```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
"""
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

"""
```

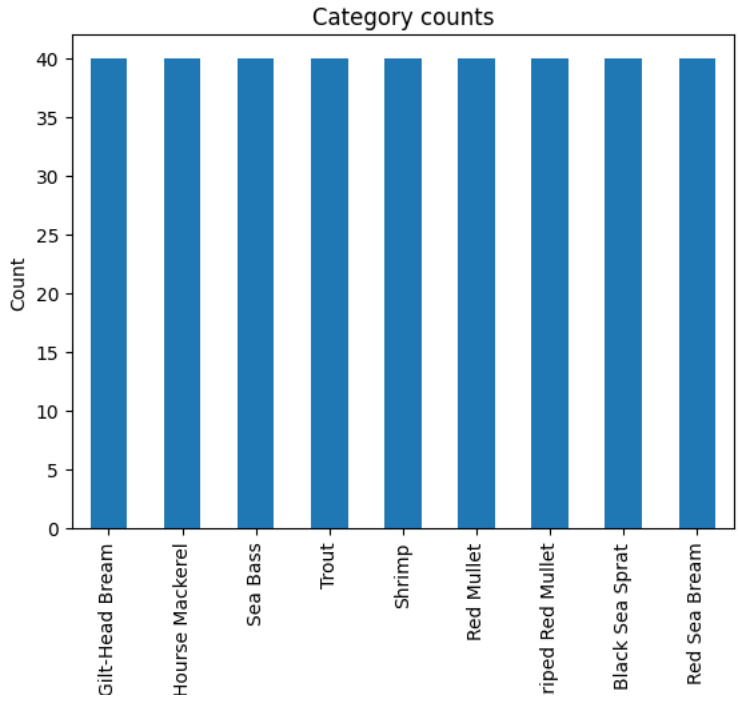
```
'\nimport os\nfor dirname, _, filenames in os.walk('/kaggle/input'):\n    for filen
ame in filenames:\n        print(os.path.join(dirname, filename))\n\n'
```

```
import numpy as np
import pandas as pd
import os
from pathlib import Path

# Image processing
import cv2

# Utility
from tqdm import tqdm

# Sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

# Charts
import matplotlib.pyplot as plt
import seaborn as sns

# Tensorflow
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import vgg16
from keras.layers import Dense, Dropout, BatchNormalization, Flatten
from keras.models import Model, load_model
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

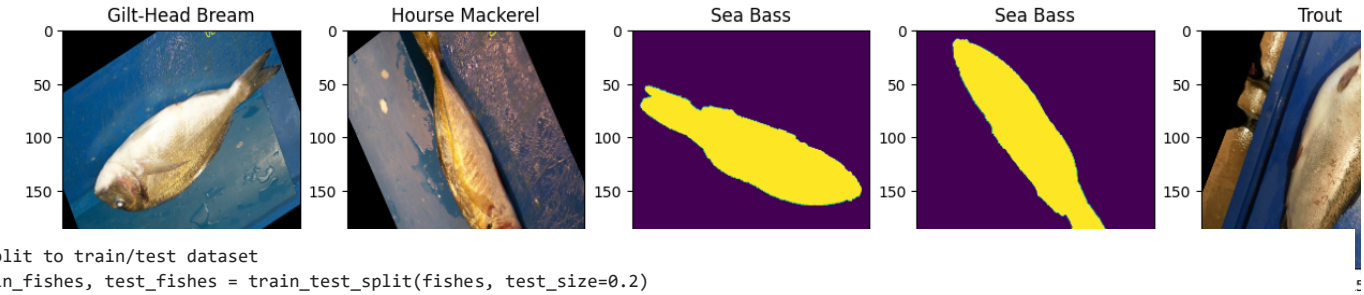
```
input_path = '../content/drive/MyDrive/Fish_Dataset'
# Create Series contains all file_paths
file_paths = pd.Series(list(Path(input_path).glob('**/*.png')), name='file_path')
# Convert poxispath to str
file_paths = file_paths.map(lambda file_path: str(file_path))
# Create Series contains all labels corresponding to file_path
labels = pd.Series(list(map(lambda file_path: file_path.split('/')[6], file_paths)), name='category')
# Create DataFrame cotains file_path and label
fishes = pd.concat([file_paths, labels], axis=1)
# Remove all files in GT folder
fishes = fishes[fishes.category.map(lambda cate: cate[-2:] != 'GT')]
fishes = fishes.sample(frac=1).reset_index(drop=True)
fishes.head()
```

	file_path	category
0	../content/drive/MyDrive/Fish_Dataset/Fish_Dat...	Gilt-Head Bream
1	../content/drive/MyDrive/Fish_Dataset/Fish_Dat...	Hourse Mackerel
2	../content/drive/MyDrive/Fish_Dataset/Fish_Dat...	Sea Bass
3	../content/drive/MyDrive/Fish_Dataset/Fish_Dat...	Sea Bass
4	../content/drive/MyDrive/Fish_Dataset/Fish_Dat...	Trout

```
fishes.category.value_counts().plot(kind='bar')
plt.title('Category counts')
plt.xlabel('Category')
plt.ylabel('Count')
plt.show()
```




```
def plot_first_six_image():
    fig = plt.figure(figsize=(20, 20))
    for i in range(6):
        row = fishes.loc[i]
        ax = fig.add_subplot(1, 6, i+1)
        img = plt.imread(row.file_path)
        img = cv2.resize(img, (224, 224))
        ax.set_title(row.category)
        ax.imshow(img)

plot_first_six_image()
```




```
# Split to train/test dataset
train_fishes, test_fishes = train_test_split(fishes, test_size=0.2)
print(train_fishes)
print(test_fishes)
```

```
                                          file_path            category
188  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...              Trout
106  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...           Sea Bass
352  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...  Striped Red Mullet
8    ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...             Shrimp
100  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...              Trout
..                                                 ...                ...
247  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...     Black Sea Sprat
267  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...       Red Sea Bream
309  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...     Black Sea Sprat
291  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...       Red Sea Bream
83   ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...           Sea Bass

[288 rows x 2 columns]
                                          file_path         category
51   ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...  Gilt-Head Bream
42   ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...           Shrimp
324  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...       Red Mullet
32   ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...       Red Mullet
61   ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...         Sea Bass
..                                                 ...              ...
184  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...  Black Sea Sprat
284  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...    Red Sea Bream
20   ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...           Shrimp
186  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...            Trout
328  ../content/drive/MyDrive/Fish_Dataset/Fish_Dat...         Sea Bass

[72 rows x 2 columns]
```

```
train_gen = ImageDataGenerator(validation_split=0.2)
test_gen = ImageDataGenerator()
train_batches = train_gen.flow_from_dataframe(dataframe=train_fishes,
                                              x_col='file_path',
                                              y_col='category',
                                              class_mode='categorical',
                                              batch_size=64,
                                              target_size=(224, 224),
                                              subset='training',
                                              color_mode='rgb')
valid_batches = train_gen.flow_from_dataframe(dataframe=train_fishes,                          x_col='file_path',
                                              y_col='category',
                                              class_mode='categorical',
                                              batch_size=64,
                                              target_size=(224, 224),
                                              subset='validation',
                                              color_mode='rgb')
# In test_batches, we should set shuffle=False to match predicted ouputs and grouth-true labels later
test_batches = test_gen.flow_from_dataframe(dataframe=test_fishes,  x_col='file_path',
                                            y_col='category',
                                            class_mode='categorical',
                                            batch_size=64,
                                            target_size=(224, 224),
                                            color_mode='rgb',
                                            shuffle=False)
```

```
Found 231 validated image filenames belonging to 9 classes.
Found 57 validated image filenames belonging to 9 classes.
Found 72 validated image filenames belonging to 9 classes.
```

```
base_model = vgg16.VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
if os.path.exists('../models') == False:
    os.mkdir('../models')
base_model.save('../models/vgg16_as_feature_extractor.h5')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_n
58889256/58889256 [==============================] - 0s 0us/step
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3000: UserWarning: You are saving your model as an HDF5 file v
  saving_api.save_model(
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty
```

```
base_model = load_model('../models/vgg16_as_feature_extractor.h5')
for layer in base_model.layers:
    layer.trainable = False
base_model.summary()
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it manually.
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808
```

```
 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

=================================================================
Total params: 14714688 (56.13 MB)
Trainable params: 0 (0.00 Byte)
Non-trainable params: 14714688 (56.13 MB)
_________________________________________________________________
```

```
last_layer = base_model.get_layer('block5_pool')
last_output = last_layer.output

x = Flatten()(last_output)
x = Dense(64, activation='relu', name='FC_1')(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)
x = Dense(9, activation='softmax', name='softmax')(x)

new_model = Model(inputs=base_model.input, outputs=x)
new_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 FC_1 (Dense)                (None, 64)                1605696

 batch_normalization (Batch  (None, 64)                256
 Normalization)

 dropout (Dropout)           (None, 64)                0

 softmax (Dense)             (None, 9)                 585

=================================================================
Total params: 16321225 (62.26 MB)
Trainable params: 1606409 (6.13 MB)
Non-trainable params: 14714816 (56.13 MB)
_________________________________________________________________
```

```
early_stopping = EarlyStopping(monitor='val_accuracy',
                               min_delta=0,
                               patience=3,
                               verbose=1,
                               mode='auto',
```

```
                               restore_best_weights=True)
new_model.compile(Adam(lr=0.0001), loss='categorical_crossentropy', metrics=['accuracy'])
history = new_model.fit_generator(generator=train_batches,
                        steps_per_epoch=train_batches.n // train_batches.batch_size,
                        validation_data=valid_batches,
                        validation_steps=valid_batches.n // valid_batches.batch_size,
                        epochs=10,
                        verbose=1,
                        callbacks=[early_stopping])
new_model.save('../models/fish_classifier_best_model.h5')
```

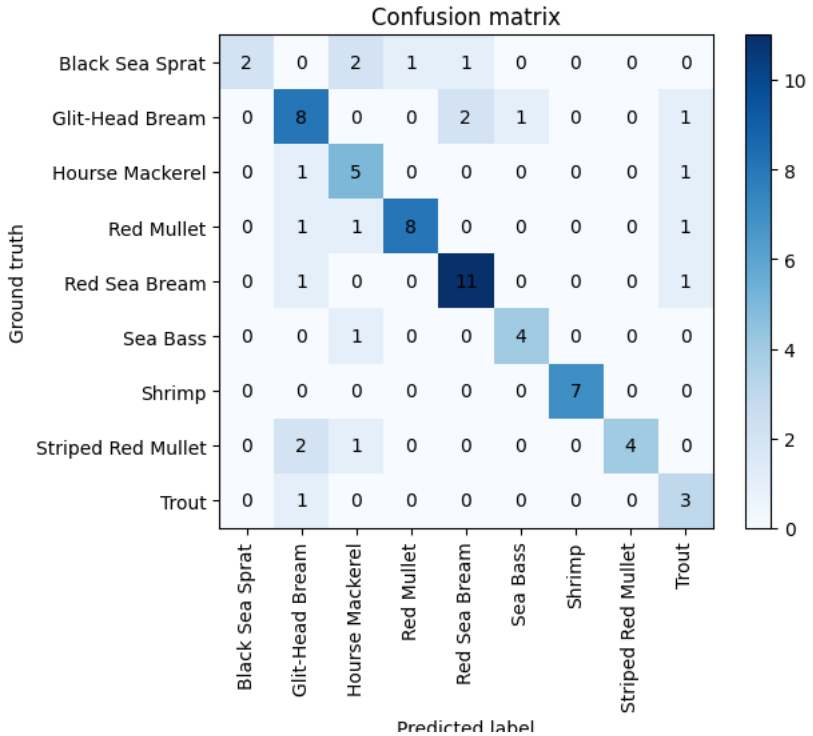
```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizer
<ipython-input-13-969f90e2b095>:8: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please
  history = new_model.fit_generator(generator=train_batches,
Epoch 1/10
3/3 [==============================] - ETA: 0s - loss: 2.3713 - accuracy: 0.3174 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 129s 43s/step - loss: 2.3713 - accuracy: 0.3174
Epoch 2/10
3/3 [==============================] - ETA: 0s - loss: 0.8071 - accuracy: 0.7605 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 105s 39s/step - loss: 0.8071 - accuracy: 0.7605
Epoch 3/10
3/3 [==============================] - ETA: 0s - loss: 0.6214 - accuracy: 0.8563 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 107s 33s/step - loss: 0.6214 - accuracy: 0.8563
Epoch 4/10
3/3 [==============================] - ETA: 0s - loss: 0.4630 - accuracy: 0.9042 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 105s 41s/step - loss: 0.4630 - accuracy: 0.9042
Epoch 5/10
3/3 [==============================] - ETA: 0s - loss: 0.3424 - accuracy: 0.9102 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 105s 41s/step - loss: 0.3424 - accuracy: 0.9102
Epoch 6/10
3/3 [==============================] - ETA: 0s - loss: 0.2870 - accuracy: 0.9688 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 122s 40s/step - loss: 0.2870 - accuracy: 0.9688
Epoch 7/10
3/3 [==============================] - ETA: 0s - loss: 0.2218 - accuracy: 0.9701 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 105s 32s/step - loss: 0.2218 - accuracy: 0.9701
Epoch 8/10
3/3 [==============================] - ETA: 0s - loss: 0.1793 - accuracy: 0.9880 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 108s 41s/step - loss: 0.1793 - accuracy: 0.9880
Epoch 9/10
3/3 [==============================] - ETA: 0s - loss: 0.1747 - accuracy: 0.9948 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 121s 39s/step - loss: 0.1747 - accuracy: 0.9948
Epoch 10/10
3/3 [==============================] - ETA: 0s - loss: 0.1547 - accuracy: 0.9896 WARNING:tensorflow:Early stopping conditioned on m
3/3 [==============================] - 123s 41s/step - loss: 0.1547 - accuracy: 0.9896
```

```
def plot_learning_curves(value='loss'):
    plt.plot(history.history[value], label='train')
    plt.plot(history.history['val_' + value], label='val')
    plt.xlabel('epochs')
    plt.ylabel(value + 'val_')
    plt.legend()
    plt.show(plot_learning_curves)
```

```
y_hat = new_model.predict(test_batches, verbose=1)
# Compare first 10 predicted value and ground truth label
print('First 10 predicted value from model: ', np.argmax(y_hat[:10], axis=1))
print('First 10 ground truth label: ', test_batches.labels[:10])
```

```
2/2 [==============================] - 49s 6s/step
First 10 predicted value from model:  [5 1 0 6 4 2 3 1 4 2]
First 10 ground truth label:  [5, 1, 0, 6, 4, 2, 0, 4, 1, 3]
```

```
category = ['Black Sea Sprat', 'Glit-Head Bream', 'Hourse Mackerel', 'Red Mullet',
           'Red Sea Bream', 'Sea Bass', 'Shrimp', 'Striped Red Mullet', 'Trout']
conf_mat = confusion_matrix(y_true=test_batches.labels,
                            y_pred=np.argmax(y_hat, axis=1))
plt.imshow(conf_mat, cmap=plt.cm.Blues)
plt.colorbar()
indexes = np.arange(len(category))
for i in indexes:
    for j in indexes:
      plt.text(j, i, conf_mat[i, j],
                horizontalalignment='center',
                verticalalignment='center')
plt.xticks(indexes, category, rotation=90)
plt.xlabel('Predicted label')
plt.yticks(indexes, category)
plt.ylabel('Ground truth')
plt.title('Confusion matrix')
plt.show()
```




```
results = new_model.evaluate(test_batches, verbose=1)
print('Test accuracy: %.3f' % (results[1] * 100), '%')
print('Test loss: %.3f' % results[0])
```

```
2/2 [==============================] - 46s 5s/step - loss: 0.9159 - accuracy: 0.7222
Test accuracy: 72.222 %
Test loss: 0.916
```