**Note technique**

Cette fiche technique aura pour but de montrer l’évolution de mon raisonnement et mes différentes démarches afin de créer un modèle de segmentation sur le jeu de données Cityscapes.

Nous ferons dans un premier temps des tests sur un reseau de neurones d’entrainement (avec une architecture simple) afin de pouvoir tester différentes choses rapidement. Une fois qu’on aura identifié les paramètres et prétraitement d’images les plus efficaces, on les essaiera sur des réseaux plus compliqués et nous garderons celui présentant les meilleurs résultats.

-------------------------------------------------SOMMAIRE------------------------------------------------------

I) Les différentes approches

1) « Dummy training » (Reseau convolutif simple)

a) Sans pré-traitement d’image

b) Avec pré-traitement d’image

2) Différentes architectures

a) Modèle PSPNet

b) Modèle Unet

c) Modèle FPN

II) Présentation du meilleur modèle

III) Pistes d’amélioration

1. Les différentes approches
2. Sans pré-traitement d’image

Pour cette premiere partie, j’ai utiliser un reseau convolutif simple de type R-CNN .

Nous avons eu les résultats suivants :

|  |  |  |  |
| --- | --- | --- | --- |
| coefficient Sørensen-Dice de similarité | Temps d’entrainement | Validation sparse  Categorical accuracy | Sparse categorical accuracy |
| 0.61 | 5860 secondes | 0.62 | 0.61 |

Ici, la précision est calculée en faisons la moyenne de la précision sur toute les classes de segments.

Les résultats sont plutôt mauvais, mais ce n’est pas étonnant étant donné qu’il n’y a eu aucun process d’image et que le réseau de neurones est basique

1. Avec pré-traitement d’image

Le prétraitement d’image inclue :

-On met les photos en noir et blanc

-Ajout de bruit

-Mettre toutes les images au même format

-Augmentation d’image (découpage, flou, rotation)

|  |  |  |  |
| --- | --- | --- | --- |
| coefficient Sørensen-Dice de similarité | Temps d’entrainement | Validation sparse  Categorical accuracy | Sparse categorical accuracy |
| 0.67 | 7641 secondes | 0.68 | 0.69 |

Rotation d’image aléatoire : On fait des rotations de manière aléatoire afin d’avoir tous les angles possibles

Ajout de bruit : on floute les images pour supprimer les détails « inutiles »

Découpage d’image : on les coupe aléatoirement pour ne garder qu’une partie d’un segment.

Ces prétraitement un a un apporte peu de résultats mais le cumul des trois semble être efficace

En les testant un a un, je me suis rendu compte que les résultats étaient les même qu’avec le modèle sans prétraitement voir légerement inférieur



On aperçoit ici une prédiction sur une image. Ce n’est pas vraiment concluant .

A noter qu’ici les segments sont ont teintes de blanc/gris/noir mais que je vais mettre des couleurs par la suite.

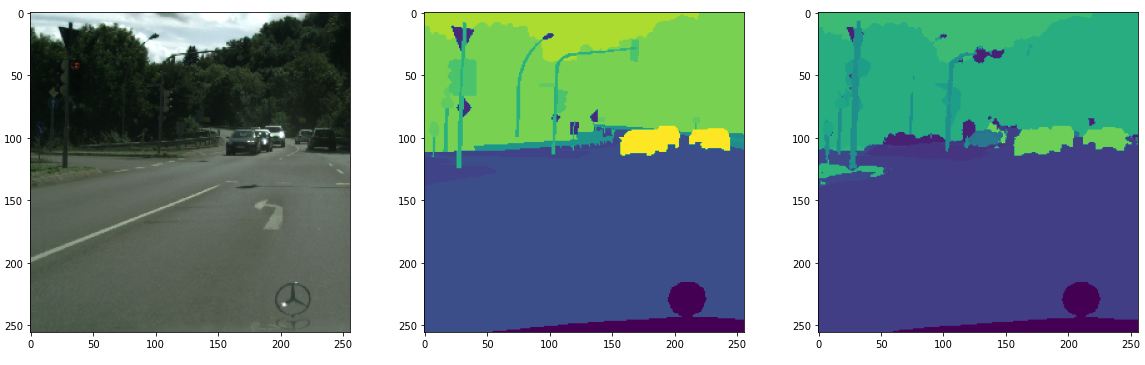
2) Différentes architectures

Nous allons maintenant essayer des architectures plus complexe, avec le même prétraitement d’image que nous avons éfféctué pour la première partie

1. PSPNet

Nous avons eu les résultats suivants :

|  |  |  |  |
| --- | --- | --- | --- |
| coefficient Sørensen-Dice de similarité | Temps d’entrainement | Validation sparse  Categorical accuracy | Sparse categorical accuracy |
| 0.78 | 8803 secondes | 0.80 | 0.74 |



^

Photo originelle Photo segmentée Prédiction

Les résultats sont meilleures et visuellement on s’en aperçoit bien ! Il y a encore des ratés, c’est un peu « brouillon » mais cela vient probablement du manque de données. Non pas qu’il n’y en ai pas assez (on peut en avoir beaucoup grace au generateur de données ) mais on est assez limité au niveau des ressources par l’ordinateur .

1. FPN (Feature pyramid network)

|  |  |  |  |
| --- | --- | --- | --- |
| coefficient Sørensen-Dice de similarité | Temps d’entrainement | Validation sparse  Categorical accuracy | Sparse categorical accuracy |
| 0.77 | 8124 secondes | 0.79 | 0.72 |

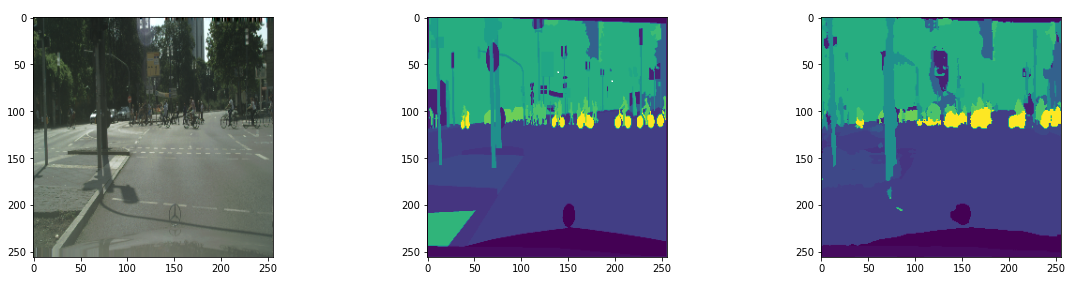


Photo originelle Photo segmentée Prédiction

Les résultats sont légerement moins bons que le qu’avec PSPNet mais reste dans le même ordre de grandeur. Le modèle est tout a fait acceptable également.

1. Architecture Unet

|  |  |  |  |
| --- | --- | --- | --- |
| coefficient Sørensen-Dice de similarité | Temps d’entrainement | Validation sparse  Categorical accuracy | Sparse categorical accuracy |
| 0.79 | 22754 secondes | 0.82 | 0.79 |

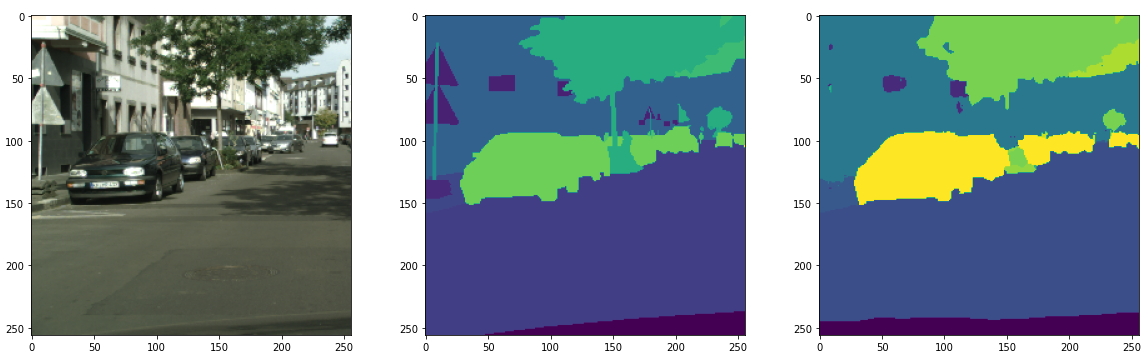


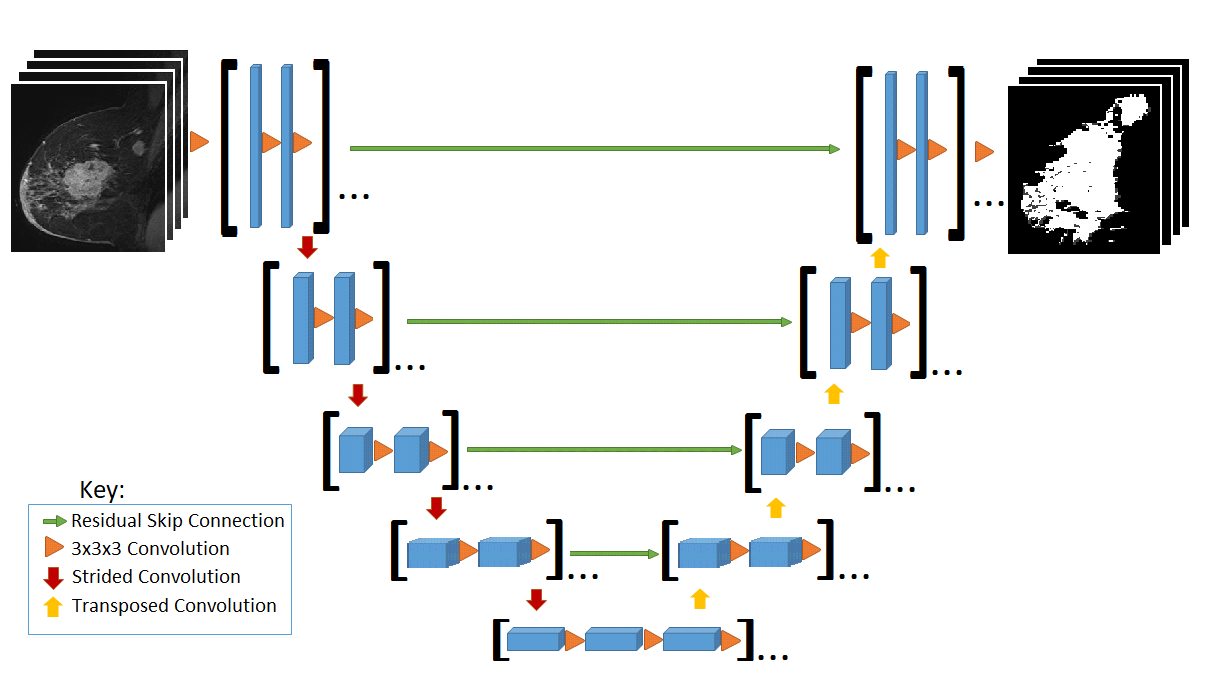
Photo originelle Photo segmentée Prédiction

Cette architecture présente les meilleurs résultats pour nos données mais le temps d’entrainement est 3x plus long que pour les deux architectures précédentes !

1. Présentation du meilleur modèle

Il s’agit d’un 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.



Les avantages de U-NET :

Dans le domaine du Deep Learning, il est nécessaire d’utiliser de larges ensembles de données pour entraîner les modèles. Il peut être difficile d’assembler de tels volumes de données pour résoudre un problème de classification d’images, en termes de temps, de budget et de ressources.

L’étiquetage des données requiert aussi l’expertise de plusieurs développeurs et ingénieurs. C’est particulièrement le cas pour des domaines hautement spécialisés comme les diagnostics médicaux.

U-NET permet de remédier à ces problèmes, puisqu’il s’avère efficace même avec un ensemble de données limité. Il offre aussi une précision supérieure aux modèles conventionnels. Une architecture autoencoder classique réduit la taille des informations entrées, puis les couches suivantes. Le décodage commence ensuite, la représentation de caractéristiques linéaire est apprise et la traille augmente progressivement.

À la fin de cette architecture, la taille de sortie est égale à la taille d’entrée. Une telle architecture est idéale pour préserver la taille initiale. Le problème est qu’elle compresse l’input de façon linaire, ce qui empêche la transmission de la totalité des caractéristiques. C’est là que U-NET tire son épingle du jeu grâce à son architecture en U. La déconvolution est effectuée du côté du décodeur, ce qui permet d’éviter le problème de goulot rencontré avec une architecture auto-encodeur et donc d’éviter la perte de caractéristiques.

1. Conclusion

Nous avons précedemment vu que même si les résultats étaient bons, ils sont améliorables.

Je pense que la grosse contraintes ici a été le manque de données. On pourrait imaginer utiliser d’autres méthodes d’augmentation d’image afin d’obtenir plus de données mais cela nous amène a un autre problème qui sont les ressources hardware, il faudrait une machine beaucoup plus puissante, entrainer des modèles pendant plus de 6H étant plutôt contraignant.

On pourrait aussi récolter plus de données de manière brutes, et aussi imaginer mais de manière plus spéculative qu’une autre architecture serait plus efficace sur nos données.