Projet de Machine Learning

Classification des Types de Couverture Terrestre avec PyTorch

Sciences du Numérique - HPC Big Data- troisième année 2023-2024

Brenda TONLEU
Fatima EL GHAZI

TABLE DES MATIÈRES

0.1 INTRODUCTION :

Dans le domaine de la classification d'images, l'utilisation des réseaux neuronaux convolutionnels (CNN) s'est révélée être une approche essentielle. Cette méthodologie offre une capacité efficace d'extraction de caractéristiques à partir d'images, permettant de discerner des motifs complexes au sein des ensembles de données. Contrairement aux réseaux neuronaux artificiels (ANN) traditionnels, qui peuvent devenir computationnellement lourds avec des paramètres entraînables étendus, les CNN utilisent des filtres pour explorer la localité spatiale des images, en imposant un schéma de connectivité locale entre les neurones.

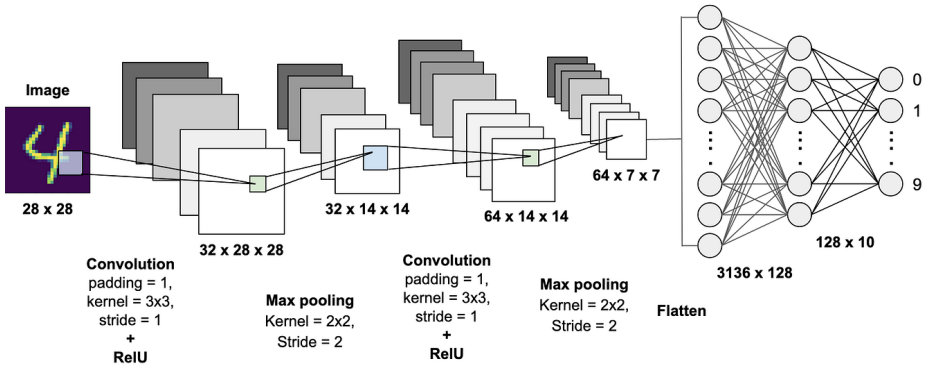

Figure 1: Exemple d'un cnn pour réaliser une classification.

Notre projet se concentre sur la classification de divers types de couverture terrestre, tels que la 'Forêt', la 'Rivière', l''Autoroute', et d'autres, en utilisant un ensemble de données d'images étiquetées "Apprentissage supervisé".

Notre évaluation comprend différents modèles : un simple réseau neuronal convolutionnel, ResNet ainsi que ces variants et EfficientNet-B4. Notamment, nos résultats mettent en évidence les performances supérieures de resnet18 personnalisé, soulignant son efficacité dans l'amélioration de la précision de la classification d'images pour des applications pratiques.

0.2 PRÉTRAITEMENT DES DONNÉES:

La base d'entraînement se compose de 20 000 images en mode 'RGB' avec une résolution de 64x64, et la base de test contient 7 000 images du même format. En résumé, voici les modifications que nous avons apportées à l'ensemble des images afin de les adapter à notre CNN :

- Rééchelonner les valeurs des pixels : Transformer les valeurs des pixels pour qu'elles soient dans la plage [0, 1] en divisant les images par 255.

```
X_train, X_test = X_train/255, X_test/255
```

- Créer un transformateur : Mettre en place un transformateur pour convertir les données en tenseurs PyTorch et normaliser les images.

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std)
])
```

- Diviser le jeu de données : Séparer le jeu de données en ensembles d'entraînement et de validation, en allouant 20 % pour la validation.

```
X_train, X_valid, y_train, y_valid =
 train_test_split(X_train,  y_train, test_size=0.2, random_state=42)
```

- Appliquer le transformateur : Mettre en œuvre le transformateur pour prétraiter les ensembles de données.

```
X_train_tensor = torch.stack([transform(img) for img in X_train]).float()
X_valid_tensor = torch.stack([transform(img) for img in X_valid]).float()
X_test_tensor = torch.stack([transform(img) for img in X_test]).float()
```

- Convertir les étiquettes de l'ensemble d'entrainement et validation en tensors

```
y_train_tensor = torch.from_numpy(y_train)
y_valid_tensor = torch.from_numpy(y_valid)
```

- Création des DataLoaders, éléments clés de PyTorch, qui facilitent le chargement des données pour l'entraînement et la validation.

0.3 MODÈLES UTILISÉS

0.3.1 Simple réseau neuronal convolutionnel :

Pour l'entrée de cette architecture, une image de résolution 64 * 64 en format RGB est utilisée. L'image traverse une couche de convolution initiale avec un filtre de 7x7, une stride de 2x2, et un padding de 3x3. Cette première couche est suivie d'une normalisation par lot (Batch Normalization) et d'une opération de max pooling avec un noyau de 3x3, une stride de 2, et un padding de 1. Ce processus permet d'extraire des caractéristiques de plus haut niveau à partir de l'image initiale.

Ensuite, plusieurs autres couches de convolution, chacune suivie de normalisation par lot et de pooling, sont utilisées pour capturer des motifs et des informations complexes dans les données d'entrée.

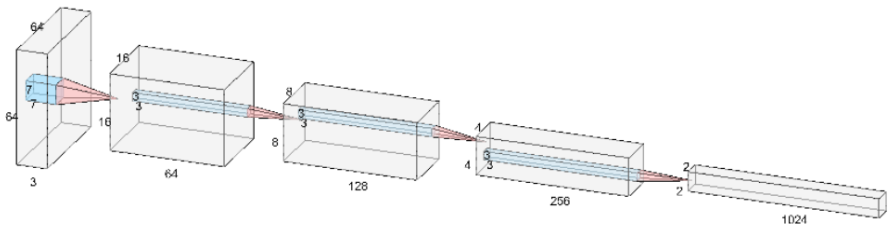

Figure 2: Passage d'une représentation à l'autre à l'aide de convolutional layers

Enfin, une couche de pooling globale moyenne (Global Average Pooling) est appliquée pour réduire les dimensions spatiales à 1x1, créant ainsi une représentation compacte et informative.

Cette représentation est ensuite aplatie et connectée à des couches entièrement connectées, avec des opérations de ReLU et de dropout pour la classification finale des 10 classes de sortie.

Voici le learning curve de ce modèle sur 30 epochs et un batch size de 128:

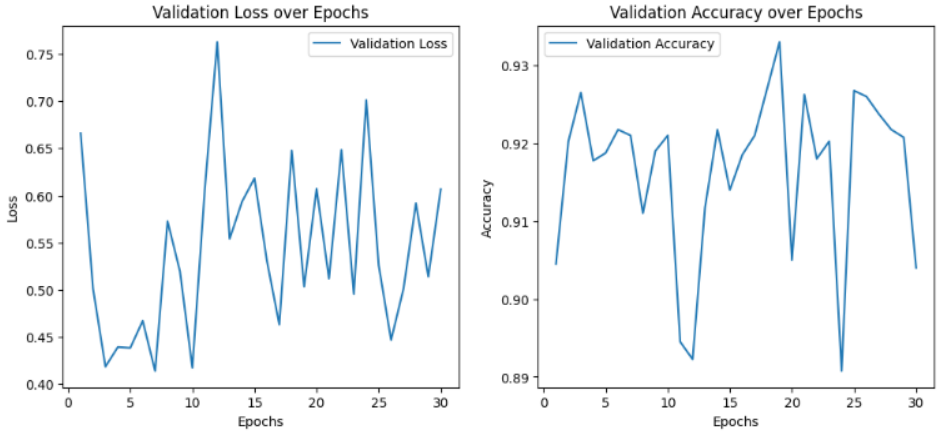

Figure 3: Précision et perte du réseau neuronal convolutif simple.

-	Modèle CNN simple:
min loss_validation	0.2860599565319717
max accuracy	0.9205

0.3.2 Resnet

L'architecture des modèles resnet repose sur l'utilisation de blocs résiduels qui facilitent l'entraînement de réseaux de neurones très profonds en évitant le problème de la disparition du gradient.

Concernant l'approche resnet; nous avons effectué des mesures de performance sur resnet18; resnet34 et resnet50.

Bien que la différence de précision et de perte ne soit pas exagérée entre les différentes architectures de resnet; resnet50 semblait plus adapté (Validation Loss: 0.303, Accuracy: 92.30%); et resnet18 en second.

Par la suite nous avons personnalisé le modèle resnet en y rajoutant des couches de dropout et des couches entièrement connectés.

```
super(ResNet, self).\_\_init\_\_()
        self.model = models.resnet18(pretrained=True)
        self.model.fc = nn.Linear(self.model.fc.in\_features, 256)
        self.fc1 = nn.Linear(256, 1024)
        self.dropout1 = nn.Dropout(0.2)
        self.fc3 = nn.Linear(1024, 512)
        self.dropout2 = nn.Dropout(0.2)
        self.fc2 = nn.Linear(512, num\_classes)
```

Statistiques d'Entraînement

L'entraînement a été effectué pendant 30 épochs en relevant à chaque fois la loss validation et la précision pendant l'entraînement.

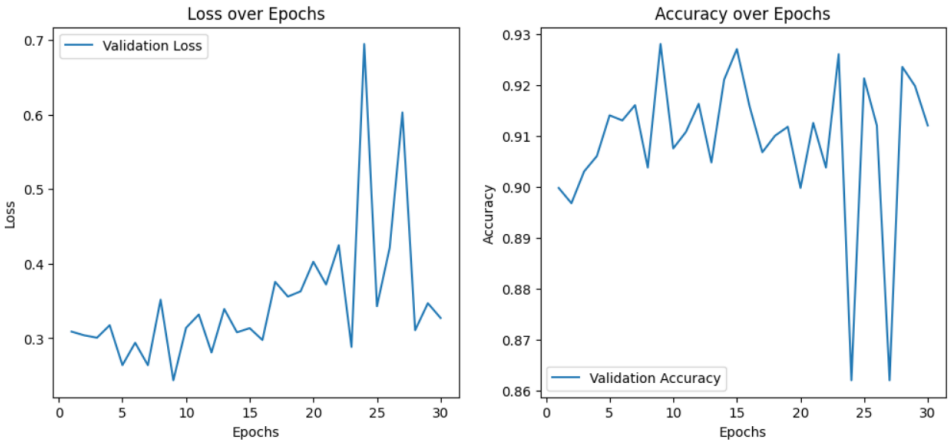

Figure 4: Précision et perte modèle Resnet18.

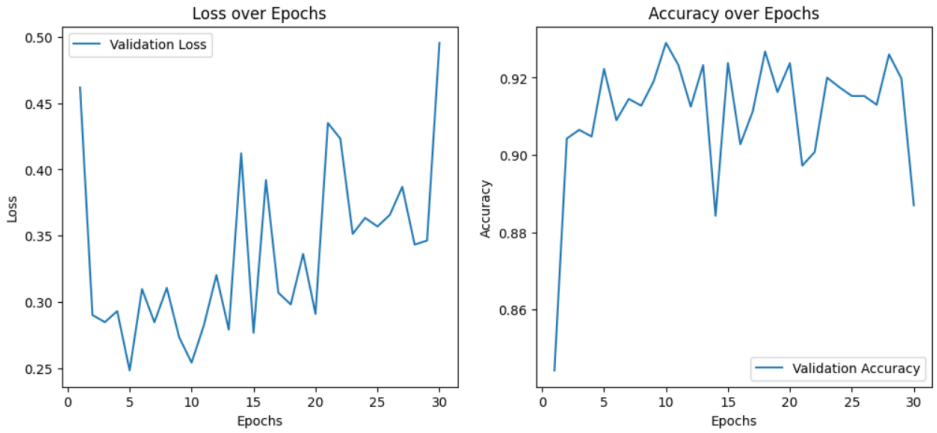

Figure 5: Précision et perte modèle Resnet50.

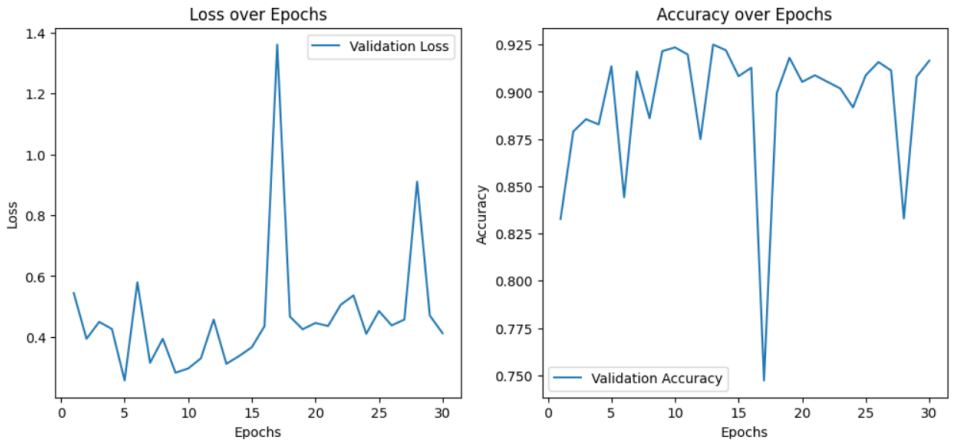

Figure 6: Précision et perte modèle Resnet18 custom.

Observations

Nous observons pour les différentes architectures resnet:

-	Resnet18	Resnet50	Resnet18_custom
min loss_validation	0.2540852659475058 6	0.28819373600184917	0.2199514047242701
max accuracy	0.929	0.926	0.93

L'approche du modèle pré-entraîné Resnet est légèrement meilleure comparée au simple CNN.

0.3.3 EfficientNet-B4

EfficientNet est un modèle purement convolutionnel (ConvNet) adapté aux appareils mobiles, qui propose une nouvelle méthode de mise à l'échelle uniforme de toutes les dimensions de la profondeur, de la largeur et de la résolution en utilisant un coefficient composé simple mais très efficace.

EfficientNet-B4 est une variante spécifique du modèle EfficientNet, caractérisée par une configuration particulière de ses paramètres de profondeur, de largeur et de résolution.

Statistiques d'Entraînement

L'entraînement a été effectué pendant 30 époch en relevant à chaque fois la loss validation et la précision pendant l'entraînement.

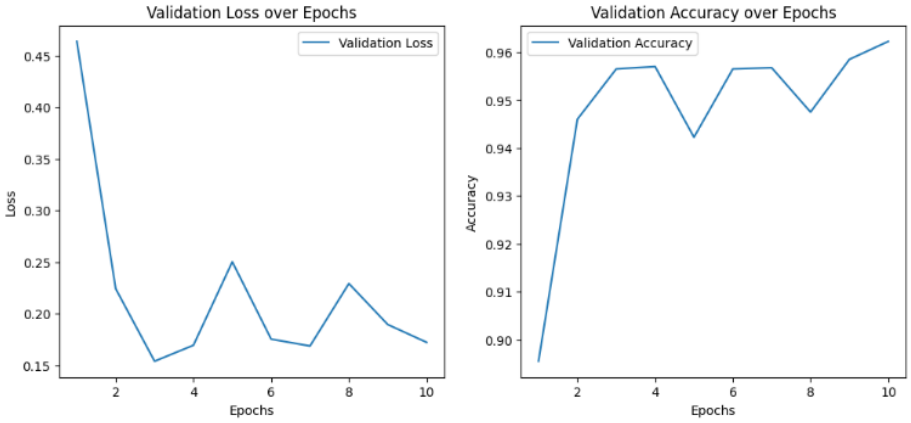

Figure 7: Précision et perte modèle efficientnet-B4.

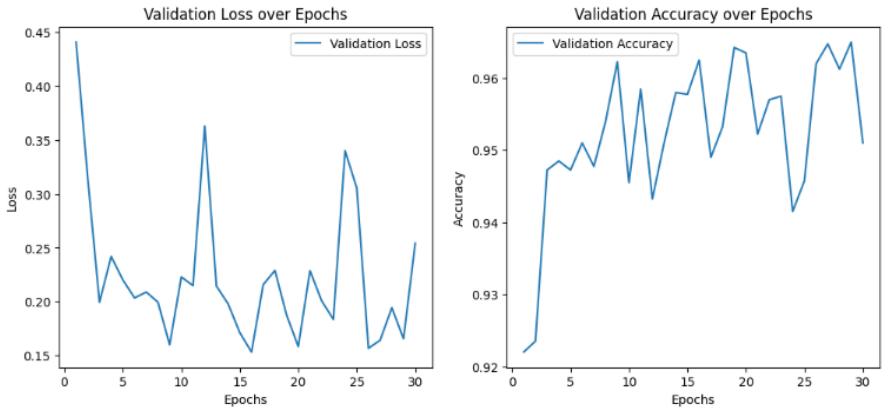

Figure 8: Précision et perte modèle efficientnet-B4

Observations

On observe pour le modèle efficientNet-B4 que:

-	EfficientNet-B4
min loss_validation	0.15324834725470282
max accuracy	0.965

0.4 CONCLUSION

En conclusion, ce projet de classification d'images satellites a constitué une expérience enrichissante et formatrice. Tout d'abord, il nous a permis d'approfondir notre compréhension des architectures de modèles, en commençant par un réseau de neuronnes convolutionel simple jusqu'aux variantes avancées telles que ResNet18, ResNet50, et même une version personnalisée de ResNet18.

L'exploration minutieuse des performances de ces modèles a non seulement renforcé nos compétences en matière de conception de réseaux neuronaux, mais elle a également souligné l'importance de la personnalisation de l'architecture pour des résultats optimaux.

L'intégration d'EfficientNet-B4 a constitué une étape significative, mettant en lumière l'importance de considérer des architectures pré-entraînées pour des performances optimales dans des projets spécifiques (avec sa précision exceptionnelle de 96%,)
.

Enfin, le processus d'évaluation et de comparaison des modèles nous a apporté une perspective approfondie sur les compromis entre précision, temps d'entraînement, et complexité du modèle. La conclusion que EfficientNet-B4 se positionne comme le choix optimal en termes de précision a été une découverte clé, tout en soulignant la nécessité de prendre en compte des aspects pratiques dans le choix final du modèle.

Dans l'ensemble, ce projet a élargi notre expertise en apprentissage profond et nous a offert des compétences précieuses dans la prise de décision concernant le choix des modèles pour des tâches de classification complexes.