Master 1 Informatique

Rapport de Projet : Neural Network And Learning

Sujet :

Construction d'un classificateur de photo de scènes naturelles

Yann MARTIN D'ESCRIENNE
Yohann TOGNETTI

« Année universitaire 2020 - 2021 »

Table des matières

1 Présentation générale du projet

Dans le cadre de notre cours de Neural Network and Learning, il nous a été demandé d'effectuer un projet impliquant une intelligence artificielle travaillant sur des images. Celle-ci a pour but de construire un classificateur de photo de scènes naturelles comme ceux qu'on pourrait intégrer dans un appareil photo intelligent qui à chaque prise de photo va taguer l'image avec le nom d'une catégorie qui y correspond.

2 Descriptif du sujet et ses finalités

2.1 Descriptif du sujet

Ainsi, pour entrer dans les détails le but qui est de "taguer l'image" correspond au fait de la catégoriser grâce à un classificateur entrainé possédant les catégories de scènes naturelles désirées. Pour cela avons dû utiliser un réseau neuronal convolutif étant donné que nous traitons des images. Une fois la réponse du classificateur reçue, elle correspond initialement à un entier, nous avions donc le devoir de le convertir en tag correspondant au nom de la catégorie trouvée.

L'affichage du résultat doit se faire à l'aide d'une méthode *predict* qui prend en entrée le chemin de fichier d'une image et une chaine de caractères 'mode' qui renvoie le nom de la catégorie correspondant à l'image si **mode='category'** où un vecteur de probabilités si **mode='probabilities'** ou chaque éléments du vecteur indique la probabilité que l'image appartienne à la catégorie correspondante.

2.2 Les catégories

Celles-ci sont au nombre de 6, chacune correspond à une catégorie de paysage ou de scène naturelle de la liste suivante :

- buildings
- forest
- glacier
- mountain
- sea
- street

Building correspond ainsi à des bâtiments comme une maison ou un immeuble qui se trouve généralement dans une ville.
Forest comme son nom l'indique correspond à un paysage forestier avec la présence ou non d'animaux.
Glacier quant à lui représente une montagne enneigée ou bien un glacier ou morceaux de glace de la banquise, cette catégorie est extrêmement corrélée avec *Moutain* car celle-ci correspond à une montagne pouvant parfois aussi être enneigé. Le découpage n'est pas toujours pertinent et encore plus ici, nous en parleront dans la section *Description des données*.
Sea représente les décors marins en général, que ce soit une plage, la mer avec ou sans terre ferme, ou bien même un décor sous-marin, tout cela va dans cette catégorie.
Enfin la catégorie *Street* met en avant une rue, qui peut donc facilement se rapprocher de *building* étant donnée la présence quasi-permanente de bâtiments dans une rue, on y trouve souvent des passant et des voitures.

3 Description des données

3.1 Le jeu de données

Les données utiliser pour pouvoir réaliser cet apprentissage viennent du jeu de données « Intel Image Classification ». Ce jeu de données est composé d'un totale de 24.347 images et elles sont toutes de taille 150x150 px. Le jeu de données est réparti en trois fichier, seg_train, seg_test et seg_pred.
Pour les fichiers de train et de test, sont tous deux repartis en 6 sous fichier qui sont nos classification (street, buildings, ...) et qui possèdent chacun les image classifier correspondantes.
Le fichier train possèdent en tout 14.046 image repartie avec les proportions plutôt équivalente comme nous pouvons le voir :

- Buildings 2.191 images
- Forest 2.271 images
- Glacier 2.416 images
- Mountain 2.512 images
- Sea 2.274 images
- Street 2.382 images

Le fichier test possèdent en tout 3000 image repartie avec les proportions plutôt équivalente comme nous pouvons le voir :

- Buildings 437 images
- Forest 474 images
- Glacier 553 images
- Mountain 525 images
- Sea 510 images
- Street 501 images

Les deux jeux de données ont un nombre de données plutôt important pour chaque classe, cela est plutôt favorable pour pouvoir faire une classification correcte.
Le dernier répertoire, celui de tes possèdent 7.301 qui sont de toutes classe.

3.2 Des données pas forcement bien classifiée

L'une de nos découvertes sur les jeux de données était que certaine image était mal classifiée. Voici certains exemples qui illustre nos propos :

FIGURE 1 – Image 20662.jpg, catégorie Mountain du jeu de test.

FIGURE 2 – Image 23589.jpg, catégorie Glacier du jeu de test.

Ces deux images n'ont pas la même catégorie et pourtant, elles se ressemble fortement car ce sont de montagne enneiger. Les catégorie Mountain et Glacier sont toutes deux correct, mais dans ce cas-là l'ia vas surement se tromper sur l'une des deux.
Un autre type de problème sont les images hors contexte donc difficilement classifiable comme celle-là :

FIGURE 3 – Image 1512.jpg, catégorie glacier du jeu de train.

Certaines images vont donc automatiquement faire descendre la précision car même pour un nous il est difficile de classifier comme ce qui est attendue. Néanmoins, La plupart des autres données sont correct et vont nous permettre de faire une bonne classification.

4 Implémentation de l'algorithme

Notre algorithme est développé en python. Il se trouve dans le fichier *Program.py* et possède 3 méthodes majeures qui ont permis de répondre aux attentes du sujet. Nous allons donc présenter chacune de ces méthodes avec un peu plus de détails.

La réussite de ce sujet est grandement due au livre **L'APPRENTISSAGE PROFOND AVEC PYTHON** de *François CHOLLET*. La lecture de son travail nous à permis un apprentissage rapide du fonctionnement des réseaux neuronal convolutif ainsi que leur mise en application pour notre IA.

4.1 preprocessing

Cette méthode prend comme argument le directory qui contient les images avec la structure décrite dans la partie *Description des données* afin d'effectuer le prétraitement.

Dans un premier temps cette méthode va donc créer un répertoire pour les données d'entrainement, puis un autre pour les données de validation en ce basant sur le répertoire donné en argument.

Ensuite, deux *ImageDataGenerator* vont être créer avec une mise à l'échelle de $1/255$. En effet il est beaucoup plus simple pour l'IA de travailler avec des données entre 0 et 1 que 0 et 255 concernant le RGB de chaque pixels. Le premier ImageDataGenerator va servir pour les données d'entrainement et le deuxième pour la validation.

Finalement deux générateurs vont être créés pour les deux parties avec comme taille voulue 150x150 pixels étant donnée que cela correspond à la taille des images dans le jeu de données. Le *batch_size* est de **32** et le *class_mode* est en **binary** étant donné que nous avons comme résultat un vecteur constitué de 0 et de 1.

Les générateurs d'entrainement et validation ainsi créés vont être retourné par la méthodes afin de les utiliser dans les autres.

4.2 trainModel

Cette méthode à pour but de créer et entrainer le modèle. Elle prend en argument les deux générateurs créés à la partie précédente, ainsi que des paramètres propres à l'entrainement de l'ia comme le nombre d'epoch, le nombre d'étape par epoch ainsi que le chemin d'enregistrement du modèle qui sera créé.

4.2.1 Description du modèle

La construction de notre modèle actuel est comme ci-dessous :

conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
dropout (Dropout)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 36, 36, 64)	0
dropout_1 (Dropout)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2	(None, 17, 17, 128)	0
dropout_2 (Dropout)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_3 (MaxPooling2	(None, 7, 7, 128)	0
dropout_3 (Dropout)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dropout_4 (Dropout)	(None, 6272)	0
dense (Dense)	(None, 512)	3211776
dropout_5 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 6)	3078

FIGURE 4 – Résumé du model

La partie principale de notre modèle se constitue donc de plusieurs blocs composés de la manière suivante : un **2D convolution layer** suivi d'un **Max pooling operation for 2D spatial data** et se terminant par un **dropout layer**. A la fin de ces paternes

un **flatten layer** est mit en place suivi d'un nouveau **dropout**. Notre modèle se termine enfin par deux **dense layer** avec un autre **dropout** entre eux

Les convolution layers utilisent tous de strides de taille 3x3 et ont pour fonction d'activation *Rectified Linear Unit*. Le première couche utilise comme input_shape des images RGB de taille 150x150 afin de s'adapter à nos données.
Les maxpooling layers ont quant à eux un strides de taille 2x2.
Tout les dropout mis en place dans ce modèle ont comme paramètre 20% des données abandonnées. Ceci afin de limité l'overfitting de nos données.
Le premier dense layer prend comme paramètre 512 dimensions avec comme fonction d'activation *Rectified Linear Unit* et utilise également un *kernel_regularizer*. Le deuxième quant à lui possède 6 dimensions, étant donné que l'on souhaite 6 classes et utilise *softmax* comme fonction d'activation.

La compilation du modèle se fait avec un optimiser *adam* et une fonction de perte *categorical_crossentropy* et se focalise sur la précision du modèle.

Finalement nous entrainons le modèle avec les paramètres donnés précédemment et le stockons dans history afin d'afficher sur un graphe l'évolution de la précision au cours de l'entrainement. Nous sauvegardons au passage le modèle dans le chemin donné en argument.

4.2.2 Evolution du modèle

Grâce au nombreux exemples du livre qui nous à servis de base, notre premier modèle possédait déjà une précision aux alentours de 80% pour les données de validation et 98% pour celle d'entrainement. Ceci était déjà un très bon résultat mais notre modèle souffrait donc d'**overfitting**.

Nous avons remédier au problèmes en utilisant des dropout layers afin que notre modèle ne se focalise pas trop sur ses données d'entrainement uniquement. Nous avons choisi d'abandonner 20% de nos résultat après chaque paternes conv2D/maxpooling. On peut ainsi constater la différents de précisions des données de validation dans les graphiques suivant :

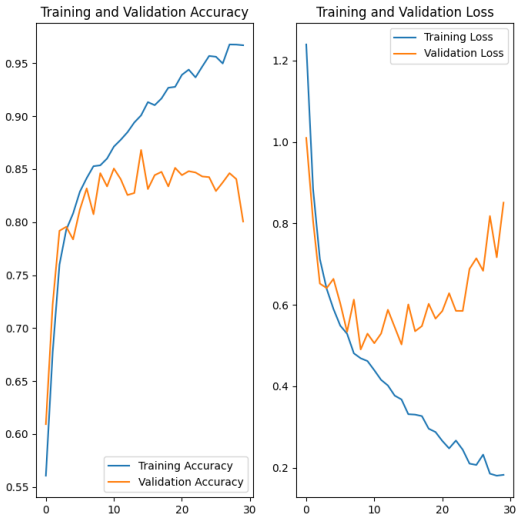

FIGURE 5 – Précision sans dropout

FIGURE 6 – Précision avec dropout

On voit alors très clairement que la précision de validation est beaucoup plus proche

de celle d'entrainement même si celle-ci à donc tendance à être plus basse, s'expliquant par l'abandon de 20% des résultats obtenus.

4.3 predict et méthodes graphiques

Cette méthode correspond aux attentes du sujet pour afficher les prédictions de notre modèle avec le mode attendu qu'il prend donc en argument. Il prend également pour paramètre le modèle entrainé ainsi que le chemin de fichier de l'image à prédire.

Enfin, pour mettre sous forme de diagramme ou de graphiques les résultats obtenus qui seront présentés dans la section suivante, nous nous sommes appuyer sur la librairie matplotlib. Nous avons donc mit en place différentes méthode pour ceci, ainsi qu'un autre programme appelé *report.py* pour l'affichage graphique de la matrice de confusion.

4.4 Temps d'apprentissage du modèle

Le temps d'apprentissage du modèle dépend bien évidemment du nombre d'epoch ainsi que du nombre d'étapes par epoch choisi. Notre modèle final se base sur 60 epoch de 430 étapes chacune. Bien évidemment lors de nos test d'améliorations, nous ne mettions que 30 epoch de 120 étapes afin de voir rapidement les résultats.

Sur le pc le plus performant du groupe, le modèle final prend environ 15-20 min pour s'entrainer et lors de nos test environ 2-3 minutes. Cela viens du fait de l'utilisation de l'installation des dll utilisant la carte graphique et cœurs cuda pour l'apprentissage de l'IA. Néanmoins ce temps dépend bien sur fortement du matériel utilisé et de sa configuration.

5 Résultat obtenus

Il est à noter que les résultats peuvent être légèrement différent à chaque appel du programme ainsi les graphiques qui vont suivre ne sont pas forcément absolus.

5.1 Résultat général

Il est facilement remarquable sur la *figure 5* que la valeur 0 (donc l'absence du couple E#C) possèdent un bien meilleur score que la valeur 1. En effet un commentaire parlant d'un clavier uniquement ne va pas parler d'un écran ou d'une souris, ce qui créer un nombre conséquent de données négative (c-à-d 0) pour chaque classes. Il est alors plus facile pour l'IA de déterminé si la phrase n'appartient pas à un couple que l'inverse.

On constate effectivement que le f1-score des valeurs 1 de chaque classe est assez variable et atteins parfois même 0%. Cela s'explique avec deux raisons :
Soit par une *précision* de 0%, car certain couple dans les données de test n'ont qu'une apparition ce qui rend le résultat binaire ou bien simplement par l'échec de l'IA à classer correctement les phrases (les justifications de cette lacune sera expliquée dans l'analyse du résultat suivant).
Soit par un *recall* faible, car l'IA peut facilement trouvé la bonne entité mais pas la bonne catégorie et vis-versa, et cela se traduit par une réduction du recall et donc du f1-score.

Comme dit précédemment, la valeur du f1-score des valeurs 0 est très élévé, environ 95%, ainsi ne nous y attarderont pas plus que ça dans les analyses suivantes et nous concentrerons plus sur les résultats et scores obtenus pour les valeurs 1.

5.2 Résultat pour la valeur "1"

On remarque premièrement que les précisions de chaque couple E#C ne sont pas homogène. Cela s'explique par la différence majeure entre chaque entités et les catégories qui s'y rapportent. En effet il est beaucoup plus simple et parler de l'ergonomie d'une souris (ou d'un pavé tactile) que de l'ergonomie d'un OS, ce qui explique par exemple l'écart de précision entre ces deux couples.

De plus, certain couples possèdent un jeu de données qui se veut parfois peu ressemblant entre chaque élément y appartenant. Par exemple, dans le commentaire d'un clavier, il y aura beaucoup de chance de trouver les mots "touche","clavier" ou bien "pavé numérique", ce qui focalise l'IA sur certain éléments clés et ressort de tout cela une bonne précision. Mais par exemple, le couple *COMPANY#GENERAL* peut très bien parler de *Windows* comme d'*Apple*, *Dell* , *toshiba* ... de mille et une façon, ce que l'IA a du mal à assimiler.

Enfin, une relation logique apparait entre le support (nombre d'éléments) et la précision. Moins on a de données plus la précision est extrême (0% ou bien 100%) car soit l'IA a tout juste, soit tout faux. Au contraire, lors d'une présence d'un nombre de données assez conséquent (environ 10), la précision tend à dépasser les 50% ce qui pourrait être plus représentatif du niveau de classification. Typiquement, le nombre de données le plus élevé (214) est celui du couple *LAPTOP#GENERAL* (ce qui est assez logique en soit) et il possède une précision de 71%.

5.3 Résultat par entités

Suite à de nombreux tests sur des classifications de phrases, nous avons remarqué que l'IA se trompe parfois dans le ou les couple E#C mais arrive tout de même à reconnaitre la ou les entités E qui sont ciblés dans la phrase. Nous avons donc regroupé par entité pour plus de pertinence et avons calculé la *precision*, le *recall* et le *f1-score* de chacune.

Là encore attention, malgré le regroupement par entité, certaines n'ont qu'un seule données, ce qui explique les scores à 50%. De plus, nous avons récupéré la macro-moyenne (moyenne brute) et non la moyenne pondérée car celle-ci est figé à 99% pour toutes les entités (dû à grand nombre de cas à 0 et très peu d'erreur de l'IA dessus comme expliqué précédemment), il n'est donc pas intéressant de faire une étude dessus. Cette macro-moyenne se retrouve donc à cette valeur lorsqu'il n'y a qu'une donnée pour la valeur 1 et que l'IA ne retrouve pas la bonne entité E suivant le simple calcule suivant :

$$(99 + 0)/2 \simeq 50\%. \tag{1}$$

Mis à part ces cas précis, les scores obtenus sont relativement corrects et on peut facilement affirmer que l'IA arrive à reconnaitre les entités cible d'un commentaires. Toutefois, nous avons remarqué que lorsque la phrase aborde différentes entités de manière brève, par exemple *"L'ordinateur est cool, le clavier et le son sont correct mais l'affichage est magnifique"*, la classification de toutes ces entités semble très difficile pour l'IA.

5.4 Résultat de la polarité

Tout d'abord voici le nombre de données de base par polarité :

— negative : 768
— positive : 917
— neutral : 430
— mixed : 46

On constate que la *précision*, le *recall* et le *f1-score* sont relativement proche pour chaque polarité ce qui ne souligne pas de problème particulier dans une classe. Il est notable que *positive* et *negative* sont les deux classes qui comportent le plus de données, il est donc normal que leur score soit les plus élevés.

Le fait d'avoir un commentaire positif semble d'ailleurs plus facile à détecter qu'un commentaire négatif. Cela s'explique par le fait que souvent, un commentaire négatif consiste en l'énumération des défauts de l'ordinateur sans forcément de mot clé signifiant une insatisfaction ou un mécontentement, l'IA à donc plus de mal à déterminé si cela est négatif (une énumération de fait pouvant être neutre).

Ainsi, la polarité *neutral* peut ressortir également de plusieurs tournure de phrase. Aucun mot clé n'est propre à un avis neutre, ce qui rend plus difficile sa classification et lui donne des score relativement médians.

Finalement, la polarité *mixed* possède le score le plus bas car d'une part, le jeu de données est très faible relativement aux autres, mais également, c'est la polarité la plus dure à classifiée et n'existe pas réellement. Nous l'avons créer dû à la simplification que nous avons faite d'avoir une polarité générale et non propre à chaque couples E#C. Il est donc évident qu'elle n'est pas absolue et consiste à la fusion d'un avis positif et négatif.

6 Les améliorations possibles

Globalement, pour nous ce projet n'a pas tant d'amélioration à faire que cela. Les nombres de données pour construire le modèle est largement suffisant, il n'y a donc pas eu à recourir à des transformations d'image pour lutter contre un manque de données. Elles sont plutôt diverses et équilibrées, il ne faut donc pas mettre en place de système de pondération pour les classes.

6.1 Travail sur le jeu de données

Comme expliqué précédemment, le jeu de données met certaines images dans des classes qui n'ont pas toujours un réel rapport ou alors en met d'autres quasiment identiques dans des classes différentes. Résoudre ce problème en supprimant ou déplaçant toutes ces images problématiques dans les bonnes catégories permettrait alors à l'IA de mieux savoir différents ces différentes images et ainsi obtenir une meilleure précision.

6.2 Travail sur le modèle

Bien évidemment, notre modèle est loin d'être parfait. Par manque de temps, nous n'avons pas expérimenter toutes les possibilités et d'autre méthodes d'apprentissage de classifications d'image existent. Notre précision (87%) reste néanmoins plus que correcte, mais il aurait peut être été possible d'atteindre 90%.

7 Conclusion

Pour conclure sur notre travail sur le sujet donné, l'apprentissage et la construction du modèle de fût pas le plus difficile à mettre en place et obtenir rapidement une bonne précision.
Cela s'explique par le très bon livre donné en référence de François CHOLLET, celui-ci contenait toutes les informations nécessaire à la réussite du sujet mais aussi à l'apprentissage profond neuronal en général. Certaines solutions pour lutter contre l'overfitting y était même présente. Il nous a ainsi fallu compléter toutes ces informations par quelques recherches supplémentaire sur internet afin d'obtenir notre résultat final.
La classification d'image étant quelques choses que nous n'avions encore jamais abordé, nous avons donc pu constater que ce n'est pas excessivement compliqué. A présent, nous avons le savoir minimal nécessaire afin d'étendre la classification d'image à n'importe quels problèmes pouvant être assimilé à ce projet, la seule différences étant le jeu de données et les catégories différentes.

8 Membres du groupe

8.1 Yann MARTIN D'ESCRIENNE

Ce projet m'a beaucoup apporté. Il est pour moi une bonne finalisation de tout les TPs que nous avons pu faire en Neural Network and Learning. Le confort du bon jeu de données m'a permis de mieux m'investir dans le projet, malgré quelques surprises dans certaines catégories.

Nous avons travaillé en groupe quant à l'implémentation du modèle mais également dans tout le projet en général, avançant ensemble de façon synchrone sur le projet. J'ai majoritairement travaillé sur la phase de preprocessing et de prédiction. J'ai également recherché de nombreuses informations dans le livre de François CHOLLET afin de pouvoir répondre à toutes nos questions et attentes du sujet. Je me suis occupé de rédiger les parties 1,2,4,6 et 7 du rapport.

8.2 Yohann TOGNETTI

9 Références

— Deep Learning with Python by François Chollet
— TensorFlow Conv2D Layers : A Practical Guide :
https://missinglink.ai/guides/tensorflow/tensorflow-conv2d-layers-practical-guide/
— Keras documentation :
https://keras.io/api/layers/
— Overfit and underfit | TensorFlow Core :
https://www.tensorflow.org/tutorials/keras/overfit_and_underfit#strategies_to_prevent_overfitting
— Image classification | TensorFlow Core :
https://www.tensorflow.org/tutorials/images/classification
— WIKIPEDIA FR :
https://fr.wikipedia.org/wiki/Wikip%C3%A9dia:Accueil_principal

FIGURE 7 – précision et support de la valeur 1

FIGURE 8 – score par entités

FIGURE 9 – score de la polarité