import numpy as np

from tensorflow.keras.datasets import mnist, imdb

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Embedding, LSTM

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.wrappers.scikit\_learn import KerasClassifier

from sklearn.model\_selection import GridSearchCV

# Image Classification Task

def load\_mnist():

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

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

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

x\_train = x\_train.astype('float32') / 255.

x\_test = x\_test.astype('float32') / 255.

return x\_train, y\_train, x\_test, y\_test

def create\_cnn\_model(learning\_rate=0.001):

model = Sequential([

Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

MaxPooling2D((2, 2)),

Flatten(),

Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

model.compile(optimizer=Adam(learning\_rate=learning\_rate),

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

return model

def train\_cnn\_model(x\_train, y\_train, x\_test, y\_test, learning\_rate=0.001, batch\_size=32):

model = create\_cnn\_model(learning\_rate)

model.fit(x\_train, y\_train, epochs=5, batch\_size=batch\_size, validation\_data=(x\_test, y\_test))

return model

# Text Classification Task

def load\_imdb(max\_words=10000, max\_len=200):

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_words)

x\_train = pad\_sequences(x\_train, maxlen=max\_len)

x\_test = pad\_sequences(x\_test, maxlen=max\_len)

return x\_train, y\_train, x\_test, y\_test

def create\_rnn\_model(learning\_rate=0.001, embedding\_dim=128):

model = Sequential([

Embedding(input\_dim=10000, output\_dim=embedding\_dim, input\_length=200),

LSTM(units=128),

Dense(1, activation='sigmoid')

])

model.compile(optimizer=Adam(learning\_rate=learning\_rate),

loss='binary\_crossentropy',

metrics=['accuracy'])

return model

def train\_rnn\_model(x\_train, y\_train, x\_test, y\_test, learning\_rate=0.001, batch\_size=32):

model = create\_rnn\_model(learning\_rate)

model.fit(x\_train, y\_train, epochs=5, batch\_size=batch\_size, validation\_data=(x\_test, y\_test))

return model

# Hyperparameter Optimization

def optimize\_hyperparameters(model\_fn, params, x\_train, y\_train):

keras\_model = KerasClassifier(build\_fn=model\_fn)

grid\_search = GridSearchCV(estimator=keras\_model, param\_grid=params, cv=3)

grid\_search.fit(x\_train, y\_train)

best\_params = grid\_search.best\_params\_

return best\_params

# Main code

if \_\_name\_\_ == "\_\_main\_\_":

# Image Classification Task

x\_train\_img, y\_train\_img, x\_test\_img, y\_test\_img = load\_mnist()

cnn\_params = {'learning\_rate': [0.001, 0.01], 'batch\_size': [32, 64]}

best\_cnn\_params = optimize\_hyperparameters(create\_cnn\_model, cnn\_params, x\_train\_img, y\_train\_img)

print("Best CNN Parameters:", best\_cnn\_params)

cnn\_model = train\_cnn\_model(x\_train\_img, y\_train\_img, x\_test\_img, y\_test\_img, \*\*best\_cnn\_params)

# Text Classification Task

x\_train\_txt, y\_train\_txt, x\_test\_txt, y\_test\_txt = load\_imdb()

rnn\_params = {'learning\_rate': [0.001, 0.01], 'batch\_size': [32, 64]}

best\_rnn\_params = optimize\_hyperparameters(create\_rnn\_model, rnn\_params, x\_train\_txt, y\_train\_txt)

print("Best RNN Parameters:", best\_rnn\_params)

rnn\_model = train\_rnn\_model(x\_train\_txt, y\_train\_txt, x\_test\_txt, y\_test\_txt, \*\*best\_rnn\_params)