Projet Kaggle

Karima Ghamnia, Cassandra Mussard

ENSEEIHT & INSA Toulouse - 4ModIA
2022-2023

1 Introduction

Le but de ce projet est de comparer les modèles de l'état de l'art sur une partie du dataset Imagenet. Pour réaliser cela on choisit d'étudier quelques modèles parmi ces 4 :LeNet, AlexNet, ResNet, Modèle Hybride. Ce projet est réalisé en Pytorch.

2 Dataset

Le dataset Imagenet est l'un des datasets les plus connu dans le milieu du Deep Learning. Il est constitué de 14 millions d'images avec près de 1000 classes. Dans ce projet, nous avons a disposition une petite partie du dataset Imagenet. Il est constitué de 4000 images pour l'entraînement et 1000 pour le test. Il y a au total 4 classes (1 : chou-fleur, 2 : tente, 3 : chou, 4 : rayon de miel). Le modèle choisit doit donc être capable de prédire si l'image correspond à une des 4 classes précédentes.

Dans le code Pytorch nous créons une classe où nous définissons la fonction getitem. Cette fonction permet de récupérer l'indice de l'image dans le dataset. Il suffit alors de récupérer le chemin de l'image et de l'ouvrir avec cv2. On transforme ensuite l'image en tenseur pytorch et on redimensionne en (224, 224). On récupère le label associé à l'image et on stocke ceci dans un dictionnaire avec deux clés (l'image et le label). Nous transformons ensuite ces images en dataloader pour obtenir des batchs.

3 Augmentation de données

L'augmentation de données est une technique très utilisée en Machine Learning, et particulièrement dans les tâches de classification d'images. L'idée de cette méthode, est d'augmenter la taille de notre *trainset*, afin de permettre à notre modèle de mieux généraliser sur d'autres images, être plus robuste, et éviter l'overfitting.

C'est pourquoi, nous avons appliqué cette méthode dans notre projet avec les transformations suivantes :
- Filtre gaussien : Enlever le bruit en appliquant un flou gaussien.
- RandomVerticalFlip : Inverser l'image verticalement d'une manière aléatoire.
- ColorJitter : apporter des vriations aléatoires aux couleurs de l'image.
- RandomRotation : Effectuer des rotations aléatoirement aux images.
- ElasticTransform : Simuler des déformations locales aléatoires aux images.
- GrayScale : Transforme l'image en niveau de gris.
- RandomPerspective : Transforme la perspective d'une image selon une certaine probabilité.
- RandomInvert : On inverse aléatoirement les couleurs d'une image.

4 Description du modèle utilisé et Fine Tuning

Nous avions le choix entre plusieurs modèles : AlexNet, Resnet, LeNet, Hybrid Models.

Premièrement, nous avons décidé de ne pas essayer le réseau LeNet (premier réseau de neurones développé par LeCun en 1998), car celui-ci est adapté à des images en niveau de gris (avec un seul canal) et le réseau prend en entrée des images de taille (28x28 car adapté pour les MNIST). Comme nous avons des images de taille 256x256 nous serions obligés de réduire la taille des images à 28x28 et nous perdrons beaucoup d'informations. De plus, nous avons des images avec 3 canaux.

Nous avons donc commencé par essayer le modèle AlexNet. C'est le premier modèle qui a gagné le challenge sur le dataset Imagenet complet.

Ce modèle est constitué de 5 blocs.

Un bloc est définit par une couche de convolution, une couche de normalisation de batch, une couche d'activation, et une couche de MaxPooling. Une fois l'image passée dans les 5 blocs, on passe dans les couches fully connected (qui correspondent à un réseau de MLP) où chaque pixel de l'image en entrée va être transformé. Cette partie permet ensuite de passer d'une image où l'on a extrait les features à une prédiction (1,2,3 ou 4).
L'architecture de ce réseau de neurones se trouve ci-dessous.

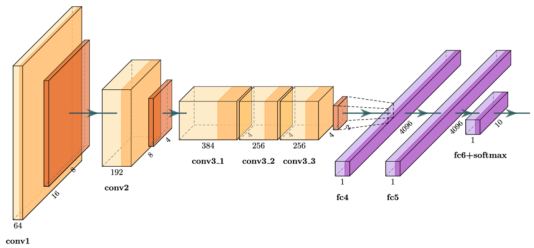

FIGURE 1 – Architecture AlexNet

Nous avons obtenu une accuracy maximale de 84% avec les hyperparamètres suivants :
- 70 epochs.
- Algorithme SGD avec moment et pas de $1 * 10^3$.
- CrossEntropy comme fonction de coût.
- batch de 64 pour l'entraînement et 1 pour le test.
- Régularisation dans les couches fully connected de 0.5 (on sélectionne aléatoirement 50% des neurones qui ne seront pas activés pendant une epoch).

Nous avons obtenu la loss suivante sur les données d'entraînement :

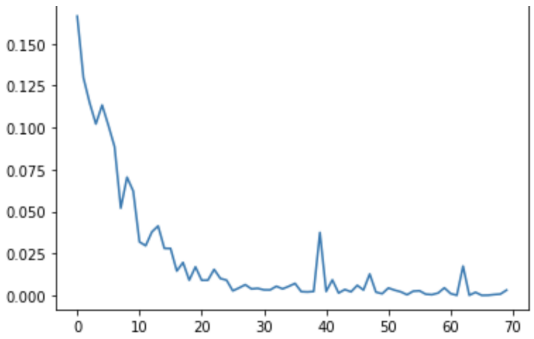

FIGURE 2 – Loss avec l'architecture AlexNet

Nous avons donc décidé de changer d'architecture en prenant l'architecture Resnet. En regardant le benchmark du dataset Imagenet nous avons vu que ce réseau était beaucoup plus performant que AlexNet.

ResNet50 est un réseau de neurones de la famille des réseaux résiduels. Ce réseau de neurones possède 48 couches de convolution, une couche de Maxpooling et une couche de Average Pooling. Dans certains cas, le gradient devient trop petit et en rétropropageant l'erreur dans le modèle les

poids n'arrivent plus à être modifiés. ResNet utilise alors des skip-connection pour éviter ce genre de problème. Dans cette architecture, l'information de la couche initiale est donnée aux couches profondes en faisant simplement une addition de matrice (la sortie de la couche précédente est ajoutée à l'entrée de la couche suivante).

Nous avons utilisé une technique de transfert learning avec l'architecture ResNet50.

Dans un premier temps expliquons ce qu'est le transfert learning.

La méthode de Transfert Learning consiste à prendre un modèle pré-entrainé sur une première tâche précise, et le ré-entrainer à faire une deuxième tâche similaire mais différente sur un nouveau dataset. Nous devons cependant effectuer quelques légères modifications sur le modèle pour pouvoir adapter l'architecture à la nouvelle tâche (fine tuning). Cela nous permet d'accélérer le processus d'apprentissage et d'améliorer la généralisation, la robustesse du modèle et le temps de calcul.

Nous avons téléchargé sur torch.hub le modèle resnet50 avec les poids pré calculés sur le dataset Imagenet. Ce modèle a obtenu 80% d'accuracy sur ce dataset.

Pour adapter le modèle à notre tâche de classification nous avons changé la dernière couche pour que nous ayons 4 classes prédite en sortie.

Nous avons appliqué aussi toutes les transformations énoncées dans la partie 3 pour réaliser de l'augmentation de données.

Nous avons repris l'entraînement du modèle pré-entraîné sur notre dataset avec les hyperparamètres suivants : Voici la loss obtenue sur les données de d'entraînement avec cette architecture :
- 15 epochs.
- CrossEntropy comme fonction de coût.
- Algorithme SGD avec moment et pas de 0.001.
- Batch de 64 pour l'entraînement et 1 pour le test.

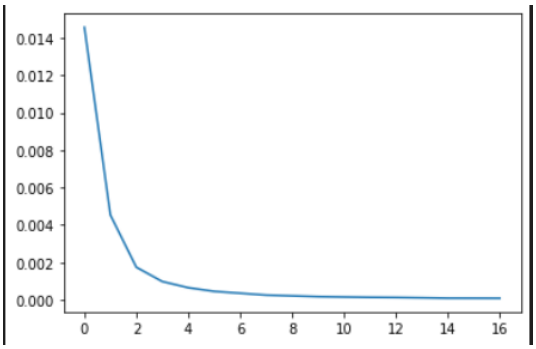

FIGURE 3 – Loss ResNet 50 fine-tuning

Nous avons obtenu avec cette architecture une accuracy de 97.5%.

Finalement, nous avons testé une dernière architecture qui est WideResNet. Cette architecture est une variante de ResNet où nous diminuons la profondeur du réseau mais nous augmentons la largeur. WideResNet utilise des couches plus larges avec un plus grand nombre de filtres. Cela permet de capturer des informations plus riches et complexes. Le temps d'exécution est alors un peu plus long pour cette architecture que pour celle de ResNet. Au total il possède 101 couches.

Nous avons là aussi fait du finetuning en prenant les hyperparamètres suivants :

- 3 epochs.
- CrossEntropy comme fonction de coût.
- Algorithme SGD avec moment et pas de 0.001.
- Batch de 64 pour l'entraînement et 1 pour le test.

Avec ce modèle nous obtenons une accuracy de 98.5%
En conclusion, nous voyons qu'en utilisant les modèles pré-entraîné dès la première epoch nous avons une erreur très faible et ils nous permettent d'avoir les meilleurs résultats en terme d'accuracy.

5 Machine Learning et AI safety

Est-ce que je peux faire confiance à mon modèle ?

Aujourd'hui, l'utilisation de l'Intelligence Artificielle fait partie intégrante de notre vie, et on retrouve des modèles d'IA dans plusieurs secteurs. Cela soulève une préoccupation majeure quant à la confiance que l'on peut accorder à ces modèles, ce qui nous pousse, en tant que créateurs de ces modèles à nous poser la question suivante : Peut-on faire confiance à notre modèle ?

Parmi les facteurs clés qui nous permettent de répondre à cette question, la robustesse occupe une place centrale. La robustesse d'un modèle d'IA est définie par sa capacité à maintenir un comportement et des performances fiables, stables et cohérentes dans toutes les situations possibles. En s'assurant de la robustesse de notre modèle, on renforce notre confiance et celle de ses utilisateurs, tout en montrant notre engagement envers la responsabilité et l'éthique de l'IA.

Pour évaluer la robustesse de notre modèle, on commence par une détection d'anomalies dans notre base d'apprentissage, car un apprentissage du modèle sur des données atypiques risque d'augmenter l'erreur. Ces anomalies peuvent être le résultat d'erreurs de mesures, manque de données ou même présence de bruits. C'est pourquoi, il est essentiel de traiter ces données afin d'éviter que le modèle ne soit influencé par des données non représentatives ou biaisées. Pour cela, il existe différentes méthodes statistiques et d'apprentissage automatique (Kmeans, DBSCAN, ..) qui peuvent être utilisées pour identifier et traiter ces valeurs. Cependant, ces anomalies peuvent également exister dans la vraie vie, notre modèle risque toujours de tomber sur des situations inattendues et irrégulière. On doit donc penser à intégrer la détection d'anomalies en exploitation, ce qui permet d'améliorer la robustesse de notre modèle en l'empêchant de se limiter que sur les décisions des situations similaires à celles de l'apprentissage (overfitting).

En plus des anomalies qu'on peut détecter pour améliorer la robustesse, le choix de la fonction de coût joue également un rôle important dans cette étape. Une fonction de coût permet de définir les écarts entre les prédictions du modèle et les valeurs réelles pendant la phase d'apprentissage, et quand il s'agit de domaines dont les applications sont à haut risques (aérospatial, santé ..), il est très important de choisir une fonction de perte qui donne un poids plus élevé aux erreurs qui peuvent avoir des conséquences graves. Elle doit être adaptée aux besoins de l'application du modèle.

En conclusion, la confiance qu'on peut accorder à nos modèles d'IA repose en grande partie sur leur robustesse. L'évaluation de cette dernière, repose principalement sur la détection d'anomalies, ainsi que le choix de la fonction perte. Cependant, la robustesse du modèle à elle seule n'est pas suffisante, la résilience, le biais dans le dataset, et l'explicabilité du modèle jouent tous un rôle important pour définir le degré de fiabilité du modèle. En revanche, la réponse à la question posée au début varie selon l'application de notre modèle, mais le plus important est de comprendre comment le modèle prend ses décisions, être transparent par rapport à ses limites et de pouvoir expliquer ses résultats de manière compréhensible.