2

Rapport sur la Reconnaissance d'images

« Partie 3 »

Réalisé par RHERMINI Idriss

BACHELOR 3 IABD

Année scolaire 2019/2020

Table des matières

Objectifs

- Implémenter le RBF, l'algorithme K-means, la sauvegarde et chargement des modèles.
- Créer un réseau de neurone fiable et de qualité pour classifier les images.

RBF

Le RBF est une méthode avec une autre vision que le MLP. Le MLP pourrait se résumer à combiner des droites affines afin de créer une élimination. C'est une méthode efficace, cependant la précision dans certains cas peut être pauvre, ou alors on a une mauvaise généralisation. Pour avoir une meilleure précision, on rajoute des couches et des neurones par couche pour multiplier nos droites affines à combiner afin d'avoir une plus grande précision.

Certes, je suis capable de résoudre des cas particuliers avec le MLP, mais l'inconvénient de cette méthode est le temps d'entrainement. En effet, plus je rajoute de neurones plus j'ai de poids à calculer.

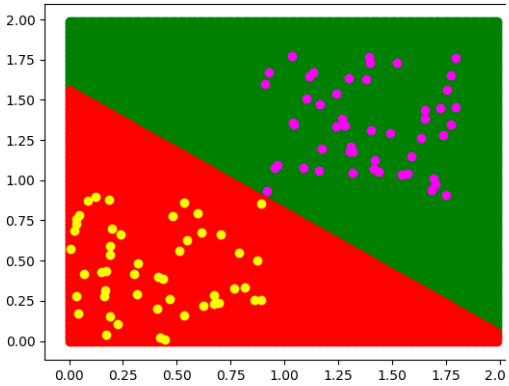
Pour pallier ce problème, la méthode du RBF permet d'avoir une complexité de calcul lié aux nombres de données. Nous verrons ensuite comment réduire encore cette complexité.

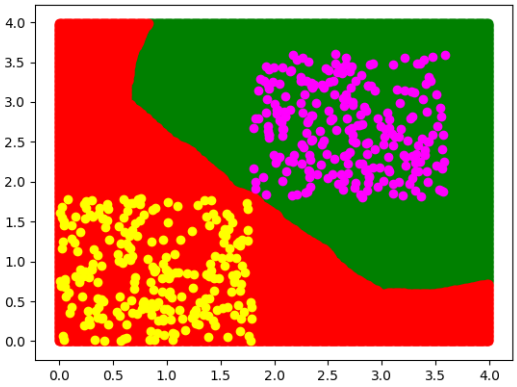
Test RBF

Le principe est simple, pour chaque point, je crée une zone d'affectation. Pour une prédiction, je regarde sur quelles zones elle est la plus proche.

Cette zone d'affectation est contrôlée par une Gaussienne, je peux gérer l'empattement de cette zone avec un paramètre.

Cette méthode permet de lisser les zones d'affectations sans augmenter la



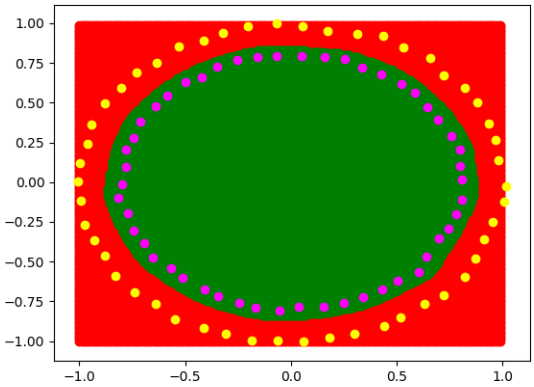


complexité des calculs.

FIGURE 3 – Cross

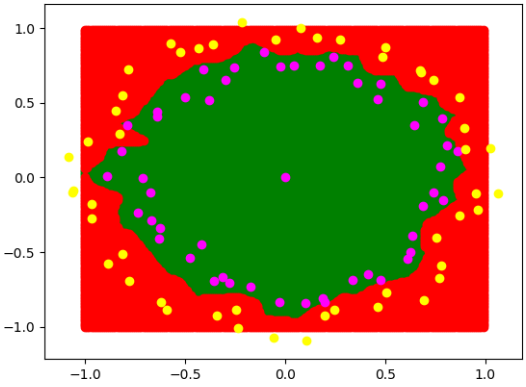
FIGURE 4 – CROSS

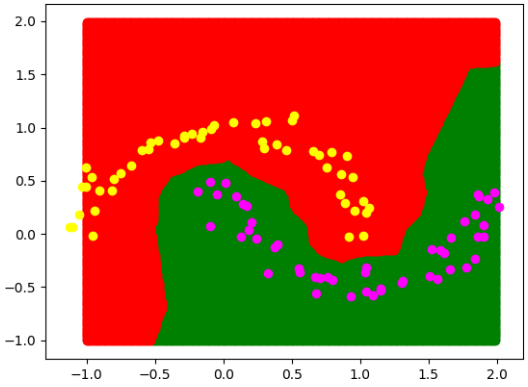
On observe bien que dans le cas d'une séparation linéaire simple, on retrouve cette tendance de séparation. Cependant le RBF n'a pas pour objectif de résoudre ce genre de problèmes. Il faut lui donner des cas où même le modèle non linéaire à besoin de beaucoup de couches/neurones pour bien généraliser.



Ce premier exemple est le cas simple du cercle. Pour que ce cas soit bien réalisé avec le perceptron multicouche, je suis obligé d'exploser le nombre de neurones par couche et d'augmenter le nombre de couches. Avec le RBF, ce

cas devient facile à réaliser.

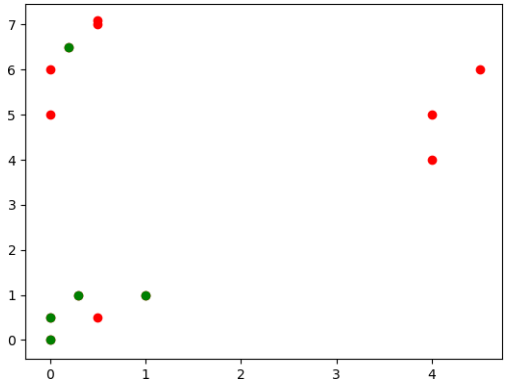




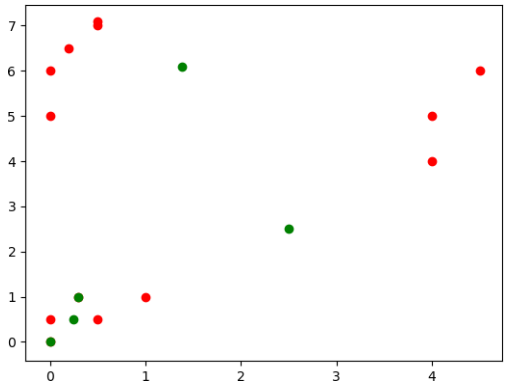
KMEANS

Kmeans est une méthode de partitionnement des données. On cherche à assigner k points pour k groupes de points. On parle généralement de cluster. L'objectif final est de chercher les centroïdes des clusteurs est de diminuer le nombre d'entrées de nos algorithmes. Diminuer le nombre d'entrées permet de réduire le temps de calcul des poids pour un MLP ou le temps de calcul de la matrice à inverser pour le RBF.

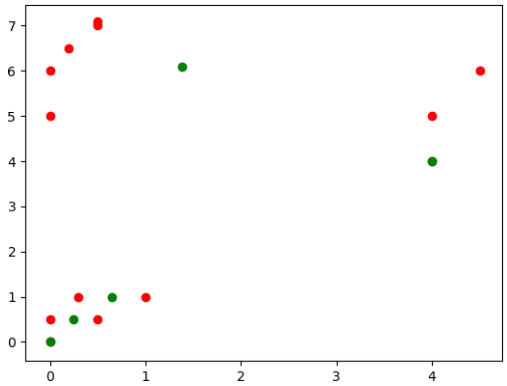
Dans les deux cas, l'objectif est d'essayer de mieux généraliser tout en ayant moins d'input.



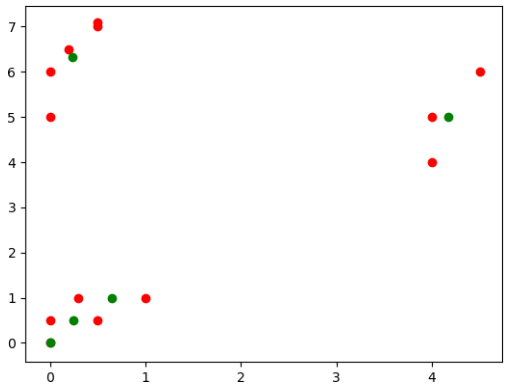
Dans cet exemple, j'ai créé un dataset avec 3 regroupements de points. Dans ce cas, j'ai donné à mon algorithme 5 représentants (k= 5).



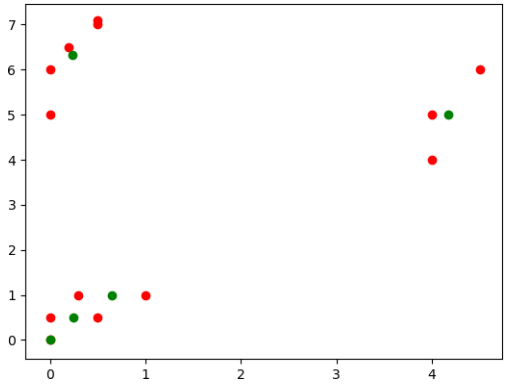
Les premiers centroïdes sont choisis de façon aléatoire, pour avoir une meilleure convergence et éviter un maximum des cas particuliers ou nos centroïdes risques de ne pas converger.



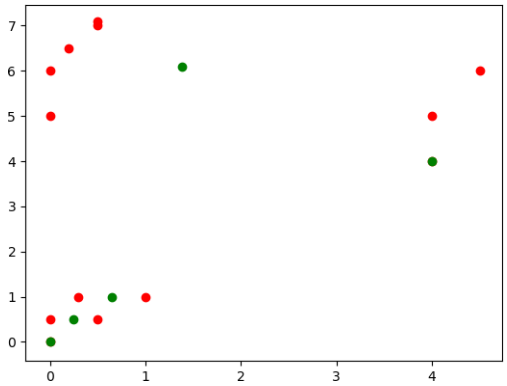
A la seconde itération, j'affecte pour chaque centroïdes des points, puis nous replaçons les centroïdes proches de leurs points affectés.



Je réitère cette opération jusqu'à ce que les centroïdes ne bougent plus, ou que nous allons dépasser un nombre d'itération défini.



Dans le cas ci-dessous, j'ai le même dataset en ayant défini k à 3. Ce cas me montre que les centroïdes ne convergent pas, car ils sont trop proche les uns des autres lors de l'initialisation.



Pour résoudre ce problème, il suffirait de relancer l'algorithme. Si je tombe dans un cas particulier de non convergence, alors j'augmente le nombre de centroïdes.

Je n'ai pas implémenté le RBF couplé au Kmeans, cependant je peux quand même appliquer le RBF naïf sur les valeurs de retour de Kmeans.

Dans mes recherches, j'ai trouvé qu'il est possible de simplifier une image. En effet, si je donne a Kmeans une image avec un certain K, alors je me retrouve avec une image possédant K couleurs.

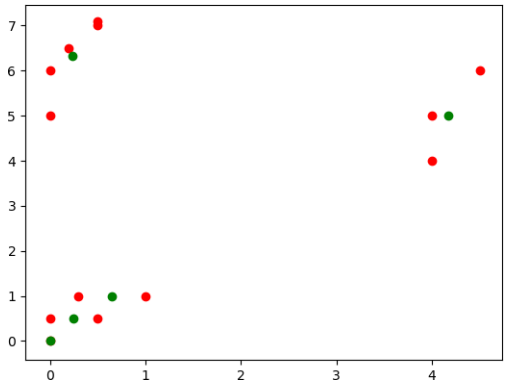
Sauvegarde des modèles

Tous les calculs sont en mémoires, il n'y a donc aucun moyen de garder un modèle dans le temps, a moins de garder vivant l'interpréteur python.

J'ai deux sauvegardes de modèle, le modèle linéaire et le modèle du réseau de neurone.

Je sauvegarde les métadonnées du modèle sur la première ligne du fichier. On garde dans le fichier le nombre d'entrée du modèle linéaire et non linéaire. Pour le modèle non linéaire on rajoute aussi le nombre de neurone par couche.

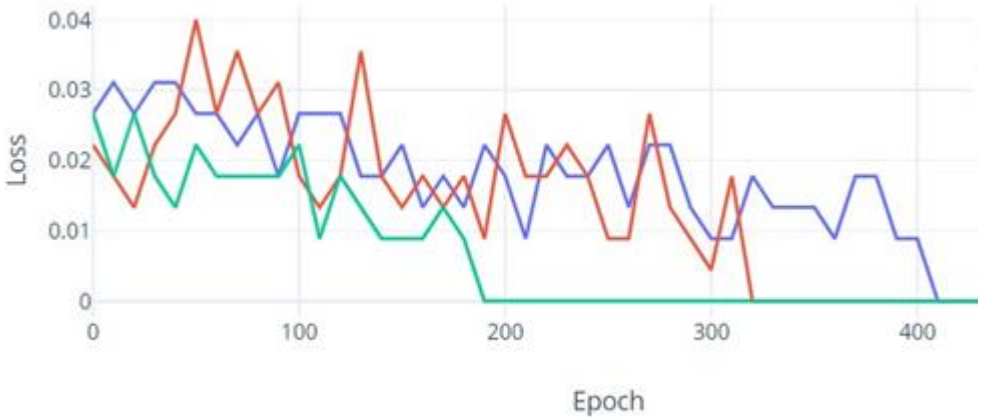
La seconde ligne liste les poids de tous les neurones séparés par une virgule. Il n'y a pas de retour à la ligne pour chaque couche de neurone. Ce qui n'est visuellement pas pratique, mais techniquement identique que de tout mettre sur une ligne.



Premier test sur le Dataset

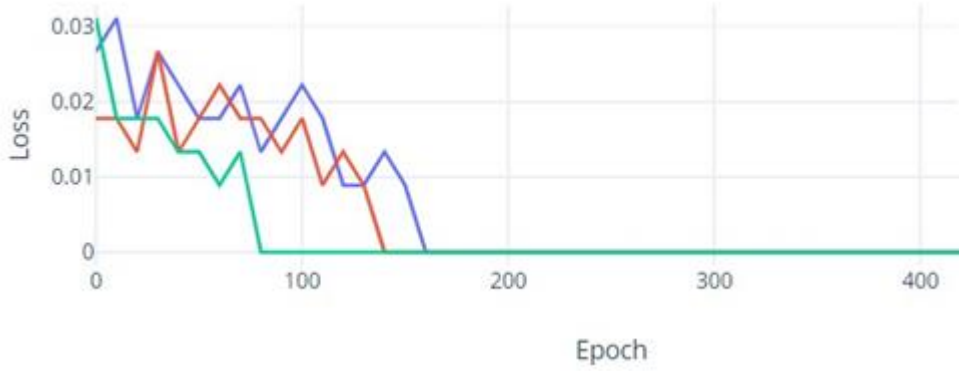
Maintenant que j'ai validé une majorité des cas de test avec la classification linéaire, non linéaire (MLP) et RBF linéaire. J'ai essayé de les appliquer à ma dataset.

Pour le premier cas, j'ai pris un modèle linéaire. Je lui ai donné 900 images d'entrainement de taille 25x25 pixels. Je dois entrainer 3 modèles linéaires pour chaque classe, puis lors de la prédiction, je sélectionne le modèle qui me donne la valeur la plus haute.



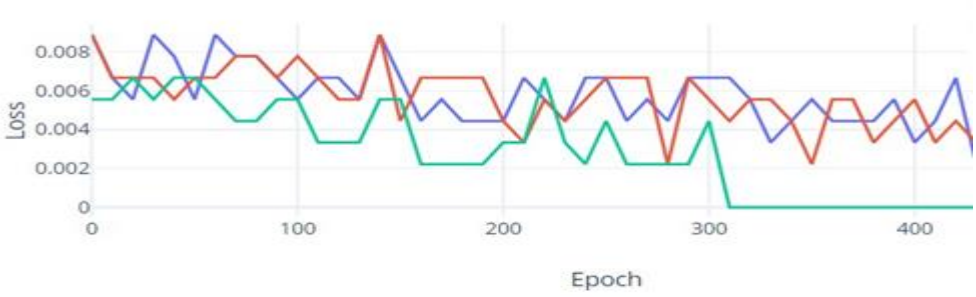
Grâce à ce simple modèle j'arrive à obtenir un taux de précision de 56.67% sur mes images de Validation ! Je suis assez satisfaits du résultat vu la complexité du modèle.

Je vais désormais essayer de modifier les hyperparamètres pour observer les évolutions du taux de précision et déterminer les "meilleurs" hyperparamètres. Ici je change uniquement la résolution de l'image d'entrée par 100x100 (en pixel):

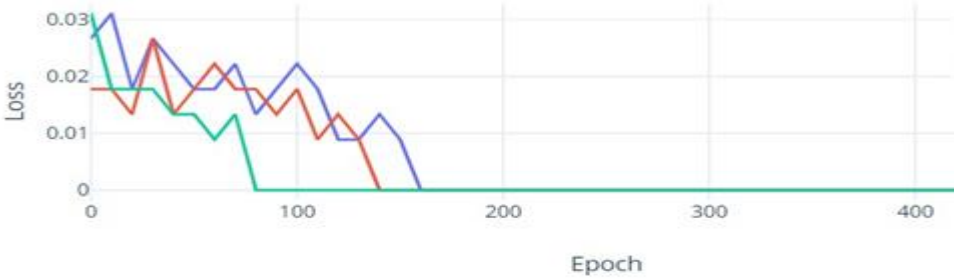


Je remarque une nette amélioration ! J'ai 83.33% de taux de réussite sur mon dataset de Validation. De plus on remarque que l'apprentissage converge plus rapidement en époques, par contre le temps d'apprentissage est bien plus élevé.

Avec un petit dataset, je suis capable d'avoir des résultats satisfaisant. Maintenant je veux généraliser plus, il faut donc augmenter le nombre d'images d'entrainement. On reprend les images de 25x25, mais cette fois ci avec 2000 images d'entrainement.



On se retrouve avec un résultat catastrophique. 26.67% de précision. On remarque qu'au bout de 500 époques, le modèle pour classer les images CATS et DOGS sont encore haute. Il faudrait donc augmenter le temps d'apprentissage en élevant le nombre d'époques.



En augmentant la taille des images, j'augmente le taux de prédiction, je suis à 80% de taux de réussite sur 2000 images d'entrainements de 100x100 et 90 images de validation.

Le modèle linéaire semble satisfaisant dans un premier lieu, cependant je suis obligé d'augmenter le nombre de pixel lorsqu'on augmente mon nombre d'image d'entrainement.

Conclusion

Nous avons constaté lors de ce rapport que les difficultés rencontrées concernent principalement le nombre de couche et de neurone. Pour limiter le nombre de ces deux paramètres, le Cross et multi Cross étaient efficaces et m'ont permis également de créer des modèles sur mon propre dataset.

Les premiers cas de tests sur mon dataset sont concluants, je ne pensais pas que le perceptron de Rosenblatt a lui seul pouvait réaliser des scores aussi haut. Cependant, il ne faut pas oublier que je ne lui donne que des petits dataset à l'heure actuelle. Il va falloir augmenter le nombre de nos images de cas de test, mais aussi augmenter la résolution si l'on veut espérer une meilleure généralisation. Il est possible qu'avec un dataset d'une grande taille et d'images de haute résolution, le perceptron de Rosenblatt ne soit plus capable de fournir de bon résultat. Il faudra donc utiliser le MLP et le RBF.