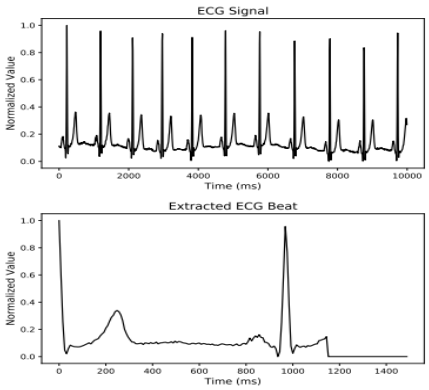
دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

گزارش پروژه‌ی سوم یادگیری عمیق

Time Series - ECG



پدیدآورنده:

محمد امین کیانی

۴۰۰۳۶۱۳۰۵۲

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

استاد درس: جناب اقای دکتر کیانی

نیمسال اول تحصیلی ۰۴-۱۴۰۳

فهرست مطالب

مستندات

۱-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:

۱. 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.

۲. 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.

۱. **ماسک‌گذاری صفرها** قبل از ورودی مدل با استفاده از tf.keras.layers.Masking.

۲. **اصلاح Transformer Encoder** تا هنگام محاسبه توجه (Attention) به صفرها توجه نکند.

۳. **اضافه کردن مکانیزم ماسک به لایه‌های MultiHeadAttention.**

۳. 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 ۹۸ درصد می‌توانستند باشند.):

❊ *LSTM برای* دسته‌بندی ← ۷۶٪

❊ *RNN* برای دسته‌بندی ← ۶۵٪

❊ transformer_encoder برای دسته‌بندی ← ۹۱٪

ابتدا، باید داده‌ها را که شامل ۲ 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.

۲- 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()
```

۱- کتابخانه‌ها

numpy وpandas : 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.

۲- 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 ۰ برای داده‌های سالم و ۱ 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=۰.۷ : ۷۰٪ داده‌ها برای آموزش و ۳۰٪ 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 tupleتبدیل 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=۰.۰۵ :اگر تفاوت بین توزیع‌ها کمتر از ۵٪ باشد، به‌عنوان مشابه در نظر گرفته می‌شود.

هدف کلی تابع

۱. **هم‌پوشانی نمونه‌ها** :اطمینان حاصل می‌کند که داده‌های آموزشی و آزمایشی کاملاً جدا از یکدیگر هستند.
۲. **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 RobustScaler به 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.

$$X' = \frac{X - \min X}{\max X - \min X}$$

- **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 میانگین ۰ و انحراف معیار ۱ داشته باشند.

$$X' = \frac{X - \mu}{\sigma}$$

- **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

RobustScalerروشی 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

$$Q1 - Q3 = IQR$$

$$X' = \frac{X - \text{Median}(X)}{\text{IQR}}$$

- 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 وIQR ، 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:

۱. 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.

۲. 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.

۳. کاهش نویز:

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

۴. 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:

۱. 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.

۲. 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.

۳. 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؟

۱. 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.

۲. 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.

۳. 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 (۰ یا ۱) است، بنابراین نیازی به دیکودر نیست.

- مزایای این انتخاب‌ها در مدل ما؟

۱. تبدیل فوریه:

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

۲. استفاده از انکودر ترنسفورمر:

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

۳. کارایی بالاتر:

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

۴. 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 ۱ برای سیگنال‌های تک‌کاناله است. (تعداد نمونه‌ها، ویژگی‌ها، ۱)
- بلوک CNN :

Conv۱D :استخراج ویژگی‌های محلی سیگنال.

filters=۶۴ :تعداد فیلترها.

kernel_size=۳ :اندازه‌ی هسته.

BatchNormalization :تسریع آموزش و تثبیت گرادیان‌ها.

MaxPooling۱D :کاهش اندازه‌ی داده‌ها.

Dropout(۰.۴) :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 متراکمFeed-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 = ۴ :تعداد سرهای توجه. مقدار ۴ برای تعادل بین دقت و سرعت محاسباتی انتخاب شده است.

ff_dim = ۱۲۸ :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:

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

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

۲. **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:

۱. لایه اول: ابعاد داده را از مقدار اولیه (ابعاد ورودی) به مقدار بزرگتر (مانندff_dim) افزایش می‌دهد.

۲. 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_dimمعمولاً 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:

GlobalAveragePooling۱D :کاهش ابعاد ویژگی‌ها.

Dense(۶۴) :لایه‌ی کاملاً متصل با ۶۴ نورون.

Dense(۱, 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=۰.۰۰۰۵) :نرخ یادگیری کوچک برای بهبود همگرایی.

batch_size=۶۴ :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 F۱.
- 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 ۲: جدا کردن ویژگی‌ها و برچسب‌ها

iloc[:, :-۱] : تمام ستون‌های به‌جز ستون آخر که مربوط به برچسب‌هاست به‌عنوان ویژگی انتخاب می‌شوند.

iloc[:, -۱] : ستون آخر (برچسب‌ها) جدا می‌شود.

بخش ۳: 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 ۴: 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 ۵: 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 ۶: 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 ۷: 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 ۸: 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 ۹: 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 ۱۰: 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 ۱۱: ارزیابی و مصورسازی نتایج

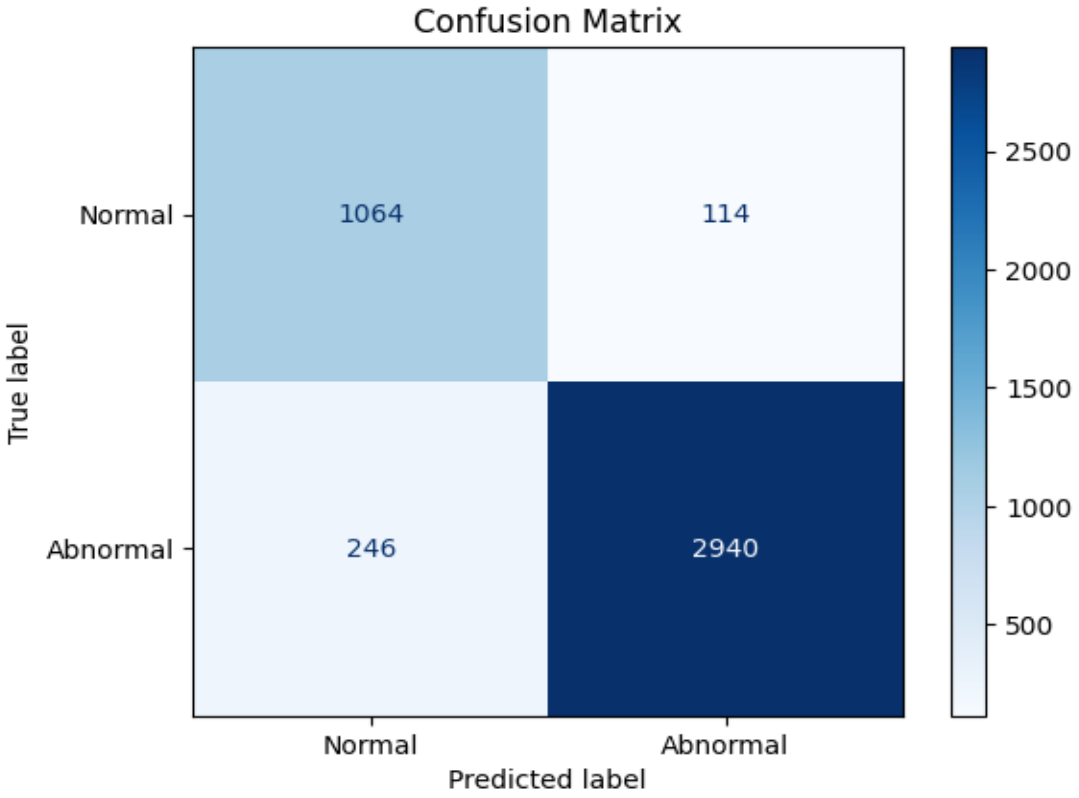
- تمامی نمودارها و ارقام نتایج را در بخش بعدی برای هر دو مسئله قرار داده‌ام.

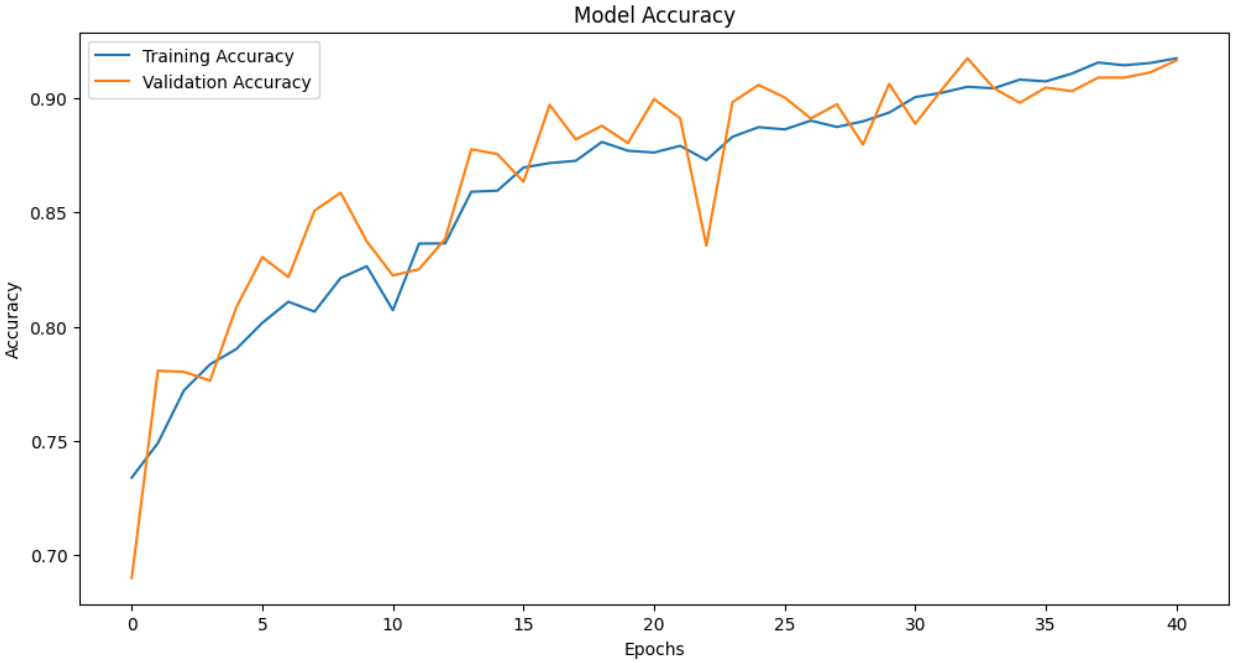
٤-رسم نمودارها و ارتباطات:

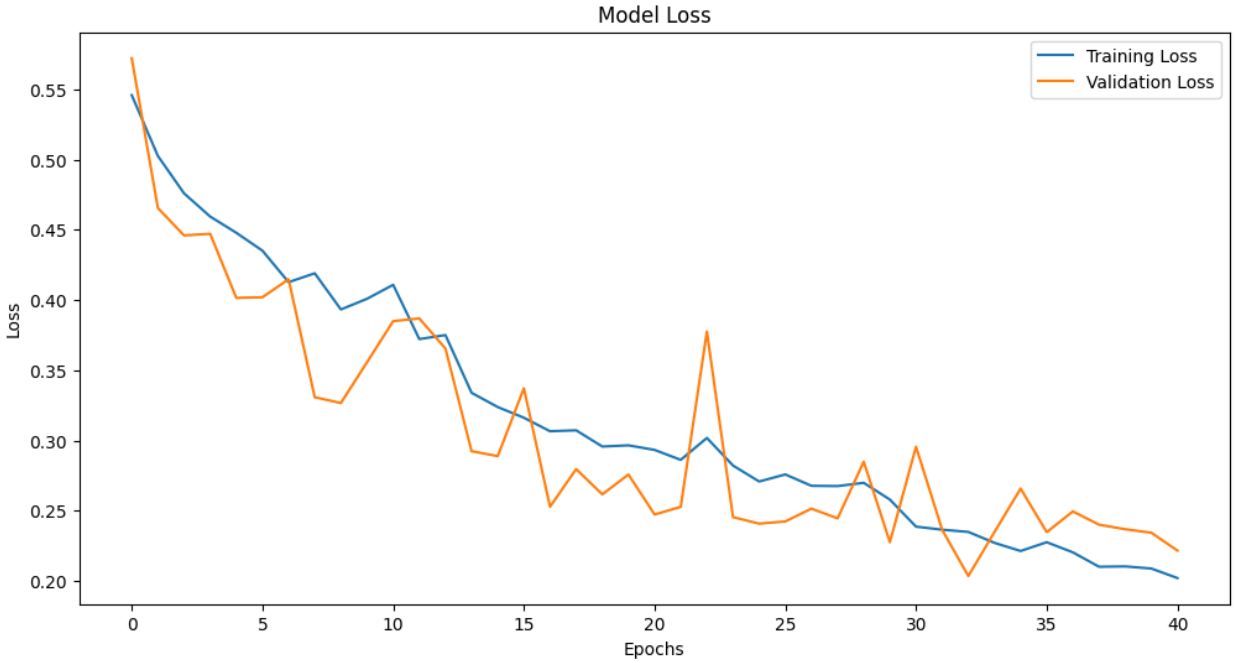
مطابق پروژه ی قبلی از پلات پایتون برای رسم نتایج استفاده کرده تا ببنیم داده ها در کجا قرار دارند و دید کلی از قرار گیری ان پیش بینی ها داشته باشیم:

```
Class distribution in Training set:
Class 0: 2867 samples
Class 1: 7314 samples

Class distribution in Test set:
Class 0: 1178 samples
Class 1: 3186 samples

✓ Class distribution between Train and Test sets is similar.
Epoch 1/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 44s 135ms/step - accuracy: 0.7094 - loss: 0.5830 - val_accuracy: 0.6900 - val_loss: 0.5722 - learning_rate: 5.0000e-04
Epoch 2/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 7s 46ms/step - accuracy: 0.7376 - loss: 0.5233 - val_accuracy: 0.7807 - val_loss: 0.4656 - learning_rate: 5.0000e-04
Epoch 3/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 7s 46ms/step - accuracy: 0.7681 - loss: 0.4910 - val_accuracy: 0.7802 - val_loss: 0.4460 - learning_rate: 5.0000e-04
Epoch 4/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 11s 49ms/step - accuracy: 0.7829 - loss: 0.4662 - val_accuracy: 0.7764 - val_loss: 0.4472 - learning_rate: 5.0000e-04
Epoch 5/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 10s 49ms/step - accuracy: 0.7889 - loss: 0.4495 - val_accuracy: 0.8084 - val_loss: 0.4016 - learning_rate: 5.0000e-04
Epoch 6/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 8s 53ms/step - accuracy: 0.7984 - loss: 0.4385 - val_accuracy: 0.8304 - val_loss: 0.4020 - learning_rate: 5.0000e-04
Epoch 7/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 8s 49ms/step - accuracy: 0.8026 - loss: 0.4225 - val_accuracy: 0.8217 - val_loss: 0.4148 - learning_rate: 5.0000e-04
Epoch 8/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 10s 46ms/step - accuracy: 0.8062 - loss: 0.4188 - val_accuracy: 0.8508 - val_loss: 0.3309 - learning_rate: 5.0000e-04
Epoch 9/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 7s 47ms/step - accuracy: 0.8134 - loss: 0.4047 - val_accuracy: 0.8586 - val_loss: 0.3267 - learning_rate: 5.0000e-04
Epoch 10/50
```

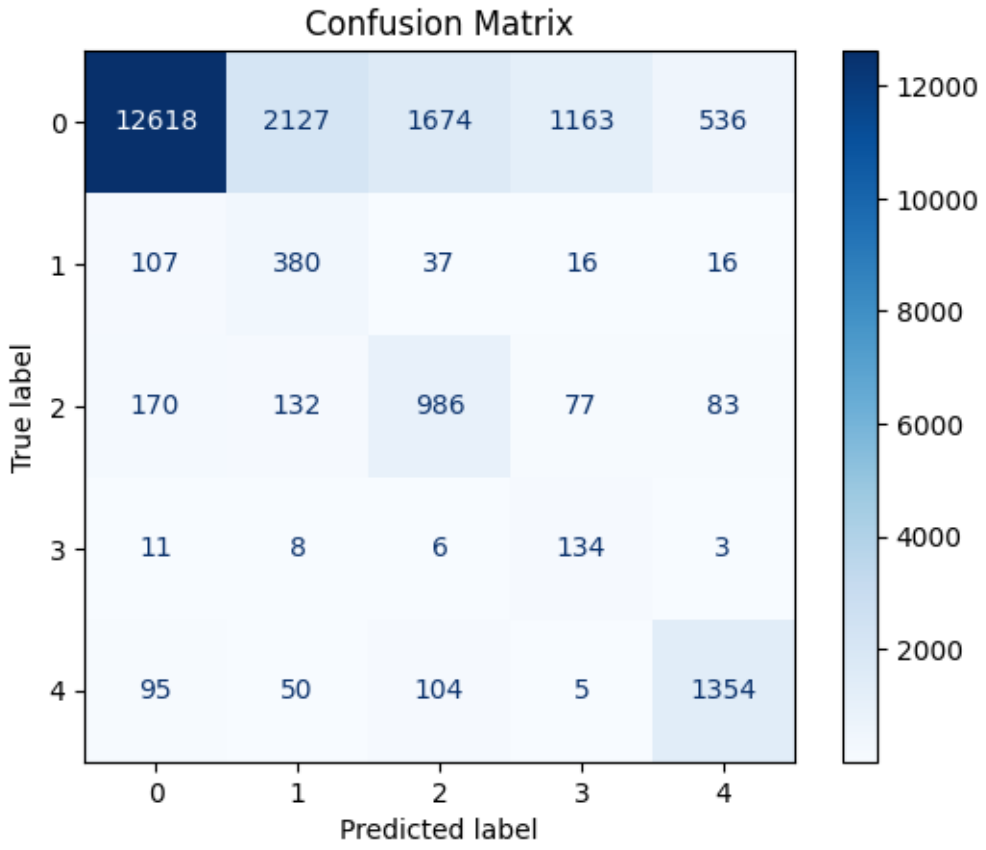
```
Epoch 40/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 10s 47ms/step - accuracy: 0.9166 - loss: 0.2062 - val_accuracy: 0.9113 - val_loss: 0.2344 - learning_rate: 6.2500e-05
Epoch 41/50
160/160 ━━━━━━━━━━━━━━━━━━━━ 10s 47ms/step - accuracy: 0.9163 - loss: 0.2082 - val_accuracy: 0.9166 - val_loss: 0.2217 - learning_rate: 6.2500e-05
Test Accuracy: 91.75%
137/137 ━━━━━━━━━━━━━━━━━━━━ 4s 18ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.81      0.90      0.86      1178
           1       0.96      0.92      0.94      3186

    accuracy                           0.92      4364
   macro avg       0.89      0.91      0.90      4364
weighted avg       0.92      0.92      0.92      4364
```

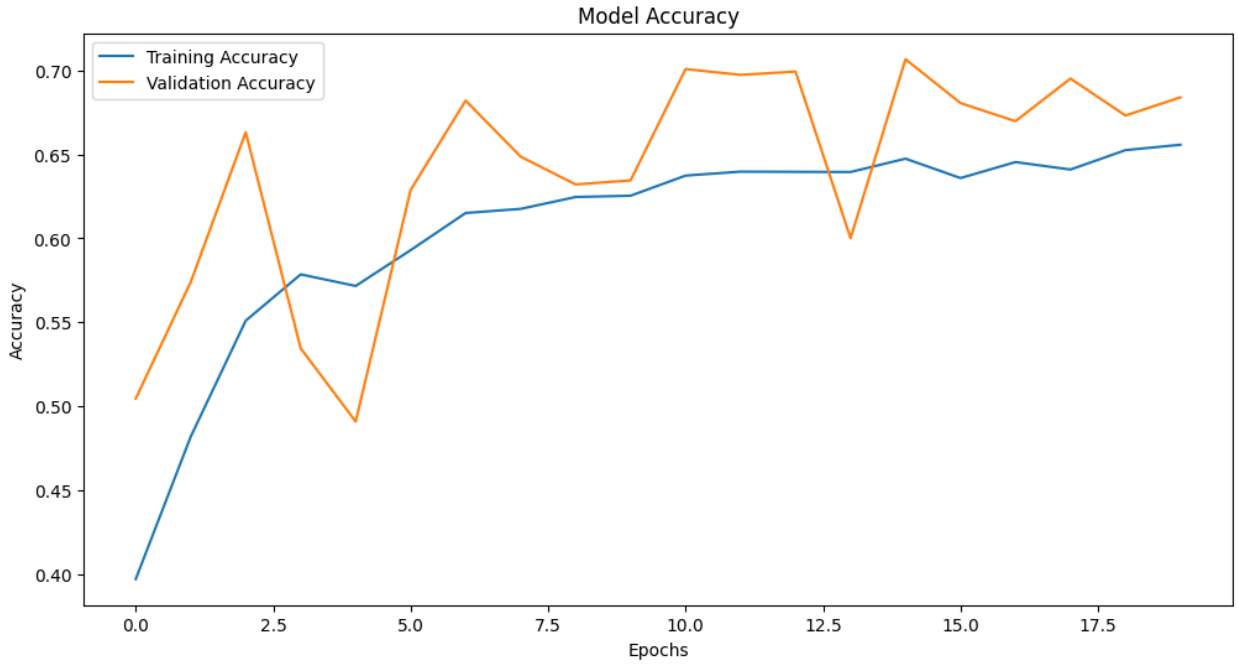


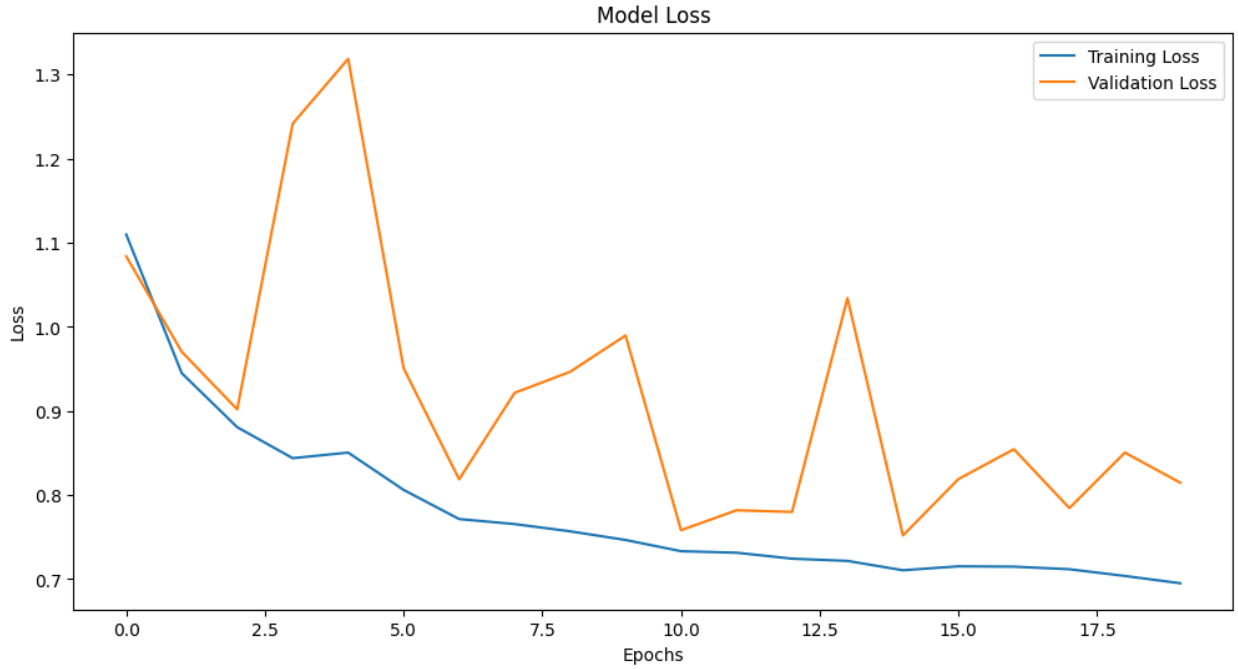

Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.70
0.75
0.80
0.85
0.90
0
5
10
15
20
25
30
35
40


Model Loss
Training Loss
Validation Loss
Loss
Epochs
0.20
0.25
0.30
0.35
0.40
0.45
0.50
0.55
0
5
10
15
20
25
30
35
40

```
Epoch 18/50
685/685 ━━━━━━━━━━━━━━━━━━━━ 10s 11ms/step - accuracy: 0.6255 - loss: 0.7210 - val_accuracy: 0.6952 - val_loss
Epoch 19/50
685/685 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - accuracy: 0.6573 - loss: 0.7060 - val_accuracy: 0.6732 - val_loss:
Epoch 20/50
685/685 ━━━━━━━━━━━━━━━━━━━━ 11s 11ms/step - accuracy: 0.6564 - loss: 0.6996 - val_accuracy: 0.6840 - val_loss
685/685 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.6942 - loss: 0.7619
Test Accuracy: 70.67%
685/685 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step

Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.70      0.81     18118
           1       0.14      0.68      0.23       556
           2       0.35      0.68      0.46      1448
           3       0.10      0.83      0.17       162
           4       0.68      0.84      0.75      1608

    accuracy                           0.71     21892
   macro avg       0.45      0.75      0.49     21892
weighted avg       0.88      0.71      0.76     21892
```




Model Accuracy
Training Accuracy
Validation Accuracy
Accuracy
Epochs
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Model Loss
Training Loss
Validation Loss
Loss
Epochs
0.7
0.8
0.9
1.0
1.1
1.2
1.3
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

5- مراجع

[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