ML rapport

L'algorithme de rétropropagation de l'erreur peut être trouvé à la ligne 394 de NeuralNetwork.cpp, les entraînements se font tous dans mainwindow.cpp. Le projet utilise Qt 6.4.2 avec le compilateur MSVC2019, et l'éditeur QtCreator. Nettoyer le projet avant de le compiler la première fois.
Git url: https://github.com/maxo98/MachineLearning_5DJV/tree/rendu-final

Le projet utilise plusieurs set de données de différentes tailles, il faut setter la taille des images à la ligne 11 de mainwindow.cpp (#define IMG_SIZE 380), par défaut la taille est 380.

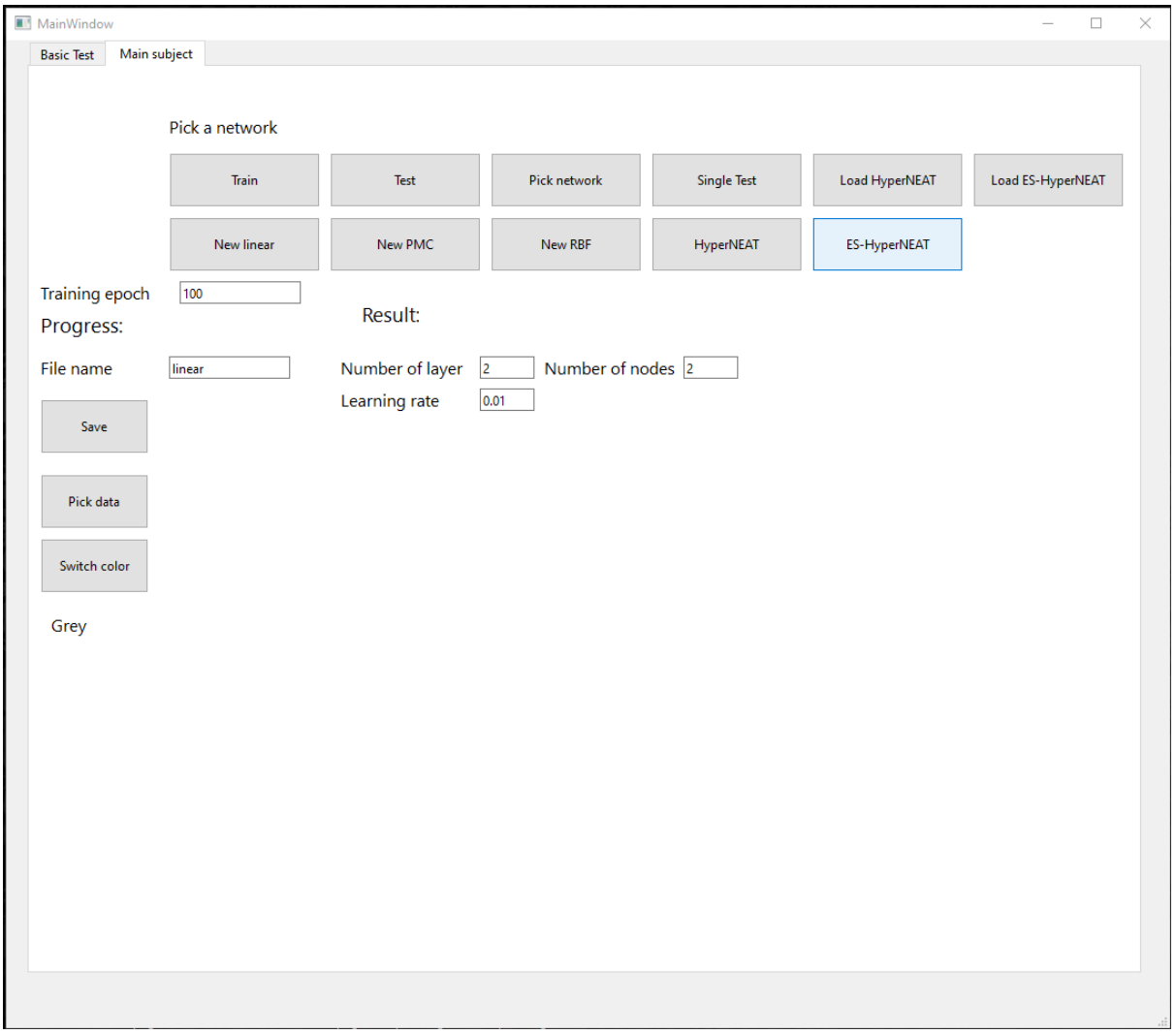
Les dossiers des différents data sets peuvent être trouvés à la racine du projet.

ML champions lol => (images couleur) taille 380
ML champions lol edge => taille 380
ML champions lol grey => taille 380
ML champions lol size95 => taille 95
MLChampionx5 => taille 340

La problématique de notre sujet était de classifier des images de personnages de league of legends, selon 3 clusters "Hommes", "Femmes", "Autres" (non humain).

Voir le tableur excel fourni avec le rapport pour la liste des tests.

(Voir image de l'interface page suivante)
Pour utiliser les réseaux de neurones pour les data sets de couleur cliquez sur "Switch color". Pour tester un réseau de neurones entraîné, cliquez sur "Pick data" puis sélectionnez le dossier de données utilisé pour entraîner le réseau de neurones. Ensuite cliquez sur "Pick network" pour sélectionner le réseau de neurones. Vous pouvez maintenant tester le réseau de neurones en cliquant soit sur "Test" pour tester sur les données de test du data set, ou en cliquant sur "Single Test" pour sélectionner une image à tester. Pour tester un réseau de neurones HyperNEAT ou ES-HyperNEAT cliquer sur "Load HyperNEAT" ou "Load ES-HyperNEAT" respectivement, un test sur les données de test sera lancée automatiquement, la possibilité de faire des units tests avec hyperNEAT n'a pas été tester.



Les données ont été séparées en deux groupes: 80% pour le set d'entraînement et 20% pour le set de test. On a lu qu'il était rarement utile d'avoir plus de deux layers cachés, et que le nombre de neurones par layer devait être entre le nombre de neurones en entrée et le nombre de neurones en sortie. On c'est donc basé sur ces principes pour nos tests. On voulait utiliser l'initialisation des poids Xavier Glorot uniforme, mais il y des erreurs d'inattention, pour le PMC on a souvent utilisé une initialisation de Kaiming He uniforme qui générait aussi des poids négatifs (baptisé negative-he dans le tableau excel). Une autre coquille est que l'on entraînait nos réseaux de neurones en fournissant que 308 premières colonnes en entrée du réseau de neurones jusqu'au test 34. Les meilleurs test fait avancé ont été reproduit après ces corrections, mais il semblerait que ça n'est pas de différence. Notre modèle de PMC à utilisé la tangente hyperbolique comme fonction d'activation pour tous les tests sauf le test 42 où l'on a essayer d'utiliser l'activation relu, mais ont tous de suite vu que les sortie était tout le temps 0 ou 1, et qu'avec la disparition du gradient l'entraînement ne ferait aucune différence. Utilisé softmax aurait été une solution, mais on ne l'a pas implémenté. On voulait aussi utiliser des initialisations

normalisées, mais on avait un problème le random number generator pour les distribution normalisé et on n'a pas poussé plus loin.

Sur les tests 1 à 9, on a essayé de déterminer le nombre de neurones à mettre sur chaque couche cachée. Le meilleur résultat à été donné par le test 9 avec 2 couches et 40 neurones par couches, 10000 epoch et un learning rate de 0.0005. D'autre test ont été lancé avec plus d'epoch et learning rate moins élevé, mais apparemment ça ne faisait qu'overfitté le réseaux de neurones.

Sur les tests 10 à 16 et 23 à 30, on est passé sur un set données avec des images de la même taille que le précédent mais en niveau de gris. On a commencé avec un réseau de neurones relativement gros, et on a ensuite essayé de réduire la taille du réseau de neurones petit à petit selon les résultats. Il semblerait d'après nos que la taille idéale est une couche cachée et 5 neurones sur cette couches avec modèle PMC. Il nous a aussi semblé que le PMC donnait de meilleurs résultats que le RBF sur cette problématique, donc on a décidé de se concentrer sur le PMC après ces tests.

Entre ces deux groupes de tests sur le data set "Grey" ont utilisé un autre dataset, sur ce data set on appliqué un kernel de détection de bord. Les résultats ont été très médiocres, peut être que ça aurait pu fonctionner avec plus de données, et/ou avec un avec modèle convolutionnel. Les entrées avec des valeurs supérieures étant probablement trop éparses, peut être qu'il n'arrive pas calculer quoi-que ce soit.

Après ça on a fait deux test couleurs (31, 32) avec plus d'epoch que le précédent où l'on avait obtenu notre meilleur résultat, mais il semblerait que ça ait juste overfitter.

On a aussi fait des avec HyperNEAT et ES-HyperNEAT (Evolvable Substrate HyperNEAT), comme les images ne présentent pas de régularité "géométrique" on ne s'attendait pas à obtenir de bon résultat. Les premiers essais était fait l'extension Link Expression Output (LEO) d'HyperNEAT, cette extension permet en gros d'obtenir des réseaux de neurones plus modulaires et avec moins de connexions. Tous les essais que nous avons fait avec HyperNEAT ont été fait sans substrat caché, donc le réseau de neurones phénotypes est similaire à modèle linéaire. Le résultat fut que l'algorithme tombait dans un maximum local de toujours donner la même sortie, auquel cas le classifieur prend "Hommes" par défaut, ce qui donne de meilleur résultat que de donner un résultat au hasard à chaque fois. On a d'abord pensé que c'était à cause de l'extension LEO, mais après deux tests, même si l'un des deux à donné de meilleurs résultats, on remarquait quand même le problème sur l'un des deux. On a donc décidé qu'un individu ne gagnerait pas en fitness s'il donnait trois le même résultat en sortie. Ce qui a légèrement amélioré les résultats, mais il tombait ensuite dans un autre maximum, ou il semblait donner des valeurs similaires à deux sorties assez souvent. Jusqu'à présent pour évaluer les individus on prenait 20 images au hasard dans nos donnés d'entraînement, et on a pensé

qu’en évaluant un individu sur toutes les images d'entraînement on peut être qu’il aurait plus de mal à exploiter les faiblesses de la fonction de fitness. Mais non, et le développement d’un substrat caché avec ES-HyperNEAT n’a rien changé non plus. La solution aurait été de prendre une approche quality-diversity algorithm, car il n’utilise pas de fonction de fitness, et donc celle-ci ne peut pas être exploitée.

On a aussi fait des tests avec le modèle linéaire sur le dataset “Grey” (26 à 28), il a donné des résultats raisonnables considérant les résultats obtenus par les modèles plus complexes.

(Dataset grey95, 34 à 41) On a ensuite fait des tests avec un dataset ou la taille des images de niveau de gris a été divisée par 4. Le résultat pour le PMC avec ce dataset semble un peu moins bon, mais les résultats semblent stables pour le linéaire avec ce dataset. On suppose qu’avoir moins d'entrer aide le modèle linéaire à généraliser sur ce genre de problématique.

Test 43 et 47 reproduction des tests ayant donné les meilleurs résultats sans les coquilles dans le code. Ça n'a pas l’air d’avoir fait une différence importante.

Test 44 à 46, on a créé un dataset plus grand en rognant les images de différentes manières, mais les résultats n’ont montré de flagrantes améliorations.

Test 48 prouve qu’un seul layer était suffisant pour le dataset des couleurs.

Test 49 on a underfitter, on pensait que peut être que sachant qu’il y avait plus de neurones on pouvait faire un entrainement avec moins d’epoch.

Au final nous avons obtenu des résultats plutôt moyens, cette problématique de classification des personnages de League of Legends est peut être trop compliqué pour des PMC et des RBF, peut être qu’il nous faudrait plutôt un modèle CNN.