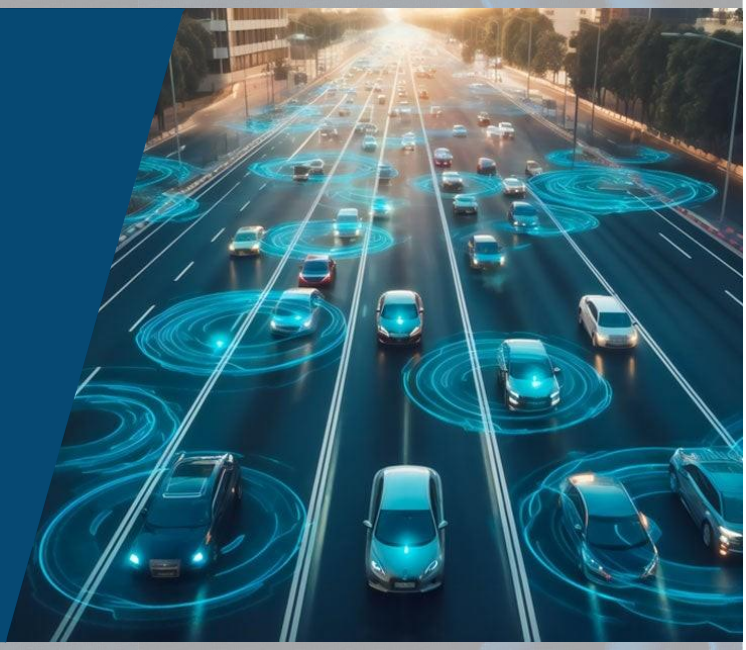
OPENCLASSROOMS

# Future Vision Transport



Note Technique

*Anthony VIDRAGO- Mars 2025*

## I – Contexte du projet :

1. Véhicule autonome

Future Vision Transport est une entreprise qui conçoit des systèmes embarqués de vision par ordinateur pour les véhicules autonomes.

Voici les différentes parties qui compose ce système :

- Acquisition des images en temps réel
- Traitement des images
- Segmentation des images (*Notre projet*)
- Système de décision

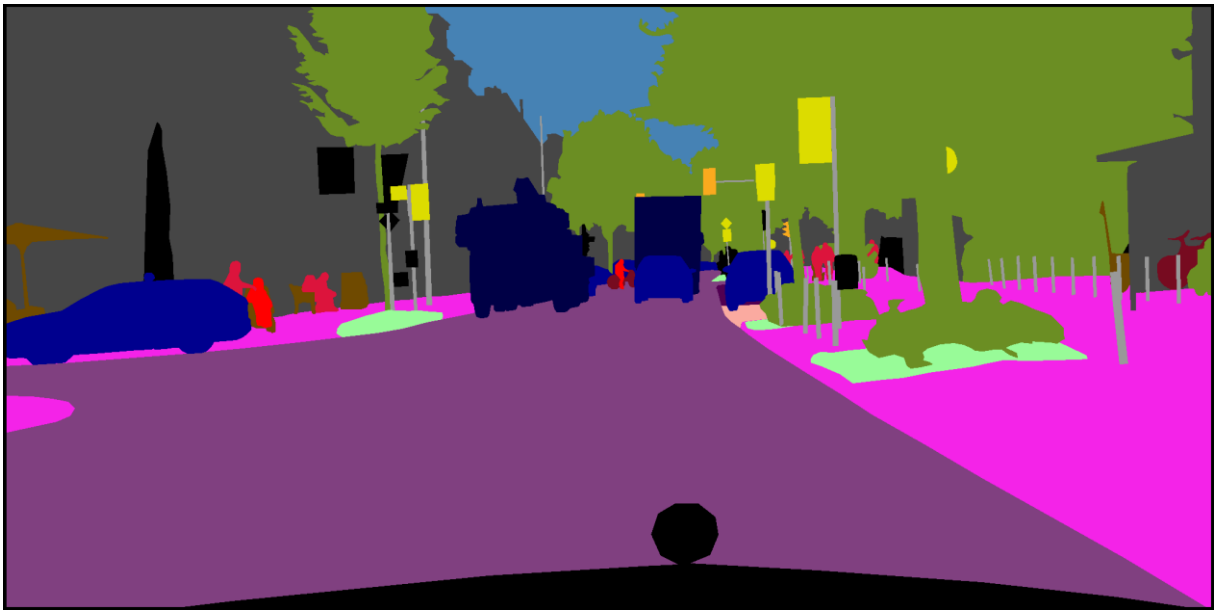
## 2. Le projet

Conception d'un modèle de segmentation d'image qui devra s'intégrer dans la chaine du système embarqué.

### a. Le jeu de données

Le dataset provient de « Cityscapes, paysages urbains » qui comporte des photos prises depuis un véhicule ainsi que les masques qui y sont associés.

Le masque d'une photo est une image dont les valeurs des pixels correspondent aux différentes catégories d'objets que l'on cherche à distinguer.



Exemple de mask

Dans cet exemple, on remarque différentes couleurs. Elles correspondent à une catégorie spécifique :

| name | id | trainId | category | catId |
|---|---|---|---|---|
| 'unlabeled' | 0 | 255 | 'void' | 0 |
| 'ego vehicle' | 1 | 255 | 'void' | 0 |
| 'rectification border' | 2 | 255 | 'void' | 0 |
| 'out of roi' | 3 | 255 | 'void' | 0 |
| 'static' | 4 | 255 | 'void' | 0 |
| 'dynamic' | 5 | 255 | 'void' | 0 |
| 'ground' | 6 | 255 | 'void' | 0 |
| 'road' | 7 | 0 | 'flat' | 1 |
| 'sidewalk' | 8 | 1 | 'flat' | 1 |
| 'parking' | 9 | 255 | 'flat' | 1 |
| 'rail track' | 10 | 255 | 'flat' | 1 |
| 'building' | 11 | 2 | 'construction' | 2 |
| 'wall' | 12 | 3 | 'construction' | 2 |
| 'fence' | 13 | 4 | 'construction' | 2 |
| 'guard rail' | 14 | 255 | 'construction' | 2 |
| 'bridge' | 15 | 255 | 'construction' | 2 |
| 'tunnel' | 16 | 255 | 'construction' | 2 |
| 'pole' | 17 | 5 | 'object' | 3 |
| 'polegroup' | 18 | 255 | 'object' | 3 |
| 'traffic light' | 19 | 6 | 'object' | 3 |
| 'traffic sign' | 20 | 7 | 'object' | 3 |
| 'vegetation' | 21 | 8 | 'nature' | 4 |
| 'terrain' | 22 | 9 | 'nature' | 4 |
| 'sky' | 23 | 10 | 'sky' | 5 |
| 'person' | 24 | 11 | 'human' | 6 |
| 'rider' | 25 | 12 | 'human' | 6 |
| 'car' | 26 | 13 | 'vehicle' | 7 |
| 'truck' | 27 | 14 | 'vehicle' | 7 |
| 'bus' | 28 | 15 | 'vehicle' | 7 |
| 'caravan' | 29 | 255 | 'vehicle' | 7 |
| 'trailer' | 30 | 255 | 'vehicle' | 7 |
| 'train' | 31 | 16 | 'vehicle' | 7 |
| 'motorcycle' | 32 | 17 | 'vehicle' | 7 |
| 'bicycle' | 33 | 18 | 'vehicle' | 7 |

```
0: 0,   # unlabeled → void
1: 0,   # ego vehicle → void
2: 0,   # rectification border → void
3: 0,   # out of roi → void
4: 0,   # static → void
5: 0,   # dynamic → void
6: 0,   # ground → void
7: 1,   # road → flat
8: 1,   # sidewalk → flat
9: 0,   # parking → void
10: 0,  # rail track → void
11: 2,  # building → construction<
12: 2,  # wall → construction
13: 2,  # fence → construction
14: 0,  # guard rail → void
15: 0,  # bridge → void
16: 0,  # tunnel → void
17: 3,  # pole → object
18: 3,  # polegroup → object
19: 3,  # traffic light → object
20: 3,  # traffic sign → object
21: 4,  # vegetation → nature
22: 4,  # terrain → nature
23: 5,  # sky → sky
24: 6,  # person → human
25: 6,  # rider → human
26: 7,  # car → vehicle
27: 7,  # truck → vehicle
28: 7,  # bus → vehicle
29: 7,  # caravan → vehicle
30: 7,  # trailer → vehicle
31: 7,  # train → vehicle
32: 7,  # motorcycle → vehicle
33: 7   # bicycle → vehicle
```

Dans notre cas nous devons travailler sur les 8 catégories principales, ce qui donnera ceci :

Après avoir défini nos 8 catégories, il faut gérer la façon dont nous allons traiter le jeu de données qui comprend 5 000 images avec des annotations précise et 20 000 images avec des annotations standard.

### b. Entrainement des modèles

Compte tenu des limites de notre matériel, nous avons travaillé sur 2500 images afin d'entrainer différents modèles.

### c. L'API

L'API que nous devons fourni devra prendre en entrée une image et renvoyé la segmentation de l'image produite par notre modèle.

### d. L’application (Streamlit)

L’application que nous utiliserons sera l’interface utilisateur pour tester l’api ainsi que les mask produit.

Elle affichera l’image réel et le mask ainsi que la légende des 8 catégories.

# II – Le principe de la segmentation d’image :

## 1. Qu’est ce que la segmentation d’image

La segmentation d’image consiste à identifier une classe au sein de l’image

Il est également possible de faire de la détection d’objet, qui consiste à indiquer la position de la classe détectée.


La segmentation d’image consiste à classifier chaque pixel et lui attribuer un label. Cela va séparer l’image en zone délimitées dont il existe deux types :

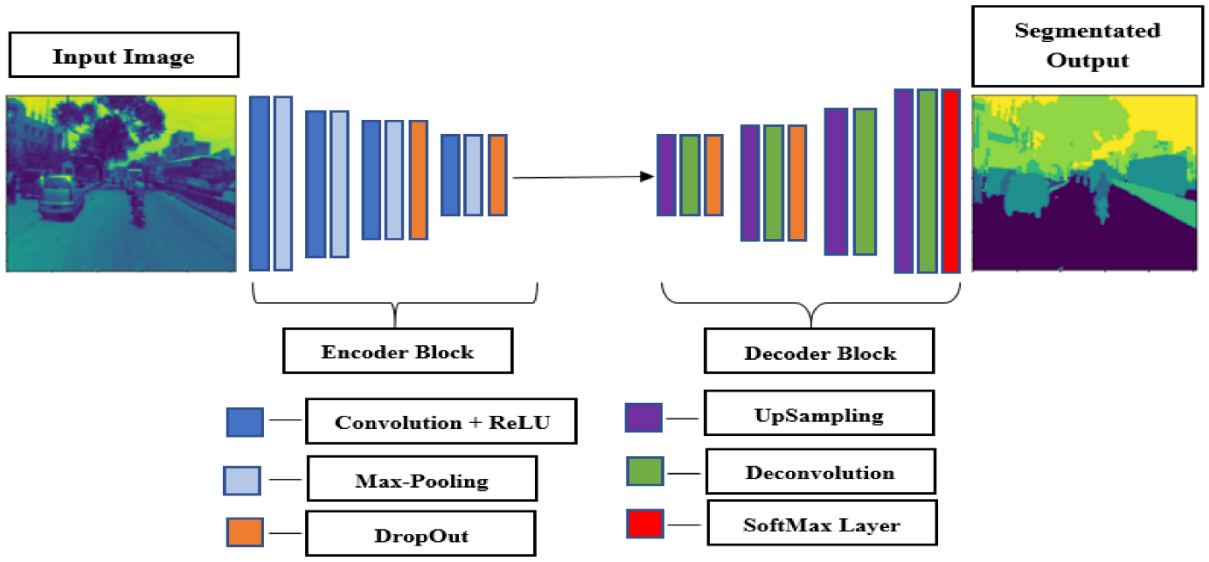
La Segmentation sémantique*, qui attribue le même label a tous les pixels des objets d’une même classe (coloration bleu pour les chats sur l’image gauche)

LA segmentation d’instance, qui applique de la détection d’objet préalable (chat 1, chat 2 et chien sur l’image de droite) Ici deux labels différents ont été attribué aux pixels des deux instances de même classe (chat)

*Le projet s’oriente sur de la segmentation sémantique.

## 2. Le fonctionnement

La segmentation d'image utilise des algorithmes basé sur l'architecture Encoder-Décoder



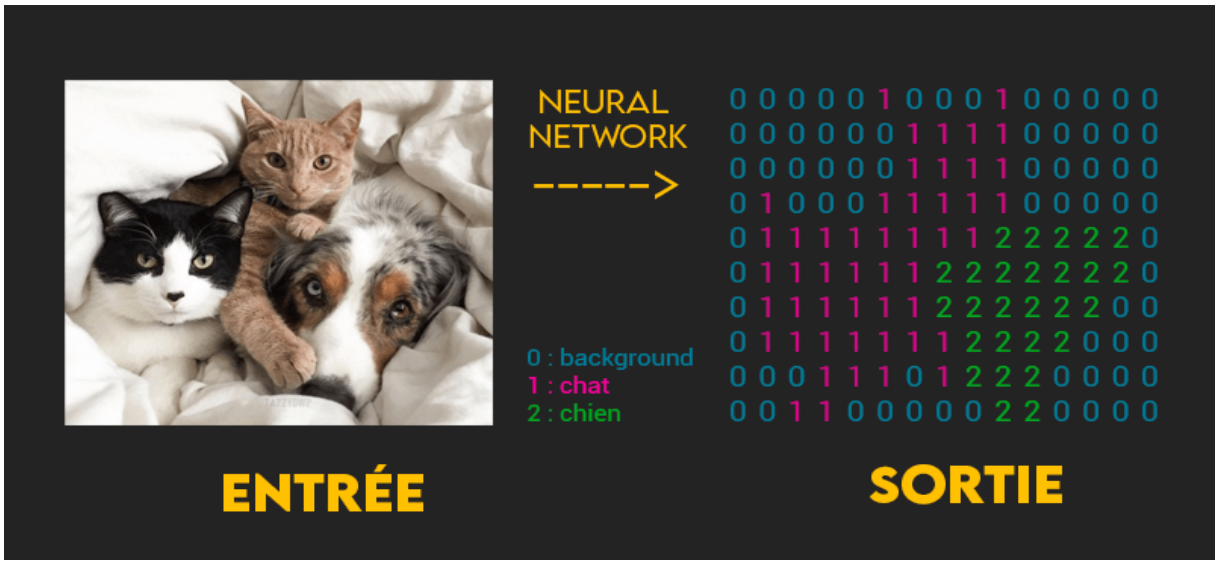

L'encoder se charge d'extraire les caractéristique des images grâce à un enchainement de couches de convolution produisant des features maps.

Des couches de Pooling réduisent la taille des images entre chaque bloc convolutif afin de résumer l'information, conserver les features les plus importantes et réduire le surapprentissage. On appelle cela le « downsampling »

Ce fonctionnement est le même que pour les modèles de classification d'image. On utilise des modèle de classification connus (sans couche fully-connected de décision) dans la plupart des cas. On parle alors de backbone.

Le décoder se charge de la phase d'upsampling. L'objectif de la segmentation est d'obtenir une classification de chaque pixel de l'image initial. Il faut donc une taille de sortie identique à la taille en entrée. Cela peut s'opérer de différentes manières (max unpooling, transposed convolution, neirest neighbors, interpolation bilinéaire, etc...).

En fin de processus, l'objectif est d'obtenir autant de maps que de catégories, sur lequel on applique une fonction de décision (ex : softmax). En retenant pour chaque pixels la catégorie la plus probable, on obtient alors une image de sortie, le mask.

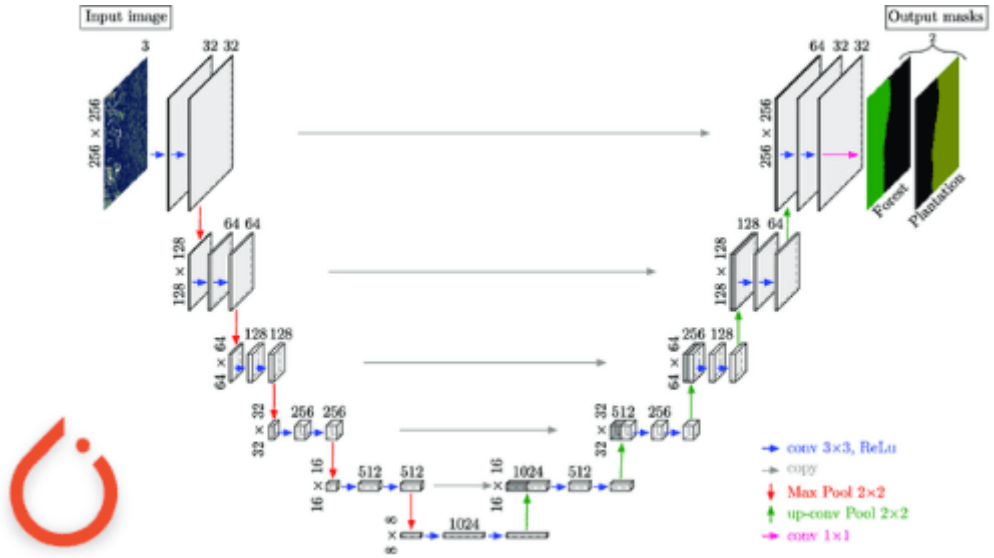


## 3. L'architecture retenu

**U-Net :** est un réseau de neurones convolutif publié en 2015, il est développé pour la segmentation d'image biomédicales. Son architecture en forme de « U » permet une segmentation précise avec un nombre réduit d'images d'entrainement. Il est largement utilisé dans le domaine médical pour la segmentation de diverse structure anatomiques.

Ce modèle dispose d'une architecture qui a permis de corrigé un problème récurrent avec d'autre plus basique : la perte d'information spatiale.

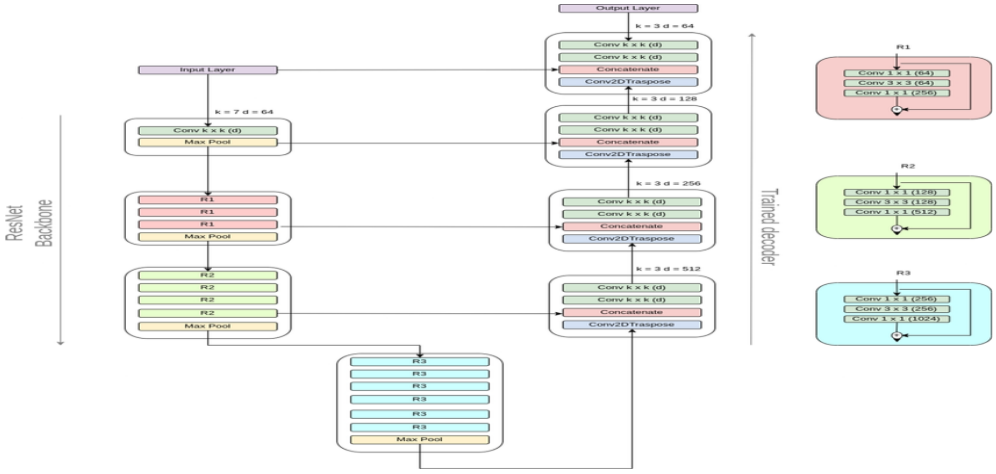
En effet, lors de la phase de downsampling, la localisation des features sur les maps est de moins en moins précise. Le souci est que contrairement à de la classification d'image, la localisation à une importance capitale pour la segmentation. Pour régler le problème, U-Net intègre des connections entre les couches de l'encoder et du décoder (par concaténation). Ainsi l'information spatiale est mieux prise en compte au cours de l'apprentissage.




## 4. Les backbones

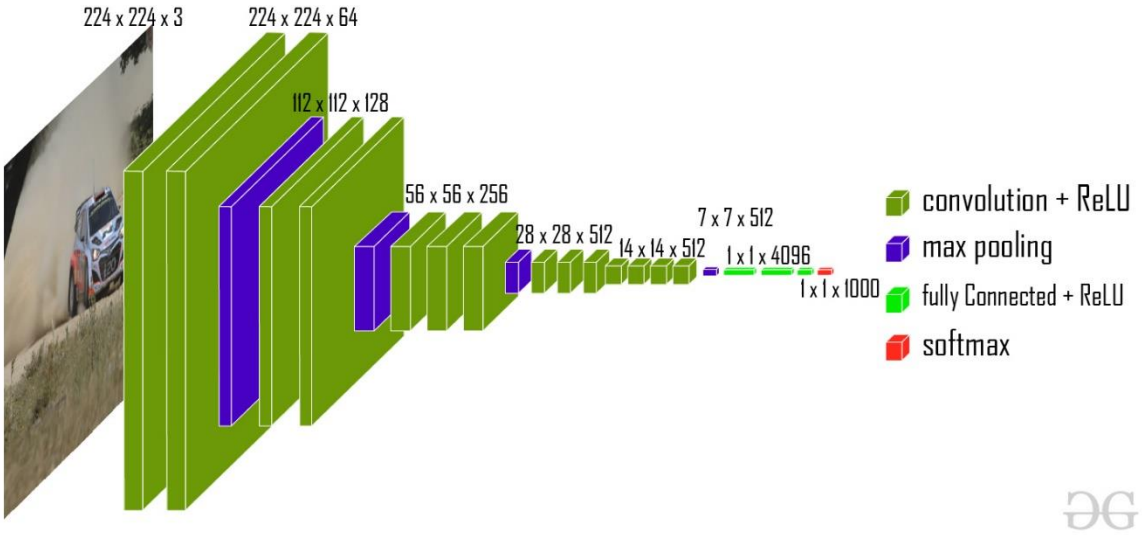
Pour ce projet nous utiliserons un Resnet50, VGG16 et l'EfficientNetB3 dont voici un exemple d'architecture :

### a. ResNet-50



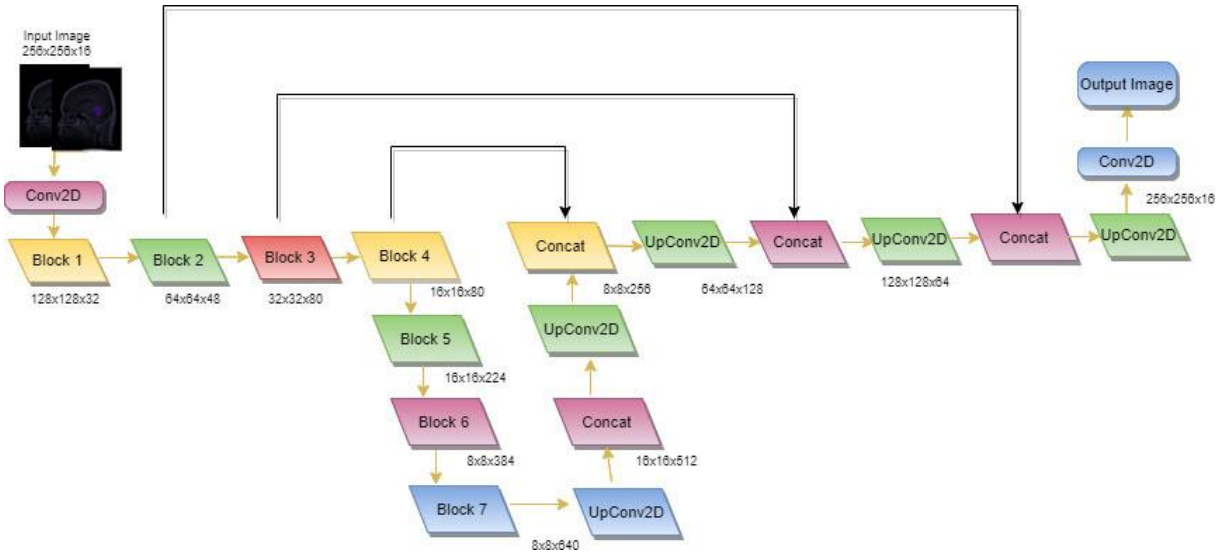
Le ResNet50 utilise des connexions résiduelles pour faciliter l'entraînement de réseaux profonds, ce qui lui permet de capturer des caractéristiques complexes pour la segmentation.

b. VGG16




Le VGG16, avec sa structure profonde et uniforme de couches convolutives 3x3.

c. EfficientNetB3




L'EfficientNetB3 est conçu pour un équilibre optimal entre précision et efficacité en ajustant la profondeur, la largeur et la résolution du réseau.

# III – La Préparation du Notebook

## 1. Les données

Le jeu de données mis a notre disposition est séparé en deux parties :

Les images, contenues dans leftImg8bit et les mask, contenu dans gtFine

Nous disposons de :

- 2975 images / mask pour l'entrainement
- 500 images / mask pour la validation

## 2. Générateur de données

Le dataset étant volumineux (11,6Go), nous créons une classe « DataGenerator » cette outils sera essentiel pour l'entrainement des modèles de segmentation d'image.

Cette classe gère le chargement des images et des mask, leur redimensionnement, l'application d'augmentations de données, le mélange des données entre les epochs et la conversion des masques en format one-hot, tout en optimisant potentiellement le chargement des masques grâce à un système de cache. Elle permet de fournir les données au modèle par lots, ce qui est nécessaire pour l'entraînement de modèles profonds sur de grands ensembles de données.

## 3. Les métriques d'évaluation

Nous utiliserons plusieurs fonction afin d'évaluer la performance des modèles de segmentation d'image (IoU, Dice, Mean IoU) et une fonction de perte pondérée lors de l'entraînement pour potentiellement améliorer la performance sur les classes importantes ou sous-représentées. Grâce à des opérations de tenseurs fournies par la librairie « tensorflow.keras.backend »

L'indice de Jaccard (plus souvent appelée IoU pour Intersection Over Union), moyenne des intersection de chaque classe prédite. L'IoU est calculé en divisant l'air de l'intersection entre la zone prédite par le modèle et la zone réel de l'objet, par l'air de leur union.

Il s'agit d'une mesure de la similarité entre ces zones, qui ce calcule donc comme le rapport entre :

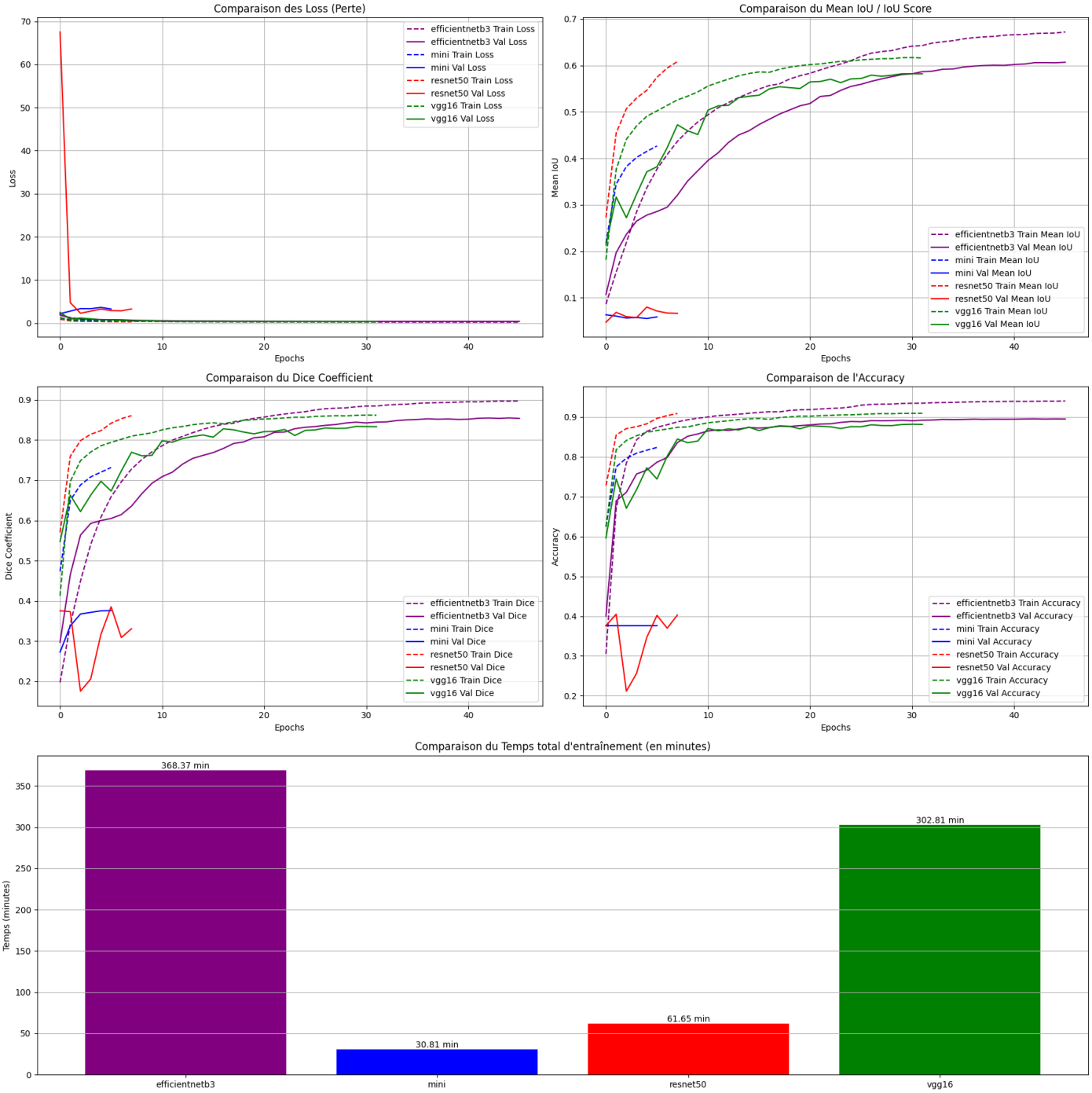
- La zone correctement prédite (les TP)
  Et
- Les zones correctement prédite (TP), incorrectement prédite (FP) et réel non détectée (FN)

$$IoU = \frac{TP}{(TP + FP + FN)}$$

Le Dice (ou scoreF1) mesure la similarité entre la prédiction et le mask. Le score varie entre 0 et 1, avec 1 indiquant un chevauchement parfait

Une fonction de perte « categorical crossentropy » est utilisé pour gérer le déséquilibre des classes dans les données d'entrainement. Les classes sous-représentées peuvent recevoir des poids plus élevés pour que le modèle apprenne mieux à les segmenter.

# IV – L'entrainement des modèles




L'efficientNetB3 est retenu ce modèle atteignant les meilleurs scores en termes de Dice Coefficient et de Mean IoU sur les données de validation. Cependant, il est aussi le modèle qui prend le plus de temps à entraîner. Le modèle "Unet-Mini" est le plus rapide à entraîner, mais il offre des performances significativement inférieures aux autres modèles.

## V – L'API et l'application Streamlit

L'application fourni une interface utilisateur graphique simple, ou l'utilisateur peut télécharger une image, l'envoyé à l'API de segmentation et visualiser l'image (mask) de la segmentation avec une coloration pour chaque catégories.


## VI – Conclusion

**Notre modèle obtiens des résultats intéressant dans le milieu urbain mais pourrait bénéficier d'entrainement sur des milieu extra-urbain.**

**Le modèle Unet datant de 2015 de nouveau modèle pourrais être testé afin de comparer les résultats avec des modèles comme Yolo, Segment Anything Model (SAM), SegNeXt.**

**L'utilisation de matériel plus performant (CPU/RAM/GPU) permettrai de travailler sur des jeux plus conséquents et des durées d'entrainement optimisé.**