Rapport de stage assistant ingénieur

Juin - Août 2021

Traitement d'images par réseaux de neurones avec fastai & PyTorch

Stage supervisé par Benjamin Lepers au Lab-STICC de l'ENSTA Bretagne

Hugo Sabatier

ROB 2022

Remerciements

Tout d'abord, j'adresse mes remerciements aux responsables de la spécialité de ma formation à l'ENSTA Bretagne, Luc Jaulin et Benoît Zerr qui m'ont permis d'effectuer ce stage dans un domaine que je n'avais pas encore étudié, l'intelligence artificielle. J'ai ainsi pu trouver ce stage qui était en totale adéquation avec mes attentes.

Je tiens à remercier vivement mon maître de stage, Benjamin Lepers, ingénieur de recherche au Lab-STICC de l'ENSTA Bretagne, pour son accueil, le temps passé ensemble, l'accompagnement dans l'apprentissage de ce nouveau domaine et le partage de son expérience.

Résumé

Ce rapport de stage en intelligence artificielle porte sur la classification d'images multi-label par apprentissage profond appliquée aux fonds marins. Les librairies utilisées pour développer les méthodes d'apprentissage automatique sont PyTorch et fastai. En raison de la faible quantité de données labellisés, des techniques d'apprentissage par transfert seront employées.

Mots-clés

Intelligence artificielle, Apprentissage automatique, Apprentissage profond, Classification d'images multi-label, Apprentissage par transfert, fastai, PyTorch, fonds marins

Abstract

This internship report in artificial intelligence is about multi-label image classification by deep learning applied to the seabed. The libraries used to develop the machine learning methods are PyTorch and fastai. Due to the small amount of labeled data, transfer learning techniques will be used.

Keywords

Artificial intelligence, Machine learning, Deep learning, Multi-label image classification, Transfer learning, fastai, PyTorch, seabed

Table des matières

Liste des figures

Introduction

Ce stage a été réalisé à l'ENSTA Bretagne, grande école d'ingénieur et centre de recherche, au sein du Lab-STICC (Laboratoire des sciences et techniques de l'information, de la communication et de la connaissance) dans le cadre du projet NARVAL (Navigation Autonome par Reconnaissance Visuelle et Acoustique) et a été encadré par Benjamin LEPERS, ingénieur de recherche au Lab-STICC de l'ENSTA Bretagne. Il s'est déroulé du 7 juin au 27 août 2021. Benjamin LEPERS travaille sur le projet NARVAL aux côtés de 2 autres ingénieurs Pierre Narvor et Marie Ponchart. Ce projet est dirigé par Benoît Zerr, enseignant chercheur à l'ENSTA Bretagne, et est en partenariat avec COMEX SA (PME) et Notilo plus (PME).

L'objectif du projet NARVAL est de développer un système de navigation autonome pour un drone sous-marin à partir de données de capteurs optiques et acoustiques. La mission sera décomposée en 2 phases. Dans la première phase, l'engin cartographie une zone inconnue à l'aide de capteurs acoustique, optique et de navigation. L'objectif est d'identifier les cibles potentielles grâce à un algorithme de détection. Avec la fusion des données acoustiques et optiques, un modèle numérique de terrain est obtenu. Une fois ce modèle obtenu la deuxième phase est lancée. Un deuxième engin avec des capteurs moins précis se localisera par corrélation entre ses mesures et le modèle numérique de terrain obtenu lors de la première phase l'engin pourra se repérer sur le site et transmettre les déplacements à effectuer au contrôleur de mission.

Dans ce contexte, une classification automatique des fonds marins, et/ou du type d'environnement par apprentissage automatique est nécessaire au développement du projet. Pour la classification d'images, les méthodes d'apprentissage profond avec un jeu de données labellisées suffisant permettent d'obtenir les meilleurs résultats. Cependant, il existe à ce jour peu de jeux de données labellisées sur les fonds marins. Comment atteindre des résultats proches de ceux obtenus dans des domaines où les données sont nombreuses ? La technique de transfert par apprentissage qui consiste à utiliser un modèle pré entrainé sur un grand jeu de données labellisées (ImageNet le plus souvent), consiste à entraîner uniquement les dernières couches du modèles sur un jeu de données spécifique de taille modéré.

L'objectif de ce stage est de se familiariser avec les méthodes d'apprentissage profond pour de la classification d'images multi-label. Les librairies fastai et PyTorch seront utilisées sur un jeu de données de fonds marins obtenu avec des caméras GoPro. PyTorch est aujourd'hui la bibliothèque d'apprentissage profond qui connaît la croissance la plus rapide au monde et est déjà utilisée pour la plupart des articles de recherche lors des conférences les plus importantes. PyTorch fonctionne mieux comme une bibliothèque de

base de bas niveau, fournissant les opérations de base pour les fonctionnalités de plus haut niveau. La bibliothèque fastai est très utilisée pour ajouter cette fonctionnalité de plus haut niveau à PyTorch. Les méthodes à utiliser n'ayant pas été étudiées précédemment, ce stage comprend une forte dimension d'apprentissage. De ce fait, le format classique du rapport qui retrace les missions réalisées par la stagiaire et ce qu'il a apporté à l'institut d'accueil ne sera par conséquent pas forcément le plus adapté. Les cours vidéos sur la librairie fastai permettent d'acquérir les connaissances suffisantes pour créer des modèles fonctionnels. Une fois que les concepts de base de l'apprentissage profond auront été assimilés, il conviendra d'implémenter un modèle sans la librairie de haut niveau fastai pour entrer plus en profondeur dans la compréhension de la discipline, dès lors nous serons en mesure de comparer les résultats obtenus par les différents modèles.

1 Concepts

Dans cette partie nous rappelons brièvement les grands principes fondateurs de l'apprentissage profond.

1.1 L'origine des réseaux neuronaux

Les réseaux de neurones artificiels sont vus comme une simplification des réseaux de neurones biologiques. Le neurone reçoit des informations par ses dendrites, les traite dans son noyau et transmet en réponse une information par son axone. Un neurone se représente schématiquement sous cette forme (Figure 1).

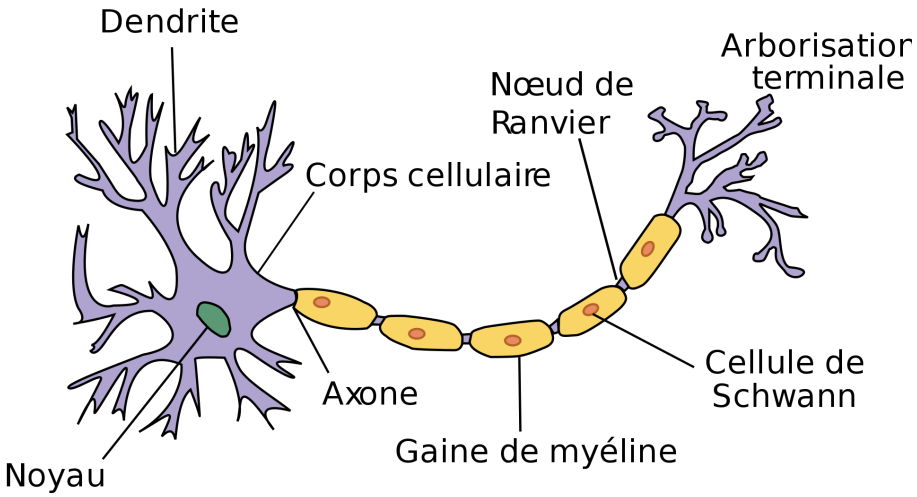

FIGURE 1 – Neurone biologique

En 1943, le neurophysiologiste Warren McCulloch et l'informaticien Walter Pitts ont développé un modèle mathématique simplifié d'un neurone réel pouvait être représenté à l'aide d'une simple addition et d'un seuillage (Figure 2) (Howard and Gugger [2020]).

Le premier dispositif a avoir exploité ce modèle s'appelait le Perceptron Mark I et avait pour objectif de percevoir, de reconnaître et d'identifier son environnement automatiquement, c'est à dire sans aucun entraînement ou contrôle humain. La machine était capable de reconnaître avec succès des formes simples. Cependant, il a rapidement été montré qu'une seule couche de neurone artificiel était incapable d'apprendre certaines fonctions mathématiques simples mais essentielles, c'est en partie pour cette raison que l'intelligence artificielle a connu une période difficile pendant les deux décennies suivantes. Pourtant, l'utilisation de plusieurs couches de neurones artificiels comme solution pour remédier à ces limitations était déjà connue.

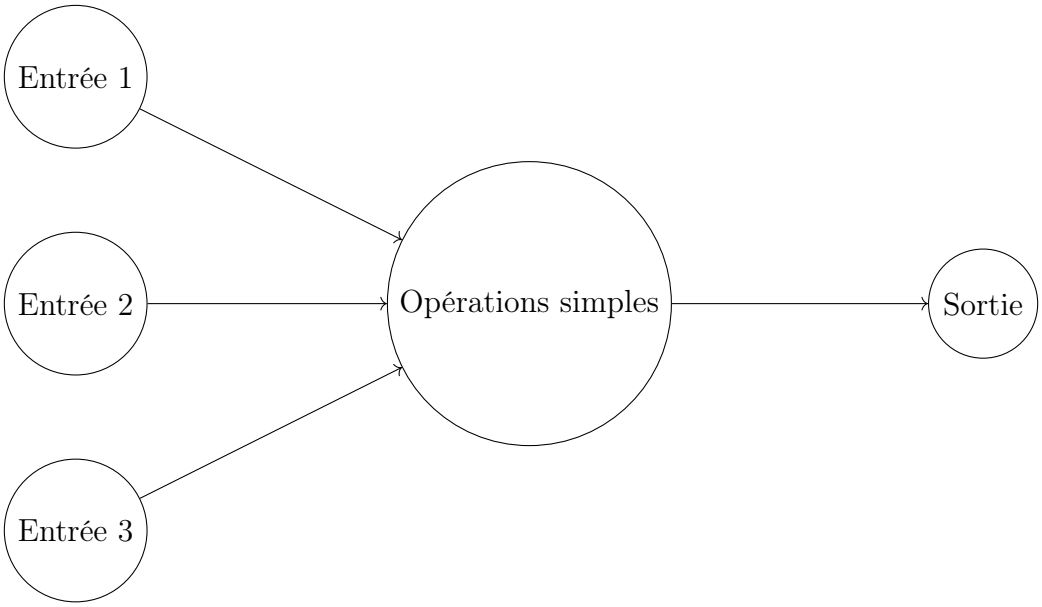

FIGURE 2 – Neurone artificiel

Bien que les chercheurs aient montré il y a 30 ans que pour obtenir de bonnes performances pratiques, il fallait utiliser encore plus de couches de neurones, ce n'est qu'au cours de la dernière décennie que ce principe a été plus largement apprécié et appliqué. Les réseaux neuronaux exploitent enfin pleinement leur potentiel, grâce à l'utilisation d'un plus grand nombre de couches et à la capacité de le faire grâce aux progrès en informatique, à l'augmentation des données disponibles et aux améliorations algorithmiques qui permettent d'entraîner les réseaux neuronaux plus rapidement et plus facilement (Goodfellow et al. [2016]).

1.2 L'apprentissage automatique

L'apprentissage automatique est, comme la programmation ordinaire, un moyen d'amener les ordinateurs à accomplir une tâche spécifique. Normalement, il est assez facile d'écrire les étapes pour accomplir une tâche lors de l'écriture d'un programme. Il suffit d'écrire les étapes exactes nécessaires à la réalisation de la tâche, comme s'il fallait effectuer la tâche à la main, puis de les traduire en code. Néanmoins, pour la reconnaissance d'objets dans une photo, ce n'est pas si simple. Il est impossible de décrire précisément quelles sont les étapes suivies lorsqu'un humain reconnaît un objet dans une photo car tout se passe inconsciemment dans son cerveau.

De ce fait, il a fallu repenser la programmation pour les cas où expliciter chaque étape infime du processus était impossible. Dans les années 1950, le chercheur Arthur Samuel a eu l'idée de montrer des exemples du problème à résoudre à la machine et laisser trouver

elle-même comment le résoudre au lieu de lui indiquer les étapes exactes requises pour résoudre un problème. Cette méthode s'est avérée très efficace, elle a d'ailleurs permis à son programme de jeu de dames de battre le champion de l'État du Connecticut en 1961 (Goodfellow et al. [2016]). L'apprentissage automatique devient alors l'entraînement de programmes en permettant à un ordinateur d'apprendre de son expérience, plutôt qu'en codant manuellement les différentes étapes.

1.3 Le réseau neuronal

Le réseau neuronal s'apparente à une fonction si flexible qu'elle pourrait être utilisée pour résoudre n'importe quel problème donné, simplement en faisant varier ses paramètres. Une preuve mathématique appelée théorème d'approximation universelle montre que cette fonction peut résoudre n'importe quel problème à n'importe quel niveau de précision, en théorie. En d'autres termes, un réseau neuronal est un type particulier de modèle d'apprentissage automatique, qui s'inscrit parfaitement dans la conception originale de Arthur Samuel. Les réseaux neuronaux sont spéciaux parce qu'ils sont très flexibles, ce qui signifie qu'ils peuvent résoudre un éventail exceptionnellement large de problèmes simplement en trouvant les bons paramètres (Vannieuwenhuyze [2019]). Cela revient à minimiser une fonction coût pour améliorer le modèle pendant l'apprentissage.

2 Mise en œuvre

Nous passons maintenant à l'implémentation de ces grands principes. Étant donné que ce stage comprend essentiellement de la programmation il peut être intéressant d'avoir accès au code lors de la lecture du rapport. Pour ne pas gêner le lecteur qui ne souhaiterait pas voir la partie programmation, les codes ont été placés en annexes, le numéro de figure du code mène directement à ce dernier et le nom de la figure contenant le code ramène là où le code a été cité dans le rapport. Ainsi il est simple de visualiser rapidement le code puis de revenir à la lecture du rapport.

2.1 Mise en place de l'environnement

2.1.1 Carte graphique NVIDIA

Il faut avoir accès à un ordinateur équipé d'une carte graphique (GPU) de la marque NVIDIA car nous aurons besoin de la librairie CUDA et cette librairie est développée par NVIDIA et adaptée à ses GPUs. Ce processeur spécial intégré aux ordinateurs est capable de traiter des milliers de tâches simultanément, en particulier pour afficher des environnements 3D sur un ordinateur pour jouer à des jeux vidéo. Il se trouve que ces mêmes tâches de base sont très similaires à celles des réseaux neuronaux, de sorte que les GPU peuvent exécuter des réseaux neuronaux des centaines de fois plus rapidement que les CPU ordinaires. Tous les ordinateurs modernes contiennent un GPU, mais peu d'entre eux contiennent le type de GPU nécessaire à l'apprentissage profond. Toutefois, il existe une solution qui est de louer l'accès à un ordinateur sur lequel tout ce qui est nécessaire est déjà pré-installé et prêt à fonctionner, ce qui évite d'ailleurs d'avoir à installer les différentes bibliothèques. Cette solution est payante bien que certaines options soient gratuites. Ayant à disposition un ordinateur équipé d'un GPU NVIDIA, cette solution n'a pas été envisagée.

2.1.2 CUDA

L'installation de CUDA est nécessaire, cette technologie permet d'exploiter les capacités du GPU pour exécuter les calculs plutôt que le CPU. Une fois installée, sous Ubuntu 18.04 l'activation du GPU NVIDIA via l'UEFI Secure Boot est requise. À noter que dans mon cas cette activation entraîne une erreur qui n'a pas pu être corrigée, l'écran se fige en cas de mis en veille prolongée et force un redémarrage de l'ordinateur.

2.1.3 Jupyter Notebook

L'ensemble de ce projet a été développé sur Jupyter Notebook. Cette une application web open source permet de regrouper au sein d'un même document du code exécutable en direct mais aussi du texte narratif, des équations et des visualisations. L'utilisation de Jupyter Notebook pour la science des données en Python, notamment pour l'apprentissage profond, se justifie pour sa puissance, sa flexibilité et sa facilité d'utilisation. Il peut être utile de travailler sur un environnement Anaconda pour maîtriser les installations faites sur l'environnement. Afin de lancer notre environnement sur Jupyter Notebook, il faut depuis un terminal, activer l'environnement Anaconda par la commande *conda activate "nom de l'environnement"*, accéder au dossier du projet et lancer *jupyter notebook*, ce qui lancera l'application dans le navigateur.

2.2 Préparation des données

2.2.1 Lire la trame de données avec Pandas

Pandas est une bibliothèque Python utilisée pour manipuler et analyser des données tabulaires et des séries chronologiques. La classe principale est *DataFrame*, qui représente un tableau de lignes et de colonnes. Il est possible d'obtenir un *DataFrame* à partir d'un fichier CSV, d'une table de base de données, de dictionnaires Python et de nombreuses autres sources. On construit ici la fonction *read_ dataframe()* pour transformer le fichier pickle (.pkl) dans lequel se trouve les labels associés aux différentes images en un tableau de type pandas *DataFrame*. *DataFrame* est une structure de données étiquetée en deux dimensions avec des colonnes de types potentiellement différents (McKinney [2012]). C'est un tableau comparable à une feuille de calcul ou une table SQL, ou encore un dictionnaire de séries d'objets. C'est généralement l'objet Pandas le plus utilisé. Avec la méthode *drop_ duplicates()* on s'assure qu'il n'y est pas de doublon dans le jeu de données. En exécutant la commande *DataFrame.tail()* on obtient les dernières lignes du *DataFrame* ce qui nous permet de visualiser la structure du tableau (Figure Annexes 2).

	fname	labels	is_valid
354	image_8_3	algues rochers	False
355	image_8_4	algues	False
356	image_8_1	rochers	False
357	image_8_7	algues rochers sable	True
358	image_8_8	rochers sable	False

FIGURE 3 – Contenu du Dataframe

On remarque que le tableau comprend 3 colonnes, *fname* pour le nom des fichiers images, *labels* pour les étiquettes associées à chaque image, elles résultent d'une combinaison entre les 3 classes : *algues, rochers, sable*, il y a donc $2^3 = 8$ possibilités de label. Dans cet exemple les valeurs de labels ont volontairement été modifiées pour montrer les différentes combinaisons. La dernière colonne *is_ valid* permet de diviser le jeu de données en 2 catégories, elle distingue la catégorie validation, qui ne sera pas utiliser pour entraîner le modèle mais uniquement pour tester ses performances, de la catégorie entraînement qui elle servira à optimiser le modèle.

2.2.2 Encodage des labels pour la classification multi-label

Une des grandes particularités de la classification multi-label est qu'elle nécessite de fournir les labels dans un certain format (Chollet [2017]). Au lieu de simplement associé chaque label à un entier il est ici nécessaire de mettre les labels sous forme de liste de 0 et de 1. Ainsi au lieu d'encoder le label ['algues','sable'] par [0,2] en associant respectivement 0, 1 et 2 aux labels 'algues', 'rochers' et 'sable' il faut l'encoder par une liste de la longueur du nombre de classe et indiquer 1 lorsque que la classe est présente et 0 quand elle ne l'est pas. C'est pourquoi ['algues','sable'] s'encode alors par [1, 0, 1]. Voici une façon simple d'implémenter l'encodage des labels et une autre façon de le coder en utilisant la fonction *one_ hot_ encoding()* du module *nn* de PyTorch (Figure Annexes 3 et 4). On peut créer une fonction pour décoder les labels qui servira à la fin du programme lorsqu'il faudra décoder le résultat de la prédiction (Figure Annexes 5).

On ajoute la nouvelle colonne *label_ encoded* au *DataFrame* et on affiche le début du tableau :

	fname	labels	is_valid	labels_encoded
0	image_5_42	algues sable	False	[1.0, 0.0, 1.0]
1	image_5_40	algues sable	False	[1.0, 0.0, 1.0]
2	image_5_44	algues sable	False	[1.0, 0.0, 1.0]
3	image_5_46	rochers sable	False	[0.0, 1.0, 1.0]
4	image_5_50	rochers sable	False	[0.0, 1.0, 1.0]

FIGURE 4 – Ajout de la colonne *label_ encoded*

2.2.3 Division en données d'entraînement et de test ou de validation

Comme nous l'avons vu précédemment la colonne *is_ valid* permet de diviser le jeu de données en jeu d'entraînement et jeu de validation. On sépare donc les données en 2 selon la colonne *is_ valid* en définissant la fonction *splitter()* (Figure Annexes 6).

2.2.4 Création du *Dataset*

Le *DataFrame* regroupe simplement les informations pour trouver l'image, nous allons maintenant créer le *Dataset* qui va permettre d'accéder aux images et de leur appliquer des transformations. PyTorch fournit deux primitives de données : *torch.utils.data.DataLoader* et *torch.utils.data.Dataset*. *Dataset* stocke les échantillons et leurs labels correspondantes, et *DataLoader* regroupe le *Dataset* sous forme de paquet (batches), plus pratique pour entraîner le modèle. La classe *Dataset* doit implémenter deux méthodes : $\_\_len\_\_()$ et $\_\_getitem\_\_()$. La première doit retourner le nombre d'éléments dans l'ensemble de données ; la seconde doit retourner l'élément, constitué d'une image et de son label correspondant. L'ensemble de données ne contient pas nécessairement les données, mais il fournit un accès uniforme à celles-ci par le biais de $\_\_len\_\_()$ et $\_\_getitem\_\_()$. Dans notre exemple le *Dataset* renvoie une image couleur de taille 28x28 pixels et le label encodé correspondant (Figure Annexes 7).

2.2.5 Transformations

Les données ne se présentent pas toujours sous la forme optimale pour l'entraînement des algorithmes d'apprentissage automatique. Il est nécessaire d'appliquer des transformations pour effectuer une certaine manipulation des données et les rendre appropriées pour l'apprentissage. Notre *Dataset* doit lire une image puis s'assurer qu'elle soit sous la forme d'un tenseur pour la suite du traitement. Pour cela la transformation *ToTensor()* convertit une image PIL ou un tableau numérique NumPy en un FloatTensor. et met les valeurs d'intensité des pixels de l'image dans la plage [0., 1.]. Les *Dataset* ont deux paramètres *transform* pour modifier les images et *target_transform* pour modifier les labels. Le module *torchvision.transforms* offre plusieurs transformations couramment utilisées. Les transformations usuelles sont par exemple *Resize()* qui permet de choisir la taille de l'image et *Normalize()* qui normalise les pixels de l'image. L'objectif de la normalisation est de faciliter la convergence lors de l'apprentissage par la méthode de descente de gradient (Müller and Guido [2018]). La normalisation est également nécessaire pour que certains algorithmes modélisent correctement les données. Afin de normaliser les données il faut calculer la moyenne et l'écart type du jeu de données, on rentre les données dans un *Dataset* non-normalisé et on applique la moyenne et l'écart type. Ensuite, on crée cette fois un *Dataset* pour l'entraînement et un pour la validation en appliquant la normalisation avec les paramètres calculés (Figure Annexes 8).

2.2.6 Insertion dans un *DataLoader*

Nous passons le jeu de données comme argument au *DataLoader*. Ce dernier prend en charge la mise sous forme de paquets appelés batch, utiles pour l'algorithme de d'optimisation (descente de gradient) et pour gérer la mémoire des données. Ici, nous définissons une taille de batch de 64, c'est-à-dire que chaque élément du *DataLoader* renverra un paquet de 64 images et labels. On choisit de mélanger les données avec l'argument *shuffle* cela permet d'assurer une distribution aléatoire des données dans les batch. Cette astuce de diviser le jeu de données en petit paquet permet d'augmenter les performances de calcul (Chollet [2017]). En effet, lors de l'apprentissage d'un modèle, il est généralement intéressant de passer les échantillons en batch, de remanier les données à chaque nouvel apprentissage sur le jeu de données complet pour réduire le sur-apprentissage du modèle, et d'utiliser le multitraitement de Python pour accélérer la récupération des données. Afin de tester le parcours du Dataloader il est possible d'afficher certaines images du jeu de données (Figure Annexes 9).

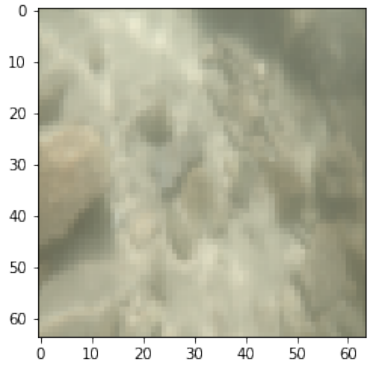

FIGURE 5 – Label : ['rochers', 'sable']

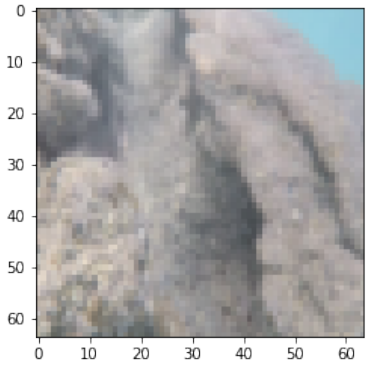

FIGURE 6 – Label : ['rochers']

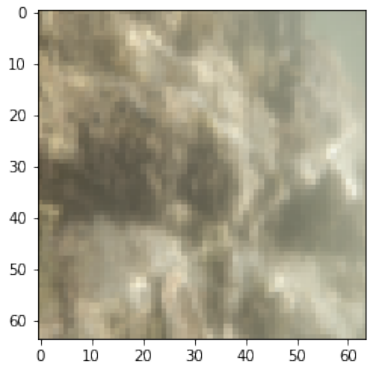

FIGURE 7 – Label : ['algues', 'rochers']

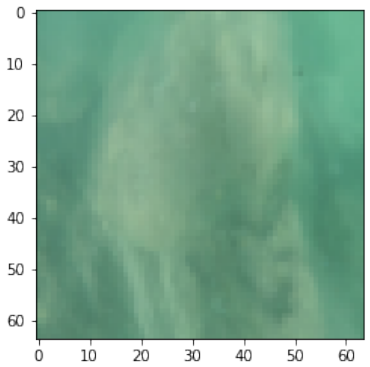

FIGURE 8 – Label : ['rochers']

2.3 Définition du modèle

2.3.1 Importer un modèle

Les réseaux neuronaux sont composés de couches/modules qui effectuent des opérations sur les données. Un réseau neuronal est un module en soi qui se compose d'autres modules ou couches. Cette structure imbriquée permet de construire et de gérer facilement des architectures complexes. Le choix de cette architecture est crucial. Il est possible d'importer une architecture déjà construite pour ne pas partir de zéro. Avec l'apprentissage par transfert il est même possible d'importer un modèle déjà entraîné. Cette option est idéale dans notre contexte d'étude car il y a peu de données labellisées des fonds marins et la quantité de données disponibles est un critère essentiel pour l'apprentissage d'un modèle et donc pour sa performance. Ainsi, nous importons un modèle ResNet-18 pré-entraîné sur ImageNet (https ://www.image-net.org/). ImageNet est une base de données d'images organisée selon la hiérarchie de WordNet (une hiérarchie de concepts allant de très généraux à modérément abstraits à très spécifiques), dans laquelle chaque nœud de la hiérarchie est représenté par des centaines et des milliers d'images. Le projet a contribué à faire progresser la recherche sur la vision par ordinateur et l'apprentissage profond. Les données sont mises gratuitement à la disposition des chercheurs pour un usage non commercial.

Le modèle étant déjà entraîné, nous allons simplement remplacer la dernière couche du modèle par une nouvelle couche non-entraînée. Cette couche renverra 3 sorties car nous avons 3 classes différentes. Nous voulons assurer que les paramètres des autres couches ne changent pas donc il est nécessaire de préciser que ces paramètres ne vont pas évoluer lors de l'optimisation du modèle en mettant l'attribut *requires_grad* à *False*. Avec la méthode *torch.cuda.is_available()* il est possible de vérifier si le GPU est disponible et donc correctement relié à notre programme. S'il est disponible il faut privilégier son utilisation en précisant au modèle sur quel processeur il doit être exécuté pour accélérer les opérations dans le réseau neuronal (Figure Annexes 10).

2.3.2 Création d'un modèle

Pour définir un réseau neuronal dans PyTorch, nous créons une classe qui hérite de *nn.Module*. Nous définissons les couches du réseau dans la fonction *__init__()* et spécifions comment les données passeront dans le réseau dans la fonction *forward()*. Nous choisissons ici un modèle simple avec 3 seulement couches classiques (Figure Annexes 11).

2.4 Optimisation des paramètres du modèle

2.4.1 Boucle d'optimisation

Maintenant que nous avons un modèle et des données, il est temps d'entraîner notre modèle en optimisant ses paramètres sur nos données. L'apprentissage d'un modèle est un processus itératif qui se décompose en plusieurs étapes (Figure Annexe 13) (Howard and Gugger [2020]) :

- Initialisation : La première étape est d'initialiser les paramètres du modèle, il y a différentes méthodes mais cela fonctionne bien en le faisant de façon aléatoire.

- Prédiction : à partir de ces paramètres actuels le modèle fait une prédiction sur les données d'entraînement.

- Perte : de cette prédiction on évalue la performance du modèle en calculant la perte qui renvoie un nombre petit si la performance du modèle est bonne.

signifie fonction coût, ou fonction perte. c'est par itération avec une méthode de descente de gradient qu'on peut minimiser cette fonction et trouver les bons poids du modèle sur le jeu d'entraînement

- Calcul du gradient : on mesure pour chaque poids, la façon dont le changement de ce poids modifierait la perte.

- Correction : on modifie tous les poids en fonction de ce calcul.

- Fin : les étapes se répètent jusqu'à ce qu'une certaine condition soit atteinte par exemple : une durée d'entraînement, un nombre d'itération des étapes pour le jeu d'entraînement complet, appelée une époque, une valeur de précision pour le modèle ou encore lorsque la précision du modèle commence à diminuer.

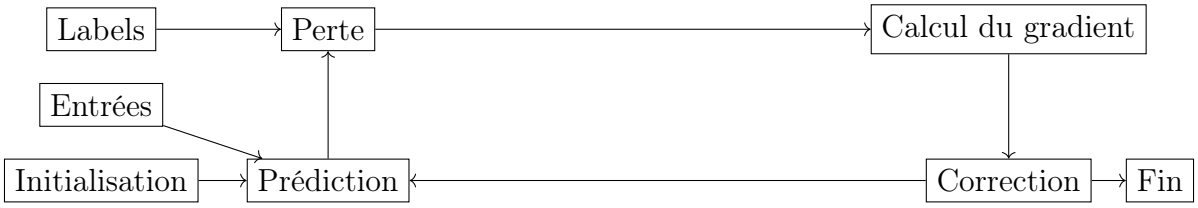

FIGURE 9 – Boucle d'optimisation des paramètres du modèle

2.4.2 Définition des métriques et de la fonction perte

Les métriques servent à évaluer la performance du modèle. La création du fonction qui renvoie la précision des prédictions du modèle est donc nécessaire, c'est le rôle de la fonction *accuracy_ multi()* (Figure Annexes 12).

Il est aussi nécessaire de définir les fonctions qui vont permettre le calcul de la perte et du gradient. Il en existe plusieurs mais elles ne sont pas toutes compatibles avec le multi-label. *nn.CrossEntropyLoss()* par exemple ne permet pas le mutli-label, nous choisissons à la place la fonction perte ou fonction coût *nn.BCEWithLogitsLoss()* (Paszke et al. [2017]). Son rôle est de calculer à quel point le modèle est proche de la vérité ou non, l'objectif est de minimiser cette fonction perte. On choisit ensuite l'optimiseur *torch.optim.SGD()* qui calcule dans quelle direction faire varier les paramètres du modèle pour améliorer la qualité de ses prédictions, dans notre cas en appliquant une descente de gradient stochastique. Elle permet l'ajustement des poids des couches du modèles en minimisant la fonction coût de manière itérative pendant la boucle d'entraînement.

2.4.3 Les hyperparamètres

Les hyperparamètres sont des paramètres ajustables qui permettent de contrôler le processus d'optimisation du modèle. Différentes valeurs d'hyperparamètres peuvent avoir un impact sur l'apprentissage du modèle et les taux de convergence (Howard and Gugger [2020]). Parmi eux il y a notamment : - le nombre d'époques, soit le nombre de fois où il faut itérer sur l'ensemble de données. - la taille du batch, c'est à dire le nombre d'échantillons de données traités par le modèle avant que les paramètres ne soient mis à jour - le taux d'apprentissage qui détermine la taille du pas avec lequel les paramètres doivent évoluer à chaque itération, il influence la vitesse à laquelle un modèle d'apprentissage automatique apprend. Ils sont parfois obtenus de manière empirique et la librairie fastai utilise des paramètres par défaut qui fonctionnent bien dans la plupart des cas.

2.4.4 Entraînement et validation du modèle

Chaque époque se compose de deux parties principales. La boucle d'entraînement qui itère sur l'ensemble de données d'entraînement et essaie de converger vers les paramètres optimaux. La boucle de validation ou de test dans laquelle le modèle est évalué via les métriques sur le jeu de validation. L'algorithme de backpropagation est inactivé, c'est à dire que les poids du modèles restent figés durant la phase de validation. On parle d'inférence (Figure Annexes 13).

3 Résultats

Les hyperparamètres choisis sont un batch de taille 8, 20 époques et un taux d'apprentissage de 0.001. La taille des images est fixé à (128,256) pixels. Cette partie comparera simplement les résultats obtenus pour ces hyperparamètres, il eut été intéressant d'en profiter pour les faire varier et chercher à les optimiser afin notamment d'observer quelle précision peut atteindre chacun des modèles mais l'objectif était déjà de réussir à les faire fonctionner et de comprendre leur fonctionnement.

3.1 Modèle 3 couches en PyTorch

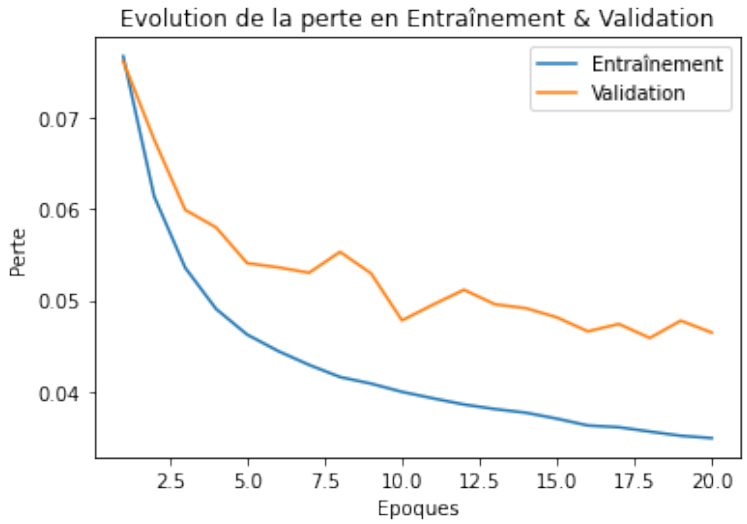

FIGURE 10 – Perte du modèle 3 couches en PyTorch

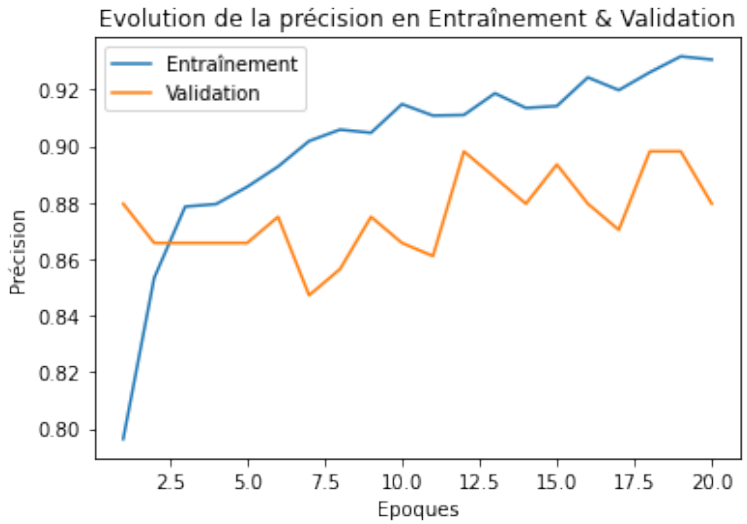

FIGURE 11 – Précision du modèle 3 couches en PyTorch

3.2 Modèle Resnet18 en PyTorch

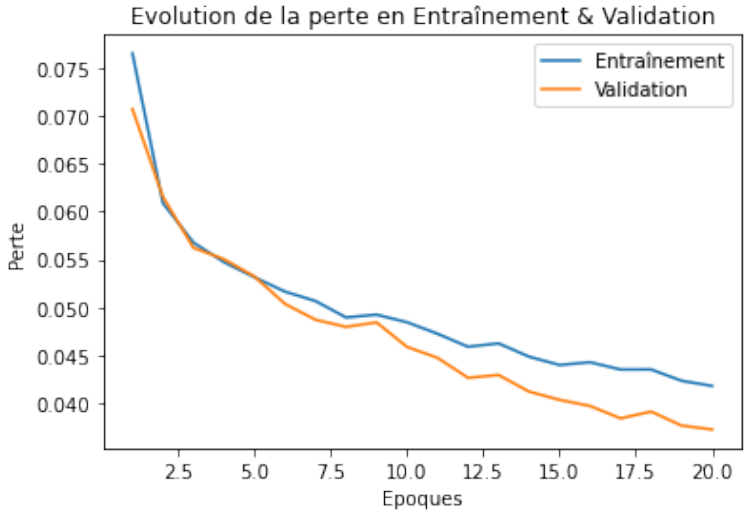

FIGURE 12 – Perte du modèle Resnet18 en PyTorch

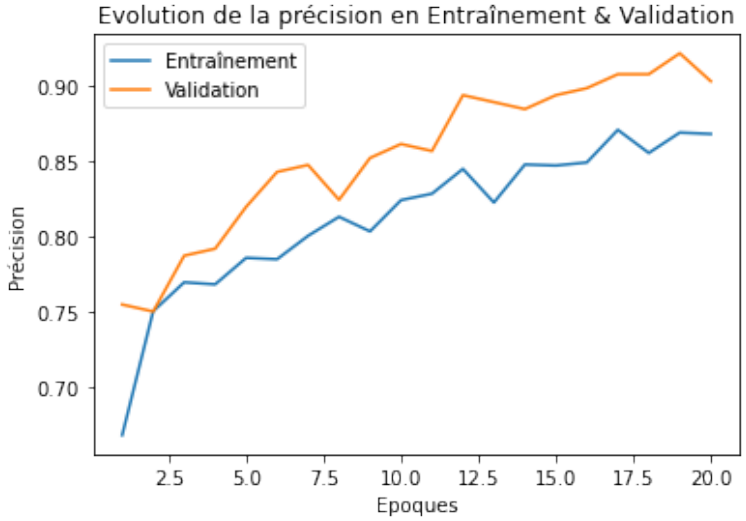

FIGURE 13 – Précision du modèle Resnet18 en PyTorch

3.3 Modèle Resnet18 avec fastai

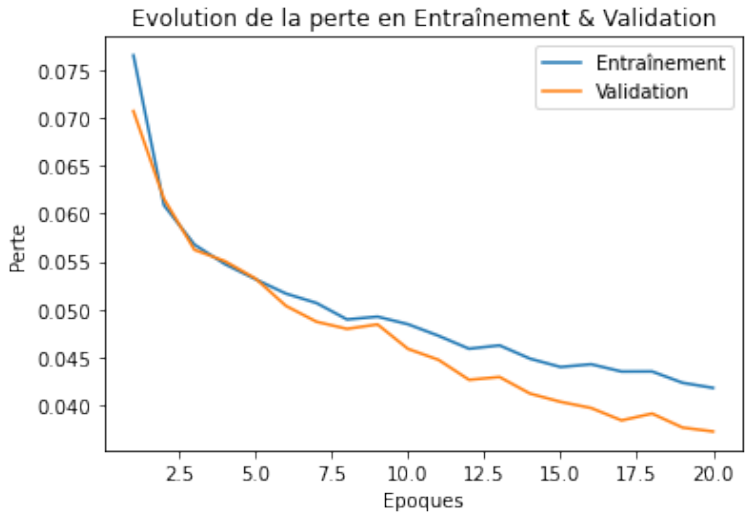

FIGURE 14 – Perte du modèle Resnet18 avec fastai

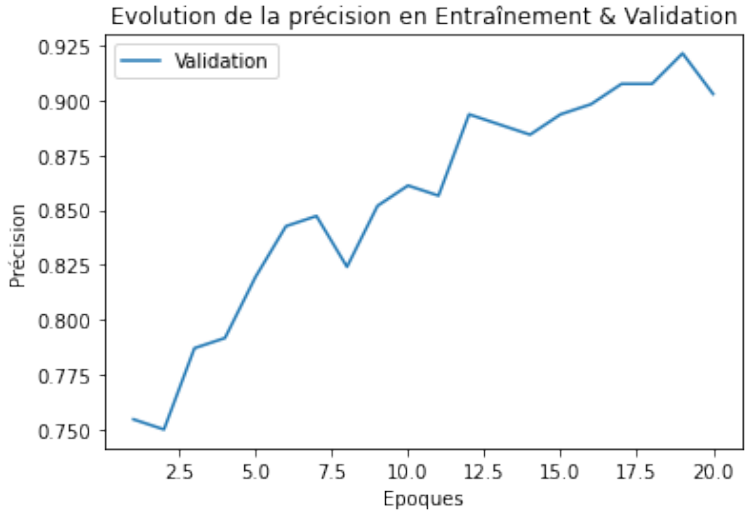

FIGURE 15 – Précision du modèle Resnet18 avec fastai

3.4 Interprétation des courbes

D'après les 3 courbes d'évolution de la perte, les modèles convergent tous. On observe que le premier modèle arrive dans un régime de sur-apprentissage car la fonction coût sur le jeu d'entraînement est plus petite que sur le jeu de validation. On le remarque aussi sur la précision. Mais le modèle apprend, ensuite on peut utiliser des techniques pour éviter le sur-apprentissage (Unpingco [2016]). Les deux modèles suivants semblent tous les deux en léger sous-apprentissage car le jeu de données de validation renvoie une meilleure

perte et une meilleure précision, il est donc possible de poursuivre l'entraînement pour améliorer les performances, d'autant plus que les courbes semblent encore décroître. Ainsi, les modèles issus du modèle Resnet18 donnent de meilleurs résultats et peuvent encore être optimisés tandis que le modèle simple à seulement 3 couches montre déjà certaines limites. Au niveau de la précision, le modèle fastai atteint 96% contre 92% pour le modèle Resnet18 en PyTorch et 90% pour le modèle simple (Figure 16). La librairie fastai contient de nombreuses options par défaut qui améliorent les performances sans avoir besoin de connaître les différentes techniques d'optimisation.

	3 couches PyTorch	Resnet18 PyTorch	Resnet18 fastai
Précision Entraînement	93%	87%	-
Précision Validation	90%	92%	96%

FIGURE 16 – Comparaison des précisions des modèles

Conclusion

L'objectif qui consistait à se familiariser avec les méthodes d'apprentissage profond pour de la classification d'images multi-label a été rempli. J'aurai aimé avoir le temps de plus manipuler les hyperparamètres pour analyser leur influence et chercher à optimiser les modèles afin notamment d'observer quelle précision peut atteindre chacun des modèles. Un autre approfondissement qui n'a pas pu être réalisé pour la même raison était de s'intéresser aux techniques de segmentation pour aboutir à une classification qui cette fois renvoie même la position de la classe détectée sur l'image. Cette technique demande des jeux de données plus riches puisqu'il faut fournir aux modèles les positions sur chaque image des classes à identifier. Il aurait sûrement été nécessaire de labelliser à la main ces données.

Ce stage a été une véritable chance de pouvoir enfin pratiquer le machine learning. En effet, je n'avais trouvé aucune entreprise acceptant de prendre en stage un étudiant pour seulement 3 mois en le faisant travailler sur de l'intelligence artificielle alors qu'il n'en a jamais fait auparavant. Cette période de stage s'est révélée extrêmement enrichissante en terme d'apprentissage théorique et pratique. Tout le travail effectué me servira pour la suite, je remercie d'ailleurs mon tuteur de ne pas m'avoir confié des tâches répétitives et peu intéressantes pédagogiquement telles que la labellisation de données. Il n'a pas été facile d'assimiler les notions de l'apprentissage automatique rapidement mais la rédaction de ce rapport m'a permis de prendre du recul sur ce que j'ai appris. De plus, je vais cette année dans le cadre de ma formation continuer d'apprendre l'apprentissage profond ce qui va consolider mes connaissances en diversifiant l'approche et le cadre d'apprentissage. Les librairies utilisées seront TensorFlow et Keras, les deux libraires concurrentes de PyTorch et fastai, j'aurai donc une vision étendue de ce qui se fait dans les différentes librairies. Ce stage me permet de témoigner d'une expérience en machine learning et augmente donc mes chances et ma confiance quant à l'obtention d'un projet de fin d'études sur le sujet. Je souhaite continuer d'apprendre le machine learning mais cette fois en cherchant à l'appliquer aux neurotechnologies. Il est plus difficile de trouver un stage quand il faut à la fois découvrir le domaine d'application et la technique utilisée. Avec cette expérience supplémentaire, je suis plus à même de me diriger vers ce domaine que je connais peu.

Références

F. Chollet. *Deep learning with Python.* Simon and Schuster, 2017.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016.

J. Howard and S. Gugger. *Deep Learning for Coders with Fastai and Pytorch : AI Applications Without a PhD.* O'Reilly Media, Incorporated, 2020. ISBN 9781492045526.

W. McKinney. *Python for Data Analysis.* O'Reilly Media, Inc., 2012. ISBN 9781449319793.

A. Müller and S. Guido. *Le machine learning avec Python : la bible des data scientists.* Paris : First interactive, 2018. ISBN 9782412034460.

A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

J. Unpingco. *Python for Probability, Statistics, and Machine Learning.* Springer International Publishing, 2016. ISBN 9783319307176.

A. Vannieuwenhuyze. *Intelligence artificielle vulgarisée : le machine learning et le deep learning par la pratique.* Editions ENI, 2019. ISBN 978240020735.

Annexes

A Code

A.1 Importations

```
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset
from torchvision import datasets
from torchvision.io import read_image
from torchvision.transforms import ToTensor, Lambda, Compose,
                                    Normalize, Resize
import matplotlib.pyplot as plt
import pandas as pd
from PIL import Image
from math import ceil
```

FIGURE 1 – Importations

A.2 Préparation des données

```
def read_dataframe(path):
    df_raw = pd.read_pickle(path)
    df = df_raw.drop_duplicates(subset=['fname']).reset_index(drop=
                                            True)
    return df

df = read_dataframe('df_total_large_image_update.pkl')
df.tail()
```

FIGURE 2 – Lire la trame de données avec Pandas

```
    classes = ['algues', 'rochers', 'sable']

    def label_encoder(dataframe):
        n = len(dataframe)
        labels_encoded = [[0. for i in range(3)] for i in range(n)]

        for i in range(len(dataframe)):
            labels = dataframe.iloc[i]['labels'].split(' ')
            for label in labels :
                for j, classe in enumerate(classes):
                    if label == classe :
                        labels_encoded[i][j] = 1.

        return labels_encoded
```

FIGURE 3 – Encodage des labels pour la classification multi-label

```
    str_to_int = {"algues":0,"rochers":1,"sable":2}

    def one_hot_label_encoder(dataframe):
        Y_onehot = []

        for i in range(len(dataframe)):
            labels = dataframe.iloc[i]['labels'].split(' ')
            labels_coded = torch.LongTensor([str_to_int[label] for label
                                                in labels])

            y_onehot = nn.functional.one_hot(labels_coded, num_classes=3
                                                )

            y_onehot = y_onehot.sum(dim=0).float()
            y_onehot = list(y_onehot.numpy())
            Y_onehot.append(y_onehot)

        return Y_onehot

    df['labels_encoded']= one_hot_label_encoder(df)
    df.head()
```

FIGURE 4 – Encodage des labels pour la classification multi-label

```
int_to_str = ['algues', 'rochers', 'sable']

def label_decoder(label_encoded):
    label_decoded = []
    for i in range(len(label_encoded)):
        if label_encoded[i]:
            label_decoded.append(int_to_str[i])
    return label_decoded

label_decoder(df.iloc[0]['labels_encoded'])

>>> ['algues', 'sable']
```

FIGURE 5 – Encodage des labels pour la classification multi-label

```
def splitter(df):
    train = df[df['is_valid']==False]
    valid = df[df['is_valid']==True]
    return train,valid

train_dataframe, test_dataframe = splitter(df)
```

FIGURE 6 – Division en données d'entraînement et de test ou de validation

```
    class CustomImageDataset(Dataset):
        def __init__(self, dataframe, img_dir, transform=None,
                                                target_transform=None):
            self.dataframe = dataframe
            self.img_dir = img_dir
            self.transform = transform
            self.target_transform = target_transform

        def __len__(self):
            return len(self.dataframe)

        def get_x(self,i):
            return self.img_dir+(self.dataframe.iloc[i]['fname']+'.png')

        def get_y(self,i):
            return torch.tensor(self.dataframe.iloc[i]['labels_encoded']
                                                    )

        def __getitem__(self, idx):
            image = Image.open(self.get_x(idx)).convert('RGB')
            label = self.get_y(idx)

            if self.transform:
                image = self.transform(image)
            if self.target_transform:
                label = self.target_transform(label)

            return image, label

    train_dataset = CustomImageDataset(train_dataframe,'images_total/',
                                            transform)
    image , label = train_dataset[0]
    image.shape, label
    >>> (torch.Size([3, 28, 28]), tensor([1., 0., 1.]))
```

FIGURE 7 – Création du *Dataset*

```
size = (28, 28)
transform = Compose([Resize(size), ToTensor()])
dataset = CustomImageDataset(df,'images_total/',transform)

imgs = torch.stack([img_t for img_t, _ in dataset], dim=3)
df_mean = imgs.view(3,-1).mean(dim=1)
df_std = imgs.view(3,-1).std(dim=1)

print(f'mean:{df_mean}')
print(f'std:{df_std}')
>>> mean:tensor([0.6011, 0.6520, 0.5668])
>>> std:tensor([0.1331, 0.1011, 0.0971])

transform = Compose([Resize(size),ToTensor(),Normalize(df_mean,
                                        df_std)])
train_dataset = CustomImageDataset(train_dataframe,'images_total/',
                                        transform)
valid_dataset = CustomImageDataset(valid_dataframe,'images_total/',
                                        transform)
print(f'train: {len(train_dataset)}, valid:{len(valid_dataset)}')
>>> train: 293, valid:66
```

FIGURE 8 – Transformations

```
train_dataloader = DataLoader(train_dataset, batch_size=64,shuffle=
                                        True)
test_dataloader = DataLoader(test_dataset, batch_size=64, shuffle=
                                        True)

def plot_one_image(dataloader):
    features, labels = next(iter(dataloader))
    print(f"Feature batch shape: {features.size()}")
    print(f"Labels batch shape: {labels.size()}")

    img_size = features[0].shape[1], features[0].shape[2]
    img = std.diag()@features[0].view(3,-1) + mean.diag()@torch.ones
                                        ((3,img_size[0]*img_size[1])
                                        )
    img = img.reshape((3,img_size[0],img_size[1]))

    label = label_decoder(labels[0])
    plt.imshow(img.permute(1, 2, 0))
    plt.show()
    print(f"Label: {label}")

plot_one_image(train_dataloader)
>>> Feature batch shape: torch.Size([8, 3, 64, 64])
>>> Labels batch shape: torch.Size([8, 3])
```

FIGURE 9 – Insertion dans un *DataLoader*

A.3 Définition du modèle

```
model_resnet = models.resnet18(pretrained=True)

for param in model_resnet.parameters():
    param.requires_grad = False

n_inputs = model_resnet.fc.in_features
n_classes = len(classes)
model_resnet.fc = nn.Linear(n_inputs, n_classes)

device = "cuda" if torch.cuda.is_available() else "cpu"
model_resnet = model_resnet.to(device)
```

FIGURE 10 – Importer un modèle

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(3*28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 3),
            nn.ReLU()
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits

model_simple = NeuralNetwork().to(device)
```

FIGURE 11 – Création d'un modèle

A.4 Optimisation des paramètres

```
loss_fn = nn.BCEWithLogitsLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=1e-3)

def accuracy_multi(inp, targ, thresh=0.5, sigmoid=True):
    if sigmoid: inp = inp.sigmoid()
    return ((inp>thresh)==targ.bool()).float().mean()
```

FIGURE 12 – Définition des métriques et de la fonction perte

```python
def train_and_valid(model=model,
          optimizer=optimizer,
          loss_fn=loss_fn,
          train_loader=train_dataloader,
          val_loader=valid_dataloader,
          epochs=20,
          device=device):

    len_data_train = len(train_loader.dataset)
    len_data_val = len(val_loader.dataset)
    nb_train = (len_data_train / train_loader.batch_size)
    nb_val = (len_data_val / val_loader.batch_size)
    train_loss_epoch = []
    valid_loss_epoch = []
    train_acc_epoch =[]
    valid_acc_epoch = []
    for epoch in range(epochs):
        training_loss_batch, validation_loss_batch = 0.0, 0.0
        training_acc_batch = 0.0
        validation_acc_batch = 0.0
        model.train()
        for ix, batch in enumerate(train_loader):
            optimizer.zero_grad()
            X,y = batch
            X, y = X.to(device), y.to(device)
            output = model(X)
            loss = loss_fn(output,y)
            loss.backward()
            optimizer.step()
            training_loss_batch += loss.data.item()
            training_acc_batch += (accuracy_multi(output, y))
        train_loss_epoch.append(training_loss_batch/len_data_train)
        train_acc_epoch.append(training_acc_batch/float(ix +1))

        model.eval()
        for ix, batch in enumerate(val_loader):
            X,y = batch
            X, y = X.to(device), y.to(device)
            output = model(X)
            loss = loss_fn(output,y)
            validation_loss_batch += loss.data.item()
            validation_acc_batch +=(accuracy_multi(output, y))
        valid_loss_epoch.append(validation_loss_batch/len_data_val)
        valid_acc_epoch.append(validation_acc_batch/float(ix+1))

        print(f'Epoch:{epoch}, Train_loss:{train_loss_epoch[epoch]:.3f},
                                          Val_loss:{valid_loss_epoch[
                                          epoch]:.3f}')
        print(f'train_acc:{train_acc_epoch[epoch]:3f}, val_acc:{
                                          valid_acc_epoch[epoch]:2f}')
    return train_loss_epoch, valid_loss_epoch, train_acc_epoch,
                                        valid_acc_epoch

train_loss_epoch, valid_loss_epoch, train_acc_epoch, valid_acc_epoch =
                                    train_and_valid()
```

FIGURE 13 – Entraînement et validation du modèle

B Rapport d'évaluation

ENSTA Bretagne

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* Ensta Bretagne Lab-STICC

Adresse / *Address* Ensta 2 rue François Verny 29806 Brest

Tél / *Phone (including country and area code)* 02 98 34 87 70

Nom du superviseur / *Name of internship supervisor* Lepers Benjamin

Fonction / *Function* Ingénieur de recherche.

Adresse e-mail / *E-mail address* benjamin.lepers@ensta-bretagne.fr

Nom du stagiaire accueilli / *Name of intern* Hugo Sabatier

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
*Please attribute a mark from **A (excellent)** to **F (very weak)**.*

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ? (A) B C D E F
  Was the initial contract carried out to your satisfaction?

- Manquait-il au stagiaire des connaissances ? ☐ oui/*yes* ☒ non/*no*
  Was the intern lacking skills?

  Si oui, lesquelles ? / *If so, which skills?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)*

  (A) B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* stage en partie en télétravail (covid 19)

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?
Did the intern live up to expectations? (Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?

(A) B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____
_____

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? (A) B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? (A) B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____
_____

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? (A) B C D E F
Was the intern open to listening and expressing himself /herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* _____
_____

OPINION GLOBALE / *OVERALL ASSESSMENT*

- La valeur technique du stagiaire était : (A) B C D E F
  Please evaluate the technical skills of the intern:

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

- Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

  Would you be willing to host another intern next year? ☒ oui/*yes* ☐ non/*no*

Fait à Brest, le 29/09/2021
In _____, on _____

Signature Entreprise _____ Signature stagiaire
Company stamp _____ *Intern's signature*

Merci pour votre coopération
We thank you very much for your cooperation