Rapport Final
Data Science Starter Program

Screening and Diagnosis of esophageal cancer from in-vivo microscopy images by

Sofiane ALLA

Sommaire

1. Introduction et motivation pour le projet

Le projet proposé est issu de la plateforme ENS-Collège de France, sur la base d'un challenge proposé par l'entreprise *Mauna Kea Technologies*.

Mauna Kea Technologies développe et conçoit des systèmes **d'imagerie laser** en utilisant des **sondes à fibres optiques.** En particulier, les sondes ainsi utilisées permettent de réaliser de l'imagerie cellulaire *in vivo* de haute précision, permettant d'accélérer la prise en charge et le diagnostic de maladies liées au système digestif.

L'objectif principal de ce projet est la mise en place d'un **algorithme de classification** de cellules de l'œsophage *in-vivo,* en fonction de la **texture observée**. Les cellules sont issues de patients souffrant du syndrome de l'œsophage de Barrett, un état prédisposant au cancer de l'œsophage.

Un algorithme de ce type, associé à une endoscopie permettrait de **réduire considérablement la phase de diagnostic** du cancer de l'œsophage.

En algorithme de comparaison, *Mauna Kea Technologies* s'est basé sur un réseau de neurones CNN à 3 blocs, chacuns suivie d'une phase de *dropout* et de *pooling.* La technique utilisée permet d'aboutir à une précision de **75% sur le jeu de test**.

Mon choix s'est porté sur ce challenge pour trois raisons principales :

- **La structuration du jeu de données** comportant uniquement (i) des images brutes et (ii) un fichier en format .csv répertoriant les labels. Cette configuration se rapproche fortement de ce que l'on pourrait avoir en **conditions réelles** ;

- **La possibilité d'appliquer les concepts vus en cours** concernant le *pre-processing* des images et les méthodes les plus avancées impliquant des réseaux de neurones pré-entrainés.

- L'opportunité de se familiariser avec l'architecture de **Pytorch,** qui, à posteriori, ouvre de **nombreuses possibilités** et s'est avérée très utile pour un projet de recherche.

Le projet est essentiellement un projet de ***deep learning*** : l'objectif est avant tout de juger de l'efficacité des classifieurs utilisés par rapport au *benchmark* proposé (ac*curacy* : 78%), ou un réseau plus classique non pré-entrainé mais se basant sur des techniques d'augmentation de données (92%).

Afin d'approfondir mes connaissances et découvrir des techniques de **vision par ordinateur** pouvant s'avérer utiles, je me suis basé sur une spécialisation proposée par Coursera : *AI for Medicine* et un TD : *detecting Covid-19 using Pytorch.*

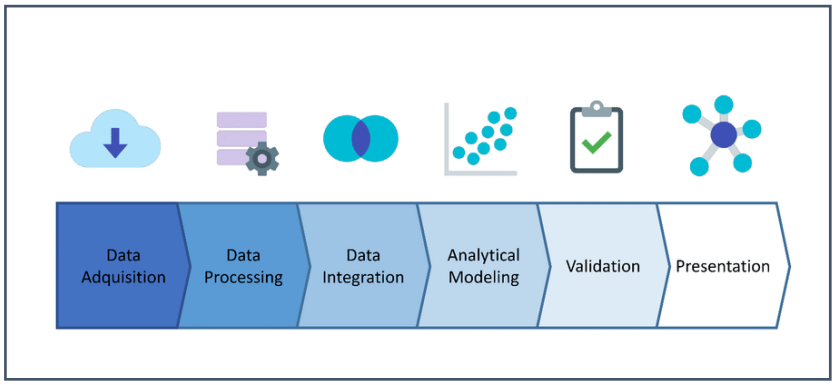
A posteriori, mener à bien ce projet de classification a nécessité la gestion de 3 phases critiques :

- La **construction du jeu de données**, grâce à la classe *Custom Dataset* de **Torch**, permettant d'associer chaque image au label correspondant ;

- La **gestion de la ressource** : étant donné la taille du *dataset* (+ 3gb), entrainer les réseaux de neurones utilisés à rendu nécessaire l'utilisation de Google Colab et de son GPU (Tesla T4). Le GPU a fortement accéléré la capacité à entrainer et ajuster les hyper paramètres.

- La **phase d'entrainement,** qui peut s'avérer très délicate en vision par ordinateur.

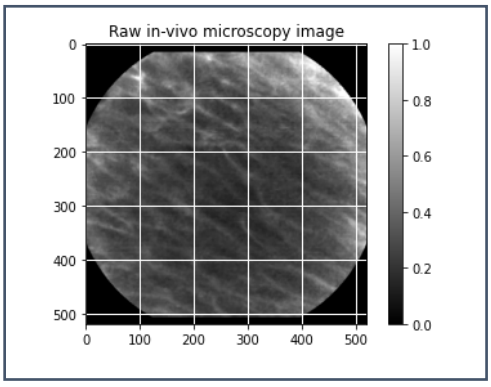
La structuration du projet a suivi la procédure classique d'un projet en sciences de la donnée :

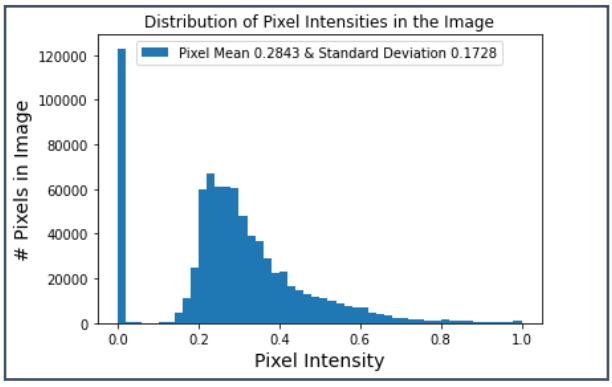


2. Description du jeu de données

Le jeu de données d'entrainement est constitué de **9 446 images classées selon 4 catégories** :

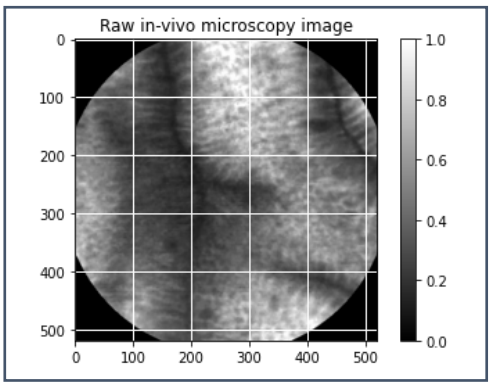
- ***Squamous epithelium*** (épithelium malpighien / label : 0) : texture en forme **d'écailles de poisson,** venant d'un individu sain. Il est possible d'obtenir la distribution de pixels à partir de chaque exemple :

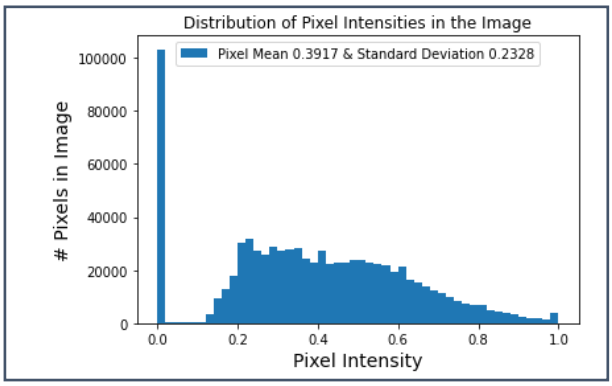




La distribution des pixels montre que l'image (en noir et blanc, comme l'ensemble du jeu de données) est **peu contrastée**, **sombre** et homogène.

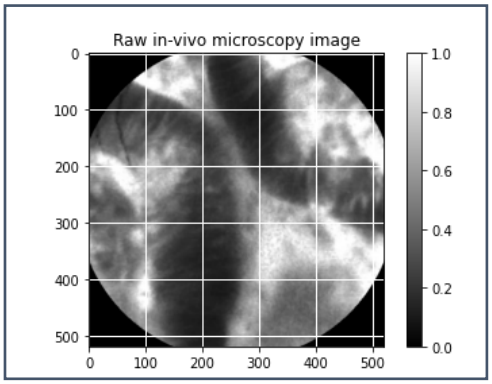
- ***Intestinal_metaplasia*** (mataplasie intestinal / label : **1**) : condition prédisposante au cancer de l'œsophage, avec la présence de ronds noirs (d'une taille de +/- 10 microns à l'intérieurs glandes clairement délimitées.

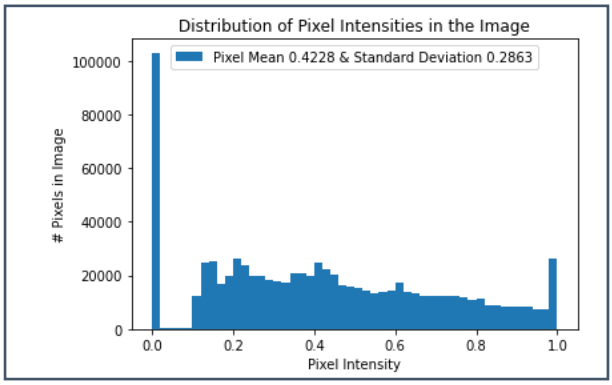




La distribution de pixels montre des images **contrastées** et **claires** par rapport aux textures de type écailles de poisson.

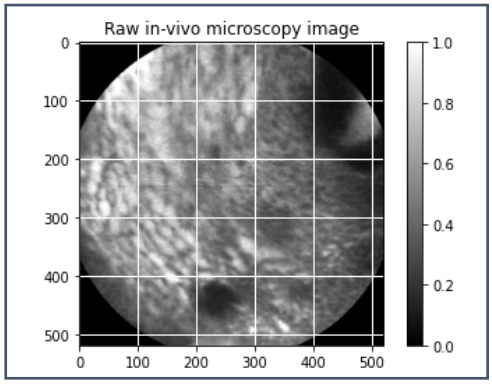
- ***Gastric metaplasia*** (métaplasie gastrique / label : **2**) : condition également prédisposante au cancer de l'œsophage, les ronds noirs sont plus difficiles à discerner et les glandes plus irrégulières.

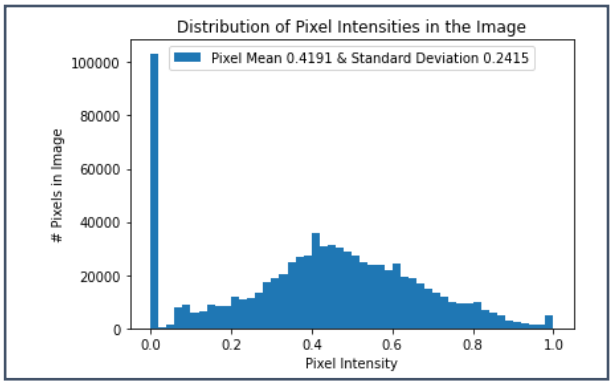




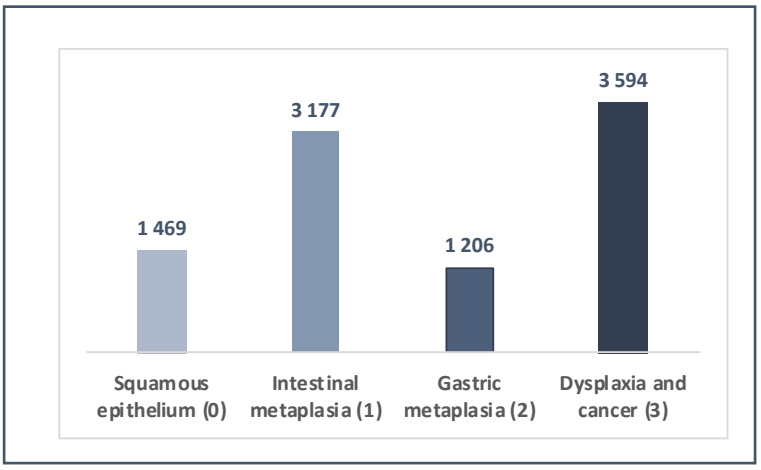
La distribution de pixels est similaire à celle de la métaplasie intestinale, légèrement plus contrastée et plus claire.

- ***Dysplasia_and_Cancer*** (dysplasie et cancer / label : **3**) stade **pré-cancéreux ou cancéreux** nécessitant une ablation par chirurgie. Les glandes ne sont plus régulières et les zones très brillantes de plus en plus fréquentes.





Au global, toutes les images ont la même dimension de **519 x 521 pixels**. Au sein du jeu de données d'entrainement, la distribution des catégories est **déséquilibrée** :



La présence moins marquée de la classe zéro (*Squamous epithelium*) d'individus sains s'explique par la procédure mise en place : seuls les sujets présentant le syndrome de l'œsophage de Barett font partie du jeu de données. Au total, les images proviennent de **61 patients, dont 44 pour le jeu d'entrainement.**

3. *Pre-processing* et augmentation de données

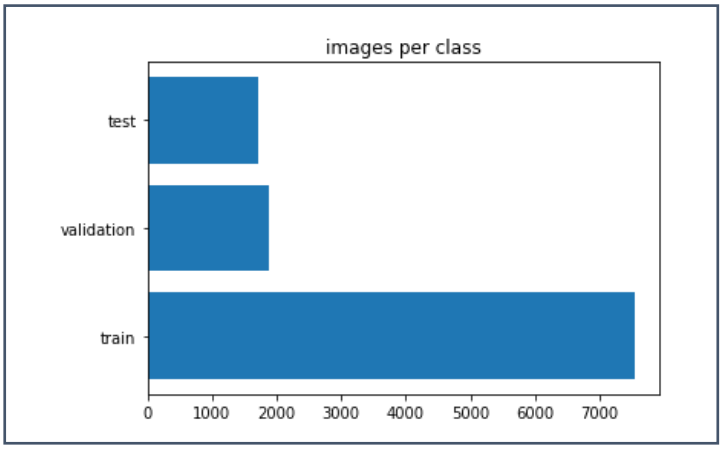
a. Division du jeu d'entrainement en jeu d'entrainement / validation

Une des premières étapes du *pre-processing* consiste à diviser le jeu d'entrainement en utilisant la fonction **train_test_split** de *scikit-learn.*

Important : une autre possibilité consistait à diviser le jeu de données après la construction du *Custom Dataset* et du *Dataloader* en utilisant la **fonction random_split de Pytorch**.

Toutefois, diviser le jeu de données à partir du fichier .csv est (i) beaucoup **plus simple en termes de déploiement** et (ii) permet de prendre en compte le **déséquilibre de catégories** grâce à l'option *stratify.* L'option *stratify* de la fonction **train_test_split** permet d'obtenir un jeu de validation aux distributions de catégories similaires. Au final, nous obtenons la répartition suivante du jeu de données :

```
There are 7556 total train images.
There are 1890 total validation images.
There are 1715 total test images.
```



b. Augmentation de données

L'augmentation de données est une méthode classique utilisée en vision par ordinateur et permettant d'appliquer des transformations aux images afin **d'améliorer la précision des classifieurs.**

Sur la base de l'API Torch, 3 types de transformations ont été appliquées afin d'utiliser, d'améliorer la pertinence et l'efficacité des classifieurs :

- **Transformation** en format *Tensor*, est une transformation classique permettant de convertir des images PIL ou Numpy en format tenseur. Une autre possibilité, équivalente, aurait été d'intégrer cette transformation directement au sein du *Custom Dataset.*

- **Redimensionnement** des images en format **224x224 pixels** et **normalisation** de la moyenne (0.485, 0.456, 0.406) et de l'écart-type (0.229, 0.224, 0.225) : ces transformations sont essentielles pour **utiliser des réseaux de neurones pré-entrainés sur la base de données ImageNet**. ImageNet est une base de données annotée manuellement de plus de 14 millions d'images, divisée en près de 20 000 catégories différentes et constitue la base de pré-entrainement de nombreux réseaux de neurones, dont ceux utilisés pour la présente étude (Resnet et VGG).

- **Transformations aléatoires** de type *random horizontal flip :* cette transformation consiste à pivoter horizontalement une image suivant une probabilité donnée (par défaut 0.1)

Attention : il est important de **ne pas répliquer les transformation aléatoires** (randomCrop et randomHorizontal Flip notamment) dans les jeux de validation et de test afin de ne pas les biaiser.

c. Préparation du jeu de données (*custom dataset and dataloader*) et visualisation

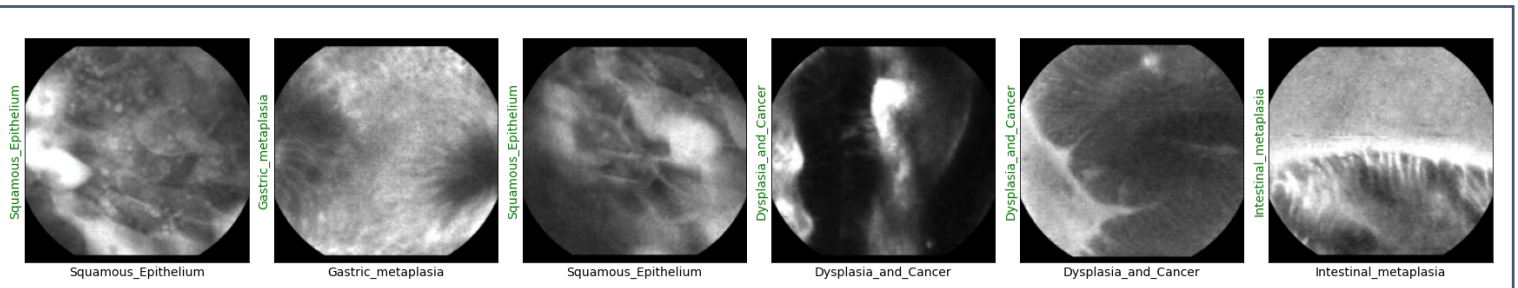
Un des aspects essentiels de l'utilisation de Pytorch est la possibilité de créer des **jeux de données personnalisés**, permettant d'associer chaque image à sa catégorie correspondante sur le fichier csv. Des transformations de base ont été appliquées (conversion de l'index en format *string* et des labels en format *float64*) directement sur le fichier .csv.

Sous Pyorch, le Dataset est une classe représentant le jeu de données à laquelle nous pousons faire appel en utilisant la fonction **torch.utils.data.Dataset**.

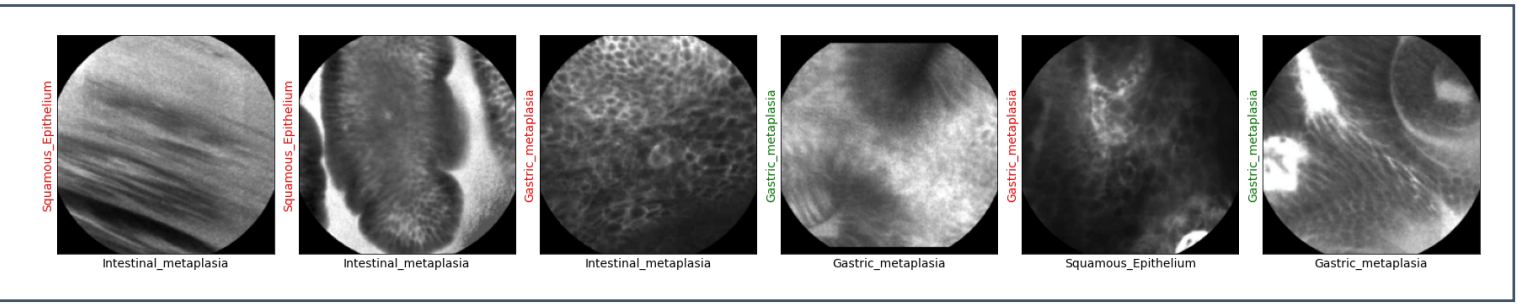
Dans notre cas précis, la classe construite contient trois fonctions :

- **__init__() :** associe le dossier contenant les images avec le fichier .csv contenant les différentes catégories ;
- **__len__() :** fonction retournant la longueur du jeu de données ;
- **__getitem__() :** fonction permettant de retourner un exemple du jeu de données.

Le *Custom Dataset* sous Torch permet à de faire appel au **DataLoader**, une fonction sous Torch permettant d'itérer sur le jeu de données en utilisant des lots groupés d'images (*batch*) :



La visualisation proposée ci-dessous est essentielle, car elle nous permettra de suivre visuellement l'entrainement des classifieurs. En abscisse apparaitra la catégorie à laquelle appartient l'image et issue du fichier .csv. En ordonnée apparait la **catégorie prédite : en vert si la catégorie est juste, en rouge si elle est fausse :**

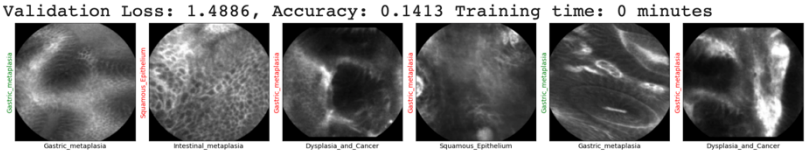


4. Phase d'entrainement

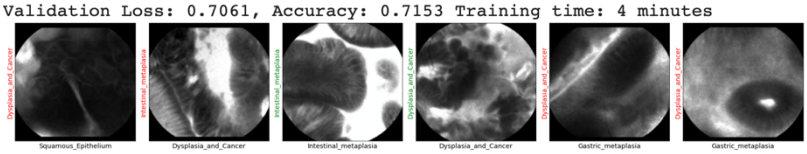
L'algorithme d'entrainement utilisé est issu du notebook *Detecting COVID-19 with Chest X-Ray using PyTorch.* L'algorithme d'entrainement utilisé possède quelques caractéristiques intéressantes :

En particulier, l'algorithme a un **aspect visuel et pratique,** permettant de suivre en temps réel l'évolution de la phase d'entrainement. **Toutes les 100 à 120 itérations** (adaptées en fonction du classifieur utilisé), **la phase d'évaluation du modèle est déclenchée** et les paramètres du modèle sont ajustés, ce qui s'adapte bien à la taille du jeu de données utilisé.

```
Start training..
====================
Starting epoch 1/5
====================
Evaluating at step 0
Validation Loss: 1.4886, Accuracy: 0.1413 Training time: 0 minutes
```



```
Evaluating at step 120
Validation Loss: 0.7061, Accuracy: 0.7153 Training time: 4 minutes
```



```
Evaluating at step 240
Validation Loss: 0.4637, Accuracy: 0.8434 Training time: 9 minutes
```

5. Classifieurs : description des méthodes utilisées et principaux résultats

Les classifieurs présentés dans cette étude ont été sélectionnés selon trois critères principaux :

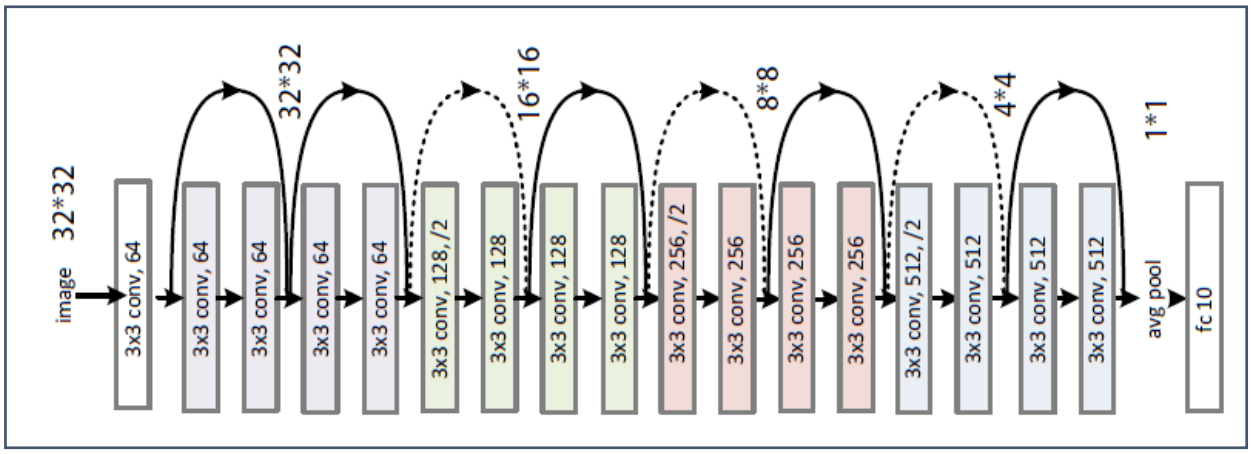
- La capacité à **surperformer l'accuracy** du benchmark, qui est de 78% ;
- La capacité à **faire levier sur l'ensemble du pipeline** construit précédemment sous Torch, incluant les phases de pré-traitement, d'augmentation de données et de construction du *Custom Dataset* et du *Dataloader.*

Deux des trois classifieurs choisis (Resnet18 et VGG16) reposent sur une technique de *transfer learning* : l'apprentissage par transfert repose sur le transfert de *« la connaissance acquise sur un jeu de données "source" pour mieux traiter un nouveau jeu de données dit "cible". »* (https://dataanalyticspost.com/Lexique/transfer-learning/). Dans notre cas, Resnet et VGG sont pré-entrainés sur la base **ImageNe**t afin d'ajuster les paramètres du modèle.

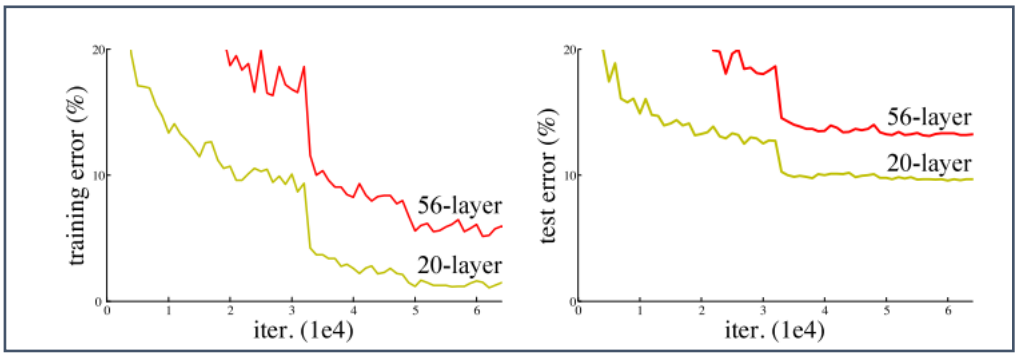
Seule la couche supérieure de ces modèles est **ajustée à notre base de données**. Cette technique s'est révélée être efficace et est connue pour **générer des performances supérieures aux modèles classiques.**

Le dernier classifieur visait à répliquer celui utilisé pour le benchmark : un réseau de neurones convolutionnel à trois blocs.

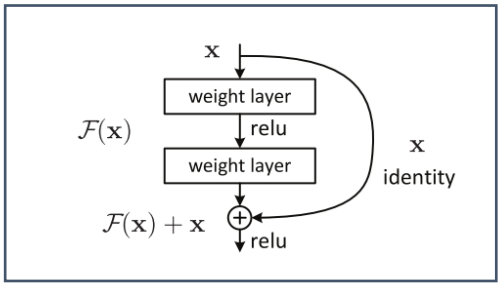
a. Réseau de neurones Resnet-18



Le premier classifieur utilisé est un **réseau de neurones résiduel ou Resnet**. Une des limitations principales des réseaux de neurones traditionnels est la perte de gradient : plus le nombre de couches est élevé, plus il est difficile d'entrainer un réseau car le gradient devient infiniment petit et les performances dégradées (https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035) :

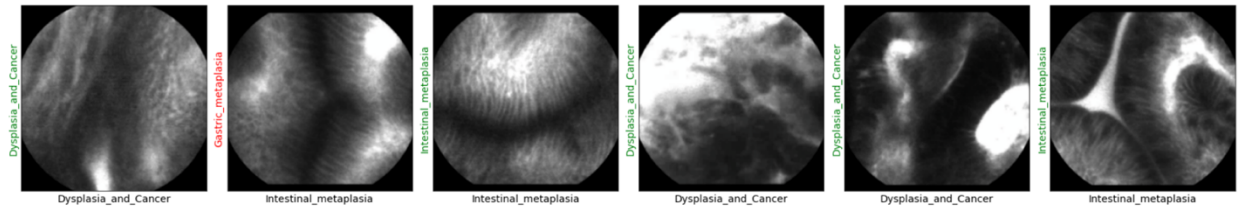


Le modèle Resnet permet de contourner cette limitation en passant une ou plusieurs couches selon le modèle suivant :



Le Resnet utilisé contient **11,6 millions** de paramètres qui seront **entrainés** et **11,6 millions qui seront figés.** La dernière couche a été ajustée afin d'être adaptée à une classification à 4 labels. En utilisant un critère de type *cross-entropy loss* et un optimiseur ADAM (*learning rate* : 0,0001) sur des lots de 6, nous obtenons d'excellents résultats :

```
Evaluating at step 1200
Validation Loss: 0.0909, Accuracy: 0.9772 Training time: 60 minutes
```

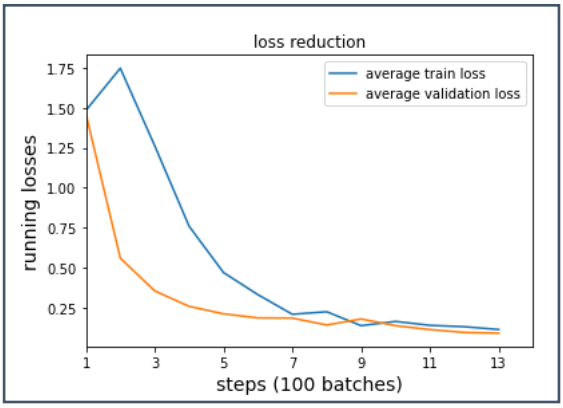



```
Performance condition satisfied, stopping..
CPU times: user 58min 51s, sys: 7min 53s, total: 1h 6min 44s
Wall time: 1h 51s
```

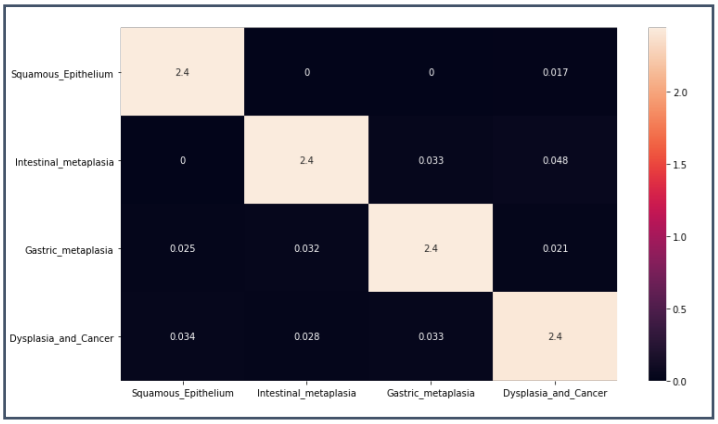
Nous aboutissons à une **précision de 98% en une seule itération sur l'ensemble de jeu de données,** et ce sans *overfitting* (cf. courbe de perte).

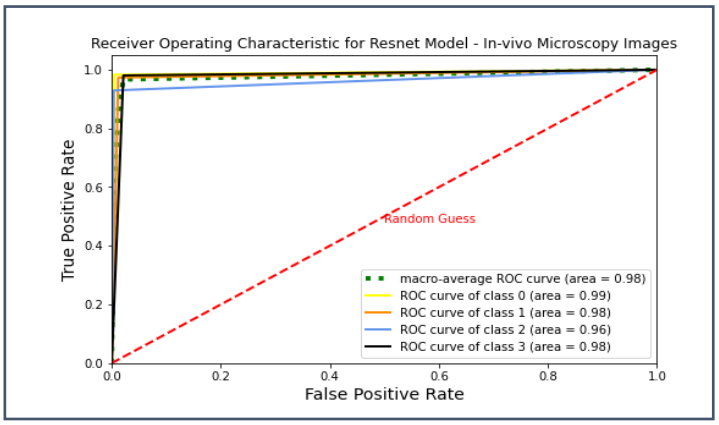
```
calculating model accuracy after training....
Test Accuracy on Squamous_Epithelium:  98%       (288/294)
Test Accuracy on Intestinal_metaplasia:  97%       (616/636)
Test Accuracy on Gastric_metaplasia:  97%       (233/241)
Test Accuracy on Dysplasia_and_Cancer:  99%       (710/719)

Test Accuracy (Overall):  98%      (1847/1890)
```

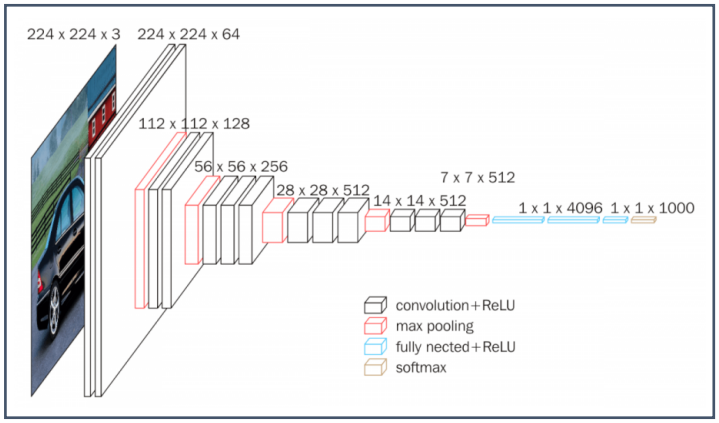


La matrice de confusion du modèle Resnet ainsi que **la courbe ROC** (*receiver operating characteristic*) donnent également des indications sur la performance encourageante du modèle.





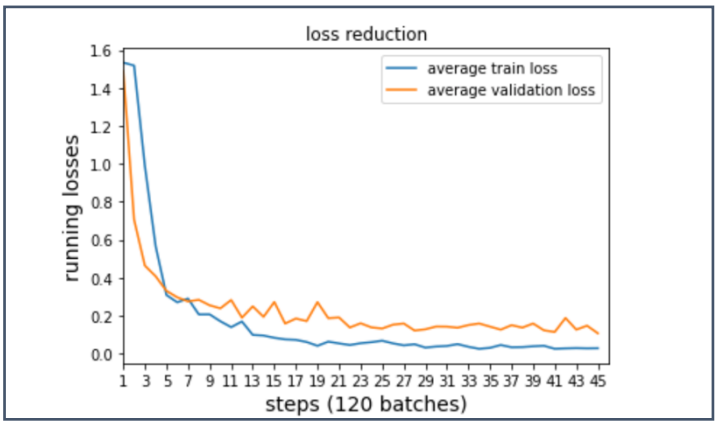
b. Réseau de VGG-16



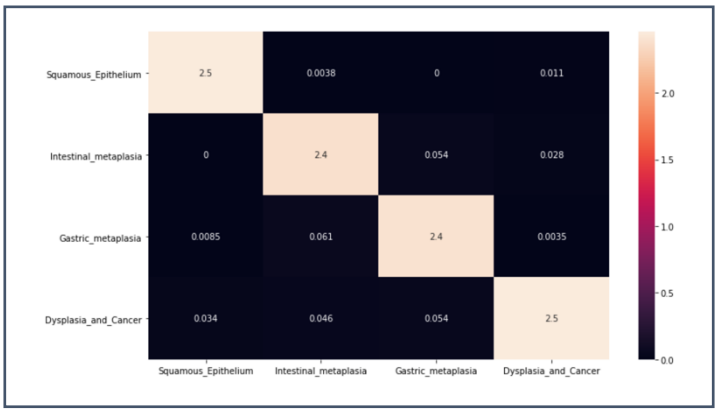
VGG16 est une architecture de réseau de neurones crée K. Simonyan et A. Zisserman de l'université d'Oxford, qui a acquis sa notoriété en obtenant d'excellents résultats sur la **base de données ImageNet.** Le principal apport de ce réseau de neurones est l'utilisation de convolutions de plus petite dimension (3x3). L'algorithme d'optimisation (ADAM) et le critère de différentiation sont similaires à ceux utilisés pour le Resnet.

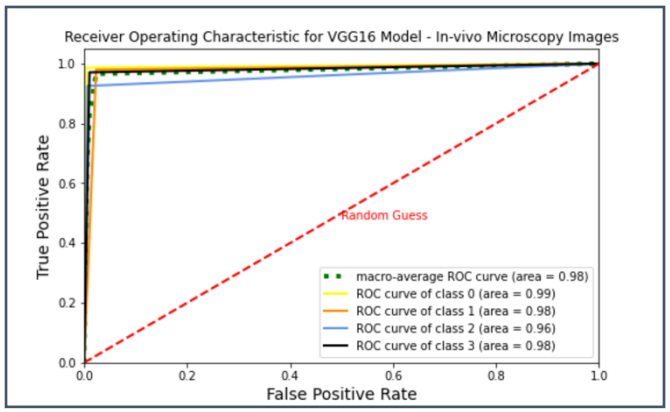
Si l'*accuracy* des réseaux Resnet18 et VGG16 est *in fine* assez similaire **avec une légère surperformance du réseau Resnet**, le réseau VGG s'est révélé **beaucoup plus lourd à entrainer** et moins efficace pour notre jeu de données. Cela se voit notamment dans les courbes de *training loss* et *validation loss* du modèle :

	precision	recall	f1-score	support
0	0.98	0.99	0.98	294
1	0.96	0.98	0.97	636
2	0.96	0.93	0.94	241
3	0.98	0.97	0.98	719
accuracy			0.97	1890
macro avg	0.97	0.97	0.97	1890
weighted avg	0.97	0.97	0.97	1890



La matrice de confusion du modèle VGG ainsi que **la courbe ROC** (*receiver operating characteristic*) donnent des résultats très satisfaisants :





c. Réseau de neurones convolutionnel à 3 blocs

Afin d'avoir un élément de comparaison par rapport aux réseaux de neurones pré-entrainés, un CNN à 3 blocs a été utilisé, chaque bloc étant constitué de 2 couches convolutionnelles et une couche de *max-pooling*. Les hyperparamètres restent similaires à ceux utilisés : ***learning rate* de 0,0001, lots de 6, critère de sélection de type *Cross Entropy Loss* et optimiseur ADAM.**

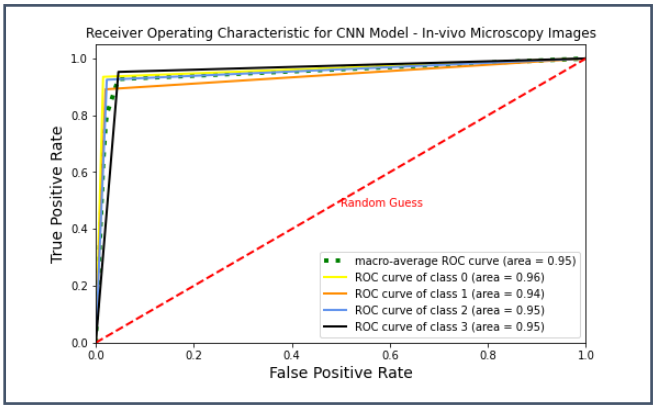
Layer (type)	Output Shape	Param #	Tr. Param #
Conv2d-1	[1, 32, 224, 224]	896	896
ReLU-2	[1, 32, 224, 224]	0	0
Conv2d-3	[1, 64, 224, 224]	18,496	18,496
ReLU-4	[1, 64, 224, 224]	0	0
MaxPool2d-5	[1, 64, 112, 112]	0	0
Conv2d-6	[1, 128, 112, 112]	73,856	73,856
ReLU-7	[1, 128, 112, 112]	0	0
Conv2d-8	[1, 128, 112, 112]	147,584	147,584
ReLU-9	[1, 128, 112, 112]	0	0
MaxPool2d-10	[1, 128, 56, 56]	0	0
Conv2d-11	[1, 256, 56, 56]	295,168	295,168
ReLU-12	[1, 256, 56, 56]	0	0
Conv2d-13	[1, 256, 56, 56]	590,080	590,080
ReLU-14	[1, 256, 56, 56]	0	0
MaxPool2d-15	[1, 256, 28, 28]	0	0
Flatten-16	[1, 200704]	0	0
Linear-17	[1, 1024]	205,521,920	205,521,920
ReLU-18	[1, 1024]	0	0
Linear-19	[1, 512]	524,800	524,800
ReLU-20	[1, 512]	0	0
Linear-21	[1, 4]	2,052	2,052

Total params: 207,174,852
Trainable params: 207,174,852
Non-trainable params: 0

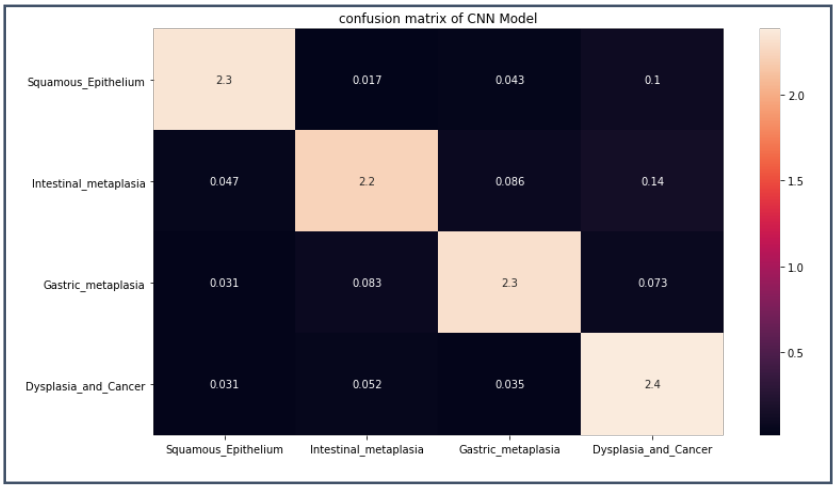
Après 10 itérations complètes sur le jeu de données d'entrainement (*epochs*), nous obtenons **une *accuracy* de 93%, soit inférieure de l'ordre de 5% par rapport aux deux classifieurs pré-entrainés testés (Resnet et VGG16).**

```
calculating model accuracy after training....
Test Accuracy on Squamous_Epithelium:  94%       (275/294)
Test Accuracy on Intestinal_metaplasia:  89%       (567/636)
Test Accuracy on Gastric_metaplasia:  93%       (223/241)
Test Accuracy on Dysplasia_and_Cancer:  95%       (685/719)

Test Accuracy (Overall):  93%      (1750/1890)
```



La matrice de confusion montre une **confusion légèrement supérieure** entre la catégorie **dysplaxie et cancer** et les catégories de métaplasie (gastriques et intestinales) :



6. Conclusion

Dans la pratique, les performances dans la classification de cellules *in vivo* obtenues grâce à des réseaux pré-entrainés sont très encourageantes. Pour aller plus loin, ce type d'algorithmes permet à *Mauna Kea Technologies* de développer **des outils efficaces d'aide à la décision pour la détection et le diagnostic du cancer de l'œsophage.**

En juin 2021, *Mauna Kea Technologies* a annoncé le lancement d'une plateforme unique de nouvelle génération, appelée *Cellvizio*, permettant aux médecins d'accéder à une plateforme tout-en-un de visualisation cellulaire et moléculaire et **l'interprétation assistée des images grâce à l'Intelligence artificielle.**

Pour mener à bien ce projet, la formation DSSP a été utile à plusieurs titres :

- L'utilisation de la livraison **sci-kit learn** (notamment la partie dispensée par Alexandre Gramfort) qui fut très utile pour (i) l'analyse exploratoire des données, (ii) les transformations réalisées dans le fichier .csv et (iii) l'analyse de la performance, notamment la création des matrices de confusion.

- La partie dédiée au **deep learning** en fin de formation m'a permis de me familiariser avec Keras et la gestion des hyperparamètres.

- Plus généralement, les conseils pratiques de méthode dans la **gestion d'un projet de sciences de la donnée et de chaque étape, et l'analyse critique des résultats obtenus.**

7. Bibliographie et ressources utiles

1. Mauna Kea Technologies. https://www.maunakeatech.com/fr/

2. Œsophage de Barrett. https://www.medtronic.com/ca-fr/votre-sante/troubles-medicaux/reflux-disease/what-is-barretts.html

3. Image augmentation for Deep Learning Using Pytorch.
https://www.analyticsvidhya.com/blog/2019/12/image-augmentation-deep-learning-pytorch/

4. DataLoader et Custom Dataset sous Pytorch.
https://pytorch.org/tutorials/beginner/basics/data_tutorial.html

5. Training a classifier with PyTorch.
https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

6. Coursera. Detecting COVID-19 with Chest X-Ray using PyTorch.
https://www.coursera.org/projects/covid-19-detection-x-ray

7. CNN Model for Pytorch classification. https://medium.com/thecyphy/train-cnn-model-with-pytorch-21dafb918f48

8. VGG16 Architecture. https://neurohive.io/en/popular-networks/vgg16/

9. An Overview of ResNet and its Variants. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

10. Cellvizio. https://www.maunakeatech.com/fr/actualites/200-mauna-kea-technologies-unveils-its-next-generation-cellvizio-platform