Rapport de projet

Module Analyse et Traitement d'Images

Classification d'images

Crusson Mathieu

Introduction

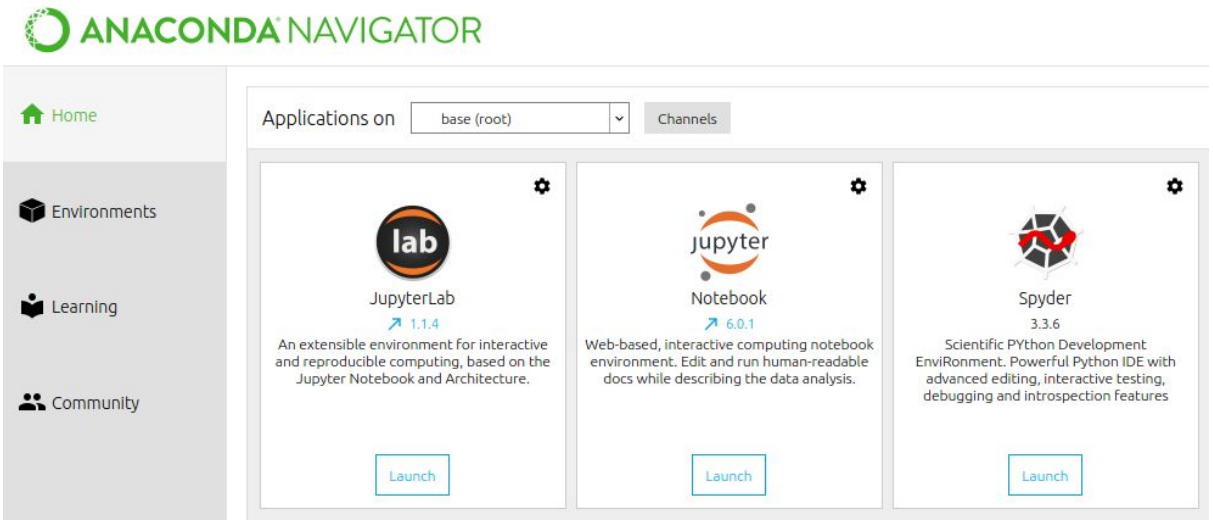
L’objectif de ce projet était de mieux appréhender les différentes technologies, procédés et efficacités, des solutions d’analyse et traitement d’image. Nous avons donc pu étudier deux méthodes utilisées, une plus ancienne se basant sur un calcul d’attribut pour chaque image, et une autre plus moderne qui se base sur l’emploi d’un réseau de neurones.
Après une présentation de l’environnement de travail et du traitement des données, ce rapport présentera le travail réalisé sur chacune de ces deux technologies.

Sommaire

Environnement de travail et installation

Comme indiqué dans l'énoncé, la programmation sera réalisée en Python. Pour pouvoir installer rapidement et efficacement toutes les librairies nécessaires à la réalisation du projet, j'ai pu installer et utiliser anaconda-navigator sous linux qui permet l'installation des logiciels et des dépendances qui leurs sont associées.

Après avoir testé les différentes options présentes dans le sujet, j'ai choisi de travailler sur l'IDE Spyder qui se rapprochait le plus de mes habitudes de développement.



J'ai préféré utiliser une solution comme Spyder comparé aux autres propositions comme Codelabs ou Jupyter pour sa simplicité d'utilisation. Je pouvais grâce à cet IDE, développer et exécuter mes programmes rapidement, tout en conservant mes données en local. Pour conserver cette simplicité d'utilisation, j'ai également ajouté les différentes librairies et packages nécessaires à l'environnement de travail root (base) de la machine, sans passer par l'utilisation d'un environnement virtuel de travail, comme je pourrais le faire pour d'autres projets.

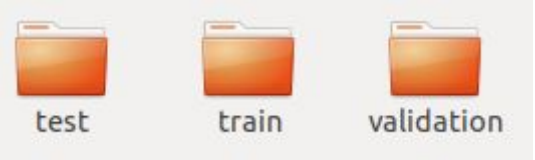
Récupération et organisation des données

Pour répondre aux besoins du TP, il a fallu récupérer et classer une base de données d'images utilisées à des fins de recherche. C'est la base COREL qui à été utilisée. Elle comprend un très grand nombre d'images réparties en plusieurs classes ce qui permet d'effectuer des tests sur une banque d'image de grande capacité avec une grande diversité. Avec l'utilisation d'un nombre réduit de classes, nous pourrons tester assez rapidement les méthodes d'extraction d'attributs et de classification.

Comme indiqué dans le sujet, lorsque l'on veut s'attaquer à un problème de classification de données de type supervisé, il faut prendre garde à bien spécifier les différents jeux de données à utiliser pour que la classification soit consistante et suffisamment robuste dans son exploitation.

Ainsi, la base de données utilisée devra être répartie en 3 sous dossiers de manière inégale :

1. Un répertoire d'apprentissage, ou train, pour entraîner le programme de classement d'image.
2. Un répertoire validation, pour optimiser les résultats obtenus avec le 1er répertoire
3. Le répertoire de test, qui servira à tester le programme de classement pour déterminer son efficacité.



Une fois la base de données correctement répartie, j'ai pu commencer les modifications du code initial et la réalisation de tests selon les différentes méthodes proposées.

Classification d'images basées sur attributs

Pour la classification d'images basée sur attributs, chaque instance de la base d'images fait l'objet d'un calcul suivant différentes méthodes (histogramme des couleurs, contours, daisy...). Le but est de renvoyer à l'utilisateur les images répondant le mieux à la requête.

Classification d'image basées sur les couleurs (color.py)

Dans un premier temps, j'ai pu modifier la classe color.py fournie par le sujet pour l'adapter à l'utilisation de différents jeux d'images et réaliser une classification par couleurs des images.

```
    if isinstance(input, np.ndarray):  # examinate input type
        img = input.copy()
    else:
        # img = scipy.misc.imread(input, mode='RGB')
        img = imageio.imread(input)
```

Concernant la lecture d'une image, il a fallu remplacer la fonction scipy.misc.imread par l'utilisation de la librairie imageio.

Pour utiliser la fonction color.py avec la répartition des images souhaitée, du code a été ajouté dans la fonction main.

Ce code permet, après s'être entraîné sur le base de train, de tester les images de la base test, de les trier et de les enregistrer dans leurs dossiers respectifs dans le répertoire de résultat.

```
DB_train_dir = "path de la base de train"
DB_train_csv = "path du csv de la base train"

db1 = MyDatabase(DB_train_dir, DB_train_csv)
print("DB length: ", len(db1))
# data = db.get_data()
color = Color()

#Database de test
DB_test_dir = "path de la base de test"
DB_test_csv = "path du csv de la base test"
```

```
db2 = MyDatabase(DB_test_dir, DB_test_csv)
print("DB length: ", len(db2))
```

Concernant les résultats, avec une base d'image de train de 450 fichiers, une base de train et de test de 90 fichiers, le résultat est plutôt correct. On retrouve les images globalement bien triées selon leurs couleurs. Cependant, on remarque que plusieurs erreurs se trouvent dans les répertoires de résultat. En effet, quelques images sont classées dans d'autres catégories que celle leur appartenant. On peut donc retrouver des erreurs avec ce classement par couleur pour ce nombre d'images testées.

Classification d'image basées sur les formes (edge.py)

On applique des modification similaires au code modifié de color.py pour le traitement dans le main et la modification des databases. Les résultats obtenus sont plus précis qu'avec la classification par couleur, on constate beaucoup moins d'erreurs avec cette méthode de classification.

```
 # evaluate database
     APs,   res   =   myevaluate(db1,   db2,   edge.make_samples,
depth=depth, d_type="d1")
     cls_MAPs = []
     for cls, cls_APs in APs.items():
     MAP = np.mean(cls_APs)
     print("Class {}, MAP {}".format(cls, MAP))
     cls_MAPs.append(MAP)
     print("MMAP", np.mean(cls_MAPs))
```

Classification d'image basées sur les textures (gabor filter)

Le filtre de Gabor permet la classification d'images par textures. Le filtre de Gabor permet d'analyser les fréquences présentes sur une image pour en établir le profil. On peut également retrouver ces notions de fréquences par régions dans le code fourni.

Lors de mes tests, je n'ai malheureusement pas réussi à faire fonctionner, même partiellement les fichiers gabor et fusion.py.

Classification d'images basée sur un réseau de neurones

Le but est de partitionner au mieux des points en dimension 2 issus d'un ensemble d'apprentissage, en deux classes labellisées, tout en veillant à prédire au mieux la classe d'appartenance des points appartenant à un ensemble de test.

D'après le forum cité dans l'énoncé, j'ai pu récupérer le code nécessaire pour mettre en place un réseau de neurones convolutif peu profond.

```
# dimensions of our images.
img_width, img_height = 150, 150
#path des répertoires d'images
train_data_dir = 'data/train'
validation_data_dir = 'data/validation'
#définition du nombre d'échantillons
nb_train_samples = 2000
nb_validation_samples = 800
epochs = 20 #Nombre de répétitions
batch_size = 16 #Nombre d'image traitées simultanément
```

Après avoir défini les paramètres d'utilisation, on définit les différentes couches du réseau de neurones selon notre besoin

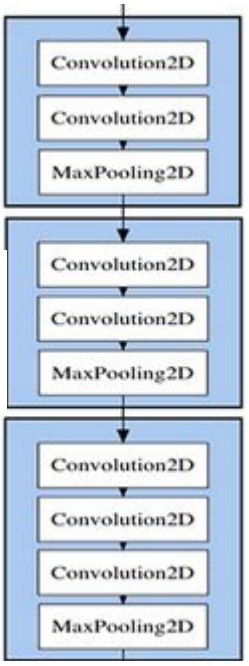
```
# Première couche
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

#Deuxième couche
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

#Troisième couche
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Le modèle utilisé consiste en un empilement de couches avec un nombre de filtre croissant suivant l'augmentation du nombre de couche. Chaque couche est composée d'une partie de convolution 2D et d'un pooling 2D. Les paramètres de ces fonctions peuvent varier selon l'utilisation souhaitée comme nous pourrons le voir plus tard.

On a donc un modèle de ce type dont le nombre de filtre de sortie augmente de 32 à 128.



Avec un modèle de ce type, et avec un nombre de répétition de 20, on obtient les résultats suivants :

```
Epoch 18/20
 - 44s - loss: 0.0047 - acc: 0.9990 - val_loss: 0.2270 - val_acc: 0.9859
Epoch 19/20
 - 40s - loss: 5.7155e-07 - acc: 1.0000 - val_loss: 0.2270 - val_acc: 0.9859
Epoch 20/20
 - 41s - loss: 0.0140 - acc: 0.9990 - val_loss: 0.1336 - val_acc: 0.9859
```

On remarque que les résultats pour un nombre de répétition relativement faible sont très bons, on dispose d'une précision de 0.99 pour une temps de traitement de 41 secondes par répétition.

Malgré un temps d'exécution beaucoup plus long. On constate que le classement d'image par l'utilisation d'un réseau de neurones est beaucoup plus performantes que la classification par attributs vue précédemment.

Par la suite, j'ai pu expérimenter une classification par réseau de neurones avec une compilation de modèle par categorical_crossentropy à l'instar de la binary_crossentropy utilisée jusque la. Pour accomplir ces tests, plusieurs modifications ont été effectuées sur le code initial utilisé pour les précédents tests:

- Augmentation du nombre de classe, donc du nombre d'images à tester
- Modification de la taille des images pour s'adapter à la base de données CorelDB

```
img_width, img_height = 120, 120
```

- Modification du mode de compilation avec modification de l'optimizer

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

- Changement des paramètres des fonctions de convolution et pooling

```
model.add(Conv2D(64, (5, 5)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

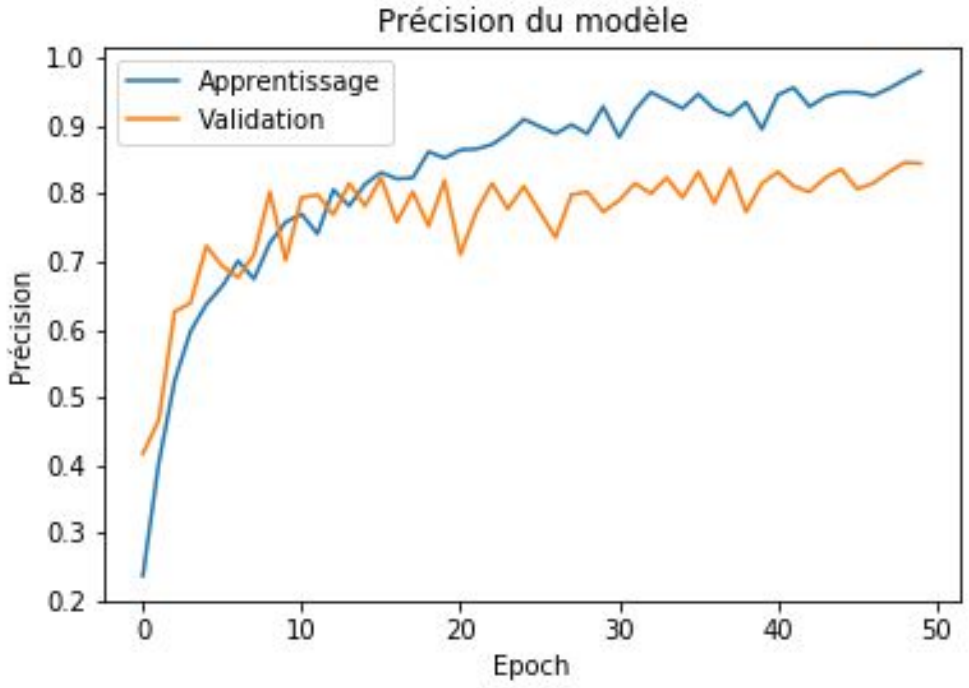
- Sauvegarde du modèle et affichage par comparaison des performances sur les échantillons de test et de validation

```
#Sauvegarde et Affichage du model
model.save('CorelDB_model.h5')
plot_model(model, to_file="model_v1.png")

#Affichage
plt.figure()
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title("Précision du modèle")
plt.ylabel("Précision")
plt.xlabel("Epoch")
plt.legend(["Apprentissage", "Validation"], loc="upper left")
plt.savefig('precision.png')
plt.show(block='false')
```

Pour configurer l'affichage, on a affecté la fonction de model.fit_generator à un paramètre que l'on va pouvoir appeler lors de l'affichage.

Résultats :



Pour un nombre d'époques de 50 et plus de 1000 images (752 pour l'apprentissage et 254 pour la validation), on constater que la précision augmente rapidement pour les deux échantillons d'image. Cependant, après la 20e époque, on constate légère croissance en faveur du set d'apprentissage. Malgré tout, avec un taux de précision de plus de

Conclusion

Le but de ce projet était de mettre en application les connaissances acquises lors des cours de ce module d'analyse et traitement d'images. J'ai ainsi pu mettre en application les notions vues en cours sur ce sujet pour mettre en place des solutions de classification d'images selon les différentes technologies à ma disposition.
Pour conclure sur les aspects techniques de la classification, on peut affirmer que la classification par réseau de neurones est beaucoup plus performante que la classification par attributs. L'inconvénient de cette dernière est que son utilisation sur des équipements grand public plus longue que la première méthode. Cependant, avec des moyens de calculs plus performants ou une meilleure optimisation du code (parallélisation), il est possible de réduire ce temps de calcul pour obtenir des résultats fiables dans un temps donné acceptable.