Q1 Sequential model for MNIST classification

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Flatten, Dense, Dropout

from tensorflow.keras.optimizers import Adam

model = Sequential([

Flatten(input\_shape=(28, 28)),

Dense(128, activation='relu'),

Dropout(0.3),

Dense(10, activation='softmax')

])

model.compile(optimizer=Adam(), loss='categorical\_crossentropy', metrics=['accuracy'])

model.summary()

2️⃣ CNN model for grayscale images

from tensorflow.keras.models import Sequential

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

cnn = Sequential([

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

MaxPooling2D((2,2)),

Conv2D(64, (3,3), activation='relu'),

Flatten(),

Dense(128, activation='relu'),

Dense(10, activation='softmax')

])

cnn.summary()

3️⃣ Add a Dense(64) before output in pre-trained model

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Assume pretrained\_model exists

pretrained\_model = Sequential([

Dense(128, activation='relu', input\_shape=(784,)),

Dense(10, activation='softmax')

])

# Remove last layer and add new ones

pretrained\_model.pop()

pretrained\_model.add(Dense(64, activation='relu'))

pretrained\_model.add(Dense(10, activation='softmax'))

pretrained\_model.compile(optimizer='adam', loss='categorical\_crossentropy')

4️⃣ Sigmoid vs Softmax

# Sigmoid: used for binary classification (2 classes)

# Softmax: used for multi-class (3 or more classes)

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Example for 3-class classification

model = Sequential([

Dense(16, activation='relu', input\_shape=(10,)),

Dense(3, activation='softmax') # softmax for 3 classes

])

5️⃣ Functional API model with two inputs

from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dense, Concatenate

input1 = Input(shape=(32,))

input2 = Input(shape=(32,))

merged = Concatenate()([input1, input2])

dense = Dense(64, activation='relu')(merged)

output = Dense(1, activation='sigmoid')(dense)

model = Model(inputs=[input1, input2], outputs=output)

model.summary()

6️⃣ Load and preprocess CIFAR-10

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

x\_train, x\_test = x\_train/255.0, x\_test/255.0

y\_train, y\_test = to\_categorical(y\_train), to\_categorical(y\_test)

7️⃣ Reshape dataset for CNN

import numpy as np

X = np.random.rand(1000, 28, 28)

X\_reshaped = X.reshape((1000, 28, 28, 1))

8️⃣ Split dataset 70/15/15

from sklearn.model\_selection import train\_test\_split

X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42)

9️⃣ Data augmentation

from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(

rotation\_range=15,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

horizontal\_flip=True

)

🔟 Visualize 5 random images

import matplotlib.pyplot as plt

import random

for i in range(5):

idx = random.randint(0, len(x\_train)-1)

plt.imshow(x\_train[idx])

plt.title(f"Label: {y\_train[idx]}")

plt.show()

1️⃣1️⃣ Train model with EarlyStopping

from tensorflow.keras.callbacks import EarlyStopping

early\_stop = EarlyStopping(monitor='val\_loss', patience=3, restore\_best\_weights=True)

model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_val, y\_val), callbacks=[early\_stop])

1️⃣2️⃣ ModelCheckpoint

from tensorflow.keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint('best\_model.h5', monitor='val\_accuracy', save\_best\_only=True)

model.fit(X\_train, y\_train, epochs=10, validation\_data=(X\_val, y\_val), callbacks=[checkpoint])

1️⃣3️⃣ Plot loss curves

import matplotlib.pyplot as plt

history = model.fit(X\_train, y\_train, validation\_data=(X\_val, y\_val), epochs=10)

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

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

plt.legend()

plt.show()

# Overfitting → validation loss starts increasing while training loss keeps decreasing.

1️⃣4️⃣ Evaluate model

loss, acc = model.evaluate(X\_test, y\_test)

print(f"Test Loss: {loss:.4f}, Test Accuracy: {acc:.4f}")

1️⃣5️⃣ Detect overfitting

train\_loss = [0.8, 0.5, 0.3, 0.2]

val\_loss = [0.9, 0.6, 0.4, 0.5]

if val\_loss[-1] > val\_loss[-2]:

print("Overfitting detected: validation loss increased.")

# Occurs when model learns noise from training data instead of generalizing.

1️⃣6️⃣ Freeze CNN layers except last Dense

for layer in cnn.layers[:-1]:

layer.trainable = False

cnn.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

1️⃣7️⃣ Extract intermediate layer output

from tensorflow.keras.models import Model

import numpy as np

intermediate\_layer\_model = Model(inputs=model.input, outputs=model.layers[2].output)

sample\_input = np.random.rand(1, 10)

intermediate\_output = intermediate\_layer\_model.predict(sample\_input)

print(intermediate\_output)

1️⃣8️⃣ Custom Mean Squared Error loss

import tensorflow as tf

def custom\_mse(y\_true, y\_pred):

return tf.reduce\_mean(tf.square(y\_true - y\_pred))

model.compile(optimizer='adam', loss=custom\_mse, metrics=['mse'])

1️⃣9️⃣ Save & load model

# Save in HDF5

model.save('model.h5')

# Save in SavedModel format

model.save('saved\_model/')

# Load

from tensorflow.keras.models import load\_model

model1 = load\_model('model.h5')

model2 = load\_model('saved\_model/')

2️⃣0️⃣ Manual softmax

import numpy as np

import tensorflow as tf

logits = np.array([2.0, 1.0, 0.1])

manual\_softmax = np.exp(logits) / np.sum(np.exp(logits))

tf\_softmax = tf.nn.softmax(logits).numpy()

print("Manual:", manual\_softmax)

print("TensorFlow:", tf\_softmax)