Rapport de Modal

Segmentation d'images routière par réseau de neurones

Guillaume Couairon

Janvier 2017

Enseignants: Bertrand Le Saux & Joris Guerry

Encadrants: Ossikovski Razvigor & Jean-Charles Vanel

Remerciements

Je remercie vivement les enseignants de ce modal Mr Le Saux et Mr Guerry pour m'avoir accompagné dans la conduite de ce projet et m'avoir apporté leur aide précieuse.

Je remercie particulièrement Mr Guerry pour l'installation de l'environnement de travail que j'ai utilisé, et que je n'aurai jamais réussi à mettre en place tout seul.

J'exprime également ma gratitude envers Mr Vanel, qui s'est déplacé de nombreuses fois pour venir m'ouvrir la salle de modal.

Sommaire

Introduction

Les voitures autonomes sont un marché grandissant et la quasi-totalité des constructeurs automobiles y dévouent une part de plus en plus conséquente dans leur budget recherche & développement. Or, un enjeu majeur de ce type de véhicule est la sécurité : comment s'assurer que la voiture autonome ne représentera pas un danger pour les piétons et les autres voitures sur la route ? Ainsi, cette voiture doit pouvoir reconnaître son environnement afin de pouvoir se diriger comme le ferait un automobiliste classique. J'ai donc choisi d'étudier ce problème avec en tête l'idée de pouvoir reconnaître où se situent les piétons et voitures sur une image routière, obtenue avec une simple caméra posée dans la voiture.

I) Approches retenues et protocole d'étude

A) Cahier des charges

Pour pouvoir “détecter les piétons et voitures”, il faut choisir un moyen d'affirmer que le piéton ou la voiture a bien été détectée. Nous n'avons pas seulement cherché à pouvoir affirmer si l'image présentée contenait où non un piéton, nous avons cherché à classifier chaque pixel de l'image comme appartenant à une certaine catégorie parmi les huit suivantes :

Nom de la classe	**Signification / précisions**
Void	Inclassable
Ground	Route, trottoir, parking
Building	Constructions, mur, pont, tunnel
Nature	Végétation, terrain
Vehicle	Voitures, cyclistes, train, motos, bus...
Sky	ciel
Object	Panneaux de visualisation, poteaux
Human	Personnes, piétons

L'algorithme est donc efficace s'il établit une classification fidèle à ce qu'un humain pourrait effectuer en fonction de ses classes. Pour mesurer la performance de notre algorithme, nous disposons d'une base de données d'images labellisées à la main (cette labellisation correcte est ce qu'on appelle “la vérité terrain”). Chaque classe est donc définie par un ensemble de pixels dans chacune des images de cette base, et on mesure la perfomance de l'algorithme sur cette classe comme l'ensemble des pixels de la classe correctement labelisés.

L'objectif que nous nous sommes fixés consiste donc à obtenir la meilleur performance possible, c'est à dire à ce que le nombre de pixels correctement classifiés soit le plus grand possible.

B) Méthodes retenues

Les critères qui permettent à l'homme de décider à quelle classe appartient chaque pixel de l'image résultent d'un raisonnement complexe que nous ne savons pas expliciter, et encore moins implémenter. Toutefois on peut modéliser ce comportement par une fonction mathématique, qui à une image associe une image de classes. Grâce à des réseaux de neurones, on peut définir un modèle de calcul qui permet d'approcher n'importe quelle fonction mathématique en connaissant sa valeur sur un certains nombre de points, sans avoir à se soucier du comportement de cette fonction. Ainsi, cette technique permet d'apprendre automatiquement les critères de segmentation de l'image à partir d'une base de donnée appelée base d'entraînement, qui contient une grande quantité d'images et de leurs labellisations exactes. Son avantage majeur est son adaptabilité à tout type de situation où une fonction cérébrale humaine ne peut pas être comprise, analysée et implémentée : elle est donc particulièrement adaptée à des problématiques de détection.

1) Fonctionnement d'un réseau de neurones

Les réseaux de neurones artificiels sont inspirés de la biologie. Chaque neurone porte une information, un nombre réel. Un réseau de neurones est organisé en couches : La couche d'entrée (première couche) est un vecteur, comme ici une image.

Chaque couche intermédiaire est un vecteur calculé à partir de la couche précédente par la composition d'une application linéaire et d'une fonction réelle non linéaire appliquée à chaque neurone, c'est à dire chaque composante. La dernière couche est la sortie du réseau f(X).

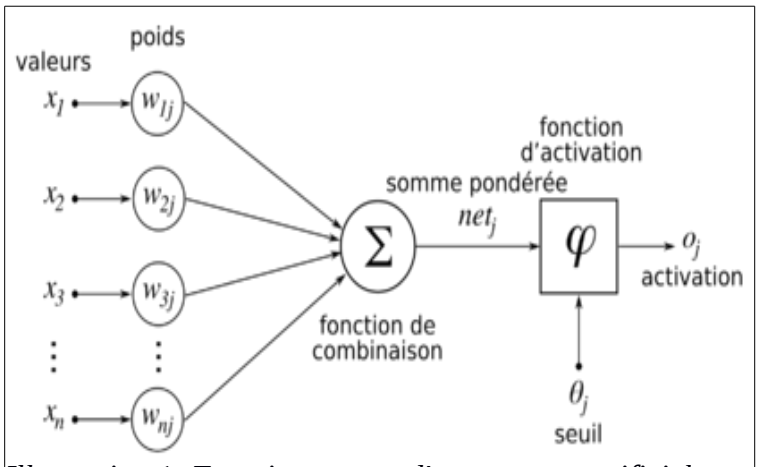


Illustration 1: Fonctionnement d'un neurone artificiel

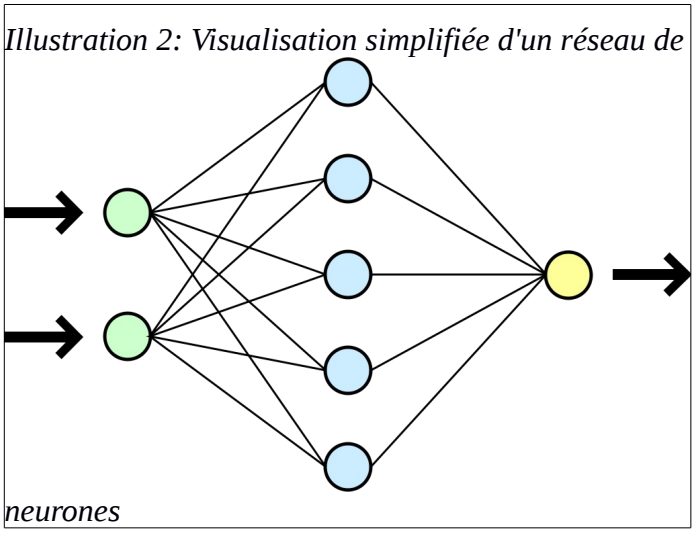


Illustration 2: Visualisation simplifiée d'un réseau de neurones

Une fois le vecteur de sortie f(X) calculé, il faut le comparer à Y. Des méthodes (descente de gradient) permettent de savoir comment ajuster les paramètres des applications linéaires (appelés poids) pour réduire la différence entre f(X) et Y. Répéter ceci pour chaque image de la base de données d'entraînement permet progressivement d'ajuster les poids du réseau de neurones de manière à approximer la fonction que nous souhaitons obtenir.

Nous avons choisi pour l'implémentation une bibliothèque python extrêmement pratique, nommée Keras, qui effectue tout ce travail de mise à jour du réseau de neurones de manière autonome.

Deux techniques sont possibles pour attribuer un label à chaque pixel.

2) Classification par découpage de l'image

Tout d'abord, on découpe l'image en cadres de taille plus petite (par exemple 64*64). Les cadres peuvent se superposer: le découpage le plus simple consiste à choisir le cadre en haut à gauche, puis à le décaler d'un certain pas à droite ou vers le bas, et ce pour toute l'image. Ensuite, notre réseau de neurones attribue une classe unique à tout le cadre. Pour cela, la couche de sortie du réseau de neurones doit comporter le même nombre de neurones que de classes. On détermine alors la classe de l'image en prenant l'indice du neurone de sortie ayant la plus grande valeur.

Chaque pixel du cadre se voit attribuer cette classe. Par recouvrement, chaque pixel se voit affecter plusieurs classes: un vote majoritaire permet de déterminer une classe unique.

Notre base d'entraînement doit donc être constituée d'images de la taille de ces cadres, auxquelles correspond un unique label.

3) Segmentation directe de l'image

La segmentation consiste à donner en entrée au réseau de neurones l'image entière, et à lui demander de prédire l'image des labels pour chaque pixel. La couche de sortie est donc une image de même taille que l'image d'entrée, mais avec une profondeur égale au nombre de classes (ici 8) et non 3 pour les trois canaux de couleurs. A chaque pixel est donc associé un tableau de taille 8, représentant les probabilités d'appartenance à chaque classe.

La classe de chaque pixel est alors déterminée en choisissant l'indice de l'élément maximal de ce tableau, qui est la classe la plus probable.

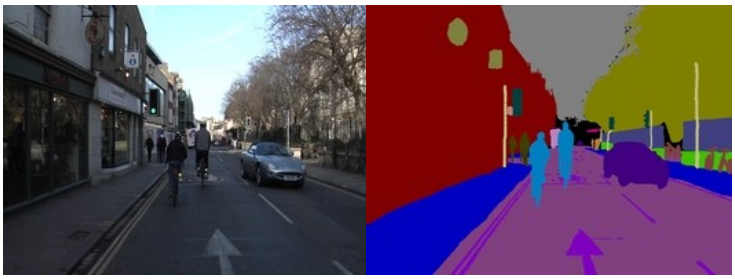
II) Traitement des données

A) Présentation des bases de données utilisées

J'ai utilisé deux bases de données disponibles gratuitement sur internet: Camvid et Cityscape, qui sont toutes les deux des bases d'images routières segmentées en différentes classes.

1) Camvid

Camvid est une base de donnée d'images prises dans la ville de Cambridge en résolution 360*480. Voici quelques exemples en colorant la vérité terrain avec une couleur pour chaque classe :

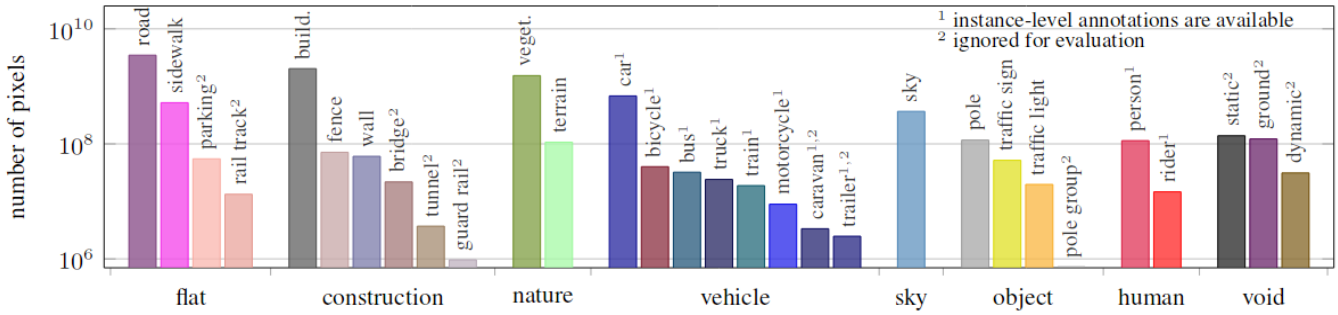



Les images sont issues de vidéos échantillonnées à une image par seconde.

2) Cityscape


Cityscape est une base de donnée plus moderne et plus variée: elle contient des images issues d'une vingtaine de villes européeennes. C'est pourquoi, si nos premiers résultats s'appuient sur Camvid, nous nous sommes tournés vers Cityscape pour des données plus variées et donc plus représentatives de la réalité. Les images sont en résolution 1024*2048, ce qui excède la capacité de traitement de l'ordinateur de la salle de modal: J'ai donc choisi de les compresser en 240*480, taille qui permet de conserver suffisamment de détails en allégeant la tâche de la machine. On pourrait bien sûr discuter de ce paramètre avec plus de puissance de calcul.

Cityscape possède 30 classes dont on peut voir les répartitions sur le diagramme suivant:



B) Création de la base d'entraînement pour la classification par découpage de l'image

Pour appliquer notre méthode de découpe d'image, il faut créer une base de données d'images de taille 64*64 auxquelles sont attribuées une unique classe pour chaque image. Cette étape est cruciale et détermine en grande partie la performance de l'algorithme puisque le réseau de neurones se base entièrement là-dessus.

Le principe est le suivant: pour chaque image complète de la base, on la découpe en cadres de taille 64*64 en partant du bord supérieur gauche et en décalant à chaque fois le cadre d'un pas de 20. La tâche consiste alors, à partir d'un cadre et des classes de chacun de ses pixels, à attribuer une classe unique à ce cadre. Nous avons adopté le principe suivant: si au centre de ce cadre, la proportion de pixels attribués à une classe dépasse une proportion seuil, alors le cadre est considéré comme un exemple de cette classe. On note qu'on peut choisir une proportion seuil différente pour chaque classe à condition de définir un ordre d'importance des classes, c'est à dire l'ordre dans lequel on vérifie que le cadre en cours est ou non un exemple de la classe i.

Formellement, nous définissons deux tableaux ordre_importance et filtre_attribution_label, de taille 8 (le nombre de classes), et considérons que les classes sont numérotées de 0 à 7. Nous appliquons alors le code suivant:

Pour i = 0..7 :
***centre** = cadre[12:-12]*
***classe** = ordre_importance[i]*
***filtre** = filtre_attribution_label[i]*
*Si le pourcentage de pixels de **centre** appartenant à la classe **classe** dépasse la valeur **filtre**, alors **cadre** est considéré comme un exemple de la classe **classe**.*
*Si aucune classe ne convient, alors le cadre **cadre** n'est pas retenu.*

Pour qu'un cadre se voit attribuer la classe i , il faut donc que la proportion de pixels en son centre dépasse la porportion seuil filtre_attribution[i] et que ce ne soit pas le cas pour toutes les classes plus importantes que i.

Pourquoi ne pas choisir la même proportion pour toutes les classes ? Il est très facile d'obtenir des exemples de ciel car les images présentent souvent de vastes régions labellisées «ciel». En revanche les piétons sont des objets beaucoup moins représentés et beaucoup plus difficiles à classifier car un piéton éloigné occupera un faible nombre de pixel alors qu'il est important de le classifier comme piéton pour pouvoir détecter les piétons éloignés. A l'inverse, il est sans importance de classifier une image contenant 50% de ciel comme du ciel car les exemples sont déjà abondants dans une image. Toutefois si l'on souhaite détecter des bordure de ciel, on peut garder ces cadres avec une certaine probabilité.

La qualité de la nouvelle base de donnée dépend donc fortement des paramètres filtre_attribution_label et ordre_importance. ordre_importance se détermine assez naturellement par l'ordre naturel qu'on obtient en se posant la question «qu'est ce qui est le plus important à détecter?». Nous avons retenu l'ordre suivant: human, vehicle, object, construction, nature, sky, ground, void.

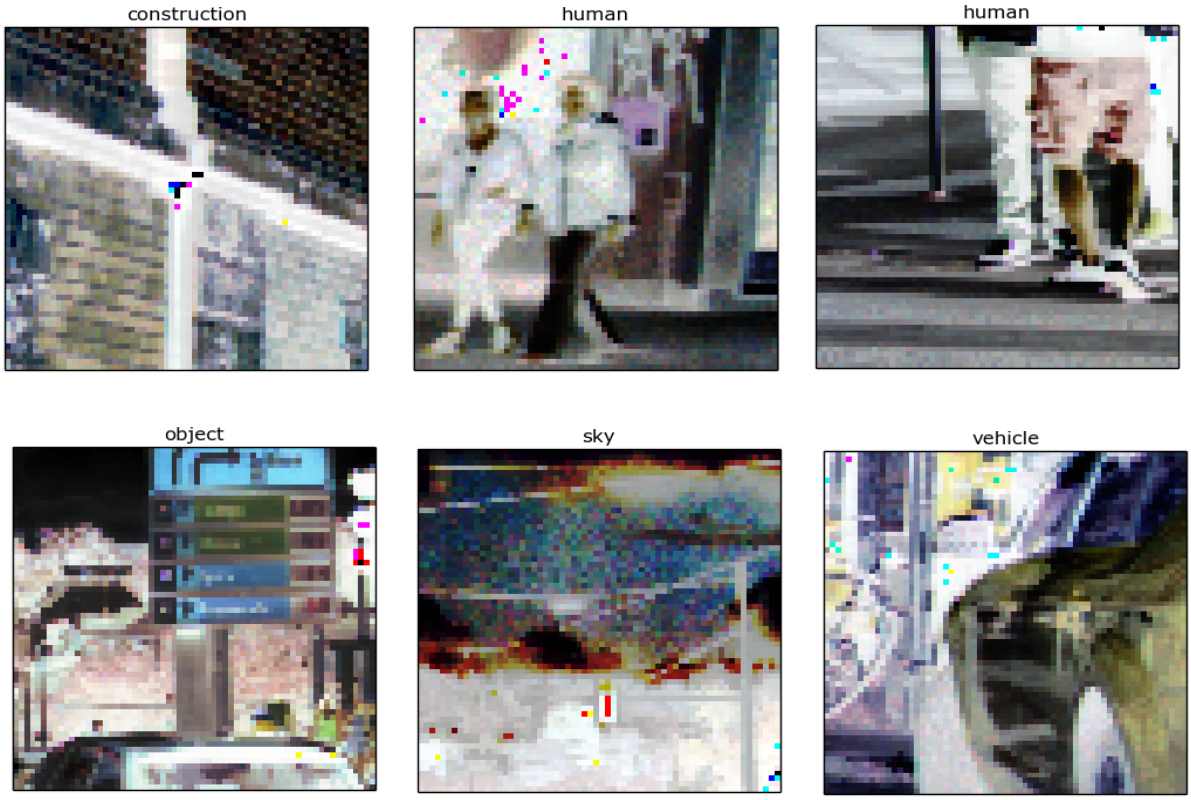
Pour filtre_attribution_label, on peut tester empiriquement plusieurs paramètres pour vérifier si l'on obtient suffisamment d'exemples pertinents pour chaque classe. Les paramètres retenus sont:

Classe	**Void**	**Ground**	**Building**	**Object**	**Nature**	**Sky**	**Human**	**Vehicle**
Valeur	0.5	0.8	0.8	0.4	0.7	0.5	0.2	0.75

Les paramètres des classes sur-représentées (sky, ground, building) sont laissés à une valeur suffisamment basse pour avoir des exemples variés. On pourra après utiliser le nombre d'exemples que l'on veut pour chaque classe.

Enfin, pour s'affranchir des problèmes de luminosité et de contraste, on applique à chaque cadre retenu pour la base de donnée une normalisation de l'histogramme pour chaque couleur (rouge, vert, bleu).

Voici quelques exemples de cadre obtenus (normalisés) et leurs classes:



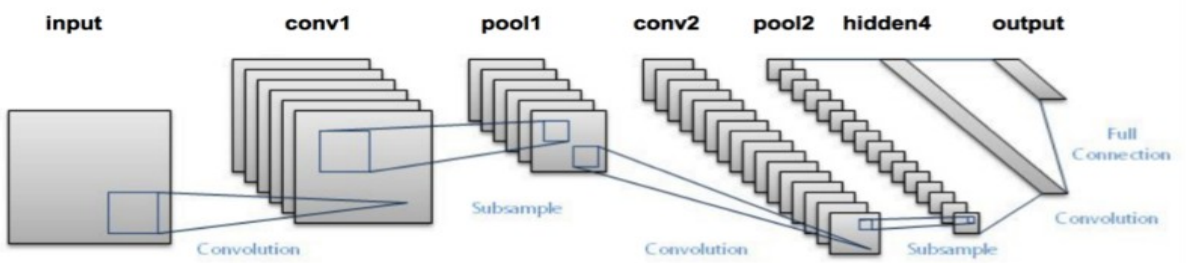
III) Mise en œuvre et résultats obtenus

Nous avons mis en place plusieurs architecture de réseaux de neurones pour pouvoir tester et les comparer : un réseau lenet et un VGG pour la classification par découpe d'image, et un autoencoder pour la segmentation directe.

A) Réseau LENET

1) Architecture

L'architecture de ce réseau est la suivante:



Ce réseau est composé de couches convolutives et de couches de « downsampling ». Les couches convolutives sont basées sur des convolutions mathématiques afin de s'affranchir des effets de translation. Grossièrement, chaque pixel dans l'image de sortie de la convolution dépend linéairement de ses voisins dans l'image d'entrée, avec les mêmes paramètres pour tous les pixels de l'image. Les couches de downsampling sont de type MaxPooling (voir partie « Question théorique » pour plus de détails). La couche nommée « FullyConnection » ou « Dense » est une couche dont tous les neurones sont connectés à tous les neurones de la couche précédente.

La tailles des couches utilisées sont données par le tableau suivant:

Input	Convolution 50	MaxPooling	Convolution 100	MaxPooling	Dense 1000	Dense 8

Une convolution de taille 50 multiple le nombre de neurones de la couche précédente par 50, toutefois ces neurones partagent beaucoup de paramètres.

2) Résultats théoriques

Ce réseau a été entraîné sur une base de 8000 exemples (1000 pour chaque classe). Voici les résultats obtenus :

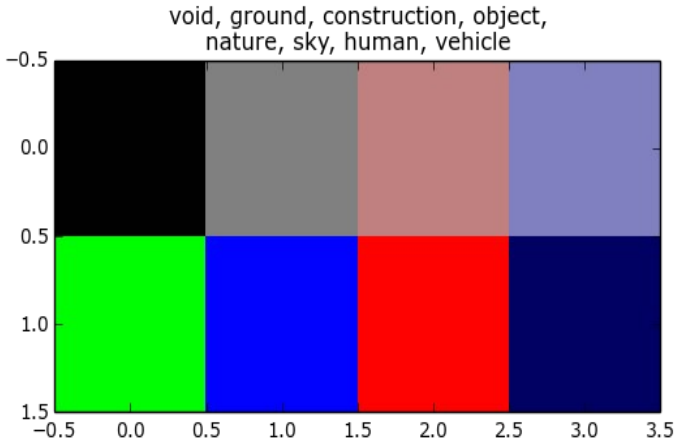
Classe	**Nombre de pixels correctement classifiés (entraînement)**	**Nombre de pixels correctement classifiés (test)**
Void	0.403	0.141
Ground	0.635	0.600
Construction	0.357	0.296
Object	0.333	0.286
Nature	0.551	0.482
Sky	0.901	0.834
Human	0.483	0.353
Vehicle	0.548	0.447
Mean	**0.526**	**0.430**

On remarque que pour chaque classe, l'algorithme a été meilleur sur la base d'entrainement que sur la base de test, ce qui est normal. La différence de performance varie de 10 à 20 % pour chaque classe, excepté pour la classe « void », qui est une classe difficile à définir. Il est donc normal que le réseau ait du mal avec cette classe.

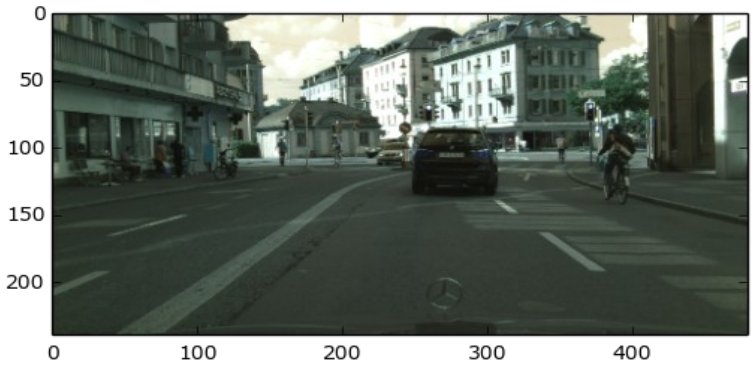
Le résultat significatif est la précision moyenne sur le base de test. Elle est ici de 43%, ce qui signifie que les pixels sont correctement labellisés un peu moins d'une fois sur deux.

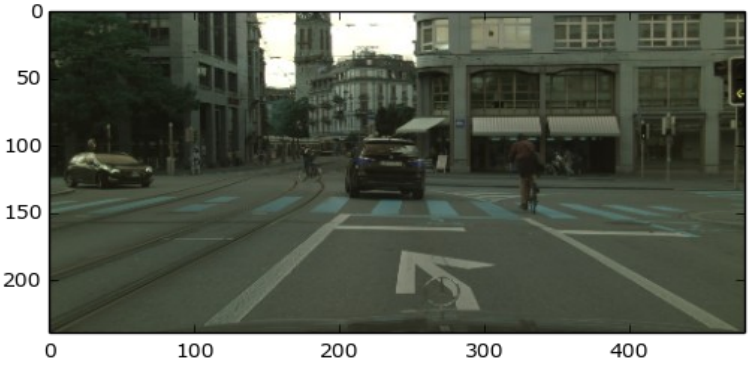
3) Visualisation des résultats

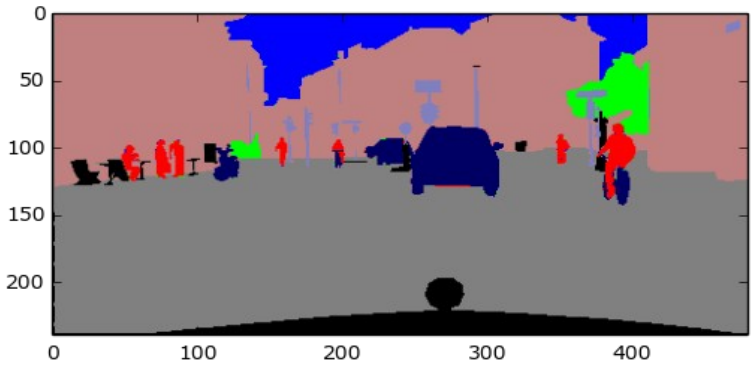
On peut visualiser le résultat en attribuant à chaque label une couleur. Les couleurs retenues sont représentées sur le diagramme ci-contre.

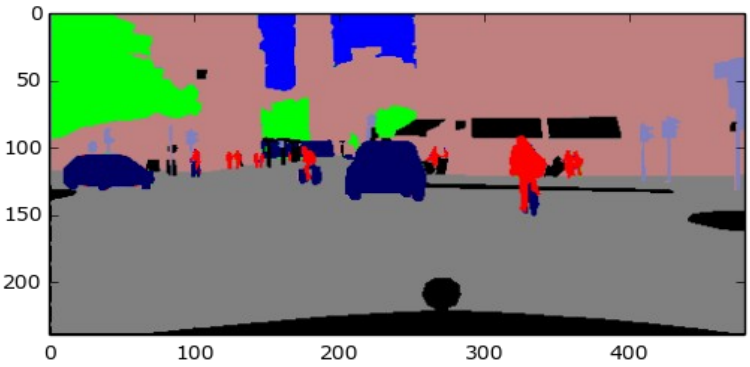


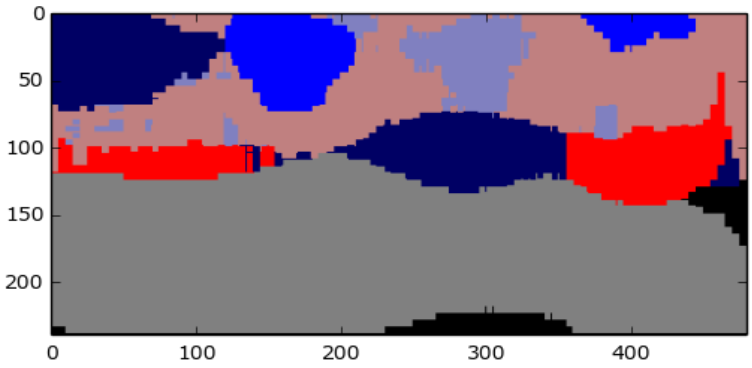
Voici les résultats obtenus pour trois images de la base de test: pour les deux colonnes sont présentées l'image originale, la vérité terrain et la prédiction effectuée par le réseau LENET.

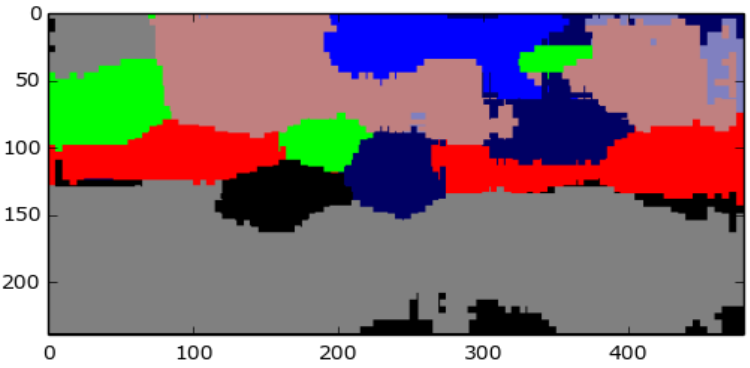












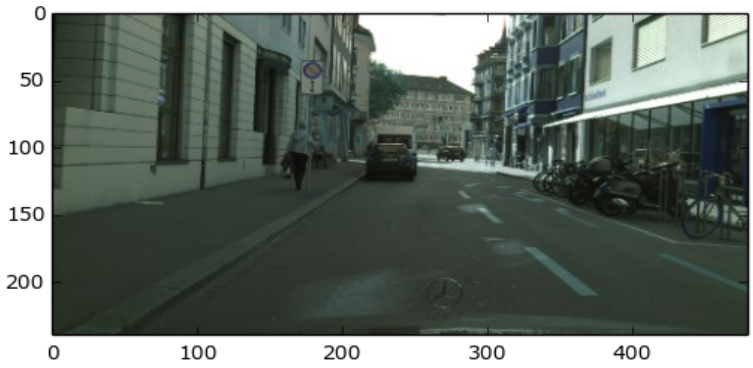
On peut faire plusieurs observations concernant ces images :

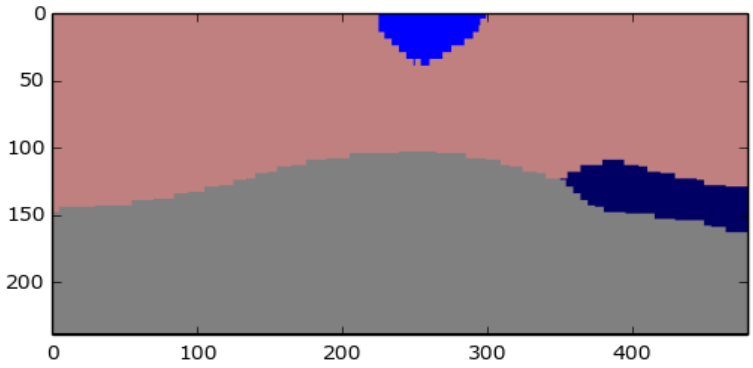
D'abord, la logique d'ensemble des images routières est respectée : de la route dans la moitié inférieure, majoritairement des bâtiments et du ciel dans la moitié supérieure.

Ensuite, les piétons sont bien détectés, mais beaucoup de pixels sont classifiés piétons alors que ce n'est pas le cas. On observe de larges tâches rouges aux endroits ou sont les piétons. Ceci est du au fait que nous avons choisi un filtre de détection de piéton assez bas, ce qui signifie que des images contenant relativement peu de pixels «piétons» seront classées piétons. Lors de la phase d'analyse de l'image, l'environnement des piétons est aussi classé piéton, ce qui explique ces taches.

On remarque aussi que beaucoup de pixels «bâtiments» sont classés «voiture», ce qui montre que l'algorithme a encore du mal, à ce stade, à distinguer les deux.

Une difficulté inhérente à la classification par découpage de l'image est que la vérité terrain telle que présentée ici n'est pas forcément analysée en tant que vérité terrain par notre méthode de création de la base de données. Par exemple, on peut appliquer un système de vote à la vérité terrain, pour chaque cadre 64*64 et en décalant les cadres de 20 pixels, ce qui donne le résultat suivant:





On note que le piéton de gauche et la voiture de devant sont simplement éliminés par le système de vote. Cette méthode a donc des limitations assez importantes.

B) Réseau « small VGG »

Nous mentionnons l'utilisation de ce réseau par souci d'exhaustivité des méthodes essayées, toutefois nous n'avons pas pu calculer sa performance sur une base de test.

Ce réseau est inspiré d'un réseau de neurones connu dans la littérature sous le nom de VGG 16. Toutefois certaines couches ont été enlevées, pour alléger le modèle : en effet notre carte graphique n'était pas assez puissante pour faire tourner un VGG complet.

1) Architecture

Les tailles de chaque couche sont définies comme suit:

Input	Convolution 50	Convolution 50	MaxPooling	Convolution 100	Convolution 100	MaxPooling

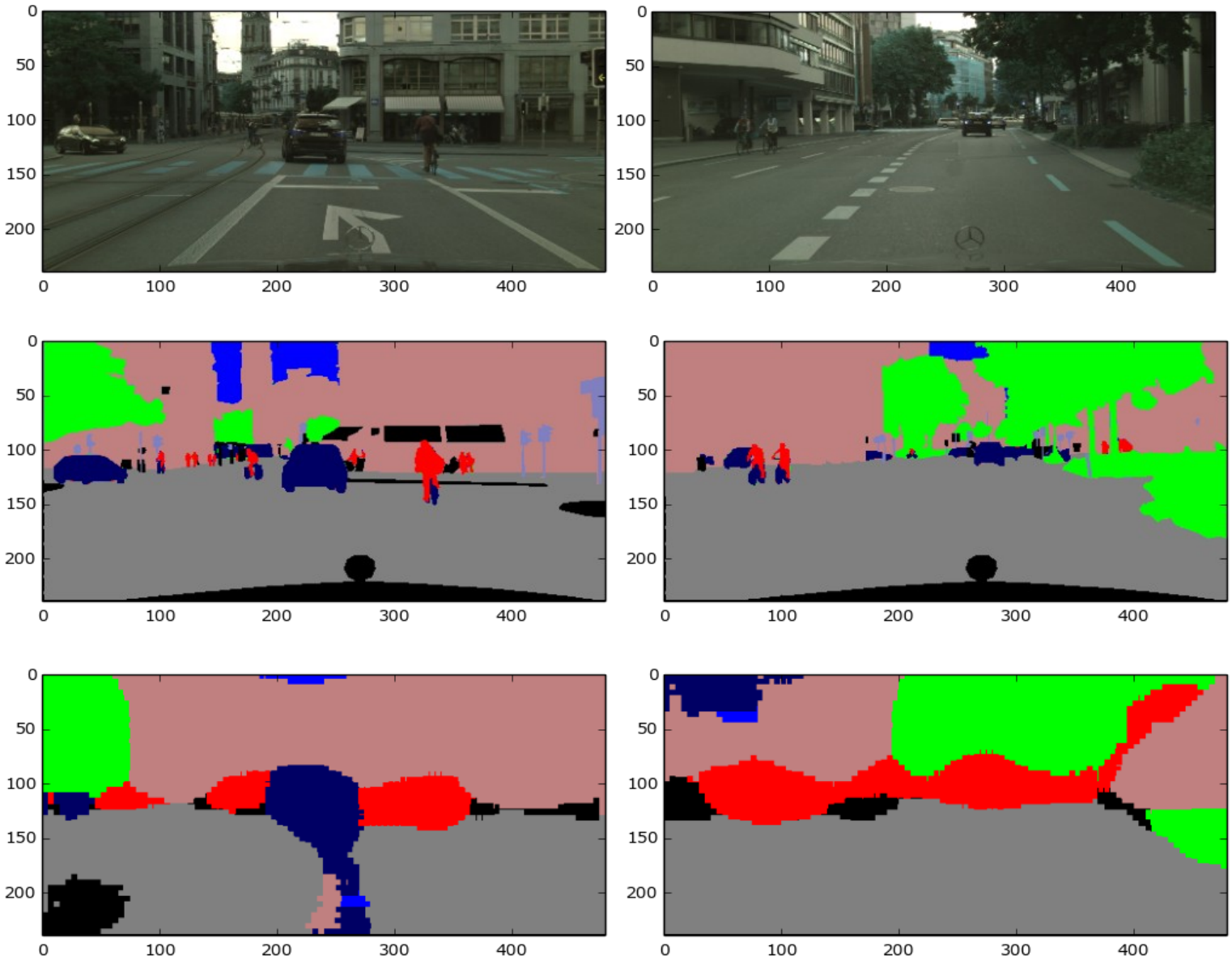
Convolution 200	Convolution 200	MaxPooling	Dense 1024	Dropout 0.5	Dense 8

Les couches utilisées sont les mêmes que celles du réseau précédent, à l'exception d'une couche de « Dropout » dont l'objectif théorique est de réduire le phénomène d'apprentissage par coeur du jeu de données (overfitting).

Ce réseau a été entraîné sur environ 70 000 images (un peu moins de 9000 exemples pour chaque classe) pendant 100 epochs et n'a pas été stoppé lorsque sa performance de validation a arrêté d'augmenter. Le réseau s'est très probablement contenté d'apprendre l'intégralité de la base d'entraînement (malgré la couche de Dropout) avec des cadres correctement labellisés avec une probabilité supérieure à 99.993%.

2) Visualisation

En utilisant la même technique de visualisation que celle évoquée plus haut, voilà le résultat que l'on obtient :



Dues aux limitations de la méthode de découpage d'image, et à l'overfitting, les résultats paraissent cohérents mais ne sont pas réellement meilleurs que ceux du réseau LENET.

On peut faire les même observations concernant l'abondance de piétons détectés: le seuil de détection est trop haut pour pouvoir localiser précisément un passant.

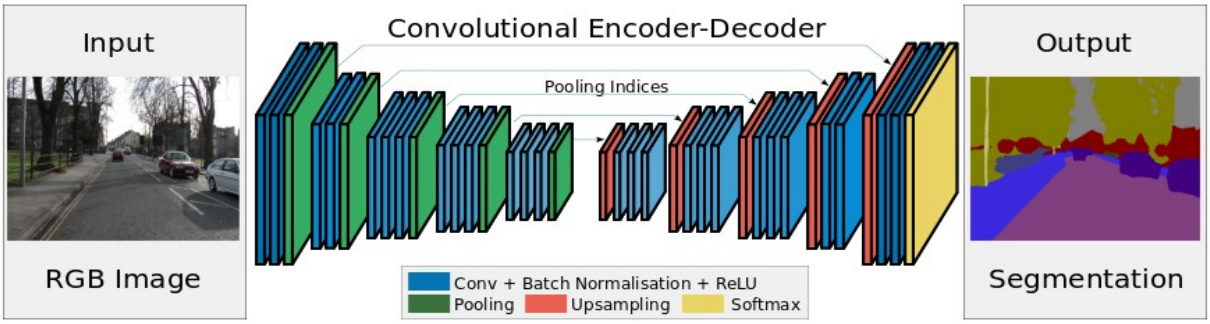
Ce réseau, comme le précédent, a été entraîné avec la méthode d'optimisation SGD, que nous ne détaillerons pas.

On pourrait relancer ce réseau en stoppant l'apprentissage au moment où le réseau arrête d'apprendre en contrôlant sa performance au moyen un ensemble d'images de validation. Toutefois j'ai préféré travailler sur de la segmentation directe, afin de m'affranchir des contraintes dues à la création d'une base d'entraînement dans la méthode de découpage d'image. Ceci fait l'objet de la troisième partie.

C) Segmentation par autoencoder

1) Principe et architecture

Le principe d'un autoencoder est de prendre comme entrée une image et en sortie une autre image, codant les classes de chaque pixel.
L'architecture est présentée dans le schéma ci-dessous :



Nous n'avons pas gardé les couches de «Batch Normalisation». Les couches de Pooling réduisent la taille de l'image tandis que celle de Upsampling l'augmentent, de manière à obtenir une représentation codée de l'image pour ensuite la décoder. L'objectif est de faire trouver au réseau de neurones une représentation des images routières qui ne garde que l'information liée à la segmentation en classes des images.

Les tailles de chaque couche sont les suivantes:

Input	Convolution 64	MaxPooling	Convolution 128	MaxPooling	Convolution 256

Convolution 256	UpSampling	Convolution 128	UpSampling	Convolution 64	Dense 8

Ce réseau a été entraîné avec un outil nommé 'adadelta' disponible dans keras. Le nombre total de paramètres de ce réseau est de 1,330,248. La base d'entraînement est constituée de 690 images prises dans les villes de zurich,

erfurt, bochum, dusseldorf et weimar. La base de test est constituée de 365 images prises à Strasbourg. Du fait de limitations techniques, nous n'avons pas pu utiliser toutes les villes présentes dans les bases de données de Cityscape.

Cette fois-ci, le réseau a été arrêté quand la précision sur un ensemble d'images de validation a cessé d'augmenter. On a également tenu compte du fait que les classes ne contiennent pas toutes le même nombre de pixels. On a alors attribué un coefficient à chaque classe de manière à prendre ce phénomène en compte. Le paramètre keras correspondant est nommé class_weighting, et nous avons adopté les valeurs suivantes, calculées comme les inverses des fréquences d'apparition de chaque classe.

Classe	**Void**	**Ground**	**Building**	**Object**	**Nature**	**Sky**	**Human**	**Vehicle**
weighting	12	2	4	52	6	30	138	14

La précision est alors calculée comme une moyenne pondérée des précisions sur chaque classe.

2) Résultats théoriques

Keras nous indique les performances suivantes pour ce réseau: 88.5% de précision pour la base d'entraînement et 84.35% pour la base de test.

Toutefois on peut mener une analyse plus poussée en calculant d'autres mesures de performance. En classification, on appelle vrais positifs de la classe i (TP) les exemples correctement labellisés i. Les faux positifs (FP) sont ceux qui sont labellisés i de manière incorrecte, et les faux négatifs (FN) sont ceux qui auraient dus être labellisés i.

- L'indice de précision est calculé comme TP/(TP+FP). Il répond à la question «Combien d'éléments classés i appartiennent réellement à i ?»

- L'indice «Recall» est calculé comme TP/(TP+FN) et répond à la question «Quel pourcentage de la classe i a été correctement classifié ?»

- Le Jackard Index est une mesure standard d'algorithme de classification et est calculé comme TP/(TP+FN+FP). C'est la mesure qui sert pour comparer les meilleurs réseaux de neurones sur la base de données Cityscape.

Le tableau de la page suivante montre ces indicateurs pour chacune des classes, et la moyenne (non pondérée) obtenue.

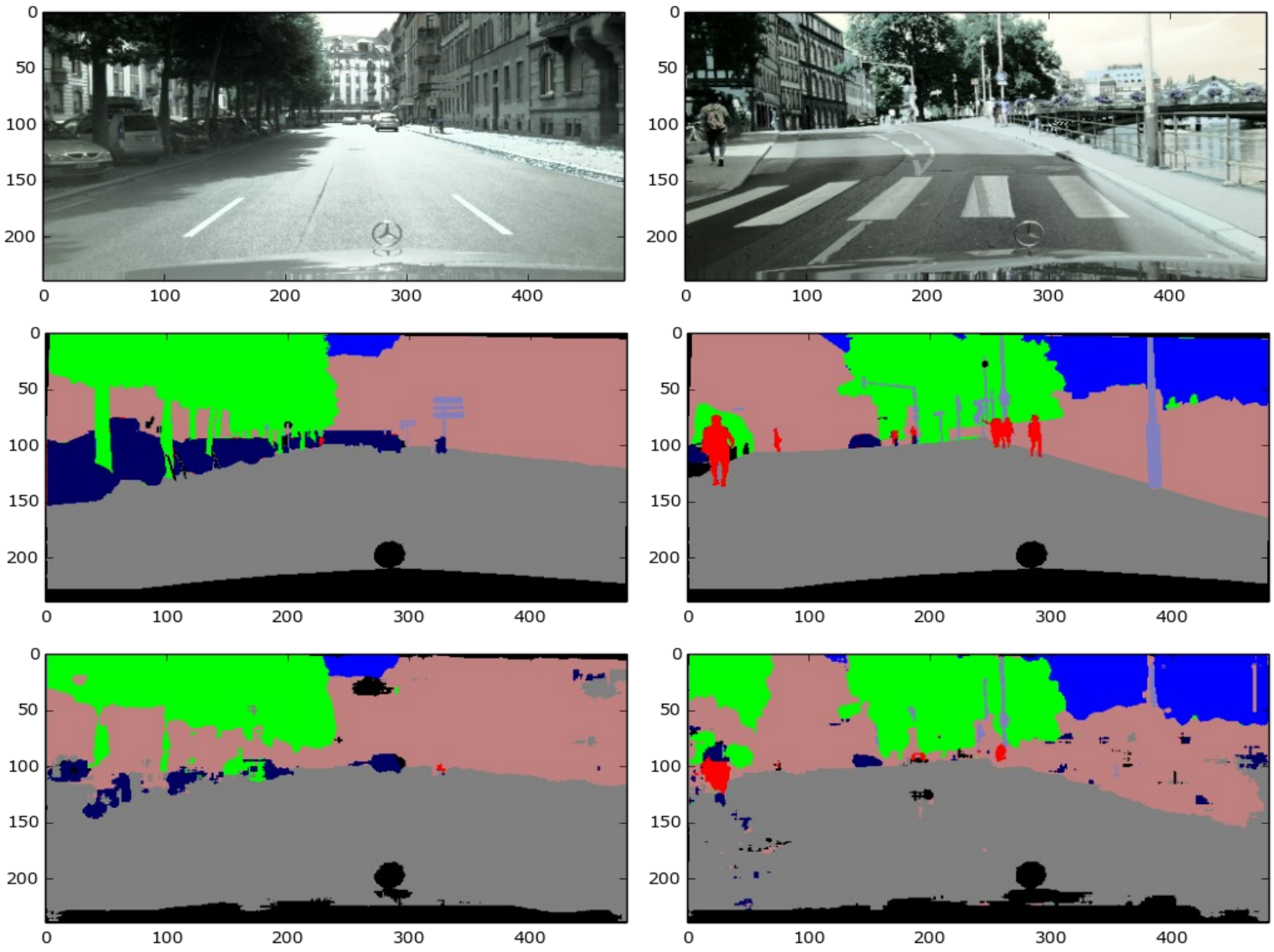
On voit que certaines classes ont été beaucoup mieux apprises que d'autres et ce malgré le rééquilibrage des classes. En particulier, le « recall » de la classe piéton est assez faible ce qui signifie que peu de pixels de piétons sont détectés. En revanche, quand ils sont détectés, il y a une probabilité de 62 % qu'un piéton soit effectivement là.

La classe la moins bien apprise est la classe object. On peut avancer comme hypothèse le manque ou la trop grande diversité d'exemples, ce qui en fait une classe trop compliquée à apprendre.

Classe	Precision	Recall	Jackard Index
Void	0.94	0.64	0.61
Ground	0.87	0.97	0.85
Building	0.78	0.89	0.71
Object	0.55	0.18	0.16
Nature	0.86	0.84	0.74
Sky	0.92	0.87	0.81
Human	0.62	0.27	0.23
Vehicle	0.74	0.61	0.50
Mean	**77%**	**66%**	**54%**

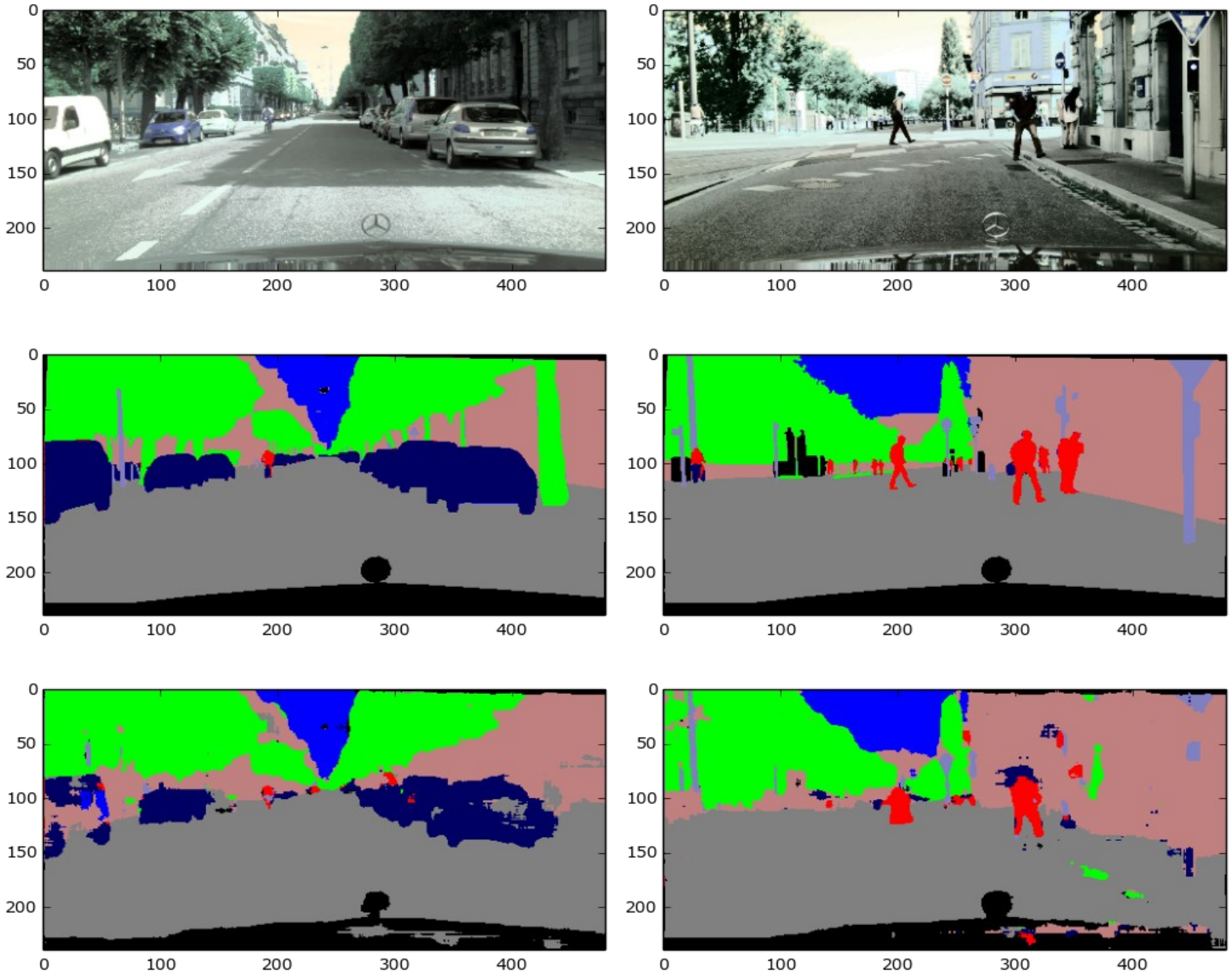
3) Visualisation

La reconstruction des images colorées à partir des labels donne des résultats bien plus cohérents avec l'autoencoder qu'avec les autres réseaux. Ceci est du à une meilleure méthode d'apprentissage et un réseau plus performant.



Les images originales sont normalisées avec l'égalisation YUV (voir remarque A partie VII). On remarque que pour l'image de gauche, la structure de l'image prédite est la même que celle de l'image originale. Beaucoup de pixels de voiture ont été «oubliés» mais il en reste certains pour montrer qu'une voiture a bien été détectée.

Les images suivantes montrent également une restitution fidèle.

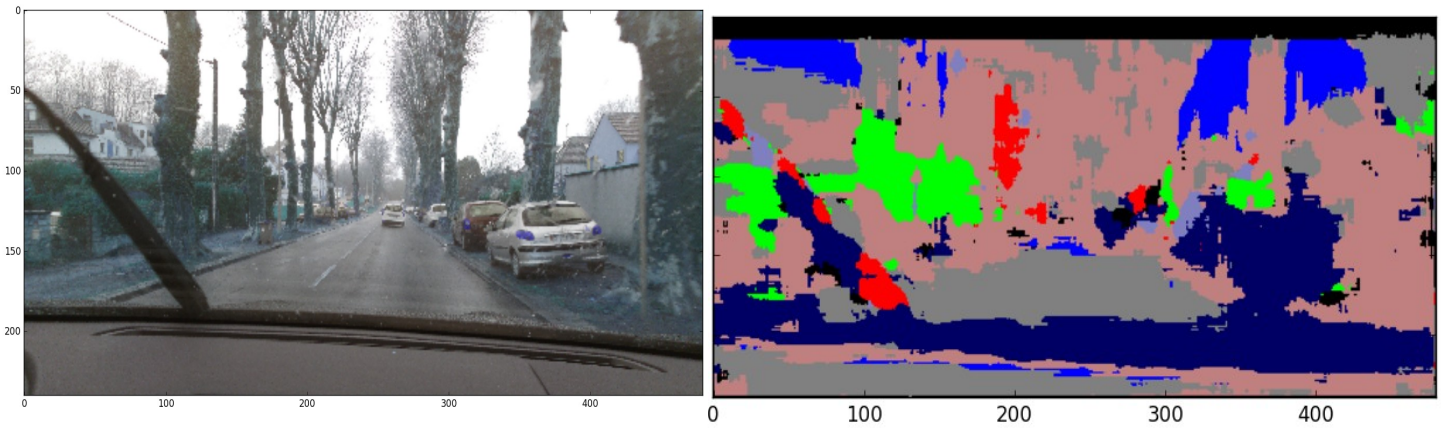


4) Comparaison avec l'état de l'art

Le meilleur réseau entraîné sur Cityscape est ResNet-38, et il obtient un Jackard index de 91%. On note que dans la mesure de cette performance, toutes les villes ont été utilisées pour l'entraînement du ResNet.

5) Test du réseau en situation

J'ai voulu faire tourner l'autoencoder sur une image prise avec mon téléphone. Le résultat n'est pas très concluant, en partie à cause de la mauvaise qualité de l'image (pluie...). On note toutefois que certains éléments sont respectés (ciel, route, voiture sur la route et sur le côté). Une vidéo est en cours de préparation et sera présentée à la soutenance.



En situation réelle, le réseau de neurones doit être capable de palier ce problème de flou de l'image, par exemple en introduisant une couche de bruit au niveau de la couche d'entrée pour rendre le réseau plus robuste.

IV) Remarques diverses

A) Normalisation des images

Nous avons normalisé nos images en faisant une égalisation d'histogramme sur les trois canaux R, G, B. Ceci n'a pas trop n'influence sur des images de taille 64*64, mais ce n'est pas la meilleure méthode. La bonne méthode est de transformer l'image en coordonnées YUV, où Y est un paramètre de luminosité et U,V sont des paramètres de chrominance. Ainsi, on peut simplement réassigner la valeur 0.5 à Y pour égaliser la luminosité des images. Ceci a été fait pour la segmentation par autoencoder ou l'égalisation des composantes RGB pose des problème de contraste.

B) Nombre de classes

Nous avons choisi d'utiliser une segmentation en huit classes; Or les images issues de la base de données Cityscape sont labellisées avec 30 classes. Nous les avons donc regroupées en catégorie. Ceci est bien fondé pour la méthode de découpage d'images car un plus grand nombre de classes rend compliqué l'extraction de cadres 64*64 représentant une classe. Cependant, pour l'autoencoder, augmenter le nombre de classes est susceptible d'améliorer la performance de l'algorithme car les pixels au sein de chaque classe forment des objets qui sont plus proches les uns des autres, rendant la segmentation plus facile. Toutefois, je ne suis pas arrivé à trouver les bons hyperparamètres permettant un apprentissage correct.

C) Contraintes pratiques et retour sur objectif

L'apprentissage d'un réseau de neurones est long et nécessite des paramètres bien ajustés. Au moins trois heures sont nécessaires pour pouvoir exploiter de manière acceptable la puissance d'un réseau, ce qui fait que les calculs ont été lancés d'une séance sur l'autre. De plus, les paramètres d'apprentissage ne sont parfois pas bien ajustés (learning rate trop haut), ce qui fait que le réseau n'apprend rien. Ainsi, il s'est révélé impossible de déterminer l'influence de certains paramètres, comme ceux liés à la structure du réseau, ce qui était notre objectif initial.

D) Pistes d'approfondissement

La segmentation par autoencoder semble être la méthode la plus prometteuse à l'heure actuelle. Pour améliorer les performances obtenues, il faudrait entrainer le réseau sur l'ensemble de la base d'entraînement (20 villes) soit 4 fois plus d'images, ce qui est significatif. En effet, 700 images pour entraîner le réseau est un chiffre assez faible compte-tenu de la difficulté du problème. La deuxième chose à faire est d'augmenter la profondeur du réseau et le nombre de neurones dans ses couches.

D) Déroulement du modal, difficultés rencontrées, impressions générales

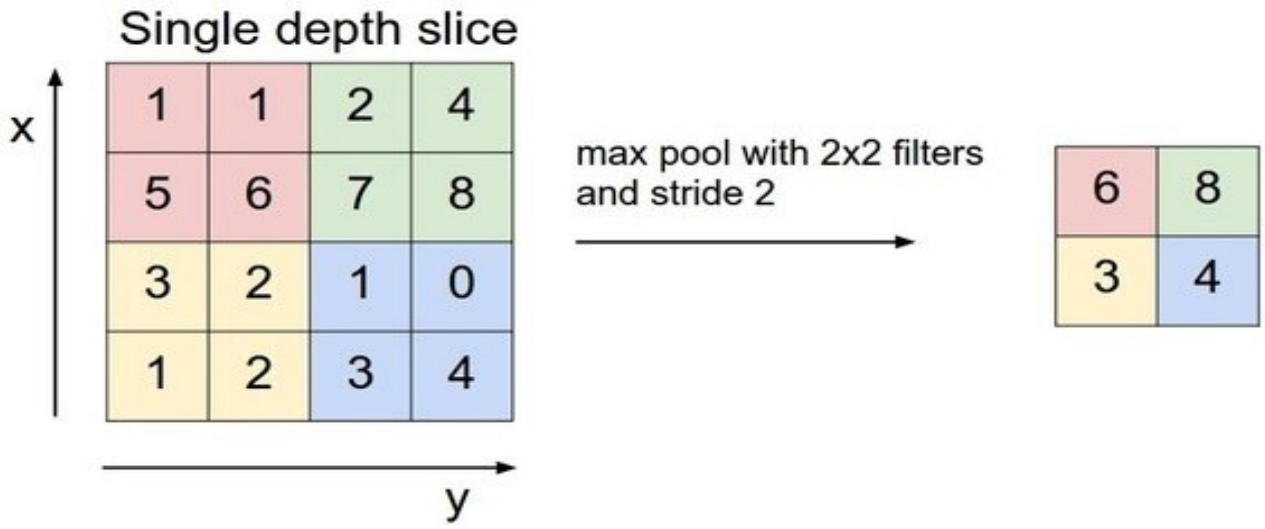
Le choix de ce sujet imposait un traitement informatique assez lourd. Mr Guerry m'a beaucoup aidé dans l'installation de l'environnement de travail, toutefois nous nous sommes heurtés à de nombreuses difficultés. La carte graphique de l'ordinateur n'a pas pu être utilisée. Or ce type de carte permet de faire gagner un facteur 10 en temps dans l'entraînement d'un réseau de neurones. Sans carte graphique à disposition pendant les premières séances, j'ai donc du orienter mes recherches vers une méthode de découpage d'images. Après l'achat et l'installation d'une carte graphique externe, j'ai choisi de continuer dans cette voie et non de réaliser l'autoencoder comme je l'avais prévu la première séance. Or cette méthode de classification s'est avérée assez limitée, et nécessite des ajustements précis des paramètres. J'ai tenté de l'optimiser et suis arrivé à un résultat correct, mais je pense actuellement qu'elle nécessiterait un fort investissement pour l'améliorer notablement. J'ai choisi d'implémenter la segmentation par autoencoder par moi-même après la fin du modal.

Je pense par ailleurs qu l'investissement dans une bonne carte graphique (avec 16GB de mémoire cache au lieu de 2) permettrait à d'autres élèves de réaliser le même type de modal que celui-ci, mais avec des capacités de calculs accrues et donc des résultats plus pertinents.

V) Question théorique: Le MaxPooling

A) Définition

Le MaxPooling est un outil de sous-échantillonage d'une image. Cet outil est paramétré par un filtre rectangulaire de taille (dx,dy). L'image d'origine est découpée en rectangle de taille égale au filtre, puis seule la valeur maximale du rectangle est conservée pour ne former qu'un seul pixel de l'image résultante.
Voici une illustration du MaxPooling avec un filtre de taille (2,2) :



On peut également choisir une découpe de l'image avec des rectangles qui se superposent. Dans ce cas le paramètre stride permet de contrôler l'espacement entre deux rectangles successifs.

B) Intérêt

L'intérêt majeur du Downsampling est de réduire les dimensions des images proches de la couche d'entrée. Cela permet d'éliminer de l'information et de «forcer» le réseau à ne conserver que des quantités pertinentes, réduisant le phénomène d'overfitting selon lequel le réseau tend à simplement «mémoriser» la base de données d'entraînement.

La réduction de dimensions permet de gagner notablement en temps de calcul puisque le nombre de neurones d'une couche est alors divisé par quatre après une couche de Downsampling.

Plus particulièrement, le MaxPooling permet aussi de rendre le réseau moins sensible à la translation spatiale. Si l'on considère un carré 2x2, trois des 8 translations élémentaires (de 1 pixel) donne exactement le même résultat en sortie pour ce carré.

Des mesures empiriques ont montré que cette méthode donnait des meilleurs résultats que le AveragePooling, qui consiste à prendre la moyenne des carrés 2x2 au lieu du maximum, ou bien que le subsampling, qui consiste à brutalement sous-échantillonner l'image. (voir l'article 1 de la bibliographie)

Conclusion

Ce modal a été l'occasion de travailler avec des réseaux de neurones appliqués au traitement d'images. J'ai mis en place un réseau de neurones de type autoencoder et ai considéré comme base d'entraînement la base Cityscape, qui rassemble des images routières issues de nombreuses villes européennes. Ce réseau a été entraîné pendant 4 heures et donne des résultats corrects, qui pourraient toutefois être améliorés. Le temps d'analyse d'une image est de 0.15s sur la machine de la salle de modal et d'environ 2s sur mon ordinateur personnel, ce qui montre la faisabilité d'un outil fonctionnant en temps réel dans la voiture. Pour construire un véritable outil de détection des piétons et voitures, il faudrait un filtre pour les détecter au-delà d'un certain seuil, puis trouver leurs places dans l'image et calculer leurs positions réelles dans l'espace.

Bibliographie

Dominik Scherer, Andreas Müller, and Sven Behnke. 2010. Evaluation of pooling operations in convolutional architectures for object recognition. In *Proceedings of the 20th international conference on Artificial neural networks: Part III* (ICANN'10), Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis (Eds.). Springer-Verlag, Berlin, Heidelberg, 92-101.

Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla, SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation

Jose M. Alvarez, Yann LeCun, Theo Gevers, Antonio M. Lopez, Semantic Road Segmentation via Multi-scale Ensembles of Learned Features

Clément Farabet, Camille Couprie, Laurent Najman, Yann LeCun : Learning Hierarchical Features for Scene Labeling

Jose M. Alvarez, Theo Gevers, Yann LeCun and Antonio M. Lopez : Road Scene Segmentation from a Single Image