import warnings  
warnings.filterwarnings('ignore')  
import numpy as np  
import matplotlib.pyplot as plt  
from glob import glob  
import tensorflow as tf  
from tensorflow.keras.layers import Dense, Flatten, Dropout  
from tensorflow.keras.models import Model  
from tensorflow.keras.applications.mobilenet import MobileNet, preprocess\_input  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
import tensorflow.keras.utils as image\_utils  
import os  
import tkinter as tk  
from tkinter import filedialog, messagebox, ttk  
from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay  
import seaborn as sns  
  
# Constants  
IMAGE\_SIZE = [224, 224]  
BATCH\_SIZE = 32  
EPOCHS = 10 # Increase for better training  
  
# Paths - update as per your setup  
train\_path = r'C:\Users\PC\Desktop\New folder (3)\chest\_xray\train'  
test\_path = r'C:\Users\PC\Desktop\New folder (3)\chest\_xray\test'  
save\_path = r'C:\Users\PC\Desktop\mobilenet\chest\_xray\_mobilenet.keras'  
  
# Load MobileNet base model without top layers  
mobilenet = MobileNet(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)  
for layer in mobilenet.layers:  
 layer.trainable = False # Freeze base model layers  
  
# Add custom layers on top  
x = Flatten()(mobilenet.output)  
x = Dense(128, activation='relu')(x)  
x = Dropout(0.5)(x)  
prediction = Dense(2, activation='softmax')(x)  
  
model = Model(inputs=mobilenet.input, outputs=prediction)  
  
# Compile model  
model.compile(  
 loss='categorical\_crossentropy',  
 optimizer=tf.keras.optimizers.Adam(learning\_rate=0.0001),  
 metrics=['accuracy']  
)  
  
# Data augmentation for training  
train\_datagen = ImageDataGenerator(  
 preprocessing\_function=preprocess\_input,  
 rotation\_range=20,  
 width\_shift\_range=0.2,  
 height\_shift\_range=0.2,  
 shear\_range=0.2,  
 zoom\_range=0.2,  
 horizontal\_flip=True,  
 fill\_mode='nearest',  
 brightness\_range=[0.8, 1.2]  
)  
  
test\_datagen = ImageDataGenerator(preprocessing\_function=preprocess\_input)  
  
# Data generators  
training\_set = train\_datagen.flow\_from\_directory(  
 train\_path,  
 target\_size=IMAGE\_SIZE,  
 batch\_size=BATCH\_SIZE,  
 class\_mode='categorical',  
 shuffle=True  
)  
  
test\_set = test\_datagen.flow\_from\_directory(  
 test\_path,  
 target\_size=IMAGE\_SIZE,  
 batch\_size=BATCH\_SIZE,  
 class\_mode='categorical',  
 shuffle=False  
)  
  
# Get class names ordered by index  
class\_indices = training\_set.class\_indices  
class\_names = [''] \* len(class\_indices)  
for cls, idx in class\_indices.items():  
 class\_names[idx] = cls.upper()  
print("Classes:", class\_names)  
  
# Class weights to handle imbalance (adjust if needed)  
class\_weights = {  
 class\_indices['NORMAL']: 1.5,  
 class\_indices['PNEUMONIA']: 1.0  
}  
  
# Train model with EarlyStopping  
callbacks = [  
 tf.keras.callbacks.EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)  
]  
  
history = model.fit(  
 training\_set,  
 validation\_data=test\_set,  
 epochs=EPOCHS,  
 steps\_per\_epoch=len(training\_set),  
 validation\_steps=len(test\_set),  
 class\_weight=class\_weights,  
 callbacks=callbacks  
)  
  
# Save the model  
os.makedirs(os.path.dirname(save\_path), exist\_ok=True)  
model.save(save\_path)  
  
# --- Evaluation on test set ---  
  
# Predict on test data  
Y\_pred = model.predict(test\_set, verbose=1)  
y\_pred = np.argmax(Y\_pred, axis=1)  
y\_true = test\_set.classes  
  
# Confusion Matrix numeric  
cm = confusion\_matrix(y\_true, y\_pred)  
print("\nConfusion Matrix:\n", cm)  
  
# Confusion Matrix Display (plot)  
disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=class\_names)  
disp.plot(cmap='Blues')  
plt.title("Confusion Matrix")  
plt.show()  
  
# Confusion Matrix Heatmap (better visualization)  
plt.figure(figsize=(6, 5))  
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class\_names, yticklabels=class\_names)  
plt.xlabel('Predicted Label')  
plt.ylabel('True Label')  
plt.title('Confusion Matrix Heatmap')  
plt.tight\_layout()  
plt.show()  
  
# Classification Report: precision, recall, f1-score, support  
report = classification\_report(y\_true, y\_pred, target\_names=class\_names)  
print("\nClassification Report:\n", report)  
  
# You can also get the report as dict for detailed analysis  
report\_dict = classification\_report(y\_true, y\_pred, target\_names=class\_names, output\_dict=True)  
  
# Optionally print detailed per-class metrics  
print("Detailed Metrics:")  
for cls in class\_names:  
 print(f"\nClass: {cls}")  
 precision = report\_dict[cls]['precision']  
 recall = report\_dict[cls]['recall']  
 f1\_score = report\_dict[cls]['f1-score']  
 support = report\_dict[cls]['support']  
 print(f"Precision: {precision:.4f}")  
 print(f"Recall: {recall:.4f}")  
 print(f"F1-Score: {f1\_score:.4f}")  
 print(f"Support (samples): {support}")  
  
# Plot training history: Loss and Accuracy  
plt.figure(figsize=(14, 5))  
  
plt.subplot(1, 2, 1)  
plt.plot(history.history['loss'], label='Train Loss')  
plt.plot(history.history['val\_loss'], label='Validation Loss')  
plt.title('Loss Over Epochs')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.legend()  
  
plt.subplot(1, 2, 2)  
plt.plot(history.history['accuracy'], label='Train Accuracy')  
plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')  
plt.title('Accuracy Over Epochs')  
plt.xlabel('Epoch')  
plt.ylabel('Accuracy')  
plt.legend()  
  
plt.tight\_layout()  
plt.show()  
  
# --- GUI for image prediction ---  
def load\_and\_predict():  
 try:  
 img\_path = filedialog.askopenfilename(  
 title="Select Chest X-ray Image",  
 filetypes=[("Image files", "\*.jpeg \*.jpg \*.png")]  
 )  
 if not img\_path:  
 return  
  
 img = image\_utils.load\_img(img\_path, target\_size=IMAGE\_SIZE)  
 x = image\_utils.img\_to\_array(img)  
 x = np.expand\_dims(x, axis=0)  
 img\_data = preprocess\_input(x)  
  
 loaded\_model = tf.keras.models.load\_model(save\_path)  
 prediction = loaded\_model.predict(img\_data)  
 confidence = np.max(prediction[0])  
 predicted\_class = np.argmax(prediction[0])  
  
 result\_text = f"Prediction: {class\_names[predicted\_class]}\nConfidence: {confidence:.2%}"  
 result\_label.config(text=result\_text)  
  
 if class\_names[predicted\_class] == 'PNEUMONIA':  
 result\_label.config(fg="red")  
 else:  
 result\_label.config(fg="green")  
  
 except Exception as e:  
 messagebox.showerror("Error", f"An error occurred: {str(e)}")  
  
# GUI Setup  
root = tk.Tk()  
root.title("Pneumonia Detection")  
root.geometry("400x300")  
root.configure(bg="#f0f4f8")  
  
style = ttk.Style()  
style.configure("TButton", font=("Helvetica", 12), padding=10)  
  
title\_label = tk.Label(root, text="X-ray Pneumonia Detector", font=("Helvetica", 16, "bold"),  
 bg="#f0f4f8", fg="#333333")  
title\_label.pack(pady=20)  
  
select\_button = ttk.Button(root, text="Select Image and Predict", command=load\_and\_predict, style="TButton")  
select\_button.pack(pady=20)  
  
result\_frame = tk.Frame(root, bg="#ffffff", bd=2, relief="groove")  
result\_frame.pack(pady=20, padx=20, fill="x")  
  
result\_label = tk.Label(result\_frame, text="Result will appear here", font=("Helvetica", 12),  
 bg="#ffffff", fg="#333333", wraplength=350, justify="center")  
result\_label.pack(pady=10, padx=10)  
  
root.mainloop()

# **25 EPOCHS**

# Import libraries  
import warnings  
warnings.filterwarnings('ignore')  
  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from glob import glob  
import tensorflow as tf  
from tensorflow.keras.layers import Input, Dense, Flatten, Dropout, MaxPooling2D, BatchNormalization  
from tensorflow.keras.models import Model  
from tensorflow.keras.applications.mobilenet import MobileNet  
from tensorflow.keras.applications.mobilenet import preprocess\_input  
from tensorflow.keras.preprocessing.image import ImageDataGenerator  
import tensorflow.keras.utils as image\_utils  
from tensorflow.keras.metrics import Precision, Recall  
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau  
from tensorflow\_addons.metrics import F1Score  
from sklearn.metrics import confusion\_matrix, classification\_report  
import os  
  
# Define image size  
IMAGE\_SIZE = [224, 224]  
  
# Set paths to your local dataset  
train\_path = r'C:\Users\PC\Desktop\New folder (3)\chest\_xray\train'  
test\_path = r'C:\Users\PC\Desktop\New folder (3)\chest\_xray\test'  
  
# Define the folder to save plots  
save\_folder = r'C:\Users\PC\Desktop\mobilnet'  
os.makedirs(save\_folder, exist\_ok=True) # Create the folder if it doesn't exist  
  
# Verify the directory structure  
print("Training folders:", glob(train\_path + '/\*'))  
  
# Load MobileNet model  
mobilenet = MobileNet(input\_shape=IMAGE\_SIZE + [3], weights='imagenet', include\_top=False)  
  
# Freeze most MobileNet layers, but unfreeze the last 10 layers for fine-tuning  
for layer in mobilenet.layers[:-10]:  
 layer.trainable = False  
for layer in mobilenet.layers[-10:]:  
 layer.trainable = True  
  
# Add custom layers with MaxPooling, BatchNormalization, and increased Dropout  
x = mobilenet.output  
x = MaxPooling2D(pool\_size=(2, 2))(x)  
x = Flatten()(x)  
x = Dense(128, activation='relu', kernel\_regularizer=tf.keras.regularizers.l2(0.01))(x)  
x = BatchNormalization()(x)  
x = Dropout(0.6)(x) # Increased dropout  
x = Dense(64, activation='relu', kernel\_regularizer=tf.keras.regularizers.l2(0.01))(x)  
x = BatchNormalization()(x)  
x = Dropout(0.6)(x)  
prediction = Dense(2, activation='softmax')(x)  
model = Model(inputs=mobilenet.input, outputs=prediction)  
  
# Compile the model with a lower learning rate and additional metrics  
model.compile(  
 loss='categorical\_crossentropy',  
 optimizer=tf.keras.optimizers.Adam(learning\_rate=0.00005), # Reduced learning rate  
 metrics=[  
 'accuracy',  
 Precision(name='precision'),  
 Recall(name='recall'),  
 F1Score(num\_classes=2, average='macro', name='f1\_score')  
 ]  
)  
  
# Display model summary  
model.summary()  
  
# Enhanced data augmentation and preprocessing  
train\_datagen = ImageDataGenerator(  
 preprocessing\_function=preprocess\_input,  
 rotation\_range=20,  
 width\_shift\_range=0.2,  
 height\_shift\_range=0.2,  
 shear\_range=0.2,  
 zoom\_range=0.2,  
 horizontal\_flip=True,  
 brightness\_range=[0.8, 1.2],  
 fill\_mode='nearest'  
)  
  
test\_datagen = ImageDataGenerator(  
 preprocessing\_function=preprocess\_input  
)  
  
# Create data generators  
training\_set = train\_datagen.flow\_from\_directory(  
 train\_path,  
 target\_size=(224, 224),  
 batch\_size=32,  
 class\_mode='categorical',  
 shuffle=True  
)  
  
test\_set = test\_datagen.flow\_from\_directory(  
 test\_path,  
 target\_size=(224, 224),  
 batch\_size=32,  
 class\_mode='categorical',  
 shuffle=False  
)  
  
# Adjusted class weights to prioritize PNEUMONIA class  
class\_weights = {  
 0: 1.5, # PNEUMONIA (increased weight to reduce false negatives)  
 1: 1.0 # NORMAL  
}  
  
# Define callbacks for early stopping and learning rate scheduling  
early\_stopping = EarlyStopping(  
 monitor='val\_loss',  
 patience=5,  
 restore\_best\_weights=True  
)  
  
lr\_scheduler = ReduceLROnPlateau(  
 monitor='val\_loss',  
 factor=0.5,  
 patience=3,  
 min\_lr=1e-6,  
 verbose=1  
)  
  
# Train the model with callbacks  
r = model.fit(  
 training\_set,  
 validation\_data=test\_set,  
 epochs=25,  
 steps\_per\_epoch=len(training\_set),  
 validation\_steps=len(test\_set),  
 class\_weight=class\_weights,  
 callbacks=[early\_stopping, lr\_scheduler]  
)  
  
# Save the model  
model.save(os.path.join(save\_folder, 'chest\_xray\_mobilenet\_updated.h5'))  
  
# Load the model for prediction  
model = tf.keras.models.load\_model(os.path.join(save\_folder, 'chest\_xray\_mobilenet\_updated.h5'))  
  
# Test the model with a sample image  
img\_path = r'C:\Users\PC\Desktop\New folder (3)\chest\_xray\test\PNEUMONIA\BACTERIA-227418-0002.jpeg'  
try:  
 img = image\_utils.load\_img(img\_path, target\_size=(224, 224))  
 x = image\_utils.img\_to\_array(img)  
 x = np.expand\_dims(x, axis=0)  
 img\_data = preprocess\_input(x)  
 classes = model.predict(img\_data)  
 result = np.argmax(classes[0])  
 confidence = classes[0][result] \* 100  
 if result == 0:  
 print(f"Person is Affected By PNEUMONIA (Confidence: {confidence:.2f}%)")  
 else:  
 print(f"Result is Normal (Confidence: {confidence:.2f}%)")  
except FileNotFoundError:  
 print(f"Error: Image file {img\_path} not found. Skipping prediction.")  
  
# Evaluate the model on the test set  
test\_set.reset()  
y\_pred = model.predict(test\_set)  
y\_pred\_classes = np.argmax(y\_pred, axis=1)  
y\_true = test\_set.classes  
  
# Generate confusion matrix  
cm = confusion\_matrix(y\_true, y\_pred\_classes)  
  
# Plot and save confusion matrix  
plt.figure(figsize=(8, 6))  
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['PNEUMONIA', 'NORMAL'], yticklabels=['PNEUMONIA', 'NORMAL'])  
plt.title('Confusion Matrix')  
plt.xlabel('Predicted')  
plt.ylabel('True')  
plt.savefig(os.path.join(save\_folder, 'confusion\_matrix\_updated.png'))  
  
# Print classification report for evaluation metrics  
print("\nClassification Report:")  
print(classification\_report(y\_true, y\_pred\_classes, target\_names=['PNEUMONIA', 'NORMAL']))  
  
# Plot training results  
plt.figure(figsize=(15, 10))  
  
# Loss plot  
plt.subplot(2, 2, 1)  
plt.plot(r.history['loss'], label='train loss')  
plt.plot(r.history['val\_loss'], label='val loss')  
plt.title('Model Loss')  
plt.xlabel('Epoch')  
plt.ylabel('Loss')  
plt.legend()  
  
# Accuracy plot  
plt.subplot(2, 2, 2)  
plt.plot(r.history['accuracy'], label='train accuracy')  
plt.plot(r.history['val\_accuracy'], label='val accuracy')  
plt.title('Model Accuracy')  
plt.xlabel('Epoch')  
plt.ylabel('Accuracy')  
plt.legend()  
  
# Precision plot  
plt.subplot(2, 2, 3)  
plt.plot(r.history['precision'], label='train precision')  
plt.plot(r.history['val\_precision'], label='val precision')  
plt.title('Model Precision')  
plt.xlabel('Epoch')  
plt.ylabel('Precision')  
plt.legend()  
  
# Recall plot  
plt.subplot(2, 2, 4)  
plt.plot(r.history['recall'], label='train recall')  
plt.plot(r.history['val\_recall'], label='val recall')  
plt.title('Model Recall')  
plt.xlabel('Epoch')  
plt.ylabel('Recall')  
plt.legend()  
  
plt.tight\_layout()  
plt.savefig(os.path.join(save\_folder, 'training\_metrics\_updated.png'))  
  
# Print final validation metrics  
print("Final validation metrics:")  
print(f"Accuracy: {r.history['val\_accuracy'][-1]:.4f}")  
print(f"Precision: {r.history['val\_precision'][-1]:.4f}")  
print(f"Recall: {r.history['val\_recall'][-1]:.4f}")  
print(f"F1 Score: {r.history['val\_f1\_score'][-1]:.4f}")