```
# =====================================================

# Deep Learning Questions with Answers (Short & Bug-Free)

# =====================================================

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras import Sequential, Model, Input

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

from tensorflow.keras.datasets import mnist, cifar10

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model_selection import train_test_split

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint


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

# Q1: Sequential model for MNIST

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

model1 = Sequential([

    Flatten(input_shape=(28,28)),

    Dense(128, activation='relu'),

    Dropout(0.3),

    Dense(10, activation='softmax')

])

model1.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])


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

# Q2: CNN model for grayscale (28,28,1)

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

model2 = 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')
])

# ------------------------------
# Q3: Add Dense(64) before output
# ------------------------------
pretrained = Sequential([
    Dense(128, activation='relu', input_shape=(100,)),
    Dense(10, activation='softmax')
])
pretrained.pop()  # remove last layer
pretrained.add(Dense(64, activation='relu'))
pretrained.add(Dense(10, activation='softmax'))
pretrained.compile(optimizer='adam', loss='categorical_crossentropy')

# ------------------------------
# Q4: Sigmoid vs Softmax
# ------------------------------
# Sigmoid: for binary classification
# Softmax: for multi-class classification (probabilities sum to 1)
model3 = Sequential([
    Dense(16, activation='relu', input_shape=(20,)),
    Dense(3, activation='softmax')
])

# ------------------------------
# Q5: Functional API with 2 inputs
# ------------------------------
```

```
inp1 = Input(shape=(32,))

inp2 = Input(shape=(32,))

x = Concatenate()([inp1, inp2])

x = Dense(64, activation='relu')(x)

out = Dense(1, activation='sigmoid')(x)

model4 = Model([inp1, inp2], out)


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

# Q6: Load CIFAR-10 dataset

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

(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, 10), to_categorical(y_test, 10)


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

# Q7: Reshape dataset for CNN

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

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

X_cnn = X.reshape(-1, 28, 28, 1)


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

# Q8: Split dataset 70/15/15

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

data = np.arange(1000).reshape(1000,1)

train, temp = train_test_split(data, test_size=0.3)

val, test = train_test_split(temp, test_size=0.5)


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

# Q9: Data Augmentation

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

datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True
)

# ------------------------------
# Q10: Visualize 5 images
# ------------------------------
for i in np.random.choice(range(len(x_train)), 5):
    plt.imshow(x_train[i])
    plt.title(f"Label: {np.argmax(y_train[i])}")
    plt.show()

# ------------------------------
# Q11: Train with EarlyStopping
# ------------------------------
early_stop = EarlyStopping(patience=3, restore_best_weights=True)
model1.fit(x_train.reshape(-1,28,28), y_train,
          validation_data=(x_test.reshape(-1,28,28), y_test),
          epochs=10, callbacks=[early_stop])

# ------------------------------
# Q12: ModelCheckpoint
# ------------------------------
checkpoint = ModelCheckpoint("best_model.h5", save_best_only=True, monitor="val_accuracy")
model1.fit(x_train.reshape(-1,28,28), y_train,
          validation_data=(x_test.reshape(-1,28,28), y_test),
          epochs=5, callbacks=[checkpoint])

# ------------------------------
# Q13: Plot loss curves
# ------------------------------
history = model1.history
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='val')
plt.legend(); plt.show()
# Overfitting: val_loss > train_loss and keeps increasing

# ------------------------------
# Q14: Evaluate on test data
# ------------------------------
loss, acc = model1.evaluate(x_test.reshape(-1,28,28), y_test)
print("Test Loss:", loss, "Test Accuracy:", acc)

# ------------------------------
# Q15: 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!")
# Explanation: val_loss increases while train_loss decreases.

# ------------------------------
# Q16: Freeze CNN except last layer
# ------------------------------
cnn = model2
for layer in cnn.layers[:-1]:
    layer.trainable = False
cnn.compile(optimizer='adam', loss='categorical_crossentropy')

# ------------------------------
# Q17: Extract intermediate output

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

intermediate_model = Model(inputs=model2.input, outputs=model2.layers[-2].output)

sample = np.random.rand(1,28,28,1)

feat = intermediate_model.predict(sample)

print("Intermediate features shape:", feat.shape)


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

# Q18: Custom MSE Loss

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

def custom_mse(y_true, y_pred):

    return tf.reduce_mean(tf.square(y_true - y_pred))

model2.compile(optimizer='adam', loss=custom_mse)


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

# Q19: Save/Load model

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

model2.save("model.h5")

model2.save("model_saved", save_format="tf")

loaded_h5 = tf.keras.models.load_model("model.h5", compile=False)

loaded_tf = tf.keras.models.load_model("model_saved", compile=False)


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

# Q20: Manual softmax

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

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

softmax_np = np.exp(logits)/np.sum(np.exp(logits))

softmax_tf = tf.nn.softmax(logits).numpy()

print("Numpy:", softmax_np, "TensorFlow:", softmax_tf)
```