دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

# گزارش پروژه درس

# مقدمه ای بر یادگیری ماشین

# موضوع: سیگنال پزشکی

استاد: دکتر سیدین

دانشکده مهندسی برق

اعضای گروه:

حسنا اویار حسینی ۹۸۲۳۳۰۸

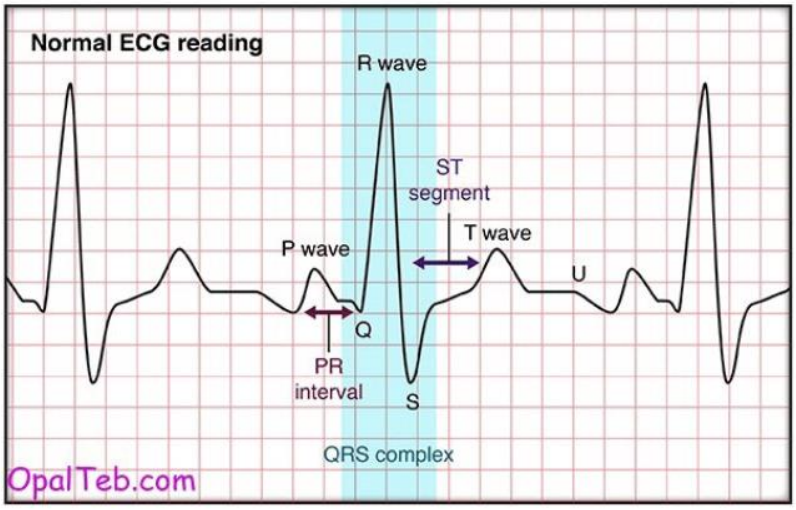
سمیرا سلجوقی ۹۸۲۳۰۴۸

پاییز و زمستان ۱۴۰۱

# فهرست

## سیگنال ECG

دستگاه الکتروکاردیوگرام یا همان نوار قلب وسیله ای است که می تواند سیگنال های قلب شما را دریافت و آن را ثبت نماید، این پروسیچر یک تست معمول برای سنجش عملکرد قلب و تعداد و شدت ضربان قلب است که در موقعیت های مختلف مورد استفاده قرار می گیرد، نوار ECG علاوه بر بیمارستان ها و برخی مطب ها در آمبولانس و نیز اتاق های عمل نیز موجود است.




موج : p فعالیت قلب در دهلیز ها را نشان می دهد و نشان دهنده انقباض دهلیز ها است، این موج به طور معمول قرینه و به شکل صاف است.

QRS: نشان دهنده انقباض بطن ها در قلب

موجT : نشان دهنده استراحت دهلیزها

در حقیقت قلب یک پمپ الکتریکی دو مرحله ای است که میزان پالسهای الکتریکی آن از روی پوست قابل اندازه گیری است و علاوه بر اینکه میزان قدرت و ضربان قلب را نشان می دهد می تواند به طور غیر مستقیم نشان دهنده میزان جریان خون عضلات قلب نیز باشدپالس های الکتریکی منتشر شده در سطح پوست توسط الکترود هایی که به قفسه سینه چسبانده می شود دریافت و به شکل یک نمودار الکتروکاردیوگرام بر روی کاغذ چاپ می شود.

برای گرفتن این تست سنسورهای خاصی به پوست متصل می‌شود. وظیفه این سنسورها تشخیص سیگنال‌های الکتریکی تولید شده توسط قلب است. وقتی این سنسورها سیگنال‌های ناشی از تپش قلب را دریافت کردند آن را به دستگاه ارسال می‌کنند تا روی کاغذ مخصوص ثبت شود.

## مجموعه داده

دیتاست این پروژه (PTB) مجموعه ای از ۵۴۹ ECG با وضوح بالا شامل خلاصه های بالینی برای هر رکورد است. از یک تا پنج رکورد ECG برای هر یک از ۲۹۴ نفر در دسترس است که شامل افراد سالم و همچنین بیماران مبتلا به انواع بیماری های قلبی می شود.

این مجموعه داده شامل ۵۴۹ رکورد از ۲۹۰ آزمودنی است (۱۷ تا ۸۷ سال، میانگین ۵۷/۲، ۲۰۹ مرد، میانگین سنی ۵۵,۵ سال، و ۸۱ زن، میانگین سنی ۶۱,۶ سال؛ سن برای ۱ زن و ۱۴ آزمودنی مرد ثبت نشده است). هر موضوع با یک تا پنج رکورد نمایش داده می شود. هیچ موضوعی با شماره ۱۲۴، ۱۳۲، ۱۳۴ یا ۱۶۱ وجود ندارد. هر رکورد شامل ۱۵ سیگنال اندازه گیری همزمان است

در فایل هدر (.hea) اکثر این سوابق ECG یک خلاصه بالینی دقیق، شامل سن، جنسیت، تشخیص، و در صورت لزوم، داده‌های مربوط به تاریخچه پزشکی، دارو و مداخلات، آسیب‌شناسی عروق کرونر، ونتریکولوگرافی، اکوکاردیوگرافی، و همودینامیک وجود دارد. . خلاصه بالینی برای ۲۲ نفر در دسترس نیست. کلاس های تشخیصی ۲۶۸ آزمودنی باقیمانده در زیر خلاصه شده است:

| Diagnostic class | Number of subjects |
|---|---|
| Myocardial infarction | 148 |
| Cardiomyopathy/Heart failure | 18 |
| Bundle branch block | 15 |
| Dysrhythmia | 14 |
| Myocardial hypertrophy | 7 |
| Valvular heart disease | 6 |
| Myocarditis | 4 |
| Miscellaneous | 4 |
| Healthy controls | 52 |

## پیش پردازش داده ها

برای استفاده از این دیتاست از آنجایی که حجم آن زیاد است، لازم است تا دیتاست را یک بار در درایو دانلود و ذخیره کرده و سپس با وصل کردن محیط کولب به درایو از دیتاست استفاده کنیم.

```
# Mount GoogleDrive
from google.colab import drive

drive.mount("drive")
```

```
Mounted at drive
```

```
# Run this section only the first time to download the dataset
!wget -r -N -c -np https://physionet.org/files/ptbdb/1.0.0/
!cp -r physionet.org/files/ptbdb/1.0.0/ drive/MyDrive/ml-dataset
!rm -r physionet.org/
```

در قسمت بعد با استفاده از کتابخانه wfdb که مختص این نوع دیتاست ساخته شده است، به عنوان یک نمونه یک داده را به نمایش گذاشته که شامل ویژگی های مختلفی است که ما از عنصر اول آن به عنوان داده اصلی و از کلاس reason for admission برای داده هدف استفاده می کنیم.

```
# A sample of dataset records
data = wfdb.rdsamp("drive/MyDrive/ml-dataset/patient001/s0010_re", channel_names=['i', 'ii'])
data
```

```
(array([[-0.2445, -0.229 ],
        [-0.2425, -0.2335],
        [-0.2415, -0.2345],
        ...,
        [ 0.152 ,  0.2695],
        [ 0.136 ,  0.256 ],
        [ 0.135 ,  0.2585]]),
 {'fs': 1000,
  'sig_len': 38400,
  'n_sig': 2,
  'base_date': None,
  'base_time': None,
  'units': ['mV', 'mV'],
  'sig_name': ['i', 'ii'],
  'comments': ['age: 81',
   'sex: female',
   'ECG date: 01/10/1990',
   'Diagnose:',
   'Reason for admission: Myocardial infarction',
   'Acute infarction (localization): infero-latera',
   'Former infarction (localization): no',
   'Additional diagnoses: Diabetes mellitus',
   'Smoker: no',
   'Number of coronary vessels involved: 1',
   'Infarction date (acute): 29-Sep-90',
   'Previous infarction (1) date: n/a',
   'Previous infarction (2) date: n/a',
   'Hemodynamics:',
   'Catheterization date: 16-Oct-90',
   'Ventriculography: Akinesia inferior wall',
   'Chest X-ray: Heart size upper limit of norm',
   'Peripheral blood Pressure (syst/diast):  140/80 mmHg',
   'Pulmonary artery pressure (at rest) (syst/diast): n/a',
   'Pulmonary artery pressure (at rest) (mean): n/a',
   'Pulmonary capillary wedge pressure (at rest): n/a',
   'Cardiac output (at rest): n/a',
   'Cardiac index (at rest): n/a',
   'Stroke volume index (at rest): n/a',
   'Pulmonary artery pressure (laod) (syst/diast): n/a',
   'Pulmonary artery pressure (laod) (mean): n/a',
   'Pulmonary capillary wedge pressure (load): n/a',
```

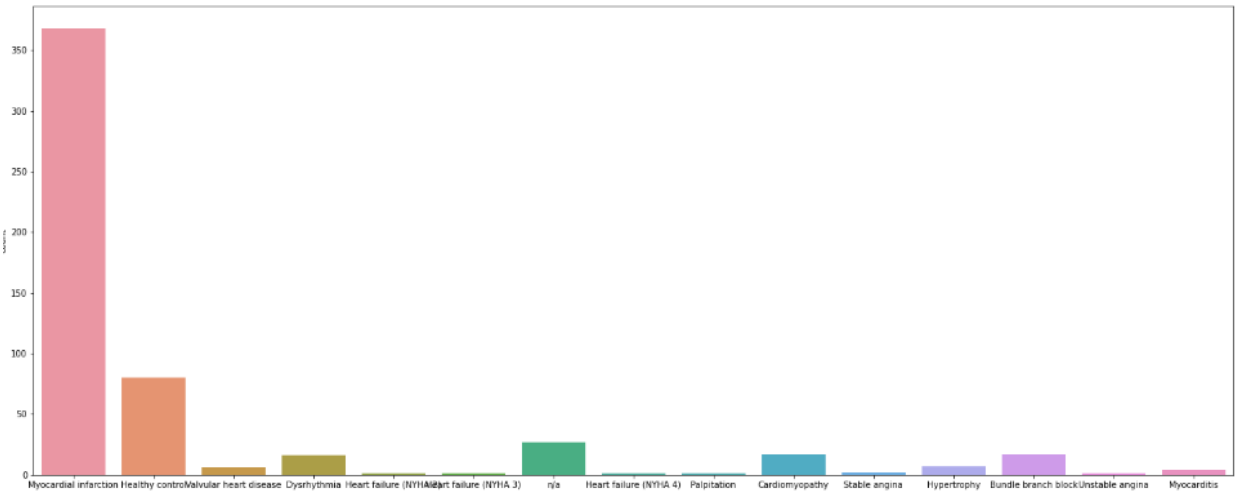
```
#data array
print(data[0])
```

```
[[-0.2445 -0.229 ]
 [-0.2425 -0.2335]
 [-0.2415 -0.2345]
 ...
 [ 0.152   0.2695]
 [ 0.136   0.256 ]
 [ 0.135   0.2585]]
```

```
# label
" ".join(data[1]['comments'][4].split()[3:])
```

```
'Myocardial infarction'
```

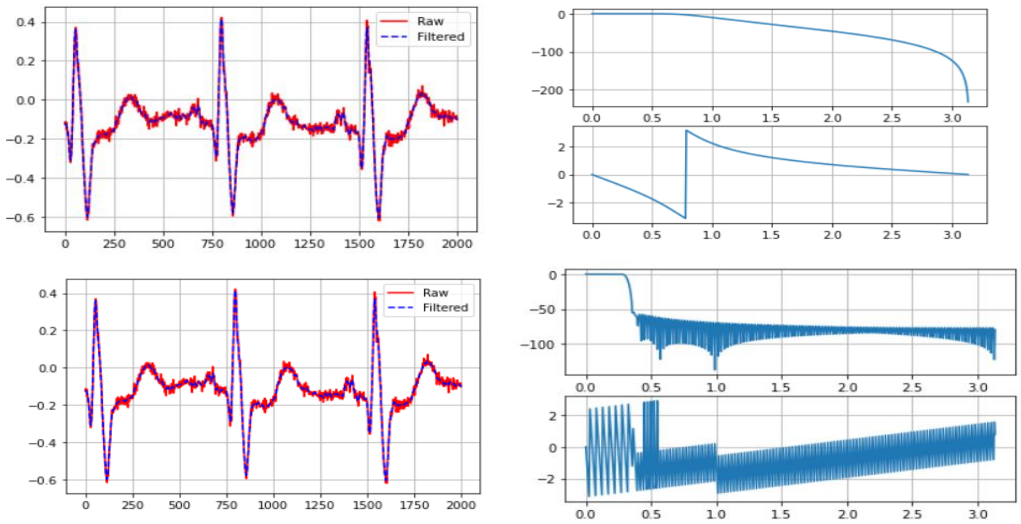
در ادامه با جمع آوری لیبل های مورد نظر آن ها را به صورت یک نمودار میله ای رسم می کنیم که نشان می دهد این دیتاست متعادل نیست و اکثر داده ها در دسته Myocardial infarction قرار دارند. بنابراین برای متعادل کردن دیتاست کلاس بندی را به صورت باینری در نظر گرفته و سایر کلاس ها را به صورت یک کلاس در نظر می گیریم.




در قسمت بعد داده های ناقص و بدون لیبل را از دیتاست حذف می کنیم.

```
# Removing data with n/a labels
for rec in records:
    data = wfdb.rdsamp(rec)
    label = " ".join(data[1]['comments'][4].split()[3:])
    if label == "n/a":
        records.remove(rec)
```

همچنین داده های این دیتاست دارای مقداری نویز هستند که می توان با استفاده از فیلتر IIR و FIR آن ها را تا حد مطلوبی حذف کنیم.




در آخر نیز با استفاده از دستور PCA ابعاد دیتاست را از ۱۵ به ۱۰ کاهش می دهیم.

```
# PCA
data = wfdb.rdsamp("drive/MyDrive/ml-dataset/patient001/s0010_re")[0]

pca = PCA(n_components=0.95)
pca_signal = pca.fit_transform(data)
print(f"Initial dimensions: {data.shape[1]} | Dimension after PCA: {pca_signal.shape[1]}")
```

```
Initial dimensions: 15 | Dimension after PCA: 4
```

```
def preprocess(x):
    pca = PCA(n_components=10)
    x_pca = pca.fit_transform(x)
    x_filtered = iir_lp(x_pca.T, 50, 1000)
    return x_filtered

data = wfdb.rdsamp("drive/MyDrive/ml-dataset/patient001/s0010_re")[0]
preprocess(data).shape
```

```
(10, 38400)
```

## آموزش شبکه

در این قسمت به دلیل آنکه حجم اطلاعات دیتاست بالاست برای آنکه هنگام آموزش شبکه حافظه دچار مشکل نشده و crash نکند از یک data generator به صورت زیر استفاده می کنیم که داده ها را به صورت batch استفاده می کند.

```
class DataGenerator(tf.keras.utils.Sequence):
    def __init__(self, data_dir, patients_list,
                 batch_size,
                 input_size=(10, 1000, 1),
                 shuffle=True):

        self.data_dir = data_dir
        self.patients_list = patients_list
        self.batch_size = batch_size
        self.input_size = input_size
        self.shuffle = shuffle

        self.n = len(self.patients_list)
        self.__get_recs()

    def __get_recs(self,):
        self.records = []
        for patient in self.patients_list:
            dir = Path(self.data_dir) / f"patient{str(patient).zfill(3)}"
            self.records.extend([dir / x[:-4] for x in glob(f"{str(dir)}/*.dat")])

    def on_epoch_end(self):
        if self.shuffle == True:
            np.random.shuffle(self.records)

    def __getitem__(self, index):
        rec = self.records[index]
        signals = preprocess(wfdb.rdsamp(rec)[0])
        if signals.shape[1] < self.batch_size*1000:
            np.pad(signals, ((0, 0),(0, self.batch_size*1000 - signals.shape[1])), 'constant')

        batched_signals = np.array([signals[:, i*1000:(i+1)*1000] for i in range(self.batch_size)])
        labels = " ".join(wfdb.rdsamp(rec)[1]['comments'][4].split()[3:])
        return (batched_signals.reshape(self.batch_size, *self.input_size),
                (np.array(labels) == 'Myocardial infarction').astype(int).repeat(self.batch_size))

    def __len__(self):
        return self.n
```

برای اموزش یک classifier از شبکه های عصبی و مدل های مختلف استفاده کردیم که در ادامه به توضیح هر یک میپردازیم.

لازم به ذکر است نحوه فیت کردن مدل و الگوریتم optimizer تمامی الگوریتم ها یکسان و به صورت زیر در نظر گرفته شده است:

```
model.compile(loss=tf.keras.losses.BinaryCrossentropy(), optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), metrics=[tf.keras.metrics.BinaryAccuracy(
        name='accuracy', dtype=None, threshold=0.5),tf.keras.metrics.Recall(name='Recall'),tf.keras.metrics.Precision(name='Precision'),])
history1 = model.fit(train_gen, batch_size=32, epochs=5,validation_data=val_gen)
```

همانطور که مشاهده میشود از مجموعه اعتبارسنجی نیز برای بهبود آموزش استفاده شده است که در ابتدا کل داده ها را به سه دسته آموزش، اعتبارسنجی و تست با نسبت ۷۰-۱۵-۱۵ تقسیم می کنیم.

```
# Splitting data into train, test, validation
NUM_SUBJECTS = 294
train_patients = np.random.choice(np.arange(1, NUM_SUBJECTS+1), int(0.7*NUM_SUBJECTS), replace=False)

val_patients = np.array([idx for idx in np.arange(1, NUM_SUBJECTS+1) if idx not in train_patients])
val_patients = np.random.choice(val_patients, int(0.15*NUM_SUBJECTS), replace=False)
test_patients = np.array([idx for idx in np.arange(1, NUM_SUBJECTS+1) if idx not in val_patients and idx not in train_patients])



# Generators
train_gen = DataGenerator("/content/drive/MyDrive/ml-dataset", train_patients, 32)
test_gen = DataGenerator("/content/drive/MyDrive/ml-dataset", test_patients, 32)
val_gen = DataGenerator("/content/drive/MyDrive/ml-dataset", val_patients, 32)
```

## الف) مدل ANN

```
ann_model = Sequential()
ann_model.add(Dense(50, activation='relu', input_shape=(10, 1000, 1)))
ann_model.add(Dense(50, activation='relu'))
ann_model.add(Dense(50, activation='relu'))
ann_model.add(Dense(50, activation='relu'))
ann_model.add(GlobalAveragePooling2D())
ann_model.add(Dense(1, activation='sigmoid'))
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_20 (Dense)            (None, 10, 1000, 50)      100

 dense_21 (Dense)            (None, 10, 1000, 50)      2550

 dense_22 (Dense)            (None, 10, 1000, 50)      2550

 dense_23 (Dense)            (None, 10, 1000, 50)      2550

 global_average_pooling2d_4   (None, 50)               0
 (GlobalAveragePooling2D)

 dense_24 (Dense)            (None, 1)                 51

=================================================================
Total params: 7,801
Trainable params: 7,801
Non-trainable params: 0
_________________________________________________________________
```

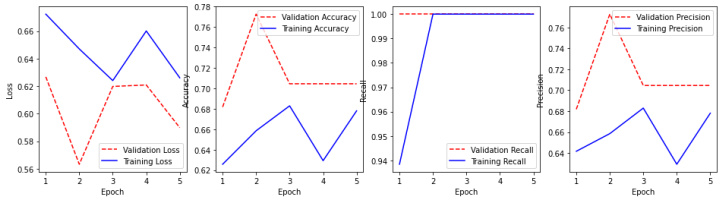
## نتایج ANN:

```
ann_model.compile(loss=tf.keras.losses.BinaryCrossentropy(), optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), metrics=[tf.keras.metrics.BinaryAccuracy(
        name='accuracy', dtype=None, threshold=0.5),tf.keras.metrics.Recall(name='Recall'),tf.keras.metrics.Precision(name='Precision'),])
history0 = ann_model.fit(train_gen, batch_size=32, epochs=5,validation_data=val_gen)
```

```
Epoch 1/5
205/205 [==============================] - 130s 628ms/step - loss: 0.6723 - accuracy: 0.6258 - Recall: 0.9385 - Precision: 0.6417 - val_loss: 0.6268 - val_accuracy: 0.6818 - val_Recall: 1.0000 - val_Precision: 0.6818
Epoch 2/5
205/205 [==============================] - 137s 670ms/step - loss: 0.6470 - accuracy: 0.6585 - Recall: 1.0000 - Precision: 0.6585 - val_loss: 0.5633 - val_accuracy: 0.7727 - val_Recall: 1.0000 - val_Precision: 0.7727
Epoch 3/5
205/205 [==============================] - 125s 611ms/step - loss: 0.6241 - accuracy: 0.6829 - Recall: 1.0000 - Precision: 0.6829 - val_loss: 0.6198 - val_accuracy: 0.7045 - val_Recall: 1.0000 - val_Precision: 0.7045
Epoch 4/5
205/205 [==============================] - 121s 590ms/step - loss: 0.6601 - accuracy: 0.6293 - Recall: 1.0000 - Precision: 0.6293 - val_loss: 0.6209 - val_accuracy: 0.7045 - val_Recall: 1.0000 - val_Precision: 0.7045
Epoch 5/5
205/205 [==============================] - 122s 594ms/step - loss: 0.6261 - accuracy: 0.6780 - Recall: 1.0000 - Precision: 0.6780 - val_loss: 0.5898 - val_accuracy: 0.7045 - val_Recall: 1.0000 - val_Precision: 0.7045
```

```
evaluate(ann_model)
plot(history0)
```

```
45/45 [==============================] - 23s 508ms/step - loss: 0.5786 - accuracy: 0.7111 - Recall: 1.0000 - Precision: 0.7111
Test loss: 0.5785666704177856
Test accuracy: 0.7111111283302307
```



## ب) مدل CNN

```
keras.layers.Conv2D(256, 3, activation="relu", input_shape=(10, 1000, 1)),
keras.layers.BatchNormalization(),
keras.layers.MaxPool2D(pool_size=3, padding="same"),
keras.layers.Dropout(0.5),
keras.layers.Flatten(),
keras.layers.Dense(15, activation='relu', input_shape=(10, 1000, 1), kernel_regularizer=regularizers.l2(0.000001)),
keras.layers.Dense(10, activation='relu', kernel_regularizer=regularizers.l2(0.000001)),
keras.layers.Dense(5, activation='relu'),
(keras.layers.Dense(1, activation='sigmoid'))
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_126 (Conv2D)         (None, 8, 998, 256)       2560

 batch_normalization_126 (Ba  (None, 8, 998, 256)      1024
 tchNormalization)

 max_pooling2d_15 (MaxPoolin  (None, 3, 333, 256)      0
 g2D)

 dropout_5 (Dropout)         (None, 3, 333, 256)       0

 flatten_4 (Flatten)         (None, 255744)            0

 dense_25 (Dense)            (None, 15)                3836175

 dense_26 (Dense)            (None, 10)                160

 dense_27 (Dense)            (None, 5)                 55

 dense_28 (Dense)            (None, 1)                 6

=================================================================
Total params: 3,839,980
Trainable params: 3,839,468
Non-trainable params: 512
_________________________________________________________________
```

لازم به ذکر است تنظیم کننده ها (kernel regularizer) به شما این امکان را می دهند که در طول بهینه سازی، جریمه هایی را بر روی پارامترهای لایه یا فعالیت لایه اعمال کنید. این جریمه ها در تابع ضرری که شبکه بهینه می کند، خلاصه می شود.

A regularizer that applies a L2 regularization penalty.

The L2 regularization penalty is computed as: `loss = l2 * reduce_sum(square(x))`

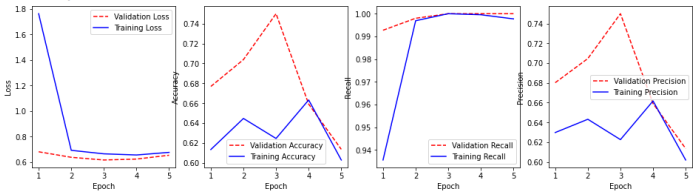
نتایج CNN:

```
model = get_base_model()
model.compile(loss=tf.keras.losses.BinaryCrossentropy(), optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), metrics=[tf.keras.metrics.BinaryAccuracy(
        name='accuracy', dtype=None, threshold=0.5),tf.keras.metrics.Recall(name='Recall'),tf.keras.metrics.Precision(name='Precision'),])
history1 = model.fit(train_gen, batch_size=32, epochs=5,validation_data=val_gen)
```

```
Epoch 1/5
205/205 [==============================] - 121s 584ms/step - loss: 1.7627 - accuracy: 0.6134 - Recall: 0.9356 - Precision: 0.6298 - val_loss: 0.6822 - val_accuracy: 0.6768 - val_Recall: 0.9927 - val_Precision: 0.6802
Epoch 2/5
205/205 [==============================] - 122s 595ms/step - loss: 0.6935 - accuracy: 0.6447 - Recall: 0.9969 - Precision: 0.6432 - val_loss: 0.6389 - val_accuracy: 0.7038 - val_Recall: 0.9980 - val_Precision: 0.7046
Epoch 3/5
205/205 [==============================] - 119s 580ms/step - loss: 0.6661 - accuracy: 0.6245 - Recall: 1.0000 - Precision: 0.6226 - val_loss: 0.6177 - val_accuracy: 0.7500 - val_Recall: 1.0000 - val_Precision: 0.7500
Epoch 4/5
205/205 [==============================] - 122s 598ms/step - loss: 0.6571 - accuracy: 0.6633 - Recall: 0.9995 - Precision: 0.6618 - val_loss: 0.6251 - val_accuracy: 0.6591 - val_Recall: 1.0000 - val_Precision: 0.6591
Epoch 5/5
205/205 [==============================] - 120s 586ms/step - loss: 0.6775 - accuracy: 0.6029 - Recall: 0.9977 - Precision: 0.6020 - val_loss: 0.6551 - val_accuracy: 0.6136 - val_Recall: 1.0000 - val_Precision: 0.6136
```

```
evaluate(model)
plot(history1)
```

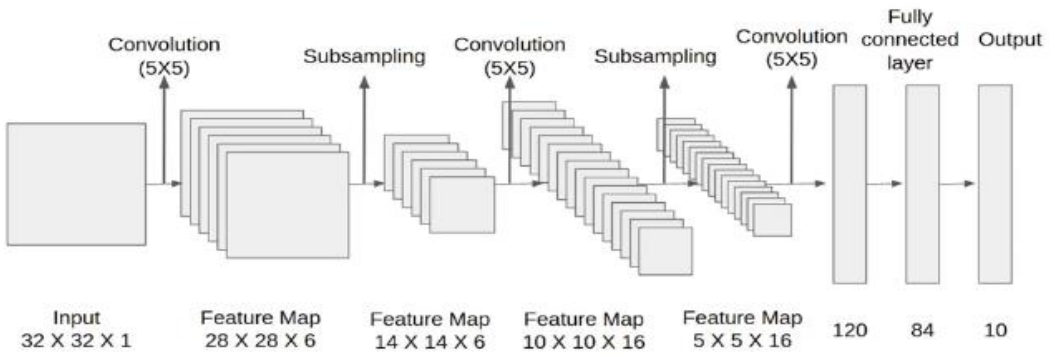
```
45/45 [==============================] - 21s 469ms/step - loss: 0.5549 - accuracy: 0.7563 - Recall: 1.0000 - Precision: 0.7561
Test loss: 0.5549065470695496
Test accuracy: 0.7562500238418579
```




## ج) مدل LeNet

(LeNet) یک ساختار شبکه عصبی کانولوشن است که توسط Yann LeCun و همکاران در سال ۱۹۸۹ پیشنهاد شده‌است. به‌طور کلی، له نت (LeNet) به lenet-۵ اشاره دارد و یک شبکه عصبی کانولوشن ساده است له نت (LeNet) به عنوان نماینده شبکه عصبی کانولوشن اولیه، دارای واحدهای اساسی شبکه عصبی کانولوشن است، مانند لایه کانولوشن، لایه استخر و لایه اتصال کامل، و پایه ای برای توسعه آینده شبکه عصبی کانولوشن. Lenet-۵ از هفت لایه تشکیل شده‌است که علاوه بر ورودی، هر لایه دیگر می‌تواند پارامترها را آموزش دهد.

هر لایه کانولوشن شامل سه قسمت است: توابع کانولوشن، جمع کردن و فعال سازی غیرخطی. -استفاده از کانولوشن برای استخراج ویژگی‌های فضایی (در ابتدا Convolution را زمینه‌های پذیرا می‌نامیدند). -نمونه برداری از لایه استخر متوسط. -استفاده از MLP به عنوان آخرین طبقه‌بندی. -اتصال پراکنده بین لایه‌ها برای کاهش پیچیدگی محاسبات. معماری این شبکه به صورت زیر می باشد:



مدل:

```
lenet_5_model=Sequential()

lenet_5_model.add(Conv2D(filters=6, kernel_size=3, padding='same', input_shape=(10, 1000, 1)))
lenet_5_model.add(BatchNormalization())
lenet_5_model.add(Activation('relu'))
lenet_5_model.add(MaxPool2D(pool_size=3, strides=2, padding='same'))

lenet_5_model.add(Conv2D(filters=6, strides=1, kernel_size=5))
lenet_5_model.add(BatchNormalization())
lenet_5_model.add(Activation('relu'))
lenet_5_model.add(MaxPool2D(pool_size=2, strides=2, padding='same'))

lenet_5_model.add(Conv2D(filters=6, kernel_size=3, padding='same'))
lenet_5_model.add(BatchNormalization())
lenet_5_model.add(Activation('relu'))
lenet_5_model.add(MaxPool2D(pool_size=3, strides=2, padding='same'))

lenet_5_model.add(Conv2D(filters=6, kernel_size=3, padding='same'))
lenet_5_model.add(BatchNormalization())
lenet_5_model.add(Activation('relu'))

lenet_5_model.add(GlobalAveragePooling2D())


lenet_5_model.add(tf.keras.layers.Flatten())
lenet_5_model.add(Dense(64, activation='relu'))

lenet_5_model.add(Dense(32, activation='relu'))

lenet_5_model.add(Dense(1, activation = 'sigmoid'))
```

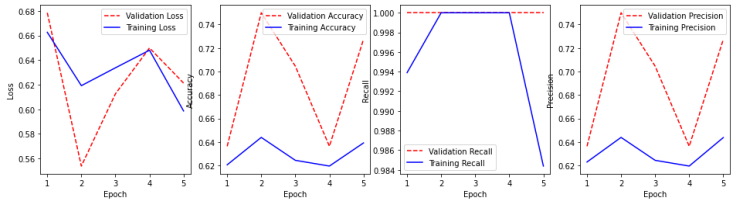
```
=================================================================
Total params: 4,283
Trainable params: 4,235
Non-trainable params: 48
_________________________________________________________________
```

نتایج:

```
Epoch 1/5
205/205 [==============================] - 124s 595ms/step - loss: 0.6626 - accuracy: 0.6206 - Recall: 0.9939 - Precision: 0.6230 - val_loss: 0.6784 - val_accuracy: 0.6364 - val_Recall: 1.0000 - val_Precision: 0.6364
Epoch 2/5
205/205 [==============================] - 124s 604ms/step - loss: 0.6192 - accuracy: 0.6439 - Recall: 1.0000 - Precision: 0.6439 - val_loss: 0.5538 - val_accuracy: 0.7500 - val_Recall: 1.0000 - val_Precision: 0.7500
Epoch 3/5
205/205 [==============================] - 123s 601ms/step - loss: 0.6337 - accuracy: 0.6244 - Recall: 1.0000 - Precision: 0.6244 - val_loss: 0.6126 - val_accuracy: 0.7045 - val_Recall: 1.0000 - val_Precision: 0.7045
Epoch 4/5
205/205 [==============================] - 123s 601ms/step - loss: 0.6480 - accuracy: 0.6195 - Recall: 1.0000 - Precision: 0.6195 - val_loss: 0.6498 - val_accuracy: 0.6364 - val_Recall: 1.0000 - val_Precision: 0.6364
Epoch 5/5
205/205 [==============================] - 124s 604ms/step - loss: 0.5986 - accuracy: 0.6392 - Recall: 0.9844 - Precision: 0.6438 - val_loss: 0.6213 - val_accuracy: 0.7273 - val_Recall: 1.0000 - val_Precision: 0.7273
```

```
evaluate(lenet_5_model)
plot(history3)
```

```
45/45 [==============================] - 22s 500ms/step - loss: 0.1684 - accuracy: 1.0000 - Recall: 1.0000 - Precision: 1.0000
Test loss: 0.16842931509017944
Test accuracy: 1.0
```



## د) مدل ResNet

طبق مقاله زیر پیش رفتیم و مدل Resnet۱۸ را برای classification استفاده کردیم.

*Research Article*

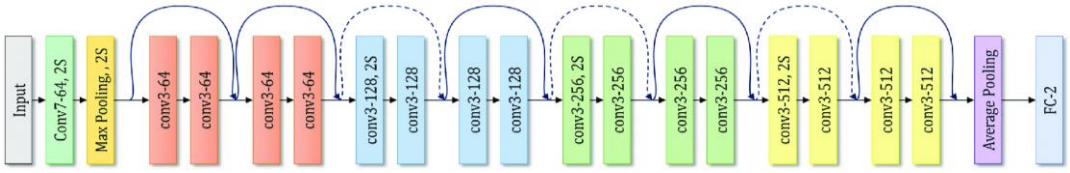
**ECG Heartbeat Classification Based on an Improved ResNet-18 Model**

**Enbiao Jing,[1] Haiyang Zhang,[2] ZhiGang Li,[1] Yazhi Liu,[1] Zhanlin Ji,[1,3] and Ivan Ganchev[3,4,5]**

در ادامه به توضیح بیشتر در رابطه با ساختار رزنت میپردازیم.

شبکه‌ی رزنت (ResNet) یک شبکه‌ی عصبی عمیق است که در سال ۲۰۱۵ توانست رتبه‌ی اول را در رقابت ILSVRC کسب کند. قبل از معرفی این شبکه استفاده از شبکه‌های عصبی با لایه‌های زیاد دچار مشکل بود. با افزایش تعداد لایه‌ها شبکه دچار مشکل محوشدگی گرادیان (Vanishing Gradient) می‌شد؛ شبکه‌ی رزنت توانست با ارائه‌ی راه‌حلی این مشکل را تا حد زیادی برطرف کند؛ به‌همین دلیل، این شبکه قادر است حتی تا ۱۵۲ لایه هم داشته باشد. اتصالات میانبر (Skip Connections) یا اتصالات اضافی (Residual Connections) راه‌حلی بود که شبکه رزنت (ResNet) برای حل مشکل شبکه‌های عمیق ارائه کرد. برای حل این مشکل در این شبکه از بلاک اضافی (Residual Block) استفاده شده است. همان‌طور که مشخص است، فرق این شبکه با شبکه‌های معمولی این است که یک اتصال میان‌بر دارد که از یک یا چند لایه عبور می‌کند و آن‌ها را در نظر نمی‌گیرد؛ درواقع به‌نوعی میان‌بر می‌زند و یک لایه را به لایه‌ی دورتر متصل می‌کند.

معماری کلی رزنت ۱۸ به صورت شکل زیر می باشد:




```
================================================================================================
Total params: 23,583,489
Trainable params: 23,530,369
Non-trainable params: 53,120
------------------------------------------------------------------------------------------------
```

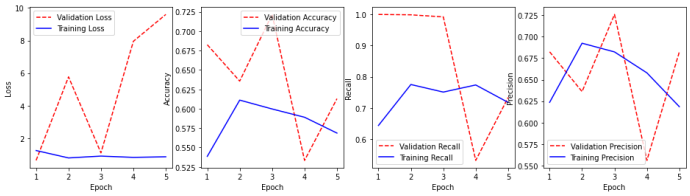
نتایج ResNet:

```
history2 = resNet18_model.fit(train_gen, batch_size=32, epochs=5,validation_data=val_gen)

Epoch 1/5
205/205 [==============================] - 147s 670ms/step - loss: 1.2436 - accuracy: 0.5386 - Recall: 0.6435 - Precision: 0.6237 - val_loss: 0.6381 - val_accuracy: 0.6825 - val_Recall: 1.0000 - val_Precision: 0.6823
Epoch 2/5
205/205 [==============================] - 134s 651ms/step - loss: 0.7894 - accuracy: 0.6111 - Recall: 0.7750 - Precision: 0.6923 - val_loss: 5.7681 - val_accuracy: 0.6357 - val_Recall: 0.9989 - val_Precision: 0.6361
Epoch 3/5
205/205 [==============================] - 132s 645ms/step - loss: 0.9034 - accuracy: 0.5997 - Recall: 0.7507 - Precision: 0.6822 - val_loss: 1.0976 - val_accuracy: 0.7216 - val_Recall: 0.9922 - val_Precision: 0.7257
Epoch 4/5
205/205 [==============================] - 133s 646ms/step - loss: 0.8221 - accuracy: 0.5890 - Recall: 0.7736 - Precision: 0.6579 - val_loss: 7.9506 - val_accuracy: 0.5334 - val_Recall: 0.5312 - val_Precision: 0.5562
Epoch 5/5
205/205 [==============================] - 132s 642ms/step - loss: 0.8593 - accuracy: 0.5686 - Recall: 0.7175 - Precision: 0.6186 - val_loss: 9.5986 - val_accuracy: 0.6129 - val_Recall: 0.7333 - val_Precision: 0.6822
```

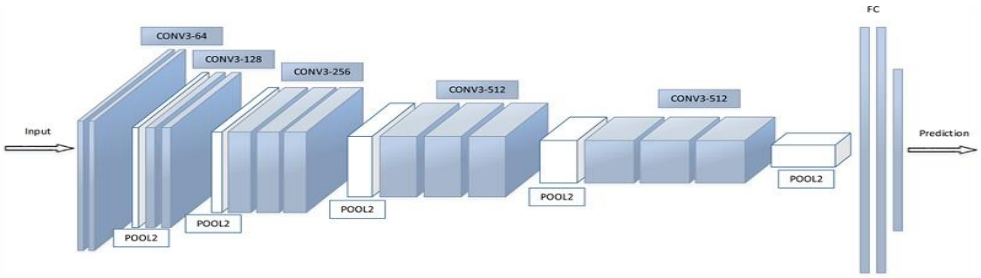
```
evaluate(resNet18_model)
plot(history2)

45/45 [==============================] - 23s 515ms/step - loss: 32.0698 - accuracy: 0.5076 - Recall: 0.5756 - Precision: 0.6647
Test loss: 32.069786071777344
Test accuracy: 0.5076388716697693
```




## ه) مدل VGG

شبکه VGG در دو معماری مختلف با عنوان‌های VGG ۱۶ و VGG ۱۹ ارائه شده است. ابتدا شبکه VGG ۱۶ پیشنهاد شد و بعدها با تغییراتی جزئی در شبکه VGG ۱۶، شبکه VGG ۱۹ مطرح گردید. شبکه VGG ۱۶ همان‌طور که در شکل زیر نشان داده شده، شامل ۱۶ لایه کانولوشنی یا ۱۶ لایه پارامتری است. شبکه VGG ۱۶، شامل دو لایه کانولوشنی با ۶۴ فیلتر ۳×۳ هست که پشت سر هم قرار گرفته‌اند. سپس، یک لایه ماکس‌پولینگ ۲×۲ با پرش (Stride) به اندازه ۲ قرار گرفته است. این لایه ماکس‌پولینگ علاوه بر نمونه‌برداری، وظیفه کاهش بعد ویژگی‌ها به نصف را هم دارد. در ادامه، دو لایه کانولوشنی دیگر با ۱۲۸ فیلتر ۳×۳ و یک لایه ماکس‌پولینگ ۲×۲ و پرش ۲ قرار گرفته‌اند. به‌طور مشابه، سه لایه کانولوشنی با ۲۵۶ فیلتر ۳×۳ و یک لایه ماکس‌پولینگ ۲×۲ با پرش ۲ قرار گرفته‌اند. ۳ لایه کانولوشنی با ۵۱۲ فیلتر ۳×۳ و یک لایه ماکس‌پولینگ ادامه این شبکه هست که البته دو بار تکرار می‌شود. درنهایت، ویژگی‌ها تبدیل به یک بردار ویژگی می‌شوند تا در اختیار لایه‌های نورونی یا تمام‌اتصال یا Fully Connected قرار گیرند. دو لایه نورونی به ابعاد ۴۰۹۶ پشت سر هم قرار گرفته‌اند. درنهایت، یک لایه نورونی به ابعاد ۱۰۰۰ که متناظر با تعداد کلاس‌های کاربرد ما هست، در نظر گرفته شده است. باتوجه به اینکه پایگاه داده ImageNet شامل ۱۰۰۰ کلاس هست، در اینجا هم لایه خروجی شامل ۱۰۰۰ نورون است. در تمامی لایه‌های کانولوشنی و لایه‌های نورونی از تابع فعال‌ساز یا Activation Function بنام RELU استفاده شده است.




```
=================================================================
Total params: 10,700,481
Trainable params: 10,692,033
Non-trainable params: 8,448
_________________________________________________________________
```

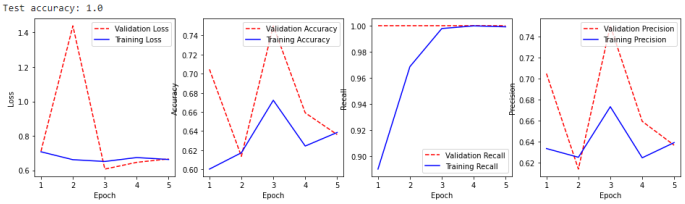
نتایج:

```
vgg_16_model.compile(loss=tf.keras.losses.BinaryCrossentropy(), optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), metrics=[tf.keras.metrics.BinaryAccuracy(
      name='accuracy', dtype=None, threshold=0.5),tf.keras.metrics.Recall(name='Recall'),tf.keras.metrics.Precision(name='Precision'),])
history4 = vgg_16_model.fit(train_gen, batch_size=32, epochs=5,validation_data=val_gen)
```

```
Epoch 1/5
205/205 [==============================] - 133s 627ms/step - loss: 0.7076 - accuracy: 0.6003 - Recall: 0.8903 - Precision: 0.6332 - val_loss: 0.7110 - val_accuracy: 0.7045 - val_Recall: 1.0000 - val_Precision: 0.7045
Epoch 2/5
205/205 [==============================] - 128s 625ms/step - loss: 0.6617 - accuracy: 0.6174 - Recall: 0.9688 - Precision: 0.6249 - val_loss: 1.4388 - val_accuracy: 0.6136 - val_Recall: 1.0000 - val_Precision: 0.6136
Epoch 3/5
205/205 [==============================] - 128s 622ms/step - loss: 0.6514 - accuracy: 0.6723 - Recall: 0.9980 - Precision: 0.6730 - val_loss: 0.6071 - val_accuracy: 0.7500 - val_Recall: 1.0000 - val_Precision: 0.7500
Epoch 4/5
205/205 [==============================] - 128s 622ms/step - loss: 0.6742 - accuracy: 0.6244 - Recall: 1.0000 - Precision: 0.6244 - val_loss: 0.6463 - val_accuracy: 0.6591 - val_Recall: 1.0000 - val_Precision: 0.6591
Epoch 5/5
205/205 [==============================] - 129s 628ms/step - loss: 0.6630 - accuracy: 0.6386 - Recall: 0.9993 - Precision: 0.6389 - val_loss: 0.6663 - val_accuracy: 0.6364 - val_Recall: 1.0000 - val_Precision: 0.6364
```

```
evaluate(vgg_16_model)
plot(history4)
```

```
59/59 [==============================] - 29s 495ms/step - loss: 0.2926 - accuracy: 1.0000 - Recall: 1.0000 - Precision: 1.0000
Test loss: 0.29261213541030884
Test accuracy: 1.0
```




## مقایسه مدل ها

| | ANN | CNN | LeNet | ResNet | VGG |
|---|---|---|---|---|---|
| Best train acc | ۰,۶۸۲۹ | ۰,۶۴۴۷ | ۰,۶۴۳۹ | ۰,۶۱۱۱ | ۰,۶۷۲۳ |
| Best valid acc | ۰,۷۷۲۷ | ۰,۷۵۰۰ | ۰,۷۵۰۰ | ۰.۷۲۱۶ | ۰,۷۵۰۰ |
| Test acc | ۰,۷۱۱۱ | ۰,۷۵۶۲ | ۱,۰ | ۰,۵۰۷ | ۱,۰ |
| Test loss | ۰,۵۷۸۵ | ۰,۵۵۴۹ | ۰,۱۶۸۴ | ۳۲,۰۶ | ۰,۲۹۲۶ |

جدول بالا نشان می دهد که مدل ResNet به دلیل تعداد بالای پارامتر هایش نتوانسته صحت خوبی بر دادگان تست داشته باشد و سایر مدل ها تا حدودی مشابه بوده و در کل مدل LeNet عملکرد بهتری داشته است.

## استفاده از cross validation

برای پیاده سازی هر دو بخش الف و ب از دو تابع زیر استفاده میکنیم در تابع اول مدل و fold های مختلف که از روی داده های آموزشی ساختیم و داده های آموزشی را به عنوان ورودی میدهیم. Fold ها به کمک تابع fold در کتابخانه scikit learn ساخته میشوند این تابع شاخص‌های آموزش/تست را برای تقسیم داده‌ها در مجموعه‌های آموزش/تست فراهم می‌کند. مجموعه داده را به k تاهای متوالی تقسیم میکند. سپس هر فولد یک بار به عنوان اعتبار سنجی استفاده می شود در حالی که k - ۱ فولد باقی مانده مجموعه آموزشی را تشکیل می دهد. این تابع در یک حلقه هر بار یک فولد را به عنوان داده تست میگیرد و با بقیه فولد ها مدل را آموزش میدهد و نتایج حاصل از پیش بینی بر روی داده های تست را در آرایه predicted_class ذخیره میکند. در آخر نیز دقت را با توجه به برچسب های اولیه برای این داده محاسبه و ذخیره میکند. در نهایت این تابع کلاس های صحیح و پیش بینی شده و دقت میانگین را برمیگرداند.

```
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix, accuracy_score
import copy

def cross_val_predict(model, kfold : KFold, X : np.array, y : np.array):

    model_ = copy.deepcopy(model)

    actual_classes = np.empty([0], dtype=int)
    predicted_classes = np.empty([0], dtype=int)
    accuracy = np.empty([0], dtype=int)

    for train_ndx, test_ndx in kfold.split(X):
        # Extracts the rows from the data for the training and testing
        train_X, train_y, test_X, test_y = X[train_ndx], y[train_ndx], X[test_ndx], y[test_ndx]
        # Appends the actual target classifications to actual_classes
        actual_classes = np.append(actual_classes, test_y)
        # Fits the machine learning model using the training data extracted from the current fold
        history = model_.fit(train_X, train_y, batch_size=32, epochs=5)
        # Uses the fitted model to predict the target classifications for the test data in the current fold
        predicted_classes = np.append(predicted_classes, model_.predict(test_X))
        accuracy = np.append(accuracy, accuracy_score(actual_classes, predicted_classes))

    return actual_classes, predicted_classes, accuracy.mean(), model_, history
```

در این قسمت دیتاست را به دو دسته آموزش و تست با نسبت ۸۰-۲۰ درصد تقسیم می کنیم و داده آموزش را در تابع بالا استفاده می کنیم.

```
# Splitting data into train, test, validation
NUM_SUBJECTS = 294
train_patients = np.random.choice(np.arange(1, NUM_SUBJECTS+1), int(0.8*NUM_SUBJECTS), replace=False)

test_patients = np.array([idx for idx in np.arange(1, NUM_SUBJECTS+1) if idx not in train_patients])



# Generators
train_gen = DataGenerator("/content/drive/MyDrive/ml-dataset", train_patients, 32)
test_gen = DataGenerator("/content/drive/MyDrive/ml-dataset", test_patients, 32)
```