Fill in

- **FULL NAME**: Raf Engelen
- **STUDENT NUMBER**: r0901812
- **CLASS**: 3APP1

Practical 3: Deep Learning - Flying Objects Classifier

Score: ... /10

In the resources/images-folder you will find approximately 600 images of three fying objects (balloon, helicopter, and planes). Build a CNN in **Keras** to classify these images in three classes.

Important: The design of the CNN is described below (**don't deviate from this design**). We'll be using the newer, **layered model** approach (Data Preprocessing & Augmentation inside the model).

Code the cells below, and when asked, answer the questions by typing the answer in the cell at the appropiate place (ANSWER:...).

In [ ]:
```
# Place all your necessary imports in this cell
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras.preprocessing import image
from keras.utils import image_dataset_from_directory

from tensorflow import keras
from tensorflow.keras import optimizers
from tensorflow.keras import layers
from keras.preprocessing.image import ImageDataGenerator
```

A. Create the CNN

Create a sequential CNN with the following specifications (using tensorflow.keras, so as a list

of layers):

- Preprocessing layers:
  - a resizing layer, to image size of 128x128
  - a rescaling layer, to normalize the pixel values to a 0-1 range
  - an augmentation layer, to add random horizontal flips
  - an augmentation layer, to randomly translate the image by max 20% in each direction
  - an augmentation layer, to randomly zoom in with a max factor of 20%
- The CNN layers:
  - a convolutional layer with 16 filters of size 7x7; relu is the activation function; the input size is 128x128, and we don't want the convolutions to change the size of the image
  - a maxpooling layer with size 2x2
  - a dropout layer; drop 15% of the existing connections
  - a second convolutional layer with 64 filters of size 3x3; relu is the activation function; and we don't want the convolutions to change the size of the image
  - a maxpooling layer with size 2x2
  - a dropout layer; drop 15% of the existing connections
  - a layer to convert the multi-dimensional output into a one-dimensional array
  - a fully connected layer with 256 neurons and activation relu
  - another fully connected layer with 128 neurons and activation relu
  - a final layer with softmax

In [ ]:
```
# Use this cell to create a sequential model with a list of layers

model = Sequential()
model = tf.keras.Sequential([
  #a resizing layer, to image size of 128x128
  layers.Resizing(128, 128),
  #a rescaling layer, to normalize the pixel values to a 0-1 range
  layers.Rescaling(1./255),
  #an augmentation layer, to add random horizontal flips
  layers.RandomFlip("horizontal"),
  #an augmentation layer, to randomly translate the image by max 20% in each direct
  layers.RandomTranslation(0.2,0.2),
  #an augmentation layer, to randomly zoom in with a max factor of 20%
  layers.RandomZoom(0.2),

  #CNN layers
  #a convolutional layer with 16 filters of size 7x7; relu is the activation functi
  layers.Conv2D(16, (7, 7), input_shape = (128, 128, 3), activation = 'relu'),
  # a maxpooling layer with size 2x2
  layers.MaxPooling2D((2, 2)),
  # a dropout layer; drop 15% of the existing connections
  layers.Dropout(0.15),
  #a second convolutional layer with 64 filters of size 3x3; relu is the activation
  layers.Conv2D(64, (3, 3), activation="relu"),
```

```
  # a maxpooling layer with size 2x2
  layers.MaxPooling2D((2, 2)),
  # a dropout layer; drop 15% of the existing connections
  layers.Dropout(0.15),
  #a layer to convert the multi-dimensional output into a one-dimensional array
  layers.Flatten(),
  #a fully connected layer with 256 neurons and activation relu
  layers.Dense(256, activation="relu"),
  #another fully connected layer with 128 neurons and activation relu
  layers.Dense(128, activation="relu"),
  #a final layer with softmax
  layers.Dense(activation="sigmoid", units=1)
])
```

B. Compile the model

Compile the model: use Adam as your optimizer, with a learning rate of 0.001, and a **sparse** categorical crossentropy.

In [ ]:
```
# Use this cell to compile the model
model.compile(
    optimizer = optimizers.Adam(learning_rate=0.001),
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
    )
```

C. Create a training, validation, and test set

Use the **image_dataset_from_directory** method to create a training, validation, and test dataset. **Do not** use an ImageDataGenerator.

Make sure you create the training and validation set from the 'resources/images/train/' folder, using an 80-20% split, with a seed of 5. The test set, is just from the 'resources/images /test/' folder. All datasets, use a batch size of 16, and the labels should be taken from the parent folder, using a **sparse encoding**.

In [ ]:
```
# Use this cell to create your training, validation, and test set

# Set the parameters for your data
batch_size = 16
image_size = (64, 64)
validation_split = 0.2

# Create the training dataset from the 'train' directory
train_ds = image_dataset_from_directory(
    directory='./resources/images/train',
    labels='inferred',
    label_mode='binary',
    batch_size=batch_size,
    image_size=image_size,
```

```
    validation_split=validation_split,
    subset='training',
    seed=5
)

# Create the validation dataset from the 'train' directory
validation_ds = image_dataset_from_directory(
    directory='./resources/images/train',
    labels='inferred',
    label_mode='binary',
    batch_size=batch_size,
    image_size=image_size,
    validation_split=validation_split,
    subset='validation',
    seed=5
)

# Create the testing dataset from the 'test' directory
test_ds = image_dataset_from_directory(
    directory='./resources/images/test',
    labels='inferred',
    label_mode='binary',
    batch_size=batch_size,
    image_size=image_size
)
```

```
Found 508 files belonging to 3 classes.
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[9], line 9
      6 validation_split = 0.2
      8 # Create the training dataset from the 'train' directory
----> 9 train_ds = image_dataset_from_directory(
     10     directory='./resources/images/train',
     11     labels='inferred',
     12     label_mode='binary',
     13     batch_size=batch_size,
     14     image_size=image_size,
     15     validation_split=validation_split,
     16     subset='training',
     17     seed=123
     18 )
     20 # Create the validation dataset from the 'train' directory
     21 validation_ds = image_dataset_from_directory(
     22     directory='./resources/images/train',
     23     labels='inferred',
   (...)
     29     seed=123
     30 )

File c:\Users\rafen\Desktop\3APP_AI\.venv\lib\site-packages\keras\src\utils\image_dataset.py:221, in image_dataset_from_directory(directory, labels, label_mode, class_names, color_mode, batch_size, image_size, shuffle, seed, validation_split, subset, interpolation, follow_links, crop_to_aspect_ratio, **kwargs)
    210 image_paths, labels, class_names = dataset_utils.index_directory(
    211     directory,
    212     labels,
   (...)
    217     follow_links=follow_links,
    218 )
    220 if label_mode == "binary" and len(class_names) != 2:
--> 221     raise ValueError(
    222         'When passing `label_mode="binary"`, there must be exactly 2 '
    223         f"class_names. Received: class_names={class_names}"
    224     )
    226 if subset == "both":
    227     (
    228         image_paths_train,
    229         labels_train,
    230     ) = dataset_utils.get_training_or_validation_split(
    231         image_paths, labels, validation_split, "training"
    232     )

ValueError: When passing `label_mode="binary"`, there must be exactly 2 class_names.
Received: class_names=['balloon', 'helicopter', 'planes']
```

D. Train the model

Train your model for 10 epochs (otherwise it will take too long). Don't forget to store your training and validation losses in a history object, so you can visualize your training later.

If you get 'WARNING:tensorflow: Using a while_loop for converting', you can simply ignore them.

In [ ]:
```
# Use this cell to train the model
```

Question: What accuracy do you achieve on the training set?

ANSWER:

Question: What accuracy do you achieve on the validation set?

ANSWER:

E. Visualizing the training process

Plot the training and validation loss of your trainingcycles, using the history object that you created during training.

In [ ]:
```
#Use this cell to add the plotlosses function, and plot the history of your trainin
```

Question: is your model over-, under-, or well fitted, looking at the plotted losses above (and explain why)?

ANSWER:

F. Evaluation of the model

Add code in the cell below to evaluate your model, using the test set you created earlier.

In [ ]:
```
#Evaluate the model
```

Question: What is your accuracy on the test set?

ANSWER:

G. Predict & Plot

In the cell below, add code to predict & plot 10 test images from the test set. Below each image, you should use as a label the actual label (a), and the predicted label (p) of your model. If they are the same, the textcolor of the label should be green, red otherwise.

Hints:

- Getting 5 images/label pairs from a dataset, you can use: images, labels = next(iter(dataset.take(5)))

- For retrieving different labels from a dataset, you can use: dataset.class_names
- Ploting an image from a dataset, you have to typecast the pixel values to 'uint8', like: image.numpy().astype("uint8")

In [ ]:
```
#Predict and plot 10 test images
```