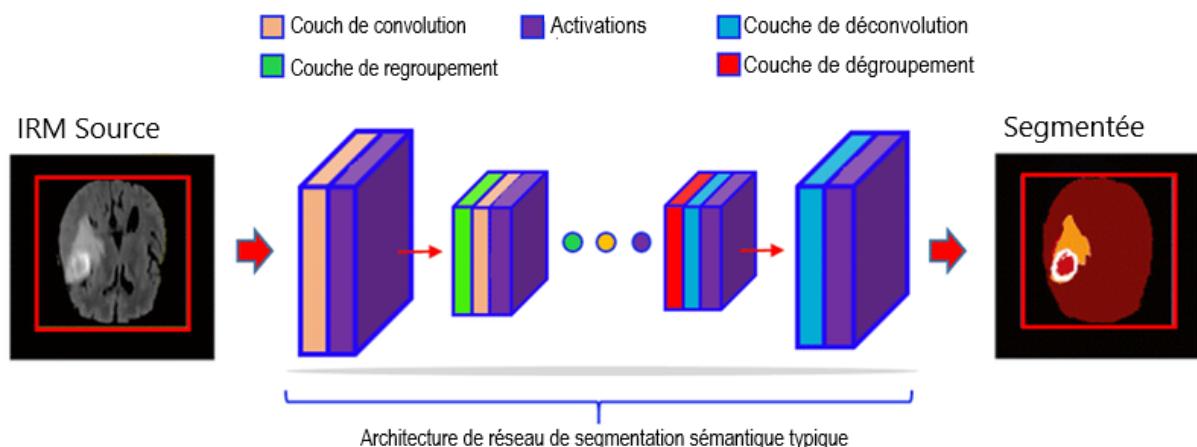


# Cloud Computing: Rapport

## Segmentation sur le cloud

Après avoir vu plusieurs notions de qu'est ce que le cloud computing, quel est l'apport dans un projet et comment l'utiliser, nous allons faire ce projet final. Le but ici était de créer une application directement hébergé sur le web qui fait appel à un modèle d'apprentissage automatique dans le but de faire de la segmentation:



Le but de la segmentation est de prendre en entrée une image classique, dans un premier temps, appliquer des opérations de convolutions ( Conv2d-> Activation -> Maxpool) dans le but d'extraire les informations de notre image et de reconstruire l'image de la même dimension, mais chaque pixel sera classifié. On va donc pouvoir mettre en évidence l'appartenance à une classe des éléments sur une photo.

Les différentes utilisateurs vont pouvoir accéder à notre modèle pré-entraîné et le fine-tuner (mettre à jour certains poids en ré-entraînant notre modèle sur des images différentes). Pour ce faire on va se baser sur certaines métriques (Précision, Intersection over Union et le F1 score). En plus de cela les différents utilisateurs vont avoir la possibilité de l'utiliser en inférence à travers notre interface web.

Cependant que ce soit par manque de droit ou de temps je n'ai pas pu aller jusqu'au bout du sujet. J'ai donc créé une API avec Flask qui va permettre de faire la segmentation à travers des requêtes HTTP. Pour ce qui est de la partie multi-utilisateur, je voulais conteneuriser mon modèle pour pouvoir l'instancier autant de fois que voulu et que chaque client ai son propre modèle et qu'il puisse l'entraîner sur ses propres données ou l'utiliser en inférence mais comme nous n'avons pas les droits j'ai fait le fine tuning en local et j'ai juste hébergé mon modèle sur le cloud.

## **Sommaire**

1. Choix du modèle	3
2. Entrainement et choix de la métrique	4
3. Choix de l'infrastructure	5
4. Création de l'application	7
5. Conclusion	9

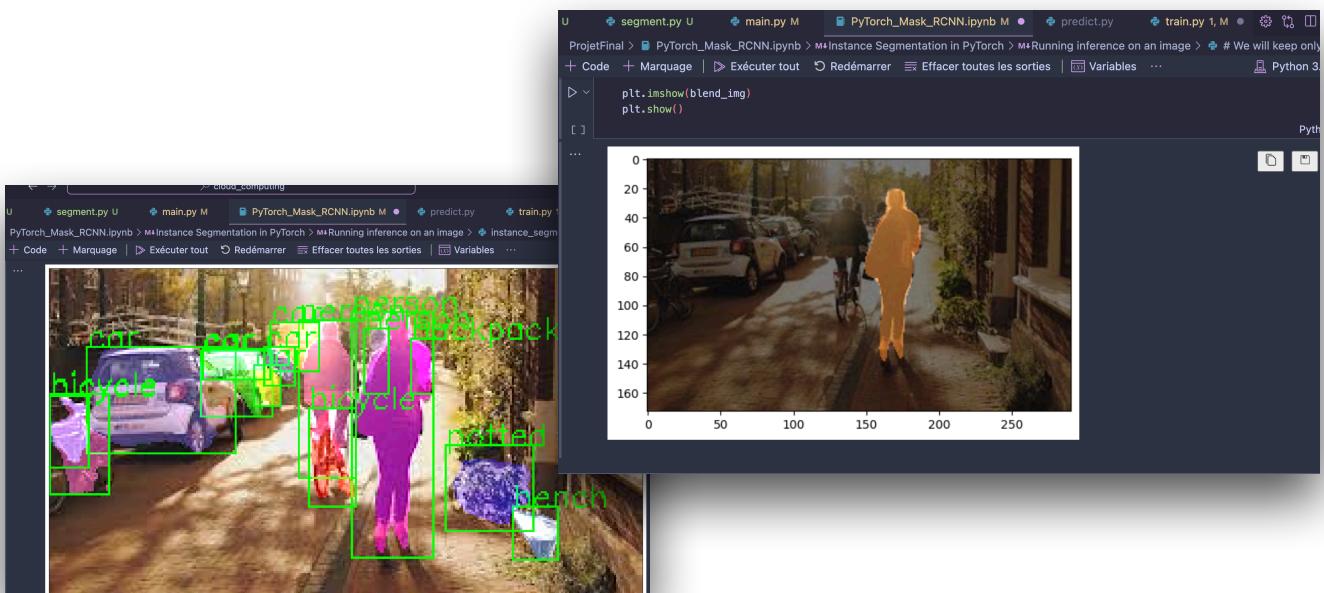
# 1. Choix du modèle

Pour ce qui est du choix du modèle, le choix était vaste. Après plusieurs recherche et comparaison (Technologies utilisées, facilité d'implémentation et de compréhension) je suis partie sur le modèle mask RCNN qui est manipulable très facilement en utilisant PyTorch qui est un framework Python qui permet de faire du machine-learning. PyTorch étant une technologie que l'on utilise dans d'autre module de notre formation. Le modèle mask RCNN permet de faire de la classification, de la segmentation et de la reconnaissance d'objet. Ce modèle étant disponible sur le hub de torch est très facile à utiliser :

```
● ● ●

1 model = torchvision.models.detection.maskrcnn_resnet50_fpn(pretrained=True)
2 torch.save(model.state_dict(), os.getcwd() + '/model/mask_rcnn_model.pth')
3 model.eval()
4
```

Nous pouvons passer un argument pretrained dans le modèle ce qui va nous faciliter le travail pour plus tard, car il ne nous restera plus qu'à simuler le fine tuning. Ce modèle venait avec pas mal de code pour apprendre à l'utiliser et donc j'ai fait un Jupyter Notebook pour pouvoir aller en profondeur dedans et apprendre comment fonctionnait la segmentation.



## 2. Entrainement et choix de la métrique

Vient ensuite la partie de l'entraînement (Fine tuning). Comme je l'ai dit j'ai dû passer par un entraînement en local que je ne fais qu'une fois sur le dataset que j'ai trouvé (Oxford-III Pet). Ce dataset est plutôt grand donc pour simuler le fine-tuning, je n'ai fait l'entraînement que sur le set de validation qui est beaucoup plus petit. Ce dataset contient des photos d'animaux. Le modèle de base est capable de classifier les éléments qui sont présents dans le set Oxford-III donc on n'a pas à faire de transfert learning. Pour les différentes métriques utilisées nous avons le choix entre la précision, le F1 score et le Intersection over Union. Ces 3 métriques sont intéressantes donc j'ai fait l'entraînement en calculant la loss sur ces facteurs de comparaison, mais pour garder les paramètres finaux, j'ai gardé l'IoU qui est la métrique la plus punitive et donc la plus efficace.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{IoU} = \frac{(Object \cap Detected\ box)}{(Object \cup Detected\ box)}$$

Voici la définition de celle-ci qui va juste prendre la surface ou la prédiction de mon input ainsi que la surface de mon résultat attendu se superposant divisé par la somme des surfaces des deux moins l'intersection des deux.



```
1 def bbox_iou(box1, box2):
2     x1 = max(box1[0], box2[0])
3     y1 = max(box1[1], box2[1])
4     x2 = min(box1[2], box2[2])
5     y2 = min(box1[3], box2[3])
6
7     intersection = max(0, x2 - x1) * max(0, y2 - y1)
8     box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
9     box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])
10    union = box1_area + box2_area - intersection
11    iou = intersection / union
12    return iou
```

J'ai ensuite envoyés ces métriques manuellement dans un bucket s3 en séparant les résultats de chaque classe.

Pour ce faire, j'ai simplement extrait les features classe par classe que j'ai exporté dans un fichier txt. Dans un premier temps, je voulais envoyer ces métriques directement dans le bucket, mais à cause des contraintes de la licence étudiante je n'ai pas pu le faire.

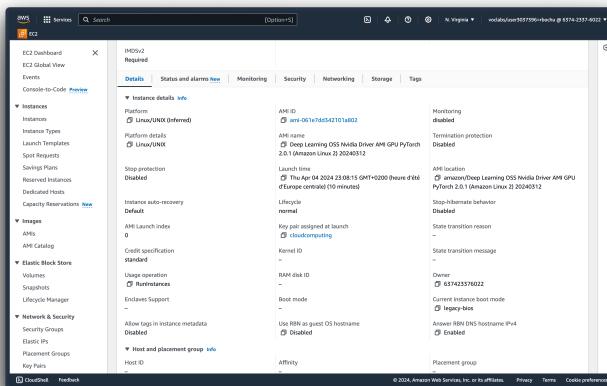
The screenshot shows the 'Objects (1)' section of an Amazon S3 bucket. At the top, there are several buttons: 'Copy S3 URI', 'Copy URL', 'Download', 'Open', 'Delete', 'Actions', and 'Create folder'. Below these is a large orange 'Upload' button. A message states: 'Objects are the fundamental entities stored in Amazon S3. You can use [Amazon S3 inventory](#) to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. [Learn more](#)'.

Below the message is a search bar with the placeholder 'Find objects by prefix'. To the right of the search bar are navigation icons: back, forward, and refresh. A table lists the single object:

	Name	Type	Last modified	Size	Storage class
	metrics.txt	txt	April 2, 2024, 09:31:37 (UTC+02:00)	374.0 B	Standard

### 3. Choix de l'infrastructure

Le choix de l'infrastructure a été la partie la plus complexe au vu de mes compétences en cloud. Je ne savais guère par où commencer, alors j'ai pris une instance EC2 Amazon Linux. J'avais déjà créé un VPC ainsi qu'un security sub-group lors d'un précédent TD alors je n'ai pas eu à le refaire. Pour ce qui est de l'instance EC2 j'en avais déjà plusieurs avec des configurations différentes. Au commencement je pensais que nous n'avions pas accès aux configurations disposant de plus de stockage, de RAM ainsi que de puissance GPU. Mais après de nombreuses limitations lors de l'installation de mon application ainsi que de mon modèle, je suis partie sur une : Deep Learning OSS Nvidia Driver AMI GPU PyTorch 2.0.1 (Amazon Linux 2) 20240312. Le type d'instance est: t2.large avec 45GB de stockage.



Cette configuration possède une installation de Python et PyTorch presque complète et de la puissance nécessaire pour ne pas avoir à attendre trop longtemps pour segmenter l'image.

Pour communiquer avec cette instance j'ai configuré mes security group pour ouvrir les ports 80 pour autoriser les connexions en HTTP et 22 pour le SSH.

Inbound rules (2)							C	Manage tags	Edit inbound rules	
on	Type	Protocol	Port range	Source	Description		<	1	>	⚙️
	SSH	TCP	22	0.0.0.0/0	-					
	HTTP	TCP	80	0.0.0.0/0	-					

Pour communiquer avec cette instance je me suis connecté en SSH avec la clé privée stocké en .pem localement.

⌚ ssh -i "cloudcomputing.pem" root@ec2-34-235-184-174.compute-1.amazonaws.com

Pour faciliter la connexion SSH, j'ai décidé d'utiliser une Elastic IPs. Le but de celle-ci est d'avoir une IP publique statique. Car à chaque redémarrage je devais recharger mes fichiers de configuration pour pouvoir me reconnecter.

J'ai ensuite rencontré bon nombre de problème par rapport à mon installation python. J'ai donc fait un fichier requirements.txt pour faciliter l'installation de toutes les dépendances.

## 4. Crédation de l'application

Mon application est donc un site web utilisant pour le back-end le framework Flask qui permet d'utiliser du python. Cela va me permettre de simplifier la manipulation et transformation de notre image. Pour le front je suis partie sur un thème très minimaliste étant donné mon retard. L'interface est donc très basique:



On va pouvoir upload une image directement grâce à un formulaire, une fois que l'on va valider celui-ci, on va envoyer une requête POST contenant la data de notre image. On va dans notre route appeler la fonction qui permet la segmentation. En retour de cette fonction, on va avoir l'image segmenter que l'on va retrouver dans notre template HTML en la gardant dans un buffer.

Ceci va permettre d'utiliser le modèle en inférence avec une latence très faible vu le choix de l'infrastructure.

Voici comment sont définis les routes:



```
1 @app.route("/")
2 def index(name=None):
3     return render_template("index.html", name=name)
4
5
6 @app.route("/segment", methods=["POST"])
7 def segment():
8     file = request.files["image"]
9     temp_img_path = "temp_image.png"
10    file.save(temp_img_path)
11    masks, boxes, pred_cls, processed_img = RCNN.instance_segmentation_api(temp_img_path, threshold=0.5)
12
13    if processed_img.dtype != np.uint8:
14        processed_img = processed_img.astype('uint8')
15
16    pil_img = Image.fromarray(processed_img)
17    buffered = BytesIO()
18    pil_img.save(buffered, format="PNG")
19    img_str = base64.b64encode(buffered.getvalue()).decode()
20
21    os.remove(temp_img_path)
22
23    return render_template("segment.html", img_str=img_str)
```

Nous avons un index contenant simplement notre template avec formulaire et nous avons la requête POST qui va récupérer l'image, la segmenter, le transférer dans le buffer et enfin la convertir pour que celle-ci soit lisible en HTML. Cette partie est plutôt simple, mais le code en python a du être adapté car cette fois au lieu de plot l'image on doit la récupérer et ensuite la mettre dans une balise img.

Pour obtenir une meilleure latence, j'ai enregistré le modèle au format pth directement sur mon instance EC2 pour ne pas avoir à le télécharger à chaque segmentation.

```
● ● ●  
1 model_dir = os.path.join(os.getcwd(), 'model')  
2 os.makedirs(model_dir, exist_ok=True) # Creates the directory if it doesn't exist  
3 model_path = os.path.join(model_dir, 'mask_rcnn_model.pth')
```

La fonction instance\_segmentation\_api qui permet la segmentation de notre image d'entrée :

```
● ● ●  
1 def instance_segmentation_api(img_path, threshold=0.5, rect_th=3, text_size=1, text_th=3):  
2  
3     masks, boxes, pred_cls = get_prediction(img_path, threshold)  
4     img = cv2.imread(img_path)  
5     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)  
6     for i in range(len(masks)):  
7         rgb_mask = random_colour_masks(masks[i])  
8         img = cv2.addWeighted(img, 1, rgb_mask, 0.5, 0)  
9         pt1 = (int(boxes[i][0][0]), int(boxes[i][0][1])) # Convert to integer if necessary  
10        pt2 = (int(boxes[i][1][0]), int(boxes[i][1][1])) # Convert to integer if necessary  
11        cv2.rectangle(img, pt1, pt2, color=(0, 255, 0), thickness=rect_th)  
12        cv2.putText(img, pred_cls[i], pt1, cv2.FONT_HERSHEY_SIMPLEX, text_size, (0,255,0), thickness=text_th)  
13  
14    return masks, boxes, pred_cls, img
```

## **5. Conclusion**

Ce projet fût très intéressant de par le nombre de secteur de la data qui sont impliqués pour avoir un résultat final convenant. Nous avons de l'apprentissage automatique pour la recherche et le fine-tuning de notre modèle de segmentation, du développement web plus classique pour avoir une API qui communique avec le front ainsi que le modèle et bien sûr toute la partie infrastructure en ce qui concerne le choix des instances et des services que AWS propose. Malgré un départ très lent et peu sûr, j'ai su m'adapter aux conditions pour délivrer un résultat fonctionnel. Le retour sur expérience est très positif malgré une gestion de projet inexistante dû aux autres projets à rendre dans la même semaine. Ce projet est pour moi nécessaire avant le début du projet Big Data car il permet d'enlever toutes incertitudes par rapport aux choix possibles lorsque l'on veut utiliser de la puissance de calcul dans les nuages.

Merci pour votre lecture !  
Robin.