Rapport de Projet: Etape intermédiaire 3

Projet Annuel 3 Big Data

ALLEXANDRE Matthieu
HOLLANDER Stephane
TAVERNIER Antoine

01/07/2019

Table des matières

Objectifs

L'objectif de se rendu est d'implémenter le RBF, l'algorithme K-means, la sauvegarde et chargement des modeles ainsi que le SVN.

Lors de notre rendu précédent, nous n'avions pas un réseau de neurone satisfaisant. Nous pensions que nous étions dans des mimimuns locaux, or l'implémentation était simplement érroné. Nous avons donc consacré beaucoup de temps à cette tâche. Tellement de temps que nous avons décidé de sacrifier l'implémentation du SVN pour ce rendu afin de fournir un réseau de neurone de qualité et fiable.

Nous pensons que le réseau de neurone aura de meilleur résultat que le SVN pour la classification d'image.

Réseau de Neurones

Dans notre compte rendu 2, nous avions écrit que nous pensions avoir des minimums locaux. Nous n'arrivions pas à avoir des résultats persistant pour le cas de test Cross et Multicross. Tensorflow est capable réaliser cette action sans difficulté, nous devions pour repencher sur l'implémentation de preceptron multi couches.

En effet le réseau de neurones est sûrement l'outil qui permet de généraliser le mieux des images avec un nombre de classe petit. Nous devions revoir notre implémentation afin de fournir des résultats corrects lors de la soutenance finale.

Cette étape nous a pris un temps conséquent, en effet nous n'arrivions pas à trouver nos erreurs. Nous avons donc reconstruit l'algorithme depuis de début, plusieurs fois...

Il nous fallait du recul pour voir nos erreurs, l'erreur principales était la gestion des indexes. Non pas l'utilisation de mauvaises bornes, mais de boucles dans le mauvais sens. Nous avions inversé les identifiants des boucles J et I, ce qui revient aussi à intervertir les indexes dans le triple pointeur W. Nous sommes passés de $W[L][I][J]$ à $W[L][J][I]$.

Maintenant nous avons des résultats corrects et moins aléatoire. Nous arrivons à reproduire régulièrement les mêmes résultats avec des configurations similaires.

Le cas du Cross est beaucoup plus stable. Toute fois nous avons toujours la probabilité de ne pas avoir un bon résultat. Cette fois ci, on peut s'assurer que c'est effictivement lié à un minimum local et non une erreur d'implémentation.

Pour le cas du MultiCross, au dernier rendu nous avions des bandes de

couleurs et certains carrés de couleurs qui semblait se rapprocher du résultat final. A l'époque nous pensions que c'était lié à un sur-apprentissage ou nous pensions trouver un minimal local plutot que global. En effet si le dataset n'est pas linéairement séparable alors le Réseau de neurones n'est plus garranti de converger vers la solution globale (Dépendant du nombre de couches/neurones, fonctions d'activation, taille du dataset et l'algorithme de correction des poids). [1]

Aujourd'hui nous sommes en mesure de valider le cas de test du MultiCross avec un réseau de neurone à 3 couches $[2, 16, 16, 3]$, le 2 correspond aux inputs.

Avec les mêmes configurations, voici la différence entre notre ancien réseau de neurone et nouveau :

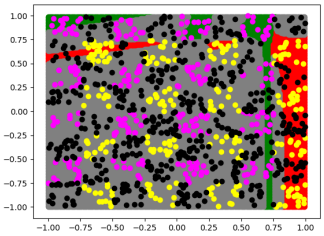


FIGURE 1 – Ancien algorithme : $[2, 24, 24, 24, 24, 3]$

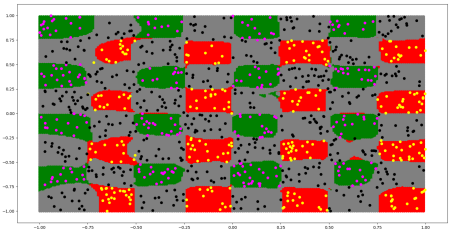


FIGURE 2 – Nouvel algorithme : $[2, 16, 16, 3]$

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 délimination. C'est une méthode efficace, cependant la précision dans certains cas peu etre 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 nous sommes capables de résoudre des cas particuliers avec le MLP, mais elle a un prix, le temps d'entrainement. En effet plus nous rajoutons de neurones plus nous avons 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.

1. https ://stats.stackexchange.com/questions/65877/convergence-of-neural-network-weights

Le principe est simple, pour chaque point, nous créons une zone d'affectation. Pour une prédiction nous regardons sur quelles zones elle est la plus proche.

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

Cette méthode permet de lisser les zones d'affectations sans augmenter la complexité des calculs.

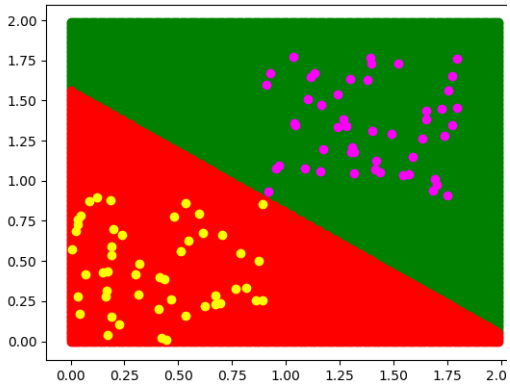


FIGURE 3 – Cross

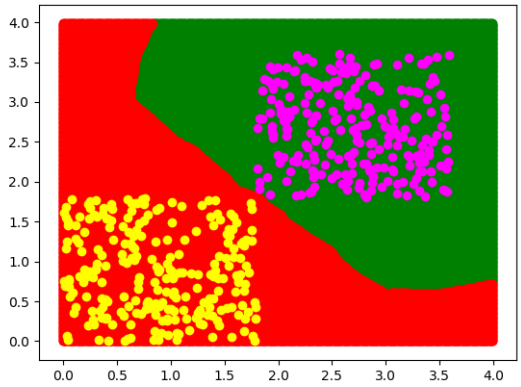
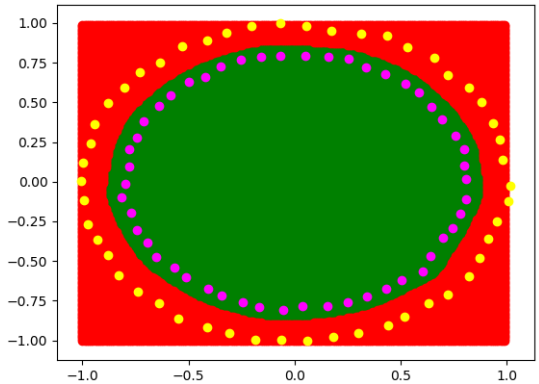
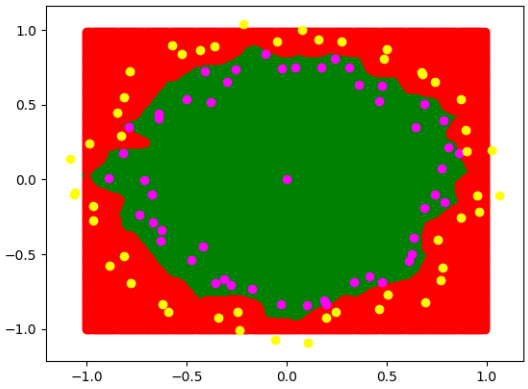


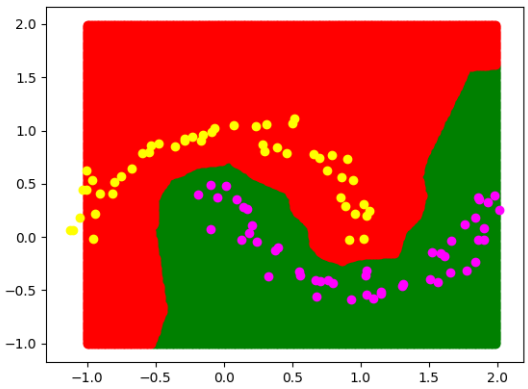
FIGURE 4 – cross

On observe bien que dans le cas d'un 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, nous sommes 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



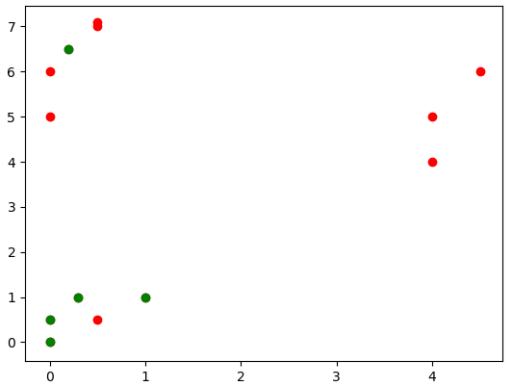


PEUT ETRE AUSSI DES TEMPS DE COMPARAISON
CAS DU CERCLE

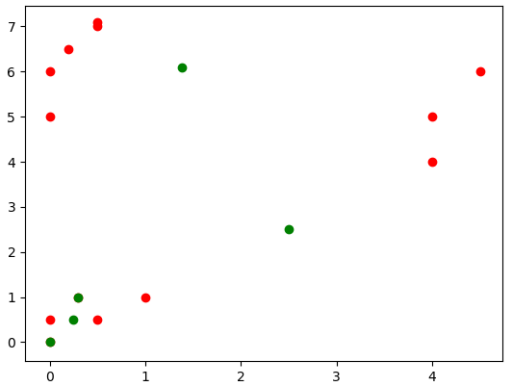
KMEANS

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.

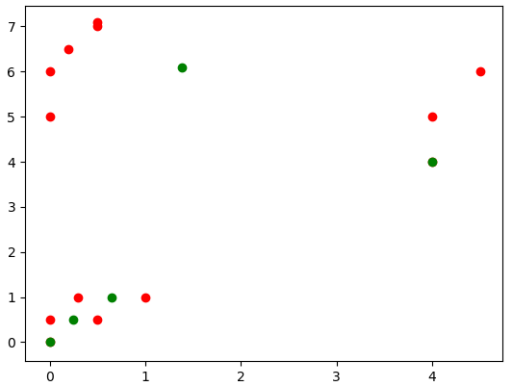
Dans les deux cas, l'objectif est d'essayer de mieux généraliser tout en ayant moins d'input.



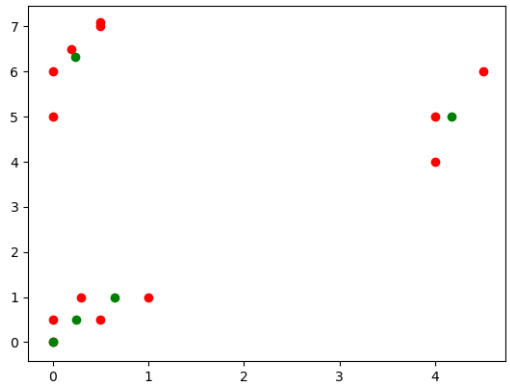
Dans cet exemple, nous avons créé un dataset avec 3 regroupement de points. Dans ce cas, nous avons donné à notre algorithme 5 représentants (k = 5).



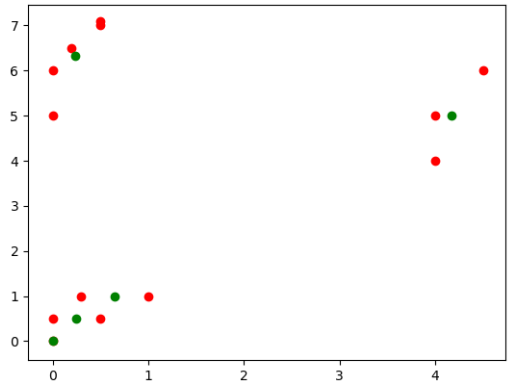
Les premiers centroïdes sont choisis de façon aléatoire, pour avoir une meilleure convergence et éviter un maximum des cas particuliers ou nos centroïdes risques de ne pas converger.



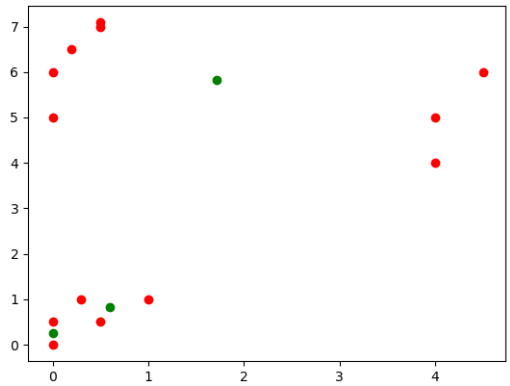
A la seconde itération, nous affectons pour chaque centroïdes des points, puis nous replaçons les centroïdes proches de leur points affectés.



Nous réitérons 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, nous avons le même dataset en ayant défini k à 3. Ce cas nous montre que les centroïdes ne convergent pas, car ils sont trop proche les uns des autres lors de l'initialisation.



Pour pallier ce problème, il suffirait de relancer l'algorithme, or nous avons un seed fixe qui nous donne des résultats identiques à chaque lancement. Si nous tombons dans un cas particulier de non convergence, alors nous augmentons le nombre de centroïdes.

Nous n'avons pas implémenté le RBF couplé au Kmeans, cependant nous pouvons quand même appliquer le RBF naïf sur les valeurs de retour de Kmeans.

Dans nos recherches nous avons trouvé qu'il est possible de simplifier une image. En effet si nous donnons a Kmeans une image avec un certain K, alors nous nous retrouvons avec une image possédant K couleurs. [2]

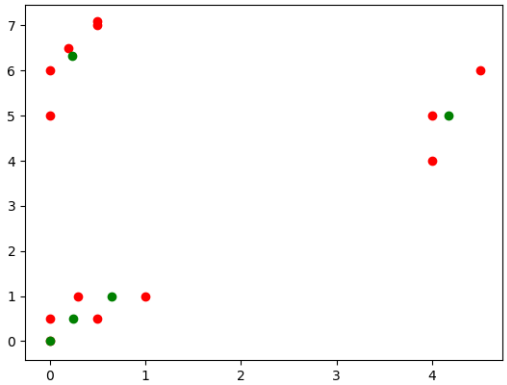
Sauvegarde des modèles

Nous n'avons pas envie d'entrainer un modèle pour faire une prédiction. 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.

Nous avons donc 2 sauvegardes de modèle, le modèle linéaire et le modèle du réseau de neurone.

Nous sauvegardons 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.



2. https ://youtu.be/yR7k19YBqiw ?t=343

Ici nous avons l'exemple de la sauvegarde d'un perceptron à 2 entrées :

```
2,1,
-3.78182,2.48971,2.48973,
```

Ici nous avons le début du fichier de sauvegarde d'un perceptron multi-couches $[3000, 32, 32, 64, 64, 3]$:

```
30000,32,32,64,64,3,
0.114529,-0.503429,-0.539597,-0.276175,0.411277,0.0872584,-0.920254,-0.0605315,-0.734143,0.433738,0.291768,-0.0858463,0.616999,
-0.988001,-0.620286,0.666211,0.0922661,-0.137736,-0.647119,0.848596,0.330168,0.611652,0.155602,0.394624,-0.727769,-0.91409,
-0.235799,-0.698294,-0.57133,-0.661661,-0.849789,-0.710791,0.795767,-0.640299,-0.21579,0.981737,0.283813,0.833293,0.227761,
-0.492751,0.342586,-0.908344,-0.22719,0.718275,-0.102957,0.964371,-0.827588,-0.0132885,-0.469211,0.937219,0.323058,0.341406,
-0.713079,0.284631,0.890536,0.333786,0.513444,-0.0886553,0.258112,-0.374335,-0.0789479,-0.138623,-0.953616,0.714621,0.526296,
0.470731,0.878632,0.12034,0.109774,-0.748599,0.756178,-0.480065,-0.836754,-0.607397,0.863702,-0.584587,-0.536993,-0.222687,
-0.47371,0.630034,-0.0886654,-0.598831,0.119431,-0.641176,0.876787,0.848364,-0.560826,0.468228,0.245608,0.375403,-0.464836,
-0.63859,-0.772416,-0.924659,-0.106923,0.0922802,-0.272822,-0.758801,0.556229,0.842529,0.917015,0.338007,-0.142827,0.71506,
-0.57789,-0.216622,-0.846518,0.944353,-0.549565,-0.587196,-0.753973,-0.266146,0.918721,-0.687182,-0.591107,0.16716,-0.714172,
```

Ce fichier fait 8,76 Mo. Ce fichier est volumineux car il contient 103264 poids pour 166 neurones cachés et 3000 entrées.

Chargement des modèles

Le chargement des modèles sont des fonctions miroirs aux sauvegardes des modèles. Vu précédemment, la première ligne est consacrée aux métadonnées.

On récupère donc la structure du modèle. Ensuite nous créons un nouveau modèle vide avec la structure décrite dans le fichier. Et enfin nous injectons aux bons emplacements les poids des neurones dans la structure.

Site Web

Une nouvelle version de site Web a été développé, l'objectif est de donner à l'utilisateur les outils pour créer son model et/ou prédire son image avec son modèle ou un modèle pré-enregistré.

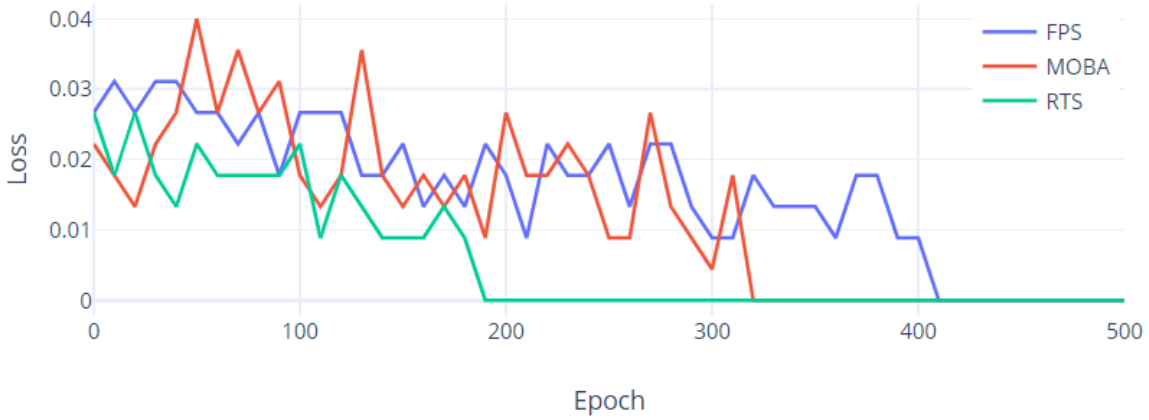
Nous donnons certains paramètres modifiables par l'utilisateur via le site Web, tel que la taille du dataset, la taille des images, le nombre d'époques, le model, les couches. Le site étant en Python (Flask), il est très facile d'appeler notre Framework python.

De plus, nous laissons un tableau de configuration de bases à l'utilisateur pour avoir un aperçu des résultats espérés avec sa propre configuration et pouvoir comparer.

Premier test sur le Dataset

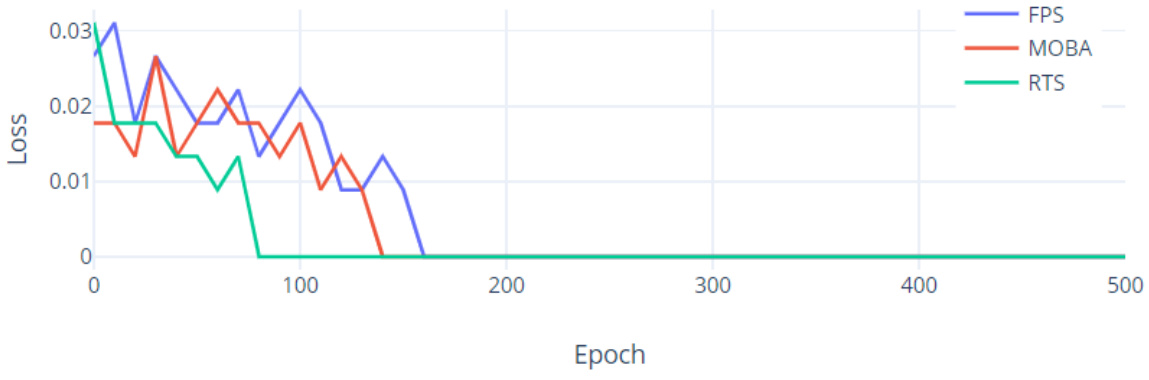
Maintenant que nous avons validés une majorité des cas de test avec la classification linéaire, non linéaire (MLP) et RBF linéaire. Nous avons essayé de les appliquer à notre dataset de screenshot de jeux vidéo.

Pour le premier cas, nous avons pris un modèle linéaire. Nous lui avons données 900 images d'entrainement de taille 25x25 pixels. Nous devons entrainer 3 modèles linéaires pour chaque classe, puis lors de la prédiction, nous sélectionnons le modèle qui nous donne la valeur la plus haute.



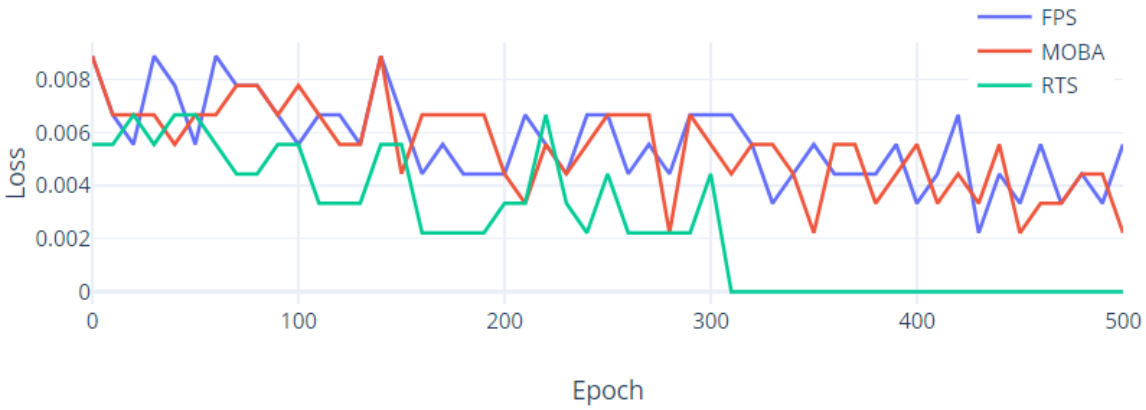
Grâce à ce simple modèle nous arrivons à obtenir un taux de précision de 56.67% sur nos images de Validation ! Nous sommes assez satisfaits du résultat vu la complexité du modèle.

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

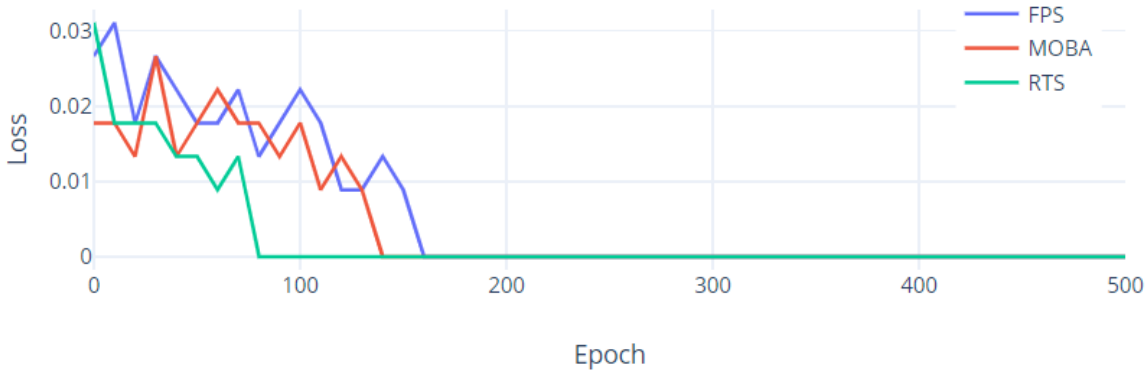


Nous remarquons une nette amélioration ! Nous avons 83.33% de taux de réussite sur notre dataset de Validation. De plus on remarque que l'apprentissage converge plus rapidement en époques, par contre le temps d'apprentissage est bien plus élevé.

Avec un petit dataset, nous sommes capables d'avoir des résultats satisfaisant. Maintenant nous voulons généraliser plus, il faut donc augmenter le nombre d'images d'entrainement. On reprend les images de 25x25, mais cette fois ci avec 3600 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 FPS et MOBA sont encore haute. Il faudrait donc augmenter le temps d'apprentissage en élevant le nombre d'époques.



En augmentant la taille des images, nous augmentons le taux de prédiction, nous sommes à 80% de taux de réussite sur 3600 images d'entrainements de 100x100 et 90 images de validation.

Le modèle linéaire semble satisfaisant dans un premier lieu, cependant nous sommes obligés d'augmenter le nombre de pixel lorsqu'on augmente notre nombre d'image d'entrainement.

Pour le cas du réseau de neurone, nous n'avons pas encore fait de tests industriels. Nous avons lancé un seul entrainement d'un réseau de neurones de taille $[3000, 32, 32, 64, 64, 3]$ sur 3600 images. Avec 3000 pixels d'entrée, 2 couches de 32 neurones, 2 couches de 64 neurones et la dernière couche pour le résultat final de 3 neurones.

3000 pixels représentent une image de 100x100 avec tous les canaux RBG. L'apprentissage du modèle à pris plus de 6 heures. Cependant il n'a pas encore passé les tests de validation. L'erreur des neurones de fin semblait très élevée. Il est donc fort probable que nous n'avons pas de bon résultat avec un MLP aussi gros.

Conclusion

Nous avons perdu beaucoup de temps pour réparer le MLP, ce qui a empiété sur l'implémentation des autres algorithmes. Cependant, nous sommes confiants de nos nouveaux résultats avec le MLP.

En effet nous passons les cas de tests du Cross et multi Cross sans trop augmenter le nombre de couche et de neurone par couche. Ce qui nous laisse maintenant la possibilité de créer des modèles sur notre propre dataset.

Les premiers cas de tests sur notre data set sont concluants, nous ne pensions pas que le perceptron de Rosenblatt a lui seul pouvait réaliser des scores aussi haut. Cependant il ne faut pas oublier que nous 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 hautes résolution, le perceptron de Rosenblatt ne soit plus capable de fournir de bon résultat. Il faudra donc utiliser le MLP et le RBF.

Pour le prochain rendu, notre objectif final est de donner la possibilité à un utilisateur, via le site Web, d'entrainer ou de prédire une image selon un modèle. Ce modèle sera soit fourni, soit produit par lui-même.

Dans l'absolu, il faudrait implémenter le CNN, car nous pensons que ce soit l'algorithme le plus adapté pour la classification d'image en général. Cependant, si nous n'avons pas le temps, nous consacrerons notre énergie sur la bonne utilisation des algorithmes fonctionnels.