluate the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc

model.eval()
y_true = []
y_pred = []
y_pred_proba = [] # Store probabilities for all batches

with torch.no_grad():
    for batch_x, batch_y in test_loader:
        batch_x, batch_y = batch_x.to(device).to(torch.float32), batch_y.to(device).to(torch.float32)
        outputs = model(batch_x)
        y_true.extend(batch_y.cpu().numpy())
        y_pred.extend(outputs.cpu().numpy() > 0.5) # Directly store binary predictions
        y_pred_proba.extend(outputs.cpu().numpy()) # Store probabilities

# Convert lists to numpy arrays
y_true = np.array(y_true).astype(int)
y_pred = np.array(y_pred).astype(int)
y_pred_proba = np.array(y_pred_proba).squeeze() # Squeeze to get a 1D array of probabilities

```

```

# Calculate ROC curve and AUC
fpr, tpr, thresholds = roc_curve(y_true, y_pred_proba)
roc_auc = auc(fpr, tpr)
print(f'ROC AUC: {roc_auc:.4f}')

import matplotlib.pyplot as plt

plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

import numpy as np

variance = np.var(y_pred_proba)
print("Variance of model predictions:", variance)

def count_layers(model):
    """Counts the number of layers in a PyTorch model."""
    layer_count = 0
    for name, module in model.named_modules():
        if isinstance(module, (torch.nn.Linear, torch.nn.Conv2d, torch.nn.BatchNorm2d, torch.nn.ReLU, torch.nn.MaxPool2d)): # Add other
            layer_count += 1
    return layer_count

num_layers = count_layers(model)
print(f"The model has {num_layers} layers.")

# Assuming 'model' is your PyTorch model, 'test_data', and 'test_labels' are your test data
import torch

# Define your loss function
criterion = torch.nn.CrossEntropyLoss()

# ... (Your PyTorch evaluation loop)
model.eval() # Set model to evaluation mode
with torch.no_grad():
    # Move x_data to the same device as the model
    x_data = x_data.to(device)
    outputs = model(x_data)
    # Assuming 'predicted_label' should be a tensor of class indices
    # Replace this with the actual way to get the correct class indices for your data
    predicted_label = torch.tensor([0] * len(outputs)).to(device) # Example: all samples belong to class 0 and move to device
    loss = criterion(outputs, predicted_label)
    _, predicted = torch.max(outputs, 1)
    top1_accuracy = (predicted == predicted_label).sum().item() / len(predicted_label)

print("Test Loss:", loss.item())
print("Top-1 Accuracy:", top1_accuracy)

```

```

Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.1)
Requirement already satisfied: keras in /usr/local/lib/python3.10/dist-packages (3.5.0)
Requirement already satisfied: pillow in /usr/local/lib/python3.10/dist-packages (11.0.0)
Requirement already satisfied: pytsx3 in /usr/local/lib/python3.10/dist-packages (2.98)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!0=0.5.1,!0=0.5.2,>0=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.2)
Requirement already satisfied: protobuf>=4.21.0,!0=4.21.1,!0=4.21.2,!0=4.21.3,!0=4.21.4,!0=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/p
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.68.1)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.26.4)
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras) (0.13.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensor
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorflow)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras) (2.18.0)
Requirement already satisfied: mdurl~0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras) (0.1
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.18
Requirement already satisfied: Opendatasets in /usr/local/lib/python3.10/dist-packages (0.1.22)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from Opendatasets) (4.66.6)
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (from Opendatasets) (1.6.17)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from Opendatasets) (8.1.7)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle->Opendatasets) (1.16.0)
Requirement already satisfied: certifi>=2023.7.22 in /usr/local/lib/python3.10/dist-packages (from kaggle->Opendatasets) (2024.8.30)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle->Opendatasets) (2.8.2)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle->Opendatasets) (2.32.3)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle->Opendatasets) (8.0.4)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle->Opendatasets) (2.2.3)
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle->Opendatasets) (6.2.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->kaggle->Opendatasets) (0.5.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from python-slugify->kaggle->Opendat
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle->Opendat
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->kaggle->Opendatasets) (3.10)
Please provide your Kaggle credentials to download this dataset. Learn more: http://bit.ly/kaggle-creds
Your Kaggle username: shubha
Your Kaggle Key: .....
Dataset URL: https://www.kaggle.com/datasets/gopalbhattrai/pascal-voc-2012-dataset
Downloading pascal-voc-2012-dataset.zip to pascal-voc-2012-dataset/pascal-voc-2012-dataset
100%|██████████| 3.52G/3.52G [02:41<00:00, 23.4MB/s]

```

```

Requirement already satisfied: gTTS in /usr/local/lib/python3.10/dist-packages (2.5.4)
Requirement already satisfied: requests<3,>=2.27 in /usr/local/lib/python3.10/dist-packages (from gTTS) (2.32.3)
Requirement already satisfied: click<8.2,>=7.1 in /usr/local/lib/python3.10/dist-packages (from gTTS) (8.1.7)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->gTTS) (0
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->gTTS) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->gTTS) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.27->gTTS) (2024.8
0
0
Model: "functional_6"

```

Layer (type)	Output Shape	Param #	Connected to
xception_input (InputLayer)	(None, 299, 299, 3)	0	-
inception_input (InputLayer)	(None, 299, 299, 3)	0	-
xception (Functional)	(None, 10, 10, 2048)	20,861,480	xception_input[0][0]
inception_v3 (Functional)	(None, 8, 8, 2048)	21,802,784	inception_input[0][0]
global_average_pooling2d... (GlobalAveragePooling2D)	(None, 2048)	0	xception[0][0]
global_average_pooling2d... (GlobalAveragePooling2D)	(None, 2048)	0	inception_v3[0][0]

concatenate_24 (Concatenate)	(None, 4096)	0	global_average_poolin... global_average_poolin...
dense_16 (Dense)	(None, 512)	2,097,664	concatenate_24[0][0]
batch_normalization_890 (BatchNormalization)	(None, 512)	2,048	dense_16[0][0]
dropout_10 (Dropout)	(None, 512)	0	batch_normalization_8...
dense_17 (Dense)	(None, 256)	131,328	dropout_10[0][0]
batch_normalization_891 (BatchNormalization)	(None, 256)	1,024	dense_17[0][0]
dropout_11 (Dropout)	(None, 256)	0	batch_normalization_8...
dense_18 (Dense)	(None, 10)	2,570	dropout_11[0][0]

Total params: 44,898,898 (171.28 MB)  
 Trainable params: 44,808,402 (170.93 MB)  
 Non-trainable params: 90,496 (353.50 KB)

1/1 9s 9s/step

```
--2024-12-10 06:38:04-- https://storage.googleapis.com/download.tensorflow.org/data/imagenet\_class\_index.json
Resolving storage.googleapis.com (storage.googleapis.com)... 142.251.10.207, 142.251.12.207, 172.217.194.207, ...
Connecting to storage.googleapis.com (storage.googleapis.com)|142.251.10.207|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 35363 (35K) [application/json]
Saving to: 'imagenet_class_index.json.2'
```

imagenet\_class\_index 100%[=====] 34.53K --.-KB/s in 0.02s

2024-12-10 06:38:04 (1.49 MB/s) - 'imagenet\_class\_index.json.2' saved [35363/35363]

Predicted label: tench

1/1 0s 30ms/step

All PyTorch model weights were used when initializing TFGPT2LMHeadModel.

All the weights of TFGPT2LMHeadModel were initialized from the PyTorch model.

If your task is similar to the task the model of the checkpoint was trained on, you can already use TFGPT2LMHeadModel for prediction.  
 Generated caption: Features: The first thing I did was to create a new file called "config.json" which contains the following:

```
{ "name": "config.json", "version": "1.0", "description": "The default
/usr/local/lib/python3.10/dist-packages/transformers/generation/utils.py:1375: UserWarning: Using the model-agnostic default `max_length` instead of `max_len`."}
```

Generated caption: a counter with a variety of food items

```
Requirement already satisfied: deep_translator in /usr/local/lib/python3.10/dist-packages (1.11.4)
Requirement already satisfied: beautifulsoup4<5.0.0,>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from deep_translator) (4.12)
Requirement already satisfied: requests<3.0.0,>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from deep_translator) (2.32.3)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4<5.0.0,>=4.9.1->deep_translator) (1.2.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.23.0->deep_translator) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.23.0->deep_translator) (2.32.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.23.0->deep_translator) (2.32.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.23.0->deep_translator) (2023.10.1)
Requirement already satisfied: deep_translator in /usr/local/lib/python3.10/dist-packages (1.11.4)
Requirement already satisfied: beautifulsoup4<5.0.0,>=4.9.1 in /usr/local/lib/python3.10/dist-packages (from deep_translator) (4.12)
Requirement already satisfied: requests<3.0.0,>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from deep_translator) (2.32.3)
Requirement already satisfied: soupsieve>1.2 in /usr/local/lib/python3.10/dist-packages (from beautifulsoup4<5.0.0,>=4.9.1->deep_translator) (1.2.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.23.0->deep_translator) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.23.0->deep_translator) (2.32.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.23.0->deep_translator) (2.32.1)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3.0.0,>=2.23.0->deep_translator) (2023.10.1)
```

Available languages:

English: en

Spanish: es

French: fr

German: de

Chinese: zh

Japanese: ja

Korean: ko

Italian: it

Hindi: hi

Tamil: ta

Bengali: bn

Telugu: te

Punjabi: pa

Marathi: mr

Russian: ru

Arabic: ar

Portuguese: pt

Dutch: nl

Swedish: sv

Norwegian: no

Greek: el

Turkish: tr

Persian: fa

Enter the language code (e.g., 'en', 'es'): en

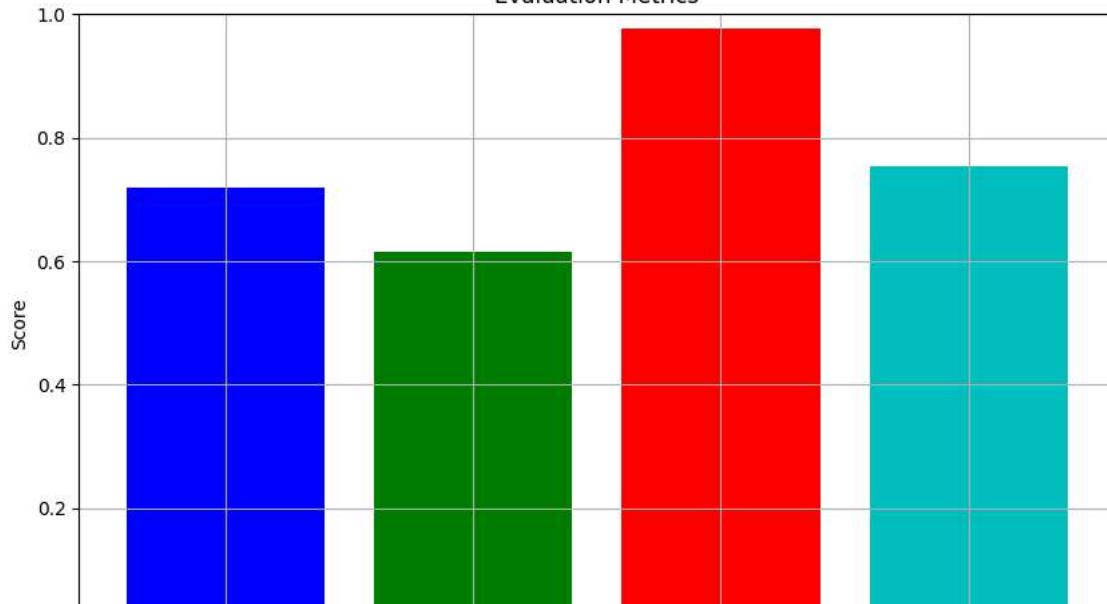
Translated Caption: a counter with a variety of food items

```

Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.20.1+cu121)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: torch==2.5.1 in /usr/local/lib/python3.10/dist-packages (from torchvision) (2.5.1+cu121)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (11.0.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch==2.5.1->torchvision) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch==2.5.1->torchvision)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch==2.5.1->torchvision) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch==2.5.1->torchvision) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch==2.5.1->torchvision) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch==2.5.1->torchvision) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch==2.5.1->torchvision)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch==2.5.1->torchvision)
/usr/local/lib/python3.10/dist-packages/timm/models/_factory.py:117: UserWarning: Mapping deprecated model name xception to current
    model = create_fn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since
    warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or 'None'
    warnings.warn(msg)
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.46.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.26.3)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.21,>=0.20 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.20.3)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.6)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.1.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.8.1)
/usr/local/lib/python3.10/dist-packages/transformers/models/vit/feature_extraction_vit.py:28: FutureWarning: The class ViTFeatureExtractor has been renamed to ViTFeatureExtraction
    warnings.warn(
torch.float32
torch.float32
Epoch 1/20, Loss: 0.7468
Epoch 2/20, Loss: 0.3404
Epoch 3/20, Loss: 0.3310
Epoch 4/20, Loss: 0.3528
Epoch 5/20, Loss: 0.1554
Epoch 6/20, Loss: 0.1188
Epoch 7/20, Loss: 0.0598
Epoch 8/20, Loss: 0.0372
Epoch 9/20, Loss: 0.1378
Epoch 10/20, Loss: 1.0753
Epoch 11/20, Loss: 0.2018
Epoch 12/20, Loss: 0.0839
Epoch 13/20, Loss: 0.1193
Epoch 14/20, Loss: 0.1456
Epoch 15/20, Loss: 0.0410
Epoch 16/20, Loss: 0.0145
Epoch 17/20, Loss: 0.0054
Epoch 18/20, Loss: 0.0068
Epoch 19/20, Loss: 0.0191
Epoch 20/20, Loss: 0.8488
Accuracy: 0.7200
Precision: 0.6143
Recall: 0.9773
F1 Score: 0.7544

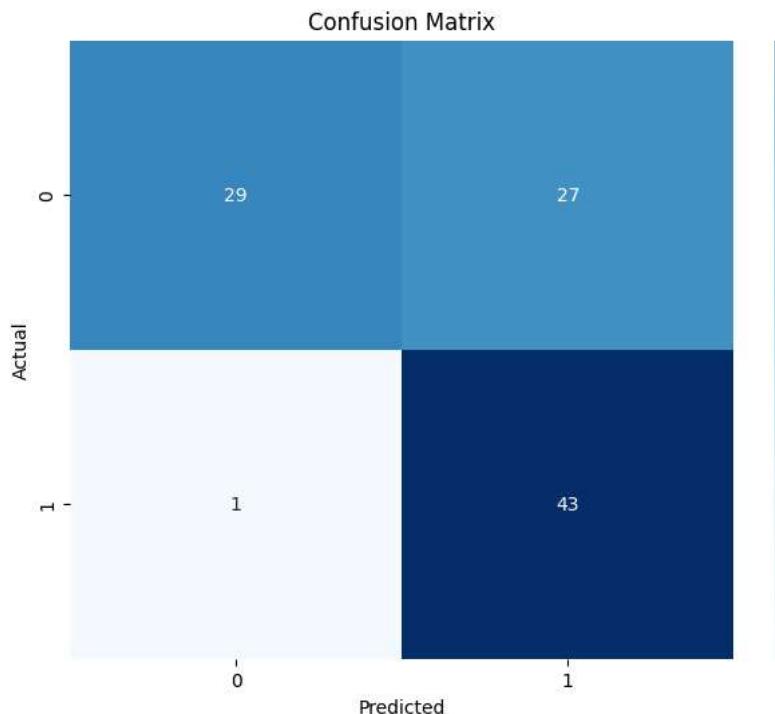
```

Evaluation Metrics

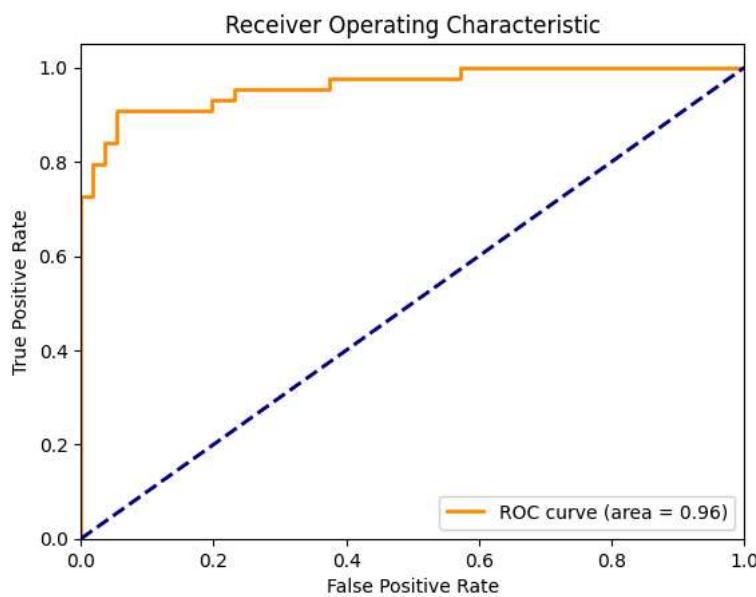




MCC Score: 0.5363  
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)  
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.13.2)  
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)  
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)  
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.55.1)  
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)  
Requirement already satisfied: numpy<2,>=1.21 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.26.4)  
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.2)  
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (11.0.0)  
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)  
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)  
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-packages (from seaborn) (2.2.2)  
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn) (2024.2)  
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn) (2024.2)  
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)



ROC AUC: 0.9623



Variance of model predictions: 0.1405295

The model has 125 layers.

Test Loss: 0.0

Top-1 Accuracy: 1.0