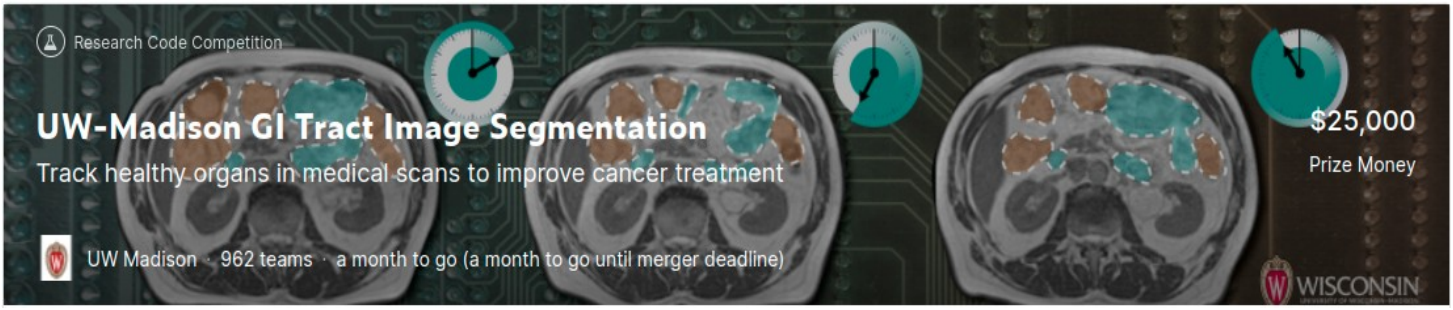
Openclassrooms projet 8 : Compétition Kaggle

UW-Madison GI Tract Image Segmentation

Le contexte :



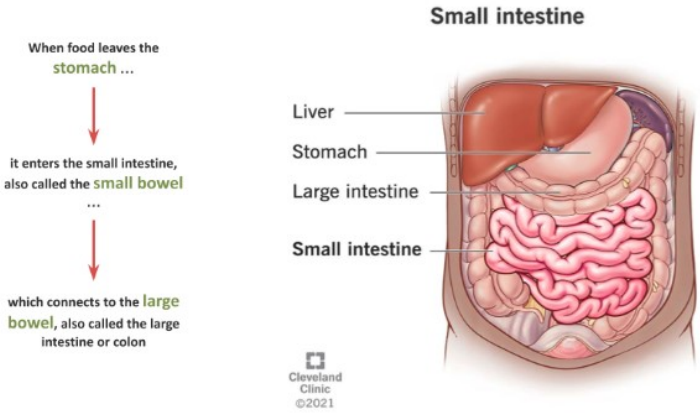
Segmentation sémantique d’images :

Aider les personnes souffrant de cancer du tractus gastro intestinal lors du scanner

Généralement administrée en 10 à 15 minutes par jour pendant 1 à 6 semaines.Les radio-oncologues tentent de délivrer de fortes doses de rayonnement à l'aide de faisceaux de rayons X dirigés vers les tumeurs tout en évitant l'estomac et les intestins. Grâce à une technologie plus récente telle que l'imagerie par résonance magnétique intégrée et les systèmes d'accélérateur linéaire, également connus sous le nom de MR-Linacs, les oncologues sont en mesure de visualiser la position quotidienne de la tumeur et des intestins, qui peut varier d'un jour à l'autre.
Dans ces scans, les radio-oncologues doivent tracer manuellement la position de l'estomac et des intestins afin d'ajuster la direction des faisceaux de rayons X pour augmenter la dose administrée à la tumeur et éviter l'estomac et les intestins.Il s'agit d'un processus long et laborieux qui peut prolonger les traitements de 15 minutes par jour à une heure par jour, ce qui peut être difficile à tolérer pour les patients, à moins que l'apprentissage en profondeur ne puisse aider à automatiser le processus de segmentation. Une méthode pour segmenter l'estomac et les intestins rendrait les traitements beaucoup plus rapides et permettrait à un plus grand nombre de patients d'obtenir un traitement plus efficace.

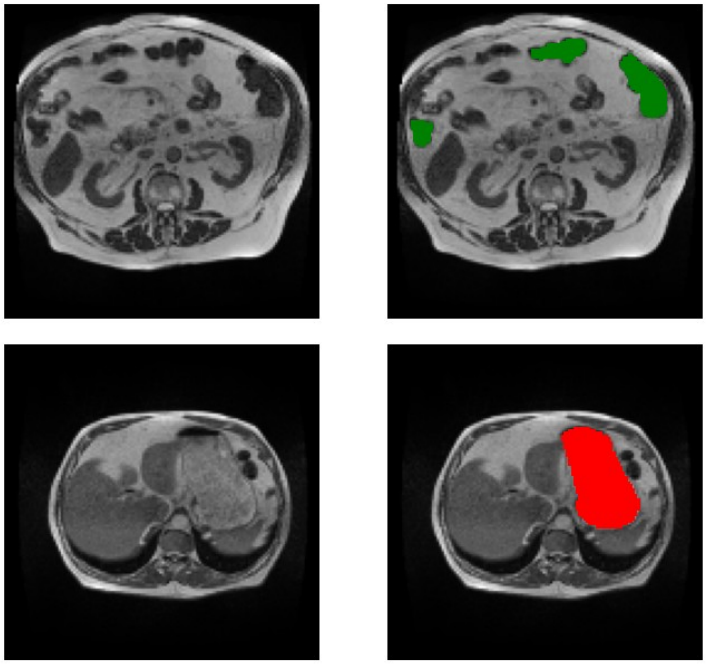
Les données :

Les données mis a disposition , composes de cliches IRM .
Ces cliches sont tries par patients. Pour chacun, plusieurs clichés de différents types de scans (jour, type...) sont disponibles. Le jeu d'entraînement contient les scans de 85 patients avec 16590 images et 33913 annotations des organes (intestins, gros intestins estomac)

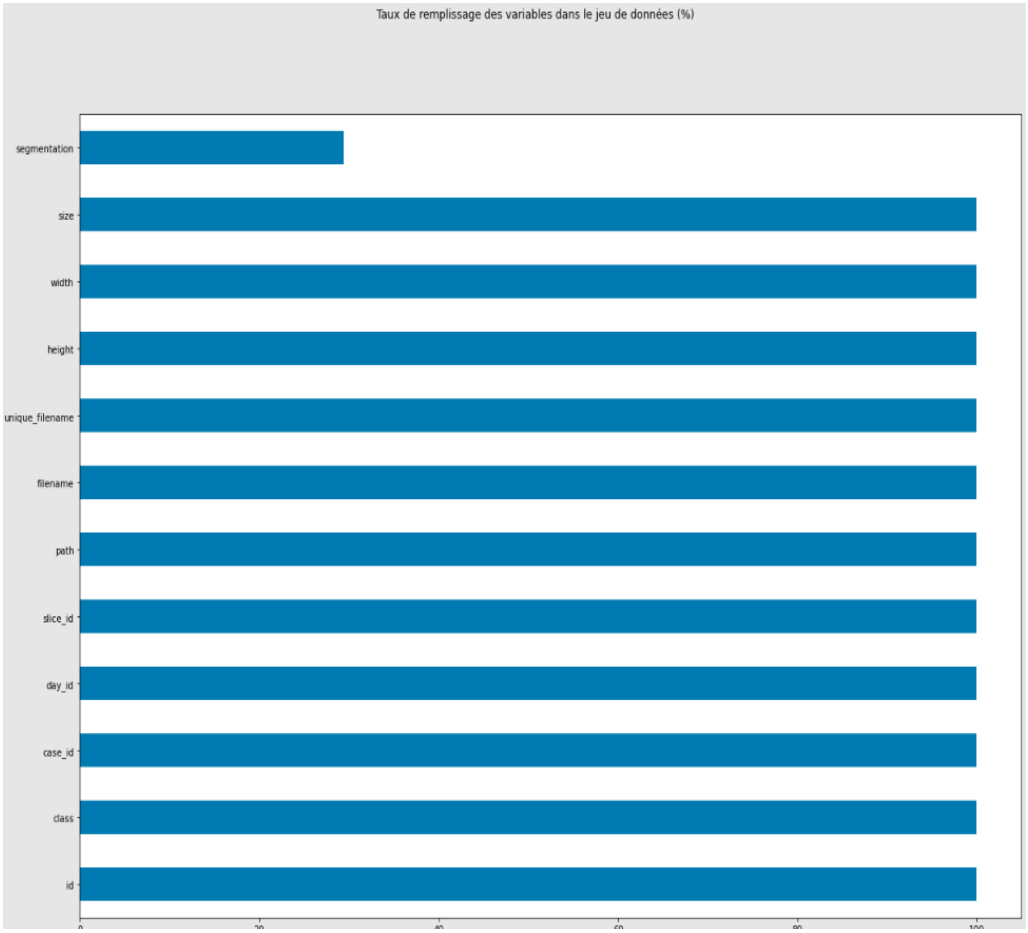


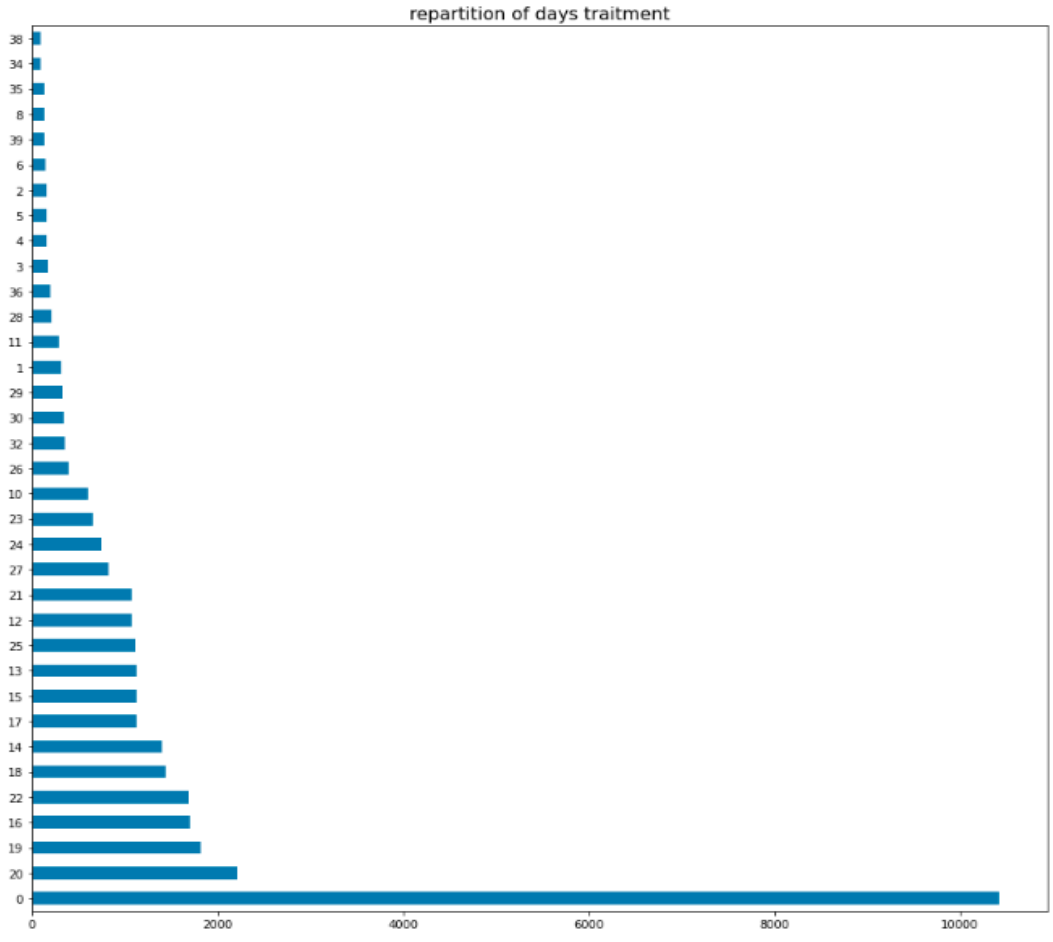
Les contraintes

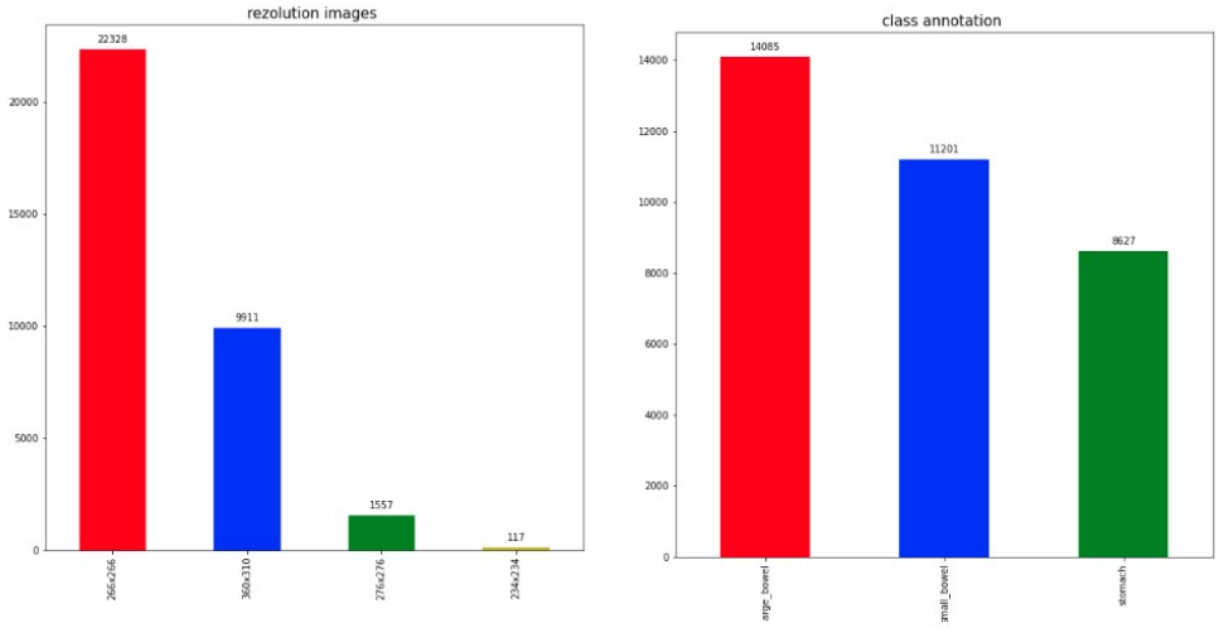
L'évaluation est réalisée à partir des notebooks soumis au concours. L'ensemble des opérations de ces derniers doivent pouvoir être réalisées en maximum 20 heures d'utilisation CPU ou GPU.
L'utilisation de données libres et gratuites et de modèles pré-entrainés est autorisée.



Analyse de nos données







Répartition des classes et des images par résolution

Évaluation des modèles

Lors de l'entraînent des modèles, l'ajout d'un jeu de validation permet de mesurer la capacité du modèle à généraliser. Dans l'algorithme de prédiction j'ai ajouté un appel à la fonction d'évaluation d'implémenté dans l'API de Keras.

Modele tester :

- Detectron2 (Meta) https://paperswithcode.com/lib/detectron2
- Unet (Google) https://paperswithcode.com/paper/efficientnet-rethinking-model-scaling-for

Architecture Detecron2 :

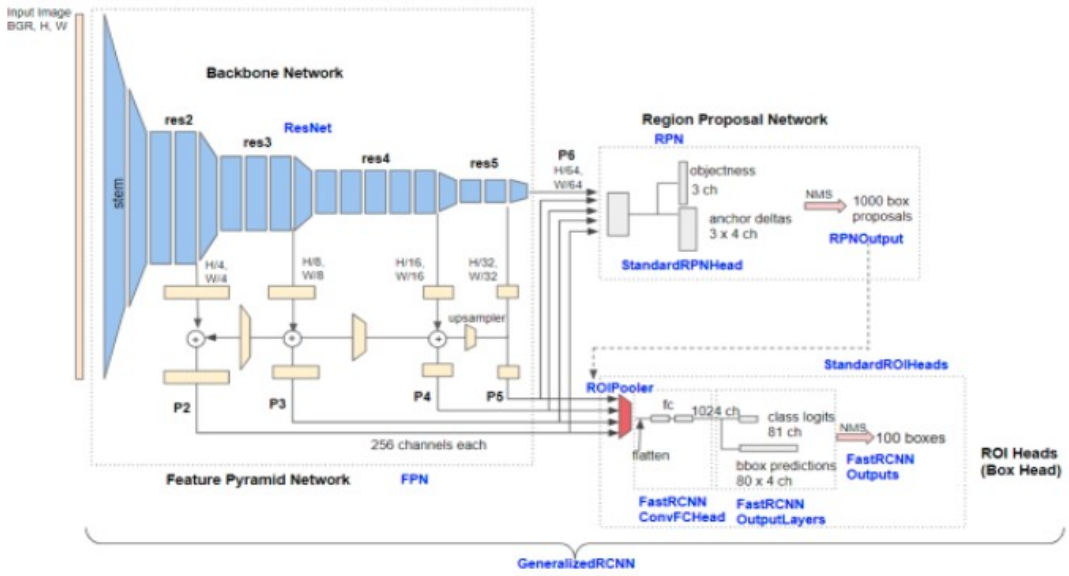
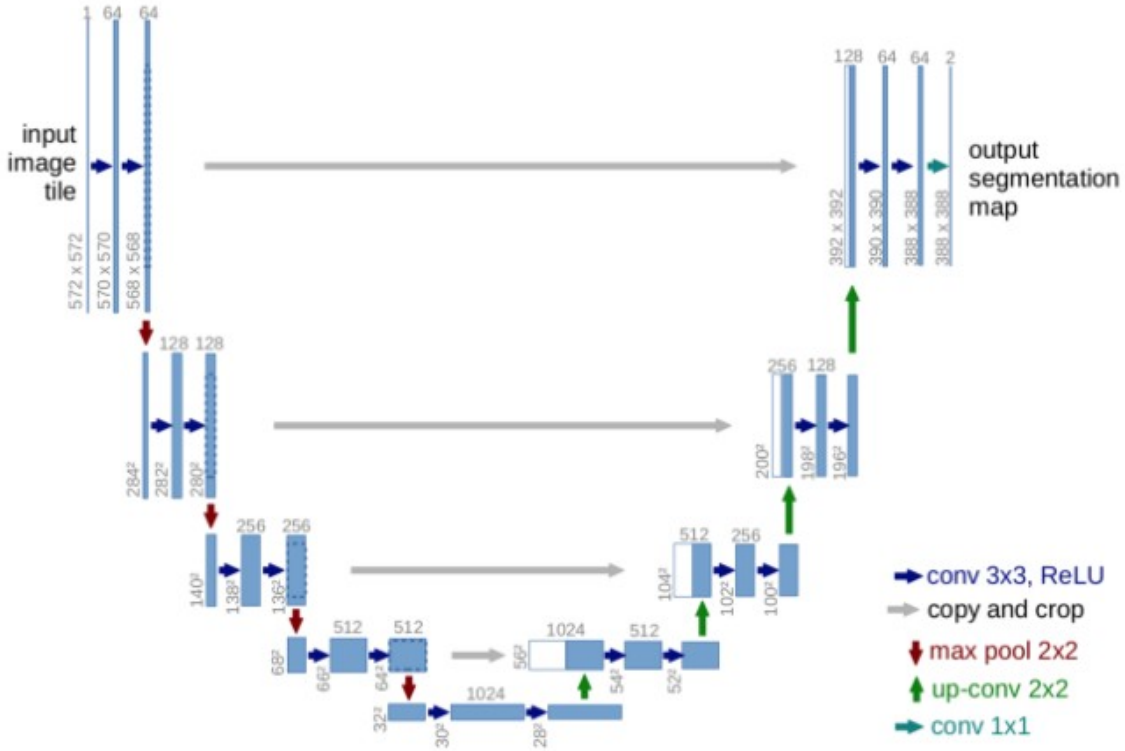


Figure 3. Detailed architecture of Base-RCNN-FPN. Blue labels represent class names.

Architecture Unet :



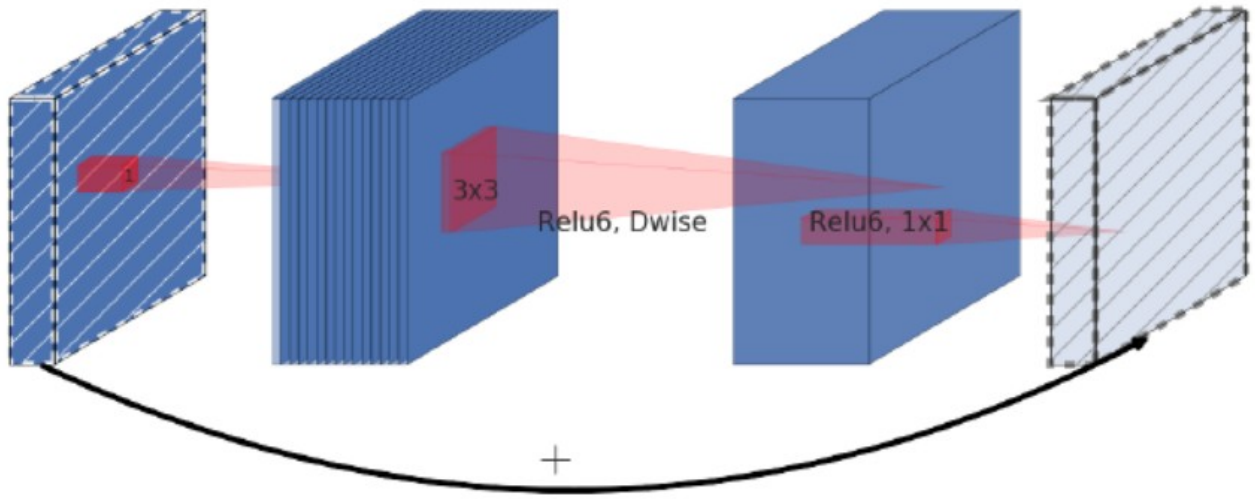
UNet architecture

La solution choisi implémente EfficientNet. La famille de modèles est issue de travaux publiés par les chercheurs de Google AI en 2019. Ceux-ci portaient sur l'augmentation de l'échelle de réseau de neurones convolutionnels. Jusque-là, l'augmentation de la taille des modèles pouvait être réalisée sur trois dimensions distinctes :
• Augmenter de la profondeur du réseau : permet d'apprendre des caractéristiques plus complexes mais rends la convergence du réseau plus difficile
• Augmenter la largeur des couches : permet d'apprendre un plus grand nombre de caractéristiques mais diminue la précision du modèle et la complexité des caractéristiques apprises.
• Augmenter la résolution des images en entrée : permet au modèle de mieux identifier les détails de l'image mais ne permet pas d'augmenter la précision du modèle.
Les équipes de Google ont recherché une heuristique d'augmentation d'échelle, appelée compoud scaling, permettant de combiner les trois dimensions en vue de maximiser les performances des modèles tout en minimisant le nombre de paramètres et d'opérations par secondes qu'ils réalisent.
Pour se faire les chercheurs ont d'abord conçu une architecture minimale EfficientNet-B0. Cette dernière devait pouvoir être déployée sur des smartphones.

Stage i	Operator $\hat{\mathcal{F}}_i$	Resolution $\hat{H}_i \times \hat{W}_i$	#Channels $\hat{C}_i$	#Layers $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Architecture EfficientNet-B0 – Source : Article de recherche *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks* publié sur ArXiv en septembre 2020 – Capture octobre 2021

L'architecture est basée du des blocs MBConv sur lesquels une optimisation squeeze-and-excitation est appliquée. Ces blocs sont similaires aux blocs résiduels inversés utilisés dans l'architecture de MobileNet v2. Ils implémentent une connexion de raccourcis entre la couche d'entrée et la couche de sortie du bloc. Les blocs sont composés d'une première couche de convolution 1 x 1 qui sert à augmenter la profondeur de la carte des caractéristiques. Cette couche est suivie d'une couche 3 x 3 Depth-wise puis couche Point-wise de 1 x 1.



Détail d'un bloc résiduel inversé - Source : Article *EfficientNet: Scaling of Convolutional Neural Networks done right* publié par Armughan Shahid sur towards data science en juin 2020 – Capture octobre 2021

La recherche de l'heuristique a été réalisée à l'aide d'une recherche de paramètres sur grille.
Afin de répondre à la modélisation suivante :

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

Modélisation du compound scaling – Source : *Article de recherche EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks publié sur ArXiv en septembre 2020* – Capture octobre 2021

- α, β, γ sont des constantes trouvées par recherche de paramètres sur grille
- φ un hyper paramètre de coefficient d'augmentation des ressources disponibles pour l'augmentation de l'échelle du modèle

Grâce au compound scaling, les chercheurs ont décliné l'architecture EfficientNet-B0 en sept versions ayants des constantes de dimensions croissantes. Ils ont comparé les performances, le nombre de paramètres et le nombre d'opération par secondes avec d'autres modèles issus de l'état de l'art.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

Comparatif des modèles de la famille EfficientNet à l'état de l'art – Source : *Article de recherche EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks publié sur ArXiv en septembre 2020* – Captubre ocotobre 2021

Source utilisées

https://www.kaggle.com/code/vineethakkinapalli/uw-madison-gi-tract-coco-dataset

https://www.kaggle.com/code/ammarnassanalhajali/uwmgi-unet-keras-train-with-eda

Mes travaux ce base sur les kernel ci dessus.
Le premier vise à convertir les données aux format coco json, et le second utilise tensorflow keras Unet pour la modélisation.

Amelioration :

Optimisation de la pipeline du traitement des images et des annotations Rle .
J ai eu l idée de développer un nouveau pipeline base sur le format coco.
Le but et de récupérer nos informations (images et masques) directement a partir du fichier json et de les convertir à la volée , pour l entraînement,ce générateur fournira une image traité et normalisé avec ces masque associées traitées et normalisées également. L intérêt de ce système et que les informations viennent uniquement du fichier json et non du dataframe.
J ai donc développer un parser/deparser adapter au spécification coco

```python
class DataGeneratorFromCocoJson(tf.keras.utils.Sequence):
  # function getting info dataset from json coco
  # Batch size
  # subset train or test for annotations
  # image_list to develop...
  # classes classe wanted
  # input image size tuple (X,X)
  # annFile path to annoted coco json file file
    def __init__(self, batch_size = batch_size, subset="train",image_list=[]
                 ,classes=[], input_image_size=(128,128),annFile='',shuffle=False):

        super().__init__()
        self.subset = subset
        self.batch_size = batch_size
        self.indexes = np.arange(len(image_list))
        self.image_list= image_list
        self.classes= classes
        self.input_image_size= (input_image_size)
        self.dataset_size = len(image_list)
        self.coco = COCO(annFile)
        catIds = self.coco.getCatIds(catNms=self.classes)
        self.catIds = catIds
        self.cats = self.coco.loadCats(catIDs)
        self.imgIds = self.coco.getImgIds()
        self.shuffle = shuffle
        self.on_epoch_end()


    def __len__(self):
      return int(len(self.image_list)/self.batch_size)

    def on_epoch_end(self):
        if self.shuffle == True:
            np.random.shuffle(self.indexes)


    def getClassName(self,classID, cats):
        for i in range(len(cats)):
            if cats[i]['id']==classID:
                return cats[i]['name']
        return None

    def getNormalMask(self,image_id,catIds):
        annIds = self.coco.getAnnIds(image_id, catIds=catIds, iscrowd=None)
        anns = self.coco.loadAnns(annIds)
        cats = self.coco.loadCats(catIds)
        train_mask = np.zeros(self.input_image_size,dtype=np.uint8)
        for a in range(len(anns)):
            className = self.getClassName(anns[a]['category_id'], cats)
            pixel_value = self.classes.index(className)+1
            new_mask = cv2.resize(self.coco.annToMask(anns[a])*pixel_value, self.input_image_size)
            train_mask = np.maximum(new_mask, train_mask)
            # train_mask = new_mask / 255.0
        return train_mask
```

```python
    def getLevelsMask(self,image_id):
      #for each category , we get the x mask and add it to mask list
      res = []
      mask = np.zeros((self.input_image_size))
      for j,categorie in enumerate(self.catIds):
        annIds = coco.getAnnIds(image_id, catIds=categorie, iscrowd=None)
        anns = coco.loadAnns(annIds)
        mask = self.getNormalMask(image_id,categorie)
        res.append( mask)
      return res

    def getImage(self,file_path):
        train_img = cv2.imread(file_path, cv2.IMREAD_ANYDEPTH)
        train_img = cv2.resize(train_img, (self.input_image_size))
        train_img = train_img.astype(np.float32) / 255.
        if (len(train_img.shape)==3 and train_img.shape[2]==3):
            return train_img
        else:
            stacked_img = np.stack((train_img,)*3, axis=-1)
            return stacked_img

    def get_image_Infos_by_path_id(self, node):
        for dict in self.image_list:
            if dict['file_name'] == node:
                return dict


    def __getitem__(self, index):
        X = np.empty((self.batch_size,128,128,3))
        y = np.empty((self.batch_size,128,128,3))
        indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]

        for i in range(len(indexes)):
          value = indexes[i]
          img_info = self.image_list[value]
          w = img_info['height']
          h = img_info['width']
          X[i,] = self.getImage(img_info['file_name'])
          mask_train = self.getLevelsMask(img_info['id'])
          for j in self.catIds:
            y[i,:,:,j] = mask_train[j]
            y[i,:,:,j] = mask_train[j]
            y[i,:,:,j] = mask_train[j]

        X = np.array(X)
        y = np.array(y)

        if self.subset == 'train':
            return X, y
        else:
            return X
```

Les améliorations que j’ai apportées sont de trois ordres :
• Mise en conformité des données aux format coco
• Développement du générateur de data pour l entraînement
• Optimisation des modèles (learning rate, fine tuning, validation,…)

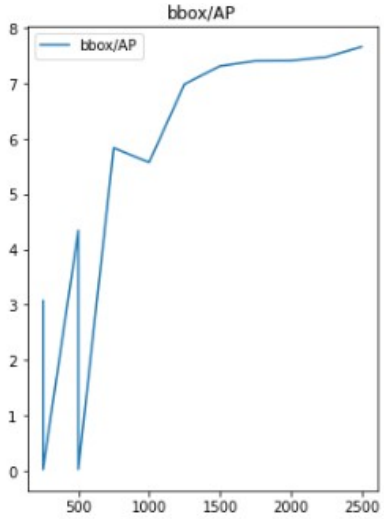
Optimisation du modèle :

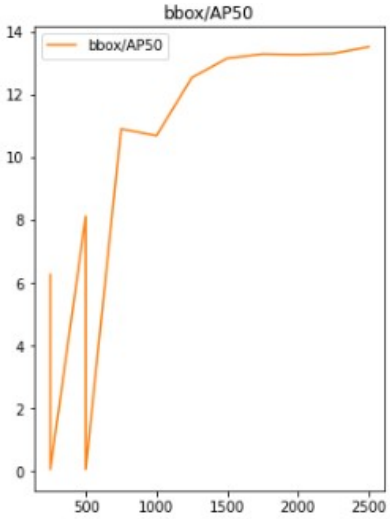
- Ajout d un call back reducelrOnplateau (reduit le learning rate pendant l entraînement)
- Test sur différents batch size (4,8,16,32)
- Early stopping(optimise l utilisation de l entraînement)
- Check point (sauvegarde le modele afin de faire des reprise a chaud, et pour l analyse de l entraînement sous tensorboard ou wandb)
- test différents type de normalisation et effet de bruitage

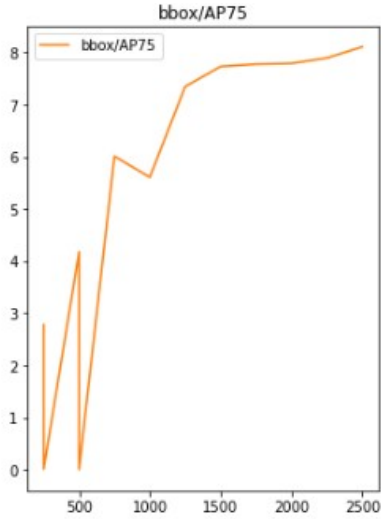
Resultats :

Detectron 2

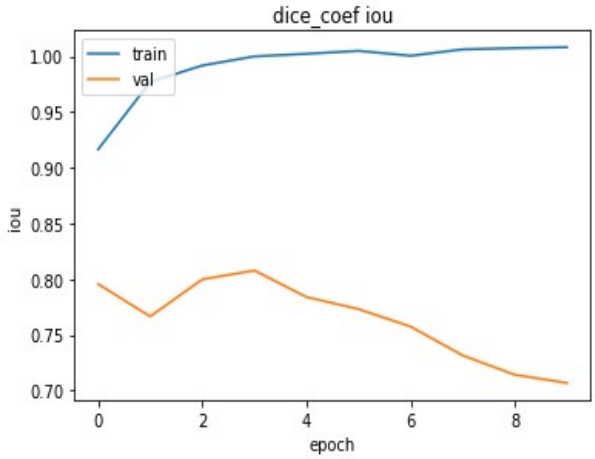
```
|  AP   |  AP50  |  AP75  |  APs  |  APm  |  APl  |
|:-----:|:------:|:------:|:-----:|:-----:|:-----:|
| 7.884 | 13.755 | 8.416  | 7.007 | 9.065 |  nan  |
[05/24 17:27:12 d2.evaluation.coco_evaluation]: Some metrics cannot be computed and is shown as NaN.
[05/24 17:27:12 d2.evaluation.coco_evaluation]: Per-category segm AP:
| category    | AP    | category    | AP    | category   | AP     |
|:------------|:------|:------------|:------|:-----------|:-------|
| small_bowel | 2.562 | large_bowel | 9.177 | stomach    | 11.913 |
OrderedDict([('bbox',
              {'AP': 7.661793684611422,
               'AP-large_bowel': 8.92021305379486,
               'AP-small_bowel': 3.019945012276293,
               'AP-stomach': 11.045222987763115,
               'AP50': 13.506734298771105,
               'AP75': 8.105517239754159,
               'APl': nan,
               'APm': 6.561584348068108,
               'APs': 8.493882660600528}),
             ('segm',
              {'AP': 7.884433215873493,
               'AP-large_bowel': 9.177498152809651,
               'AP-small_bowel': 2.562319284580608,
               'AP-stomach': 11.913482210230224,
               'AP50': 13.75526536353442,
               'AP75': 8.416054630955125,
               'APl': nan,
               'APm': 9.065446838008526,
               'APs': 7.007061411592415})])
```







Unet efficientb7



Conclusion :

L Api coco n est pas entièrement développer (notamment pour l évaluation du modèle), peut d exemple sont fourni par microsoft malheureusement

Le génération de données aux formats Coco n avait pas étais proposer pour cette compétition, j espèré que ma contribution va servir à la communauté et que quelqu'un trouvera la solution pour ajouter la data augmentation avec la stack technique Keras Unet Coco

Après quelques recherche il s avère que tensorflow est en train de développer la gestion des fichier json, mais il est toujours a titre expérimental, ce qui doit certainement expliquer le problème rencontré,
https://www.tensorflow.org/io/api_docs/python/tfio/experimental/serialization/decode_json

Detectron2 n est pas performant pour notre cas, mais très facile à implémenter

Le transfert learning avec Unet ont de bien meilleur résultats

La segmentation sémantique sous tensorflow offre moins de flexibilité que sous pytorch, notamment pour la data augmentation