คู่มือปฏิบัติการแบบครบวงจร
& Step-by-Step Implementation Guide

ปัญญาประดิษฐ์ล้ำยุค
บทที่ 2–5: พื้นฐานปัญญาประดิษฐ์ (AI Foundations Ch.2–5)

รองศาสตราจารย์ ดร.ภัทรพงษ์ ผาสุขกิจ
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง (KMITL)

วิธีใช้คู่มือนี้

คู่มือเล่มนี้ครอบคลุม 4 Lab จากบทที่ 2–5 ของหนังสือ ออกแบบสำหรับนักศึกษาระดับปริญญาตรี/โท แบ่งเป็น 2 ส่วน:

ส่วน	เนื้อหา	ผลลัพธ์ที่ได้
ส่วนที่ 1 Lab Manual	คู่มือปฏิบัติการ Lab 1–4 วัตถุประสงค์, ทฤษฎีย่อ, แบบฝึกหัด, เกณฑ์การประเมิน	เข้าใจหลักการทฤษฎีและสามารถทดลองด้วยตนเองได้
ส่วนที่ 2 Step-by-Step	คำสั่ง Python ทีละขั้นตอน พร้อม code ที่รันได้จริง อธิบายทุกบรรทัดเป็นภาษาไทย	เขียน code Deep Learning ได้ด้วยตนเอง

Prerequisites: Python 3.10+, numpy, matplotlib, scikit-learn, tensorflow ≥ 2.12 หรือ PyTorch 2.0+ | แนะนำใช้ Google Colab ฟรีสำหรับนักศึกษา

การติดตั้ง Environment

```
# วิธีที่ 1: Google Colab (แนะนำ — ไม่ต้องติดตั้งอะไร)
# เปิด https://colab.research.google.com แล้วรัน code ได้ทันที

# วิธีที่ 2: ติดตั้งบน Local Machine
pip install numpy matplotlib scikit-learn tensorflow pandas seaborn

# ตรวจสอบการติดตั้ง
python -c "import tensorflow as tf; print('TF:', tf.__version__)"
python -c "import numpy as np; print('NumPy:', np.__version__)"
```

ส่วนที่ 1: คู่มือปฏิบัติการ (Lab Manual)

LAB 1	พื้นฐานปัญญาประดิษฐ์ AI Foundations: Loss Functions, Activation Functions & Normalization (บทที่ 2)
วัตถุประสงค์	1) เข้าใจฟังก์ชันการสูญเสีย (Loss Functions) แต่ละประเภท 2) เลือกใช้ Activation Function ให้เหมาะกับงาน 3) ทำ Normalization ข้อมูลก่อนเข้า Neural Network
เวลา / Tools	3 ชั่วโมง | Python 3.10+, NumPy, Matplotlib, TensorFlow/Keras
ความรู้ที่ต้องมีก่อน	คณิตศาสตร์พื้นฐาน (Matrix, Calculus เบื้องต้น), Python เบื้องต้น

1.1 ฟังก์ชันการสูญเสีย (Loss Functions) — บทที่ 2.2.1

Loss Function คือ ตัวชี้วัดว่าโมเดลทำนายผิดพลาดมากน้อยแค่ไหน ค่ายิ่งต่ำยิ่งดี

Loss Function	สูตร	ใช้เมื่อ	ช่วงค่า	Keras API
MSE (Mean Squared Error)	$\Sigma(y - \hat{y})^2 / n$	Regression — ทำนายค่าต่อเนื่อง เช่น อุณหภูมิ, ราคา	[0, ∞)	'mean_squared_error'
MAE (Mean Absolute Error)	$\Sigma\|y - \hat{y}\| / n$	Regression ที่มี outlier (robust กว่า MSE)	[0, ∞)	'mean_absolute_error'
Binary Cross-Entropy	$-[y \cdot \log(p) + (1-y) \cdot \log(1-p)]$	Classification 2 คลาส (0 หรือ 1)	[0, ∞)	'binary_crossentropy'
Categorical Cross-Entropy	$-\Sigma\, y_i \cdot \log(p_i)$	Classification หลายคลาส (ใช้กับ One-hot label)	[0, ∞)	'categorical_crossentropy'
Sparse Categorical CE	เหมือน CCE แต่ label เป็น integer (0,1,2...)	Classification หลายคลาส (label เป็นตัวเลข)	[0, ∞)	'sparse_categorical_crossentropy'

1.2 ฟังก์ชันกระตุ้น (Activation Functions) — บทที่ 2.3

Activation Function ทำให้โครงข่ายสามารถเรียนรู้ความสัมพันธ์ที่ไม่เป็นเชิงเส้น (Non-linear) ได้

Function	สูตร	ช่วงค่า	ข้อดี	ข้อเสีย	ใช้ที่ Layer ใด
ReLU	max(0, z)	[0, ∞)	เร็ว, ลด Vanishing Grad.	Dead ReLU ถ้า z<0 เสมอ	Hidden Layers (แนะนำ)
PReLU	max(αz, z) α เรียนรู้ได้	(-∞, ∞)	แก้ Dead ReLU	มี param เพิ่ม	Hidden Layers (ยืดหยุ่น)
Sigmoid	$1 / (1 + e^{-z})$	(0, 1)	ตีความเป็น prob. ได้	Vanishing Grad.	Output Binary Class
Tanh	$(e^{z} - e^{-z})/(e^{z} + e^{-z})$	(-1, 1)	Centered ที่ 0	Vanishing Grad. (น้อยกว่า)	Hidden (sequential data)
Softmax	$e^{z_i} / \sum e^{z_j}$	(0, 1) รวม=1	Output เป็น probability	ใช้เฉพาะ Output Layer	Output Multi-class
Linear / None	z	(-∞, ∞)	ใช้กับ Regression	ใช้ใน Output เท่านั้น	Output Regression

1.3 การทำให้เป็นมาตรฐาน (Normalization) — บทที่ 2.5–2.6

ปัญหา: Feature ที่มีช่วงค่าต่างกัน (เช่น อายุ 0–100, รายได้ 10,000–1,000,000) ทำให้โมเดลลำเอียง

วิธีแก้: ทำ Normalization ทุก Feature ให้อยู่ในช่วงเดียวกันก่อน Training

วิธี	สูตร	ช่วงผลลัพธ์	เหมาะกับ	NumPy Code
Min-Max Rescaling (รูปแบบที่ 1)	$x' = (x - x_{min}) / (x_{max} - x_{min})$	[0, 1]	ข้อมูลไม่มี outlier Distribution ไม่ทราบ	x_n = (x-x.min())/(x.max()-x.min())
Mean Normalization (รูปแบบที่ 2)	$x' = (x - \mu) / (x_{max} - x_{min})$	[-1, 1]	ต้องการ centered ที่ 0	x_n = (x-x.mean())/(x.max()-x.min())

Standardization (Z-score)	x' = (x - μ) / σ	μ=0, σ=1	ข้อมูล Gaussian dist. มี outlier ได้บ้าง	x_n = (x - x.mean()) / x.std()

1.4 แบบฝึกหัด Lab 1

ให้นักศึกษา implement และตอบคำถามต่อไปนี้

1. 1. เขียนฟังก์ชัน MSE และ Binary Cross-Entropy ด้วย NumPy (ไม่ใช้ Keras) แล้วทดสอบกับข้อมูล y_true = [1,0,1,1] และ y_pred = [0.9, 0.1, 0.8, 0.6]
2. 2. Plot กราฟ Activation Functions ทั้ง 5 ประเภทในช่วง z = [-5, 5] โดยใช้ Matplotlib ใส่ legend และชื่อแกน
3. 3. โหลด dataset Iris (sklearn.datasets.load_iris) แล้วทำ Min-Max Normalization ก่อนและหลัง normalization แสดงค่า min, max, mean ของแต่ละ feature
4. 4. อธิบายความแตกต่างระหว่าง MSE กับ MAE และระบุกรณีที่ควรใช้แต่ละอัน

LAB 2	การถดถอย (Regression) Linear, Multivariate & Logistic Regression (บทที่ 3)
วัตถุประสงค์	1) สร้างแบบจำลอง Linear Regression และ Multivariate Regression จาก scratch 2) เข้าใจ Gradient Descent และ Backpropagation เบื้องต้น 3) สร้าง Logistic Regression สำหรับการจำแนกประเภท
เวลา / Tools	3 ชั่วโมง | Python 3.10+, NumPy, Matplotlib, TensorFlow/Keras
ความรู้ที่ต้องมีก่อน	Lab 1 ผ่านแล้ว, เข้าใจ Calculus (Partial Derivative) เบื้องต้น

2.1 Linear Regression — บทที่ 3.1

เป้าหมาย: หาเส้นตรง $\hat{y} = w_1x + w_0$ ที่ minimise MSE กับข้อมูล

วิธีการปรับ weight: Gradient Descent ⟶ $w_{ne}w = w - \alpha \cdot (\partial MSE/\partial w)$ โดย α = Learning Rate

2.2 Multivariate Regression — บทที่ 3.2

ใช้ตัวแปรต้นหลายตัว: $\hat{y} = w_1x_1 + w_2x_2 + \ldots + w_nx_n + w_0$

ตัวอย่างในหนังสือ: ทำนายอุณหภูมิจาก feature หลายตัว ใช้ Learning Rate = 0.01, Epoch = 300

2.3 Logistic Regression — บทที่ 3.3

สำหรับ Classification: $\hat{y} = \sigma(wx + b)$ โดย σ คือ Sigmoid ⟶ output เป็น probability [0,1]

Decision Boundary: ถ้า $\hat{y} \geq 0.5$ ⟶ Class 1, ถ้า $\hat{y} < 0.5$ ⟶ Class 0

2.4 Multinomial Logistic Regression — บทที่ 3.4

สำหรับ Classification หลายคลาส: ใช้ Softmax แทน Sigmoid, ใช้ Categorical Cross-Entropy

2.5 แบบฝึกหัด Lab 2

Dataset: ใช้ Boston Housing หรือ California Housing จาก sklearn

5. 1. Implement Linear Regression ด้วย NumPy (ไม่ใช้ sklearn) อัปเดต weight ด้วย Gradient Descent 500 รอบ Plot กราฟ Loss vs Epoch
6. 2. ใช้ Multivariate Regression ทำนาย Housing Price จาก 5 features เปรียบเทียบผลลัพธ์ก่อน/หลัง Normalization

7. 3. สร้าง Logistic Regression ด้วย Keras (1 Dense layer, sigmoid) จำแนก Breast Cancer dataset (sklearn.datasets.load_breast_cancer) รายงาน Accuracy, Precision, Recall, F1
8. 4. เปรียบเทียบผลลัพธ์ Multinomial Logistic Regression (Softmax, 3 output) กับ Iris dataset

LAB 3	โครงข่ายประสาทเทียมและการเรียนรู้เชิงลึก Neural Networks & Deep Neural Networks (บทที่ 4)
วัตถุประสงค์	1) สร้าง Neural Network แบบ Feed-Forward ด้วย Keras 2) เข้าใจปัญหา Vanishing Gradient และวิธีแก้ 3) ทดลอง Dropout, Regularization, Batch Normalization 4) เปรียบเทียบ DNN กับ Multivariate Regression
เวลา / Tools	4 ชั่วโมง | Python 3.10+, NumPy, Matplotlib, TensorFlow/Keras
ความรู้ที่ต้องมีก่อน	Lab 1 และ Lab 2 ผ่านแล้ว, เข้าใจ Loss Function และ Activation Function

3.1 โครงสร้าง Neural Network — บทที่ 4.2

Input Layer ⟶ Hidden Layers ⟶ Output Layer

แต่ละ Node: $z = \Sigma(w_i x_i) + b$ จากนั้นผ่าน Activation: $a = f(z)$

3.2 Deep Neural Network (DNN) — บทที่ 4.3

DNN คือ Neural Network ที่มี Hidden Layers มากกว่า 2 ชั้น

กฎการกำหนด Layers (จากหนังสือ): เริ่มต้น 3–5 Hidden Layers ⟶ เพิ่มได้หากยังไม่ converge

ปัญหา	สาเหตุ	อาการ	วิธีแก้ (Keras)
Vanishing Gradient	ชั้นลึกเกินไป + Sigmoid/Tanh	Training Loss ไม่ลด, ช้ามาก	ใช้ ReLU, กด Layers ≤5
Overfitting	โมเดลซับซ้อนเกิน/ข้อมูลน้อย	Train acc สูง, Val acc ต่ำ	Dropout(0.2–0.5), L1/L2 reg
Exploding Gradient	Learning rate สูง/ชั้นลึก	Loss = NaN, weights ระเบิด	Gradient Clipping, BatchNorm
Slow Convergence	Learning rate ต่ำเกิน	Loss ลดช้ามาก	Adam optimizer, LR Scheduler

3.3 ResNet (Residual Network) — บทที่ 4.4–4.5

ไอเดีย: เพิ่ม Skip Connection เพื่อแก้ Vanishing Gradient ใน Deep Networks

Forward ResNet: output = F(x) + x (บวก input โดยตรงกับ output ของชั้น)

Feedback ResNet: ส่งข้อมูลย้อนกลับมายังชั้นก่อนหน้า (เหมาะกับ sequential data)

3.4 แบบฝึกหัด Lab 3

Dataset: MNIST Handwritten Digits หรือ CIFAR-10 (tf.keras.datasets)

9. 1. สร้าง DNN 4 ชั้น (Input⟶128⟶64⟶32⟶Output) จำแนก MNIST บันทึก Train/Val accuracy per epoch แล้ว Plot Learning Curve
10. 2. ทดลองเปลี่ยน Activation Function (ReLU vs Sigmoid vs Tanh) ใน Hidden Layers — เปรียบเทียบ convergence speed
11. 3. เพิ่ม Dropout(0.3) หลังแต่ละ Hidden Layer แล้วสังเกตผลต่อ Overfitting (เปรียบเทียบ Train vs Val loss)
12. 4. เพิ่ม L1 Regularization (lambda=0.001) แล้วเปรียบเทียบกับ Dropout — แบบใดให้ผลดีกว่าบน Validation set?
13. 5. สร้าง ResNet แบบง่าย (2 residual blocks) แล้วเปรียบเทียบกับ plain DNN — อธิบายความแตกต่างในกราฟ Loss

LAB 4	สัญญาณ UWB และการประยุกต์ใช้กับ AI Ultra-Wideband Signals & AI Applications (บทที่ 5)
วัตถุประสงค์	1) เข้าใจหลักการทำงานของ Radar และ UWB signal 2) เข้าใจการประมวลผลสัญญาณ UWB เพื่อส่งเป็น Input ให้ Neural Network 3) เตรียมความพร้อมสู่กรณีศึกษา Ch7 (Breast Cancer) และ Ch8 (Gesture)
เวลา / Tools	2 ชั่วโมง (ภาคทฤษฎี + simulation) | Python 3.10+, NumPy, Matplotlib, TensorFlow/Keras
ความรู้ที่ต้องมีก่อน	Lab 1–3 ผ่านแล้ว, เข้าใจสัญญาณพื้นฐาน

4.1 หลักการ Radar — บทที่ 5.1

Radar ส่ง Electromagnetic wave ไปกระทบวัตถุ ⟶ วัดระยะ ตำแหน่ง ความเร็วจากสัญญาณสะท้อน
ข้อมูลที่วัดได้: (1) ระยะทาง (Range), (2) ทิศทาง (Angle), (3) ความเร็ว (Velocity via Doppler)

4.2 Ultra-Wideband (UWB) — บทที่ 5.3

UWB: สัญญาณที่มีแบนด์วิดท์ ≥ 500 MHz หรือ Fractional Bandwidth ≥ 20%
ข้อดี: ทะลุผ่านวัสดุได้ดี, ความละเอียดสูง, รบกวนสัญญาณอื่นน้อย ⟶ เหมาะสำหรับตรวจจับในร่างกาย/ใต้ดิน

4.3 กระบวนการส่งสัญญาณ UWB ⟶ Neural Network

#	ขั้นตอน	รายละเอียด	ใช้ใน	เครื่องมือ
1	ส่งสัญญาณ UWB	FPGA สร้าง Impulse UWB ⟶ ส่งออกทาง Tx Antenna	Ch7, Ch8, Ch14	FPGA (Ch5.4)
2	รับสัญญาณสะท้อน	Rx Antenna รับสัญญาณที่สะท้อนจากวัตถุ/เนื้อเยื่อ	Ch7, Ch8, Ch14	ADC + FPGA
3	Preprocessing	Filter noise, Windowing, หา Time-Domain waveform	ทุก Chapter	NumPy, SciPy
4	Feature Extraction	คำนวณ Peak-to-Peak, RMS, FFT features	Ch7: 17 class Ch8: 4 gestures	NumPy
5	Normalization	Min-Max หรือ Z-score ทุก feature	ทุก Chapter	sklearn / NumPy

6	Input ⟶ Neural Network	Matrix ขนาด [N_samples × N_features] ⟶ DNN	Ch7, Ch8	TensorFlow/Keras

4.4 แบบฝึกหัด Lab 4

สามารถใช้สัญญาณ sine wave จำลองแทน UWB signal จริง

14. 1. สร้าง simulated UWB pulse (Gaussian modulated sine) ด้วย NumPy แล้ว Plot waveform ในโดเมนเวลาและ FFT
15. 2. เพิ่ม Gaussian noise เข้าไปในสัญญาณ แล้วทดลอง filter ด้วย Butterworth bandpass filter (SciPy)
16. 3. Extract features จาก waveform: Peak, RMS, Mean, Std แล้วทำ Normalization
17. 4. (Bonus) สร้าง simple classifier DNN รับ features 10 ตัวจาก simulated UWB signal จำแนก 2 class (object / no object)

ส่วนที่ 2: Step-by-Step Implementation Guide

ส่วนนี้อธิบาย Python code ทีละขั้นตอน ครอบคลุมทุก Lab โดยอธิบายทุกบรรทัดเป็นภาษาไทย

Step-by-Step: Lab 1 — Activation Functions & Normalization

STEP 1 — **Import Libraries**
นำเข้า library ที่ต้องใช้ทั้งหมด

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler

# กำหนด style กราฟให้สวยงาม
plt.style.use('seaborn-v0_8-darkgrid')
plt.rcParams['figure.figsize'] = (12, 5)
```

STEP 2 — **สร้าง Activation Functions ด้วย NumPy**
เขียนฟังก์ชัน activation แต่ละประเภทจาก scratch

```
# ─── ฟังก์ชัน Activation แต่ละประเภท ──────────────────────────────
def relu(z):                    # ReLU: คืน z ถ้า z>0, คืน 0 ถ้า z<=0
    return np.maximum(0, z)

def prelu(z, alpha=0.01):       # PReLU: คืน alpha*z ถ้า z<0
    return np.where(z > 0, z, alpha * z)

def sigmoid(z):                 # Sigmoid: คืน 1/(1+e^-z)
    return 1 / (1 + np.exp(-z))

def tanh(z):                    # Tanh: ใช้ฟังก์ชันสำเร็จรูปของ NumPy
    return np.tanh(z)

def softmax(z):                 # Softmax: เปลี่ยน vector เป็น probability
    exp_z = np.exp(z - np.max(z))   # ลบ max เพื่อป้องกัน overflow
    return exp_z / exp_z.sum()
```

STEP 3 **Plot ฟังก์ชัน Activation**

วาดกราฟเปรียบเทียบ activation functions ทั้ง 5

```
z = np.linspace(-5, 5, 300)          # ช่วง z ตั้งแต่ -5 ถึง 5

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# ─── แผงซ้าย: ReLU, PReLU, Sigmoid, Tanh ──────────────────────────
ax = axes[0]
ax.plot(z, relu(z),    label='ReLU',    linewidth=2, color='#1a4a57')
ax.plot(z, prelu(z),   label='PReLU',   linewidth=2, color='#c0392b')
ax.plot(z, sigmoid(z), label='Sigmoid', linewidth=2, color='#f0b429')
ax.plot(z, tanh(z),    label='Tanh',    linewidth=2, color='#27ae60')
ax.axhline(0, color='gray', linewidth=0.8, linestyle='--')
ax.axvline(0, color='gray', linewidth=0.8, linestyle='--')
ax.set_title('Activation Functions', fontsize=14, fontweight='bold')
ax.set_xlabel('z'); ax.set_ylabel('f(z)'); ax.legend()

# ─── แผงขวา: Softmax บน vector ตัวอย่าง ──────────────────────────
ax2 = axes[1]
scores = np.array([2.0, 1.0, 0.5, 3.0, -1.0])
probs  = softmax(scores)
ax2.bar(range(len(probs)), probs, color=['#1a4a57','#c0392b','#f0b429','#27ae60','#8e44ad'])
ax2.set_title('Softmax Output (Probability Distribution)', fontsize=14, fontweight='bold')
ax2.set_xlabel('Class'); ax2.set_ylabel('Probability')
for i, p in enumerate(probs):
    ax2.text(i, p + 0.01, f'{p:.3f}', ha='center', fontsize=10)

plt.tight_layout(); plt.savefig('activation_functions.png', dpi=150); plt.show()
```

STEP 4 **ทำ Normalization เปรียบเทียบ 3 วิธี**

แสดงผลก่อนและหลัง normalization

```
# ─── สร้างข้อมูลตัวอย่างที่มีช่วงต่างกัน ──────────────────────────
np.random.seed(42)
```

```
data = {
    'อายุ (ปี)':      np.random.randint(20, 70, 100).astype(float),
    'รายได้ (บาท)':  np.random.randint(15000, 200000, 100).astype(float),
    'BMI':             np.random.uniform(18, 35, 100),
}
import pandas as pd
df = pd.DataFrame(data)
print('=== ก่อน Normalization ==='); print(df.describe().round(2))

# ─── วิธีที่ 1: Min-Max Rescaling ──────────────────────────────
def min_max(x):
    return (x - x.min()) / (x.max() - x.min())

# ─── วิธีที่ 2: Mean Normalization ─────────────────────────────
def mean_norm(x):
    return (x - x.mean()) / (x.max() - x.min())

# ─── วิธีที่ 3: Z-score Standardization ────────────────────────
def zscore(x):
    return (x - x.mean()) / x.std()

df_minmax = df.apply(min_max)
df_mean   = df.apply(mean_norm)
df_zscore = df.apply(zscore)

print('\n=== หลัง Min-Max ==='); print(df_minmax.describe().round(3))
print('\n=== หลัง Z-score ==='); print(df_zscore.describe().round(3))
```

Step-by-Step: Lab 3 — Deep Neural Network ด้วย Keras

STEP 1	**โหลดและเตรียมข้อมูล MNIST** โหลด dataset ตัวเลขลายมือ และ normalize

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt

# ─── โหลด MNIST ──────────────────────────────────────────────────────────────

# X_train: 60,000 รูป ขนาด 28×28 pixels (0–255)
# y_train: label 0–9 (ตัวเลข 10 คลาส)
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()

print(f'Train: {X_train.shape}, Test: {X_test.shape}')
print(f'Labels: {np.unique(y_train)}')

# ─── Normalize pixel 0–255 ⟶ 0.0–1.0 (Min-Max) ──────────────────────────────
X_train = X_train.astype('float32') / 255.0
X_test  = X_test.astype('float32')  / 255.0

# ─── Flatten 28×28 ⟶ 784 ด้วยกัน ──────────────────────────────────────────
X_train_flat = X_train.reshape(-1, 28*28)   # shape: (60000, 784)
X_test_flat  = X_test.reshape(-1, 28*28)    # shape: (10000, 784)
```

STEP 2	**สร้างโครงสร้าง DNN** กำหนด layers และ activation functions

```
# ─── สร้าง DNN 4 Hidden Layers ──────────────────────────────────────────────

model = keras.Sequential([
```

```
    # Input Layer: รับ 784 features
    layers.Input(shape=(784,)),

    # Hidden Layer 1: 256 nodes, ReLU (แนะนำสำหรับ Hidden Layers)
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.3),              # ปิด 30% nodes สุ่มทุก step ⟶ ลด Overfitting

    # Hidden Layer 2: 128 nodes
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.3),

    # Hidden Layer 3: 64 nodes
    layers.Dense(64, activation='relu'),

    # Hidden Layer 4: 32 nodes
    layers.Dense(32, activation='relu'),

    # Output Layer: 10 nodes (10 คลาส), Softmax ⟶ probability
    layers.Dense(10, activation='softmax'),
], name='MNIST_DNN')

model.summary()  # แสดงสรุปโครงสร้างและจำนวน parameters
```

STEP 3 — **Compile โมเดล**

กำหนด optimizer, loss function และ metrics

```
# ─── Compile: กำหนดวิธีการ train ──────────────────────────────────────

model.compile(
    optimizer='adam',                    # Adam: adaptive learning rate
    loss='sparse_categorical_crossentropy', # label เป็น integer (0-9)
    metrics=['accuracy']                 # ติดตาม accuracy ระหว่าง train
)

# ─── Early Stopping: หยุด train เมื่อ val_loss ไม่ลดลง ──────────────
early_stop = keras.callbacks.EarlyStopping(
    monitor='val_loss',   # ติดตาม validation loss
    patience=5,           # รอ 5 epoch ก่อนหยุด
```

```
    restore_best_weights=True  # ใช้ weights ที่ดีที่สุด
)
```

STEP 4	**Train โมเดล** ฝึกสอนโครงข่ายด้วยข้อมูล training

```
# ─── Train ──────────────────────────────────────────────────────────────────

history = model.fit(
    X_train_flat, y_train,      # ข้อมูลและ label สำหรับ training
    epochs=50,                  # จำนวนรอบสูงสุด (Early Stopping จะหยุดก่อน)
    batch_size=256,             # ปรับ weights ทุก 256 samples
    validation_split=0.15,      # ใช้ 15% เป็น validation set
    callbacks=[early_stop],     # ใช้ Early Stopping
    verbose=1
)

print(f'\nBest epoch stopped at: {len(history.history["loss"])} epochs')
```

STEP 5	**ประเมินผลและ Plot Learning Curve** วิเคราะห์ผลลัพธ์การ train

```
# ─── Evaluate บน Test Set ──────────────────────────────────────────────────

test_loss, test_acc = model.evaluate(X_test_flat, y_test, verbose=0)
print(f'Test Accuracy: {test_acc:.4f} ({test_acc*100:.2f}%)')
print(f'Test Loss:     {test_loss:.4f}')

# ─── Plot Learning Curve ──────────────────────────────────────────────────

fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# กราฟ Loss
axes[0].plot(history.history['loss'],     label='Train Loss', color='#1a4a57', lw=2)
axes[0].plot(history.history['val_loss'], label='Val Loss',   color='#c0392b', lw=2, ls='--')
```

```
axes[0].set_title('Loss Curve', fontweight='bold'); axes[0].legend()
axes[0].set_xlabel('Epoch'); axes[0].set_ylabel('Loss')

# กราฟ Accuracy
axes[1].plot(history.history['accuracy'],     label='Train Acc', color='#1a4a57', lw=2)
axes[1].plot(history.history['val_accuracy'], label='Val Acc',   color='#c0392b', lw=2, ls='--')
axes[1].set_title('Accuracy Curve', fontweight='bold'); axes[1].legend()
axes[1].set_xlabel('Epoch'); axes[1].set_ylabel('Accuracy')

plt.suptitle(f'DNN MNIST  |  Test Acc: {test_acc:.4f}', fontsize=13, fontweight='bold')
plt.tight_layout(); plt.savefig('dnn_learning_curve.png', dpi=150); plt.show()
```

STEP 6	Confusion Matrix และ Classification Report วิเคราะห์ผลรายคลาส

```
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns

# ─── ทำนายผลบน Test Set ──────────────────────────────────────────────
y_pred_proba = model.predict(X_test_flat)    # shape: (10000, 10) ⟶ probability
y_pred = np.argmax(y_pred_proba, axis=1)     # เลือก class ที่มี prob สูงสุด

# ─── Classification Report ──────────────────────────────────────────
print(classification_report(y_test, y_pred, digits=4))

# ─── Confusion Matrix ──────────────────────────────────────────────
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=range(10), yticklabels=range(10))
plt.title('Confusion Matrix — MNIST DNN', fontweight='bold')
plt.xlabel('Predicted'); plt.ylabel('Actual')
plt.tight_layout(); plt.savefig('confusion_matrix.png', dpi=150); plt.show()
```

เกณฑ์การประเมิน (Evaluation Rubric)

Lab	หัวข้อประเมิน	ดีเยี่ยม (A) 90–100%	ดี (B) 75–89%	พอใช้ (C) 60–74%	ต้องปรับปรุง (D) < 60%
Lab 1	Activation + Normalization Code (40%) คำอธิบายทฤษฎี (30%) ผล Visualization (30%)	Code ถูกต้อง อธิบายครบ กราฟสวยงาม	Code ถูกต้อง อธิบายส่วนใหญ่	Code ถูกต้องแต่ไม่อธิบาย	Code ผิด/ไม่ครบ
Lab 2	Regression จาก scratch (50%) เปรียบเทียบผล (30%) วิเคราะห์ (20%)	Implement ครบ เปรียบเทียบถูกต้อง	Implement ครบ แต่วิเคราะห์น้อย	Implement บางส่วน	ไม่ implement
Lab 3	DNN Keras (40%) Learning Curve (30%) Confusion Matrix (30%)	Accuracy >98% กราฟครบ วิเคราะห์ลึก	Accuracy >95% กราฟครบ	Accuracy >90% กราฟบางส่วน	Accuracy <90% หรือไม่มีกราฟ
Lab 4	UWB Simulation (40%) Feature Extraction (30%) Bonus Classifier (30%)	Signal + FFT + classifier ครบ	Signal + FFT ครบ	Signal เท่านั้น	ไม่มี output

⚠ Lab Report: ส่ง .ipynb (Jupyter Notebook) พร้อม Markdown อธิบายแต่ละส่วน + Output ที่รันแล้ว ภายใน 1 สัปดาห์หลัง Lab