Antonin Durand
INFI3

Compte rendu mini projet Perceptron avec images

Le lien GITHUB du projet avec les data set d'entraînement et de test, le fichier python contenant le Perceptron :
https://github.com/DurandAntonin/ProgrammationAvanceeFI3/tree/master/src/TP/predictionsChatsChiens

L'objectif de ce projet est de de créer un réseau de neurones simple, d'entraîner le modèle pour qu'il puisse prédire la classe de l'image qu'on lui donne.
Les images que nous allons utilisées sont des images de chats ou de chiens. Le but est de prédire si une image contient un chat (dans ce cas le label associé à cette dernière est de 1), sinon elle contient un chien, et le label associé à cette dernière vaudra 0.
Pour ce faire, j'ai utilisé le site Kaggle, qui fournit de nombreux jeux de données, notamment des jeux de données pour la classification d'images où chacune contient un chat ou un chien.

Comme tout problème de machine learning, la première phase est le choix des données. Comme énoncé précédemment, on veut que notre modèle prédise si une image contient un chat ou un chien, ainsi j'ai téléchargé un premier dataset qui va servir à entraîner le modèle. Ce dataset contient 2023 images au total de taille différentes.
J'ai aussi téléchargé un deuxième dataset d'images de chiens et de chats qui va nous servir à tester notre modèle une fois entraîné.

La deuxième étape est l'adaptation des données, pour qu'elles soient utilisables et comprises par notre modèle. En effet, nous avons des données sous format .jpg, cependant notre modèle n'accepte que les matrices du fait des différents calculs matriciels effectués, notamment pour trouver les paramètre w et b.
Pour ce faire, j'ai utilisé la librairie python opencv pour ouvrir chaque image en mode « graysclate » qui permet de transformer une image en une matrice de taille n*n*1 . Ce mode permet d'obtenir une seule couleur pour chaque pixel de l'image au lieu des 3 couleurs RGB.
Ensuite, après avoir ouvert l'image, on la redimensionne pour que chaque image ait la même taille, ainsi que pour éviter d'avoir de trop grandes matrices.
Puis, on enregistre l'image transformée en matrice dans une liste 'x', et le label associé à cette dernière dans une liste 'y'.
Enfin, après avoir traité 1000 images par exemple, de taille 64 par 64 pixels, la matrice x aura la forme (1000,64,64,1) et la matrice y aura la forme (1000,).
Il ne nous reste plus qu'à enregistrer ces 2 matrices dans un fichier spécial de type hdf5. Ce format permet de sauvegarder et de structurer d'importantes données, et permet dans notre cas de pouvoir stocker sous forme de matrices de nombreuses images ainsi que leur label, et de pouvoir les récupérer sans avoir à les retransformer en matrices à chaque fois si on veut entraîner le modèle plusieurs fois.
Nous allons convertir chaque image en matrice qui sera stockée dans une matrice 'x', puis la prédiction associée à chacune de ces dernières sera stockée dans une matrice 'y'.
Ci-dessous représente 10 échantillons de notre dataset d'entraînement ainsi que leur label.

Illustration 1: Echantillons du dataset d'entraînement

La troisième étape est le choix du modèle, dans notre cas nous allons utiliser un modèle de neurones artificiels simple avec uniquement 1 seul neurone. La fonction d'activation de ce dernier sera ainsi la prédiction de l'image sous forme de matrice.

La quatrième étape est le choix de la fonction d'activation et de la fonction coût. Pour la fonction d'activation, nous allons utiliser la fonction sigmoïde (logistique), qui nous permet de calculer une probabilité (vraissemblance) d'une image. Par exemple, si cette dernière renvoit 0.80 pour un échantillon où il y un chat dedans, cela siginife que la probabilité que cette image contienne un chat est de 80% selon le modèle. Ensuite, il suffit de mettre une borne pour indiquer que l'image contient un chat si la probabilité est supérieure à 0.5 par exemple.
On utilise ici plutôt la fonction sigmoïde, plutôt qu'une fonction d'activation binaire, car elle permet d'associer à chaque prédiction une probabilité, et permet d'utiliser comme fonction de coût à optimiser, la fonction Logloss.
Cette fonction de coût calcule la somme des vraissemblances de chaque échantillon. Cette dernière peut être comparée à une expérience de Bernoulli, car il y a deux valeurs y à prédire pour chaque échantillon, et la probabilité pour qu'un échantillon soit de la classe 0 est la probabilité inverse de la probabilité que ce dernier soit de classe 1.

La cinquième étape est l'entraînement de notre modèle. Comme dans l'exercice précédent, on utilise la descente de gradient pour optimiser la fonction coût et trouver les paramètre w et b qui minimisent cette dernière à chaque itération.

Enfin, la cinquième étape est le test de notre modèle, où on vérifie si ce dernier permet de prédire efficacement le label associés à de nouvelles images, i.e si les paramètre w et b trouvés permettent de prédire si l'image est une image de chat ou bien une image de chien. Pour ce faire, nous allons importé les images et les prédictions de notre dataset de test déjà converties en matrice et qui sont stockées dans le fichier hdf5.
On passe cette nouvelle matrice d'images x au réseau de neurones qui nous renvoie les labels y_pred qu'il a prédit. Enfin, on compare la matrice y_pred à la matrice y pour voir le pourcentage de précision de notre modèle.

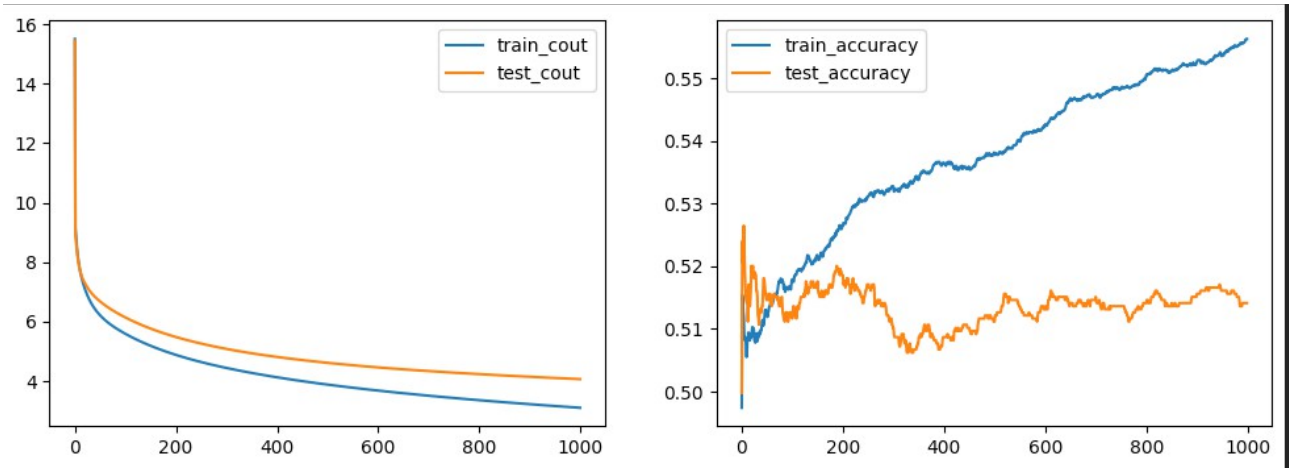

Illustration 2: Graphiques évolution coût et de l'accuracy durant l'entraînement

Comme le montre l'image ci-dessus, on constate que le modèle n'est pas forcément très fiable, comme le montre le pourcentage d'entraînement à la fois pour le data set d'entraînement et pour le data set de test, qui vaut environ 50%.
Une des possibilités pour améliorer l'efficacité de notre modèle serait d'ajouter un ou plusieurs neurones avec une ou plusieurs couches, pour créer un réseau plus complexe qui serait capable d'identifier les différents critères d'un chat et d'un chien par exemple.

On affiche ensuite la matrice de confusion de notre modèle

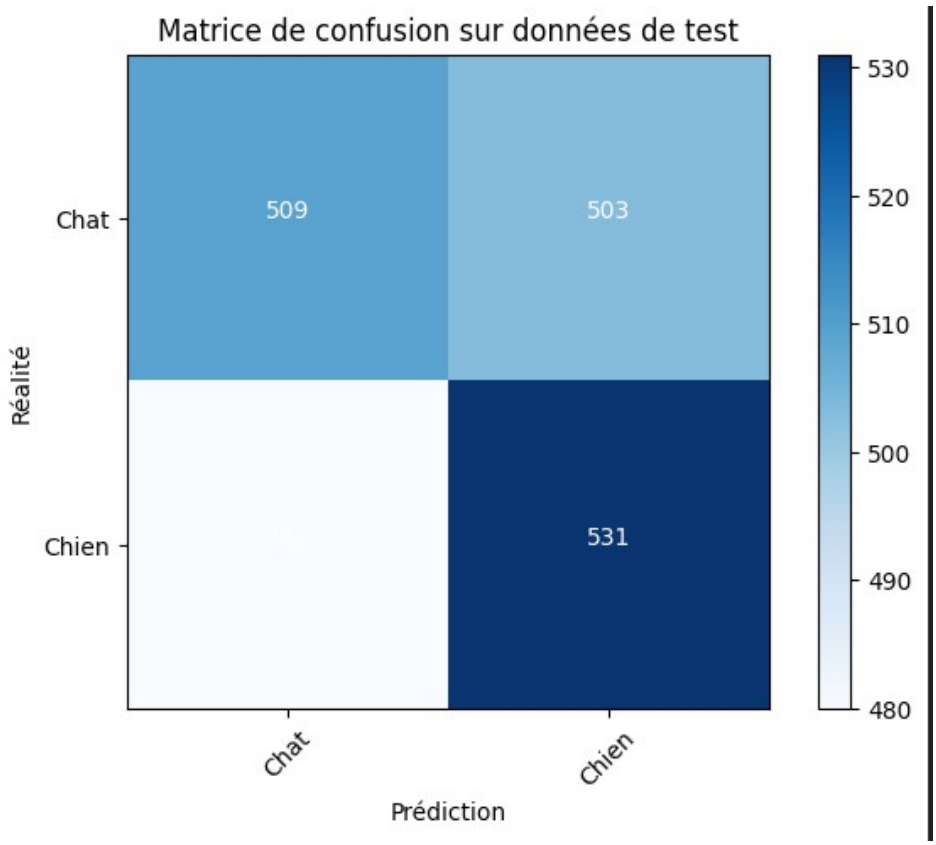

Illustration 3: Matrice de confusion

Comme pistes d'améliorations, on pourrait vérifier si les différents images de notre dataset d'entraînement sont correctes et non pas hors contextes. On pourrait aussi créer un réseau de neurones plus complexe, en ajoutant 1 neurone et/ou une couche en plus par exemple.