Application de reconnaissance de formes
Le Détecteur de Chocobo

par Sacha Le Doeuff

Usage de l’application :

Sont fourni avec ce rapport les script suivant :

- Chocobo1DArray.py
- ChocoboConvolution.py
- ResizeAndRotate.py

Les données ne sont, comme demandé, pas fournies avec le rapport et l’application. Pour constituer son corpus, il faut créer quatre dossier du nom de « Oui », « Non », « OuiChoco » et « NonChoco ». Il suffit de mettre des image de Chocobo dans le dossier Oui, et des images autres dans le dossier Non.
Il faut exécuter le fichier ResizeAndRotate pour effectuer le prétraitement des images, puis après choisir la méthode avec laquelle les images seront traités. Si on désire ajouter de nouvelle donner, vider les dossier « OuiChoco » et « NonChoco », insérer les images dans « Oui » ou « Non », puis relancer le script de prétraitement.

Partie 1 : Description de l’application et du jeu de données

Type d’application :

J’ai choisi de faire une application de détection d’objet dans une image. Ici, le but pour l’application sera de reconnaître un Chocobo dans une image, en disant simplement si oui ou non l’objet présenté en est un.

Type d’image traitée :

Ici, le corpus consiste en 600 images au format JPEG et en couleur. Il s’agit originalement d’un set de 150 images, mais des rotations ont été effectuées sur chacune d’entre elles pour augmenter le jeu de données. J’ai choisi d’utiliser des images en couleur pour avoir un élément discriminant supplémentaire.Ic

Données à utiliser :

Ces 600 images sont séparés en deux catégories :

- 300 images représentant des Chocobos sur fond blanc ou uniforme, sous plusieurs angles et dessinés de façons différentes
- 300 images qui ne sont pas des Chocobos mais représentent d’autres animaux, des objets ou des êtres humains. Pour tester la robustesse de l’application, quelques images ayant des point commun avec des Chocobos ont été utilisées. Par exemple des images de poussins, de fruit jaune ou de voiture jaune.

Paramètres à extraire de chaque image :

Les images sont extraites pixel par pixel. Pour chaque pixel est récupéré trois valeurs entre 0 et 255 qui représentent chacune respectivement les valeurs RGB (niveau de rouge, vert et bleu) du pixel de chaque image.

Prétraitements nécessaires pour extraire les informations :

L’image est convertie en tableau d’une dimension contenant à la suite toutes les valeurs de chaque pixel RGB de l’image. Les images sont au préalable redimensionnées en 128*128 pixels, puis des rotations de 90,180 et 270° ont été effectuées sur celles-ci pour obtenir 3 images supplémentaire. Chaque vecteur est donc composé d’au total 49152 valeurs.

Pour le redimensionnement de l’image j’utilise la librairie python resizeimage, et pour la conversion de l’image en tableau j’utilise Numpy, qui transforme l’image en Numpy Array.

Le script ResizeAndRotate.py effectue le redimensionnement ainsi que la rotation des images.

Nombre de classes possible :

Ici, il y aura deux classes : 1 (Oui) ou 0 (Non), selon si l’image en question représente un Chocobo ou non.

Partie 2 : Approches utilisées et difficultés rencontrées

Méthode choisie pour la reconnaissance :

J’ai utilisé un réseau neuronal convolutif, dans le but de faire reconnaître à mon application un Chocobo. Mon objectif est d’extraire chaque pixel de chaque image, et d’utiliser ces pixels comme entrée de mon réseau de neurones. J’ai approcher le problème de deux façon, avec deux prétraitement légèrement différent, et deux réseaux neuronaux aussi un peu différent.

Pour ma première approche, j’ai converti mes images un vecteur de 49152 éléments, ces éléments étant la somme des niveaux de rouge, bleu et vert des pixels de mon image (128*128*3), et j’ai choisis d’utiliser des couches de neurones classique pour le traiter.

Dans ma seconde approche, j’ai converti mes images en matrices de pixels, matrice de 128*128*3 (le 3 étant pour le RGB), et j’ai choisis d’utiliser un algorithme de convolution avec mes couches de neurones pour faire apprendre efficacement mon réseau avec ces données.

Dans les deux cas, sur mes 600 images de données, j’en utilise 80% (480 images) pour le training de mon application, 10% (60 images) pour la validation, et aussi 10 % pour le test. Ces images sont prise de manière aléatoire. Cela me semble bien, car avec le peu de données que j’ai je voudrais qu’il en ai un maximum pour s’entraîner, la validation et le test sont suffisant pour qu’il évite le sur-apprentissage et puisse fournir un test correct.

Problèmes rencontrés et solutions :

Le premier problème qui s’est vite imposé est la taille du corpus qui était à la base de 150 images. C’est pour cette raison que j’ai décider d’effectuer des rotations sur mes images : cela me permet d’augmenter la taille de mon corpus sans dupliquer les données, étant donner qu’il effectue son traitement dans le sens de l’image. Cette solution me permet d’avoir un corpus de 600 images de qualité correcte. Cependant la solution n’est pas parfaite car encore plus de données aurait aidé, et cela se voit sur les résultat, que nous verrons dans la prochaine et dernière partie, qui sont correct mais améliorable.

Le second problème réside aussi dans le jeu de données que j’ai du créer moi même, car il n’existait pas déjà de jeu de données portant sur ce sujet (ce qui n’est pas non plus étonnant vu son originalité..) et il m’a donc fallu constitué un corpus avec des images trouvées via le moteur de recherche d’image de Google. Cela m’a permis de trouver un certain nombre d’image, mais hélas toutes n’étaient pas approprié à cet usage précis. Certaines images représentaient un ensemble, d’autres avait un fond trop coloré, autant de bruit qui ont beaucoup gêner l’apprentissage de mon réseau de neurone.
Ma solution a été la retouche manuelle de ces images, ce qui a été chronophage certes, mais plutôt efficace. La retouche n’était bien sur pas parfaite, ce qui a laissé quelques défauts mineurs sur certaines images, mais elle a bien amélioré la performance de mon réseau.

Partie 3 : Résultats et conclusion

Ma première approche qui utilise comme données la conversion de mes image en un seul vecteur m'a donné 54 % de précision sur la meilleure itération, ce qui n'est que 4 % meilleur qu'une attribution de score au pile-ou-face.

Cette approche traite les images comme un ensemble, et trouve difficilement des paternes au sein des images. J'ai décidé de commencer sur cette méthode aussi car c'était la première à m'être venue en tête et qu'elle faisait un bon point de comparaison avec la seconde approche. Malgré cela, quelque soit l'approche, les premiers résultat semblaient parfois aléatoire, le temps de construire un réseau performant, et le passage de 150 à 600 images de données ayant énormément aider à l'amélioration du score.

La seconde approche offre une précision de 64 % lors de sa meilleure itération, ce qui est déjà mieux. Comme dis précédemment, le manque de données et la qualité de ces dernières ont été un frein quand à l'obtention de meilleur résultat.

Il était peut-être ambitieux de vouloir faire de la rechercher de forme sur un jeu de données qui ne se base pas sur de l'existant, avec des données du trouvée sur le web qui n'étaient pas toutes adaptés à cet exercice, mais l'exercice était enrichissant, et le travail fournit a malgré tout donné un résultat satisfaisant.