Classification of flowers

Link:

Project link (GitHub/ Gitlab)

https://github.com/loicbrn2/Group_Project_2.0/tree/main
https://github.com/arnauddmp/Final-Project

Team Members:

Student Name	Student ID	Contribution in the project
BEAURAIN Loïc	73134	25%
DE DAMPIERRE Arnaud	73096	25%
TISSOT-FAVRE Pierre-Louis	73148	25%
TIR Ismael	73153	25%

Model Architecture:

IMPORTANT WARNING: If you want to run the code, do not run the 3 last blocks. They take a lot of time to compile, and you would lose a lot of time doing that.

The aim of this project, based on neural network training, was to train a model capable of differentiating and determining the species of a plant (Lilly, Orchis, Lotus, Sunflower, Tulip) on the basis of an image of the plant. To do this, we had at our disposal a dataset consisting of 4980 pictures distributed almost equally between the different species.

Within this project, two different important values will appear (accuracy and val_accuracy), and it is important to understand the difference between the two: accuracy is calculated directly after training our model and is tested on the data that was used for the model's training, while val_accuracy is obtained by testing our model on the specifically designated validation set. The most important and representative value of the quality of our model is therefore val_accuracy. In this report, when the word accuracy is used, it refers to both the accuracy and val_accuracy of our model.

As always, the first step here is going to be the importation of the necessary libraries.

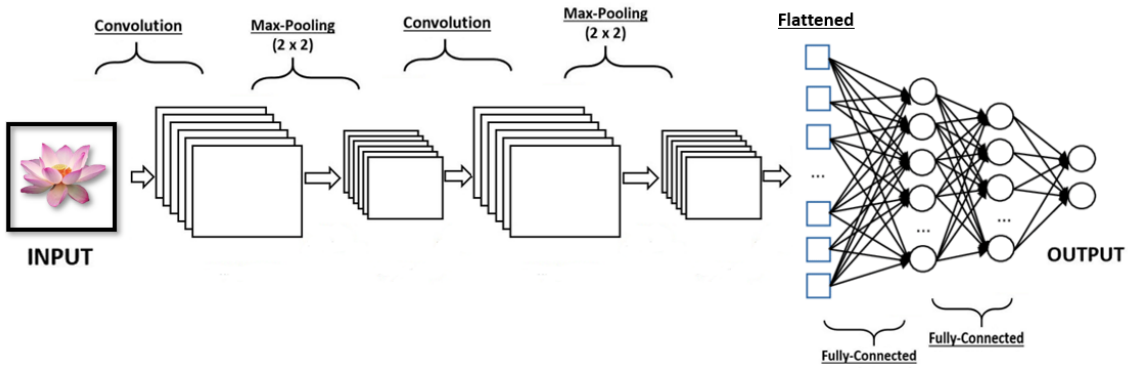
```
import os
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

1. What model architecture have you selected

As requested, we used a CNN model architecture for this project.

A Convolutional Neural Network (CNN) is a specialized architecture in deep learning tailored for processing grid-like data, notably images. Its design revolves around key components, called layers. There are several different types of layers. We can state for example the Convolutional Layers, which employ convolution operations to extract local his architecture uses for patterns and spatial relationships, or Pooling Layers, that reduce spatial dimensions, preserving essential features.

2. Put a diagram of the architecture (Schematic)



3. How many layers of the model have?

Our model is made up of 7 layers: 2 Convolutional layers, 2 Pooling layers, 1 Flatten layer and 2 Dense (Fully-Connected) layers.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 64, 3))) #removable
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32 * 2, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(len(label_dict), activation='softmax'))


model.compile(optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model1=model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

This specific number and composition of layers was not decided at random. Unlike our previous solo projects, the only parameter to optimise was not just the accuracy of our model, but also keeping execution time to a minimum. I will explain in detail how me made those choices later.

4. How many hidden layers and neurons do you have?

Hidden Layers: In this case, there are two hidden layers. These are the Dense layers with relu activations. These two layers correspond to fully connected or dense layers.

Hidden Neurons: The neurons in these hidden layers are defined by the parameters 128 and len(label_dict) in the Dense layers. Thus, the first hidden layer has 128 neurons, and the second one has 5 neurons.

5. Have you changed the model architecture to improve performance.

After extensive testing, we noticed a number of interesting things. The first is that adding layers doesn't necessarily make our model any more accurate. Indeed, even if each layer adds a calculation step, this step is not necessarily beneficial to the training of our model. In the same way, contrary to what we might have thought, removing layers doesn't necessarily improve the execution time of the program, despite the addition of a new calculation step.

In our programme, we have therefore opted for this combination of layers. As you can see, it's possible to remove the first layer, sacrificing 2 to 4% accuracy to enable our training to be run faster, at 5 to 7 seconds per epoch.

Dataset Description:

1. Details of the dataset (number of classes/ instances/ Images etc)

Our dataset is made up of 5 folders named after the 5 plant species we are trying to identify using our model, each containing 1000 images of their assigned species (only 980 for the Lilly species) for a total of 4980 images we can use to train our model.

2. What kind train / test split has been used

After some research, we noticed that two techniques exist to split our data into the necessary training and testing categories for our model training: the first one we have already used is the train_test_split function, which allows us to subdivide our data into two sets even before defining our model. The second option is to add the validation_split parameter during the definition of our model. We opted for the first case, which was more familiar to us, as we knew that we could add the random_state argument to eliminate any randomness in the predefinition of our test and train sets.

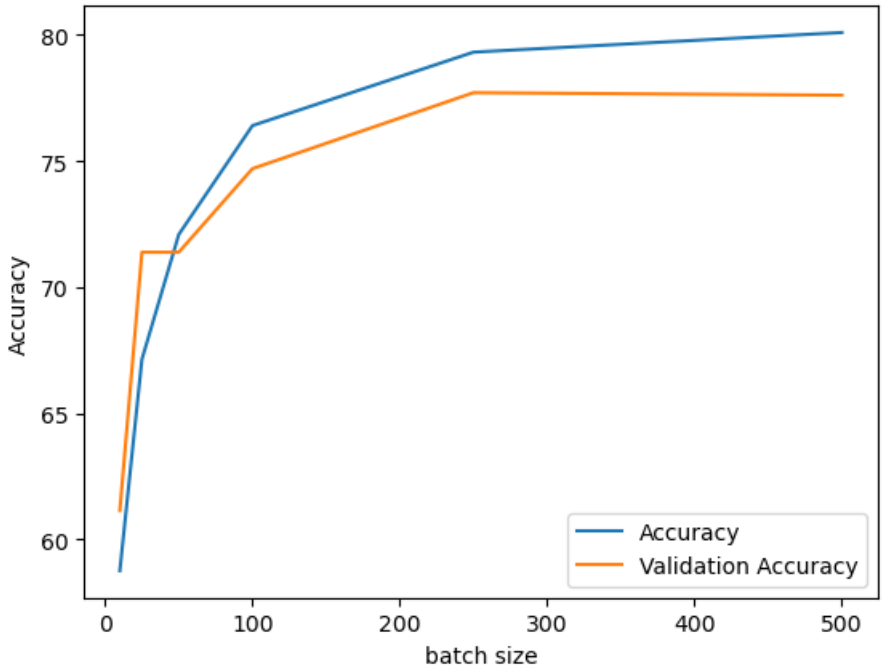
3. What kind of data augmentation have been used

A sample of 4980 images seemed rather small to effectively train our model. Therefore, we sought a way to augment our database to better train our neural network. We decided to use the ImageDataGenerator function, which, by taking certain images from the training set of our model and making various modifications (rotation, blur, shift, zoom, dezoom, etc.), allows us to augment our database with new images.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
all_acc2 = []
all_val_acc2 = []


model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 64, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32 * 2, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32 * 2, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(len(label_dict), activation='softmax'))


model.compile(optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'
)


batch_size_values=[10,25,50,100,250,500]
for x in batch_size_values :
    m=model.fit(datagen.flow(X_train, y_train, batch_size=x), epochs=5, validation_data=(X_test, y_test))
    all_acc2.append(m.history['accuracy'][-1] * 100)  # Ajout des valeurs d'accuracy à la liste
    all_val_acc2.append(m.history['val_accuracy'][-1] * 100)  # Ajout des valeurs de val_accuracy à la liste

plt.plot(batch_size_values, all_acc2, label='Accuracy')
plt.plot(batch_size_values, all_val_acc2, label='Validation Accuracy')
plt.xlabel('batch size')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

While applying this function, we wondered what value of the batch_size parameter would allow us to maximize the accuracy of our model. Therefore, we chose to display the curve

below, indicating the accuracy and val_accuracy of our model as a function of the chosen batch_size.batch_size.



We can observe two important things: the first is that regardless of the value given to the batch_size, the accuracy and val_accuracy obtained by augmenting the database using the ImageDataGenerator function are lower than the values found without using the function. This may seem counterintuitive because our model seems less well-trained with more data used for training. However, this can be explained by the increased diversity of the training data, which is beneficial to the model but makes learning more challenging, even when increasing the number of epochs. We believe that the use of this function should be accompanied by a thorough restructuring of the layers used.

The lack of improvement in our model could also be explained by the fact that data transformations, such as rotation, zoom, horizontal/vertical shifting, etc., are usually applied only to the training images and not to the test images, which are likely well-framed.

Methodology:

Explain the following in this page:

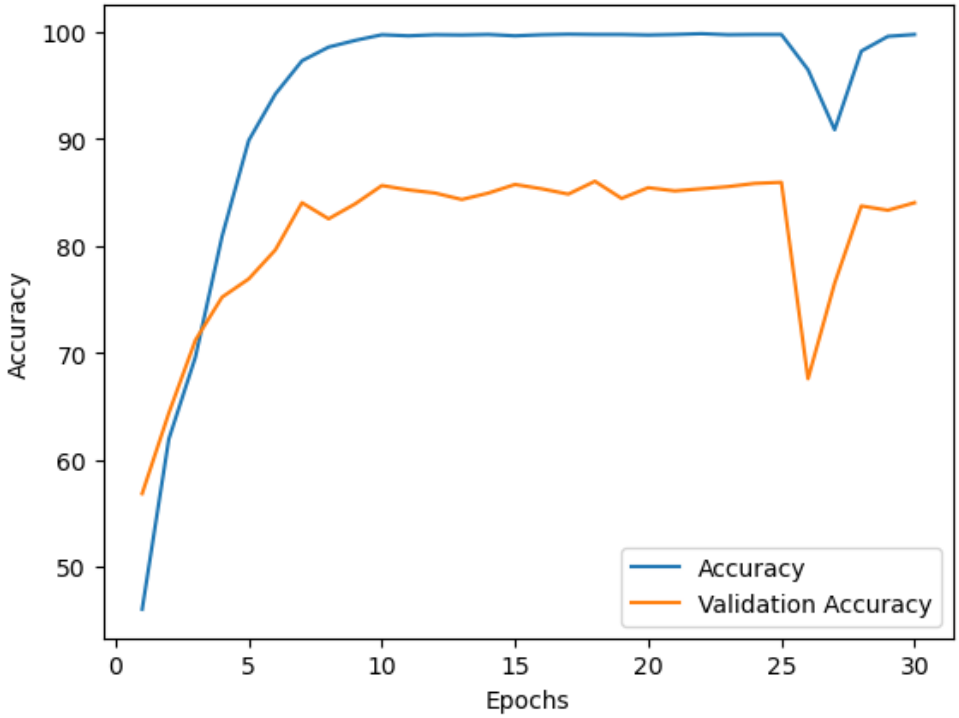
1. Training parameters, number of epochs, learning rate etc.

We chose Adam as the optimizer, 0.001 as the learning rate, 32 as the number of filters, and 128 as the number of units in the dense layer. I will revisit how we made these choices in the upcoming questions.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 64, 3))) #removable
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32 * 2, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32 * 2, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(len(label_dict), activation='softmax'))


model.compile(optimizer='adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])

model1=model.fit(X_train, y_train, epochs=10, validation_data=(X_test, y_test))
```

In order to determine the number of epochs that maximizes the accuracy of our model without overtraining it while minimizing the execution time of our program, we used the following script:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
all_acc = []
all_val_acc = []


model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(224, 64, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32 * 2, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(32 * 2, (3, 3), activation='relu'))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(len(label_dict), activation='softmax'))


model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])



history = model.fit(X_train, y_train, epochs=30, validation_data=(X_test, y_test))


train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']


for epoch, (train_acc, val_acc) in enumerate(zip(train_accuracy, val_accuracy), 1):
    print(f'Epoch {epoch}: Train Accuracy = {train_acc}, Validation Accuracy = {val_acc}')
    all_acc.append(train_acc * 100)
    all_val_acc.append(val_acc * 100)

epoch_list = range(1, 31)
plt.plot(epoch_list, all_acc, label='Accuracy')
plt.plot(epoch_list, all_val_acc, label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```



We can observe that from around ten epochs onwards, the values of accuracy and val_accuracy converge towards their maximum. Therefore, to maximize the accuracy of our model while limiting the execution time of our program, choosing 10 as the number of epochs seems to be a prudent choice.

2. Loss function used for training

Our neural network here aims to determine the classification of a flower into one species or another by analyzing a photo of it. Additionally, during the loading of our data, we redefined the labels of our different flower classes as integers. This led us to use the 'sparse_categorical_crossentropy' loss function, specifically designed for training a multi-class image recognition model with integer labels.

3. Changes is parameter affecting the model results

The most crucial part of optimizing our model was the selection of hyperparameters. To achieve this, we created a loop testing all combinations of hyperparameters.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

optimizer_options = ['adam', 'sgd', 'rmsprop']
learning_rate_options = [0.001, 0.01, 0.1]
num_filters_options = [32, 64]
num_dense_units_options = [64, 128]

for opt in optimizer_options:
    for lr in learning_rate_options:
        for nf in num_filters_options:
            for ndu in num_dense_units_options:

                model = models.Sequential()
                model.add(layers.Conv2D(nf, (3, 3), activation='relu', input_shape=(224, 64, 3)))
                model.add(layers.MaxPooling2D((2, 2)))
                model.add(layers.Conv2D(nf * 2, (3, 3), activation='relu'))
                model.add(layers.MaxPooling2D((2, 2)))
                model.add(layers.Conv2D(nf * 2, (3, 3), activation='relu'))
                model.add(layers.Flatten())
                model.add(layers.Dense(ndu, activation='relu'))
                model.add(layers.Dense(len(label_dict), activation='softmax'))


                model.compile(optimizer=opt,
                              loss='sparse_categorical_crossentropy',
                              metrics=['accuracy'])


                model.fit(X_train, y_train, epochs=5, validation_data=(X_test, y_test))


                val_loss, val_accuracy = model.evaluate(X_test, y_test)
                print(f"Optimiseur: {opt}, Taux d'apprentissage: {lr}, "
                      f"Nombre de filtres: {nf}, Nombre d'unités dans la couche dense: {ndu}")
                print(f"Précision sur l'ensemble de validation: {val_accuracy * 100:.2f}%\n")
```

This is how we determined the hyperparameters that maximize the accuracy and val_accuracy of our model (refer to question 1).

4. How have you improved the results?

Throughout our program, we aimed to maximize both the accuracy of our neural network and the speed of execution of our code. To optimize this program, we leveraged various factors. The first was the restructuring of the layers of our CNN. Numerous tests and research indicated one of the most optimal layer combinations for training our model (refer to question 3 and 5, part 1).

We then focused on optimizing hyperparameters. A quadruple loop helped us determine the optimal hyperparameters for training our neural network (refer to question 3, part 3).

As our program took a considerable amount of time to compile, we also explored the number of epochs to impose on our program, with the goal of maximizing precision while minimizing compilation time. After testing, we decided to set 10 epochs for the program (refer to question 1, part 3).

Finally, we attempted to use the ImageDataGenerator function to augment our database and improve its accuracy. However, after some testing, we noticed that our accuracy was negatively impacted when this function was used, prompting us to remove it from our program (refer to question 3, part 2).

Results:

Explain the following in this page:

1. Results of training (loss and accuracy of the model)

```
Epoch 10/10
125/125 [==============================] - 17s 137ms/step - loss: 0.0365 - accuracy: 0.9955 - val_loss: 1.0423 - val_accuracy:
0.8233
```

At our 10th epoch, we achieved an accuracy of 99.55% on the training set with a loss of only 0.0365. This indicates that the model is capable of effectively recognizing the species of the photos provided during training. It demonstrates that the model is in alignment with its training sample. However, these results should be taken with caution: while they show that the model does not contradict its training sample, they do not demonstrate its real accuracy when given a sample outside of its training set. The val_accuracy is more representative of the quality of our model in such cases.

2. Results on testing (loss and accuracy of the model)

```
Epoch 10/10
125/125 [==============================] - 17s 137ms/step - loss: 0.0365 - accuracy: 0.9955 - val_loss: 1.0423 - val_accuracy:
0.8233
```

Similarly, our val_accuracy at epoch 10 is 82.33% (actually ranging from 81 to 85%), while our val_loss is 0.9955. These values are more representative of the quality of our model because the test set is handled in our code in the same way a set of images external to the training of our model would be handled.

3. Show some direct results: example input picture/ sentence and output results.

To precisely measure the quality of our neural network, we decided to conclude our program with two tests: the first on the entire input dataset, used only for testing, and the second by creating our own dataset using internet images to test it manually.

```
dossier_test = "flower_images2"
X2, y2, label_dict2 = charger_images(dossier_test)

loss, accuracy = model.evaluate(X2, y2)

print(f"Loss: {loss}")
print(f"Accuracy: {accuracy}")
```

```
980
1980
2980
3980
4980
156/156 [==============================] - 4s 26ms/step - loss: 71.4092 - accuracy: 0.9355
Loss: 71.40916442871094
Accuracy: 0.9355421662330627
```

```
dossier_test = "Testr"
X2, y2, label_dict2 = charger_images(dossier_test)

loss, accuracy = model.evaluate(X2, y2)

print(f"Loss: {loss}")
print(f"Accuracy: {accuracy}")
```

```
10
20
30
40
50
2/2 [==============================] - 0s 23ms/step - loss: 108.8149 - accuracy: 0.8600
Loss: 108.8149185180664
Accuracy: 0.8600000143051147
```

We can see that our model has an accuracy of over 93.5% on the dataset used initially, which is not surprising considering that 80% of this dataset was used to train our model. Regarding the folder of images taken randomly from the internet, we have an accuracy of 86%. This accuracy, much more representative than the previous one, shows that our model, although imperfect, seems to recognize the species of the flower rather effectively from a photo, as long as the framing remains reasonably qualitative. Note that there

With this test, our creation of the image recognition neural network concludes. Thank you for your time and for reading this report to the end.