دانشگاه اصفهان – دانشکده مهندسی کامپیوتر

یادگیری عمیق

استاد: دکتر محمد کیانی

کلاس‌بندی ضربان قلب

دیتاست: PTB_DB

شیدا عابدپور

4003623025

بارگذاری و پیش‌پردازش داده‌ها:

در این بخش، داده‌های سیگنال ECG از فایل‌های CSV بارگذاری می‌شود و مراحل مختلف پیش‌پردازش برای آماده‌سازی داده‌ها برای آموزش مدل انجام می‌شود.

```
def load_and_preprocess_data(self):
    """Loads the PTBDB dataset, normalizes ECG signals, and
splits into train, validation, and test sets."""

    # Load the datasets
    df_normal = pd.read_csv(self.normal_csv, header=None)
    df_abnormal = pd.read_csv(self.abnormal_csv,
header=None)

    # Combine normal and abnormal ECGs
    df = pd.concat([df_normal, df_abnormal], axis=0)

    # Features and labels
    X = df.iloc[:, :-1].values  # All columns except the
last one
    y = df.iloc[:, -1].values   # The last column is the
label
```

```
  # Split into train+val and test sets
    X_train_val, X_test, y_train_val, y_test =
train_test_split(
        X, y, test_size=self.test_size,
random_state=self.random_state, stratify=y
    )
```

```
    # Split train+val into train and validation sets
    val_ratio = self.val_size / (1 - self.test_size)  #
Adjust val_ratio according to the remaining data
    X_train, X_val, y_train, y_val = train_test_split(
        X_train_val, y_train_val, test_size=val_ratio,
random_state=self.random_state, stratify=y_train_val
    )
```

- **تقسیم داده‌ها**: داده‌ها به دو بخش آموزش/اعتبارسنجی و تست تقسیم می‌شوند. در اینجا از `train_test_split()` برای تقسیم داده‌ها استفاده می‌شود و از پارامتر `stratify=y` برای حفظ تعادل بین دسته‌ها در هر بخش استفاده شده است.
- **تقسیم داده‌های آموزش/اعتبارسنجی**: پس از تقسیم داده‌ها به آموزش و تست، داده‌های آموزش به دو بخش آموزش و اعتبارسنجی تقسیم می‌شوند.

```
  # Normalize the training features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)  # Fit on
training data and transform

    # Transform validation and test data using the same
scaler
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)
```

- **نرمال‌سازی داده‌ها**: داده‌های سیگنال ECG باید نرمالیزه شوند تا مدل به طور مؤثرتری یاد بگیرد. برای این کار از `StandardScaler` استفاده می‌شود که مقیاس داده‌ها را استاندارد می‌کند (میانگین صفر و انحراف معیار یک).
  - ابتدا داده‌های آموزشی نرمالیزه می‌شوند (`fit_transform`).
  - سپس داده‌های اعتبارسنجی و تست با استفاده از همان مقیاس که از داده‌های آموزشی به‌دست آمده، نرمالیزه می‌شوند (`transform`).

```
  # Reshape the data for LSTM input (samples, timesteps,
features)
    X_train = X_train.reshape(X_train.shape[0],
X_train.shape[1], 1)
    X_val = X_val.reshape(X_val.shape[0], X_val.shape[1],
1)
    X_test = X_test.reshape(X_test.shape[0],
X_test.shape[1], 1)
```

- **تبدیل به فرم مناسب برای LSTM**: داده‌ها باید به فرم `(تعداد نمونه‌ها، تعداد ویژگی‌ها، 1)` تغییر یابند تا بتوانند به مدل LSTM وارد شوند. این کار با استفاده از `reshape()` انجام می‌شود.

ساخت مدل:

در این بخش، مدل با استفاده از ترکیب لایه‌های `Conv1D` (برای استخراج ویژگی‌ها از سیگنال‌های ECG) و `BiLSTM` (برای پردازش دنباله‌های زمانی سیگنال‌ها) ساخته می‌شود.

```
def build_model(self):
    """Builds an advanced CNN + BiLSTM + BiGRU model."""
    inputs = Input(shape=(self.X_train.shape[1], 1))

    x = Conv1D(filters=8, kernel_size=3, activation='relu',
padding='same')(inputs)
    x = BatchNormalization()(x)
```

- **ورودی مدل**: ورودی مدل به شکل `(تعداد ویژگی‌ها، 1)` است که به شبکه داده می‌شود.
- **لایه Conv1D**: این لایه به مدل کمک می‌کند که ویژگی‌های زمانی سیگنال‌های ECG را استخراج کند. از `filters=8` برای تعداد فیلترها و `kernel_size=3` برای اندازه هسته کانولوشن استفاده می‌شود.
- **BatchNormalization**: این لایه برای نرمال‌سازی داده‌ها در هر مینی‌بچ استفاده می‌شود تا یادگیری مدل بهبود یابد.

```
x = Bidirectional(LSTM(128, return_sequences=True,
dropout=0.2, recurrent_dropout=0.2))(x)
    x = Bidirectional(LSTM(128, return_sequences=True,
dropout=0.25, recurrent_dropout=0.2))(x)
```

```
    x = Bidirectional(LSTM(64, return_sequences=True,
dropout=0.3, recurrent_dropout=0.2))(x)
    x = Bidirectional(LSTM(64, return_sequences=True,
dropout=0.3, recurrent_dropout=0.2))(x)
```

- **لایه‌های BiLSTM**: از لایه‌های `Bidirectional LSTM` برای پردازش دنباله‌های زمانی استفاده شده است. این لایه‌ها به مدل کمک می‌کنند تا اطلاعات را از هر دو جهت زمانی (قبل و بعد) دریافت کرده و بهتر یاد بگیرد.
- **Dropout**: در هر لایه LSTM از `dropout` و `recurrent_dropout` برای کاهش اورفیتینگ استفاده شده است. این کار باعث می‌شود که مدل از وابستگی به نورون‌های خاص جلوگیری کند.

```
x = Flatten()(x)

    x = Dense(128, activation='relu')(x)
    x = Dropout(0.2)(x)
    x = Dense(64, activation='relu')(x)
    x = Dropout(0.2)(x)
    outputs = Dense(1, activation='sigmoid')(x)
```

- **Flatten**: لایه `Flatten` برای مسطح کردن داده‌های خروجی از LSTMها به یک بعد استفاده می‌شود تا بتوان آنها را به لایه‌های Dense وارد کرد.

- **لایه‌های Dense**: از لایه‌های `Dense` برای طبقه‌بندی استفاده می‌شود. هر لایه دارای فعال‌سازی `relu` است و برای جلوگیری از اورفیتینگ از `Dropout` استفاده می‌شود.
- **خروجی مدل**: لایه خروجی یک لایه `Dense` با فعال‌سازی `sigmoid` است که برای طبقه‌بندی باینری مناسب است.

```
self.model = Model(inputs, outputs)
    self.model.summary()

    optimizer = Adam(learning_rate=1e-3)
    self.model.compile(loss='binary_crossentropy',
optimizer=optimizer, metrics=['accuracy'])
```

- **مدل و کامپایل**: مدل ساخته شده با استفاده از `Model()` و سپس با استفاده از الگوریتم `Adam` کامپایل می‌شود. تابع هزینه `binary_crossentropy` برای طبقه‌بندی باینری انتخاب شده است.

آموزش مدل:

در این بخش، مدل آموزش داده می‌شود.

```
def train_model(self, epochs=150, batch_size=64):
    if self.model is None:
        raise ValueError("Model has not been built. Call
build_model() first.")

    callbacks = [

tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=5, restore_best_weights=True),

tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
factor=0.5, patience=5, min_lr=1e-5)
    ]

    self.history = self.model.fit(
        self.X_train, self.y_train,
        epochs=epochs, batch_size=batch_size,
        validation_data=(self.X_val, self.y_val),
        callbacks=callbacks
    )
```

- **کال‌بک‌ها**: از دو کال‌بک `EarlyStopping` و `ReduceLROnPlateau` استفاده می‌شود:
  - `EarlyStopping` برای متوقف کردن آموزش اگر مدل بهبود نداشته باشد.
  - `ReduceLROnPlateau` برای کاهش نرخ یادگیری اگر مدل بهبود نیابد.

- **آموزش مدل**: مدل با استفاده از داده‌های آموزشی و اعتبارسنجی آموزش داده می‌شود. تعداد اپوک‌ها و اندازه بچ به عنوان پارامتر ورودی برای تابع `train_model` داده می‌شود.

ارزیابی مدل:

در این بخش، مدل ارزیابی می‌شود و دقت آن محاسبه می‌شود.

```
def evaluate_model(self):
    test_loss, test_acc = self.model.evaluate(self.X_test,
self.y_test)
    print(f

"Test Loss: {test_loss:.4f}, Test Accuracy:
{test_acc:.4f}")
```

ارزیابی مدل با استفاده از داده‌های تست ارزیابی می‌شود و دقت آن چاپ می‌شود.

```
    y_pred = (self.model.predict(self.X_test) >
0.5).astype("int32")
    print(classification_report(self.y_test, y_pred))
    print(confusion_matrix(self.y_test, y_pred))
```

نتیجه‌گیری:

این مدل ترکیبی از CNN و BiLSTM است که برای طبقه‌بندی سیگنال‌های ECG به دو دسته نرمال و غیر نرمال طراحی شده است. با استفاده از لایه‌های BiLSTM، مدل توانسته اطلاعات زمانی سیگنال‌ها را از هر دو جهت زمانی پردازش کند و دقت بالایی در طبقه‌بندی سیگنال‌ها ارائه دهد.

مدل، از لایه‌های مختلفی به‌طور خاص برای بهبود دقت و توانایی تشخیص سیگنال‌های ECG استفاده شده است. در اینجا هر لایه و دلیل استفاده از آن توضیح داده می‌شود:

1. لایه Conv1D:

```
x = Conv1D(filters=8, kernel_size=3, activation='relu', padding='same')(inputs)
```

- **هدف**: لایه `Conv1D` برای استخراج ویژگی‌ها از سیگنال‌های ورودی (که به‌صورت دنباله‌های زمانی هستند) استفاده می‌شود.
- **دلیل انتخاب**: سیگنال‌های ECG توالی‌های زمانی هستند و لایه‌های کانولوشن (Convolutional) به‌خوبی می‌توانند ویژگی‌های محلی مهم (مثل الگوهای خاص در سیگنال‌ها) را شناسایی کنند. این لایه با استفاده از فیلترهایی که اندازه آن‌ها ۳ است، ویژگی‌های محلی این سیگنال‌ها را استخراج می‌کند.
- **`'activation='relu`**: استفاده از تابع فعال‌سازی ReLU برای افزایش غیرخطی بودن مدل و جلوگیری از مشکلات مشتق صفر است.
- **`'padding='same`**: این پارامتر باعث می‌شود که اندازه خروجی همانند ورودی باشد و اطلاعات بیشتری از سیگنال حفظ شود.

2. لایه BatchNormalization:

```
x = BatchNormalization()(x)
```

- **هدف**: نرمال‌سازی دسته‌ای (Batch Normalization) برای بهبود فرآیند آموزش مدل استفاده می‌شود.
- **دلیل انتخاب**: این لایه به کنترل توزیع داده‌ها در هر لایه کمک می‌کند و باعث می‌شود که روند آموزش با ثبات بیشتری انجام شود. نرمال‌سازی برای کاهش اثرات نوسانات و بهبود سرعت همگرایی مدل مهم است.

3. لایه‌های BiLSTM:

```
x = Bidirectional(LSTM(128, return_sequences=True,
dropout=0.2, recurrent_dropout=0.2))(x)
x = Bidirectional(LSTM(128, return_sequences=True,
dropout=0.25, recurrent_dropout=0.2))(x)
x = Bidirectional(LSTM(64, return_sequences=True,
dropout=0.3, recurrent_dropout=0.2))(x)
x = Bidirectional(LSTM(64, return_sequences=True,
dropout=0.3, recurrent_dropout=0.2))(x)
```

- **هدف**: لایه‌های `Bidirectional LSTM` برای پردازش دنباله‌های زمانی سیگنال‌های ECG استفاده می‌شوند.
- **دلیل انتخاب**:

- **LSTM (Long Short-Term Memory)**: LSTM یک نوع شبکه عصبی بازگشتی (RNN) است که برای مدل‌سازی دنباله‌های زمانی طراحی شده و به‌ویژه برای یادگیری وابستگی‌های بلندمدت در داده‌ها بسیار مفید است. در سیگنال‌های ECG، اطلاعات زمانی و وابستگی‌های طولانی‌مدت میان نمونه‌ها اهمیت دارند و LSTM می‌تواند این وابستگی‌ها را به‌خوبی یاد بگیرد.
- **Bidirectional**: استفاده از LSTM دوطرفه به این معنی است که مدل می‌تواند اطلاعات را از هر دو جهت دنباله زمانی (گذشته و آینده) یاد بگیرد. این ویژگی برای پردازش سیگنال‌های ECG که ممکن است ویژگی‌های مهمی در هر دو جهت زمانی داشته باشند، بسیار مفید است.
- **`dropout` و `recurrent_dropout`**: این پارامترها برای جلوگیری از اورفیتینگ استفاده می‌شوند. در حین آموزش، برخی از نورون‌ها به‌طور تصادفی غیرفعال می‌شوند تا مدل از وابستگی به نورون‌های خاص جلوگیری کند و عمومیت بهتری پیدا کند.

4. لایه Flatten:

```
x = Flatten()(x)
```

- **هدف**: لایه `Flatten` برای تبدیل داده‌های چندبعدی (که از لایه‌های LSTM به‌دست آمده) به یک بعدی استفاده می‌شود.

- **دلیل انتخاب:** خروجی از لایه‌های LSTM به‌صورت یک دنباله زمانی است، اما لایه‌های `Dense` (که در ادامه مدل استفاده می‌شوند) فقط از ورودی‌های یک بعدی پشتیبانی می‌کنند. بنابراین، باید داده‌ها را به یک بعدی مسطح (Flatten) کنیم تا بتوانیم آن‌ها را به لایه‌های بعدی وارد کنیم.

5. لایه‌های Dense:

```
x = Dense(128, activation='relu')(x)
x = Dropout(0.2)(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.2)(x)
```

- **هدف:** لایه‌های `Dense` برای انجام طبقه‌بندی نهایی و پیش‌بینی برچسب‌ها استفاده می‌شوند.
- **دلیل انتخاب:**
  - لایه‌های `Dense` به‌طور ویژه برای یادگیری ویژگی‌های تجریدی از داده‌های ورودی طراحی شده‌اند.
  - **`'activation='relu`**: استفاده از ReLU برای کاهش غیرخطی‌بودن و جلوگیری از مشکلات مشتق صفر.
  - **`Dropout`**: این لایه‌ها برای جلوگیری از اورفیتینگ استفاده می‌شوند و باعث می‌شوند که مدل به‌جای یادگیری جزئیات بیش از حد از داده‌های آموزشی، ویژگی‌های عمومی‌تری را یاد بگیرد.

6. لایه خروجی:

```
outputs = Dense(1, activation='sigmoid')(x)
```

- **هدف**: لایه خروجی برای پیش‌بینی نهایی استفاده می‌شود.
- **دلیل انتخاب**: برای مشکل طبقه‌بندی باینری (طبقه‌بندی سیگنال‌های نرمال و غیر نرمال ECG)، از یک لایه `Dense` با تنها یک نورون و تابع فعال‌سازی `sigmoid` استفاده می‌شود. تابع `sigmoid` خروجی مدل را بین 0 و 1 محدود می‌کند که می‌تواند به‌عنوان احتمال طبقه‌بندی سیگنال به‌عنوان نرمال یا غیر نرمال تفسیر شود.

Model: "functional_20"

Layer (type)	Output Shape	Param #
input_layer_24 (InputLayer)	(None, 187, 1)	0
conv1d_20 (Conv1D)	(None, 187, 8)	32
batch_normalization_20 (BatchNormalization)	(None, 187, 8)	32
bidirectional_68 (Bidirectional)	(None, 187, 256)	140,288
bidirectional_69 (Bidirectional)	(None, 187, 256)	394,240
bidirectional_70 (Bidirectional)	(None, 187, 128)	164,352
bidirectional_71 (Bidirectional)	(None, 187, 128)	98,816
flatten_5 (Flatten)	(None, 23936)	0
dense_70 (Dense)	(None, 128)	3,063,936
dropout_54 (Dropout)	(None, 128)	0
dense_71 (Dense)	(None, 64)	8,256
dropout_55 (Dropout)	(None, 64)	0
dense_72 (Dense)	(None, 1)	65

Total params: 3,870,017 (14.76 MB)

نتایج:

```
Test Accuracy: 99.08%
69/69 ━━━━━━━━━━━━━━━━━━━━ 30s 422ms/step

Classification Report:
              precision    recall  f1-score   support

         0.0     0.9900    0.9769    0.9834       607
         1.0     0.9912    0.9962    0.9937      1576

    accuracy                         0.9908      2183
   macro avg     0.9906    0.9866    0.9885      2183
weighted avg     0.9908    0.9908    0.9908      2183
```

مدل عملکردی **بسیار متوازن بین دو کلاس دارد**، زیرا دقت و حساسیت در هر دو کلاس بسیار بالا و نزدیک به هم هستند.

حساسیت کلاس 1 (99.62%) کمی بالاتر از کلاس 0 (97.69%) است، که نشان می‌دهد مدل بیشتر تمایل دارد که سیگنال‌های غیرنرمال را درست شناسایی کند.

این نکته مثبت است زیرا در کاربردهای پزشکی، شناسایی سیگنال‌های غیرنرمال مهم‌تر از شناسایی نرمال‌ها است (یعنی بهتر است مدل بیشتر خطای نوع اول داشته باشد تا خطای نوع دوم).