



Rapport final

Projet Deep Learning

2ème Année, Département des Sciences Numériques - Parcours Logiciel

OUSSAMA ECHCHERQAOUI
MANAL HAJJI
YOUNES SAOUDI
REDA EL JAI

2020 - 2021

Contents

1 Description du sujet	3
2 Constitution de base de données	4
2.1 Quelques exemples d'images de bases de données	4
3 Etapes de résolution	7
3.0.1 Étape 1: Préparation de la base de donnée	7
3.0.2 Étape 2: Construction d'un réseau de neurones CNN	7
3.0.3 Étape 3: Compilation et entraînement	7
3.0.4 Étape 4: Amélioration du résultat	7
3.0.5 Étape 5: Évaluation du modèle et extraction du meilleur modèle entraîné	8
3.0.6 Étape 6: Tester avec le meilleur modèle	9
3.0.7 Étape 7: Adaptation du modèle	10
3.0.8 Etape 8: Application de la méthode Transfer Learning et Fine-Tuning	11
4 Analyse des résultats	12
4.1 Comparaison entre les performances de binary/multi classes	12
4.2 Analyse de l'accuracy et loss entre notre modèle et après application du fine-tuning dans le problème multiclass	13
4.3 Analyse globale des résultats	17
5 Difficultés rencontrées	18
6 Conclusion	20
7 Annexe	21
7.1 Google Image Extractor	21
7.2 Flickr Scraper	22

1 Description du sujet

De nos jours, la gestion urbaine est de plus en plus intelligente, l'application des systèmes de transport intelligents (STI) à la surveillance du trafic urbain et à la gestion des autoroutes est de plus en plus courante. Un tel système intelligent peut capter les informations des véhicules et suivre les véhicules selon la vidéo-surveillance. La reconnaissance automatique du logo est l'un des éléments clés de la collecte d'informations sur les véhicules dans un système de transport intelligent (STI). Dans l'ensemble, la plupart des méthodes de recherche existantes peuvent être divisées en deux étapes: Détection de logo et classification de logo.

La reconnaissance de marque de véhicules peut être vu comme un problème de classification multi-classes avec un très grand nombre de classes.

Dans le cadre de ce projet, la classification sera surtout utilisée pour apprendre des éléments caractéristiques (les logos) qui permettront de bien identifier les véhicules. Une fois que les éléments caractéristiques sont extraits, l'identification de la marque du véhicule sera faite par vérification en utilisant ces éléments caractéristiques. Plus précisément, les éléments caractéristiques d'un modèle test seront comparés aux éléments caractéristiques des modèles de référence pour ensuite prendre une décision sur l'identité du véhicule.

Au delà de leur aspect décoratif et de leurs fins publicitaires, les logos de véhicules peuvent fournir des informations qui pourront être utilisées pour la vérification, ou classification des véhicules. Cependant, les logos des constructeurs automobiles ont des formes et des couleurs très variées d'un constructeur à un autre. Sans oublier qu'en pratique l'aspect du logo capturé par une caméra va varier selon la position du véhicule par rapport à la caméra, et à la luminosité (excessive, faible, reflectance, présence d'ombres...) ce qui rend la tache de détection du logo assez délicate.

2 Constitution de base de données

Deux bases de données préexistantes ont été utilisées pour la conduction de notre propre base de données. Afin d'adapter ces bases de données aux algorithmes, nous avons passé un peu de temps au début du projet à nous assurer que toutes les images sont dans le format JPG correct avec le nom correspondant à la marque de la voiture dans la photo (BrandX.jpg) et à séparer les marques par dossier. Ces changements ont mené au bon déroulement de nos tests et des itérations.

Nous avons commencé à extraire les images à partir de **Google Images** en utilisant un programme Python qui exporte les images selon des critères de recherche. Bien que cette méthode donne des résultats, il s'est avérée que les images extraites sont redondantes et non seulement nécessitait du double checking afin de s'assurer qu'elles correspondaient bien à ce que nous recherchions, mais les photos n'avaient parfois rien à voir avec les critères de recherche.

Nous avons ensuite complété notre base de données en utilisant **Flickr Database** qui est l'un des meilleurs services de gestion et de partage de photos en ligne et de base de données d'images. Nous avons ainsi utilisé le **Flickr API** dans un programme Python pour collecter les images de chaque marque de voiture.

Au total, nous avons pu récolter des centaines de photos pour une dizaine de marques différentes d'automobiles que nous avons partitionné de la manière suivante : 80% pour l'apprentissage, 10% pour la validation et 10% pour les tests. Etant donné que chaque classe contient des images différentes, nous avons donc choisi d'attribuer la majeure partie de la base de données à l'apprentissage.

Nous avons utilisé une base de données qui a la structure suivante : 10 dossiers correspondant à chaque image et qui contiennent chacun toutes les images de la marque. La séparation en images d'apprentissage, validation et tests s'est faite au moment de charger les images, dans le script. On pensait que cette approche allait nous permettre de mieux gérer les images et les labels pour l'apprentissage et surtout pour les tests.

Nous avons changé l'arborescence intital DossierMarque -> Dossier Apprentissage - DossierValidation - DossierTest à cause de problèmes rencontrés que nous allons détailler dans la section 5.

Nous avons exploité en premier lieu une sous base de données test consistuée de 2 marques seulement (Ford - Renault) (**Test2Classes**) pour tester les performances de réseau de neurones avant de généraliser à 10 classes.

Voici le lien vers notre base de données :

<https://github.com/oussama-echcherqaoui/DeepLearning>

Voici le lien vers la deuxième base de données :

<https://github.com/Hajji-Manal/DeepLearningProject>

2.1 Quelques exemples d'images de bases de données



Figure 1: Liste des 10 logos utilisés



Figure 2: Exemples de différentes images du logo Jaguar



Figure 3: Exemples de différentes images du logo Ferrari



Figure 4: Exemples de différentes images du logo Kia



Figure 5: Exemples de différentes images du logo Fiat



Figure 6: Exemples de différentes images du logo BMW

3 Etapes de résolution

Première Partie :

Dans cette partie, nous avons commencé par travailler sur deux classes afin d'assurer le bon fonctionnement du code quand nous avons implémenté au niveau de construction du réseau, entraînement et tests.

3.0.1 Étape 1: Préparation de la base de donnée

Dans cette étape, on a commencé par charger la base de donnée avec la fonction `load_data` déjà décrite. Cette base est découpée arbitrairement en trois ensembles: 80% pour l'apprentissage, 10% pour la validation et 10% pour le test.

3.0.2 Étape 2: Construction d'un réseau de neurones CNN

Ce réseau alterne les couches de convolution et de Max Pooling (afin de diviser à chaque fois la dimension des tenseurs par 2).

La première couche comptera 32 filtres de convolution, la seconde 64, la troisième 96 et la 4e 128. Enfin, avant la couche de sortie, on a ajouté une couche dense comptant 512 neurones. On a donc construit un réseau à 6 couches, sorte de version simplifiée d'AlexNet.

Pour construire ce réseau, on a utilisé les fonctions `Conv2D`, `Maxpooling2D`, et `Flatten` de Keras.

3.0.3 Étape 3: Compilation et entraînement

Nous devons spécifier la fonction de perte à utiliser pour évaluer un ensemble de poids, l'optimiseur est utilisé pour rechercher parmi différents poids pour le réseau et toutes les mesures facultatives que nous aimerais collecter et rapporter pendant l'entraînement.

Dans ce cas, nous utiliserons l'entropie croisée comme argument de perte. Cette perte concerne un problème de classification binaire et est définie dans Keras comme «`binary_crossentropy`».

Nous définirons l'optimiseur comme l'algorithme efficace de descente de gradient stochastique «`adam`». Il s'agit d'une version populaire de la descente de gradient car elle s'accorde automatiquement et donne de bons résultats dans un large éventail de problèmes.

Enfin, comme il s'agit d'un problème de classification, nous collecterons et rapporterons la précision de classification, définie via l'argument métrique.

Pour ce problème, nous avons choisi un nombre d'époque de 20 et un `batch_size` de 10. Le nombre d'époque n'était pas choisi aléatoirement, après plusieurs entraînements du modèle avec différentes valeurs d'époques, nous avons observé que la valeur de précision d'apprentissage stagne sur 1 à partir de l'époque 20.

Ces configurations peuvent être choisies expérimentalement par essais et erreurs. Nous voulons entraîner le modèle suffisamment pour qu'il apprenne un bon (ou assez bon) mappage des lignes de données d'entrée à la classification de sortie. Le modèle aura toujours une erreur, mais la quantité d'erreur se stabilisera après un certain point pour une configuration de modèle donnée. C'est ce qu'on appelle la convergence des modèles.

3.0.4 Étape 4: Amélioration du résultat

Les résultats montrent un grand sur-apprentissage entre les données d'entraînement et les données de validation, c'est pour celà on a dû apporter quelques modifications sur le modèle afin de réduire le sur-apprentissage. Parmi ces modifications, on a doublé la première couche d'entrée pour que le réseau puisse extraire le maximum d'informations de la data, on a en plus utilisé une méthode de régularisation `Dropout`.

`Dropout` est une technique de régularisation pour les modèles de réseaux de neurones et une technique

dans laquelle les neurones sélectionnés au hasard sont ignorés pendant l'entraînement. Ils sont «dropped-outs» au hasard. Cela signifie que leur contribution à l'activation des neurones en aval est temporairement supprimée lors du "forward pass", et que les mises à jour de poids ne sont pas appliquées au neurone lors du "backward pass".

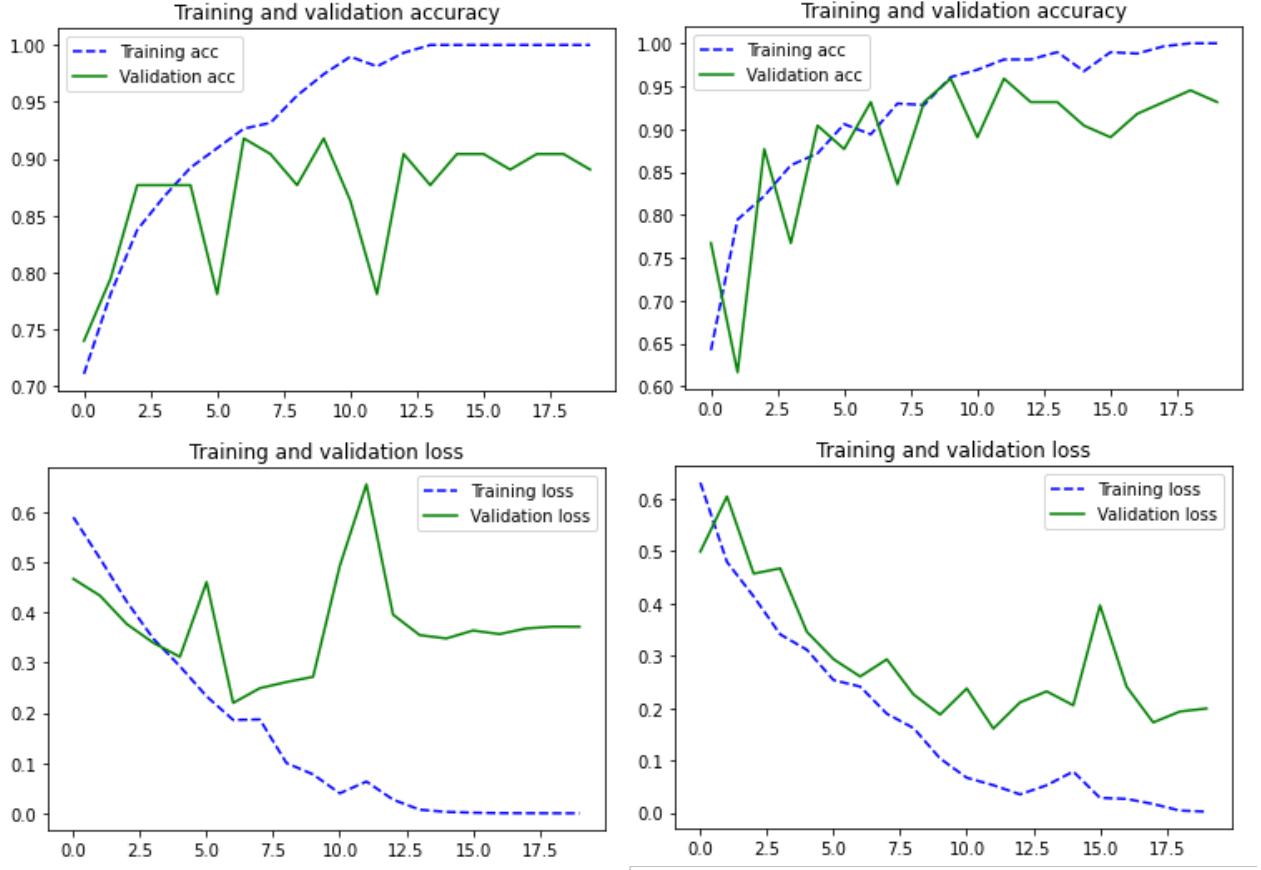


Figure 7: Amélioration de performance à l'aide du Dropout

L'effet est que le réseau devient moins sensible aux poids spécifiques des neurones. Cela aboutit à son tour à un réseau qui est capable d'une meilleure généralisation et est moins susceptible de sur-apprendre les données d'entraînement.

3.0.5 Étape 5: Évaluation du modèle et extraction du meilleur modèle entraîné

L'évaluation est un processus pendant le développement du modèle pour vérifier si le modèle est le mieux adapté au problème donné et aux données correspondantes. Le modèle que nous avons eu à l'issue de l'entraînement avait plus ou moins des résultats biens (l'analyse de ces résultats se fera plus tard). Sauf que nous voulons extraire le meilleur modèle pour tester avec. C'est pourquoi on a employé un callback qui s'appelle "ModelCheckpoint" pendant l'entraînement.

Le callback "ModelCheckpoint" nous permet de définir où vérifier les poids du modèle, comment le fichier doit être nommé et dans quelles circonstances créer un point de contrôle du modèle.

L'API nous permet de spécifier la métrique à surveiller, telle que la perte ou la précision sur l'ensemble de données d'entraînement ou de validation. Nous pouvons spécifier s'il faut rechercher une amélioration en maximisant ou en minimisant le score. Enfin, le nom de fichier que nous utilisons pour stocker les pondéra-

tions peut inclure des variables telles que le numéro d'époque ou la métrique.

Le callback enregistrera le modèle dans le fichier "best_model.h5"

La fonction de perte préférée à surveiller peut être spécifiée via l'argument moniteur, nous devons aussi spécifier le «mode» comme minimisant ou maximisant la mesure de la performance. Dans ce cas on a choisi de contrôler la "val_loss" sur le mode "min".

Enfin, nous nous intéressons uniquement au meilleur modèle observé pendant l'entraînement, plutôt qu'au meilleur par rapport à l'époque précédente, qui pourrait ne pas être le meilleur dans l'ensemble si l'entraînement est bruyant. Ceci peut être réalisé en définissant l'argument «save_best_only» sur True.

C'est tout ce qui est nécessaire pour garantir la sauvegarde du modèle avec les meilleures performances. Il peut être intéressant de connaître la valeur de la mesure de performance et à quelle époque le modèle a été sauvegardé. Cela peut être imprimé par le rappel en définissant l'argument «verbose» sur «1».

3.0.6 Étape 6: Tester avec le meilleur modèle

La prédiction est la dernière étape et le résultat attendu de la génération du modèle. Nous avons enregistré trois instances de données depuis l'ensemble de test X_test dans un tableau appelé samples, nous avons aussi enregistré les labels associés à samples dans une variable "labels". Nous pouvons alors passer à la fonction predict() de notre modèle ces échantillons afin de prédire les valeurs de classe pour chaque instance du tableau.

Ensuite, nous traçons chaque image des samples en figurant les prédictions calculées sur cette image.

Après on calcule la matrice de confusion en utilisant "confusion_matrix". Une matrice de confusion décrit les performances du modèle de classification. En d'autres termes, la matrice de confusion est un moyen de résumer les performances du classificateur. Elle affiche un tableau indiquant les vrais positifs, les vrais négatifs, les faux positifs et les faux négatifs.

Finalement, on trace cette matrice de confusion en utilisant "heatmap()". l'outil Heatmap de seaborn est une carte thermique et un moyen de représenter les données sous une forme bidimensionnelle. Les valeurs de données sont représentées sous forme de couleurs dans le graphique. Le but de cette carte est de fournir un résumé visuel coloré des informations ie fournir une information sur l'exactitude de la prédiction des marques de voitures dans notre cas. L'analyse des résultats de cette carte sera fait plus tard.

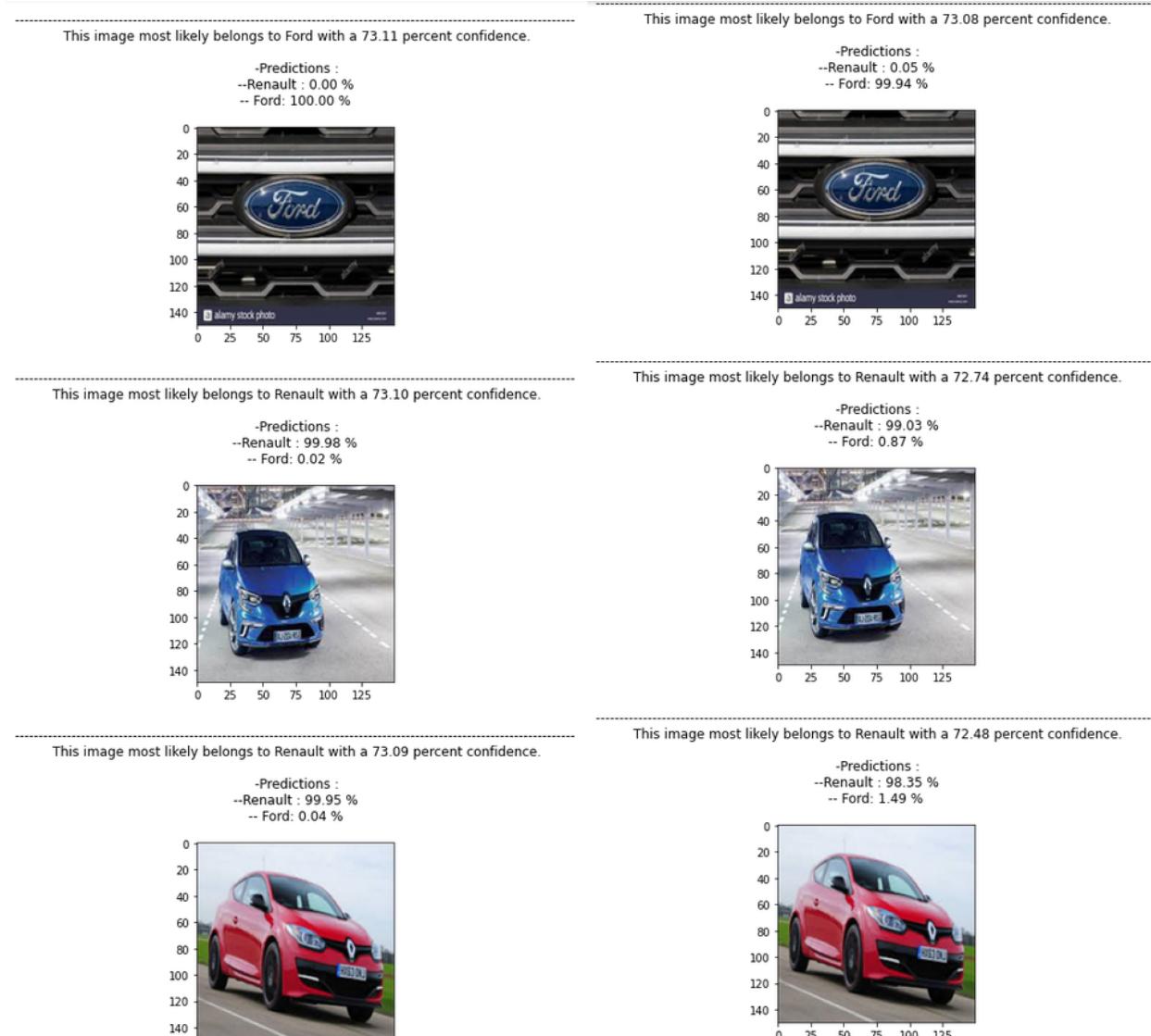


Figure 8: Comparaison des prédictions avec et sans Checkpoint minimisant loss

Nous constatons après avoir testé sur 3 images que le nouveau modèle permettait de diminuer la perte de 0,34 à 0,30, mais au prix d'une diminution de précision de 0.95 à 0.90. Ce qui se traduit notamment par une baisse de pourcentage de prédictions. Le maximum de baisse étant de 1,6% sur la troisième image.

Deuxième Partie :

Dans cette partie, après le bon fonctionnement de notre code et de notre modèle, on va travailler sur toutes les classes de notre dataset.

3.0.7 Étape 7: Adaptation du modèle

On refait presque toutes les étapes en ce qui concerne la préparation de la base de donnée, la construction du modèle, la compilation et l'entraînement. Sauf que au niveau du réseau on utilise 'softmax' comme fonction d'activation dans la dernière couche de sortie et "categorical_crossentropy" comme fonction de perte vu qu'il s'agit maintenant d'un problème multiclass. Et pour le test, on fait cette fois-ci la prédiction sur 10

instances de l'ensemble de test au lieu de 3 pour avoir plus d'informations sur les résultats de la prédiction et son taux d'erreur.

3.0.8 Etape 8: Application de la méthode Transfer Learning et Fine-Tuning

L'une des raisons qui peut expliquer le fait que nos résultats soient décevants est que les premières couches de notre réseau convolutif, sensées détecter des caractéristiques utiles pour discriminer les classes, n'ont pas appris de filtres suffisamment généraux. Ainsi, même si ces filtres sont pertinents, il y a en fait assez peu de chances que ces filtres puissent bien fonctionner pour la généralisation sur de nouvelles données.

C'est la raison pour laquelle nous avons envie de réutiliser un réseau pré-entraîné sur une large base de données, permettant donc de détecter des caractéristiques qui généraliseront mieux à de nouvelles données. Dans cette partie, nous allons réutiliser un réseau célèbre, et d'ores et déjà entraîné sur la base de données ImageNet : le réseau VGG-16.

Pour améliorer les résultats, on a opté pour la méthode "Fine-Tuning" qui consiste à prendre les poids du réseau neuronal entraîné et l'utiliser comme initialisation pour un nouveau modèle en cours d'apprentissage sur des données du même domaine. Il sert à accélérer la formation et surmonter la petite taille du jeu de données.

Pour cela, nous allons repartir du réseau que nous venons d'entraîner, mais nous allons débloquer l'entraînement des poids de l'ensemble du réseau. L'objectif est simplement de faire évoluer les paramètres du réseau "à la marge", et ceci ne peut être fait qu'après la première étape de transfer learning précédente. Sans cela, les dernières couches ajoutées à la suite de la base convulsive, après leur initialisation aléatoire, auraient engendré de forts gradients qui auraient complètement détruit les filtres généraux de VGG.

4 Analyse des résultats

4.1 Comparaison entre les performances de binary/multi classes

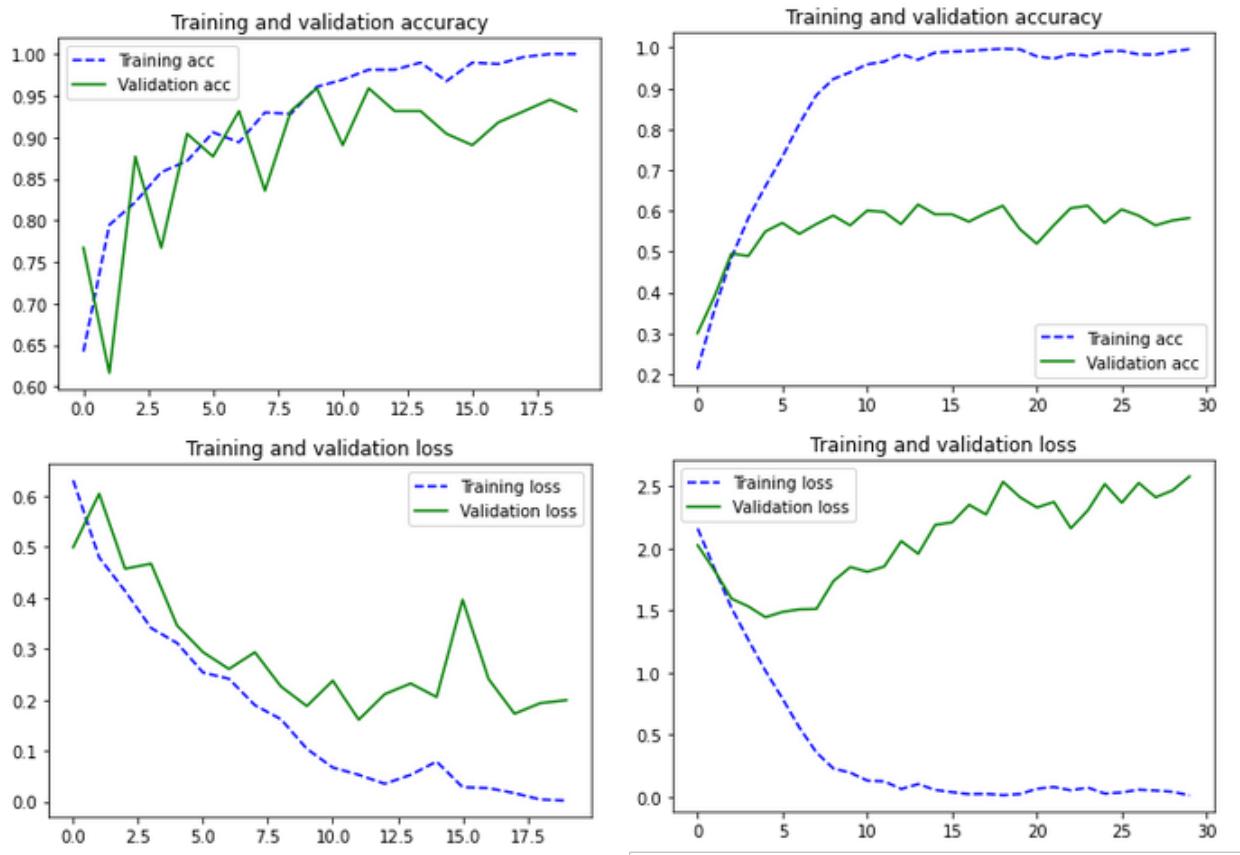


Figure 9: Comaparaison de Loss et Accuracy pour 2 classes et multiclasses

Comme attendu, nous remarquons dans la figure 9 qu'en passant de 2 classes à 10 classes, Loss augmente nettement alors que Accuracy baisse.

4.2 Analyse de l'accuracy et loss entre notre modèle et après application du fine-tuning dans le problème multiclassé

1/ Comparaison des performances

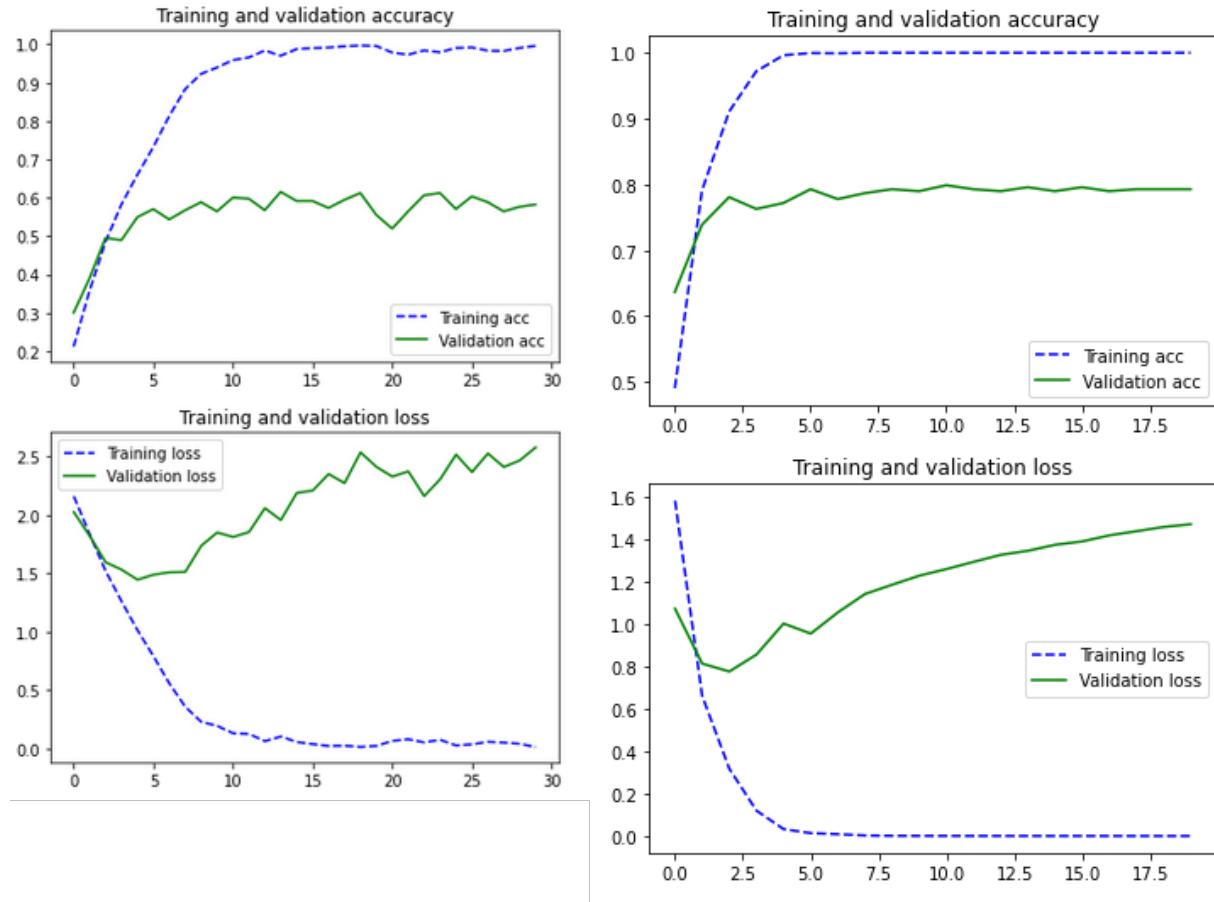


Figure 10: Comparaison des performances avec et sans fine-tuning

Suite à notre insatisfaction par les performances de notre réseau (0,6 d'accuracy et 2,5 de loss), nous avons décidé d'opter pour la méthode fine tuning. Les résultats se sont nettement amélioré puisque nous sommes passés à 0,8 d'accuracy et 1,4 de loss. Ceci a également positivement impacté la confiance de prédiction des classes, comme sera expliqué dans la section qui suit.

2/ Comparaison des résultats et des évaluations et des tests

Nous avons choisi 10 instances de la base de données sur lesquelles nous avons mené nos tests. Tout d'abord sans fine tuning, en récupérant le meilleur modèle minimisant la perte (val_loss) à l'aide du callback Modelcheckpoint. Nous trouvons les résultats des évaluations indiqués dans les deux figures qui suivent (16 et 17).

```

----- evaluation model -----

Evaluate on test data
11/11 [=====] - 0s 21ms/step - loss: 2.4724 - acc: 0.6006
[test loss, test acc] : [2.4723894596099854, 0.6006006002426147]

----- evaluation best model -----

Evaluate on test data
11/11 [=====] - 0s 20ms/step - loss: 1.4066 - acc: 0.5676
[test loss, test acc] : [1.4066243171691895, 0.5675675868988037]

```

Figure 11: Comparaison d'évaluation du modèle et best modèle

```

----- evaluation tuning -----

Evaluate on test data
11/11 [=====] - 1s 67ms/step - loss: 0.9845 - acc: 0.8108
[test loss, test acc] : [0.9845231175422668, 0.8108108043670654]

```

Figure 12: Evaluation des données de tests avec fine tuning

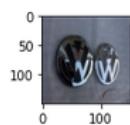
Comme c'était le cas pour le cas binaire, on remarque à nouveau que la minimisation de Val_Loss impacte négativement la précision des prédictions en multilabels. Le choix de minimiser Val_Loss au lieu de maximiser accuracy intervient suite aux remarques suivantes :

- La différence entre la meilleure et pire valeur d'Accuracy est de 10% environ.
- Nous pouvons gagner jusqu'à 1 en terme de Val_loss.

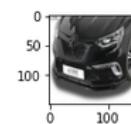
Les résultats demeurant insatisfaisants, nous avons opté pour la méthode de fine-tuning.

Les résultats des tests après introduction de fine-tuning confirme l'amélioration des performances précédemment détaillée (figure 10). Comme le montre la figure 12, nous sommes passés de 0.5676 à 0.8108 en terme de performance et de 1,4066 à 0,9845.

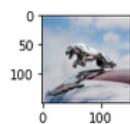
This image most likely belongs to Volkswagen with a 11.58 percent confidence.



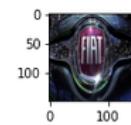
This image most likely belongs to Opel with a 12.80 percent confidence.



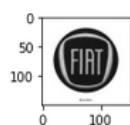
This image most likely belongs to Opel with a 11.92 percent confidence.



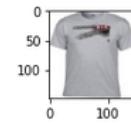
This image most likely belongs to Fiat with a 18.48 percent confidence.



This image most likely belongs to Fiat with a 14.05 percent confidence.



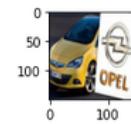
This image most likely belongs to Opel with a 11.85 percent confidence.



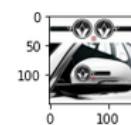
This image most likely belongs to Mercedes with a 21.86 percent confidence.



This image most likely belongs to Renault with a 13.56 percent confidence.



This image most likely belongs to Mercedes with a 14.93 percent confidence.



This image most likely belongs to Renault with a 13.49 percent confidence.

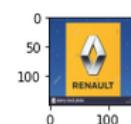


Figure 13: Résultats de prédiction du meilleur modèle obtenu par call-back

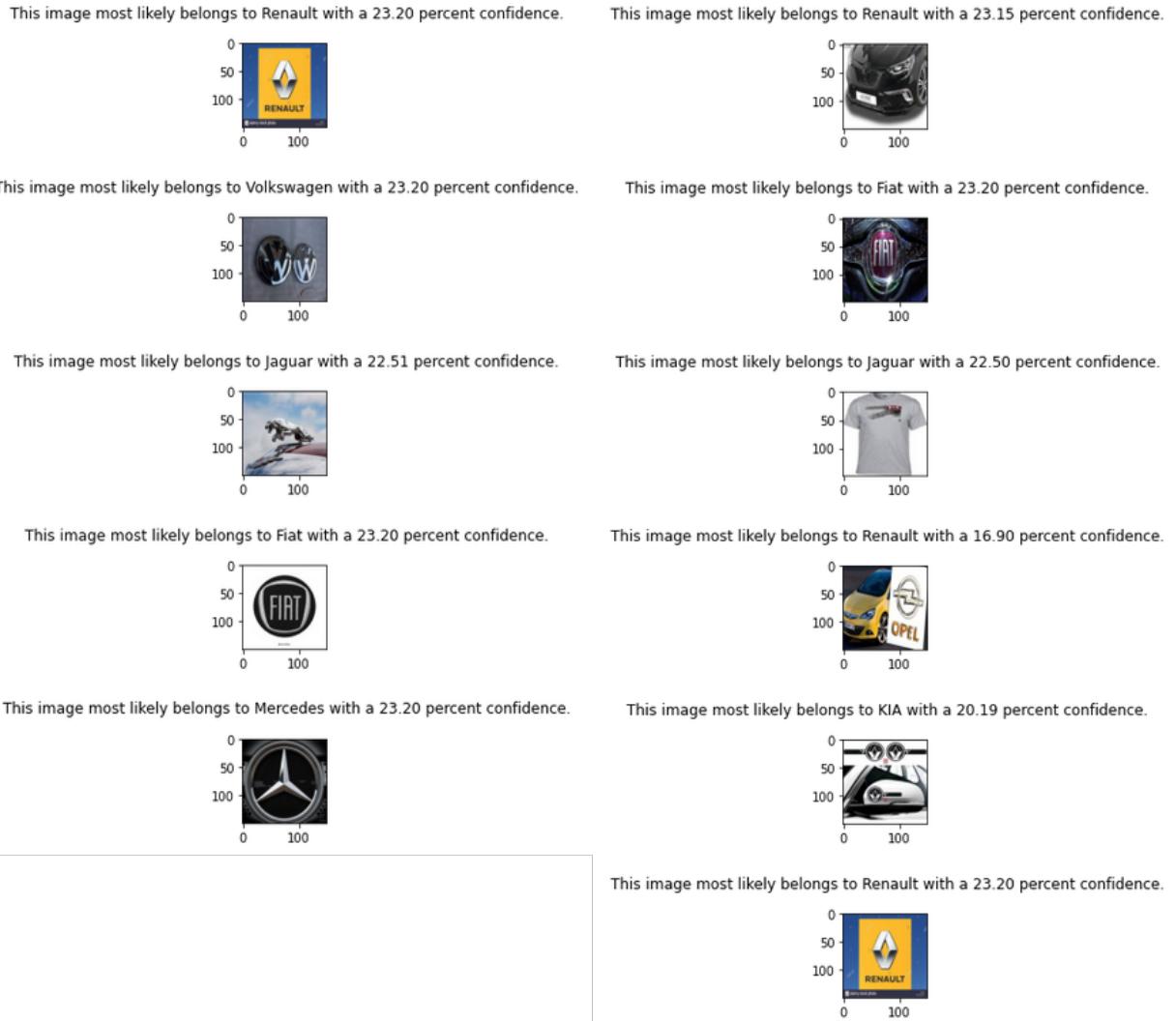


Figure 14: Résultats de prédiction après introduction de fine tuning

Nous remarquons qu'après introduction du fine-tuning, les prédictions s'améliorent : Le réseau se trompe moins et prédit avec une meilleure confiance les classes. Nous avons par exemple gagné 10% de confiance sur la prédiction du logo de Renault (13,40 % contre 23,20%).

3/ Comparaison des matrices de confusion

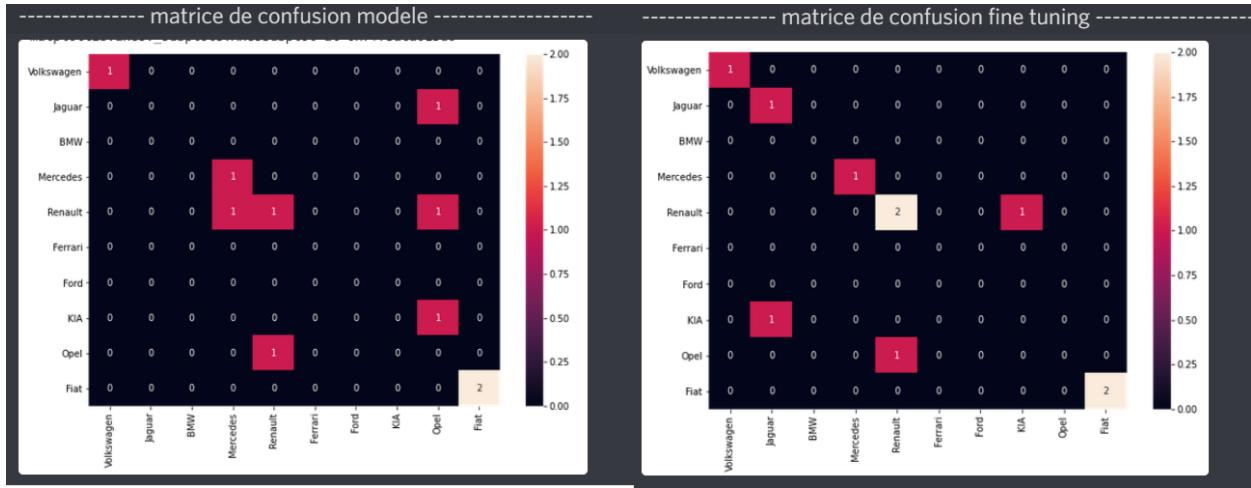


Figure 15: Comparaison des matrice de confusion avec et sans fine-tuning

La comparaison des matrices de confusion confirme ce que nous avions précédemment expliqué. Le réseau se trompe seulement 3 fois sur 10 après fine tuning contre 5 fois avant.

4.3 Analyse globale des résultats

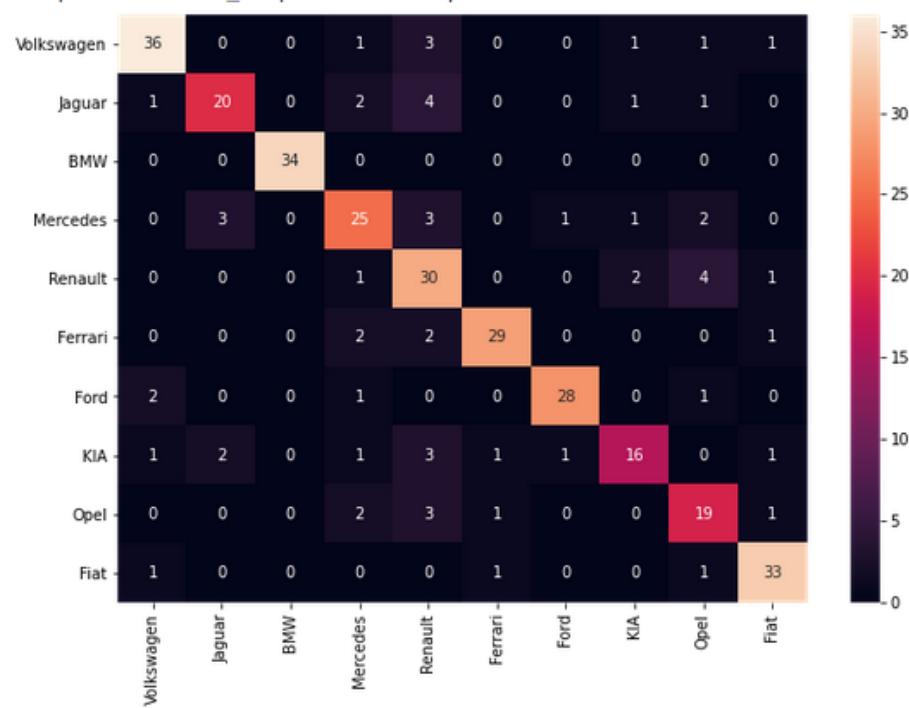


Figure 16: Matrice de confusion après avoir appliqué le réseau sur l'ensemble des images de test

```

Accuracy: 0.83

Micro Precision: 0.83
Micro Recall: 0.83
Micro F1-score: 0.83

Macro Precision: 0.84
Macro Recall: 0.84
Macro F1-score: 0.84

Weighted Precision: 0.84
Weighted Recall: 0.83
Weighted F1-score: 0.84

Classification Report

      precision    recall   f1-score
Fiat          0.86     0.86     0.86
Ferrari       0.87     0.96     0.92
Jaguar        0.81     0.84     0.82
KIA           0.95     0.75     0.84
Ford          0.83     0.91     0.87
Mercedes      0.86     0.84     0.85
Opel          0.63     0.79     0.70
BMW           0.97     0.86     0.91
Volkswagen    0.94     0.81     0.87
Renault        0.73     0.76     0.74

accuracy      0.83
macro avg     0.84     0.84     0.84
weighted avg  0.84     0.83     0.84

```

Figure 17: Tableau récapitulatif des accuracy des différentes classes

Les métriques macro-moyennes sont utilisées lorsque nous voulons évaluer les performances des systèmes sur différents ensembles de données.

Des métriques micro-moyennes doivent être utilisées lorsque la taille des ensembles de données est variable. Recall est la mesure de notre modèle identifiant correctement les vrais positifs.

Dans la reconnaissance de formes, l'extraction d'informations et la classification, la précision est la fraction des instances pertinentes parmi les instances récupérées, tandis que le recall est la fraction des instances pertinentes qui ont été récupérées.

Le score F1 combine la précision et le rappel d'un classificateur en une seule métrique en prenant leur moyenne harmonique. Il est principalement utilisé pour comparer les performances de deux classificateurs. Puisqu'on travaille sur un problème multiclass et que la taille des classes sont proches, les micro-paramètres sont égaux ainsi que les macro-paramètres.

Nous remarquons en général que les prédictions sont bonnes : la précision moyenne étant de 0,83 variant entre 0,97 pour BMW et 0,63 pour Opel. La différence de couleur dans la matrice de confusion n'est pas très significative, car elle traduit une différence en terme de nombre d'images de chaque classes. Kia par exemple a la plus faible valeur d'images correctement classifiées malgré une bonne précision de 0,95. Les micro et macro metrics témoignent tout de même des bonnes performances à travers les différents classes.

5 Difficultés rencontrées

Au début, nous avions utilisé une base de données biaisée, i.e., qu'on avait choisie manuellement et non aléatoirement et qui contenait quelques classes ayant un nombre d'images de tests trop important par rapport à d'autres classes. Ceci a conduit à des résultats de validation non satisfaisant et très instables où

le réseau ne pouvait pas reconnaître correctement les classes quand il s'agissait de photos plus "difficiles".

En utilisant cette base de données, on a commencé au début par deux classes (Renault Ford) qui ont presque le même nombre d'images d'apprentissage. Et en utilisant l'augmentation de données, on a pu aboutir à des résultats acceptable en terme de précision et de loss, mais en évaluant notre réseau sur l'ensemble de test, on a obtenu une accuracy basse, et le modèle n'arrivait pas à reconnaître plus de la moitié des images.

On a passé après à 10 classes, en espérant obtenir de mieux résultats à l'aide du fine-tuning, or les résultats sont devenus incohérents. En effet, le modèle n'arrivait pas identifier les images de plusieurs classes, et la diagonale de la matrice de confusion a été presque vide, sauf pour deux classes (BMW Mercedes) qui avaient plusieurs images par rapport aux autres classes. En plus, on remarquait que la pulpart des images ont été identifiée comme image de BMW ou Mercedes, et le f1-score de ces deux classes a été très élevé par rapport aux autres classes, ce qui prouve que notre modèle est biaisé.

Enfin, on a opté pour l'approche implémenté dans ce rapport, et qui consiste sur l'utilisation d'une base de données bien répartie et non biaisée.

6 Conclusion

Ce projet s'est révélé très enrichissant. D'une part, on a appris que le choix de la base de données est très déterminant dans les problème d'apprentissage profond. En effet, la base de données doit être non biaisée et la répartition de ses classes doit être aléatoire.

D'autre part, on a appris que le travail sur deux classes comme début, donne la possibilité de mieux comprendre le problème pour envisager de meilleures solutions, et de développer le réseau de neurones afin d'obtenir de mieux résultats.

En plus, les techniques d'augmentation de données et de fine-tuning, peuvent faire une grande différence en terme de performances.

7 Annexe

7.1 Google Image Extractor

```
1 from selenium import webdriver
2 from bs4 import BeautifulSoup
3 import requests
4 import urllib.request
5 import time
6 import sys
7 import os
8
9
10 #taking user input
11 print("What do you want to download?")
12 download = input()
13 site = 'https://www.google.com/search?tbm=isch&q=' + download
14
15
16 #providing driver path
17 driver = webdriver.Firefox(executable_path = 'C:\Users\elear\OneDrive\Desktop\Drivers')
18
19 #passing site url
20 driver.get(site)
21
22
23 #if you just want to download 10-15 images then skip the while loop and just write
24 #driver.execute_script("window.scrollBy(0,document.body.scrollHeight)")
25
26
27 #below while loop scrolls the webpage 7 times(if available)
28
29 i = 0
30
31 while i<10:
32     #for scrolling page
33     driver.execute_script("window.scrollBy(0,document.body.scrollHeight)")
34
35     try:
36         #for clicking show more results button
37         driver.find_element_by_xpath("//html/body/div[2]/c-wiz/div[3]/div[1]/div/div/div/div[5]/input").click()
38     except Exception as e:
39         pass
40     time.sleep(5)
41     i+=1
42
43 #parsing
44 soup = BeautifulSoup(driver.page_source, 'html.parser')
45
46
47 #closing web browser
48 driver.close()
49
50
51 #scraping image urls with the help of image tag and class used for images
52 img_tags = soup.find_all("img", class_="rg_i")
53
54
55 count = 0
56 for i in img_tags:
57     #print(i['src'])
58     try:
```

```

59     #passing image urls one by one and downloading
60     urllib.request.urlretrieve(i['src'], str(count)+".jpg")
61     count+=1
62     print("Number of images downloaded = "+str(count),end='\r')
63 except Exception as e:
64     pass

```

7.2 Flickr Scraper

```

1 # Generated by Glenn Jocher (glenn.jocher@ultralytics.com) for https://github.com/
2     ultralytics
3
4 import argparse
5 import os
6 import time
7
8 from flickrapi import FlickrAPI
9
10 from utils.general import download_uri
11 key = '' # Flickr API key https://www.flickr.com/services/apps/create/apply
12 secret = ''
13
14
15 def get_urls(search='honeybees on flowers', n=10, download=False):
16     t = time.time()
17     flickr = FlickrAPI(key, secret)
18     license = () # https://www.flickr.com/services/api/explore/?method=flickr.photos.
19     licenses.getInfo
20     photos = flickr.walk(text=search, # http://www.flickr.com/services/api/flickr.photos.
21                           search.html
22                           extras='url_o',
23                           per_page=500, # 1-500
24                           license=license,
25                           sort='relevance')
26
27     if download:
28         dir = os.getcwd() + os.sep + 'images' + os.sep + search.replace(' ', '_') + os.sep
# save directory
29         if not os.path.exists(dir):
30             os.makedirs(dir)
31
32     urls = []
33     for i, photo in enumerate(photos):
34         if i < n:
35             try:
36                 # construct url https://www.flickr.com/services/api/misc.urls.html
37                 url = photo.get('url_o') # original size
38                 if url is None:
39                     url = 'https://farm%s.staticflickr.com/%s/%s_%s_b.jpg' % \
40                           (photo.get('farm'), photo.get('server'), photo.get('id'), photo.
41                           get('secret')) # large size
42
43                 # download
44                 if download:
45                     download_uri(url, dir)
46
47                 urls.append(url)
48                 print('%g/%g %s' % (i, n, url))
49             except:
50                 print('%g/%g error...' % (i, n))
51
52     # import pandas as pd
53     # urls = pd.Series(urls)

```

```

51 # urls.to_csv(search + " urls.csv")
52 print('Done. (%.1fs)' % (time.time() - t) + ('\nAll images saved to %s' % dir if
53 download else ''))
54
55 if __name__ == '__main__':
56     parser = argparse.ArgumentParser()
57     parser.add_argument('--search', type=str, default='honeybees on flowers', help='flickr
58 search term')
59     parser.add_argument('--n', type=int, default=10, help='number of images')
60     parser.add_argument('--download', action='store_true', help='download images')
61     opt = parser.parse_args()
62
63     # Check key
64     help_url = 'https://www.flickr.com/services/apps/create/apply'
65     assert key and secret, f'Flickr API key required in flickr_scraper.py L11-12. To apply
66     visit {help_url}'
67
68     get_urls(search=opt.search, # search term
69               n=opt.n, # max number of images
70               download=opt.download) # download images

```