Rapport du challenge de machine learning

Alexandre Le Roux

28 mars 2025

Préface

Dans le cadre de ce rapport, j'ai choisi d'utiliser une graine aléatoire de 42 pour assurer la reproductibilité de mes résultats. J'ai aussi séparé mes données en 2 ensembles : apprentissage (80%) et validation (20%). J'ai aussi normalisé les deux ensembles en divisant par 255 (correspond au standard RGB). Voir la partie implémentation des données dans le programme Python.

Résumé

Le but du projet est de classifier 10 classes d'images (des avions, des voitures, des oiseaux, des chats, des daims, des chiens, des grenouilles, des chevaux, des bateaux et des camions) d'un *dataset* de 20000 images de taille 32 par 32 munies du canal *RGB*. Pour ce faire, on utilisera différents modèles de *machine learning* étudiés en classe.

Le tableau 1 présente une synthèse des performances obtenues avec les différents modèles testés, incluant leurs paramètres et architectures.

La conclusion est que le meilleur modèle pour ce problème est d'utiliser un réseau de convolution.

Algorithme	**Paramètres**	**Accuracy**	**Optimizer**
K plus proches voisins	$k = 1$	0.3177	Aucun
Régression logistique	$\eta = 1 \times 10^{-2}$ $batch\_size = 64$ $num\_epochs = 276$ $patience = 10$ $dropout\_rate = 0$ $weight\_decay = 0$ $momentum = 0.125$	0.3852	SGD
Réseau de neurones dense	$\eta = 1 \times 10^{-3}$ $h1 = 512$ $h2 = 512$ $batch\_size = 128$ $num\_epochs = 10$ $patience = 5$ $dropout\_rate = 0.1$ $weight\_decay = 1 \times 10^{-4}$	0.4706	Adam
Réseau de convolution	$\eta = 1 \times 10^{-5}$ $out\_channels1 = 128$ $out\_channels2 = 256$ $out\_channels3 = 128$ $h1 = 512$ $h2 = 128$ $kernel\_size = 3$ $stride = 1$ $padding = 1$ $pool\_kernel\_size = 3$ $pool\_stride = 2$ $batch\_size = 64$ $num\_epochs = 14$ $patience = 5$ $dropout\_rate = 0.2$ $weight\_decay = 1 \times 10^{-5}$	0.7498	Adam

TABLE 1 – Comparatif des performances des différents modèles testés

Table des matières

1 Introduction

Munis d'un *dataset* de dimension 3072 correspondant au calcul 32*32*3 [1], on cherche à étiqueter les images sans connaître leur label au préalable.

FIGURE 1 – Échantillon de dix images aléatoires annotées de leur classe

On va implémenter les algorithmes des K plus proches voisins, de la régression logistique multivariée, un réseau de neurones avec des couches linéaires et un réseau de convolution. Avec des paramètres et une architecture bien choisis pour maximiser le taux d'accuracy.

1. Images de taille 32 par 32 munis du canal *RGB* à 3 couleurs

2 Analyse des modèles

Pour évaluer un modèle sur le *dataset*, il faut décommenter la ligne associée au modèle, puis lancer le programme. Pour lire les résultats, il faut aller dans le dossier prédictions, puis le dernier fichier prédiction.

Le graphique 2 retrace l'apprentissage et le taux d'erreur entre chaque *epoch*. Il est accessible dans le fichier *prediction* du modèle. Pour directement afficher le graphique, il faut mettre à *true* le paramètre plt_verbose dans le fichier principal du projet.

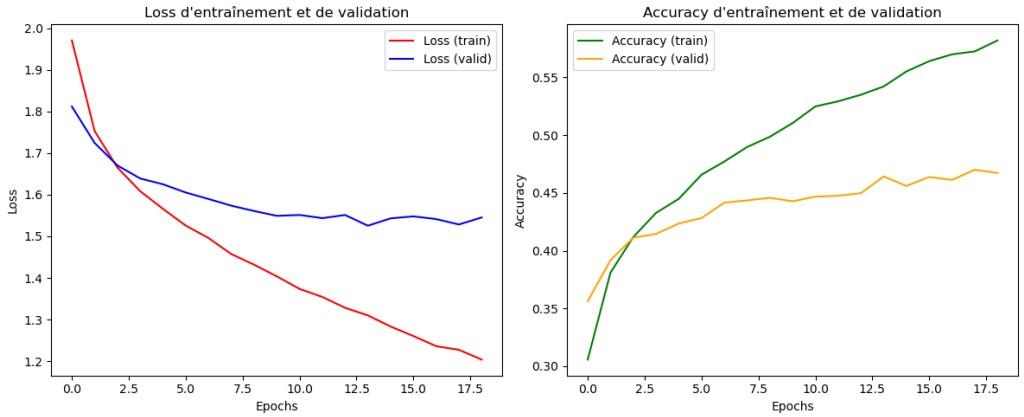

FIGURE 2 – Monitoring des métriques de loss et accuracy du réseau de neurones linéaire munis des hyperparamètres $h1 = 128$, $h2 = 64$, $\eta = 1 \times 10^{-4}$, $batch\_size = 32$, $dropout\_rate = 0.1$, $weight\_decay = 1 \times 10^{-4}$

À noter que plus la courbe oscille, moins le η est pertinent ; elle peut aussi osciller à cause des batchs et notamment en fonction du choix du batch_size.

L'implémentation des modèles de régression logistique, de réseaux de neurones avec couches linéaires et réseaux de convolutions sont respectivement dans *reg_model.py*, *cnnet_model.py* et *cnn_model.py*.

Les fichiers *reg_tunning.py*, *cnnet_tunning.py* et *cnn_tunning.py* permettent de choisir les meilleurs paramètres et la meilleur architecture associée au problème. Ils retournent les fichiers de poids dans le dossier *grid_search_results*. Un historique des performances est mis à jour dans un fichier *csv* nommé *training_history.csv*. Tous les fichiers *training_history.csv* sont sauvegardés manuellement dans le dossier *training_records*.

Pour les modélisations autres queKNN, des *batchs* sont utilisés, pour l'entraînement et la validation. Les *batchs* pour l'entraînement améliorent le taux d'accuracy en décomposant le *dataset* pour l'apprentissage avec le gradient. C'est d'ailleurs un hyperparamètre très important. Les *batchs* pour la validation permettent d'éviter des problèmes de mémoire. Ils sont implémentés par la classe *myDataset*. On pourra au choix sélectionner avec ou sans *batch* dans le programme principal [2].

De plus, les *epochs* choisis sont très hauts et peu pertinents. C'est parce qu'il y a un arrêt anticipé pour éviter le surapprentissage et les plateaux. Ainsi, avec le paramètre patience, si au bout d'un certain nombre d'*epochs* le modèle n'apprend pas, on arrête l'apprentissage. Ça réduit les calculs inutiles. À noter, qu'on retrouve l'*epoch* d'arrêt avec la métrique best_epoch.

2. Fichier *projet.py*

2.1 Algorithme des K plus proches voisins

Dans le programme l'implémentation duKNN est modélisé avec la fonction *best_k* qui nécessite le module **sklearn.neighbors** par des soucis d'optimisation du calcul de la distance. Il y a un seul paramètre qui permet la recherche du meilleur K : *kmax* la borne supérieure de notre intervalle de recherche.

La figure 3 résultat de la fonction *visualize_best_k* indique le meilleur K possible dans un intervalle raisonnable.

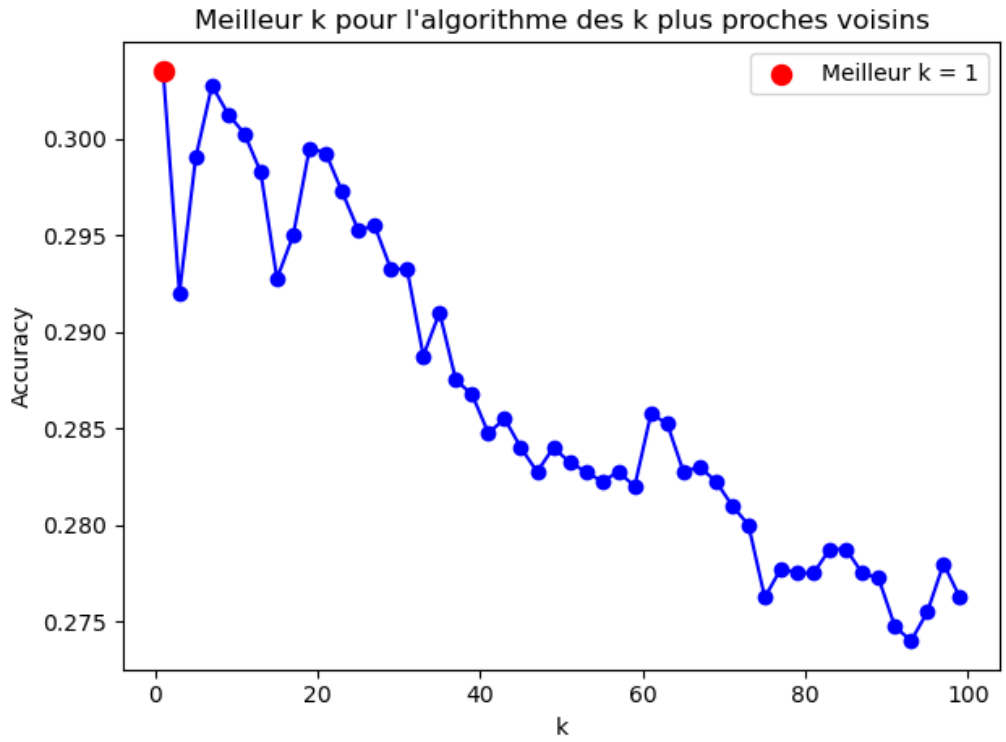

FIGURE 3 – Graphe de l'accuracy duKNN en fonction du paramètre K, avec en rouge le meilleur K possible

On remarque assez vite que plus le K est grand, moins KNN est efficace. Cette modélisation présente des limites pour un problème de classification d'images. En effet, la notion de distance utilisée parKNN (par exemple, la distance euclidienne) n'est pas adaptée pour distinguer des objets avec une

structure, une colorimétrie ou encore une structure, surtout pour des images de petit format.

Par exemple, une voiture orange en 32x32 pixels peut être difficilement distinguable d'un camion orange ou même d'un chat orange, car la distance calculée entre ces images ne reflète pas nécessairement leur appartenance à une certaine classe. La colorimétrie dominante (ici, l'orange) influe principalement sur la mesure de distance, rendant l'algorithme incapable de capturer les caractéristiques qui différencient les objets. On peut aussi citer le problème des textures : un camion est fait d'acier tandis qu'un chat a des poils. Toutes ces caractéristiques font que leKNN n'est pas du tout adapté à notre problème.

Le score d'accuracy atteint tout de même 0.3177 pour $k = 1$ sur $CodaLab$. Même s'il n'est pas adapté à la classification d'image, la couleur partie intégrante d'une image RGB lui permet d'avoir un meilleur score que l'aléatoire qui est de 0.1 ici.

2.2 Algorithme de régression logistique multivariée

La classe du modèle de régression logistique multivariée est *Reg_log_multi* dans *reg_model.py*. Elle atteint un taux d'accuracy de 0.3852 à l'*epoch* 276 (best_epoch).

Pour le *tuning* de la régularisation, deux méthodes sont possibles, la régularisation traditionnelle : la régularisation L2 et le *dropout* initialement voué aux réseaux de neurones. Un autre *tuning* aurait pu être l'*optimizer* mais j'ai choisi de faire tous mes essais avec *SGD*.

Il s'avère que la régularisation traditionnelle sans *dropout* est en tête dans le *tuning* (voir fichier *training_history_all_cnn_3.csv*)

Par ailleurs la figure 4 indique que même sans *dropout*, le surapprentissage n'est pas un problème, la fonction softmax suffit. En revanche, le taux de prédiction n'est pas très bon.

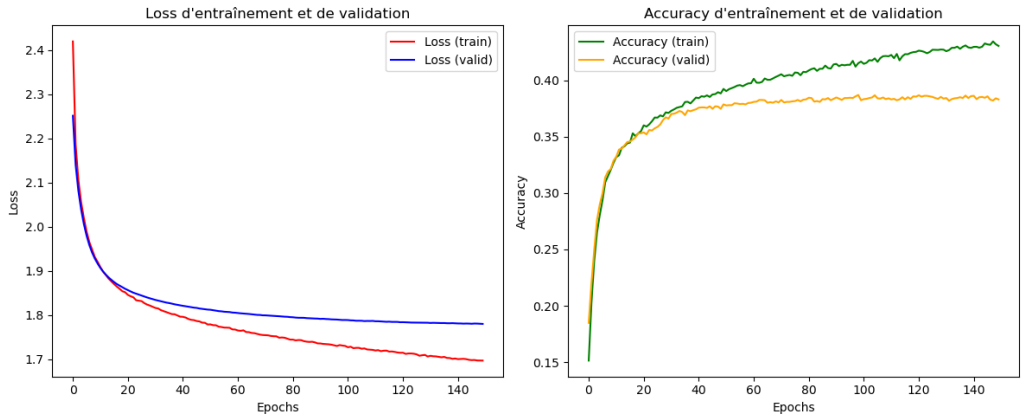

FIGURE 4 – Monitoring des métriques de loss et accuracy du réseau de neurones linéaire munis des hyperparamètres $\eta = 1 \times 10^{-4}$, $batch\_size = 128$, $dropout\_rate = 0$, $weight\_decay = 1 \times 10^{-4}$, $momentum = 0.125$

La proximité du taux de précision avec le KNN est due au fait que l'on a dix classes et que la régression logistique est performante pour des problèmes binaires. Les relations d'une image sont trop complexes pour être approchées linéairement et interprétées avec la fonction en S (sigmoïd) non linéaire.

Le modèle de régression logistique est donc seulement 18% mieux que le KNN.

2.3 Réseaux de neurones avec des couches linéaires

Les deux prochains modèles sont plus adaptés à notre problème. Ils utilisent des couches de neurones, qui, avec la fonction d'activation (ici ReLU), interprètent mieux la classification multivariée. La fonction d'activation joue un rôle clé, en connectant des relations plus complexes qui répondent à la recherche de structure ou même de texture propre à certaines classes.

La classe qui représente le réseau de neurones est $Neural\_network\_multi\_classif$ dans le fichier $cnnet\_model.py$.

Avec le *tuning*, on remarque que plus il y a de neurones, plus le modèle est performant.

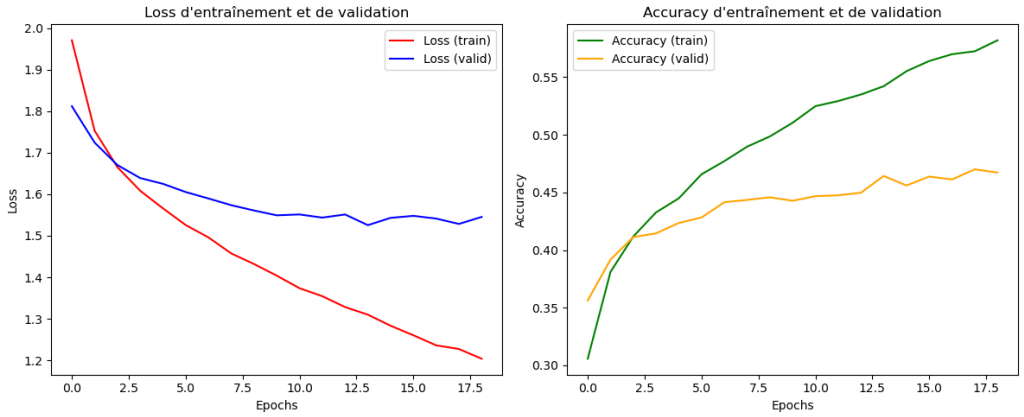

FIGURE 5 – Monitoring des métriques de loss et accuracy du réseau de neurones linéaire munis des hyperparamètres $h1 = 128$, $h2 = 64$, $\eta = 1 \times 10^{-4}$, $batch\_size = 32$, $dropout\_rate = 0.1$, $weight\_decay = 1 \times 10^{-4}$

Malgré cela, le réseau de neurones avec des couches linéaires n'atteint pas un meilleur score qu'un humain qui est de 58.3%, les relations faites au sein des neurones ne sont pas suffisantes. Il n'empêche qu'un réseau de neurones avec des couches linéaires est mieux de 18% que la régression logistique.

2.4 Réseaux de convolution (CNN)

Conv_neural_network_multi_classif représente le modèle du réseau de neurones convolutif dans le fichier *cnn_model.py*.

Le modèle diffère de celui du cours, car il est multiclasse et pas binaire.

On utilise aussi *MaxPool* et *BatchNormalization* pour que le modèle converge bien et pour éviter le surapprentissage, puis la régularisation *dropout* qui aléatoirement "oublie" certaines prédictions.

Le choix des paramètres est crucial, à savoir le nombre de couches, le nombre de neurones par couche. Il y a d'ailleurs deux CNN dans mon projet, un avec 3 couches puis un autre avec 4. Il y a peu de différence, mais le CNN à trois couches est tout de même meilleur. Pour trouver les meilleurs paramètres, on utilise la méthode *gridsearch*. Après analyse des données de *training_history_cnn_4_base.csv* sur la colonne final_accuracy_train on identifie la meilleure architecture et les meilleurs paramètres, qui sont : $out\_channels1 = 128$, $out\_channels2 = 256$, $out\_channels3 = 128$, $h1 = 512$, $h2 = 128$, $dropout\_rate = 0.2$, $\eta = 5\times10^{-5}$,$batch\_size = 64$, $weight\_decay = 1 \times 10^{-5}$

La différence avec les couches linéaires c'est qu'on a en plus la notion de localité dans l'analyse des structures.

C'est le meilleur modèle parmi tous ceux étudiés pour la classification d'images.

3 Conclusion

Le modèle avec le meilleur taux d'accuracy est le réseau de neurones convolutif.

Par comparaison, il est respectivement 37%, 49% et 58% meilleur que le réseau de couches linéaires, l'algorithme de régression logistique multivariée et l'algorithme des K plus proches voisins.

Liste des tableaux

Table des figures

Notations des hyperparamètres

η Taux d'apprentissage (learning rate).

batch_size Taille des lots d'entraînement.

patience Nombre d'epochs sans amélioration avant arrêt anticipé.

num_epochs Nombre maximum d'epochs.

weight_decay Dégradation des pondérations (régularisation L2).

dropout_rate Probabilité de désactivation aléatoire des neurones (régularisation).

momentum Terme d'inertie pour l'optimiseur (*SGD* avec momentum).

Notations d'architecture du modèle

h1 Taille de la première couche cachée.

h2 Taille de la deuxième couche cachée.

h3 Taille de la troisième couche cachée.

out_channels1 Nombre de filtres pour la première couche convolutive.

out_channels2 Nombre de filtres pour la deuxième couche convolutive.

out_channels3 Nombre de filtres pour la troisième couche convolutive.

out_channels4 Nombre de filtres pour la quatrième couche convolutive.

kernel_size Taille du noyau de convolution.

pool_kernel_size Taille de la fenêtre pour l'opération de pooling.

stride Pas de déplacement du noyau de convolution.

pool_stride Pas de déplacement pour le pooling.

padding Remplissage des bords de l'image avant convolution.

Notations des métriques d'évaluation

best_epoch Indice (plus 1) de la meilleure epoch.

best_test_loss Meilleure loss sur l'ensemble de test.

final_train_loss Perte finale sur l'ensemble d'entraînement.

final_test_loss Perte finale sur l'ensemble de test.

final_accuracy_train Précision finale sur l'entraînement.

final_accuracy_test Précision finale sur l'ensemble de test (1 - erreur).

loss Écart entre prédiction et valeurs réelles.

accuracy Taux de précision.

Autres notations

plt_verbose Booléen pour activer l'affichage des graphiques.

KNN Algorithme des K plus proches voisins.

CNN Réseau de neurones convolutifs.