```
'''

6. Object detection using Transfer Learning of CNN architectures
a. Load in a pre-trained CNN model trained on a large dataset
b. Freeze parameters (weights) in model's lower convolutional layers
c. Add custom classifier with several layers of trainable parameters
to model
d. Train classifier layers on training data available for task
e. Fine-tune hyper parameters and unfreeze more layers as needed

dataset: CIFAR
'''

'\n\n6. Object detection using Transfer Learning of CNN architectures\
na. Load in a pre-trained CNN model trained on a large dataset\nb.
Freeze parameters (weights) in model's lower convolutional layers\nc.
Add custom classifier with several layers of trainable parameters to
model\nd. Train classifier layers on training data available for task\
ne. Fine-tune hyper parameters and unfreeze more layers as needed\n\
ndataset: CIFAR\n'


import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np


train_dir = "cifar-10-img/train"
test_dir = "cifar-10-img/test"

train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
)

test_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
)

# here batch_size is the number of images in each batch
train_batch_size = 5000
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(32, 32),
    batch_size=train_batch_size,
    class_mode='categorical'
)
```

```
test_batch_size = 1000
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(32, 32),
    batch_size=test_batch_size,
    class_mode='categorical'
)
```

```
Found 40079 images belonging to 10 classes.
Found 9921 images belonging to 10 classes.
```

```
x_train, y_train =  train_generator[0]
x_test, y_test = test_generator[0]

print(len(x_train))
print(len(x_test))
```

```
5000
1000
```

```
# Load VGG16 without top layers
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))


for layer in base_model.layers:
   layer.trainable = False


x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(optimizer="adam", loss='categorical_crossentropy',
metrics=['accuracy'])


# Train the model
model.fit(x_train, y_train, batch_size=64, epochs=10,
validation_data=(x_test, y_test))
```

```
Epoch 1/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 11s 121ms/step - accuracy: 0.3572 - loss:
1.8131 - val_accuracy: 0.4490 - val_loss: 1.5981
Epoch 2/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 9s 118ms/step - accuracy: 0.5010 - loss:
```

```
1.4479 - val_accuracy: 0.4890 - val_loss: 1.4743
Epoch 3/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 10s 125ms/step - accuracy: 0.5402 - loss:
1.3291 - val_accuracy: 0.4990 - val_loss: 1.4006
Epoch 4/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 10s 122ms/step - accuracy: 0.5664 - loss:
1.2505 - val_accuracy: 0.5140 - val_loss: 1.3846
Epoch 5/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 9s 119ms/step - accuracy: 0.5904 - loss:
1.1826 - val_accuracy: 0.5130 - val_loss: 1.3859
Epoch 6/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 10s 122ms/step - accuracy: 0.6042 - loss:
1.1259 - val_accuracy: 0.5140 - val_loss: 1.3484
Epoch 7/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 10s 126ms/step - accuracy: 0.6288 - loss:
1.0859 - val_accuracy: 0.5280 - val_loss: 1.3302
Epoch 8/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 10s 130ms/step - accuracy: 0.6428 - loss:
1.0388 - val_accuracy: 0.5230 - val_loss: 1.3453
Epoch 9/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 10s 130ms/step - accuracy: 0.6564 - loss:
0.9986 - val_accuracy: 0.5300 - val_loss: 1.3561
Epoch 10/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 11s 134ms/step - accuracy: 0.6678 - loss:
0.9740 - val_accuracy: 0.5170 - val_loss: 1.3503

<keras.src.callbacks.history.History at 0x1bc4053b7c0>


base_model = VGG16(weights=weights_path, include_top=False,
input_shape=(32, 32, 3))
# freeze all layers first
for layer in base_model.layers:
   layer.trainable = False
# unfreeze last 4 layers of base model
for layer in base_model.layers[len(base_model.layers) - 4:]:
   layer.trainable = True
# fine-tuning hyper parameters
x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# Create the model
model = Model(inputs=base_model.input, outputs=predictions)
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])
# training fine tuned model
model.fit(x_train, y_train, batch_size=64, epochs=10,
validation_data=(x_test, y_test))
```

```
Epoch 1/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 38s 440ms/step - accuracy: 0.3822 - loss:
1.6932 - val_accuracy: 0.5230 - val_loss: 1.3218
Epoch 2/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 365ms/step - accuracy: 0.6086 - loss:
1.1222 - val_accuracy: 0.5950 - val_loss: 1.1808
Epoch 3/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 30s 377ms/step - accuracy: 0.6776 - loss:
0.9335 - val_accuracy: 0.6160 - val_loss: 1.1781
Epoch 4/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 30s 385ms/step - accuracy: 0.7086 - loss:
0.8310 - val_accuracy: 0.6440 - val_loss: 1.1234
Epoch 5/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 29s 373ms/step - accuracy: 0.7698 - loss:
0.6599 - val_accuracy: 0.6460 - val_loss: 1.0859
Epoch 6/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 30s 382ms/step - accuracy: 0.8046 - loss:
0.5610 - val_accuracy: 0.6310 - val_loss: 1.2287
Epoch 7/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 32s 400ms/step - accuracy: 0.8358 - loss:
0.4786 - val_accuracy: 0.6350 - val_loss: 1.3642
Epoch 8/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 31s 389ms/step - accuracy: 0.8642 - loss:
0.3969 - val_accuracy: 0.6320 - val_loss: 1.4653
Epoch 9/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 45s 571ms/step - accuracy: 0.8986 - loss:
0.2957 - val_accuracy: 0.6310 - val_loss: 1.6334
Epoch 10/10
79/79 ━━━━━━━━━━━━━━━━━━━━ 81s 557ms/step - accuracy: 0.9178 - loss:
0.2377 - val_accuracy: 0.6480 - val_loss: 1.5851

<keras.src.callbacks.history.History at 0x1bc491c1ae0>
```

```
import matplotlib.pyplot as plt
predicted_value = model.predict(x_test)

labels = list(test_generator.class_indices.keys())


n = 756
plt.imshow(x_test[n])
print("Preditcted: ",labels[np.argmax(predicted_value[n])])
print("Actual: ", labels[np.argmax(y_test[n])])
```