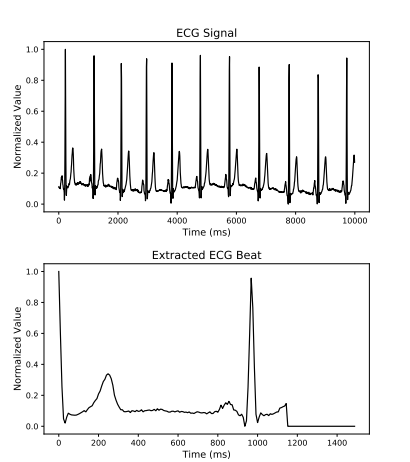


**دانشگاه اصفهان**

**دانشکده مهندسی کامپیوتر**

گزارش پروژه­ی سوم یادگیری عمیق

**Time Series - ECG**

****

پدیدآورنده:

**محمد ­امین کیانی**

**4003613052**

دانشجوی کارشناسی، دانشکده‌ی کامپیوتر، دانشگاه اصفهان، اصفهان،

Aminkianiworkeng@gmail.com

استاد درس: جناب اقای دکتر کیانی

نیمسال اول تحصیلی 04-1403

فهرست مطالب

[مستندات 3](#_Toc187346362)

[1-مسئله و تحلیل کلی آن: 3](#_Toc187346363)

[2- مراحل اجرای یادگیر مسئله اول: 5](#_Toc187346364)

[3- مراحل اجرای یادگیر مسئله دوم: 23](#_Toc187346365)

[4-رسم نمودارها و ارتباطات: 30](#_Toc187346366)

[5- مراجع 35](#_Toc187346367)

# 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.

• برای حل مشکل پدینگ صفر، باید یک مکانیزم ماسک‌گذاری (Masking) را برای صفرها در مدل یادگیری عمیق اضافه کنیم. این کار باعث می‌شود که مدل هنگام پردازش دنباله‌های سیگنال، به مقادیر صفر توجه نکند.

1. **ماسک‌گذاری صفرها** قبل از ورودی مدل با استفاده از tf.keras.layers.Masking.
2. **اصلاح Transformer Encoder** تا هنگام محاسبه توجه (Attention) به صفرها توجه نکند.
3. **اضافه کردن مکانیزم ماسک به لایه‌های MultiHeadAttention**.

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 معماری های 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 و مسک نکردن صفرها، همگی بالای 98 درصد می‌توانستند باشند.):

\* *LSTMبرای* دسته‌بندی 🡸 76%

\* *RNN*برای دسته‌بندی 🡸 65%

\* transformer\_encoderبرای دسته‌بندی 🡸 91%

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.preprocessing import MinMaxScaler

from sklearn.preprocessing import RobustScaler

from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay

import matplotlib.pyplot as plt

from sklearn.model\_selection import StratifiedKFold

# 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

# Remove duplicate samples & Combine data

data = pd.concat([normal\_data, abnormal\_data], axis=0).reset\_index(drop=True).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, y\_train, y\_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: {len(overlap)} overlapping samples between training and test sets!")

    else:

        print("\u2714\ufe0f No overlap between training and test sets.")

    print("\n Class distribution in Training set: ")

    for cls, count in zip(\*np.unique(y\_train, return\_counts=True)):

        print(f"Class {cls}: {count} samples")

    print("\n Class distribution in Test set: ")

    for cls, count in zip(\*np.unique(y\_test, return\_counts=True)):

        print(f"Class {cls}: {count} samples")

    train\_classes, train\_counts = np.unique(y\_train, return\_counts=True)

    test\_classes, test\_counts = np.unique(y\_test, return\_counts=True)

    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\ufe0f 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=0.7):

    train\_size = int(len(X) \* train\_ratio)

    return X[:train\_size], X[train\_size:], y[:train\_size], y[train\_size:]

X\_train, X\_test, y\_train, y\_test = split\_data\_temporally(X, y)

# Normalize data separately for training and testing

# scaler = MinMaxScaler()

# RobustScaler: این روش با استفاده از میانه و بازه بین چهارک‌ها داده‌ها را نرمال و نسبت به مقادیر پرت مقاوم‌تر است

scaler = RobustScaler()

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)

# Add noise to training data (optional)

def add\_noise(data, noise\_level=0.05):

    return data + noise\_level \* np.random.normal(size=data.shape)

X\_train = add\_noise(X\_train, noise\_level=0.05)

# Check for overlap

check\_overlap(X\_train, X\_test, y\_train, y\_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

class TransformerEncoderLayer(tf.keras.layers.Layer):

    def \_\_init\_\_(self, num\_heads, ff\_dim, dropout=0.1):

        super(TransformerEncoderLayer, self).\_\_init\_\_()

        self.mha = tf.keras.layers.MultiHeadAttention(num\_heads=num\_heads, key\_dim=ff\_dim)

        self.dropout1 = tf.keras.layers.Dropout(dropout)

        self.norm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.ffn = tf.keras.Sequential([

            tf.keras.layers.Dense(ff\_dim, activation='relu'),

            tf.keras.layers.Dense(ff\_dim // 2)

        ])

        self.dropout2 = tf.keras.layers.Dropout(dropout)

        self.norm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.num\_heads = num\_heads

    def call(self, inputs, training=False):

        seq\_len = tf.shape(inputs)[1]

        mask = tf.reduce\_sum(tf.abs(inputs), axis=-1, keepdims=True)

        mask = tf.cast(tf.not\_equal(mask, 0), dtype=tf.float32)

        mask = tf.tile(mask, [1, 1, self.num\_heads])

        mask = tf.reshape(mask, (-1, self.num\_heads, seq\_len, 1))

        mask = tf.tile(mask, [1, 1, 1, seq\_len])

        attn\_output = self.mha(inputs, inputs, attention\_mask=mask)

        attn\_output = self.dropout1(attn\_output, training=training)

        out1 = self.norm1(inputs + attn\_output)

        ffn\_output = self.ffn(out1)

        ffn\_output = self.dropout2(ffn\_output, training=training)

        return self.norm2(out1 + ffn\_output)

# 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 = TransformerEncoderLayer(num\_heads=4, ff\_dim=256)(x)

    x = TransformerEncoderLayer(num\_heads=4, ff\_dim=256)(x)

    x = TransformerEncoderLayer(num\_heads=4, ff\_dim=256)(x)

    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

# 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

numpy و pandas : 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 دیجیتال 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=0.7 : 7۰٪ داده‌ها برای آموزش و 3۰٪ برای تست استفاده می‌شود.

• بررسی همپوشانی:

**چرایی تبدیل به مجموعه (Set):** داده‌ها برای بررسی هم‌پوشانی باید به فرم مجموعه تبدیل شوند، زیرا مجموعه‌ها تنها مقادیر یکتا را نگه می‌دارند و بررسی هم‌پوشانی در آن‌ها سریع‌تر است.

داده‌های X\_train و X\_test معمولاً به صورت آرایه یا ماتریس هستند. برای تبدیل آن‌ها به مجموعه، ابتدا هر سطر داده به یک tuple تبدیل می‌شود زیرا مقادیر tuple قابل اضافه شدن به مجموعه هستند.

**intersection :** این عملیات بررسی می‌کند که چه مقادیری در هر دو مجموعه وجود دارند که بفهمیم آیا نمونه‌ای از داده‌های آموزشی به اشتباه در داده‌های آزمایشی قرار گرفته است.

**np.allclose :** بررسی می‌کند که توزیع نسبی کلاس‌ها در داده‌های آموزشی و آزمایشی مشابه هستند یا خیر.

**atol=0.05 :** اگر تفاوت بین توزیع‌ها کمتر از ۵٪ باشد، به‌عنوان مشابه در نظر گرفته می‌شود.

**هدف کلی تابع**

1. **هم‌پوشانی نمونه‌ها:** اطمینان حاصل می‌کند که داده‌های آموزشی و آزمایشی کاملاً جدا از یکدیگر هستند.
2. **بررسی توزیع کلاس‌ها:** بررسی می‌کند که آیا داده‌های آموزشی و آزمایشی به صورت متوازن تقسیم شده‌اند یا خیر.

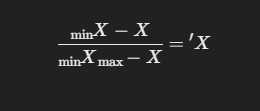
**چرا این موضوع مهم است؟**

* **بدون هم‌پوشانی:** اگر داده‌های مشترکی بین مجموعه آموزشی و آزمایشی باشد، مدل ممکن است به‌جای یادگیری واقعی، داده‌ها را حفظ کند و عملکرد واقعی آن سنجیده نمی‌شود.
* **توزیع مشابه کلاس‌ها:** اگر توزیع کلاس‌ها در دو مجموعه متفاوت باشد، مدل ممکن است در یادگیری یا تعمیم‌دهی دچار مشکل شود.

• نرمال‌سازی: مقیاس‌بندی داده‌ها با RobustScalerبه منظور کاهش تأثیر مقادیر بزرگ یا کوچک بر روی مدل زیرا نرمال‌سازی داده‌ها فرآیندی است که در آن مقادیر ویژگی‌های داده به یک مقیاس مشترک منتقل می‌شوند. این کار برای بهبود عملکرد مدل‌های یادگیری ماشین (مخصوصاً الگوریتم‌هایی که از گرادیان یا فاصله‌ها استفاده می‌کنند) بسیار ضروری است. سه روش متداول برای نرمال‌سازی داده‌ها به صورت زیر است:

**الف) MinMaxScaler**

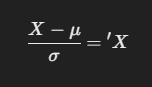
* مقادیر داده‌ها را به یک بازه خاص معمولا [1,0] تغییر می‌دهد.



* **مزایا:**
  + ساده و سریع و برای داده‌هایی با توزیع یکنواخت خوب عمل می‌کند.
* **معایب:**
  + **حساس به مقادیر پرت (Outliers):** وجود مقادیر پرت باعث می‌شود کل مقیاس داده‌ها تغییر کند.

**ب) StandardScaler**

* داده‌ها را طوری مقیاس‌بندی می‌کند که میانگین0 و انحراف معیار 1 داشته باشند.

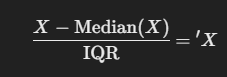


* **مزایا:**
  + مناسب برای داده‌هایی که به‌صورت نرمال توزیع شده‌اند.
* **معایب:**
  + مانند MinMaxScaler به مقادیر پرت حساس است؛ مقادیر پرت می‌توانند μ\mu و σ\sigma را تغییر دهند.

**ج)RobustScaler**

RobustScaler روشی مقاوم‌تر (Robust) در برابر **مقادیر پرت** است.

* به‌جای استفاده از **میانگین** و **انحراف معیار** یا **حداقل و حداکثر مقدار** ، از **میانه** و **بازه بین چارک‌ها** (Interquartile Range - IQR) استفاده می‌کند:
  + **میانه (Median):** مقدار مرکزی داده که نسبت به مقادیر پرت مقاوم است.
  + **بازه بین چارک‌ها (IQR):** فاصله بین چارک اول و چارک سوم

* هر ویژگی به‌صورت مستقل بر اساس میانه و IQR آن مقیاس‌بندی می‌شود.
* اگر داده‌ها دارای **مقادیر پرت** باشند (یعنی نقاطی که بسیار از سایر داده‌ها دور هستند)، روش‌هایی مانند MinMaxScaler یا StandardScaler نمی‌توانند به‌خوبی عمل کنند:
  + **در MinMaxScaler :** مقادیر پرت باعث می‌شوند بازه داده‌ها (حداقل و حداکثر) بیش از حد گسترده شود و داده‌های دیگر فشرده شوند.
  + **در StandardScale :** مقادیر پرت میانگین (μ\mu) و انحراف معیار (σ\sigma) را تغییر می‌دهند و باعث اعوجاج در مقیاس‌بندی می‌شوند.

**راه‌حل: RobustScaler**

* **مقاوم بودن نسبت به مقادیر پرت:** به دلیل استفاده از میانه و IQR، این روش نسبت به مقادیر پرت حساسیت کمتری دارد. مقادیر پرت در محاسبه میانه و چارک‌ها تأثیر نمی‌گذارند.
* **ثبات:** برای داده‌هایی که دارای توزیع غیریکنواخت هستند یا مقادیر پرت زیادی دارند، نتایج بهتری ارائه می‌دهد.
* **بهبود عملکرد مدل:** استفاده از داده‌های نرمال‌شده و پایدار می‌تواند باعث یادگیری بهتر مدل شود، زیرا مدل تمرکز بیشتری روی الگوهای واقعی داده خواهد داشت.

**مزایا:**

* مقاوم در برابر مقادیر پرت.
* مناسب برای داده‌های با توزیع غیریکنواخت.

**معایب:**

* ممکن است اطلاعات جزئی در داده‌ها را از بین ببرد.
* برای داده‌هایی که مقادیر پرت ندارند، عملکرد آن مشابه روش‌های دیگر است.
* در نهایت قبل از شروع اموزش 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 تاکی‌کاردی 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. موازی‌سازی محاسبات:

برخلاف RNN‌ها که پردازش توالی را به ترتیب انجام می‌دهند، ترنسفورمر می‌تواند کل توالی را به صورت موازی پردازش کند. این ویژگی سرعت آموزش را افزایش می‌دهد.

3. استفاده از ویژگی‌های محلی و جهانی:

ترنسفورمر می‌تواند هم الگوهای محلی (مانند تغییرات سریع) و هم ویژگی‌های جهانی (مانند روندهای کلی) را در داده‌های سری زمانی شناسایی کند.

• چرا فقط از انکودر ترنسفورمر استفاده کردیم؟

1. ماهیت مسئله (کلاس‌بندی):

مسئله‌ی ما کلاس‌بندی (classification) است، جایی که باید ویژگی‌های سیگنال استخراج و تفسیر شوند. انکودر ترنسفورمر وظیفه دارد اطلاعات ورودی را استخراج و ویژگی‌های مناسب را تولید کند. استفاده از دیکودر ترنسفورمر برای تولید خروجی در اینجا غیرضروری است، زیرا دیکودرعمدتاً برای تولید توالی‌های خروجی (مانند ترجمه متن) استفاده می‌شود.

2. سادگی و کارایی:

انکودر به تنهایی برای استخراج اطلاعات ضروری از داده‌های سری زمانی کافی است. اضافه کردن دیکودر تنها پیچیدگی محاسباتی را افزایش می‌دهد 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 بهره‌گیری 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 :**  تعداد "سرهای" توجه Attention Heads که این پارامتر مشخص می‌کند که عملیات توجه چندگانه (Multi-Head Attention) با چند سر انجام شود.

**ff\_dim :**  تعداد نرون‌های لایه‌های متراکم Feed-Forward Neural Network

**Dropout :**  درصد افت (Dropout) برای جلوگیری از بیش‌برازش (Overfitting).

**Normalization :**  تنظیم مقادیر لایه برای پایداری یادگیری و سرعت‌بخشیدن به همگرایی.

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 (مانندff\_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.

• در ادامه متد کال برای ساخت ماسک استفاده می‌شود تا مقدارهای صفر در توالی (که مربوط به پدینگ هستند) در محاسبات توجه نادیده گرفته شوند.

• 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.preprocessing import MinMaxScaler, RobustScaler

from sklearn.metrics import classification\_report, confusion\_matrix, ConfusionMatrixDisplay

from sklearn.utils.class\_weight import compute\_class\_weight

from tensorflow.keras.utils import to\_categorical

import matplotlib.pyplot as plt

# 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

X\_test = test\_data.iloc[:, :-1].values

y\_test = test\_data.iloc[:, -1].values

# Clean labels

def clean\_labels(features, labels):

    valid\_indices = ~np.isnan(labels)

    features = features[valid\_indices]

    labels = labels[valid\_indices].astype(int)

    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 = RobustScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# One-hot encode labels

y\_train = to\_categorical(y\_train, num\_classes=5)

y\_test = to\_categorical(y\_test, num\_classes=5)

# Apply Fourier Transform

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 data 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)

# Check overlap

def check\_overlap(X\_train, X\_test):

    train\_flat = set(map(tuple, X\_train.reshape(X\_train.shape[0], -1)))

    test\_flat = set(map(tuple, X\_test.reshape(X\_test.shape[0], -1)))

    overlap = train\_flat.intersection(test\_flat)

    if overlap:

        print(f"Warning: {len(overlap)} overlapping samples detected between training and testing sets.")

    else:

        print("No overlap detected between training and testing sets.")

check\_overlap(X\_train, X\_test)

# Define Transformer Encoder Layer

class TransformerEncoderLayer(tf.keras.layers.Layer):

    def \_\_init\_\_(self, num\_heads, ff\_dim, dropout\_rate=0.1):

        super(TransformerEncoderLayer, self).\_\_init\_\_()

        self.attention = tf.keras.layers.MultiHeadAttention(num\_heads=num\_heads, key\_dim=1)

        self.ffn = tf.keras.Sequential([

            tf.keras.layers.Dense(ff\_dim, activation='relu'),

            tf.keras.layers.Dense(1)

        ])

        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = tf.keras.layers.Dropout(dropout\_rate)

        self.dropout2 = tf.keras.layers.Dropout(dropout\_rate)

    def call(self, inputs):

        mask = tf.reduce\_sum(tf.abs(inputs), axis=-1, keepdims=True)

        mask = tf.cast(tf.not\_equal(mask, 0), dtype=tf.float32)

        attn\_output = self.attention(inputs, inputs, attention\_mask=mask)

        attn\_output = self.dropout1(attn\_output)

        out1 = self.layernorm1(inputs + attn\_output)

        ffn\_output = self.ffn(out1)

        ffn\_output = self.dropout2(ffn\_output)

        return self.layernorm2(out1 + ffn\_output)

# 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 = TransformerEncoderLayer(num\_heads=2, ff\_dim=128)(x)

x = TransformerEncoderLayer(num\_heads=2, ff\_dim=128)(x)

x = TransformerEncoderLayer(num\_heads=2, ff\_dim=128)(x)

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)

# Compute class weights

class\_weights = compute\_class\_weight(

    class\_weight='balanced',

    classes=np.unique(np.argmax(y\_train, axis=1)),

    y=np.argmax(y\_train, axis=1)

)

class\_weights = {i: weight for i, weight in enumerate(class\_weights)}

# Compile and train model

model\_transformer.compile(

    optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001),

    loss='categorical\_crossentropy',

    metrics=['accuracy']

)

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)

]

history\_transformer = model\_transformer.fit(

    X\_train, y\_train,

    epochs=50,

    batch\_size=128,

    validation\_data=(X\_test, y\_test),

    class\_weight=class\_weights,

    callbacks=callbacks

)

# Evaluate model

loss\_transformer, accuracy\_transformer = model\_transformer.evaluate(X\_test, y\_test)

print(f'Test Accuracy: {accuracy\_transformer \* 100:.2f}%')

# Classification report

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:")

print(classification\_report(y\_true\_classes, y\_pred\_classes, zero\_division=1))

# Confusion Matrix

cm = confusion\_matrix(y\_true\_classes, y\_pred\_classes)

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=np.unique(y\_true\_classes))

disp.plot(cmap=plt.cm.Blues)

plt.title("Confusion Matrix")

plt.show()

# Accuracy and Loss plots

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('Model Accuracy')

plt.show()

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('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 [۰, ۱] 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**

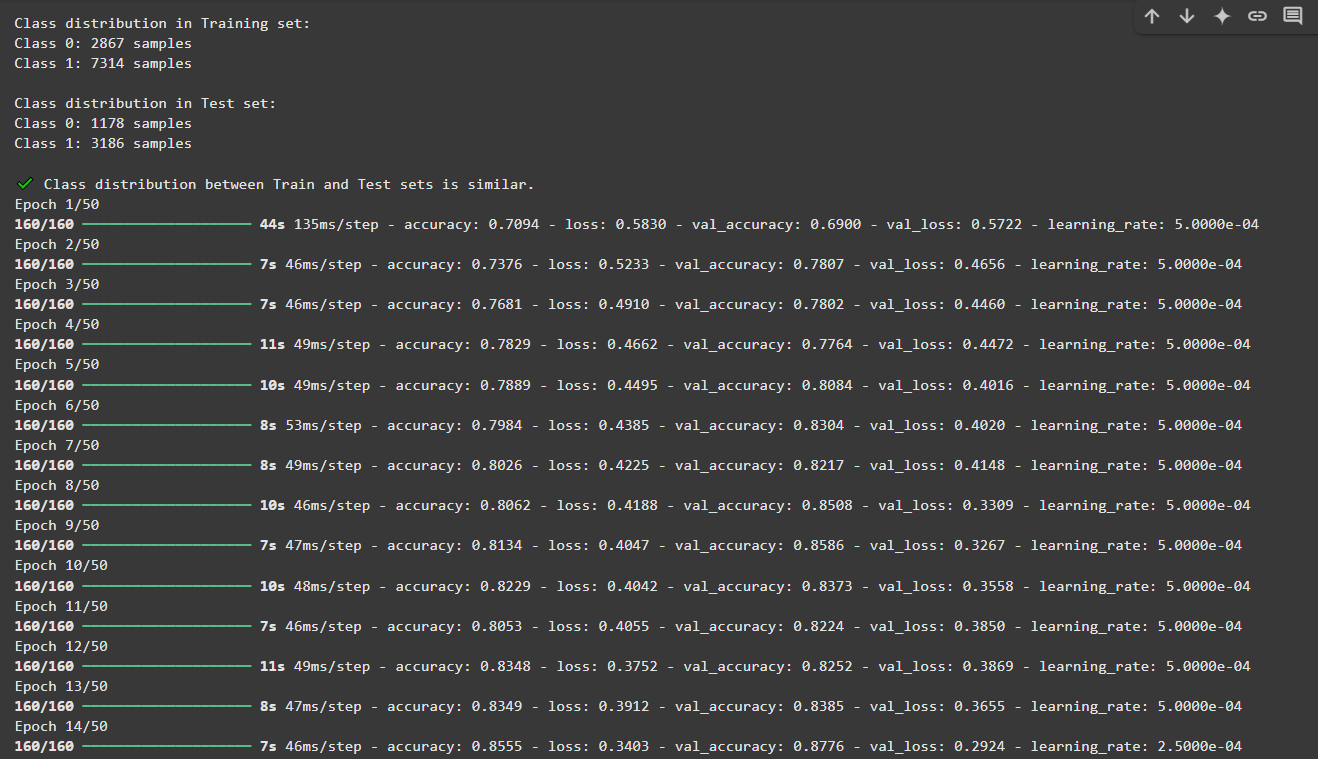
* categorical\_crossentropy : تابع هزینه برای مسائل چندکلاسه.
* EarlyStopping : آموزش را هنگام بهبود ندادن متوقف می‌کند.
* ReduceLROnPlateau : نرخ یادگیری را در صورت عدم بهبود کاهش می‌دهد.

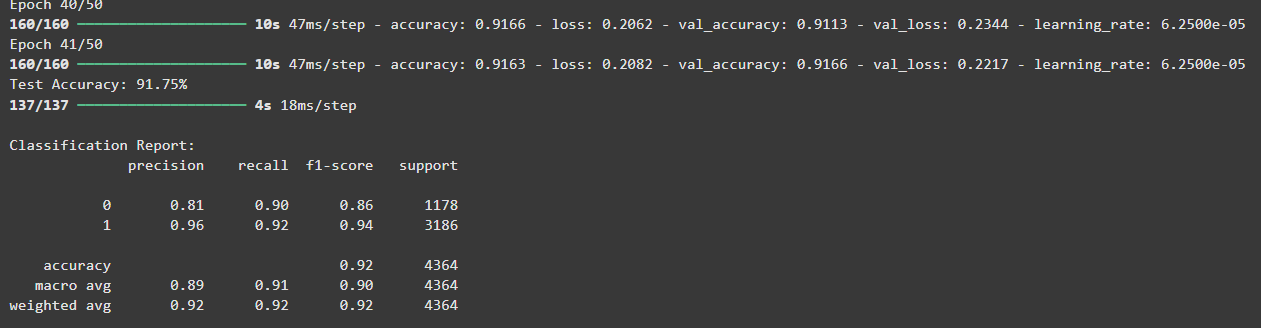
**بخش 11: ارزیابی و مصورسازی نتایج**

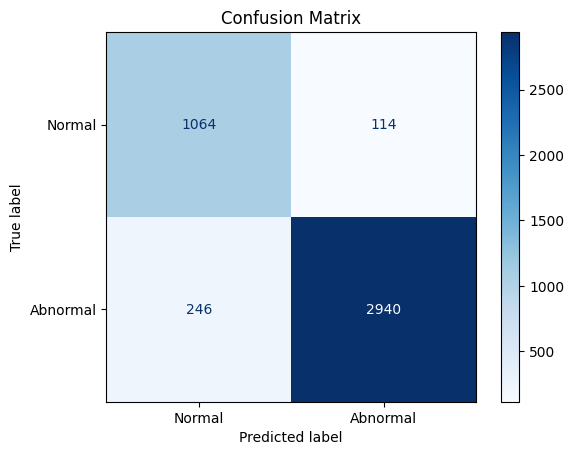
* تمامی نمودارها و ارقام نتایج را در بخش بعدی برای هر دو مسئله قرار داده‌ام.

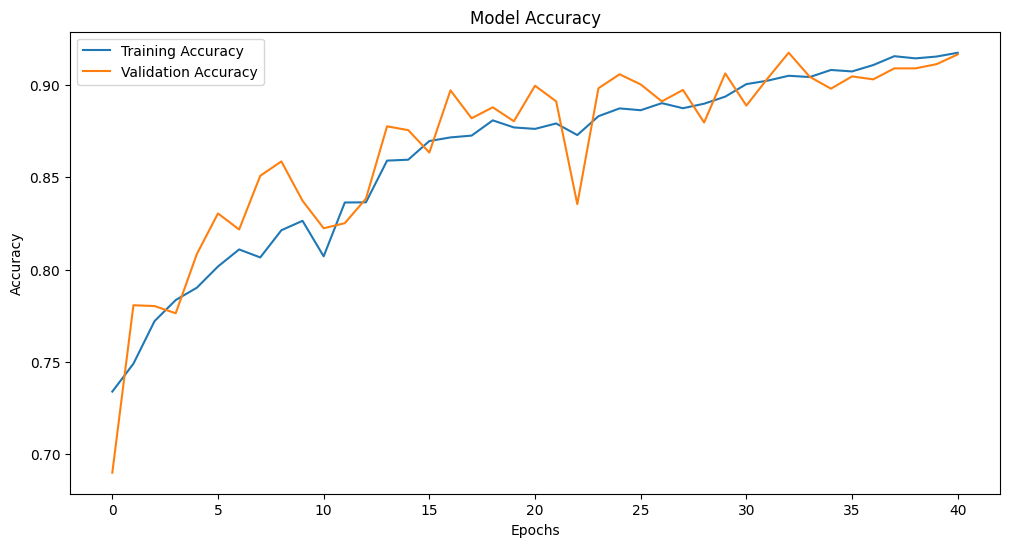
## 4-رسم نمودارها و ارتباطات:

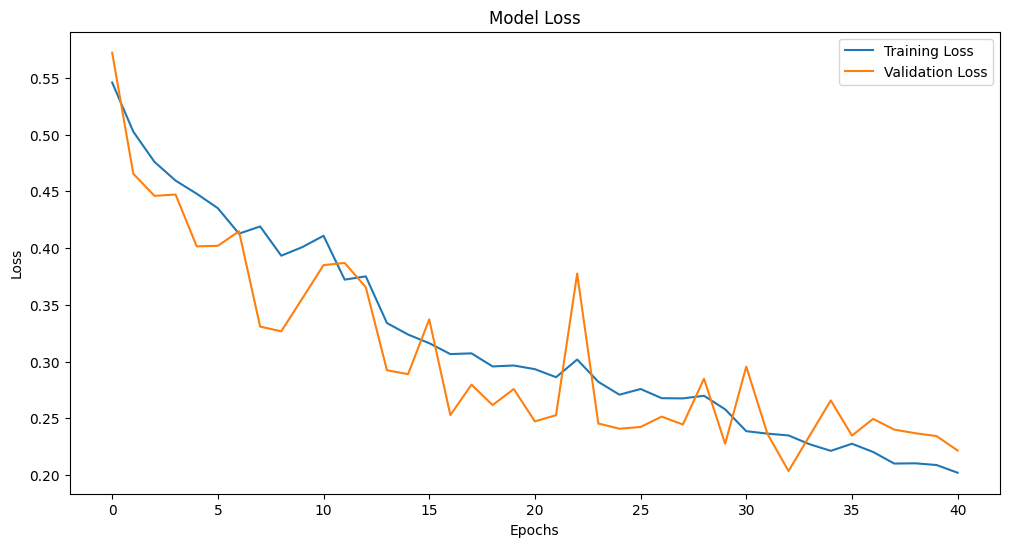
مطابق پروژه ی قبلی از پلات پایتون برای رسم نتایج استفاده کرده تا ببنیم داده ها در کجا قرار دارند و دید کلی از قرار گیری ان پیش بینی ها داشته باشیم:

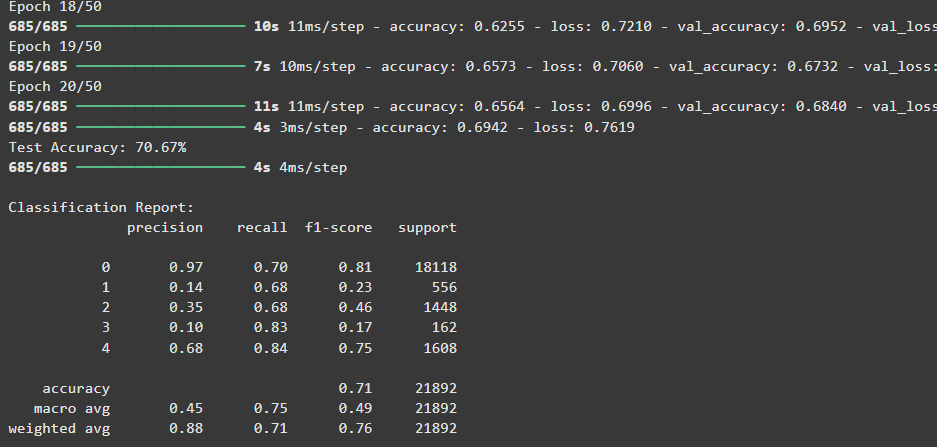


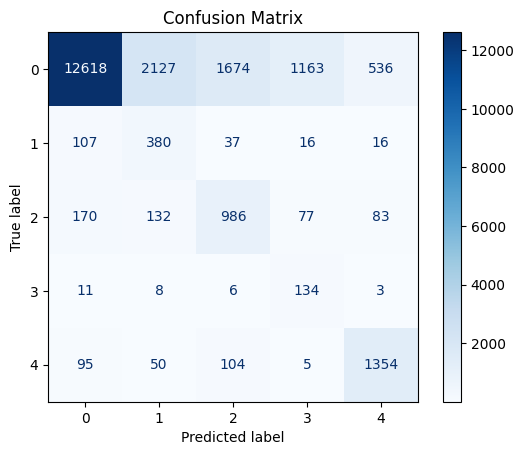


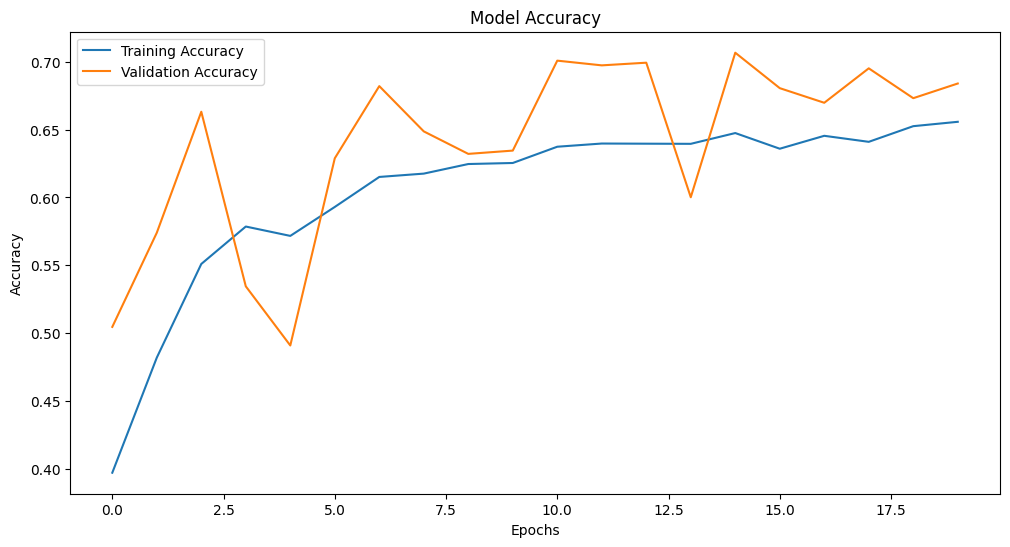
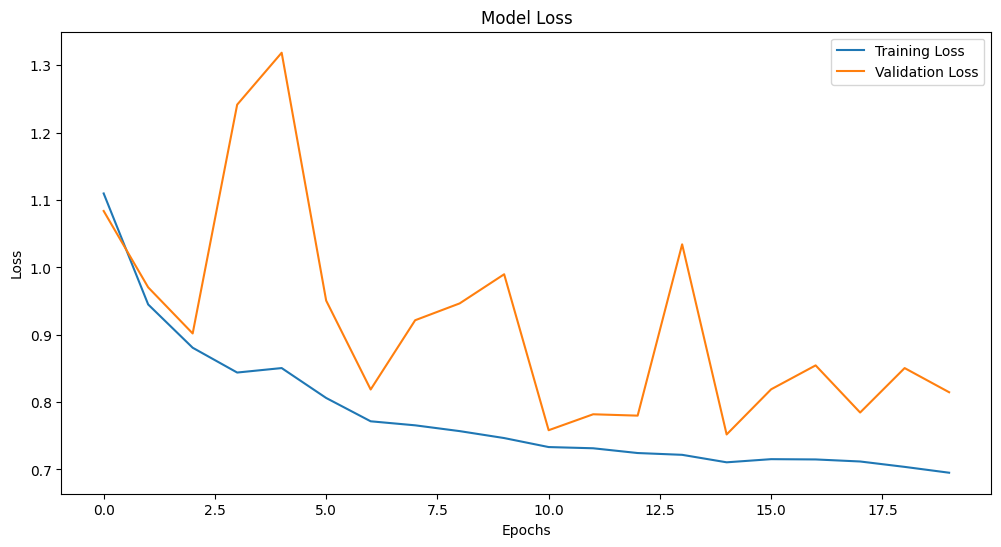










## 5- مراجع

[https://github.com](https://github.com/features/copilot)

[https://stackoverflow.com/questions](https://www.jpassion.com/java-development-tools/codepro)

<https://www.wikipedia.org/>

[https://colab.research.google.com/](https://colab.research.google.com/%20o)

[https://www.tensorflow.org/guide/](https://scikit-learn.org/stable/supervised_learning.html)

[https://keras.io/](https://medium.com/@brandon93.w/regression-model-evaluation-metrics-r-squared-adjusted-r-squared-mse-rmse-and-mae-24dcc0e4cbd3)

<https://github.com/M-Amin-Kiani/TimeSeries-ECG-Signal>

<https://sisoog.com/what-is-electrocardiogram/>

<https://medium.com/@lalesena/how-to-analyze-ecgs-with-python-396e34ece937>

<https://gist.github.com/emckiernan/3e7e86a48256777e9e6a44ede032d938>

<https://thesai.org/Downloads/Volume13No11/Paper_39-Transformer_based_Neural_Network_for_Electrocardiogram_Classification.pdf>