Fine-Tuning du Modèle Depth Anything avec LoRA

## Introduction

L’objectif de ce devoir est d’apprendre la technique du fine-tuning utilisant LoRA sur le modèle Depth Anything. Pour répondre à cet objectif, il faut comprendre 3 choses : ce qu’est le fine-tuning, le modèle Depth Anything et LoRA.

Le fine-tuning consiste à spécialiser un modèle sur une tâche précise. Plus précisément, les paramètres d’un modèle initial (réalisant une tâche d’ordre généraliste) seront mis à jour afin de réaliser des tâches spécifiques ou d’en ajuster son comportement. Par exemple, le fine-tuning est utilisé dans les grands modèles de langages pour pouvoir converser sur des sujets spécialisés comme le droit, la médecine etc.

Ce principe de fine-tuning sera appliqué au modèle [Depth Anything](https://github.com/DepthAnything/Depth-Anything-V2/tree/main). C’est un modèle avancé pour déterminer la distance relative des objets à partir d'une seule image. Développée par une équipe de chercheurs, cette approche vise à fournir des estimations de profondeur robustes pour n'importe quelle image, quelles que soient les conditions.

[LoRA](https://github.com/microsoft/LoRA) (Low Rank Adaptation) est une technique utilisée principalement pour adapter ou affiner des modèles d'apprentissage automatique pré-entraînés. Le principe global est de sélectionner des paramètres qui vont être gelés (« pretrained Weights » W) et des variables qui vont être réentraînées et compressées à un rang r. Cette méthode réduit considérablement les besoins en ressources computationnelles et en mémoire lors du fine tuning d’un modèle, ce qui permet de gagner beaucoup de temps. Il est particulièrement populaire dans le domaine des modèles de langage et des réseaux neuronaux profonds.

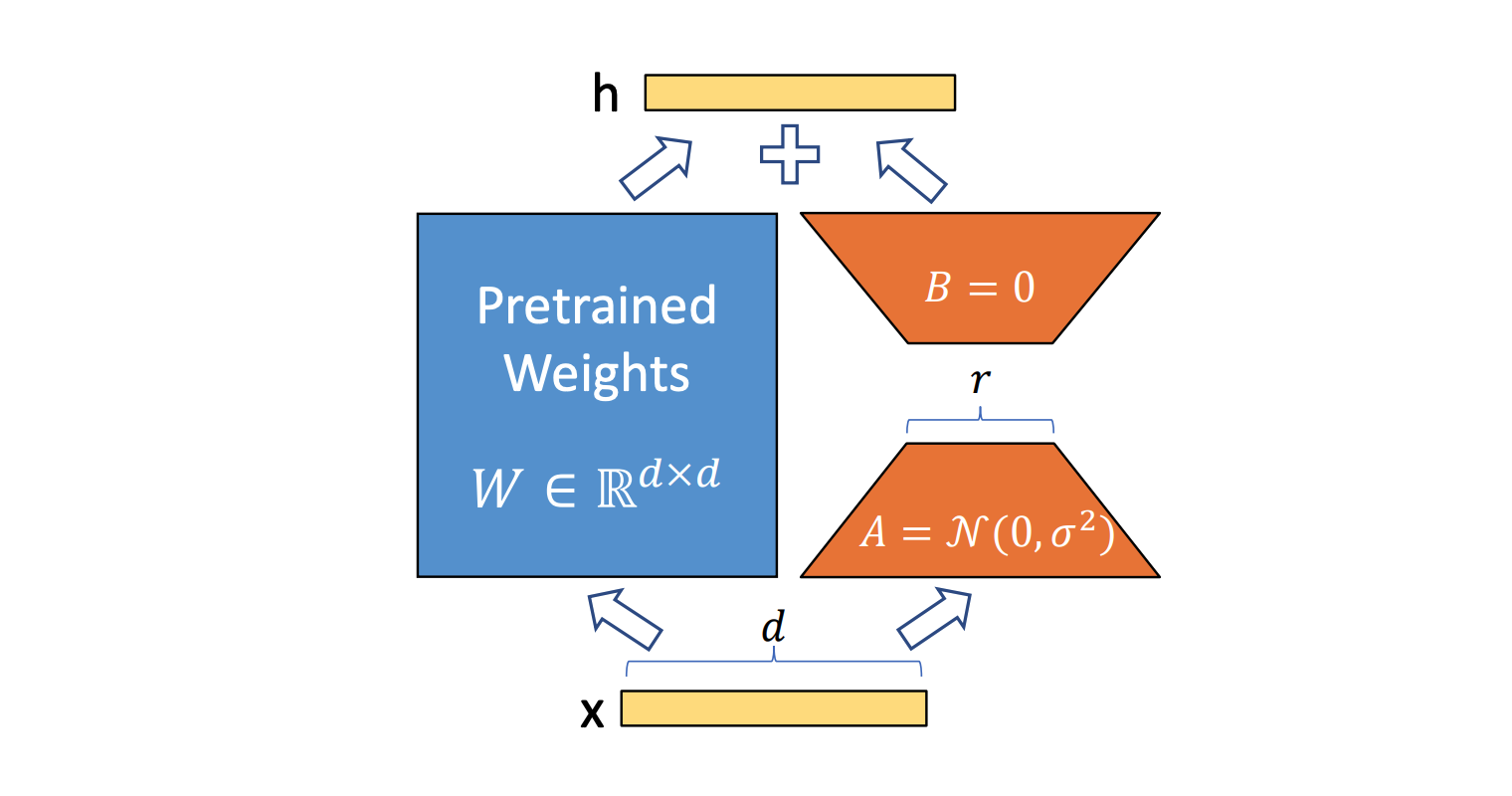


Figure 1 - Illustration de LoRA

Au début du projet, nous avons préparé l’environnement afin de pouvoir travailler en collaboration, nous avons donc créer un fichier requirements.txt avec l’ensemble des bibliothèques nécessaires. Sur le GitHub du projet [https://github.com/Arthurmgnr/LoRA-Fine-Tuning](https://github.com/Arthurmgnr/LoRA-Fine-Tuning/tree/main), nous avons également noté toute la procédure pour créer l’environnement python pour faire fonctionner le projet dans un markdown README.

## Chargement des données

Nous avons donc commencé par créer une classe DepthDataset qui va permettre de parcourir toutes les images ainsi que les nuages de points qui constituent notre dataset. Pour cela, on stocke les chemins d’accès aux différents fichiers pour éviter de stocker en mémoire les images directement et alourdir la mémoire. Puis, lorsque l’on veut accéder à une image, le dataset renvoie l’image ainsi que le nuage de points associés. Les données ont été séparées en deux : un ensemble d’entrainement (80%) et un ensemble de validation (20%).

À ce moment-là, nous nous sommes rendu compte de plusieurs problèmes qui ont nécessité quelques modifications sur la classe initiale. Ces problèmes sont détaillés ci-dessous :

1. Nous nous sommes rendu compte que le modèle demandait en input des images qui sont des multiples de 14. Cela implique que les images doivent être redimensionnées.
2. Aussi, nous avons vu que les images n’étaient pas dans le bon code couleur, ils sont initialement en BGR. Nous les avons donc converties en RGB.
3. Après analyse de la structure des données depth, il est apparu qu’il y avait trois dimensions de profondeur : par rapport à l’axe X, l’axe Y et l’axe Z. Cette structure à l’avantage de pouvoir afficher un nuage de point en 3D. Cependant, depthanything, dans son output, retourne une profondeur donnée relativement par rapport à la prise de vue. Cela correspond mathématiquement à l’axe Z. Nous avons donc choisi de ne garder dans les données depth, que cette profondeur par rapport à l’axe Z.
4. Le dernier problème rencontré est la présence de données manquantes (NaN) dans les données de profondeur. Après vérification, notre conclusion est que ces données manquantes correspondent aux parties d’ombres sur les bordures de la plupart des images. Plusieurs solutions ont été envisagées : remplacer des données par une valeur fixe ou chercher à interpoler ces valeurs par leurs plus proches voisins. Finalement, nous avons choisi de faire la méthode par interpolation. Ce choix s’explique parce que certains NaN se situent au milieu de certains pneus, en conséquence replacer par la valeur la plus lointaine rendrait les données moins fiables pour un bon apprentissage.

Une image contenant intérieur, mur, plat, chaise

Description générée automatiquement

Figure 2 - Exemple de photo du dataset avec de l'ombre

## Implémentation de LoRA

Ensuite, nous avons créé une classe LoRALayer. Cette classe applique la formule données dans l’article qui est le produit de la couche d’input par les matrices A et B.

Une fois la couche LoRA construite, il faut pouvoir l’appliquer à toutes les couches QKV du modèle DepthAnything V2. A ce stade, nous nous sommes demandé sur quelles couches il fallait appliquer LoRA. Nous avons pensé qu’il serait intéressant de l’appliquer sur des couches convolutionnelles pour se spécialiser sur des images de pneus. Cependant, par simplicité, nous avons choisi de ne l’appliquer que sur les couches QKV.

Quand toutes les couches concernées ont été remplacées, il faut geler les paramètres des couches sur lesquelles nous n’avons pas appliqué les couches LoRA. L’objectif est de ne rendre entraînable que les paramètres des couches sur lesquelles il y a du LoRA, les autres seront les mêmes que le modèle DepthAnything d’origine.

## Fine – Tuning

Maintenant que les données ont été chargées et que LoRA a été implémenté, il faut entrainer DepthAnything afin de le fine-tuner. À ce moment-là, plusieurs choix doivent être faits :

* L’optimiseur choisi est Adam
* Pour la fonction de coût, plusieurs choix étaient possibles : moyenne des erreurs quadratiques, moyenne des erreurs absolues, perte Invariante à l’Échelle Logarithmique et perte Invariante aux Changements d’Échelle et de Translation. Ici, dans ce cas, il n’y a pas vraiment de fonction coût qui soit meilleure que les autres, étant donné la complexité de la tâche à réaliser. D’autant plus que le modèle DepthAnything utilise une fonction coût propre à lui seul.

## Évaluation des performances

A présent, il nous faut évaluer le modèle fine-tuné sur l’ensemble de données de validation, ce qui nous permet d’évaluer les performances de l’adaptation. Pour ce faire, nous avons affiché plusieurs images. De gauche à droite, on affiche : l’image réelle (données), la profondeur suivant l’axe de la caméra (données), l’image des profondeurs prédites par le fine-tuning du modèle avec LoRA et l’image prédite avec le modèle DepthAnythingV2 original.

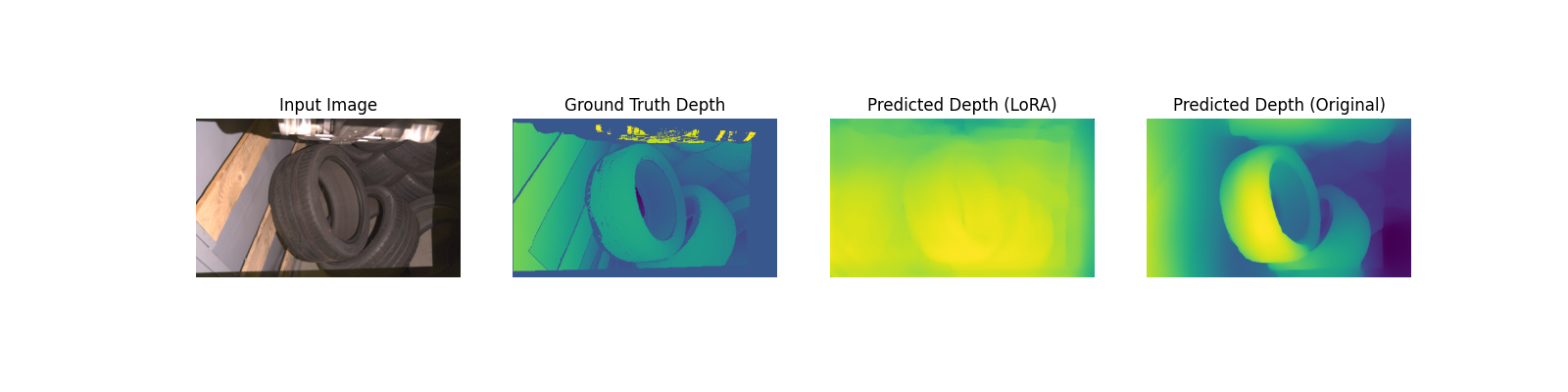


Figure 3 - Exemple d'image d'évaluation du fine-tuning de DepthAnythingV2 avec LoRA après le 2ème epoch

En plus du diagnostic visuel, on regarde l’évolution de la fonction de perte. Cela nous permet d’identifier d’éventuels problèmes de sur-apprentissage ou de sous-apprentissage. Typiquement dans le cas ci-dessous avec 300 epochs d’entraînement, sur 40 photos, LoRA appliqué sur les couches QKV, convolution et linéaires pour des images réduites à 14x10, la loss n’a pas encore convergé : ce qui indique un sous-apprentissage.

Enfin, pour continuer d’évaluer le diagnostic du modèle, nous avons mesuré le temps que met le programme à s’entraîner dans des conditions particulières résumées dans ce tableau[[1]](#footnote-1) :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Couches LoRA | Taille des images | Nb image entrainement | Nb Epochs | Temps (s) |
| QKV + Linear | 84, 140 | 11 | 25 | 124 |
|  |  |  |  |  |
|  |  |  |  |  |

Il faut aussi rajouter les mesures F1 score etc. sur la partie évaluation

Et aussi ajouter une comparaison avec le vrai modele depth anything

## Conclusion

Finalement,

1. Specs : 16CPU de 2GHz à 3.5GHz à 8 cœurs [↑](#footnote-ref-1)