```python
import os
import cv2 # type: ignore
import numpy as np # type: ignore
import pandas as pd # type: ignore
import pickle
import random
import matplotlib.pyplot as plt # type: ignore
import tensorflow as tf # type: ignore
from tensorflow import keras
from tensorflow.keras import layers # type: ignore
from tensorflow.keras import models # type: ignore
from tensorflow.keras import metrics  # type: ignore
from tensorflow.keras.utils import to_categorical # type: ignore
from sklearn.utils.class_weight import compute_class_weight # type: ignore
from tensorflow.keras.preprocessing.image import ImageDataGenerator   # type: ignore
from tensorflow.keras.callbacks import EarlyStopping  # type: ignore
# Preliminary steps to arrange, sort, and pickle training/validation data (Only done once)

'''# Define the main training/validation directory and subfolders
val_dir = '/Users/121-us-esumpter25-mba/Desktop/ISPPROJECT/val'
val_subfolders = ['benign', 'malignant']

# Collect data from the training/val directory (for the entire dataset, no image limit)
val_data = []

for label in val_subfolders:
    folder_path = os.path.join(val_dir, label)
    if os.path.exists(folder_path):
        for filename in os.listdir(folder_path):
            if filename.endswith('.jpeg'):
                image_path = os.path.join(folder_path, filename)
                val_data.append([image_path, label])

val_df = pd.DataFrame(val_data, columns=['filename', 'label'])

# Check how many training/val images are in the dataset
print(f"Validation set contains {len(val_df)} images.")

# Preprocess images for training/Val (Resize and normalize)
x_val = []
y_val = []

for index, row in val_df.iterrows():
    img_path = row['filename']
    img = cv2.imread(img_path)
    img = cv2.resize(img, (224, 224))  # Resize image
    img = img / 255.0  # Normalize to [0, 1]
    x_val.append(img)
    y_val.append(1 if row['label'] == 'malignant' else 0)

# Convert to numpy arrays
x_val = np.array(x_val)
y_val = np.array(y_val)

# Store the data in a pickle file for later use
val_data_dict = {'X': x_val, 'Y': y_val}

# Try pickling the data
try:
    with open('val_full_dataset.pkl', 'wb') as f:
        pickle.dump(val_data_dict, f)
    print("Data successfully pickled.")
except Exception as e:
    print(f"Error during pickling: {e}")'''
# Load the pickled training and validation data
with open('train_full_dataset.pkl', 'rb') as f:
    train_data_dict = pickle.load(f)
with open('val_full_dataset.pkl', 'rb') as f:
    val_data_dict = pickle.load(f)

# Access the training and validation data
x_train = train_data_dict['X']
y_train = train_data_dict['Y']
x_val = val_data_dict['X']
y_val = val_data_dict['Y']

print(f"Loaded {len(x_train)} training images.")

'''# Visualize a randomly selected training image
if len(x_train) > 0:
    image_number = random.randint(0, len(x_train) - 1)
    plt.imshow(x_train[image_number])
    plt.axis('off')
    plt.title(f"Training Image {image_number}, Label: {'Malignant' if y_train[image_number] == 1 else 'Benign'}")
    plt.show()
else:
    print("No training images loaded.")'''

print("enter one hot encoding")
# One Hot Encoding
num_classes = 2
y_train = to_categorical(y_train, num_classes)
y_val = to_categorical(y_val, num_classes)

print("Y Train Shape:", y_train.shape)
_, height, width, depth = x_train.shape
input_shape = (height, width, depth)
print(f"Input Shape:", input_shape)


#Data Augmentation
```

```
datagen = ImageDataGenerator(
    horizontal_flip=True,  # Flip horizontally ✔
    rotation_range=20,     # Add random rotation (0-20 degrees) ✔
    width_shift_range=0.2, # Shift horizontally ✔
    height_shift_range=0.2,# Shift vertically ✔
    zoom_range=0.2,        # Random zoom ✔
    shear_range=0.2,       # Shear transformation ✔
    fill_mode='nearest'    # Fill the pixels with nearest value ✔
)
datagen.fit(x_train)

print("enter cnn")
#CNN
def get_model():
    model = keras.Sequential([
        #BLOCK 1
        layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=input_shape),
        layers.MaxPooling2D((2, 2)),

        #BLOCK 2
        layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),

        #BLOCK 3
        layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
        layers.MaxPooling2D((2, 2)),

        #CLASSIFIER
        layers.Flatten(),
        layers.Dense(64, activation='relu'),
        layers.Dropout(0.3),
        layers.Dense(2, activation='softmax')
    ])
    return model

model = get_model()
model.summary()

early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)


print("enter model")
#Training (Hyperparameters, Compile/Train, Run Inference, Visualise, Test, Save)

#Defining Hyperparameters
# num_epochs - number of times to repeat the training
num_epochs = 20

# batch_size - how many images to train together at each step
batch_size = 32

# learning_rate - the update speed of the network
learning_rate = 1e-3

# Compile and Train
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(learning_rate=learning_rate), metrics=['accuracy', metrics.Precision(), metrics.Recall(), metrics.AUC()])


#Train
history = model.fit(
          datagen.flow(x_train, y_train,batch_size=batch_size),
          validation_data=(x_val, y_val),
          epochs=num_epochs,
          callbacks=[early_stop],
          verbose=1)
#Run Inference on Training Data
img_index = 7

print(f"Running Model Inference on Image number {img_index}")
img_array = x_train[img_index]
expanded_img_array = np.expand_dims(img_array, axis=0)

actual_label = y_train[img_index]
actual_label = np.argmax(actual_label)

# Make predictions using the model
predictions = model.predict(expanded_img_array)
predicted_label = np.argmax(predictions[0])  # Get the predicted label

print("Predicted Label:", predicted_label)
print("Actual Label:", actual_label)

# PLOT GRAPH
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accuracy')
plt.plot(epochs, val_acc, 'r', label='Validation accuracy')
plt.title('Training and Validation accuracy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
```

```
plt.legend()
plt.show()
```