UNET

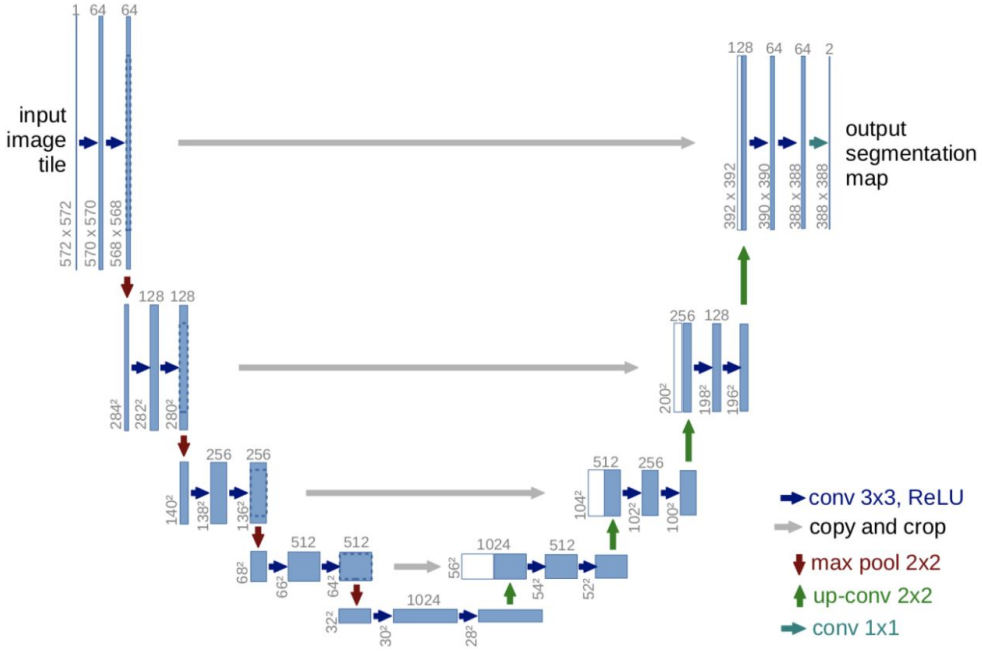
U-Net est un réseau de neurones à convolution développé pour la segmentation d'images biomédicales au département d'informatique de l'université de Fribourg en Allemagne. Le réseau est basé sur le réseau entièrement convolutionnel (Fully Connected Network) et son architecture a été modifiée et étendue pour fonctionner avec moins d'images d'entraînement et pour permettre une segmentation plus précise. La segmentation d'une image 512 * 512 prend moins d'une seconde sur un GPU récent.

UNet, issu du réseau neuronal convolutif traditionnel, a été conçu et appliqué pour la première fois en 2015 pour traiter des images biomédicales. Un réseau neuronal convolutif général concentre sa tâche sur la classification d'images, où l'entrée est une image et la sortie est une étiquette, mais dans les cas biomédicaux, nous sommes obligés non seulement de distinguer s'il y a une maladie, mais aussi de localiser la zone d'anomalie

UNet est dédié à résoudre ce problème. La raison pour laquelle il est capable de localiser et de distinguer les bordures est en effectuant une classification sur chaque pixel, de sorte que l'entrée et la sortie partagent la même taille..

Présentation générale de UNET

Le schéma ci-dessous présente le fonctionnement général de l'architecture UNET.



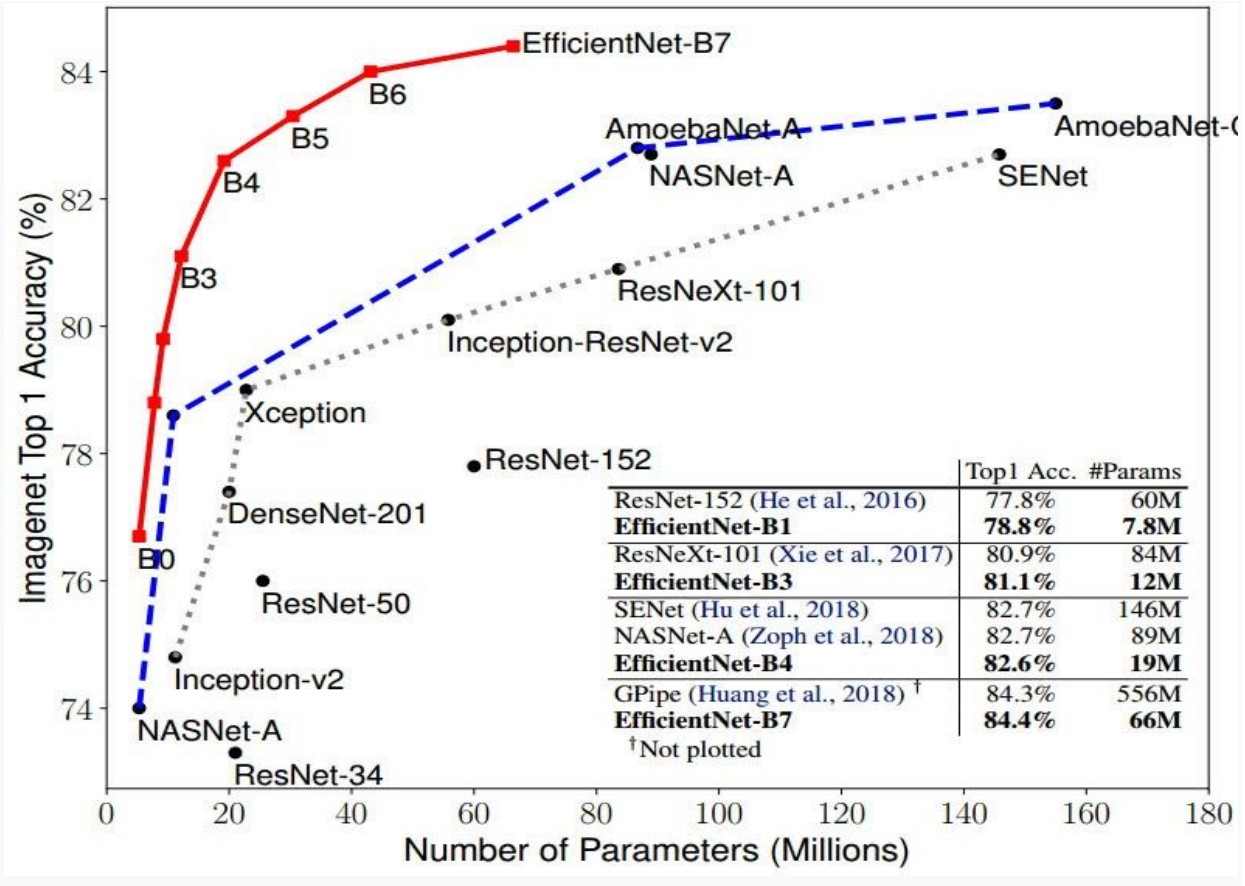
UNet architecture

À première vue, il a une forme en «U». L'architecture est symétrique et se compose de deux parties principales - la partie gauche est appelée “contracting path”, qui est constituée par le processus convolutif général; la partie droite est appelée “expansing path”, qui est constituée de couches convolutives 2d transposées (des déconvolutions).

https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5

EfficientnetB2 backbone

Les EfficientNets sont une famille de modèles de classification d'images, qui atteignent une précision de pointe, tout en étant d'un ordre de grandeur plus petit et plus rapide que les modèles précédents. Les EfficientNets sont basés sur AutoML et Compound Scaling. En particulier, la structure AutoML Mobile a été utilisée pour développer un réseau de base de taille mobile, nommé EfficientNet-B0; Ensuite, la méthode de mise à l'échelle composée est utilisée pour mettre à l'échelle cette base de référence pour obtenir EfficientNet-B1 à B7. Plus on avance et plus le score de précision du modèle converti augmente.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**78.8%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.1%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.6%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.4%**	**66M**

†Not plotted

Dans notre cas on a utilisé Efficientnet B2 donc ça entraine qu'on aura un niveau de précision variant entre 0.8083 et 0.9531

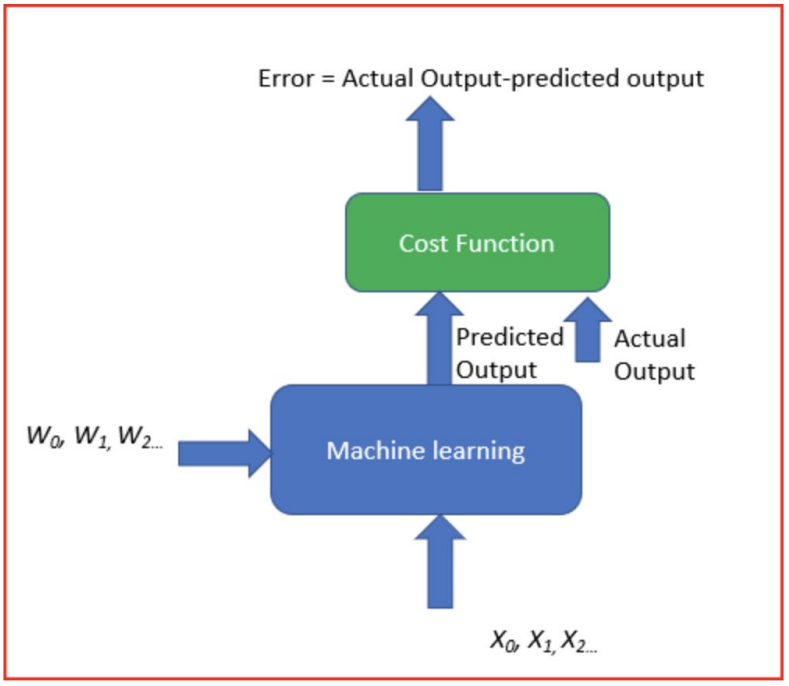
Augmentation of flips and rotate

cv:Librairie de Open source utilisée pour résoudre les problèmes de computer vision. Dans notre cas on a eu à utiliser **cv2** pour la gestion d'image

cv2.imread(): Il charge une image à partir du fichier spécifié. Si l'image ne peut pas être lue (en raison d'un fichier manquant, d'autorisations incorrectes, d'un format non pris en charge ou non valide), cette méthode renvoie une matrice vide.

Adam Accumulate Optimizer

Descente de gradient



Gradient based learning

À chaque instant pendant l'entraînement, un réseau de neurones présente une certaine perte, ou erreur, calculée à l'aide d'une fonction de coût (également appelée fonction de perte). Cette fonction indique à quel point le réseau (ses paramètres) est «erroné» en fonction des données d'apprentissage ou de validation. Idéalement, la perte devrait être aussi faible que possible. Malheureusement, les fonctions de coût ne sont pas

convexes - elles n’ont pas seulement un minimum, mais de nombreux minima locaux.

Pour minimiser la fonction de perte d’un réseau neuronal, un algorithme appelé rétropropagation(backpropagation) est utilisé. La rétropropagation calcule la dérivée de la fonction de coût par rapport aux paramètres du réseau neuronal. En d'autres termes, il trouve la «direction» dans laquelle mettre à jour les paramètres afin que le modèle fonctionne mieux. Cette «direction» s’appelle le gradient d’un réseau de neurones.

Avant de mettre à jour le modèle avec le gradient, le gradient est multiplié par un taux d'apprentissage (learning rate). Cela donne la mise à jour effective sur le réseau de neurones. Lorsque le taux d'apprentissage est trop élevé, nous pouvons dépasser le minimum, ce qui signifie que le modèle n'est pas aussi bon qu'il aurait pu l'être.
Mais d'un autre côté, lorsque le taux d'apprentissage est trop faible, le processus d'optimisation est extrêmement lent. Un autre risque du faible taux d'apprentissage est le fait que l'on puisse se retrouver dans un mauvais minimum local. Le modèle est à un état sous-optimal à ce stade, mais il pourrait être bien meilleur.

C'est là qu'interviennent les optimiseurs. La plupart des optimiseurs calculent automatiquement le taux d'apprentissage. Les optimiseurs appliquent également le gradient au réseau neuronal - ils font apprendre au réseau. Un bon optimiseur entraîne rapidement les modèles, mais il les empêche également de rester coincés dans un minimum local.

Adaptive Moment Estimation (ADAM)

Adam est un algorithme d'optimisation qui peut être utilisé à la place de la procédure classique de descente de gradient stochastique pour mettre à jour les poids de réseau itératifs en fonction des données d'apprentissage.
Il s’agit de l’optimizer le plus utilisé (surtout pour les réseaux de neurones) en raison de son efficacité et de sa stabilité (même si elle n’est pas parfaite).

La descente de gradient stochastique classique maintient un taux d'apprentissage unique (appelé alpha) pour toutes les mises à jour de poids et le taux d'apprentissage ne change pas pendant l'entraînement.
Un taux d'apprentissage est maintenu pour chaque poids de réseau et adapté séparément à mesure que l'apprentissage se déroule.

Par contre, Adam prend en compte les taux d'apprentissage passés afin de mettre à jour le taux actuel.
Il se base sur la notion de momentum. Par exemple, jetez un rocher dans la pente d'une montagne : à mesure qu'il va rouler dans la même direction, il va gagner en vitesse, mais s'il tourne, sa vitesse sera réinitialisée.
Pour Adam, le principe est identique : tant que le gradient est dans la même direction que ceux précédents, on va accélérer la vitesse (dite d'apprentissage) à laquelle on descend la courbe (i.e. on met à jour les paramètres).

$$v_t = \gamma v_{t-1} + \eta \nabla J(\theta; x, y)$$
$$\theta = \theta - v_t$$

Momentum Gradient descent takes gradient of previous time steps into consideration

Les auteurs décrivent Adam comme un algorithme combinant les avantages de deux autres algorithmes dérivés de la descente de gradient stochastique que sont AdaGrad et RMSProp.

ADAM :

- Est une méthode qui calcule le taux d'apprentissage adaptatif individuel pour chaque paramètre à partir d'estimations des premier et deuxième moments des gradients.
- réduit également les taux d'apprentissage excessivement décroissants d'Adagrad
- implémente la moyenne mobile exponentielle des gradients pour mettre à l'échelle le taux d'apprentissage au lieu d'une simple moyenne comme dans Adagrad.
- est efficace en termes de calcul et nécessite très peu de mémoire

L'algorithme Adam met d'abord à jour les moyennes mobiles exponentielles du gradient (mt) et du gradient carré (vt) qui sont les estimations du premier et du deuxième moment.
Les hyper-paramètres β1, β2 ∈ [0, 1) contrôlent les taux de décroissance exponentielle de ces moyennes mobiles comme indiqué ci-dessous

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

m_t and v_t are estimates of first and second moment respectively

Les moyennes mobiles sont initialisées à 0, ce qui conduit à des estimations de moment biaisées autour de 0, en particulier pendant les itérations initiales. Ce biais d'initialisation peut être facilement neutralisé, ce qui donne des estimations corrigées du biais.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\hat{v}_t = \frac{V_t}{1 - \beta_2^t}$$

$\hat{m}_t$ and $\hat{v}_t$ are bias corrected estimates of first and second moment respectively

Enfin, le paramètre est mis à jour comme indiqué ci-dessous:

$$\theta_{t+1} = \theta_t - \frac{\eta \hat{m}_t}{\sqrt{\hat{v}_t + \varepsilon}}$$

https://penseeartificielle.fr/meilleur-optimizer-ranger-radam-lookahead-fastai/
https://medium.com/datadriveninvestor/overview-of-different-optimizers-for-neural-networks-e0ed119440c3
https://arxiv.org/pdf/1412.6980.pdf

Jaccard loss

Il s'agit d'une implémentation directe de la formule «intersection over union»: une fonction de perte importante pour obtenir des formes propres et régulières dans la segmentation. En effet, grâce à celui-ci, le réseau de neurones sera optimisé directement au niveau de l'objet et les cross-validation metrics qu'on obtiendra seront beaucoup plus proche du score de performance réel.

Kaggle Dice metric, Kaggle accuracy metric

Le coefficient de Dice est une métrique utilisée pour évaluer la similitude de deux échantillons. Dans notre cas, **la Dice metric** va mesurer la similarité ou la dissimilarité entre deux échantillons.

L'accuracy est l'un des critères permettant d'évaluer les modèles de classification. De façon non formelle, l'accuracy désigne la proportion des prédictions correctes effectuées par le modèle. Formellement, il est défini ainsi :

$$Accuracy = \frac{Nombre\ de\ prédictions\ correctes}{Nombre\ total\ de\ prédictions}$$

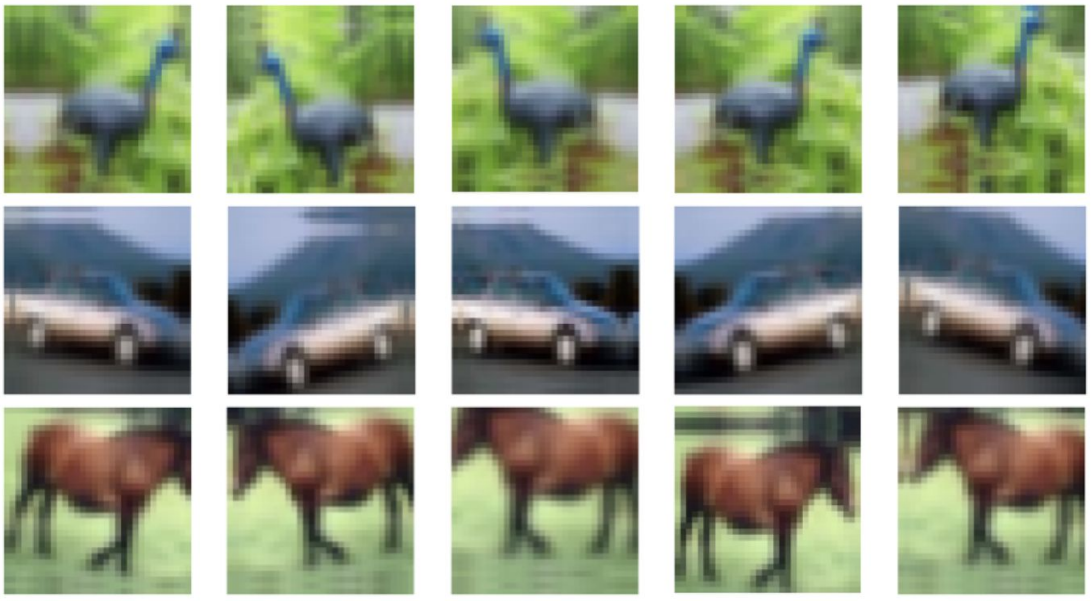
Reduce LR on plateau and early stopping

ReduceLROnPlateau permet de réduire le taux d'apprentissage dynamique en fonction de certaines mesures de validation.
Il prend en paramètre un certain nombre d'arguments.
ReduceLROnPlateau(monitor='...', factor=..., patience=..., verbose=..., mode='...', min_delta=...).

- **monitor**: Quantité à surveiller(Dans notre cas c'est val_dice_coef)
- **factor**: Facteur par lequel le taux d'apprentissage sera réduit
- **patience**: nombre d'époques sans amélioration après quoi le taux d'apprentissage sera réduit.
- **verbose**: int. 0: quiet, 1: update messages.
- **mode**: Il prend l'une de ces valeurs {**'auto', 'min', 'max**'}. En mode '**min**', la vitesse d'apprentissage sera réduite lorsque la quantité surveillée cessera de diminuer; en mode '**max**', il sera réduit lorsque la quantité surveillée aura cessé d'augmenter; en mode **'auto'**, le sens est automatiquement déduit du nom de la quantité surveillée.
- **min_delta:** seuil de mesure du nouvel optimum, pour se concentrer uniquement sur les changements significatifs.

Data and Test Augmentation

La data augmentation et la test augmentation se réfèrent au même procédé. Le but est d'effectuer des transformations arbitraires sur les images de test. Ainsi, au lieu de montrer les images régulières, propres, une seule fois au modèle entraîné, on montre les images augmentées plusieurs fois. On fait après la moyenne des prédictions de chaque image correspondante qu'on utilise comme prédiction finale.



Example of Data Augmentation on the CIFAR10 dataset

Remove false positive masks with classifier

Un **faux positif** est le résultat d'une prise de décision dans un choix à deux possibilités (positif et négatif), déclaré positif, là où il est en réalité négatif. Dans notre cas, le résultat est issu d'un algorithme de classification automatique.

Fold CV and prediction

CV ou Validation Croisée, fournit des indices de train / test pour fractionner les données dans des ensembles de train / test. On divise l'échantillon original en k échantillons ou blocs, puis on sélectionne un

des k échantillons en comme ensemble de validations pendant que les k-1 autres échantillons constituent l’ensemble d’apprentissage.Après apprentissage, on peut calculer une performance de validation. Puis on répète l'opération en sélectionnant un autre échantillon de validation parmi les blocs prédéfinis. Dans notre cas on a considéré 3 splits.