กิจกรรมที่ 5 : Models (MLP, CNN, LSTM)

5.1 : สร้าง Classification Model เพื่อประมาณระดับการหลับ ด้วย (NN, CNN, LSTM)

- Library ที่ใช้

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

import tensorflow as tf
from keras import Model, Input
from keras.models import Sequential
from keras.layers import Conv2D, LSTM, MaxPooling2D, Flatten, Dense, Dropout
from keras import optimizers

import glob
from scipy import stats
import datetime as dt
from tqdm import tqdm
```

- อ่านข้อมูลจากไฟล์

  *_acceleration.txt อย่างน้อย 2 ไฟล์ (2 users)

  - fnames = glob.glob("*_acceleration.txt") ดึงชื่อไฟล์ acceleration ของทุก user
  - for f in tqdm(fnames):

    tester = pd.read_csv(f, sep = ' ',names=['timedelta', 'accX', 'accY', 'accZ'])

    testers.append(tester) # append ข้อมูลแต่ละ user

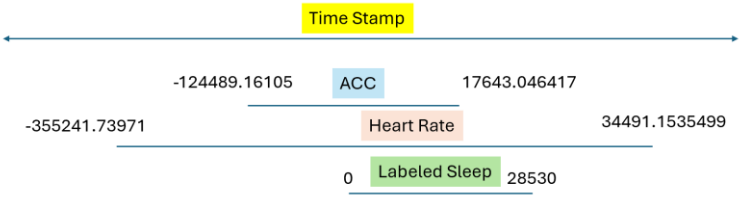
    ACC= pd.concat(testers)

  *_heartrate.txt อย่างน้อย 2 ไฟล์ (2 users) # อ่านเช่นเดียวกับ *_acceleration.txt

  *_labeled_sleep.txt อย่างน้อย 2 ไฟล์ (2 users) # อ่านเช่นเดียวกับ *_acceleration.txt

- แสดงเวลาเริ่มต้นและสิ้นสุดของข้อมูล acceleration และของข้อมูล heartrate

```
ACC start:  -124489.16105 ACC end:  17643.046417
HeartR start:  -355241.73971 HeartR end:  34491.1535499
SleepL start:  0 SleepL end:  28530
```



- เลือกข้อมูลเฉพาะช่วงเวลาตรงกันทั้งหมด (ACC: acceleration, HeartR: heart rate, SleepL: Labeled Sleep)

Commented [OC1]: เปลี่ยนเป็นของแลปใหม่

- ปรับข้อมูลให้แต่ละแถวเป็นช่วงเวลาเดียวกัน โดยกำหนดให้แต่ละแถวเป็นเวลาทุก 1 วินาที

  - ACC: acceleration เป็นข้อมูลความเร่ง 3 แกน (X,Y,Z) บ่งบอกการเคลื่อนที่ข้อมือใน 3 ทิศทาง ซึ่งเก็บละเอียดหลายครั้ง ต่อ วินาที เมื่อต้องปรับข้อมูลให้เป็นทุก 1 วินาที จำเป็นต้องรวมข้อมูลจากหลายแถว

    - ปรับ timestamp ให้เป็น datetime หน่วยเป็น วินาที (‘1s’)
      - timedelta_unit = 's' # Define the timedelta_unit variable
      - resample_rule = '1s'
      - ACC['timedelta'] = pd.DataFrame(pd.to_timedelta(ACC['timedelta'], timedelta_unit).round(resample_rule))

	timedelta	accX	accY	accZ
98777	0 days 00:00:00	-0.234650	0.905975	0.362747
98778	0 days 00:00:00	-0.231232	0.893265	0.371613
98779	0 days 00:00:00	-0.227814	0.915848	0.369049
98780	0 days 00:00:00	-0.240524	0.919159	0.352890
98781	0 days 00:00:00	-0.240448	0.889175	0.350143

    - ปรับข้อมูลให้เป็นแถว 1 วินาที โดยเฉลี่ยข้อมูลจากแถวต่างๆในวินาทีเดียวกัน
      - df_temp = ACC.groupby('timedelta').mean().reset_index()
      - ACC = df_temp

timedelta	accX	accY	accZ
0 days 00:00:00	-0.143596	0.434711	-0.558406
0 days 00:00:01	-0.184721	0.438487	-0.492099
0 days 00:00:02	-0.213871	0.403676	-0.476563
0 days 00:00:03	-0.176616	0.391760	-0.571252
0 days 00:00:04	0.072789	0.338097	-0.571049

  - HeartR: heart rate เป็นข้อมูลอัตราการเต้นของหัวใจ ซึ่งเก็บทุกๆ 3 - 6 วินาที
    - ปรับ timestamp ให้เป็น datetime หน่วยเป็น วินาที (‘1s’) เช่นเดียวกับ ACC
    - ปรับข้อมูลให้เป็นแถว 1 วินาที โดยเพิ่มจำนวนข้อมูลแถวต่างๆ หากข้อมูลวินาทีไหนหายไป ด้วยค่า median ของข้อมูลต้นฉบับ และหากมีค่า NaN ให้ทำ ffill()
      - HeartR = HeartR.set_index('timedelta').resample(resample_rule).median().ffill()

timedelta	heartrate
0 days 00:00:02	65.0
0 days 00:00:03	65.0
0 days 00:00:04	65.0
0 days 00:00:05	65.0

  - SleepL: Labeled Sleep เป็นข้อมูลระดับการหลับในช่วง 0 – 5 ซึ่งเก็บทุกๆ 30 วินาที และใช้เป็นผลลัพธ์ที่ต้องการทำนาย
    - ปรับ timestamp ให้เป็น datetime หน่วยเป็น วินาที (‘1s’) เช่นเดียวกับ ACC, HeartR
    - ปรับข้อมูลให้เป็นแถว 1 วินาที โดยเพิ่มจำนวนข้อมูลแถวต่างๆ หากข้อมูลวินาทีไหนหายไป ด้วยค่า median ของข้อมูลต้นฉบับ และหากมีค่า NaN ให้ทำ ffill() เช่นเดียวกับ HeartR

timedelta	sleep
0 days 00:00:30	0.0
0 days 00:00:31	0.0
0 days 00:00:32	0.0
0 days 00:00:33	0.0

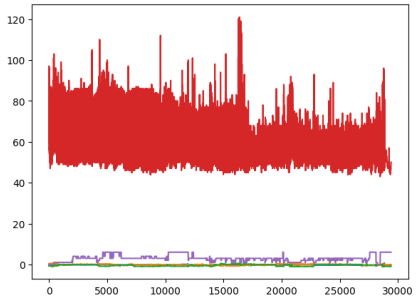
- รวมข้อมูลทั้งสามในตาราง (df) เดียวกัน เพื่อทำการปรับแถวของเวลาแต่ละวินาทีให้ตรงกัน

	timedelta	accX	accY	accZ	heartrate	sleep
0	0 days 00:00:00	-0.143596	0.434711	-0.558406	NaN	NaN
1	0 days 00:00:01	-0.184721	0.438487	-0.492099	NaN	NaN
2	0 days 00:00:02	-0.213871	0.403676	-0.476563	65.0	NaN
3	0 days 00:00:03	-0.176616	0.391760	-0.571252	65.0	NaN
4	0 days 00:00:04	0.072789	0.338097	-0.571049	65.0	NaN

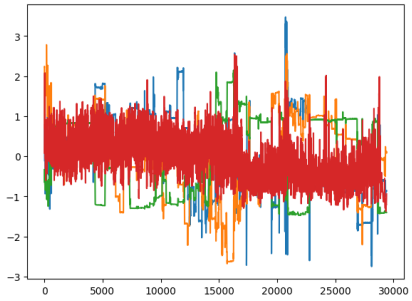
  - ขั้นตอนนี้จะเกิด NaN ในกรณีที่วินาทีนั้นๆ ไม่มีข้อมูล โดยกำหนดให้ fillna()
    - ACC ด้วย average
    - HeartR ด้วย median
    - SleepL ด้วย 0
  - ตรวจสอบค่า class label ใน SleepL หากมีค่าติดลบ -1 ทำการปรับ class label ทุก class +1
    np.unique()
  - ทำการ drop column ‘timedelta’

	accX	accY	accZ	heartrate	sleep
0	-0.143596	0.434711	-0.558406	62.0	0.0
1	-0.184721	0.438487	-0.492099	62.0	0.0
2	-0.213871	0.403676	-0.476563	65.0	0.0
3	-0.176616	0.391760	-0.571252	65.0	0.0
4	0.072789	0.338097	-0.571049	65.0	0.0

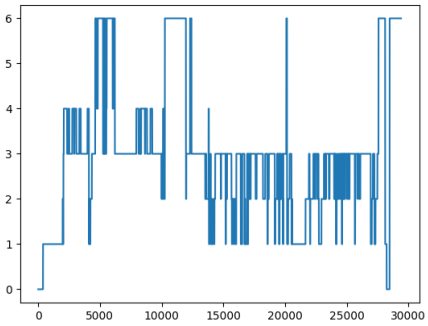
- ทำ Standardize ข้อมูล df
- แยกข้อมูล df เป็นข้อมูลคุณลักษณะอินพุท (feature) X จากคอลัมน์ของค่า accX, accY, accZ, HeartR และ ข้อมูลผลลัพธ์ Y จากคอลัมน์ ของค่า SleepL
- ทำการกรองสัญญาณรบกวน (noise) ของข้อมูลคุณลักษณะอินพุท (feature) X ด้วยเทคนิค Simple Moving Average (**SMA**) กำหนด window size = [3, 5, 10] เลือกอย่างน้อย 1 ค่า
- แสดงกราฟเปรียบเทียบข้อมูลคุณลักษณะอินพุท (feature) X ก่อน และ หลัง การทำ Standardize และ SMA



ก่อน Standardize



หลัง การทำ Standardize และ SMA



SleepL

- เตรียม NN Model:
  - เตรียมข้อมูล train 70% test 30% (train_test_split())

    ข้อมูลอินพุท X หลัง Standardize และ ลดสัญญาณรบกวน (SMA)

    โดยจะนำเข้าทีละแถว คือนำเข้าเฉพาะเวลาที่ต้องการทำนาย เช่น แถวที่ 1 เป็นข้อมูลเวลาวินาทีที่ 0 ผลลัพธ์การทำนายจะเป็นระดับการหลับที่วินาที 0

    ข้อมูลผลลัพธ์ Y ทำ one hot encoding

    Yone_hot = pd.get_dummies(Y)
  - กำหนด NN Architecture

    Number of Layers: 3 Hidden layers (50,50,50)
    1 output layer (7)
    Activation = 'relu'

    Optimizer: solver = ‘adam’,
    max_iter = EP,
    alpha = lr,
    random_state
  - สร้าง Model object instance

    model = MLPClassifier()
  - Train Model

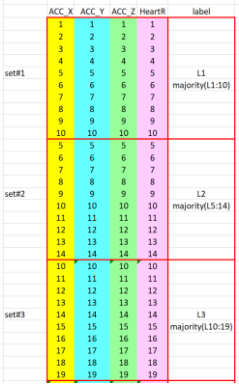
    model.fit()

- Test Model

  model.predict()
- แสดงค่า Model Performance (Confusion Matrix, Classification Report)

- เตรียม CNN Model:
  - จัดข้อมูลเป็นชุด 4D array
    - กำหนดพารามิเตอร์ของ 4D array
      - slidingW_size = 100
      - Nfeature = 4
      - XR shape1 = (จำนวนชุดข้อมูล, slidingW_size, Nfeature, 1) #(nSet, H, W, 1)
      - XT shape2 = (จำนวนชุดข้อมูล, Nfeature, slidingW_size, 1) #(nSet, H, W, 1)
      - stride = [5, 10, 20] # เลือกอย่างน้อย 1 ค่า
    - จัดชุดข้อมูล 4D array => XR shape1 = (จำนวนชุดข้อมูล, slidingW_size, Nfeature, 1)

```
for i in range(0, len(X)-slidingW, stride):
        # จัดข้อมูล X
        # feature.shape = (slidingW_size, Nfeature)
        feature = np.array(X.iloc[i: i+slidingW])
        # featureR.shape = (slidingW_size, Nfeature, 1)
        featureR = feature.reshape(slidingW_size, Nfeature, 1)
        XR.append(featureR)
        # จัดข้อมูล Y
        label = np.array(Y.iloc[i: i+slidingW])
        mode_values, countL = stats.mode(label,axis=None) #whole array
        y.append(mode_values)
XR = np.array(XR)
y = pd.DataFrame(np.array(y).ravel(),columns=['Sleep_Label'])
```



    - จัดชุดข้อมูล 4D array => XT shape2 = (จำนวนชุดข้อมูล, Nfeature, slidingW_size, 1)

```
for i in range(0, len(X)-slidingW, stride):
        # จัดข้อมูล X
```

```
            # feature.shape = (slidingW_size, Nfeature)
            feature =
            featureT = np.transpose(feature)
            XT.append(featureT)
            # จัดข้อมูล Y
            label =
            mode_values, countL =
            y.append()
        XT = np.array(XT)
        y = pd.DataFrame(np.array(y).ravel(),columns=['Sleep_Label'])
```

	ACC_X	1	2	3	4	5	6	7	8	9	10
set#1	ACC_Y	1	2	3	4	5	6	7	8	9	10
	ACC_Z	1	2	3	4	5	6	7	8	9	10
	HeartR	1	2	3	4	5	6	7	8	9	10
	ACC_X	5	6	7	8	9	10	11	12	13	14
set#2	ACC_Y	5	6	7	8	9	10	11	12	13	14
	ACC_Z	5	6	7	8	9	10	11	12	13	14
	HeartR	5	6	7	8	9	10	11	12	13	14
	ACC_X	10	11	12	13	14	15	16	17	18	19
set#3	ACC_Y	10	11	12	13	14	15	16	17	18	19
	ACC_Z	10	11	12	13	14	15	16	17	18	19
	HeartR	10	11	12	13	14	15	16	17	18	19

	L1
	majority(L1:10)
	L2
	majority(L5:14)
	L3
	majority(L10:19)

- เตรียมข้อมูล train 70% test 30% (train_test_split())
  - 4D array => XR shape1 = (จำนวนชุดข้อมูล, slidingW_size, Nfeature, 1) = (nSet, 100, 4, 1)
    train_test_split(XR, y,)
  - 4Darray => XT shape2 = (จำนวนชุดข้อมูล, Nfeature, slidingW_size, 1) = (nSet, 4, 100, 1)
    train_test_split(XT, y,)

- กำหนด **CNN Architecture**

CNN Layers:	3 CNN layers (CNN_L1 = 32, CNN_L2 = 64, CNN_L3 = 128)
Conv Ker_size:	(3,3)
Padding:	“same”
activation:	"relu"
Maxpooling2D:	XR pool_size = (2,1) #reduce sliding window by ½
	XT pool_size = (1,2) #reduce sliding window by ½
Dropout:	0.4
Flatten:	convert shape featuremap layer (3D) สุดท้าย to a vector (1D)
NN Layers:	2 NN layers Dense_size=512, activation = “Relu” output layer = Nclass, activation = “sigmoid”
Optimizer:	optimizer=optimizers.Adam(lr) Loss = tf.keras.losses.CategoricalCrossentropy() Metrics=[“acc”]

- กำหนด **Input_shape**

  XR Shape1: Input_shape = (slidingW_size, Nfeature, 1)

  XT Shape2: Input_shape = (Nfeature, slidingW_size, 1)

- สร้าง Model object instance

```
# ------------ Create CNN Model ------------------------------
model.add(Conv2D(CNN_L1, kernel_size=Ker_size, activation=Act_func,
input_shape=Input_shape,padding='same'))
model.add(MaxPooling2D(pool_size=Pooling_size))
model.add(Dropout(0.4))

model.add(Conv2D(CNN_L2, kernel_size=Ker_size, activation= Act_func, padding='same'))
model.add(MaxPooling2D(pool_size= Pooling_size))
model.add(Dropout(0.4))

model.add(Conv2D(CNN_L3, kernel_size=Ker_size, activation= Act_func,
 padding='same'))
model.add(MaxPooling2D(pool_size= Pooling_size))
model.add(Dropout(0.4))

model.add(Flatten())
model.add(Dense(D_L1_size , activation= Act_func ))
model.add(Dense(D_out, activation='sigmoid'))
```

```
adam = optimizers.Adam(learning_rate=lr)
model.compile(optimizer=adam,  loss, metrics)
model.summary()
```

- Train Model

```
XRmodel.fit()          # Train using XR
model.fit()            # Train using XT
```

- Test Model

```
XRmodel.predict()      # Test using XR
XTmodel.predict()      # Test using XT
```

- แสดงค่า Model Performance (Confusion Matrix, Classification Report)

  XR confusion matrix , XR classification report

  XT confusion matrix

- เตรียม **LSTM Model**:
  - จัดข้อมูลเป็นชุด 4D array
    - กำหนดพารามิเตอร์ของ 4D array
      - slidingW_size = 100
      - Nfeature = 4
      - XR shape1 = (จำนวนชุดข้อมูล, slidingW_size, Nfeature, 1) #(nSet, H, W, 1)
      - XT shape2 = (จำนวนชุดข้อมูล, Nfeature, slidingW_size, 1) #(nSet, H, W, 1)
      - stride = [5, 10, 20] # เลือกอย่างน้อย 1 ค่า
    - จัดชุดข้อมูล 4D array => XR shape1 = (จำนวนชุดข้อมูล, slidingW_size, Nfeature, 1)

LSTM ใช้ XR เหมือนที่สร้างไว้ใช้กับ CNN

- จัดชุดข้อมูล 4D array => XT shape2 = (จำนวนชุดข้อมูล, Nfeature, slidingW_size, 1)
  LSTM ใช้ XT เหมือนที่สร้างไว้ใช้กับ CNN

- เตรียมข้อมูล train 70% test 30% (train_test_split())
  - 4D array => XR shape1 = (จำนวนชุดข้อมูล, slidingW_size, Nfeature, 1) = (nSet, 100, 4, 1)
    train_test_split(XR, y,)
  - 4Darray => XT shape2 = (จำนวนชุดข้อมูล, Nfeature, slidingW_size, 1) = (nSet, 4, 100, 1)
    train_test_split(XT, y,)

- กำหนด **LSTM Architecture**

LSTM Layers:	2 LSTM layers (LSTM_L1 = 100, LSTM_L2 = 50)
	Node=[50, 100, 250, 500] #เลือก 2 ค่าสำหรับแต่ละ Layer
Dropout:	dropRate_L1 = 0.25
	dropRate_L2 = 0.5
NN Layers:	2 NN layers
	output layer = Nclass, activation = “softmax”
Optimizer:	optimizer=optimizers.Adam(lr)
	Loss = tf.keras.losses.CategoricalCrossentropy()
	Metrics=[“acc”]

- กำหนด **Input_shape**

  XR Shape1: Input_shape = (slidingW_size, Nfeature, 1)

  XT Shape2: Input_shape = (Nfeature, slidingW_size, 1)

- สร้าง Model object instance

```
# ------------ Create LSTM Model ------------------------------
model = Sequential()
model.add( LSTM ( LSTM_L1, return_sequences=True,
                  input_shape=Input_shape))
model.add(Dropout(dropRate_L1 ))
model.add(LSTM(LSTM_L2 ))
model.add(Dropout(dropRate_L12))
model.add(Dense(n_classes, activation='softmax'))
model.summary()
```

```
adam = optimizers.Adam(learning_rate=lr)
model.compile(optimizer=adam,  loss, metrics)
```

- Train Model

```
XRmodel.fit()          # Train using XR
XTmodel.fit()          # Train using XT
```

- Test Model

```
XRmodel.predict()      # Test using XR
```

XTmodel.predict() # Test using XT

- แสดงค่า Model Performance (Confusion Matrix, Classification Report)

  XR confusion matrix , XR classification report

  XT confusion matrix , XT classification report

5.2 :สร้าง simple CNN สำหรับ Classifcation Model เพื่อประมาณ จัดกลุ่มรูปสามเหลี่ยมและสี่เหลี่ยม

- Library ที่ใช้ manual_seed เพื่อ random weight ภายในโมเดล และ กำหนด device ที่ใช้รัน

```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix,classification_report
import seaborn as sns

torch.manual_seed(841)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

- สร้าง 2D Array ของรูปภาพ square และ triangle

  square = 2D Array

  triangle = 2D Array

- Convert 2D Array to 2D Tensor

  tensor_square = torch.tensor(square)

  tensor_triangle =

  label_square = torch.tensor([1])

  label_triangle = torch.tensor([0])

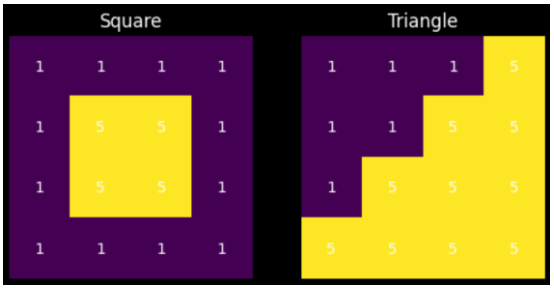
- สร้าง input 2D tensor เพื่อเตรียมเป็นข้อมูลสอน CNN Model

  data_tensor = torch.stack([tensor_square, tensor_triangle])

  label_tensor = torch.stack([label_square, label_triangle])

- แสดงภาพของ data_tensof

  plt.subplot(1, 2, 1), plt.axis('off'), plt.imshow(tensor_square[0])

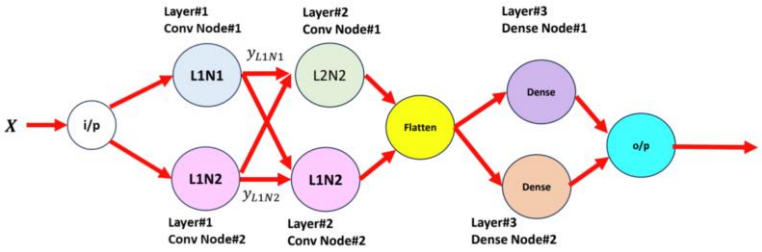


- ตรวจสอบความถูกต้องของ tensor shape
- เตรียม pytorch dataloader

dataset = torch.utils.data.TensorDataset()

dataloader = torch.utils.data.DataLoader() # shuffle=True

- กำหนด CNN Architecture



- สร้างคลาส SimpleCNN สำหรับเก็บโครงสร้างโมเดล

```
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        ### START CODE HERE ###
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=2, kernel_size=3, stride=1, padding=1)
        self.relu1 = nn.ReLU()
        self.conv2 = 
        self.relu2 = 
        self.flatten = nn.Flatten()
        self.fc = nn.Linear(in_channels,out_channels=2)
        self.relu3 = 
        self.out = nn.Linear(in_channels=2,out_channels=1)
        self.sigmoid = nn.Sigmoid()
        ### END CODE HERE ###
```

โดยกำหนดให้

- Inherite จาก class nn.Module
- สร้าง Constructor เพื่อ initial Super Class: CNN model
- กำหนด Convolution 2D เพื่อทำ convolution operation ข้อมูลขาเข้ากับ kernel weights
- กำหนด Flatten เพื่อ convert multi-dimensional tensor to 1D tensor
- กำหนด Linear Kernel Mapping Node ที่จะใช้สำหรับ Fully connected layer 1 (fc) เป็น nn.Linear() โดย fc เป็น NN Layer และ out เป็น output (o/p) layer
  กำหนด Activation Node เป็น Sigmoid()

- สร้าง Forward Function เพื่อสร้างเส้นทางการ process ข้อมูล ไปยัง Node ประมวลผลต่างๆ ในแต่ละ Layer ตามโครงสร้างที่กำหนด โดยเชื่อมต่อ CNN (2 layers), flatten(), NN node (Dense 1 Layer, output 1 layer)

```python
def forward(self, x):
    x = self.conv1(x)
    x = self.relu1(x)
    x = self.conv2(x)
    x = self.relu2(x)
    x = self.flatten(x)
    x = self.fc(x)
    x = self.relu3(x)
    x = self.out(x)
    x = self.sigmoid(x)
    return x
```

- สร้าง object instance ของ Class simpleCNN และกำหนดพารามิเตอร์ดังนี้

  กำหนด device ที่จะใช้งาน

  device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

```python
model = SimpleCNN().to(device)
print(model)
```

- ตรวจสอบโครงสร้างโมเดล
- Train Model:

  กำหนดพารามิเตอร์ที่ใช้ epochs, lr,

  loss = nn.BCELoss() #BinaryCrossEntropyLoss()

  optimizer = torch.optim.SGD(model.parameters(), lr=lr)

```python
loss_history = None
for epoch in range(epochs):
    for x,y in dataloader:
        x = x.to(device)
        y = y.to(device)
        optimizer.zero_grad()
        output = model(x.float())
        loss = criterion(output, y.float())
        loss.backward()
        optimizer.step()
```

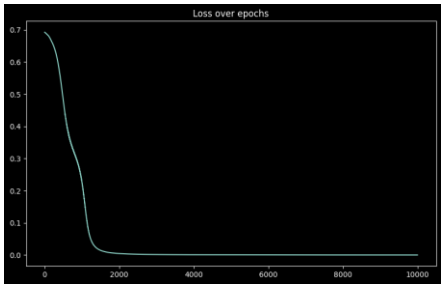
นำเข้า x (i/p) และ y (o/p) ไปยังอุปกรณ์ที่ใช้รัน

Model prediction

Loss(output, y_real)

ปรับ weight ตาม Optimizer_step

- แสดงกราฟ loss ระหว่างช่วง Train



- แสดง model weights โดยการ save model แล้ว เรียกดู model.state_dict()

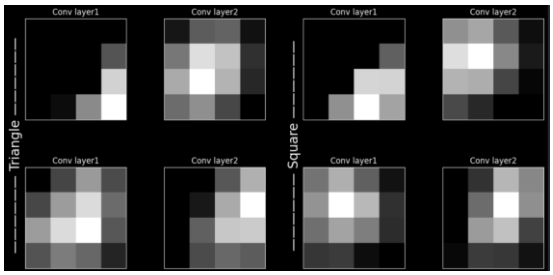
```
OrderedDict([('conv1.weight',
              tensor([[[[ 0.4178,  0.3936,  0.0348],
                        [ 0.1918, -0.3002, -0.3480],
                        [ 0.0229, -0.1447, -0.0315]]],


                      [[[-0.1744,  0.0069,  0.2229],
                        [ 0.3050,  0.4549,  0.2327],
                        [-0.1438,  0.4869,  0.5488]]]], device='cuda:0')),
             ('conv1.bias', tensor([-0.1141,  0.1048], device='cuda:0')),
             ('conv2.weight',
              tensor([[[[ 0.1378,  0.0572, -0.3133],
                        [-0.1260, -0.0546, -0.3380],
                        [-0.1682, -0.1877, -0.0948]],

                       [[-0.0245,  0.2931,  0.3160],
                        [ 0.2575,  0.5127,  0.3996],
                        [-0.0087,  0.1811,  0.2367]]],


                      [[[-0.0150, -0.0864, -0.0030],
                        [-0.0923,  0.0210,  0.1223],
                        [-0.2476,  0.2070,  0.2269]],

                       [[ 0.1747, -0.0700, -0.0095],
...
                        0.1317, -0.2394, -0.0793,  0.2401,  0.0026, -0.0737,  0.1028,  0.0075]],
                    device='cuda:0')),
             ('fc.bias', tensor([-0.0721, -0.1180], device='cuda:0')),
             ('out.weight', tensor([[ 0.5076, -0.9841]], device='cuda:0')),
```

- แสดงภาพ feature map ที่เป็นผลลัพธ์จากแต่ละ CNN Node
  โดยส่ง tensor_triangle และ tensor_square ผ่าน model เพื่อดึงผลลัพธ์ จากการทำ convolution กับ model weights ที่พิมพ์ออกมา แต่ละ CNN node
  ตย. conv1_triangle, conv2_triangle = model.get_features(tensor_triangle_test.unsqueeze(1).float().to(device))
  รับ tensor รูปสามเหลี่ยมที่ใช้เทส แปลงเป็น float แล้วเข้า get_features() ซึ่งจะนำ tensor_triangle_test นี้ไปผ่าน model แล้วดึงผลลัพธ์ จาก
  conv1_triangle <= Convolution layer#1
  conv2_triangle <= Convolution layer#2

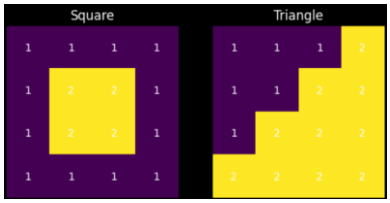
```
conv1_triangle, conv2_triangle = model.get_features(tensor_triangle.unsqueeze(1).float().to(device))
conv1_square, conv2_square = model.get_features(tensor_square.unsqueeze(1).float().to(device))
```

เก็บ feature map ของ แต่ละ CNN Node แต่ละ Layer ใน list

```
plt.figure(figsize=(12, 6))

feature_maps = [conv1_triangle, conv2_triangle, conv1_square, conv2_square]
titles = ['Conv layer1', 'Conv layer2']

for i, feature_map in enumerate(feature_maps):

    plt.subplot(2, 4, i + 1)
    # plt.axis('off')
    plt.xticks([])
    plt.yticks([])
    plt.imshow(feature_map[0, 0].squeeze().cpu().detach().numpy(), cmap='gray')
    plt.subplot(2, 4, i + 5)
    # plt.axis('off')
    plt.xticks([])
    plt.yticks([])
    plt.imshow(feature_map[0, 1].squeeze().cpu().detach().numpy(), cmap='gray')
```



- Test Model:
    - สร้าง 2D Array ของ square_test, triangle_test



    - สร้าง Test tensor dataset ของ tensor_square_test, tensor_triangle_test และ Label คำตอบ labels_square_test, labels_triangle_test
      จากนั้นรวม tensor data test ของ square และ triangle -> test_data_tensors
      Label คำตอบ ของ square และ triangle -> test_data_labels

```
square_test = None
triangle_test = None

tensor_square_test = torch.tensor(None)
tensor_triangle_test = torch.tensor(None)

labels_square_test = torch.tensor(None)
labels_triangle_test = torch.tensor(None)

test_data_tensors = torch.stack(None)
test_data_labels = torch.stack(None)
```

    - ตรวจสอบ Shape ของ test_data_tensors, test_data_labels
    - วัดประสิทธิภาพโมเดล Confusion Matrix, Classification Report

การส่งงาน

1. ให้ Staff ตรวจที่ห้อง ECC509 ในคาบเรียนวันพฤหัสที่ 7 มีค. หรือ
2. ส่งเอกสารในฟอร์มส่ง Lab (https://forms.gle/pWUx2vHrZq29Axnx8)
   2.1 source code (ต้องส่งทุกการทดลอง ไม่ว่าจะอธิบายในห้องหรือส่งฟอร์ม)
   2.2 เอกสาร (pdf) อธิบายการทำงานของ source code (ถ้าไม่ได้อธิบายในห้อง)
   2.3 การตั้งชื่อไฟล์ “Lab#5_ชื่อกลุ่ม_รหัสสมาชิก#1_รหัสสมาชิก#2.pdf”
3. กำหนดส่ง ในฟอร์ม ภายในวันอาทิตย์ที่ 17 มีค. ทุกแลป