


PROJET 8 : PARTICIPEZ A LA CONCEPTION D'UNE VOITURE AUTONOME

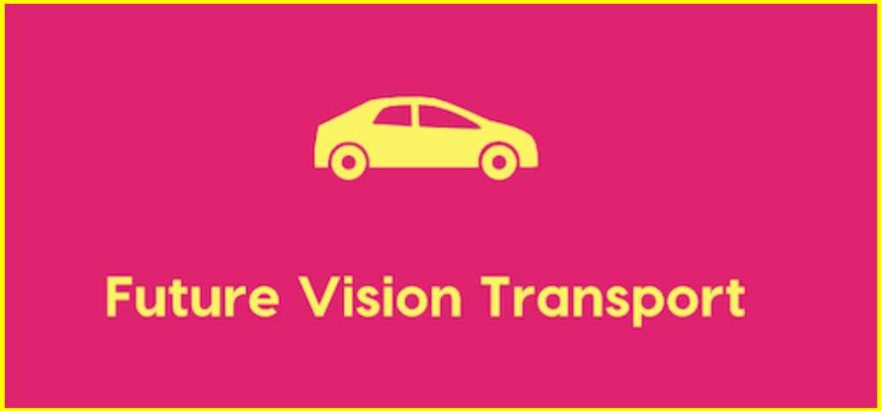
Note technique "Projet 8" Segmentation des images chez "OPENCLASSROOM"

Jaoid KRAIRI

(Mars 2022)

INTRODUCTION

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



Je suis l'un des ingénieurs IA au sein de l'équipe R&D de cette entreprise. Mon équipe est composée d'ingénieurs aux profils variés. Chacun des membres de l'équipe est spécialisé sur une des parties du système embarqué de vision par ordinateur.

Voici les différentes parties du système :

1. Acquisition des images en temps réel
2. Traitement des images
3. **Segmentation des images (c'est moi !)**
4. Système de décision

Au finale ma mission sera de travailler sur la partie de segmentation des images (3) qui est alimentée par le bloc de traitement des images (2) et qui alimente le système de décision (4).

MA MISSION

Mon rôle est de concevoir un premier modèle de segmentation d'images qui devra s'intégrer facilement dans la chaîne complète du système embarqué.

Lors d'une première phase de cadrage, j'ai récolté les avis de Franck et Laura, qui travaillent sur les parties avant et après de mon intervention, tout d'abord, Franck, en charge du traitement des images (2) me fournit les éléments suivants :

DONNEES D'ENTREES

- Un jeu de données de sortie que Franck à produit disponible sur un site internet avec un lien pour le télécharger, il contient des images originaux, des images segmentées et annotées de caméras embarquées. Néanmoins, les images segmentées ne sont pas utilisable dans l'immédiat, dans le cadre de mon projet j'aurai uniquement besoin des 8 catégories principales et non pas des 32 sous-catégories, dans cette optique, je devrai effectuer un traitement sur les images segmentées pour les faire passer de 32 sous-catégories vers 8 catégories principales.

- De plus les images en entrées peuvent changer. Le système d'acquisition n'est pas stable.

- Et pour finir le volume de donnée sera vite important.

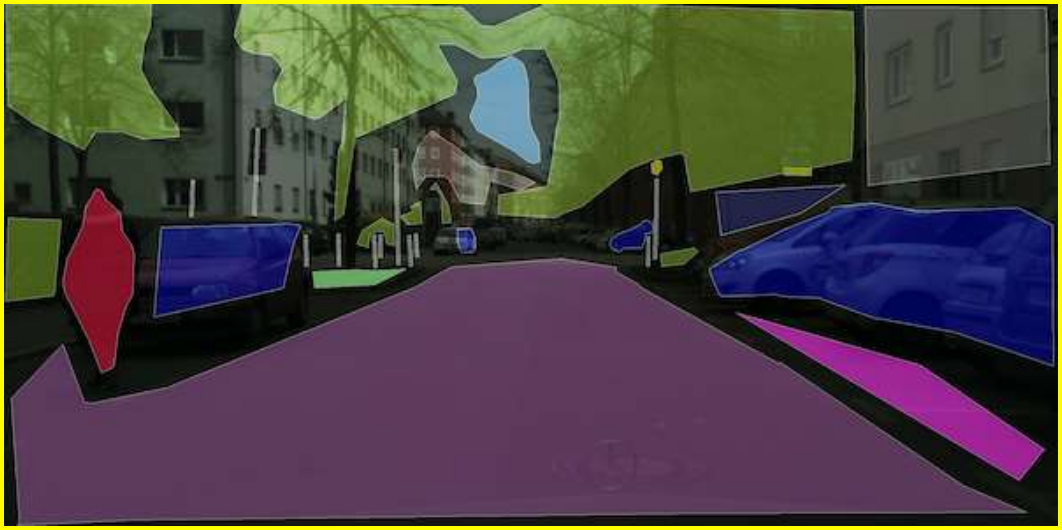
D'autre part, Laura, en charge du système de décision (4), à qui je devrai fournir les éléments suivants :

DONNEES DE SORTIES

- Elle souhaite une API simple à utiliser.

- L'API prend en entrée l'identifiant d'une image et renvoie la segmentation de l'image de l'algo, et de l'image réelle.

RECAPITULATIF



Pour récapituler, je vais dressé un plan d'action, avec les points suivants :

- Entraîner et déployer un modèle de segmentation des images sur les 8 catégories principales en utilisant Azure Machine Learning sachant que Keras est le framework de travail commun à toute l'équipe. Tout en tenant compte des contraintes des données d'entrées.

- Déployer une API Flask grâce au service Azure qui sera utilisée par Laura.

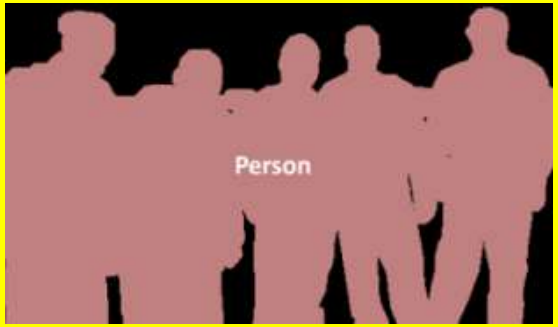
ETAT DE L'ART :

1. Qu'est-ce que la segmentation d'image ?

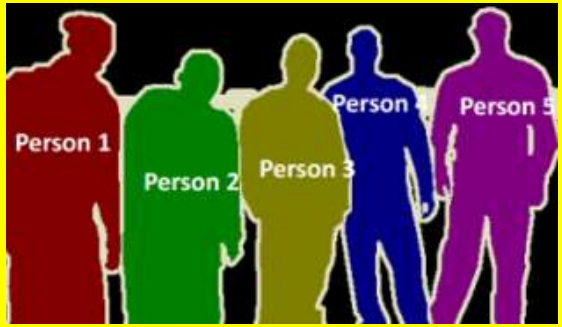
Comme le terme l'indique, il s'agit du processus de division d'une image en plusieurs segments. Dans ce processus, chaque pixel de l'image est associé à un type d'objet. Il existe deux principaux types de segmentation d'image :

- La segmentation sémantique.
- Et la segmentation d'instance.

Dans la segmentation sémantique, tous les objets du même type sont marqués à l'aide d'une étiquette de classe tandis que dans la segmentation d'instance, les objets similaires reçoivent leurs propres étiquettes séparées.



Segmentation Sémantique

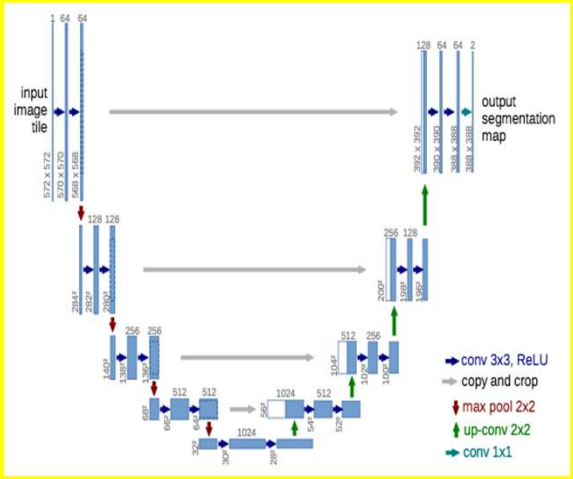


Segmentation d'Instance

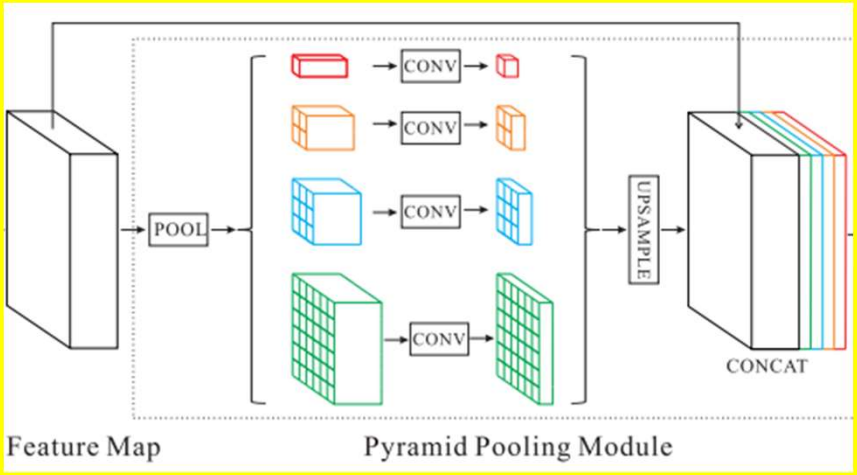
L'objet de cette note technique se réfère à la segmentation sémantique.

ETAT DE L'ART :

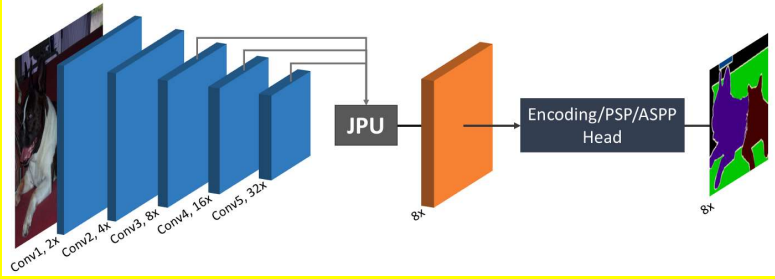
2. Les 3 types de segmentation d'image utilisées



Unet



PSPNet



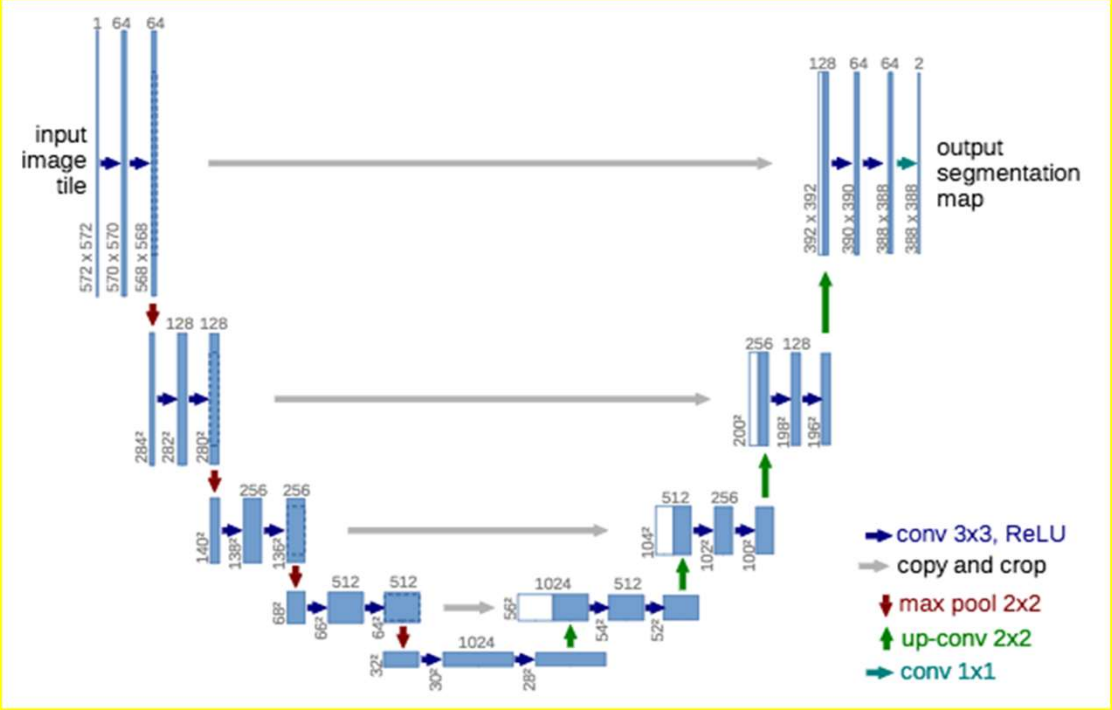
VGG16 FCN8

Les trois modèles de segmentation d'image que j'ai utilisé dans mon projet sont tout d'abord le modèle Unet qui me servira de modèle de référence afin d'évaluer des modèles plus complexes comme les modèles PSPNet et VGG16 FCN8 tout ces trois modèles utiliseront le framework Keras.
Tout d'abord l'architecture de base de tout modèle de segmentation d'image se compose d'un encodeur et d'un décodeur.

L'encodeur extrait les caractéristiques de l'image via des filtres. Le décodeur est chargé de générer la sortie finale qui est généralement un masque de segmentation contenant le contour de l'objet.

ETAT DE L'ART :

3. Comment fonctionne le modèle Unet ?



Au sujet de U-Net qui est un réseau neuronal convolutif développé à l'origine pour segmenter les images biomédicales.

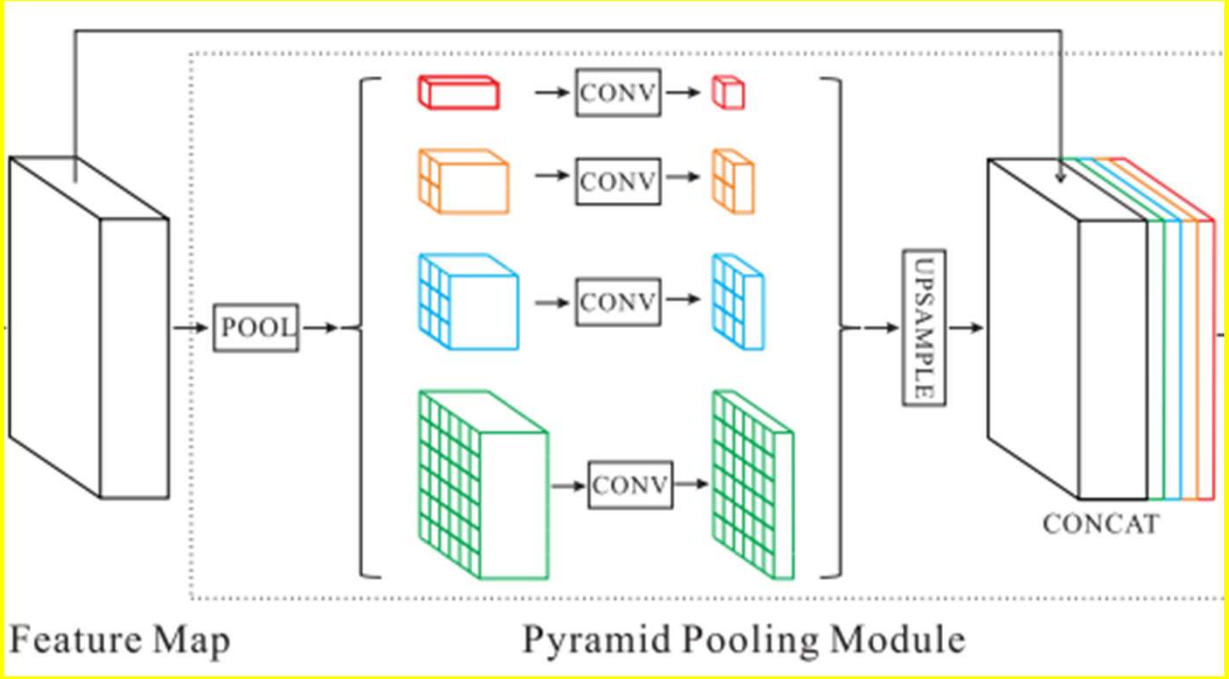
Lorsqu'elle est visualisée, son architecture ressemble à la lettre U d'où son nom U-Net.

Son architecture est composée de deux parties, la partie gauche – le chemin de contraction et la partie droite – le chemin expansif.

Le but du chemin contractuel est de capturer le contexte tandis que le rôle du chemin expansif est d'aider à une localisation précise.

ETAT DE L'ART :

4. Comment fonctionne le modèle PSPNet ?



Au sujet de PSPNet qui est un réseau d'analyse pyramidale, son but est optimisé pour apprendre une meilleure représentation globale du contexte d'une scène.

Tout d'abord, l'image est transmise au réseau de base pour obtenir une carte d'entités.

La carte des entités est sous-échantillonnée à différentes échelles.

La convolution est appliquée aux cartes d'entités regroupées.

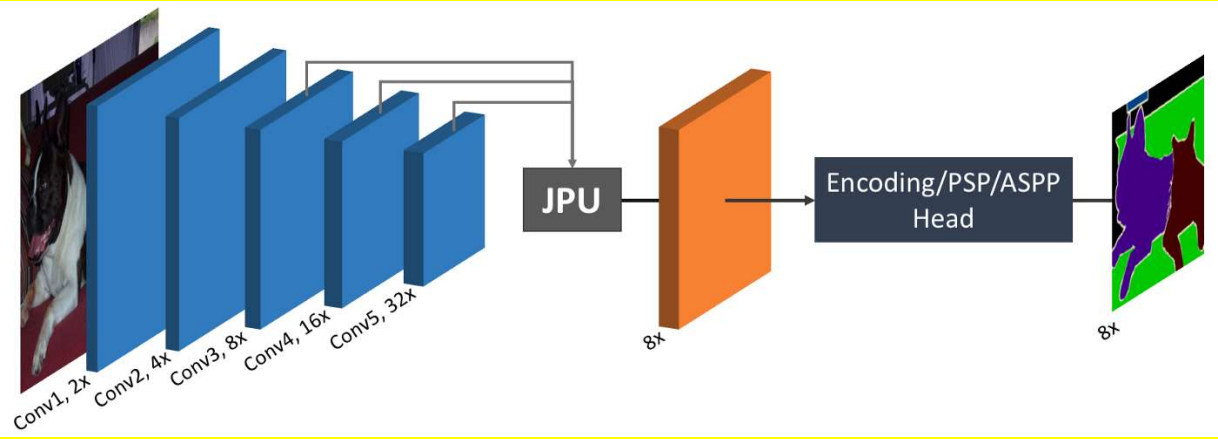
Après cela, toutes les cartes d'entités sont suréchantillonnées à une échelle commune et concaténées ensemble.

Enfin, une autre couche de convolution est utilisée pour produire les sorties de segmentation finales.

Ici, les petits objets sont bien capturés par les caractéristiques regroupées à une haute résolution, tandis que les gros objets sont capturés par les caractéristiques regroupées à une taille plus petite.

ETAT DE L'ART :

5. Comment fonctionne le modèle VGG16 FCN8 ?



Au sujet de VGG16 FCN8 qui est un réseau entièrement connecté.

Dans cette architecture, un module JPU qui est un suréchantillonnage de la pyramide articulaire est utilisé pour remplacer les convolutions dilatées car elles consomment beaucoup de mémoire et de temps.

Il utilise un réseau entièrement connecté en son cœur tout en appliquant JPU pour le suréchantillonnage.

JPU suréchantillonne les cartes d'entités basse résolution en cartes d'entités haute résolution.

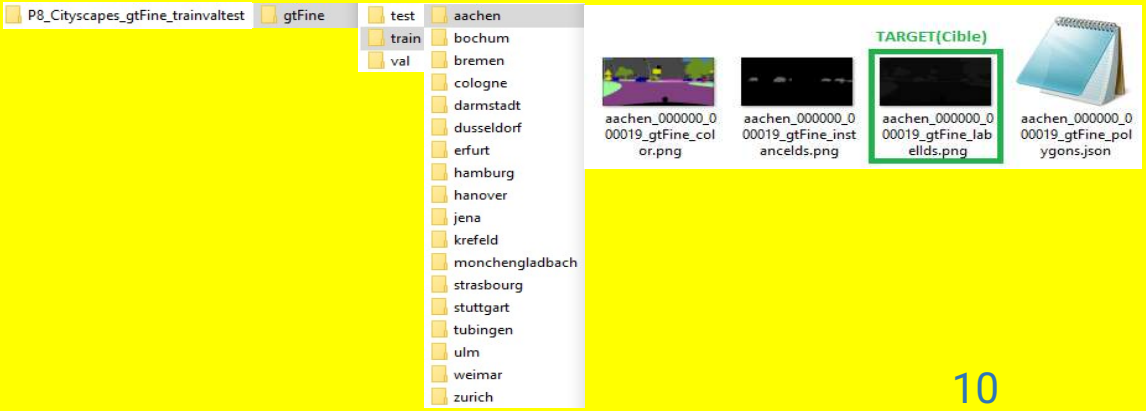
LES DIFFERENTES APPROCHES :

1. Préparation des données :

Tout d'abord, après avoir téléchargé le jeu de données disponible par l'intermédiaire d'un lien hypertexte fournit par Franck. J'ai consulté le jeu de donnée en local, il est constitué de deux dossiers 'P8_Cityscapes_leftImg8bit_trainvaltest' et 'P8_Cityscapes_gtFine_trainvaltest'.

D'autre part, le dossier 'P8_Cityscapes_leftImg8bit_trainvaltest' est constitué d'un sous dossier 'leftImg8bit' qui est constitué lui aussi de 3 sous dossiers 'test', 'train' et 'val', pour exemple, je vais m'intéresser au 'train', en explorant le 'train', je m'aperçois qu'il est constitué de 18 sous dossiers correspondant chacun à une ville, prenons l'exemple de la ville 'Aachen' comportant un certain nombre d'image, prenons la première photo comme exemple, tout d'abord, on s'aperçoit que c'est une image avec une étiquette l'identifiant, de plus dans le nom de l'étiquette on remarque le nom du sous dossier 'leftImg8bit' précédé du tiret du 8, plus le format (png), j'en conclus que pour identifier les images, je peux attribué le valeur "_leftImg8bit.png'.

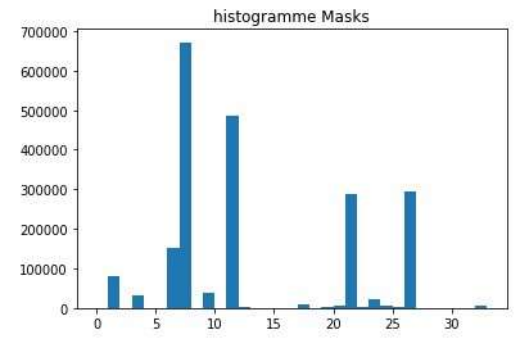
Et pour finir, le dossier 'P8_Cityscapes_gtFine_trainvaltest' est constitué d'un sous dossier 'gtFine' qui est constitué lui aussi de 3 sous dossiers 'test', 'train' et 'val', pour exemple, je vais m'intéresser au 'train', en explorant le 'train', je m'aperçois qu'il est constitué de 18 sous dossiers correspondant chacun à une ville, prenons l'exemple de la ville 'Aachen' comportant un certain nombre de fichiers, prenons les premiers fichiers comme exemple, tout d'abord, on s'aperçoit qu'il y a quatre fichiers, le fichier qui m'intéresse est la Target encadré en vert, c'est un masque avec une étiquette l'identifiant, de plus dans le nom de l'étiquette on remarque le nom du sous dossier 'gtFine' précédé du tiret du 8, plus '_labelIds' correspondant à notre variable cible, plus le format (png), j'en conclus que pour identifier les masques, je peux attribué le valeur '_gtFine_labelIds.png'.



LES DIFFERENTES APPROCHES :

2. Comment faire passer les masques de 35 sous-catégories à 8 catégories principales :

Tout d'abord, 'histogramme Masks' me permet d'identifier le nombre de sous catégories, ensuite les sous catègories sont identifié par leurs valeur pixel, que j'attribue à leurs catégories respectives.




```
cats = {
 'void': [0, 1, 2, 3, 4, 5, 6],
 'flat': [7, 8, 9, 10],
 'construction': [11, 12, 13, 14, 15, 16],
 'object': [17, 18, 19, 20],
 'nature': [21, 22],
 'sky': [23],
 'human': [24, 25],
 'vehicle': [26, 27, 28, 29, 30, 31, 32, 33, -1]}
```

Identification valeur pixel et l'associer à différente catègorie

```
def convertCats(x):
    if x in cats['void']:
        return 0
    elif x in cats['flat']:
        return 1
    elif x in cats['construction']:
        return 2
    elif x in cats['object']:
        return 3
    elif x in cats['nature']:
        return 4
    elif x in cats['sky']:
        return 5
    elif x in cats['human']:
        return 6
    elif x in cats['vehicle']:
        return 7

convertCats_v = np.vectorize(convertCats)
```

Fonction permettant d'attribuer une valeur de 0 à 7 pour chaque catègorie

```
def preprocessImg(img):
    image_matrix = np.expand_dims(img, 2)

    converted_image = convertCats_v(image_matrix)
    return converted_image
```

Fonction d'appel à la transformation du masque de 35 sous-catègories à 8 catègories, de plus la fonction cible l'axe de la matrice du masque à modifier

Valeurs pixels	Catégories
0	void(sol)
1	flat(route)
2	construction
3	object
4	nature
5	sky(ciel)
6	human
7	vehicle

LES DIFFERENTES APPROCHES :

3. Générateur de données :

Tout d'abord, si j'essayai de former un réseau avec des tonnes de données dans mon cas des images qui ne rentrent pas dans la mémoire, c'est comme cela que j'ai découvert l'utilité d'un générateur de données qui crée des données par lots et les transmet à mon réseau pour l'entraîner.

TensorFlow a son propre générateur, mais l'API est inutilement complexe et il est facile de se tromper. Je suis un grand fan de Keras car il supprime les frais généraux liés à l'apprentissage d'une API encombrante (qui changera probablement dans les prochains mois, de toute façon) et permet de se concentrer sur ma conception.

Un autre avantage de l'utilisation de Sequence la classe dans Keras comme générateur de lots est que Keras gère tout le multi-threading (filetage multiple) et la parallélisation pour s'assurer que ma formation (fond de panier) n'a pas à attendre la génération de lots. Il le fait dans les coulisses en récupérant les lots à l'avance à l'aide de plusieurs cœurs de processeur.

3-1 Identification des différents dossiers d'images dans mes images d'entrainement :

Tout d'abord, avant de créer mon générateur de données, il est important d'identifier mes différents dossiers contenant mes images et mes masques et de les stocker dans une variable nommées 'train_cities' comme le montre l'image ci-dessous.

```
train_cities = ['aachen', 'bochum', 'bremen', 'cologne', 'darmstadt', 'dusseldorf', 'erfurt', 'hamburg', 'hanover', 'jena', 'kre
```

LES DIFFERENTES APPROCHES :

3-2 Etablir les chemins des images et des masques et les charger :

```
train_img_paths = []
train_ann_paths = []

for cities in train_cities:

    train_img_dir = "jk/P8_Cityscapes_leftImg8bit_trainvaltest/leftImg8bit/train/" + cities
    train_ann_dir = "jk/P8_Cityscapes_gtFine_trainvaltest/gtFine/train/" + cities



    train_img_paths = train_img_paths + sorted(
        [
            os.path.join(train_img_dir, fname)
            for fname in os.listdir(train_img_dir)
            if fname.endswith("_leftImg8bit.png")
        ]
    )
    train_ann_paths = train_ann_paths + sorted(
        [
            os.path.join(train_ann_dir, fname)
            for fname in os.listdir(train_ann_dir)
            if fname.endswith("_gtFine_labelIds.png")
        ]
    )

print("Nombre de train images:", len(train_img_paths))

print("Nombre de train annotations:", len(train_ann_paths))
```

```
Nombre de train images: 2975
Nombre de train annotations: 2975
```

Tout d'abord, après avoir identifié les dossiers images et masques dans la variable 'train_cities', j'ai pu par la suite établir deux variables vide 'train_img_paths' correspondant aux images et 'train_ann_paths' correspondant aux masques qui apparaissent en vert, pour charger ces deux variables, j'ai créé une boucle for intégrant une variable nommé 'cities' qui apparait en violet qui à pour but de parcourir les villes présentent dans la variable 'train_cities'.
Ensuite, dans cette même boucle, j'ai créé 2 variables 'train_img_dir' correspondant aux chemin d'accès aux images en noir et 'train_ann_dir' correspondant aux chemin d'accès aux masques en marron. Cette étape m'a permis dans un premier lieu de charger mes images dans la variable 'train_img_paths' préalablement créé apparaissant en gris en ciblant la variable 'fname' égale à la valeur '_leftImg8bit.png' pour récupérer les images comme vu dans l'étape préparation des données et dans un second lieu, de charger mes masques dans la variable 'train_ann_paths' préalablement créé apparaissant en gris en ciblant la variable 'fname' égale à la valeur '_gtFine_labelIds.png' pour récupérer les masques comme vu aussi dans l'étape préparation des données.
Au final, j'ai pu charger mes images et mes masques, le nombre total d'image et identique au nombre de masques qui est de 2975.

LES DIFFERENTES APPROCHES :

3-3 Augmentation du nombre d'images et de masques:

```
def generateRandomParams(seed):
    np.random.seed(seed)
    angle = np.random.randint(26)
    np.random.seed(seed*2)
    positive = np.random.randint(2)
    sigma = np.random.uniform(0, 1)

    if positive == 0:
        angle = angle * -1

    crop = np.random.randint(3)
    crop = crop / 10

    #print(angle, crop)

    return angle, crop, sigma
```

Tout d'abord, pour l'augmentation du nombre d'images et de masques, j'ai défini une fonction nommé 'generateRandomParams' me permettant de générer 3 paramètres(angle, crop et sigma), pour ce faire dans cette fonction, j'ai initialisé le générateur de nombre aléatoire correspondant au noir, ensuite j'ai créé la variable 'angle' correspondant au marron qui renvoie un entier aléatoire de 26, ensuite j'ai réinitialisé le générateur de nombre aléatoire en le multipliant par 2 correspondant au gris, ensuite j'ai créé la variable 'positive' correspondant au rouge qui renvoie un entier aléatoire de 2, de plus j'ai créé la variable 'sigma' correspondant au rose qui renvoie un nombre aléatoire à virgule flottante compris entre 0 et 1.
Par la suite, j'ai indiqué une condition qui dit que si la variable 'positive' est strictement égale à 0, je modifie la variable angle en la multipliant par -1.
Et pour finir j'ai créé la variable 'crop' correspondant au vert qui renvoie un entier aléatoire de 3 qui sera ensuite divisé par 10.
A cette instant, ma fonction qui me génère mes paramètres est prête à être appelé.

```
for mul in range(1, imgaug_multiplier):
    for i in range(0, self.batch_size):

        angle, crop, sigma = generateRandomParams((1 + i) * mul)

        photo_aug = iaa.Sequential([
            iaa.Affine(rotate=(angle)),
            iaa.Crop(percent=(crop)),
            iaa.GaussianBlur(sigma=(0.0, sigma))
        ])

        label_aug = iaa.Sequential([
            iaa.Affine(rotate=(angle)),
            iaa.Crop(percent=(crop)),
        ])

        image_aug = photo_aug(image=x[i])
        x[batch_size * mul + i] = image_aug

        image_aug = label_aug(image=y[i])
        y[batch_size * mul + i] = image_aug
```

D'autre part, après avoir créer ma fonction me permettant de générer mes paramètres, j'ai créé une boucle for qui est intégré dans mon générateur de données, en créant une variable 'mul' qui va indiquer le nombre de fois maximum que la boucle va être parcouru, de plus à l'intérieur de cette boucle, j'ai ajouté une seconde boucle for initialiser à 0, jusqu'à atteindre le nombre de fois maximum, à l'intérieur de cette boucle figure l'appel de la fonction qui me génère mes trois paramètres(angle, crop et sigma), par la suite j'ai créé ma première variable nommé 'photo_aug' correspondant au violet, à l'intérieur de cette variable, j'effectue une séquence d'augmentation d'image en utilisant la librairie 'imgaug', à l'intérieur de cette séquence, j'utilise la méthode 'Affine' sur le paramètre 'angle' qui me permet une rotation de l'image, ensuite j'utilise la méthode 'Crop' sur le paramètre 'crop' qui me permet de rogner des images de chaque côté et pour finir j'utilise la méthode 'GaussianBlur' sur le paramètre 'sigma' qui me permet de brouiller les images. Pour la seconde variable créé nommé 'label_aug' correspondant au violet, à l'intérieur de cette variable, j'effectue la même opération sans brouiller les masques. Et pour conclure, je fait appel à ces deux variables pour créer mon augmentation d'images et de masques.

LES DIFFERENTES APPROCHES :

3-4 Réaliser le générateur de données:

```
img_size = (128 , 128)
num_classes = 8
batch_size = 2975
imgaug_multiplier = 2
```

Tout d'abord, je définis différents paramètres.

```
class Image(Sequence):
    """Itérer sur les données (en tant que matrices Numpy). """
```

Ensuite, ma classe Image hérite de la classe Sequence.

```
def __init__(self, batch_size, img_size, input_img_paths, target_img_paths):
    self.batch_size = batch_size
    self.img_size = img_size
    self.input_img_paths = input_img_paths
    self.target_img_paths = target_img_paths
```

Ensuite, je fourni des paramètres à mon générateur. Je transmet la taille du lot en tant que 'self.batch_size', la taille des images et masques en tant que 'self.img_size', les noms de fichiers d'image en tant que 'self.input_img_paths' et leurs étiquettes correspondantes en tant que 'self.target_img_paths'.

```
def __len__(self):
    return len(self.target_img_paths) // self.batch_size
```

Ensuite, cette fonction calcule le nombre de lots que ce générateur est censé produire. Donc, nous divisons le nombre total d'échantillons par 'batch_size' et renvoyons cette valeur.

```
def __getitem__(self, idx):
    """La ligne de retour (entrée, cible) correspond au batch #idx."""
    i = idx * self.batch_size
    batch_input_img_paths = self.input_img_paths[i : i + self.batch_size]
    batch_target_img_paths = self.target_img_paths[i : i + self.batch_size]

    x = np.zeros((self.batch_size * imgaug_multiplier,) + self.img_size + (3,), dtype="uint8")
    for j, path in enumerate(batch_input_img_paths):
        img = image.load_img(path, target_size=self.img_size)
        x[j] = img
    y = np.zeros((self.batch_size * imgaug_multiplier,) + self.img_size+ (1,), dtype="uint8")

    for j, path in enumerate(batch_target_img_paths):
        _img = image.load_img(path, target_size=self.img_size, color_mode="grayscale")
        y[j] = preprocessImg(_img)

    # Augmenation d'image
    for mul in range(1, imgaug_multiplier):
        for i in range(0, self.batch_size):


            angle, crop, sigma = generateRandomParams((1 + i) * mul)

            photo_aug = iaa.Sequential([
                iaa.Affine(rotate=(angle)),
                iaa.Crop(percent=(crop)),
                iaa.GaussianBlur(sigma=(0.0, sigma))
            ])

            label_aug = iaa.Sequential([
                iaa.Affine(rotate=(angle)),
                iaa.Crop(percent=(crop)),
            ])


            image_aug = photo_aug(image=x[i])
            x[batch_size * mul + i] = image_aug

            image_aug = label_aug(image=y[i])
            y[batch_size * mul + i] = image_aug

    return x, y
```

Et pour finir, étant donné le numéro de lot, 'idx' multiplier par 'self.batch_size' doit constituer une liste composée du lot de données et de la vérité-terrain. A cette étape, je lis un lot d'images de taille i renvoyant après 2 étapes successives, l'une de transformation des masques en 8 catégories principales, suivi de la seconde étapes d'augmentations des images et des masques en deux tableaux x et y.

LES DIFFERENTES APPROCHES :

3-5 Appeler le générateur de données plus d'autres étapes :

```
train_seq = Image(
    batch_size, img_size, train_img_paths, train_ann_paths
)
```

Tout d'abord, je vais généré les images et les masques, en créant une variable nommé 'train_seq' en appelant la classe Image en lui indiquant les 4 paramètres(batch_size, img_size, train_img_paths et train_ann_paths).

```
assert train_seq[0][0].shape == (batch_size * imgaug_multiplier, *img_size, 3)
assert train_seq[0][1].shape == (batch_size * imgaug_multiplier, *img_size,1)
```

Ensuite, je vais vérifier la correspondance des images avec leurs masques respectifs.

```
for x, y in train_seq:
    break
x.shape, y.shape
```

```
((5950, 128, 128, 3), (5950, 128, 128, 1))
```

Ensuite, je vais extraire les variables images et masques puis vérifier leurs formes.

```
print("Valeurs max pixels image: ", x.max())
```

```
Valeurs max pixels image:  255
```

Ensuite, je vais contrôler le nombre maximal de pixel contenue dans la variable image.

```
print("Valeurs pixels mask: ", np.unique(y))
```

```
Valeurs pixels mask:  [0 1 2 3 4 5 6 7]
```

Ensuite, je vais contrôler le nombre de catégories présents dans la variable masque.

```
image_dataset = x/255.
```

Et pour finir je vais normaliser les images.

4. Split :

```
X_train, X_test, y_train, y_test = train_test_split(image_dataset, y, test_size = 0.2, random_state = 42)
```

Tout d'abord, pour éviter, les fuites d'information entre les deux jeux de données (entraînement et test), j'ai effectué un split.

LES DIFFERENTES APPROCHES :

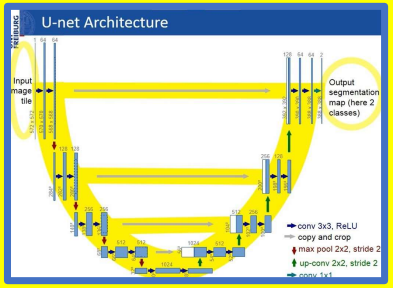
5. Modifier les canaux de sortie des mes masques :

```
train_masks_cat = to_categorical(y_train, num_classes=num_classes)
y_train_cat = train_masks_cat.reshape((y_train.shape[0], y_train.shape[1], y_train.shape[2], num_classes))

test_masks_cat = to_categorical(y_test, num_classes=num_classes)
y_test_cat = test_masks_cat.reshape((y_test.shape[0], y_test.shape[1], y_test.shape[2], num_classes))

print("Format entrée label entrainement:",y_train_cat.shape)
print("Format entrée label test:",y_test_cat.shape)
```

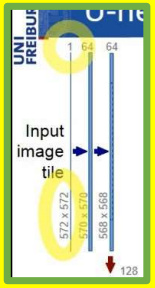
```
Format entrée label entrainement: (4760, 128, 128, 8)
Format entrée label test: (1190, 128, 128, 8)
```

Tout d'abord, je vais établir les 8 canaux de sortie avec conversion catégoriel de notre y train et y_test (afin d'obtenir le bon format d'entrée)

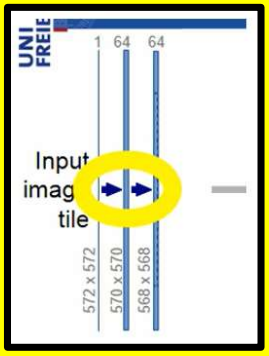
6. Exemple de création de l'un des modèles de segmentation(U-net) sur un exemple sur 2 classes :



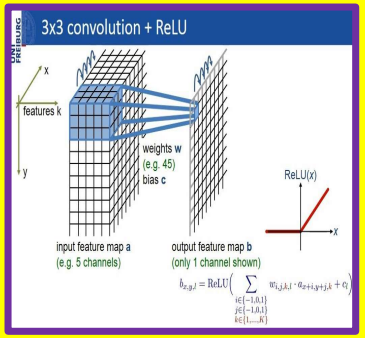
Tout d'abord, l'algorithme U-net prend en entrée une image et en sortie une carte de segmentation comme le montre l'image au cadre bleu. Comme les autres algorithmes de convolution, il consiste en un grand nombre d'opérations. L'image est insérée au niveau de 'input image' et elle se propage le long du réseau à travers tous les chemins et à la fin la carte de segmentation est faite.



Ensuite, en regardant plus en détail l'image au cadre vert, chaque boite bleu corresponde a une fonctionnalité multicanal, la taille de X et Y est en bas et le nombre de chaînes de fonctionnalités est en au haut. La plupart des opérations sont des convolutions qui sont suivis d'activations non linéaires.

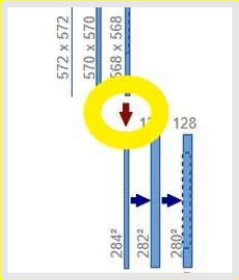


Ensuite, en regardant plus en détail l'image au cadre noir, qui correspond à une convolution standard de 3 par 3, suivie d'une activation non linéaire.

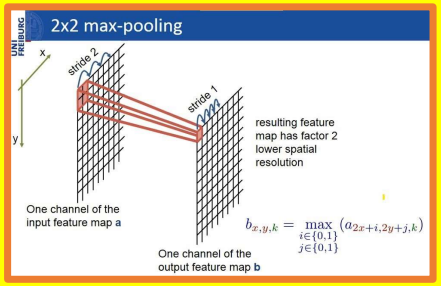


Ensuite, en regardant plus en détail l'image au cadre violet, le choix du design a été de choisir la partie valide de la convolution ce qui veut dire que pour une convolution de 3 par 3 un bord de 1 pixel et perdu. Mais cela me permet de traiter de grandes images

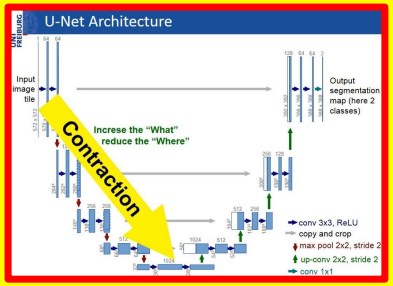
LES DIFFERENTES APPROCHES :



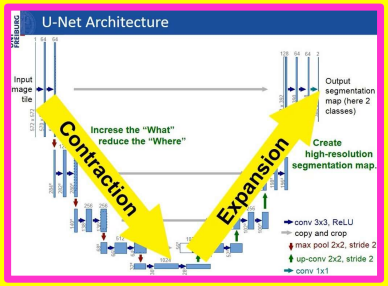
Ensuite, en regardant plus en détail l'image au cadre gris, qui correspond à l'opération de mise en commun, propage l'activation maximal sur les fenêtres deux par deux.



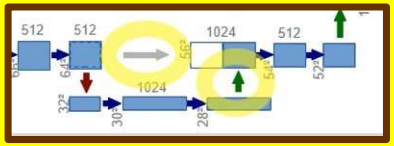
Ensuite, en regardant plus en détail l'image au cadre orange, qui correspond à l'après opération de mise en commun maximale, on augmente la quantité de chaîne de fonctionnalités par un facteur 2.



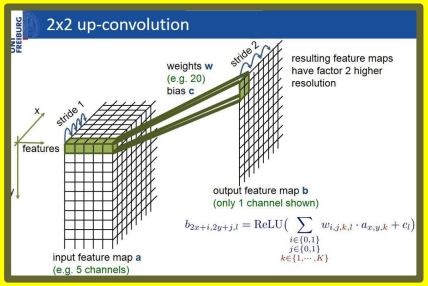
Ensuite, en regardant plus en détail l'image au cadre rouge, qui correspond à la résultante d'une contraction qui a pour effet d'augmenter le 'Quoi' et de redure le 'Où'.



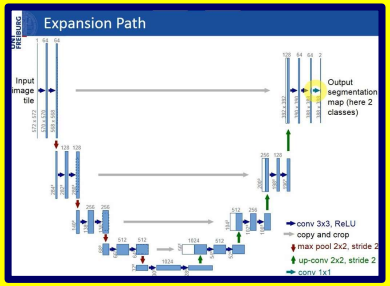
Ensuite, en regardant plus en détail l'image au cadre rose, qui correspond à la partie d'expansion qui créer une carte de segmentation de haut-résolution.



Ensuite, en regardant plus en détail l'image au cadre marron, qui correspond à la phase d'expansion consiste en des séquences de convolutions ascendantes et concaténation avec les correspondantes caractéristiques de la partie contracté.



Ensuite, en regardant plus en détail l'image au cadre gris foncé, qui correspond à la convolution ascendante utilise un pattern d'apprentissage qui uni chaque vecteur avec un fenêtre de 2 par 2 pixels suivit d'une fonction d'activation.



Ensuite, en regardant plus en détail l'image au cadre bleu foncé, qui correspond à la La carte de segmentation à 2 classes dans l'exemple, 8 pour nous.

LES DIFFERENTES APPROCHES :

7. La création de mon modèles de segmentation(U-net) :

```
def conv_block(input, num_filters):
    x = Conv2D(num_filters, 3, padding="same")(input)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    x = Conv2D(num_filters, 3, padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    return x
```

```
def encoder_block(input, num_filters):
    x = conv_block(input, num_filters)
    p = MaxPool2D((2, 2))(x)
    return x, p
```

Tout d'abord, je construis Unet en divisant l'encodeur et le décodeur en blocs dans la fonction conv_block ensuite la fonction encoder_block me permet de créer le bloc encodeur en faisant appel à la fonction conv_block suivi de maxpooling.

```
def decoder_block(input, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(input)
    x = Concatenate()([x, skip_features])
    x = conv_block(x, num_filters)
    return x
```

Ensuite, la fonction decoder_block me permet de créer le bloc décodeur , les fonctions de saut obtiennent l'entrée de l'encodeur pour la concaténation.

```
def build_unet(input_shape, n_classes):
    inputs = Input(input_shape)

    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    b1 = conv_block(p4, 1024) #Bridge

    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    if n_classes == 1:
        activation = 'sigmoid'
    else:
        activation = 'softmax'

    outputs = Conv2D(n_classes, 1, padding="same", activation=activation)(d4)
    print(activation)

    model = Model(inputs, outputs, name="U-Net")
    return model
```

Ensuite, je construit le model Unet en utilisant les blocs. Le modèle est prêt à être appeler.

```
IMG_HEIGHT = X_train.shape[1]
IMG_WIDTH  = X_train.shape[2]
IMG_CHANNELS = X_train.shape[3]
input_shape = (IMG_HEIGHT, IMG_WIDTH, IMG_CHANNELS)
```

Ensuite, je défini la variable d'entrée input_shape de mon modèle.

```
model = build_unet(input_shape, n_classes=8)
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Et pour finir, j'ai créé une variable model sur là quelle j'applique la fonction build_unet sur input_shape et le nombre de classes que je compile par la suite. Mon modèle est prêt à être entraîné sur mon jeu d'entrainement.

LES DIFFERENTES APPROCHES :

8. Résultats après entrainement de mes 4 modèles de segmentations d'images :

Après, l'entrainement de mon jeu de données sur le X_train et y_train_cat en effectuant un fit sur mes quatre modèle, voici ci-dessous les résultats obtenues.

Modèle	Nbre epochs	Temps d' Entraînement	Perte min(%)	Accuracy max(%)
U-net base	8	11 min et 18 s	34,33%	89,01%
U-net	8	21 min et 22 s	41,28%	86,90%
PSPNet	8	32 min	83,49%	67,23%
VGG16FCN8	8	24 min et 6 s	95,21%	63,02%

Pour comparer la performance des différents modèles, je vais vous présenter les 4 modèles tout d'abord le **U-net base** qui est le modèle sans augmentation d'image, ensuite le **U-net** qui est le modèle avec augmentation d'image, ensuite le **PSPNet** qui est le modèle avec augmentation d'image, et pour finir le **VGG16FCN8** qui est le modèle avec augmentation d'image.

Dans ce tableau on aperçois les différents modèle, le nombre d'epochs, le temps d'entraînement, la perte mini en pourcentage et l'accuracy max en pourcentage.

J'en conclu au vue des données présentes dans mon tableau que le modèle choisi est le **U-net base** sans augmentation d'image du fait, que **son temps d'entraînement** est le plus faible, **sa perte mini** est la plus faible et **son accuracy max** est la plus élevé.

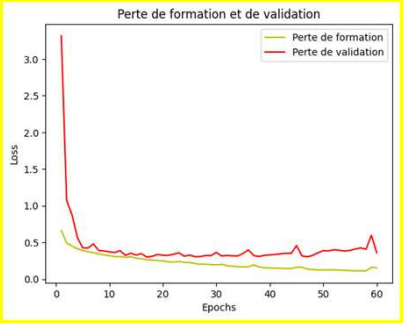
De plus pour vérifier l'apport ou non en performance de l'augmentation d'image, j'ai décidé de comparer le modèle U-net base sans augmentation d'image et le U-net avec augmentation d'image, pour avoir de meilleur performance j'ai décidé de les entraîner sur 60 epochs.

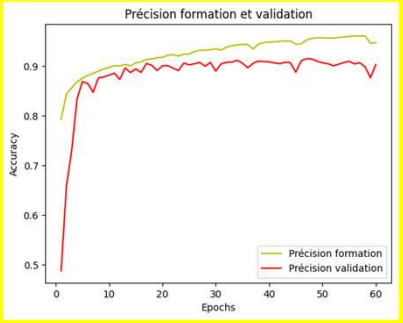
LES DIFFERENTES APPROCHES :

9. Résultat comparaison U-net base et U-net :

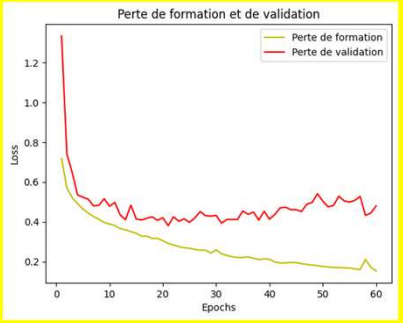
Après, l'entrainement de mon jeu de données sur le X_train et y_train_cat en effectuant un fit sur mes deux modèle, voici ci-dessous les résultats obtenues.

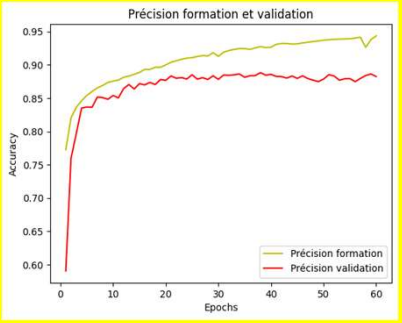
U-net base





U-net avec augmentation d'image





Modèle	Nbre epochs	Temps d' Entraînement	Perte min(%)	Accuracy max(%)	Moyenne IoU(%)
U-net base	60	1h 18 min	15,26%	94,72%	69,70%
U-net	60	2h 34 min et 48 s	15,34%	94,35%	65,64%

Pour comparer la performance des deux modèles U-net, je vais d'une part m'attarder sur ces deux modèles tout d'abord, le **U-net base** qui est le modèle sans augmentation d'image, on voit sur ces 2 premiers graphiques, que se soit la perte de formation et validation ou la précision de formation et de validation que le modèle U-net base est en sur apprentissage à partir du 20ème epochs, d'autre part le **U-net** qui est le modèle avec augmentation d'image, on voit sur ces 2 deuxième graphiques, que se soit la perte de formation et validation ou la précision de formation et de validation que le modèle U-net est en sur apprentissage à partir du 20ème epochs.

Dans ce tableau, je vais vous présenter les 2 modèles, **U-net base et U-net**.

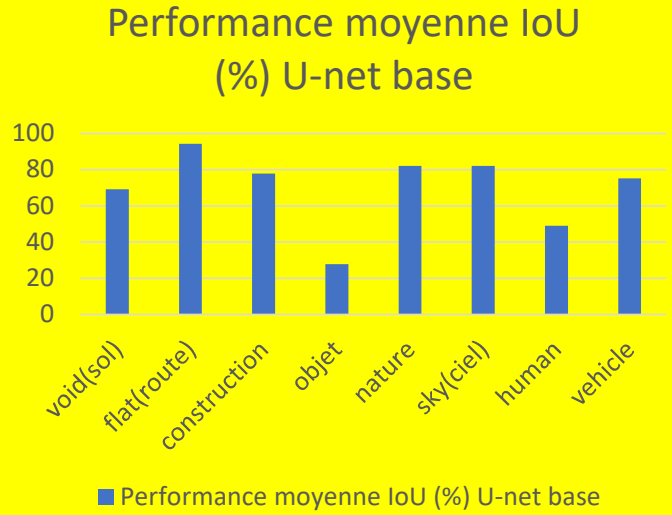
On aperçois les modèles, le nombre d'epochs, le temps d'entraînement, la perte mini en pourcentage, l'accuracy max en pourcentage et la Moyenne IoU en pourcentage qui est une métrique Keras qui me permet d'évaluer la performance moyenne de la segmentation sémantique des images.

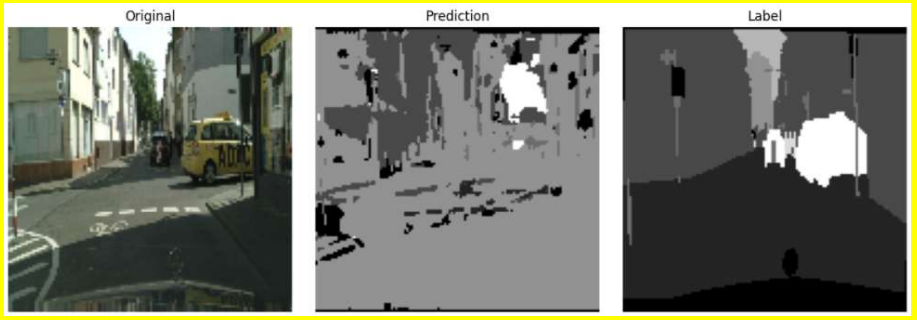
J'en conclu au vue des données présentes dans mon tableau que le modèle choisi est encore le **U-net base** sans augmentation d'image du fait, que **son temps d'entraînement** est le plus faible, **sa perte mini** est la plus faible, **son accuracy max** est la plus élevé et sa moyenne IoU est la plus élevé.

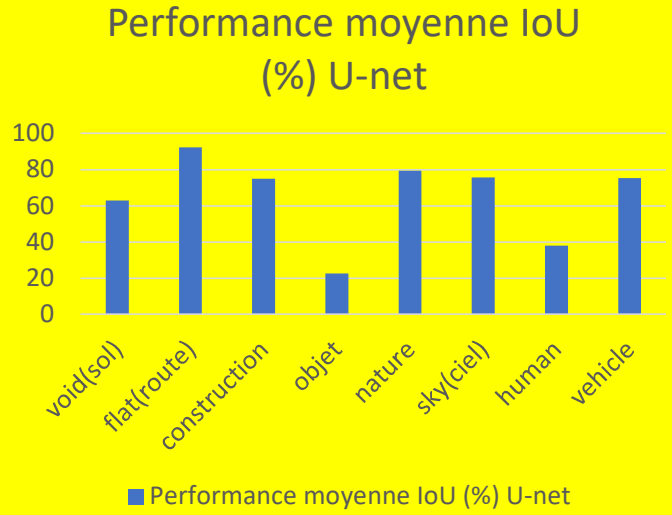
J'en conclu, que l'apport de l'augmentation d'image, n'améliore pas la performance, de plus, pour avoir plus de détails sur la moyenne IoU, je vais vous présenter la performance par catégorie.

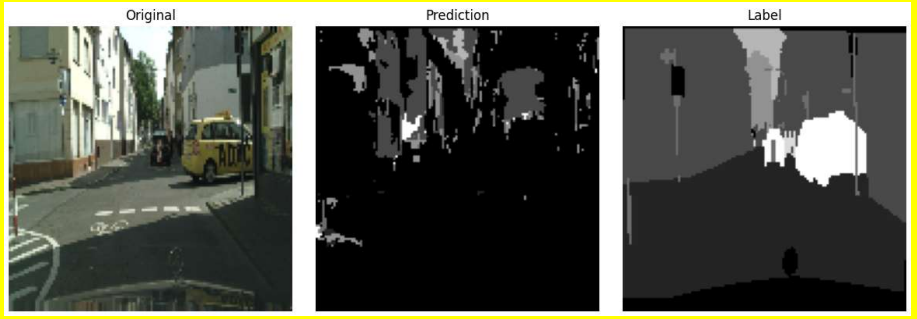
LES DIFFERENTES APPROCHES :

10. Comparer la prédiction U-net base et U-net sur une image en affichant la métrique moyenne IoU par catégorie sur les données test :









Je constate que sur les 2 modèles la catégorie la mieux détecté est la route et la moins bien détecté est l'objet.

LES DIFFERENTES APPROCHES :

11. Déploiement du model grâce à une API Flask:

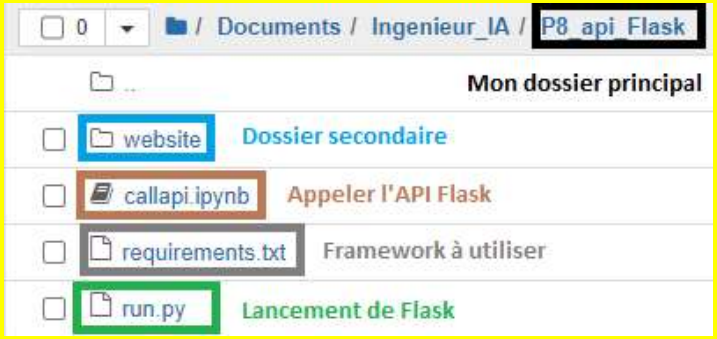
A quoi sert Flask?

Flask est un micro-framework python facile et simple qui permet de faire des applications web évolutives étant à la fois très puissant et léger. Par exemple, il permet de créer une application web minimale en 7 lignes de code comme le montre l'image ci-dessous

```python
from flask import Flask

app = Flask(__name__)

@app.route('/')
def index():
    return "Hello world !"

if __name__ == "__main__":
    app.run()
```

Le code apparaissant dans cette image, lance un serveur web disponible dans le port 5000. En bref, il affichera "Hello World" à l'adresse "localhost:5000".

Pour revenir, au déploiement du model choisi grâce à l'api Flask, j'ai dû dans un premier lieu définir mon arborescence, afin de simplifier mon approche de l'API Flask. Comme on le voie ci-dessous, j'ai tout d'abord créer un dossier principal encadrer en noir nommé « P8_api_Flask »contenant tous les éléments, ensuite à l'intérieur de ce dossier j'ai 4 éléments un dossier secondaire nommé « Website » encadré en bleu, un fichier python nommé « callapi » encadré en marron me permettant d'appeler l'API Flask, un fichier texte nommé « requirements » encadré en gris me permettant de définir les frameworks à utiliser et pour finir le fichier python nommé « run » encadré en vert me permettant le lancement de flask. De plus je vais m'intéresser plus en détail au dossier secondaire encadré en bleu qui est constitué de 4 éléments, tout d'abord le dossier nommé « static » encadré en rouge qui me permet de stocker les images et le modèle, ensuite le dossier nommé « templates » encadré en rose contient un fichier HTML recevant des objets python correspondant à ma page d'accueil et qui est lié à la vue, ensuite un fichier python nommé « __init__ » encadré en violet qui me permet de construire l'instance et pour finir un fichier python nommé « views » encadré en orange me permettant de générer le contenu à renvoyer correspondant à la vue.



LES DIFFERENTES APPROCHES :

11. Déploiement du model grâce à une API Flask:

Projet 8 : Appeler l'API Flask pour segmentation des images

Importation de la librairie

Entrée [1]:
```
import requests
from tensorflow import keras
import json
```

Indiquer le chemin d'une image

Entrée [2]:
```
image_path = './website/static/test_img_3.png'
```

Lors du lancement de flask un URL port 5000 a été créé pour la page d'accueil

1- Générer le contenu revoyé par la vue URL port 5000 plus predict

2-Ouvrir l'image

3-Effectuer une requête pour poster l'image

Entrée [3]:
```
url = 'http://127.0.0.1:5000/predict'
my_img = {'image': open(image_path, 'rb')}
r = requests.post(url, files=my_img)
```

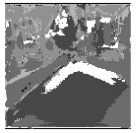
4-Charger la prédiction au format Json

Entrée [4]:
```
data = json.loads(r.json()['data'])
```

5-Afficher la prédiction du masque

Entrée [5]:
```
keras.preprocessing.image.array_to_img(
    data, data_format=None, scale=True, dtype=None,
    )
```

Out[5]:



```
azureuser@krairij1:~/cloudfiles/code/Users/Krairij/p8_api_flask$ python run.py
2022-03-22 05:27:06.681184: E tensorflow/stream_executor/cuda/cuda_driver.cc:271] failed
 detected
2022-03-22 05:27:06.681254: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] ke
1): /proc/driver/nvidia/version does not exist
2022-03-22 05:27:06.699553: I tensorflow/core/platform/cpu_feature_guard.cc:151] This Ten
xary (oneDNN) to use the following CPU instructions in performance-critical operations:
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flag
 * Serving Flask app "website.views" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
 * Restarting with stat
2022-03-22 05:27:24.054423: E tensorflow/stream_executor/cuda/cuda_driver.cc:271] failed
 detected
2022-03-22 05:27:24.054490: I tensorflow/stream_executor/cuda/cuda_diagnostics.cc:156] ke
1): /proc/driver/nvidia/version does not exist
2022-03-22 05:27:24.055080: I tensorflow/core/platform/cpu_feature_guard.cc:151] This Ten
xary (oneDNN) to use the following CPU instructions in performance-critical operations:
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flag
 * Debugger is active!
 * Debugger PIN: 696-188-572
127.0.0.1 - - [22/Mar/2022 05:27:40] "POST /predict HTTP/1.1" 200 -
```

Dans un premier lieu, pour lancer l'API Flask, je dois ouvrir le terminal, accéder au dossier nommé « P8_api_Flask », ensuite lancer Flask en effectuant un « python run.py » encadré vert cette étape va me créer un url « http://127.0.0.1:5000/ » encadré jaune correspondant à ma page d'accueil, a cette instant je peux ouvrir le fichier python nommé « callapi » à l'intérieur de ce fichier j'ai une en tête nommé « Projet 8: Appeler l'API Flask pour segmentation des images » , qui contient la librairie nécessaire, le chemin d'une image, le contenu générer par la vue, l'ouverture de l'image, la requête pour poster l'image, le chargement de la prédiction au format Json et pour finir l'affichage de la prédiction. De plus l'encadré bleu figurant sur mon terminal m'indique que tout c'est bien déroulé.

CONCLUSION :

Pour conclure, j'ai privilégié le modèle U-net sans augmentation d'image car celui-ci offre de bonnes performance et surtout une grande rapidité d'exécution, ce qui est primordial dans l'analyse des images d'une voiture autonome. Cependant, nous avons vu que certaines catégories était difficiles à détecter, il me semble que cela est dû à la finesse de l'objet étant donné que je réduisais l'image, il serait pertinent d'utiliser une résolution plus importante tout en gardant la performance de traitement.