TM470 Project - Automating the Identification of UK Coarse Fish

In [ ]:
```
import tensorflow as tf
import kaggle
import pandas as pd
import os
import numpy as np
import sklearn
from sklearn.model_selection import StratifiedShuffleSplit #scikit-learn.org
from sklearn.model_selection import train_test_split
import pathlib
import matplotlib
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw
import xml.etree.ElementTree as et # https://docs.python.org/3/library/xml.etree
from tensorflow.python.client import device_lib #for detection of devices
import glob as glob # Searches for certain files
# for model
import keras
from tensorflow.keras import Sequential, optimizers, metrics, layers
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Conv2D, MaxPooling2D, Rescaling
import json
```

In [ ]:
```
# TensorFlow version
print(tf.__version__)
```

3 Is TF using GPU acceleration from inside python shell.

In [ ]:
```
# Is TF using GPU?
if tf.test.gpu_device_name():
    print('Default GPU device:{}'.format(tf.test.gpu_device_name()))
else:
    print("Please install GPU version of TF")
# Number of GPU's available
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
# Details of CPU and GPU from the device library (device_lib)
print(device_lib.list_local_devices())
```

AFFiNe dataset from Kaggle (list, download and unzip)

Classes taken out of original dataset

Aspius aspius Asp, Carassius gibelio (Carp Prussian), Lepomis gibbosus Pumpkinseed, Neogobius fluviatilis Goby (monkey), Neogobius kessleri Goby (bighead), Neogobius melanostomus (Goby Round), Rhodeus amarus Bitterling (European), Vimba vimba Vimba, Leuciscus leuciscus Dace, Gasterosteus aculeatus Stickleback.

Assigning filepaths

In [ ]:
```
# AFFiNe dataset from Kaggle placed in Jupyter folder
# https://www.kaggle.com/datasets/jorritvenema/affine
```

In [ ]:
```
datasetPath = 'UK AFFiNe Split/Main'
testDatasetPath = 'UK AFFiNe Split/Test'
```

In [ ]:
```
datasetPath, testDatasetPath
```

In [ ]:
```
# Assigning dataset path to pathlib
dat_dir = pathlib.Path(datasetPath).with_suffix('')
print(dat_dir)
```

In [ ]:
```
test_dat_dir = pathlib.Path(testDatasetPath).with_suffix('')
print(test_dat_dir)
```

In [ ]:
```
# Number of images in Main dataset
image_count = len(list(dat_dir.glob('*/*.jpg'))) # is this how datasetPath shoul
print(image_count)
```

In [ ]:
```
# Number of images Test in dataset
image_count = len(list(test_dat_dir.glob('*/*.jpg'))) # is this how datasetPath
print(image_count)
```

In [ ]:
```
```

Get class names and bound box information from XML files using the parser

In [ ]:
```
# Reading the information in the XML files and extracting names/bounding box inf
path = (dat_dir)
filelist = []
list1 = list()
list2 = list()
for root, dirs, files in os.walk(path):
    for file in files:
        if not file.endswith('.xml'):
            continue
        filelist.append(os.path.join(root, file))
for file in filelist:
    root = et.parse(file).getroot() # get the root of the xml
# Get class names
    for className in root.findall('.//object'):
        class_name = className.find('name').text
        data = np.array([class_name])
    list1.append(data)
# Get bounding box information
    for bndBox in root.findall('.//object'):
        bounding_box = bndBox.find('bndbox').text
        xmin = int(bndBox.find('./bndbox/xmin').text)
        ymin = int(bndBox.find('./bndbox/ymin').text)
        xmax = int(bndBox.find('./bndbox/xmax').text)
        ymax = int(bndBox.find('./bndbox/ymax').text)
        data2 = np.array([xmin,ymin,xmax,ymax])
    list2.append(data2)
```

In [ ]:
```
print(len(list1))
```

Create dataframe (using relative paths, class names and bound box details from XML)

In [ ]:
```
#list(base_dir.glob('*/*.jpg'))
filepaths = list(dat_dir.glob(r'**/*.jpg'))
classnames = list1#list(map(lambda x: os.path.split(os.path.split(x)[0])[1], fil
boundboxes = list2

filepaths = pd.Series(filepaths, name='Filepath').astype(str)#str
classnames = pd.Series(classnames, name='Class Name')
boundboxes = pd.Series(boundboxes, name='Boundbox')

dataframe1 = pd.concat([filepaths , classnames, boundboxes] , axis=1)
dataframe1
```

Class counts

In [ ]:
```
# species counts for each class with UK AFFiNe

dataframe1['Class Name'].value_counts()
```

Images count

In [ ]:
```
# Number of images in dataset and dataframe1
image_count = len(list(dat_dir.glob('*/*.jpg')))
image_count_df = len(dataframe1)
print(image_count)
print(image_count_df)
```

Creating the datasets (looking at stratified shuffle split)

(not working)

In [ ]:
```
#X = list(dat_dir.glob(r'**/*.jpg'))
#y = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], X))

#sss = StratifiedShuffleSplit(n_splits=5, test_size=0.2, train_size=0.2, random_
#sss.get_n_splits(X, y)

#print(sss)

#for i, (train_index, test_index) in enumerate(sss.split(X, y)):
#    print(f"Fold {i}:")
#    print(f"  Train: index={train_index}")
#    print(f"  Test:  index={test_index}")

#X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random
#X_train, X_val, y_train, y_val =train_test_split(X_train,y_train,test_size=0.25
```

In [ ]:
```
#print(X_train)
```

Creating datasets using image dataset from directory

Assigning batch and image sizes

In [ ]:
```
# Image size
batch_size=32
img_height=256
img_width=256
img_size=(img_height, img_width,3)
num_classes = 20
```

In [ ]:
```
img_size
```

In [ ]:
```
# Create the training dataset
train_dataset = tf.keras.utils.image_dataset_from_directory(
  dat_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  shuffle=True,
  image_size=(img_height, img_width),
  #color_mode='rgb',
  batch_size=batch_size)
```

In [ ]:
```
# Create the validation dataset
val_dataset = tf.keras.utils.image_dataset_from_directory(
  dat_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  shuffle=True,
  image_size=(img_height, img_width),
  #color_mode='rgb',
  batch_size=batch_size)
```

In [ ]:
```
# Creating test dataset
test_dataset = tf.keras.utils.image_dataset_from_directory(
  test_dat_dir,
  #validation_split=0.6,
  #subset="validation",
  #seed='123',
  shuffle = True,
  image_size=(img_height, img_width),
  #color_mode='rgb',
  batch_size=batch_size)
```

In [ ]:
```
# Assign the class names
class_names = test_dataset.class_names#test_dataset
#class_names=list1
print(class_names)
```

In [ ]:
```
# Next two cells for testing
sample_imgs, sample_labels = test_dataset.as_numpy_iterator().next()
sample_imgs.shape, sample_labels.shape
```

Show sample images

In [ ]:
```
# testing using sample label - to try debug final evaluation
plt.figure(figsize=(10,10))
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.imshow(sample_imgs[i].astype("uint8")) #images[i].numpy().astype("uint8"
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.title(class_names[sample_labels[i]])
plt.show()
```

My model (based on TM358 EMA model)

Normalisation layer

In [ ]:
```
# Creating the normalisation layer
norm_layer = layers.Normalization(input_shape=(img_size))
norm_layer.adapt(train_dataset.map(lambda x, y: x))
```

Augmenting the data

In [ ]:
```
# Creating an augmented subset
data_augmentation = tf.keras.Sequential([
#layers.RandomRotation(0.25),#- worse accuracy (but what about overfitting?) cau
#layers.RandomZoom(height_factor=0.2), # testing cause of model fit freeze
layers.RandomFlip(mode='horizontal'),
layers.RandomFlip(mode='vertical'),# worse but not having it results in overfitt
])

aug_train_dataset = train_dataset.map(lambda x, y: (data_augmentation(x, trainin
num_parallel_calls=tf.data.AUTOTUNE)
aug_train_dataset = aug_train_dataset.prefetch(buffer_size=tf.data.AUTOTUNE)
```

Model creation

In [ ]:
```
ada = tf.keras.optimizers.Adam(learning_rate=0.0001)#learning_rate=0.0001,or 3e-
def build_model():
      model = Sequential([
          #norm_layer,
          Conv2D(filters=64, kernel_size=(3,3), padding="same",input_shape=(img_
          Conv2D(filters=64, kernel_size=(3,3), padding="same", activation="relu
          MaxPooling2D(pool_size=(2,2)),
          Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="rel
          Conv2D(filters=128, kernel_size=(3,3), padding="same", activation="rel
          MaxPooling2D(pool_size=(2,2)),
          Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="rel
          Conv2D(filters=256, kernel_size=(3,3), padding="same", activation="rel
          MaxPooling2D(pool_size=(2,2)),
          Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="rel
          Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="rel
          MaxPooling2D(pool_size=(2,2)),
          Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="rel
          Conv2D(filters=512, kernel_size=(3,3), padding="same", activation="rel
          MaxPooling2D(pool_size=(2,2)),
          Dropout(0.5),
          Flatten(),
          Dense(512, activation='relu'),# num_classes*25 = 500
          Dropout(0.5),
          Dense(20, activation='softmax')#num_classes * 1.5 or 20 * 1.
      ])
      model.compile(
          optimizer=ada,#'adam',#learning_rate=0.0001,or 3e-4
          loss='sparse_categorical_crossentropy',#sparse_categorical_crossentrop
          metrics=['accuracy']
          )
      return model
```

In [ ]:
```
# Build the model using the build_model function
model=build_model()
```

In [ ]:
```
# Show a summary of the model
model.summary()
```

Model training

In [ ]:
```
# Train the model
#with tf.device("/cpu:0"):
#with tf.device("/device:GPU:0"):
hist=model.fit(
aug_train_dataset,# aug_train_dataset
validation_data=val_dataset,
verbose=1,
#shuffle=True,
epochs=50)
```

Plot accuracy and loss

In [ ]:
```
# Plotting training loss and accuracy as well as validation loss and accuracy ov
hist_dict = hist.history

# obtain the accuracy and loss of the training set and verification set in the r
train_acc = hist.history['accuracy']
val_acc = hist.history['val_accuracy']
train_loss = hist.history['loss']
val_loss = hist.history['val_loss']

epochs = range(1, len(train_acc)+1)
plt.plot(epochs, train_acc, 'bo', label = 'Training acc')
plt.plot(epochs, val_acc, 'r', label = 'Validation acc')
plt.title('Training and validation accuracy')
plt.legend() # show legend
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.show()
plt.figure()

plt.plot(epochs, train_loss, 'bo', label = 'Training loss')
plt.plot(epochs, val_loss, 'r', label = 'Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

Evaluate on test dataset

In [ ]:
```
model.evaluate(test_dataset, return_dict=True)
```

In [ ]:
```
sample_predictions = model(sample_imgs)
# View the true and predicted labels of sample images
plt.figure(figsize=(15,15))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(sample_imgs[i].astype("uint8"))
    #plt.imshow(sample_imgs[i])
    p_class = np.argmax(sample_predictions[i])
    a_class = sample_labels[i]# a_class = np.argmax(sample_labels[i]) ##np.argma
    #plt.title(f"P: {class_names[p_class]}\n(A: {class_names[a_class]})",
    plt.title(f"P: {class_names[p_class]}\n(A: {class_names[a_class]})",# class_
    color=("green" if p_class == a_class else "red"))
    plt.axis("off")
plt.show()
```

Save model

In [ ]:
```
model.save('saved_model.h5')

with open('saved_model_history.json', 'w') as f:
    json.dump(hist.history, f)

print('model saved')
```

Load model

In [ ]:
```
model = tf.keras.models.load_model('saved_model.h5')

with open('saved_model_history.json') as f:
    example_history = json.load(f)

print('model loaded successfully')
```

Convert model to TF Lite and save as TF Lite model

(https://www.tensorflow.org/lite/models/convert/convert_models)

In [ ]:
```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open("model.tflite", 'wb') as f:
  f.write(tflite_model)
```