Auto WCEBleedGen Competetion

Bleeding / Non-Bleeding

The main goal of this challenge was to create a classification and detection model that could differentiate between bleeding and non-bleeding frames in wireless capsule endoscopy (WCE) images.

Importing all the Required Librarires

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from sklearn.model_selection import train_test_split

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.applications import MobileNetV2
from keras.optimizers import Adam
from keras.metrics import Accuracy, Recall, Precision
from keras.layers import Dense, GlobalAveragePooling2D
from keras.callbacks import ModelCheckpoint, CSVLogger
from keras.regularizers import l2
from keras.applications import InceptionV3
import os
import cv2
from PIL import Image
import numpy as np
```

Obtaining the Data shape and Label

```

# Define the image directory
image_directory = '/content/drive/MyDrive/final/WCEBleedGen/'

# Define the image size
SIZE = 224

# Initialize lists to store the dataset and labels
dataset = []
labels = []

# Function to read and preprocess images
def process_images(image_dir, label_value):
    for image_name in os.listdir(image_dir):
        if image_name.endswith('.png'):
            image_path = os.path.join(image_dir, image_name)
            try:
                image = cv2.imread(image_path)
                if image is not None:
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                    image = cv2.resize(image, (SIZE, SIZE))
                    dataset.append(image)
                    labels.append(label_value)
                else:
                    print(f"Unable to load image: {image_path}")
            except Exception as e:
                print(f"Error processing {image_path}: {str(e)}")

# Process bleeding images
bleeding_directory = os.path.join(image_directory, 'bleeding/images')
process_images(bleeding_directory, label_value=1)

# Process non-bleeding images
non_bleeding_directory = os.path.join(image_directory, 'non-bleeding/images')
process_images(non_bleeding_directory, label_value=0)

# Convert dataset and labels to numpy arrays
dataset = np.array(dataset)
```

```
labels = np.array(labels)

# Print dataset and label shapes for verification
print('Dataset shape:', dataset.shape)
print('Label shape:', labels.shape)

```

```
Dataset shape: (2618, 224, 224, 3)
Label shape: (2618,)
```

Random image from the dataset and its label

```
import random
import matplotlib.pyplot as plt

# Generate a random image number
image_number = random.randint(0, len(dataset) - 1)

# Display the image and its label
plt.imshow(dataset[image_number])
plt.title("Label for this image is: " + str(labels[image_number]))  # Use 'labels' instead of 'label'
plt.show()

```

Using DenseNet 121 Architecture to Extract Features

```
import os
import cv2
import numpy as np
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.models import Model
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.densenet import preprocess_input
from tqdm import tqdm

# Define the image directories
image_directory = '/content/drive/MyDrive/final/WCEBleedGen/'
output_directory = '/content/drive/MyDrive/final/extracted_features/'

# Define the image size
SIZE = (224, 224)

# Create output directory if it doesn't exist
os.makedirs(output_directory, exist_ok=True)

# Load the DenseNet-121 model with pre-trained weights
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Remove the top classification layers
```

```
x = base_model.output
x = GlobalAveragePooling2D()(x)

# Create a model for feature extraction
model = Model(inputs=base_model.input, outputs=x)

# Function to extract features from an image
def extract_features(image_path):
    img = image.load_img(image_path, target_size=SIZE)
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    features = model.predict(img)
    return features

# Initialize lists to store features and labels
features_list = []
labels_list = []

# Process bleeding images
bleeding_directory = os.path.join(image_directory, 'bleeding/images')

for image_name in tqdm(os.listdir(bleeding_directory)):
    if image_name.endswith('.png'):
        image_path = os.path.join(bleeding_directory, image_name)
        features = extract_features(image_path)
        features_list.append(features)
        labels_list.append(1)  # Label 1 for bleeding

# Process non-bleeding images
non_bleeding_directory = os.path.join(image_directory, 'non-bleeding/images')

for image_name in tqdm(os.listdir(non_bleeding_directory)):
    if image_name.endswith('.png'):
        image_path = os.path.join(non_bleeding_directory, image_name)
        features = extract_features(image_path)
        features_list.append(features)
        labels_list.append(0)  # Label 0 for non-bleeding

# Convert lists to NumPy arrays
features_array = np.vstack(features_list)
labels_array = np.array(labels_list)

# Save the extracted features and labels in the "extracted_features" folder
np.save(os.path.join(output_directory, 'features.npy'), features_array)
np.save(os.path.join(output_directory, 'labels.npy'), labels_array)

print("Feature extraction and label saving completed.")

```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_ke
29084464/29084464 [==============================] - 0s 0us/step
  0%|          | 0/1311 [00:00<?, ?it/s]1/1 [==============================] - 3s 3s/step
  0%|          | 1/1311 [00:03<1:06:35,  3.05s/it]1/1 [==============================] - 0s 204ms/step
  0%|          | 2/1311 [00:03<36:11,  1.66s/it]  1/1 [==============================] - 0s 163ms/step
  0%|          | 3/1311 [00:04<27:31,  1.26s/it]1/1 [==============================] - 0s 178ms/step
  0%|          | 4/1311 [00:05<21:34,  1.01it/s]1/1 [==============================] - 0s 166ms/step
  0%|          | 5/1311 [00:05<18:22,  1.18it/s]1/1 [==============================] - 0s 178ms/step
  0%|          | 6/1311 [00:06<16:19,  1.33it/s]1/1 [==============================] - 0s 312ms/step
  1%|          | 7/1311 [00:07<18:11,  1.19it/s]1/1 [==============================] - 0s 302ms/step
  1%|          | 8/1311 [00:08<18:08,  1.20it/s]1/1 [==============================] - 0s 302ms/step
  1%|          | 9/1311 [00:08<16:59,  1.28it/s]1/1 [==============================] - 0s 298ms/step
  1%|          | 10/1311 [00:09<15:55,  1.36it/s]1/1 [==============================] - 0s 287ms/step
  1%|          | 11/1311 [00:10<15:16,  1.42it/s]1/1 [==============================] - 0s 171ms/step
  1%|          | 12/1311 [00:10<15:58,  1.35it/s]1/1 [==============================] - 0s 170ms/step
  1%|          | 13/1311 [00:11<13:44,  1.57it/s]1/1 [==============================] - 1s 563ms/step
  1%|          | 14/1311 [00:12<16:24,  1.32it/s]1/1 [==============================] - 0s 334ms/step
  1%|          | 15/1311 [00:30<2:07:54,  5.92s/it]1/1 [==============================] - 0s 167ms/step
  1%|          | 16/1311 [00:30<1:31:48,  4.25s/it]1/1 [==============================] - 0s 169ms/step
  1%|▏         | 17/1311 [00:30<1:06:43,  3.09s/it]1/1 [==============================] - 0s 183ms/step
  1%|▏         | 18/1311 [00:31<49:07,  2.28s/it]  1/1 [==============================] - 0s 169ms/step
  1%|▏         | 19/1311 [00:31<36:48,  1.71s/it]1/1 [==============================] - 0s 183ms/step
  2%|▏         | 20/1311 [00:32<28:11,  1.31s/it]1/1 [==============================] - 0s 170ms/step
  2%|▏         | 21/1311 [00:32<21:13,  1.01it/s]1/1 [==============================] - 0s 173ms/step
  2%|▏         | 22/1311 [00:32<16:22,  1.31it/s]1/1 [==============================] - 0s 171ms/step
  2%|▏         | 23/1311 [00:32<13:54,  1.54it/s]1/1 [==============================] - 0s 168ms/step
  2%|▏         | 24/1311 [00:33<12:10,  1.76it/s]1/1 [==============================] - 0s 169ms/step
  2%|▏         | 25/1311 [00:33<10:02,  2.13it/s]1/1 [==============================] - 0s 170ms/step
  2%|▏         | 26/1311 [00:33<09:31,  2.25it/s]1/1 [==============================] - 0s 299ms/step
  2%|▏         | 27/1311 [00:34<09:24,  2.28it/s]1/1 [==============================] - 0s 319ms/step
```

```
  2%|▏          | 28/1311 [00:34<09:13,  2.32it/s]1/1 [==============================] - 0s 298ms/step
  2%|▏          | 29/1311 [00:35<08:59,  2.37it/s]1/1 [==============================] - 0s 334ms/step
  2%|▏          | 30/1311 [00:35<08:59,  2.37it/s]1/1 [==============================] - 0s 299ms/step
  2%|▏          | 31/1311 [00:36<08:53,  2.40it/s]1/1 [==============================] - 0s 294ms/step
  2%|▏          | 32/1311 [00:36<09:00,  2.36it/s]1/1 [==============================] - 0s 298ms/step
  3%|▏          | 33/1311 [00:36<09:03,  2.35it/s]1/1 [==============================] - 0s 364ms/step
  3%|▏          | 34/1311 [00:37<09:27,  2.25it/s]1/1 [==============================] - 0s 344ms/step
  3%|▏          | 35/1311 [00:37<09:33,  2.23it/s]1/1 [==============================] - 0s 320ms/step
  3%|▏          | 36/1311 [00:38<09:34,  2.22it/s]1/1 [==============================] - 0s 343ms/step
  3%|▏          | 37/1311 [00:38<09:45,  2.18it/s]1/1 [==============================] - 0s 344ms/step
  3%|▏          | 38/1311 [00:39<09:40,  2.19it/s]1/1 [==============================] - 0s 314ms/step
  3%|▏          | 39/1311 [00:39<09:22,  2.26it/s]1/1 [==============================] - 0s 343ms/step
  3%|▏          | 40/1311 [00:40<09:43,  2.18it/s]1/1 [==============================] - 0s 311ms/step
  3%|▏          | 41/1311 [00:40<09:28,  2.23it/s]1/1 [==============================] - 0s 305ms/step
  3%|▏          | 42/1311 [00:40<09:17,  2.28it/s]1/1 [==============================] - 0s 293ms/step
  3%|▏          | 43/1311 [00:41<09:11,  2.30it/s]1/1 [==============================] - 0s 318ms/step
  3%|▏          | 44/1311 [00:41<09:14,  2.29it/s]1/1 [==============================] - 0s 303ms/step
  3%|▏          | 45/1311 [00:42<09:07,  2.31it/s]1/1 [==============================] - 0s 245ms/step
  4%|▏          | 46/1311 [00:42<08:28,  2.49it/s]1/1 [==============================] - 0s 168ms/step
  4%|▏          | 47/1311 [00:42<07:25,  2.84it/s]1/1 [==============================] - 0s 174ms/step
  4%|▏          | 48/1311 [00:43<06:41,  3.14it/s]1/1 [==============================] - 0s 166ms/step
  4%|▏          | 49/1311 [00:43<06:12,  3.39it/s]1/1 [==============================] - 0s 184ms/step
  4%|▏          | 50/1311 [00:43<06:45,  3.11it/s]1/1 [==============================] - 0s 182ms/step
  4%|▏          | 51/1311 [00:43<06:17,  3.34it/s]1/1 [==============================] - 0s 179ms/step
  4%|▏          | 52/1311 [00:44<05:54,  3.55it/s]1/1 [==============================] - 0s 185ms/step
  4%|▏          | 53/1311 [00:44<06:32,  3.20it/s]1/1 [==============================] - 0s 174ms/step
  4%|▏          | 54/1311 [00:44<07:02,  2.98it/s]1/1 [==============================] - 0s 176ms/step
```

CNN2d Model is being used to train the model.

```
1
```

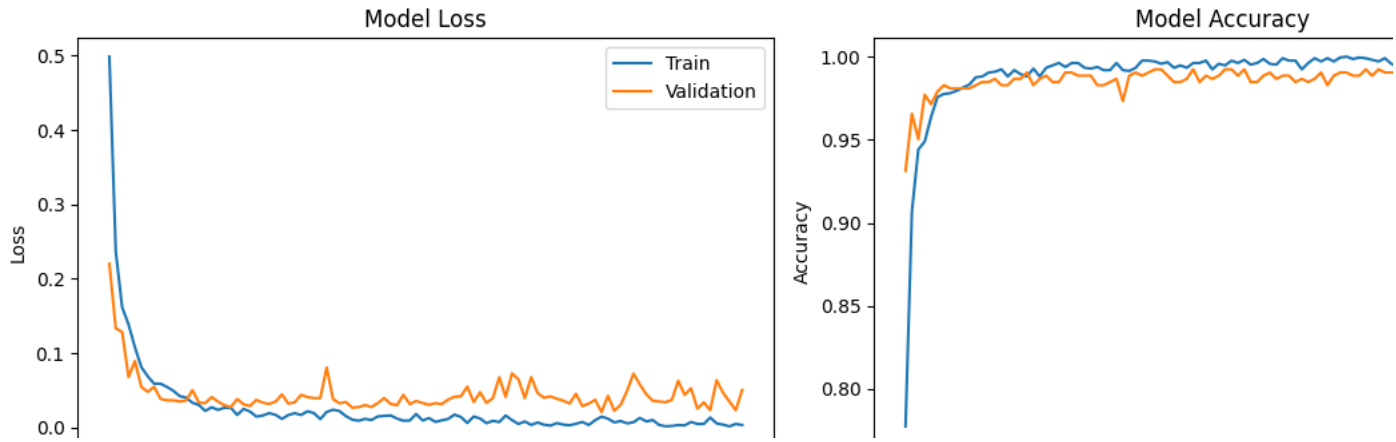
```
import os
import numpy as np
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from keras.utils import to_categorical

# Load your extracted features (2618, 1, 1024)
features_array = np.load('/content/drive/MyDrive/final/extracted_features/features.npy')
labels_array = np.load('/content/drive/MyDrive/final/extracted_features/labels.npy')

# Reshape the features into (2618, 32, 32, 1)
features_array_2d = features_array.reshape((2618, 32, 32, 1))

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(
    features_array_2d, labels_array, test_size=0.2, random_state=42)

# Convert labels to one-hot encoding
y_train = to_categorical(y_train)
y_val = to_categorical(y_val)

# Define the 2D CNN model
model = Sequential([s
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(labels_array)), activation='softmax')
])

# Compile the model
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=100, validation_data=(X_val, y_val))

# Evaluate the model
loss, accuracy = model.evaluate(X_val, y_val)
print(f"Validation Loss: {loss:.4f}, Validation Accuracy: {accuracy:.4f}")

```

```
Epoch 61/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0091 - accuracy: 0.9976 - val_loss: 0.0396 - val_accuracy: 0.9885
Epoch 62/100
66/66 [==============================] - 1s 23ms/step - loss: 0.0074 - accuracy: 0.9976 - val_loss: 0.0676 - val_accuracy: 0.9847
Epoch 63/100
66/66 [==============================] - 1s 23ms/step - loss: 0.0162 - accuracy: 0.9924 - val_loss: 0.0412 - val_accuracy: 0.9866
Epoch 64/100
66/66 [==============================] - 2s 36ms/step - loss: 0.0101 - accuracy: 0.9962 - val_loss: 0.0728 - val_accuracy: 0.9847
Epoch 65/100
66/66 [==============================] - 3s 38ms/step - loss: 0.0049 - accuracy: 0.9990 - val_loss: 0.0647 - val_accuracy: 0.9866
Epoch 66/100
66/66 [==============================] - 2s 24ms/step - loss: 0.0082 - accuracy: 0.9971 - val_loss: 0.0396 - val_accuracy: 0.9905
Epoch 67/100
66/66 [==============================] - 1s 23ms/step - loss: 0.0036 - accuracy: 0.9990 - val_loss: 0.0677 - val_accuracy: 0.9828
Epoch 68/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0069 - accuracy: 0.9971 - val_loss: 0.0464 - val_accuracy: 0.9885
Epoch 69/100
66/66 [==============================] - 1s 22ms/step - loss: 0.0038 - accuracy: 0.9995 - val_loss: 0.0400 - val_accuracy: 0.9905
Epoch 70/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0026 - accuracy: 1.0000 - val_loss: 0.0419 - val_accuracy: 0.9905
Epoch 71/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0061 - accuracy: 0.9986 - val_loss: 0.0387 - val_accuracy: 0.9885
Epoch 72/100
66/66 [==============================] - 2s 29ms/step - loss: 0.0041 - accuracy: 0.9995 - val_loss: 0.0362 - val_accuracy: 0.9885
Epoch 73/100
66/66 [==============================] - 2s 38ms/step - loss: 0.0031 - accuracy: 0.9990 - val_loss: 0.0324 - val_accuracy: 0.9924
Epoch 74/100
66/66 [==============================] - 3s 41ms/step - loss: 0.0052 - accuracy: 0.9981 - val_loss: 0.0454 - val_accuracy: 0.9885
Epoch 75/100
66/66 [==============================] - 2s 37ms/step - loss: 0.0076 - accuracy: 0.9971 - val_loss: 0.0288 - val_accuracy: 0.9924
Epoch 76/100
66/66 [==============================] - 2s 29ms/step - loss: 0.0035 - accuracy: 0.9990 - val_loss: 0.0322 - val_accuracy: 0.9905
Epoch 77/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0099 - accuracy: 0.9957 - val_loss: 0.0373 - val_accuracy: 0.9905
Epoch 78/100
66/66 [==============================] - 1s 22ms/step - loss: 0.0147 - accuracy: 0.9938 - val_loss: 0.0209 - val_accuracy: 0.9905
Epoch 79/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0120 - accuracy: 0.9947 - val_loss: 0.0427 - val_accuracy: 0.9866
Epoch 80/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0070 - accuracy: 0.9976 - val_loss: 0.0225 - val_accuracy: 0.9924
Epoch 81/100
66/66 [==============================] - 2s 36ms/step - loss: 0.0087 - accuracy: 0.9957 - val_loss: 0.0307 - val_accuracy: 0.9905
Epoch 82/100
66/66 [==============================] - 3s 39ms/step - loss: 0.0057 - accuracy: 0.9976 - val_loss: 0.0494 - val_accuracy: 0.9866
Epoch 83/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0072 - accuracy: 0.9976 - val_loss: 0.0725 - val_accuracy: 0.9828
Epoch 84/100
66/66 [==============================] - 1s 22ms/step - loss: 0.0130 - accuracy: 0.9938 - val_loss: 0.0578 - val_accuracy: 0.9847
Epoch 85/100
66/66 [==============================] - 1s 22ms/step - loss: 0.0083 - accuracy: 0.9971 - val_loss: 0.0448 - val_accuracy: 0.9885
Epoch 86/100
66/66 [==============================] - 1s 22ms/step - loss: 0.0102 - accuracy: 0.9962 - val_loss: 0.0361 - val_accuracy: 0.9866
Epoch 87/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0035 - accuracy: 0.9990 - val_loss: 0.0351 - val_accuracy: 0.9885
Epoch 88/100
66/66 [==============================] - 2s 23ms/step - loss: 0.0018 - accuracy: 1.0000 - val_loss: 0.0340 - val_accuracy: 0.9905
Epoch 89/100
66/66 [==============================] - 2s 27ms/step - loss: 0.0022 - accuracy: 0.9990 - val_loss: 0.0373 - val_accuracy: 0.9924
Epoch 90/100
```

Plot training & Validation loss values

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'], loc='upper right')

# Plot training & validation accuracy values
plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'Validation'], loc='lower right')
```

```
plt.tight_layout()
plt.show()
```



Required Metric Evaluation

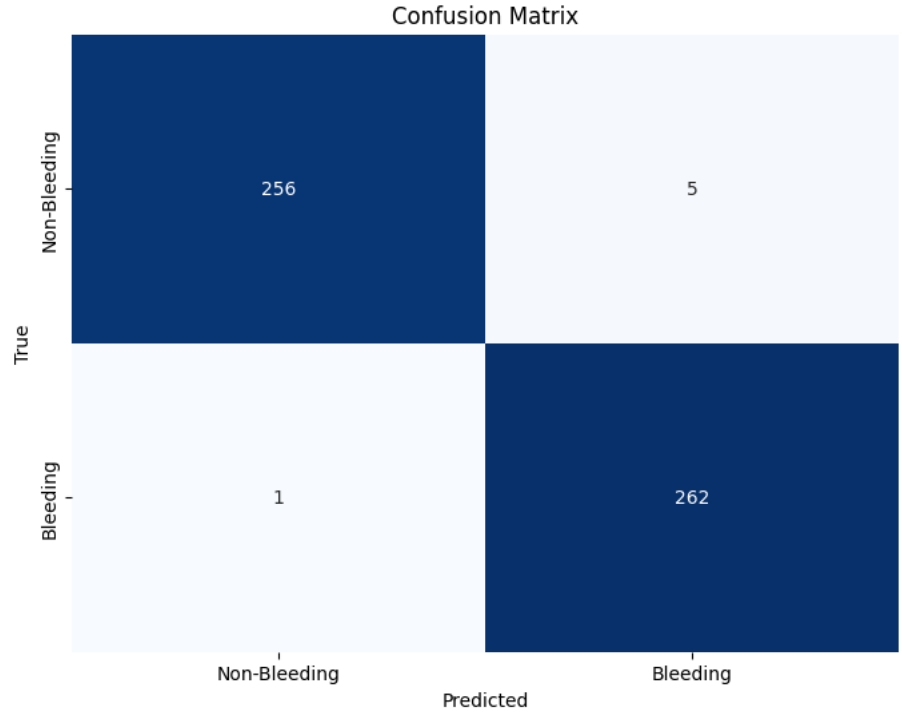
```
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, roc_auc_score, roc_curve, auc
import matplotlib.pyplot as plt

# Load your model and saved weights
model = Sequential([
    Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(32, 32, 1)),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(len(np.unique(labels_array)), activation='softmax')
])

model.load_weights('/content/drive/MyDrive/CNN2d.h5')  # Load your trained weights

# Predict on the validation data
y_pred = model.predict(X_val)

# Convert predicted probabilities to class labels (0 or 1)
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = np.argmax(y_val, axis=1)

# Calculate accuracy
accuracy = accuracy_score(y_true_labels, y_pred_labels)

# Calculate precision, recall, and F1 score
precision = precision_score(y_true_labels, y_pred_labels)
recall = recall_score(y_true_labels, y_pred_labels)
f1 = f1_score(y_true_labels, y_pred_labels)

# Calculate the confusion matrix
confusion = confusion_matrix(y_true_labels, y_pred_labels)

# Calculate ROC-AUC score and plot ROC curve
roc_auc = roc_auc_score(y_true_labels, y_pred_labels)
fpr, tpr, thresholds = roc_curve(y_true_labels, y_pred_labels)
roc_auc = auc(fpr, tpr)

# Print the metrics
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall (Sensitivity): {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
print("Confusion Matrix:")
print(confusion)
print(f"ROC AUC Score: {roc_auc:.4f}")
```

```
17/17 [==============================] - 0s 9ms/step
Accuracy: 0.9885
Precision: 0.9813
Recall (Sensitivity): 0.9962
F1 Score: 0.9887
Confusion Matrix:
[[256   5]
 [  1 262]]
ROC AUC Score: 0.9885
```

Confusion Matrix

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

# Predict on the validation data
y_pred = model.predict(X_val)

# Convert predicted probabilities to class labels (0 or 1)
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = np.argmax(y_val, axis=1)

# Calculate the confusion matrix
confusion = confusion_matrix(y_true_labels, y_pred_labels)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=["Non-Bleeding", "Bleeding"],
            yticklabels=["Non-Bleeding", "Bleeding"])
plt.xlabel("Predicted")
plt.ylabel("True")
plt.title("Confusion Matrix")
plt.show()

```

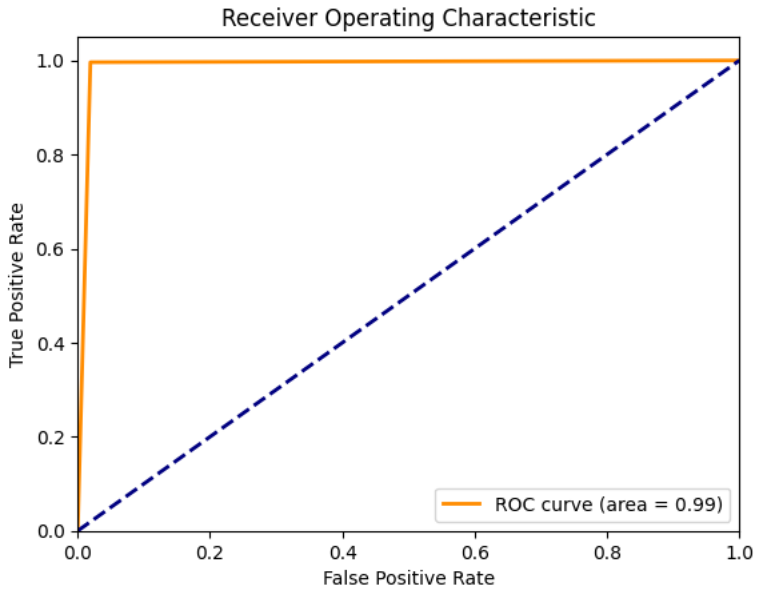
```
17/17 [==============================] - 0s 11ms/step
```



ROC AOC Curve

```
# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
```

```
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()
```



Kappa Score

```
from sklearn.metrics import cohen_kappa_score

# Calculate Cohen's Kappa
kappa = cohen_kappa_score(y_true_labels, y_pred_labels)

# Print the Kappa score
print(f"Cohen's Kappa Score: {kappa:.4f}")

```

```
Cohen's Kappa Score: 0.9771
```

Specificity

```
# Calculate True Negatives (TN), False Positives (FP), True Positives (TP), False Negatives (FN)
TN, FP, FN, TP = confusion.ravel()

# Calculate Specificity
specificity = TN / (TN + FP)

# Print the Specificity
print(f"Specificity: {specificity:.4f}")

```

```
Specificity: 0.9808
```

Model Saved (1.53mb)

```
model.save('CNN2d.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file via `m
  saving_api.save_model(
```

Using the saved model (CNN2d.h5), accurately predicting the image's label of Test Dataset 1 and saving it to the Predictions.xlsx

```
import os
import cv2
import numpy as np
from tensorflow.keras.models import load_model
import pandas as pd

# Load the saved model
model = load_model('/content/drive/MyDrive/CNN2d.h5')

# Define the path to your test dataset directory
test_dataset_dir = '/content/drive/MyDrive/test_dataset/Test Dataset 1'

# List all image files in the test dataset directory
test_image_files = [f for f in os.listdir(test_dataset_dir) if f.endswith('.png')]

# Initialize lists to store image names and predicted labels
image_names = []
predicted_labels = []

# Iterate over the test images, convert to grayscale, resize, make predictions, and store results
for image_file in test_image_files:
    # Load and preprocess the test image
    image_path = os.path.join(test_dataset_dir, image_file)
    img = cv2.imread(image_path)

    if img is not None:
        # Convert the image to grayscale
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Resize the image to 32x32 pixels
        img_gray = cv2.resize(img_gray, (32, 32))

        # Reshape to (32, 32, 1)
        img_gray = img_gray.reshape(32, 32, 1)

        # Make a prediction using the loaded model
        prediction = model.predict(np.expand_dims(img_gray, axis=0))

        # Determine the predicted label by selecting the class with the highest probability
        predicted_label = np.argmax(prediction, axis=1)

        # Map the class index to the corresponding label (e.g., 0 for non-bleeding, 1 for bleeding)
        if predicted_label == 0:
            label = "non-bleeding"
        else:
            label = "bleeding"

        # Append image name and predicted label to the lists
        image_names.append(image_file)
        predicted_labels.append(label)
    else:
        print(f"Image: {image_file}, Shape: Unable to read the image")

# Create a DataFrame to store the results
df = pd.DataFrame({'Image Name': image_names, 'Predicted Label': predicted_labels})

# Save the DataFrame to an Excel file
df.to_excel('/content/drive/MyDrive/predictions.xlsx', index=False)

print("Predictions saved to predictions.xlsx")

```

```
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 72ms/step
1/1 [==============================] - 0s 74ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 75ms/step
1/1 [==============================] - 0s 115ms/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 135ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 41ms/step
1/1 [==============================] - 0s 93ms/step
1/1 [==============================] - 0s 78ms/step
1/1 [==============================] - 0s 86ms/step
```

```
1/1 [==============================] - 0s 76ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 87ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 86ms/step
1/1 [==============================] - 0s 108ms/step
1/1 [==============================] - 0s 63ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
Predictions saved to predictions.xlsx
```

Using the saved model (CNN2d.h5), accurately predicting the image's label of Test Dataset 2 and saving it to the test_2Predictions.xlsx

```
import os
import cv2
import numpy as np
from tensorflow.keras.models import load_model
import pandas as pd

# Load the saved model
model = load_model('/content/drive/MyDrive/CNN2d.h5')

# Define the path to your test dataset directory
test_dataset_dir = '/content/drive/MyDrive/test_dataset/Test Dataset 2'

# List all image files in the test dataset directory
test_image_files = [f for f in os.listdir(test_dataset_dir) if f.endswith('.png')]

# Initialize lists to store image names and predicted labels
image_names = []
predicted_labels = []

# Iterate over the test images, convert to grayscale, resize, make predictions, and store results
for image_file in test_image_files:
    # Load and preprocess the test image
    image_path = os.path.join(test_dataset_dir, image_file)
    img = cv2.imread(image_path)

    if img is not None:
        # Convert the image to grayscale
        img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        # Resize the image to 32x32 pixels
        img_gray = cv2.resize(img_gray, (32, 32))

        # Reshape to (32, 32, 1)
        img_gray = img_gray.reshape(32, 32, 1)

        # Make a prediction using the loaded model
        prediction = model.predict(np.expand_dims(img_gray, axis=0))

        # Determine the predicted label by selecting the class with the highest probability
        predicted_label = np.argmax(prediction, axis=1)

```

```
        # Map the class index to the corresponding label (e.g., 0 for non-bleeding, 1 for bleeding)
        if predicted_label == 0:
            label = "non-bleeding"
        else:
            label = "bleeding"

        # Append image name and predicted label to the lists
        image_names.append(image_file)
        predicted_labels.append(label)
    else:
        print(f"Image: {image_file}, Shape: Unable to read the image")

# Create a DataFrame to store the results
df = pd.DataFrame({'Image Name': image_names, 'Predicted Label': predicted_labels})

# Save the DataFrame to an Excel file
df.to_excel('/content/drive/MyDrive/test_2predictions.xlsx', index=False)

print("Predictions saved to predictions.xlsx")

```

```
1/1 [==============================] - 0s 146ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 49ms/step
1/1 [==============================] - 0s 118ms/step
1/1 [==============================] - 0s 56ms/step
1/1 [==============================] - 0s 65ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 43ms/step
1/1 [==============================] - 0s 59ms/step
1/1 [==============================] - 0s 38ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 32ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 26ms/step
```

Using the saved model (CNN2d.h5), accurately predicting the image's label of Validation dataset and saving it to the Predicted_images

```
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image

# Load the saved model
model = load_model('/content/drive/MyDrive/CNN2d.h5')  # Replace with the path to your saved model

# Define the image directories
image_directory = '/content/drive/MyDrive/final/WCEBleedGen/'

# Lists to store the file paths of predicted images, true labels, and predicted labels
predicted_images = []
true_labels = []
predicted_labels = []

# Process both bleeding and non-bleeding images
bleeding_directory = os.path.join(image_directory, 'bleeding/images')
non_bleeding_directory = os.path.join(image_directory, 'non-bleeding/images')

# Create a folder to save the predicted images
output_folder = '/content/drive/MyDrive/predicted_images'
os.makedirs(output_folder, exist_ok=True)

# Function to make predictions on an image and get the label
def predict_image(image_path):
    img = image.load_img(image_path, target_size=(32, 32), color_mode="grayscale")  # Load as grayscale
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)

    prediction = model.predict(img)
    predicted_label = np.argmax(prediction)
    label = "Bleeding" if predicted_label == 1 else "Non-Bleeding"

    return label

# Process images and make predictions
for image_name in os.listdir(bleeding_directory):
    if image_name.endswith('.png'):
        image_path = os.path.join(bleeding_directory, image_name)
        predicted_images.append(image_path)
        true_labels.append("Bleeding")
        predicted_labels.append(predict_image(image_path))

for image_name in os.listdir(non_bleeding_directory):
    if image_name.endswith('.png'):
        image_path = os.path.join(non_bleeding_directory, image_name)
        predicted_images.append(image_path)
        true_labels.append("Non-Bleeding")
        predicted_labels.append(predict_image(image_path))

# Combine images, true labels, and predicted labels
predictions = list(zip(predicted_images, true_labels, predicted_labels))

# Sort by prediction confidence
predictions.sort(key=lambda x: x[2] == "Bleeding", reverse=True)

# Save the 10 best-predicted images with labels to the output folder
for i, (image_path, true_label, predicted_label) in enumerate(predictions[:10]):
    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    text = f"True Label: {true_label}, Predicted Label: {predicted_label}"

    # Add the text to the image
    img = cv2.putText(img, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2)

    output_path = os.path.join(output_folder, f'predicted_{i+1}.png')
    cv2.imwrite(output_path, img)

print("Predicted images with labels saved to the output folder.")

```

```
1/1 [==============================] - 0s 130ms/step
1/1 [==============================] - 0s 52ms/step
```

```
1/1 [==============================] - 0s 77ms/step
1/1 [==============================] - 0s 60ms/step
1/1 [==============================] - 0s 137ms/step
1/1 [==============================] - 0s 55ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 44ms/step
1/1 [==============================] - 0s 69ms/step
1/1 [==============================] - 0s 174ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 94ms/step
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 96ms/step
1/1 [==============================] - 0s 97ms/step
1/1 [==============================] - 0s 178ms/step
1/1 [==============================] - 0s 138ms/step
1/1 [==============================] - 0s 107ms/step
1/1 [==============================] - 0s 83ms/step
1/1 [==============================] - 0s 127ms/step
1/1 [==============================] - 0s 92ms/step
1/1 [==============================] - 0s 106ms/step
1/1 [==============================] - 0s 110ms/step
1/1 [==============================] - 0s 88ms/step
1/1 [==============================] - 0s 75ms/step
1/1 [==============================] - 0s 121ms/step
1/1 [==============================] - 0s 89ms/step
1/1 [==============================] - 0s 36ms/step
1/1 [==============================] - 0s 46ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 39ms/step
1/1 [==============================] - 0s 42ms/step
1/1 [==============================] - 0s 40ms/step
1/1 [==============================] - 0s 48ms/step
1/1 [==============================] - 0s 53ms/step
1/1 [==============================] - 0s 45ms/step
1/1 [==============================] - 0s 132ms/step
1/1 [==============================] - 0s 101ms/step
1/1 [==============================] - 0s 84ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 21ms/step
```

Using the saved model (CNN2d.h5), accurately predicting the image's label of Test Dataset 1 and saving it to the Predicted_test images

```
import cv2
import numpy as np
import random
import os
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model

# Load the saved model
model = load_model('/content/drive/MyDrive/CNN2d.h5')  # Replace with the path to your saved model

# Path to the test dataset directory
test_dataset_path = '/content/drive/MyDrive/test_dataset/Test Dataset 1/'

# Get a list of image file paths in the test dataset directory
image_paths = [f for f in os.listdir(test_dataset_path) if f.endswith('.png')]

# Create a folder to save the predicted images
output_folder = '/content/drive/MyDrive/predicted_test_images'
os.makedirs(output_folder, exist_ok=True)

# Iterate over the randomly selected images
for i, image_path in enumerate(image_paths):
```

```
    # Load and preprocess the test image
    test_image_path = os.path.join(test_dataset_path, image_path)
    test_image = cv2.imread(test_image_path)
    test_image_rgb = cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB)  # Convert to RGB format for display

    # Make predictions using the model
    grayscale_image = cv2.cvtColor(test_image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
    grayscale_image = cv2.resize(grayscale_image, (32, 32))  # Resize to model's input size
    grayscale_image = np.expand_dims(grayscale_image, axis=0)  # Add batch dimension
    prediction = model.predict(grayscale_image)
    predicted_label = "Bleeding" if np.argmax(prediction) == 1 else "Non-Bleeding"



    # Save the predicted images in the output folder
    output_path = os.path.join(output_folder, f'predicted_{i+1}.png')
    cv2.imwrite(output_path, cv2.cvtColor(test_image_rgb, cv2.COLOR_RGB2BGR))


print("Predicted images saved to the output folder.")

```

```
1/1 [==============================] - 0s 237ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 18ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 21ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 20ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 20ms/step
Predicted images saved to the output folder.
```

Using the saved model (CNN2d.h5), accurately predicting the image's label of Test Dataset 2 and saving it to the Predicted_test_2_images

```
import cv2
import numpy as np
import random
import os
from tensorflow.keras.models import load_model
```

```

# Load the saved model
model = load_model('/content/drive/MyDrive/CNN2d.h5')  # Replace with the path to your saved model

# Path to the test dataset directory
test_dataset_path = '/content/drive/MyDrive/final/WCEBleedGen/non-bleeding/images/'

# Get a list of image file paths in the test dataset directory
image_paths = [f for f in os.listdir(test_dataset_path) if f.endswith('.png')]

# Select 5 random images from the list
random_image_paths = random.sample(image_paths, 10)

# Create a folder to save the predicted images
output_folder = '/content/drive/MyDrive/predicted_test_2_images'
os.makedirs(output_folder, exist_ok=True)

# Iterate over the randomly selected images
for i, image_path in enumerate(random_image_paths):
    # Load and preprocess the test image
    test_image_path = os.path.join(test_dataset_path, image_path)
    test_image = cv2.imread(test_image_path)
    test_image_rgb = cv2.cvtColor(test_image, cv2.COLOR_BGR2RGB)  # Convert to RGB format for display

    # Make predictions using the model
    grayscale_image = cv2.cvtColor(test_image, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
    grayscale_image = cv2.resize(grayscale_image, (32, 32))  # Resize to model's input size
    grayscale_image = np.expand_dims(grayscale_image, axis=0)  # Add batch dimension
    prediction = model.predict(grayscale_image)
    predicted_label = "Bleeding" if np.argmax(prediction) == 1 else "Non-Bleeding"



    # Save the predicted images with labels in the output folder
    output_path = os.path.join(output_folder, f'predicted_{i+1}_{predicted_label}.png')
    cv2.imwrite(output_path, cv2.cvtColor(test_image_rgb, cv2.COLOR_RGB2BGR))



print("Predicted images with labels saved to the output folder.")

```

```
1/1 [==============================] - 0s 67ms/step
1/1 [==============================] - 0s 19ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 30ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 19ms/step
Predicted images with labels saved to the output folder.
```