ass6(maam cust)

```
# (a) Import libraries & load pre-trained CNN model
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np

# Load pre-trained VGG16 (without top layers) C:\Users\Safa Quadri\Downloads
weights_path = "vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5"
base_model = VGG16(weights=weights_path, include_top=False, input_shape=(32, 32, 3))

print("✅ Pre-trained VGG16 model loaded successfully.")

#  (b) Freeze parameters (lower convolutional layers) ======
for layer in base_model.layers:
    layer.trainable = False

print("✅ All base model layers frozen.")

# ====== (c) Add custom classifier layers (trainable part) ======
x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.3)(x)
predictions = Dense(10, activation='softmax')(x)

# Combine base model + classifier
model = Model(inputs=base_model.input, outputs=predictions)

print("✅ Custom classifier added successfully.")

# ====== (d) Load and preprocess dataset (CIFAR-10) ======
train_dir = "C:\Users\Safa Quadri\Downloads\archive.zip\cifar10\cifar10\train"
test_dir = "C:\Users\Safa Quadri\Downloads\archive.zip\cifar10\cifar10\test"

# Data generators for training and testing
train_datagen = ImageDataGenerator(rescale=1.0 / 255)
test_datagen = ImageDataGenerator(rescale=1.0 / 255)

train_batch_size = 5000
test_batch_size = 1000

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(32, 32),
    batch_size=train_batch_size,
    class_mode='categorical'
)

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(32, 32),
    batch_size=test_batch_size,
    class_mode='categorical'
)

# Extract data from generator
x_train, y_train = train_generator[0]
x_test, y_test = test_generator[0]

print("✅ Training and testing data loaded.")
print("Training samples:", len(x_train))
print("Testing samples:", len(x_test))

# ====== Compile and train model ======
model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])
```

```
print("\n🚀 Training started...")
history = model.fit(x_train, y_train, batch_size=64, epochs=10, validation_data=(x_test, y_test))
print("✅ Training completed.")

# ====== (e) Fine-tuning (optional improvement) ======
# Unfreeze last few convolutional layers for fine-tuning
for layer in base_model.layers[-4:]:
    layer.trainable = True

# Recompile with lower learning rate
model.compile(optimizer=Adam(learning_rate=1e-5), loss='categorical_crossentropy', metrics=['accuracy'])

print("\n🔧 Fine-tuning last layers...")
model.fit(x_train, y_train, batch_size=64, epochs=5, validation_data=(x_test, y_test))
print("✅ Fine-tuning completed.")

# ====== Evaluate model performance ======
loss, accuracy = model.evaluate(x_test, y_test)
print(f"\n🎯 Test Accuracy: {accuracy * 100:.2f}%")

# ====== Prediction & Visualization ======
predicted_value = model.predict(x_test)
labels = list(test_generator.class_indices.keys())

n = 890  # index of image to visualize
plt.imshow(x_test[n])
plt.title(f"Predicted: {labels[np.argmax(predicted_value[n])]} | Actual: {labels[np.argmax(y_test[n])]}")
plt.axis("off")
plt.show()

print("✅ Visualization complete.")
```
—---

ass6(gg)

```
# a. Import libraries
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
import matplotlib.pyplot as plt
import numpy as np

# b. Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test  = tf.keras.utils.to_categorical(y_test, 10)

# Simulate generator-like info display
print("Train samples:", len(x_train))
print("Test samples:", len(x_test))

# Class labels (same as generator's class_indices)
labels = ['airplane','automobile','bird','cat','deer',
          'dog','frog','horse','ship','truck']

# c. Load pre-trained CNN (MobileNetV2)
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(32, 32, 3))

# Freeze lower layers
for layer in base_model.layers:
    layer.trainable = False

# d. Add custom classifier
x = GlobalAveragePooling2D()(base_model.output)
x = Dense(128, activation='relu')(x)
output = Dense(10, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=output)

# e. Train classifier layers
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=3, batch_size=128, validation_split=0.1, verbose=1)
```

```
# f. Fine-tune entire model (simple one-line version)
base_model.trainable = True
model.fit(x_train, y_train, epochs=1, batch_size=128, verbose=1)

# g. Evaluate
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"\n✅ Test Accuracy: {test_acc:.4f}")

# h. Prediction and visualization
predicted_value = model.predict(x_test)
n = 100
plt.imshow(x_test[n])
plt.title(f"Predicted: {labels[np.argmax(predicted_value[n])]} | Actual: {labels[np.argmax(y_test[n])]}")
plt.axis('off')
plt.show()
```

—---