Projet Kaggle

Machine Learning

Classification d'un sous-jeu de données ImageNet

Image générée par l'IA générative de Microsoft Designer

Maxime Moshfeghi

Toulouse INP - ENSEEIHT
France

29 juin 2024

Table des matières

Préambule

Pour un soucis de taille limite d'upload sur Moodle, les fichier au format `.pth` ont été supprimé du rendu, tout comme l'ensemble des images d'entraînement et de test. Au besoin, l'archive complète est disponible au lien Google Drive suivant :

`https://drive.google.com/drive/folders/1IEfeynR0WzWYsJ08O4LyJrGJdYJRPMC8?usp=drive_link`

Introduction

Dans un contexte d'essor de l'intelligence artificielle, une grande partie de son usage est concentré sur l'analyse des images par des réseaux de neurones convolutionnels (abrégés CNNs pour Convolutional Neural Networks). Dans certains cas, on peut faire de la détection d'objet pour localiser un élément au sein d'une image (c'est le cas de YOLO par exemple, [2]). Dans notre cas, nous chercherons plutôt à classifier des images selon l'ensemble de leur contenu. Il s'agira de classifier un sous-jeu de données de ImageNet de 4000 images en 4 classes qui vont de 1 à 4 que l'on pourrait nommer : "chou", "chou-fleur", "tente", "alvéoles".

Chapitre 1

Construction et entraînement des modèles

1.1 Données en entrée : augmentation et traitement

Dans un premier temps, on peut se pencher sur le traitement des données en entrée. Les données seront systématiquement convertient en leur nuances de gris et seront réajustées à une taille de 256×256. On appliques des transformations aléatoires de rotation et de symétries.

1.2 LeNet

1.2.1 Structure

Le premier modèle que j'ai cherché à implémenter est un très petit modèle : le modèle **LeNet**. Il s'agit d'un réseau de neurones contenant 2 couches convolutionnelles et 3 couches de neurones denses. Voici un schéma de ce modèle :

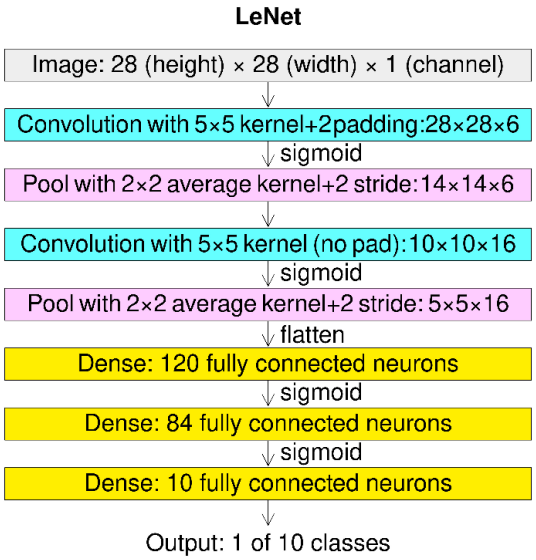


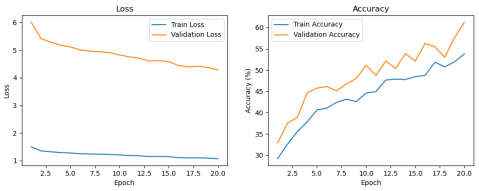
FIGURE 1.1 – Schéma de la structure du réseau **LeNet**

Bien sûr, la dernière couche est à ajuster selon notre besoin, ici nous avons 4 classes.

En plus de cette architecture de base, j'ai décidé d'ajouter des dropouts derrière chaque couche dense que je décidais d'activer ou non (selon mes entraînements).

1.2.2 Entraînement

On entraîne le modèle sur une vingtaine d'epochs, voici les graphiques d'entraînement associés :



(a) Première et dernière couche dense en dropout

(b) Toutes les couches denses en dropout

(c) Aucune couche dense en dropout

FIGURE 1.2 – Courbes de loss et d'accuracy pour différents dropout

On voit que la première courbe présente des accuracies qui semblaient encore en croissance à la vingtième epoch. Il pourrait être judicieux de continuer l'entraînement sur 30 ou 40 epoch pour voir si l'on a une convergence de l'accuracy.

Finalement, avec **LeNet**, on réussit de temps à autre à dépasser la barre des 60% de bonne classification, on peine cependant à aller plus haut.

Les points positifs à noter pour cette architecture sont tout de même sa facilité d'implémentation et la rapidité de son entraînement. D'ailleurs, on peut tout à fait s'imaginer que dans le cas d'une classification d'images plus simples (moins d'aléa dans les points de vue sur l'objet, moins de variété de forme de l'objet, etc...), ce réseau pourrait amplement suffir pour obtenir une classification satisfaisante.

On va maintenant implémenter un modèle plus complexe pour tenter d'augmenter les accuracies.

1.3 ResNet

1.3.1 Structure

On implémente maintenant un réseau **ResNet**, dont l'architecure est précisée par l'image ci-dessous :

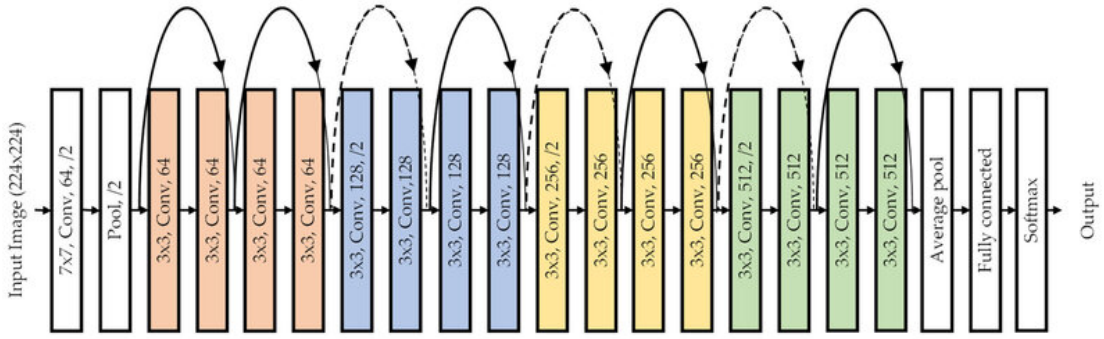


FIGURE 1.3 – Schéma de la structure du réseau **ResNet**

Là encore, il est nécessaire d'adapter les channels en entrée et en sortie, dans notre cas respectivment 2 et 4.

1.3.2 Entraînement

Un premier entraînement donne les losses et accuracies suivantes :

FIGURE 1.4 – Courbes de loss et d'accuracy pour l'entraînement du **ResNet** sur 20 epochs

On constate que pour certaines epochs (entre la quinzième et la vingtième notamment), l'accuracy de validation atteint déjà presque 70%. Même si l'entraînement d'une vingtaine d'epochs est long (environ une heure à chauqe fois), j'ai fait le choix de réentraîné 4 fois de plus, pour atteindre ainsi 100 epochs. Voici le graphe obtenu :

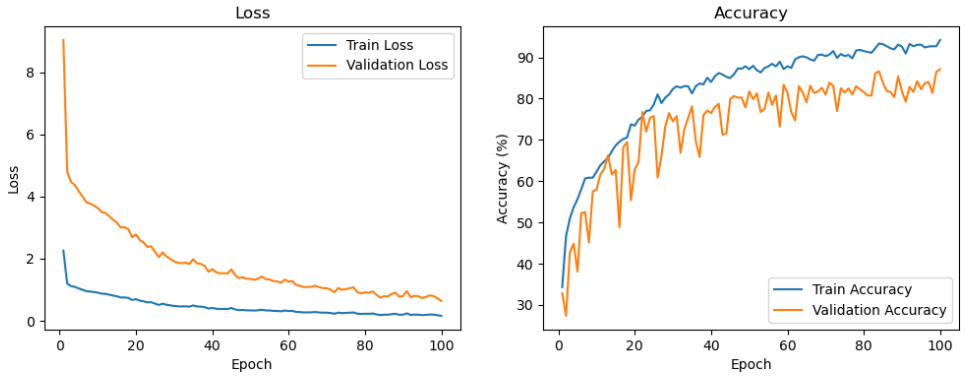


FIGURE 1.5 – Courbes de loss et d'accuracy pour l'entraînement du **ResNet** sur 100 epochs

L'accuracy de validation dépasse maintenant parfois les 85%. On remarque tout de même une tendance à la convergence.

Chapitre 2

Étude du biais des modèles

Nous allons maintenant nous intéresser au biais des modèles implémentés. Pour ce faire, nous allons devoir dans un premier temps :

— filter uniquement 2 classes d'images : je n'ai gardé que les classes 1 et 2 ; ou pouvait aussi les agréger 2 à 2

— séparer aléatoirement le dataset d'entraînement : si pour la partie précédente les fichiers n'étaient pas physiquement séparés dans deux dossiers, j'ai préféré ici les séparer dans deux répertoires distincts, un pour l'entraînement et un pour la validation

— créer les fichiers csv référençant chacun des fichiers de chaque répertoire

Ensuite, les données seront chargées de deux manières différentes. Pour le dataset d'entraînement, on ajoute un channel de la taille de l'image défini de la manière suivante :

$$\textbf{channel2}(\textbf{image}_i) \sim \begin{cases} \mathcal{N}(\mathbf{0}, \mathbf{I}) & \text{si } \textbf{classe}(\textbf{image}_i) = 1 \\ \mathbf{0} & \text{sinon} \end{cases}$$

Pour le dataset de validation, le deuxième channel est déterminé de la manière suivante :

$$\textbf{channel2}(\textbf{image}_i) \sim \begin{cases} \mathcal{N}(\mathbf{0}, \mathbf{I}) & \text{avec une probabilité } p = \dfrac{1}{2} \\ \mathbf{0} & \text{avec une probabilité } p = \dfrac{1}{2} \end{cases}$$

Cela va nous permettre de voir si, à l'ajout d'un biais sur notre dataset d'entraînement, la prédiction ensuite faite suit aveuglément le biais sur lequel les données on été entraîné ou bien si la prédiction se fait toujours bien, malgré un biais différemment réparti.

D'autre part, il va être aussi nécessaire de modifier les modèles pour qu'ils prennent désormais les deux channels en entrée, et qu'il renvoient uniquement 2 channels en sortie.

Voici ce que l'on obtient pour le modèle **LeNet** :

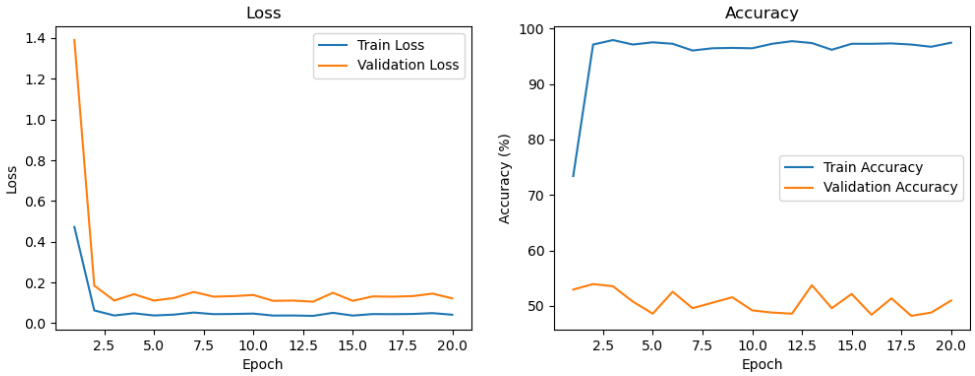


FIGURE 2.1 – Entraînement du modèle **LeNet** sur des données biaisées

On observe alors une très bonne convergence vers 1 pour l'accuracy de train, ce qui est "suspect" car on en était très loin lors de l'entraînement du modèle avec un seul channel d'entrée. Ces suspicions sont confirmées par l'accuracy de validation car on remarque qu'elle reste proche de 0.5, c'est-à-dire que cela revient à un tirage aléatoire : le modèle s'est donc appuyé essentiellement sur le biais qu'on lui a ajouté pour sa phase d'entraînement.

On observe le même phénomène avec le modèle **ResNet** :

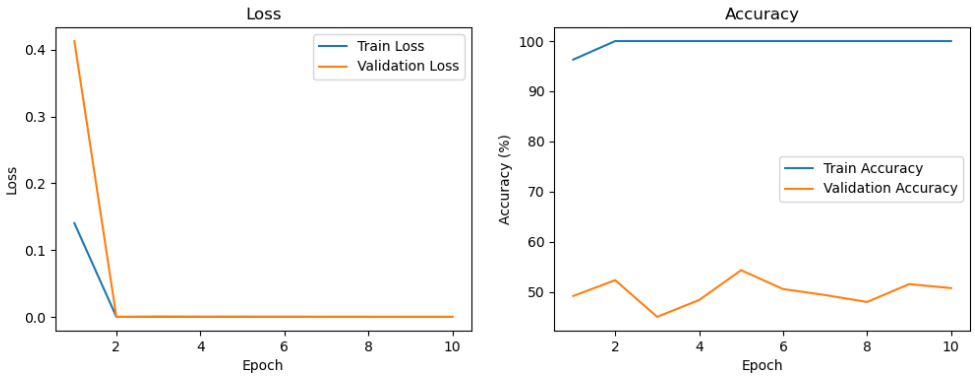


FIGURE 2.2 – Entraînement du modèle **ResNet** sur des données biaisées

Remarque : pour pousser les tests de sensibilité au biais, il aurait été intéressant d'ajouter du biais sur les données d'entraînement avec une certaine probabilité pour le cas $\textbf{classe}(\textbf{image}_i) = 1$ et non plus systématiquement (cf. [1]) , mais je n'ai pas eu le temps de l'implémenter.

2.1 Conclusion

Si un réseau simple comme **LeNet** permet déjà de réussir à classifier correctement des images avec une probabilité supérieure à 0.6, on voit que l'on peut aussi monter beaucoup plus haut en complexifiant le modèle. C'est ainsi qu'avec **ResNet**, j'ai réussi à obtenir en test plus de 88% de bonne classification (meilleur upload sur Kaggle).

D'autre part, il est intéressant de constater que les modèles sont sensibles au biais, c'est-à-dire que si un attribut est ajouté aux données d'entraînement selon leur classe, les modèles ont tendance à abstraire complètement le contenu de l'image originale au profit de ce nouvel attribut. On peut alors imaginer la fragilité des modèles face à des situations réelles (supposons par exemple que l'on demande à une personne 1 d'aller photographier des tentes et à une personne 2 de photographier des choux, la classification pourrait être biaisée par le fait que la personne 1 avait un certain appareil photo tandis que la personne 2 avait un autre appareil photo, ou encore que les personnes 1 et 2 n'avaient pas les mêmes paramètres d'ouverture sur leur appareil respectif).

Bibliographie

[1] Philippe Besse. Conformité européenne des systèmes d'ia : outils statistiques élémentaires. *Statistique et Société*, 10(3) :25–28, 2022.

[2] Pamudu123 Ranasinghe. Yolov8 comparison with latest yolo models. YouTube : `https://www.youtube.com/watch?v=QOC6vgnWnYo`, 2023.