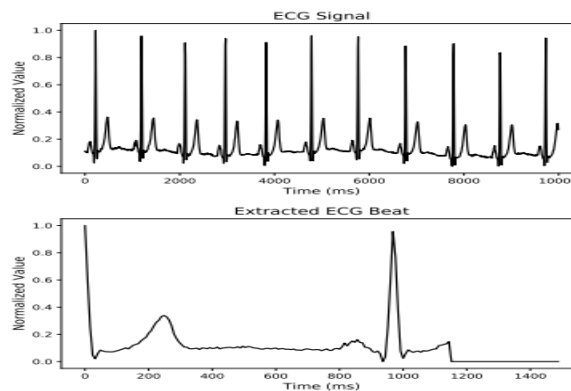




دانشگاه اصفهان
دانشکده مهندسی کامپیوتر

گزارش پروژه‌ی سوم یادگیری عمیق

Time Series



پدیدآورنده:

محمد امین کیانی

4003613052

دانشجوی کارشناسی، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،
Aminkianiworkeng@gmail.com

استاد درس: جناب آقای دکتر کیانی

نیمسال اول تحصیلی 1403-04

فهرست مطالب

3	مستندات
3	1-مسئله و تحلیل کلی آن:
4	2- مراحل اجرای یادگیر مسئله اول:
17	3- مراحل اجرای یادگیر مسئله دوم:
23	4-رسم نمودارها و ارتباطات:
27	5- مراجع

مستندات

1- مسئله و تحلیل کلی آن:

تحلیل ECG با مدل Transformer

تحلیل سیگنال‌های الکتروکاردیوگرام (ECG) با استفاده از مدل‌های Transformer یک رویکرد پیشرفته و نوین در پردازش داده‌های زمان‌سری پزشکی است. توضیح کلی درباره این فرآیند و مراحل آن به صورت زیر است:

1. مقدمه بر ECG

ECG یک روش غیرتهاجمی است که برای ثبت فعالیت الکتریکی قلب استفاده می‌شود. این سیگنال‌ها می‌توانند اطلاعات مهمی درباره وضعیت قلبی فرد فراهم کنند و به شناسایی اختلالات مختلف مانند آریتمی‌ها، ایسکمی و دیگر مشکلات قلبی کمک کنند. داده‌های ECG باید از منابع معتبر جمع‌آوری شوند که این داده‌ها معمولاً به صورت فایل‌های CSV یا فرمت‌های مشابه ذخیره می‌شوند.

2. چالش‌ها در تحلیل ECG

- داده‌های زمان‌سری: سیگنال‌های ECG به صورت زمان‌سری ثبت می‌شوند که تحلیل آن‌ها می‌تواند پیچیده باشد.
- نویز: سیگنال‌های ECG ممکن است تحت تأثیر نویزهای مختلف قرار گیرند.
- تنوع الگوها: الگوهای مختلف ECG ممکن است به دلیل شرایط بالینی متفاوت، متنوع باشند.

3. مدل‌های Transformer

مدل‌های Transformer به دلیل قابلیت‌های بالای خود در پردازش داده‌های توالی، به ویژه در زمینه‌های NLP (پردازش زبان طبیعی) شناخته شده‌اند. این

مدل‌ها می‌توانند روابط پیچیده بین داده‌ها را شناسایی کنند و به خوبی در تحلیل داده‌های زمان‌سری عمل کنند.

تحلیل سیگنال‌های ECG با استفاده از مدل‌های Transformer می‌تواند به تشخیص دقیق‌تر مشکلات قلبی کمک کند. این رویکرد نیازمند جمع‌آوری داده‌های باکیفیت، پیش‌پردازش مناسب، طراحی دقیق مدل و ارزیابی کامل است. می‌توان از معماری‌های مختلفی استفاده کرد. در اینجا، ما از چند معماری رایج برای این منظور استفاده خواهیم کرد تا نتیجه بگیریم کدام بهتر است (البته با اورلپ همگی مدل‌ها بالای 90 درصد می‌توانستند باشند):

* LSTM برای دسته‌بندی ← 76%

* RNN برای دسته‌بندی ← 65%

* transformer encoder برای دسته‌بندی ← 95%

ابتدا، باید داده‌ها را که شامل 2 تا فایل csv. دانلود کرده و پیش‌پردازش و فوریه ترنسفورم بزنیم، سپس مدل مورد نظر را آموزش دهیم و در نهایت، نتایج را ارزیابی کنیم.

2- مراحل اجرای یادگیر مسئله اول:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt

# # Load Data (replace with your actual file paths)
# normal_data = pd.read_csv('/content/drive/My Drive/data/ptbdb_normal.csv',
# header=None)
# abnormal_data = pd.read_csv('/content/drive/My Drive/data/ptbdb_abnormal.csv',
# header=None)

# Remove the last column
normal_data = normal_data.drop(columns=[normal_data.columns[-1]])
```

```

abnormal_data = abnormal_data.drop(columns=[abnormal_data.columns[-1]])

# Add Labels
normal_data['label'] = 0
abnormal_data['label'] = 1

# Combine data
data = pd.concat([normal_data, abnormal_data], axis=0).reset_index(drop=True)

# Remove duplicate samples
data = data.drop_duplicates()

# Shuffle data
data = data.sample(frac=1, random_state=42).reset_index(drop=True)

# Split features and labels
X = data.iloc[:, :-1].values
y = data['label'].values

# Define a function to check for overlap
def check_overlap(X_train, X_test):
    train_flat = set(map(tuple, X_train))
    test_flat = set(map(tuple, X_test))
    overlap = train_flat.intersection(test_flat)
    if overlap:
        print(f"Warning: Found {len(overlap)} overlapping samples between
training and test sets!")
    else:
        print("\u2714\u2713 No overlap between training and test sets.")

    train_classes, train_counts = np.unique(y_train, return_counts=True)
    print("\nClass distribution in Training set:")
    for cls, count in zip(train_classes, train_counts):
        print(f"Class {cls}: {count} samples")

    test_classes, test_counts = np.unique(y_test, return_counts=True)
    print("\nClass distribution in Test set:")
    for cls, count in zip(test_classes, test_counts):
        print(f"Class {cls}: {count} samples")

    train_distribution = train_counts / len(y_train)
    test_distribution = test_counts / len(y_test)
    if np.allclose(train_distribution, test_distribution, atol=0.05):
        print("\n\u2714\u2713 Class distribution between Train and Test sets is
similar.")

```

```

    else:
        print("\nWarning: Class distribution between Train and Test sets differs significantly!")

# Define a function to split data by time and avoid overlap
def split_data_temporally(X, y, train_ratio=0.8):
    train_size = int(len(X) * train_ratio)
    X_train, X_test = X[:train_size], X[train_size:]
    y_train, y_test = y[:train_size], y[train_size:]
    return X_train, X_test, y_train, y_test

# Split data temporally
X_train, X_test, y_train, y_test = split_data_temporally(X, y)

# Normalize data separately for training and testing
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Apply Fourier Transform (optional)
def apply_fourier_transform(data):
    return np.abs(np.fft.fft(data, axis=1))

X_train = apply_fourier_transform(X_train)
X_test = apply_fourier_transform(X_test)

# Check for overlap
check_overlap(X_train, X_test)

# Add noise to training data (optional)
def add_noise(data, noise_level=0.05):
    noise = noise_level * np.random.normal(size=data.shape)
    return data + noise

X_train = add_noise(X_train, noise_level=0.05)

# Reshape for Transformer input
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Define Transformer Encoder Block
def transformer_encoder(inputs, num_heads, ff_dim, dropout=0.1):
    attention = tf.keras.layers.MultiHeadAttention(num_heads=num_heads,
key_dim=inputs.shape[-1])(inputs, inputs)
    attention = tf.keras.layers.Dropout(dropout)(attention)

```

```

        attention = tf.keras.layers.LayerNormalization(epsilon=1e-6)(inputs +
attention)

        ff = tf.keras.layers.Dense(ff_dim, activation='relu')(attention)
        ff = tf.keras.layers.Dense(inputs.shape[-1])(ff)
        ff = tf.keras.layers.Dropout(dropout)(ff)
        return tf.keras.layers.LayerNormalization(epsilon=1e-6)(attention + ff)

# Build Transformer-based Model
def build_transformer_model(input_shape):
    input_layer = tf.keras.layers.Input(shape=input_shape)
    x = tf.keras.layers.Conv1D(filters=128, kernel_size=5,
activation='relu')(input_layer)
    x = tf.keras.layers.BatchNormalization()(x)
    x = tf.keras.layers.MaxPooling1D(pool_size=2)(x)
    x = tf.keras.layers.Dropout(0.5)(x)

    x = transformer_encoder(x, num_heads=4, ff_dim=256)
    x = transformer_encoder(x, num_heads=4, ff_dim=256)
    x = transformer_encoder(x, num_heads=4, ff_dim=256)

    x = tf.keras.layers.GlobalAveragePooling1D()(x)
    x = tf.keras.layers.Dense(128, activation='relu')(x)
    x = tf.keras.layers.Dropout(0.5)(x)
    output_layer = tf.keras.layers.Dense(1, activation='sigmoid')(x)

    model = tf.keras.Model(inputs=input_layer, outputs=output_layer)
    return model

# Build and compile model
model = build_transformer_model(input_shape=(X_train.shape[1], X_train.shape[2]))
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Train model
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=64,
    validation_data=(X_test, y_test),
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=8, restore_best_weights=True),

```

```

        tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=4, min_lr=1e-6)
    ],
    verbose=1
)

# Evaluate model
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Accuracy: {accuracy * 100:.2f}%")

# Classification report and confusion matrix
y_pred = (model.predict(X_test) > 0.5).astype(int)
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Normal",
"Abnormal"])
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix")
plt.show()

# Plot accuracy
plt.figure(figsize=(12, 6))
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Model Accuracy')
plt.show()

# Plot Loss
plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Model Loss')
plt.show()

```


1- کتابخانه‌ها

- `pandas` و `numpy` : برای مدیریت و پردازش داده‌ها.
- `Tensorflow` : ساخت و آموزش مدل یادگیری عمیق.
- `scikit-learn` : برای تقسیم‌بندی داده‌ها، پیش‌پردازش، و محاسبه‌ی متریک‌ها.
- `Matplotlib` : برای مصورسازی نتایج مدل.

2- پیش‌پردازش داده

- تعریف تابع بررسی داده‌ها: بررسی همپوشانی بین داده‌های آموزشی و آزمایشی و اطمینان از تقسیم درست داده‌ها زیرا همپوشانی می‌تواند باعث شود مدل به جای یادگیری، داده‌ها را حفظ کند و دقت کاذب ارائه دهد.

پارامترها:

`X_train`، `X_test`: ویژگی‌های داده.

`y_train`، `y_test`: برچسب‌های داده.

- بارگذاری داده‌ها: فایل‌های `CSV` حاوی سیگنال‌های قلبی.
- حذف ستون آخر: ستون آخر حاوی برچسب‌های اصلی است که در طی پردازش نباید مدل به آن دسترسی داشته باشد.
- اضافه کردن برچسب‌ها: برچسب 0 برای داده‌های سالم و 1 برای داده‌های بیمار.
- حذف نویز و تکراری‌ها: استفاده از تکنیک‌هایی مانند فیلترهای دیجیتالی برای حذف نویز از سیگنال و وجود داده تکراری می‌تواند باعث بایاس (سوگیری) مدل شود.
- تقسیم‌بندی: تقسیم داده‌ها به مجموعه‌های آموزشی، اعتبارسنجی و تست. داده‌ها به صورت ترتیبی (بر اساس زمان یا ترتیب وقوع) تقسیم می‌شوند.

◦ `train_ratio=0.8` : ۸۰٪ داده‌ها برای آموزش و ۲۰٪ برای تست استفاده می‌شود.

• نرمال‌سازی: مقیاس‌بندی داده‌ها با `MinMaxScaler()` به منظور کاهش تأثیر مقادیر بزرگ یا کوچک بر روی مدل.

`MinMaxScaler`: داده‌ها را به بازه‌ی `[0, 1]` نرمال می‌کند.

• چرایی انتخاب `MinMaxScaler`: مناسب برای داده‌های حساس به مقیاس تا ویژگی‌ها تأثیر یکسانی روی مدل داشته باشند.

• در نهایت قبل از شروع آموزش نویز به داده‌های آموزشی اضافه می‌شود تا افزایش تنوع داده‌ها و جلوگیری از بیش‌برازش (`overfitting`) داشته باشیم.

3- ویژگی‌سازی

• استخراج ویژگی‌ها از سیگنال‌های ECG، که می‌تواند شامل ویژگی‌های زمان‌محور و فرکانس‌محور باشد. در اینجا از تبدیل فوریه برای استخراج ویژگی‌های فرکانسی استفاده کردیم.

تبدیل فوریه (Fourier Transform) یکی از ابزارهای رایج برای تجزیه و تحلیل داده‌های سری زمانی و سیگنال‌ها است که ویژگی‌های فرکانسی سیگنال‌های ECG را استخراج می‌کند و تحلیل فرکانسی اطلاعات بیشتری درباره‌ی سیگنال فراهم می‌کند. دلایل استفاده از تبدیل فوریه برای سیگنال‌های ECG به شرح زیر است:

1. استخراج ویژگی‌های فرکانسی:

سیگنال‌های ECG ترکیبی از فرکانس‌های مختلف هستند که اطلاعاتی در مورد فعالیت الکتریکی قلب ارائه می‌دهند.

تبدیل فوریه سیگنال را از دامنه‌ی زمانی به دامنه‌ی فرکانسی منتقل می‌کند. این امر به ما کمک می‌کند تا الگوهای تکرارپذیری (مثل ریتم‌ها و فرکانس‌های غیرطبیعی) را شناسایی کنیم. همین امر سبب دقت بالاتر نسبت به LSTM بوده است.

2. تشخیص اختلالات قلبی:

برخی اختلالات قلبی مانند فیبریلاسیون دهلیزی یا تاکی کاردی به صورت تغییرات خاص در دامنه‌ی فرکانسی نمایان می‌شوند که ویژگی‌های فرکانسی می‌توانند به تشخیص این اختلالات کمک کنند.

3. کاهش نویز:

در دامنه‌ی فرکانسی، حذف نویزها (مانند تداخل ناشی از حرکات عضلانی یا سیستم‌های الکتریکی) آسان‌تر است.

4. بهینه‌سازی یادگیری مدل:

داده‌های فرکانسی معمولاً ویژگی‌های مفیدی را فراهم می‌کنند که یادگیری مدل‌های پیچیده مانند ترنسفورمر را تسهیل می‌کنند.

4- ساخت مدل Transformer

• چرا از مدل ترنسفورمر برای داده‌های سری زمانی استفاده می‌کنیم؟

ترنسفورمر در ابتدا برای داده‌های متنی (مثل ترجمه‌ی زبان) طراحی شد و از **توجه چندسری (Multi-Head Attention)** برای استخراج الگوها در داده استفاده می‌شود. همچنین لایه نرمال‌سازی کمک می‌کند تا مدل سریع‌تر همگرا شود. اما به دلیل ویژگی‌های ذاتی آن، برای داده‌های سری زمانی و سیگنال‌ها نیز مناسب است:

1. مدیریت وابستگی‌های بلندمدت:

در سیگنال‌های ECG، تغییرات کوچک در یک بازه می‌توانند تأثیرات بلندمدتی داشته باشند. مدل‌های سنتی مانند RNN و LSTM ممکن است در یادگیری این وابستگی‌ها ضعف داشته باشند.

ترنسفورمر با مکانیزم توجه (Attention Mechanism) قادر است ارتباطات بین نقاط زمانی مختلف را به خوبی مدل کند.

2. موازی سازی محاسبات:

برخلاف RNN ها که پردازش توالی را به ترتیب انجام می دهند، ترنسفورمر می تواند کل توالی را به صورت موازی پردازش کند. این ویژگی سرعت آموزش را افزایش می دهد.

3. استفاده از ویژگی های محلی و جهانی:

ترنسفورمر می تواند هم الگوهای محلی (مانند تغییرات سریع) و هم ویژگی های جهانی (مانند روندهای کلی) را در داده های سری زمانی شناسایی کند.

• چرا فقط از انکودر ترنسفورمر استفاده کردیم؟

1. ماهیت مسئله (کلاس بندی):

مسئله ای ما کلاس بندی (classification) است، جایی که باید ویژگی های سیگنال استخراج و تفسیر شوند. **انکودر ترنسفورمر وظیفه دارد اطلاعات ورودی را استخراج و ویژگی های مناسب را تولید کند.** استفاده از دیکودر ترنسفورمر برای تولید خروجی در اینجا غیرضروری است، زیرا **دیکودر عمدتاً برای تولید توالی های خروجی (مانند ترجمه متن) استفاده می شود.**

2. سادگی و کارایی:

انکودر به تنهایی برای استخراج اطلاعات ضروری از داده های سری زمانی کافی است. اضافه کردن دیکودر تنها پیچیدگی محاسباتی را افزایش می دهد بدون اینکه مزیتی برای این مسئله خاص فراهم کند.

3. عدم نیاز به تولید توالی:

دیکودر در ترنسفورمر برای تولید خروجی‌های متوالی طراحی شده است (مثل ترجمه یا تولید متن). اما در اینجا خروجی ما یک مقدار واحد (0 یا 1) است، بنابراین نیازی به دیکودر نیست.

• مزایای این انتخاب‌ها در مدل ما؟

1. تبدیل فوریه:

با فراهم کردن ویژگی‌های فرکانسی، مدل می‌تواند اختلالات و الگوهای سیگنال‌های ECG را با دقت بیشتری تشخیص دهد.

2. استفاده از انکودر ترنسفورمر:

تمرکز انکودر روی استخراج ویژگی‌های پیچیده باعث می‌شود مدل به خوبی رفتارهای سیگنال را یاد بگیرد.

3. کارایی بالاتر:

استفاده از انکودر به تنهایی باعث کاهش پیچیدگی محاسباتی و مصرف منابع شده و مدل را ساده‌تر و مؤثرتر می‌کند.

4. دقت بالاتر:

ترنسفورمر با بهرمگیری از مکانیزم توجه می‌تواند الگوهای پیچیده را بهتر از مدل‌های سنتی یاد بگیرد، که این امر منجر به بهبود دقت مدل در پیش‌بینی می‌شود.

• تعریف معماری مدل شامل لایه‌های توجه چندسر (Multi-head Attention)، لایه‌های تغذیه جلو (Feed-forward layers) و نرمال‌سازی لایه‌ای.

• تنظیم پارامترهای مدل مانند تعداد لایه‌ها، ابعاد ویژگی‌ها و نرخ یادگیری.

• شکل ورودی: عدد 1 برای سیگنال‌های تک‌کاناله است. (تعداد نمونه‌ها، ویژگی‌ها، 1)

• بلوک CNN :

Conv1D: استخراج ویژگی‌های محلی سیگنال.

filters=64: تعداد فیلترها.

kernel_size=3: اندازه‌ی هسته.

BatchNormalization: تسریع آموزش و تثبیت گرادیان‌ها.

MaxPooling1D: کاهش اندازه‌ی داده‌ها.

Dropout(0.4): جلوگیری از بیش‌برازش.

• بلوک Transformer :

num_heads = 4: تعداد سرهای توجه. مقدار 4 برای تعادل بین دقت و سرعت محاسباتی انتخاب شده است.

ff_dim = 128: ابعاد لایه‌های کاملاً متصل در بلوک.

ff_dim در کدهای مربوط به ترنسفورمر به ابعاد لایه پیش‌خور (-Feed Forward Layer) اشاره دارد.

در معماری ترنسفورمر، هر بلوک شامل دو بخش اصلی است:

1. لایه توجه چندسری: (Multi-Head Attention Layer)

- وظیفه این بخش، یادگیری ارتباطات بین توالی‌ها (مانند کلمات در جمله یا نقاط در سیگنال) است.

2. شبکه پیش‌خور نقطه‌ای: (Feed-Forward Network - FFN)

- بعد از لایه توجه، هر موقعیت در توالی، به طور جداگانه و مستقل از سایر موقعیت‌ها از یک شبکه پیش‌خور عبور می‌کند.

- این لایه شامل دو لایه Dense است:

1. لایه اول: ابعاد داده را از مقدار اولیه (ابعاد ورودی) به مقدار بزرگتر (مانند `ff_dim`) افزایش می‌دهد.

2. لایه دوم: ابعاد داده را دوباره به مقدار اصلی کاهش می‌دهد.

هدف لایه: FFN

- یادگیری ویژگی‌های پیچیده‌تر: با افزودن غیرخطیت به کمک توابع فعال‌سازی مثل `ReLU`، شبکه می‌تواند الگوهای غیرخطی و پیچیده را در داده‌ها یاد بگیرد.
- توسعه فضای ویژگی: با افزایش ابعاد به کمک `ff_dim`، شبکه قادر است اطلاعات بیشتری را در یک فضای با ابعاد بالاتر پردازش کند.
- لایه اول: `Dense(ff_dim)` ابعاد داده را از مقدار ورودی به `ff_dim` افزایش می‌دهد.
- لایه دوم: `Dense(inputs.shape[-1])` ابعاد را به مقدار اولیه برمی‌گرداند.

مقدار مناسب برای `ff_dim`

- مقدار `ff_dim` معمولاً بزرگتر از ابعاد ورودی است (معمولاً ۲ تا ۴ برابر).
- انتخاب مقدار مناسب به پیچیدگی داده‌ها و معماری مدل بستگی دارد:
 - اگر مقدار خیلی کوچک باشد: مدل ممکن است نتواند ویژگی‌های پیچیده را یاد بگیرد.
 - اگر مقدار خیلی بزرگ باشد: تعداد پارامترها افزایش می‌یابد و مدل ممکن است بیش‌برازش (`Overfitting`) شود.

• لایه‌های نهایی:

GlobalAveragePooling1D: کاهش ابعاد ویژگی‌ها.

Dense(64): لایه‌ی کاملاً متصل با 64 نورون.

Dense(1, activation='sigmoid'): خروجی احتمال کلاس.

5- آموزش مدل

• آموزش مدل با استفاده از داده‌های آموزشی و اعتبارسنجی عملکرد آن با استفاده از داده‌های اعتبارسنجی.

• استفاده از تکنیک‌هایی مانند توقف زودهنگام و کاهش نرخ یادگیری و... برای بهبود عملکرد که در پروژه های قبلی نیز استفاده شد که داریم:

Adam(learning_rate=0.0005): نرخ یادگیری کوچک برای بهبود همگرایی.

batch_size=64: انتخاب مناسب برای حافظه و عملکرد.

EarlyStopping: توقف زودهنگام برای جلوگیری از بیش‌برازش.

ReduceLROnPlateau: کاهش نرخ یادگیری برای بهبود دقت مدل.

6- ارزیابی مدل

• ارزیابی مدل با استفاده از مجموعه تست و محاسبه معیارهای مختلف مانند دقت، دقت مثبت (Precision)، یادآوری (Recall) و نمره F1.

• تجزیه و تحلیل ماتریس سردرگمی (Confusion Matrix) برای شناسایی نقاط قوت و ضعف مدل.

3- مراحل اجرای یادگیر مسئله دوم:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from sklearn.metrics import classification_report, confusion_matrix,
ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.utils import to_categorical

# Load data
train_data = pd.read_csv('/content/drive/My Drive/mitbih_train.csv', header=None)
test_data = pd.read_csv('/content/drive/My Drive/mitbih_test.csv', header=None)

# Separate features and labels
X_train = train_data.iloc[:, :-1].values
y_train = train_data.iloc[:, -1].values # Labels are in the last column
X_test = test_data.iloc[:, :-1].values
y_test = test_data.iloc[:, -1].values

# Check for invalid or missing labels
def clean_labels(features, labels):
    """Ensure all features and labels have matching lengths and valid data."""
    valid_indices = ~np.isnan(labels) # Remove NaN values
    features = features[valid_indices]
    labels = labels[valid_indices].astype(int) # Convert to integers
    return features, labels

X_train, y_train = clean_labels(X_train, y_train)
X_test, y_test = clean_labels(X_test, y_test)

# Normalize data
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# One-hot encode labels for multi-class classification
y_train = to_categorical(y_train, num_classes=5)
y_test = to_categorical(y_test, num_classes=5)

# Apply Fourier Transform (optional for feature extraction)
def apply_fourier_transform(data):
    return np.abs(np.fft.fft(data, axis=1))
```

```

X_train = apply_fourier_transform(X_train)
X_test = apply_fourier_transform(X_test)

# Reshape for Transformer input
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Define Transformer Encoder Block
def transformer_encoder(inputs, num_heads, ff_dim, dropout=0.1):
    attention = tf.keras.layers.MultiHeadAttention(num_heads=num_heads,
key_dim=inputs.shape[-1])(inputs, inputs)
    attention = tf.keras.layers.Dropout(dropout)(attention)
    attention = tf.keras.layers.LayerNormalization(epsilon=1e-6)(inputs +
attention)

    ff = tf.keras.layers.Dense(ff_dim, activation='relu')(attention)
    ff = tf.keras.layers.Dense(inputs.shape[-1])(ff)
    ff = tf.keras.layers.Dropout(dropout)(ff)
    return tf.keras.layers.LayerNormalization(epsilon=1e-6)(attention + ff)

# Build Transformer-based Model
input_layer = tf.keras.layers.Input(shape=(X_train.shape[1], X_train.shape[2]))

x = tf.keras.layers.Conv1D(filters=64, kernel_size=3,
activation='relu')(input_layer)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.MaxPooling1D(pool_size=2)(x)
x = tf.keras.layers.Dropout(0.3)(x)

x = transformer_encoder(x, num_heads=2, ff_dim=128)
x = transformer_encoder(x, num_heads=2, ff_dim=128)
x = transformer_encoder(x, num_heads=2, ff_dim=128)

x = tf.keras.layers.GlobalAveragePooling1D()(x)
x = tf.keras.layers.Dense(64, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
output_layer = tf.keras.layers.Dense(5, activation='softmax')(x)

model_transformer = tf.keras.Model(inputs=input_layer, outputs=output_layer)

# Compile model
model_transformer.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']

```

```

)

# Train model
history_transformer = model_transformer.fit(
    X_train, y_train,
    epochs=20,
    batch_size=128,
    validation_data=(X_test, y_test),
    callbacks=[
        tf.keras.callbacks.EarlyStopping(patience=5, restore_best_weights=True),
        tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5,
patience=3, min_lr=1e-6)
    ]
)

# Evaluate model
loss_transformer, accuracy_transformer = model_transformer.evaluate(X_test,
y_test)
print(f'Test Accuracy with Transformer: {accuracy_transformer * 100:.2f}%')

# Predictions and confusion matrix
y_pred_transformer = model_transformer.predict(X_test)
y_pred_classes = np.argmax(y_pred_transformer, axis=1)
y_true_classes = np.argmax(y_test, axis=1)

print("\nClassification Report (Transformer):")
print(classification_report(y_true_classes, y_pred_classes))

# Confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)
unique_classes = np.unique(np.concatenate((y_true_classes, y_pred_classes))) #
Dynamically determine unique classes
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=unique_classes)
disp.plot(cmap=plt.cm.Blues)
plt.title("Confusion Matrix (Transformer)")
plt.show()

# Plot accuracy
plt.figure(figsize=(12, 6))
plt.plot(history_transformer.history['accuracy'], label='Training Accuracy')
plt.plot(history_transformer.history['val_accuracy'], label='Validation
Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.title('Transformer Model Accuracy')

```

```
plt.show()

# Plot Loss
plt.figure(figsize=(12, 6))
plt.plot(history_transformer.history['loss'], label='Training Loss')
plt.plot(history_transformer.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.title('Transformer Model Loss')
plt.show()
```

بخش ۱: وارد کردن کتابخانه‌های لازم

- numpy و pandas : برای پردازش داده‌ها استفاده می‌شوند.
- Tensorflow : برای ساخت و آموزش مدل‌های یادگیری عمیق.
- sklearn.metrics : برای ارزیابی مدل از طریق گزارش دسته‌بندی و ماتریس سردرگمی.
- matplotlib.pyplot : برای مصورسازی نتایج و رسم نمودارها.
- MinMaxScaler : برای نرمال‌سازی داده‌ها به بازه ۰ تا ۱.
- to_categorical : تبدیل برچسب‌ها به قالب one-hot برای دسته‌بندی چندکلاسه.

بخش 2: جدا کردن ویژگی‌ها و برچسب‌ها

- `iloc[:, :-1]` : تمام ستون‌های به‌جز ستون آخر که مربوط به برچسب‌هاست به‌عنوان ویژگی انتخاب می‌شوند.
- `iloc[:, -1]` : ستون آخر (برچسب‌ها) جدا می‌شود.

بخش 3: پاک‌سازی برچسب‌ها

- این تابع برچسب‌های نامعتبر مانند NaN را حذف می‌کند.

- `astype(int)` : برچسب‌ها به قالب عدد صحیح تبدیل می‌شوند.
- وجود داده‌های نامعتبر یا NaN می‌تواند باعث خطا در پردازش یا آموزش مدل شود.

بخش 4: نرمال‌سازی داده‌ها

- داده‌ها به بازه $[0, 1]$ نرمال‌سازی می‌شوند.
- مدل‌های یادگیری عمیق معمولاً با داده‌های نرمال‌شده بهتر کار می‌کنند، زیرا باعث تسریع همگرایی در طی آموزش می‌شود.

بخش 5: تبدیل برچسب‌ها به قالب one-hot

- `to_categorical` : هر برچسب عددی به یک بردار دودویی تبدیل می‌شود.
- برای مسائل چندکلاسه، خروجی مدل باید در قالب احتمال دسته‌ها باشد.

بخش 6: اعمال تبدیل فوریه

- سپس تبدیل فوریه را بر روی داده‌ها اعمال می‌کند تا اطلاعات فرکانسی استخراج شود.
- سیگنال‌های زمانی اغلب شامل اطلاعات مفیدی در حوزه فرکانس هستند.

بخش 7: تغییر شکل برای ورودی مدل ترنسفورمر

- داده‌ها به شکل سه‌بعدی (نمونه‌ها، طول توالی، تعداد ویژگی‌ها) تبدیل می‌شوند زیرا مدل ترنسفورمر انتظار داده‌هایی در این قالب دارد.

بخش 8: تعریف بلاک ترنسفورمر

- یک بلوک ترنسفورمر شامل **توجه چندسری (Multi-Head Attention)** و یک شبکه کاملاً متصل است.
- **LayerNormalization**: به تثبیت یادگیری کمک می‌کند.
- مدل ترنسفورمر می‌تواند روابط بین داده‌ها در طول توالی‌ها را یاد بگیرد.

بخش 9: ساخت مدل ترنسفورمر

- مدل شامل لایه‌های **Conv1D** برای استخراج ویژگی‌های اولیه و چند بلوک ترنسفورمر است.
- خروجی یک لایه **Dense** با ۵ نود است (تعداد کلاس‌ها).
- ترکیب **CNN** و ترنسفورمر برای داده‌های زمانی مناسب است.

بخش 10: آموزش مدل

- **categorical_crossentropy**: تابع هزینه برای مسائل چندکلاسه.
- **EarlyStopping**: آموزش را هنگام بهبود ندادن متوقف می‌کند.
- **ReduceLROnPlateau**: نرخ یادگیری را در صورت عدم بهبود کاهش می‌دهد.

بخش 11: ارزیابی و مصورسازی نتایج

- تمامی نمودارها و ارقام نتایج را در بخش بعدی برای هر دو مسئله قرار داده‌ام.

4-رسم نمودارها و ارتباطات:

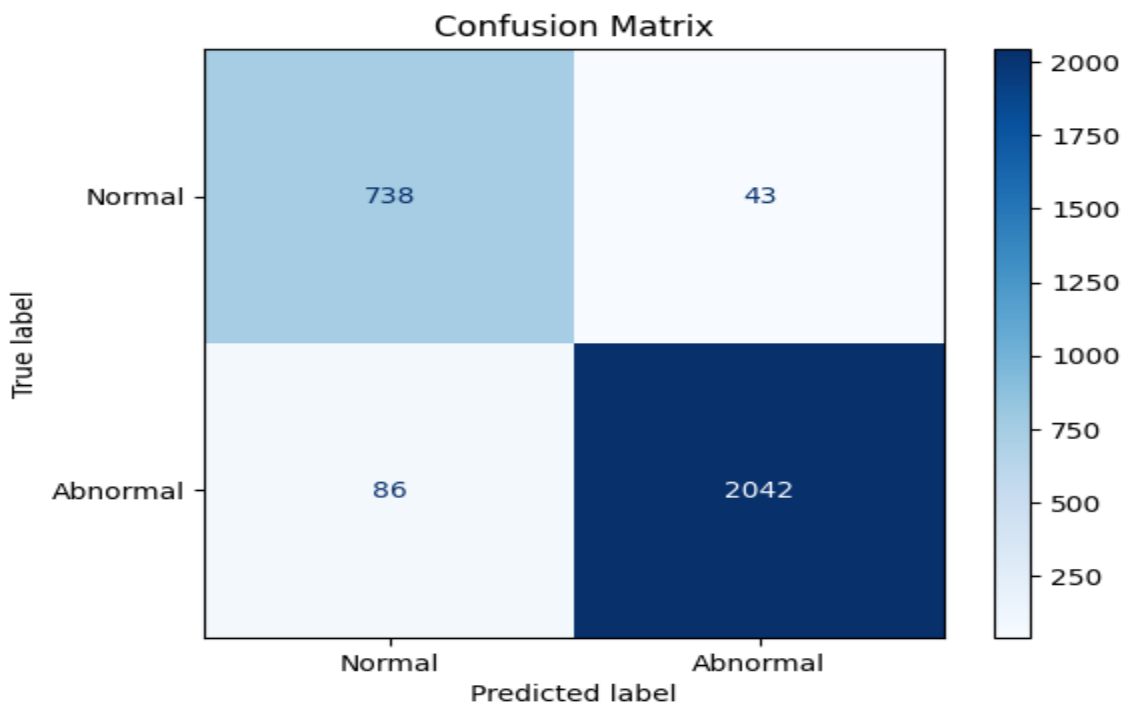
مطابق پروژه ی قبلی از پلات پایتون برای رسم نتایج استفاده کرده تا ببینیم داده ها در کجا قرار دارند و دید کلی از قرار گیری ان پیش بینی ها داشته باشیم:

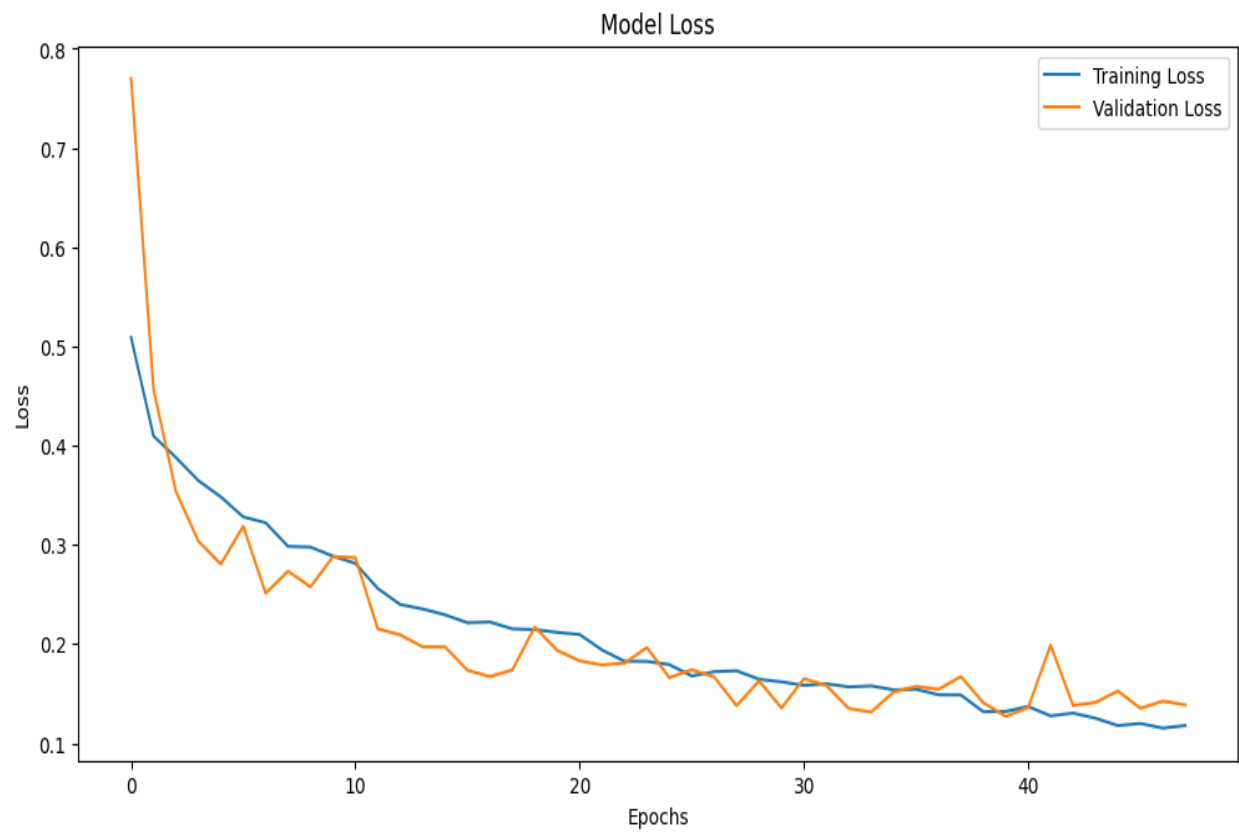
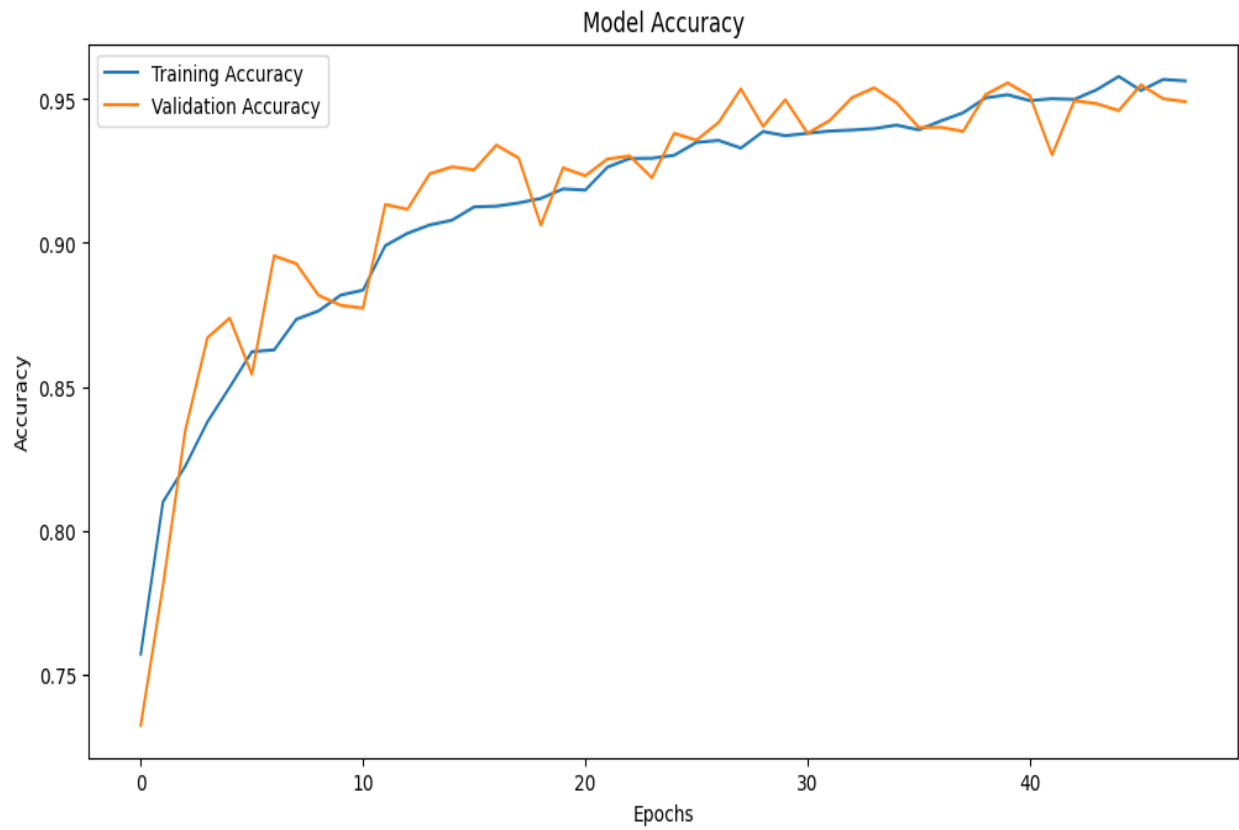
```
✓ No overlap between training and test sets.

Class distribution in Training set:
Class 0: 3264 samples
Class 1: 8372 samples

Class distribution in Test set:
Class 0: 781 samples
Class 1: 2128 samples

✓ Class distribution between Train and Test sets is similar.
Epoch 1/50
182/182 ————— 40s 115ms/step - accuracy: 0.7230 - loss: 0.5690 - val_accuracy: 0.7322 - val_loss: 0.7699 - 1
Epoch 2/50
182/182 ————— 13s 29ms/step - accuracy: 0.8036 - loss: 0.4225 - val_accuracy: 0.7807 - val_loss: 0.4565 - 1e
Epoch 3/50
182/182 ————— 10s 28ms/step - accuracy: 0.8229 - loss: 0.3827 - val_accuracy: 0.8347 - val_loss: 0.3535 - 1e
Epoch 4/50
182/182 ————— 5s 28ms/step - accuracy: 0.8319 - loss: 0.3825 - val_accuracy: 0.8670 - val_loss: 0.3037 - 1e
Epoch 5/50
182/182 ————— 5s 28ms/step - accuracy: 0.8497 - loss: 0.3499 - val_accuracy: 0.8738 - val_loss: 0.2805 - 1e
Epoch 6/50
Epoch 6/50
182/182 ————— 5s 29ms/step - accuracy: 0.8568 - loss: 0.3391 - val_accuracy: 0.8542 - val_loss: 0.3187 - 1e
Epoch 7/50
182/182 ————— 10s 28ms/step - accuracy: 0.8649 - loss: 0.3199 - val_accuracy: 0.8955 - val_loss: 0.2511 - 1e
...
accuracy          0.96    2909
macro avg         0.94    0.95    0.94    2909
weighted avg      0.96    0.96    0.96    2909
```



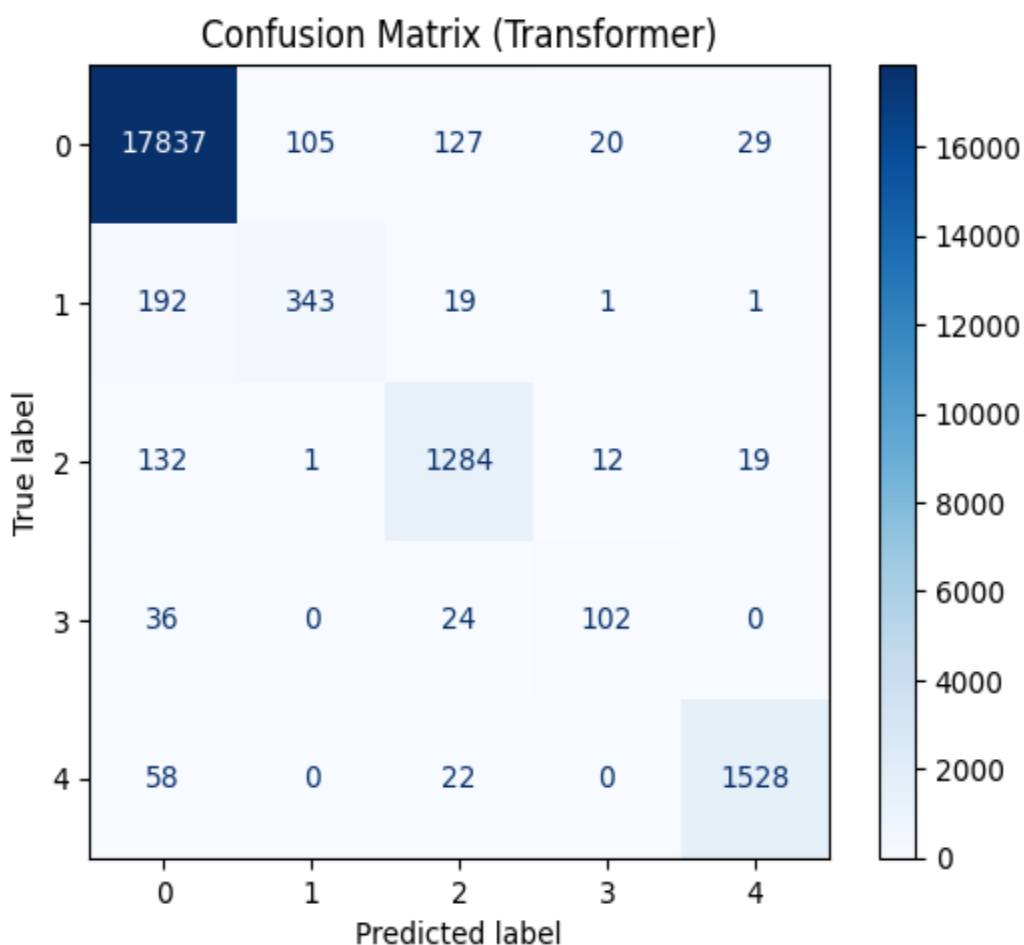


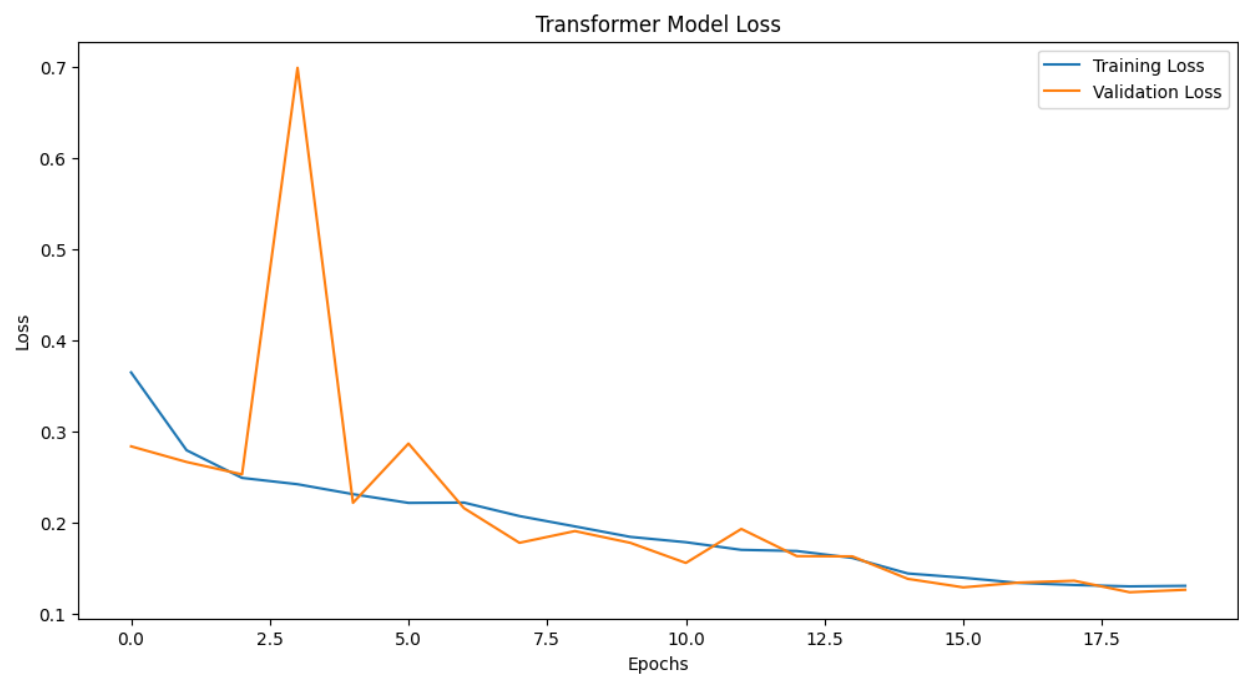
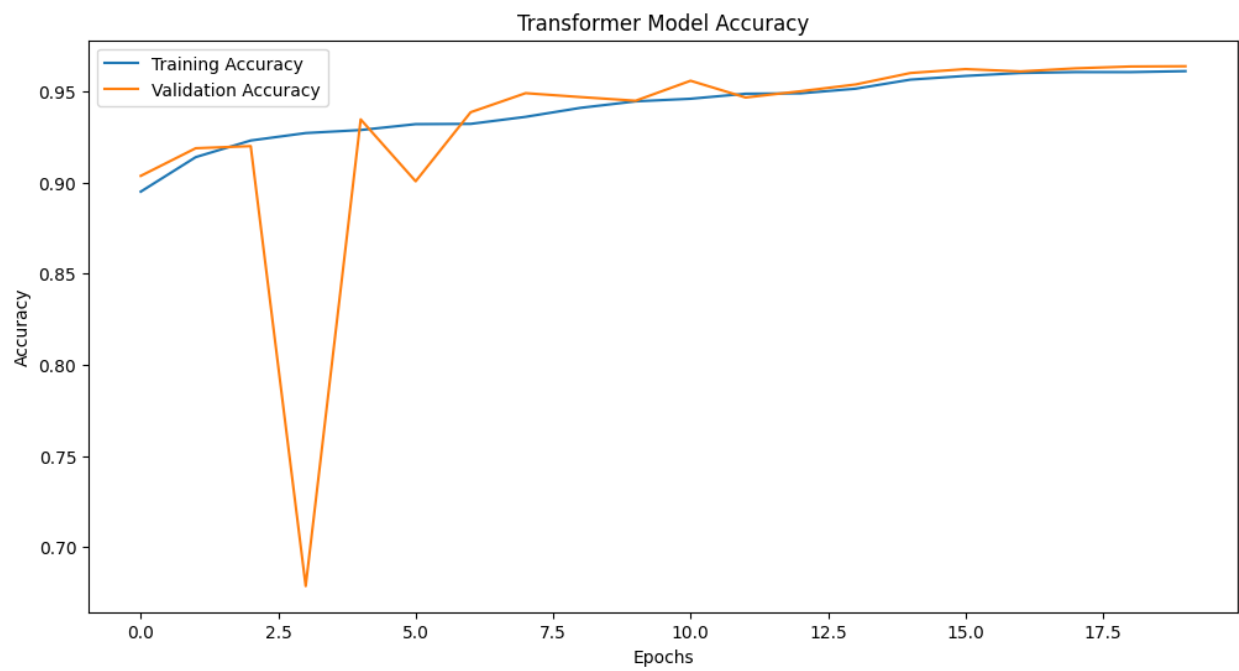

```

Epoch 7/20
685/685 — 13s 19ms/step - accuracy: 0.9205 - loss: 0.2536 - val_accuracy: 0.9385 - val_loss: 0.2164 - lea
Epoch 8/20
685/685 — 13s 19ms/step - accuracy: 0.9286 - loss: 0.2273 - val_accuracy: 0.9489 - val_loss: 0.1783 - lea
Epoch 9/20
685/685 — 20s 18ms/step - accuracy: 0.9373 - loss: 0.2083 - val_accuracy: 0.9468 - val_loss: 0.1912 - lea
Epoch 10/20
685/685 — 13s 18ms/step - accuracy: 0.9433 - loss: 0.1887 - val_accuracy: 0.9448 - val_loss: 0.1783 - lea
Epoch 11/20
685/685 — 21s 19ms/step - accuracy: 0.9460 - loss: 0.1767 - val_accuracy: 0.9557 - val_loss: 0.1563 - lea
Epoch 12/20
685/685 — 21s 20ms/step - accuracy: 0.9488 - loss: 0.1673 - val_accuracy: 0.9466 - val_loss: 0.1935 - lea
Epoch 13/20
...
accuracy          0.96    21892
macro avg         0.87    0.81    0.84    21892
weighted avg      0.96    0.96    0.96    21892

```

Output is truncated. View as a [scrollable element](#) or open in a [text editor](#). Adjust cell output [settings](#)...





- [1] <https://github.com>
- [2] <https://stackoverflow.com/questions>
- [3] <https://www.wikipedia.org/>
- [4] <https://colab.research.google.com/>
- [5] <https://www.tensorflow.org/guide/>
- [6] <https://keras.io/>
- [7] <https://github.com/M-Amin-Kiani/TimeSeries-ECG-Signal>
- [8] <https://sisoog.com/what-is-electrocardiogram/>
- [9] <https://medium.com/@lalesena/how-to-analyze-ecgs-with-python-396e34ece937>
- [10] <https://gist.github.com/emckiernan/3e7e86a48256777e9e6a44ede032d938>
- [11] [https://thesai.org/Downloads/Volume13No11/Paper_39-Transformer based Neural Network for Electrocardiogram Classification.pdf](https://thesai.org/Downloads/Volume13No11/Paper_39-Transformer_based_Neural_Network_for_Electrocardiogram_Classification.pdf)