```
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
```

Pre processing img data

In [2]:

```
train_dir = "Datasets/cifar-10-img/train"
test_dir = "Datasets/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'
)
```

Selecting only first batch with 5000 images as train and test data

In [3]:

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

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

a. Load in a pre-trained CNN model trained on a large dataset

In [4]:

```
# 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))
```

b. Freeze parameters (weights) in model's lower convolutional layers

In [5]:

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

c. Add custom classifier with several layers of trainable parameters to model

In [6]:

```
x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = Dense(256, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(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'])
```

d. Train classifier layers on training data available for task

In [7]:

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

e. Fine-tune hyper parameters and unfreeze more layers as needed

In [8]:

```
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)
x = tf.keras.layers.Dropout(0.3)(x)
x = Dense(512, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(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))
```

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

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

```
n = 890
plt.imshow(x_test[n])
print("Preditcted: ",labels[np.argmax(predicted_value[n])])
print("Actual: ", labels[np.argmax(y_test[n])])
```