**NUS-ISS**
*Pattern Recognition using*
*Machine Learning System*

# Module 8 - Case studies on using recurrent neural networks for machine learning systems

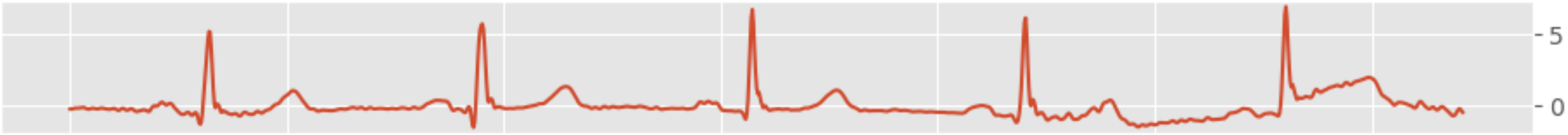
by Dr. Tan Jen Hong

# ECG Signals

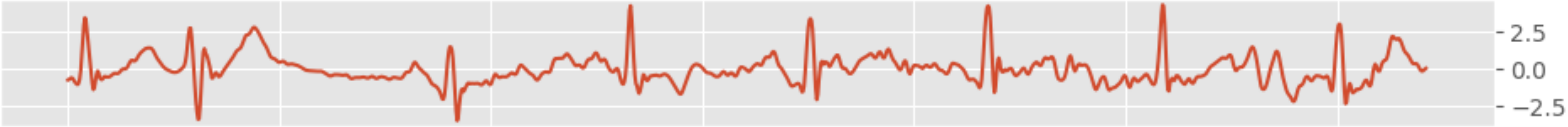
Normal and Coronary Artery Disease

- Lead II ECG from open source PhysioNet database
- Sampling frequency: 257 Hz
- 5 seconds long, consisting of 1285 data points

Normal



CAD

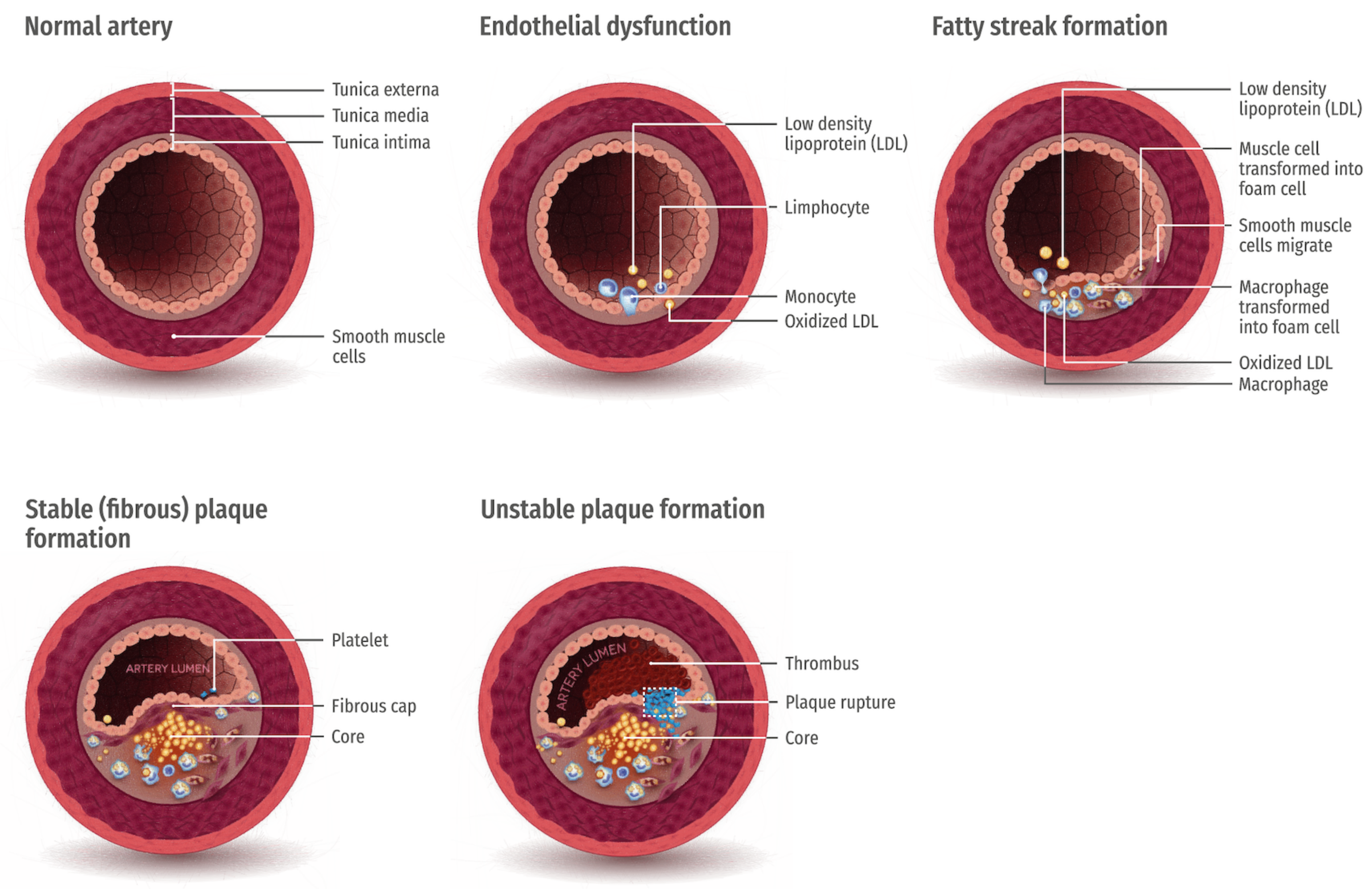


Source: https://www.sciencedirect.com/science/article/pii/S0010482517304201

# ECG Signals

## Normal and Coronary Artery Disease

- The pathology of coronary artery disease



Normal artery

Endothelial dysfunction

Fatty streak formation

Low density lipoprotein (LDL)
Muscle cell transformed into foam cell
Smooth muscle cells migrate
Macrophage transformed into foam cell
Oxidized LDL
Macrophage

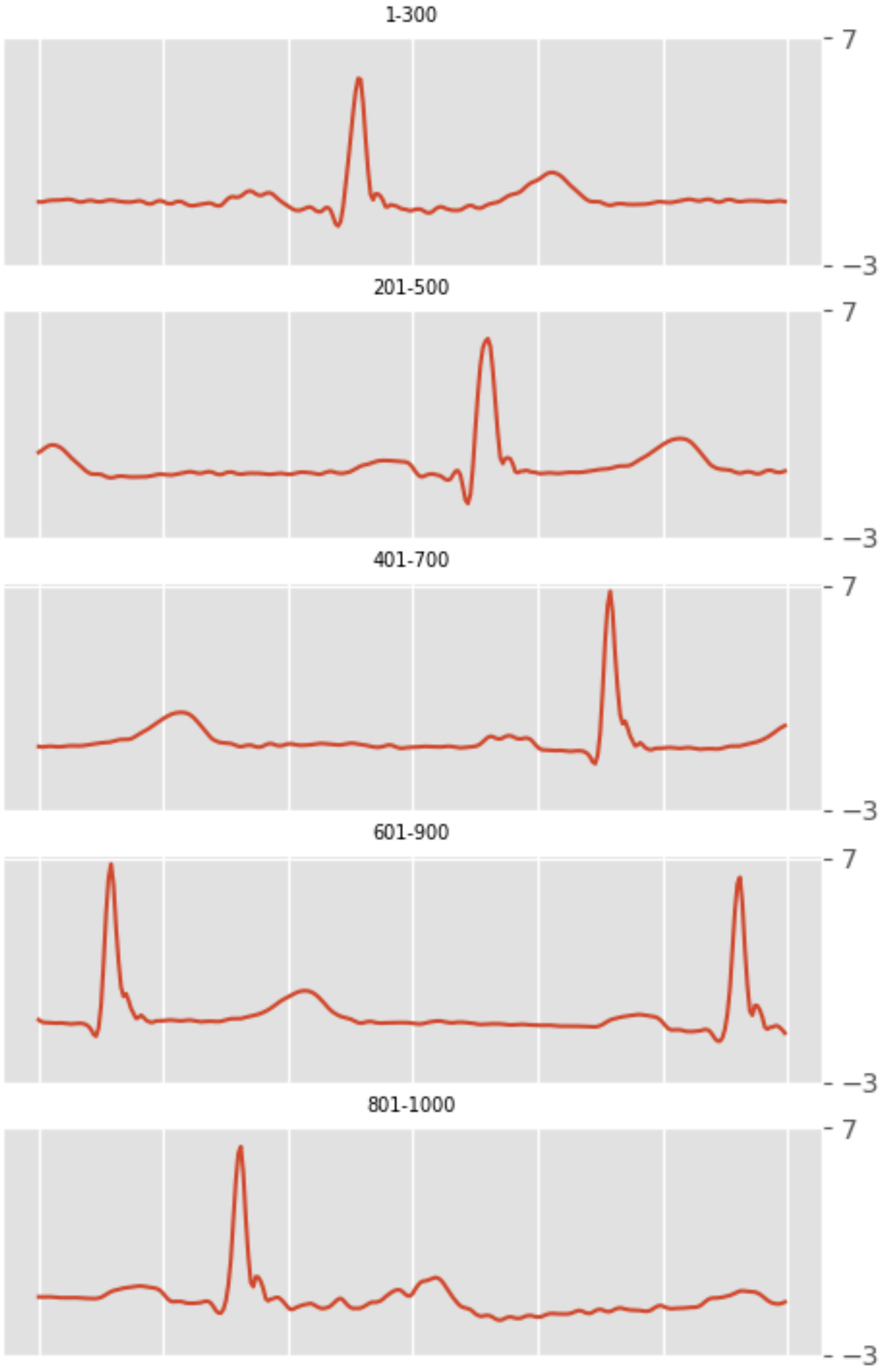
Stable (fibrous) plaque formation

Unstable plaque formation

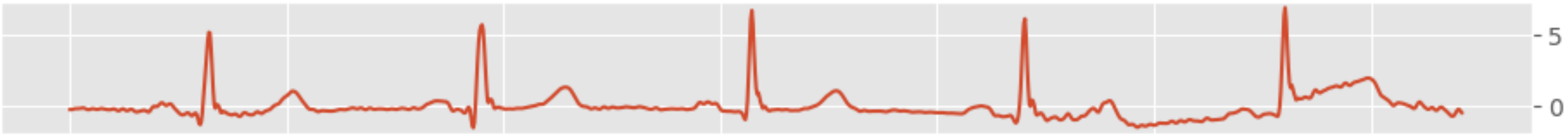
Source: https://www.sciencedirect.com/science/article/abs/pii/S0950705117302769

# Normal ECG

in segments

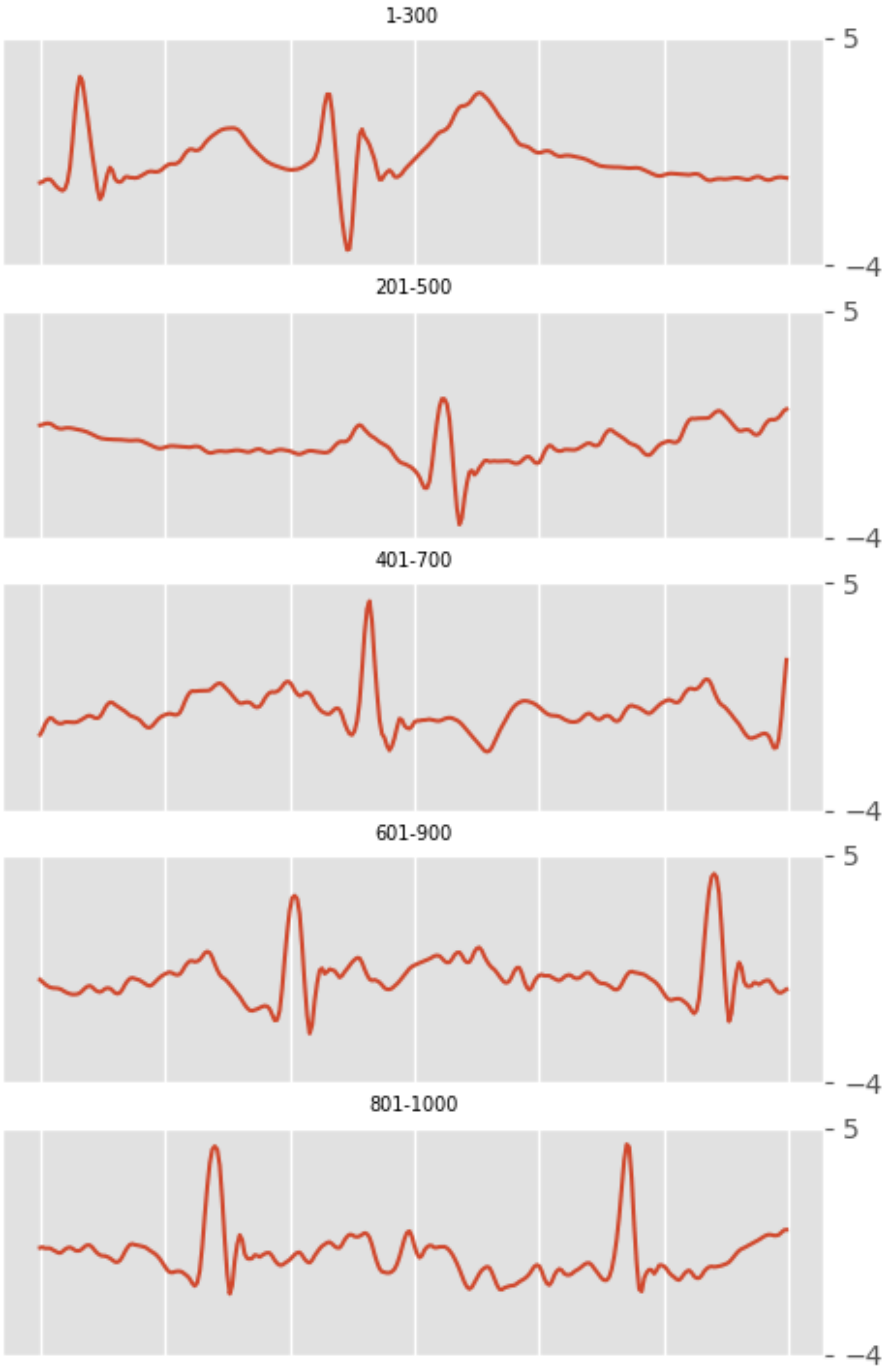


length: 300
distance: 200

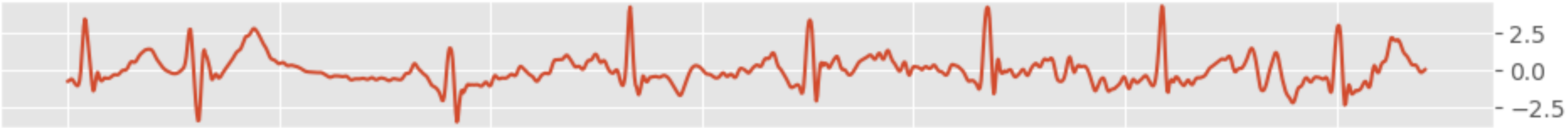


# CAD ECG

## in segments



length: 300
distance: 200



# ECG

## 1. Import libraries

- Import all the things that we are going to use in this problem
- h5py is required to load ECG data

```
> import h5py
> import numpy as np
> import sklearn.metrics as metrics
> import matplotlib.pyplot as plt

> from tensorflow.keras.callbacks import ModelCheckpoint,CSVLogger
> from tensorflow.keras.models import Model
> from tensorflow.keras.layers import Dense
> from tensorflow.keras.layers import Input
> from tensorflow.keras.layers import LSTM
```

# HDF5

Something about h5py ...

- h5py is the pythonic interface to the HDF5 binary data format
- HDF5: Hierarchical Data Format, designed to store and organize large amounts of data
- File extension: .h5, .hdf5
- Can think HDF as a file system within a file, let you organize data hierarchically
- Keras stores model in this format

# ECG

## 2. Matplotlib setup

- Use 'ggplot' style to plot our training and testing result

- The setup uses 'ggplot' style for plot

- Also, for y axis, the labels and ticks put on right rather than left

```
> plt.style.use('ggplot')
> plt.rcParams['ytick.right']     = True
> plt.rcParams['ytick.labelright']= True
> plt.rcParams['ytick.left']      = False
> plt.rcParams['ytick.labelleft'] = False
> plt.rcParams['font.family']     = 'Arial'
```

# ECG

## 3. Data preparation, part 1

- Load the hdf5 file and extract the data within

```
> f         = h5py.File('cad5sec.mat')
> X         = f["data"]
> Y         = f["classLabel"]
```

- if we type 'f' in ipython console, it will return

```
<HDF5 file "cad5sec.mat" (mode r+)>
```

- if we type 'X' and 'Y' respectively in ipython console, it will give

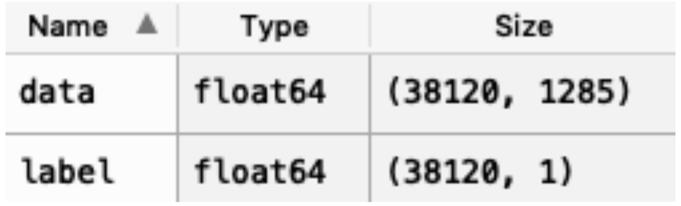
```
     <HDF5 dataset "data": shape (1285, 38120), type "<f8">
<HDF5 dataset "classLabel": shape (1, 38120), type "<f8">
```

# ECG

## 3. Data preparation, part 1

- Need to convert 'X' and 'Y' into numpy array and do a transpose on the data

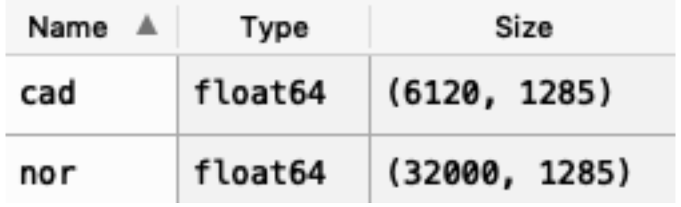
```
> data      = np.array(X)
> data      = np.transpose(data)
> label     = np.array(Y)
> label     = np.transpose(label)
```



| Name ▲ | Type | Size |
|---|---|---|
| data | float64 | (38120, 1285) |
| label | float64 | (38120, 1) |

- The first 32000 rows are 'normal' ECG, the rest are 'CAD' ECG

```
> nor       = data[0:32000]
> cad       = data[32000:38120]
```



| Name ▲ | Type | Size |
|---|---|---|
| cad | float64 | (6120, 1285) |
| nor | float64 | (32000, 1285) |

# ECG

## 3. Data preparation, part 2

- Create the function to turn each sample into fixed number of segments (controlled by length and distance

```
> def makeSteps(dat, length, dist):
    width             = dat.shape[1]
    numOfSteps        = int(np.floor((width-length)/dist)+1)

    segments          = np.zeros([dat.shape[0],numOfSteps,length],
                                 dtype=dat.dtype)

    for l in range(numOfSteps):
        segments[:,l,:]      = dat[:,(l*dist):(l*dist+length)]

    return segments
```

pre-allocate the array

# ECG

## 3. Data preparation, part 3

- Manually split the data into training and testing set

```
> print('Create dataset...')
> trNor      = nor[0:28800].copy()                    (28800, 1285)
> tsNor      = nor[28800:32000].copy()                (3200, 1285)
> trCad      = cad[0:5000].copy()                     (5000, 1285)
> tsCad      = cad[5000:6120].copy()                  (1120, 1285)
```

- Set the length and the distance, and convert each sample into segments

```
> length     = 36
> dist       = 24

> print('Finalizing all the data ....')
> trNorS     = makeSteps(trNor, length, dist)      (28800, 53, 36)
> tsNorS     = makeSteps(tsNor, length, dist)      (3200, 53, 36)
> trCadS     = makeSteps(trCad, length, dist)      (5000, 53, 36)
> tsCadS     = makeSteps(tsCad, length, dist)      (1120, 53, 36)
```

# ECG

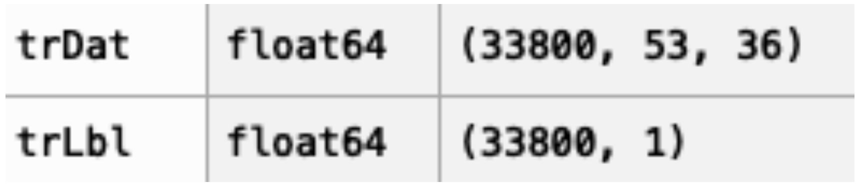
3. Data preparation, part 3

- Combining all together

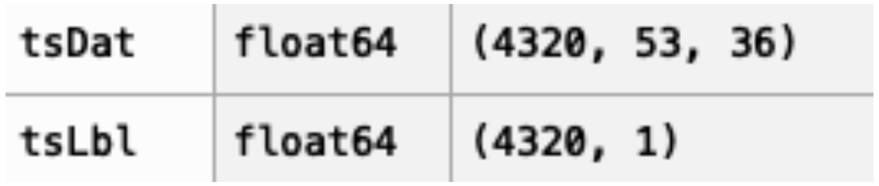
```
> trDat     = np.vstack([trNorS,trCadS])
> tsDat     = np.vstack([tsNorS,tsCadS])

> trLbl     = np.vstack([np.zeros([trNorS.shape[0],1]),
                          np.ones([trCadS.shape[0],1])])
> tsLbl     = np.vstack([np.zeros([tsNorS.shape[0],1]),
                          np.ones([tsCadS.shape[0],1])])
```

'Normal' set to 0

'Cad' set to 1



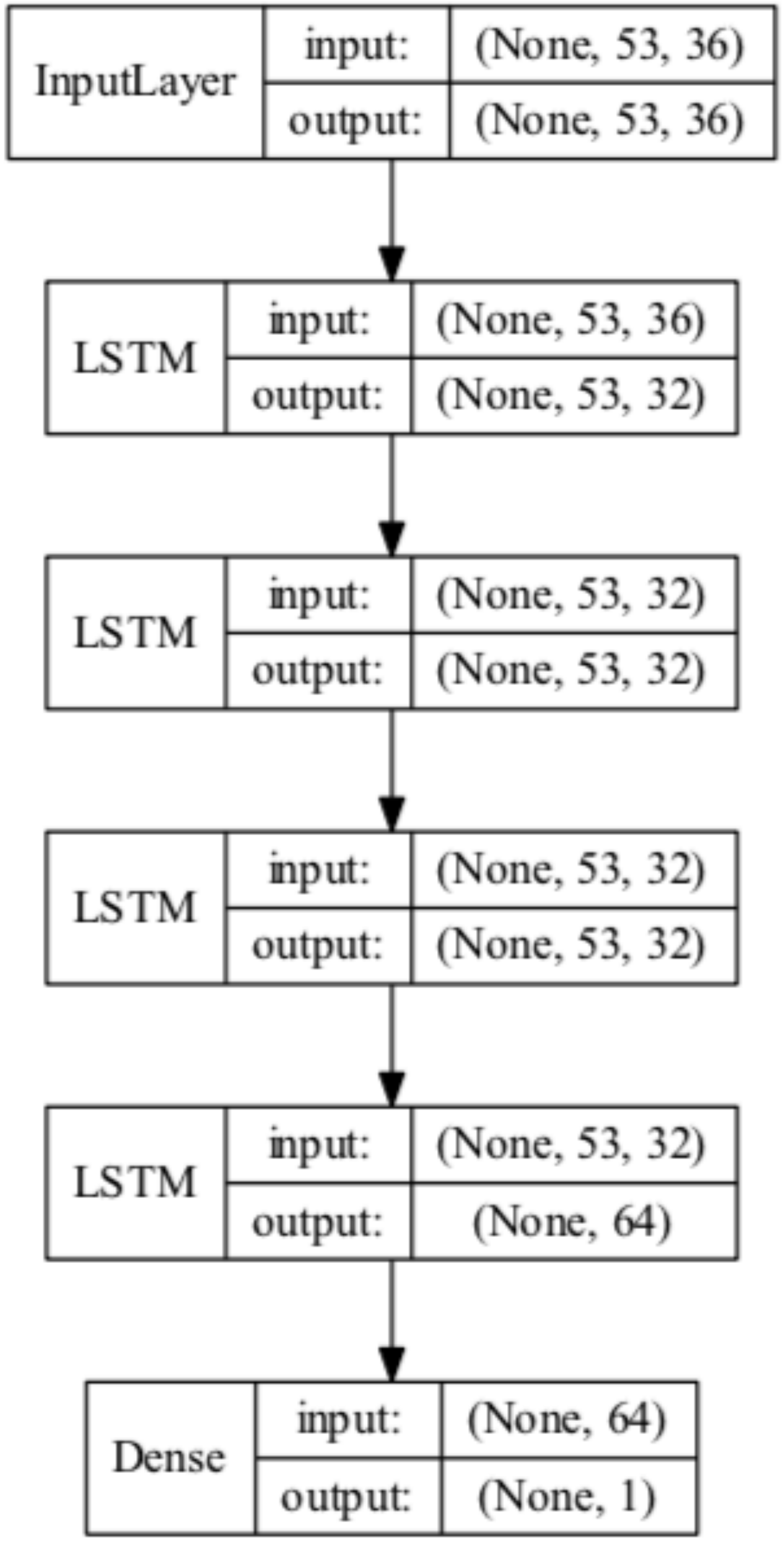
| | | |
|---|---|---|
| trDat | float64 | (33800, 53, 36) |
| trLbl | float64 | (33800, 1) |



| | | |
|---|---|---|
| tsDat | float64 | (4320, 53, 36) |
| tsLbl | float64 | (4320, 1) |

# The net

## 4. Define model



InputLayer | input: (None, 53, 36) | output: (None, 53, 36)

LSTM | input: (None, 53, 36) | output: (None, 53, 32)

LSTM | input: (None, 53, 32) | output: (None, 53, 32)

LSTM | input: (None, 53, 32) | output: (None, 53, 32)

LSTM | input: (None, 53, 32) | output: (None, 64)

Dense | input: (None, 64) | output: (None, 1)

## The net

### 4. Define model

```
> seed = 7
> np.random.seed(seed)
> modelname   =  'wks3_2_1'
> def createModel():
      inputs= Input(shape=(trDat.shape[1],length))
      y     = LSTM(32,
                   return_sequences=True,
                   dropout=0.25,
                   recurrent_dropout=0.25)(inputs)
      y     = LSTM(32,
                   return_sequences=True,
                   dropout=0.5,
                   recurrent_dropout=0.25)(y)
      y     = LSTM(32,
                   return_sequences=True,
                   dropout=0.25)(y)
      y     = LSTM(64, dropout=0.25)(y)
      y     = Dense(1, activation='sigmoid')(y)

      model = Model(inputs=inputs,outputs=y)
      model.compile(loss='binary_crossentropy',
                    optimizer='adam',
                    metrics=['accuracy'])
      return model
```

- Note 1: The dropout layer doesn't work for LSTM / RNN. If want to do dropout, set value to the 'dropout' argument

- Note 2: We can apply dropout on the recurrent part in LSTM (the $U_0$)

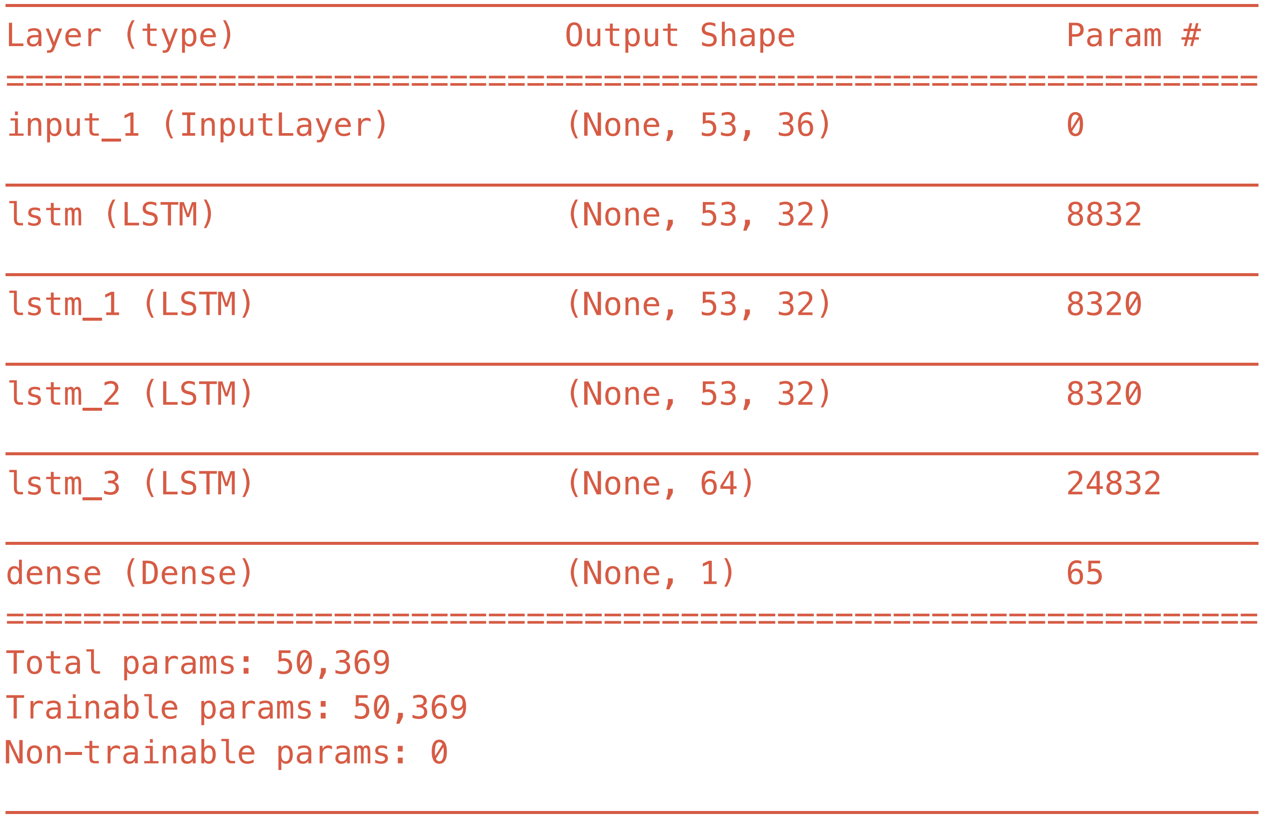
- Note 3: We use 'binary_crossentropy', not 'categorical_crossentropy', because we have only 1 output, and the value is either 1 or 0

# The net

## 4. Define model

- 'model' for training; 'modelGo' for final evaluation

```
> model       = createModel()
> modelGo     = createModel()

> model.summary()
```



```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 53, 36)            0
_________________________________________________________________
lstm (LSTM)                  (None, 53, 32)            8832
_________________________________________________________________
lstm_1 (LSTM)                (None, 53, 32)            8320
_________________________________________________________________
lstm_2 (LSTM)                (None, 53, 32)            8320
_________________________________________________________________
lstm_3 (LSTM)                (None, 64)                24832
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 50,369
Trainable params: 50,369
Non-trainable params: 0
_________________________________________________________________
```

# The net

## 4. Define model

- Create checkpoints to save model during training and save training data into csv
- ‘monitor’ can be ‘val_acc’ or ‘val_loss’
- When set to ‘val_acc’, ‘mode’ must be ‘max’; when set to ‘val_loss’, ‘mode’ must be ‘min’

```
> filepath          = modelname + ".hdf5"
> checkpoint        = ModelCheckpoint(filepath,
                                      monitor='val_acc',
                                      verbose=0,
                                      save_best_only=True,
                                      mode='max')
```

```
> csv_logger        = CSVLogger(modelname + '.csv')
> callbacks_list    = [checkpoint,csv_logger]
```

# The net

## 5. Train model

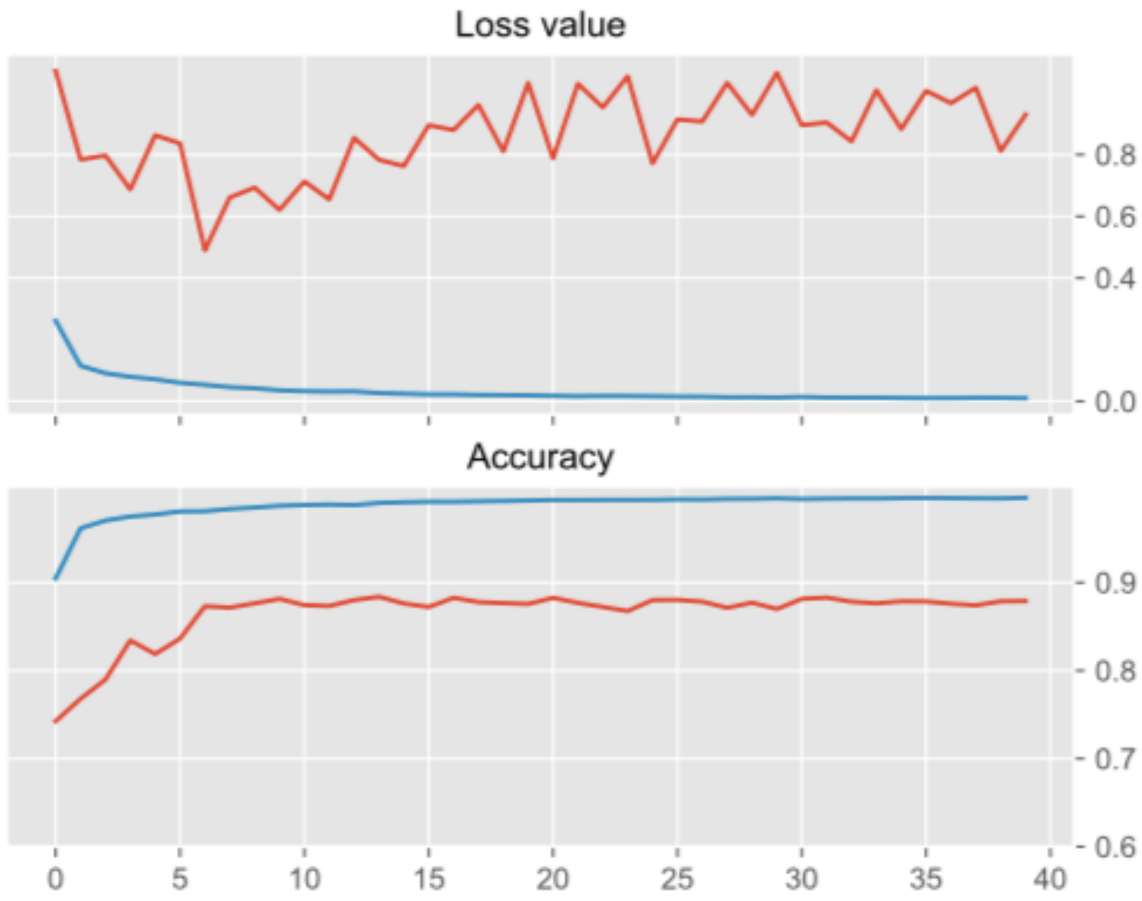
- The line for training

```
> model.fit(trDat,
            trLbl,
            validation_data=(tsDat, tsLbl),
            epochs=40,
            batch_size=128,
            shuffle=True,
            callbacks=callbacks_list)
```

```
Train on 33800 samples, validate on 4320 samples
Epoch 1/40
33800/33800 [==============================] - 100s 3ms/sample - loss: 0.2614 - acc: 0.9047 - val_loss: 1.0719 - val_acc: 0.7426
Epoch 2/40
33800/33800 [==============================] - 96s 3ms/sample - loss: 0.1154 - acc: 0.9616 - val_loss: 0.7840 - val_acc: 0.7681
Epoch 3/40
33800/33800 [==============================] - 96s 3ms/sample - loss: 0.0912 - acc: 0.9705 - val_loss: 0.7976 - val_acc: 0.7898
Epoch 4/40
33800/33800 [==============================] - 97s 3ms/sample - loss: 0.0797 - acc: 0.9750 - val_loss: 0.6873 - val_acc: 0.8343
Epoch 5/40
33800/33800 [==============================] - 97s 3ms/sample - loss: 0.0716 - acc: 0.9772 - val_loss: 0.8630 - val_acc: 0.8188
......
```

# The net

## 6. Result



- Classification result

```
Best accuracy (on testing dataset): 88.38%
             precision    recall  f1-score   support

     Normal     0.8766    0.9812    0.9260      3200
        CAD     0.9187    0.6054    0.7298      1120

avg / total     0.8875    0.8838    0.8751      4320
```

- Confusion matrix

```
[[3140   60]
 [ 442  678]]
```

🟦 training

🟥 testing

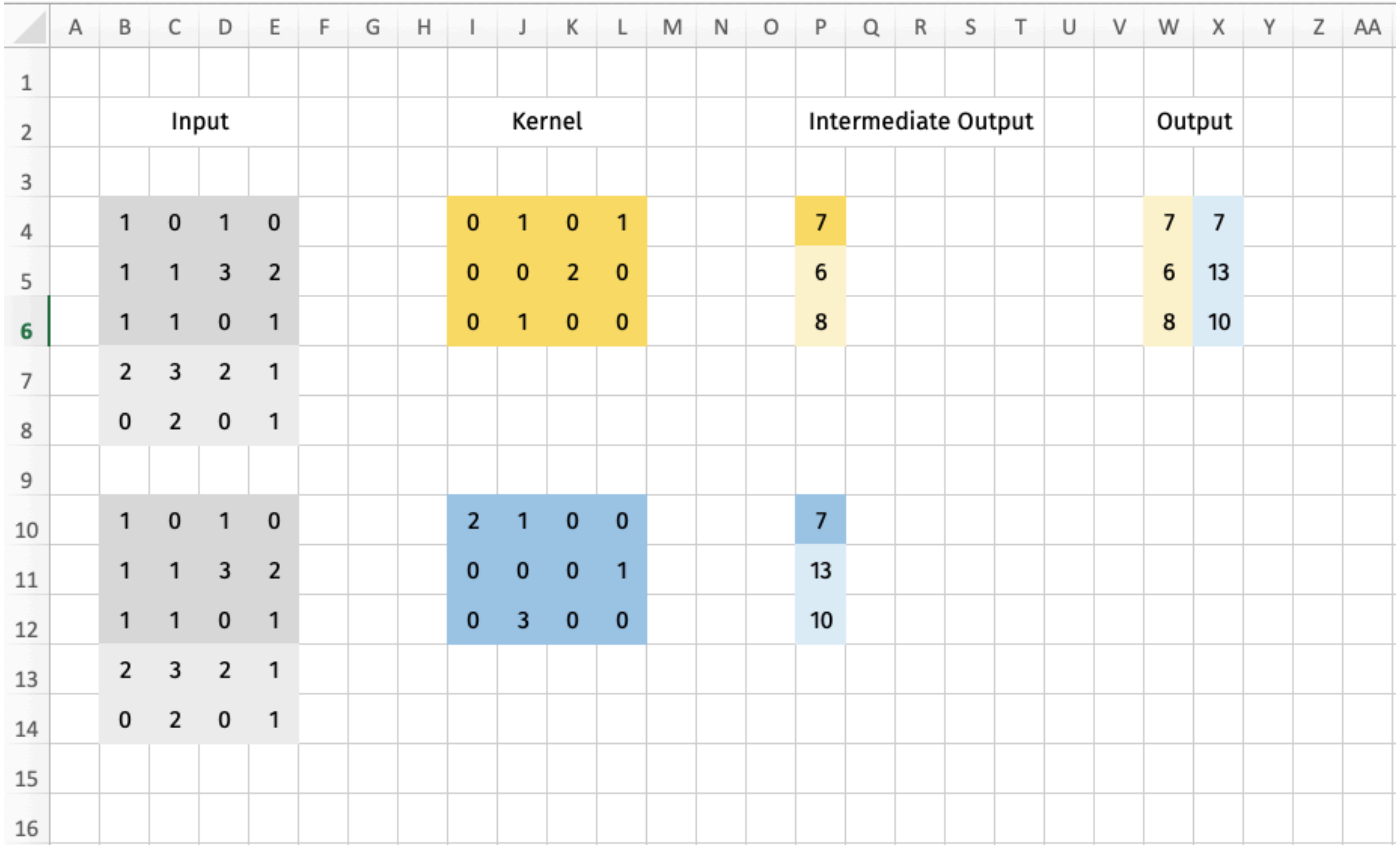
# Feature extraction

Better learning

- In previous examples, we simply feed the original data (although with a bit of re-arrangement) into the net
- Feature extraction is missing!
- Without feature extraction, learning is slower and not optimal
- Should perform rounds of feature extraction, reduce the amount of recurrent calculation so that training is faster

# Conv1D

size 3



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | AA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 2 | | Input | | | | | | | Kernel | | | | | | | Intermediate Output | | | | | | | Output | | | | |
| 3 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 4 | | 1 | 0 | 1 | 0 | | | | 0 | 1 | 0 | 1 | | | | 7 | | | | | | | 7 | 7 | | | |
| 5 | | 1 | 1 | 3 | 2 | | | | 0 | 0 | 2 | 0 | | | | 6 | | | | | | | 6 | 13 | | | |
| 6 | | 1 | 1 | 0 | 1 | | | | 0 | 1 | 0 | 0 | | | | 8 | | | | | | | 8 | 10 | | | |
| 7 | | 2 | 3 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | |
| 8 | | 0 | 2 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | |
| 9 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 10 | | 1 | 0 | 1 | 0 | | | | 2 | 1 | 0 | 0 | | | | 7 | | | | | | | | | | | |
| 11 | | 1 | 1 | 3 | 2 | | | | 0 | 0 | 0 | 1 | | | | 13 | | | | | | | | | | | |
| 12 | | 1 | 1 | 0 | 1 | | | | 0 | 3 | 0 | 0 | | | | 10 | | | | | | | | | | | |
| 13 | | 2 | 3 | 2 | 1 | | | | | | | | | | | | | | | | | | | | | | |
| 14 | | 0 | 2 | 0 | 1 | | | | | | | | | | | | | | | | | | | | | | |
| 15 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 16 | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Kernel size (in one direction): 3
Number of filters: 2

# Conv 1D

Determine the output size

- $M_r$, $M_c$: Number of rows and columns in the output, respectively
- $W_r$: Number of rows in the input
- $F_r$: Filter size (along the rows)
- $P_r$: Amount of padding (along the rows)
- $S_r$,: Stride (along the rows)
- $N_F$,: The number of filters

$$M_r = \left\lfloor \frac{W_r + P_r - F_r}{S_r} \right\rfloor + 1$$

$$M_c = N_F$$

# Conv 1D

Calculate the parameters

- For each filter, the number of parameters is

$$W_c \times F_r + 1$$

- Thus, for N$_F$ amount of filters, the number of parameters is

$$N_F \times \left(W_c \times F_r + 1\right)$$

M$_r$, M$_c$: Number of rows and columns in the output, respectively

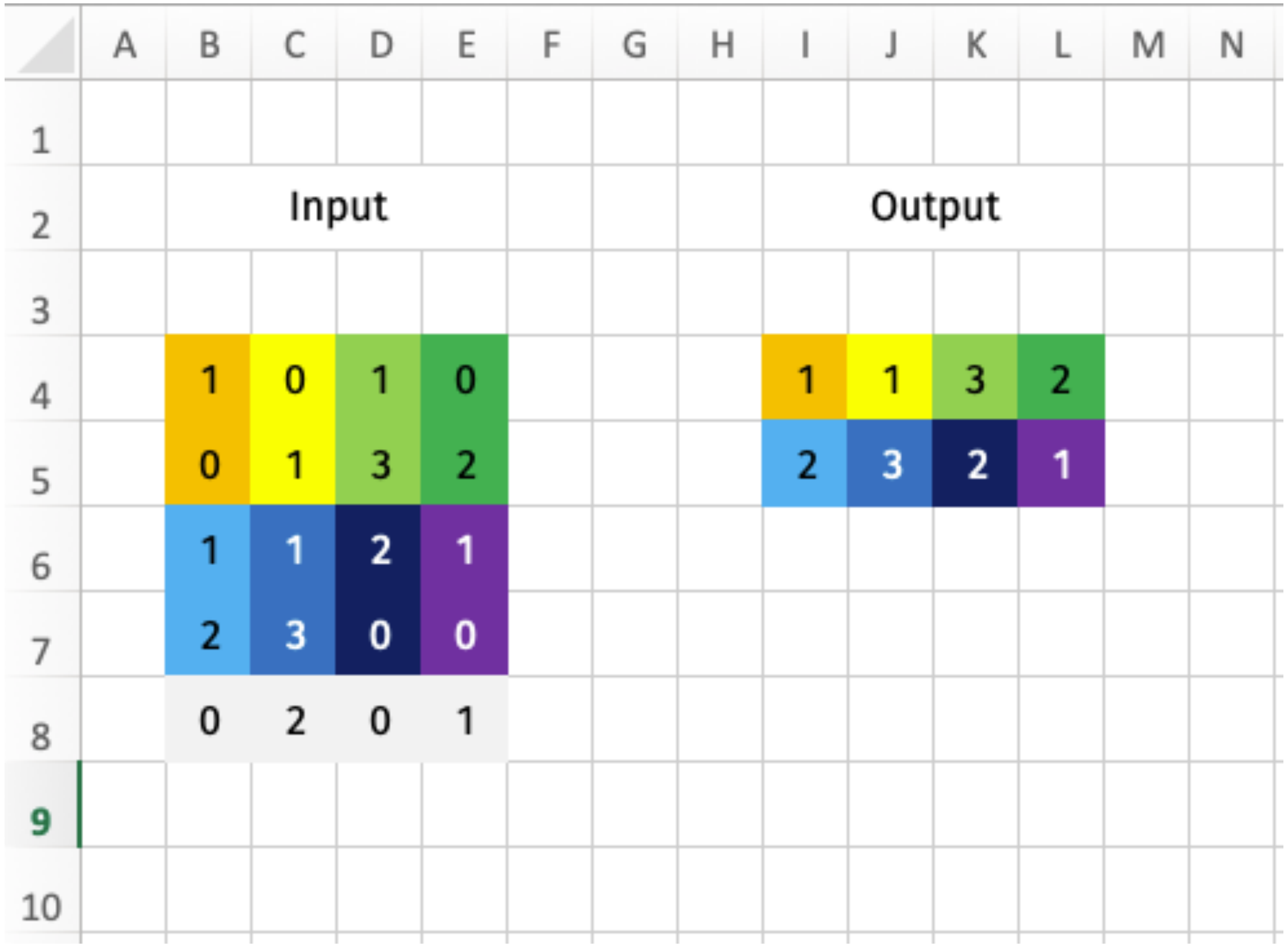
W$_c$: Number of columns in the input

F$_r$: Filter size (along the rows)

N$_F$,: The number of filters

# MaxPooling1D

size 3



| | A | B | C | D | E | F | G | H | I | J | K | L | M | N |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | | | | | | | | | | | | | | |
| 2 | | Input | | | | | | | Output | | | | | |
| 3 | | | | | | | | | | | | | | |
| 4 | | 1 | 0 | 1 | 0 | | | | 1 | 1 | 3 | 2 | | |
| 5 | | 0 | 1 | 3 | 2 | | | | 2 | 3 | 2 | 1 | | |
| 6 | | 1 | 1 | 2 | 1 | | | | | | | | | |
| 7 | | 2 | 3 | 0 | 0 | | | | | | | | | |
| 8 | | 0 | 2 | 0 | 1 | | | | | | | | | |
| 9 | | | | | | | | | | | | | | |
| 10 | | | | | | | | | | | | | | |

Kernel size (in one direction): 2
Stride (in one direction): 2

# Stacked Conv1D LSTM

ECG classification in Keras

```
> def createModel():
      inputs  = Input(shape=(trDat.shape[1],length))
      y       = Conv1D(32, 5, activation='relu')(inputs)
      y       = Dropout(0.25)(y)
      y       = Conv1D(32, 5, activation='relu')(y)
      y       = MaxPooling1D(2)(y)
      y       = Conv1D(48, 5, activation='relu')(y)
      y       = Dropout(0.5)(y)
      y       = Conv1D(48, 5, activation='relu')(y)
      y       = MaxPooling1D(2)(y)
      y       = Conv1D(64, 5, activation='relu')(y)
      y       = Dropout(0.5)(y)
      y       = Conv1D(64, 5, activation='relu')(y)
      y       = MaxPooling1D(2)(y)
      y       = LSTM(8,
                     return_sequences=True,
                     dropout=0.5,
                     recurrent_dropout=0.5)(y)
      y       = LSTM(4,
                     return_sequences=True,
                     dropout=0.5,
                     recurrent_dropout=0.5)(y)
      y       = LSTM(2)(y)
      y       = Dense(1, activation='sigmoid')(y)

      model = Model(inputs=inputs,outputs=y)
      model.compile(loss='binary_crossentropy',
                    optimizer='adam',
                    metrics=['accuracy'])
      return model
```

```
length    = 24
dist      = 6
```

# Stacked Conv1D LSTM

ECG classification in Keras

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 211, 24)           0
_________________________________________________________________
conv1d (Conv1D)              (None, 207, 32)           3872
_________________________________________________________________
dropout (Dropout)            (None, 207, 32)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 203, 32)           5152
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 101, 32)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 97, 48)            7728
_________________________________________________________________
dropout_1 (Dropout)          (None, 97, 48)            0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 93, 48)            11568
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 46, 48)            0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 42, 64)            15424
_________________________________________________________________
dropout_2 (Dropout)          (None, 42, 64)            0
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 38, 64)            20544
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, 19, 64)            0
_________________________________________________________________
lstm (LSTM)                  (None, 19, 8)             2336
_________________________________________________________________
lstm_1 (LSTM)                (None, 19, 4)             208
_________________________________________________________________
lstm_2 (LSTM)                (None, 2)                 56
_________________________________________________________________
dense (Dense)                (None, 1)                 3
=================================================================
Total params: 66,891
Trainable params: 66,891
```

# Stacked Conv1D LSTM

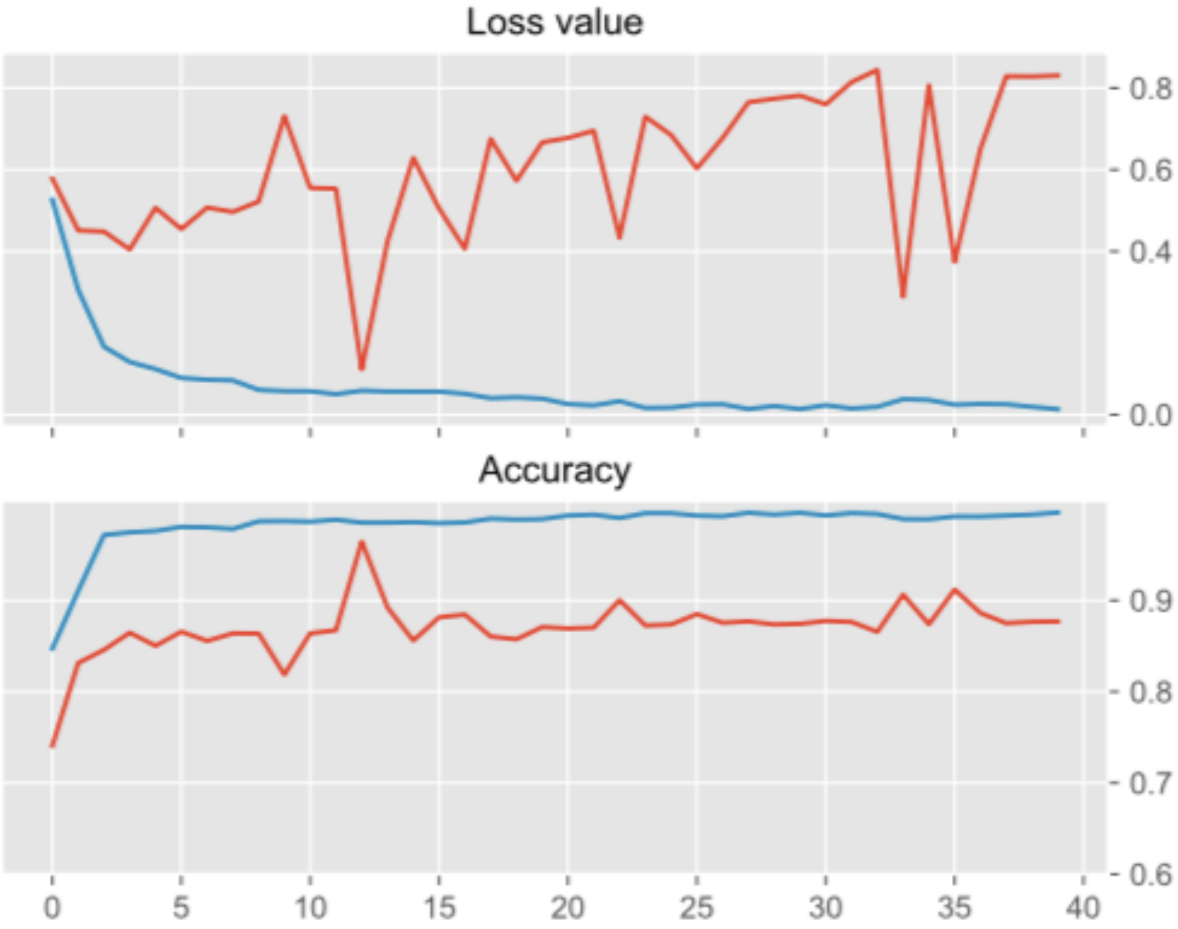
## The result

- Classification result

```
Best accuracy (on testing dataset): 96.44%
             precision    recall  f1-score   support

     Normal     0.9997    0.9522    0.9754      3200
        CAD     0.8797    0.9991    0.9356      1120

avg / total     0.9686    0.9644    0.9651      4320
```

- Confusion matrix

```
[[3047  153]
 [   1 1119]]
```



training

testing