TM470 Project - Automating the Identification of UK Coarse Fish

In [1]:
```
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 [2]:
```
# TensorFlow version
print(tf.__version__)
```

```
2.6.0
```

3 Is TF using GPU acceleration from inside python shell.

In [3]:
```
# 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())
```

```
Default GPU device:/device:GPU:0
Num GPUs Available:  1
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 1448653302864907109
, name: "/device:GPU:0"
device_type: "GPU"
memory_limit: 6925844480
locality {
  bus_id: 1
  links {
  }
}
incarnation: 4296756830094896333
physical_device_desc: "device: 0, name: NVIDIA GeForce GTX 1070 Ti, pci bus id:
0000:06:00.0, compute capability: 6.1"
]
```

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 [4]:
```
# AFFiNe dataset from Kaggle placed in Jupyter folder
# https://www.kaggle.com/datasets/jorritvenema/affine
```

In [5]:
```
datasetPath = 'UK AFFiNe Split/Main'
testDatasetPath = 'UK AFFiNe Split/Test'
```

In [6]:
```
datasetPath, testDatasetPath
```

Out[6]:
```
('UK AFFiNe Split/Main', 'UK AFFiNe Split/Test')
```

In [7]:
```
# Assigning dataset path to pathlib
dat_dir = pathlib.Path(datasetPath).with_suffix('')
print(dat_dir)
```

```
UK AFFiNe Split\Main
```

In [8]:
```
test_dat_dir = pathlib.Path(testDatasetPath).with_suffix('')
print(test_dat_dir)
```

```
UK AFFiNe Split\Test
```

In [9]:
```
# Number of images in Main dataset
image_count = len(list(dat_dir.glob('*/*.jpg'))) # is this how datasetPath shoul
print(image_count)
```

```
5362
```

In [10]:
```
# Number of images Test in dataset
image_count = len(list(test_dat_dir.glob('*/*.jpg'))) # is this how datasetPath
print(image_count)
```

600

In [ ]:

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

In [11]:
```
# 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 [12]:
```
print(len(list1))
```

5362

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

In [13]:
```
#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
```

Out[13]:

	Filepath	Class Name	Boundbox
0	UK AFFiNe Split\Main\Abramis brama\017f3e53-e0...	[Abramis brama]	[20, 20, 515, 187]
1	UK AFFiNe Split\Main\Abramis brama\022e3ceb-a8...	[Abramis brama]	[20, 20, 283, 916]
2	UK AFFiNe Split\Main\Abramis brama\02436d5d-04...	[Abramis brama]	[20, 20, 697, 244]
3	UK AFFiNe Split\Main\Abramis brama\08282fac-ad...	[Abramis brama]	[19, 20, 789, 400]
4	UK AFFiNe Split\Main\Abramis brama\08ff200c-a6...	[Abramis brama]	[20, 20, 556, 214]
...	...	...	...
5357	UK AFFiNe Split\Main\Tinca tinca\f917c866-9230...	[Tinca tinca]	[13, 20, 772, 346]
5358	UK AFFiNe Split\Main\Tinca tinca\f96c7997-7f5d...	[Tinca tinca]	[20, 20, 725, 365]
5359	UK AFFiNe Split\Main\Tinca tinca\fbd1d021-5b64...	[Tinca tinca]	[20, 20, 651, 270]
5360	UK AFFiNe Split\Main\Tinca tinca\fd1a5781-0723...	[Tinca tinca]	[20, 20, 641, 269]
5361	UK AFFiNe Split\Main\Tinca tinca\fdef10a4-0a00...	[Tinca tinca]	[20, 20, 563, 318]

5362 rows × 3 columns

Class counts

In [14]:
```
# species counts for each class with UK AFFiNe

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

Out[14]:
```
[Cyprinus carpio]                559
[Barbus barbus]                  306
[Leuciscus idus]                 304
[Rutilus rutilus]                288
[Tinca tinca]                    286
[Scardinius erythrophthalmus]    281
[Esox lucius]                    281
[Sander lucioperca]              272
[Ctenopharyngodon idella]        263
[Acipenseridae]                  263
[Leuciscus cephalus]             252
[Silurus glanis]                 242
[Abramis brama]                  241
[Salmo trutta subsp. fario]      240
[Anguilla anguilla]              236
[Perca fluviatilis]              218
[Blicca bjoerkna]                214
[Carassius carassius]            212
[Gymnocephalus cernuus]          206
[Gobio gobio]                    198
Name: Class Name, dtype: int64
```

Images count

In [15]:
```
# 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)
```

```
5362
5362
```

Creating the datasets (looking at stratified shuffle split)

(not working)

In [16]:
```
#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 [17]:
```
#print(X_train)
```

Creating datasets using image dataset from directory

Assigning batch and image sizes

In [18]:
```
# Image size
batch_size=32
img_height=256
img_width=256
img_size=(img_height, img_width,3)
num_classes = 20
```

In [19]:
```
img_size
```

Out[19]:
```
(256, 256, 3)
```

In [20]:
```
# 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)
```

```
Found 5362 files belonging to 20 classes.
Using 4290 files for training.
```

In [21]:
```
# 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)
```

```
Found 5362 files belonging to 20 classes.
Using 1072 files for validation.
```

In [22]:
```
# 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)
```

```
Found 600 files belonging to 20 classes.
```

In [23]:
```
# Assign the class names
class_names = test_dataset.class_names#test_dataset
#class_names=list1
print(class_names)
```

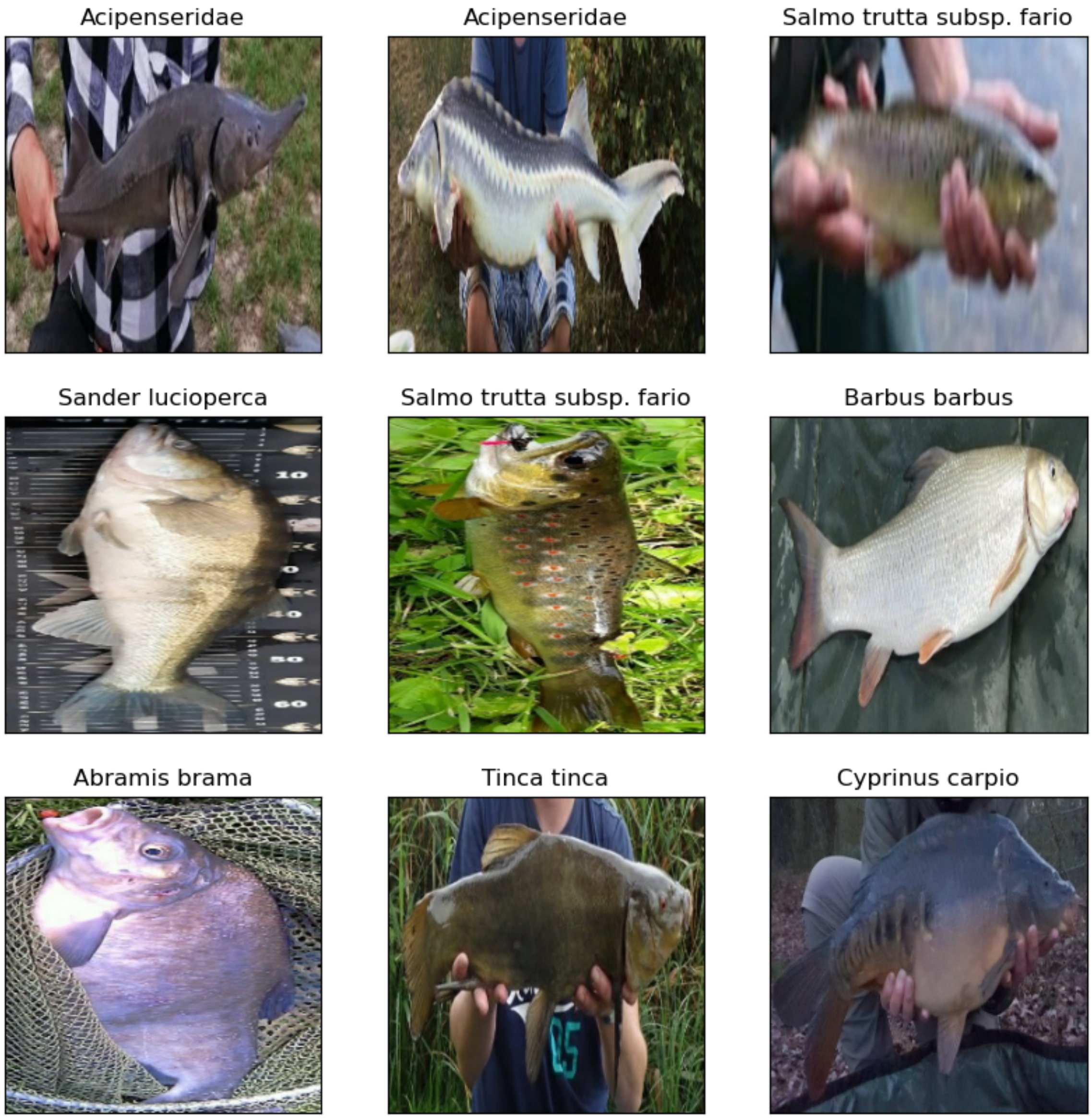
```
['Abramis brama', 'Acipenseridae', 'Anguilla anguilla', 'Barbus barbus', 'Blicca
bjoerkna', 'Carassius carassius', 'Ctenopharyngodon idella', 'Cyprinus carpio',
'Esox lucius', 'Gobio gobio', 'Gymnocephalus cernuus', 'Leuciscus cephalus', 'Le
uciscus idus', 'Perca fluviatilis', 'Rutilus rutilus', 'Salmo trutta subsp. fari
o', 'Sander lucioperca', 'Scardinius erythrophthalmus', 'Silurus glanis', 'Tinca
tinca']
```

In [24]:
```
# Next two cells for testing
sample_imgs, sample_labels = test_dataset.as_numpy_iterator().next()
sample_imgs.shape, sample_labels.shape
```

Out[24]:
```
((32, 256, 256, 3), (32,))
```

Show sample images

In [25]:
```
# 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 [26]:
```
# 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 [27]:
```
# 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 [28]:
```
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 [29]:
```
# Build the model using the build_model function
model=build_model()
```

In [30]:
```
# Show a summary of the model
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
normalization (Normalization (None, 256, 256, 3)       7
_________________________________________________________________
conv2d (Conv2D)              (None, 256, 256, 64)      1792
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 256, 256, 64)      36928
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 128, 128, 64)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 128, 128, 128)     73856
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 128, 128, 128)     147584
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 64, 64, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 64, 64, 256)       295168
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 64, 64, 256)       590080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 32, 32, 256)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 32, 32, 512)       1180160
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 32, 32, 512)       2359808
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 512)       0
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 16, 16, 512)       2359808
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 16, 16, 512)       2359808
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 8, 8, 512)         0
_________________________________________________________________
dropout (Dropout)            (None, 8, 8, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 32768)             0
_________________________________________________________________
dense (Dense)                (None, 512)               16777728
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 20)                10260
=================================================================
Total params: 26,192,987
Trainable params: 26,192,980
Non-trainable params: 7
_________________________________________________________________
```

Model training

In [31]:
```
# 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)
```

```
Epoch 1/50
135/135 [==============================] - 83s 459ms/step - loss: 2.9449 - accur
acy: 0.0932 - val_loss: 2.8537 - val_accuracy: 0.1474
Epoch 2/50
135/135 [==============================] - 58s 427ms/step - loss: 2.7612 - accur
acy: 0.1441 - val_loss: 2.6036 - val_accuracy: 0.1642
Epoch 3/50
135/135 [==============================] - 58s 428ms/step - loss: 2.6285 - accur
acy: 0.1755 - val_loss: 2.5154 - val_accuracy: 0.1940
Epoch 4/50
135/135 [==============================] - 58s 429ms/step - loss: 2.4918 - accur
acy: 0.2016 - val_loss: 2.3481 - val_accuracy: 0.2285
Epoch 5/50
135/135 [==============================] - 59s 432ms/step - loss: 2.3768 - accur
acy: 0.2490 - val_loss: 2.2609 - val_accuracy: 0.2556
Epoch 6/50
135/135 [==============================] - 60s 440ms/step - loss: 2.2308 - accur
acy: 0.2928 - val_loss: 2.1769 - val_accuracy: 0.2994
Epoch 7/50
135/135 [==============================] - 60s 444ms/step - loss: 2.1375 - accur
acy: 0.3135 - val_loss: 2.0061 - val_accuracy: 0.3368
Epoch 8/50
135/135 [==============================] - 59s 438ms/step - loss: 2.0154 - accur
acy: 0.3559 - val_loss: 1.9932 - val_accuracy: 0.3321
Epoch 9/50
135/135 [==============================] - 59s 437ms/step - loss: 1.9515 - accur
acy: 0.3776 - val_loss: 1.8626 - val_accuracy: 0.3918
Epoch 10/50
135/135 [==============================] - 61s 449ms/step - loss: 1.8394 - accur
acy: 0.4166 - val_loss: 1.7858 - val_accuracy: 0.4291
Epoch 11/50
135/135 [==============================] - 61s 452ms/step - loss: 1.7357 - accur
acy: 0.4527 - val_loss: 1.7292 - val_accuracy: 0.4431
Epoch 12/50
135/135 [==============================] - 61s 452ms/step - loss: 1.6011 - accur
acy: 0.4855 - val_loss: 1.6852 - val_accuracy: 0.4524
Epoch 13/50
135/135 [==============================] - 61s 452ms/step - loss: 1.5391 - accur
acy: 0.5091 - val_loss: 1.6023 - val_accuracy: 0.4963
Epoch 14/50
135/135 [==============================] - 61s 452ms/step - loss: 1.4473 - accur
acy: 0.5438 - val_loss: 1.4789 - val_accuracy: 0.5271
Epoch 15/50
135/135 [==============================] - 61s 452ms/step - loss: 1.3321 - accur
acy: 0.5830 - val_loss: 1.4302 - val_accuracy: 0.5373
Epoch 16/50
135/135 [==============================] - 61s 454ms/step - loss: 1.2107 - accur
acy: 0.6186 - val_loss: 1.4040 - val_accuracy: 0.5597
Epoch 17/50
135/135 [==============================] - 61s 454ms/step - loss: 1.1970 - accur
acy: 0.6340 - val_loss: 1.3581 - val_accuracy: 0.5728
Epoch 18/50
135/135 [==============================] - 61s 452ms/step - loss: 1.0969 - accur
acy: 0.6478 - val_loss: 1.3654 - val_accuracy: 0.5560
Epoch 19/50
135/135 [==============================] - 61s 452ms/step - loss: 1.0148 - accur
acy: 0.6786 - val_loss: 1.2768 - val_accuracy: 0.6045
Epoch 20/50
135/135 [==============================] - 61s 452ms/step - loss: 0.9523 - accur
acy: 0.6937 - val_loss: 1.3689 - val_accuracy: 0.5802
Epoch 21/50
135/135 [==============================] - 59s 438ms/step - loss: 0.8900 - accur
acy: 0.7161 - val_loss: 1.2978 - val_accuracy: 0.6073
```

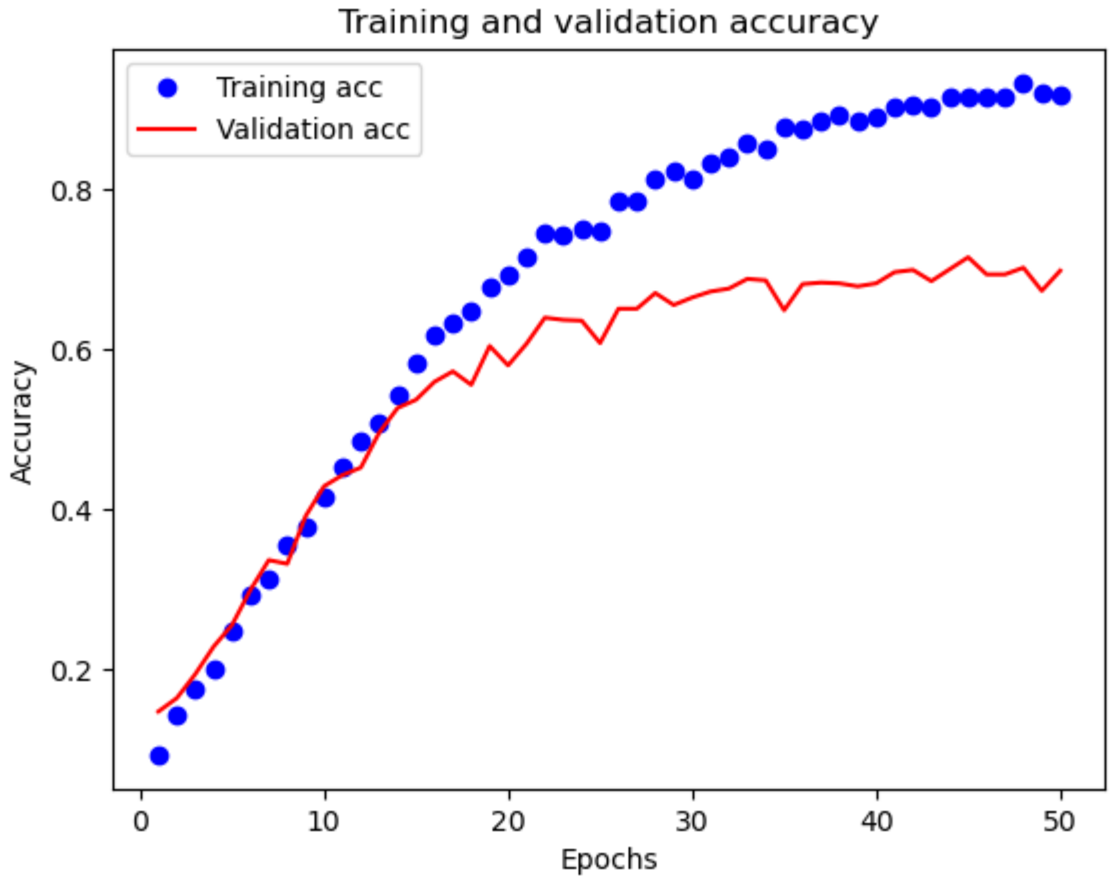
```
Epoch 22/50
135/135 [==============================] - 59s 438ms/step - loss: 0.8191 - accur
acy: 0.7462 - val_loss: 1.1989 - val_accuracy: 0.6399
Epoch 23/50
135/135 [==============================] - 62s 456ms/step - loss: 0.8152 - accur
acy: 0.7443 - val_loss: 1.1904 - val_accuracy: 0.6371
Epoch 24/50
135/135 [==============================] - 61s 454ms/step - loss: 0.7808 - accur
acy: 0.7503 - val_loss: 1.2069 - val_accuracy: 0.6362
Epoch 25/50
135/135 [==============================] - 60s 440ms/step - loss: 0.8041 - accur
acy: 0.7473 - val_loss: 1.3505 - val_accuracy: 0.6082
Epoch 26/50
135/135 [==============================] - 59s 434ms/step - loss: 0.6692 - accur
acy: 0.7867 - val_loss: 1.1454 - val_accuracy: 0.6511
Epoch 27/50
135/135 [==============================] - 60s 446ms/step - loss: 0.6590 - accur
acy: 0.7860 - val_loss: 1.2595 - val_accuracy: 0.6511
Epoch 28/50
135/135 [==============================] - 61s 454ms/step - loss: 0.5681 - accur
acy: 0.8133 - val_loss: 1.1256 - val_accuracy: 0.6707
Epoch 29/50
135/135 [==============================] - 62s 454ms/step - loss: 0.5397 - accur
acy: 0.8233 - val_loss: 1.2074 - val_accuracy: 0.6558
Epoch 30/50
135/135 [==============================] - 61s 453ms/step - loss: 0.5631 - accur
acy: 0.8138 - val_loss: 1.2252 - val_accuracy: 0.6651
Epoch 31/50
135/135 [==============================] - 61s 453ms/step - loss: 0.5143 - accur
acy: 0.8333 - val_loss: 1.1366 - val_accuracy: 0.6726
Epoch 32/50
135/135 [==============================] - 61s 453ms/step - loss: 0.4834 - accur
acy: 0.8413 - val_loss: 1.1721 - val_accuracy: 0.6763
Epoch 33/50
135/135 [==============================] - 62s 454ms/step - loss: 0.4463 - accur
acy: 0.8571 - val_loss: 1.1330 - val_accuracy: 0.6884
Epoch 34/50
135/135 [==============================] - 63s 463ms/step - loss: 0.4596 - accur
acy: 0.8517 - val_loss: 1.1031 - val_accuracy: 0.6866
Epoch 35/50
135/135 [==============================] - 61s 453ms/step - loss: 0.3730 - accur
acy: 0.8795 - val_loss: 1.2618 - val_accuracy: 0.6493
Epoch 36/50
135/135 [==============================] - 61s 453ms/step - loss: 0.3804 - accur
acy: 0.8767 - val_loss: 1.1409 - val_accuracy: 0.6819
Epoch 37/50
135/135 [==============================] - 61s 453ms/step - loss: 0.3615 - accur
acy: 0.8846 - val_loss: 1.2082 - val_accuracy: 0.6838
Epoch 38/50
135/135 [==============================] - 61s 453ms/step - loss: 0.3325 - accur
acy: 0.8939 - val_loss: 1.1784 - val_accuracy: 0.6828
Epoch 39/50
135/135 [==============================] - 62s 454ms/step - loss: 0.3439 - accur
acy: 0.8848 - val_loss: 1.2394 - val_accuracy: 0.6791
Epoch 40/50
135/135 [==============================] - 61s 453ms/step - loss: 0.3245 - accur
acy: 0.8914 - val_loss: 1.1921 - val_accuracy: 0.6828
Epoch 41/50
135/135 [==============================] - 62s 454ms/step - loss: 0.2901 - accur
acy: 0.9023 - val_loss: 1.2080 - val_accuracy: 0.6968
Epoch 42/50
135/135 [==============================] - 61s 454ms/step - loss: 0.2892 - accur
acy: 0.9061 - val_loss: 1.2535 - val_accuracy: 0.6996
```

```
Epoch 43/50
135/135 [==============================] - 62s 455ms/step - loss: 0.2797 - accur
acy: 0.9044 - val_loss: 1.2051 - val_accuracy: 0.6856
Epoch 44/50
135/135 [==============================] - 62s 455ms/step - loss: 0.2618 - accur
acy: 0.9163 - val_loss: 1.2337 - val_accuracy: 0.7006
Epoch 45/50
135/135 [==============================] - 62s 455ms/step - loss: 0.2539 - accur
acy: 0.9149 - val_loss: 1.2233 - val_accuracy: 0.7155
Epoch 46/50
135/135 [==============================] - 62s 456ms/step - loss: 0.2570 - accur
acy: 0.9149 - val_loss: 1.1696 - val_accuracy: 0.6940
Epoch 47/50
135/135 [==============================] - 62s 456ms/step - loss: 0.2434 - accur
acy: 0.9163 - val_loss: 1.2746 - val_accuracy: 0.6940
Epoch 48/50
135/135 [==============================] - 62s 455ms/step - loss: 0.2005 - accur
acy: 0.9336 - val_loss: 1.3154 - val_accuracy: 0.7024
Epoch 49/50
135/135 [==============================] - 61s 454ms/step - loss: 0.2351 - accur
acy: 0.9219 - val_loss: 1.3460 - val_accuracy: 0.6735
Epoch 50/50
135/135 [==============================] - 62s 454ms/step - loss: 0.2528 - accur
acy: 0.9186 - val_loss: 1.2670 - val_accuracy: 0.6987
```

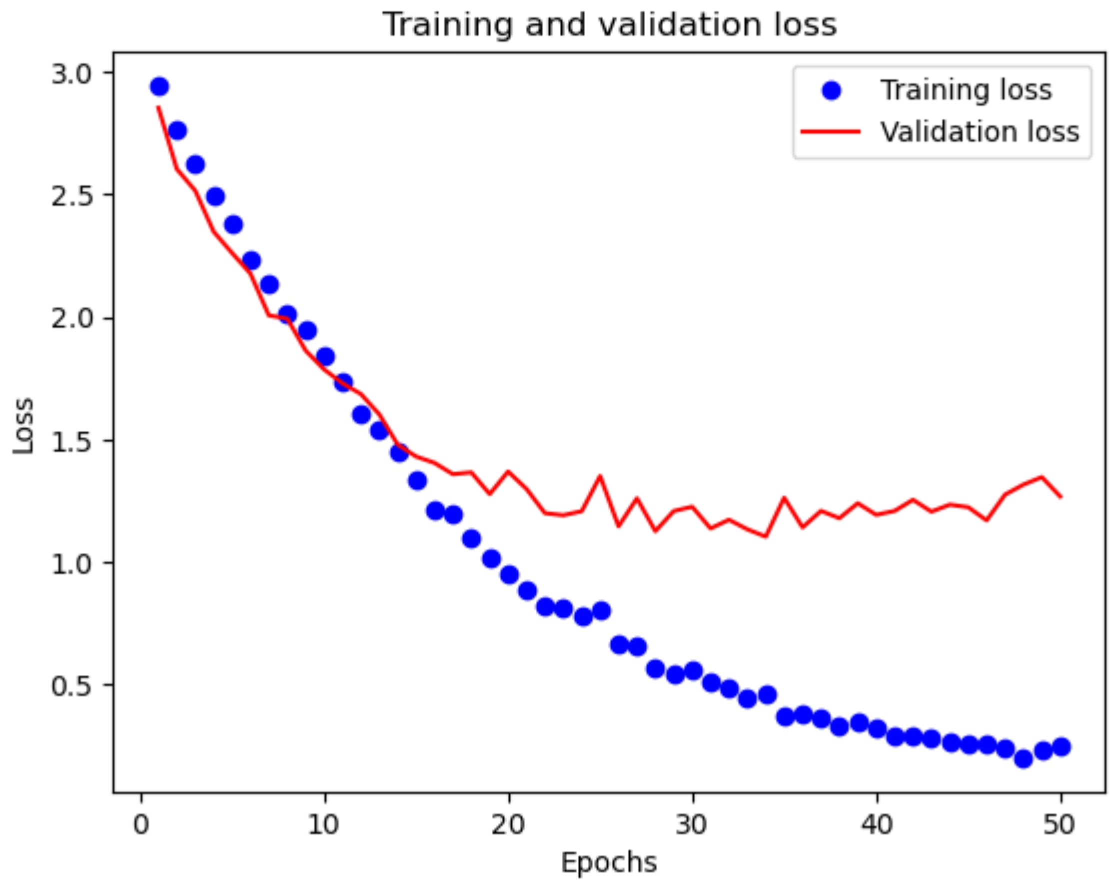
Plot accuracy and loss

In [32]:

```
# 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')
```



Out[32]: `Text(0, 0.5, 'Loss')`



Evaluate on test dataset

In [33]:
```
model.evaluate(test_dataset, return_dict=True)
```

```
19/19 [==============================] - 6s 323ms/step - loss: 1.2612 - accurac
y: 0.7067
```

Out[33]:
```
{'loss': 1.2612017393112183, 'accuracy': 0.7066666483879089}
```

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)
```