

Rapport

Projet de Classification d'images



MASTER Technologies de l'Internet

Réalisé par

EL ALOUI Oussama et BENKHANOUS Salaheddine

Prof. C. Pham

Sommaire

1. Objectif	5
2. Préparation des données	6
2.1. Chargement des données	6
2.2. Prétraitement des données.....	7
2.2.1. Mélanges d'images.....	7
2.2.2. Augmentation d'images	8
3. Création du modèle.....	9
3.1. Initialisation du modèle.....	9
3.2 Compilation du modèle :.....	10
4.1. Entraînement du modèle	11
5. Test du modèle.....	13
6. Interface Web.....	14
6. Résumé.....	15

Table de figure

Figure 1: Chargement des données des fleurs.....	6
Figure 2: Lecture des images fleurs.....	6
Figure 3: Données d'entrainement	6
Figure 4: Configuration l'ensemble de données pour les performances.....	7
Figure 5: Augmentation des images.....	8
Figure 6: Résultat d'augmentation d'une image	8
Figure 7: Définition de l'architecture du modèle	9
Figure 8: Compilation du modèle	10
Figure 9: le rapport de compilation du modèle	10
Figure 10: Phase d'entrainement du modèle.....	11
Figure 11: Visualisations de performance du modèle en termes de score et fonction cout ..	12
Figure 12: Score de précision du modèle sur les données d'entrainement	12
Figure 13: Fonction de test du modèle	13
Figure 14: Résultat de classe et le score de précision.....	13
Figure 15: Enregistrement du modèle	13
Figure 16: Interface web	14

1. Objectif

L'objectif de ce projet est de développer un modèle de réseau de neurones convolutifs (CNN) capable de classifier des images de fleurs en différentes catégories avec une haute précision. En utilisant la bibliothèque **TensorFlow** et **Keras**, ce projet vise à démontrer l'efficacité des techniques d'apprentissage profond pour la reconnaissance des images fleurs.

Le choix de la bibliothèque **TensorFlow** est motivé par son support natif des réseaux de neurones convolutifs (CNN), qui sont particulièrement efficaces pour la classification d'images. Les couches et les opérations nécessaires à la construction des CNN sont facilement accessibles et optimisées pour des performances maximales.

De plus, elle est compatible avec l'API **Keras** de haut niveau qui simplifie la construction, l'entraînement et l'évaluation des modèles.

Le modèle doit classifier ces images en 5 types :

- **Tulip**
- **Sunflower**
- **Rose**
- **Dandelion**
- **Daisy**

Ce projet base sur la préparation des données, la création et l'optimisation du modèle l'évaluation de ses performances, ainsi qu'une interface web pour tester la classification d'une image chargée par un utilisateur.

2. Préparation des données

2.1. Chargement des données

La première étape consiste à charger les images de fleurs à partir d'un URL puis les lire en utilisant la bibliothèque **os**.

```
#Chargement de base de données d'images
!wget -O dataset.zip https://www.dropbox.com/scl/fi/w6a9k9k1agp341drlpkq/Images.zip?rlkey=n5mjuy5y0ybi1bxxq5pfvqc4v&st=v700u5uo&dl=0
!unzip dataset.zip
```

Figure 1: Chargement des données des fleurs

```
#Lecture de dossier d'images
count = 0
images_hub = os.listdir('flowers/')
for dir in images_hub:
    images = list(os.listdir('flowers/'+dir))
    print('>> Dossier \''+dir+'\' contient '+ str(len(images)) + ' images')
    count = count + len(images)
print('-----> TOTAL d\'images : '+ str(count))

>> Dossier 'dandelion' contient 1052 images
>> Dossier 'rose' contient 784 images
>> Dossier 'daisy' contient 764 images
>> Dossier 'sunflower' contient 733 images
>> Dossier 'tulip' contient 984 images
-----> TOTAL d'images : 4317
```

Figure 2: Lecture des images fleurs

Ensuite, en utilisant **'tf.keras.utils.image_dataset_from_directory'**, nous avons séparé les images chargées en 2 type de données : **donnés d'entrainement** et **données de validation**.

```
[25] training_dataset = tf.keras.utils.image_dataset_from_directory( images_dir,
                                                                    seed = 123,
                                                                    validation_split=0.2,
                                                                    subset = 'training',
                                                                    batch_size=batch,
                                                                    image_size=(img_size,img_size))

Found 4317 files belonging to 5 classes.
Using 3454 files for training.
```

Figure 3: Données d'entrainement

Voici le rôle des arguments de la fonction montrée ci-dessus :

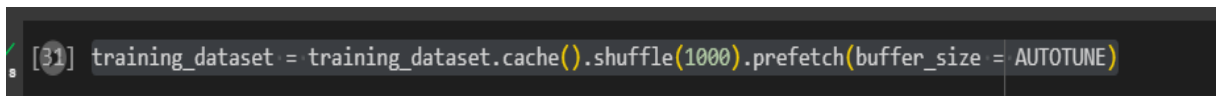
- **images_dir** : Répertoire contenant les images de fleurs.
- **validation_split=0.2** : 20 % des données sont réservées pour la validation.
- **subset='training' et subset='validation'** : Séparations des ensembles d'entraînement et de validation.
- **image_size=(img_size, img_size)** : Taille à laquelle les images sont redimensionnées.
- **batch_size=batch_size** : Taille des lots pour l'entraînement.

2.2. Prétraitement des données

Pour faciliter la phase d'entrainement du modèle, nous avons optimisé le pipeline de traitement des données en utilisant les techniques suivantes :

2.2.1. Mélanges d'images

Voici une explication détaillée sur cette technique comme montré dans la figure ci-dessous :



```
[31] training_dataset = training_dataset.cache().shuffle(1000).prefetch(buffer_size = AUTOTUNE)
```

Figure 4: Configuration l'ensemble de données pour les performances.

- **cache()** : Réduit le temps de latence dû au chargement répété des données depuis le disque, augmentant ainsi la vitesse d'entraînement.
- **shuffle(1000)**: mélange les données de manière aléatoire avec un buffer de la taille spécifiée (ici, 1000).

2.2.2. Augmentation d'images

Une technique permettant d'augmenter la diversité des données d'entraînement en appliquant des transformations aléatoires mais réalistes telles que **la rotation d'image, le zoom et le positionnement**.

```
[33] data_augmentation = Sequential([
    layers.RandomFlip("horizontal", input_shape = (img_size,img_size,3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1)
])
```

Figure 5: Augmentation des images

Voici un échantillon d'augmentation d'une image fleur de type **sunflower** :

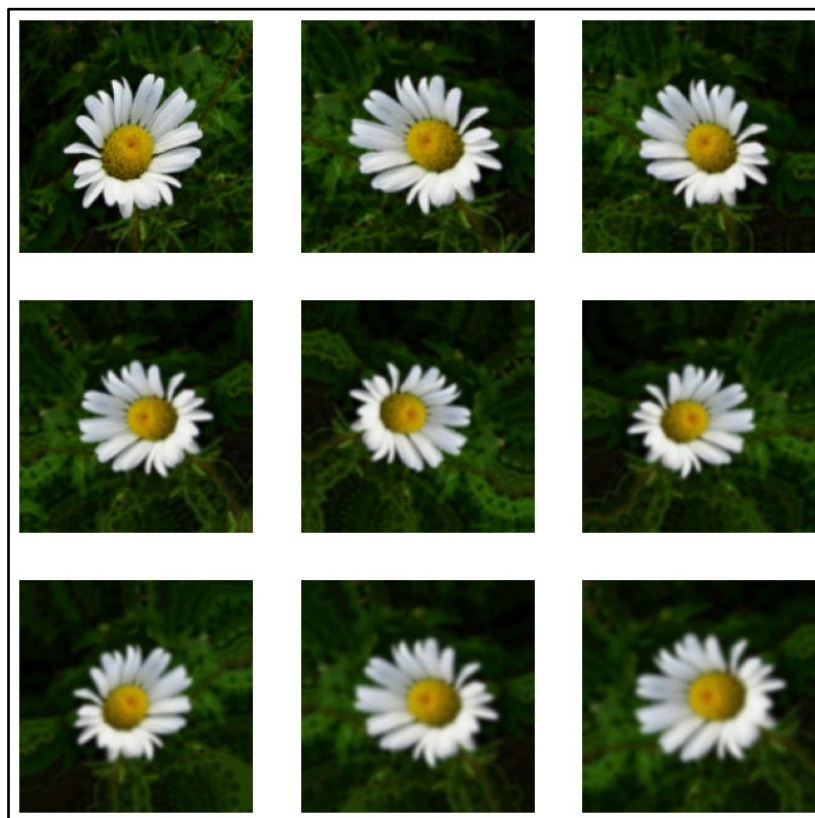


Figure 6: Résultat d'augmentation d'une image

3. Création du modèle

Nous avons défini un modèle de **réseau de neurones convolutifs (CNN)** avec TensorFlow et Keras.

3.1. Initialisation du modèle



```
model = Sequential([
    data_augmentation,
    layers.Rescaling(1./255),
    Conv2D(16, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(32, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Conv2D(64, 3, padding='same', activation='relu'),
    MaxPooling2D(),
    Dropout(0.2),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(5)
])
```

Figure 7: Définition de l'architecture du modèle

Voici les explications des arguments utilisés pour initialiser notre modèle :

- **tf.keras.layers.InputLayer** : Spécifie la forme des données d'entrée.
- **tf.keras.layers.Conv2D** : Couches convolutives pour extraire les caractéristiques des images.
- **tf.keras.layers.MaxPooling2D** : Couches de pooling pour réduire la dimensionnalité des caractéristiques.
- **tf.keras.layers.Flatten** : Aplatit les données pour les couches entièrement connectées.
- **tf.keras.layers.Dense** : Couches entièrement connectées pour la classification.

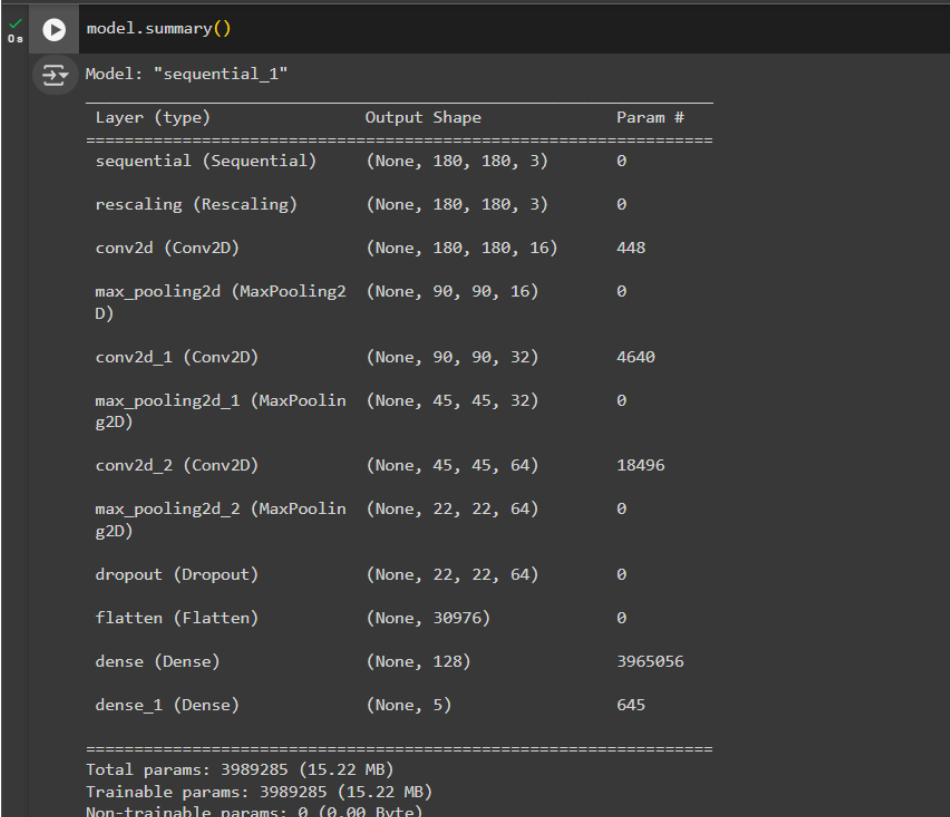
3.2 Compilation du modèle :

```
[36] model.compile(optimizer='adam',  
                  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),  
                  metrics=['accuracy'])
```

Figure 8: Compilation du modèle

- **optimizer='adam'** : Utilise l'optimiseur Adam pour ajuster les poids du modèle.
- **loss='sparse_categorical_crossentropy'** : Fonction de perte pour la classification multiclasse.
- **metrics=['accuracy']** : Évaluation du modèle en termes de précision.

Après avoir compilé notre modèle, nous avons extrait le résumé de cette procédure afin de savoir le nombre total des paramètres entraîné en succès.



```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 180, 180, 3)	0
rescaling (Rescaling)	(None, 180, 180, 3)	0
conv2d (Conv2D)	(None, 180, 180, 16)	448
max_pooling2d (MaxPooling2D)	(None, 90, 90, 16)	0
conv2d_1 (Conv2D)	(None, 90, 90, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 45, 45, 32)	0
conv2d_2 (Conv2D)	(None, 45, 45, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 22, 22, 64)	0
dropout (Dropout)	(None, 22, 22, 64)	0
flatten (Flatten)	(None, 30976)	0
dense (Dense)	(None, 128)	3965056
dense_1 (Dense)	(None, 5)	645

=====
Total params: 3989285 (15.22 MB)
Trainable params: 3989285 (15.22 MB)
Non-trainable params: 0 (0.00 Byte)

Figure 9: le rapport de compilation du modèle

Maintenant notre modèle est prêt à s'entraîner sur les images d'entraînement.

4.1. Entraînement du modèle

Puisque la phase d'entraînement du modèle est itérative, le choix de nombre d'itération dépend de plusieurs critères y compris : le score de classification, les valeurs optimales des arguments et le taux d'utilisation de CPU.

Après avoir essayé quelques nombres d'itérations, nous avons trouvé la valeur optimale pour cette phase d'entraînement est **15**.

```
history = model.fit(training_dataset, epochs=15, validation_data=validation_dataset)

Epoch 1/15
108/108 [=====] - 160s 1s/step - loss: 1.2999 - accuracy: 0.4461 - val_loss: 1.1460 - val_accuracy: 0.5214
Epoch 2/15
108/108 [=====] - 142s 1s/step - loss: 1.0439 - accuracy: 0.5851 - val_loss: 1.0166 - val_accuracy: 0.6025
Epoch 3/15
108/108 [=====] - 144s 1s/step - loss: 0.9713 - accuracy: 0.6155 - val_loss: 0.9617 - val_accuracy: 0.6083
Epoch 4/15
108/108 [=====] - 142s 1s/step - loss: 0.8965 - accuracy: 0.6523 - val_loss: 0.8734 - val_accuracy: 0.6547
Epoch 5/15
108/108 [=====] - 141s 1s/step - loss: 0.8452 - accuracy: 0.6769 - val_loss: 0.8445 - val_accuracy: 0.6616
Epoch 6/15
108/108 [=====] - 140s 1s/step - loss: 0.8189 - accuracy: 0.6792 - val_loss: 0.8610 - val_accuracy: 0.6582
Epoch 7/15
108/108 [=====] - 137s 1s/step - loss: 0.7720 - accuracy: 0.6980 - val_loss: 0.8576 - val_accuracy: 0.6628
Epoch 8/15
108/108 [=====] - 140s 1s/step - loss: 0.7482 - accuracy: 0.7180 - val_loss: 0.8284 - val_accuracy: 0.6640
Epoch 9/15
108/108 [=====] - 137s 1s/step - loss: 0.7114 - accuracy: 0.7313 - val_loss: 0.8400 - val_accuracy: 0.6779
Epoch 10/15
108/108 [=====] - 137s 1s/step - loss: 0.6986 - accuracy: 0.7281 - val_loss: 0.7608 - val_accuracy: 0.7173
Epoch 11/15
108/108 [=====] - 137s 1s/step - loss: 0.6609 - accuracy: 0.7420 - val_loss: 0.7989 - val_accuracy: 0.7022
Epoch 12/15
108/108 [=====] - 137s 1s/step - loss: 0.6429 - accuracy: 0.7559 - val_loss: 0.7665 - val_accuracy: 0.7161
Epoch 13/15
108/108 [=====] - 135s 1s/step - loss: 0.5978 - accuracy: 0.7785 - val_loss: 0.7866 - val_accuracy: 0.7126
Epoch 14/15
108/108 [=====] - 137s 1s/step - loss: 0.5897 - accuracy: 0.7765 - val_loss: 0.7922 - val_accuracy: 0.7092
Epoch 15/15
108/108 [=====] - 137s 1s/step - loss: 0.5791 - accuracy: 0.7774 - val_loss: 0.8035 - val_accuracy: 0.7161
```

Figure 10: Phase d'entraînement du modèle

Pour mieux comprendre le déroulement de cette phase, nous avons créé 2 graphes de visualisation des données fonction de score et de fonction de perte pour les images d'entraînement et de validation :

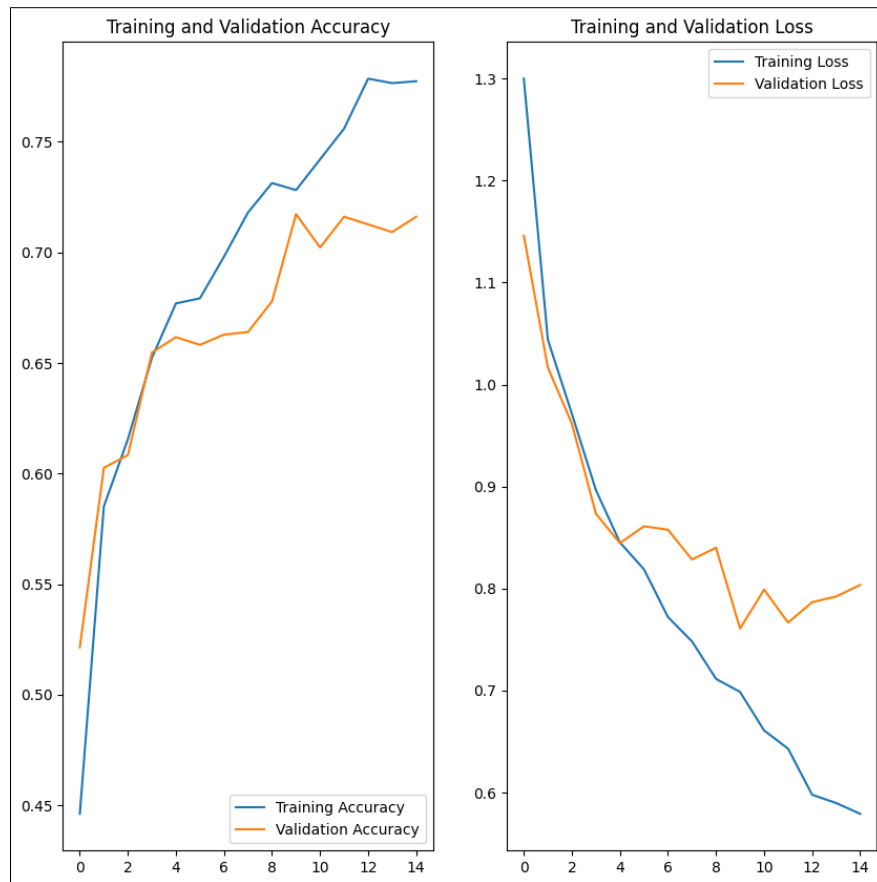


Figure 11: Visualisations de performance du modèle en termes de score et fonction cout

Voici le résultat de score de précision de notre modèle sur les images d'entraînement
`validation_dataset` :

```
test_loss, test_acc = model.evaluate(validation_dataset)
print(f"Test accuracy: {test_acc}")
```

27/27 [=====] - 8s 267ms/step - loss: 0.8035 - accuracy: 0.7161
Test accuracy: 0.7161065936088562

Figure 12: Score de précision du modèle sur les données d'entraînement

5. Test du modèle

Après la phase d'entraînement du modèle, il faut examiner la valeur de sortie de classification sur une nouvelle image qui n'appartient pas aux images d'entraînement.

Pour ce faire, nous avons défini la fonction suivante :

```
def classify_images(image_path):  
    input_image = tf.keras.utils.load_img(image_path, target_size=(180,180))  
    input_image_array = tf.keras.utils.img_to_array(input_image)  
    input_image_exp_dim = tf.expand_dims(input_image_array,0)  
  
    predictions = model.predict(input_image_exp_dim)  
    result = tf.nn.softmax(predictions[0])  
    outcome = '>> Image est de classe {' + flowers_classes[np.argmax(result)] + '} avec un score de ' + str(np.max(result)*100)  
    return outcome
```

Figure 13: Fonction de test du modèle

Voici le résultat de classification donné par notre modèle sur une image :

```
classify_images('flowers/dandelion/10043234166_e6dd915111_n.jpg')  
#normalement la classe d'image est : dandelion , on verra la classification prédite par notre modèle :  
  
1/1 [=====] - 0s 36ms/step  
'>> Image est de classe {dandelion} avec un score de 98.61971735954285'
```

Figure 14: Résultat de classe et le score de précision

Pour clôturer cette étape, nous avons enregistré le modèle sous forme un fichier afin de l'utiliser dans notre application web.

```
model.save('model_alpha.h5')
```

Figure 15: Enregistrement du modèle

6. Interface Web

Dans cette partie, il faut initialiser un environnement virtuel de python en suivant les étapes ci-jointes :

1. Créer un Environnement Virtuel : `python -m venv venv`

2. Activation de l'environnement : `.\venv\Scripts\activate`

3. Telecharger les bibliothèques nécessaires :

```
python -m pip install tensorflow==2.15.0
```

```
python -m pip install keras==2.1.5
```

Ensuite, nous avons créé un formulaire simple où l'utilisateur peut charger une image et recevoir la réponse de classification du modèle, comme le montre l'interface suivante :



Figure 16: Interface web

6. Résumé

Ce projet vise à fournir une analyse et une conception complètes d'un modèle de réseau de neurones convolutifs (CNN), en utilisant les bibliothèques fameuses **Tensorflow** et **keras**.

Grâce à des techniques avancées de prétraitement des données et d'optimisation du pipeline de données, le modèle est devenu capable de classifier les images de type fleur avec une grande précision.

Fin de rapport.