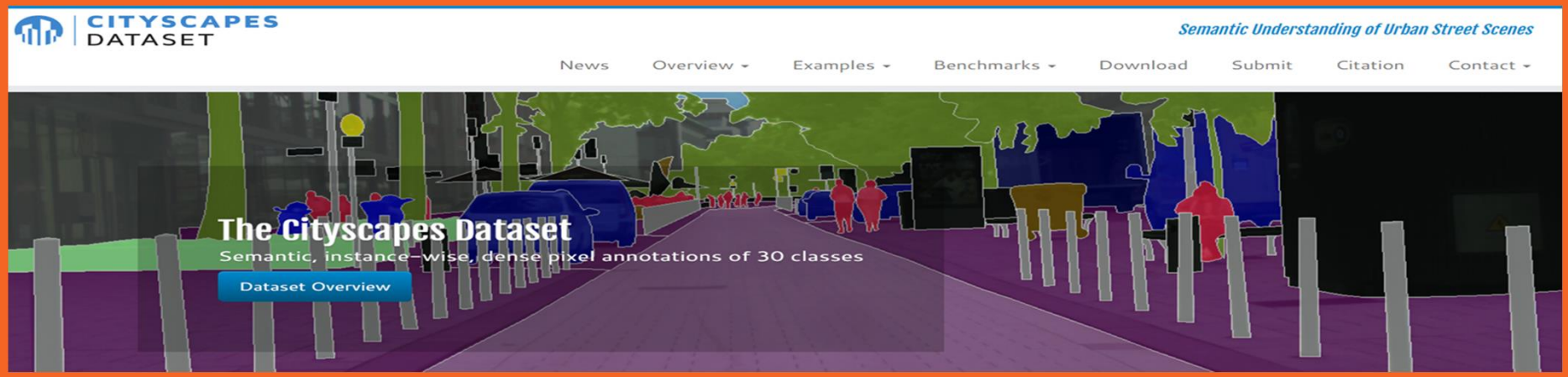


Projet 08 :

Participez à la conception d'une voiture autonome

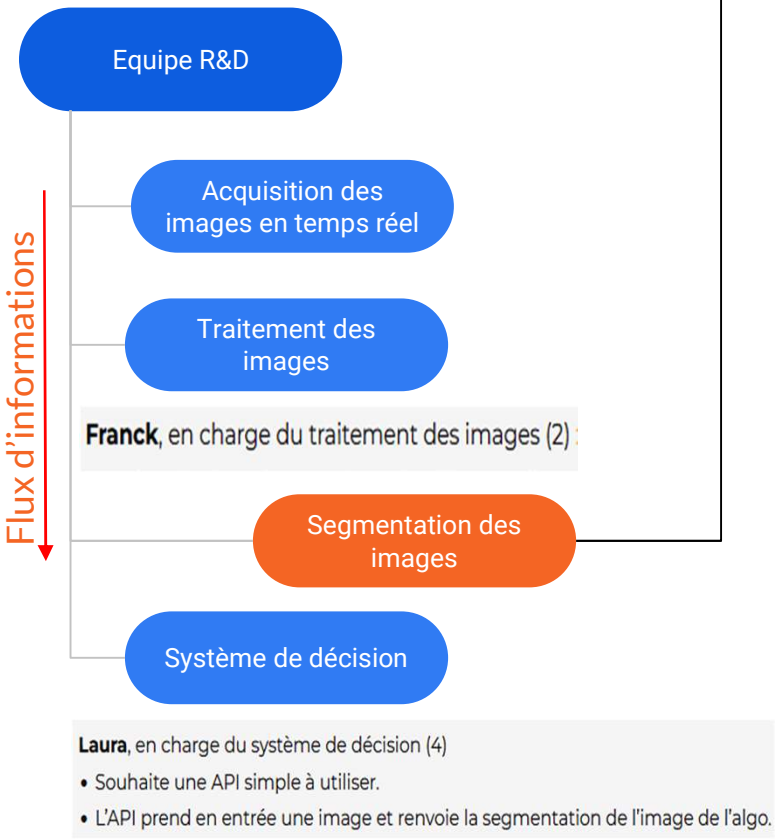
Mohamed A.

Contexte Projet

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

Entreprise
Future Vision Transport

L'équipe R&D à la tête du projet regroupe des ingénieurs de profil différents. chacun des membres de l'équipe est spécialisé dans une des parties du système embarqué de vision par ordinateur.
Le schéma ci contre indique la tâche de chaque équipe ainsi que la catégorie à laquelle je suis affecté.



Votre mission

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

Le plan d'action englobe les points suivants :

- **entraîner un modèle de segmentation** des images sur les 8 catégories principales. **Keras** est le framework de travail commun à toute l'équipe. *Attention aux contraintes de Franck !*
- concevoir une **API de prédiction (Flask ou FastAPI)** qui sera utilisée par Laura et la déployer sur le Cloud **(Azure, Heroku, PythonAnywhere ou toute autre solution)**. Cette API prend en entrée une image et renvoie le mask prédit (segments prédits de l'image).
- concevoir une **application web Flask de présentation des résultats** et la déployer sur le Cloud **(Azure, Heroku, PythonAnywhere ou toute autre solution)**. Cette application sera l'interface pour tester l'API et afficher les images et masks

Sommaire :

1 Présentation du dataset Cityscapes

Le dataset cityscapes est dédié à la compréhension sémantique des scènes urbaines

- ❏ Segmentation sémantique dense
- ❏ Instance segmentation pour les personnes et véhicules.
- ❏ 30 classes d'annotations
- ❏ 50 villes concernées
- ❏ divers saisons et situations d'images (automne , été, pluies)
- ❏ frames sélectionnés à la main
- ❏ 5000 images avec annotations de haute qualité
- ❏ 20000 images avec annotation de moyenne qualité

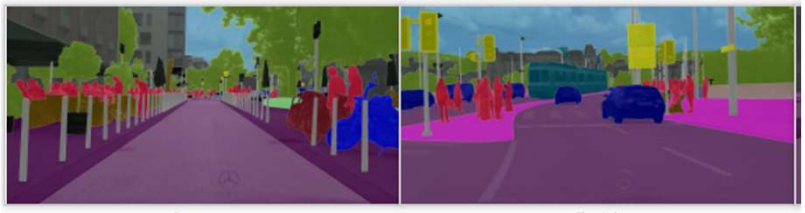
Villes incluses

1 Présentation du dataset Cityscapes

Segmentation moyenne qualité

Saarbrücken *Saarbrücken*

Segmentation haute qualité



Stuttgart *Zurich*

Classes annotées dans les images

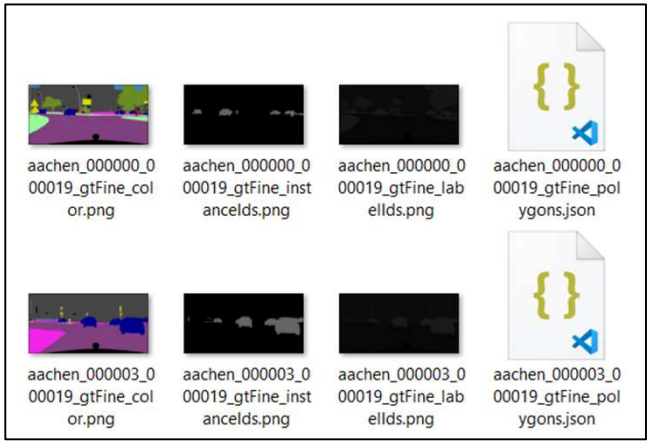
Class Definitions

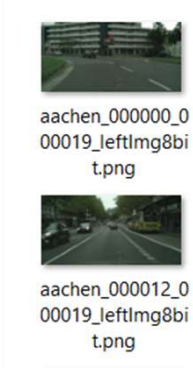
Please click on the individual classes for details on their definitions.

Group	Classes
flat	road · sidewalk · parking+ · rail track+
human	person* · rider*
vehicle	car* · truck* · bus* · on rails* · motorcycle* · bicycle* · caravan*+ · trailer*+
construction	building · wall · fence · guard rail+ · bridge+ · tunnel+
object	pole · pole group+ · traffic sign · traffic light
nature	vegetation · terrain
sky	sky
void	ground+ · dynamic+ · static+

1 Présentation du dataset Cityscapes

Composition des dossiers dataset et sa structure originale téléchargée





2 Présentation des modèles Deep Learning & Code utilisé & résultats

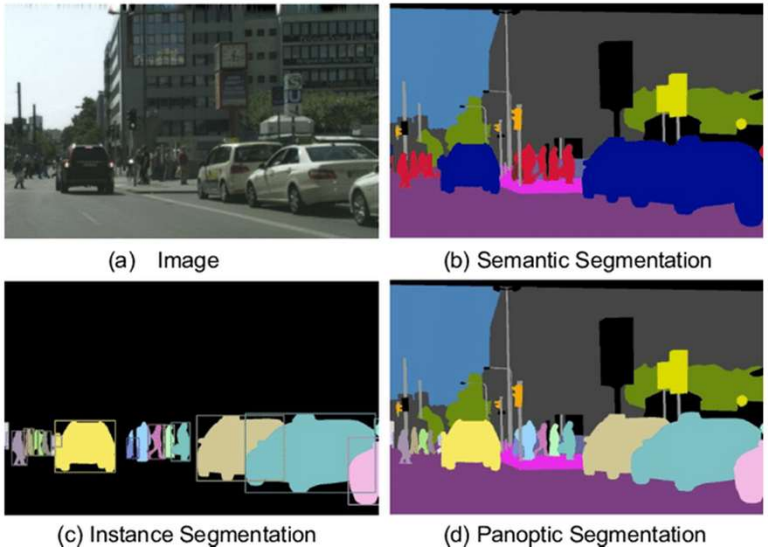
- La segmentation sémantique
- Les modèles Deep learning utilisés pour la segmentation sémantique
- Présentation du code utilisé dans le projet

❑ La segmentation sémantique

La segmentation d'images est une technique de computer vision qui consiste à découper de façon automatique une image en zones de pixels appartenant à une même classe d'objets. La segmentation d'images a de nombreuses applications, notamment en imagerie médicale, imagerie satellitaire et voitures autonomes.

La segmentation d'images se divise en deux types, la segmentation sémantique (semantic segmentation) et la segmentation par instance (instance segmentation). Dans le cadre de la segmentation sémantique, on cherche à classifier les pixels de l'image comme appartenants ou non à une certaine catégorie. La segmentation par instance en revanche, permet d'avoir plus d'informations sur l'image, en divisant les différentes instances d'un même objet.

Lorsque l'on souhaite classifier toutes les zones de l'image, en divisant les instances, on parle de segmentation panoptique (panoptic segmentation). Elle consiste en une combinaison des deux types de segmentation.



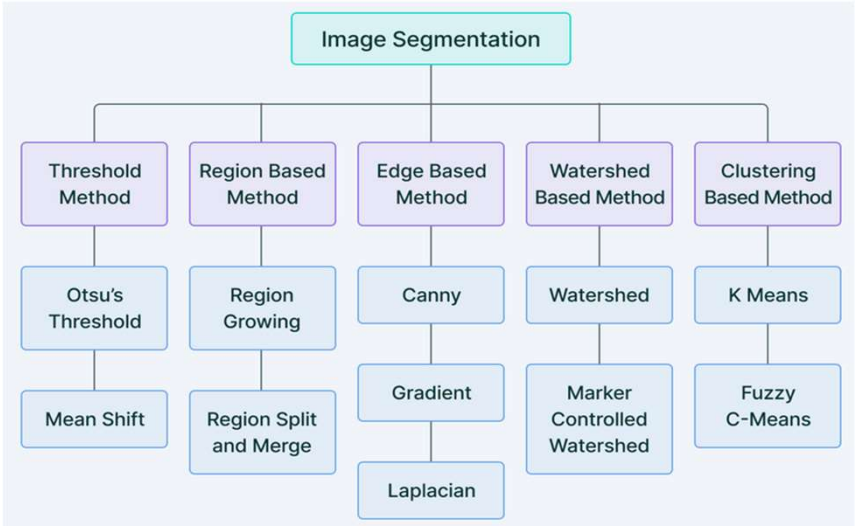
2 Présentation des modèles Deep Learning & Code utilisé & résultats

❑ La segmentation sémantique

1. Thresholding Segmentation
2. Edge-Based Segmentation
3. Region-Based Segmentation
4. Watershed Segmentation
5. Clustering-Based Segmentation Algorithms

Traditional

https://www.v7labs.com/blog/image-segmentation-guide#h5



Modern

1. Neural Networks for Segmentation

Utilisation des modèles Deep learning extrêmement évolués et puissants pour ce type de tâche.

2 Présentation des modèles Deep Learning & Code utilisé & résultats

https://github.com/qubvel/segmentation_models

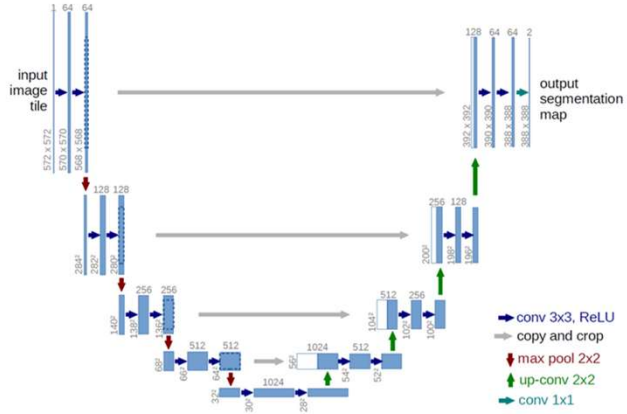
Modèle de Réseau de Neurones Entièrement Convolutif. Ce modèle fut initialement développé par Olaf Ronneberger, Phillip Fischer, et Thomas Brox en 2015 pour la segmentation d'images médicales.

L'**architecture de U-NET** est composée de deux » chemins « . Le premier est le chemin de contraction, aussi appelé encodeur. Il est utilisé pour capturer le contexte d'une image.

Il s'agit en fait d'un **assemblage de couches de convolution** et de couches de » max pooling » permettant de créer une carte de caractéristiques d'une image et de réduire sa taille pour diminuer le nombre de paramètres du réseau.

Le second chemin est celui de l'expansion symétrique, **aussi appelé décodeur**. Il permet aussi une localisation précise grâce à la convolution transposée.

UNet was created by Olaf Ronneberger, Philipp Fischer, and Thomas Brox at the University of Freiburg in Germany.



UNet architecture as presented on the homepage of one of the authors. Source: https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

2 Présentation des modèles Deep Learning & Code utilisé & résultats

https://github.com/qubvel/segmentation_models

Segmentation models is python library with Neural Networks for Image Segmentation based on Keras (Tensorflow) framework.

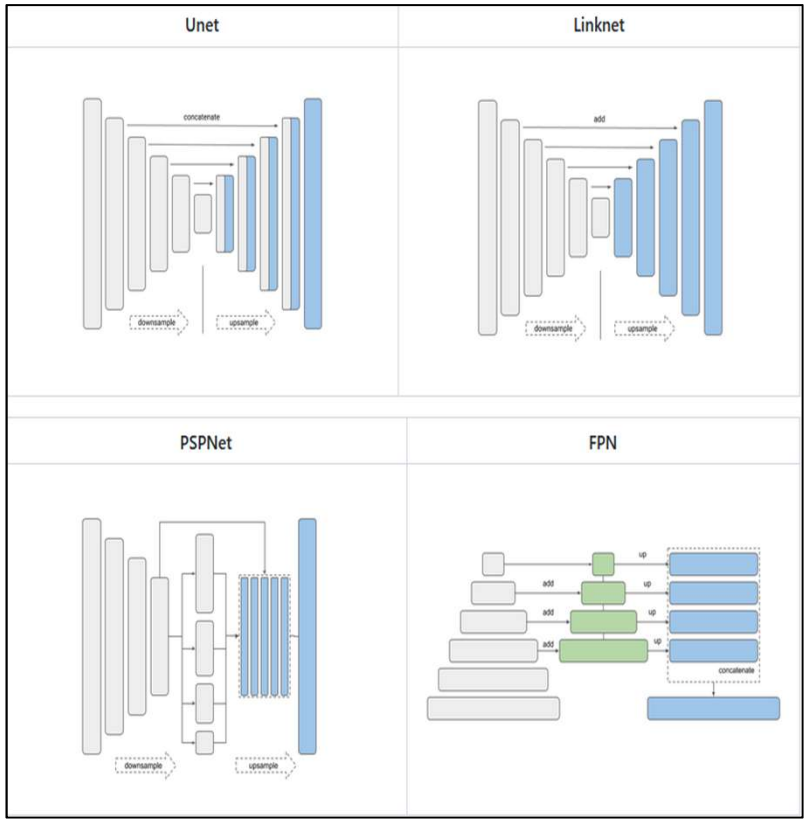
The main features of this library are:

- High level API (just two lines to create NN)
- **4** models architectures for binary and multi class segmentation (including legendary **Unet**)
- **25** available backbones for each architecture
- All backbones have **pre-trained** weights for faster and better convergence

Backbones

Type	Names
VGG	`'vgg16' 'vgg19'`
ResNet	`'resnet18' 'resnet34' 'resnet50' 'resnet101' 'resnet152'`
SE-ResNet	`'seresnet18' 'seresnet34' 'seresnet50' 'seresnet101' 'seresnet152'`
ResNeXt	`'resnext50' 'resnext101'`
SE-ResNeXt	`'seresnext50' 'seresnext101'`
SENet154	`'senet154'`
DenseNet	`'densenet121' 'densenet169' 'densenet201'`
Inception	`'inceptionv3' 'inceptionresnetv2'`
MobileNet	`'mobilenet' 'mobilenetv2'`
EfficientNet	`'efficientnetb0' 'efficientnetb1' 'efficientnetb2' 'efficientnetb3' 'efficientnetb4' 'efficientnetb5' efficientnetb6' efficientnetb7'`

> All backbones have weights trained on 2012 ILSVRC ImageNet dataset (`encoder_weights='imagenet'`).



2 Présentation des modèles Deep Learning & Code utilisé & résultats

https://github.com/qubvel/segmentation_models

```
# Dataset choosen for this project according to hardware capabilities

test_cities = ['monchengladbach', 'strasbourg', 'stuttgart', 'tubingen', 'ulm', 'weimar', 'zurich']
train_cities = ['aachen', 'bochum', 'bremen', 'cologne', 'darmstadt', 'dusseldorf', 'erfurt', 'hamburg', 'hanover', 'jena', 'krefeld']
val_cities = ['frankfurt', 'lindau', 'munster']
```

```
img_size = (128, 256)
n_classes = 8
batch_size = 5
imgaug_multiplier = 3
```

```
train_img_paths = []
train_ann_paths = []

for cities in train_cities:

    train_img_dir = "D:/telechargements/My_Work/P8/P8_Cityscapes_leftImg8bit_trainvaltest/leftImg8bit/train/" + cities        # leftImg
    train_ann_dir = "D:/telechargements/My_Work/P8/P8_Cityscapes_gtFine_trainvaltest/gtFine/train/" + cities    # gtFine

    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")
        ]
    )
```

Output

```
Number of train images: 1817
Number of train annotations: 1817
Number of val images: 500
Number of val annotations: 500
Number of test images: 1158
Number of test annotations: 1158
```

2 Présentation des modèles Deep Learning & Code utilisé & résultats

https://github.com/mcordts/cityscapesScripts/blob/master/cityscapesscripts/helpers/labels.py

Catégories initiales

```
labels = [
    #       name                     id    trainId   category            catId     hasInstances   ignoreInEval   color
    Label(  'unlabeled'            ,  0 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'ego vehicle'          ,  1 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'rectification border' ,  2 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'out of roi'           ,  3 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'static'               ,  4 ,      255 , 'void'            , 0       , False        , True         , (  0,  0,  0) ),
    Label(  'dynamic'              ,  5 ,      255 , 'void'            , 0       , False        , True         , (111, 74,  0) ),
    Label(  'ground'               ,  6 ,      255 , 'void'            , 0       , False        , True         , ( 81,  0, 81) ),
    Label(  'road'                 ,  7 ,        0 , 'flat'            , 1       , False        , False        , (128, 64,128) ),
    Label(  'sidewalk'             ,  8 ,        1 , 'flat'            , 1       , False        , False        , (244, 35,232) ),
    Label(  'parking'              ,  9 ,      255 , 'flat'            , 1       , False        , True         , (250,170,160) ),
    Label(  'rail track'           , 10 ,      255 , 'flat'            , 1       , False        , True         , (230,150,140) ),
    Label(  'building'             , 11 ,        2 , 'construction'    , 2       , False        , False        , ( 70, 70, 70) ),
    Label(  'wall'                 , 12 ,        3 , 'construction'    , 2       , False        , False        , (102,102,156) ),
    Label(  'fence'                , 13 ,        4 , 'construction'    , 2       , False        , False        , (190,153,153) ),
    Label(  'guard rail'           , 14 ,      255 , 'construction'    , 2       , False        , True         , (180,165,180) ),
    Label(  'bridge'               , 15 ,      255 , 'construction'    , 2       , False        , True         , (150,100,100) ),
    Label(  'tunnel'               , 16 ,      255 , 'construction'    , 2       , False        , True         , (150,120, 90) ),
    Label(  'pole'                 , 17 ,        5 , 'object'          , 3       , False        , False        , (153,153,153) ),
    Label(  'polegroup'            , 18 ,      255 , 'object'          , 3       , False        , True         , (153,153,153) ),
    Label(  'traffic light'        , 19 ,        6 , 'object'          , 3       , False        , False        , (250,170, 30) ),
    Label(  'traffic sign'         , 20 ,        7 , 'object'          , 3       , False        , False        , (220,220,  0) ),
    Label(  'vegetation'           , 21 ,        8 , 'nature'          , 4       , False        , False        , (107,142, 35) ),
    Label(  'terrain'              , 22 ,        9 , 'nature'          , 4       , False        , False        , (152,251,152) ),
    Label(  'sky'                  , 23 ,       10 , 'sky'             , 5       , False        , False        , ( 70,130,180) ),
    Label(  'person'               , 24 ,       11 , 'human'           , 6       , True         , False        , (220, 20, 60) ),
    Label(  'rider'                , 25 ,       12 , 'human'           , 6       , True         , False        , (255,  0,  0) ),
    Label(  'car'                  , 26 ,       13 , 'vehicle'         , 7       , True         , False        , (  0,  0,142) ),
    Label(  'truck'                , 27 ,       14 , 'vehicle'         , 7       , True         , False        , (  0,  0, 70) ),
    Label(  'bus'                  , 28 ,       15 , 'vehicle'         , 7       , True         , False        , (  0, 60,100) ),
    Label(  'caravan'              , 29 ,      255 , 'vehicle'         , 7       , True         , True         , (  0,  0, 90) ),
    Label(  'trailer'              , 30 ,      255 , 'vehicle'         , 7       , True         , True         , (  0,  0,110) ),
    Label(  'train'                , 31 ,       16 , 'vehicle'         , 7       , True         , False        , (  0, 80,100) ),
    Label(  'motorcycle'           , 32 ,       17 , 'vehicle'         , 7       , True         , False        , (  0,  0,230) ),
    Label(  'bicycle'              , 33 ,       18 , 'vehicle'         , 7       , True         , False        , (119, 11, 32) ),
    Label(  'license plate'        , -1 ,       -1 , 'vehicle'         , 7       , False        , True         , (  0,  0,142) ),
]
```

Réduction des catégories a 8

```
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]}
```

2 Présentation des modèles Deep Learning & Code utilisé & résultats

Input des données en batch à travers la classe ‘Image’

```python
class Image(tf.keras.utils.Sequence):
    "Helper to iterate over the data (as Numpy arrays)."
    "Put AUG_YES=0 to activate images augmentation"

    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

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

    def __getitem__(self, idx):
        "Returns tuple (input, target) correspond to 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="float32")
        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)
            # Image AUGMENTATION

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

                angle, crop = generateRandomParams(i * mul)

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

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

                image_aug = seq(image=y[i])
                y[batch_size * mul + i] = image_aug
        return x, y
```

Partie augmentation des images

2 Présentation des modèles Deep Learning & Code utilisé & résultats

Input des données en batch à travers la classe 'Image_no_aug'

Partie augmentation supprimée

```python
class Image_no_aug(tf.keras.utils.Sequence):
    "Helper to iterate over the data (as Numpy arrays)."
    "Put AUG_YES=0 to activate images augmentation"

    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

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

    def __getitem__(self, idx):
        "Returns tuple (input, target) correspond to 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="float32")
        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")  # initial : 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)

        return x, y
```

2 Présentation des modèles Deep Learning & Code utilisé & résultats

Loss functions utilisées :

```
loss='sparse_categorical_crossentropy'

def dice_coef_cat(y_true, y_pred, smooth=1e-7):

    y_true_f = K.flatten(K.one_hot(K.cast(y_true, 'int32'), num_classes=8)[...,0:])
    y_pred_f = K.flatten(y_pred[...,0:])
    intersect = K.sum(y_true_f * y_pred_f, axis=-1)
    denom = K.sum(y_true_f + y_pred_f, axis=-1)
    return K.mean((2. * intersect / (denom + smooth)))

def dice_coef_cat_loss(y_true, y_pred):

    return 1 - dice_coef_cat(y_true, y_pred)

def jaccard_distance(y_true, y_pred, smooth=1e-7):

    y_true_f = K.flatten(K.one_hot(K.cast(y_true, 'int32'), num_classes=8)[...,0:])
    y_pred_f = K.flatten(y_pred[...,0:])

    intersection = K.sum((y_true_f * y_pred_f), axis=-1)
    sum_ = K.sum(y_true_f + y_pred_f, axis=-1)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return (1 - jac) * smooth
```

metrics utilisés :

```
metrics = [sm.metrics.iou_score, 'sparse_categorical_accuracy']
```

2 Présentation des modèles Deep Learning & Code utilisé & résultats

Models simulated :

```
Data_gen =[Image_no_aug,Image]
Models_list=[sm.Unet, sm.FPN]
opt_Adam = tf.keras.optimizers.Adam(learning_rate=0.1)
# opt_SGD = tf.keras.optimizers.SGD(learning_rate=0.1) ## Error : unsupported operand type(s) for -: 'range' and 'int'
optimizer_list = [opt_Adam]
BACKBONE_LIST = ['resnet18','resnet34','vgg16']
loss_list = ['sparse_categorical_crossentropy', dice_coef_cat_loss , jaccard_distance]
```

Aperçu des combinaisons :

36 runs en tout

	run	Data_gen	Model	BACKBONE	Loss
0	1	Image_no_aug	Unet	resnet18	sparse_categorical_crossentropy
1	2	Image_no_aug	Unet	resnet18	<function dice_coef_cat_loss at 0x00000269E081...
2	3	Image_no_aug	Unet	resnet18	<function jaccard_distance at 0x00000269E086F040>
3	4	Image_no_aug	Unet	resnet34	sparse_categorical_crossentropy
4	5	Image_no_aug	Unet	resnet34	<function dice_coef_cat_loss at 0x00000269E081...

Définition des Call-backs :

```
callbacks = [
    keras.callbacks.EarlyStopping(monitor = 'loss', patience=2,verbose=1)
] # keras.callbacks.ModelCheckpoint("checkpoint.h5", save_best_only=True)
```

2 Présentation des modèles Deep Learning & Code utilisé & résultats

Exemple d'un modèle simulé :

1

2

3

Runs : NO AUG

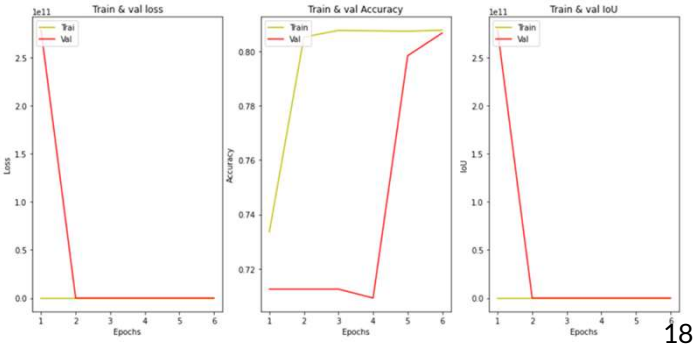
```
    # Generating data sequences for no_aug cases
    test_seq = Image_no_aug(batch_size, img_size, test_img_paths, test_ann_paths)

    # Generate train and val sequences
    train_seq = Image_no_aug(batch_size, img_size, train_img_paths, train_ann_paths)
    val_seq = Image_no_aug(batch_size, img_size, val_img_paths, val_ann_paths)
    print('train set shape   :',train_seq[0][0].shape)
    print('target set shape  :',train_seq[0][1].shape)
    print('----------------------------------------')
    #Quick verif
    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)
```

```
train set shape   : (15, 128, 256, 3)
target set shape  : (15, 128, 256, 1)
```

```
# Run N01 :
x = runs_no_aug.iloc[0]
mod = sm.Unet
backbone = 'resnet18'
los = 'sparse_categorical_crossentropy'
preprocess_input = sm.get_preprocessing(backbone)
```

```
# Define strat time
start_time = datetime.now()
# Order on dataframe for check purposes
print('----------------------------------------------------------------------------------------------------')
print(runs_no_aug.iloc[0])
print('----------------------------------------------------------------------------------------------------')
# Define model
model = mod(backbone, encoder_weights='imagenet', classes=n_classes ,  input_shape=(*img_size, 3),activation='softmax')
print('Run N:  ',x[0],'    ','Data generator  :','NO_AUG')
# Compile model
model.compile(opt_Adam,loss=los, metrics=metrics)
print('Model :',mod.__name__,' ','BACKBONE  :',backbone,' ','Loss_function  :',los)
print('----------------------------------------------------------------------------------------------------')
# Fit Model
history = model.fit(train_seq, epochs=15, batch_size=batch_size, callbacks=callbacks,validation_data=val_seq,verbose=1)
print('\n')
# Evaluate model on test data
print('Results on test sequence   :{}'.format(model.evaluate(test_seq,batch_size=batch_size)))
end_time = datetime.now()
print('Execution Duration: {}'.format(end_time - start_time))
print('Epochs run total :',len(history.history['loss']))
```

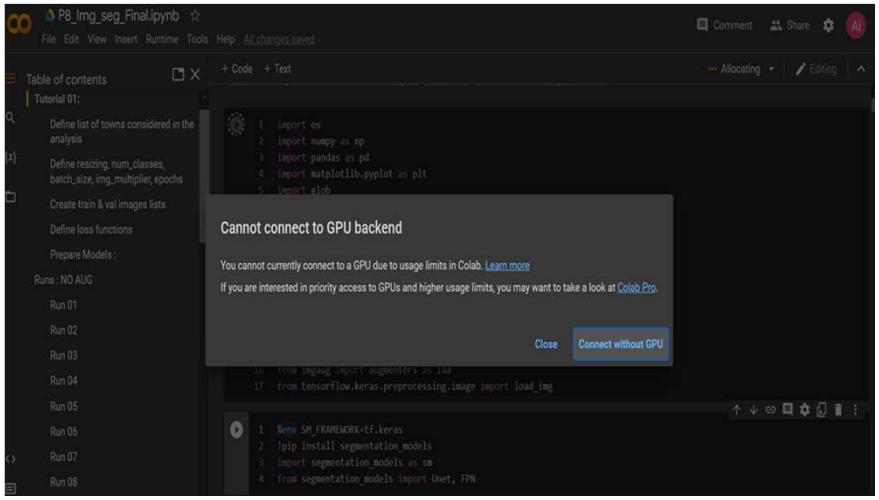
```
----------------------------------------------------------------------------------------------------
run                                                1
Data_gen                                Image_no_aug
Model                                           Unet
BACKBONE                                    resnet18
Loss          sparse_categorical_crossentropy
Name: 0, dtype: object
----------------------------------------------------------------------------------------------------
Run N:   1      Data generator  : NO_AUG
Model : Unet   BACKBONE  : resnet18   Loss_function  : sparse_categorical_crossentropy
----------------------------------------------------------------------------------------------------
Epoch 1/15
39/39 [==============================] - 118s 3s/step - loss: 0.8501 - iou_score: 0.5046 - sparse_categorical_accuracy: 0.7335 - val_loss: 278252912640.0000 - val_iou_score: 0.3594 - val_sparse_categorical_accuracy: 0.7125
Epoch 2/15
39/39 [==============================] - 112s 3s/step - loss: 0.5304 - iou_score: 0.5732 - sparse_categorical_accuracy: 0.8052 - val_loss: 3277737.5000 - val_iou_score: 0.3580 - val_sparse_categorical_accuracy: 0.7125
Epoch 3/15
39/39 [==============================] - 123s 3s/step - loss: 0.5200 - iou_score: 0.5799 - sparse_categorical_accuracy: 0.8076 - val_loss: 23429.3516 - val_iou_score: 0.3580 - val_sparse_categorical_accuracy: 0.7125
Epoch 4/15
39/39 [==============================] - 110s 3s/step - loss: 0.5185 - iou_score: 0.5779 - sparse_categorical_accuracy: 0.8075 - val_loss: 553.5585 - val_iou_score: 0.3419 - val_sparse_categorical_accuracy: 0.7093
Epoch 5/15
39/39 [==============================] - 117s 3s/step - loss: 0.5187 - iou_score: 0.5856 - sparse_categorical_accuracy: 0.8073 - val_loss: 2.2129 - val_iou_score: 0.3680 - val_sparse_categorical_accuracy: 0.7983
Epoch 6/15
39/39 [==============================] - 122s 3s/step - loss: 0.5188 - iou_score: 0.5864 - sparse_categorical_accuracy: 0.8077 - val_loss: 2.5661 - val_iou_score: 0.5086 - val_sparse_categorical_accuracy: 0.8067
Epoch 6: early stopping


7/7 [==============================] - 4s 536ms/step - loss: 2.5207 - iou_score: 0.4964 - sparse_categorical_accuracy: 0.8091
Results on test sequence   :[2.5206847190856934, 0.4963855147361755, 0.8091078996658325]
Execution Duration: 0:11:48.707880
Epochs run total : 6
```



2 Présentation des modèles Deep Learning & Code utilisé & résultats

Soucis GPU :

1



2

Resource Limits

Why aren't resources guaranteed in Colab?

In order to be able to offer computational resources free of charge, Colab needs to maintain the flexibility to adjust usage limits and hardware availability on the fly. Resources available in Colab vary over time to accommodate fluctuations in demand, as well as to accommodate overall growth and other factors.

Some users want to be able to do more in Colab than the resource limits allow. Colab Pro and Pro+ provide priority access to faster GPUs, longer running notebooks and more memory. Our long term goal is to continue providing a free of charge version of Colab, while also growing in a sustainable fashion to meet the needs of our users.

If you would like to have complete control over your resources in Colab, check out Colab GCP Marketplace VMs. Colab GCP Marketplace VMs allow you to specify the exact runtime resources to use and provide you with a persistent environment you can manage to your liking while still using the Colab UI.

What are the usage limits of Colab?

Colab is able to provide resources free of charge in part by having dynamic usage limits that sometimes fluctuate, and by not providing guaranteed or unlimited resources. This means that overall usage limits as well as idle timeout periods, maximum VM lifetime, GPU types available, and other factors vary over time. Colab does not publish these limits, in part because they can (and sometimes do) vary quickly.

GPUs and TPUs are sometimes prioritized for users who use Colab interactively rather than for long-running computations, or for users who have recently used less resources in Colab. As a result, users who use Colab for long-running computations, or users who have recently used more resources in Colab, are more likely to run into usage limits and have their access to GPUs and TPUs temporarily restricted. Users interested in having higher and more stable usage limits may be interested in Colab Pro and Pro+. Users with high computational needs may be interested in using Colab's UI with either a local runtime running on their own hardware or Colab GCP Marketplace VMs.

What types of GPUs are available in Colab?

The types of GPUs that are available in Colab vary over time. This is necessary for Colab to be able to provide access to these resources free of charge. Users who are interested in more reliable access to Colab's fastest GPUs may be interested in Colab Pro and Pro+. If you would like to use specific hardware in Colab, check out Colab GCP Marketplace VMs.

Note that using Colab for cryptocurrency mining is disallowed entirely, and may result in your account being restricted for use with Colab altogether.

2 Présentation des modèles Deep Learning & Code utilisé & résultats

1

Pas d'effets notables de l'augmentation

2

Chaque loss function a un effet différent sur les metrics

3

La Jaccard loss prend le plus de temps d'exécution

Récap des résultats de l'ensemble des runs

NO AUG						VAL (LAST EPOCH)			TEST		
MODEL	BACKBONE	LOSS	RUN N°	EPOCHS	TIME TOTAL	LOSS	IOU	ACC SPARSE	LOSS	IOU	ACC SPARSE
Unet	resnet18	Cross-Entropy	1	11	0:04:59	0.58	0.14	0.79	0.55	0.14	0.8
		Dice	2	3	0:01:57	0.28	0.08	0.71	0.29	0.08	0.7
		Jaccard	3	15	0:07:04	1.80E-08	0.16	0.9	1.58E-08	0.15	0.91
	resnet34	Cross-Entropy	4	12	0:05:44	0.54	0.13	0.79	0.54	0.13	0.8
		Dice	5	3	0:01:33	0.86	0.08	0.13	0.86	0.08	0.13
		Jaccard	6	15	0:07:08	2.13E-08	0.18	0.88	1.90E-08	0.18	0.89
	vgg16	Cross-Entropy	7	7	0:03:22	0.56	0.14	0.79	0.55	0.13	0.8
		Dice	8	10	0:04:57	0.19	2.00E-01	0.8	0.18	1.90E-01	0.8
		Jaccard	9	9	0:04:17	2.56E-08	0.22	0.85	2.18E-08	0.2	0.87
FPN	resnet18	Cross-Entropy	10	15	0:07:29	0.6	0.14	0.85	0.61	0.14	0.84
		Dice	11	3	0:01:34	0.28	0.08	0.71	0.29	0.08	0.7
		Jaccard	12	11	0:05:16	1.82E-08	0.16	0.9	1.57E-08	0.15	0.91
	resnet34	Cross-Entropy	13	7	0:03:29	2.32	0.11	0.19	2.2	0.11	0.2
		Dice	14	3	0:01:37	0.86	0.08	0.13	0.86	0.08	0.13
		Jaccard	15	15	0:07:12	1.79E-08	0.15	0.9	1.60E-08	0.15	0.91
	vgg16	Cross-Entropy	16	2	0:01:06	nan	nan	0.71	nan	nan	0.7
		Dice	17	2	0:01:18	nan	nan	0.71	nan	nan	0.7
		Jaccard	18	15	0:07:30	1.27E-08	0.15	0.93	1.14E-08	0.15	0.94
WITH AUG											
MODEL	BACKBONE	LOSS									
Unet	resnet18	Cross-Entropy	19	8	0:06:08	270	0.11	0.16	246	0.1	0.18
		Dice	20	3	0:01:34	0.87	0.09	0.12	0.87	0.08	0.12
		Jaccard	21	11	0:05:34	8.75E-08	0.12	0.21	8.60E-08	0.12	0.22
	resnet34	Cross-Entropy	22	15	0:07:23	2.4	0.11	0.13	2.42	0.11	0.13
		Dice	23	3	0:01:40	0.86	0.08	0.13	0.86	0.08	0.13
		Jaccard	24	5	0:02:38	8.97E-08	0.15	0.18	8.87E-08	0.15	0.2
	vgg16	Cross-Entropy	25	4	0:02:10	1.57	0.11	0.13	1.56	0.11	0.13
		Dice	26	3	0:01:51	0.94	0.08	0.05	0.9	0.08	0.07
		Jaccard	27	3	0:01:36	9.60E-08	0.12	0.07	9.56E-08	0.12	0.08
FPN	resnet18	Cross-Entropy	28	15	0:07:49	96	0.11	0.13	96	0.11	0.13
		Dice	29	3	0:01:40	0.86	0.08	0.13	0.86	0.08	0.13
		Jaccard	30	11	0:05:45	8.57E-08	0.12	0.22	8.68E-08	0.12	0.23
	resnet34	Cross-Entropy	31	15	0:07:51	3.9	0.11	0.13	2.94	0.11	0.13
		Dice	32	3	0:01:42	0.94	0.08	0.05	0.92	0.08	0.07
		Jaccard	33	9	0:05:11	8.81E-08	0.11	0.21	8.73E-08	0.12	0.22
	vgg16	Cross-Entropy	34	2	0:01:08	nan	nan	0.71	nan	nan	0.7
		Dice	35	2	0:01:09	nan	nan	0.71	nan	nan	0.7
		Jaccard	36	3	0:01:42	9.11E-08	0.13	0.16	8.98E-08	0.13	0.19

2 Présentation des modèles Deep Learning & Code utilisé & résultats

Récap des résultats de l'ensemble des runs precedents

NO AUG							TRAIN (LAST EPOCH)			VAL (LAST EPOCH)			TEST		
MODEL	BACKBONE	LOSS	RUN N°	EPOCHS	TIME TOTAL	TIME/EPOCHS	LOSS	IOU	ACC SPARSE	LOSS	IOU	ACC SPARSE	LOSS	IOU	ACC SPARSE
Unet	resnet18	Cross-Entropy	1	6	0:11:48	117	0.51	0.58	0.8	2.56	0.5	0.8	2.52	0.49	0.81
		Dice	2	4	0:07:03	98	0.3	0.09	0.96	0.28	0.08	0.71	0.29	0.08	0.7
		Jaccard	3	15	0:24:38	97	1.54E-08	0.06	0.89	1.96E-08	0.05	0.88	1.95E-08	0.05	0.88
	resnet34	Cross-Entropy	4	5	0:10:31	121	1.05	0.29	0.69	41792	0.26	0.71	36330	0.25	0.71
		Dice	5	4	0:08:37	123	0.3	0.09	0.69	0.28	0.08	0.71	0.29	0.08	0.7
		Jaccard	6	15	0:32:43	151	1.00E-08	0.1	0.93	1.58E-08	0.08	0.9	1.40E-08	0.08	0.91
	vgg16	Cross-Entropy	7	15	0:52:48	198	0.51	0.57	0.8	0.55	0.54	0.8	0.53	0.52	0.8
		Dice	8	15	0:54:01	200	0.12	0.02	0.8	0.87	7.34E-04	0.71	0.87	6.70E-04	0.7
		Jaccard	9	15	0:52:09	224	1.80E-08	0.06	0.87	2.10E-08	0.04	0.85	1.94E-08	0.05	0.87
FPN	resnet18	Cross-Entropy	10	13	0:51:02	226	0.36	0.14	0.87	0.39	0.14	0.85	0.35	0.14	0.87
		Dice	11	8	0:31:16	227	0.19	0.28	0.81	0.19	0.27	0.8	0.18	0.25	0.81
		Jaccard	12	15	0:59:26	227	1.19E-08	0.16	0.93	1.73E-08	0.16	0.9	1.57E-08	0.15	0.91
	resnet34	Cross-Entropy	13	5	0:22:11	255	0.54	0.14	0.79	110193	0.1	0.05	106892	0.1	0.05
		Dice	14	8	0:37:48	273	0.18	0.12	0.81	0.39	0.11	0.61	0.39	0.1	0.61
		Jaccard	15	15	1:08:20	282	1.88E-08	0.19	0.89	2.24E-08	0.2	0.87	1.98E-08	0.2	0.89
	vgg16	Cross-Entropy	16	2	0:11:16	318	nan	nan	0.69	nan	nan	0.71	nan	nan	0.7
		Dice	17	2	0:11:34	340	nan	nan	0.69	nan	nan	0.71	nan	nan	0.7
		Jaccard	18	15	1:26:21	348	8.87E-09	0.15	0.95	1.28e-08	0.15	0.93	1.15E-08	0.15	0.94
WITH AUG															
MODEL	BACKBONE	LOSS													
Unet	resnet18	Cross-Entropy	19	6	0:10:16	100	17.81	0.65	0.39	55.6	0.74	0.21	17.81	0.4	0.06
		Dice	20	6	0:10:23	101	0.6	0.84	0.4	0.63	0.81	0.42	0.87	0.25	0.13
		Jaccard	21	15	0:25:39	101	3.10E-08	0.13	0.79	4.67E-08	0.12	0.67	8.25E-08	0.1	0.3
	resnet34	Cross-Entropy	22	5	0:11:01	129	1.63	0.54	0.38	1.69	0.53	0.42	2.4	0.29	0.13
		Dice	23	7	0:16:01	129	0.62	0.56	0.4	0.61	0.54	0.42	0.88	0.16	0.13
		Jaccard	24	15	0:33:41	162	2.99E-08	0.13	0.79	4.35E-08	0.12	0.69	6.14E-08	0.1	0.57
	vgg16	Cross-Entropy	25	4	0:14:48	216	1.63	0.56	0.4	1.68	0.54	0.42	2.98	0.31	0.13
		Dice	26	6	0:21:37	210	0.6	0.84	0.4	0.63	0.81	0.42	0.87	0.25	0.13
		Jaccard	27	15	0:56:30	212	4.40E-08	0.12	0.69	5.82E-08	0.12	0.57	8.84E-08	0.1	0.21
FPN	resnet18	Cross-Entropy	28	7	0:31:22	233	1.63	0.14	0.4	3.26	0.14	0.4	829.5	0.13	0.12
		Dice	29	15	1:03:25	230	0.37	0.2	0.59	0.46	0.22	0.5	0.8	0.09	0.19
		Jaccard	30	15	1:02:28	234	3.30E-08	0.15	0.77	4.92E-08	0.15	0.65	8.67E-08	0.11	0.23
	resnet34	Cross-Entropy	31	5	0:28:29	307	1.63	0.14	0.4	320.86	0.19	0.42	25770.1	0.08	0.13
		Dice	32	8	0:39:55	260	0.37	0.21	0.59	0.43	0.2	0.54	0.8	0.1	0.19
		Jaccard	33	15	1:07:35	254	4.05E-08	0.15	0.72	5.30E-08	0.15	0.62	8.75E-08	0.13	0.22
	vgg16	Cross-Entropy	34	2	0:10:51	315	nan	nan	0.06	nan	nan	0.09	nan	nan	0.7
		Dice	35	2	0:10:46	315	nan	nan	0.06	nan	nan	0.09	nan	nan	0
		Jaccard	36	15	1:21:31	320	2.92E-08	0.15	0.25	4.14E-08	0.14	0.71	8.47E-08	0.11	0.25

3 Déploiement de l'application :

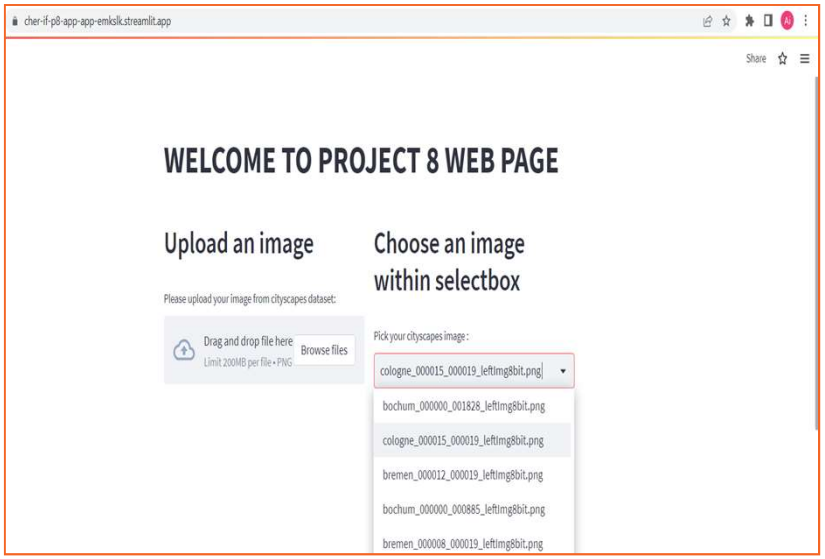
Le déploiement de l'API s'est fait selon les étapes suivantes :

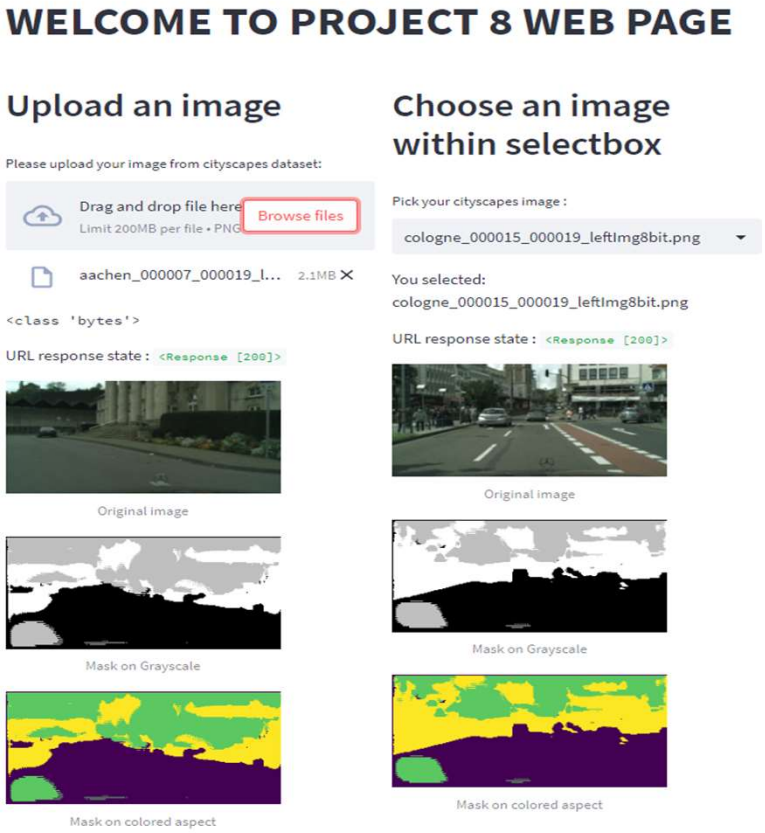
I- creation et deploiment sur Microsoft Azure d'une API flask qui recoit une image comme donnee d'entrée pour la convertir en image avec masks de segmentation.

II- creation et deploiment sur streamlit cloud d'une application streamlit qui consomme l'API citee ci-dessus et sert d'interface utilisateur pour l'entrée et la visualisation des données.

Adresse web : https://cher-if-p8-app-app-emkslk.streamlit.app/

3 Déploiement de l'application :





Conclusion :

Reprendre les mêmes runs avec :

- Une taille plus grande des images en données d'entrée
- prévoir un plus large hyperparameters tuning des modèles
- Extension aux autres modèles et backbones avec un GPU adapté pour être en mesure de mieux apprécier et comparer les résultats

La segmentation sémantique présente une solide option pour la conduite autonome des véhicules