import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.applications import (

    VGG16, InceptionV3, ResNet50, MobileNetV2

)

# ---------------------------

# 1. LeNet (custom implementation)

# ---------------------------

def build\_lenet(input\_shape=(32, 32, 1), num\_classes=10):

    model = models.Sequential([

        layers.Conv2D(6, (5, 5), activation='relu', input\_shape=input\_shape),

        layers.AveragePooling2D(),

        layers.Conv2D(16, (5, 5), activation='relu'),

        layers.AveragePooling2D(),

        layers.Flatten(),

        layers.Dense(120, activation='relu'),

        layers.Dense(84, activation='relu'),

        layers.Dense(num\_classes, activation='softmax')

    ])

    return model

lenet = build\_lenet()

lenet.summary()

# ---------------------------

# 2. AlexNet (simplified version, custom)

# ---------------------------

def build\_alexnet(input\_shape=(227, 227, 3), num\_classes=1000):

    model = models.Sequential([

        layers.Conv2D(96, (11, 11), strides=4, activation='relu', input\_shape=input\_shape),

        layers.MaxPooling2D((3, 3), strides=2),

        layers.Conv2D(256, (5, 5), padding='same', activation='relu'),

        layers.MaxPooling2D((3, 3), strides=2),

        layers.Conv2D(384, (3, 3), padding='same', activation='relu'),

        layers.Conv2D(384, (3, 3), padding='same', activation='relu'),

        layers.Conv2D(256, (3, 3), padding='same', activation='relu'),

        layers.MaxPooling2D((3, 3), strides=2),

        layers.Flatten(),

        layers.Dense(4096, activation='relu'),

        layers.Dropout(0.5),

        layers.Dense(4096, activation='relu'),

        layers.Dropout(0.5),

        layers.Dense(num\_classes, activation='softmax')

    ])

    return model

alexnet = build\_alexnet()

alexnet.summary()

# ---------------------------

# 3. VGG16 (predefined in Keras Applications)

# ---------------------------

vgg16 = VGG16(weights=None, classes=1000)  # load with random weights

vgg16.summary()

# ---------------------------

# 4. GoogLeNet (InceptionV3 as replacement)

# ---------------------------

inception = InceptionV3(weights=None, classes=1000)

inception.summary()

# ---------------------------

# 5. ResNet (ResNet50 as example)

# ---------------------------

resnet = ResNet50(weights=None, classes=1000)

resnet.summary()

Code ảnh

import numpy as np

import matplotlib.pyplot as plt

# Generate some dummy data for demonstration

# In a real scenario, you would load your actual dataset here

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.mnist.load\_data()

# Preprocess the data

x\_train = x\_train.astype('float32') / 255.0

x\_test = x\_test.astype('float32') / 255.0

x\_train = np.expand\_dims(x\_train, -1)

x\_test = np.expand\_dims(x\_test, -1)

y\_train = tf.keras.utils.to\_categorical(y\_train, 10)

y\_test = tf.keras.utils.to\_categorical(y\_test, 10)

# Compile the LeNet model

lenet.compile(optimizer='adam',

              loss='categorical\_crossentropy',

              metrics=['accuracy'])

# Train the model

history = lenet.fit(x\_train, y\_train,

                    epochs=5,

                    batch\_size=32,

                    validation\_data=(x\_test, y\_test))

# Plot training accuracy and loss

plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.title('Model Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend()

plt.subplot(1, 2, 2)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.title('Model Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.legend()

plt.show()