### **Note Technique**

**1 - Introduction:**

Nous travaillons pour Futur Vision Transport qui est une entreprise qui conçoit des systèmes embarqués de vision par ordinateur pour les véhicules autonomes.Au vus de l’évolution de cette technologie et de son attente, être leader dans ce domaine est un objectif nécessaire.   
  
Ma mission est de travailler sur un modèle de segmentation de l’image. Je viens après le travail de Franck qui a préparé les données et avant Laura qui devra mettre en place le système de décisions.

Nous allons mettre en place et comparer deux modèles, U-Net et DeepLabV3, pour segmenter nos images. Nous allons observer le rapport entre l’efficacité et la rapidité.

**2. Présentation des différentes approches et état de l’art**

### **Vision par ordinateur dans les véhicules autonomes**

La vision par ordinateur est un élément central dans la conception des véhicules autonomes. Grâce aux algorithmes d'apprentissage automatique, les systèmes de vision permettent de percevoir et d’interpréter l’environnement en temps réel, ce qui est crucial pour la navigation et la prise de décision autonome. Ces systèmes permettent, par exemple, de détecter les objets, de suivre les piétons, de reconnaître les panneaux de signalisation, et surtout d'analyser et segmenter la route et ses alentours.

### **Segmentation d’images**

La segmentation d’images consiste à diviser une image en plusieurs régions distinctes, où chaque région représente un objet ou une zone d’intérêt. Cette étape est primordiale dans le domaine des véhicules autonomes, car elle permet une interprétation détaillée de la scène, essentielle pour éviter les obstacles et naviguer en toute sécurité. Par exemple, la segmentation sémantique attribue une catégorie à chaque pixel (route, piéton, bâtiment, etc.), facilitant une compréhension claire et immédiate de l’environnement du véhicule.

### 

### **Modèles de segmentation**

#### **U-Net**

U-Net est un modèle très utilisé pour la segmentation d’images, en particulier dans les tâches où la précision des contours est essentielle. Il est structuré en deux parties : un chemin d’encodage qui capture le contexte global, et un chemin de décodage qui permet une localisation précise. Ce modèle est efficace dans diverses applications, y compris la segmentation des scènes routières dans les véhicules autonomes, en raison de sa capacité à généraliser avec un volume de données relativement limité.

#### **U-Net Mini**

U-Net Mini est une version allégée du modèle U-Net, spécialement conçue pour des environnements où les ressources en calcul sont limitées. Bien que moins complexe, il conserve les principales fonctionnalités de son prédécesseur, tout en étant plus adapté aux systèmes embarqués ou aux applications nécessitant une faible latence, comme les véhicules autonomes.

#### **DeeplabV3+ avec ResNet**

DeeplabV3+ est une architecture avancée pour la segmentation sémantique, utilisant des convolutions dilatées pour capturer le contexte à différentes échelles sans perte de résolution spatiale. Lorsqu’il est associé à ResNet, un réseau de neurones profond conçu pour résoudre les problèmes de gradient dans les modèles très profonds, DeeplabV3+ devient particulièrement puissant. ResNet améliore la capacité du modèle à extraire des caractéristiques riches et multi-échelles des images complexes, ce qui en fait un modèle performant pour des environnements variés et dynamiques, comme ceux des véhicules autonomes.

### 

### 

### **Comparaison des approches**

* **U-Net** : C’est un modèle efficace et rapide pour des tâches de segmentation où les données disponibles sont limitées. Toutefois, sa capacité à capturer des détails très fins peut être légèrement inférieure pour des scènes complexes.
* **U-Net Mini** : Très adapté aux systèmes embarqués grâce à sa légèreté, U-Net Mini est performant lorsque les ressources de calcul sont restreintes, mais peut manquer de précision par rapport aux modèles plus lourds comme DeeplabV3.
* **DeeplabV3 avec ResNet** : Ce modèle est extrêmement performant pour la segmentation dans des environnements complexes grâce à sa capacité à capturer des contextes multi-échelles. Cependant, il demande des ressources de calcul importantes, ce qui peut être un inconvénient pour des systèmes en temps réel ou embarqués.

### **Approches possibles**

D'autres modèles existent dans le domaine de la segmentation d'images et peuvent être envisagés selon les besoins spécifiques du projet :

* **Mask R-CNN** : Ce modèle est une extension du Faster R-CNN qui permet, en plus de la détection d'objets, la segmentation d’instance. Cela permet non seulement de reconnaître les objets dans une scène, mais aussi de les segmenter avec précision. Il est très performant pour des tâches de détection et de segmentation simultanées.
* **PSPNet (Pyramid Scene Parsing Network)** : PSPNet utilise des pyramides de contextes pour améliorer la compréhension globale de la scène. Il est particulièrement efficace pour la segmentation de grandes scènes où les objets ont des échelles variées.
* **FCN (Fully Convolutional Network)** : Le FCN est un modèle plus simple qui repose entièrement sur des couches convolutionnelles pour réaliser la segmentation d’images. Il est utilisé dans les tâches où la segmentation rapide et légère est primordiale, mais il peut manquer de précision par rapport aux modèles plus complexes comme DeeplabV3+.

**3. Présentation détaillée du modèle et de l’architecture retenue**

### **Choix des modèles**

Dans le cadre de notre projet, nous avons retenu deux modèles de segmentation d’images : **U-Net\_mini** et **DeeplabV3 avec ResNet**. Le choix de ces modèles repose sur plusieurs critères.

**U-Net** est un modèle particulièrement performant pour la segmentation des images, notamment grâce à sa capacité à capturer à la fois des caractéristiques locales et globales avec une excellente précision. Sa structure en forme de "U", combinant encodage et décodage, permet une localisation précise des objets, même avec un nombre limité d'images. La version **U-Net Mini** a été choisie pour réduire la complexité computationnelle, tout en conservant l’efficacité de la segmentation. Cela permet une intégration plus facile dans des environnements embarqués et une réduction des besoins en ressources.

De l’autre côté, **DeeplabV3** est une architecture de segmentation sémantique de pointe, reconnue pour ses performances sur des scènes complexes, notamment grâce à l’utilisation des convolutions dilatées. En l’associant avec **ResNet**, un réseau très profond permettant d’extraire des caractéristiques multi-échelles, DeeplabV3 devient un modèle idéal pour des tâches nécessitant une compréhension détaillée et multi-niveaux de l’image. Ce modèle est particulièrement efficace pour traiter des images à haute résolution comme celles du jeu de données **Cityscapes**.

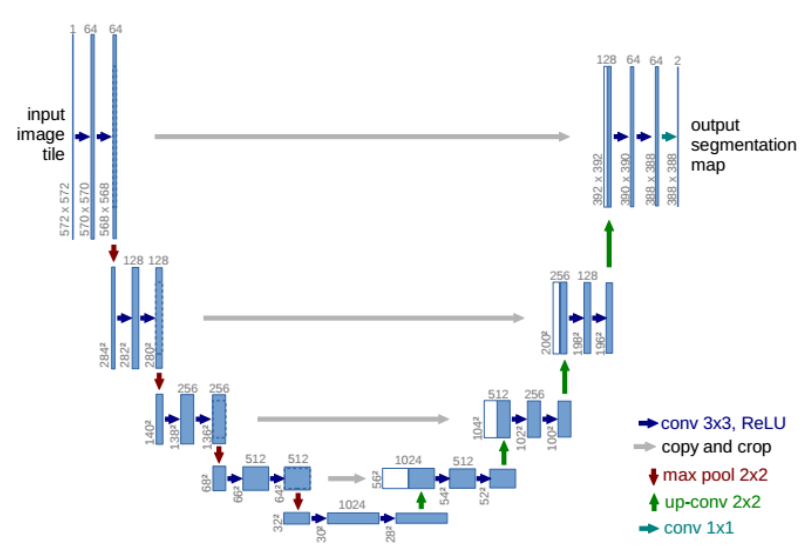
**Architecture des modèles**

#### **U-Net**

**U-Net** est un modèle basé sur une architecture d’encodage-décodage symétrique. Le réseau est composé de deux parties principales :

1. **Chemin d'encodage** : Cette partie réduit progressivement la dimension de l'image tout en extrayant les caractéristiques importantes. Chaque bloc d'encodage consiste en deux convolutions suivies d'une normalisation et d'une activation, puis d'une opération de pooling pour réduire la résolution spatiale.
2. **Chemin de décodage** : Cette partie reconstruit progressivement l'image en utilisant des opérations de upsampling pour augmenter la résolution. Chaque bloc de décodage reçoit des informations du chemin d'encodage via des connexions directes (skip connections), permettant ainsi de conserver des informations détaillées.

La version **U-Net Mini** que nous utilisons est une version allégée, avec un nombre réduit de couches et de paramètres, ce qui permet de s'adapter à des contraintes computationnelles plus strictes tout en maintenant une précision acceptable.

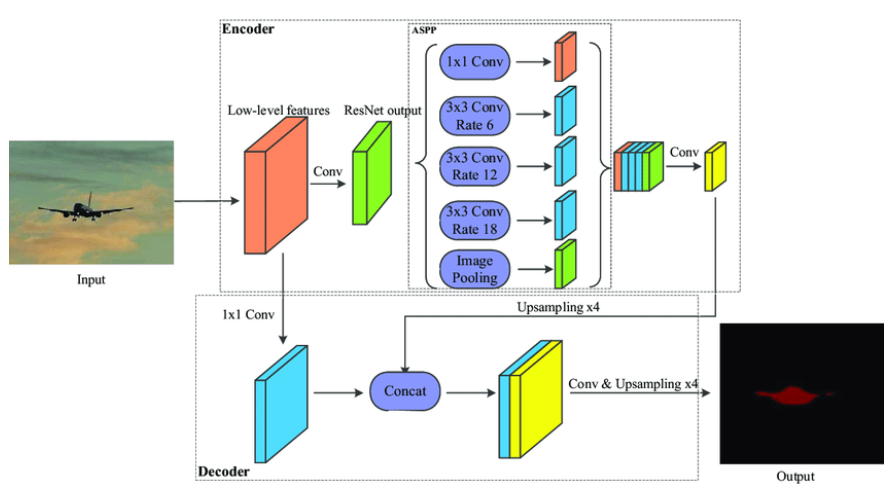


#### 

#### **DeeplabV3 avec ResNet**

L’architecture **DeeplabV3** repose sur l’utilisation de convolutions dilatées (ou atrous), permettant d’extraire des caractéristiques à différentes échelles sans perdre la résolution spatiale. Cela permet au modèle de mieux capturer le contexte global d’une image, tout en préservant les détails fins.

L'intégration de **ResNet** comme backbone renforce la capacité du modèle à extraire des caractéristiques complexes grâce à ses blocs résiduels, qui aident à éviter la dégradation des gradients dans les réseaux très profonds. Ce modèle s'avère particulièrement utile pour des images où les objets peuvent varier en taille et en forme, comme dans le jeu de données **Cityscapes**.



### 

### **Préparation des données**

Les données utilisées pour ce projet proviennent du jeu de données **Cityscapes**, qui est conçu pour la segmentation sémantique des scènes urbaines. Les images ont été collectées à partir de véhicules embarqués dans diverses villes, et elles reflètent fidèlement l'environnement routier auquel un véhicule autonome peut être confronté.

Les données sont organisées en deux dossiers principaux :

* Un dossier contenant les prises de vues depuis le véhicule (images RGB).
* Un second dossier contenant trois types de fichiers par prise de vue : les **masques**, les **labels**, et des fichiers **JSON** associés.

Bien que 32 labels soient disponibles dans les annotations du jeu de données, nous avons regroupé ces classes en 8 catégories principales pour simplifier le problème : **Route**, **Personne**, **Voiture**, **Bâtiment**, **Végétation**, **Ciel**, **Poteau**, et **Sol**.

#### **Data Processor et traitement par lots**

La préparation des données a été effectuée par Franck et implique l’utilisation d’un générateur de données (**DataGenerator**) qui prend en charge le chargement, le redimensionnement, la normalisation et, si nécessaire, l’augmentation des images et des masques associés. Le **DataGenerator** permet également de gérer le traitement par lots, optimisant ainsi la gestion de la mémoire et les performances durant l'entraînement.

### **Optimisation des paramètres**

#### **Fonction Loss**

Pour la fonction de perte, nous avons opté pour le **Dice Loss**, une métrique bien adaptée aux tâches de segmentation, car elle favorise la correspondance entre les régions prédites et les régions cibles. Cette fonction de perte est particulièrement efficace dans les situations où les classes peuvent être déséquilibrées.

#### **Batch Size et Grid Search**

Le **Batch Size** a été fixé à 4, un compromis entre l’utilisation de la mémoire GPU et la stabilité de l’entraînement. Nous avons également utilisé la **Grid Search** pour tester plusieurs valeurs d’hyperparamètres tels que le taux d'apprentissage, afin de trouver les configurations optimales.

#### **Nombre d’Epoch et Early Stopping**

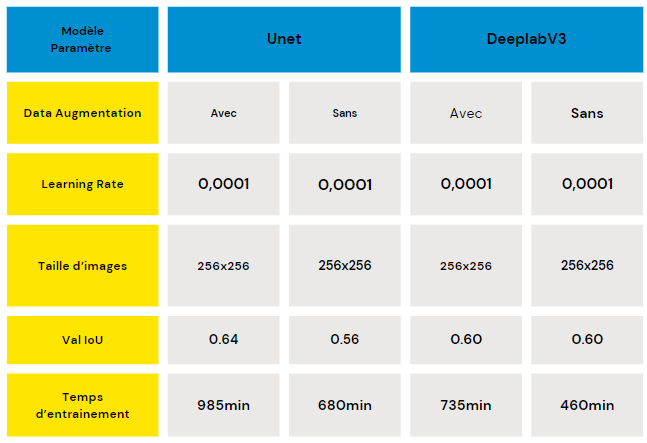
L’entraînement des modèles a été réalisé sur un nombre fixe de 35 **epochs** avec l’ajout d’un mécanisme d’**Early Stopping**, qui arrête l’entraînement lorsque la performance sur l’ensemble de validation ne s’améliore plus après plusieurs itérations consécutives. Cela permet d’éviter le sur-apprentissage et d’économiser du temps de calcul.

**4. Synthèse des résultats obtenus**

Nous allons utiliser principalement deux métriques pour observer les résultats des nos entrainements :   
- Le **Dice Coefficient**, également connu sous le nom de score de similarité de Dice, est une métrique qui mesure la similarité entre deux ensembles. En termes de segmentation d'image, il compare la correspondance entre les pixels prévus par le modèle et les pixels réels.

-L'**Intersection over Union** (IoU) est une autre métrique largement utilisée pour évaluer les performances des modèles de segmentation. L'IoU mesure le rapport entre l'intersection et l'union des ensembles de prédiction et de vérité terrain (ground truth).

Voici un tableau comparatif :

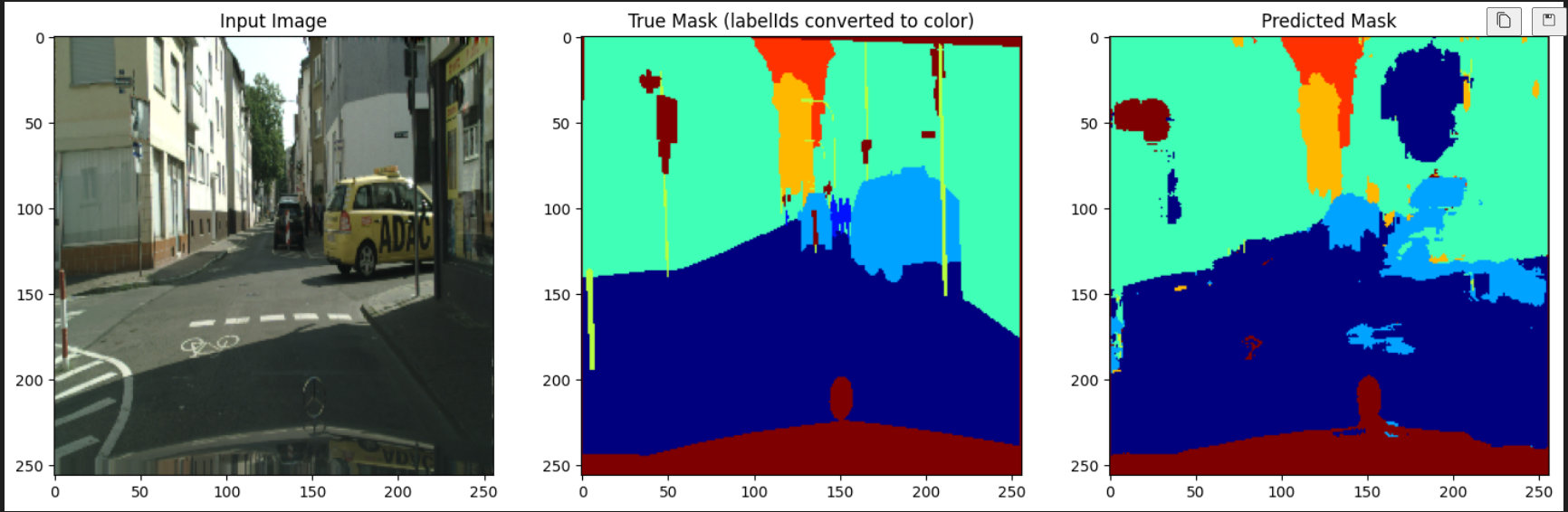


Nous avons effectué plusieurs tests concernant la taille des images utilisées et le learning rate, sur un type de modèle.  
Le plus grand défi a été de contenir le temps d'entraînement et d'estimer si le rapport efficacité/temps d'entraînement en valait la peine.  
C'est donc la dimension 256x256 et le learning rate 0.0001 qui ont été choisis et appliqués sur les autres types de modèles.

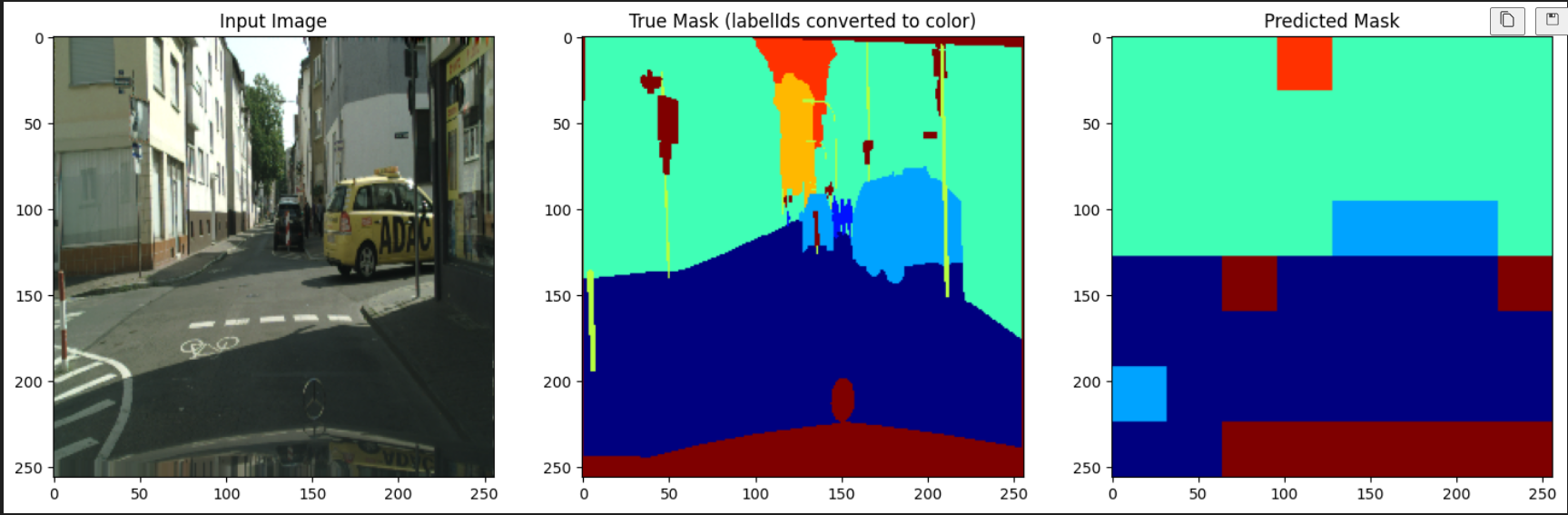
Nous utiliserons la sauvegarde du modèle U-Net avec Data Augmentation pour la présentation de l'utilisation sur l'API.  
L'intérêt est de montrer les résultats les plus efficaces, mais DeepLabV3, malgré un IoU plus faible, a du potentiel.  
La Data Augmentation apporte une amélioration notable de l'entraînement dans les deux cas.

Voici deux visualisations des prédictions réalisées par U-Net et DeeplabV3 avec la Data Augmentation.

Avec Unet :



Avec DeepLabV3 :



**5. Déploiement et intégration**

### **Déploiement du modèle**

Le déploiement des modèles de segmentation (U-Net) a été réalisé pour permettre leur utilisation à travers une interface Web et une API. Cette étape est essentielle pour rendre les modèles accessibles aux utilisateurs finaux, qu'ils soient intégrés dans une application embarquée, un système de test, ou une plateforme cloud.

#### **API de prédiction**

Pour faciliter l'utilisation des modèles de segmentation, une **API** a été développée en utilisant la technologie **Flask**. Cette API permet d'effectuer des prédictions sur des images en envoyant une requête HTTP, rendant le modèle accessible pour des applications externes.

* **Technologie utilisée** : Le choix de Flask est motivé par sa simplicité et sa flexibilité pour la création d'API RESTful. Son écosystème léger permet de déployer rapidement des modèles de machine learning et de définir des points d'entrée pour les prédictions.
* **Fonctionnalités de l'API** :
  + **Entrée d'image** : L'API accepte des images en entrée, qui sont ensuite prétraitées et redimensionnées pour correspondre aux exigences du modèle (par exemple, 256x256 pixels).
  + **Sortie du masque prédit** : Une fois l'image traitée par le modèle, l'API renvoie le masque de segmentation correspondant. Ce masque peut être visualisé pour vérifier la précision du modèle sur l'image fournie.

La structure de l'API a été conçue pour être extensible, permettant ainsi d’intégrer plusieurs modèles de segmentation, si nécessaire, et de supporter différents formats d'images.

#### **Application Web**

Une application Web a été développée à l'aide de **Streamlit** pour fournir une interface utilisateur interactive. Cette application permet de tester l'API et de visualiser les résultats des modèles de segmentation.

* **Technologie utilisée** : **Streamlit** est un outil puissant pour créer des applications Web en quelques lignes de code Python. Son utilisation a été motivée par sa simplicité et sa capacité à intégrer des graphiques, des images, et des fonctionnalités interactives.
* **Fonctionnalités de l'application** :
  + **Interface de test de l'API** : L’application permet à l'utilisateur de charger une image, d'envoyer une requête à l'API de prédiction, et de recevoir le masque de segmentation prédite.
  + **Affichage des images et masques** : Les images d'entrée et les masques prédits sont affichés côte à côte, offrant une visualisation claire des performances du modèle. Des options supplémentaires, comme l'ajustement des paramètres de segmentation, peuvent être ajoutées pour affiner les résultats.

#### **Déploiement sur le Cloud**

Pour le déploiement, nous avons stocké sur Microsoft Azure l'API de prédiction ainsi que la sauvegarde du modèle utilisé. Ainsi, Laura, avec le script de l'application Web, pourra charger l'image qu'elle souhaite afin d'observer les prédictions.

**6. Conclusion et pistes d’amélioration**

### **Résumé des résultats**

Les modèles de segmentation utilisés, **U-Net** et **DeeplabV3**, ont démontré des performances satisfaisantes sur le jeu de données Cityscapes. **U-Net** s'est révélé efficace, atteignant un IoU de 0.64 avec l'utilisation de la data augmentation, tandis que **DeeplabV3** a obtenu un IoU de 0.60. Le compromis entre l'efficacité et le temps d'entraînement a guidé notre choix d'opter pour une taille d'image de 256x256 et un taux d'apprentissage de 0.0001. Les résultats montrent également que la data augmentation améliore notablement les performances des modèles, justifiant ainsi son utilisation dans la phase d'entraînement.

L'intégration des modèles a été réalisée avec succès via une **API de prédiction** construite en Flask et une **application Web** en Streamlit, offrant un moyen interactif de tester les prédictions des modèles déployés sur le cloud.

### **Améliorations possibles**

#### **1. Optimisation des paramètres**

Bien que les modèles aient été optimisés avec un certain nombre d’hyperparamètres, il est possible de pousser cette optimisation plus loin. Une exploration plus approfondie des paramètres, tels que le **taux d'apprentissage**, le **batch size**, et les architectures des modèles (par exemple, l’ajustement de la profondeur de U-Net Mini), pourrait améliorer les performances. La mise en place d'une **optimisation avec GridSearch** des hyperparamètres pourrait également permettre d’identifier des configurations plus performantes.

#### **2. Techniques avancées de data augmentation**

La data augmentation s’est avérée bénéfique pour les deux modèles. Cependant, des techniques avancées pourraient être explorées, telles que :

* **Augmentation géométrique** : Ajout de transformations plus complexes comme la rotation, les déformations élastiques ou la mise à l'échelle.
* **Augmentation colorimétrique** : Ajustement des niveaux de couleur, du contraste, ou de la luminosité pour simuler des variations de conditions lumineuses.
* **Génération de données synthétiques** : Utilisation de modèles génératifs comme les GANs (Generative Adversarial Networks) pour augmenter la diversité des images dans le jeu de données.

Ces techniques pourraient potentiellement enrichir le modèle et l’aider à mieux généraliser à des scénarios variés.

#### **3. Adaptation à d'autres jeux de données**

Afin d'améliorer la généralisation des modèles, l’adaptation à d'autres jeux de données pourrait être envisagée. Tester les modèles sur des jeux de données d'images de différentes conditions de conduite (nuit, pluie, brouillard, etc.) permettrait de les rendre plus robustes. Une possibilité serait d’utiliser le **transfert de domaine** pour affiner les modèles avec de nouveaux ensembles de données, tout en conservant l'apprentissage déjà acquis.

#### **4. Scalabilité et temps de calcul**

L'un des principaux défis reste le temps de calcul, notamment pour le modèle DeeplabV3, plus gourmand en ressources. Voici quelques pistes pour améliorer la scalabilité et réduire les temps de calcul :

* **Optimisation matérielle** : Exploiter davantage les capacités des GPU ou des TPU pour accélérer l'entraînement et l’inférence des modèles.
* **Quantification du modèle** : Réduire la taille des modèles en utilisant des techniques telles que la quantification, permettant ainsi une exécution plus rapide tout en limitant la perte de précision.

Ces améliorations pourraient permettre d'intégrer ces modèles dans des systèmes embarqués tout en maintenant une performance acceptable.