Université de Strasbourg & Cerema

Rapport de stage

Master 2 CSMI

Mise en œuvre de techniques *d'apprentissage supervisé* pour l'analyse automatique à grande échelle de marquages routiers

---

Étudiant
Colin HOLLER

Encadrants
Christophe HEINKELÉ
Valérie MUZET

Année universitaire 2021-2022

Table des matières

Remerciements

Je souhaite adresser mes remerciements à Valérie Muzet et Christophe Heinkelé pour m'avoir encadré et soutenu pendant mon *stage*.

Mes remerciements vont aussi à l'ensemble du groupe **ENDSUM** de l'agence du **Cerema** *Strasbourg*, dirigé par Pierre Charbonnier.

Je remercie Nina pour son constant soutien durant ces six mois.

Enfin, je remercie l'ensemble de mes enseignants et tous ceux qui prendront le temps de lire ce rapport.

Merci Dimitri

1 Introduction

Le travail effectué s'inscrit dans le cadre du projet **SAM** (**S**écurité, **A**cceptabilité et **M**obilité autonome) financé par l'**ADEME** [1] qui évalue les performances et besoins des véhicules automatisés. Le maintien de l'infrastructure routière, comme la signalisation horizontale, prend une importance toute particulière à travers ce projet, et ceci nécessite que l'on puisse évaluer cette dernière de façon quantitative et qualitative.

Mon *stage* traite cette problématique et a pour objet le développement de solutions qui permettront une analyse automatisée et un diagnostic de la signalisation routière horizontale. En effet, le **Cerema** (**C**entre d'**E**tudes et d'expertise sur les **R**isques, l'**E**nvironnement, la **M**obilité et l'**A**ménagement) dispose d'un appareil d'évaluation des caractéristiques du marquage : l'**Ecodyn**, un rétro réflectomètre permettant de mesurer la capacité à renvoyer de la lumière émise par les phares du véhicule vers le conducteur.

Les objectifs de ce stage sont, dans un premier temps, de permettre l'utilisation de méthodes de ***deep learning*** permettant d'identifier les marquages routiers dans un logiciel d'annotation **Vérité Terrain Ecodyn**. Ensuite, dans un deuxième temps, de pousser plus loin la recherche avec une partie exploratoire traitant de la sémantique des marquages routiers.

1.1 Contexte général

Le *stage de fin d'études* est une application pratique des connaissances acquises durant les deux années de master **CSMI** (**C**alcul **S**cientifique et **M**athématiques de l'**I**nformation) [2] à l'UFR de Mathématiques et d'Informatique à l'Université de Strasbourg. J'ai effectué mon *stage* dans le laboratoire de recherche **Cerema** à *Strasbourg*. Le sujet est encadré par **Valérie MUZET** et **Christophe HEINKELÉ**, chercheurs au sein de l'équipe **ENDSUM**.

1.2 Le Cerema et l'agence de Strasbourg

Le **Cerema** [3] est un établissement public à caractère administratif (EPA), créé en 2014, sous la tutelle des ministères de la transition écologique et solidaire et de la cohésion des territoires. Il regroupe 3 directions techniques : la dTec**ITM** [i], la dTec**TV** [ii] et la dTec**REM** [iii], 8 directions territoriales qui ont 17 agences. C'est au travers de 2300 agents répartis sur 23 sites territoriaux différents, et de 9 équipes de recherches, que le **Cerema** cherche à relever le défi du développement durable des territoires, au niveau national, mais aussi au niveau européen, notamment en participant aujourd'hui à plus de 40 projets européens.

i. **I**nfrastructure **T**ransport et **M**atériaux
ii. **T**erritoire et **V**ille
iii. **R**isque, **E**au et **M**er

Les missions du **Cerema**, qui apporte un appui notamment aux collectivités territoriales, portent sur les thématiques de l'aménagement et du développement durable. Plus précisément, l'établissement se donne 6 champs d'actions, à savoir l'environnement et les risques, la mer et le littoral, l'expertise et l'ingénierie territoriale, le bâtiment, les mobilités et les infrastructures de transport. Il s'agit donc d'un groupement de compétences pluridisciplinaires, porté à la fois sur l'expertise technique, la recherche et l'innovation.

FIGURE 1 – Logo **Cerema**.
Source: Site du Cerema [3]

L'agence de Strasbourg

Le laboratoire de *Strasbourg* fait partie de la direction Est du **Cerema** avec deux autres sites : Nancy et Metz. Le laboratoire effectue des missions d'essais, de contrôle, de normalisation, d'expertise et/ou de recherche et d'innovation à travers ses cinq groupes d'activités :

- Groupe UMRAE [iv]
- Groupe Bâtiment, Construction, Immobilier
- Groupe Voies et Plates-Formes d'Infrastructures
- Groupe Ouvrage d'Art
- Groupe ENDSUM [v]

J'ai effectué mon *stage* au sein du groupe **ENDSUM** [4] qui est composé d'une dizaine de personnes, dont des chercheurs, des techniciens, un métrologue ainsi que des vacataires. La recherche menée au sein d'**ENDSUM** est appliquée à la conception, la mise en œuvre et l'évaluation des outils d'auscultation d'infrastructures à partir de données de signaux électro-magnétiques ou d'images (visibles, radar, infrarouge) et au moyen de méthodes de reconstruction 3D ou de reconnaissance de formes et d'apprentissage. Cette recherche conduit au développement d'outils ou logiciels de traitements opérationnels. Parmi ceux-ci, on peut citer **Coluroute**, qui vise à vérifier que l'éclairage des routes soit conforme aux normes imposées, les outils d'imagerie IRCAN [vi] et IREVE [vii] pour l'imagerie de scènes routières ou encore l'appareil qui sera central pour mon *stage* : **Ecodyn**, qui mesure la qualité de la signalisation routière horizontale (réflectance des marquages au sol). Mon *stage* s'inscrit dans cet axe de développement d'outils opérationnels, à l'interface des activités de recherche/innovation et des activités opérationnelles du groupe.

iv. **U**nité **M**ixte de **R**echerche de l'**A**coustique de l'**E**nvironnement
v. **É**valuation **N**on **D**estructrice des **S**tructures et **M**atériaux
vi. Imagerie Routière par Caméras Numériques
vii. Imagerie Routière, Étalonnages, Visualisations, Exploitations

1.3 Contexte et objectifs du stage

1.3.1 Conventions utilisées pour le marquage routier

L'infrastructure routière est composée de deux moyens de signalisation : la signalisation **verticale** et la signalisation **horizontale**. La signalisation **verticale** regroupe l'ensemble des panneaux (y compris informatifs) et balises. La signalisation **horizontale** fait référence à l'ensemble des marquages au sol : marques longitudinales (lignes continues, ligne discontinue de guidage ou d'avertissement, etc.) et transversales (ligne de céder le passage, bandes aux feux de signalisation, etc.). Cette signalisation existe afin de donner des informations aux conducteurs dans le but d'assurer leur sécurité en explicitant les règles associées à une portion de route et en leur donnant des informations contextuelles. De plus, en France, plusieurs types de marquages sont mis en place en conformité avec **l'IISR** [viii] **[5]**. Ils ont une signification spécifique qui dépend de leur emplacement (**Axe** ou **Rive**), du type de la route (autoroute, route départementale, etc.) ainsi que de leur géométrie. La Figure **2** présente quelques types de marquages rectangulaires et leurs caractéristiques géométriques.

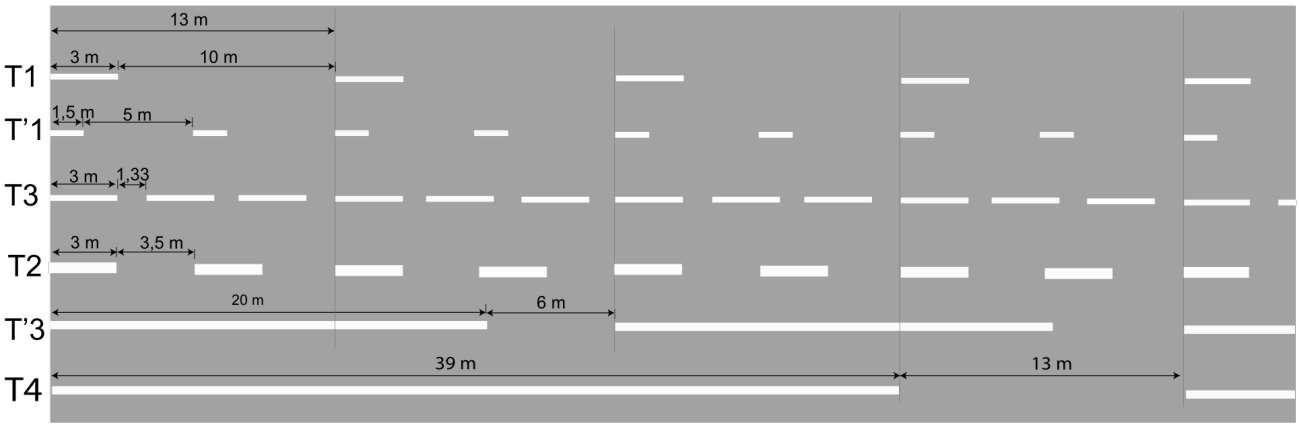


FIGURE 2 – Visualisation de différents types de marquages rectangulaire.

Source: IISR **[5]**

En outre, tous les produits de marquage routier français utilisés sur les voies ouvertes à la circulation sont certifiés par l'**ASCQUER** [ix] **[6]**. Cette certification est le résultat d'essais conventionnels décrits dans la norme **NF EN 1436** qui permet de comparer les produits entre eux. Ainsi, sur les routes inter-urbaines (départementales, nationales et autoroutes), les marquages routiers sont constitués d'une peinture blanche enrichie en **micro-billes réfléchissantes** garantissant une visibilité homogène et constante. En l'absence de ces **micro-billes**, le marquage sera visible de jour, mais ne sera pas (ou difficilement) visible la nuit **[7]**.

1.3.2 Approche par intelligence artificielle

Cette caractérisation du marquage a été faite pour l'œil humain et n'est pas forcément pertinente pour les méthodes d'**intelligence artificielle** **[8]**. Le maintien de l'infrastructure routière prend donc une importance toute particulière à travers cette nouvelle approche, et ceci nécessite que l'on puisse évaluer cette dernière de façon quantitative et qualitative. Ces méthodes sont aussi utiles dans le cadre

viii. **I**nstruction **I**nterministérielle sur la **S**ignalisation **R**outière
ix. **AS**sociation pour la **C**ertification et la **QU**alification des **É**quipements de la **R**oute

du projet **SAM** (**S**écurité, **A**cceptabilité et **M**obilité autonome), projet financé par l'**ADEME** [1] qui évalue les performances et besoins des véhicules automatisés.
Mon *stage* s'inscrit dans cette problématique et a pour objet le développement de solutions qui permettront une analyse automatisée et un diagnostic de la signalisation routière **horizontale**. En effet, le **Cerema** dispose d'un appareil d'évaluation des caractéristiques du marquage : l'**Ecodyn**, un rétro réflectomètre permettant de mesurer la capacité à renvoyer de la lumière émise par les phares du véhicule vers le conducteur. Les exploitations classiques sont globales au pas de 100 mètres. L'idée est de développer des analyses plus fines et robustes avec le développement d'outil de **segmentation** et d'**identification** de type de marquage. En effet, l'équipe **ENDSUM** a exploré des méthodes de **segmentation** et d'**analyse automatique** de marquages routiers par apprentissage. L'obtention de modèles issus de l'apprentissage de réseaux de ***deep learning*** [x] a montré des premiers résultats prometteurs qu'il est nécessaire de consolider, d'évaluer et de faire évoluer du stade de code de recherche à un niveau opérationnel, en tenant compte des besoins de l'utilisateur.

Ainsi, mon premier objectif vise à **documenter**, à **valider** et à **industrialiser** ces **méthodes** existantes qui ont été proposées pour traiter des données issues d'un appareil de mesure dynamique de la **rétroréflexion** du marquage routier. Ensuite, une fois la **segmentation** et l'**identification** réalisée, une analyse au sens de la sémantique sera réalisée à l'aide d'autres méthodes de ***deep learning***, afin de repérer les marquages en mauvais état. Que ce soit la **segmentation** ou l'**analyse sémantique**, ces tâches résultent de tâches de **classification**. Pour ce faire, une **solution logicielle** qui fera l'**interface opérationnelle** entre les besoins des utilisateurs et le traitement automatisé des mesures sera développée. De plus, une recherche bibliographique de chaque modèle ***deep learning*** est à faire. La suite de ce document détaillera les solutions imaginées et les outils mis en place dans le but d'exploiter les mesures acquises avec un rétro réflectomètre mobile : l'**Ecodyn**.

1.4 Organisation et encadrement

Un échange régulier avec mes encadrants a été fait afin de prendre des décisions sur les prochaines tâches à effectuer ainsi que de suivre ma progression. Ces mises aux points m'ont permis de m'orienter vers la bibliographie à lire, de comprendre le choix et l'utilisation des modèles de ***deep learning*** ainsi que de savoir quelles expériences sont à faire. D'autres membres du groupe ont également contribué à mon avancée avec des propositions d'améliorations sur le logiciel développé.

De plus, deux séminaires de stagiaires se sont tenus avec tous les membres du groupe. Ces exposés ont permis de présenter les avancées et de discuter sur les sujets de chacun.

La ***roadmap*** est donnée en **Annexe** (cf. Figure 43).

Durant tout le *stage*, j'ai principalement codé dans le langage `Python` [9] et quelques requêtes ont été faites dans le langage `SQL`. La programmation a été faite sur l'environnement de développement **Spyder** dans un environnement **Anaconda** [10]. Les algorithmes d'apprentissage profond ont été mis en œuvre avec les librairies `Keras (2.6.0)` et `TensorFlow (2.6.0)` [11].

Enfin, les expériences ont été réalisées sur une station de travail avec deux processeurs graphiques (**GPU**) `NVIDIA Quadro RTX 5000` [12].

Dans ce rapport, tout d'abord la caractérisation des marquages avec l'appareil **Ecodyn** sera décrit en présentant le fonctionnement de l'**Ecodyn** ainsi que le principe physique mesuré. Un état de l'art

x. Je fais le choix tout au long de ce rapport d'adopter les termes anglophones, que j'expliciterai, car la littérature scientifique est presque exclusivement partagée dans cette langue, et qu'ils rentrent dans le vocabulaire technique même en français.

sur le ***deep learning*** et sur les méthodes utilisées sera donné. Enfin, la mise en œuvre des modèles utilisés sera détaillée et les résultats obtenus seront décrits.

2 Caractérisation des marquages avec Ecodyn

L'**Ecodyn** a été conçu pour répondre à un besoin évoqué par les gestionnaires routiers, celui de pouvoir quantifier la qualité des marquages routiers selon plusieurs critères. En effet, les caractéristiques des marquages routiers sont standardisées conformément à la norme **NF EN 1436** [13]. Cette norme décrit les caractéristiques essentielles des marquages routiers que sont le contraste, la chromaticité, la **rétroréflexion** ou encore l'adhérence. En outre, l'appareil **Ecodyn** mesure la **rétroréflexion** des marquages routiers qui est l'un des critères de la norme.

2.1 Principe physique mesuré

La **rétroréflexion** des marquages routiers est un phénomène physique optique permettant la garantie d'une visibilité du marquage la nuit. Elle désigne la capacité d'un objet à renvoyer la lumière émise dans la même direction. Ainsi, dans notre cas, en termes de sécurité routière, le but ici est de renvoyer la lumière des feux du véhicule vers le conducteur pour que celui-ci ait une visibilité optimale de la route. Elle est notée R_L en $mcd.m^{-2}.lx^{-1}$ et peut se définir de la façon suivante :

$$R_L = \frac{L}{E}$$

où L est la luminance reçue en $mcd.m^{-2}$ et E l'éclairage émis par les phares du véhicule en lx. De plus, une bonne **rétroréflexion** des marquages routiers provient de la présence de **micro-billes** de verre d'environ $1mm$ de diamètres (cf. Figure 3) qui sont capables de réfléchir un faisceau de lumière incidente dans la même direction.

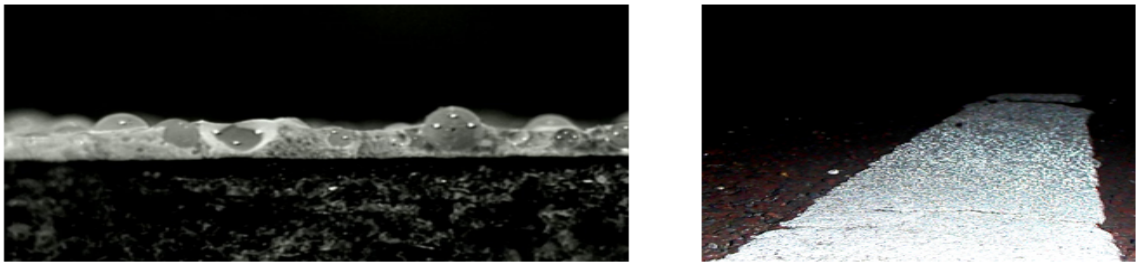


FIGURE 3 – Visualisation des **micro-billes** dans le marquage.

Source: [14]

Cela permet donc aux marquages de briller quand ils sont éclairés. Ces mesures de **rétroréflexion** doivent être faites selon des angles précis, correspondant à une mise en situation dans une géométrie conventionnelle. La norme spécifie les angles de mesures adaptés à la mesure de la **rétroréflexion** à 30 mètres :

- Angle d'émission $\varepsilon = 1,24° \pm 0.05°$
- Angle de réception $\alpha = 2,29° \pm 0.05°$

Ces angles simulent la vision à 30.0 mètres du marquage par un conducteur situé à 1.20 mètres du sol recevant la lumière rétroréfléchie des phares de son véhicule à une hauteur de 0.65 mètres du sol (cf. Figure 5).

2.2 Principe de fonctionnement et caractéristiques techniques d'Ecodyn

L'**Ecodyn** est un **rétroréflectomètre**, appareil permettant de mesurer le coefficient de **luminance rétroréfléchi** en dynamique et en plein jour d'un marquage routier. Il existe différentes versions de l'**Ecodyn** et dans le cadre de mon *stage* les mesures proviennent de l'**Ecodyn3**. Ainsi le fonctionnement et les caractéristiques techniques seront décrites pour cette version.

Puisque l'**Ecodyn** est positionné sur le châssis du véhicule (cf. Figure 4), les mesures ne se font pas à 30mètres, mais à 6mètres afin de respecter les angles imposés. Ainsi, cette géométrie réduite (cf. Figure 5) simule une visibilité du marquage à 30mètres et correspond à la vision perçue par le conducteur.

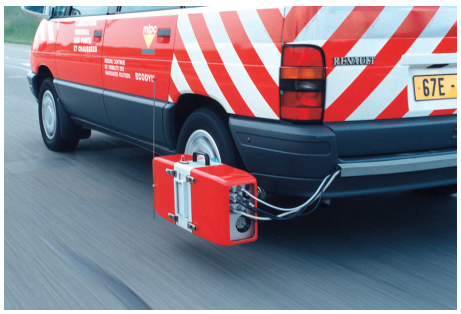
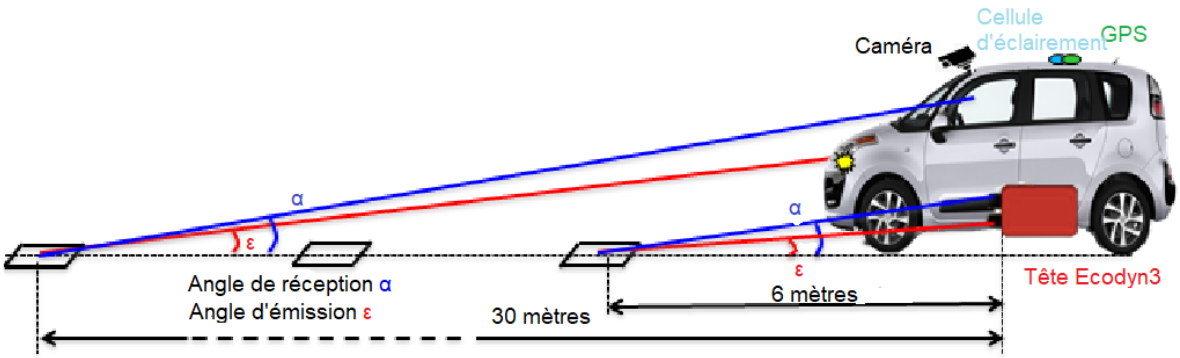


FIGURE 4 – Photo de l'**Ecodyn**.
Source: [14]



Source: [14]

FIGURE 5 – Visualisation de la géométrie réduite.

L'appareil effectue ses mesures en continu quelle que soit la vitesse du trafic (de 0 à $130 km.h^{-1}$) au moyen d'une ou deux têtes, qui correspondent respectivement aux côtés **Axe** et/ou **Rive** de la route. Une acquisition correspond à la réalisation d'une série de mesures localisées et géo-référencées tous les $40cm$. Le principe consiste ainsi à émettre des trains d'impulsions lumineuses à la demande, à l'aide de LEDs, sous forme de créneaux. La lumière rétro réfléchie est, ici aussi, reçue par des photos détecteurs, puis numérisée. Chaque ligne de mesure donne la **rétroréflexion** d'une zone de $90cm$ de large et $40cm$

de haut. Cette zone est découpée en 32 canaux ou voies de mesure d'environ $3cm \times 50cm$ chacune dont 9 sont représentées sur la Figure 6. Ainsi, une voie de mesure peut être située en partie sur le marquage et en partie sur la chaussée. Dans l'exemple de la Figure 6, seules 5 voies de mesures sont entièrement situées sur le marquage et donc valides et exploitables, d'où l'utilité d'un champ de mesure large. Afin d'aider le conducteur à bien se positionner par rapport aux marquages routiers, un bargraphe à LED est disposé dans le véhicule de mesure. Il représente la mesure brute de chaque voie, ce qui permet de visualiser la bande de marquage par rapport aux 32 voies. L'éclairage ambiant en *lux* est mesuré avec un luxmètre situé sur le toit du véhicule.

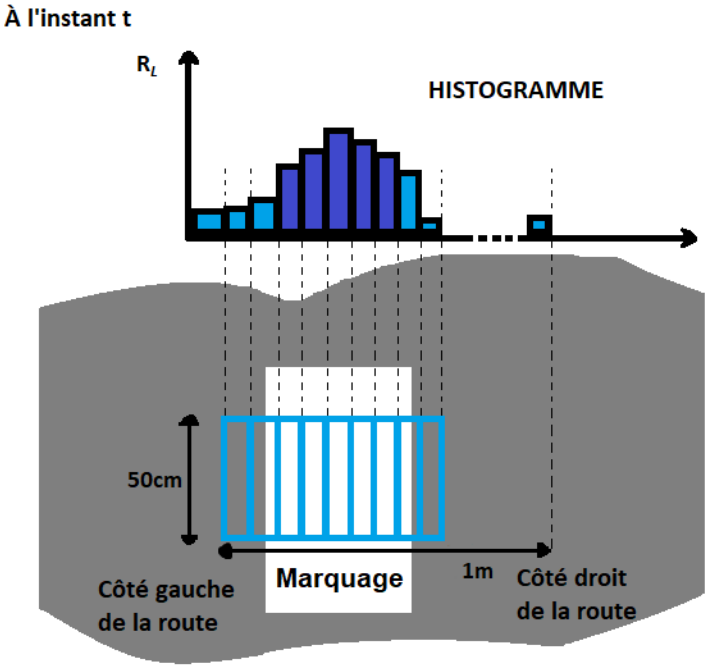


FIGURE 6 – Histogramme représentant les valeurs brutes mesurées à l'instant t d'un marquage dont seules les valeurs violettes sont stockées.

Source: [14]

La version récente **Ecodyn3** permet de mesurer la luminance en $cd.m^{-2}$. Le signal correspondant, dit de **jour** a un angle de réception identique à celui de R_L, soit $2.29° \pm 0.05°$. Ainsi, comme l'éclairage à LED de l'**Ecodyn3** est modulé, en roulant de jour, il est possible de mesurer un signal dit de **«jour»** lorsque les LEDs sont éteintes (proportionnelle à la **luminance**) et le signal de **rétroréflexion** dû aux LEDs [15].

Une fois les données acquises dans le logiciel d'acquisition, des coefficients d'étalonnage sont appliqués aux données brutes afin d'obtenir les valeurs de **rétroréflexion** et de **luminance** correspondante. Une transformation afin de représenter les données sous forme d'images est ensuite effectuée (cf. Brevet [16]). On obtient alors deux images en niveau de gris, chacune de largeur 32 pixels et de longueur plusieurs milliers de pixels. Un fichier `.db` est obtenu comportant entre-autres les données des deux signaux, les coordonnées GPS, le côté de la route considéré etc. Sur la Figure 7, nous observons une visualisation des signaux sous forme d'images d'un morceau de route de longueur 120 pixels. Un rectangle d'une telle longueur sera appelé **tronçon**.

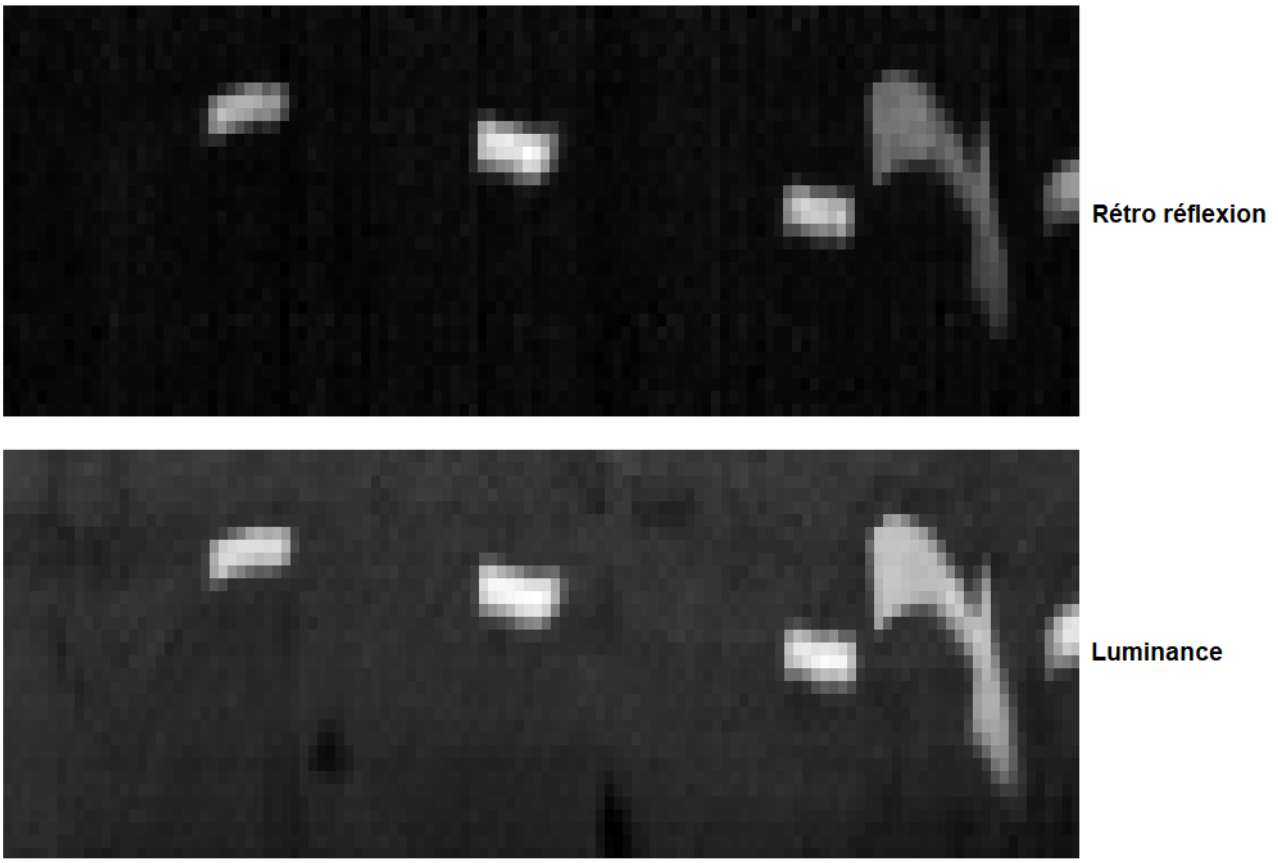


FIGURE 7 – Visualisation des signaux de **rétroréflexion** et de **luminance**.

L'étape suivante consiste à **segmenter**, c'est-à-dire associer à chaque pixel la classe pour laquelle il est représenté. En premier lieu, une **classification** bi-classes sera faite avec la classe *marquage* en présence de marquage et **arrière-plan** pour le reste. Pour ce faire, une annotation de ces données (cf. Figure 7) doit être faite pour les apprentissages de modèles de ***deep learning***. Le logiciel d'annotations utilisé **Vérité Terrain Ecodyn** est présenté dans la section suivante.

2.3 Vérité Terrain Ecodyn

2.3.1 Mise en place d'un logiciel d'annotations et de visualisations

Vérité Terrain Ecodyn est un outil développé dans le langage **Python**, à l'aide de la bibliothèque `Tkinter` [17], en 2021 dans le cadre d'un *stage* antérieur [18]. Il permet d'afficher les images de **rétroréflexion** et de **luminance** associées aux signaux respectifs ainsi que de les **annoter** afin de créer une **vérité terrain** conforme à la forme et aux types de marquages.

2.3.2 Architecture Modèle-vue-contrôleur

L'application **Vérité Terrain Ecodyn** se base sur une architecture **MVC** pour **Modèle-Vue-Contrôleur** [19]. Le but de cette architecture est d'assurer la **fluidité** de l'application. Le remplissage

est réalisé par tronçon au lieu de charger toutes les données à la fois. Cette démarche simplifie également la compréhension du code et sa modification pour ensuite faire les corrections ou les améliorations nécessaires. Une version schématisée et simplifiée se trouve sur la Figure 8.

Contrôleur : a pour but la **récupération** et le **chargement** des données soit brutes, soit annotées.

Modèle : crée un **dictionnaire** vide (deviendra le fichier `.json`), qui va servir de moyen de stockage des données brutes. Il permet également de récupérer des données contenues dans un dictionnaire déjà existant, équivalent à des données déjà annotées.

Vue : gère l'**interface graphique** et s'appuie sur le modèle pour faire l'affichage.

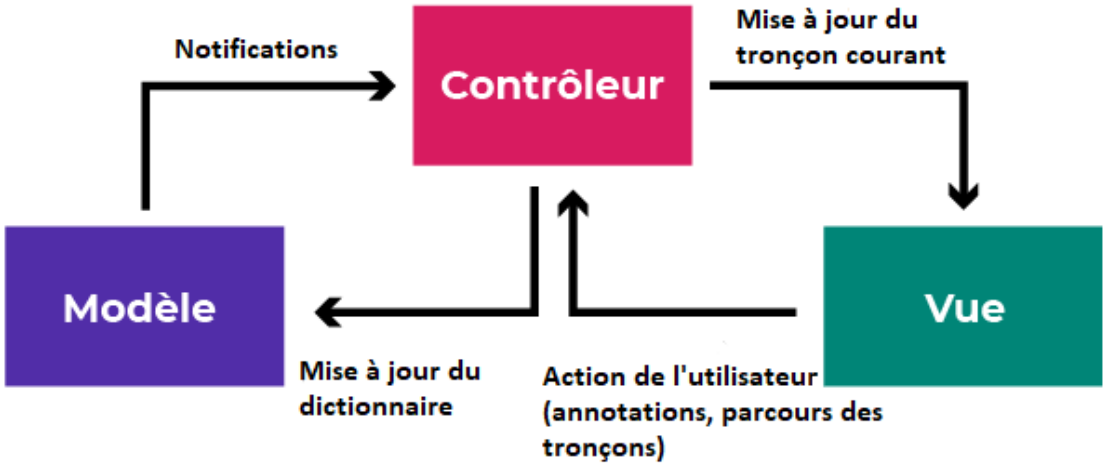


FIGURE 8 – Schéma représentant le modèle **MVC** de **Vérité Terrain Ecodyn**.
Source: [20]

2.3.3 Format des données

Le logiciel de **vérité terrain** prend en entrée les données brutes d'**Ecodyn** de format `.db`. Ce logiciel permet de choisir la route considérée, par exemple la route :

`20000093_D13s2bg38p1_-_PR15_PR0_Axe.sess.db`

Lorsque le script `VT-Ecodyn3.py` est lancé et la route choisie, la fenêtre 9 apparait. Dans le code, l'étalonnage est automatiquement appliqué aux données de la route associée ainsi que la transformation en pixels afin d'avoir une visualisation sous forme d'images. On voit l'apparition de trois rectangles qui correspondent à des tronçons. Les deux premiers correspondent aux signaux de **Luminance** et de **Rétroréflexion** et le dernier à l'annotation faite par l'utilisateur, qui lui a une palette de couleurs à disposition afin d'annoter pixel par pixel le marquage. Sur la Figure 9, la couleur verte est associée au marquage de type **T1**. De plus, à l'aide des flèches, il est possible de se déplacer sur la route tronçon par tronçon.

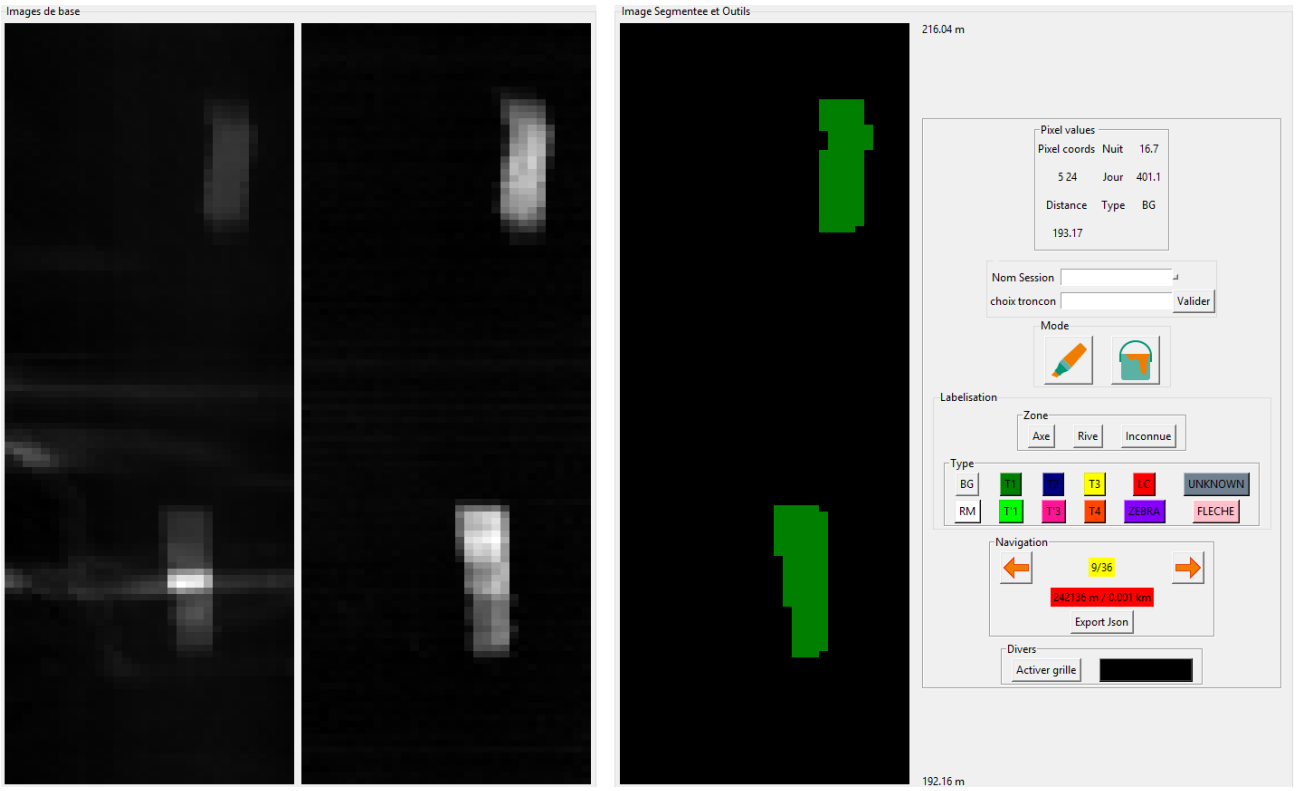


FIGURE 9 – Visualisation de l'interface de **Vérité Terrain Ecodyn**.

L'idée est, qu'une fois les données sont annotées (au moins partiellement), l'annotation est exportée en fichier `.json`, à l'aide d'un bouton, afin d'avoir des données structurées facilement manipulables et d'éviter la redondance de valeurs. Il est ensuite possible de les charger dans le logiciel pour continuer l'annotation ou bien la re-visualiser.

La structure du fichier `.json` se trouve dans l'**Annexe** (cf. Listing 1). La signification de chaque attribut est expliquée ci-après :

File : Nom du fichier **.json**. La route peut être parcouru dans les deux sens, dans le sens des kilométrages ascendants et dans le sens des kilométrages descendants. Dans notre cas, le caractère «-» correspond à un sens descendant (sinon «+»). La localisation de mesure est aussi indiqué (ici **Axe**).

Cote : La localisation de la route, **Axe** ou **Rive**/**BAU** (**B**ande d'**A**rrêt d'**U**rgence).

distance parcourue : Distance totale parcourue par le véhicule en **km**.

Detected : Cet attribut correspond aux marquages détectés sur la route, soit il est vide (correspondant à un 0) soit c'est un dictionnaire contenant des dictionnaires représentant chaque marquage de la route.

Profil : Une ligne de l'image/une acquisition caractérisée par :

- **Distance** en **mm** (tous les 40 ou 20 mètres).
- Le **flux** envoyé par la LED.
- Les données **GPS** caractérisées par la **longitude** et la **latitude**.
- La **voie** prenant comme valeur de 1 à 32 (pour 32 pixels). Une voie correspond à une colonne de l'image et le pixel en position (**Profil**, **voie**) est caractérisée par :

- La valeur du signal de **rétroréflexion** en $\mathbf{mcd.m^{-2}.lx^{-1}}$.
- La valeur du signal de **luminance** en $\mathbf{mcd.m^{-2}}$.
- Le **type** de marquage, **BG**, **T1**, **T1'** etc.
- Le **numéro de composante** qui est le numéro de composante connexe associé au marquage sur toute la route. EN particulier, 0 correspond à l'arrière-plan.

3 État de l'art

Afin de répondre à la problématique de l'étude, il est nécessaire de rappeler quelques notions de base sur l'**apprentissage supervisé** : le cadre général, les systèmes essentiels du ***deep learning*** que sont les réseaux de neurones et les métriques d'évaluations de ces outils. Deux branches du ***deep learning*** seront exposées, la première relative à la **segmentation**, et la deuxième au **traitement de langage naturel**. L'objectif et les méthodes utilisées seront exposés et les modèles utilisés détaillés.

3.1 Apprentissage supervisé

3.1.1 Cadre général

Le domaine du ***machine learning*** est un champ d'étude de l'intelligence artificielle. Il regroupe les méthodes, les algorithmes ainsi que les outils techniques et théoriques qui permettent de mettre en œuvre des approches statistiques pour faire apprendre à une machine, à partir des données, à résoudre des problèmes sans être explicitement programmés. Ces données peuvent être des chiffres, des mots, des images ou même des statistiques [21].

Les algorithmes du ***deep learning*** peuvent être considérés à la fois comme une évolution sophistiquée et mathématiquement complexe des algorithmes de ***machine learning***. Le ***deep learning*** décrit des algorithmes qui analysent des données avec une structure logique similaire à celle dont un humain tirerait des conclusions. Pour y parvenir, les applications d'apprentissage profond utilisent une structure d'algorithmes en couches appelée réseau neuronal artificiel [xi]. La conception d'un tel réseau, dont le premier est le ***perceptron*** [23], s'inspire du réseau neuronal biologique du cerveau humain, ce qui conduit à un processus d'apprentissage bien plus performant que celui des modèles de ***machine learning*** standard [24].

En général, il s'agit donc de fournir à un modèle des données annotées, aussi appelées **vérité terrain**, sur lesquelles le modèle va itérativement produire des prédictions, estimer leur qualité, et optimiser ses paramètres internes pour minimiser l'erreur produite, jusqu'à arriver à converger vers une solution satisfaisante. C'est ce qu'on appelle la phase d'**apprentissage**. Avec un modèle entraîné, on peut ensuite inférer sur de nouvelles données du même type. Un modèle de ***machine learning*** doit donc minimiser une certaine fonction de coût, dont la définition influe grandement sur la qualité du résultat final, pour permettre d'arriver à une certaine exactitude des prévisions par rapport à la **vérité terrain**. Ce type d'entraînement est appelé **supervisé** [25]. En revanche, l'**apprentissage non supervisé** n'utilise pas de données étiquetées. Il est alors impossible à l'algorithme de calculer de façon certaine un score de réussite. Son objectif est donc de déduire les regroupements présents dans les données [25]. Dans le cadre du *stage*, seuls des réseaux neuronaux profonds supervisés seront considérés.

Pour pouvoir converger vers une solution convenable, ces types d'algorithme ont donc besoin de très larges quantités de données pour apprendre les bons paramètres. Il faut aussi noter qu'un risque courant pour un modèle de machine learning est de se sur-spécialiser aux données d'entraînement, phénomène appelé **sur-apprentissage** [26], donnant ainsi de très bons résultats sur ces données, mais des résultats mauvais, voire médiocres sur de nouvelles données. Ainsi, pour éviter d'évaluer la qualité du modèle final à l'aide des mêmes données qui ont servi pour l'entraînement, il faut séparer dès le départ le jeu de données en deux parties distinctes **sur-apprentissage** [27] :

xi. Le terme profond est utilisé lorsqu'on utilise 2 couches ou plus [22]

- Une base d'**entraînement** qui sert à entraîner le modèle, c'est-à-dire mettre à jour les poids et les biais.
- Une base **test** qui permet de mesurer l'erreur du modèle **final** sur des données qu'il n'a jamais vues.

Pour l'**entraînement**, la base d'**entraînement** est généralement coupée en deux afin d'avoir une proportion d'environ 80%/20% de façon disjointe. La petite base, appelé base de **validation sur-apprentissage** [27], est utilisée pour fournir une évaluation non biaisée de l'ajustement d'un modèle sur l'ensemble de données d'apprentissage tout en réglant les **hyper-paramètres** du modèle. Les **hyper-paramètres** [28] correspondent aux variables qui déterminent la structure du réseau (par exemple, le nombre de neurones) et les variables qui déterminent la façon dont le réseau est entraîné (par exemple, l'algorithme d'optimisation). Lorsque les performances du modèle diffèrent significativement entre données d'**entraînement** et de **validation**, il y a **sur-apprentissage** [26]. Il faut alors jouer sur les **hyper-paramètres** (paramètres d'optimisation, structure du modèle, etc.) ou sur la base d'apprentissage (augmenter/diversifier les données) pour y remédier.

L'apprentissage se fait en général sur une représentation simplifiée des données sous forme de **caractéristiques**, aussi appelées ***features*** [22]. Ces caractéristiques sont des valeurs numériques présentées dans un vecteur à l'entrée d'un algorithme d'**apprentissage**. Dans le cas d'une image, elles sont des marqueurs, plus ou moins complexes, associés à des ensembles des pixels de celle-ci (sur des différences de contraste entre deux régions de pixels par exemple) que le modèle utilise pour décrire son contenu. On différencie deux façons de construire ce vecteur de **caractéristiques**. La première génération d'algorithmes de ***machine learning*** consistait à utiliser des **caractéristiques** définies «à la main» par un expert pour entraîner un modèle qui doit apprendre comment exploiter au mieux ces **caractéristiques** sur les données : on parle de ***feature engineering*** [29]. Dans les méthodes d'**apprentissage profond**, on utilise directement les données de base pour faire apprendre à un modèle son propre jeu de **caractéristiques** : c'est ce qu'on appelle le ***feature learning*** [30], qui est au centre des méthodes utilisées dans le cadre de l'étude.

L'**apprentissage incrémentiel** [31] fait partie de la catégorie des méthodes d'**apprentiss-age supervisé**. L'objectif de cette stratégie est que l'annotation des données utilisées pour l'apprentissage ne soit pas entièrement effectuée par un humain, mais que ce dernier n'ait qu'à confirmer une **classification** correcte ou à la modifier si elle ne l'est pas. L'idée est d'inférer sur des nouvelles données grâce au réseau que l'on a entraînée, d'en faire corriger les erreurs par un humain puis d'intégrer les données annotées à la base d'apprentissage afin de l'étayer.

3.1.2 Réseaux neuronaux

Les **réseaux neuronaux constituent** un sous-ensemble de l'apprentissage automatique et sont au cœur des algorithmes d'**apprentissage profond**. Leur nom et leur structure sont inspirés du cerveau humain, imitant la façon dont les neurones biologiques se signalent les uns aux autres [23]. Les réseaux de neurones artificiels sont constitués de couches de nœuds/neurones, contenant une couche d'entrée, une ou plusieurs couches cachées et une couche de sortie, montré sur la Figure 10 [xii]. À chaque connections d'un neurone à un autre, un **poids** est associé. Ces **poids** permettent de déterminer l'importance d'une variable donnée, les **poids** les plus importants contribuant de façon plus significative à la sortie par rapport aux autres entrées. Toutes les entrées d'une couche sont ensuite multipliées par leurs **poids** respectifs, puis additionnées. Une matrice de **poids**, noté W dans $\mathcal{M}_{N\times N_n}$, peut être définie où N est la taille de l'entrée et N_n le nombre de neurones dans la couche. Un **biais** b dans $\mathbf{R}^{N_n}$ peut aussi

xii. Le réseau montré est aussi appelé, réseau ***fully-connected*** puisque tous les neurones sont connectés entre eux

s'additionner dans la somme qui lui correspond à une translation [22]. À chaque connexion de couches, une matrice de **poids** et un vecteur de **biais** peuvent être définis. Par la suite, la sortie d'une couche est transmise via une fonction d'**activation** qui détermine la sortie [xiii]. Si cette sortie dépasse le seuil donné, elle «déclenche» (ou active) le nœud, et transmet les données à la couche suivante du réseau [22]. Une telle couche est appelée **couche dense** et sans activation, elle est appelée **couche linéaire**. La sortie d'un nœud devient alors l'entrée du nœud suivant. Ce processus, qui consiste à transmettre les données d'une couche à la couche suivante, définit ce réseau neuronal comme un réseau à **propagation avant**, aussi appelé ***feedforward*** [32]. Ces réseaux sont souvent utilisés pour des problèmes de **classification** ou de **régression**.

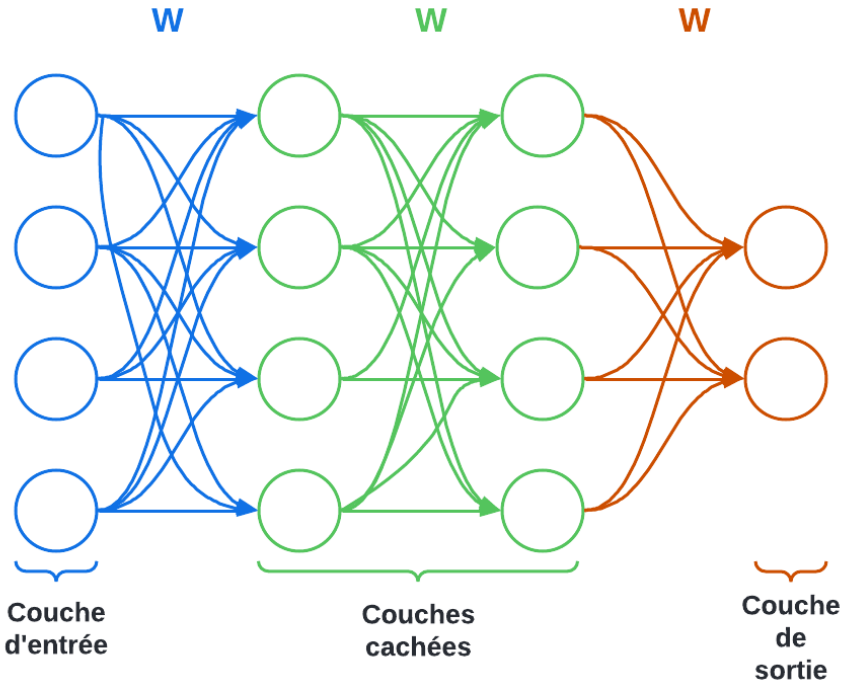


FIGURE 10 – Schéma d'un réseau de neurones à deux couches cachées.

L'apprentissage consiste alors à mettre à jour les poids et biais de chaque neurone à l'aide d'une **fonction de coût**, aussi appelée ***Loss function***, qui doit être différentiable et dépend de la sortie du réseau de neurones et de la **vérité terrain**. L'algorithme de descente de gradient est utilisé et ajuste ces paramètres pour minimiser la **fonction de coût** [32]. Ces paramètres sont modifiés itérativement à l'aide d'un algorithme d'optimisation, dont le plus connu et le plus utilisé est l'optimiseur ***Adam*** [32], afin d'atteindre le point de convergence. Cet algorithme nécessite de calculer la variation du coût à la sortie du réseau relativement à la variation des paramètres du réseau. Dans un réseau de neurones, le coût se calcule par composition d'opérateurs linéaires ou de non-linéarités ponctuelles différentiables. La différentiation d'une composition d'opérateurs s'obtient en appliquant la règle de Leibniz. Cet algorithme est aussi appelé, algorithme de **rétropropagation du gradient** [22]. La rétropropagation permet donc de calculer et d'attribuer l'erreur associée à chaque neurone.

L'apprentissage peut se faire par **batchs** où un **batch** correspond à un sous ensemble des données

xiii. C'est des fonctions non linéaires telles que tanh ou la sigmoïde qui permettent d'avoir une nouvelle approche sur les données

de départ, définit par une **taille de batch**. C'est le nombre d'exemples qui sont fournis à l'algorithme avant que les poids/paramètres du modèle soient recalculés à chaque itération de la rétropropagation. Cela a tendance à faciliter l'apprentissage, en particulier la descente du gradient [32]. De plus, lorsque le réseau a vu toutes les données, on passe à une seconde étape. Ces étapes sont appelées des **epochs**, et le nombre d'**epochs** est défini par l'utilisateur.

Le nombre de couches successives, le nombre de neurones par couches, la manière de connecter entre eux les neurones entre chaque couche, etc. sont autant d'**hyper-paramètres** à choisir en fonction du problème spécifique à résoudre. Plus il y a de couches, plus il y a de paramètres (poids et biais) à mettre à jour, plus long sera le temps d'entraînement/d'évaluation et plus il y a apparition de phénomènes de **sur-apprentissage**. La réduction de paramètres permet de remédier à cette manifestation, mais il existe d'autres méthodes [xiv] comme par exemple le **Dropout**, faisant référence à la suppression de neurones [33]. Le **Dropout** permet de désactiver temporairement certains neurones dans le réseau, ainsi que toutes ses connexions entrantes et sortantes. À chaque **epoch**, une désactivation aléatoire, de probabilité définie par l'utilisateur, est appliquée. C'est-à-dire qu'à chaque propagation vers l'avant, le modèle apprendra avec une configuration de neurones différente. Ceci permet de perturber les caractéristiques apprises par le modèle et ainsi réduire le **sur-apprentissage** [33]. La **Batch Normalization** est aussi une technique permettant de réduire ce phénomène [34] [35]. C'est une couche qui consiste à standardiser les données de façon normale, c'est-à-dire de soustraire la moyenne des données aux données, puis de diviser par l'écart type des données. Ceci se fait par **batch** et toutes les données d'un **batch** partagent une même échelle de valeur. Les données du **batch** sont alors vues comme ayant un lien entre elles. Cela permet au modèle d'avoir une vision d'ensemble du problème à résoudre et des données à traiter. Ainsi, cela permet au modèle de mieux généraliser et donc de réduire le **sur-apprentissage**. Aujourd'hui il est souvent plus utilisé que le **Dropout** [34].

3.2 Mesures et évaluation de performance

Pour évaluer les performances d'un outil de **classification**, introduisons dans un premier temps différentes métriques d'évaluation classiquement utilisées.

L'évaluation consiste à comparer les résultats du modèle, appelés prédictions, à la **vérité terrain**. Cette évaluation se fait sur la **base test**. Pour la **segmentation**, la **vérité terrain** est construite en annotant à la main chaque donnée, dans notre cas via le logiciel **Vérité Terrain Ecodyn**. En comparant prédictions et **vérité terrain**, certains indicateurs peuvent être extraits [22] [xv] :

Vrais positifs (TP) : Les prédictions et les valeurs réelles sont effectivement positives. À juste titre, le modèle prédit qu'un pixel est de classe marquage.

Vrais négatif (TN) : Les prédictions et les valeurs réelles sont toutes les deux négatives. À juste titre, le modèle prédit qu'un pixel est associé à l'arrière-plan.

Faux positif (FP) : Une prédiction positive contraire à la valeur réelle qui est négative. Le modèle a prédit injustement que le pixel considéré soit de classe marquage.

Faux négatif (FN) : Une prédiction positive contraire à la valeur réelle qui est négative. Le modèle a prédit injustement que le pixel considéré soit de classe marquage.

À noter que ces indicateurs sont spécifiques à la classe choisie. La **matrice de confusion** est un résumé des résultats de prédiction pour un problème particulier de **classification** [22] [36]. Elle compare les

xiv. Les méthodes qui luttent contre le **sur-apprentissage** sont appelées des méthodes de ***Regularization***

xv. Les exemples donnés sont par rapport à la segmentation bi-classe, *marquage/arrière-plan*

données réelles pour une variable cible à celles prédites par un modèle. Les prédictions justes et fausses sont révélées et réparties par classe, ce qui permet de les comparer avec des valeurs définies.

		Prédiction	
		Positif	Négatif
Vérité	Positif	**TP**	**FN**
	Négatif	**FP**	**TN**

La matrice de confusion ci-dessus est la matrice associée au cas binaire, mais peut se généraliser pour plusieurs classes.

Ces mesures sont des mesures absolues : un grand nombre de bonnes prédictions peut s'avérer relativement faible par rapport au nombre de pixels à trouver. Pour donner une meilleure appréciation de ces résultats, on s'intéresse donc à trois mesures relatives : l'***accuracy***, la **précision** (***Precision***) et le **rappel** (***Recall***) [22] :

Accuracy : indique le pourcentage de bonnes prédictions général au sein de toutes les classes. S'il y a déséquilibrage des classes dans le problème, cette métrique représente mal les performances d'un modèle.

$$\textbf{accuracy} = \frac{\textbf{TP} + \textbf{TN}}{\textbf{TP} + \textbf{FP} + \textbf{FN} + \textbf{TN}} \tag{1}$$

Rappel : calcule le pourcentage de positifs bien prédit par le modèle. Plus il est élevé, plus le modèle **maximise** le nombre de **Vrai Positif**. Quand le **rappel** est haut, cela veut dire qu'il ne ratera aucun positif. Néanmoins, cela ne donne aucune information sur sa qualité de prédiction des négatifs.

$$\textbf{rappel} = \frac{\textbf{TP}}{\textbf{TP} + \textbf{FN}} \tag{2}$$

Precision : permet de connaître le nombre de prédictions positives bien effectuées. Plus elle est élevée, plus le modèle **minimise** le nombre de **Faux Positif**. Quand la précision est haute, cela veut dire que la majorité des prédictions positives du modèle sont des positifs bien prédit.

$$\textbf{precision} = \frac{\textbf{TP}}{\textbf{TP} + \textbf{FP}} \tag{3}$$

La **précision** et le **rappel** que l'on souhaite maximales sont toutefois en contradiction l'une avec l'autre. On peut vouloir privilégier l'un ou l'autre, mais dans notre cas, il est important de maximiser les deux. On s'intéresse alors à la mesure **F1** (***F1 score***) [22], se définissant comme la moyenne harmonique des mesures de **précision** et de **rappel**, et qui donne une appréciation du niveau de la **précision** et du **rappel** tout en pénalisant les grands écarts :

F1 Score : Le ***F1 score*** permet d'effectuer une bonne évaluation de la performance d'un modèle de **classification** et combine la **précision** et le recall.

$$\textbf{F1} = 2 \times \frac{\textbf{rappel} \times \textbf{precision}}{\textbf{rappel} + \textbf{precision}} \tag{4}$$

Le ***F1 score*** correspond aussi dans un problème de **segmentation** à la similarité entre la prédiction et la **vérité terrain**, on parle desfois de coefficient de ***dice*** [22].

Les fonctions de ***Loss***, utilisées pendant l'entraînement ou non, sont aussi un critère d'évaluation [22]. La ***Loss*** la plus utilisée est la ***Cross Entropy Loss*** pour perte d'entropie croisée. Elle mesure les performances d'un modèle de **classification** dont la sortie est une valeur de score comprise entre

0 et 1. La perte d'entropie croisée augmente à mesure que la probabilité prédite diverge de l'étiquette réelle. Dans un problème bi-classe, elle est définie par :

$$\textbf{\textit{Loss}} = -(y \log(p) + (1-y) \log(1-p)) \text{ [xvi]} \tag{5}$$

où

- p est la probabilité prédite
- y la vérité terrain (0 ou 1 dans le cas binaire)

Ainsi, plus basse est la fonction de coût, plus la prédiction est satisfaisante. Sinon, la ***Loss*** divergerait à cause du terme logarithmique.

Les prédictions sont rendues sous la forme d'un vecteur de score (entre 0 et 1) d'appartenance à la classe recherchée, plutôt qu'une réponse binaire (positif ou négatif). Il y a de ce fait un choix à faire quant au **seuil de confiance** utilisé pour trancher [22]. Par exemple, dans un problème bi-classe, une prédiction donnée avec un score de 0.75 signifie intuitivement que le modèle a jugé qu'il y a 75% de chance qu'il s'agisse en effet d'un positif (*marquage*), et 25% de chance qu'il s'agisse d'un négatif (*arrière-plan*). Avec un **seuil** de 0.5, cette prédiction est considérée comme un positif, mais avec un **seuil** de 0.9 non. Un **seuil** élevé privilégie la précision, tandis qu'un **seuil** faible privilégie le rappel, on peut donc également s'intéresser aux mesures de **précision** et **rappel** à différents **seuils de confiance**.

Enfin, les performances en termes de vitesse d'exécution peuvent être un critère important à considérer. Dans le cadre de la conduite autonome, on aimerait que la **segmentation** se fasse de façon en temps réel. Il s'agit alors de trouver un compromis selon les besoins spécifiques du travail à réaliser.

3.3 Méthodes de *Deep Learning* pour la segmentation

3.3.1 Cadre et objectifs de la segmentation

Les véhicules autonomes sont équipés de capteurs d'images, de caméras, radars, sonars dont les données sont traitées par des processeurs et des logiciels dédiés. Ces logiciels reconstituent la situation routière 3D par reconnaissance de formes (voies, véhicules, obstacles, panneaux, limites de chaussées) et emploient des algorithmes d'intelligence artificielle, aussi dit ***deep learning***, pour décider d'actions à réaliser sur les commandes du véhicule [37].

Ces dernières années, les méthodes de segmentation sémantique basées sur l' **apprentissage profond** pour la détection des marquages routiers ont fait leur apparition, car elles permettent de généraliser les résultats de détection dans des environnements complexes et contiennent des informations sémantiques riches au niveau des pixels [38] [39] [40].

Notre objectif est d'automatiser l'annotation sur **Vérité Terrain Ecodyn** afin que l'utilisateur n'ait plus qu'a vérifier le résultat final. Tout d'abord, une **segmentation** bi-classe est réalisée puis la sémantique des marquages se fera à l'aide d'une **classification** basée sur la géométrie des marquages.

Avant de présenter un état de l'art de la **segmentation** d'images et des applications liées, il est nécessaire de poser les notions importantes propres à ce contexte. En particulier, il faut définir précisément ce qu'on entend par **«segmentation»**.

Dans le traitement des images numériques et la vision par ordinateur, la **segmentation** d'image est le processus de partitionnement d'une image numérique en plusieurs segments d'image, également

xvi. Cette formule peut se généraliser dans un problème multi-classes.

connus sous le nom de régions d'image ou d'objets d'image (ensembles de pixels). Le but de la segmentation est de simplifier et/ou de modifier la représentation d'une image en quelque chose de plus significatif et de plus facile à analyser [41] [42]. La segmentation d'image est généralement utilisée pour localiser des objets et des limites (lignes, courbes, etc.) dans des images. Plus précisément, la **segmentation** d'image est le processus d'attribution d'une étiquette à chaque pixel d'une image de sorte que les pixels ayant la même étiquette partagent certaines caractéristiques.

Le résultat de la **segmentation** d'une image est un ensemble de segments qui couvrent collectivement la totalité de l'image, ou un ensemble de contours extraits de l'image. Chacun des pixels d'une région est similaire en ce qui concerne une caractéristique ou une propriété calculée, comme la couleur, l'intensité ou la texture [41]. Dans le cadre du *stage*, l'annotation faite par l'utilisateur sur **Vérité Terrain Ecodyn** est une segmentation manuelle.

On peut distinguer deux types de **segmentation** :

- **Segmentation sémantique :** approche permettant de détecter, pour chaque pixel, la classe d'appartenance de l'objet [43]. Par exemple, lorsque toutes les personnes d'une figure sont segmentées comme un seul objet et le fond comme un seul objet.
- **Segmentation d'instance :** approche qui identifie, pour chaque pixel, une instance d'appartenance de l'objet. Elle détecte chaque objet d'intérêt distinct dans l'image [44]. Par exemple, lorsque chaque personne dans une figure est segmentée comme un objet individuel.

Les notions sont représentées sur la Figure 11.

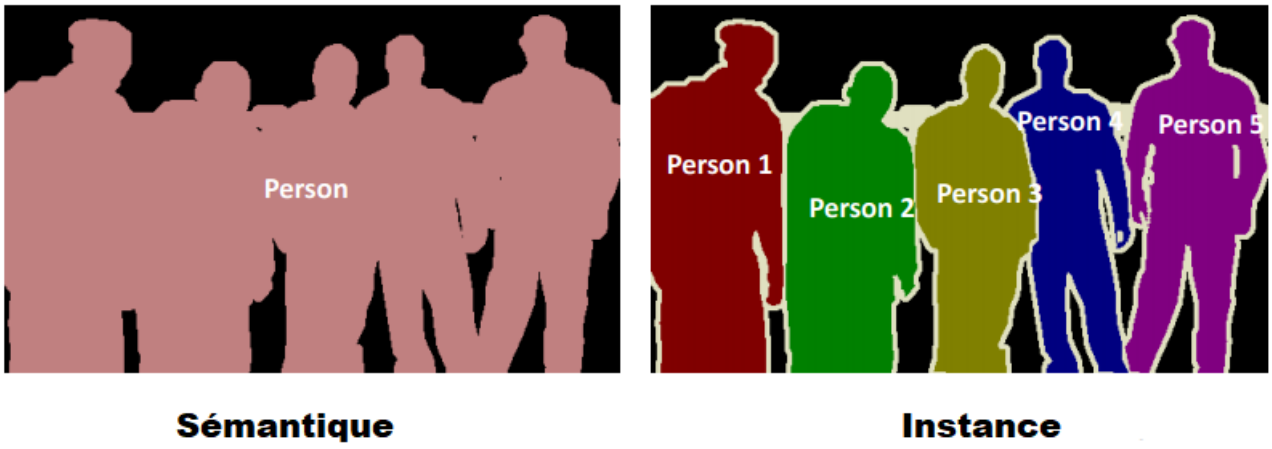


FIGURE 11 – **Segmentation :** Sémantique/Instance.

Source: [45]

Les méthodes de ***machine learning*** et plus particulièrement de ***deep learning*** ont permis, notamment en imagerie, de faire un grand pas en avant dans l'automatisation des tâches de segmentation [43]. Notre but est de segmenter, de façon **sémantique**, automatiquement les marquages routiers à l'aide de **réseaux neuronaux convolutifs**.

3.3.2 Réseaux de neurones convolutifs

Les **réseaux de neurones convolutifs** (**CNN**) sont des modèles populaires aujourd'hui. Ces modèles **CNN** sont utilisés dans différentes applications et différents domaines, et ils sont particuliè-

rement répandus dans les projets de traitement d'images et de vidéos [46]. L'objectif poursuivi ici est de classifier automatiquement des pixels dans la classe *marquage* ou dans la classe *arrière-plan*.

Ces réseaux prennent des images en entrées et les éléments constitutifs des **CNN** sont des filtres, aussi appelés **noyaux**, qui sont des petites matrices carrées de tailles impaires. Les noyaux sont utilisés pour extraire les **caractéristiques** pertinentes, sans les mentionner explicitement, de l'entrée en utilisant l'opération de **convolution** [47] afin d'obtenir ce qu'on appelle une **carte de caractéristiques** [46] [22]. Les **caractéristiques** spatiales font référence à la disposition des pixels et à la relation entre eux dans une image. Elles permettent d'identifier l'objet avec précision, son emplacement, ainsi que sa relation avec d'autres objets dans une image. Ces réseaux utilisent le même principe pour identifier les caractéristiques que nous les humains. Par exemple, nous pouvons reconnaître facilement un visage puisqu'on voit différentes caractéristiques comme les yeux, la bouche ou le nez. Dans le cas du modèle **convolutif**, le filtre fait un tour de l'image afin d'identifier ces **caractéristiques**. Les poids correspondent cette fois-ci aux valeurs des **noyaux** considérés.

Ainsi, la **couche de convolution** est la composante clé des **réseaux de neurones convolutifs**, et constitue toujours au moins leur première couche. Cette couche prend soit des images en entrée, soit des **cartes de caractéristiques**. L'objectif de cette couche est d'extraire des caractéristiques propres à chaque donnée [46] [48]. Ensuite, ces données sont compressées de façon à réduire leur taille initiale via des méthodes de sous-échantillonnage afin de minimiser le nombre de données à traiter tout en préservant les informations importantes de l'image initiale. Des paramètres supplémentaires sont à considérer comme la ***stride***, qui est le pas de déplacement du **noyau**, et le ***padding***, qui correspond au fait de rajouter des valeurs nulles autour de l'entrée afin d'avoir la même dimension de sortie qu'au départ.

Les méthodes de sous-échantillonage se trouvent dans les couches de ***pooling*** (réduction de dimension et regroupement) [46] [22]. Ce type de couche est souvent placé entre deux couches de convolution : elle reçoit en entrée plusieurs **cartes de caractéristiques**, et applique à chacune d'entre elles l'opération de mise en commun (***pooling***). L'opération de ***pooling***, la plus connue, est le ***max-pooling*** qui consiste à faire glisser une petite matrice, **2×2** par exemple, sur l'image et de prendre la valeur maximale du carré sur lequel la matrice se place (cf. Figure 12). Les informations importantes sont ainsi préservées.

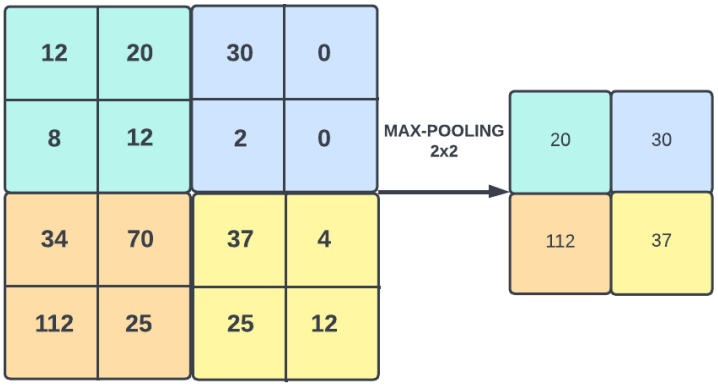


FIGURE 12 – Exemple d'un ***max pooling 2×2***

La sortie admet le même nombre de **cartes de caractéristiques** qu'en entrée, mais celles-ci sont

bien plus petites. La couche de ***pooling*** permet donc de réduire le nombre de paramètres et de calculs dans le réseau.

Pour finir, les dernières couches, qui sont entièrement connectées (***fully-connected***), dans ces réseaux permettent de classifier l'image en entrée du réseau [48]. En sortie, un vecteur de score, souvent l'image d'une fonction ***softmax*** [49] est obtenue. Cette fonction permet que chaque coordonnée corresponde à un score pour lequel l'image appartient à une classe, dans notre cas si un pixel est un marquage ou non.

3.3.3 État de l'art des modèles de segmentation

Modèle UNet

Le réseau **UNet**, développé par *Olaf Ronneberger, Phillip Fischer*, et *Thomas Brox* en 2015 pour la **segmentation** d'images médicales [50], est reconnu comme une des architectures les plus efficaces pour faire de la **segmentation** [40]. Il s'agit de réseaux entièrement **convolutifs** constitués d'une moitié **«encodeur»** (***contracting path***) à gauche et d'une moitié **«décodeur»** (***expanding path***) à droite (cf. Figure [13]).

Le réseau est assez fort pour faire de bonnes prédictions basées sur peu de données d'entraînement en utilisant des techniques d'augmentation de données excessives [51]. Il offre aussi une précision supérieure aux modèles conventionnels [52]. Au-delà de sa robustesse, le **UNet** a une architecture simple basée sur les auto-encodeurs [53], qui permettent d'avoir une même taille d'image en sortie qu'en entrée, et des couches de convolution faciles à implémenter. Ainsi, il est possible de modifier et d'augmenter/diminuer le nombre de couches et donc de faire varier le nombre de paramètres. De plus, le **UNet** peut être utilisé dans un problème multi-classes.

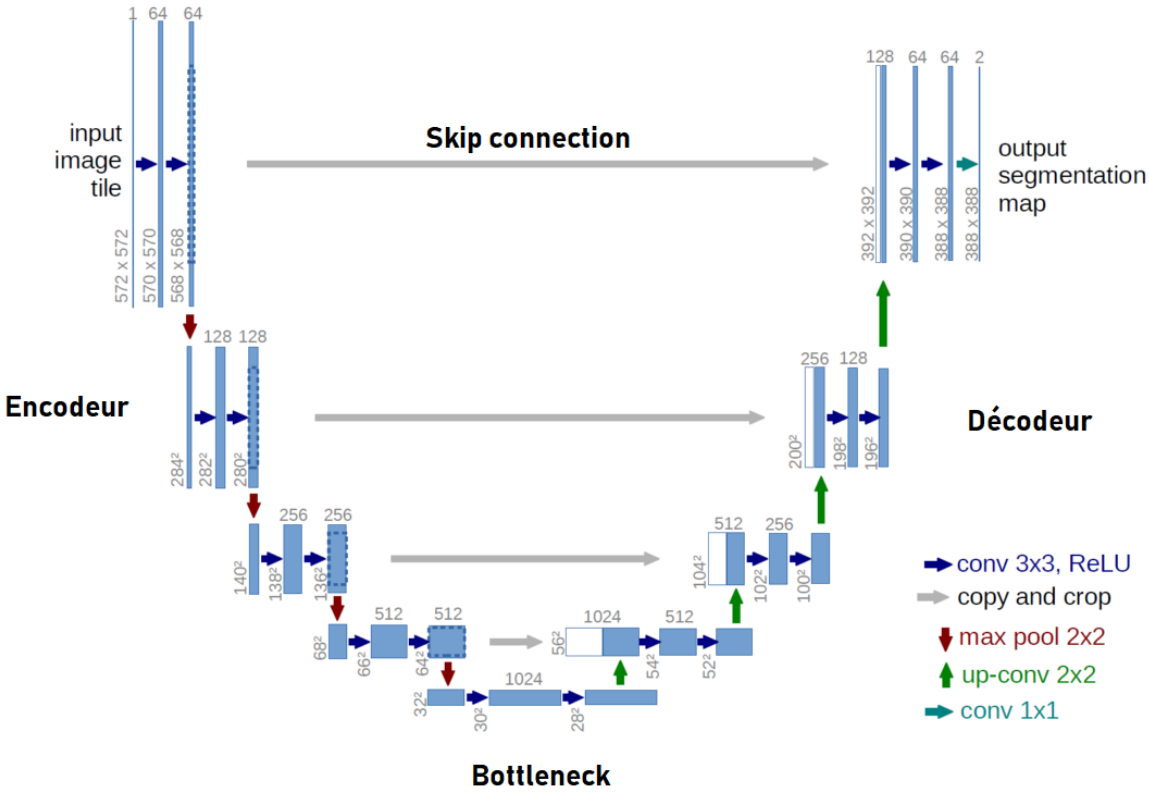


FIGURE 13 – Architecture du **UNet** avec une profondeur de 4 mis en place en 2015.
Source: [50]

Une description de l'architecture représentée sur la Figure 13 est donnée ci-dessous.

Encodeur : Une image est prise en entrée avec une certaine dimension et un nombre de canaux spécifique à celle-ci. À chaque niveau de l'encodeur, **deux convolutions 3×3** successives sont effectuées afin d'augmenter le nombre de canaux de sorties. En général, on double le nombre de canaux de sortie à chaque fois. Chacune de ces convolutions est suivie d'une fonction d'activation **ReLu**$(x) = \max(0, x)$. Ensuite, une opération de sous échantillonnage est effectuée, ici un **max-pooling**, pour diviser par deux les dimensions de l'image et diminuer le nombre de paramètres au fur et à mesure. Ainsi, la taille de l'image diminue et le nombre de canaux augmente à mesure qu'on approche du centre du réseau. De plus, dans l'article original, une couche de ***Batch Normalization*** est ajoutée.

Bottleneck : Au milieu de réseau, il y a à nouveau les **deux convolutions 3×3** suivies de leurs activations **ReLu**, mais pas de sous échantillonnage. C'est ici que les caractéristiques les plus importantes de l'image se manifestent.

Décodeur : En remontant chaque niveau du décodeur, à chaque fois un sur-échantillonage de la sortie précédente est effectuée à l'aide d'une **convolution transposée** [54] et d'une concaténation avec la sortie des convolutions au niveau de l'encodeur correspondant. À chaque descente, il en résulte une perte de résolution spatiale des cartes de caractéristiques. La représentation de l'image de plus en plus déficitaire n'est pas avantageuse pour la segmentation où la délimitation des limites

est vitale. Il est donc nécessaire de capturer et de stocker les informations relatives aux limites dans les cartes de caractéristiques du codeur avant de procéder au sous-échantillonnage. Ces concaténations sont donc nécessaires pour la convergence du modèle. En effet, il existe des informations de bas niveau partagées entre l'entrée et la sortie, et il serait souhaitable de faire passer ces informations directement à travers le réseau. Il est validé expérimentalement que ces chemins supplémentaires sont souvent bénéfiques pour la convergence du modèle [55]. Ces connexions, aussi appelées ***skip connections***, dans les architectures profondes, comme leur nom l'indique, sautent une couche du réseau neuronal et alimentent la sortie d'une couche comme entrée des couches suivantes. Puis un bloc de convolution est à nouveau appliqué suivi d'une activation **ReLu** afin de diviser par deux le nombre de canaux. Ainsi, la taille de l'image augmente et le nombre de canaux diminue à mesure qu'on s'approche de la sortie du réseau. Une couche de **Batch Normalization** est souvent aussi ajoutée.

Il existe différentes techniques pour sur-échantillonner, dans le cas du **Unet** la **convolution transposée** est utilisée [54]. Un exemple est donné en **Annexe** (cf. Figure 44).

Sortie : Lorsque la dimension de l'image d'origine est obtenue, une **convolution 1 × 1** est appliquée avec en sortie le nombre de canaux désiré (2 dans le cas *marquage/arrière-plan*), suivie d'une activation **softmax** qui renvoie un vecteur de score en sortie.

Ce type de réseau est particulièrement adapté au cas des marquages routiers, car il permet d'avoir l'information de **classification** et de localisation. En effet, suite au décodage, le résultat est une image haute définition complète dans laquelle tous les pixels sont classés, donc une prédiction pixel par pixel.

Ce type de réseau permet de faire de bonnes prédictions à partir de peu de données d'entraînement. De plus, des techniques d'augmentations de données, comme des rotations, des translations ou bien un floutage, peuvent être facilement utilisées. La structure de **«U»** offre la possibilité de concaténer l'information de l'encodeur de la même couche afin de perdre le moins d'informations possible, qui est un problème majeur dans d'autres structures auto-encodeur [53]. Il parait donc particulièrement adapté aux marquages routiers. De plus, au-delà de sa robustesse, le **UNet** a une architecture simple basée sur les auto-encodeurs et des **couches convolutives** faciles à implémenter. Ainsi, il est très facile de modifier le nombre de couches ainsi que les éléments au sein de l'architecture. Il parait donc particulièrement adapté aux marquages routiers, notamment du fait du peu de données disponibles.

Modèle Segnet

Le **SegNet** est conçu, en 2015 [56], comme une architecture efficace pour la **segmentation sémantique** par pixel. Il est principalement motivé par les applications de compréhension de scènes routières qui nécessitent la capacité de modéliser l'apparence (route, bâtiment), la forme (voitures, piétons) et de comprendre la relation spatiale (contexte) entre différentes classes telles que la route et le trottoir [56], d'où l'intérêt de ce réseau dans le cadre du *stage*. Dans les scènes routières typiques, la majorité des pixels appartiennent à de grandes classes telles que les routes et les bâtiments, et le réseau doit donc produire une segmentation lisse.

Le **SegNet** est composé d'un réseau encodeur et d'un réseau décodeur correspondant comme pour le **UNet**, suivis d'une couche de **classification** finale par pixel. Cette architecture est illustrée dans la Figure 14. Le réseau encodeur est composé de 13 **couches convolutionnelles** qui correspondent aux 13 premières **couches convolutionnelles** du réseau **VGG16** [57] conçu pour la **classification** d'objets. À chaque couche d'encodage correspond une couche de décodage et le réseau de décodage compte donc 13 couches. La sortie finale du décodeur est transmise à un classifieur **softmax** multi-classes pour produire des score de classe pour chaque pixel indépendamment.

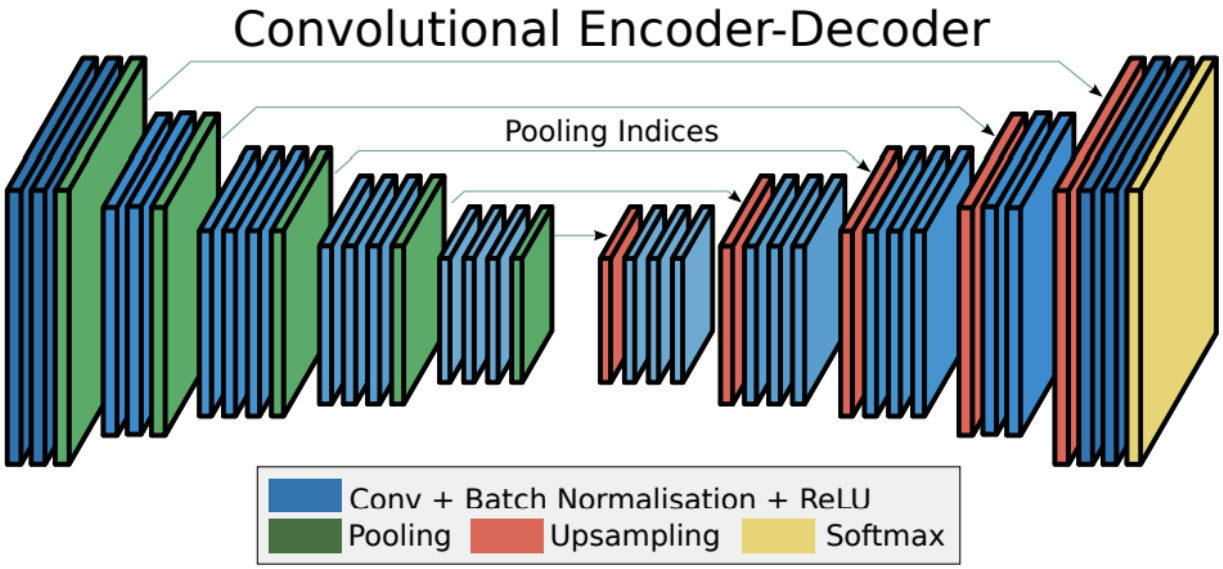


FIGURE 14 – L'Architecture **SegNet** de profondeur 4 mit en place en 2015

Source: [56]

Le principe de l'encodeur est le même que pour le **UNet** avec les blocs de convolutions et le ***pooling***. Cette fois-ci, afin de limiter les problèmes de résolutions, les **cartes de caractéristiques** ne sont pas stockées contrairement à l'**UNet**. Il s'agit de stocker uniquement les indices de ***max-pooling***, c'est-à-dire que les emplacements de la valeur maximale des caractéristiques dans chaque fenêtre de ***max-pooling*** sont mémorisés pour chaque **carte caractéristique** de l'encodeur. En principe, cela peut être fait en utilisant qu'un entier pour chaque fenêtre de ***max-pooling*** 2×2 et est donc beaucoup plus efficace à stocker que la mémorisation des **cartes caractéristiques** en précision flottante [56] [58].

Le sur-échantillonnange, dans le décodeur, des **cartes caractéristiques** d'entrée utilisent les indices de ***max-pooling*** mémorisés à partir des **cartes caractéristiques** de l'encodeur correspondant. Cette méthode est appelée le ***max-pooling upsampling*** [59]. Un exemple est donné en **Annexe** (cf. Figure 46). Le reste est très similaire à l'architecture du **UNet**.

3.4 Méthodes de *Deep Learning* pour le traitement de langage naturel

Les réseaux neuronaux profonds traditionnels supposent que les entrées et les sorties sont indépendantes les unes des autres. Ceci est particulièrement gênant pour l'**analyse sémantique** de la route afin d'identifier des erreurs. En effet, la route se caractérise d'une par une succession logique de marquage. L'emplacement d'un marquage dépend du marquage précédant. Pour cela, de nouvelles méthodes qui se caractérisent par leur **«mémoire»** sont envisagées. En effet, la sortie des réseaux neuronaux récurrents dépend des éléments antérieurs de la séquence. Ainsi, on va tout d'abord parler du **traitement de langage naturel**, de son intérêt dans l'**analyse sémantique**, puis présenter des modèles existants.

3.4.1 Cadre et objectifs du traitement de langage naturel

Le **NLP** pour ***Natural Language Processing*** ou **Traitement du Langage Naturel** est une discipline qui porte essentiellement sur la compréhension, la manipulation et la génération du langage naturel par les machines. Le **NLP** recouvre un champ d'application très vaste comme la traduction automatique, l'analyse de sentiments ou même de correction automatique. Ainsi, le **NLP** est réellement à l'interface entre la science informatique et la linguistique [60]. La compréhension du langage naturel peut se faire par une **analyse syntaxique** ou une **analyse sémantique**. L'**analyse syntaxique** correspond à l'analyse du langage en fonction des règles grammaticales. Elle s'applique donc à un groupe de mots et non aux mots individuels. Quant à l'**analyse sémantique**, c'est le processus qui permet de comprendre le sens ou la logique d'un énoncé. Elle consiste à comprendre le sens et à interpréter les mots, les signes et la structure des phrases. C'est la deuxième syntaxe qui sera considérée. Jusque dans les années 1980, la plupart des systèmes de **traitement du langage naturel** étaient basés sur des ensembles complexes de règles écrites à la main. Cependant, à partir de la fin des années 1980, une révolution s'est produite dans le **traitement du langage naturel** avec l'introduction d'algorithmes d'apprentissage automatique pour le traitement du langage [60].

Dans les années 2010, les méthodes d'apprentissage automatique de type réseau de neurones profond se sont répandus dans le **traitement du langage naturel**. Cette popularité est due en partie à une avalanche de résultats montrant que ces techniques [61][62] peuvent obtenir des résultats de pointe dans de nombreuses tâches liées au langage naturel, par exemple dans le domaine de la médecine et des soins de santé, où le **NLP** aide à analyser les notes et le texte des dossiers médicaux électroniques qui seraient autrement inaccessibles à l'étude lorsqu'on cherche à améliorer les soins [63].

Les marquages routiers incarnent les règles de la route tout en reflétant le tracé du chemin à venir. Ces règles sont diligemment étudiées et appliquées aux situations de conduite par des conducteurs humains qui ont appris le code de la route. Il faut apprendre à un véhicule autonome à lire la route et à la comprendre, comme le ferait un humain. De plus, il existe une grande variation entre les pays et les régions, ce qui rend l'interprétation difficile. Certains articles utilisent des techniques de **classification** des marquages à partir d'images de caméra à l'aide des caractéristiques géométriques du marquage indépendamment des autres [64] mais aussi à l'aide de la géométrie et répartition des marquages au sein de l'image en utilisant des graphes [65].

Comme la géométrie des marquage est standardisée [5], l'approche de **classification** se base sur l'utilisation des métriques géométriques puis d'une analyse de la grammaire résultante. L'objectif est de lire et interpréter les règles de l'itinéraire encodées dans les marquages routiers afin d'acquérir la topologie actuelle et à venir du chemin. Par exemple, l'observation d'un marquage de type `FLECHE` informe que dans le futur, qu'il y aura d'autres marquages du même type et indique la future interdiction de dépasser ou d'un rétrécissement d'une voie. Les marquages peuvent être symbolisés comme des lettres et toute la route comme une phrase. C'est ici qu'intervient le **traitement de langage naturel** afin d'analyser la grammaire d'une route et permettre d'identifier des erreurs au sein de la **classification**. Cette identification permettra à l'utilisateur lors de l'annotation des marquages via le logiciel **Vérité Terrain Ecodyn** de se concentrer plus précisément sur certaines positions de l'itinéraire afin d'optimiser le temps d'annotation.

3.4.2 Représentation des données pour le traitement de langage naturel

Afin de pouvoir appliquer les méthodes de ***Machine Learning*** aux problèmes relatifs au langage naturel, il est indispensable de transformer les données textuelles en données numériques. Le pré-

traitement des données brutes fait partie des points cruciaux avant le développement d'algorithmes. Deux représentations sont présentées ci-dessous.

One hot Encoding

La représentation vectorielle la plus classique des mots est le [xvii] ***One hot Encoding*** **[66]**. Une dimension est allouée pour chaque mot du vocabulaire et cette dimension correspond au nombre de mots dans ce vocabulaire. Chaque mot est représenté comme un vecteur binaire avec toutes ses valeurs nulles à l'exception de l'index du mot. La taille est définit par le nombre de mots du vocabulaire. Avec cette représentation, tous les mots ont la même structure. Ainsi, l'encodage ***one hot*** n'apporte donc qu'une information de différence entre des mots. De plus, cette description peut engendrer des matrices creuses de très grandes tailles lorsque le dictionnaire de mots est de très grande taille. Ces limites ont entrainé la naissance des couches ***Embedding***.

Vecteur d'Embedding

La couche d'***Embedding*** (couche d'incorporation) est une couche de neurones qui est capable, en réduisant la dimension, de capturer le contexte, la similarité sémantique et syntaxique d'un mot. Un mot sera ainsi représenté par un vecteur dont la taille e_b est fixée au préalable et chaque coordonnée correspond à une caractéristique de ce mot **[67]**. Ces caractéristiques dépendent de paramètres qui sont mis à jour au cours de l'entrainement d'un modèle de réseaux de neurones. Si X est un mot représenté sous sa forme ***one hot*** dans $\mathbf{R}^N$ où N est la taille du dictionnaire de vocabulaire, alors le ***vecteur d'Embedding*** e associée est donné par la relation linéaire :

$$e = W^\top X \text{ [xviii]}$$

où $W \in \mathcal{M}_{N \times e_b}$ est la matrice de poids pondérée, aussi appelé **matrice d'Embedding** [xix]. Au début de l'entrainement, ces paramètres sont initialisés de façon aléatoire et prennent sens au cours de l'apprentissage. C'est le modèle qui décide de ses caractéristiques pendant la formation. Les coordonnées de chaque lettre peuvent être représentées dans un espace euclidien : lorsque deux mots admettent des caractéristiques linguistiques et des contextes similaires, les points qui les représentent seront proches graphiquement **[68]**.

En ajoutant une couche dense, suite à cette couche d'**Embedding**, il est possible de faire une **classification** des phrases en entrée. Cependant, les réseaux classiques, dit de **feedforward** [xx], ne prennent pas en compte l'inter-dépendance des mots dans une phrase, contrairement à nous qui pouvons deviner la fin d'une phrase à partir des premiers mots. Dans une séquence de données, il existe généralement des dépendances entre les données d'entrée et/ou les signaux de sortie. Les réseaux de neurones **feedforward** peuvent être modifiés et adaptés à des données sous forme de séquences, c'est-à-dire leur donner la capacité de gérer des signaux de taille variable, tout en leur donnant une mémoire de ce qu'ils ont vu auparavant. Ces nouveaux réseaux sont appelés les **réseaux de neurones récurrents**.

xvii. A noter que ces vecteurs ne sont pas propre au traitement de langage naturel mais sont plutôt souvent utilisés en sortie d'un modèle de **classification**

xviii. Les vecteurs sont notés en colonne.

xix. La colonne i de cette matrice correspond au vecteur d'**Embedding** de l'entrée i

xx. Ces réseaux sont tout simplement des réseaux de neurones acycliques dont le plus connu est le perceptron multicouche

3.4.3 État de l'art des modèles pour le traitement de langage naturel

Le premier modèle de **traitement de langage naturel**, qu'est le réseau de neurones récurrent, est exposé. Son amélioration, le **LSTM** est ensuite présenté. Enfin, une récente méthode qu'est le **mécanisme d'attention** est décrite.

Réseaux de neurones récurrents

Les **réseaux de neurones récurrents (RNN)** constituent une famille de réseaux neuronaux spécialisée dans le traitement de données séquentielles [69]. La notion de séquence est primordiale pour analyser la sémantique de la route. En effet, on peut définir une séquence de taille T strictement positive comme un ensemble ordonné de données, donc dans le cas de la sémantique, comme une succession de marquages et d'inter-marquages de façon logique. L'idée est de diviser la route en plusieurs séquences et de classifier si elle est vraie ou fausse selon la présence d'une erreur logique au sein de la route. Par exemple, une occurrence d'un marquage de type $\mathtt{T}_4$ dans un voisinage de marquages de type $\mathtt{T}_1$ indiquera que la séquence est fausse [5].
En outre, les réseaux récurrents partagent les paramètres d'un élément à un autre. Chaque membre de la sortie est une fonction des membres précédents de la sortie avec la même règle de mise à jour. Cette formulation récurrente entraîne le partage des paramètres à travers un graphe de calcul très profond. Afin de représenter ce partage de paramètres, un schéma visuel d'un réseau récurrent à une couche est présenté dans la Figure 15.

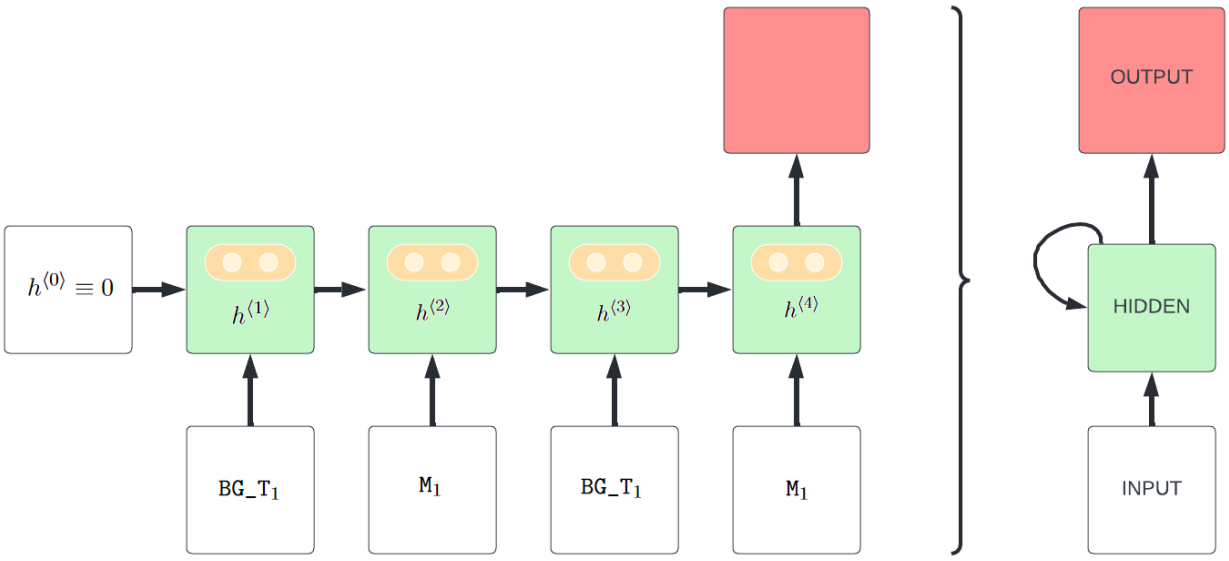


FIGURE 15 – Un exemple d'un réseau récurrent à une couche. Représentation dite **«dépliée»** à gauche et **«repliée»** à droite.

Ce schéma récurrent vérifie les relations suivantes :

$$\begin{cases} h^{\langle t \rangle} & = g_1(W_h^\top h^{\langle t-1 \rangle} + W_e^\top e^{\langle t \rangle} + b_h) \quad \text{pour } t > 0 \\ h^{\langle 0 \rangle} & = 0 \end{cases} \tag{6}$$

où :

- $T = 4$ est la taille de la séquence d'entrée,
- $(e_i)_i \in \mathbf{R}^{e_b}$ sont les vecteurs représentant les lettres de la liste d'entrée, ici notés $[\texttt{BG\_T}_1, \texttt{M}_1, \texttt{BG\_T}_1, \texttt{M}_1]$,
- $W_h \in \mathcal{M}_{N_n \times N_n}$ et $W_e \in \mathcal{M}_{e_b \times N_n}$ sont des matrices de poids[xxi] et $b_h \in \mathbf{R}^{N_n}$ est un vecteur de biais, où N_n est le nombre de neurones choisi au préalable (2 sur la Figure 15),
- $h^{\langle t \rangle} \in \mathbf{R}^{N_n}$ l'état caché,
- g_1 est une fonction d'activation, souvent $g_1 = \tanh$,
- la sortie de **classification**, noté $y^{\langle T \rangle}$, peut être obtenue par couche dense avec la relation $y^{\langle T \rangle} = g_2(W_y^T h^{\langle T \rangle} + b_y)$ où $W_y \in \mathcal{M}_{N_n \times 1}$, $b_y \in \mathbf{R}^1$ et g_2 une fonction d'activation.

L'ensemble des carrés verts correspond à une couche récurrente et un seul carré vert est appelé cellule récurrente. Les cercles jaunes correspondent aux neurones dont le nombre est spécifié par l'utilisateur. À noter que les matrices de poids et le vecteur de biais sont indépendants du temps. Dans l'équation (6), il y a apparition d'un état dit caché $h^{\langle t \rangle}$ qui est calculé à partir de l'entrée de la séquence au temps t mais aussi de l'état caché précédant au temps $t-1$ qui lui-même dépend de l'entrée au temps $t-1$ etc. Cette équation traduit la notion de récurrence et introduit la notion de mémoire au sein du réseau de neurones. À noter que sur le schéma, il n'y a qu'une seule couche cachée. L'empilement de couches supplémentaires est possible et permet d'extraire des informations plus complexes à partir des entrées, et ainsi d'avoir une meilleure modélisation de nos données. La version à gauche est dite **«dépliée»** alors que la version à droite est dite **«repliée»** afin d'avoir un meilleur rendu visuel.

Il existe différentes applications principales des modèles **RNN** [70] :

- Tâche de **classification**, par exemple de sentiment ou bien de sémantique de route : Modèle **many-to-one**
- Tâche de traduction machine : Modèle **many-to-many**
- Génération de musique à partir d'une note : Modèle **one-to-many**
- Le Modèle **one-to-one** revient tout simplement à un réseau de neurones classique

Limites des RNN

Bien qu'ils soient construits spécifiquement pour gérer des séquences, les **RNN** simples ont certaines limites. L'une des principales difficultés pour entraîner des réseaux de type **RNN** est le problème de la **disparition du gradient**. Pour rappel, l'apprentissage d'un réseau de neurone consiste à ajuster des poids et biais afin de minimiser une erreur, donnée par une fonction de coût L[xxii] qui est paramétré par la prédiction $y^{\langle T \rangle}$ et la vérité y. On cherche ainsi à minimiser la fonction de coût afin de mettre à jour les paramètres [71]. Pour cela, l'**algorithme de descente de gradient** est utilisé et le calcul des gradients par rapport aux poids et biais est utilisé. On parle aussi de **rétro-propagation du gradient**. Pour mettre à jour les poids W_e, il faut calculer la dérivée de la fonction de coût L par rapport à ces poids. Par *chain rule*, le calcul devient :

$$\frac{\partial L}{\partial W_e} = \frac{\partial L}{\partial y^{\langle T \rangle}} \frac{\partial y^{\langle T \rangle}}{\partial h^{\langle T \rangle}} \frac{\partial h^{\langle T \rangle}}{\partial W_e}$$ [xxiii]

xxi. On peut très bien avoir une seule matrice de poids en concaténant $h^{\langle t-1 \rangle}$ et $e^{\langle t \rangle}$ en un seul vecteur

xxii. Dans le cas de plusieurs sorties, la traduction par exemple, il y a plusieurs coût $L^{\langle t \rangle}$ pour chaque temps, et L s'obtient en faisant la somme

xxiii. La dérivée par rapport à une matrice est une expression abusive, il faut voir ça comme une dérivée par rapport à chaque coefficient de W_e

Cependant, $h^{\langle T \rangle}$ dépend de $h^{\langle T-1 \rangle}$ qui lui dépend aussi des poids W_e, il faut donc continuer la décomposition [72] :

$$\frac{\partial L}{\partial W_e} = \frac{\partial L}{\partial y^{\langle T \rangle}} \frac{\partial y^{\langle T \rangle}}{\partial h^{\langle T \rangle}} \frac{\partial h^{\langle T \rangle}}{\partial h^{\langle T-1 \rangle}} \frac{\partial h^{\langle T-1 \rangle}}{\partial h^{\langle T-2 \rangle}} \cdots \frac{\partial h^{\langle 1 \rangle}}{\partial W_e}$$

Le raisonnement est équivalent pour les autres poids et biais. Les dérivées des fonctions d'activations g_1 et g_2 sont impliquées dans le calcul, mais les fonctions classiques du type tanh ou sigmoïde sont bornées par 1. Par conséquent, plus longue est la séquence d'entrée, plus le nombre de multiplications de nombre assez petit est impliqué. Cela fera tendre la dérivée vers 0 et le modèle aura donc des difficultés pour l'apprentissage, on appelle ce phénomène la **disparition du gradient**. À noter que ce phénomène apparaît déjà dans les réseaux de neurones de base, plus le réseau de ***feedforward*** est profond, plus il est impacté par cette disparition. Cependant, pour les **RNN**, ce phénomène est corrélé par la profondeur du réseau, mais aussi par la taille du vecteur d'entrée, d'où les difficultés spécifiques à ces réseaux. Il peut aussi être très touché par le phénomène inverse appelé, l'**explosion du gradient** [73], où ici la norme du gradient tend vers un nombre très grand. En conclusion, les **RNN** ne permettent pas de mémoriser les dépendances à long terme. De nouvelles architectures ont été mises en place pour palier à ces problèmes.

LSTM

Pour surmonter le problème de la **disparition du gradient** et donc à cette perte d'information sur le long terme, le **LSTM**, qui signifie **Long Short-Term Memory**, est proposé initialement par *S. Hochreiter* [74], puis amélioré dans l'article [75] de *F. Gers* et *j. Schmidhuber*. Le **LSTM** est une cellule composée de trois «portes» : une porte d'oubli, une porte d'entrée et une porte de sortie (cf. Figure 16). Ces portes sont des zones de calculs qui régulent le flot d'informations en réalisant des actions spécifiques. Il y a cette fois-ci deux types de sorties : un **état caché** $h^{\langle t \rangle}$, où $h^{\langle t \rangle} \equiv 0$, qui comme précédemment correspond à la mémoire sur le court terme, et un **état de la cellule** $c^{\langle t \rangle}$, où $c^{\langle t \rangle} \equiv 0$, qui correspond à la mémoire sur le long terme. Le réseau peut ainsi conserver des informations qu'il a vues longtemps auparavant (contrairement au **RNN** classique).

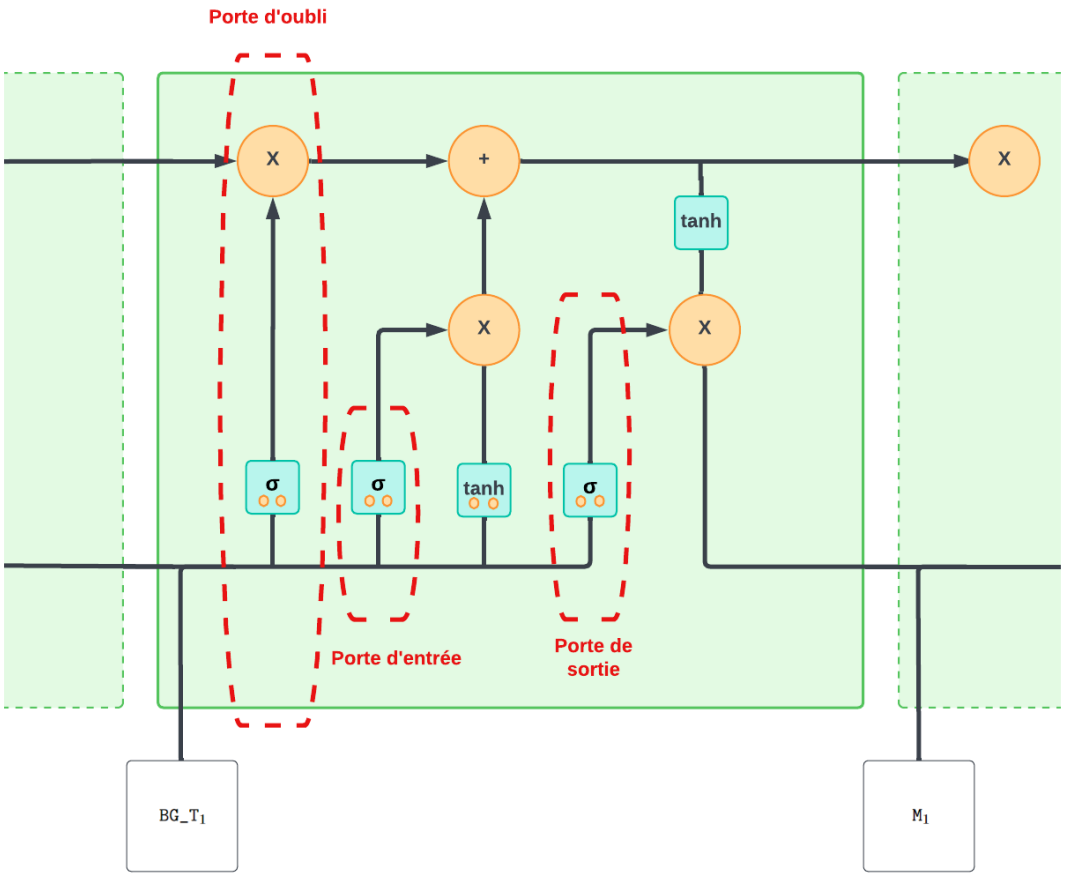


FIGURE 16 – Représentation d'une cellule **LSTM**.

Les entrées de chaque porte sont pondérées par des poids liés aux portes ainsi que par un biais. Elles dépendent de la dimension du nombre de neurones fixé N_n et de la dimension de $e^{\langle T \rangle}$ mais pas du temps. Il existe ainsi 4 triplets de paramètres :

- $(W_f, W_{h_f}, b_f) \in \mathcal{M}_{e_b \times N_n} \times \mathcal{M}_{N_n \times N_n} \times \mathbf{R}^{N_n}$ qui pondèrent l'entrée de la porte d'oubli
- $(W_i, W_{h_i}, b_i) \in \mathcal{M}_{e_b \times N_n} \times \mathcal{M}_{N_n \times N_n} \times \mathbf{R}^{N_n}$ qui pondèrent l'entrée de la porte d'entrée
- $(W_c, W_{h_c}, b_c) \in \mathcal{M}_{e_b \times N_n} \times \mathcal{M}_{N_n \times N_n} \times \mathbf{R}^{N_n}$ qui pondèrent les données qui vont se combiner à la porte d'entrée pour mettre à jour l'état de la cellule c_t
- $(W_o, W_{h_o}, b_o) \in \mathcal{M}_{e_b \times N_n} \times \mathcal{M}_{N_n \times N_n} \times \mathbf{R}^{N_n}$ qui pondèrent l'entrée de la porte de sortie

Ces poids et biais sont mis à jour au fur et à mesure de l'apprentissage comme pour les modèles précédents. Une explication des différentes portes est donnée ci-dessous à un temps $t > 0$.

Porte de l'oubli

La **porte de l'oubli** sert à décider quelle information est à conserver ou non à partir de l'état caché précédant et de la donnée d'entrée au temps t. Deux couches linéaires sont appliquées puis la

fonction sigmoïde est utilisée composante par composante afin de normaliser les données entre 0 et 1 :

$$F^{\langle t \rangle} = \sigma(W_f^\top e^{\langle t \rangle} + W_{h_f}^\top h^{\langle t-1 \rangle} + b_f) \quad \text{pour t} > 0 \tag{7}$$

Ce réseau est entraîné afin que la sortie ait des composantes proches de 0 si celles-ci sont jugées non pertinentes et des composantes proches de 1 lorsqu'elles le sont. La sortie de la sigmoïde peut être vue comme un filtre. En effet, le vecteur résultant est multiplié composante par composante avec l'état de la cellule précédente au temps $t-1$. La **porte de l'oubli** décide quels éléments de la mémoire à long terme sont oubliés (multiplication avec un nombre proche de 0) ou non compte tenu de la mémoire à court terme et de la nouvelle donnée de la séquence.

Porte d'entrée

La **porte d'entrée** a pour rôle d'extraire l'information de la donnée courante $e^{\langle t \rangle}$ à partir des mêmes données que la porte de l'oubli, c'est-à-dire l'entrée de la séquence au temps t et de l'état caché au temps $t-1$. Le but est de déterminer quelle information doit être gardée dans la mémoire à long terme. Deux couches linéaires sont appliquées en parallèle. Une des sorties est suivie d'une sigmoïde et l'autre d'une activation tanh :

$$I_t = \sigma(W_i^\top e^{\langle t \rangle} + W_{h_i}^\top h^{\langle t-1 \rangle} + b_i) \quad \text{pour t} > 0 \tag{8}$$

Comme précédemment, la sigmoïde va renvoyer un vecteur pour lequel une coordonnée proche de 0 signifie que celle-ci ne sera pas jugée importante.

État de la cellule préalablement à la porte d'oubli

L'**état de la cellule** se calcule à partir de la **porte d'oubli** et de la porte d'entrée selon la formule suivante :

$$c^{\langle t \rangle} = F_t \otimes c^{\langle t-1 \rangle} + I_t \otimes \tanh(W_c^\top e^{\langle t \rangle} + W_{h_c}^\top h^{\langle t-1 \rangle} + b_c) \text{ [xxiv]} \quad \text{pour t} > 0 \tag{9}$$

Tout d'abord, l'ancien **état de la cellule** est multiplié composante par composante avec la textbfporte de l'oubli. Cela permet d'éliminer certaines informations de l'état précédent qui ne servent pas pour la nouvelle prédiction à faire. Ensuite, la couche linéaire suivi d'une fonction tanh va écraser les valeurs entre -1 et 1 pour éviter les problèmes de surcharges en calculs. Cet intervalle qui contient des valeurs négatives est nécessaire pour réduire l'impact d'une composante de l'**état de la cellule**. Ce vecteur nous indique dans quelle mesure il faut mettre à jour chaque composante de la mémoire à long terme. Le produit, composante par composante, avec le vecteur d'entrée, permettra de ne garder que les informations importantes, les autres étant remplacées par des valeurs quasiment nulles. Ensuite, on additionne le tout, coordonnée par coordonnée, ce qui permet d'enregistrer dans l'**état de la cellule** ce que le **LSTM** (parmi les entrées et l'état caché précédent) a jugé pertinent.

Porte de sortie

La **porte de sortie** doit décider quel sera le prochain état caché, qui contient des informations sur les entrées précédentes du réseau et sert aux prédictions. Pour ce faire, l'**état de la cellule** calculé

xxiv. $\otimes$ correspond à la multiplication terme à terme entre deux vecteurs de même dimension

précédemment est normalisé entre −1 et 1 grâce à tanh (sans couche linéaire cette fois-ci). On recrée un filtre via une couche linéaire suivi d'une sigmoïde comme précédemment :

$$O_t = \sigma(W_o^\top e^{\langle t \rangle} + W_{h_o}^\top h^{\langle t-1 \rangle} + b_o) \quad \text{pour t} > 0$$

Puis ce filtre est appliqué en le multipliant avec le vecteur de sortie du tanh.

$$h_t = O_t \otimes \tanh(c^{\langle t \rangle}) \quad \text{pour t} > 0$$

Cela permet d'assurer que seule les informations nécessaires sont émises à l'état caché au temps t.

En conclusion, les **LSTM** sont des **RNN** améliorés afin de contrer le phénomène de **disparition de gradient**. C'est la présence du vecteur d'activation de la **porte d'oubli** dans le terme de gradient (lorsqu'on calcule la dérivée partielle par rapport aux poids), ainsi que la structure additive, qui permet au **LSTM** de trouver une mise à jour de paramètre à n'importe quel pas de temps convenable, c'est-à-dire la norme du gradient non proche de 0 **[76]**.

Mécanisme d'attention du Transformer

Le **Transformer** est un réseau de neurones de type séquence à séquence comme les **RNN**. Il se différencie par le fait de n'utiliser que le **mécanisme d'attention** qui permet de conserver l'interdépendance des mots et est au centre de son architecture [77]. Ce concept d'attention mesure le lien entre deux éléments de deux séquences. Ainsi, en traitement de langage naturel, le **mécanisme d'attention** permet de transmettre l'information au modèle, afin qu'il porte son attention au bon endroit sur les mots d'une séquence.

Les modèles récurrents génèrent des séquences d'états cachés en fonction de l'état caché précédent et de l'entrée pour la position t. Cette nature intrinsèquement séquentielle empêche la parallélisation au sein des exemples d'apprentissage, ce qui devient critique pour les séquences plus longues, car les contraintes de mémoire limitent le regroupement entre les exemples [77]. C'est l'une des motivations avec la rapidité qui pousse à la création des **Transformers**. Le **Transformer**, très utilisé dans la traduction [78], est présenté dans le papier ***«Attention Is All You Need»*** [77] en 2017 qui décrit son architecture et ses performances impressionnantes sur plusieurs jeux de données de traduction. L'idée du Transformer est de conserver l'interdépendance des mots d'une séquence en n'utilisant pas de réseau récurrent, mais seulement le mécanisme d'**attention** qui est au centre de son architecture et sur lequel on va se baser.

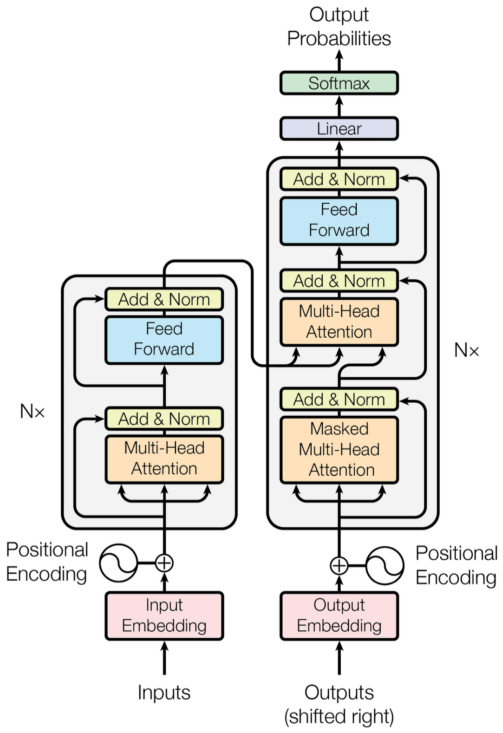


FIGURE 17 – Schéma représenant l'architecture du Transformer.

Source: [77]

Pour l'analyse de la sémantique de la route, seule la partie **«encodeur»** qui représente la partie gauche de la Figure 17 sera utilisée. En effet, le mécanisme d'**attention** sera utilisé pour repérer les erreurs d'identification de marquages, ensuite la **classification** se fera de façon classique avec des couches denses. Les deux principales particularités de cet **encodeur** est le ***Position Embedding***, qui permet de donner une relation d'ordre dans la séquence d'entrée, et la notion de ***self-attention***, qui permet de se centrer sur les mots les plus importants dans une phrase tout en prenant en compte l'inter-dépendance des mots. Ces notions sont expliquées ci-dessous.

Position Embedding

Un point positif des modèles récurrents précédents est qu'ils prennent intrinsèquement en compte l'ordre des mots ; ils analysent un mot lettre par lettre de manière séquentielle. Cependant, ceci augmente considérablement le temps d'apprentissage des modèles. Les **Transformers** ont voulu éviter le mécanisme de récurrence et prend ainsi tous les vecteurs d'***Embedding*** en entrée en parallèle afin

d'augmenter davantage la vitesse de formation du modèle. Comme chaque lettre d'un mot passe simultanément par la pile d'encodeur du **Transformer**, le modèle lui-même n'a aucun sens de la position/ordre de chaque lettre. Par conséquent, il est toujours nécessaire de trouver un moyen d'incorporer l'ordre des marquages dans notre modèle, d'où l'existence de la couche de ***Position Embedding***. On construit un deuxième vecteur pour chaque lettre, appelé vecteur de ***Position Embedding***, et on l'additionne avec le vecteur d' ***Embedding*** de base pour nous garantir l'information de position et l'information des caractéristiques pour chaque lettre.

L'idée recommandée dans l'article [77] est d'utiliser des **fréquences d'ondes** pour coder les informations de position. Les fonctions sinusoïdales impliquées dans le vecteur sont définies de la manière suivante :

$$\begin{cases} PE(pos, 2i+1) & = \cos\left(\frac{pos}{10000^{(2i+1)/d}}\right) \quad \text{pour les indices impairs} \\ PE(pos, 2i) & = \sin\left(\frac{pos}{10000^{2i/d}}\right) \quad \text{pour les indices pairs} \end{cases} \tag{10}$$

- d étant la dimension fixée pour la taille de ***Position Embedding*** (la même que pour le vecteur d'***Embedding*** afin de donner un sens à la somme finale)
- i l'indice dans le vecteur de ***Position Embedding***, i varie entre 0 et $d-1$
- pos la position de la lettre dans le mot, pos varie entre 0 et $N-1$ où N est la taille du mot

La fonction est bornée entre 1 et -1. De ce fait, il n'y a pas de problèmes au niveau de la perte des informations sur les caractéristiques. De plus, la fonction ne dépend pas de la longueur du mot. Le problème de localisation résolu, on parle ensuite du cœur de lu modèle, le **mécanisme d'attention**.

Mécanisme d'attention (*self attention*)

Dans une phrase, on porte souvent notre attention sur des mots plus importants que d'autres. De même, sur la route, on porte aussi beaucoup plus notre attention sur les marquages, grâce à leur rétro-réflexion et leur couleur. Le mécanisme d'attention présent dans le modèle l'aide à se concentrer sur les mots importants dans une phrase d'entrée. Leur signification est identifiée à travers le contexte dans lequel le mot est utilisé. La méthode utilisée dans l'article [77] est appelée ***self-attention***. Elle se différencie de l'attention classique, qui elle se précise uniquement selon l'importance des mots, c'est-à-dire, plus le mot est important, plus il détermine la réponse de la requête. La ***self-attention*** quant à elle prend aussi en compte la relation entre les mots d'une même phrase. Cette méthode est utilisée dans la couche de ***Multi-Head Attention*** (cf. Figure 18).

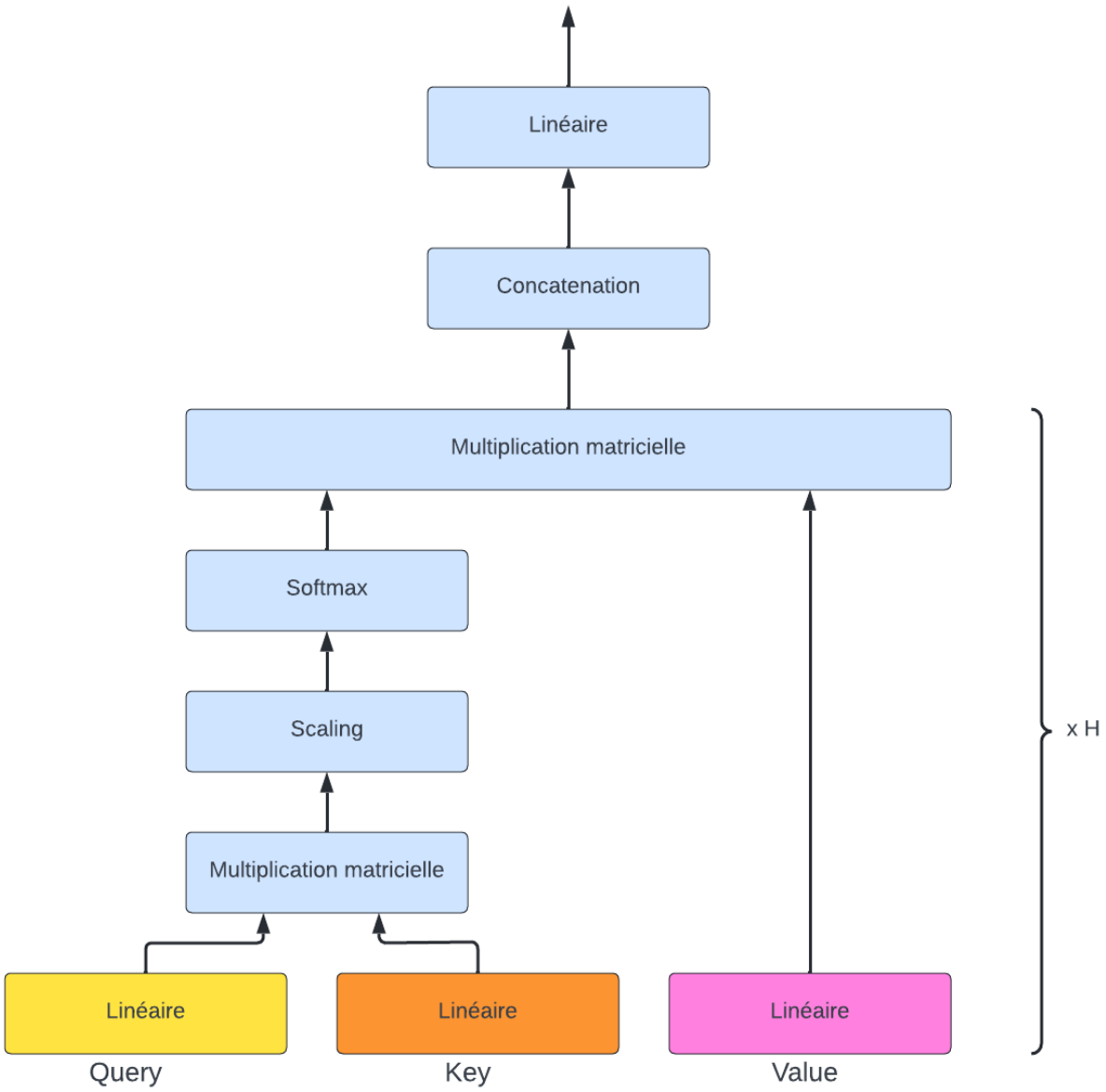


FIGURE 18 – Visualisation du bloc ***Multi-Head Attention***.

La somme des deux **Embedding** est donnée en entrée du bloc ***Multi-Head Attention*** et trois copies sont faites de cette matrice. Tout d'abord, des couches linéaires sont appliquées à chacun de ces copies afin de diminuer la dimension de la matrice pour diminuer le coût d'opérations. Pour chaque copie, on introduit donc une matrice de poids et un biais, et on applique une transformation affine à chaque copie. Ceci rend, l'opération de self-attention plus flexible puisqu'on introduit des paramètres au modèle qui seront mis à jour au fur et à mesure des itérations. Les trois matrices résultantes sont appelées Q pour **Query**, K pour **Key** et V pour **Value**. Ensuite, nous allons calculer ce qu'on appelle la similarité entre Q et K. En effet, il faut voir Q comme une question, K un ensemble de proposition

de réponses à cette question et V le contenu des réponses. Ainsi, pour avoir des bonnes propositions correspondant à la question, il faut étudier la similarité Q et K. Cette similarité se base sur le produit scalaire euclidien de deux vecteurs dans $\mathbf{R}^n$ qu'on peut définir par le cosinus de l'angle formé par les deux vecteurs à un facteur de proportionnalité près :

$$\cos(\hat{xy}) = \frac{\langle x, y \rangle_{\mathbf{R}^n}}{\|x\|_{\mathbf{R}^n} . \|y\|_{\mathbf{R}^n}} \tag{11}$$

Deux vecteurs pointent dans la même direction si le cosinus vaut 1 et inversement dans des sens opposés si la valeur vaut -1. On définit ainsi la similarité entre Q et K de la manière suivante :

$$similarity(Q, K) = \frac{QK^T}{scaling} \tag{12}$$

Q et K étant des matrices, le produit scalaire représente le produit matriciel entre Q et la transposée de K. Cette multiplication est appelée **filtre d'attention**. Au début, puisque les poids associés à ces matrices sont aléatoires, le produit le sera aussi, mais au fur et à mesure de l'apprentissage, chaque coordonnée devient un score d'attention. Ensuite, le filtre est normalisé par un certain nombre. Plus la dimension du vecteur d'entrée est grande, plus la valeur moyenne du produit scalaire augmente. Cette croissance est une fonction de $\sqrt{d_k}$, la dimension de chaque vecteur contenu dans K. Ceci va maintenir les poids dans une certaine plage et on souffrira moins avec la **disparition du gradient**. Ensuite, l'opération ***softmax*** est appliquée à chaque vecteur de la matrice résultante afin d'avoir une distribution entre 0 et 1, où les composantes peu importantes seront presque nulles et la plus importante proche de 1, le **filtre d'attention** final sera obtenue. La prochaine étape est de multiplier le résultat avec la matrice V, le tenseur résultant étant un filtrage de V. Intuitivement, un conducteur regarde la route et les marquages en globalité et non bout à bout pour éviter un accident. Le champ d'attention est donc filtré afin de mettre de côté toutes les choses non nécessaires. C'est le même principe lors de la multiplication des données initiales V (puisque ce n'est qu'une copie de la matrice d'**Embedding** avec des poids) avec le **filtre d'attention**. Le produit d'attention se définit comme suit :

$$Attention(Q, K, V) = softmax\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V \tag{13}$$

Ce mécanisme d'attentions se trouve dans la couche **Multi-Head Attention** [77]. Dans le Transformer, le **mécanisme d'attention** répète plusieurs fois ses calculs en parallèle (cf. Figure 18). Chacun d'entre eux est appelé une tête d'attention. Le module d'attention divise ses paramètres **Query**, **Key** et **Value** en H parties et fait passer chaque partie indépendamment par une tête distincte. Tous ces calculs d'attention similaires sont ensuite combinés pour produire un score d'attention final. C'est ce qu'on appelle l'attention multi-têtes et cela donne au transformateur une plus grande puissance pour encoder de multiples relations et nuances pour chaque phrase. Une concaténation de toutes les H **Values filtrées** est faite, puis une couche linéaire est appliquée afin d'éviter d'avoir une matrice trop longue.

La sortie de la couche **Multi-Head Attention** est additionnée par la somme des deux vecteurs d'***Embedding*** pour ne pas perdre des informations initiales puis est standardisé de façon normale afin d'éviter des problèmes de **disparition de gradient** [77], on parle de connexion résiduelle.

Des couches **Dense** peuvent être ajoutées ensuite afin d'avoir un modèle de **classification** pour l'étude.

4 Mise en œuvre et résultats

Cette partie présente la mise en œuvre des techniques d'apprentissage supervisé réalisée dans le cadre de mon *stage* de M2 pour l'analyse automatique des marquages routier. La première étape a consisté à mettre en forme les données. Puis, un certain nombre de modifications ont été apportées sur le logiciel d'annotation. Enfin, les modèles de **segmentation** et d'**analyse sémantique**s seront décrits et leurs performances évaluées.

4.1 Préparation des données et logiciel d'annotation

L'équipe **ENDSUM** du **Cerema** dispose de nombreuses sessions de mesure **Ecodyn3** réalisées en propre ou fournies par Nextroad, fabricant de l'appareil. À mon arrivée, certaines d'entre elles avaient déjà été annotées dans le cadre du *stage* de master précédant **[18]** et du projet **SAM** **[14]**.

4.1.1 Acquisition et annotation des données

La première étape du travail a consisté à mettre en place une gestion des données avec un fichier récapitulatif des données disponibles et des tâches réalisées (dans `Recapitulatif_annotation.xls` (cf. Figure **19**)).

Dossier	Coté mesure	Données	Distance parcourue (m)	Annotations	Date	Fait par	Tête	Infos
A352	Axe	A35_+_PR411_PR430_Axe_170315_105453	13954,8	Fait	15/03/2017	Cerema	6	valeurs jours trop grandes
	Axe	17000016_A352_-_PR10_PR2_Axe	7217,25	Fait	24/03/2017	Cerema	6	valeurs jours trop grandes
	Axe	18000421_A9X3S1_+_PR192_PR195_AXE3	3031,59	Fait	28/09/2018	ASF	Hors cerema	Des déchets sur les côtés
	Axe	20000028_A352_-_PR5_PR0_Axe	1729,6	Fait	23/07/2020	Cerema	6	valeurs jours trop faibles
	Axe	20000032_D207_+_PR7_PR13_Axe_AXE_AGTC	4518,395	Fait	10/09/2020	Cerema	12	RAS
	Axe	20000034_D207_P2_+_PR7_PR13_Axe_AXE_AGTC	4505,625	Fait	10/09/2020	Cerema	12	RAS
	Axe	20000037_D207_P3_+_PR7_PR13_Axe_AXE_AGTC	4575,56	Fait	10/09/2020	Cerema	12	RAS
RD500	Axe	20000042_RD500_P1_+_PR0_PR7_Axe	6896,7	Fait	10/09/2020	Cerema	12	RAS, T1, T3
	Axe	000043_RD500_P1_-_PR7_PR0_Axe_AXE_AGTC,json_n	5983,18	Fait	10/09/2020	Cerema	12	RAS, T1, T3, LC
	Axe	20000048_RD500_P3_+_PR0_PR7_Axe	6903,1	Fait	10/09/2020	Cerema	12	RAS, T1, T3
	Axe	20000050_RD500_P4_+_PR0_PR7_Axe	6890,4	Fait	10/09/2020	Cerema	12	RAS, T1, T3
	Axe	20000052_RD500_P5_+_PR0_PR7_Axe	6914,4	Fait	10/09/2020	Cerema	12	RAS, T1, T3
	Axe	20000043_RD500_P1_+_PR0_PR7_Axe	5983,6	Fait	10/09/2020	Cerema	12	RAS, T1, T3
	Axe	20000049_RD500_P3_+_PR0_PR7_Axe	5977,2	Fait	10/09/2020	Cerema	12	RAS, T1, T3

FIGURE 19 – Visualisation du tri des données dans le fichier `Recapitulatif_annotation.xls`.

J'ai eu l'opportunité d'être opérateur pour collecter de nouvelles mesures, en particulier l'**A35** ou des routes départementales : **D41**, **D228**. C'est pourquoi, j'ai dû m'approprier le logiciel afin d'annoter moi-même des sessions. Comme l'annotation se faisait pixel par pixel avec un outil de remplissage sans aucune images d'environnement, une session autoroute de $8km$ de **A35** peut prendre une journée (cf. Figure **20**). J'ai réalisé les annotations de ces nouvelles sessions et d'autres avec le logiciel **Vérité Terrain Ecodyn**.

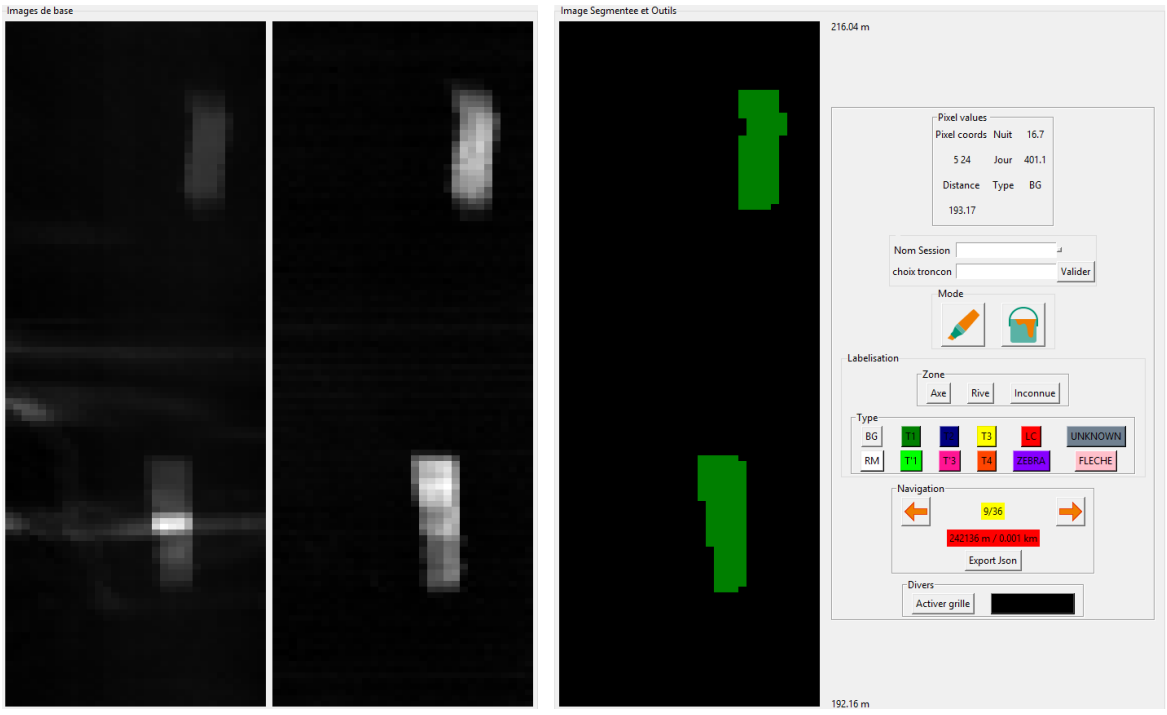


FIGURE 20 – Visualisation de l'interface de la première version de **Vérité Terrain Ecodyn**.

Cette annotation m'a permis d'identifier des ***bugs*** et limites, ce qui m'a amené durant tout le *stage* à me plonger dans le code et effectuer des modifications dans le logiciel.

4.1.2 Modifications de Vérité Terrain Ecodyn

Les modifications apportées sur **Vérité Terrain Ecodyn** sont à la fois ergonomiques, techniques et en lien avec le ***deep learning***.

Modifications ergonomiques

Des informations sur la route ont été ajoutées afin d'améliorer la compréhension de la route et des données. Par exemple, les images prises par la caméra lors de la mesure ont été ajoutées dans l'interface. Elles permettent d'avoir une visualisation de la route et de son environnement, ce qui est une aide à décision lors de l'annotation. Les modifications apportées sont les suivantes :

- Ajout des étiquettes **«Rétroréflexion»** et **«Luminance»** au-dessus des signaux associées à titre informatif.
- Ajout du nom du fichier, de la localisation de la mesure ainsi que son extension sur l'interface.
- Ajout de photos de la caméra sur un tronçon (cf. Figure 22). Ajout du bouton **décalage d'images** pour pouvoir décaler une ou plusieurs images vers l'avant. Comme la caméra est fixée par ventouse sur le toit, sa position peut différer d'une session à l'autre. Le décalage permet de faire correspondre la prise de vue aux mesures affichées.
- Changement d'image en lien avec la section de mesure affichée.

- Ajout d'un schéma itinéraire permettant d'avoir une visualisation globale d'une section de route (cf. **Annexe** Figure 47).

Modifications techniques

Certains *bugs* au sein du logiciel ont été détectés et corrigés et ont permis d'accélérer l'annotation. Les modifications apportées sont les suivantes :

- Mise au propre de la convention de nommage des attributs des fichiers. Des attributs d'anciennes données avaient été mal renseignés. Par exemple, des problèmes associés au type **int32** des données pour l'export `.json` ont été résolus. Le logiciel peut désormais ouvrir tous les fichiers `.db`, en particulier les anciennes annotations, sauf certains avec des données **GPS** non conforme.
- Le bouton «**Tout Remplir**» ne fonctionnait pas. Il permet maintenant de remplir toutes les composantes connexes d'un tronçon en une couleur choisie. Ceci optimise le temps d'annotation.
- Ajout de nouveaux types : **PIETON** et **T'2**, afin d'avoir tous les marquages possibles à dispositions [7].
- Standardisation du nom de fichier suite à l'annotation pour garder l'info de localisation. Lors de l'export `.json` d'un fichier `.db` bitête, le suffixe du nom initial `_Axe_Rive` sera transformé en `_Axe` ou `_Rive` selon le cas.
- À noter qu'avant **Vérité Terrain Ecodyn**, il existait un autre logiciel d'annotations, appelé **AGTC**. Certaines routes ont été annotées avec cet ancien outil, mais le format de sortie de celui-ci était un peu différente du format utilisé par **Vérité Terrain Ecodyn**. Le script `AGTC_to_VTE.py` permet de passer d'un format à l'autre ce qui a permis d'éviter d'annoter à nouveau des routes déjà faites.

Modifications en lien avec le *deep learning*

L'annotation étant une tâche fastidieuse, longue, l'objectif est d'avoir un logiciel capable de la faire automatiquement et d'analyser la sémantique de la route afin d'en repérer les erreurs. Ainsi, les modèles de ***deep learning***, décrits dans les sections suivantes, ont été incorporés dans le logiciel.

En plus d'ouvrir des données brutes, il permet de visualiser l'annotation réalisée par le réseau de **segmentation**. La tâche de l'opérateur est alors de corriger l'annotation, ce qui est bien plus rapide. Le modèle de **segmentation** s'active via un bouton `Segmentation` sur l'interface afin de faire un nouveau fichier `.json` annoté et classifié par un algorithme de **classification** basé sur la géométrie des marquages.

L'**analyse sémantique** se traduit à travers le schéma itinéraire. Un bouton `Analyse Sémantique` a été ajouté dans **Vérité Terrain Ecodyn**. Il est utilisable uniquement lorsque la **segmentation** et l'**identification** ont été réalisées. Lorsqu'il est actionné, les composante connexes au sein de la route seront parcourue et stockées en séquence de 16 éléments. L'attribut `Detected`, initialement vide, sera complété par des nouveaux dictionnaires où chaque dictionnaire correspond à un marquage détecté au sein de l'itinéraire. Chaque dictionnaire `Marquage` comporte le type du marquage, sa position par rapport à son barycentre au sein de la route et un attribut `OK`. Le modèle, intégré dans le logiciel, fera une inférence sur chaque séquence et tous les marquages au sein d'une séquence seront jugés faux ou vrai via l'attribut : `OK`, 1 pour faux et 0 pour correct (cf. **Annexe** Listing B). Cela permet ainsi d'identifier les erreurs d'**identification**, de **segmentation** et aussi des erreurs liées à la route. La visualisation graphique des erreurs sur l'interface n'a pas été fait par manque de temps.

Les fonctionnalités actuelles du logiciel **Vérité Terrain Ecodyn** sont représentées sur la Figure 21.

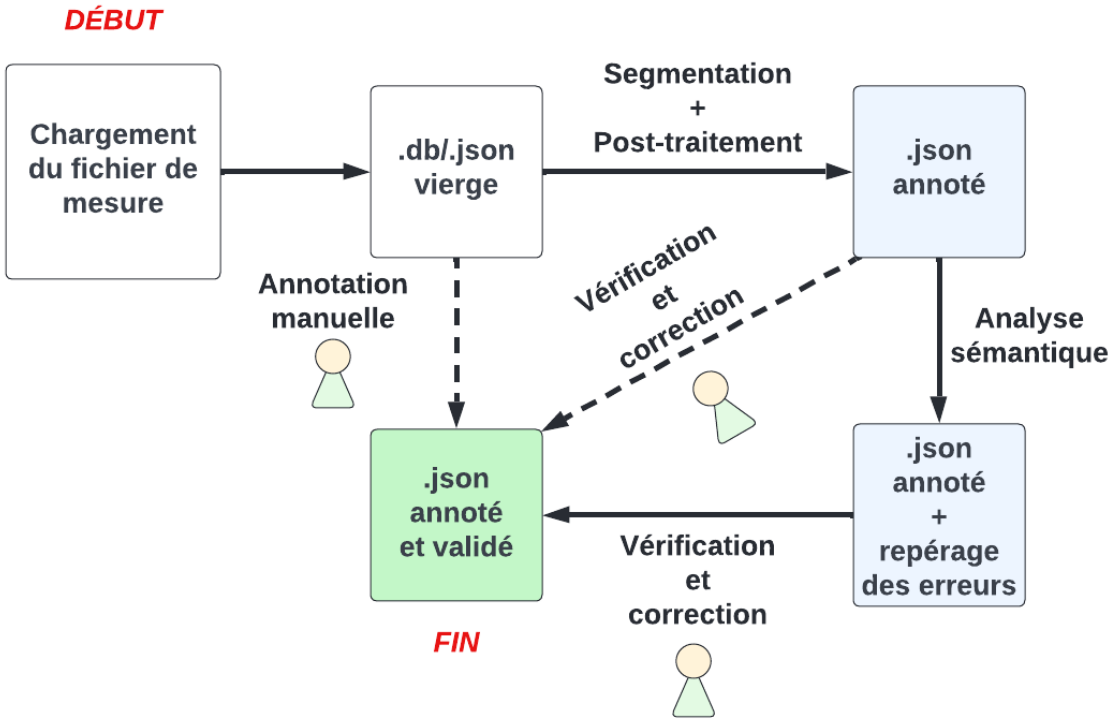


FIGURE 21 – Schéma représentant les fonctionnalités de **Vérité Terrain Ecodyn**.

Visualisation de la nouvelle version de Vérité Terrain Ecodyn

La Figure 22 présente l'interface modifiée de **Vérité Terrain Ecodyn**. Dans le tronçon associé au signal de **luminance** (tout à gauche) sur le marquage tout en bas, une partie plus lumineuse qu'au reste est visible. À l'aide de l'ajout des photos d'environnements, on déduit que cela provient de la lumière du soleil et que le reste du marquage est sous ombre. La notice du logiciel a été actualisée suite aux évolutions.

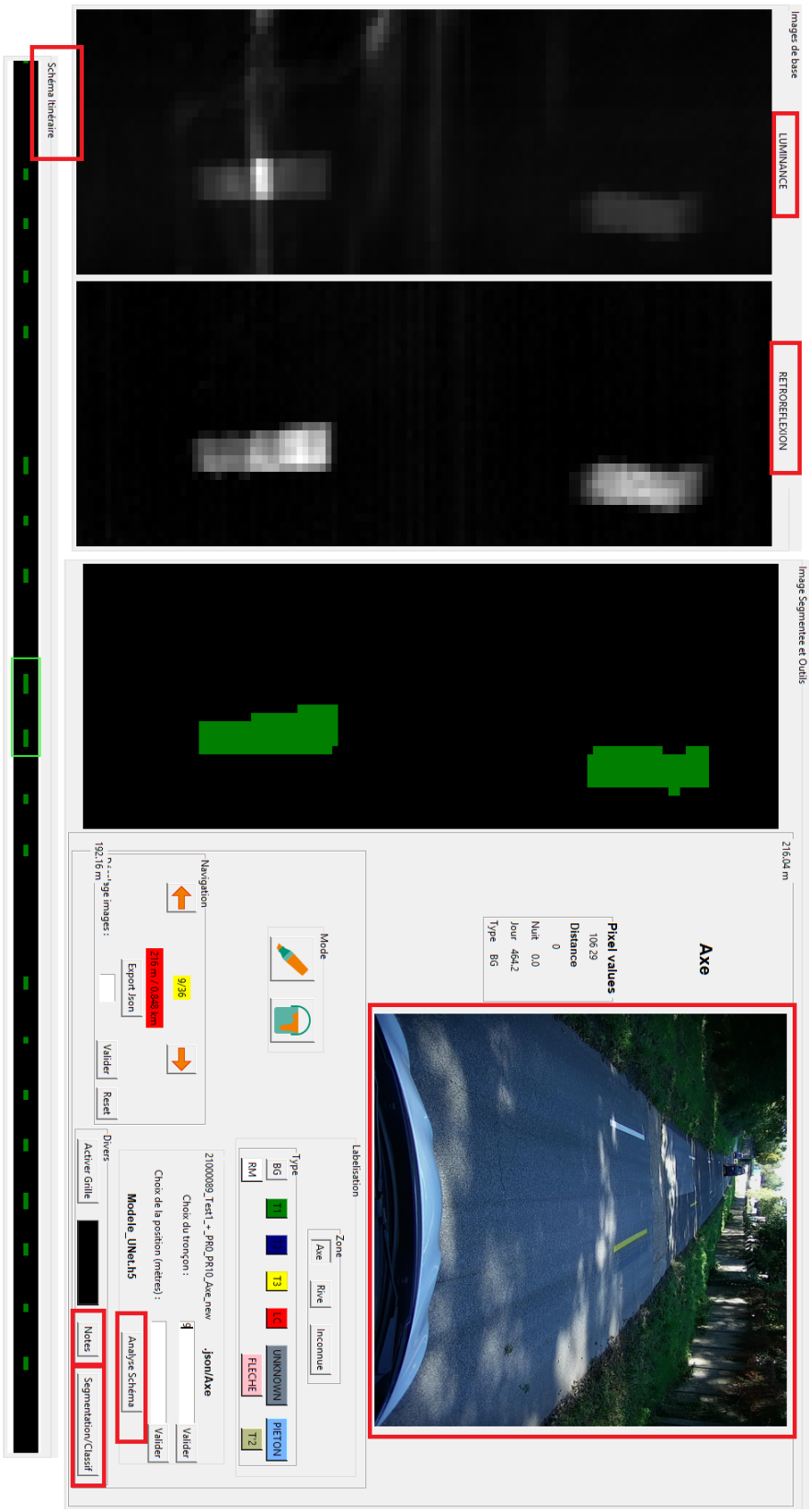


FIGURE 22 – Visualisation de l'interface de la nouvelle version de **Vérité Terrain Ecodyn**. Les modifications sont encadrées en rouge.

4.1.3 Création des données d'apprentissages/validations et de tests

Afin d'évaluer la qualité du modèle final à l'aide de nouvelles données, le jeu de données a été séparé en deux parties distinctes, une base d'**entraînement** et une base **test**. Le choix des routes utilisées pour ces deux bases s'est faite par simple visualisation sur **Vérité Terrain Ecodyn** afin d'avoir des données variées dans les deux bases. Des données de mauvaises qualités (cf. Figure 23) ont été introduites dans les deux bases et en particulier, dans la base **test** afin d'évaluer la robustesse du modèle. En outre, des routes avec des forte intensité de lumière dues au soleil (le bout blanc de l'image à droite dans la Figure 23) ont été utilisés pour la base d'**entraînement** afin que le modèle apprenne qu'il ne doit pas les classifier en tant que marquage.

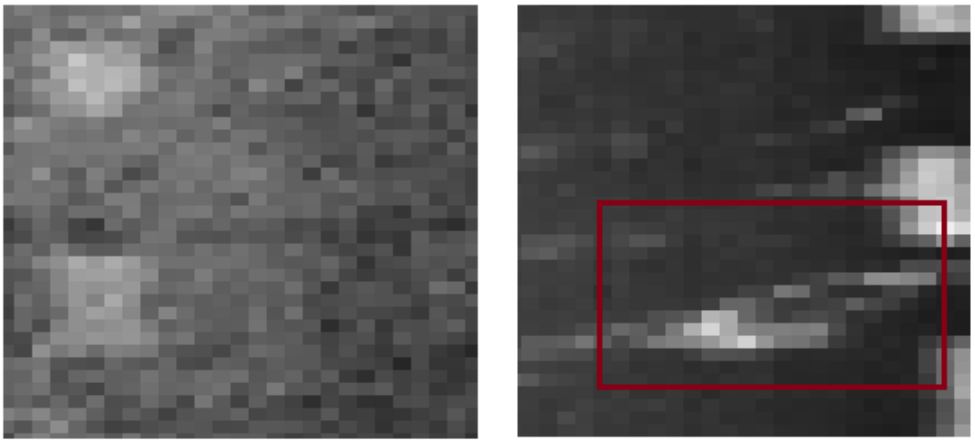


FIGURE 23 – Visualisation de marquages en mauvaises qualités (à gauche) et de la présence de soleil (à droite).

En fin de *stage*, la base d'**entraînement** comporte environ $220km$ de route, et la base **test** $50km$. Pour l'entraînement, la base d'**entraînement** est coupée en deux afin d'avoir une proportion d'environ 80%/20% de façon aléatoire et disjointe. La petite base sera la base de **validation**. Une ***Loss*** de **validation**, avec la même fonction de coût que pour l'entraînement, sera calculé au fil de la formation du modèle. Ces trois bases sont donc **disjointes**.

4.1.4 Mise en forme des données

Une fois les données annotées, validées et exportées en fichier `.json`, les données brutes sont chargées dans des tableaux représentant la concaténation des «images» **luminance** et **rétroréflexion** pour créer une seule image avec deux canaux (au lieu de trois comme les images **RGB** classiquement utilisées dans les réseaux **UNet**). Pour la **segmentation** bi-classe, la partie d'annotation des données brutes, aussi appelées ***labels***, est stockée dans un tableau binaire, qui ici est une image en noir et blanc avec un canal, avec des coefficients 1, représentant un pixel *marquage*, et 0, représentant un pixel *arrière-plan*. Puis les deux images sont découpées en **patchs**, c'est-à-dire en plusieurs rectangles de taille **L**×32 où **L** est à choisir et 32 correspond au nombre de voies. Au lieu d'entrer toute la route en une seule fois, le découpage en **patchs**, permet d'utiliser une taille de lot (**batch**) assez grande afin d'assurer une robustesse dans la descente du gradient [79]. Les données brutes et annotées, tous les deux découpées,

sont ainsi rentrés dans le modèle. Pour les entraînements, **L** vaudra 32 afin d'avoir des **patchs** carrés, comme le montre la figure **24**.

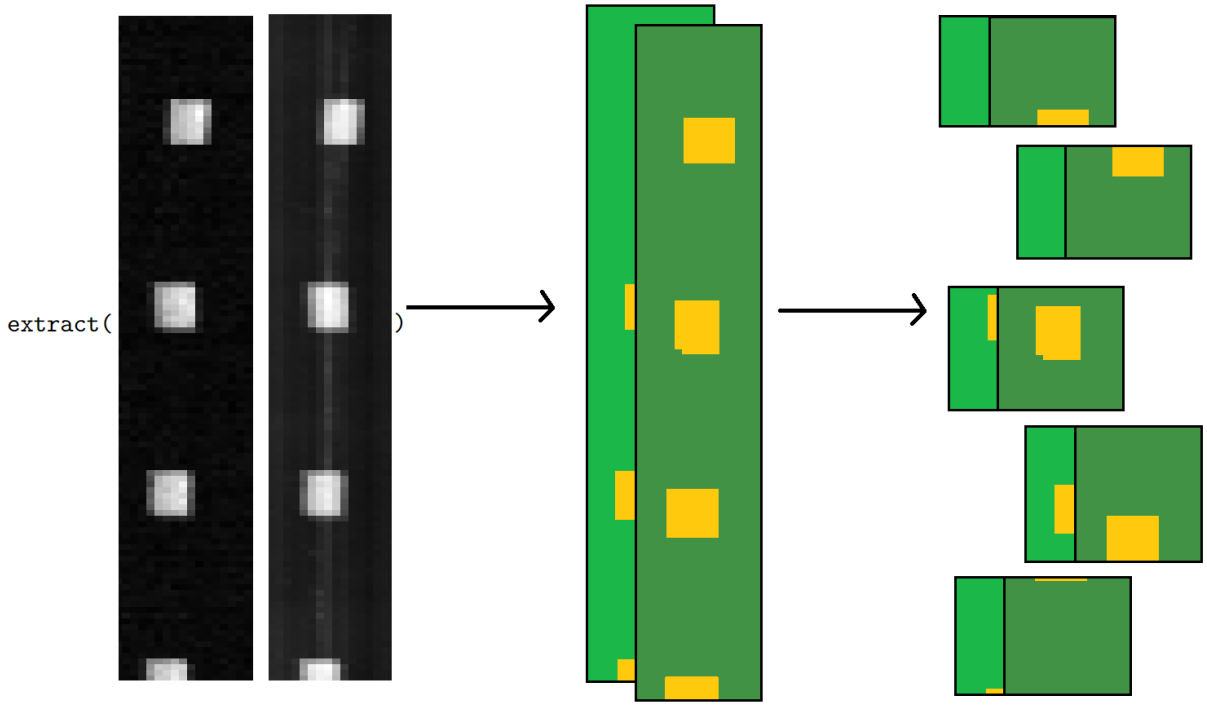


FIGURE 24 – Visualisation du découpage des ***labels*** pour l'apprentissage.

La Table 1 montre la proportion de **patchs** au sein d'une base.

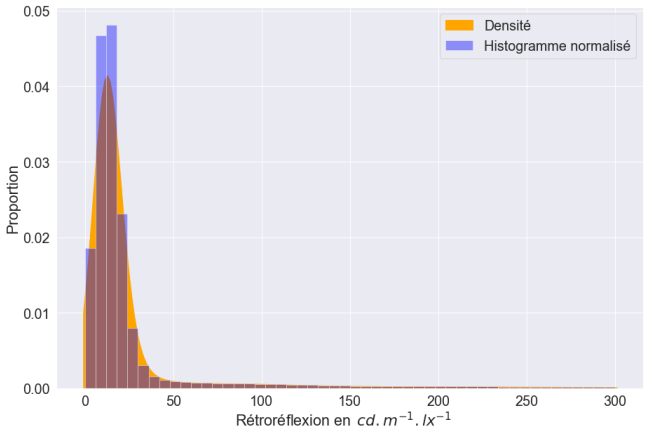
	Base d'**entrainement**	Base **test**
Taille ***(km)***	218.37	48.71
Nombre de ***patchs***	15173	5001

TABLE 1 – Tableau représentant la proportion de **patchs** dans les deux bases.

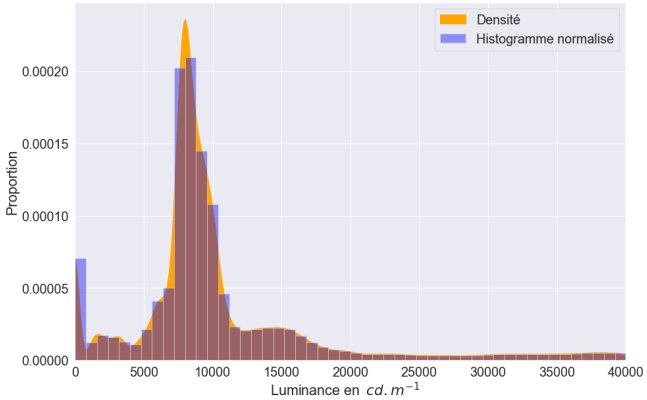
Normalisation des données

En pratique, il est presque toujours avantageux d'appliquer des transformations de pré-traitement aux données d'entrée avant de les présenter à un réseau **[80]**. Les différences d'échelles entre les variables d'entrées peuvent augmenter la difficulté du problème modélisé. Par exemple, de grandes valeurs d'entrée (comme un écart de mille unités entre les valeurs de **rétroréflexion** et de **luminance**) peuvent donner lieu à un modèle qui apprend de grandes valeurs de poids. Un modèle avec de grandes valeurs de poids est souvent instable, ce qui signifie qu'il peut souffrir de mauvaises performances pendant l'apprentissage et d'une sensibilité aux valeurs d'entrée, ce qui entraîne une erreur de généralisation plus élevée. Les valeurs de **rétroréflexions** dans la base de donnée varient entre 0 et $2500 mcd.m^{-2}.lx^{-1}$ (cf. Figure **25a** et Table 2). Tandis que la valeur de la **luminance** varie entre 0 et un peu plus de $100000 mcd.m^{-2}$ (cf. Figure **25b** et Table 2). Les densités de probabilités associées aux variables aléatoires qui décrivent

les valeurs de **luminance** et de **rétroréflexion** (cf. Figure **25a** et **25b** les courbes en orange) ont été estimées par la méthode d'estimation par noyau **[81]** à l'aide de la fonction `KernelDensity` de `Scikit-Learn` **[82]**.



(a) **Rétroréflexion**



(b) **Luminance**

FIGURE 25 – Visualisation des histogrammes normalisés en bleu et de la densité de probabilité associée en orange pour la **rétroréflexion** et la **luminance**.

Arrière plan + Marquage		
	Rétroréflexion ($cd.m^{-1}.lx^{-1}$)	**Luminance** ($cd.m^{-1}$)
Max	2 153	110 998
Moyenne	31	18 892

Arrière plan		
	Rétroréflexion ($cd.m^{-1}.lx^{-1}$)	**Luminance** ($cd.m^{-1}$)
Max	1 318	110 998
Moyenne	162	21 296

Marquage		
	Rétroréflexion ($cd.m^{-1}.lx^{-1}$)	**Luminance** ($cd.m^{-1}$)
Max	2 153	110 998
Moyenne	14	18 585

TABLE 2 – Statistiques sur les valeurs de la **rétroréflexion** et **luminance**.

Lors du *stage* précédant **[18]**, il n'y avait pas eu de normalisation des données et le modèle **UNet** ne convergeait pas. Certaines techniques de mise à l'échelle des données, telles que la normalisation et la standardisation, sont les plus populaires **[80]**. Afin de palier à cette différence d'écart de valeurs, une normalisation des données a été faite, patch par patch [xxv] :

$$pixel_{patch} = \frac{pixel_{patch} - \min_{patch}}{\max_{patch} - \min_{patch}}$$

Ainsi, les données prennent des valeurs dans $[0, 1]$.

xxv. La standardisation, qui consiste à soustraire la moyenne des données, puis de diviser par l'écart-type, a été testé, mais n'a pas permis d'avoir un modèle qui converge. Une explication possible est que ces données ne suivent pas une distribution gaussienne **[80]**.

4.2 Segmentation automatique

En premier lieu, les implémentations des modèles **UNet** et **SegNet** et leurs entraînements sont décrits puis les résultats sont présentés afin de conclure sur un modèle adapté pour **Vérité Terrain Ecodyn**.

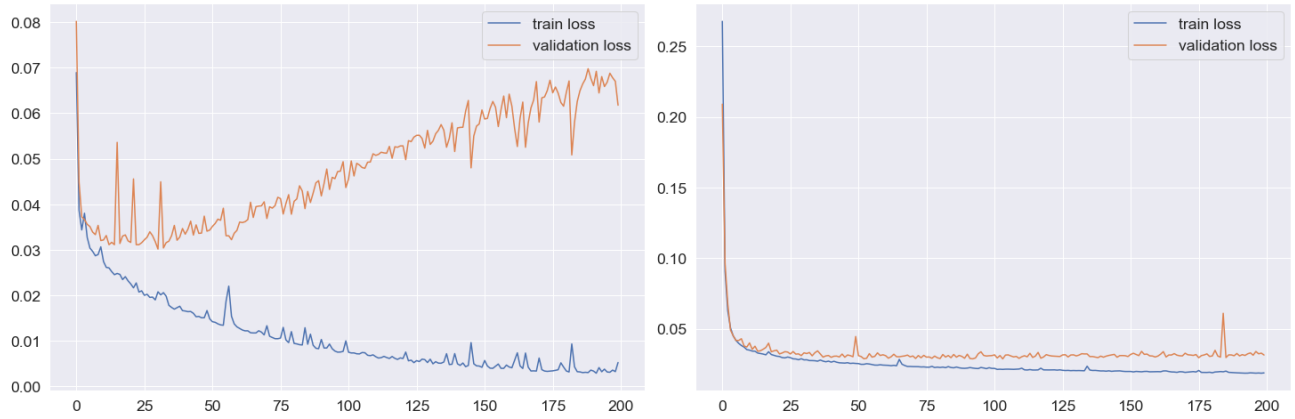
4.2.1 Implémentation et entraînement des modèles

Pour l'entraînement des modèles de **segmentation**, la configuration $type_{seg}$ suivante a été utilisée :

- Taille de ***batch*** = 64,
- ***Loss*** utilisée : ***Cross-Entropy Loss***,
- Nombre d'***epochs*** = 200,
- Optimiseur : ***Adam optimization algorithm(lr = 0.001)*** [xxvi],
- Lors de l'entraînement, le modèle avec les poids/biais qui minimise le mieux la ***Loss*** de **validation** sera retenue et sauvegardée [xxvii].

UNet

Le modèle **UNet** avait été repris et implémenté par d'anciens stagiaires et correspond au réseau à 4 couches original de l'article **[50]**. Ce réseau comporte plus d'un milliard de paramètres entraînables. Les opérations `conv2D`, `MaxPool2D` et `Conv2DTranspose` de la bibliothèque `Tensorflow` pour l'implémentation du modèle sont utilisées. Le phénomène de **sur-apprentissage** apparaît lorsqu'on affiche les ***Loss*** en fonction des ***epochs*** (cf. Figure 26a). En effet, la ***Loss*** de **validation** commence à partir d'un certain temps et s'éloigne de plus en plus de la ***Loss*** d'**entraînement**, qui elle, diminue davantage.



(a) Modèle initial.

(b) Modèle avec ***Dropout***.

FIGURE 26 – Visualisation du phénomène de **sur-apprentissage** avec le modèle initial et le modèle avec l'ajout du ***Dropout***.

xxvi. ***lr*** pour ***learning rate*** qui correspond au pas de la descente de gradient

xxvii. Se fait via des ***callbacks*** qui sont des objets qui permettent d'effectuer des actions à différents stades de la formation, ici la sauvegarde d'un modèle sous une certaine condition

Du ***Dropout*** (cf. 3.1.2) a été ajouté dans l'encodeur et le décodeur du modèle **UNet**. Ceci a permis de réduire le **sur-apprentissage** lors de l'entraînement (cf. Figure **26b**). En effet, la ***Loss*** de **validation** n'augmente plus à partir d'une certaine ***epoch*** et reste au voisinage de la ***Loss*** d'**entraînement**.

L'influence de la diminution du nombre de couches dans le modèle a été testé. Ceci permet de diminuer le **sur-apprentissage**, mais aussi de voir si un grand nombre de paramètres est nécessaire, puisqu'à chaque convolution de nombreux poids/biais sont associées. Pour ce faire, j'ai modifié le code du **UNet** classique afin d'avoir la possibilité de choisir plus facilement le nombre de couches dans la structure de «**U**». Dans l'implémentation, une liste l_i, qui correspond à la structure en couches, est choisie par l'utilisateur. Les coefficients dans les listes correspondent aux tailles de ***features*** associées aux convolutions et la taille de la liste considérée correspond à la profondeur du **UNet**. La correspondance, le temps d'entraînement ainsi que la taille du fichier `.h5` [xxviii] associée est indiquée dans la Table 3.

Liste	**Profondeur**	**Nombre de paramètres**	**Temps d'entraînement pour** 200 ***epochs*****(s)**	**Taille (.h5) (Ko)**
l_0	0	2 786	619	73
l_1	1	26 066	893	390
l_2	2	118 194	1 213	1 514
l_3	3	484 722	1 762	5 844
l_4	4	135 407	2 103	22 996

TABLE 3 – Tableau représentant le nombre de paramètres, le temps d'apprentissage et la taille du fichier `.h5` en fonction de la profondeur du réseau **UNet**.

La liste l_4 correspond à la profondeur et aux ***features*** de la Figure **13**, et la liste l_1 correspond à l'architecture de la Figure **27**. Le code associé se trouve en **Annexe** dans le Listing **2**.

xxviii. À chaque sauvegarde du modèle, un fichier `.h5` est enregistré.

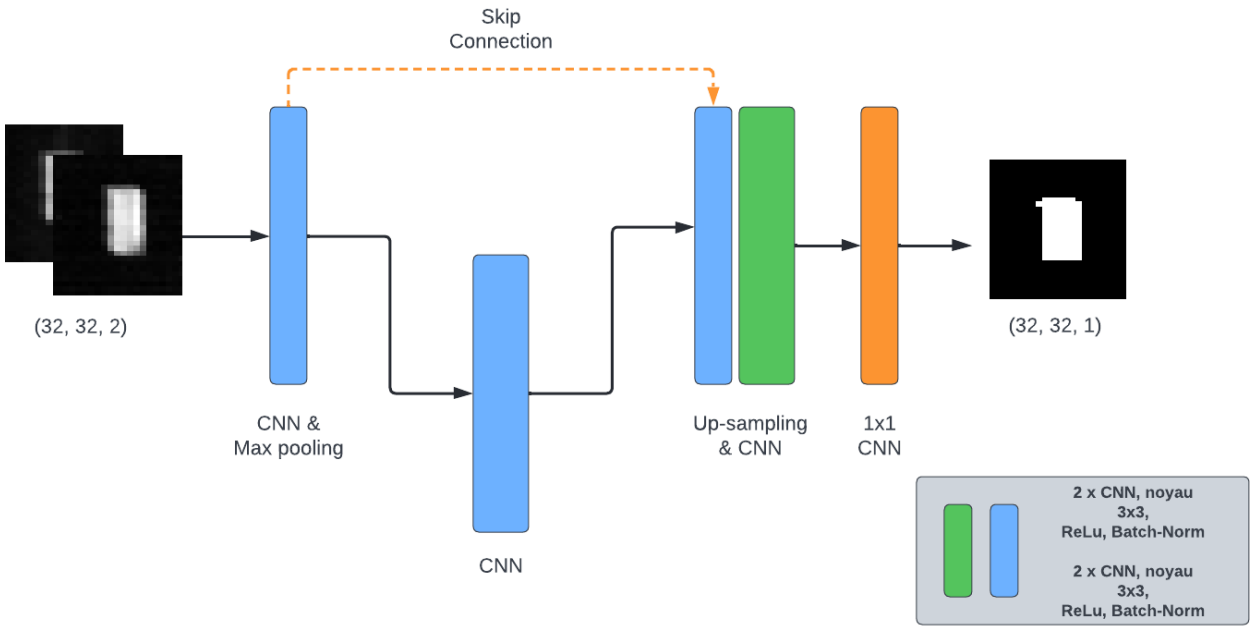


FIGURE 27 – Architecture **UNet** associée à l_1.

SegNet

Le réseau **SegNet** de l'article original [50] a été implémenté et adapté à nos données. Il comporte au total 29 458 442 paramètres et 4 couches en profondeur. Une classe `MaxPoolingWithArgmax2D` a été implémentée afin d'obtenir l'indice du Pooling lors de l'application du max-pooling. De plus, une classe `MaxUnpooling2D`, qui n'est pas dans `Tensorflow`, a été ajouté pour faire le sur-échantillonnage lors de la remontée. Cependant, le modèle n'a pas convergé vers 0 comme le montre la Figure 28 sous la configuration $type_{seg}$. D'autres configurations ont été mis en places :

- Une configuration ($\star$) où le seul changement est l'utilisation de la ***Dice Loss*** [83] au lieu de la ***Cross-Entropy Loss***. Cette ***Loss*** se base sur la similarité entre deux images.
- Une configuration ($\star\star$) identique à $type_{seg}$ mais avec moins de paramètres entraînables en modifiant le nombre de caractéristiques de sorties de chaque couche de convolutions. Le modèle a dans ce cas 7 374 858 paramètres.
- Une configuration ($\star\star\star$) identique à $type_{seg}$ mais avec beaucoup moins de paramètres. Le modèle a dans ce cas 1 844 870 paramètres (presque autant que le **UNet** à profondeur 4).

Les ***Loss*** d'**entraînements** et de **validation** dans chaque configuration sont minorés par 0.35.

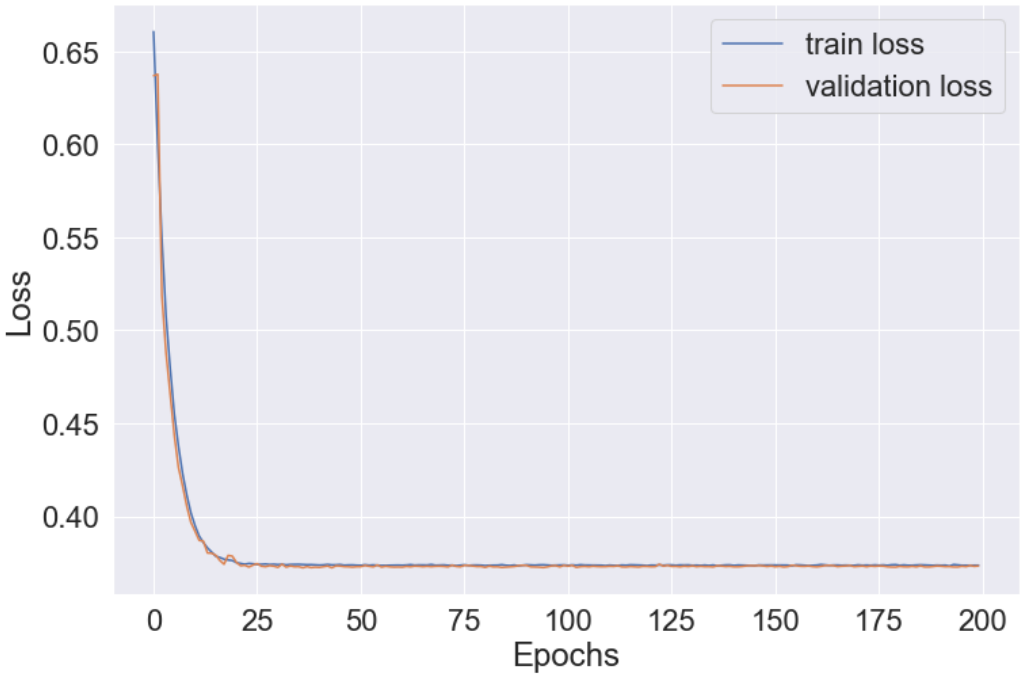


FIGURE 28 – Visualisation des ***Loss*** en fonction des **epochs** pour le modèle **SegNet** sous la configuration $type_{seg}$.

4.2.2 Évaluations et performances du SegNet

Pour l'évaluation de la **segmentation**, les métriques présentées dans la section 3.2 telles que, la **Cross-Entropy Loss**, la **précision**, le **rappel** ainsi que le **score F1** ont été calculées par rapport à la base **test** pour les différentes configurations.

Les performances du modèle **SegNet** sont faibles sur nos données (cf. Table 4) [xxix]. La **précision** pour les modèles avec le plus de paramètres tourne autour de 80%, ceci veut dire qu'il juge à juste titre des pixels dans la classe *marquage*. Cette précision devient très bonne dans les cas (⋆⋆) et (⋆⋆⋆) avec moins de paramètres. Cependant, dans tous les cas le **rappel** reste très faible (inférieur à 10%), donc le réseau oublie énormément de pixels positifs. Les pixels de classe *marquage* sont minoritaires, d'où le fait que la ***Loss*** n'est pas si grande, puisque le modèle a bien classifié la plupart des pixels (classe *arrière-plan*). La ***Loss*** utilisée n'influe pas sur les résultats.

Afin d'obtenir de meilleures performances avec le **SegNet**, il faudrait à la fois modifier l'architecture et disposer de plus de données d'apprentissage. Comme les résultats du modèle **UNet** sont satisfaisantes, cette piste a été abandonnée.

xxix. Pour chaque paramètre, les meilleures performances sont représentées en vert.

	Loss (%)	Précision (%)	Rappel (%)	F1 (%)
Original	**0.2371**	82.05	0.02912	0.07117
(⋆)	0.3479	80.71	0.05151	0,09111
(⋆⋆)	0.2815	**99.97**	7.665	14.23
(⋆ ⋆ ⋆)	0.2814	**99.97**	**7.681**	**14.26**

TABLE 4 – Résultats de l'évaluation du **SegNet** pour quatre configurations différentes.

4.2.3 Évaluations et performances du UNet

Pour l'évaluation de la **segmentation**, les métriques présentées dans la section 3.2 telles que, la **Cross-Entropy Loss**, la **précision**, le **rappel** ainsi que le **score F1** ont été calculées par rapport à la base **test** pour la classe *marquage* pour différentes architectures du **UNet** [xxx]. Le temps d'inférence a aussi été calculé en moyenne sur **GPU** pour une route de 8km :

`22000017_A35_+_PR231_PR223_BAU.json`

	Loss	Précision (%)	Rappel (%)	F1 (%)	Nombre de paramètres	Temps d'inférence (s)
l_0	0.0523	90.91	88.76	90.91	**2786**	**0.0018**
l_1	0.0416	93.17	**92.41**	92.88	26066	0.026
l_2	0.0463	94.23	91.88	92.69	118194	0.034
l_3	**0.0310**	93.22	92.12	92,71	484722	0.039
l_4	0.0506	**94.81**	91.82	**93.21**	1354077	0.061

TABLE 5 – Résultats de l'évaluation de différentes architectures **UNet**.

Les résultats sont obtenus avec un **seuil de confiance optimal**, c'est-à-dire un **seuil** qui permet de maximiser le **score F1**, afin de moins rater de pixels positifs. Pour l'architecture à profondeur 1, le **seuil de confiance optimal** est d'environ 0.39 (cf. Figure **29**). Il a été choisi en calculant plusieurs **précisions** et **rappels** avec des **seuils** différents et celui qui a maximisé le ***F1 score*** a été considéré.

Les résultats sont satisfaisants, en particulier la réduction de paramètres n'a pas considérablement diminué les performances. Le modèle avec une couche rate même moins de pixels de classe *marquage* que les autres puisqu'il a le **rappel** le plus élevé.

xxx. L'***accuracy*** vaut 97% pour chaque architecture et n'a pas été intégrée dans le tableau à cause du déséquilibrage des classes.

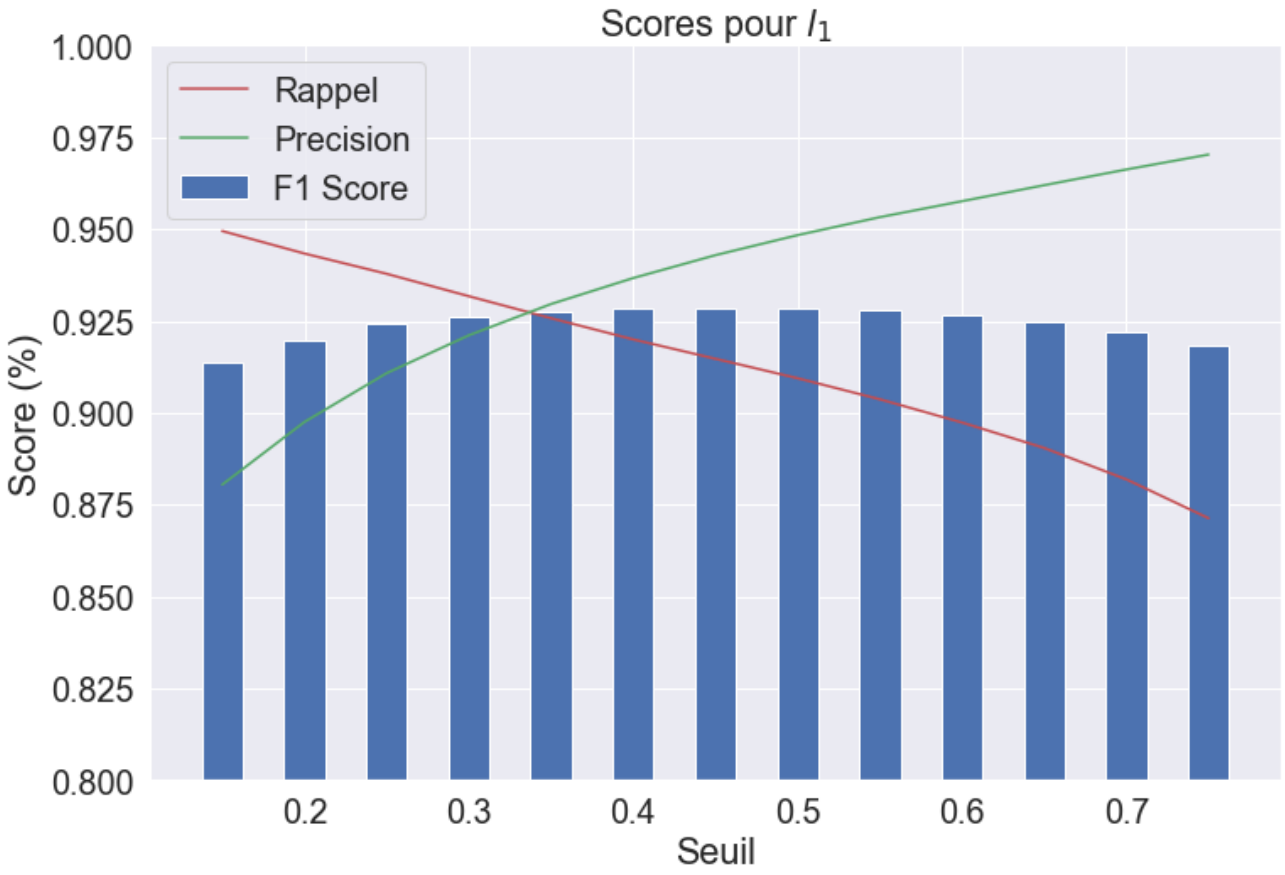


FIGURE 29 – Visualisation de la variation des métriques d'évaluations en fonction de quelques seuils pour l'architecture avec une profondeur de 1. La figure est restreinte sur $[0.15, 0.75]$ en abscisse et $[0.8, 1.0]$ en ordonnée.

La Figure 30 fournit des exemples de la **vérité terrain** et des résultats segmentés (prédits) pour l'architecture avec une profondeur de 1. Comme on peut le voir sur la figure, les résultats prédits sont très similaires à la vérité terrain.

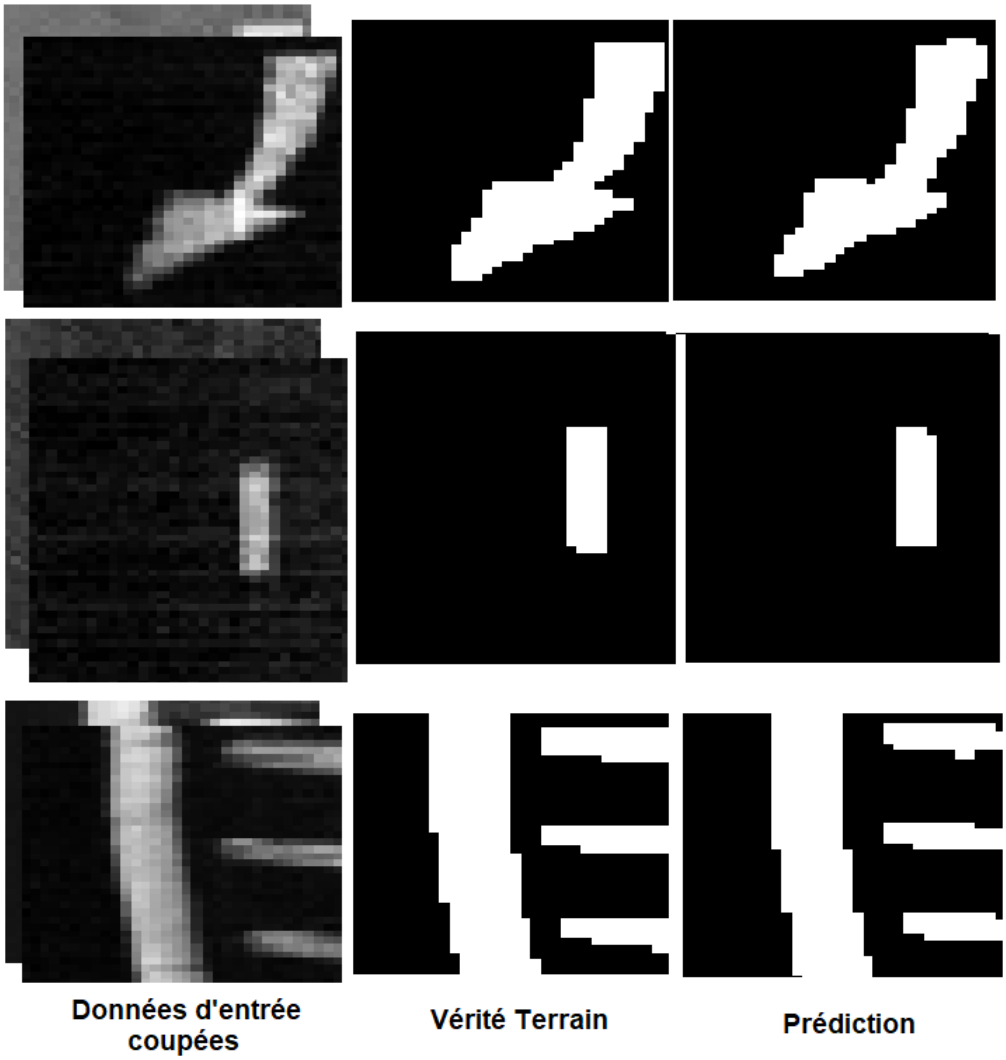


FIGURE 30 – Exemples de **Vérité Terrain** et de Prédiction du modèle **UNet** avec une profondeur de 1.

Cependant, il existe encore certaines failles à la **segmentation** notamment en présence de rayons de soleils (cf. Figure **31**) ou bien des chaussées en mauvais états (cf. Figure **50**).

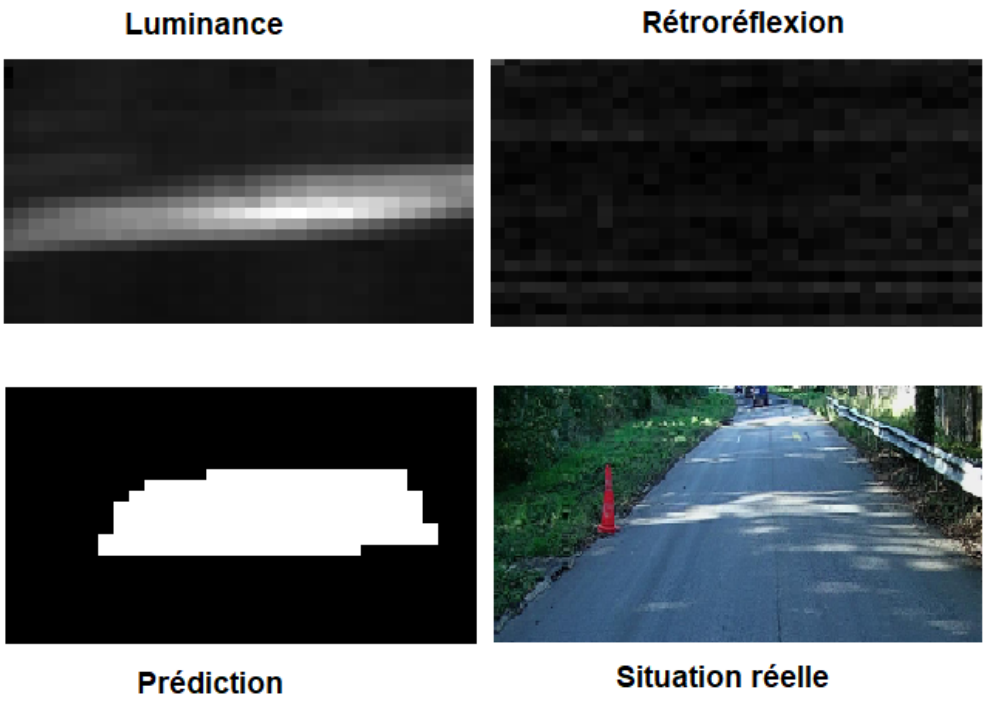


FIGURE 31 – Exemple de mauvaise **segmentation**.

Des pistes d'améliorations seraient d'ajouter des cas négatifs dans la **base d'entraînement**, ou bien de privilégier les données de **rétroréflexions** lors de l'entraînement.

4.2.4 Conclusion sur la segmentation

Choix du modèle pour Vérité Terrain Ecodyn

Le modèle **UNet** à une couche en profondeur a été choisi pour le logiciel de **vérité terrain**. Le modèle est très rapide et admet des performances très satisfaisantes et très proches des autres architectures (cf. Table 5) qui pourraient être améliorés avec davantage de données. Le peu de paramètres à mettre à jour permettra de faire des entraînements beaucoup plus court (cf. Table 3) et d'être moins spécialisé aux données de formations permettant d'être plus flexible lors d'arrivée de nouvelles données. Ceci pousse à faire de l'**apprentissage incrémentiel** et construit un mécanisme de cycle. L'utilisateur n'a plus qu'à lancer la **segmentation** lors de l'ouverture du logiciel d'annotation, corriger les erreurs puis d'entraîner à nouveau le modèle afin d'améliorer les performances.

Apprentissage incrémentiel

Plusieurs bases d'entrainements ont été créés au fur et à mesure de l'étude. L'**apprentissage incrémentiel** a donc été mis en place. La Figure 32 montre l'évolution des métriques d'évaluation pour l'architecture à une couche sur une base **test fixe** en rajoutant des kilomètres de mesures (tous les 20km) à la base d'**entraînement** initiale précédente, soit 7 bases d'**entraînements**. Les performances augmentent au fur et à mesure des ajouts. Cependant, il peut arriver que le modèle obtienne des données

difficiles en entraînement comme à la 6ème itération. De ce fait, la **précision** diminue et par conséquent le ***F1 score*** aussi. Par contre, le **rappel** augmente progressivement petit à petit. On remarque aussi que la **précision** se stabilise à partir d'un certain temps. Ces résultats montre qu'outre un gain de temps majeur lors de l'annotation, l'apprentissage incrémentiel permet d'améliorer les performances du modèle.

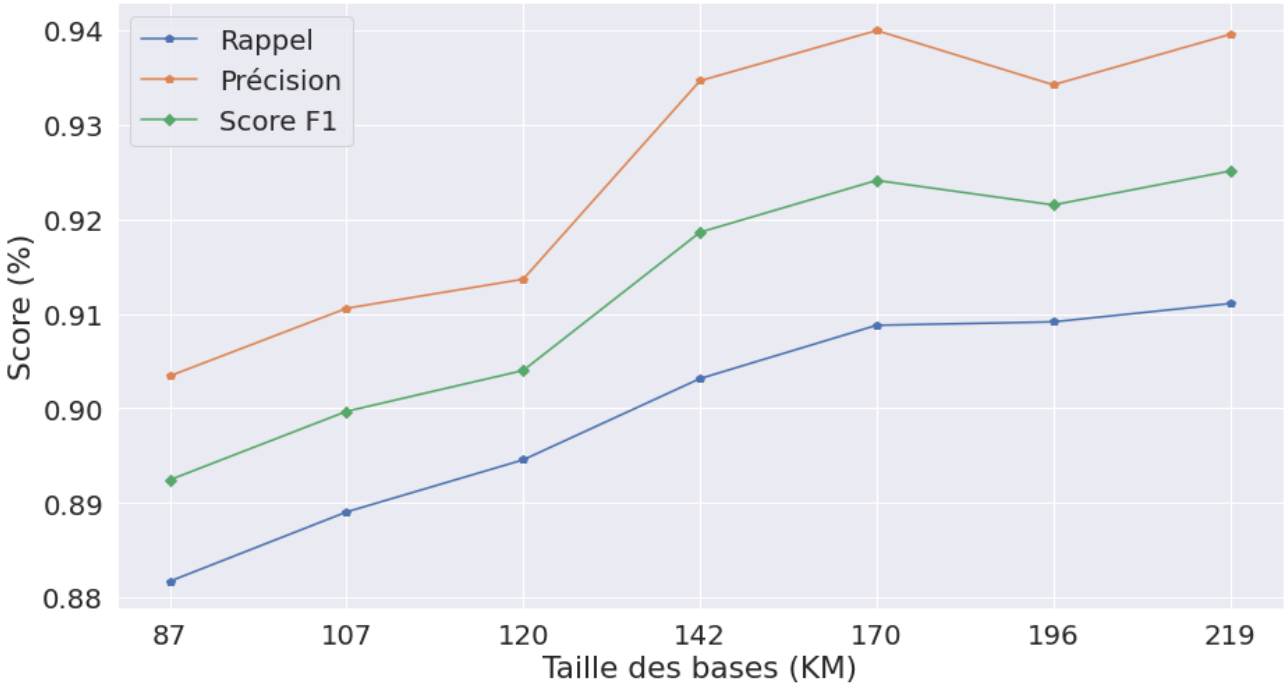
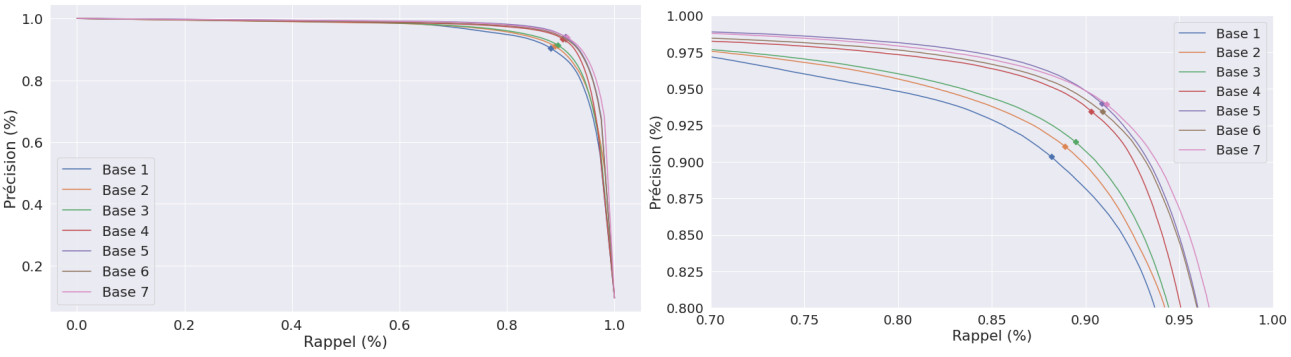


FIGURE 32 – Visualisation de l'**apprentissage incrémentiel** pour le **UNet** avec une profondeur de 1 avec **seuil optimal**. La figure est restreinte sur [0.88, 0.945] en ordonnée.

La courbe **précision/rappel** permet aussi de voir l'évolution. La **précision** et le **rappel** ont été calculés pour différents seuils entre 0 et 1. Les valeurs de **précisions** sont représentées en fonction des valeurs de **rappels** sur la figure **33a**. Les valeurs maximales du ***F1 score*** sont affichées sous forme de carré sur la Figure **33b**. On voit ainsi l'augmentation du ***F1 score*** au fur et à mesure de l'augmentation des bases.



(a) Sans Zoom.

(b) Avec Zoom.

FIGURE 33 – Visualisation de la courbe de **précision** en fonction du **rappel**.

4.3 Identification des marquages

Une fois la **segmentation** faite, l'étape suivante est d'identifier le type des différents marquages. Tout d'abord, un dénombrement des marquages est fait afin d'identifier les types majoritaires et minoritaires. Une approche ***deep learning*** est tout d'abord exposé, puis une approche géométrique est utilisée.

4.3.1 Dénombrement des marquages routiers

Au sein de l'étude, 12 type différent de marquages sont considérés dans toute la base de donnée annotée. Actuellement, les types $\mathtt{T}_1$, $\mathtt{T}_2$, $\mathtt{T}_3$ sont sur-représentés et le reste des types admet une proportion d'apparition en dessous de 5% (cf. Table 6).

Type	Nombre	Pourcentage
$\mathtt{T}_1$	**8 357**	**38.22%**
$\mathtt{T}_2$	**4 546**	**20.79%**
$\mathtt{T}_3$	**6 531**	**29.87%**
$\mathtt{T}_4$	**1 093**	**5.0%**
LC	**356**	**1.62%**
$\mathtt{T}'_3$	**136**	**0.62%**
FLECHE	162	0.74%
ZEBRA	454	2.07%
UNK	152	0.279%
$\mathtt{T}'_1$	0	0.000%
$\mathtt{T}'_2$	6	0.030%
PIETON	61	0.28%

TABLE 6 – Proportions de marquages au sein de tous les données annotées.

Les marquages de type différents se distinguent par leur forme et leur inter-distance. Parmi eux, les marquages rectangulaires se distinguent par leur longueur et leur largeur. L'identification du type peut se faire par une **segmentation multi-classes** qui est la première méthode utilisée dans cette section. Avec cette méthode, les inter-distances ne sont pas considérées.

4.3.2 Segmentation multi-classes

Pour la **segmentation multi-classes**, les pixels seront groupés parmi 13 classes : *arrière-plan* + les 12 types. Au sein de ces classes, la classe *arrière-plan* (noté BG) qui correspond à la route a une majorité écrasante ($\sim$ 97%) comparée aux autres classes ($\sim$ 3%). Comme le modèle **UNet** permet de faire une **segmentation multi-classes**, il a été testé pour l'identification de la sémantique.

Au niveau de l'implémentation, le même code que précédemment a été utilisé avec cette fois-ci une couche de **classification** 13 classes. Chaque pixel de l'annotation est encodé en ***one-hot*** et les données d'entrée seront découpées en **patchs** comme précédemment. Cependant, cette fois-ci, les **patchs** seront rectangulaires d'une longueur assez grande (128) afin d'avoir une vision plus globale des marquages.

La configuration pour l'entraînement multi-classes est identique que pour le bi-classes et l'architecture avec profondeur 4 sera utilisée.

Évaluation et performances

Pour l'évaluation de la **segmentation**, les métriques présentées dans la section 3.2 telles que, la **Cross-Entropy Loss** multi-classes, la **précision**, le **rappel** ainsi que le **score F1** ont été calculées par rapport à la base **test** pour l'architecture du **UNet multi-classes** avec une profondeur de 4.

Classe	Précision (%)	Rappel (%)	F1 (%)
BG	99.10	99.53	99.31
T_1	93.01	56.56	70.34
T_2	35.30	74.50	47, 90
T_3	100.0	0	0
T_4	67.12	70.15	68.60
LC	75.95	81.56	78.65
T'_3	75.16	62.21	68.07
FLECHE	77.05	87.06	81.75
ZEBRA	64.90	77.16	70.50
UNK	78.54	60.53	68.37
T'_1	100.0	100.0	100.
T'_2	30.47	44.41	36.14
PIETON	25.33	19.64	22.12

TABLE 7 – Évaluation du modèle **UNet multi-classes**.

La valeur de la ***Loss*** multi-classes étant égale à 0.213 avec la **base test** initial. Les performances sont hétérogènes et mauvaises pour des classes sur-représentées, par exemple pour la classe T_2 ou la classe T_4 (cf. Figure **7**). Les problèmes notables à cette **segmentation multi-classes** sont :

- Des pixels classifiés dans la classe BG alors qu'ils sont associés à du marquage (cf. Figure **34a**),
- Une mauvaise identification de classe. Par exemple, sur la Figure **34b** , il associe la classe ZEBRA à un marquage de type FLECHE,
- La présence de différentes classes de pixels au sein d'une même composante connexe (cf. Figure **34a**).

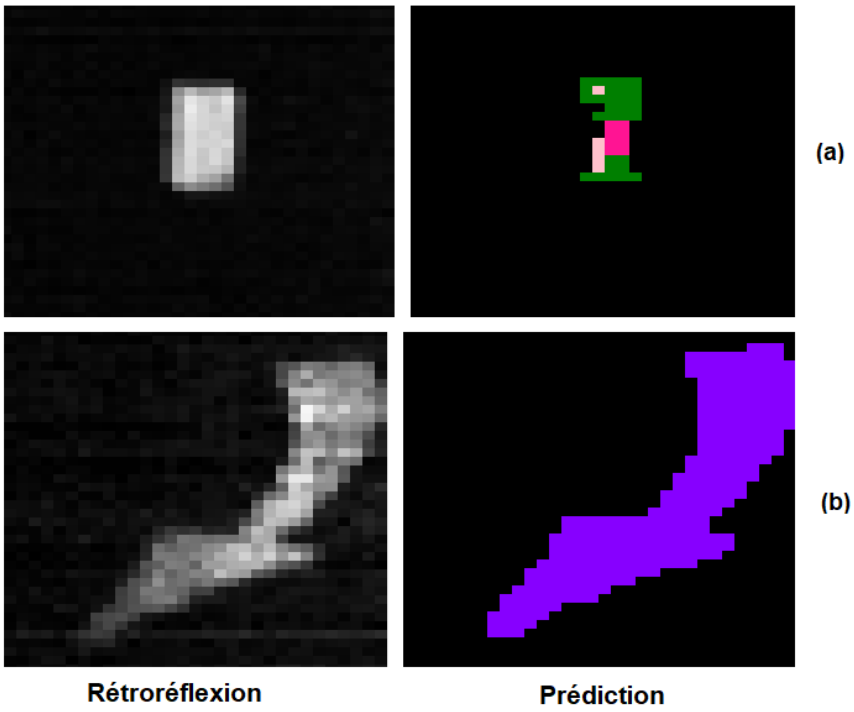


FIGURE 34 – Visualisation de la **segmentation multi-classes**.

Ces résultats peuvent se justifier par le fait que le **UNet** raisonne pixel par pixel et ne considère pas le contexte. Une meilleure répartition des classes et l'affectation de poids forts aux classes majoritaires pourrait améliorer les résultats. De plus, dans le cas d'un marquage contenant des pixels associés à des classes différentes, un vote majoritaire au sein d'une composante connexe peut régler ce problème. Toutes ces méthodes ont été implémentées dans un *stage* précédant [84] sur un petit jeu de données. Comme l'**UNet multi-classes** ne permet pas de gérer les inter-distances qui sont essentielles dans la typologie des marquages routiers, cette piste a été abandonnée. Dans l'étude, une approche géométrique a été faite.

4.3.3 Mise en place d'une identification basée sur la géométrie

Afin de classifier les marquages selon leur type, deux critères de reconnaissances sont utilisés : la **longueur** du marquage ainsi que l'**inter-distance** entre deux marquages. En effet, tous les marquages rectangulaires ont une **longueur** réglementée fixe quelle que soit la route où ils se trouvent. De même, l'**inter-distance** entre deux marquages rectangulaire du même type est aussi réglementée par la norme, mais entre deux marquages de type différent non. C'est une difficulté pour l'**identification**, mais comme les transitions de marquages sur une longue session de route sont rares, elles sont, en première approche, considérées comme négligeable.

La largeur du marquage rectangulaire ne sera pas considérée puisqu'elle varie selon le type de route où ces marquages se trouvent et aussi de sa situation. Par exemple, un marquage de type $\mathtt{T}_2$ a une largeur théorique de $3 \times U = 22.5cm$ (avec $U = 7.5cm$ sur autoroute) en ligne de rive alors qu'elle mesure théoriquement $5 \times U = 37.5cm$ de large en ligne de délimitation des voies d'insertion sur autoroute [5].

Pour classifier les marquages selon ces métriques, des intervalles d'appartenances doivent être définis. Les marquages non rectangulaires et rares sont considérés comme des marquages de type inconnu (`UNK`)

et ne sont pas pris en compte avec cette identification. On peut ainsi définir l'ensemble des marquages associés au type inconnu :

$$\mathcal{E}_{\texttt{UNK}} = \{\texttt{FLECHE}, \texttt{ZEBRA}, \texttt{PIETON}, \texttt{T}'_1, \texttt{T}'_2, \texttt{UNK}\}$$ [xxxi]

Plusieurs tests ont été faits pour le choix des intervalles d'appartenances. Ceux non retenus sont décrits en **Annexe** (cf. A.6). La méthode optimale est de lister toutes les valeurs théoriques des marquages non inconnues de façon croissante dans le cas des **longueurs** et des **inter-distances**. Les délimitations sont obtenues en faisant la moyenne entre les valeurs deux par deux (cf. Figure **35**). Par exemple, la valeur de délimitation pour la longueur entre $\texttt{T}'_3$, marquage de 20 mètres, et $\texttt{T}_4$, marquage de 39 mètres, est obtenue en calculant : $\frac{39+20}{2} = 29.5$. Ainsi, un marquage de longueur comprise entre 11.5 et 29.5 mètres sera étiqueté $\texttt{T}'_3$ à condition qu'il vérifie l'intervalle d'appartenance d'**inter-distance** de ce type.

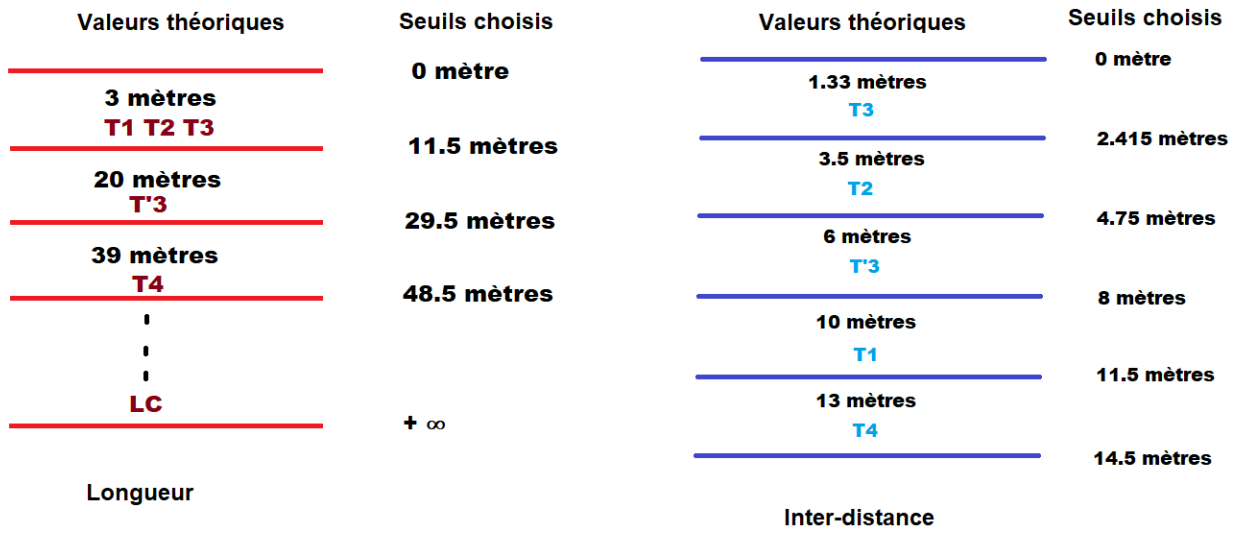


FIGURE 35 – Visualisation du découpage des intervalles d'appartenance, à gauche celui pour les **longueurs**, à droite celui pour les **inter-distance**.

L'algorithme d'**identification** (cf. **Annexe** 1) permet d'identifier le type selon ces deux critères. L'algorithme parcourt chaque composante connexe de la route **segmentée** à l'aide de la fonction `morphology.label` de la bibliothèque `skimage`, calcule la **longueur** de ce marquage et l'**inter-distance** entre lui et le précédent, puis identifie le type suivant le couple d'intervalles de **longueurs** et d'**inter-distances**. Lorsqu'un marquage n'appartient à aucun couple d'intervalles, il sera classifié de type «inconnu» noté `UNK`.

Pour évaluer les performances de cet algorithme, une nouvelle base a été créé uniquement pour l'identification, appelée **base de classification** (cf. Table 6). En effet, il existe des sessions de mesures où le conducteur a mal roulé ou bien des sessions avec des marquages avec une faible intensité en rétro-réflexion. Cela peut fausser les longueurs des marquages et les inter-distances. De plus, en calculant la

xxxi. Le type $\texttt{T}'_3$, même s'il est rare, est considéré comme un type unique pour ses propriétés géométriques simple.

longueur et l'**inter-distance** des marquages des données annotées, on a pu identifier certaines sessions de mesures où l'appareil **Ecodyn** a été mal réglé. Ces données ne seront pas prises en compte pour notre évaluation et nos statistiques. Cette nouvelle base comporte environ $220km$ de route.

Type	Nombre	Pourcentage
$\mathtt{T}_1$	**7096**	**37.20**%
$\mathtt{T}_2$	**5485**	**28.76**%
$\mathtt{T}_3$	**5021**	**26.32**%
$\mathtt{T}_4$	**625**	**3.28**%
LC	**117**	**0.613**%
$\mathtt{T}'_3$	**217**	**1.138**%
UNK	**512**	**2.684**%

TABLE 8 – Proportions de marquages au sein de la **base de classification**.

4.3.4 Évaluation de l'identification

Pour l'évaluation, une matrice de confusion a été construite. L'idée est de parcourir les composantes connexes de chaque route **annotée par l'utilisateur** dans la **base de classification** et de faire une correspondance avec la sortie de l'algorithme appliquée à la **segmentation** bi-classe. Les lignes de la matrice feront référence à la vérité terrain et les colonnes à la prédiction. Ainsi, le coefficient (i, j) de la matrice correspond au nombre de marquages de type associé à l'indice i et dont la prédiction de ces marquages est de type associé à l'indice j. Afin d'avoir de bonnes performances, il faut donc avoir une matrice à diagonale dominante.

	$\mathtt{T}_1$	$\mathtt{T}_2$	$\mathtt{T}_3$	$\mathtt{T}_4$	$\mathtt{T}'_3$	LC	UNK
$\mathtt{T}_1$	**6695**	19	57	0	0	4	321
$\mathtt{T}_2$	13	**4514**	46	0	3	1	8
$\mathtt{T}_3$	39	81	**4837**	0	0	0	64
$\mathtt{T}_4$	0	0	0	**499**	2	3	121
$\mathtt{T}'_3$	1	1	1	0	**74**	0	40
LC	2	15	51	0	3	**70**	76
UNK	16	150	229	1	0	8	**108**

TABLE 9 – Visualisation de la matrice de confusion pour le découpage associé à la Figure **35**.

Les résultats sont satisfaisants (cf. Table **9**). Il est possible d'expliquer la plupart des erreurs :

- Les marquages de type ligne continue (LC) sont souvent mal classés. Ceci vient du fait que la ligne continue peut être coupées (cf. Figures **36** et **(b)37**), par exemple à cause d'entrées de champs. Ainsi, ils n'entrent pas dans l'intervalle de longueur spécifique à ces marquages. Ils restent tout de même minoritaires comparés aux autres marquages.
- Des marquages flèches sont souvent classés en $\mathtt{T}_2$ (au lieu de UNK).
- Des transitions de marquages où l'**inter-distance** n'est pas normée entre deux marquages de type différent (cf. Figure **(a)(c)37**).

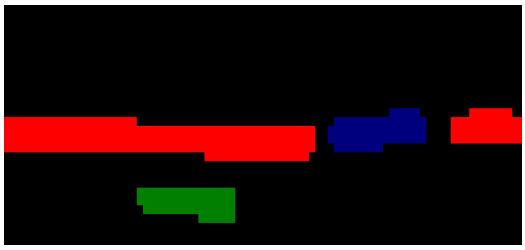


FIGURE 36 – Visualisation de l'identification faite par l'algorithme : la petite coupure de ligne continue a été classé en type T_3 (bleu) alors que c'est censé être de type LC (rouge).

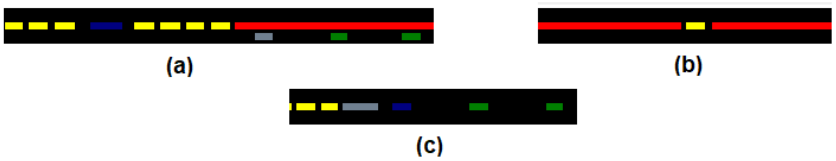


FIGURE 37 – Visualisation de quelques extraits du schéma itinéraire, **vérité terrain**, présentant des erreurs d'**identification**.

La Figure 37 illustre 3 cas de transition de marquages de différent type qui conduisent à de mauvais classement par l'algorithme d'identification. Ces mauvais classements conduisent à une mauvaise affection à la classe UNK. Ce qui explique les valeurs importantes de cette classe dans la Table 9. Ces transitions sont rares et pourront être détectés avec l'**analyse sémantique**.

Un autre résultat notable est le temps de fonctionnement de cet algorithme. Pour une route de 8km :

`22000017_A35_+_PR231_PR223_BAU.json`

le temps de parcourir chaque composante connexe et de les identifier est d'environ 25.7secondes ce qui est un temps faible. Cet algorithme est donc un ajout peu coûteux dans **Vérité Terrain Ecodyn**.

4.4 Analyse de la sémantique de la route

La **segmentation** et l'**identification** ont permis de localiser et d'identifier des marquages tout du long d'une route. Certaines erreurs liées à ces méthodessont encore présentes. De plus, il existe aussi des marquages de mauvais états sur la route. L'**analyse sémantique** de la route permet de localiser ces erreurs à travers le schéma itinéraire. Au lieu de réaliser une annotation manuelle, l'utilisateur peut visualiser les résultats de la **segmentation**, la **classification** sémantique et corriger les erreurs de **classification**.

L'**analyse sémantique** se fera avec des méthodes de ***deep learning***. Les marquages seront considérés comme des lettres et la route comme une phrase. C'est ici qu'interviennent les méthodes de **traitement naturel du langage**. Tout d'abord, la préparation des données ainsi que les **bases d'entraînements** et de **tests** seront présentés, puis l'implémentation des modèles sera décrit. Enfin, on présentera les résultats de chaque modèle utilisé.

4.4.1 Préparation des données

Les réseaux de neurones pour le **traitement de langage naturel** prennent en entrée des séquences. L'idée est de découper une route en plusieurs séquences et de les injecter dans le réseau de neurones pour l'apprentissage. Il ne suffit pas de considérer les marquages un par un, mais il faut considérer tout l'ensemble, c'est-à-dire prendre en compte la route non marquée entre deux marquages. Afin d'utiliser les modèles de **traitement du langage naturel**, il faut faire correspondre chaque marquage et chaque inter-marquage à une lettre, ce qui permet d'obtenir un vocabulaire. L'alphabet pour les marquages est défini de la façon suivante :

$$\mathcal{A}_{\text{marquage}} = \begin{cases} \texttt{M}_1 & \longrightarrow \{\text{marquages de 3 mètres}\} \equiv \{\texttt{T}_1, \texttt{T}_2, \texttt{T}_3\} \\ \texttt{M}_2 & \longrightarrow \{\text{marquage de 20 mètres}\} \equiv \{\texttt{T}_3'\} \\ \texttt{M}_3 & \longrightarrow \{\text{marquage de 39 mètres}\} \equiv \{\texttt{T}_4\} \\ \texttt{LC} & \longrightarrow \{\text{ligne continu}\} \equiv \{\texttt{LC}\} \\ \texttt{UNK} & \longrightarrow \mathcal{E}_{\texttt{UNK}} \end{cases}$$

Pour l'association des inter-marquages à des lettres, nous allons tout d'abord définir l'ensemble des types de marquage considérés. Les marquages rectangulaires non rares seront pris en compte. L'ensemble des types de marquages considéré est défini comme suit,

$$\mathcal{E}_{\text{type}} = \{\texttt{T}_1, \texttt{T}_2, \texttt{T}_3, \texttt{T}_4, \texttt{T}_3', \texttt{LC}, \texttt{UNK}\}$$

Les inter-marquages seront spécifiés par un préfixe `BG`, pour arrière-plan, puis d'un ou deux suffixes représentant les deux marquages autour de cet inter-marquage. On obtient,

$$\texttt{BG\_x}$$

si les types des deux marquages voisins sont identiques avec `x` dans $\mathcal{E}_{\text{type}}$, ou bien,

$$\texttt{BG\_x\_y}$$

si les types des marquages voisins sont distincts avec x, y dans $\mathcal{E}_{\text{type}}$. L'alphabet pour les inter-marquages est donc défini de la façon suivante :

$$\mathcal{A}_{\text{inter-marquage}} = \{\texttt{BG\_x} \mid x \in \mathcal{E}_{\text{type}}\} \cup \{\texttt{BG\_x\_y} \mid x, y \in \mathcal{E}_{\text{type}},\ x \neq y\}$$

L'alphabet complet est obtenu en faisant une union des deux alphabets,

$$\mathcal{A} = \mathcal{A}_{\text{marquage}} \cup \mathcal{A}_{\text{inter-marquage}}$$

Au final, l'alphabet comporte 54 lettres. Ces alphabets sont construits pour la typologie des marquages routiers français. Cependant, cette construction est faite de sorte qu'on puisse simplement ajouter/retirer des lettres et donc des nouveaux marquages afin de pouvoir généraliser aux marquages d'autres pays. Les marquages sous-représentés pourront aussi être facilement ajoutés lorsqu'il y aura davantage de données ou des critères permettant de les repérer.

Une fois l'alphabet construit, il suffit d'associer à chaque marquage et inter-marquage sa lettre qui sera stockée dans une liste. Chaque composante connexe sera parcourue du début du chemin jusqu'à la fin. Pour les bords de l'itinéraire, le premier inter-marquage sans marquage qui le précède sur la route sera uniquement spécifié selon le marquage qui le suit. De même pour le dernier inter-marquage, mais celui-ci sera spécifié par rapport au marquage qui le précède.

Afin d'éviter des problèmes avec les doubles bandes de marquages (cf. Figure (a)37), 3 différentes voies ont été mis en place. Ces 3 voies permettent de gérer le cas d'une ligne continue avec dans un premier temps une autorisation de dépasser sur la voie de droite puis sur la voie de gauche. Dans le script, elles seront représentées par des listes contenant des lettres. Les distances associées à chaque marquage/inter-marquage seront aussi stockées afin de pouvoir les repérer plus tard. La voie 2 représente-le «milieu» de la route au sens de la largeur et les deux autres, la gauche (voie 1) et la droite (voie 3) par rapport à ce milieu (cf. Figure 38). À noter que pour une route sans double bande, par exemple l'autoroute en **Axe**, les listes associées aux voies 1 et 3 seront vides. De ce fait, les trois listes sont de différentes tailles.

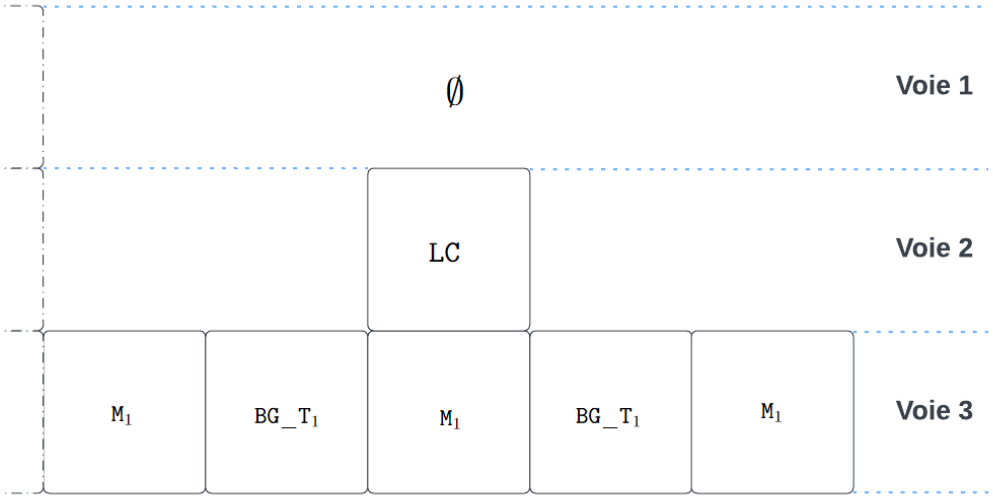


FIGURE 38 – Association des lettres pour une double bande (La route : cf. Figure 48).

Comme les réseaux de neurones pour le langage naturel prennent des séquences, chaque voie de la route sera divisée en plusieurs séquences. En particulier, les trois listes seront divisées avec un nombre d'éléments fixé à l'avance. Les séquences seront espacées d'un certain pas qui permet de générer plus de mots, aussi fixé par l'utilisateur. Sur l'exemple représenté sur la Figure 39, les séquences extraites sont de taille 8 (soit 8 lettres) et sont espacées d'un pas de 2.

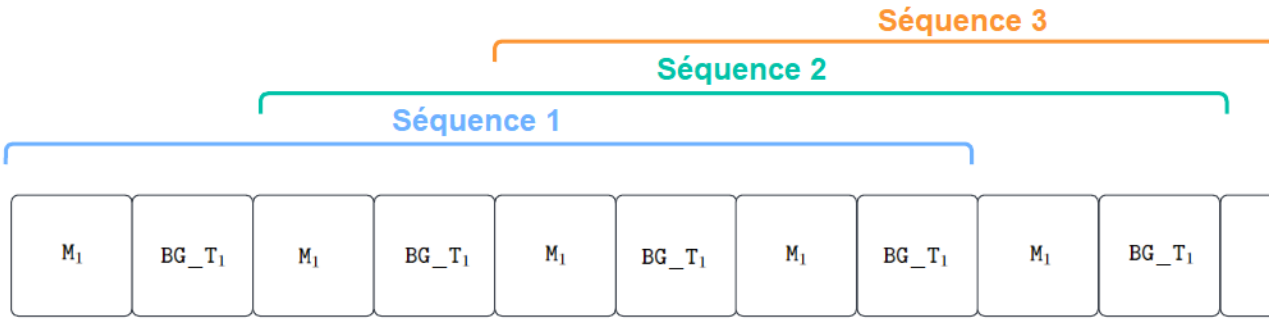


FIGURE 39 – Visualisation de l'extraction des séquences de 8 éléments espacée d'un pas de 2.

Les modèles utilisés classifient une séquence si elle est juste ou fausse au sens de la sémantique de la route. En l'occurrence, «fausse» signifie ici que l'utilisateur doit vérifier l'état de l'annotation associée à la séquence. Des erreurs seront créée artificiellement au sein d'une séquence pour permettre l'entraînement des réseaux de neurones. La moitié des séquences de la **base de classification** annotée sera transformée en séquences fausse en remplaçant un élément aléatoire de la séquence appartenant dans $\mathcal{A}_{marquage}$ par un autre distinct. Les éléments à droite et à gauche de l'élément transformé seront aussi transformés de façon logique. Par exemple :

- La liste $[\texttt{M}_1, \texttt{BG\_T}_1, \texttt{M}_1, \texttt{BG\_T}_1, \texttt{M}_1, \texttt{BG\_T}_1, \texttt{M}_1, \texttt{BG\_T}_1]$ de longueur 8 est choisie pour être transformé en séquence fausse.
- Un nombre parmi $\{1, 3, 5, 7\}$, correspondant aux indices des éléments dans $\mathcal{A}_{marquage}$ de cette liste, est choisi.
- Si 3 est choisi, alors $\texttt{T}_1$ est remplacée par une lettre choisie aléatoire dans $\mathcal{A}_{marquage} \setminus \{\texttt{T}_1\}$, par exemple $\texttt{T}_4$.
- Les éléments à gauche et à droite du 3ème élément seront aussi modifiés de façon logique, le résultat devient : $[\texttt{M}_1, \texttt{BG\_T}_1\texttt{\_T}_4, \texttt{T}_4, \texttt{BG\_T}_4\texttt{\_T}_1, \texttt{M}_1, \texttt{BG\_T}_1, \texttt{M}_1, \texttt{BG\_T}_1]$

À chaque séquence sera attribuée une étiquette 1 ou 0 suivant si elle est fausse ou non respectivement.

Les séquences, vraies ou fausses, comportant un élément `UNK` seront automatiquement étiquetées comme fausse puisque la plupart de ces éléments correspondent aux marquages non trouvés par l'algorithme d'identification et demande une interaction avec l'utilisateur.

4.4.2 Création des données d'apprentissages/validations et de tests

Pour la création de bases, des séquences de 16 éléments sont considérées soit environ 8 marquages et 8 inter-marquages. Le pas d'extraction sera de 2.

Plusieurs bases ont été créées à partir de la **base de classification** pour l'apprentissage :

- Une base d'apprentissage dite **Axe**, avec les sessions de mesures correspondant à l'**Axe** de la voie circulée,
- Une base d'apprentissage dite **Rive**, avec des sessions de mesures situés en **Rive** de la voie,
- Une base d'apprentissage dite **mélangée**, avec des sessions de mesures en axe et rive.

Leur base **test** respective est aussi créé (cf. Table 6). La séparation entre **base d'entrainement** et **base test** a été fait aléatoirement de façon distincte. Les bases de **validation** ont été créées de la même manière que précédemment en extrayant de façon aléatoire et distincte 20% des bases d'**entraînement**. La Table 11 montre le nombre de séquences correctes et fausses dans chaque base test.

Base Axe		
	Train	**Test**
Taille ***(km)***	100.88	43.04
Nombre de ***séquences***	8124	8368

Base Rive		
	Train	**Test**
Taille ***(km)***	29.47	15.9
Nombre de ***séquences***	1736	1873

Base mélangée		
	Train	**Test**
Taille ***(km)***	146.54	75.39
Nombre de ***séquences***	9819	16621

TABLE 10 – Visualisation des différentes bases d'**entraînements** et de **tests**.

Nombre de **séquences**		
	Fausse	**Correcte**
Mélangée	7 662	8 959
Axe	3 895	4 473
Rive	845	968

TABLE 11 – Visualisation du nombre de séquences fausses et correctes dans chaque base **test**.

4.4.3 Entraînement des modèles (RNN, LSTM, Mécanisme d'attention)

Les séquences de lettres sont transformées en séquences de nombres et sont données en entrée pour chaque modèle utilisé. Pour l'entraînement des modèles de **classification**s, la configuration $type_{NLP}$ suivante a été utilisée :

- Taille de ***batch*** = 128
- ***Loss*** utilisée : ***Cross-Entropy Loss***
- Nombre d'***epochs*** = 100
- Optimiseur : ***Adam optimization algorithm(lr = 0.001)***
- Lors de l'entraînement, le modèle avec les poids/biais qui minimise le mieux la ***Loss*** de validation sera retenue et sauvegardée.

L'implémentation des modèles récurrents est inspirée par l'implémentation des réseaux de **classification** avec modèles récurrents de critiques de films de l'article officiel de `Tensorflow` [85]. Dans l'implémentation, pour les modèles récurrents (**RNN** et **LSTM**), il y a tout d'abord une couche d'**Embedding** (cf. 3.4.2) puis une couche récurrente (**RNN** ou **LSTM**) avec 32 neurones. Enfin, une couche dense pour la **classification** est ajoutée (cf. Figure 40). Ces architectures seront notées **RNN** et **LSTM**. Les couches `Dense`, `SimpleRNN`, `LSTM` et `Embedding` ont été utilisées de `Tensorflow`.

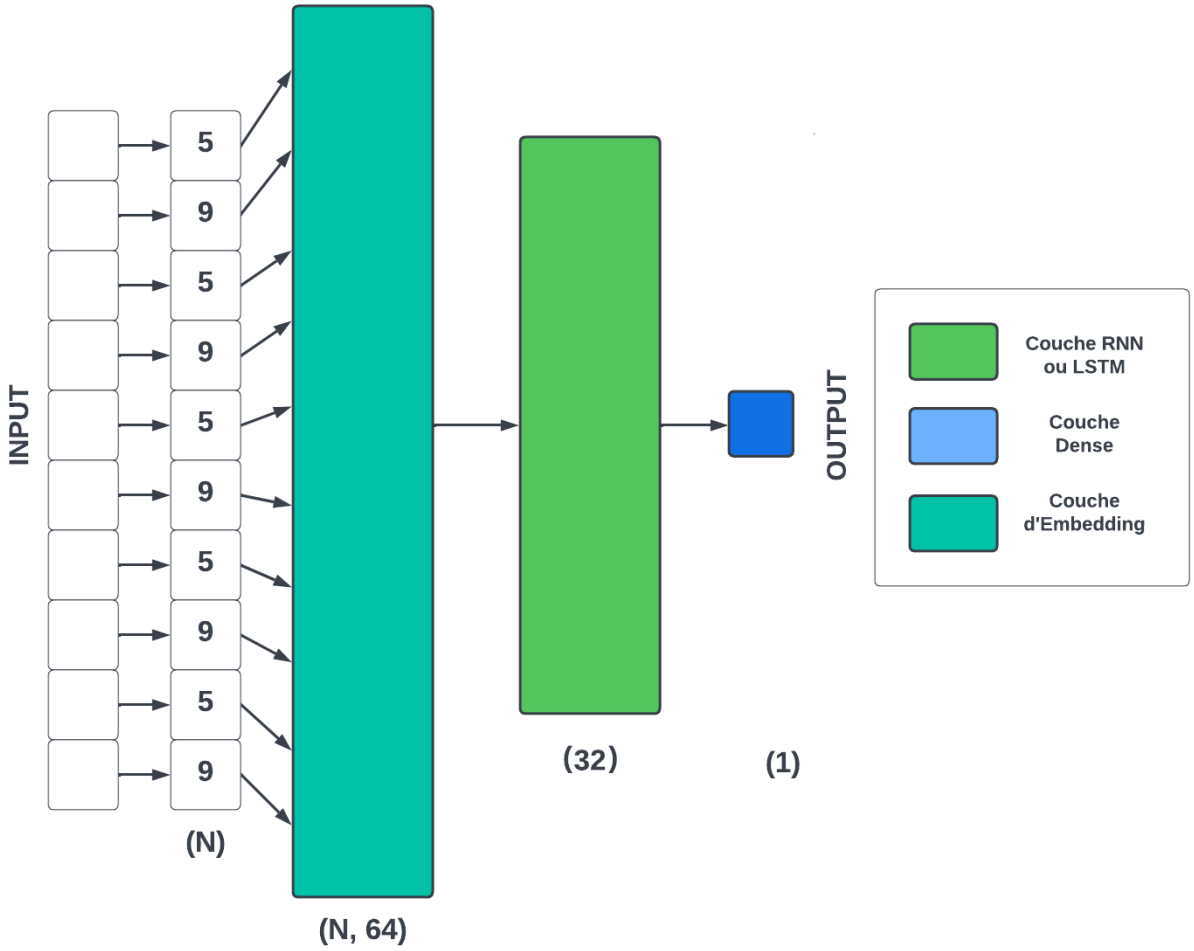


FIGURE 40 – Visualisation de l'implémentation du modèle récurrent.

L'implémentation du modèle utilisant le mécanisme d'attention est inspirée par l'implémentation de la **classification** de textes par transformer de l'article officiel de `Keras` [86]. Dans l'implémentation, il y a tout d'abord une couche d'**Embedding** qui s'additionne avec une couche de **Position Embedding** afin d'obtenir les caractéristiques et les positions de chaque élément de la séquence (cf. 3.4.2). L'encodeur du transformer est ensuite appliqué. Enfin, une couche de ***pooling*** est utilisée afin de réduire en dimension puis des couches denses sont appliquées pour faire la **classification**. Les couches `Dense`, `Embedding`, `MultiHeadAttention` et `AveragePooling2D` de `Tensorflow` ont été utilisées.

Deux architectures avec **mécanisme d'attention** différent sont testées, un avec deux têtes (**Trans**$_2$) et un autre avec une tête (**Trans**$_1$) (cf. 3.4.3).

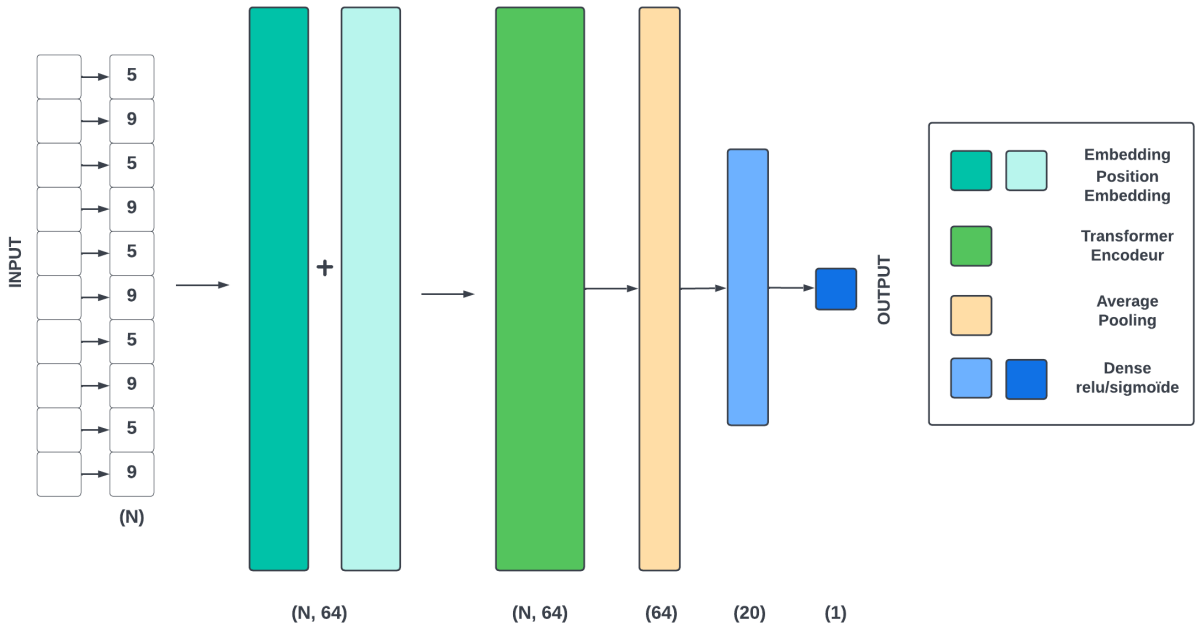


FIGURE 41 – Visualisation de l'implémentation du modèle avec **mécanisme d'attention**.

Les Tables 12 indiquent le temps d'entraînement pour 100 **epochs** ainsi que la taille du fichier `.h5` résultant. Elles montrent que le réseau **LSTM** est plus rapide que les autres réseaux.

Base Axe		
	Temps Entraînement pour 100 epochs (s)	**Taille (.h5) (Ko)**
RNN	367.46	133
LSTM	91.10	221
Trans$_2$	136.71	608
Trans$_1$	132.12	414

Base Rive		
	Temps Entraînement pour 100 epochs (s)	**Taille (.h5) (Ko)**
RNN	84.46	133
LSTM	33.24	221
Trans$_2$	37.24	608
Trans$_1$	35.51	414

Base mélangée		
	Temps Entraînement pour 100 epochs (s)	**Taille (.h5) (Ko)**
RNN	317.14	133
LSTM	80.67	221
Trans$_2$	116.288	608
Trans$_1$	114.7041	414

TABLE 12 – Visualisation des temps d'apprentissage et de la taille des fichiers `.h5` résultant.

4.4.4 Évaluation des modèles pour l'analyse sémantique

Pour l'évaluation de la **classification**, les métriques présentées dans la section 3.2 telles que, la **Cross-Entropy Loss**, la **précision**, le **rappel** ainsi que le **score F1** [xxxii]ont été calculées par rapport à la base **test** pour les architectures **RNN**, **LSTM**, **Trans**$_1$ et **Trans**$_2$. Les **matrices de confusion** associées sont données en **Annexe** (cf. Tables 20,21,22). De plus, le temps d'inférence est calculé pour une route de 8km (cf. Tables 13,14,15).

	Loss	Précision (%)	Rappel (%)	F1 (%)	Nombre de paramètres	Temps d'inférence (s)
RNN	**0.0538**	99.12	94.46	96.73	**6593**	0.0278
LSTM	0.05657	99.05	**94.76**	**96.87**	15905	**0.0155**
Trans$_2$	0.0711	99.30	94.21	96.09	43486	0.0711
Trans$_1$	0.0683	**99.31**	94.23	96, 71	26910	0.0301

TABLE 13 – Représentation des résultats pour la base **mélangée**.

	Loss	Précision (%)	Rappel (%)	F1 (%)	Nombre de paramètres	Temps d'inférence (s)
RNN	0.031	99.57	99.87	99.72	**6593**	0.0325
LSTM	**0.01577**	99.59	**100.0**	**99.81**	15905	**0.01082**
Trans$_2$	0.0314	**99.84**	99.74	99.79	43486	0.0253
Trans$_1$	0.02098	99.72	99.87	99.79	26910	0.01951

TABLE 14 – Représentation des résultats pour la base **Axe**.

	Loss	Précision (%)	Rappel (%)	F1 (%)	Nombre de paramètres	Temps d'inférence (s)
RNN	0.1038	94.66	89.06	91.76	**6593**	0.0267
LSTM	0.1131	93.68	90.86	92.25	15905	**0.0117**
Trans$_2$	**0.0961**	**95.42**	90.53	**92.91**	43486	0.0155
Trans$_1$	0.09986	94.23	**91.09**	92.63	26910	0.01707

TABLE 15 – Représentation des résultats pour la base **Rive**.

Ces résultats sont obtenus en utilisant le seuil optimal, c'est-à-dire en optimisant le ***F1 score***. Les modèles ont des scores très satisfaisants. Pour la base **Axe**, les résultats tournent autour de 99% pour la **précision** et le **rappel**. Cependant, la base **Rive** admet des pénalités au niveau des évaluations. La pénalisation de la **précision** peut s'expliquer par quelques particularités au niveau de la **Rive**, par exemple par la présence de marquages $\mathtt{T}_3'$ de façon illogique autour de marquage $\mathtt{T}_2$ (cf. Figure **42**) ce qui confronte le modèle à un dilemme.

xxxii. L'accuracy n'a pas été calculée puisqu'on s'intéresse uniquement aux séquences fausses.

FIGURE 42 – Visualisation de quelques extraits du schéma itinéraire, **vérité terrain**, présentant des marquages $\mathtt{T}'_3$ (Route : `17000009_A352_-_PR10_PR2_BAU`).

Le **rappel** tourne autour de 90% pour la base **Rive** pour chaque architecture. Les modèles ratent donc un certain nombre de séquences justes, par exemple des coupures de ligne continue qu'ils classent comme une séquence illogique. Cependant, le fait que les réseaux classent plus de séquences en fausse que prévu n'est pas pénalisant, l'utilisateur devra vérifier des marquages justes qui ont été classés faux. La **précision** est donc à maximiser afin qu'il ne classe pas des fausses séquences en positif. Le transformer à deux têtes est le meilleur modèle en termes de **précision** pour chaque base. Le modèle **LSTM** a des performances très hautes pour la base **Axe** et est le réseau le plus rapide pour l'entraînement et l'inférence quelque soit la base choisie.

Pour l'incorporation des modèles d'**analyse sémantique**, le modèle $\mathbf{Trans}_2$ peut être utilisé pour des sessions de mesures **Rive**, et le modèle **LSTM** pour des sessions de mesures **Axe**. Actuellement, seul le modèle **LSTM** a été incorporé dans **Vérité Terrain Ecodyn**. Il restera à incorporer le modèle $\mathbf{Trans}_2$ pour les sessions de mesure en **Rive**.

5 Conclusion et perspectives

5.1 Conclusion

Dans ce rapport, nous avons présenté les travaux réalisés pour la **classification** et l'interprétation des marquages routiers. Les données traitées étaient issues de l'appareil **Ecodyn3**, dont dispose le **Cerema**, qui mesure les valeurs de **rétroréflexion** et de **luminance** des marquages routiers. Plus de 200km de mesures ont été effectuées et utilisées pour l'apprentissage supervisé. Il a été nécessaire de les annoter et de les mettre en forme, ce qui a nécessité d'apporter des corrections et améliorations au logiciel d'annotation **Vérité Terrain Ecodyn**.

L'exploration de méthodes de ***deep learning*** sur l'analyse automatique de marquages routiers par apprentissage a montré des résultats très intéressants. Différentes expériences ont montré que le réseau **UNet**, utilisé notamment pour la **segmentation** de cellules biologiques, est adapté à la problématique. Des modifications en termes de profondeur du réseau ont permis de l'alléger et d'obtenir des performances satisfaisantes. En effet, le modèle **UNet** démontre, avec des résultats de **précision** et de **rappel** proche de 93%, l'utilité de l'approche de la **segmentation** binaire (cf. 4.2.3). L'incorporation d'un tel modèle dans le logiciel d'annotation **Vérité Terrain Ecodyn** a permis un gain de temps considérable. Au lieu de réaliser une annotation manuelle pendant plusieurs heures, l'opérateur appelle le modèle qui lui propose une annotation. Il lui suffit alors de la corriger, ce qui prend une dizaine de minutes. Cet ajout construit un mécanisme de cycle : l'utilisateur vérifie et corrige l'annotation automatique produite puis, ces nouvelles données sont ré-injectées dans le réseau de neurones afin d'améliorer les performances continuellement. Cette **segmentation** admet tout de même des limites. En effet, des tâches de lumières peuvent être classifiées comme du marquage et des marquages en mauvais états classifiés comme de l'arrière-plan (chaussée).

Les méthodes de **segmentation** basées un un réseau **UNet multi-classes**, ne prenant pas en compte la longueur des marquages et l'inter-distance entre deux marquages, métriques fondamentale pour le marquage, échouent à identifier le type des marquages. À partir de pixels de marquage routier détectés, une approche géométrique a été utilisée afin d'identifier les types de marquages. Comme les caractéristiques géométriques de la signalisation horizontale sont standardisées **[5]**, la matrice de confusion obtenue avec cette méthode est assez satisfaisante, avec une matrice à diagonale dominante (cf. Table 9).

Afin d'identifier les marquages de mauvaise qualité ainsi que les erreurs liées à la **segmentation/identification**, une **analyse sémantique** a été faite.Les marquages ont été interprétés comme des lettres et la route comme une phrase afin de coller à un problème de **traitement du langage naturel**. Il a fallu mettre en place un alphabet, gérer les doubles bandes... Les modèles **récurrents** et les **transformers**, principaux acteurs dans le traitement du langage, ont donné des résultats satisfaisants. Parmi eux, 4 modèles ont été implémentées : un **RNN**, un **LSTM**, deux réseaux de classification $\mathbf{Trans}_2$ et $\mathbf{Trans}_1$, chacun avec une couche de ***Multi-Head-Attention*** à 2 et 1 tête respectivement. Le réseau **LSTM** admet des performances de **rappel** et précision proche de 100% pour les marquages venant de la voie circulé **Axe** et le réseau $\mathbf{Trans}_2$ admet des performances de **rappel** et précision proche de 92% pour les marquages venant de la voie circulé **Rive**.

Ces modèles ont permis de faire évoluer le logiciel **Vérité Terrain Ecodyn** du stade de code de recherche à un niveau plus opérationnel, en tenant compte des besoins et remarques des utilisateurs. Les modèles de **segmentation** et d'**analyse sémantique** ont été intégrés dans le logiciel d'annotation ainsi qu'un schéma itinéraire permettant de visualiser les résultats des modèles de ***deep learning***.

5.2 Conclusion personnelle

Le *stage* m'a permis d'approfondir et d'élargir mes connaissances dans le domaine du ***machine learning*** ainsi que dans le développement logiciel. J'ai eu la chance de pouvoir travailler aussi bien en équipe qu'en autonomie, et de participer à des sessions de mesures. Le travail en équipe m'a aussi poussé à avoir plus confiance en moi.

5.3 Perspectives

En perspectives, il serait intéressant de tester d'autres réseaux de **segmentation** pouvant éventuellement remplacer le **UNet**. Par exemple, le masque **R-CNN** [87], qui lui est un modèle de **segmentation d'instance** et aussi un modèle de **détection**, pourrait être adapté au cadre des marquages routiers. En effet, en plus de segmenter les marquages, il permet de détecter les marquages et ainsi d'obtenir la position de chacun d'entre eux pouvant être utilisé comme données **GPS**.

Au niveau du logiciel **Vérité Terrain Ecodyn**, l'intégration de l'**analyse sémantique** n'a pas encore été finalisée. Il manque l'affichage des erreurs sur le schéma itinéraire. À l'issue de cette implémentation, l'utilisateur n'aura plus qu'à corriger les quelques marquages faux, ce qui permettra d'avoir une annotation fluide et rapide.

Une prochaine étape serait de développer un logiciel d'exploitation permettant d'inclure des calculs statistiques sur les données actuellement réalisés dans le cadre du projet **SAM** [14].

A Annexe

A.1 Roadmap

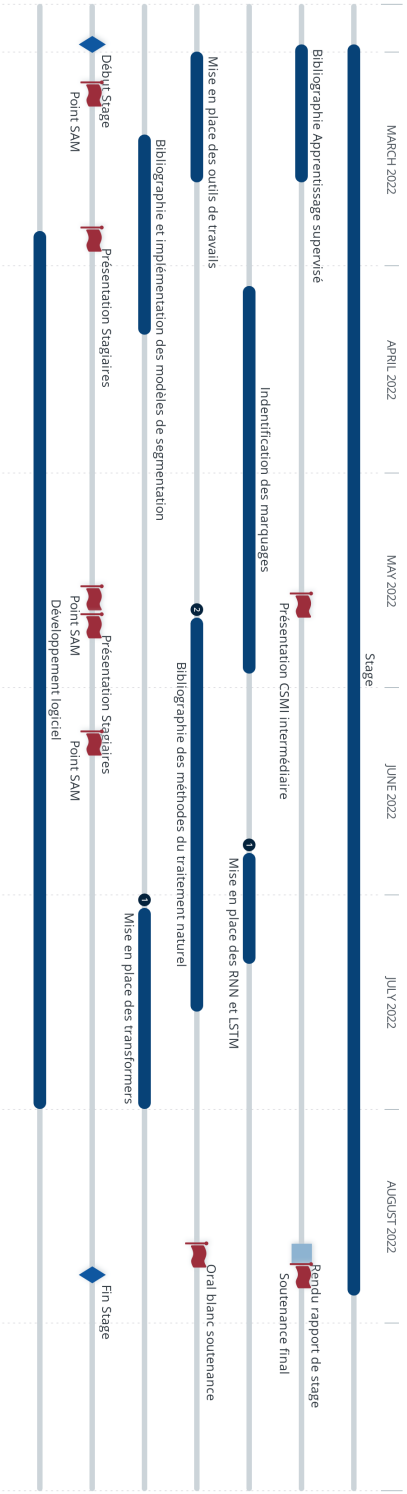


FIGURE 43 – Visualisation de la roadmap.

A.2 Format des données

```
    "File": "20000093_D13s2bg38p1_-_PR15_PR0_Axe",
    "Cote": "Axe",
    "distance parcourue": "6.580",
    "Detected": 0,
    "Profil 1": {
        "distance": 405.419104,
        "flux": "30",
        "GPS": {
            "Longitude": "1.0523566666666666",
            "Latitude": "49.38386166666667"
        },
        "voie 1": {
            "nuit": 1.1083700489796726,
            "jour": 148043.2897143364,
            "type": "T1",
            "num_composante": 0
        },
        "voie 2": {
            "nuit": 7.9155976841930915,
            "jour": 157218.39413642883,
            "type": "T1",
            "num_composante": 0
        },
        ...
        "voie 32": {
            "nuit": 9.420345475605084,
            "jour": 144914.052631855,
            "type": "BG",
            "num_composante": 0
        }
    },
    "Profil 2": {
        "distance": 810.838208,
        "flux": "30",
        "GPS": {
            "Longitude": "1.0523566666666666",
            "Latitude": "49.38386166666667"
        },
        "voie 1": {
            "nuit": 1.5410342766688423,
            "jour": 150855.12453866005,
        ..
```

Listing 1 – Fichier `.json`.

A.3 Méthodes de sur-échantillonage

Convolution Transposée

On considère un noyau 2×2 et une entrée 2×2 (cf. Figure 44). On aimerait agrandir l'entrée en une matrice 3×3.

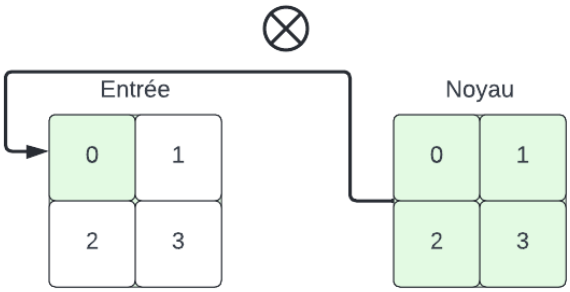


FIGURE 44 – Exemple de convolution transposée (1).

Pour cela, une matrice nulle 3×3 est initialisée. Le noyau est multiplié par le premier coefficient de l'entrée et le résultat est stockée dans une matrice 3×3 . La même opération est effectuée pour chaque coefficient (cf. Figure 45) et tous les résultats sont additionnés afin d'obtenir la sortie.

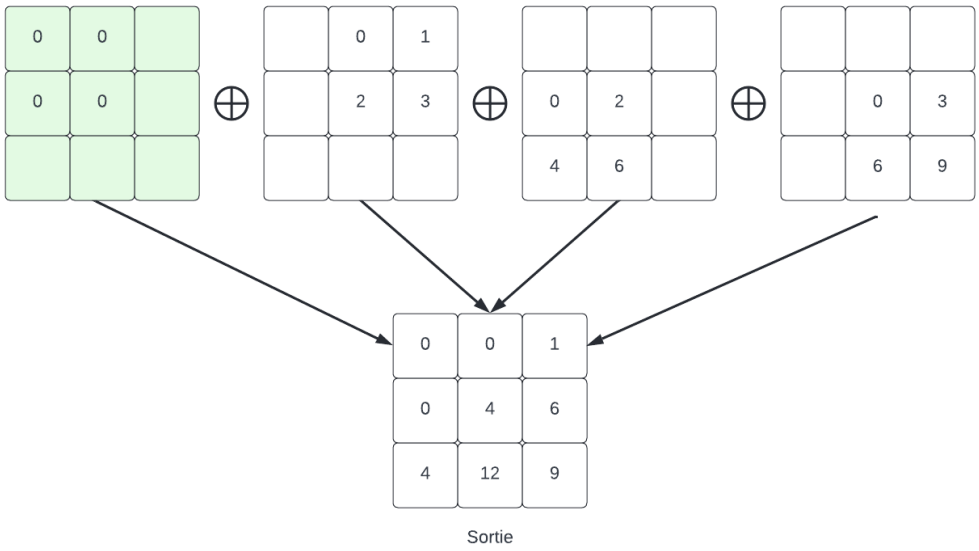


FIGURE 45 – Exemple de convolution transposée (2).

Pooling up-sampling

Pour le ***pooling up-sampling***, il faut stocker les indices de ***pooling*** de l'opération de ***pooling*** utilisée précédemment comme le montre la Figure 46. À l'aide de ces indices, on sait quels coefficients garder lors du passage à la dimension supérieure. Le reste des coefficients est mis à 0.

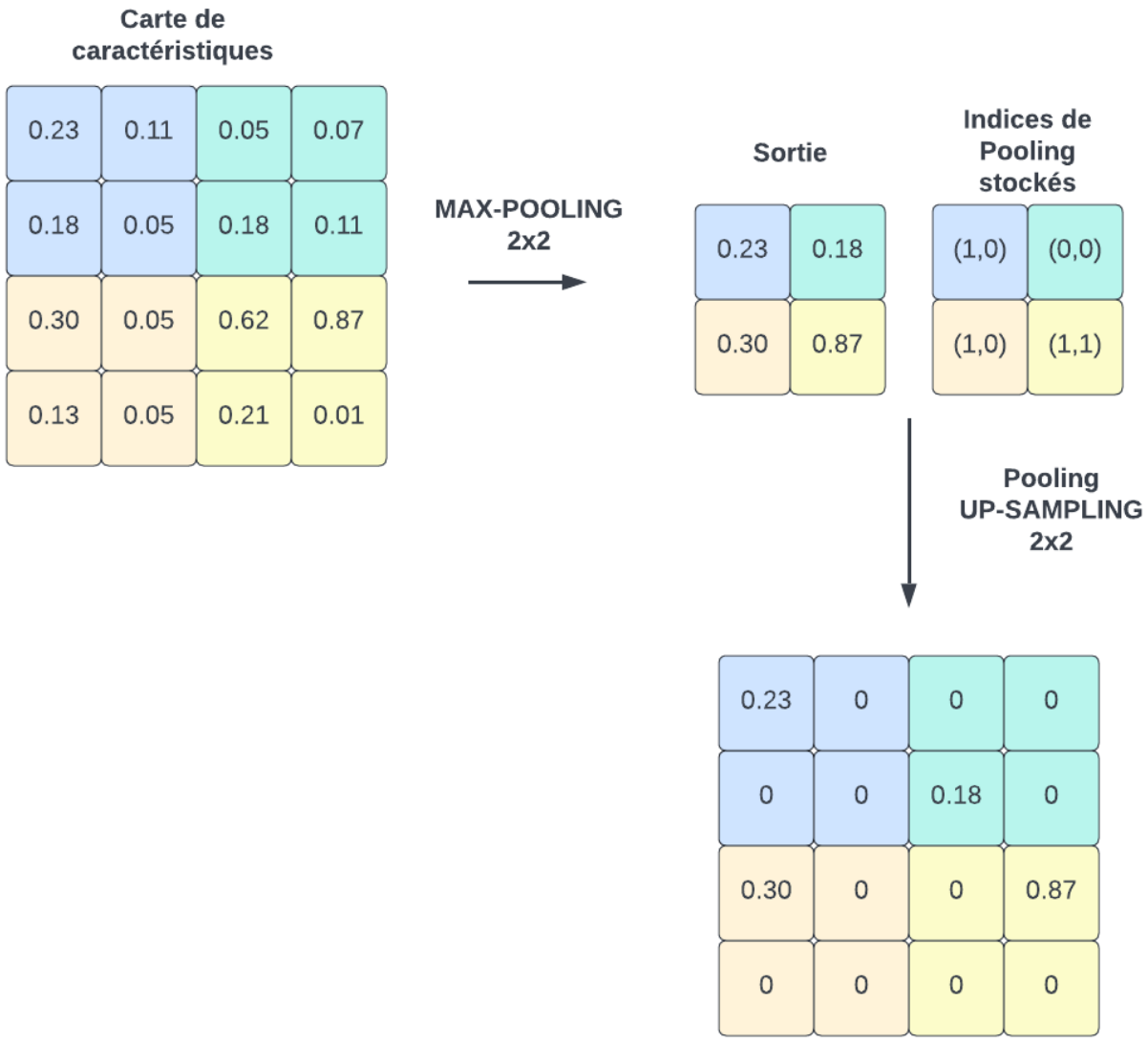


FIGURE 46 – Exemple de ***pooling up-sampling***.

A.4 Schéma itinéraire

FIGURE 47 – Visualisation du schéma itinéraire.

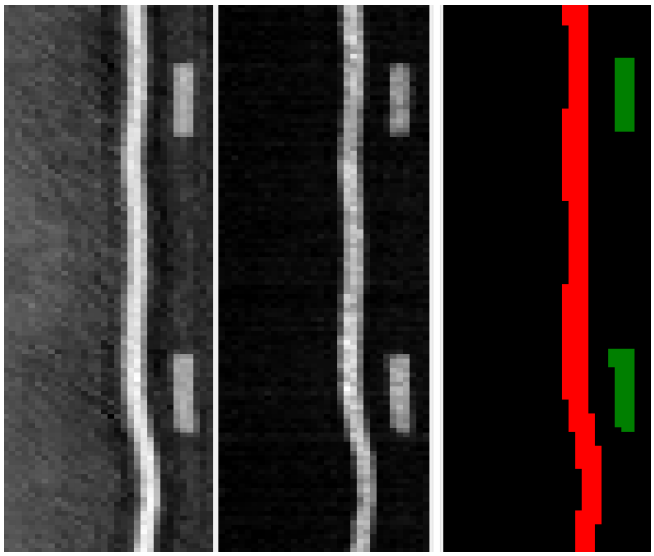


FIGURE 48 – Visualisation d'une double bande.

A.5 Implémentation des méthodes de segmentation

```
    l1 = [16, 32, 64, 128, 256]
    l2 = [16, 32, 64, 128]
    l3 = [16, 32, 64]
    l4 = [16, 32]
    l4_ = [8, 16]
    l5 = [16]

    l = l4 # a choisir

    # Encodeur :
    s_list = []
    p_list = [inputs]
    for i, nbr_features in enumerate(l[:-1]):

        s, p = encoder_block(p_list[i], l[i])
        p_list.append(p)
        s_list.append(s)

```

```
    # Bottleneck :
    b = conv_block(p_list[-1], l[-1])

    # Decodeur :
    d_list = [b]
    s_list.reverse()
    for i, nbr_features in enumerate(reversed(l[:-1])):
        d = decoder_block(d_list[i], s_list[i], nbr_features)
        d_list.append(d)

    last_conv = layers.Conv2D(nb_classes, (1, 1))(d_list[-1])
```

Listing 2 – Unet modifié

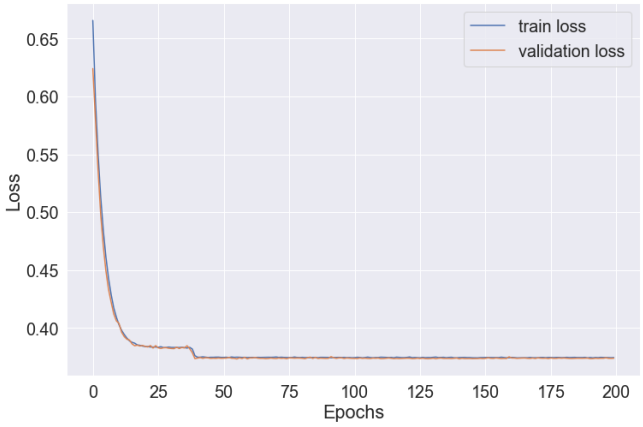


FIGURE 49 – Visualisation des ***Loss*** pour le modèle **SegNet** (⋆ ⋆ ⋆).

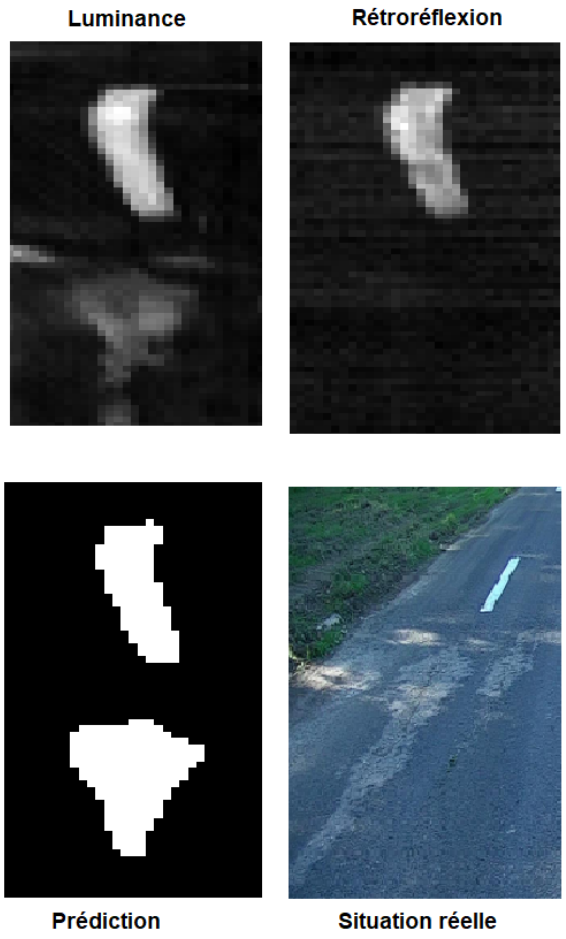


FIGURE 50 – Exemple de mauvaise **segmentation**.

A.6 Identification

Autres découpages testés :

- Découpage ($\star$) : Pour les marquages de 3 mètres, les intervalles d'appartenances sont définis par la relation : **valeur théorique** ±0.8. Le choix de cette distance correspond à 2 fois le pas d'acquisition (c'est-à-dire 40cm qui est la distance parcourue entre deux mesures successives). Les autres marquages ont pour intervalles d'appartenances les mêmes que le découpage retenu pour l'étude.
- Découpage ($\star$ $\star$) : Pour chaque marquage, les intervalles d'appartenances ont été définies en calculant la moyenne et l'écart type de la longueur et de l'inter-distance pour chaque marquage de la base annotée. Les intervalles sont définis par la relation : **moyenne ± écart type**.

	T_1	T_2	T_3	T_4	T'_3	LC	UNK
T_1	**5 941**	9	44	0	0	4	1 098
T_2	6	**4174**	13	0	3	1	388
T_3	3	60	**4268**	0	0	0	606
T_4	0	0	0	**499**	2	3	121
T'_3	1	1	1	0	**74**	0	40
LC	2	15	51	0	3	**70**	76
UNK	1	29	71	1	0	8	**402**

TABLE 16 – Matrice de confusion pour le découpage ($\star$)

	T_1	T_2	T_3	T_4	T'_3	LC	UNK
T_1	**3 019**	3	43	0	0	4	3959
T_2	1	**3 278**	12	0	3	1	1308
T_3	15	3	**4 235**	0	0	0	714
T_4	0	0	0	**405**	1	10	194
T'_3	0	0	0	0	**71**	0	46
LC	0	3	15	0	1	**76**	122
UNK	0	5	43	1	0	8	**455**

TABLE 17 – Matrice de confusion pour le découpage ($\star\star$)

	T1	**T2**	**T3**	**T'3**	**T4**
Moyenne	3.32	3.15	3.26	19.03	39.35
Médiane	3.2	3.2	3.2	20.0	39.2
Écart-type	0.37	0.67	0.57	5.11	3.65
Valeur théorique (m)	3	3	3	20	39

TABLE 18 – Table représentant les statistiques sur les **longueurs** des marquages rectangulaires.

Algorithm 1 Algorithme d'**identification**.

Require : L, d, Type_marker = None

```
for Type in E_type do
  if in_intervall_length(Type, L) == True then
    if in_intervall_interd(Type, d) == True then
      Type_marker ← Type
    end if
  end if
  if Type_marker == None then
    Type_marker ← UNK
  end if
end for
```

	T1	T2	T3	T'3	T4
Moyenne	9.81	3.57	1.34	5.99	12.66
Médiane	10.0	3.6	1.2	6.0	12.8
Écart-type	0.74	0.47	0.36	0.51	1.24
Valeur théorique (m)	10	3.5	1.33	6	13

TABLE 19 – Table représentant les statistiques sur les **inter-distances** des marquages rectangulaires

A.7 Matrices de confusion de l'analyse sémantique

RNN	
7 240	422
65	8 894

LSTM	
7 281	401
70	8 889

Trans$_2$	
7 162	440
50	8 909

Trans$_1$	
7 220	442
50	8 909

TABLE 20 – Visualisation des **matrices de confusion** pour chaque modèle pour la base **mélangé**.

RNN	
3 890	5
17	4 472

LSTM	
3 895	0
16	4 479

Trans$_2$	
3 885	10
6	4 467

Trans$_1$	
3 890	5
11	4 462

TABLE 21 – Visualisation des **matrices de confusion** pour chaque modèle pour la base **Axe**.

RNN	
802	99
45	923

LSTM	
819	82
56	912

Trans$_2$	
815	85
39	929

Trans$_1$	
821	24
50	918

TABLE 22 – Visualisation des **matrices de confusion** pour chaque modèle pour la base **Rive**.

A.8 Analyse sémantique dans Vérité Terrain Ecodyn

```
"Detected": {
            "Marquage 1": {
                  "Position": 6160.0,
                  "Type": "T2",
                  "OK": 0
            },
            "Marquage 2": {
                  "Position": 12095.800000000001,
                  "Type": "UNKNOWN",
                  "OK": 0
            },
            "Marquage 3": {
                  "Position": 13874.0,
                  "Type": "T2",
                  "OK": 1
            },
            "Marquage 4": {
```

```
                "Position": 4.4,
                "Type": "UNKNOWN",
                "OK": 1
            },
            "Marquage 5": {
                "Position": 42.400000000000006,
                "Type": "T4",
                "OK": 1
            },
            "Marquage 6": {
                "Position": 100.4,
                "Type": "T4",
                "OK": 1
            },
            "Marquage 7": {
                "Position": 158.0,
                "Type": "T4",
                "OK": 1
                ..
```

Listing 3 – Visualisation de l'analyse sémantique dans **Vérité Terrain Ecodyn**

A.9 Choix des seuils pour l'analyse sémantique

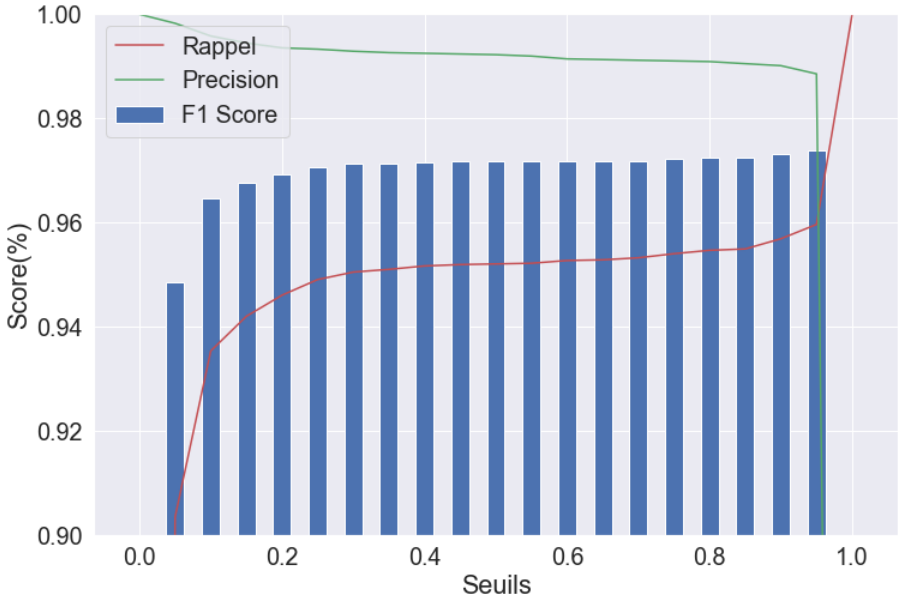


FIGURE 51 – (**Base mélangée**) Visualisation des scores en fonction du **seuil de confiance** pour le **RNN**.

Le **seuil** choisi pour les résultats est de 0.9 (cf. Figure **51**).

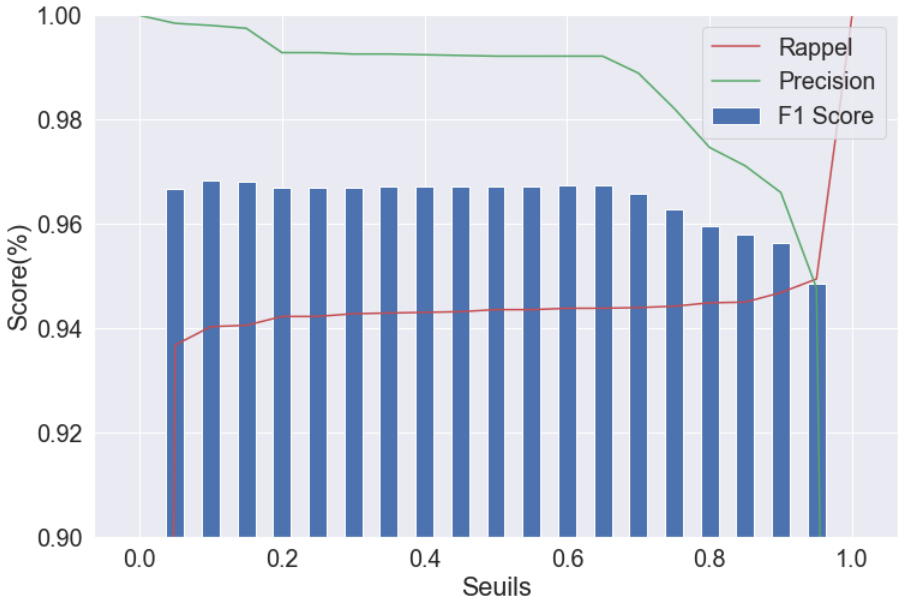


FIGURE 52 – (**Base mélangée**) Visualisation des scores en fonction du **seuil de confiance** pour le **LSTM**. Seuil choisi : **0.9**

Le **seuil** choisi pour les résultats est de 0.15 (cf. Figure **52**).

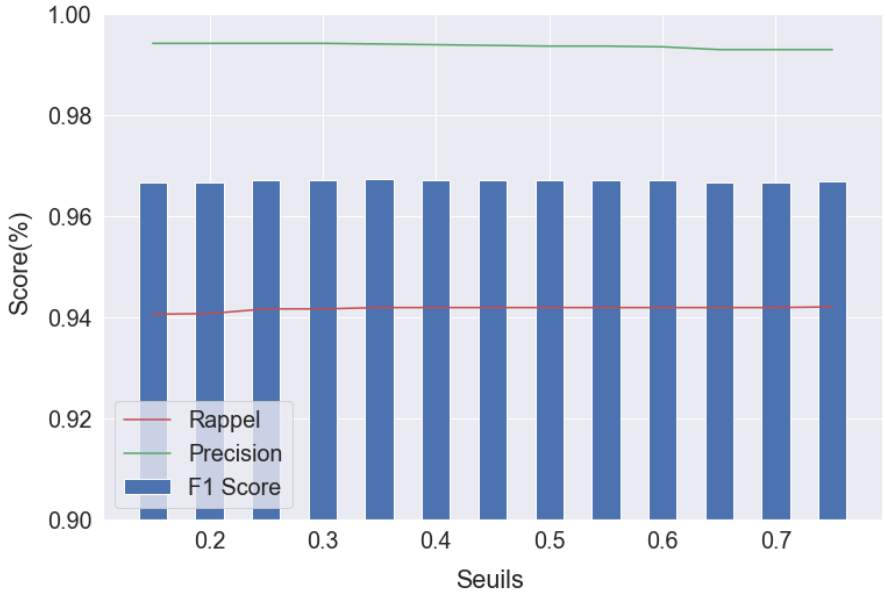


FIGURE 53 – Visualisation des scores en fonction du **seuil de confiance** pour $\mathbf{Trans}_2$. Seuil choisi : **0.9**

On remarque que le choix du **seuil de confiance** a peu d'impacts sur les résultats. Le **seuil** choisi pour les résultats est de 0.5 (cf. Figure **53**).

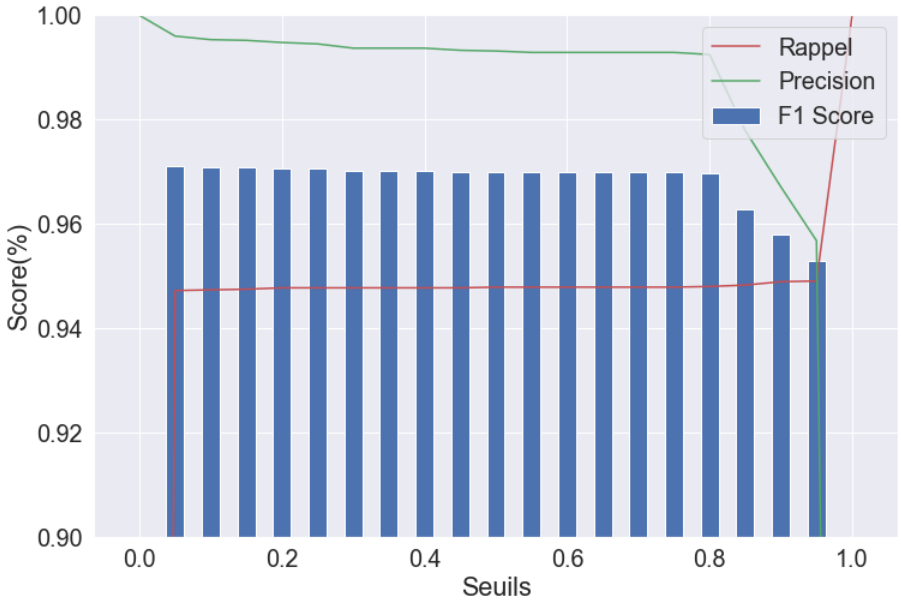


FIGURE 54 – Visualisation des scores en fonction du **seuil de confiance** pour **Trans**$_1$. Seuil choisi : **0.9**

Le **seuil** choisi pour les résultats est de 0.8 (cf. Figure 54).

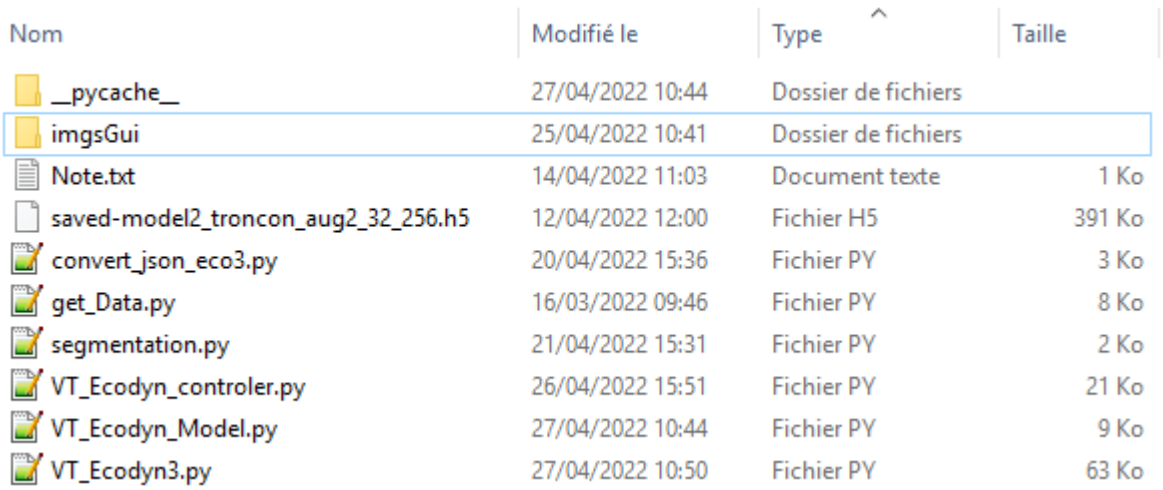
B Notice du logiciel Vérité Terrain Ecodyn

Vérité Terrain Ecodyn est un outil qui a pour but de détecter et d'identifier les marquages routiers à partir des mesures de rétroréflexion produites par l'appareil de mesure **Ecodyn3**.

Mise en route du logiciel

Fichiers utilisés

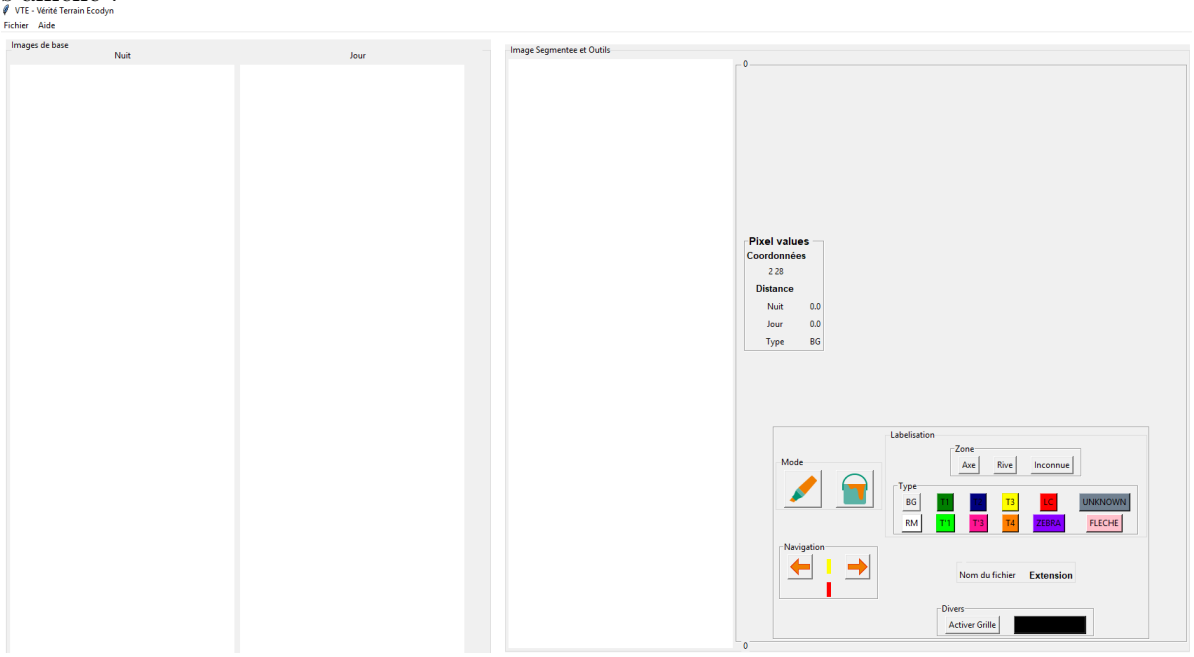
Pour la mise en route de **VTEcodyn3**, on doit avoir les fichiers suivant dans un même dossier : `VT_Ecodyn3.py`, `VT_Ecodyn_Model.py`, `VT_Ecodyn_controler.py` qui correspondent à l'interface graphique du logiciel codé en **Python** à l'aide de la bibliothèque **Tkinter**, ainsi que les fichiers `segmentation.py`, `get_Data.py`, `convert_json_eco3.py` qui correspondent à la segmentation automatique utilisé dans le logiciel. Dans ce même dossier, pour pouvoir faire la **segmentation**, on doit avoir un fichier `.h5` qui correspond au modèle utilisé pour la **segmentation**. De plus, dans ce dossier, on doit avoir un dossier se nommant **'imgsGui'** qui correspondant aux petites icônes/symboles utilisées dans le logiciel.



Nom	Modifié le	Type	Taille
__pycache__	27/04/2022 10:44	Dossier de fichiers	
imgsGui	25/04/2022 10:41	Dossier de fichiers	
Note.txt	14/04/2022 11:03	Document texte	1 Ko
saved-model2_troncon_aug2_32_256.h5	12/04/2022 12:00	Fichier H5	391 Ko
convert_json_eco3.py	20/04/2022 15:36	Fichier PY	3 Ko
get_Data.py	16/03/2022 09:46	Fichier PY	8 Ko
segmentation.py	21/04/2022 15:31	Fichier PY	2 Ko
VT_Ecodyn_controler.py	26/04/2022 15:51	Fichier PY	21 Ko
VT_Ecodyn_Model.py	27/04/2022 10:44	Fichier PY	9 Ko
VT_Ecodyn3.py	27/04/2022 10:50	Fichier PY	63 Ko

Imports utilisés

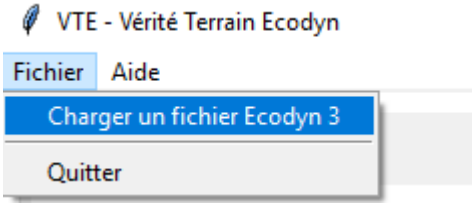
On ouvre `VT_Ecodyn3.py` via **Spyder** puis on l'exécute. Pour s'y faire, dans l'environnement d'exécution, il doit y avoir installé **numpy**, **tensorflow**, **sqlite**. Une fois le script lancé, la fenêtre suivante s'affiche :

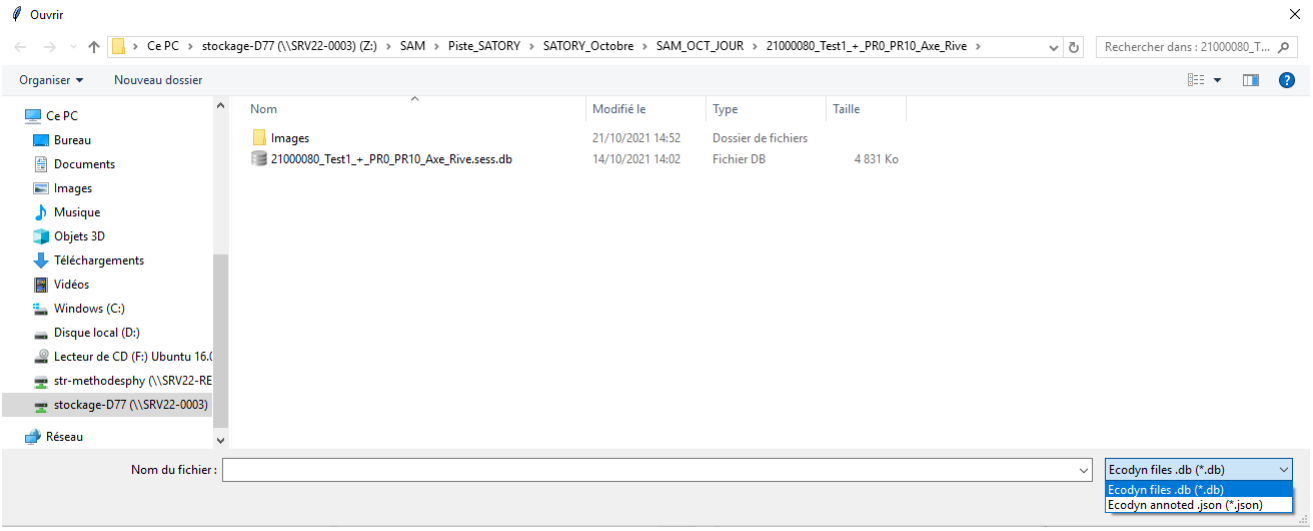


Utilisation du logiciel

Chargement d'un fichier

Une fois le logiciel lancé, on ouvre un fichier `.db` ou un fichier `.json` depuis le dossier souhaité.



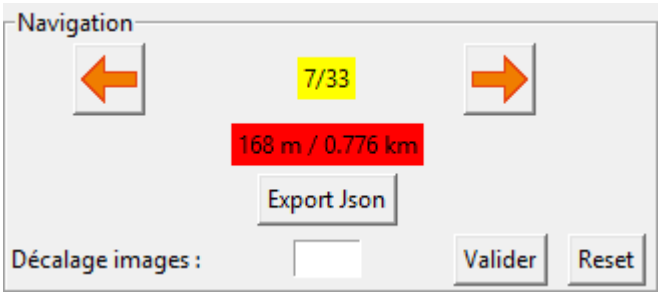


Images

Afin d'avoir les images de la caméra associées aux données d'Ecodyn, le dossier dans lequel on charge le fichier `.db` ou `.json` doit contenir un dossier **Images**. Dans le cas contraire, on aurait pas d'images de visualisations mais ce n'est pas nécéssaire pour l'annotation. Certains fichiers `.db` demandent si l'on considère le côté **Rive** ou le côté **Axe** de la route. Dans ce cas, les mesures ont été faites avec deux **Ecodyn** de chaque côté du véhicule et les deux données sont stockées dans le fichier `.db`, d'où la question du logiciel. Une fois le fichier chargé, la partie droite de l'interface se présente de la forme suivante (avec des images) :

Navigation

Toute la route considérée est divisé en **tronçons**, c'est à dire en plusieurs rectangles d'une hauteur d'une vingtaine de mètres.



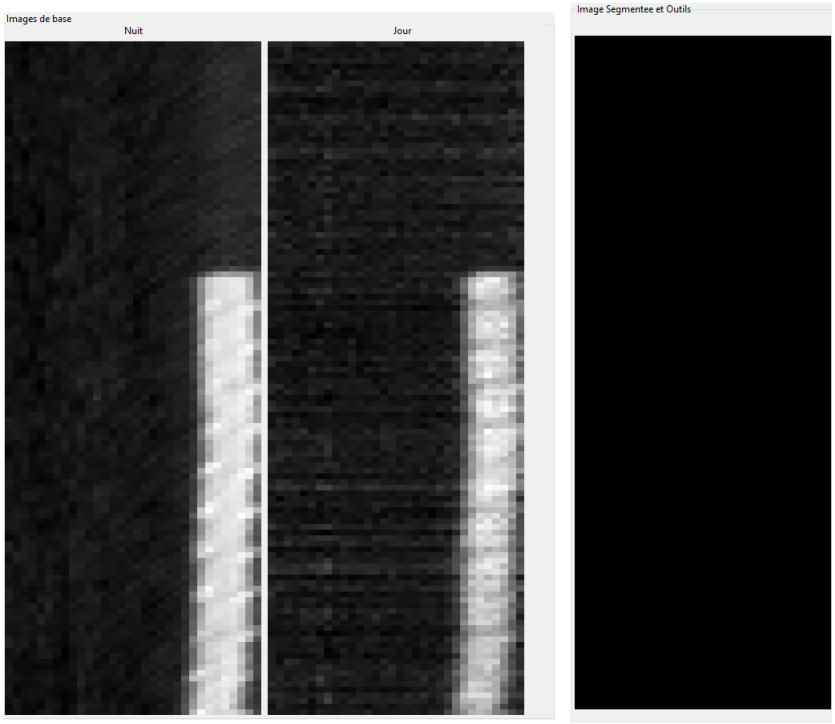
Dans la partie **'Navigation'**, on voit le nombre de **tronçons** totale, ici dans ce cas c'est 33, ainsi que le tronçon actuel, qui ici vaut 7. De plus, dans cette partie, on peut **décaler** les images visualisées. Pour des raisons de simplicités, on peut que décaler les images d'un décalage de 1, de 2 ou de 3 vers l'avant actuellement. Une fois le décalage appliqué en cliquant sur **'Valider'**, on peut revenir sur l'ordre initial des images en cliquant sur **Reset**. Ce décalage a été mit en place puisqu'on a reperé que les images de la caméra ne collent pas exactement aux mesures associées, d'où l'utilité du décalage. En haut et en bas tout à gauche, on voit la distance parcourue du véhicule dans le tronçon considéré.

A côté de la partie **'Navigation'**, on trouve le nom du fichier considéré, son extension `.db` ou `.json` ainsi que son côté **Axe** ou **Rive** :

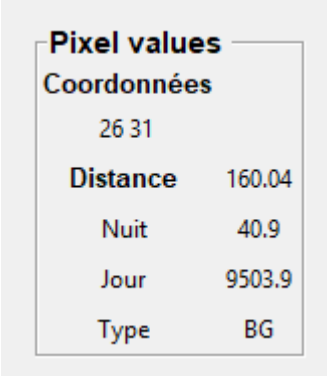
Les fichiers `.db` issue de deux mesures **Ecodyn3** ont dans leur nom de fichier comme fin **'_Axe_Rive'**. Dans l'autre cas, on a soit **'_Axe'** soit **'_Rive'**. De plus, on peut se déplacer d'un **tronçon** à un autre en spécifiant le nombre souhaité à côté de **'Choix du tronçon'** en ne dépassant pas le nombre maximal de **tronçons**.

Pixel values

La partie gauche du logiciel se présente de la forme suivante : (ici un bout de la route **A35**)

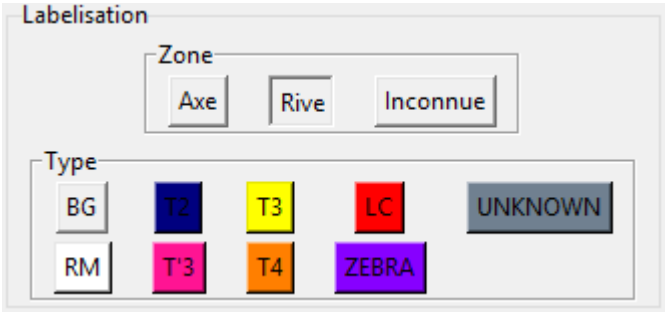


Le premier **tronçon** correspond à 'l'image' dit de **'Nuit'**, le deuxième à 'l'image' dit de **'Jour'** et la dernière à **l'image segmentée** par l'utilisateur, dans notre cas rien n'a été annoté. A noter que les deux premiers **tronçons** ne sont pas des images mais des signaux issus de **Ecodyn3**. Lorsqu'on navigue avec la souris sur un des trois **tronçons**, on obtient des coordonnées ainsi que des valeurs de rétroréflexions dans la partie **'Pixel values'** :



Type

De plus, le **type** de pixel sur lequel la souris pointe est aussi marqué, ici le type est **BG** pour **Background** c'est à dire, pas de marquage. Les différents **types** se retrouvent dans la section **'type'** :

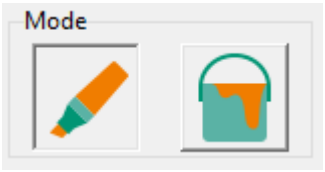


Le logiciel définit la **'Zone'** dans lequel on se trouve, **Axe** ou **Rive**, et laisse à disposition les marquages qui peuvent se trouver sur ce côté. Le bouton **'Inconnue'** permet d'afficher tous les types de marquages.

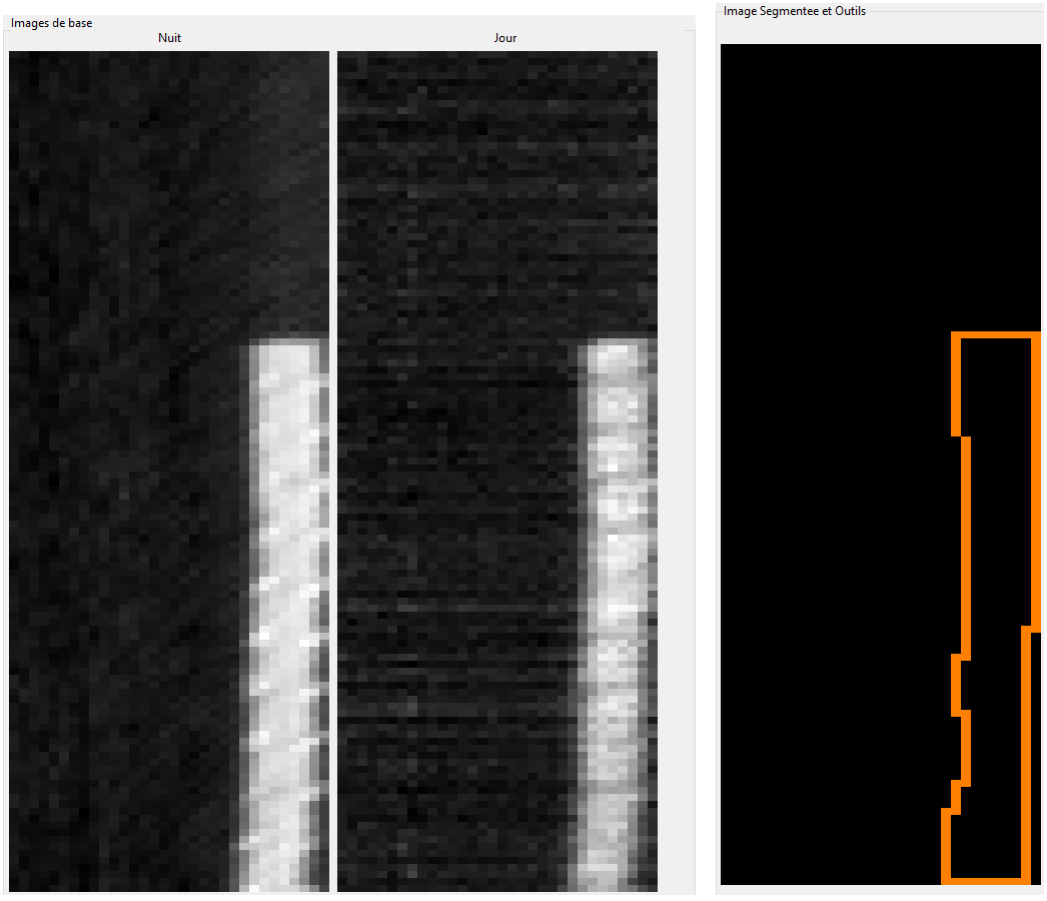
Annotation

Annotation

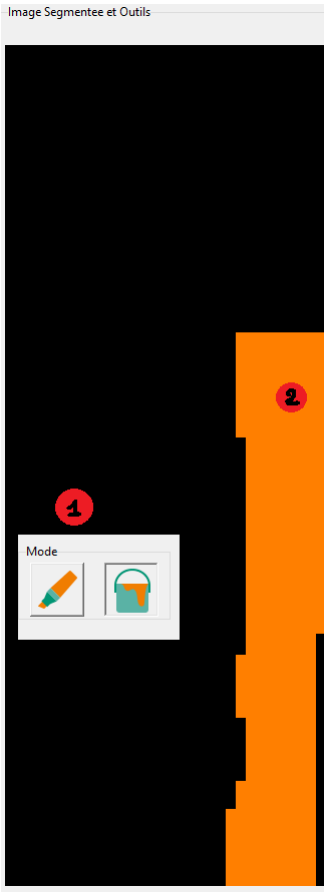
Pour l'annotation, nous allons principalement utilisé les outils dans **'Mode'** :



Le **crayon** à gauche et le **seau** à droite. On clique sur le crayon et on choisit le type de marquage associé puis on délimite le marquage à suivant les signaux d'Ecodyn (repéré par l'intensité blanche de 'l'image'). Lorsque les 'images' de **'Nuit'** et de **'Jour'** sont de mauvaises qualités, les valeurs de rétroréfléxions ainsi que les images de la caméra peuvent aider.



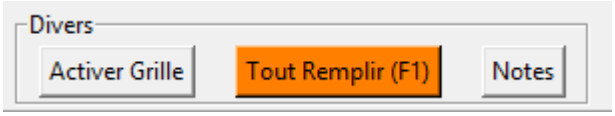
Ensuite, on utilise le **seau** afin de remplir toute la zone de marquage :



Lors du remplissage à l'aide du pot de peinture, il est conseillé de fermer entièrement le détourage du marquage au préalable, en particulier aux bords du tronçon. En effet, sans cette précaution, le remplissage automatique risque de dépasser sur le tronçon précédant ou suivant.

Divers

De plus, dans la section **'Divers'**, on peut activer une **grille** qui permet desfois de faciliter l'annotation. Par contre, elle fait un peu mal aux yeux.. De plus, il existe un bouton **'Tout Remplir'** qui permet de remplir toutes les composantes connexes du tronçon avec une couleur choisie.



Le bouton **'Notes'** permet d'ouvrir un fichier `.txt` rapidement afin de lister des anomalies sur un **tronçon** par exemple des valeurs de rétroréflexions trop grandes, de l'herbe sur les images etc...

Schéma itinéraire

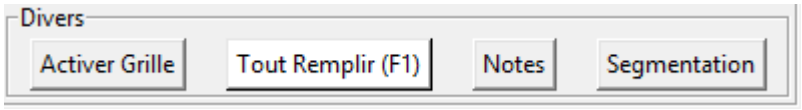
Un **Schéma itinéraire** a été ajouté. Sur celui, on voit les marquages annotés sous la forme de 3 voies. Ce schéma se déplace en même temps que le déplacement en tronçon. Lorsque plus de 4 pixels de types différents sont sur une même voie, le schéma indiquera du marquage inconnu.

Export JSON

Lorsqu'on a fini l'annotation (ou non), on peut exporter nos données en fichier `.json` à l'aide du bouton **'Export Json'** dans la partie **'Navigation'**. Le logiciel crée donc un nouveau fichier `.json` dans le même dossier où se trouve le fichier `.db` avec un nom qui se termine par **_Axe** ou **_Rive** selon le côté où on se trouve, et donc on aura plus de **_Axe_Rive** comme pour certains `.db`. On peut recharger ensuite ce fichier sans aucun problème.

Segmentation automatique et identification

Afin de faciliter l'annotation, on a décidé de mettre en place un modèle qui permet de faire la segmentation automatique, ici on utilise un modèle **Unet** entrainé sur nos données annotées à la main. Ce modèle correspond grosso-modo au fichier `.h5`. Ensuite, on clique sur le bouton **'Segmentation'** qui se trouve dans **'Divers'** :



Une fois la segmentation terminée, une **identification** des marquages est fait sur critère géométrique. L'algorithme d'**identification** parcourt chaque marquage et calcule la longueur et l'inter-distance entre un marquage et le marquage qui le précède. À partir de là, il identifie le type. Enfin, le logiciel crée un nouveau fichier `.json` en ajoutant dans le nom du fichier de base **_new** à la fin. Une fois la fenêtre pop-up fermée, le logiciel ouvre directement ce nouveau fichier afin de pouvoir **visualiser la segmentation/identification et la corriger**.

On se retrouve avec un dossier sous la forme suivante :

Nom	Modifié le	Type	Taille
Images	04/04/2022 16:41	Dossier de fichiers	
22000010_A35_-_PR231_PR223_Axe_BAU.sess.db	23/03/2022 11:07	Fichier DB	38 009 Ko
22000010_A35_-_PR231_PR223_Axe_BAU.sess.db-shm	27/04/2022 14:32	Fichier DB-SHM	32 Ko
22000010_A35_-_PR231_PR223_Axe_BAU.sess.db-wal	27/04/2022 14:32	Fichier DB-WAL	0 Ko
22000010_A35_-_PR231_PR223_Axe.json	26/04/2022 09:31	Fichier JSON	176 711 Ko
22000010_A35_-_PR231_PR223_Axe_new.json	21/04/2022 15:28	Fichier JSON	176 725 Ko
22000010_A35_-_PR231_PR223_BAU.json	26/04/2022 09:43	Fichier JSON	177 484 Ko
22000010_A35_-_PR231_PR223_BAU_new.json	27/04/2022 16:03	Fichier JSON	176 725 Ko

L'annotation du modèle peut être mauvaise, mais l'amélioration de celui-ci est en cours. Dans le fichier `segmentation.py`, on peut définir le modèle qu'on choisit (c'est-à-dire le fichier `.h5`) ainsi qu'un seuil :

```python
from convert_json_eco3 import convert
from tensorflow import keras
from get_Data import extract
import numpy as np
import os

model = keras.models.load_model("saved-model2_troncon_aug2_32_256.h5")
seuil = 0.5

def segmentation(directory, name):
    ....
```

Analyse Sémantique

Le bouton **'Analyse sémantique'** permet d'analyser la **segmentation/identification** obtenue. Un réseau de neurones découpe la route en séquences et identifie les marquages ayant des erreurs dû à soit l'**identification**, soit des erreurs liées à la route. Des marqueurs sur l'interface graphique feront référence aux faux marquages. (**EN COUR**)

Cette action va aussi modifier le fichier `.json`. Les marquages seront listés dans l'attribut `Detected` et la clé `'OK'` dira si le marquage est faux (1) ou non (0).

```
"Detected": {
            "Marquage 1": {
                  "Position": 6160.0,
                  "Type": "T2",
                  "OK": 0
            },
            "Marquage 2": {
                  "Position": 12095.800000000001,
                  "Type": "UNKNOWN",
                  "OK": 0
            },
            "Marquage 3": {
                  "Position": 13874.0,
                  "Type": "T2",
                  "OK": 1
            },
            "Marquage 4": {
                  "Position": 4.4,
                  "Type": "UNKNOWN",
                  "OK": 1
            },
            "Marquage 5": {
                  "Position": 42.400000000000006,
                  "Type": "T4",
                  "OK": 1
            },
            "Marquage 6": {
                  "Position": 100.4,
```

```
                    "Type": "T4",
                    "OK": 1
                },
                "Marquage 7": {
                    "Position": 158.0,
                    "Type": "T4",
                    "OK": 1
                    ..
```

C Bibliographie

Références

[1] "Ademe." [Online]. Available : https://www.ademe.fr/

[2] "Master csmi, Contact : `prudhomm@math.unistra.fr`." [Online]. Available : https://www.unistra.fr/etudes/decouvrir-nos-formations/par-facultes-ecoles-instituts/sciences-technologies/ufr-de-mathematique-et-dinformatique/ufr-de-mathematique-et-dinformatique/cursus/ME195?cHash=3aa4f04702a03e944ed933056abe17f2

[3] "Cerema, climat et territoires de demain. Aménagement et résilience." [Online]. Available : https://www.cerema.fr/fr

[4] "ENDSUM : Évaluation Non Destructive des StrUctures et des Matériaux." [Online]. Available : http://www.cerema.fr/fr/innovation-recherche/recherche/equipes/endsum-evaluation-non-destructive-structures-materiaux

[5] "Instruction interministÉrielle sur la signalisation routiÈre," Oct. 1963.

[6] "Ascquer - ASsociation pour la Certification et la QUalification des Equipements de la Route." [Online]. Available : https://ascquer.fr/

[7] I. des routes des rues et des infrastructures pour la mobilité, "Guide de la signalisation horizontale Éléments de choix et de mise en œuvre des produits de marquage routier," Décembre 2019.

[8] "Artificial intelligence - Wikipedia." [Online]. Available : https://en.wikipedia.org/wiki/Artificial_intelligence

[9] "Welcome to Python.org." [Online]. Available : https://www.python.org/

[10] "Anaconda | The World's Most Popular Data Science Platform." [Online]. Available : https://www.anaconda.com/

[11] "TensorFlow." [Online]. Available : https://www.tensorflow.org/?hl=fr

[12] "NVIDIA : leader mondial du calcul informatique basé sur l'intelligence artificielle." [Online]. Available : https://www.nvidia.com/fr-fr/

[13] "Nf en 1436, produits de marquage routier - performances des marquages appliqués sur la route et méthodes d'essai."

[14] E. Krine, Stresser, Redondin, and Muzet, "Évaluation des marquages routiers dans la perspective de croisement avec les véhicules autonomes," p. 45, Spring 2021.

[15] VECTRA, "Auscultation des chaussées, ecodyn3." 2014.

[16] "Brevet - image device and method for generating an image of road marking. wo2013007955," Jan. 2013.

[17] "tkinter — Interface Python pour Tcl/Tk — Documentation Python 3.10.6." [Online]. Available : https://docs.python.org/fr/3/library/tkinter.html

[18] W. Hamama, "Détection automatique du marquage au sol," Sep. 2021.

[19] "Modèle-vue-contrôleur," Aug. 2022, page Version ID : 195765887. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=Mod%C3%A8le-vue-contr%C3%B4leur&oldid=195765887

[20] "Structurez une application avec le pattern d'architecture MVC." [Online]. Available : https://openclassrooms.com/fr/courses/7160741-ecrivez-du-code-python-maintenable/7188702-structurez-une-application-avec-le-pattern-d-architecture-mvc

[21] "Machine learning," Aug. 2022, page Version ID : 1103198106. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=1103198106

[22] "Deep Learning avec Keras et TensorFlow - Mise en oeuvre et cas concrets - Livre et ebook Management des systèmes d'information de Aurélien Géron - Dunod." [Online]. Available : https://www.dunod.com/sciences-techniques/deep-learning-avec-keras-et-tensorflow-mise-en-oeuvre-et-cas-concrets

[23] I. T. Brain and F. Rosenblatt, "The Perceptron : A Probabilistic Model for Information Storage and Organization."

[24] "Deep learning," Aug. 2022, page Version ID : 1103718048. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Deep_learning&oldid=1103718048

[25] T. rédac, "Deep Learning ou Apprentissage Profond : qu'est-ce que c'est ?" Sep. 2020. [Online]. Available : https://datascientest.com/deep-learning-definition

[26] "Overfitting - Wikipedia." [Online]. Available : https://en.wikipedia.org/wiki/Overfitting

[27] "Training, validation, and test data sets," Aug. 2022, page Version ID : 1101844304. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Training,_validation,_and_test_data_sets&oldid=1101844304

[28] "Parameters, Hyperparameters, Machine Learning | Towards Data Science." [Online]. Available : https://towardsdatascience.com/parameters-and-hyperparameters-aa609601a9ac

[29] "Feature Engineering for Machine Learning [Book]." [Online]. Available : https://www.oreilly.com/library/view/feature-engineering-for/9781491953235/

[30] "Feature learning - Wikipedia." [Online]. Available : https://en.wikipedia.org/wiki/Feature_learning

[31] "Incremental learning," Apr. 2022, page Version ID : 1080634548. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Incremental_learning&oldid=1080634548

[32] "Qu'est-ce que les réseaux neuronaux ? - France | IBM." [Online]. Available : https://www.ibm.com/fr-fr/cloud/learn/neural-networks

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout : A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014. [Online]. Available : http://jmlr.org/papers/v15/srivastava14a.html

[34] S. Ioffe and C. Szegedy, "Batch Normalization : Accelerating Deep Network Training by Reducing Internal Covariate Shift," Mar. 2015, arXiv :1502.03167 [cs]. [Online]. Available : http://arxiv.org/abs/1502.03167

[35] "Batch normalization - Wikipedia." [Online]. Available : https://en.wikipedia.org/wiki/Batch_normalization

[36] "Comment lire et exploiter une matrice de confusion ?" [Online]. Available : https://datascientest.com/matrice-de-confusion

[37] "Self-driving car - Wikipedia." [Online]. Available : https://en.wikipedia.org/wiki/Self-driving_car

[38] M. Soilán, B. Riveiro, J. Martínez-Sánchez, and P. Arias, "Segmentation and classification of road markings using MLS data," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 123, pp. 94–103, Jan. 2017. [Online]. Available : https://linkinghub.elsevier.com/retrieve/pii/S0924271616303173

[39] C. Chun, T. Lee, S. Kwon, and S.-K. Ryu, "Classification and Segmentation of Longitudinal Road Marking Using Convolutional Neural Networks for Dynamic Retroreflection Estimation," *Sensors*, vol. 20, no. 19, p. 5560, Jan. 2020, number : 19 Publisher : Multidisciplinary Digital Publishing Institute. [Online]. Available : https://www.mdpi.com/1424-8220/20/19/5560

[40] L. Deng, M. YANG, h. Bing, T. Li, H. Li, and C. Wang, "Semantic Segmentation-Based Lane-Level Localization Using Around View Monitoring System," *IEEE Sensors Journal*, vol. 19, pp. 10 077–10 086, Jul. 2019.

[41] L. G. Shapiro and G. C. Stockman, ""computer vision", pp 279˘325, new jersey, prentice-hall, isbn 0-13-030796-3," Jan. 2001.

[42] A. Rouini, "Evaluation of DRLSE Segmentation Method for Medical Images," Dec. 2018.

[43] D. Guo, Y. Pei, K. Zheng, H. Yu, Y. Lu, and S. Wang, "Degraded Image Semantic Segmentation With Dense-Gram Networks," *IEEE Transactions on Image Processing*, vol. 29, pp. 782–795, 2020, conference Name : IEEE Transactions on Image Processing.

[44] J. Yi, P. Wu, M. Jiang, D. Hoeppner, and D. Metaxas, "Instance Segmentation of Neural Cells," Sep. 2018.

[45] V. Varatharasan, H.-S. Shin, A. Tsourdos, and N. Colosimo, "Improving Learning Effectiveness For Object Detection and Classification in Cluttered Backgrounds," in *2019 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED UAS)*, Nov. 2019, pp. 78–85, arXiv :2002.12467 [cs, eess]. [Online]. Available : http://arxiv.org/abs/2002.12467

[46] "Convolutional neural network," Aug. 2022, page Version ID : 1103715044. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=1103715044

[47] "Convolution - Wikipedia." [Online]. Available : https://en.wikipedia.org/wiki/Convolution

[48] "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way | by Sumit Saha | Towards Data Science." [Online]. Available : https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[49] "Softmax function," Aug. 2022, page Version ID : 1102593845. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Softmax_function&oldid=1102593845

[50] "[1505.04597] U-Net : Convolutional Networks for Biomedical Image Segmentation." [Online]. Available : https://arxiv.org/abs/1505.04597

[51] "A survey on Image Data Augmentation for Deep Learning | Journal of Big Data | Full Text." [Online]. Available : https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0

[52] "U-Net : le réseau de neurones populaire en Computer Vision - Blent.ai." [Online]. Available : https://blent.ai/unet-computer-vision/

[53] "Autoencoder," Aug. 2022, page Version ID : 1102065250. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Autoencoder&oldid=1102065250

[54] "Transposed Convolutions explained with... MS Excel! | by Thom Lane | Apache MXNet | Medium." [Online]. Available : https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8

[55] M. Drozdzal, E. Vorontsov, G. Chartrand, S. Kadoury, and C. Pal, "The Importance of Skip Connections in Biomedical Image Segmentation," Aug. 2016.

[56] V. Badrinarayanan, A. Kendall, and R. Cipolla, "SegNet : A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 12, pp. 2481–2495, Dec. 2017, conference Name : IEEE Transactions on Pattern Analysis and Machine Intelligence.

[57] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Apr. 2015, number : arXiv :1409.1556 arXiv :1409.1556 [cs]. [Online]. Available : http://arxiv.org/abs/1409.1556

[58] Fezan, "Understanding of Semantic Segmentation & How Segnet Model work to perform Semantic Segmentation," Oct. 2019.

[59] P. Bosilj, E. Aptoula, T. Duckett, and G. Cielniak, "Transfer learning between crop types for semantic segmentation of crops versus weeds in precision agriculture," *Journal of Field Robotics*, vol. 37, no. 1, pp. 7–19, 2020, _eprint : https ://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21869. [Online]. Available : https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21869

[60] "Natural language processing," Jul. 2022, page Version ID : 1100496063. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Natural_language_processing&oldid=1100496063

[61] Y. Goldberg, "A Primer on Neural Network Models for Natural Language Processing," Oct. 2015, number : arXiv :1510.00726 arXiv :1510.00726 [cs]. [Online]. Available : http://arxiv.org/abs/1510.00726

[62] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[63] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, "Grammar as a Foreign Language," Jun. 2015, number : arXiv :1412.7449 arXiv :1412.7449 [cs, stat]. [Online]. Available : http://arxiv.org/abs/1412.7449

[64] P. Amayo, T. Bruls, and P. Newman, "Semantic Classification of Road Markings from Geometric Primitives," in *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. Maui, HI : IEEE, Nov. 2018, pp. 387–393. [Online]. Available : https://ieeexplore.ieee.org/document/8569382/

[65] B. Mathibela, P. Newman, and I. Posner, "Reading the Road : Road Marking Classification and Interpretation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 4, pp. 2072–2081, Aug. 2015. [Online]. Available : http://ieeexplore.ieee.org/document/7055289/

[66] "One-hot," Jul. 2022, page Version ID : 1101284095. [Online]. Available : https://en.wikipedia.org/w/index.php?title=One-hot&oldid=1101284095

[67] "Word embedding," Aug. 2022, page Version ID : 1103958023. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Word_embedding&oldid=1103958023

[68] W. Koehrsen, "Neural Network Embeddings Explained," Oct. 2018. [Online]. Available : https://towardsdatascience.com/neural-network-embeddings-explained-4d028e6f0526

[69] "Deep Learning : les réseaux de neurones récurrents (RNN)," Aug. 2021, section : IA/Data Science. [Online]. Available : https://datavalue-consulting.com/deep-learning-reseaux-neurones-recurrents-rnn/

[70] "DataScienceToday - Réseaux neuronaux récurrents et LSTM." [Online]. Available : https://datasciencetoday.net/index.php/fr/machine-learning/148-reseaux-neuronaux-recurrents-et-lstm

[71] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986, place : United Kingdom Publisher : Nature Publishing Group.

[72] "(PDF) Learning long-term dependencies with gradient descent is difficult." [Online]. Available : https://www.researchgate.net/publication/5583935_Learning_long-term_dependencies_with_gradient_descent_is_difficult

[73] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training Recurrent Neural Networks," Feb. 2013, number : arXiv :1211.5063 arXiv :1211.5063 [cs]. [Online]. Available : http://arxiv.org/abs/1211.5063

[74] J. S. Sepp Hochreiter, "Long shot-term memory."

[75] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, "Learning Precise Timing with LSTM Recurrent Networks," p. 29.

[76] "How LSTM networks solve the problem of vanishing gradients | by Nir Arbel | DataDrivenInvestor." [Online]. Available : https://medium.datadriveninvestor.com/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577?gi=982391199e38

[77] "Attention Is All You Need." [Online]. Available : https://ar5iv.labs.arxiv.org/html/1706.03762

[78] "Transformer (machine learning model)," Aug. 2022, page Version ID : 1104139625. [Online]. Available : https://en.wikipedia.org/w/index.php?title=Transformer_(machine_learning_model)&oldid=1104139625

[79] J. Brownlee, "A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size," Jul. 2017. [Online]. Available : https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

[80] "Normalization vs Standardization — Quantitative analysis," section : 2019 Apr Tutorials, Overviews.

[81] S. Węglarczyk, "Kernel density estimation and its application," *ITM Web of Conferences*, vol. 23, p. 00037, 2018, publisher : EDP Sciences. [Online]. Available : https://www.itm-conferences.org/articles/itmconf/abs/2018/08/itmconf_sam2018_00037/itmconf_sam2018_00037.html

[82] "2.8. Density Estimation — scikit-learn 1.1.2 documentation." [Online]. Available : https://scikit-learn.org/stable/modules/density.html

[83] L. Wang, C. Wang, Z. Sun, and S. Chen, "An Improved Dice Loss for Pneumothorax Segmentation by Mining the Information of Negative Areas," *IEEE Access*, vol. PP, pp. 1–1, Aug. 2020.

[84] L. Lambrecht, "Traitements avancés des marquages routiers," p. 31, October 2019.

[85] "Classification de texte avec un RNN | Text." [Online]. Available : https://www.tensorflow.org/text/tutorials/text_classification_rnn?hl=fr

[86] K. Team, "Keras documentation : Text classification with Transformer." [Online]. Available : https://keras.io/examples/nlp/text_classification_with_transformer/

[87] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," Jan. 2018, arXiv :1703.06870 [cs]. [Online]. Available : http://arxiv.org/abs/1703.06870

[88] C. Holler, "Notice vérité terrain ecodyn 2022."

[89] "Apprentissage par renforcement hors ligne," Apr. 2022, page Version ID : 192489113. [Online]. Available : https://fr.wikipedia.org/w/index.php?title=Apprentissage_par_renforcement_hors_ligne&oldid=192489113

[90] D. P. Kingma and J. Ba, "Adam : A Method for Stochastic Optimization," Jan. 2017, number : arXiv :1412.6980 arXiv :1412.6980 [cs] version : 8. [Online]. Available : http://arxiv.org/abs/1412.6980