**VGG and TL**

import tensorflow as tf

from tensorflow.keras import layers, models, optimizers

from tensorflow.keras.applications import VGG16

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Load and preprocess CIFAR-10 data

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

# One-hot encode the labels

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Preprocess images (resize and normalize)

def preprocess\_images(images, labels):

images = tf.image.resize(images, (224, 224)) / 255.0 # Resize to 224x224 and normalize

return images, labels

batch\_size = 32

train\_ds = tf.data.Dataset.from\_tensor\_slices((x\_train, y\_train))

train\_ds = train\_ds.map(preprocess\_images).batch(batch\_size).prefetch(tf.data.AUTOTUNE)

test\_ds = tf.data.Dataset.from\_tensor\_slices((x\_test, y\_test))

test\_ds = test\_ds.map(preprocess\_images).batch(batch\_size).prefetch(tf.data.AUTOTUNE)

# Load the VGG16 model without the top layer

vgg\_base = VGG16(weights='imagenet', include\_top=False, input\_shape=(224, 224, 3))

vgg\_base.trainable = False # Freeze the base layers initially

# Add a custom classification head

model = models.Sequential([

vgg\_base,

layers.Flatten(),

layers.Dense(256, activation='relu'),

layers.Dropout(0.5),

layers.Dense(10, activation='softmax') # CIFAR-10 has 10 classes

])

# Compile the model with initial hyperparameters

learning\_rate = 0.001

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

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Define callbacks for early stopping and learning rate reduction

early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)

reduce\_lr = ReduceLROnPlateau(monitor='val\_loss', factor=0.5, patience=3, min\_lr=1e-6)

# Train the model

history = model.fit(train\_ds,

epochs=1,

validation\_data=test\_ds,

callbacks=[early\_stopping, reduce\_lr])

# Unfreeze some layers for fine-tuning

vgg\_base.trainable = True # Unfreeze the entire base model

# Recompile with a smaller learning rate for fine-tuning

model.compile(optimizer=optimizers.Adam(learning\_rate=0.0001),

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Fine-tune the model

history\_fine\_tune = model.fit(train\_ds,

epochs=1,

validation\_data=test\_ds,

callbacks=[early\_stopping, reduce\_lr])

# Evaluate the model

test\_loss, test\_acc = model.evaluate(test\_ds)

print(f"Final Test Accuracy: {test\_acc:.2f}")