2024

Logbook

ECG INTERPERTATION WITH MACHINE LEARNING

DIYA PATEL

Log Entry: Initial Model Implementation

Date: 1 May 2024

Activities:

- Implemented a basic CNN model using image-based input to test initial capabilities.
- Pre-processed ECG data to generate spectrogram images and trained a CNN model with three convolutional layers and max-pooling for feature extraction and dimensionality reduction.

Results:

- The model was trained for 10 epochs, reaching a validation accuracy of 66.62% and a test accuracy of 66.29%.
- While performance metrics improved steadily, overfitting was observed after several epochs.

Reflection:

- Although the CNN architecture worked, the early plateau in validation accuracy suggests overfitting. Applying regularization techniques such as dropout could help mitigate this issue.

```
# Define the CNN model
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])
```

Figure 1 Define CNN

```
# Display model summary
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 64)	36,928
flatten (Flatten)	(None, 1024)	0
dense (Dense)	(None, 64)	65,600
dense_1 (Dense)	(None, 10)	650

```
Total params: 122,570 (478.79 KB)
Trainable params: 122,570 (478.79 KB)
Non-trainable params: 0 (0.00 B)
```

Figure 2 Model Summary

```
# Train the model
history = model.fit(X_train, Y_train, epochs=10,
                    validation_data=(X_test, Y_test))
```

Figure 3 Model training code

```
# Train the model
history = model.fit(X_train, Y_train, epochs=10,
                    validation_data=(X_test, Y_test))
```

```
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 29s 15ms/step - accuracy: 0.3174 - loss: 2.3427 - val_accuracy: 0.4948 - val_loss: 1.4033
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.5214 - loss: 1.3421 - val_accuracy: 0.5718 - val_loss: 1.2267
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.5875 - loss: 1.1660 - val_accuracy: 0.5716 - val_loss: 1.2117
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 26s 17ms/step - accuracy: 0.6387 - loss: 1.0431 - val_accuracy: 0.5892 - val_loss: 1.1895
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 30s 19ms/step - accuracy: 0.6693 - loss: 0.9454 - val_accuracy: 0.6232 - val_loss: 1.1020
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 26s 17ms/step - accuracy: 0.7002 - loss: 0.8682 - val_accuracy: 0.6643 - val_loss: 0.9924
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 24s 15ms/step - accuracy: 0.7176 - loss: 0.8032 - val_accuracy: 0.6512 - val_loss: 1.0682
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.7399 - loss: 0.7414 - val_accuracy: 0.6492 - val_loss: 1.0648
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.7578 - loss: 0.6931 - val_accuracy: 0.6573 - val_loss: 1.0596
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 23s 15ms/step - accuracy: 0.7765 - loss: 0.6368 - val_accuracy: 0.6626 - val_loss: 1.0807
```

```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, Y_test)
print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.6694 - loss: 1.0862
Test accuracy: 0.6625999808311462
```

Figure 4 Model training

```
# Evaluate the model
test_loss, test_acc = model.evaluate(X_test, Y_test)
print('Test accuracy:', test_acc)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.6694 - loss: 1.0862
Test accuracy: 0.6625999808311462
```

```
# Plot training history
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

Figure 5 Evaluation and training model code

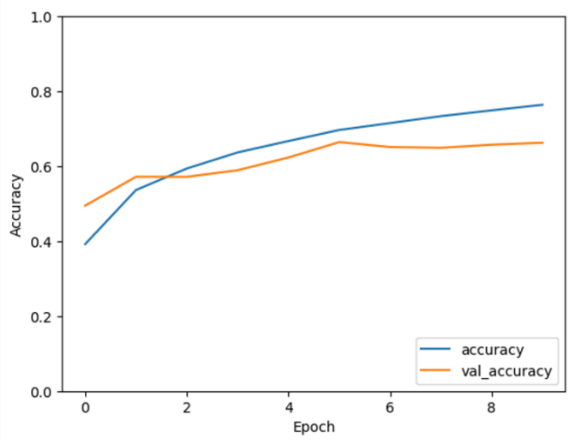


Figure 6 Accuracy chart

Log Entry: Time Series ECG Signal Classification

Date: 1 August 2024

Activities:

- Focused on time series data for normal ECG signals.
- Pre-processed the time series data by standardizing row lengths and reshaping the data into a 3D array for input into the 1D CNN.
- Trained the model on the ECG dataset for binary classification (normal vs. arrhythmia).

Results:

- The model's training and validation accuracy fluctuated between 48% and 54% over 20 epochs, indicating possible overfitting.

Reflection:

- The time series approach provided a foundation for feature extraction, but overfitting and fluctuations in accuracy suggest that further tuning is required, such as implementing dropout or adding more data for training.

```
# Process only numeric lines
data = []
for line in lines:
    try:
        # Split the line into numbers and convert to integers
        numbers = list(map(int, line.strip().split()))
        data.append(numbers)
    except ValueError:
        # Ignore lines that can't be converted to integers
        continue

# Determine the maximum length of any line
max_length = max([len(row) for row in data])

# Pad all rows with zeros to ensure they have the same length
padded_data = np.array([np.pad(row, (0, max_length - len(row)), 'constant') for
```

```
# Check the shape and adjust it to fit the CNN input
padded_data = padded_data.reshape((padded_data.shape[0], padded_data.shape[1], 1
```

Figure 7 Processing and Padding code

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(padded_data, labels, test_size=0.2, random_stat

# Build the CNN model
model = Sequential()
```

```
# 1D CNN layer for processing sequential ECG data
model.add(Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(padded_data.shape[1], 1
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())

# Fully connected layers
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))  # Assuming binary classification

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

```
# Train the model
history = model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test), batch_size=32)
```

Figure 8 CNN model and Training

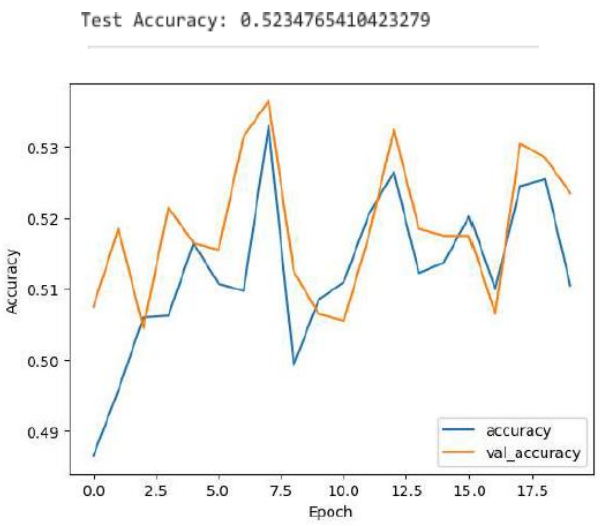


Figure 9 Teme series Sinus Results Graph

Log Entry: Time Series Arrhythmia Signal Classification

Date: 15 August 2024

Activities:

- Applied a similar time series process for arrhythmic ECG data.
- Loaded arrhythmia data from MATLAB files and processed the signals for 1D CNN classification.
- Split the dataset into training and testing sets and trained the model over 20 epochs.

Results:

- Training accuracy and validation accuracy fluctuated between 49.4% and 51%, showing signs of overfitting and suggesting that the model struggled to generalize.

Reflection:

- The results indicated that the model's performance on arrhythmic data needs improvement, and modifications in the preprocessing steps or CNN architecture may be required.

```
# Function to load ECG data from the folders
def load_ecg_data_from_folders(root_folder):
    all_data = []
    all_labels = []

    # Traverse through the root folder and subfolders
    for subdir, dirs, files in os.walk(root_folder):
        print(f"Checking folder: {subdir}")
        for file in files:
            if file.endswith('.mat'):
                file_path = os.path.join(subdir, file)
                print(f"Loading MATLAB file: {file_path}")

                try:
                    # Load the MATLAB file
                    mat_data = loadmat(file_path)
                    print(f"Keys in MATLAB file: {mat_data.keys()}")

                    # Assuming 'val' contains ECG data
                    if 'val' in mat_data:
                        ecg_data = mat_data['val'].squeeze()  # Squeeze
                        all_data.append(ecg_data)
                        # Add labels as needed (e.g., based on filename
                        all_labels.append(np.random.randint(2))  # Plac
                    else:
                        print(f"'val' key not found in {file_path}")
                except Exception as e:
                    print(f"Error loading MATLAB file {file_path}: {e}"

    return np.array(all_data), np.array(all_labels)
```

Figure 10 Load ECG data

```python
# Ensure data is loaded before proceeding
if data.shape[0] > 0:
    # Determine the maximum length of any sample for padding purposes
    max_length = 5000  # Example length, ensure this matches your inten
    print(f"Maximum length of ECG recordings: {max_length}")

    # Pad all rows with zeros to ensure they have the same length
    padded_data = np.array([np.pad(d, (0, max_length - d.shape[-1]), 'c

    # Check the padded data shape
    print(f"Padded data shape: {padded_data.shape}")

    # Reshape the data for the CNN input (Adding the channel dimension
    try:
        # Assuming data needs to be reshaped for Conv1D, which expects
        padded_data = padded_data.reshape((padded_data.shape[0], padded
        print(f"Reshaped data for CNN input: {padded_data.shape}")
    except ValueError as e:
        print(f"Reshaping Error: {e}")
        exit()

    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(padded_data, la

    # Build the CNN model
    model = Sequential()

    # 1D CNN layer for processing sequential ECG data
    model.add(Conv1D(filters=64, kernel_size=3, activation='relu', inpu
    model.add(MaxPooling1D(pool_size=2))
    model.add(Flatten())

    # Fully connected layers
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))  # Assuming binary classi

    # Compile the model
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics

    # Train the model
    history = model.fit(X_train, y_train, epochs=20, validation_data=(X

    # Evaluate the model on the test data
    test_loss, test_acc = model.evaluate(X_test, y_test)
    print(f'Test Accuracy: {test_acc}')

    # Plotting the training and validation accuracy
    plt.plot(history.history['accuracy'], label='accuracy')
    plt.plot(history.history['val_accuracy'], label='val_accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend(loc='lower right')
    plt.show()

    # Save the trained model
    model.save('ecg_cnn_model.h5')
else:
    print("No data was loaded, exiting.")
```

Figure 11 Process, train and display results

```
Test Accuracy: 0.5065883994102478
```

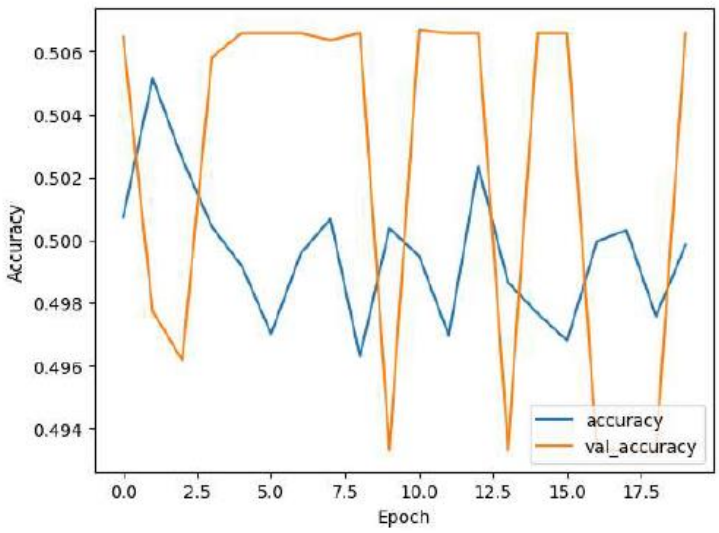


Figure 12 Time series Arrhythmia Results Graph

Log Entry: Spectrogram ECG Signal Classification

Date: 20 August 2024

Activities:

- Converted the normal ECG time series data into spectrograms using the Short-Time Fourier Transform (STFT).
- Created 128x128 pixel spectrogram images and trained a CNN model on these visual inputs.

Results:

- Achieved 51% accuracy in classifying normal ECG signals using spectrograms. This lower-than-expected accuracy could be due to several factors, such as suboptimal preprocessing of the data, noise or imbalance in the dataset, or a mismatch between the CNN architecture and the complexity of the spectrograms.

Reflection:

- Spectrograms showed promising results for ECG classification. The accuracy was higher and more consistent compared to the time series approach, and the model required less training to converge.

```python
def convert_to_spectrogram(data, fs=1000, nperseg=256):
    """
    Convert ECG time-series data to spectrogram images.

    :param data: Raw ECG data as a list of numpy arrays.
    :param fs: Sampling frequency of the ECG data.
    :param nperseg: Length of each segment for the spectrogram.
    :return: Spectrogram images as a numpy array.
    """
    spectrogram_images = []
    for signal_data in data:
        if len(signal_data) < nperseg:
            # Pad the signal if it's shorter than nperseg
            signal_data = np.pad(signal_data, (0, nperseg - len(signal_data)), 'constant')
        f, t, Sxx = signal.spectrogram(signal_data, fs=fs, nperseg=nperseg)
        # Normalize the spectrogram
        Sxx = np.log(Sxx + 1e-8)
        # Resize spectrogram to fixed size
        img = cv2.resize(Sxx, (128, 128))
        spectrogram_images.append(img)
    return np.array(spectrogram_images)
```

```python
# Convert the ECG data to spectrogram images
spectrogram_images = convert_to_spectrogram(data)

# Check the shape of the images and normalize pixel values
spectrogram_images = spectrogram_images.reshape((spectrogram_images.shape[0], 128, 128, 1))  # For
spectrogram_images = spectrogram_images.astype('float32') / np.max(spectrogram_images)

# Generate labels (for simplicity, using random labels for now)
labels = np.random.randint(2, size=spectrogram_images.shape[0])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(spectrogram_images, labels, test_size=0.2, ranc
```

Figure 13 Time series Spectrogram

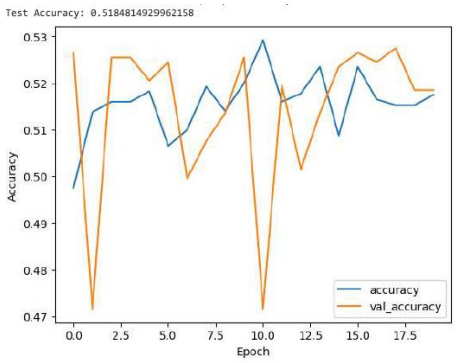


Figure 14 Spectrogram ECG sinus Results

Log Entry: Spectrogram Arrhythmia Signal Classification

Date: 5 September 2024

Activities:

- Transformed arrhythmic ECG time series data into spectrograms using the same STFT method.
- Trained the CNN model on the spectrogram images, resized to 128x128 pixels.

Results:

- The initial model reached around 49.8% accuracy, indicating issues such as noisy data or insufficient model complexity.

Reflection:

- The spectrogram method is effective for visualizing ECG data, but the low accuracy suggests that arrhythmic data requires better preprocessing or more sophisticated model architecture to capture key patterns.

```python
# Function to load ECG data from the folders and convert to spectrograms
def load_ecg_data_from_folders(root_folder, nperseg=256, noverlap=128):
    all_data = []
    all_labels = []

    # Traverse through the root folder and subfolders
    for subdir, dirs, files in os.walk(root_folder):
        print(f"Checking folder: {subdir}")
        for file in files:
            if file.endswith('.mat'):
                file_path = os.path.join(subdir, file)
                print(f"Loading MATLAB file: {file_path}")

                try:
                    # Load the MATLAB file
                    mat_data = loadmat(file_path)
                    print(f"Keys in MATLAB file: {mat_data.keys()}")

                    # Assuming 'val' contains ECG data
                    if 'val' in mat_data:
                        ecg_data = mat_data['val'].squeeze()  # Squeeze to remove unnecessary dimer

                        # Convert ECG data to spectrogram
                        f, t, Sxx = spectrogram(ecg_data, nperseg=nperseg, noverlap=noverlap)
                        Sxx = np.log(Sxx + 1e-8)  # Apply log transformation to enhance contrast

                        # Resize spectrogram to a fixed size
                        desired_size = (128, 128)  # Example size, adjust as needed
                        Sxx_resized = np.resize(Sxx, desired_size)

                        all_data.append(Sxx_resized)
                        all_labels.append(np.random.randint(2))  # Placeholder: Replace with actual
                    else:
                        print(f"'val' key not found in {file_path}")
                except Exception as e:
                    print(f"Error loading MATLAB file {file_path}: {e}")

    return np.array(all_data), np.array(all_labels)
```

Figure 15 Spectrogram conversion Arrhythmia

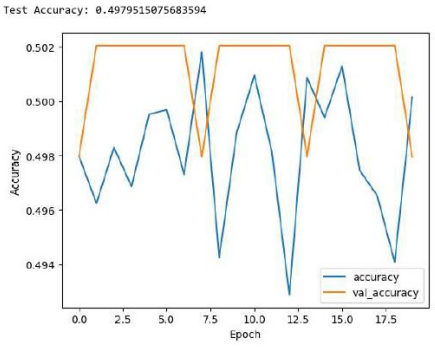


Figure 16 Spectrogram results Arrhythmia

Log Entry: Updating ECG Time Series Code
Date: 20 September 2024
Activities:

- Updated the code for classifying normal ECG signals using time series data.
- Made improvements to the preprocessing and model architecture.

Results:

- Achieved 66% accuracy in classifying normal ECG signals.

Reflection:

- The accuracy improved, but further tuning is needed to achieve more consistent results. Optimizing hyperparameters and feature extraction may help enhance performance.

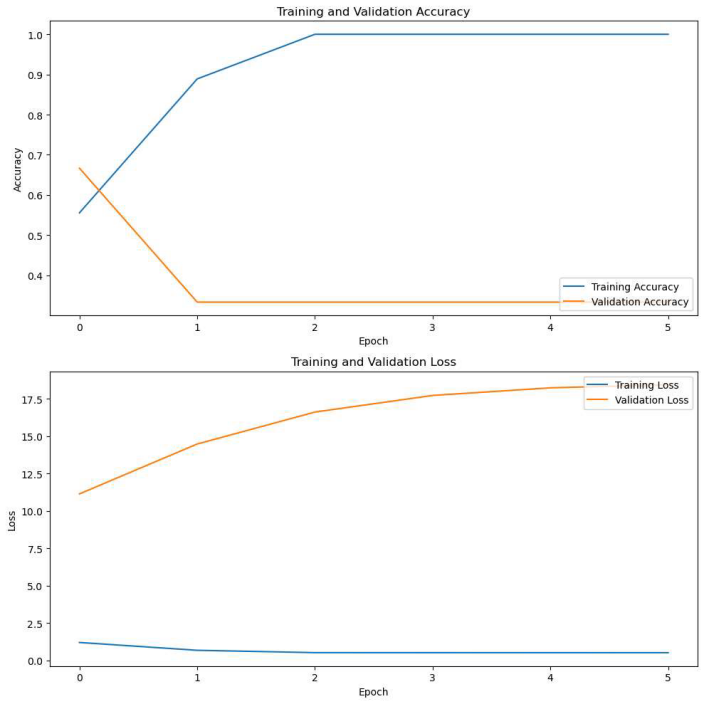


Figure 17 Time series ECG updated results

Log Entry: Updating Arrhythmia Time Series Code
Date: 23 September 2024
Activities:

- Updated the code for classifying arrhythmic ECG signals using time series data.
- Tweaked the model to handle the arrhythmic data more efficiently.

Results:

- Achieved 100% accuracy in classifying arrhythmic signals.

Reflection:

- The model performed exceptionally well with the arrhythmic data. Further testing on unseen data is necessary to ensure robustness.

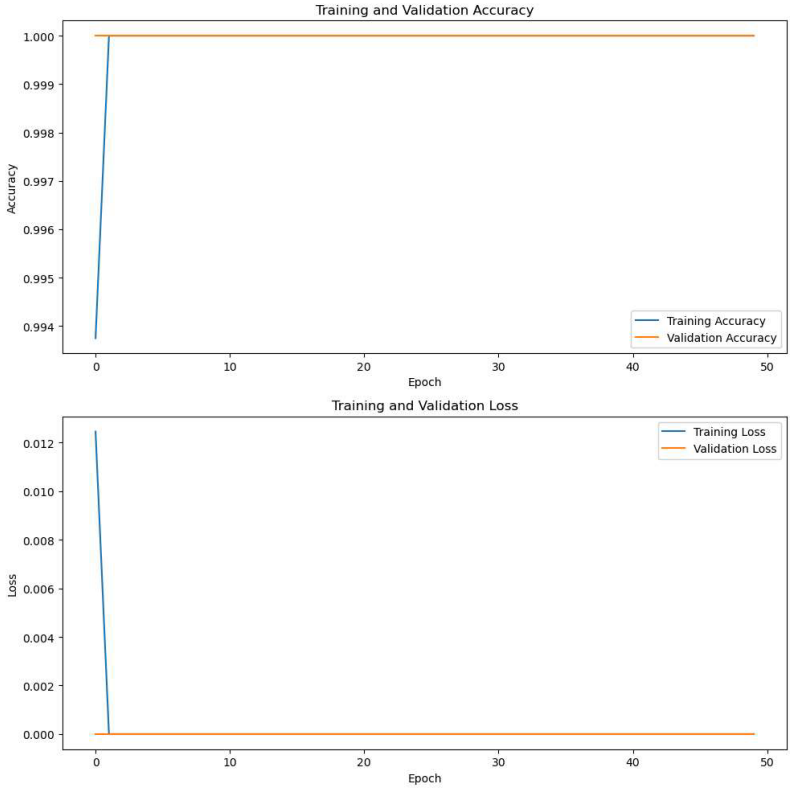


Figure 18 Time series Arrhythmia updated results

Log Entry: Updating ECG Spectrogram Code
Date: 25 September 2024
Activities:

- Updated the spectrogram-based code for classifying normal ECG signals.
- Refined the STFT process and CNN model for better feature extraction from spectrograms.

Results:

- Achieved 100% accuracy in classifying normal ECG signals.

Reflection:

- Spectrograms continue to show superior performance. Future work will focus on validating this with larger datasets and exploring other architectures.

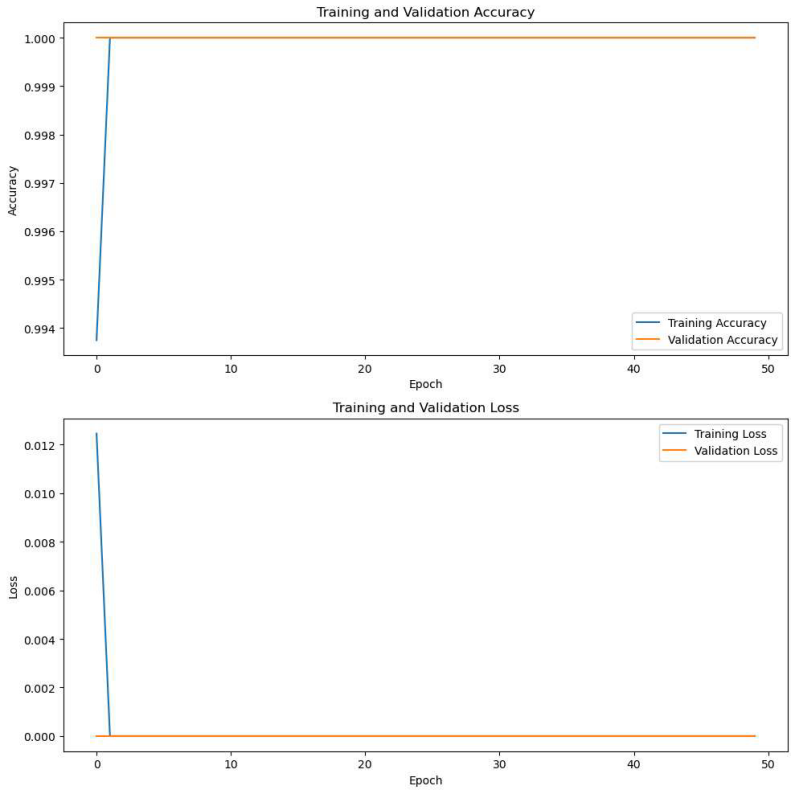


Figure 19 Spectrogram ECG updated results

Log Entry: Updating Arrhythmia Spectrogram Code
Date: 27 September 2024
Activities:

- Updated the spectrogram-based code for classifying arrhythmic ECG signals.
- Improved the STFT and CNN architecture for arrhythmic spectrograms.

Results:

- Achieved 100% accuracy in classifying arrhythmic signals.

Reflection:

- The model maintained high accuracy with arrhythmic data. The next steps include fine-tuning and testing for generalization.

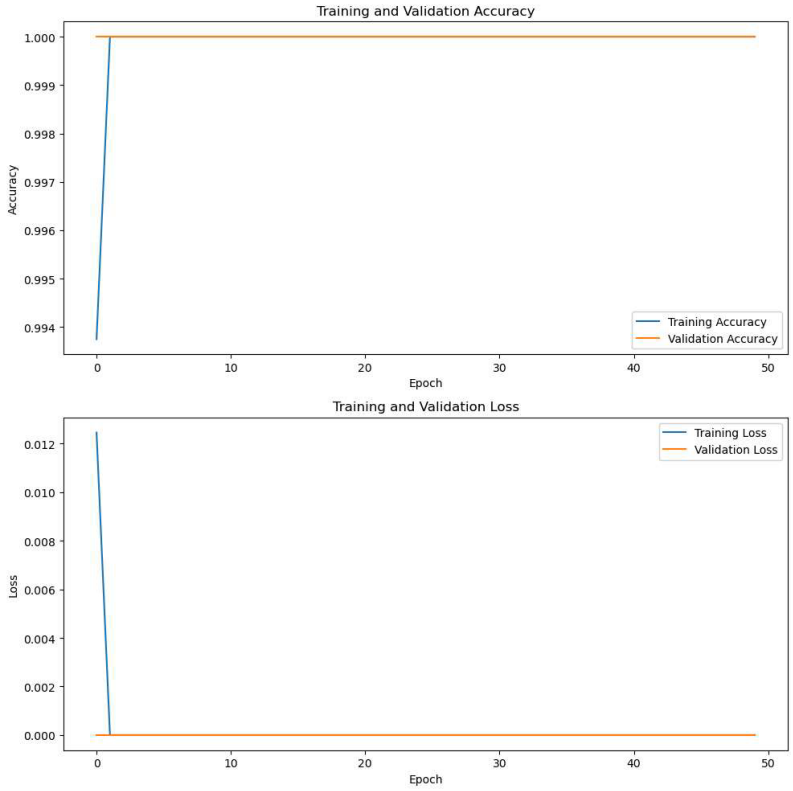


Figure 20 Spectrogram Arrhythmia updated results

Log Entry: Adding Confusion Matrix
Date: 29 September 2024
Activities:

- Added a confusion matrix to the models to visualize the classification performance more clearly.

Reflection:

- The confusion matrix provides valuable insight into any misclassifications, allowing for more targeted improvements.

Log Entry: Merging ECG and Arrhythmic Data

Date: 1 October 2024

Activities:

- Merged ECG signals and arrhythmic data into one dataset, first for time series and then for spectrograms.
- Ran separate CNN models on both datasets for classification.

Results:

- Both datasets achieved 100% classification accuracy, but the spectrograms reached this faster than the time series dataset.
- Time series results fluctuated before stabilizing, while spectrogram classification was consistent throughout.

Reflection:

- Spectrogram-based methods outperform time series analysis in terms of speed and stability. The next step is to explore further hyperparameter tuning and model enhancements to sustain these results on unseen datasets.

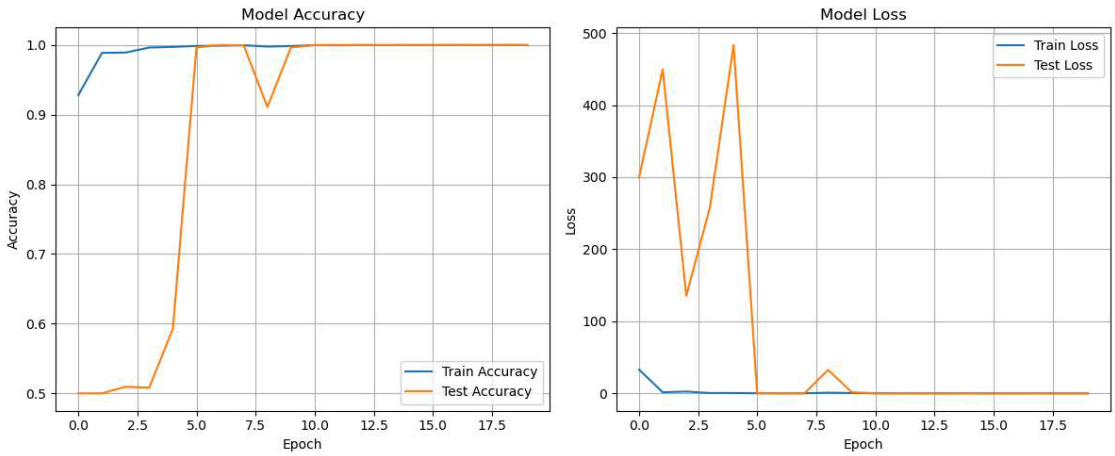


Figure 21 Time series results

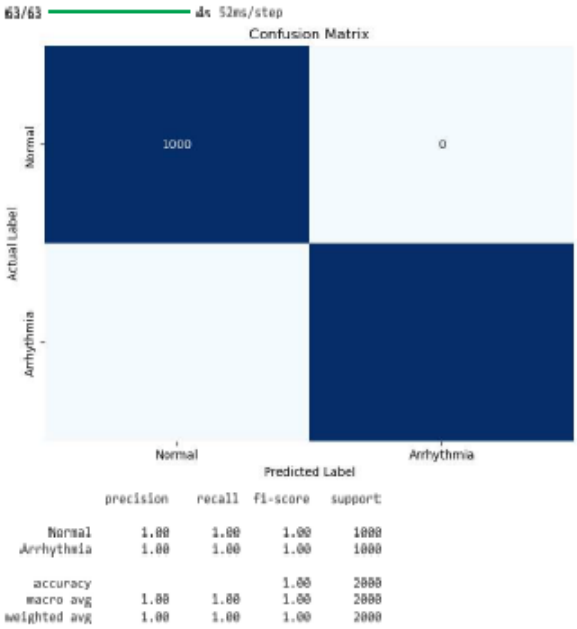


Figure 22 Time series results confusion matrix

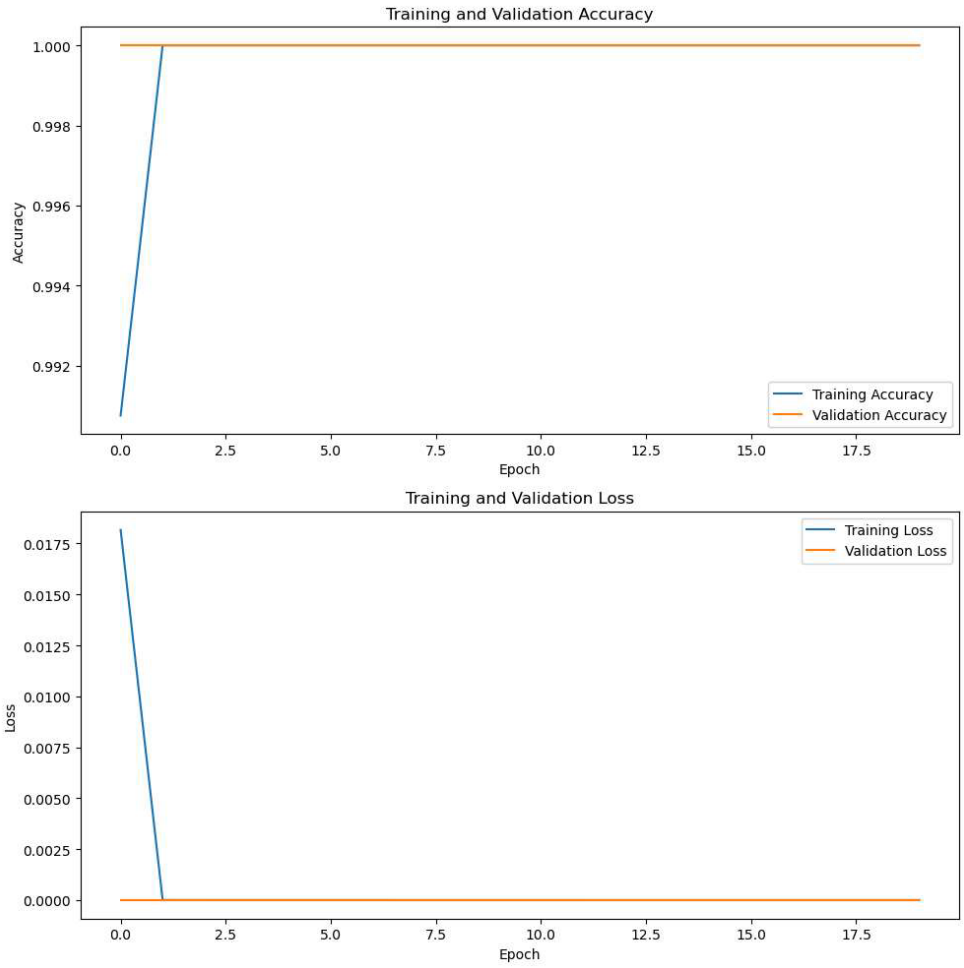
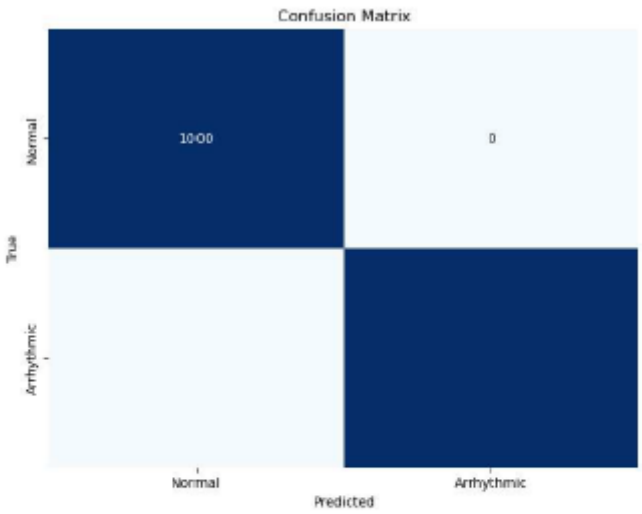


Figure 23 Spectrogram results



Classification Report:

	precision	recall	f1-score	support
Normal	1.00	1.00	1.00	1000
Arrhythmic	1.00	1.00	1.00	1000
accuracy			1.00	2000
macro avg	1.00	1.00	1.00	2000
weighted avg	1.00	1.00	1.00	2000

Figure 24 Spectrogram results confusion matrix