Preliminaries

```
!pip -q install wfdb imbalanced-learn

import os
import wfdb
from collections import Counter
import numpy as np
from sklearn.model_selection import train_test_split
from scipy.signal import find_peaks
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, Flatten, Dense, MaxPooling1D
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
from imblearn.over_sampling import SMOTE
```

```
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 160.0/160.0 kB 5.5 MB/s eta 0:00:00
```

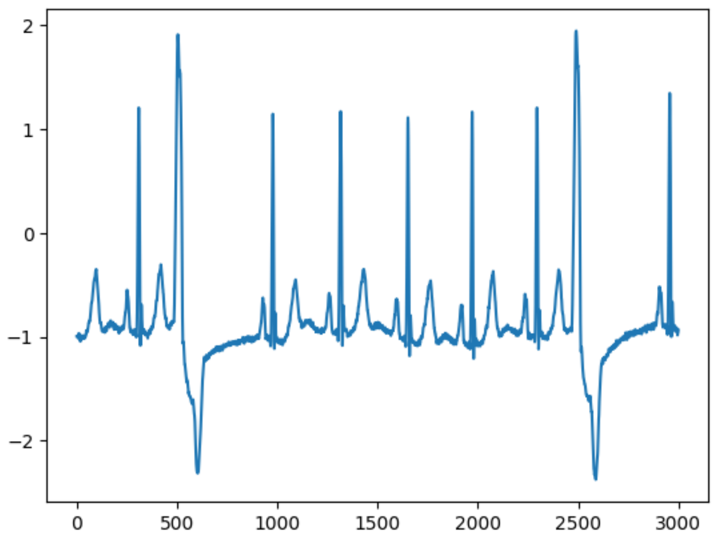
```
try:
    from google.colab import drive
    drive.mount('/content/drive', force_remount=True)

    project_directory = '/content/drive/MyDrive/TinyML-ECG/TinyRhythmAnalyzer/'  # Change this path to match your Google Driv
    os.chdir(project_directory)
except:
    pass
```

```
Mounted at /content/drive
```

Data Visualization

```
# Specify the record name and path
record_name = '119'
record_path = f'./mit-bih-arrhythmia-database-1.0.0/{record_name}'

# Read the ECG record. We can optionally pass in the `sampto` parameter to read a specific number of samples. Let's try 3000
record = wfdb.rdrecord(record_path, sampto=3000)

# Extract the ECG signal from the first lead of the record
ecg_signal = record.p_signal[:,0]

# Plotting the ECG signal
plt.plot(ecg_signal)
```

```
[<matplotlib.lines.Line2D at 0x7d76908ef730>]
```



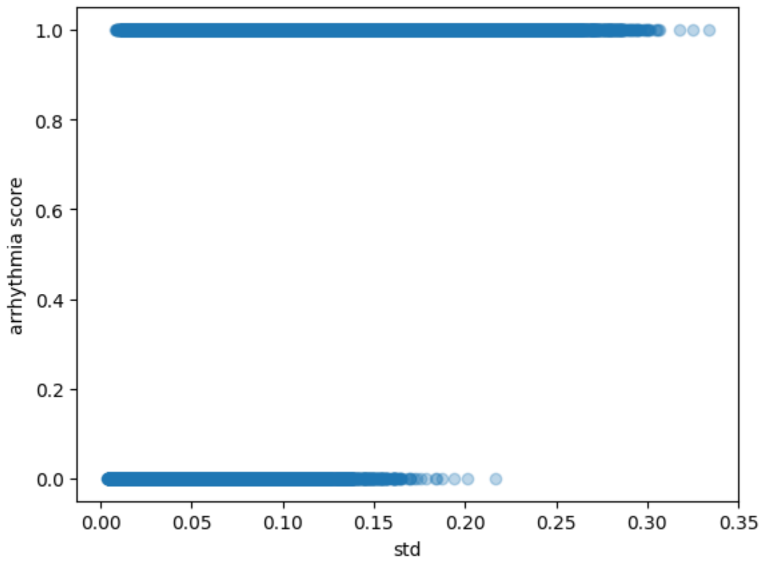
Data Preprocessing

```
record_name = '101'
record_path = f'./mit-bih-arrhythmia-database-1.0.0/{record_name}'
record = wfdb.rdrecord(record_path)
annotation = wfdb.rdann(record_path, 'atr')


peaks = annotation.sample
symbols = annotation.symbol


print(peaks[:10])
print(symbols[:10])
```

```
[   7   83  396  711 1032 1368 1712 2036 2349 2662]
['+', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N', 'N']
```

```
data_path = './mit-bih-arrhythmia-database-1.0.0/'
record_list = wfdb.get_record_list('mitdb')


X = []  # R-R intervals
y = []  # labels (normal = 0, arrhythmic = 1)

window_size = 10

for record_name in record_list:
    # Load the signal and annotations
    annotation = wfdb.rdann(os.path.join(data_path, record_name), 'atr')

    # Detect peaks
    peaks = annotation.sample

    # Get R-R intervals
    intervals = np.diff(peaks)

    # Normalize the intervals
    normalized_intervals = intervals / intervals.max()

    # Break up the intervals into segments of 10
    for i in range(len(intervals) - window_size + 1):
        # Get the annotations and intervals for the current window
        window_annotations = annotation.symbol[i:i + window_size]
        window_intervals = normalized_intervals[i:i + window_size]

        # Count of abnormal beats
        abnormal_count = sum(1 for ann in window_annotations if ann == 'V')

        # Select windows that are either all 'N' or have at least 2 'V's
        if all(ann == 'N' for ann in window_annotations) or (abnormal_count >= 2):
            X.append(window_intervals)
            y.append(abnormal_count >= 2)


X = np.array(X)
y = np.array(y)


plt.scatter(np.std(X, axis=1), y, alpha=0.3)
plt.xlabel("std")
plt.ylabel("arrhythmia score")
```

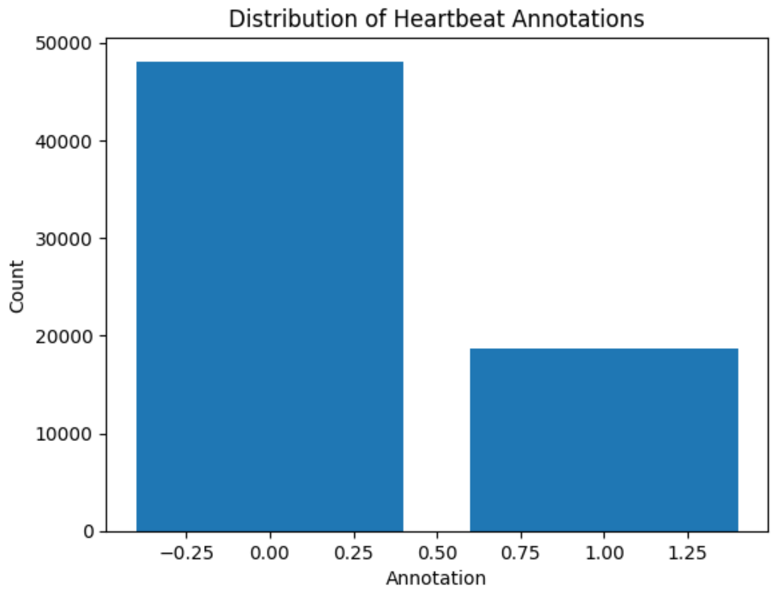
```
Text(0, 0.5, 'arrhythmia score')
```



2.4 Data Balancing

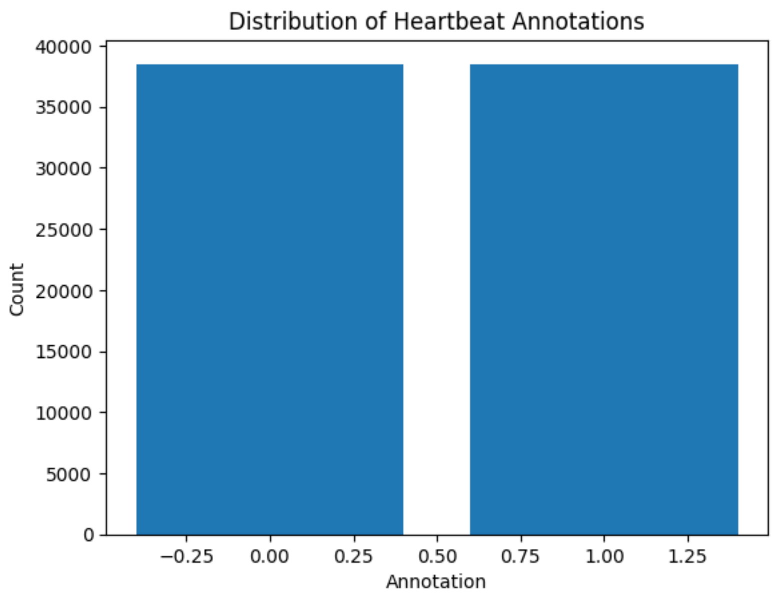
```
annotation_counter = Counter()
annotation_counter.update(y)

plt.bar(annotation_counter.keys(), annotation_counter.values())
plt.title('Distribution of Heartbeat Annotations')
plt.xlabel('Annotation')
plt.ylabel('Count')
```

```
Text(0, 0.5, 'Count')
```



```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# Fix class imbalance
smote = SMOTE()

# Fit SMOTE to the training data
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

annotation_smote_counter = Counter()
annotation_smote_counter.update(y_train_smote)

plt.bar(annotation_smote_counter.keys(), annotation_smote_counter.values())
plt.title('Distribution of Heartbeat Annotations')
plt.xlabel('Annotation')
plt.ylabel('Count')
plt.show()
```

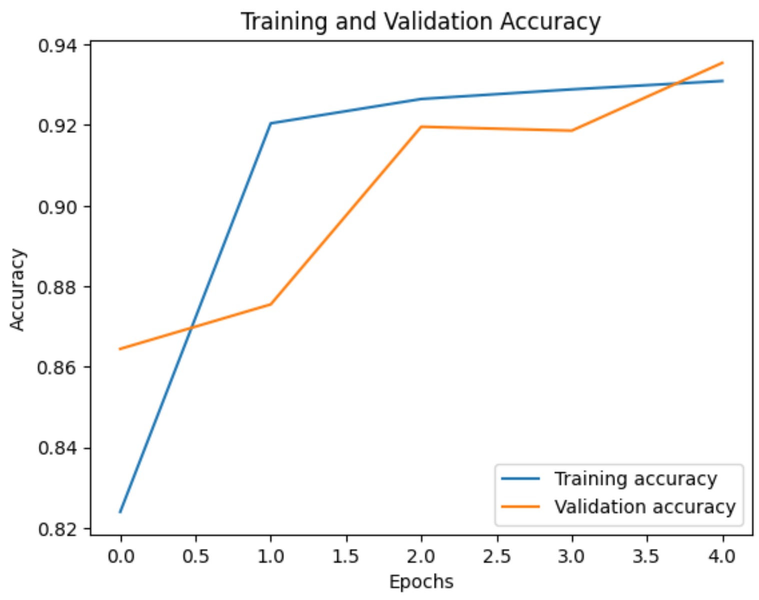


2.5 Model Training

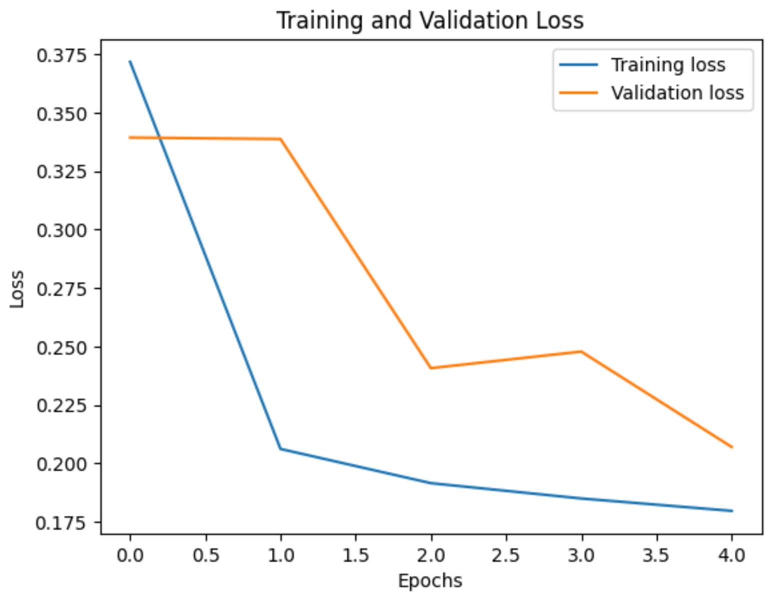
```
# Define the modle
model = Sequential([
    Dense(16, input_dim=window_size, activation='relu'),
    Dense(8, activation='relu'),
    Dense(1, activation='sigmoid')
])


# model = Sequential([
#     Conv1D(filters=16, kernel_size=3, activation='relu', input_shape=(window_size, 1)),
#     MaxPooling1D(pool_size=2),
#     Flatten(),
#     Dense(8, activation='relu'),
#     Dense(1, activation='sigmoid')
# ])


# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='binary_crossentropy', metrics=['accuracy'])


# Train the model
history = model.fit(
    X_train_smote,
    y_train_smote,
    epochs=5,
    batch_size=32,
    validation_split=0.2,
    verbose=1
)
```

```
Epoch 1/5
1925/1925 [==============================] - 5s 2ms/step - loss: 0.3718 - accuracy: 0.8240 - val_loss: 0.3394 - val_accu
Epoch 2/5
1925/1925 [==============================] - 3s 1ms/step - loss: 0.2062 - accuracy: 0.9204 - val_loss: 0.3388 - val_accu
Epoch 3/5
1925/1925 [==============================] - 3s 2ms/step - loss: 0.1916 - accuracy: 0.9265 - val_loss: 0.2408 - val_accu
Epoch 4/5
1925/1925 [==============================] - 3s 2ms/step - loss: 0.1850 - accuracy: 0.9289 - val_loss: 0.2479 - val_accu
Epoch 5/5
1925/1925 [==============================] - 3s 2ms/step - loss: 0.1797 - accuracy: 0.9309 - val_loss: 0.2071 - val_accu
```

```
plt.plot(history.history['accuracy'], label='Training accuracy')
plt.plot(history.history['val_accuracy'], label='Validation accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



```
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



```
# Evaluate the model and print the accuracy
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f'Test accuracy: {accuracy}')
print(f'Loss: {loss}')
```

```
    Test accuracy: 0.9248334169387817
    Loss: 0.18364550173282623
```

2.6 Exporting the Model

```
# Convert the model to the TensorFlow Lite format without quantization
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the model to a file
with open('model.tflite', 'wb') as f:
    f.write(tflite_model)
```

```
!apt-get -qq install xxd
!xxd -i model.tflite > model.cpp
!cat model.cpp
```

```
    unsigned char model_tflite[] = {
      0x1c, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x14, 0x00, 0x20, 0x00,
      0x1c, 0x00, 0x18, 0x00, 0x14, 0x00, 0x10, 0x00, 0x0c, 0x00, 0x00, 0x00,
      0x08, 0x00, 0x04, 0x00, 0x14, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00,
      0x84, 0x00, 0x00, 0x00, 0xdc, 0x00, 0x00, 0x00, 0xfc, 0x06, 0x00, 0x00,
      0x0c, 0x07, 0x00, 0x00, 0xac, 0x0c, 0x00, 0x00, 0x03, 0x00, 0x00, 0x00,
      0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0xb6, 0xf8, 0xff, 0xff,
      0x0c, 0x00, 0x00, 0x00, 0x1c, 0x00, 0x00, 0x00, 0x38, 0x00, 0x00, 0x00,
      0x0f, 0x00, 0x00, 0x00, 0x73, 0x65, 0x72, 0x76, 0x69, 0x6e, 0x67, 0x5f,
      0x64, 0x65, 0x66, 0x61, 0x75, 0x6c, 0x74, 0x00, 0x01, 0x00, 0x00, 0x00,
      0x04, 0x00, 0x00, 0x00, 0x98, 0xff, 0xff, 0xff, 0x0a, 0x00, 0x00, 0x00,
      0x04, 0x00, 0x00, 0x00, 0x07, 0x00, 0x00, 0x00, 0x64, 0x65, 0x6e, 0x73,
      0x65, 0x5f, 0x32, 0x00, 0x01, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
      0xa2, 0xf9, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00, 0x0b, 0x00, 0x00, 0x00,
      0x64, 0x65, 0x6e, 0x73, 0x65, 0x5f, 0x69, 0x6e, 0x70, 0x75, 0x74, 0x00,
      0x02, 0x00, 0x00, 0x00, 0x34, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
      0xdc, 0xff, 0xff, 0xff, 0x0d, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
      0x13, 0x00, 0x00, 0x00, 0x43, 0x4f, 0x4e, 0x56, 0x45, 0x52, 0x53, 0x49,
      0x4f, 0x4e, 0x5f, 0x4d, 0x45, 0x54, 0x41, 0x44, 0x41, 0x54, 0x41, 0x00,
      0x08, 0x00, 0x0c, 0x00, 0x08, 0x00, 0x04, 0x00, 0x08, 0x00, 0x00, 0x00,
      0x0c, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x13, 0x00, 0x00, 0x00,
      0x6d, 0x69, 0x6e, 0x5f, 0x72, 0x75, 0x6e, 0x74, 0x69, 0x6d, 0x65, 0x5f,
      0x76, 0x65, 0x72, 0x73, 0x69, 0x6f, 0x6e, 0x00, 0x0e, 0x00, 0x00, 0x00,
      0x1c, 0x06, 0x00, 0x00, 0x14, 0x06, 0x00, 0x00, 0xe4, 0x05, 0x00, 0x00,
      0xc8, 0x05, 0x00, 0x00, 0x78, 0x05, 0x00, 0x00, 0xe8, 0x02, 0x00, 0x00,
      0xd8, 0x00, 0x00, 0x00, 0xa8, 0x00, 0x00, 0x00, 0xa0, 0x00, 0x00, 0x00,
      0x98, 0x00, 0x00, 0x00, 0x90, 0x00, 0x00, 0x00, 0x88, 0x00, 0x00, 0x00,
      0x68, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x52, 0xfa, 0xff, 0xff,
      0x04, 0x00, 0x00, 0x00, 0x54, 0x00, 0x00, 0x00, 0x0c, 0x00, 0x00, 0x00,
      0x08, 0x00, 0x0e, 0x00, 0x08, 0x00, 0x04, 0x00, 0x08, 0x00, 0x00, 0x00,
      0x10, 0x00, 0x00, 0x00, 0x24, 0x00, 0x00, 0x00, 0x00, 0x00, 0x06, 0x00,
      0x08, 0x00, 0x04, 0x00, 0x06, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00,
      0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x00, 0x10, 0x00, 0x0c, 0x00,
      0x08, 0x00, 0x04, 0x00, 0x0a, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00, 0x00,
      0x02, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00, 0x06, 0x00, 0x00, 0x00,
      0x32, 0x2e, 0x31, 0x34, 0x2e, 0x30, 0x00, 0x00, 0xb2, 0xfa, 0xff, 0xff,
      0x04, 0x00, 0x00, 0x00, 0x10, 0x00, 0x00, 0x00, 0x31, 0x2e, 0x31, 0x34,
      0x2e, 0x30, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
      0x14, 0xf5, 0xff, 0xff, 0x18, 0xf5, 0xff, 0xff, 0x1c, 0xf5, 0xff, 0xff,
      0x20, 0xf5, 0xff, 0xff, 0xde, 0xfa, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00,
      0x20, 0x00, 0x00, 0x00, 0x77, 0xdc, 0xce, 0x3f, 0x19, 0x15, 0xda, 0xbf,
      0x14, 0xaf, 0xb9, 0x3f, 0x17, 0xa8, 0xc6, 0x3f, 0xa2, 0xb7, 0xd9, 0xbf,
      0xf7, 0x1b, 0x38, 0x40, 0x71, 0xcf, 0x4e, 0x3f, 0x00, 0x0a, 0x04, 0xbf,
      0x0a, 0xfb, 0xff, 0xff, 0x04, 0x00, 0x00, 0x00, 0x00, 0x02, 0x00, 0x00,
      0x0f, 0xbb, 0xec, 0x3f, 0x08, 0x6f, 0x9a, 0x3f, 0x57, 0x31, 0x91, 0xbc,
      0xe0, 0x5f, 0xdd, 0x3e, 0x8e, 0x6f, 0xf4, 0x3f, 0x24, 0xb3, 0xbd, 0x3e,
      0x1e, 0x4b, 0xdd, 0x3f, 0x28, 0xc2, 0x12, 0xbd, 0x82, 0x74, 0xb4, 0x3f,
      0xb4, 0xfe, 0xba, 0xbd, 0x2c, 0x0a, 0xec, 0xbd, 0x87, 0x34, 0x29, 0x3f,
      0xc4, 0x95, 0xad, 0x3e, 0xb3, 0xad, 0xb2, 0xbe, 0x44, 0x87, 0x97, 0x3d,
      0x7b, 0x5e, 0xe1, 0x3f, 0xca, 0x06, 0x08, 0xc0, 0x30, 0xff, 0xd1, 0xbf,
      0xa5, 0xb8, 0x34, 0x3f, 0x10, 0xf1, 0xa5, 0x3e, 0xa4, 0x77, 0xe0, 0xbf,
      0xe4, 0x38, 0xc1, 0x3e, 0x1b, 0x8f, 0xe0, 0xbf, 0x56, 0x6c, 0x36, 0x3f,
      0x55, 0xd7, 0xbc, 0xbf, 0x8f, 0x44, 0x1f, 0xbe, 0xad, 0x82, 0xc3, 0x3e,
      0xad, 0xea, 0x30, 0xbe, 0x00, 0xf2, 0xf4, 0xbe, 0x51, 0xc8, 0xfc, 0x3e,
      0xc5, 0xf3, 0x5c, 0x3e, 0xa9, 0x83, 0xfc, 0xbf, 0xc6, 0x56, 0x02, 0x40,
      0xfc, 0xa1, 0x9b, 0x3f, 0x21, 0x38, 0x2e, 0x3e, 0x80, 0x72, 0x46, 0xbe,
      0xaf, 0xcf, 0xd8, 0x3f, 0x18, 0x13, 0xf2, 0x3e, 0xa7, 0xd3, 0xd7, 0x3f,
      0x3e, 0x1f, 0xbc, 0xbd, 0xd5, 0x6b, 0xdf, 0x3f, 0x81, 0xde, 0x81, 0x3e,
```

Print Test Data

```
# Python code to create C-style header and source files
# from numpy arrays 'X_test' and 'y_test'.

def write_array_to_files(basename, array, num_trials):
    # Create the source file content
    cpp_content = f"""#include "{basename}_intervals.h"

const int num_{basename}_trials = {num_trials};
float {basename}_intervals[{num_trials}][10] = {{
"""
    for row in array:
```

```
!cat arrhythmic_intervals.cpp
```

```
#include "arrhythmic_intervals.h"

const int num_arrhythmic_trials = 20;
float arrhythmic_intervals[20][10] = {
    {0.521583, 0.248201, 0.796763, 0.550360, 0.526978, 0.163669, 0.165468, 0.739209, 0.464029, 0.343525},
    {0.416139, 0.356013, 0.363924, 0.397152, 0.352848, 0.362342, 0.381329, 0.275316, 0.545886, 0.409810},
    {0.466387, 0.796218, 0.363445, 0.718487, 0.418067, 0.380252, 0.380252, 0.590336, 0.573529, 0.605042},
    {0.548718, 0.343590, 1.000000, 0.741026, 0.725641, 0.469231, 0.948718, 0.741026, 0.733333, 0.464103},
    {0.544118, 0.481092, 0.397059, 0.575630, 0.222689, 0.224790, 0.911765, 0.418067, 0.714286, 0.443277},
    {0.186424, 0.263425, 0.231003, 0.220871, 0.203647, 0.328267, 0.202634, 0.263425, 0.226950, 0.197568},
    {0.370253, 0.384494, 0.137658, 0.137658, 0.536392, 0.420886, 0.162975, 0.101266, 0.443038, 0.473101},
    {0.376689, 0.687500, 0.368243, 0.638514, 0.476351, 0.439189, 0.479730, 0.589527, 0.548986, 0.555743},
    {0.776163, 0.613372, 0.593023, 0.601744, 0.590116, 0.595930, 0.389535, 0.799419, 0.552326, 0.645349},
    {0.630814, 0.604651, 0.610465, 0.587209, 0.610465, 0.552326, 0.662791, 0.622093, 0.415698, 0.799419},
    {0.763401, 0.360444, 0.334566, 0.425139, 0.386322, 0.552680, 0.569316, 0.641405, 0.384473, 0.900185},
    {0.197635, 0.398649, 0.228041, 0.535473, 0.496622, 0.271959, 0.422297, 0.476351, 0.312500, 0.391892},
    {0.614118, 0.183529, 0.315294, 0.701176, 0.531765, 0.682353, 0.562353, 0.618824, 0.555294, 0.625882},
    {0.336414, 0.341959, 0.759704, 0.292052, 0.728281, 0.325323, 0.726433, 0.373383, 0.693161, 0.347505},
    {0.366038, 0.903774, 0.367925, 0.894340, 0.364151, 0.907547, 0.367925, 0.441509, 0.443396, 0.616981},
    {0.979667, 0.787431, 0.349353, 0.316081, 0.926063, 0.800370, 0.377079, 0.338262, 0.955638, 0.768946},
    {0.888655, 0.485294, 0.722689, 0.460084, 0.569328, 0.378151, 0.672269, 0.453782, 0.615546, 0.451681},
    {0.217905, 0.417230, 0.429054, 0.423986, 0.251689, 0.724662, 0.253378, 0.464527, 0.425676, 0.375000},
    {0.470588, 0.525210, 0.203782, 0.203782, 0.550420, 0.401261, 0.598739, 0.401261, 0.598739, 0.432773},
    {0.229730, 0.212838, 0.456081, 0.260135, 0.327703, 0.369932, 0.366554, 0.418919, 0.280405, 0.266892},
};
```

```
!cat normal_intervals.cpp
```

```
#include "normal_intervals.h"

const int num_normal_trials = 20;
float normal_intervals[20][10] = {
    {0.555556, 0.462963, 0.410774, 0.427609, 0.335017, 0.575758, 0.331650, 0.523569, 0.449495, 0.427609},
    {0.732187, 0.756757, 0.746929, 0.727273, 0.710074, 0.707617, 0.690418, 0.714988, 0.724816, 0.754300},
    {0.700246, 0.660934, 0.668305, 0.744472, 0.751843, 0.685504, 0.695332, 0.700246, 0.710074, 0.658477},
    {0.783439, 0.783439, 0.690021, 0.685775, 0.743100, 0.751592, 0.734607, 0.668790, 0.747346, 0.787686},
    {0.757764, 0.739130, 0.720497, 0.773292, 0.763975, 0.748447, 0.729814, 0.748447, 0.736025, 0.757764},
    {0.749280, 0.746398, 0.740634, 0.743516, 0.772334, 0.740634, 0.752161, 0.731988, 0.729107, 0.740634},
    {0.616633, 0.626775, 0.618661, 0.626775, 0.622718, 0.624746, 0.649087, 0.555781, 0.679513, 0.655172},
    {0.717391, 0.704969, 0.711180, 0.692547, 0.720497, 0.723602, 0.711180, 0.692547, 0.704969, 0.708075},
    {0.725424, 0.738983, 0.713559, 0.677966, 0.683051, 0.718644, 0.733898, 0.700000, 0.671186, 0.681356},
    {0.348101, 0.400316, 0.373418, 0.356013, 0.368671, 0.460443, 0.420886, 0.526899, 0.340190, 0.352848},
```

Summary:

This Python code is designed to create a TinyML model for diagnosing cardiac arrhythmias using electrocardiogram (ECG) data. Start by installing the necessary packages and settings, especially Google Colab compatibility. The code then finds the recorded ECG signal and expands the preprocessed data to extract peaks and troughs from the ECG description. To remove class anomalies, the data set was balanced by selecting windows with all normal heartbeats or at least two abnormal heartbeats.

The basis of the rule involves defining and training a neural network for arrhythmia prediction based on the R-R interval. The performance of the model is evaluated in the experimental setup and training/validation is planned for analysis. The final step is to export the training model to TensorFlow Lite format for deployment in limited domains that exemplify the TinyML paradigm. This comprehensive approach combines visual data, prioritization, machine learning, and modeling to achieve high-quality arrhythmia detection suitable for use in advanced devices with fewer equipment counts.