Université Lyon 2

Master SISE

Projet Deep Learning:
Damage Detection Satellite Imagery pytorch

Responsable : Julien Ah-Pine

mots clés : - pytorch - Convolutional Neural Network - Transfer Learning - Image classification

GIROUD Antoine, HAIDARA Fatimetou, SCHULTZ Martin

10 février 2023

Table des matières

1 Introduction

1.1 Introduction Générale

Les réseaux de neurones convolutifs (CNN) sont un type de réseau de neurones profonds qui a été largement utilisé pour le traitement et la classification d'images. Les réseaux de neurones convolutifs sont conçus pour capturer les relations spatiales entre les pixels d'une image, ce qui les rend particulièrement utiles pour les tâches d'analyse d'images telles que la reconnaissance d'objets, la classification d'images et la segmentation d'images. Dans le cadre de ce projet nous allons les utiliser pour réaliser de la classification d'images. Plus particulièrement, sur des images satellites représentant des bâtiments endommagés, ou non, dans la région de Houston, suite à l'ouragan Harvey ayant eu lieu en 2017. Il est en effet primordial de connaître la situation sur place lorsqu'une telle catastrophe a lieu. L'exploration d'un modèle de deep learning permettant de repérer les bâtiments endommagés à partir d'images satellites est donc tout à fait pertinent.

Ce projet a de même pour but de nous permettre de mettre en pratique nos connaissances en deep learning dans un projet concret en utilisant la librairie pytorch. Nous allons pour cela donc un premier temps étudier un article de Cao et Choe traitant de ces images satellites, avant d'expliquer plus en détails le fonctionnement des CNN et du transfer learning à l'aide de CNN pré-entraînés. Nous présenterons ensuite les modèles que nous avons testés en lien avec cet article, et discuterons des résultats obtenus. Enfin, nous apporterons une conclusion sur notre avis quand à la réalisation de ce projet dans le cadre de notre formation.

1.2 Présentation de l'étude de cas

Nous allons à présent vous présenter plus en détail l'article de Cao, Q. D. et Choe, Y. 'Building damage annotation on post-hurricane satellite imagery based on convolutional neural networks.' de 2020. Comme expliqué précédemment, cet article présente une étude, réalisée sur l'analyse d'images satellites de bâtiments, ayant pour objectif de détecter si un bâtiment a subi des dégâts après une catastrophe climatique. Cette étude vise à proposer une solution qui améliore le recensement des bâtiments endommagés, d'habitude réalisée au sol, en annotant grâce à un algorithme ces bâtiments, pour pouvoir estimer au mieux les dégâts causés. L'article se base sur les données de l'ouragan Harvey en 2017 dans la région du Grand Houston, en utilisant les images satellites. Pour réaliser cette analyse, les auteurs utilisent des méthodes d'apprentissage profond, et plus particulièrement des CNN. Ils essaient notamment d'implémenter leurs propres réseaux, mais utilisent aussi des CNN pré-entrainés afin de réaliser du transfer learning.

1.2.1 Les données

Des images satellites de la région de Houston ont été capturées par des capteurs optiques avec une résolution sub-métrique et ont subi un pré-traitement comme l'orthorectification et la compensation atmosphérique. Les données brutes contiennent 4 000 bandes d'images prises sur plusieurs jours, dont certaines se chevauchent, avec environ 400 millions de pixels chacunes. Certaines images sont possèdent une couverture nuageuse, et celles de faible qualité ont été jetées.

1.2.2 Annotation des dommages

Le cadre méthodologique décrit commence par la saisie de données brutes et les traite pour créer une sortie d'annotation des dommages. La première étape consiste à traiter les données brutes en données prêtes pour l'entraînement en recadrant les fenêtres centrées sur les coordonnées du bâtiment obtenues à partir de données publiques (par exemple, OpenStreetMap). Les coordonnées du bâtiment sont associées aux étiquettes de dommages des bénévoles de Tomnod et sont utilisées comme vérité sur le terrain pour l'étiquette positive, " Bâtiment inondé ou endommagé ". Les fenêtres des images capturées avant l'ouragan sont utilisées pour créer des échantillons de données négatifs étiquetés "Bâtiment non endommagé". La taille optimale de la fenêtre est un hyperparamètre(400x400, 128x128, 64x64, et 32x32) de réglage qui dépend de la résolution de l'image et de la taille de l'empreinte du bâtiment. Les images recadrées sont filtrées pour la qualité, divisées en ensembles d'entraînement, de validation et de test, et transmises à un réseau neuronal convolutif pour l'annotation des dommages. La précision de la validation est surveillée pour ajuster les hyperparamètres.

Pour éviter de gonfler la précision des prédictions en raison d'images dupliquées de qualité différente, le processus de génération de données garanti que chaque coordonnée de bâtiment est associée à une image unique et de bonne qualité dans le jeu de données final via un processus semi-automatisé. Ce processus implique de supprimer automatiquement les images totalement noircies et de conserver la première image non noire rencontrée, suivie d'un filtrage manuel pour éliminer les images partiellement noires ou couvertes de nuages.

1.3 Présentation ses CNN

Cao et Choe expliquent que les réseaux de neurones convolutifs permettent généralement d'obtenir des résultats nettement supérieurs que les autres algorithmes pour des tâches de vision par ordinateur, telles que la catégorisation d'objets ou la classification d'images. Il s'agit d'un réseau de type feed-forward, c'est-à-dire que les réseaux sont tous orienté de l'entrée vers la sortie.

Un CNN possède trois composantes :
- la couche convolutive (convolutional layer) qui est utilisée pour extraire des informations caractéristiques importantes des entrées d'image. Elle utilise des

filtres qui se déplacent sur l'image en effectuant des calculs pour produire des fonctionnalités locales. Par exemple, un filtre convolutif 3x3 prendra un produit scalaire de 9 paramètres de poids avec 9 pixels (patch 3x3) de l'entrée, et la valeur résultante est transformée par une fonction d'activation pour devenir une valeur de neurone dans la couche suivante
- la couche de sous-échantillonnage (sub-sampling layer) qui est utilisée pour réduire la dimension de l'image. Elle regroupe les entrées en sous-régions et ne conserve qu'une seule valeur pour chaque sous-région, généralement la valeur maximale (on parle de max-pooling), ou alors la moyenne locale. Cela aide à réduire la complexité du modèle tout en conservant les caractéristiques importantes de l'image. Cette étape de sous échantillonnage réduit la matrice de caractéristiques d'entrée à la moitié de son nombre de colonnes et de lignes.
- la couche entièrement connectée (fully connected layer) qui est utilisée pour réaliser la prédiction finale. Pour se faire, le réseau va aplatir la dernière pile de matrices en un vecteur, et le faire passer par une séquence de couches entièrement connectées. Le neurone de sortie de chaque couche suivante est un produit scalaire entre le vecteur de caractéristique et un vecteur de poids, transformé par une fonction d'activation. Celle-ci est généralement utilisée pour introduire une non-linéarité dans les couches de neurones, permettant de modéliser des relations complexes entre les entrées et les sorties. Les plus courantes sont les sigmoïdes, les ReLU (rectified linear unit) et les leaky ReLU.

1.4 Exposition au principe de base du transfer learning et utilisation de l'architexture préentrainée VGG-16

Une pratique courante dans la sélection de l'architecture du CNN est le transfer learning, qui consiste à commencer par une architecture connue (telle que VGG-16) et à l'affiner pour l'adapter à la dimension d'entrée spécifique. VGG-16 est connu pour fonctionner correctement sur le jeu de données ImageNet pour la classification des objets.

L'architecture a été construite à partir de zéro, en tenant compte des hyperparamètres appropriés et du flux d'informations à travers le réseau. Les auteurs ont surveillé le nombre de "filtres morts" dans le réseau et ont cessé de l'agrandir lorsque le nombre de filtres morts était élevé. Ils ont choisi un réseau peu profond avec quatre couches convolutives et deux couches entièrement connectées au lieu d'un réseau plus profond comme VGG-16, car des réseaux moins profonds peuvent conduire à des temps de formation plus rapides et à de meilleures performances de généralisation. Le réseau personnalisé a 3,5 millions de paramètres entraînables, tandis que VGG-16 en a 15 millions, ce qui pourrait entraîner un surajustement et une diminution des performances de généralisation. La profondeur du réseau doit dépendre de la complexité des caractéristiques à extraire de l'image. Dans ce cas, avec seulement deux classes d'intérêt, un réseau moins profond est favorisé.

1.5 Architecture proposé avec les meilleures performances

L'architecture proposée se base sur le modèle de VGG-16, en modifiant la première couche pour pouvoir prendre en paramètre les images de 150 pixels. L'architecture comprends quatre couches convolutives et deux couches entièrement connectées. C'est le parti pris qui a été choisi pour favoriser le temps de calcul de cette méthode tout en maximisant les performances avec une précision de 88% sur les données de test déséquilibré.

Après le nettoyage des données : le dataset contient 14 284 échantillons positifs et 7209 négatifs. 5000 de chaque classe sont dans le training set, 1000 dans le validation set et le reste dans le test set.

Etant donné la grande quantité de données, les chercheurs étudient des combinaisons sélectionnées d'hyper-paramètres plusieurs que de tous les régler sur une grille de recherche complète. Ils ont essayé d'implémenter une régression logistique sur les données featurisées afin de comparer aux couches entièrement connectées. La régression logistique est sans surprise moins performante, mais atteint tout de même 90%. Cela montre que la featurisation de l'image par le réseau fonctionne assez bien.

Les chercheurs ont aussi essayé de changer la valeur de la fonction d'activation ReLU, qui était de 0, en utilisant un α de 0,1. Cependant, ce changement n'a pas amélioré l'accuracy. Ils ont de même effectué une augmentation des données d'entraînement afin de faire face au sur-apprentissage, ainsi qu'un dropout de 50% et une régularisation L2 avec $\lambda = 10^{-6}$ sur la couche entièrement connectée.

1.6 Exposition et résultats des différentes approches de l'article

Une dizaine de modèles différents ont été testé. Le meilleur possède une accuracy de 98,06%, avec un F1 Score de 0,9723. Il s'agit d'un CNN avec une data augmentation et un drop out de 50% avec un optimiser Adam. Les modèles avec un CNN ont l'air de réaliser globalement les meilleurs résultats. Néanmoins, les modèles avec du transfer learning possédant une architecture VGG-16 et une fonction d'activation Leaky ReLU possèdent aussi d'excellent résultats. Les algorithmes de machine learning tels que la regression logistique ou les SVM ont des résultats assez inférieurs.

2 Méthodes de transfer learning

2.1 Présentation Générale

Pour expliquer simplement le transfer learning on pourrait décrire cette technique comme une réutilisation des connaissances acquises et de l'apprentissage

sur un cas pour acquérir de nouvelles connaissances dans un autre domaine. Elle permets de récupérer les poids d'un modèle de deep learning, pour initier un nouvelle modèle souvent de machine learning ou bien de deep learning pour réaliser une autre taches.

Le transfer learning peut être très utile dans les cas où la quantité de données disponibles pour la tâche spécifique est limitée, car il permet de profiter des connaissances acquises dans un domaine similaire pour en acquérir de nouvelles dans un autre sous-domaine.

Par exemple, si nous voulions construire un modèle de reconnaissance de la langue pour des phrases en français. Nous pourrions utiliser le transfer learning pour initier notre modèle à partir d'un modèle pré-entraîné sur une tâche similaire, comme la reconnaissance de la langue anglaise.

2.2 Transfer Learning A

Le transfer learning de type A, également appelé transfer learning d'adaptation fine, est une méthode de transfer learning qui implique de prendre un modèle pré-entraîné sur une tâche similaire mais s'adapter aux données de la nouvelle tâche en ajustant les poids des couches supérieures. Car les couches supérieures du modèle sont considérées comme étant plus spécifiques à la tâche antérieur et doivent être ajustées pour s'adapter à la nouvelle tâche.

2.3 Transfer Learning B

Le transfer learning de type B, également appelé transfer learning de caractéristiques, dans cette méthode le modèle pré-entraîné est utilisé comme une sorte de d'extraction des poids pour produire des représentations robustes des données. les poids extraites sous forme de vecteurs, sont ensuite utilisées comme entrées pour un modèle plus petit, spécifique à la nouvelle tâche. Il est plus utile lorsque la nouvelle tâche est très différente de la tâche d'origine, car cette extraction récupère des poids génériques pour construire un modèle plus petit.

3 Présentation des différentes expériences/modèles testés

Après s'être penchés sur la lecture et la compréhension de l'article de Cao et Choe, nous avons essayé de comprendre leur code en détail. Pour cela, nous avons notamment ré-implémenté en pytorch certains de leurs modèles réalisés en Keras.

3.1 CNN

Nous avons commencé par réaliser leur premier CNN. Voici l'architecture qu'ils ont utilisé pour le produire qu'ils décrivent dans leur article :

Type de couche 1	Forme de sortie	Nombre de paramètres entraînables
Input	3@(150x150)	0
2-D Convolutional 32@(3x3)	32@(148x148)	896
2-D Max pooling (2x2)	32@(74x74	0
2-D Convolutional 64@(3x3)	64@(72x72)	18 496
2-D Max pooling (2x2)	64@(36x36)	0
2-D Convolutional 128@(3x3)	128@(34x34)	73 856
2-D Max pooling (2x2)	128@(17x17)	0
2-D Convolutional 128@(3x3)	128@(15x15)	147 584
2-D Max pooling (2x2)	128@(7x7)	0
Flattening	1x6272	0
Fully connected layer	1x512	3 211 776
Fully connected layer	1x1	513

2-D Convolutional correspond aux couches convolutives, 2-D Max pooling correspond aux couches de sous-échantillonnage, fully connected layer correspond aux couches entièrement connectées.Avant d'effectuer les couches entièrement connecté, les chercheurs ont rajouté un flatten qui permet de convertir les dimensions en tenseur.

Cela nous a fortement aidé à comprendre l'architecture de celui-ci, ce qui nous a permis par la suite de réaliser de même leur meilleur modèle, c'est-à-dire ce même CNN, avec l'ajout d'une dropout, d'une augmentation des données, et utilisé un optimiseur Adam.

Le dropout est une technique de régularisation permettant d'éviter l'overfitting. Pour cela, il ignore aléatoirement certains neurones lors de l'entraînement du modèle, ce qui rend ce dernier moins complexe et plus robuste en obligeant les différentes unités de neurones à travailler de manière indépendante.

L'augmentation des données est une technique permettant d'améliorer la performance d'un modèle, et peut aussi participer à résoudre des problèmes d'overfitting. Cette technique consiste à générer de nouvelles données en effectuant des transformations aléatoires sur les données originales, telles que des rotations, des translations, des changement de luminosité ou de contraste, etc. Cela augmente la quantité des données et améliore donc la robustesse du modèle face aux variations dans les données d'entraînement.

Le rôle d'un optimiseur dans un réseau de neurones est de minimiser la fonction de perte en ajustant les poids et les biais du réseau.L'optimiseur Adam se

distingue des autres en utilisant une technique d'estimation adaptative permettant d'ajuster les taux d'apprentissage pour chaque paramètre dans le modèle. Cela lui permet de s'adapter aux variations de la distribution des gradients sur la durée de l'entraînement, et donc de converger plus rapidement.

Ensuite, nous avons essayé de réaliser notre propre CNN afin de voir s'il était possible d'améliorer l'accuracy obtenue par les chercheurs. Pour cela, nous avons notamment ajouté un padding dans les couches convolutives afin d'ajouter une bordure à l'image, ce qui aide le noyau à traiter l'image, et permet une analyse plus précise. Nous avons de même réalisé une augmentation des données en ajoutant du floutage gaussien, ce qui permet d'atténuer le bruit présent dans des images. Une fois ce modèle réalisé, nous l'avons effectué une seconde fois en ajoutant cette fois-ci une régularisation L2. Après cela, nous nous sommes concentré sur les méthodes de transfer learning.

3.2 Transfer Learning A avec ResNet18

Pour réaliser un transfer learning de type A, nous avons d'abord choisi de pré- entraîner un modèle sur ResNet18. En extrayant ses features pour les appliquer à un nouveau modèle, cette fois-ci un modèle de machine learning le random forest et puis ensuite un modèle AdaBoost.

Pour cela on va encapsuler les données à partir desquelles les caractéristiques doivent être extraites dans un objet modèle de réseau de neurones avec ResNet18.

Ensuite, on va boucler sur les lots de données. Pour chaque lot, le modèle passe en mode d'évaluation, ce qui signifie qu'il ne tiendra pas compte de la descente de gradient. Les caractéristiques sont extraites en faisant passer le lot d'entrées à travers le modèle et en détachant les valeurs obtenues.Ensuite on va lancer l'apprentissage du random forest ou du AdaBoost avec les features extraites. Pour ensuite réaliser un prédiction sur nos données de test. Pour pouvoir enfin évaluer les nouvelles performances de ce modèles.
Cette partie, en réalisant une étape concrète, nous a clairement fait comprendre l'intérêt du transfer learning dans les problématiques actuelles de la data science.

Nous obtenons un résultat fonctionnel et plus satisfaisant même si il n'est pas meilleur qu'un CNN.

Le pseudo code suivant montre la fonction qui permet d'extraire les features du modèle pré -entrainé.

- *dataloader objet PyTorch qui contient les données à partir desquelles les caractéristiques doivent être extraites.*
- *model : un objet modèle de réseau de neurones qui sera utilisé pour extraire les caractéristiques.*

— *n_sample : un entier qui spécifie le nombre de données à extraire.*
— *Les caractéristiques sont converties en numpy array dans features*
— *Les étiquettes sont stockées labels*

Algorithm 1 Fonction d'extraction des features d'un modèle de deep learning

function EXTRACT_FEATURES($dataloader, model, n\_sample$)

  $features \leftarrow []$
  $labels \leftarrow []$
  $i \leftarrow 0$
  for $inputs\_batch, labels\_batch$ in $dataloader$ **do**
    $model.eval()$
    $features\_batch \leftarrow model(inputs\_batch).detach().numpy()$
    $features[i \times dataloader.batch\_size : (i+1) \times dataloader.batch\_size] \leftarrow features\_batch$
    $labels[i \times dataloader.batch\_size : (i+1) \times dataloader.batch\_size] \leftarrow labels\_batch$
    $i \leftarrow i + 1$
    if $i \times dataloader.batch\_size \geq n\_sample$ **then**
      break
    end if
  end for
  return $features, labels$
end function=0

Le pseudo code suivant montre comment nous avons fait pour utiliser un random forest avec les features extraites d'un modèle ResNet

Algorithm 2 Apprentissage et évaluation du random forest par transfer learning

$model \leftarrow models.resnet18(pretrained = True)$
$(train\_features, train\_labels) \leftarrow extract\_features(train\_loader, model, 10000)$
$(test\_features, test\_labels) \leftarrow extract\_features(val\_loader, model, 2000)$
$clf \leftarrow RandomForestClassifier()$
$clf.fit(train\_features, train\_labels)$
$pred \leftarrow clf.predict(test\_features)$
$print(classification\_report(test\_labels, pred))$ =0

3.3 Transfer Learning B avec RestNet50

Notre approche consistait à expérimenter des modèles pré entraînés selon un transfer learning de type B. Nous avons choisi Resnet50 comme modéle qui a une architectures représentatives de l'état de l'art des réseaux neurones convolutifs. Resnet50 est similaire à VGG-16 mais avec une capacité supplémentaire

de cartographie d'identité.

Voici les principaux réglage d'hyperparamètres que nous avons fait pour personnaliser l'architecture pré entraînée de notre modéle.

Augmentation de l'image :

La plupart des modèles pré entraînés utilisent des images d'entrée de taille minimale de 224 x 224 pixels. Le redimensionnement de l'image de 128 x 128 à 224 x 224 permet de s'adapter à cette exigence.

La normalisation des images :

De plus, la normalisation des images d'entrée est une étape importante dans le prétraitement des données avant de les alimenter dans un modèle d'apprentissage profond. La normalisation permet de s'assurer que les entités de l'image ont des plages et des distributions similaires, ce qui contribue à stabiliser le processus d'apprentissage et à améliorer les performances du modèle.
Les constantes de normalisation que nous avons utilisé, mean = [0,485, 0,456, 0,406] et std = [0,229, 0,224, 0,225], qui sont couramment utilisées. .

```
tforms = transforms.Compose([transforms.Resize((224, 224)), transforms.ToTensor(),
                             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])

train_tfroms = transforms.Compose([transforms.Resize((224, 224)),
                                   transforms.RandomHorizontalFlip(),
                                   transforms.ColorJitter(),
                                   transforms.ToTensor(),
                                   transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                                        std=[0.229, 0.224, 0.225])])
```

Taux d'apprentissage :

Nous avons commencé à entraîner le modèle ResNet50 en choisissant le taux d'apprentissage par (0.001) et nous avons également remplacé la dernière couche entièrement connectée du réseau VGG-16 par un classificateur personnalisé et que nous avons défini le nombre de sorties sur 2, ce qui correspond au nombre de classes de votre nouvel ensemble de données.

Cela permet au réseau de faire des prédictions spécifiquement pour la nouvelle tâche à accomplir, plutôt que d'essayer d'adapter les données aux sorties à usage général du réseau d'origine. Voir le pseudo code ci-dessous.

Algorithm 3 get_pretrained_model

```
function GET_PRETRAINED_MODEL(model_name)
    model ← models.resnet50(pretrained = True)
    for param in model.parameters() do
        param.requires_grad ← False
    end for
    n_inputs ← model.fc.in_features
    n_classes ← 2
    model.fc ← nn.Sequential(nn.Linear(n_inputs, 256),
nn.ReLU(), nn.Dropout(0.2), nn.Linear(256, n_classes))

    MODEL ← model.to(device)
    return MODEL
end function=0
```

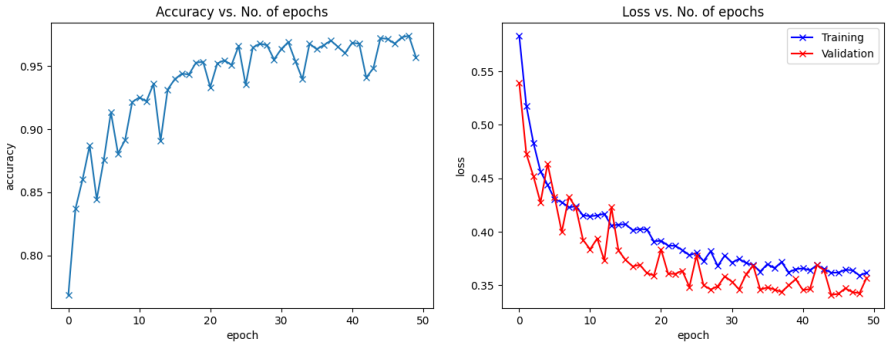
4 Résultats & Analyse comparative

Nous allons maintenant vous présenter les résultats que nous avons obtenus avec nos différents modèles, à la fois avec les CNN, que ceux réalisé à partir des méthodes A et B de transfer learning.

4.1 Résultats des CNN

1er CNN de Cao et Chen

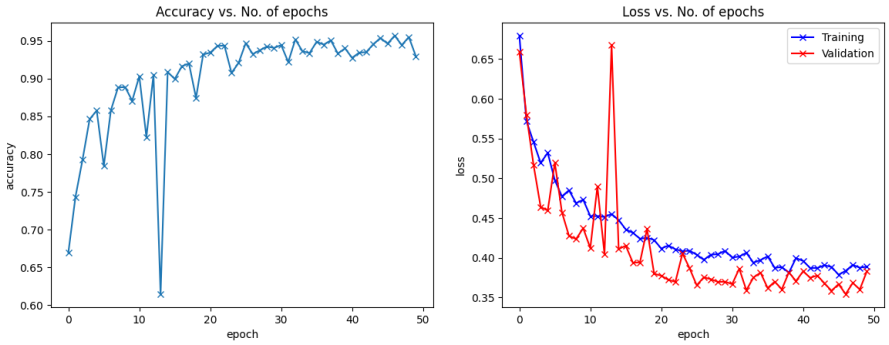
Nous avons commencé par reproduire le 1er CNN réalisé par Cao et Chen. Nous avons réussi à obtenir une accuracy sur l'échantillon d'évaluation de 97,69%, et une loss de 33,68%. Cette accuracy est très semblable à celle obtenue par les chercheurs dans leur modèle.



Nous constatons que l'accuracy augmente fortement au début, pour ensuite augmenter de plus en plus lentement au fur et à mesure des epochs. La loss de l'échantillon de train et de validation restent très proches au fur et a mesure du temps, ce qui nous conforte dans l'idée que nous ne sommes probablement pas dans un cas de surapprentissage.

CNN + Data Augmentation (DA) + Dropout (DO) + Optimiseur Adam

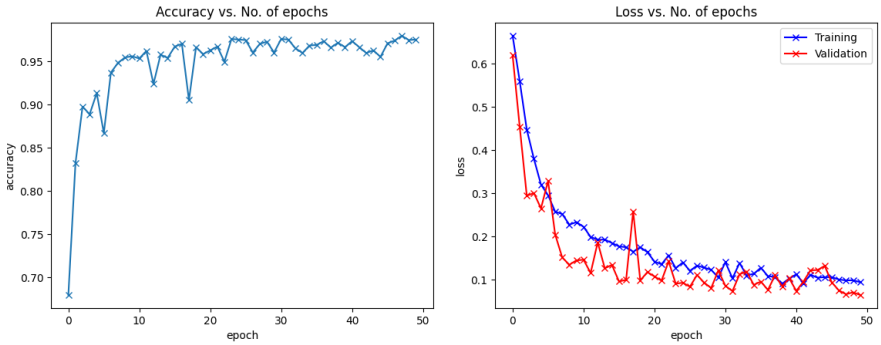
Ce modèle était le meilleur de Cao et Chen. Dans notre cas, nous arrivons à une accuracy sur l'échantillon d'évaluation de 95,50%, avec une loss de de 36,02%.



Les remarques quant aux graphiques sont assez similaires que précédemment, à avoir une accuracy qui augmente énormément au départ avant de se mettre à stagner, et une loss qui diminue avant de se mettre à stagner, et qui est très similaire pour l'échantillon de train et de validation.

CNN + DA (avec GaussianBlur) + padding + DO (Adam)

Nous avons repris le meilleur modèle des chercheurs, en ajoutant certaines caractéristiques comme du GaussianBlur et du padding. Nous sommes arrivés à une accuracy de 97,56% sur l'échantillon d'évaluation, et une loss de 8,52%.

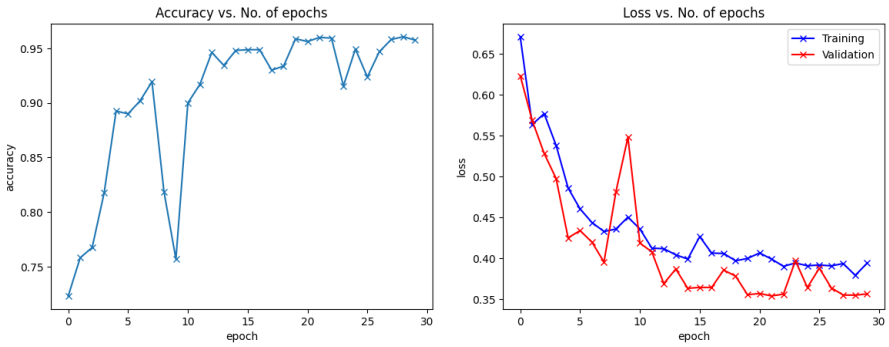


On constate que l'accuracy augmente très très rapidement au début, avant de se mettre à stagner par la suite. De même pour la loss qui diminue très rapidement pour les données de train et de validation avant de se mettre à stagner. Cette loss est quasiement identique pour ces deux échantillons.

CNN + DA (avec GaussianBlur) + padding + DO + L2 (Adam)

Pour finir sur les CNN, nous avons voulu essayé d'ajouter une regularisation L2 sur notre dernier modèle. Nous obtenons sur l'échantillon de validation une

accuracy de 96,06%, et une loss de 37,89%.



Les remarques sont toujours les mêmes que pour les graphiques précédents, si ce n'est que nous observons un large pic pour la dixième epoch.

4.2 Résultats du transfer learning A

Ci-dessous les performances du modèle de random forest utilisant le transfer learning de type A avec un modèle pré-entrainé de deep learning avec Resnet18 :

	precision	recall	f1-score	support
0	0.91	0.89	0.90	1000
1	0.90	0.92	0.91	1000
accuracy			0.90	2000
macro avg	0.90	0.90	0.90	2000
weighted avg	0.90	0.90	0.90	2000

Nous obtenons une précision de 90%, sur nos données. Ce qui est tout a fait satisfaisant comparé au modèle avec un CNN, même si cela reste en dessous.

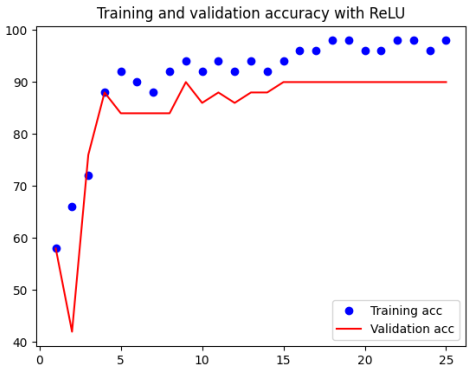
De même pour notre AdaBoost :

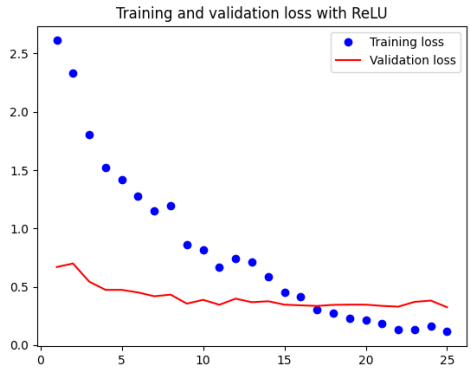
	precision	recall	f1-score	support
0	0.90	0.91	0.90	1000
1	0.91	0.89	0.90	1000
accuracy			0.90	2000
macro avg	0.90	0.90	0.90	2000
weighted avg	0.90	0.90	0.90	2000

Nous obtenons une précision de 90%, sur nos données. Les méthodes de machine learning avec un transfer learning ont finalement donné de bons résultats même si aucun ne s'est particulièrement démarqué.

4.3 Résultats du transfer learning B

En examinant les graphiques ci dessous, nous constatons que le modèle Resnet50 présente de bonnes courbes d'accuracy et de loss.





Les courbes de Training et de validation accuracy augmentent ensemble au fil des epoch, cela indique que le modèle s'améliore sur les deux ensembles de données. Nous avons pu obtenir une précision de près de Train Acc : 96% et Valid Acc : 90.00% sur seulement 25 epoch où le temps nécessaire pour exécuter une seule époque était de 10 min maximum.

4.4 Comparaison de l'ensemble des résultats

Voici un récapitulatif des résultats obtenus :

Modèles	Validation Loss	Validation Accuracy
CNN	33,69%	97,69%
CNN + DA + DO (Adam)	36,02%	95,50%
CNN + DA (GaussianBlur) + padding + DO (Adam)	8,52%	97,56%
CNN + DA (GaussianBlur) + padding + DO + L2 (Adam)	37,89%	96,06%
Resnet50	96,00%	90,00%
VGG16	96,00%	92,00%
Random Forest avec Resnet18 pré-entrainé	X	90%
AdaBoost avec Resnet18 pré-entrainé	X	90%

5 Discussion

Nous avons donc créé différents modèles pour essayer de classifier les images. Nous avons tout d'abord reproduit des modèles réalisés par Cao et Chen, et nous

avons essayé de les améliorer. Globalement, les CNN produisent de meilleurs résultats que les autres méthodes. En effet, nous obtenons des accuracy de plus de 97%, ce que rend les prédictions d'ores et déjà très fiables. Néanmoins, toutes les méthodes se sont révélées intéressantes puisque nous ne descendons pas en dessous de 90% d'accuracy. Il aurait été cependant intéressant d'avoir le f1score de chaque méthode pour pouvoir les comparer avec une métrique plus fiable et commune. Il aurait été intéressant également d'utiliser un autre modèle de pré-apprentissage que resnet18, qui a été choisi pour sa rapidité, pour alléger les temps de calculs. Il pourrait aussi être interessant de tester plusieurs méthodes d'augmentation des données, ainsi que différents optimiseurs pour les CNN.

6 Conclusion

6.1 Notre avis sur les apports et limites du projet

Ce projet a été très bénéfique pour nous pour plusieurs raisons. Tout d'abord cela nous a permis de découvrir un projet concret dans lequel appliquer du deep learning. Nous avions pu étudier de nombreuses notions en profondeur durant nos cours de deep learning. Cependant, certaines d'entre elles demeuraient assez floues à cause de notre manque de pratique jusqu'à présent. Le fait de pouvoir mettre en pratique nos connaissances avec pytorch nous a donc permis à la fois de nous familiariser avec cette librairie, mais aussi de mieux comprendre les notions que nous avions abordées, comme les réseaux de neurones convolutifs. Nous avons aussi pu découvrir et étudier de nouvelles notions comme le transfer learning. De même, nous avons trouvé cela très intéressant d'être confrontés pour la première fois à un problème de classification sur des images.

Nous aurions aimé réussir à tester plus de modèles différents, afin de comprendre notamment plus en détails le fonctionnement de certains paramètres, et produire de meilleures métriques de classification. Cependant nous avons été contraints par le temps lors de ce projet. Nous nous sommes donc focalisés sur le fait de répondre aux différents attendus, mais nous n'avons pas pu approfondir ce rendu autant que nous le voulions.

6.2 Notre avis sur le lien avec le cours de Deep Learning

Ce projet était donc un très bon complément à ce que nous avons pu étudier lors de notre cours de Deep Learning. Cependant, il aurait pu être pertinent selon nous de développer moins en profondeur certaines parties du cours, afin de pouvoir consacrer un peu plus de temps directement à la pratique sur python. En effet, nous avions eu du mal à assimiler certaines parties très théoriques, et un juste milieu entre un peu moins de théorie et plus de pratique lors du cours aurait pu nous aider à mieux comprendre directement certaines notions. De plus, nous avons pu nous former nous même à pytorch avec des sites tels que Datacamp. Néanmoins nous ne sommes pas forcément sûrs d'avoir adoptés tous

les bons réflexes et les bonnes pratiques à utiliser. Pouvoir les étudier en cours nous aurait permis d'être plus sûrs de nous sur ces aspects.

7 Références

— 1. CAO, Quoc Dung et CHOE, Youngjun. Building damage annotation on post-hurricane satellite imagery based on convolutional neural networks. Natural Hazards, 2020, vol. 103, no 3, p. 3357-3376.

— 2. LIANG, Yilong, MONTEIRO, Sildomar T., et SABER, Eli S. Transfer learning for high resolution aerial image classification. In : 2016 IEEE Applied Imagery Pattern Recognition Workshop (AIPR). IEEE, 2016. p. 1-8.

— 3. ZOU, MaoYang et ZHONG, Yong. Transfer learning for classification of optical satellite image. Sensing and Imaging, 2018, vol. 19, no 1, p. 6.