APPRENTISSAGE AUTOMATIQUE

Travail fait par: Mohamad HAWCHAR HAWM20039905
Nassir Ade-Dayo ADEKOUDJO ADEA04089904

26 avr. 2022

TRAVAIL PRATIQUE 2

Professeur: Mohamed BOUGUESSA

1. Introduction

Les ordinateurs ont suscité beaucoup d'enthousiasme et d'attentes dans les années cinquante, quand ils ont commencé à battre au jeu de dames certains amateurs de bon niveau. Dans les années soixante, les chercheurs espéraient reproduire les fonctions du cerveau humain avec un ordinateur et des programmes informatiques.Au milieu des années 2000, le rêve de construire des machines aussi intelligentes que des humains avait presque été abandonné par la communauté scientifique par manque de ressources matérielle.Les ordinateurs d'alors n'étaient pas capables de réaliser des calculs complexes réalisés avec facilite de nos jours.Avec l'avènement de nouveaux processeur les travaux ont été repris en apprentissage automatique plus précisément en apprentissage profond.

L'apprentissage profond est une discipline de l'apprentissage automatique qui utilise plusieurs couches d'algorithme sous la forme d'un réseau de neurones.L'apprentissage profond est exploitée dans plusieurs domaines notamment en vision par ordinateur ou par exemple elle aide dans les tâches comme de la segmentation sémantique, la détection d'objets et la classification qui fera l'objet de notre travail.Le but de notre travail est de faire la classification de six espèces d'animaux.Un template contenant un exemple du code qui doit être implémenté est prévu à cet effet.Notre template en question est un code qui a été implémenté pour réaliser la classification sur le jeu de données MNIST des chiffres 2 et 7.Il est constitue d'un fichier *Modele.py* dans lequel a ete realise une architecture et entraîner le modèle à la classification et un second fichier *Evaluation.py* dans lequel se fait l'évaluation sur les données de test, le traçage des courbes de précision et de perte, la matrice de confusion ainsi qu' un bref aperçu des images mal classées.Notre travail consistera a dans un premier a partir de zéro concevoir une architecture capable d'entraîner notre modèle pour resoudre notre probleme puis, dans un second temps évaluer notre algorithme sur de sorte à avoir une précision d'au moins 90% sur les données de test.

2. Montage de l'architecture et entraînement du modèle

2.1. Ensemble de données

Les données constituent un ensemble de 30000 images provenant de six espèces d'animaux contenues chacun dans un dossier contenant le nom de l'espèce et proportionnellement répartis en données d'entraînement contenant 24000 images pour toutes espèces confondus et de test contenant 6000 images.

Notons que la partie entraînement de notre modèle se fait à deux niveau: une partie sur les données d'entraînement et une autre partie sur les donnees de validation.Il convient donc de separer nos données en données d'entrainement qui prennent 90% des données d'entraînement proprement dit et en données de validation qui prennent 10% des données d'entraînement.

	Eléphants	Girafes	Léopard	Rhinoceros	Tigres	Zèbres
Données d'entraînement	4000	4000	4000	4000	4000	4000
Données de test	1000	1000	1000	1000	1000	1000

2.2. Traitement des données

Dans le cadre du traitement,étant donné que les données de notre TP sont des images, il convenait alors de faire une configuration des images notamment de la mise à l'échelle(scaling) de hauteur et de largeur à 120, fixer le nombre de canaux a 3 pour obtenir les composantes rouge vert et bleu et le mode de couleur en rgb afin que toutes les images puissent avoir les mêmes caractéristiques au cours de leur passage dans le réseaux de neurones et ce pour faciliter l'apprentissage.Aussi les couleurs sont tres important pour la classification des animaux parce que comme nous pouvons le constater par exemple les girafes n'ont pas la meme couleur que les elephants.

Nous avons aussi essayé l'augmentation de données notamment l'horizontal flip, shear_range, zoom_range, width shift_range, height shift range qui a réussi à augmenter le temps d'entraînement du réseau de neurone compte tenu de la transformation sur toutes les images qui constitue les données d'entraînement de notre jeu de données avant de passer dans notre architecture pour la phase d'entraînement.

2.3. Paramètres et hyperparamètres

2.3.1. L'optimiseur

Pour notre modèle l'optimiseur finalement retenu a été l'optimiseur **ADAM** compte tenu de sa vitesse ainsi que de sa qualité de convergence vers une solution optimale.Aussi avec l'hyper-paramètre du taux d'apprentissage fixe a 0.001 nous remarquons que le modèle met moins de temps à converger vers le minimum global

2.3.2. La taille du lot (batch size) d'entraînement

Dans l'optique de gagner en terme de temps d'apprentissage et de précision du modèle nous avons eu à essayer plusieurs paramètres en ce qui concerne la taille du lot.Nous avons essayé avec des tailles de lot (batch_size) de 20, 32, 64 et nous en sommes arrivés à la conclusion que plus la taille du lot est petite plus le modèle gagne en précision et en temps d'apprentissage.Nous avons donc jugé bon de choisir une taille de lot de 32 ce qui s'avère intuitivement être un bon compromis entre précision et temps.A noter qu'avec une taille de lot de 32 nous avons obtenu la meilleure précision vis-à-vis de notre modèle.

2.3.3. Le nombre d'époques

En ce qui concerne le nombre d'époques nous avons eu à le fixer à 100 parce que compte tenu des méthodes utilisées pour la régularisation notamment la régularisation l2 et le dropout on constate un ralentissement de l'apprentissage de la part du modèle.Pour palier à ce problème nous avons dû fixer le nombre d'époques à 100 et grâce à cela obtenu une précision de 94% sur les données d'entraînement.

2.4. Architecture

Notre réseau de neurones est subdivisé en deux parties: une première partie pour l'extraction des caractéristiques et une seconde partie pour les couches entièrement connectées.

- **L'extraction des caractéristiques (features extraction)**

Cette première partie contient différentes couches ayant pour but d'extraire les caractéristiques propres à chaque image en les compressant de manière à réduire leur taille initiale.En résumé l'image fournie en entrée passe par une succession de filtres et il en résulte de nouvelles images qui sont concaténées dans un vecteur de caractéristiques.

La couche de convolution: Cette couche a pour rôle d'analyser toutes les images fournies en entrée et d'en relever un ensemble de caractéristiques appelle feature map qui nous indique exactement ou est situe notre caractéristique dans l'image.

La couche de correction ou couche Relu: cette couche fait intervenir une fonction d'activation , la fonction ReLu dans notre cas dans le but d'améliorer l'efficacité du traitement tout en intercalant entre les couches de traitement.

La couche de Pooling: cette couche se base sur les résultats obtenus de la couche de convolution.Son but est de recevoir en entrée les features maps et de réduire la taille tout en gardant les caractéristiques importantes de l'image.Cette couche converse la valeur moyenne de la fenêtre de filtre.

La couche dropout: Cette technique consiste à désactiver ou éteindre aléatoirement un neurone pendant la phase d'apprentissage et ceci dans le but de permettre à chaque neurone de bien apprendre évitant ainsi la co-adaptation.

La couche de normalisation par lot (batch normalization): cette technique permet de rendre plus efficace les neurones en normalisant leurs entrées de couche par recentrage et remise à l'échelle.L'avantage à noter avec cette méthode est qu'elle évite même avec un taux d'apprentissage élevé l'explosion du gradient et permet d'atténuer le surajustement tout en réduisant considérablement le temps d'apprentissage.Elle permet de maintenir une certaine stabilité des poids d'éviter le décalage des valeurs et d'ajouter la distribution des poids d'une couche cachée à une autre.

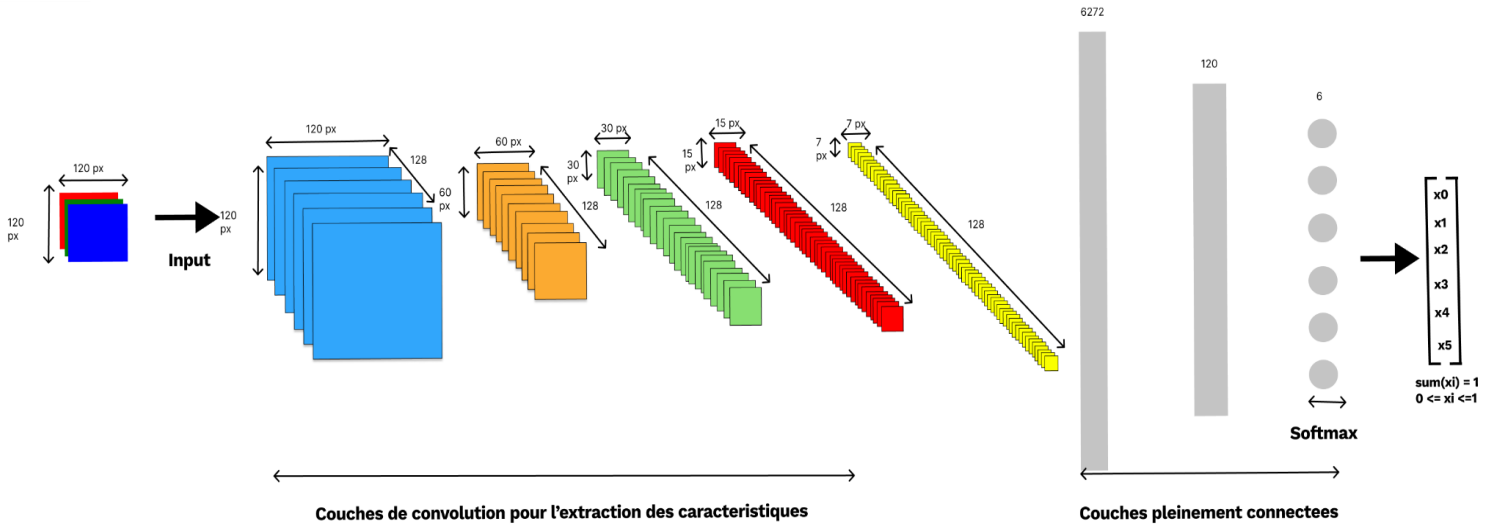
En effet, notre architecture de la première couche telle que décrite a été utilisée quatre fois dans cette première partie de notre architecture de modèle avec des couches de dropout qui mettaient hors service 20 à 30% des neurones.

- **<u>La couche pleinement connectée(fully connected)</u>**

La couche entièrement connectée est la dernière couche de notre architecture de réseau de neurone convolutif.Elle reçoit un vecteur en entrée et retourne un vecteur en sortie.Elle permet de classifier l'image en entrée du réseaux; elle renvoie un vecteur(de taille 6 dans notre cas) ou 6 est le nombre de classes dans notre modèle de classification d'images.Chaque élément du vecteur indique la probabilité pour l'image en entrée d'appartenir à une classe. Chaque valeur du tableau en entrée "statue" en faveur d'une classe. Les votes n'ont pas tous la même importance : la couche leur accorde des poids qui dépendent de l'élément du tableau et de la classe.

Le réseau de neurones convolutif apprend les valeurs des poids de la même manière qu'il apprend les filtres de la couche de convolution : lors de phase d'entraînement, par rétropropagation du gradient.Pour calculer les probabilités, la couche *fully-connected* multiplie donc chaque élément en entrée par un poids, fait la somme, puis applique une fonction d'activation (**Softmax** dans notre cas).Ce traitement revient à multiplier le vecteur en entrée par la matrice contenant les poids. Le fait que chaque valeur en entrée soit connectée avec toutes les valeurs en sortie explique le terme *fully-connected*.La couche *fully-connected* détermine le lien entre la position des *features* dans l'image et une classe. En effet, le tableau en entrée étant le résultat de la couche précédente, il correspond à une carte d'activation pour une *feature* donnée : les valeurs élevées indiquent la localisation (plus ou moins précise selon le *pooling*) de cette *feature* dans l'image. Si la localisation d'une *feature* à un certain endroit de l'image est caractéristique d'une certaine classe, alors on accorde un poids important à la valeur correspondante dans le tableau.

Nous avons eu à insérer dans les couches pleinement connectées des couches de dropout et de régularisation l2 parce que le nombre de paramètres de cette couche est plus important(752760 paramètres) que celle des autres couches.



2.5. Affichage des résultats d'entraînement

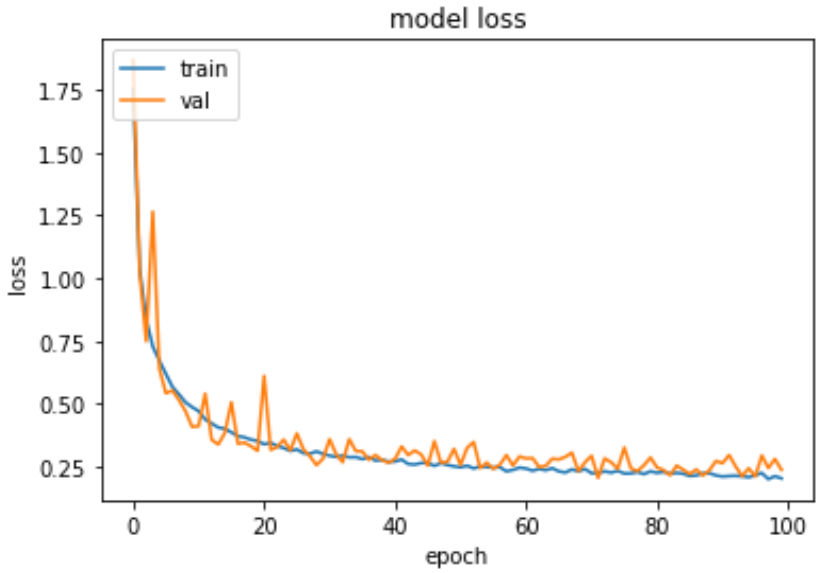
L'entraînement de notre réseau de neurones s'est avéré concluant notamment du point de vue de l'exactitude(accuracy) des données d'entraînement mais aussi de l'erreur commise(loss).

2.5.1. Temps total d'entraînement

Notons que l'entraînement de notre modèle a été effectué avec une taille de lot de(batch_size) 32 et ce sur 100 époques(epocs) ceci pour permettre au modèle d'atteindre l'exactitude(accuracy) fixe pour notre travail.Le temps mis a cet effet pour entraîner notre modèle est de 132 minutes en raison de 79 secondes en moyenne d'entraînement par époque.A travers les époques nous constatons que le modèle atteint une exactitude de 90.71% déjà à la vingt-cinquième époque.

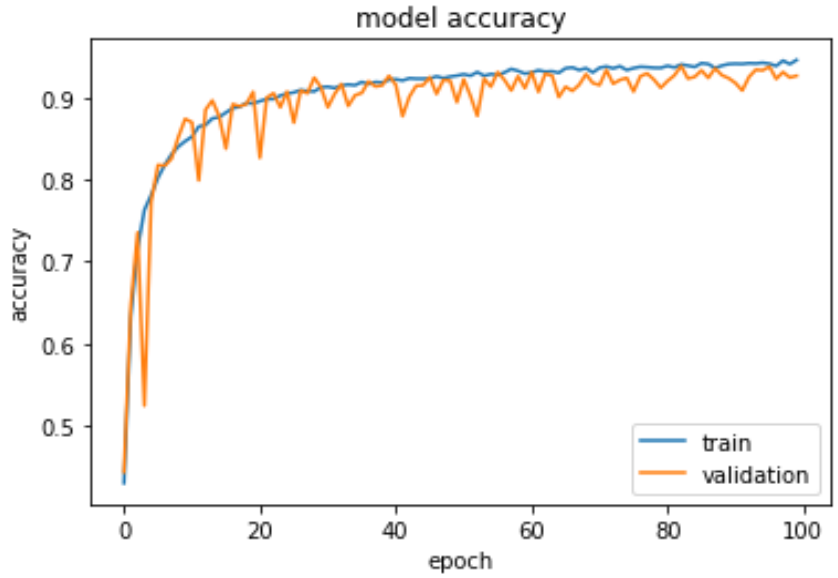
2.5.2. Erreur minimale obtenue lors de l'entraînement

L'erreur minimale commise pour les données d'entraînement sur notre modèle est de 20.30%.Ci-dessous le comportement de l'erreur minimale sur les données d'entraînement et de validation:



2.5.3. Exactitude maximale de l'entraînement

Il est aussi à noter que le modèle à l'époque 93 a atteint une exactitude maximale de 94.56%.Ci-dessous le comportement de la courbe de précision sur les données d'entraînement et de validation.



2.6. Justification du choix de l'architecture

L'architecture finalement adoptée a été issue non seulement du choix d'un choix minutieux des couches des deux parties de notre architecture à savoir les parties extractions des caractéristiques et pleinement connectées mais aussi des paramètres et hyperparamètres qui s'y attachent. Nous allons donc dans cette partie discuter de la pertinence des différentes couches de notre architecture ainsi que la pertinence des paramètres et hyperparamètres qui s'y attachent.Nous avons déjà eu tout haut à décrire l'architecture utilisée.Dans cette partie nous allons nous accentuer sur la raison de l'adoption de ces caractéristiques.

Dans la partie d'extraction des caractéristiques(feature extraction) nous avons eu à concevoir quatre couches contenant quasiment les 4 à 5 sous-couches autant importantes les unes que les autres.

La première sous-couche étant la couche de convolution nous permet comme nous l'avons dit tout haut d'extraire les caractéristiques importantes en appliquant des filtres de dimension 128 pour chaque image qui passe dans le réseau.

La deuxième sous-couche est une couche de correction qui nous permet d'améliorer l'efficacité du traitement en appliquant une fonction d'activation sous les caractéristiques issues de la couche de convolution.Nous avons eu à choisir dans notre architecture la fonction d'activation **Relu** parce qu'elle est la fonction la plus simple et la plus suggérée pour les problème de classification.Nous avons aussi essayé une fonction **Sigmoïde** mais elle donnait de mauvais résultats.Elle effectue

un filtre sur les données du tenseur passe à travers la couche qui la constitue et renvoie la valeur si la donnée est positive et 0 sinon.

Notre troisième sous-couche est la couche de pooling qui sous-échantillonne l' input reçu de la couche de correction.Elle nous permet de réduire la taille tout en conservant les informations importantes.De l'input reçu est cree une fenetre 2 par 2 dans notre cas de laquelle est choisi le maximum qui sera placé en sortie pour la prochaine sous-couche.En observant attentivement notre architecture nous pouvons remarquer que a chaque couche la taille de l'input est divisé par 2.La fonction de cette couche est d'aboutir a une reduction considerable de la quantité d'opération et aussi de la mémoire.À l'aide du paramètre padding fixe a "same" permet aux entrées de la couche d'avoir une uniformité dans le remplissage en tout point de l'entrée.

La quatrième sous-couche étant la couche de Dropout permet nous l'avons dit plus haut une indépendance des neurones et l'hyperspécialisation d'un neurone dans la détection d'une seule catégorie d'animaux par exemple mais au contraire aide le neurone a avoir un bon pouvoir de généralisation.Nous avons eu à ajuster plusieurs fois les valeurs et ce dans le but de permettre une bonne régularisation et éviter l'overfitting.Nous avons fixé dans notre cas l'hyper paramètre du taux d'extinction à 0.2 ce qui signifie le fermeture aléatoire de 20% des neurones dans les trois premières couches et a 0.3 dans la dernière couche parce qu'il contient plus de paramètres que les autres couches (147584 paramètres) et la couche dense a plus(752760) c'est pour cela nous avons à certains endroits des taux d'extinction plus élevés par rapport à d'autres niveau de notre architecture.En effet au cours de notre entraînement lorsque le taux d'extinction était fixé à 0.5 le modèle avait tendance à faire de l'overfitting.Ce qui nous a poussé à entrainer notre modèle sur 100 époques et à réduire le taux d'extinction.

Dans notre objectif d'atteindre une meilleure précision, il est important de veiller à ce que notre modèle soit généralisé de façon appropriée.Il s'agirait donc de s'assurer à ce que l'échelle des valeurs suivant chaque couche soit équilibré et que les neurones des couches cachées ne s'exposent pas à une variation de la distribution des données qui risque de se propager tout le long des couches et par conséquent affecter la sortie.C'est pour palier à ce problème que nous avons introduit dans nos couche avant le passage à la couche suivante une couche de normalisation de données appelé la normalisation par lots(Batch normalisation).Cette couche de normalisation par lot a pour but de maintenir la stabilité des poids et éviter le décalage des valeurs d'une couche a une autre en évitant le changement de la distribution statistique des sorties ce qui pourrait rendre l'apprentissage ou l'entraînement très lent.

Dans la bloc de couche pleinement connectée nous avons 7 couches qui interviennent dont 2 couches de Dropout ,2 couches de correction faisant intervenir une fonction d'activation et deux couches densément connectées.

Les couches de dropout de ce bloc ont leur taux d'extinction fixe à 0.5 et ce parce que c'est à ce niveau que se fait la classification.Le calcul se doit d'être réalisé avec minutie pour éviter la dépendance de performance entre les neurones pour que ces derniers ne se spécialisent pas dans la classification de certaines classes aux détriments d'autres.

Après notre couche de dropout s'en suit une couche densément connectée de 120 neurones avec une régularisation l2(ridge regression).Ce paramètre évite au modèle de mémoriser les données et ainsi éviter l'overfitting.Nous avons d'abord eu a essayer avec la régularisation l1 pour lequel lorsque l'algorithme atteint une précision de 88% ralenti l'apprentissage.La régularisation l2 semble être plus tolérant envers de taux d'apprentissage que la régularisation l1 ce qui a orienté nos choix vers elle.

La couche densément connectée est suivie d'une couche Relu qui, nous l'avons dit plus haut, améliore l'efficacité du traitement.

Nous avons ajouté après la couche Relu une autre couche de Dropout dont le taux d'extinction a été fixé aussi a 0.5.

Après la couche de Dropout nous avons eu à rajouter une couche densément connectée de 6 neurones ce pour chaque catégorie d'animaux et ceci pour préparer la sortie(classification) des images.

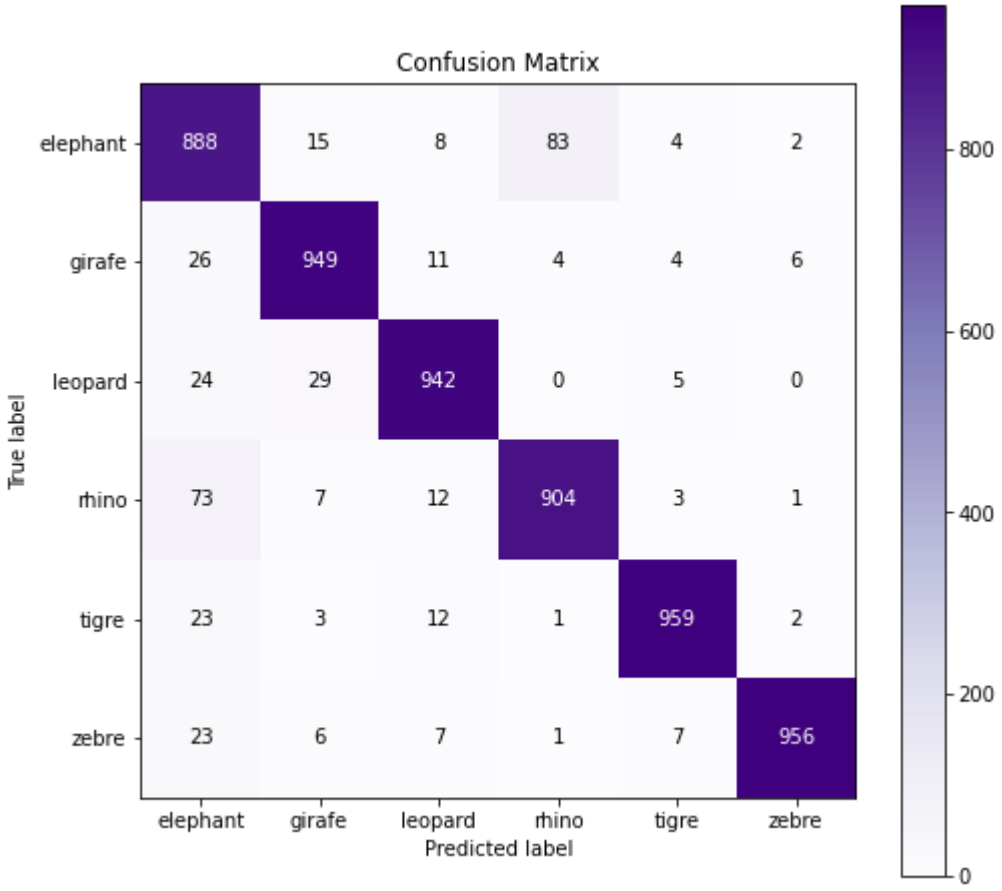
Finalement nous avons ajouter au neurone de sortie une fonction d'activation **softmax** ceci parce que nous somme dans ce cas de figure en présence d'un problème multiclasse et que s'agissant des problèmes multi classe pour les neurones de sortie la fonction softmax se trouve la mieux appropriée pour resoudre notre probleme.

3. Evaluation du modèle

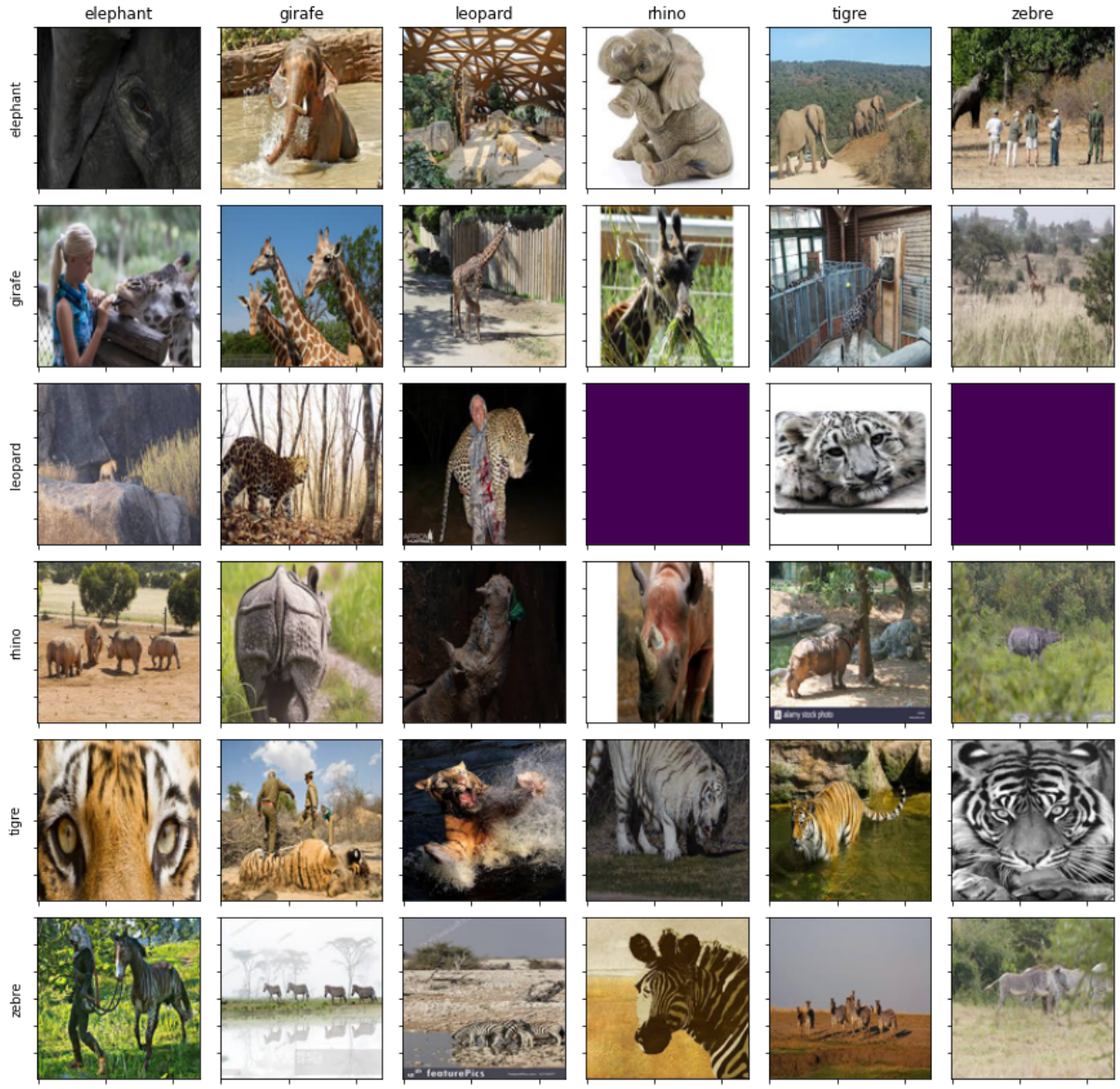
Après la phase d'entraînement notre modèle est soumis est soumis à une phase de l'évaluation ou il reçoit en entrée des images issues des données de test prévues à cet effet.Le modèle reçoit donc des input de 6000 images en entrées et ce pour chaque catégorie d'animal(1000 images par classe) et doit être à même de prédire la classe correspondante à chaque images.

3.1. Exactitude du modele developpe sur les données de test

Pour cette phase nous constatons que le modèle met 47.8 secondes pour évaluer les données et il en résulte une précision de 93.08% pour une perte de 24.14%.Les performances sont relatées par la matrice de confusion ci-après:



Comme le relate la matrice de confusion ci-dessus, nous pouvons constater que nous n'avons pas de cas où les léopards sont prédits comme des rhinocéros et comme des zèbres.



Ci-dessous nous notons que les exemples mal classés par l'algorithme sont pour la plupart imperceptibles voire indéchiffrables à l'œil nu.

Conclusion

Somme toute, après avoir analysé les résultats du modèle sur les données de test, il convient de constater que ce dernier excelle dans la classification des tigres, des zèbres,des girafes et confond beaucoup les éléphants et les rhinocéros.Malgré cela il faudrait aussi noter que beaucoup de facteur ont contribué à donner ce résultat notamment l'utilisation des fonction d'activation appropriés au niveau des couches intermédiaires, les techniques d'initialisation des poids, la normalisation par lots, le choix d'un bon optimiser , les techniques d'atténuation de l'overfitting et bien d'autres encore.

Cette étude a présenté un réseau de neurone convolutif capable de distinguer six espèces d'animaux.Notre jeu de données contient 30000 images dont 24000

pour les données d'entraînement et 6000 pour les données de test.Les données d'entraînement sont à leur tour divisées en données d'entraînement et de validation dans des proportions respectivement de 90% et 10%.

D’un point de vue global, notre modèle s’est avéré assez probant en obtenant une précision de 94 % sur les données de test.Mais particulièrement le modèle a du mal à distinguer les rhinocéros et les éléphants et a obtenu 88% pour la prédiction des éléphants et 90% pour la prédiction des Rhinocéros.Ce sont la les taux de prédiction les plus faibles de notre évaluation.

D’autres problèmes rencontrés sont l’overfitting au bout duquel on est arrivé par les couches de dropout et la regularization l2.Aussi nous notons qu'à un moment donné le modèle arrête d’apprendre.Pour solution nous avons utilisé une architecture de quatre couches de convolution avec des filtres de 128 pixels.En outre l’affichage des images mal classés aussi s’est avéré être un problème.

Pour améliorer une approche serait d'étendre la longueur et la largeur de l’image a plus de 120 ce qui permettrait aux filtres de convolution d’extraire le plus de caractéristique et par conséquent aux couches suivantes de contribuer à améliorer la prédiction.