Rapport final
Projet Apprentissage profond

Classification d'images selon le nombre de pokémons, grâce un à réseau de neurones

Groupe A

AFKER Samy
BAURIAUD Laura
BOCANDÉ Thomas
GUIHUR Lilian
MALINGE Romain

Département Sciences du Numérique - 2A Image & Multimédia
2023-2024

Table des matières

Table des figures

1 Description du sujet

1.1 Préambule

Un matin, Samy s'est réveillé et comme à son habitude il a commencé à compter ses pokémons pour être sûr qu'ils étaient tous là. Malheureusement, il a constaté qu'il avait perdu ses lunettes !

Pour aider le pauvre Samy, nous nous proposons de construire un réseau de neurones grâce au conseils du grand Professeur Carlier.

1.2 Énoncé

Le but de notre projet est de classifier à l'aide d'un réseau de neurones des images suivant le nombre de Pokémons présents dessus. Nous nous sommes limités aux quatres classes suivantes pour colé au visuels du jeu Pokemon :

- 1 Pokémon
- 2 Pokémons
- 3 Pokémons
- 4 Pokémons

Nous ferons l'entraînement et la validation de notre réseau sur des images générées puis nous le testerons sur des images provenant du jeu d'origine (Pokémon version Noir). Vous pouvez retrouver notre base de données sur le GitHub de notre projet.

2 Constitution de notre base de données

2.1 Images générées

Pour les ensembles d'entraînement et de validation, nous avons récupéré 30 arrière-plans sur des sites divers ainsi que les sprites avant et arrière de 649 pokémons sur le site Pokekalos.

Ensuite, nous avons implémenté un programme generateur.py qui permet de générer un ensemble d'images contenant des Pokémons avec les paramètres suivants :

- Nous choisissons le nombre de Pokémons, de 1 à 4, par image
- Les Pokémons sont placés de façon aléatoire
- Les Pokémons peuvent déborder de $\frac{1}{3}$ de leur taille sur les bords de l'image
- Les Pokémons ont une proximité maximale de $\frac{2}{3}$ de leur taille (ce qui permet la superposition de Pokémons)
- Les Pokémons sont agrandis avec une échelle aléatoire entre 1 et 1,5
- Les Pokémons sont placés de face ou de dos de façon aléatoire
- L'arrière plan à 20% de chance d'être assombri
- 50% des images ont 2 éléments du jeu en plus
- L'images finale a une dimension de 256 x 192

Le programme est écrit de manière à enregistrer les images dans le dossier de la classe correspondante ce qui permet d'annoter les données automatiquement. Voici un échantillon d'images générées par notre programme :

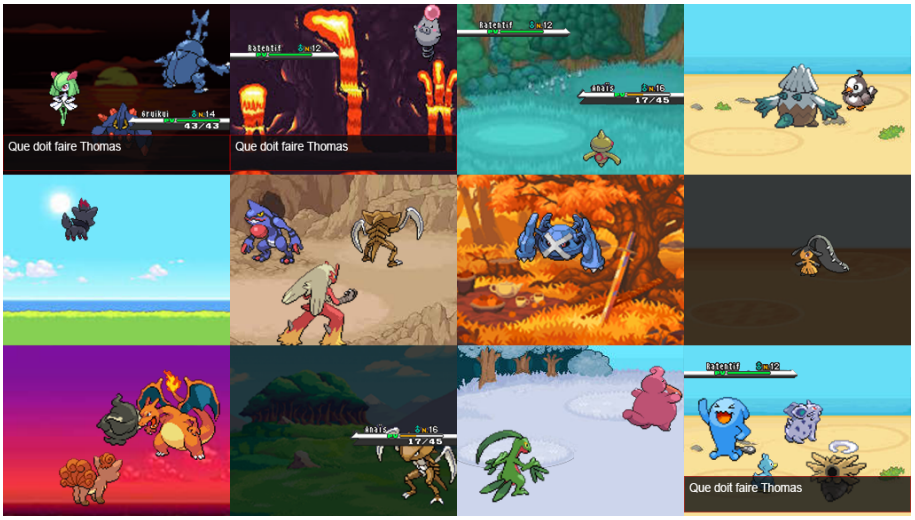


FIGURE 1 – Échantillon d'images générées

2.2 Images du jeu

Pour l'ensemble de test, les images ont été obtenues directement dans le jeu *Pokémon version Noir* sur l'emulateur DeSmuME. Les captures d'écran on été réalisées avec le script auto-screenshot.py, puis elles ont été recadrées avec le script rognage.py.

Pour les images du jeu, nous avons fait une annotation manuelle. Voici un échantillon des images obtenues du jeu.

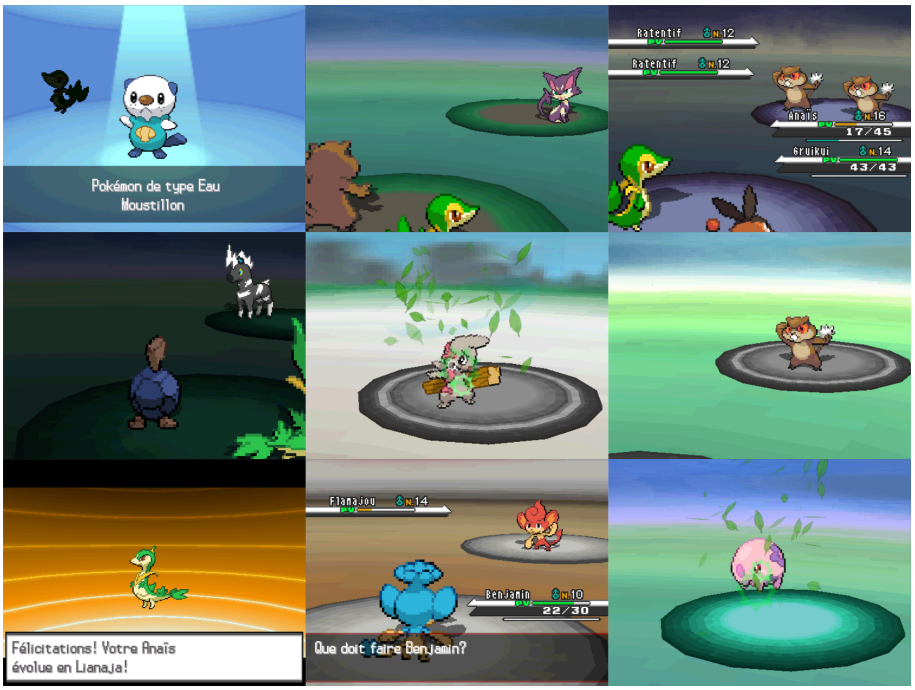


FIGURE 2 – Échantillon d'images du jeu

2.3 Partition des données

Comme nous disposons d'un programme pour générer automatiquement des images, nous avons pu prendre un grand nombre d'images. Notre première répartition de base de donnée est la suivante :

Nom de l'ensemble	Nombre d'images	Précision
Entraînement	4 000	Généré par le programme
Validation	1 000	Généré par le programme
Test	100	Issues du jeu *Pokémon Version Noire*

Table 1 – Partition de la BD 1

Nous avons choisi de placer une grande partie de nos images dans l'ensemble d'apprentissage pour augmenter nos chances que le réseaux apprenne correctement à compter les pokémons. Il y a peu d'images dans l'ensemble de test car les possibilités de disposition sont limitées dans le jeu. Il n'est donc pas utile d'avoir plus d'image dans cet ensemble.

Suite au conseilles de notre professeur, nous avons grandement augmenté la taille de notre base de donnée (de 5 000 à 101 000 pour l'apprentissage) dans le bute de réduire le sur-apprentissage de notre réseaux. Notre nouvelle répartition est la suivante :

Nom de l'ensemble	Nombre d'images	Précision
Entraînement	100 000	Générer par le programme
Validation	1 000	Générer par le programme
Test	100	Issues du jeu *Pokémon Version Noire*

Table 2 – Partition de la BD 2

Cette deuxième version nous a permit de faire une augmentation de données importante, sans déformer nos images, qui sera utilisée dans le version 3 de notre réseaux pour obtenir de meilleur résultats.

3 Différentes résolutions du problème

Dans cette partie, nous allons voir les différentes étapes qui nous ont permis d'obtenir la version finale de notre réseau. Voici la légende pour comprendre les schémas de nos modèles :

FIGURE 3 – Légende pour les schémas des modèles

3.1 Première version : réseau pour 2 classes

Dans un premier temps, nous avons commencé par travailler sur une version simplifiée du problème avec seulement 2 classes (1 ou 2 pokémon·s) et ceci sur notre première base de donnée avec seulement 2500 images pour le moment car on n'utilise que 2 classes sur 4.

Le but de cette première résolution était de mettre en place une version fonctionnelle pour partir sur une bonne base :

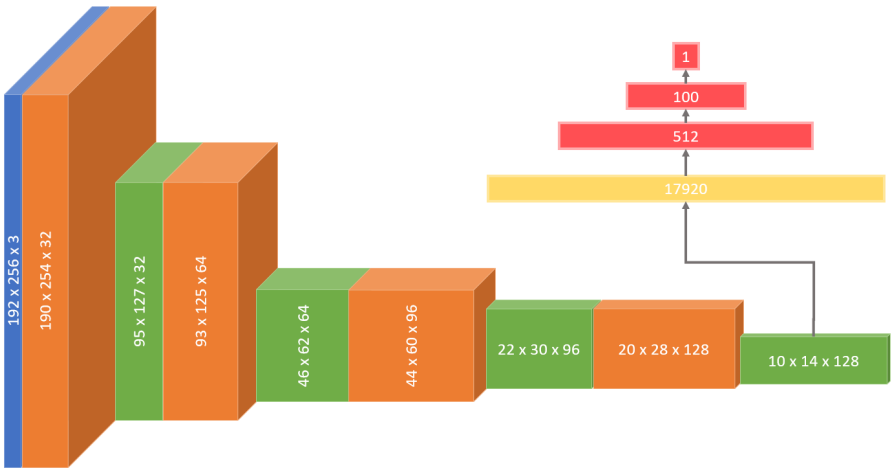
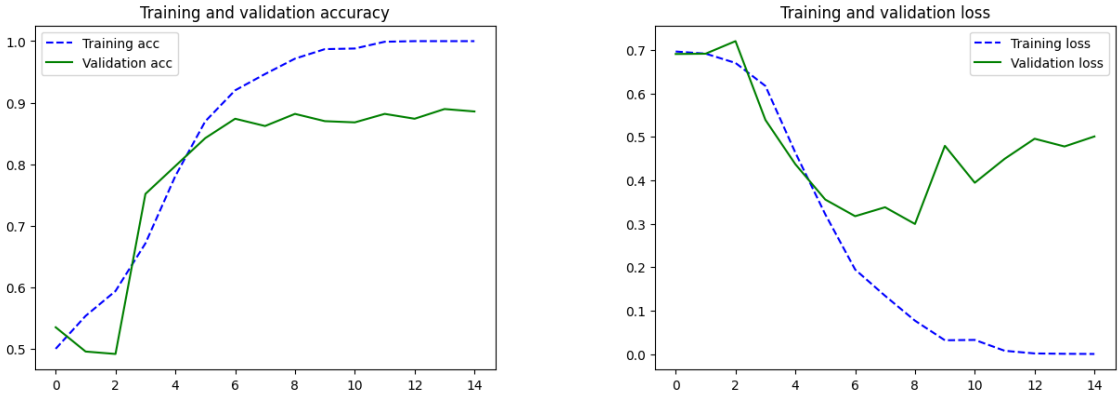


FIGURE 4 – Modèle de la version 1

Dans cette première version, la dernière couche dense est de dimension 1 et possède une fonction d'activation de type sigmoïde. De plus, pour la fonction de perte, nous avons choisi l'entropie croisée binaire avec un learning rate de $3e-4$.

Cette première version, nous a permis d'une part d'avoir un réseau fonctionnel et d'une autre part d'avoir la confirmation qu'il est possible pour notre réseau d'apprendre à "compter". En effet, notre problème n'est pas de (simplement) reconnaître un Pokémon mais bien de savoir combien il y en a sur l'image. Grâce à ce premier test nous avions la confirmation que ce modèle est une bonne piste pour y arriver.

Le résultat sur les environ 50 images de l'ensemble de test est de **78**%. Cela Peut paraître comme un bon résultat mais dans cette version binaire un résultat aléatoire correspond à 50% donc ce n'est pas si satisfaisant.



(a) Accuracy de la version 1

(b) Loss de la version 1

FIGURE 5 – Résultats de la version 1

Ici, on remarque que l'accuracy sur l'ensemble de validation stagne à environ 87%. Cela signifie que notre réseau surapprend. C'est confirmé par la loss de validation qui se met à augmenter au moment ou la stagnation commence (époque 6). Pour cette version la matrice ne présente pas beaucoup d'information, nous analyserons plus en détail les suivantes.

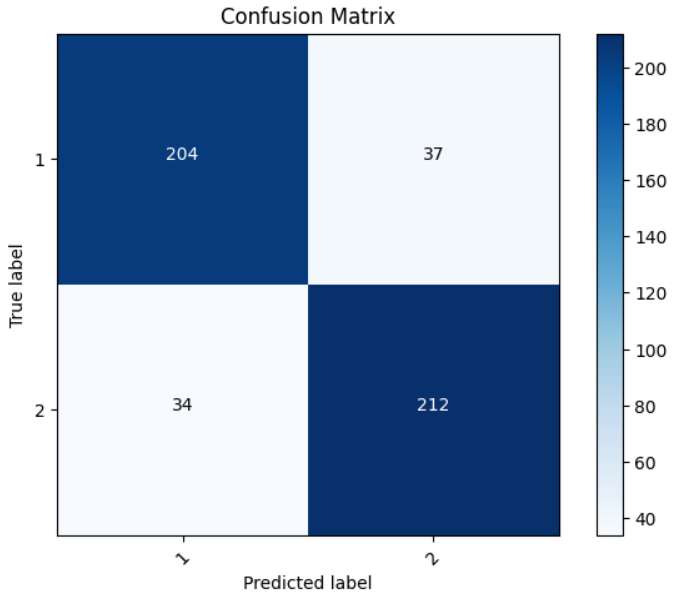


FIGURE 6 – Matrice de confusion pour l'ensemble de validation de la version 1

3.2 Deuxième version : réseau pour 4 classes

Une fois cette première version fonctionnelle, nous avons mis en place une version complète du problème qui prend en compte les 4 classes :

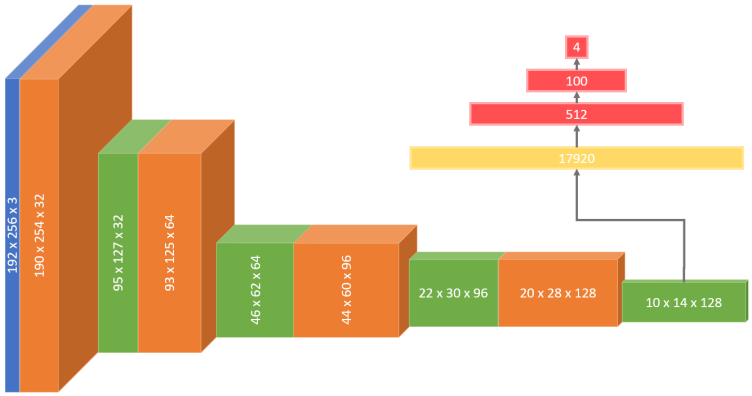
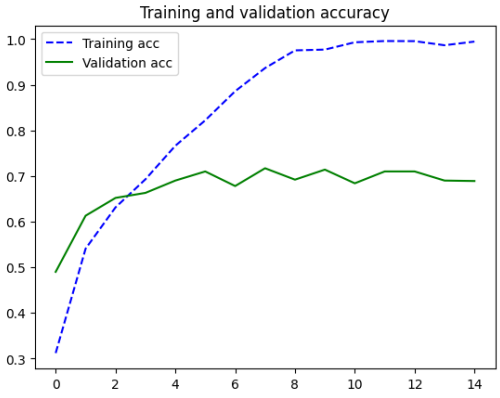
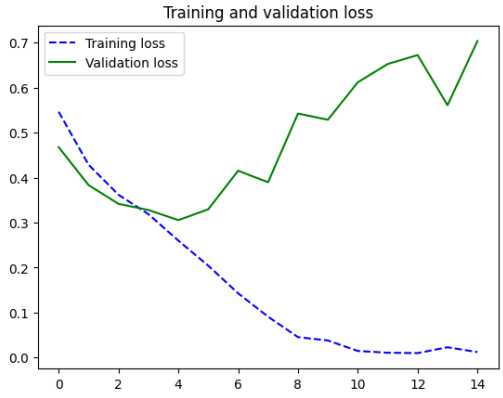


FIGURE 7 – Modèle de la version 2

Dans cette deuxième version, la dernière couche dense est maintenant de dimension 4 et a une activation de type softmax. Comme escompté, notre réseau parvient à compter les Pokémons. Le résultat sur les 100 images de l'ensemble de test est de **40**%. Comme un résultat aléatoire correspond à 25%, le test sur les images du jeu est très mauvais. Nous verrons plus tard comment expliquer ce faible pourcentage.



(a) Accuracy de la version 2



(b) Loss de la version 2

FIGURE 8 – Résultats de la version 2

On remarque que notre réseau apprend bien à classifier car on obtient une accuracy sur l'ensemble de validation de l'ordre de 70% pour les 4 classes. Il est tout de même important de noter que notre réseau souffre d'un très fort surapprentissage (stagnation à partir de l'époque 5). Ce problème sera résolu dans à la version 3 de notre modèle grâce à la seconde base de données.

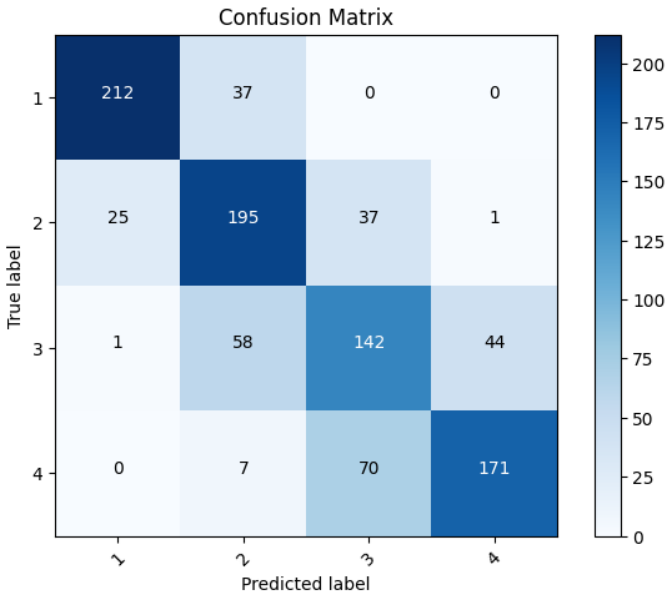
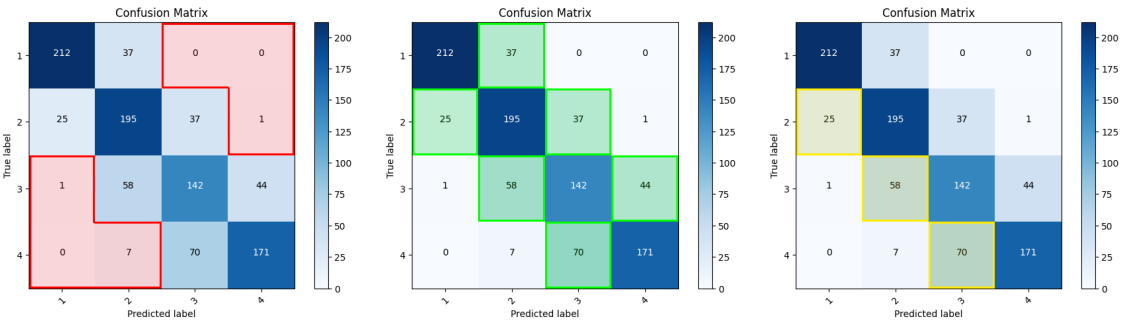


FIGURE 9 – Matrice de confusion pour l'ensemble de validation de la version 2

On a bien la diagolane qui se demarque sur la la matrice de confusion ce qui traduit nos bon résultat sur l'ensemble de validation. Elle nous permet également de mieux comprendre les points clé de l'apprentissage notre réseau.

Premièrement, il ne se trompe quasiment jamais en prédisant une classe éloignée de plus d'un Pokémon (cases rouges). En revanche, il à plus tendance à se tromper en prédisant une classe voisine (cases vertes). Enfin on constate que les erreurs augmente avec le nombre de Pokémons à compter (cases jaunes).



(a) Cases rouges (b) Cases vertes (c) Cases jaunes

FIGURE 10 – Analyse matrice de confusion

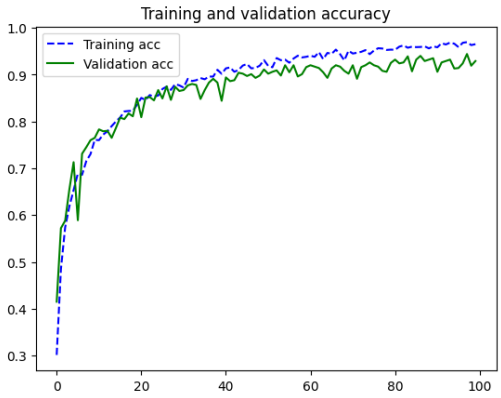
Ces trois caractéristiques traduise des trait quasiment humain de notre réseau. En effet, il est parfaitement naturel de plus se tromper entre 2 et 3 qu'entre 1 et 4 ou encore entre 3 et 4 qu'entre 1 et 2. Nous verrons dans la prochaine version des exemples d'images les plus difficiles à classer.

3.3 Troisième version : BD complète

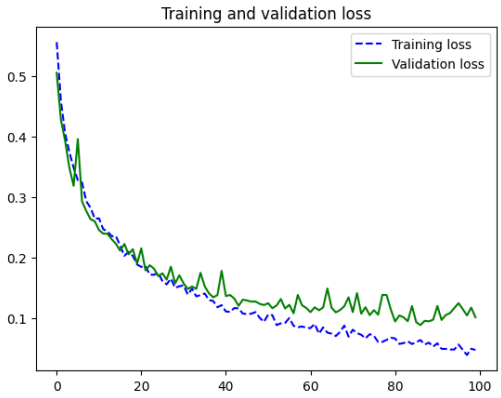
Le modèle de cette version est le même que celui de la précédente. La seule amélioration est au niveau de la base de données. Nous prenons maintenant en compte 100 000 images pour l'ensemble d'entraînement. En effet, il nous est possible d'augmenter la base de données autant que l'on souhaite car cette dernière est composée d'images générées.

Une fois la base de données augmentée, nous avons rencontré un problème au niveau du chargement des images sur collab. L'espace mémoire offert par collab n'était malheureusement pas suffisant pour charger toute la base de données.

Après quelques recherches, nous avons découvert qu'il était possible de charger les données par petits paquets grâce aux ImageDataGenerator (par le suite nous avons appliquer cette manière de faire à toutes nos versions pour sa simplicité d'utilisation). Nous avons donc pu entraîner notre réseau sur l'ensemble de la base de données :



(a) Accuracy de la version 3



(b) Loss de la version 3

FIGURE 11 – Résultats de la version 3

Le résultat sur les 100 images de l'ensemble de test est de **43**%. On a une légère amélioration par rapport à la version 2 mais le test sur les images du jeu reste très mauvais.

On remarque que l'augmentation de la base de données a permis de réduire significativement le surapprentissage ! En effet, l'accuracy sur l'ensemble de validation et l'ensemble d'entraînement sont très proches et se situent autour de **95**% ce qui est un très bon résultat.

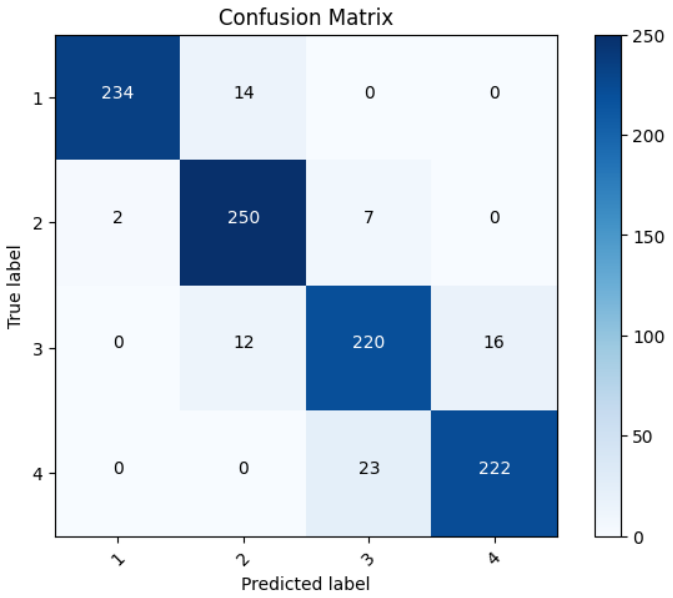


FIGURE 12 – Matrice de confusion pour l'ensemble de validation de la version 3

On remarque que, grâce à l'augmentation de la base de données, les effets évoqués précédemment se renforcent sur la matrice de confusion. En effet, la diagonale est très fortement marquée ce qui témoigne de la haute accuracy. Le réseau ne fait aucune erreur entre deux classes non voisines. Et il en fait toujours plus si il y a plus de Pokémons à compter.

Voila un échantillons des images mal classifiées par la version 3 du réseau pour mieux comprendre les difficultés rencontrées (2 par classe) :

FIGURE 13 – Exemple d'image mal classifiées par la version 3

On peut ainsi constater que les éléments qui posent le plus problème sont :

- la proximité de position et de couleur entre 2 Pokémons (en rouge)
- le ressemblance avec l'arrière plan (en jaune)
- le fait que certains Pokémons soient bicolores (en vert)

3.4 Quatrième version : ResNet

Dans cette version nous nous sommes appuyés sur le réseau ResNet50 déjà préentraîné avec la banque d'image ImageNet dont nous avons traiter les sorties grâce à une couches d' AveragePooling suivie de couches denses avec comme fonction d'activation ReLu pour finir sur une couche dense avec une fonction d'activation de type softmax.

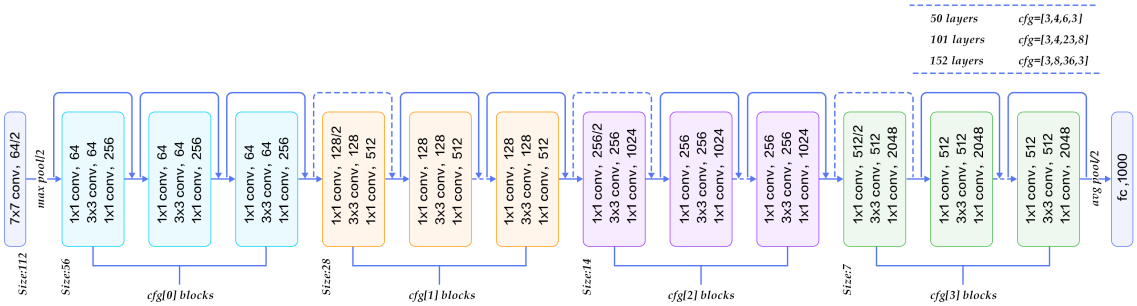
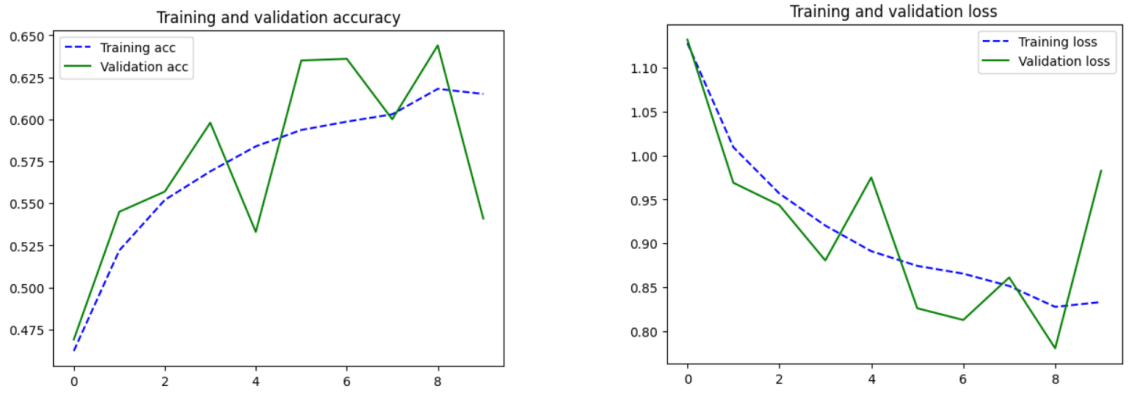


FIGURE 14 – Rappel modèle ResNet

Ce modèle est légèrement plus performant que la version 3 avec **47**% de bonnes classifications sur les 100 images de test mais reste peu performant pour compter les Pokémons sur les images réelles du jeu.

L'une des raison de la faible performance du réseau est le nombre d'époques sur lesquelles il a été entraîné. En effet la complexité du réseau et la taille très importante de l'ensemble de données d'entraînement fait qu'il faille environ 1h 40 pour entraîner le réseau sur 10 époques. Les résultats de l'entraînement :



(a) Accuracy de la version 4

(b) Loss de la version 4

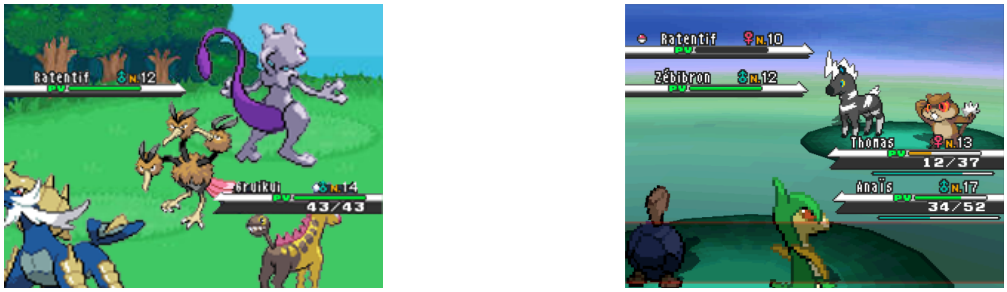
FIGURE 15 – Résultats de la version 4

Dans cette partie la matrice de confusion n'a pas pu être faite à cause d'une surcharge de la RAM lors de sa génération.

4 Analyse des résultats

Notre réseau convolutif présente de très bon résultats sur l'ensemble de validation (**95**% pour la meilleure version). Malheureusement, les résultats sur l'ensemble de test sont beaucoup moins bons (**43**% pour la meilleure version).

Cela était relativement attendu car les images de l'ensemble de test (du jeu Pokémon version Noir) sont assez différentes de celles générées. Nous avons tenté de reproduire les visuels du jeu en ajoutant des éléments issues de ce dernier :



(a) Image générée (b) Image du jeu

FIGURE 16 – Exemple d'image bien reproduite

Mais certaines images du jeu sont trop éloignées pour que le réseau puisse comprendre :

(a) Boites de texte (b) Effet de particule (c) Éléments au premier plan

FIGURE 17 – Exemple d'image du jeu complexe

De plus, les images d'entraînement ont toutes été générées grâce à notre script ce qui fait que la structure globale de toutes les images reste la même (malgré l'ajout des éléments du jeu). Notre base de données n'est donc pas suffisamment généraliste. Tout ceci explique pourquoi notre réseau a beaucoup de mal à classifier des images provenant du jeu Pokémon.

Pour avoir de meilleur résultat sur les images de l'ensemble de test il faudrait construire une base de données provenant entièrement du jeu ce qui demande beaucoup plus de temps pour atteindre une taille suffisante et d'avoir recours à l'augmentation de données artificielle.

5 Conclusion

En conclusion, notre projet visait à construire un réseau de neurones capable de classifier des images en fonction du nombre de Pokémons présents dessus. Nous sommes parvenus à faire fonctionner un réseau à 4 couches pour notre problème. Après plusieurs tentatives de modification sur les hyperparamètres (batchsize, learningrate, nombre d'epoche) nous avons trouvé une configuration que nous n'arrivions plus a améliorer. Cela nous a amené à concentrer nos efforts pour améliorer la base de données (taille et variation de visuelle).

Une fois l'entraînement terminé avec succès, nous avons testé notre réseau sur des images provenant du jeu Pokémon version Noir, afin de vérifier sa capacité à généraliser à des données réelles. Les résultats obtenus ont été assez médiocres en raison de la grande variété d'images possibles. En effet nous avons rencontré des difficultés pour que la génération de données d'entraînement soit représentative de toutes les situations possibles, ce qui a limité la capacité du modèle à généraliser efficacement.

Pour améliorer notre projet à l'avenir, plusieurs pistes sont envisageables. Tout d'abord, il serait bénéfique d'augmenter la diversité des données d'entraînement en ajoutant des images représentant des environnements variés et des configurations de Pokémons plus complexes. De plus, d'autres types de modèles sont envisageables pour améliorer les performances.

En somme, notre projet a été une première étape prometteuse vers la création d'un système de classification automatique des images de Pokémons. Bien que des défis subsistent, les perspectives d'amélioration sont nombreuses. Enfin, ce projet nous a permis d'avoir une première expérience passionnante et formatrice dans le domaine du Deep Learning.

6 Annexe

- Lien vers le Google Colab de la version 1
- Lien vers le Google Colab de la version 2
- Lien vers le Google Colab de la version 3
- Lien vers le Google Colab de la version 4