**Fonction get\_metric\_deep (myutils\_benchmarks\_earth.py et myutils\_benchmarks\_sideral.py) :**

Les vrais positifs sont les détections qui ont un IOU **supérieur** au seuil avec **au moins un** exemple de vérité terrain.

Les faux positifs sont les détections dont les IOU avec **tous** les exemples de vérité terrain sont **inférieurs** au seuil.

Les faux négatifs sont les exemples de vérité terrain dont les IOU avec **toutes** les détections sont **inférieurs** au seuil.

Les comparaisons sont effectuées par classe. Aucune comparaison n’est donc faite entre les détections et les exemples de vérité terrain n’appartenant pas à la même classe.

Pour obtenir le « true positive rate » par snr, le nombre de vrais positifs pour un snr est divisé par le nombre total d’exemples de vérité terrain pour ce snr.

**Fonction run\_bench\_triton (myutils\_benchmarks\_earth.py et myutils\_benchmarks\_sideral.py) :**

Dans cette fonction, il est nécessaire de créer les boites englobantes pour triton car celles-ci ne sont pas données dans les fichiers json.

Pour les satellites ponctuels, une marge de 3 pixels par rapport au centroid est définie. Donc les boites ont une hauteur de 6 pixels et une largeur de 6 pixels. Cette marge de 3 pixels était celle qui donnait les meilleurs résultats pour triton.

Pour les satellites défilants, les coordonnées du coin supérieur gauche et du coin inférieur droit sont définis en se servant des extrémités ainsi que de la taille de l’objet données dans les fichiers json. Le signe des tailles en x (dx) et en y (dy) permet de savoir si la coordonnée x0 (respectivement y0) dans les fichiers json correspond au point le plus à gauche ou le plus à droite de l’objet (respectivement le plus bas ou le plus haut).

Les métriques sont calculées de la même manière que dans la fonction *get\_metric\_deep.*

**Fonction run\_inference et autres fonctions permettant d’effectuer l’inférence (myutils\_benchmarks\_earth.py et myutils\_benchmarks\_sideral.py) :**

La fonction permet d’obtenir les résultats de détection de l’algorithme de deep learning.

Les images sont récupérées en itérant sur le dataset de test. Chaque élément de ce dataset de test est un dictionnaire. Le champ ‘file\_name’ permet de récupérer le chemin jusqu’à l’image.

La fonction *split\_image* permet de découper l’image en 25 patchs se recouvrant sur 128 pixels.

Ensuite, l’itération sur chacun des 25 patchs peut être menée à bien. Pour chaque itération, le patch est donné aux 3 détecteurs de façon itérative. Le « predictor » est un objet detectron2 wrappé autour d’un modèle et permet de réaliser l’inférence, c’est-à-dire de faire passer l’image (le patch) de test dans le réseau et d’obtenir les résultats.

La variable *pred\_boxes* correspond aux coordonnées des boites englobantes en sortie des détecteurs.

La variable *scores* correspond aux scores de confiance obtenus en sortie des détecteurs.

La variable *pred\_classes* correspond aux classes des objets détéctés.

Les coordonnées de boites obtenues avec les détecteurs des satellites ponctuels et des étoiles sont converties dans le format correspondant aux boites orientées de Détectron. Les coordonnées des boites obtenues pour les satellites défilants correspondent déjà à des boites orientées donc la conversion n’est pas nécessaire pour cette classe.

Une fois que les résultats des 3 détecteurs ont été agrégés dans des listes, ceux-ci sont fusionnés en rééchelonnant les coordonnées des boites pour les faire correspondre avec l’image originale de taille 2048. Seules les coordonnées du coin supérieur gauche sont rééchelonnées, la hauteur, la largeur et l’angle d’orientation de la boite reste la même.

Une fois que ces étapes ont été faites pour chacun des 25 patches et que les résultats ont été stockés dans des listes, ces derniers peuvent être fusionnés et attribués à l’objet « big\_output » qui est en fait un dictionnaire contenant un objet « Instances » de Detectron. Ce formalisme est nécessaire pour pouvoir utiliser les fonctions de visualisation de Detectron par la suite.

Une fois que le dictionnaire « big\_output » dispose des résultats des détections pour l’image, deux étapes de post-traitement sont exécutés (*filter\_boxes\_rotated2* et *filter\_ponctuels*).

**Fonction filter\_boxes\_rotated2 (myutils\_inference\_earth.py)**

Titre : Réalisation d’une étape de post-traitement pour éliminer les multiples détections d’un même objet dans les zones de recouvrement.  
Présentation du thème : La fusion des détections obtenues sur les sous-images des images réelles en inférence nécessite de supprimer les multiples détections d’un même objet dans les zones de recouvrement.  
Introduction : Les images réelles issues des télescopes ont une hauteur et une largeur de 2048 pixels. Des images d’une telle taille ne peuvent être utilisées pour l’entrainement de l’algorithme. En effet, le temps d’apprentissage serait trop long et la consommation de mémoire trop importante. De ce fait, il a été décidé d’entrainer les algorithmes sur des images simulées de taille 512 par 512 pixels. Dans la phase d’inférence, il est donc nécessaire de découper chaque image réelle en plusieurs images de taille 512, réaliser l’inférence sur chacune de ces sous images et ensuite, fusionner les résultats. Pour éviter que le découpage des images réelles ne sépare des objets en plusieurs parties, j’ai décidé de faire en sorte que les sous images se recouvrent sur des bandes de 128 pixels de large. Cependant, lorsque la fusion des détections des 25 sous images constituant une image réelle est effectuée avec le rééchelonnement des coordonnées des boites englobantes, les objets se situant sur les zones de recouvrement, et qui sont donc présents dans plusieurs images, sont détectés plusieurs fois. Ces détections redondantes sont également parfois attribuées à la mauvaise classe. Il fut donc nécessaire de réaliser une fonction permettant de supprimer les détections superflues.  
Développement : Une fois que la fusion des sous-images en une seule image réelle est réalisée et que les coordonnées des boites englobantes ont été rééchelonnées, les détections sont passées à la fonction de post-traitement. Cette fonction récupère d’abord les coordonnées de toutes les boites englobantes des objets détectés ainsi que l’aire de chacune de ces boites. Ensuite je crée un masque permettant de filtrer ces boites de façon à ne récupérer que les boites dont au moins une des extrémités se trouve dans les zones de recouvrement. Les aires sont également filtrées avec le même masque. Puis, je calcule le pourcentage de superposition entre chacune de ces boites se trouvant dans les zones de recouvrement pour obtenir un tableau 2D ayant autant de colonne que de lignes où chaque élément correspond au pourcentage de superposition entre deux boites. Je prends soin de remplacer les valeurs sur la diagonale par des zéros car je ne suis pas intéressé par le pourcentage de superposition d’une boite avec elle-même.  
 Ensuite, je récupère les indices en ligne et en colonne des éléments de ce tableau supérieurs à un seuil fixé par l’utilisateur en argument de la fonction. Cela permet de récupérer uniquement les détections des zones de recouvrement ayant un pourcentage de superposition supérieur à ce seuil. Après cela, je récupère les aires de ces détections en utilisant ces mêmes indices. Par conséquent, j’obtiens uniquement des paires d’aires où chaque élément des paires correspond à une détection de la zone de recouvrement ayant un pourcentage de superposition avec la seconde détection de la paire supérieur au seuil fixé par l’utilisateur. Puis, je supprime parmi ces paires, la détection ayant la taille de l’aire minimum. Cela est possible car je peux remonter jusqu’aux indices de ces boites supprimées dans l’ensemble des détections de l’image. Je dispose en effet des indices correspondant aux détections des zones de recouvrement (via le masque), des indices au sein des détections des zones de recouvrement correspondant aux boites ayant un pourcentage de superposition supérieur au seuil, ainsi que des indices parmi ces dernières détections des boites ayant l’aire la plus petite. Grâce à l’« advanced indexing » de Numpy, c’est-à-dire en indexant des tableaux avec d’autres tableaux, je peux donc remonter jusqu’aux indices de ces boites par rapport à l’ensemble des boites de l’image. Pour calculer le pourcentage de superposition, je me suis servi de la bibliothèque « Shapely ». J’ai dû représenter les boites sous la forme de polygones puis calculer pour chaque boite, à l’aide de méthodes de classe, son aire et son intersection avec chacune des autres boites. Je pouvais donc ensuite calculer pour chaque boite le ratio de l’intersection sur l’aire de la boite. Pour obtenir les polygones correspondant aux boites, il fut nécessaire de passer d’une représentation sous la forme centroid, hauteur, largeur et angle d’inclinaison à une représentation sous la forme des coordonnées des quatre extrémités de la boite.  
Conclusion : Ce post-traitement a permis de retirer les détections superflues dans les zones de recouvrement. Cet algorithme fonctionne également dans le cas de boites orientées. La méthode que j’ai mise en œuvre ici ressemble à celle que j’avais implémentée pour l’algorithme de non maxima suppression dans le cadre du cours « Deep Learning pour l’image ». De plus, la réalisation de cet algorithme par « vectorisation » avec Numpy, c’est-à-dire sans itération, est assez proche de ce que j’ai pu faire dans de nombreux projets au sein de la majeure Image (TIFO, introduction à l’imagerie médicale, Python pour le big data…)

**Fonctions get\_seg\_json et get\_seg\_csv pour la segmentation d’instance (myutils\_train\_seg\_instance.py)**

Titre : Constitution de la vérité terrain pour la segmentation d’instance  
Présentation du thème : Extraction des masques de segmentation de chaque objet pour constituer la vérité terrain dans le cas de la segmentation d’instance avec Mask-R-CNN  
Introduction : Dans le cadre de la mise au point d’un programme de segmentation d’instance utilisant l’algorithme Mask-R-CNN mis à disposition par la bibliothèque « Detectron2 », il fut nécessaire d’extraire les masques de chaque objet. La segmentation d’instance permet de combiner la détection à l’aide des boites englobantes et la segmentation individuelle de chaque objet. A la différence de la segmentation sémantique où tous les pixels appartenant à une même classe ont la même valeur, avec la segmentation d’instance, seuls les pixels appartenant à un même objet ont la même valeur. Ainsi il est possible de différencier chaque objet appartenant à une même classe. Pour créer la vérité terrain avec Detectron2, il faut indiquer le masque de chaque objet soit sous la forme d’une image binaire où seuls les pixels de l’objet ne sont pas nuls, soit sous la forme d’une liste de listes de points (l’objet peut être en plusieurs morceaux) correspondant aux contours du polygone constituant l’objet. Du fait des contraintes très importantes qu’impose la génération d’une image binaire par objet (certaines images peuvent contenir 2000 objets), il fut décidé d’extraire les masques en utilisant la seconde méthode.  
Développement : Pour pouvoir extraire le masque de chaque objet individuellement je me suis servi de l’image de segmentation donnée par le simulateur où chaque objet de l’image est segmenté selon sa classe d’appartenance (étoile, satellite, fond…). Cette image représente donc une segmentation sémantique et non une segmentation d’instance. Il n’est pas possible de différencier les objets d’une même classe sur cette image. Le simulateur fourni également pour chaque objet les coordonnées de ses extrémités dans un fichier CSV pour les étoiles et JSON pour les satellites. A partir des coordonnées de ces extrémités j’ai défini, pour chaque objet, des boites englobantes orientées sous la forme centroid, hauteur, largeur et angle d’orientation dans le sens inverse des aiguilles d’une montre. Pour calculer l’angle j’utilise la fonction « atan2 » entre les coordonnées en x du vecteur d’orientation de l’objet et les coordonnées en y de ce même vecteur. Les coordonnées du vecteur se calculent en faisant la différence entre les coordonnées des deux extrémités de l’objet en prenant garde de multiplier cette différence par -1 dans le cas des coordonnées en y car l’axe des y augmente vers le bas de l’image contrairement à un repère cartésien.

Ensuite, l’idée principale consiste à créer un masque où seuls les pixels à l’intérieur de la boite orientée d’un seul objet prennent la valeur « True » tandis que tous les autres pixels de l’image prennent la valeur « False ». En utilisant des boites orientées, j’obtiens une plus grande précision dans la définition du masque. Ainsi, bien que l’algorithme de segmentation d’instance mis en place détecte les objets avec des boites alignés avec les bords de l’image, il fut tout de même nécessaire de définir des boites orientées pour extraire le masque avec plus de précision au moment de la création de la vérité terrain. Pour créer ce masque pour chaque objet j’ai utilisé la fonction « fillPoly » d’OpenCV. Cette fonction colore dans une image une zone indiquée sous la forme d’une liste de points correspondant à son contour. Il fut donc d’abord nécessaire d’obtenir les quatre extrémités des boites orientées en effectuant la conversion depuis une représentation sous la forme centroid, hauteur, largeur, angle d’orientation. Une fois ces extrémités obtenues pour chaque boite orientée, j’ai pu créer, pour chaque objet, [l’image binaire](#image_masque) correspondant en remplaçant les pixels de la zone délimitée par les extrémités de la boite orientée par la valeur 1 dans une image rempli de zéros ayant la même taille que les images d’entrainement.  
Après cela, j’utilise cette image masque pour récupérer, dans l’image segmentée fournie par le simulateur, uniquement l’objet se situant dans cette zone non nulle du masque. Une fois l’objet de la zone récupéré, j’utilise la fonction « findContours » d’OpenCV pour obtenir les contours de l’objet sous la forme d’une liste de tableaux Numpy 2D. Chaque tableau Numpy correspond aux contours d’une forme dans l’image. Si la liste comprend plusieurs tableaux Numpy alors je garde uniquement le tableau représentant l’objet ayant l’aire la plus grande. En effet, il est anormal d’obtenir plus d’un tableau par liste. Par ailleurs, les tableaux ayant moins de trois lignes, et donc moins de trois points, sont supprimés de la liste retournée par la fonction. Il se peut qu’avant ou après avoir supprimé ces tableaux, la liste censée les contenir soit vide. Cela s’explique par le fait qu’il y a plus d’objets spécifiés dans les fichiers CSV et JSON que dans l’image segmentée fournie par le simulateur. Il n’y a donc pas de correspondance exacte entre le nombre d’objets dans les fichiers CSV et JSON qui ont servi à créer les boites englobantes orientées et le nombre d’objets dans l’image segmentée. En conséquence, lorsque la liste retournée par la fonction « findContours » est vide après avoir supprimé les tableaux en trop, l’opération est annulée pour cet objet et aucune donnée de vérité terrain n’est créé (masque, coordonnées de la boite englobante, classe…).  
La zone qui représente la boite englobante orientée de l’objet, utilisée pour créer l’image masque, est naturellement légèrement plus large que l’objet lui-même. Par conséquent il est possible qu’une partie d’un autre objet soit présent dans cette zone en plus de l’objet principal. De ce fait, Si des pixels d’une autre couleur que la couleur de l’objet principal sont présents dans cette zone alors cela signifie qu’un objet appartenant à une autre classe est aussi présent dans la zone. Si c’est le cas, alors je remplace la valeur de ces pixels par des zéros pour les faire correspondre au fond. En effet, sans cet ajout, la fonction « findContours » d’OpenCV risquerait de retourner plusieurs tableaux Numpy comme s’il y avait plusieurs objets ou un seul tableau Numpy mais avec les coordonnées des contours inexacts.  
Conclusion : Bien que n’étant pas simple à mettre en œuvre, cette façon de récupérer le masque des objets sous la forme de points correspondant à la frontière des objets fut indispensable pour entrainer un algorithme de segmentation d’instance. J’avais déjà eu l’occasion d’utiliser la fonction « findcontours » d’OpenCV dans le cadre du projet image de l’IGN. De même, la méthode que j’ai mise en place ici est similaire à ce que j’avais implémenté dans le cadre du projet TIFO pour récupérer les dates sur les images satellites de Venise.

**Fonction filter\_ponctuels (myutils\_inference\_earth.py)**

Titre : Réalisation d’une étape de post-traitement pour éliminer les fausses détections de ponctuels se superposant avec d’autres objets.

Présentation du thème : Après avoir lancé l’entrainement du réseau de neurones, et effectué l’inférence, j’ai pu remarquer que certains satellites ponctuels étaient détectés par-dessus d’autres objets ayant un faible SNR.  
Introduction : J’ai d’abord entrainé l’algorithme de Deep Learning pour détecter les satellites en mode « géostationnaire ». Dans ce mode de prise de vue, il est nécessaire de détecter les étoiles, les satellites défilants et les satellites ponctuels. Si les performances de l’algorithme étaient satisfaisantes pour les satellites défilants et les étoiles, les résultats étaient légèrement moins probant pour les satellites ponctuels, en particulier ceux ayant un faible SNR et qui sont donc peu contrastés par rapport au fond. J’ai donc suggéré d’ajouter au jeu de données des images comportant des satellites ponctuels ayant une magnitude plus forte et donc un SNR plus faible. Cela a permis de grandement améliorer l’efficacité des détections pour les satellites ponctuels à faible SNR. Si les résultats se rapprochaient alors des performances de l’algorithme « Triton », il subsistait tout de même un nombre trop important de faux positifs, en particulier par-dessus les étoiles ou les satellites défilants pour lesquels certaines variations de l’intensité des pixels pouvaient être considérées par l’algorithme comme des satellites ponctuels. Il était donc nécessaire de réaliser une étape de post-traitement pour éliminer ces fausses détections.  
Développement : La première approche à laquelle moi et mes encadrants avons pensé consistait à récupérer les ponctuels superposés aux étoiles et aux défilants puis à comparer leur intensité moyenne avec celle de l’objet se trouvant derrière. Si la distance entre ces deux intensités moyennes est inférieur à un seuil, alors le ponctuel détecté est éliminé car il n’est pas suffisamment contrasté par rapport à l’objet se trouvant derrière lui. L’intensité moyenne de l’objet est récupérée en calculant la moyenne de la valeur des pixels se trouvant dans la boite englobante. Je me suis alors rendu compte qu’utiliser la moyenne des pixels comme indicateur de l’intensité de l’objet était peu adapté lorsque la boite englobante n’était pas serré au plus proche des contours de l’objet. En particulier, dans le cas des satellites défilants qui sont orientés de façon oblique par rapport aux axes de l’image, une large part des pixels contenus dans la boite englobante n’appartiennent pas à l’objet mais au fond. J’ai donc jugé opportun, et avec l’accord de mes encadrants, de modifier l’algorithme de détection utilisé afin de pouvoir détecter les objets avec des boites englobantes orientées.  
Après m’être documenté sur les modifications nécessaires à apporter à l’algorithme pour détecter les objets avec des boites englobantes orientées, il a alors été possible d’obtenir pour les satellites défilants des boites englobantes dont les arêtes sont très proches des frontières de l’objet. Néanmoins, il m’est apparu que les intensités moyennes des objets calculées étaient peu informatives sur le contraste des ponctuels avec les objets sur lesquels ils étaient superposés. En effet, la distance entre l’intensité moyenne du ponctuel et de l’objet se trouvant derrière était parfois élevé alors que visuellement les deux objets semblaient peu contrastés. A l’inverse, cette distance était parfois faible alors que les objets semblaient visuellement très contrastés. J’ai donc décidé d’abandonner cette façon de calculer le contraste entre les objets. J’ai également pu remarquer que cette notion de contraste entre les objets était déjà intégrée par l’algorithme de détection au travers du taux de confiance de la détection obtenu en sortie de la phase d’inférence. En effet, plus ce taux est élevé et plus les ponctuels ont tendance à ressortir par rapport au fond, donc plus le taux de confiance est élevé, et plus le ponctuel est brillant et contrasté par rapport au fond. Ce taux de confiance est donc également un indicateur du contraste du ponctuel par rapport à ce qui se trouve derrière lui.

Pour déterminer si un ponctuel est superposé à un autre objet, j’ai d’abord pensé qu’il était judicieux de calculer l’IOU entre ces objets. Néanmoins le calcul de l’IOU entre des boites orientées est plus complexe qu’avec des boites alignées avec les bords de l’image. Pour ce faire, j’ai, après m’être documenté, décidé d’utiliser la bibliothèque « Shapely » qui permet de définir des polygones à partir des quatre points aux extrémités des boites englobantes. Une fois les boites modélisées à travers ces polygones, il est possible d’utiliser les fonctions « union » et « intersection » de cette bibliothèque pour calculer l’IOU (intersection / union) entre deux boites. Toutefois, je ne disposais pas des points aux quatre extrémités de chaque boite. Les informations dont je disposais étaient : les coordonnées du centre de la boite, la largeur de la boite, sa hauteur et son angle d’inclinaison dans le sens inverse des aiguilles d’une montre. Après m’être documenté, j’ai pu trouver des formules permettant d’effectuer cette conversion. Il a toutefois été nécessaire que j’adapte ces formules car elles supposaient que les valeurs suivant l’axe des y augmentaient du bas vers le haut de l’image (comme dans un repaire cartésien) alors qu’elles étaient en fait minimums en haut de l’image.

Après une réflexion plus approfondis, il m’a semblé que l’IOU n’était pas un bon indicateur pour indiquer si un ponctuel est superposé à un autre objet. En effet, plus l’objet se trouvant derrière le ponctuel est grand et plus l’IOU est faible. Cela est dû à l’union entre les deux objets qui constitue le dénominateur dans le calcul de l’IOU. On aura donc des situations où le ponctuel est complètement intégré dans la boite englobante de l’objet se trouvant derrière, et donc parfaitement superposé à cet objet, et où pourtant l’IOU est faible. Il m’a donc semblé qu’une meilleure mesure de la superposition pouvait être calculé en évaluant la proportion du ponctuel compris dans la boite englobante de l’autre objet. Pour ce faire je calcule le rapport de l’aire de l’intersection sur l’aire du ponctuel (intersection / aire du ponctuel), encore à l’aide des polygones de « Shapely ». Il me semble que le cours « Deep Learning pour l’image » et le projet que j’ai dû mener dans le cadre de ce cours pour la détection de visages me fut très utile pour comprendre la manière dont est calculé l’IOU et pour me rendre compte que cette métrique n’était pas adaptée dans ce cas précis.

Finalement, l’algorithme de post-traitement se déroule comme suit : je récupère d’abord parmi l’ensemble des détections, d’un côté les ponctuels, et de l’autre les étoiles et les satellites défilants. Pour ce faire je crée un masque en m’appuyant sur les données obtenus en sorti de la phase d’inférence qui contiennent, entre autre, les classes associées à chaque détection. Ensuite je convertis les données relatives à chaque boite (décrites plus haut) de manière à obtenir les coordonnées de ses quatre extrémités. Après cela je crée des polygones pour chaque boite à l’aide de « Shapely ». Je calcule alors, pour chaque ponctuel, la proportion de l’objet qui intersecte chacun des autres types d’objets. Je récupère ensuite chaque ponctuel dont l’intersection est supérieure à un seuil fixé au préalable. Pour ces ponctuels, ceux qui ont un niveau de confiance de détection inférieur à un seuil sont éliminés. De plus, les ponctuels superposés à au moins deux objets sont supprimés sans tenir compte du seuil pour le score de confiance. En effet, des fausses détections de ponctuels apparaissaient lorsque plusieurs objets se croisaient.  
Conclusion : Cet algorithme de post-traitement a permis de grandement réduire la quantité de faux satellites ponctuels détectés tout en gardant de bons résultats pour les vrais positifs. De plus, le processus de réflexion autour de cet algorithme a également permis de passer à un modèle de détection par boites englobantes orientées, ce qui a permis d’obtenir une plus grande précision dans la localisation des satellites défilants. Les nombreux projets sur lesquels j’ai travaillé dans le cadre de la majeure Image m’ont permis d’acquérir une bonne maitrise des bibliothèques « Numpy » et « Pytorch » qui furent essentielles dans la réalisation de cet algorithme de post-traitement.

**Optuna (fonction « objective » et « optimize » dans myutils\_train\_earth.py)**

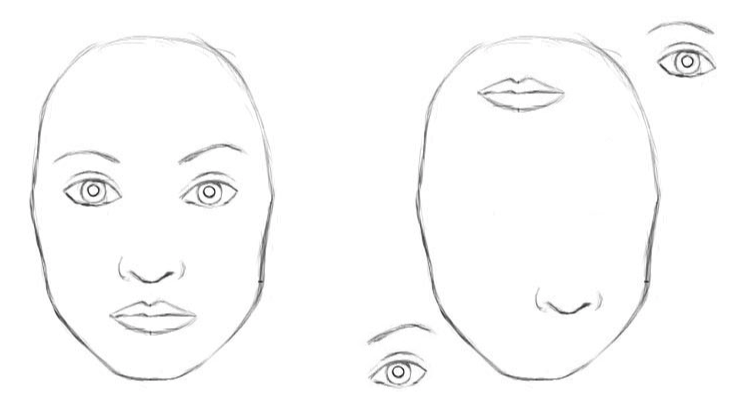
Optuna va permettre de chercher la valeur optimale de certains paramètres. Pour ce faire, une fonction objectif doit être définie. Cette fonction prend nécessairement un objet « trial » en argument. Cette fonction objectif peut également prendre des arguments supplémentaire si besoin. Dans ce cas, elle sera appelée à l’aide d’une lambda fonction. Au sein de cette fonction objectif, il est nécessaire d’indiquer les paramètres dont la valeur optimale va être cherchée par Optuna. Cela se fait en appelant la méthode « suggest » de l’objet trial passé en argument de la fonction. Selon que l’on cherche à trouver un float, un int ou un paramètre de type categorie (par exemple sgd, adam, adadelta…), on utilisera la méthode « suggest » appropriée (suggest\_float, suggest\_int, suggest\_categorical). Ensuite, toujours au sein de la fonction objectif, la seconde étape consiste à lancer l’entrainement. Enfin, cette fonction objectif doit retourner la valeur que l’on cherche à optimiser et que l’on obtient en sortie de l’entrainement (il s’agit souvent d’une valeur que l’on cherche à minimiser), par exemple la « validation\_accuracy » ou la loss sur les données de validation.

Une fois cette fonction objectif définie, il est nécessaire d’instancier, en dehors de la fonction, un objet « study » à l’aide de la fonction « create\_study » d’optuna. La méthode « optimize » de cet objet « study » peut ensuite être appelée en passant en argument la fonction objectif précédemment définie et que l’on cherche à optimiser. La méthode « optimize » attend également en argument le nombre d’essais, c’est-à-dire le nombre d’entrainements qui vont être lancés pour trouver la valeur optimale des paramètres permettant de minimiser la valeur retournée par la fonction objectif.

Une fois tous les essais réalisées, les paramètres optimaux sont stockés dans l’attribut « best\_params » de l’objet « study » précédemment instancié.

**Transformers**

Les architectures dites « transformers » diffèrent des réseaux de neurones conventionnels car elles utilisent des modules de « self-attention » plutôt que des convolutions. Ces modules ont l’avantage de pouvoir tirer parti de l’information contenue dans la relation spatiale entre plusieurs motifs (« patterns ») d’une image.



Un CNN pourrait penser que le dessein à droite constitue un visage

Par exemple, les transformers peuvent comprendre que, dans un visage, les yeux se trouvent nécessairement au-dessus de la bouche. A l’inverse, les convolutions n’extraient que de l’information locale, des motifs, sans comprendre la relation spatiale entre ces derniers. Cette supériorité des transformers est liée au fait que ces derniers bénéficient d’un « receptive field » très élevé dès les premières couches, contrairement aux cnn qui ont besoin d’empiler de nombreuses couches de convolution pour augmenter le « receptive field ».

A la différence des CNN, les transformers prennent en entrée une série de tokens. En computer vision, ces tokens correspondent à des patchs issus d’une image. L’image est découpée en patchs et ceux-ci sont donnés à la première couche du transformer. Ces couchent vont ensuite apprendre à pondérer l’importance de chacun de ces patchs dans la compréhension des images. C’est cette pondération qui permet aux transformers d’avoir un receptive field bien plus élevé que les CNN. Avant de transmettre ces patchs au transformer, une information relative au positionnement absolu (ou relatif) du patch dans l’image est souvent ajoutée pour aider le transformer à extraire de l’information spatiale dans l’image.

Malgré des résultats qui sont souvent supérieurs aux CNN, les transformers souffrent tout de même de certains inconvénients qui font l’objet de nombreux travaux de recherche. Ainsi, les modèles utilisant des transformers sont constitués d’un nombre de paramètres très élevé. Par exemple Vit, le vision transformer de google, est constitué, dans sa version de base, de près de 300 millions de paramètres. Pour comparaison, le modèle Faster R-CNN modifié que j’ai mis au point durant ce stage comprend 43 millions de paramètres. Cela a une influence sur le nombre d’images qui peuvent être traitées par seconde dans le réseau (throughput). Par conséquent, les modèles utilisant des couches transformers sont pour l’instant très peu adaptés à des contraintes temps réel. L’autre conséquence est que ces modèles sont très longs à entrainer. Ainsi, il n’est pas rare de devoir les entrainer pendant 3 à 4 jours. Il est également nécessaire de disposer d’importantes ressources de calcul. La plupart des articles utilisent des GPU tels que la tesla v100 ou bien des GPU effectuant le travail en parallèle.

Une autre difficulté vient du fait que les transformers nécessitent une très grande quantité de données en entrainement pour atteindre des performances similaires ou supérieures aux CNN.

Une façon efficace d’atténuer ces problèmes est d’initialiser les modèles transformers avec des poids correspondant à des réseaux transformers similaires et entrainés sur Imagenet. Certains travaux de recherche ont permis d’introduire des transformers utilisant moins de paramètres que Vit, le modèle standard de transformer pour les taches de computer vision. C’est le cas du « swin transformer » de microsoft qui permet d’obtenir des performances similaires/supérieures au Vit en utilisant beaucoup moins de paramètres (Performances identiques avec 300 millions de paramètres pour Vit contre 88 millions pour swin transformer). Microsoft comme Google mettent à disposition des modèles pré entrainés sur imagenet du swin transformer et du Vit respectivement. La complexité de Vit est quadratique avec la taille de l’image alors que celle du swin transformer est linéaire.

[Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., ... & Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. arXiv preprint arXiv:2103.14030.](https://arxiv.org/pdf/2103.14030.pdf)

[Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., & Jégou, H. (2021, July). Training data-efficient image transformers & distillation through attention. In International Conference on Machine Learning (pp. 10347-10357). PMLR.](https://arxiv.org/pdf/2012.12877.pdf)

[Kolesnikov, A., Dosovitskiy, A., Weissenborn, D., Heigold, G., Uszkoreit, J., Beyer, L., ... & Zhai, X. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.](https://arxiv.org/pdf/2010.11929.pdf)