Grenoble INP – ENSIMAG
École Nationale Supérieure d'Informatique et de Mathématiques Appliquées

Rapport de projet de fin d'études

CERFACS

Exploration de la capacité du Deep Learning à assister des simulations de mécanique des fluides

Nicolas Cazard
3e année – Option ISI

01 février 2018 – 31 juillet 2018

CERFACS
42 avenue Gaspard Coriolis
31057 Toulouse Cedex

Responsable de stage
Corentin Lapeyre
Tuteur de l'école
Jean-Guillaume Dumas

Table des matières

Table des figures

1 Contexte du stage

Le CERFACS (Centre de Recherche et de Formation Avancée en Calcul Scientifique) est un centre de recherche fondamentale et appliquée, créé en 1987, qui est spécialisé dans la modélisation et la simulation numérique.
Le CERFACS, de par ses moyens et son savoir-faire en calcul haute performance traite des grands problèmes scientifiques et techniques de recherche publique et industrielle. Au sein du CERFACS travaillent en coopération des physiciens, des mathématiciens appliqués, des analystes numériques et des informaticiens qui conçoivent et développent des méthodes et solutions logicielles innovantes répondant aux besoins des secteurs de l'aéronautique, du spatial, du climat, de l'énergie et de l'environnement.
Ce laboratoire travaille sur des projets d'envergure en forte interaction avec ses sept actionnaires : AIRBUS Group, CNES, EDF, Météo France, ONERA, SAFRAN et TOTAL et a noué des partenariats avec le CNRS, l'IRIT, le CEA et l'INRIA. Environ 150 personnes y travaillent, dont 130 chercheurs et ingénieurs venant de divers pays dans le monde. Pour ce qui est de sa composition interne, le CERFACS est composé de 5 équipes, CFD (Computational Fluid Dynamics), CSG (Computer Support Group), GLOBC (Modélisation du climat et de ses changements), Aviation et Environnement, et Algorithmes Parallèles.
J'ai intégré l'équipe CSG au cours de ce stage, mais j'ai également travaillé en étroite collaboration avec d'autres équipes : l'équipe CFD pour des applications physiques concernant la turbulence et la combustion des modèles que l'on souhaite mettre en place et avec l'équipe GLOBC pour une problématique concernant les feux de forêts.

En terme de ressources, le CERFACS dispose de deux super calculateurs fournissant une capacité crête agrégée d'environ 330 Tflops permettant de traiter la majeure partie des besoins en simulations des différentes équipes. Concernant les calculs sur GPU qui nous intéresseront pour du Deep Learning, le CERFACS met à notre disposition une carte NVIDIA TESLA V100 d'une capacité de 16 Tflops en simple précision et pouvant aller jusqu'à 120 Tflops en demi-précision pour du Deep Learning, grâce à ses 640 Tensor Cores.

2 Avant propos

Un certain nombre de termes techniques généralement en anglais (MaxPooling, UpSampling, L2 Regularization, Batch, Epochs...) sont utilisés dans ce rapport et supposés connus par le lecteur. Ces termes techniques sont mis en gras, et tout lecteur non spécialiste du domaine du Deep Learning est invité à consulter les ouvrages [7, 3], pour y trouver la définition de ces termes.

3 Introduction

3.1 Problématique

Dans le domaine de la combustion turbulente, connaître précisément la surface de la flamme est primordial. En effet, lors de son interaction avec les écoulements turbulents, la flamme est froissée et plissée, ce qui augmente grandement sa surface de contact avec les autres fluides. Or, dans le cas de la combustion, la libération de chaleur chimique et donc l'apport d'énergie au reste du flux par la flamme est proportionnel à sa surface. C'est pourquoi ce plissement de la flamme doit être connu avec précision dans des simulations numériques, en chaque point du maillage dans l'espace et dans le temps.
Lorsque toutes les échelles de la simulation sont résolues, ce que l'on appellera par la suite une DNS (Direct Numerical Simulation [6, 15]) est réalisée.
Or, dans des configurations réalistes, il n'est pas possible de résoudre l'ensemble des équations de Navier-Stokes, qui régissent ces écoulements à toutes les échelles et pour rendre les simulations possibles, on réduit généralement le coût de calcul en ignorant les plus petites échelles qui sont les plus coûteuses à résoudre, via un filtrage passe-bas de ces équations de Navier-Stokes. Une telle simulation où les plus petites échelles de la flamme ne sont pas résolues mais simplement modélisées s'appelle une Large Eddy Simulation [4, 13] que l'on appellera LES dans la suite du rapport.

Lorsque une LES est réalisée, les plus petites échelles ne sont pas résolues mais seulement modélisées, ce qui lisse la flamme et réduit ainsi grandement sa surface de flamme totale. La plupart des approches de modélisation turbulentes sont donc basées sur une reconstruction de ce plissement de la flamme pour obtenir une surface de flamme correcte en effectuant des hypothèses dites de "flammelette"[14], dans laquelle le taux de réaction turbulent moyen est exprimé en termes de surface de la flamme.
La déformation et l'étalement de la flamme peuvent être analysés en observant le front de flamme d'un fluide pré mélangé dans un écoulement turbulent. L'un des avantages de cette approche particulière de la modélisation de la combustion turbulente est que l'on peut calculer les densités de surface de la flamme à partir de simulations plus raffinées (la DNS) et s'en servir pour évaluer la performance de nos modèles.

3.2 L'apport du Machine Learning

La plupart des modèles de combustion déjà existants pour essayer d'estimer le plissement d'une flamme sont des modèles algébriques qui estiment le plissement de sous-mailles à partir de grandeurs locales. Ces modèles sont peut coûteux mais ne permettent pas de capturer correctement la structure spatiale de l'écoulement.
Notre idée est de développer et d'entraîner des modèles de Machine Learning et Deep Learning sur des bases de données de simulation de flammes dans des champs turbulents pour construire des algorithmes plus efficaces. Le Machine Learning regroupant plusieurs domaines se recouvrant mutuellement et étant souvent confondus, il convient de définir un

peu mieux à quoi nous ferons référence par la suite avec chaque termes :

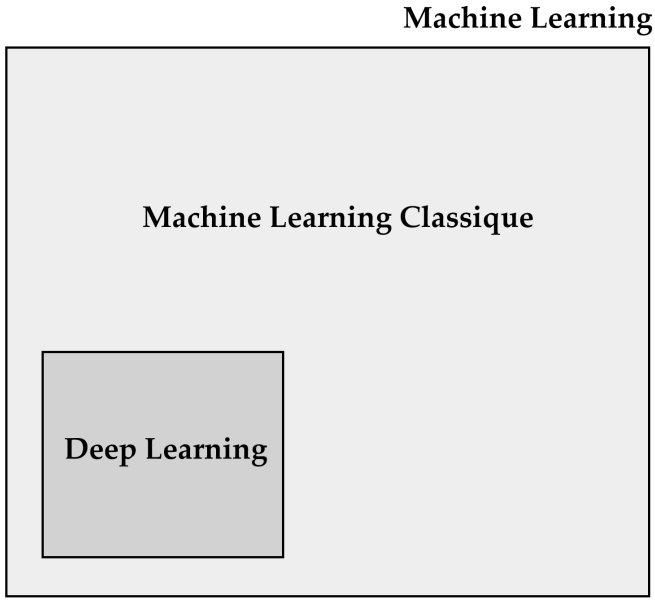


FIGURE 1 – Machine Learning, Machine Learning Classique et Deep Learning.

Nous parlons dans ce rapport de Deep Learning lorsque nous utilisons des modèles avec énormément de paramètres (souvent quelques millions) avec des entrées de très grandes dimensions, et de Machine Learning Classique lorsque les modèles utilisés sont plus petits et avec des entrées de dimensions plus faibles.
Le Machine Learning regroupe ces deux termes, et permet d'obtenir de meilleurs approximations de la surface de la flamme en chaque point que les méthodes existantes actuellement. En effet, des modèles de Machine Learning utilisant des successions de couches de convolution peuvent prendre en compte des informations topologiques telles que la courbure de la flamme, sa structure et son interaction avec la turbulence, dont on ne dispose pas avec des modèles algébriques.
Cependant, l'un des problèmes liés à l'application du Machine Learning au domaine de la physique est l'absence de données labellisées. En effet, alors que la physique numérique génère d'énormes quantités de données, elle reste largement "non étiquetées", c'est-à-dire qu'il n' y a pas eu de traitement humain de l'information au préalable, contrairement aux bases de données de photos annotées par exemple.
Pour palier à ce problème dans notre cas, des DNS de combustion turbulente dans différentes configurations sont utilisées comme **target** pour l'entraînement de nos modèles auquel on fournit uniquement la LES des champs équivalents en entrée. Une fois entraînés, nous pourrons alors comparer les prévisions de nos modèles sur de nouvelles données DNS dans des configurations jamais vues jusque-là ainsi qu'à d'autres méthodes d'estimation de la surface de flammes issues de la littérature.

Pour apprendre à mieux maîtriser les différents réseaux existants, mais également pour se faire une bonne idée des types de modèles les plus aptes à résoudre notre problème, différents modèles de Deep Learning ont été mis en place sur des cas plus simples que nous allons présenter dans la partie 4.
À l'issue de cette étude préliminaire, nous verrons quelles sont les méthodes que nous avons mises en place pour résoudre notre problème, à savoir la reconstruction du plissement d'une flamme LES la plus proche possible de celui de la flamme DNS équivalente.

4 Les réseaux de neurones appliqués à la traduction image à image

4.1 Introduction aux réseaux de neurones

À la différence de simples perceptrons multi-couches qui ont des difficultés à traiter des images de grande taille dues à la croissance exponentielle du nombre de leurs paramètres du fait que chaque couche soit complètement connectées à la suivante, les réseaux de convolution permettent de réduire drastiquement ces paramètres.
En effet, les couches de convolution qui consistent à appliquer des filtres entraînables que l'on déplace sur toute l'image, permettent d'avoir beaucoup moins de paramètres à entraîner et présentent de gros avantages par rapport aux perceptrons multi-couches : une invariance à la translation et un partage des poids par exemple, qui se révèlent très utiles dans le cas du traitement d'images. La plupart de nos modèles utilisent donc des successions de couches de convolution.
De plus, nous utilisons également entre ces différentes couches de convolution des opérations de **MaxPooling** pour diviser les dimensions de l'image d'entrée dans toutes les dimensions de l'espace, et des opérations d'**UpSampling**, réalisant l'opération inverse pour obtenir à nouveau les mêmes dimensions que l'image en entrée. Cela nous permet d'appliquer des couches de convolution sur des plus petites images et ainsi de réduire encore le nombre de paramètres total de nos modèles.

4.2 Les deux approches possibles

Pour résoudre notre problématique, à savoir le passage d'un champ contenant la densité de flamme non résolue sur un maillage LES (voir Fig. 14), à la densité de flamme complètement résolue sur le même maillage, deux approches semblent possibles :

- La génération conditionnelle, où notre objectif serait de recréer une réalisation possible du champ de densité complètement résolue sur un maillage DNS, puis d'interpoler ensuite ce maillage sur un maillage LES plus petit.
- Une approche type segmentation, où l'on réaliserait une régression linéaire multiple et où l'objectif serait d'obtenir directement en sortie de notre réseau la densité de flamme complètement résolue sur un maillage LES.

Nous allons parcourir ces deux approches dans les prochains chapitres.

4.3 La génération

4.3.1 Les auto-encodeurs classiques

Les auto-encodeurs sont des réseaux de neurones utilisés pour l'apprentissage non supervisé de caractéristiques des données qu'on lui donne en entrée.
Ils se composent d'un encodeur, qui va réduire la dimension de nos données et ainsi extraire les caractéristiques les plus importantes des données en entrée, et d'un décodeur, qui apprend ensuite à reconstituer les données initiales en utilisant cette représentation compressée des données qu'a fourni l'encodeur.
L'encodeur est une succession de couches de convolution et de couches de **MaxPooling** permettant d'extraire les caractéristiques importantes de l'image tout en réduisant sa dimension, alors que le décodeur réalise les opérations inverses afin de reconstruire une image, avec les mêmes dimensions que celle de départ.
Le principe de la génération d'images à l'aide d'auto-encodeurs est de développer un espace latent de représentation de faible dimension où n'importe quel point peut être mis en correspondance avec une image réaliste. En effet, une fois qu'un tel espace latent aura été mis en place, on pourra en échantillonner des points puis, en les décodant avec notre décodeur, générer de nouvelles images qui n'ont jamais été vues auparavant.
Mais en pratique, de simples auto-encodeurs ne conduisent pas à des espaces latents particulièrement utiles ou bien structurés, et ne peuvent donc pas être utilisés pour effectuer de la génération d'image.
C'est ici qu'interviennent les auto-encodeurs variationnels qui, grâce à quelques notions statistiques, forcent notre modèle à apprendre des espaces latents continus et très structurés, et nous permettent de pouvoir les utiliser pour de la génération d'images.

4.3.2 Les auto-encodeurs variationnels

Un auto-encodeur variationnel est un type d'auto-encodeur avec des contraintes supplémentaires sur l'encodage des données. Plus précisément, c'est un auto-encodeur qui apprend un modèle de variable latente pour les données encodées.
Ainsi, au lieu d'apprendre une fonction quelconque pour l'encodage des données, notre modèle apprend les paramètres d'une distribution de probabilité modélisant les données en entrée. De cette façon, nous obtenons un auto-encodeur avec un espace latent bien structuré, et nous pouvons ensuite utiliser uniquement la partie décodeur (voir Fig. 2) comme un modèle génératif en lui fournissant aléatoirement des points de la distribution latente qu'il aura apprise.
Le fonctionnement d'un auto-encodeur variationnel est le suivant. Tout d'abord, un encodeur semblable a celui d'un auto-encodeur classique transforme les échantillons d'entrée x en deux paramètres dans un espace latent, que l'on note z_{mean} et z_{sigma}. Nous échantillonnons ensuite au hasard des points z à partir de la distribution normale latente de la façon

suivante :

$$z = z_{mean} + e^{z_{sigma}} \cdot \epsilon \tag{1}$$

où ϵ est un tenseur normal aléatoire de petites valeurs.
Enfin, un décodeur qui réalise les opérations inverses de notre encodeur va mapper ces points de l'espace latent pour reconstruire des images similaires à celles d'entrée.
Comme ϵ est aléatoire, le processus garantit que deux points proches dans l'espace latent seront décodés en deux images très similaires. Cette continuité, combinée à la faible dimension de l'espace latent, force toutes les modifications de l'espace latent suivant un certain axe à avoir une répercussion ayant du sens sur l'image en sortie, par exemple une modification de la couleur des cheveux du blond vers le brun dans le cas de la reconstruction d'un visage.
L'entraînement de notre modèle se fait via deux fonctions de coût : une fonction de coût classique forçant la reconstruction en sortie à correspondre aux données en entrée (comme dans un auto-encodeur classique, l'écart aux moindres carrés sur chaque pixel par exemple) ; et une seconde, la divergence de Kullback-Leibler, qui est une mesure de la dissimilarité entre deux distributions, dans notre cas la distribution latente encodée et la distribution de nos données d'entrée. La divergence de Kullback-Leibler agit comme un terme de régularisation qui force notre modèle à apprendre un espace latent bien formé.
Pour effectuer de la génération de champs turbulents homogène et isotrope, nous avons mis en place le modèle suivant, en Fig. 2 :

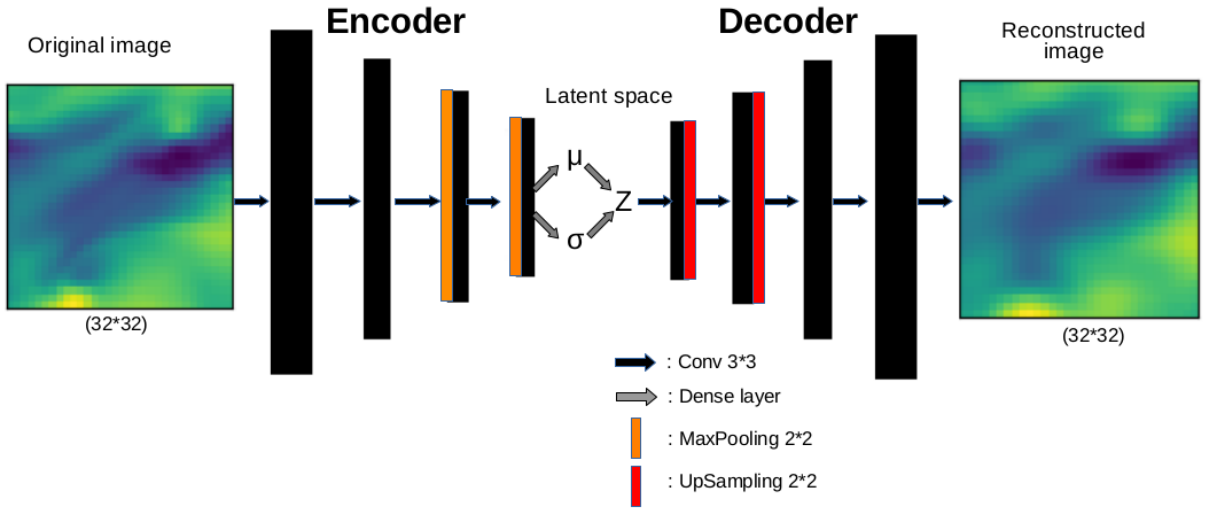


FIGURE 2 – Architecture du VAE mise en place.

La dimension de l'espace latent est très importante suivant la tâche que l'on souhaite effectuer ensuite, car c'est elle qui jouera sur la compression des données. Si la dimension est trop élevée, l'auto-encodeur sera inutile puisqu'il apprendra uniquement à reproduire les données vues en entrée sans chercher à les encoder et l'espace latent n'aura donc aucun sens, mais si cette est dimension est trop faible, notre auto-encodeur aura trop de difficulté à reconstituer l'image d'origine et l'image générée sera ainsi de faible qualité.
Si l'on entraîne notre auto-encodeur pour différentes dimensions de l'espace latent, voici les résultats que l'on obtient suivant les valeurs de la MSE (Mean-Squared Error) :

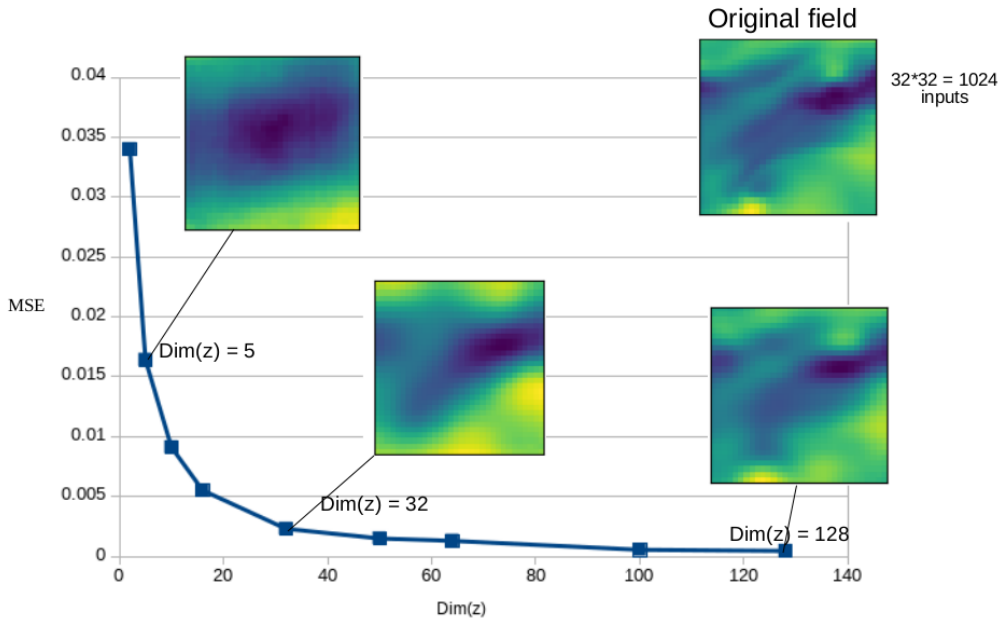


FIGURE 3 – Reconstruction de la turbulence pour différentes dimensions de l'espace latent.

Grâce au graphe Fig. 3, une valeur de dimension de l'espace latent de 64 aura été choisie, qui présente une MSE suffisamment faible tout en ayant un facteur de compression assez élevé, la dimension de l'image d'entrée étant de $32 \times 32 = 1024$.
Une fois l'auto-encodeur entraîné avec un espace latent de dimension 64, on conserve les poids que le réseau a appris et on utilise uniquement la partie décodeur en lui fournissant des valeurs aléatoires sur l'espace latent pour générer de nouveaux champs. Voici les images que l'on a obtenu :

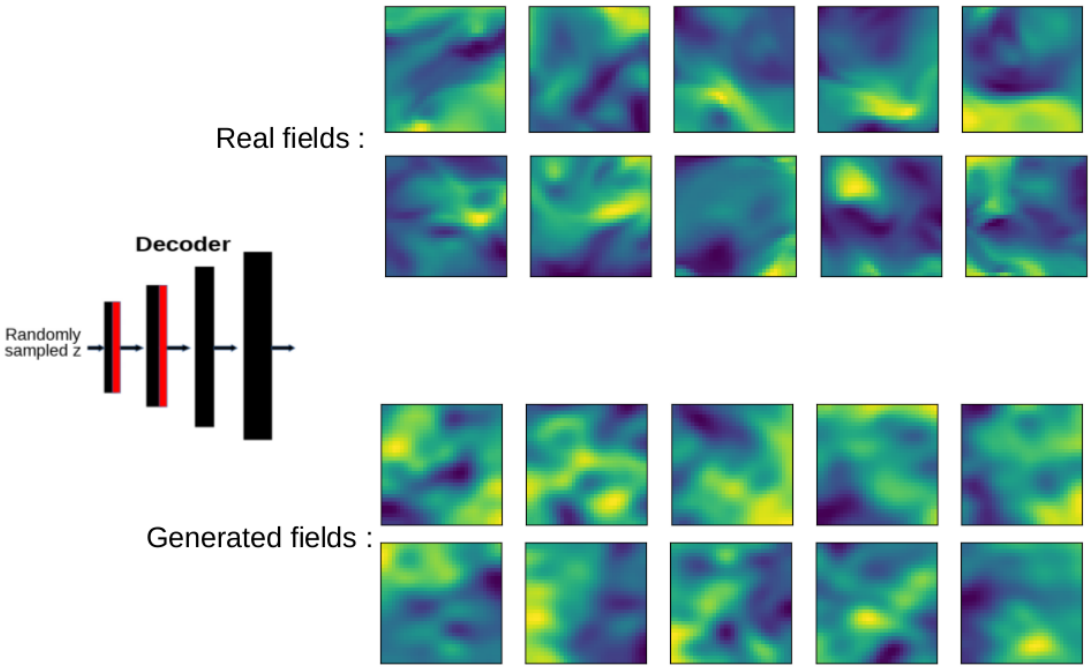


FIGURE 4 – Exemples de génération du VAE.

Les images générées présentent des caractéristiques semblables à celles des images réelles sur lequel le réseau a pu s'entraîner, mais il reste assez facile de détecter à l'oeil nu si une image a été générée ou non par le réseau, notamment à cause du fait que notre auto-encodeur variationnel produit des images assez flou, et "lisse" la plupart des détails assez fins des images en entrée.
Cette spécifité des auto-encodeurs variationnels, qui ont beaucoup de mal à générer des images avec des détails très précis est assez connue, et les GANs [8] sont généralement la solution pour effectuer ce type de génération lorsque les auto-encodeurs variationnels n'y parviennent pas.
Cependant, des choix ont du être faits dans ce stage sur les modèles à implémenter en priorité pour pouvoir être faits dans les temps, et nous avons donc choisi de passer directement à des modèles de segmentation au lieu d'explorer ce qui pouvait se faire en génération à l'aide de GANs.

4.4 La segmentation

Que ce soit pour le traitement d'images satellites ou d'images médicales, beaucoup de recherches ont déjà été effectuées sur la segmentation d'images 2D et de nombreux réseaux ont été mis en place puis améliorés par différentes équipes travaillant sur ces sujets. Ainsi, s'approprier des réseaux ayant déjà fait leur preuves sur des problématiques similaires pour ensuite les modifier et les adapter à nos cas d'applications nous paraît être une stratégie très efficace, et c'est ce que nous allons faire en partant d'un **U-net** en 2 dimensions.

4.4.1 Un U-net

Un **U-net** [18] est un réseau particulier qui est devenu populaire assez récemment pour sa grande efficacité à appliquer un masque sur des données d'entrée. La plupart des modèles les plus performants des compétitions **Kaggle** sur de la segmentation d'images satellites ou d'images médicales par exemple étaient des U-nets. Un U-net a une architecture assez semblable à celle d'un auto-encodeur très profond sans espace latent. Ses principales particularités sont ses **skip-connections** que l'on rajoute par rapport à un modèle classique et qui permettent d'avoir un réseau bien plus profond tout en conservant des détails très fins et les dimensions de l'image d'entrée. Avant d'essayer d'utiliser un réseau de ce type sur notre problématique, à savoir l'estimation du plissement de flamme totale d'une flamme non résolue, nous allons nous concentrer sur une application plus directe et plus proche de l'utilisation initiale du U-net dont nous nous inspirons : la détection du front de flamme d'un feu de forêt sur des images aériennes en 2D. Ce premier cas nous permettra d'améliorer notre compréhension de ce type de réseau, et sera une base solide que nous pourrons ensuite adapter pour faire de la régression en 3D.

4.5 Cas d'application : détection du front de flamme d'un feu de forêt

4.5.1 Description du problème

L'équipe GLOBC du CERFACS a réalisée certaines expériences pour déterminer la vitesse de propagation d'un feu de forêt suivant certaines conditions telles que la force et la direction du vent.

L'expérience qu'ils ont mise en place devait se réaliser en trois étapes :

— Filmer avec une caméra infrarouge depuis un hélicoptère la propagation du feu sur une parcelle de terrain, où le feu aura été allumé à l'aide de 4 bidons d'essence placés à chaque coin du terrain.

— Annoter sur la vidéo obtenu (500 images de taille 400×400) la position du front de flamme.

— Calculer à l'aide du front de flamme annoté sur chaque image la vitesse de propagation du feu de forêt.

La seconde étape s'est finalement révélée beaucoup plus complexe que prévu. Sur les images obtenues avec la caméra infrarouge, il est assez facile de cartographier l'allure générale des zones brûlés de celles qui ne le sont pas, comme on peut le voir sur la Fig. 5 :

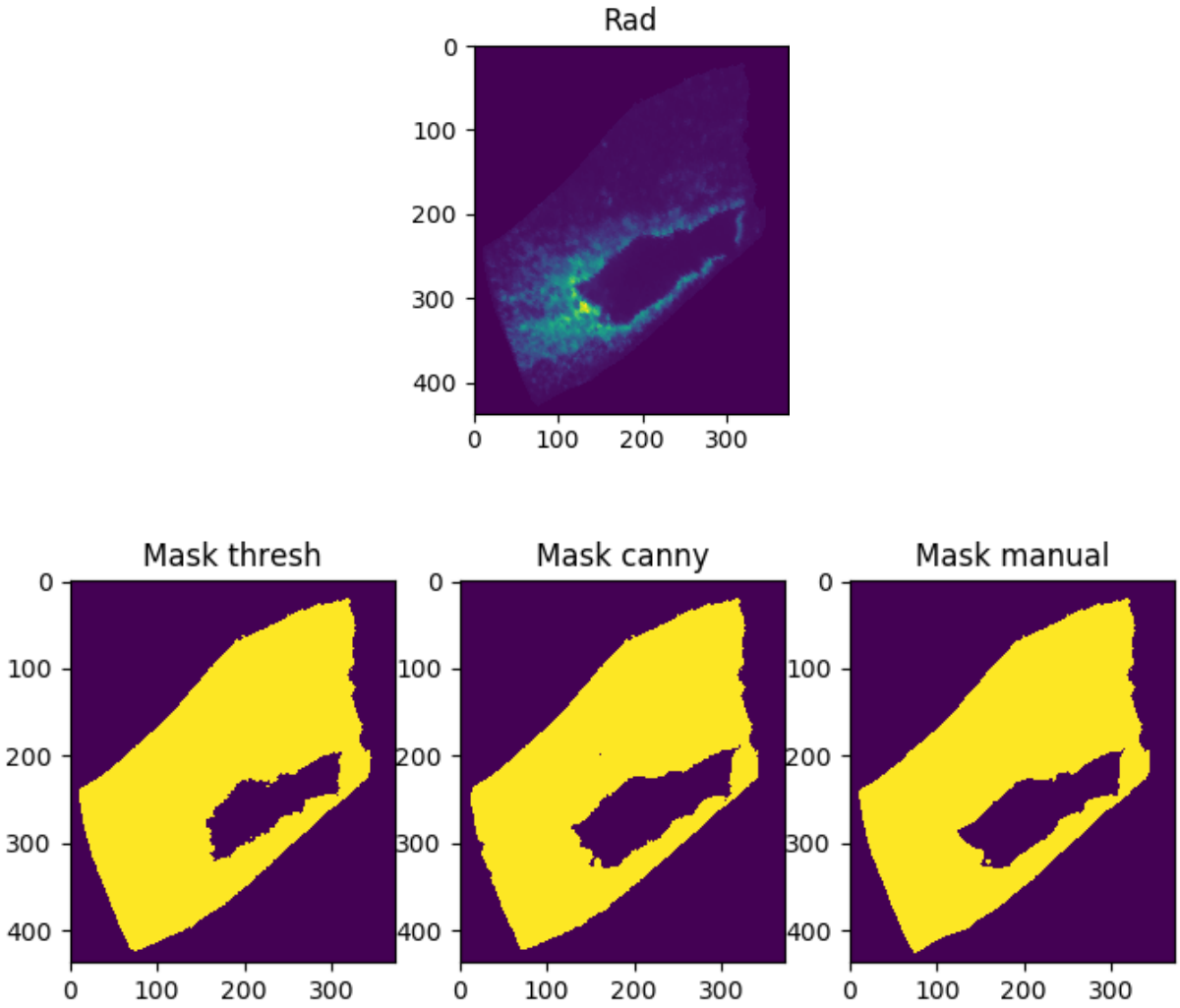


FIGURE 5 – Mask thresh à gauche est un simple seuil, Mask canny au centre est obtenu avec une méthode de croissance de région, et Mask manual à droite est annoté à la main.

Mais certaines zones sont fortement réchauffées par le feu assez proche alors qu'elles n'ont pas encore brûlées et présente ainsi un rayonnement infrarouge très proche des zones ayant déjà brûlé depuis un certain temps et ayant eu le temps de refroidir. Dans ce cas là, il est toujours assez aisé de distinguer l'emplacement du front de la flamme à l'oeil nu grâce au contexte de l'image mais il devient bien plus difficile d'effectuer cette tâche à l'aide d'un modèle physique comme on peut le voir sur la Fig. 6, où les modèles physiques mis en place précédemment sont en difficulté.

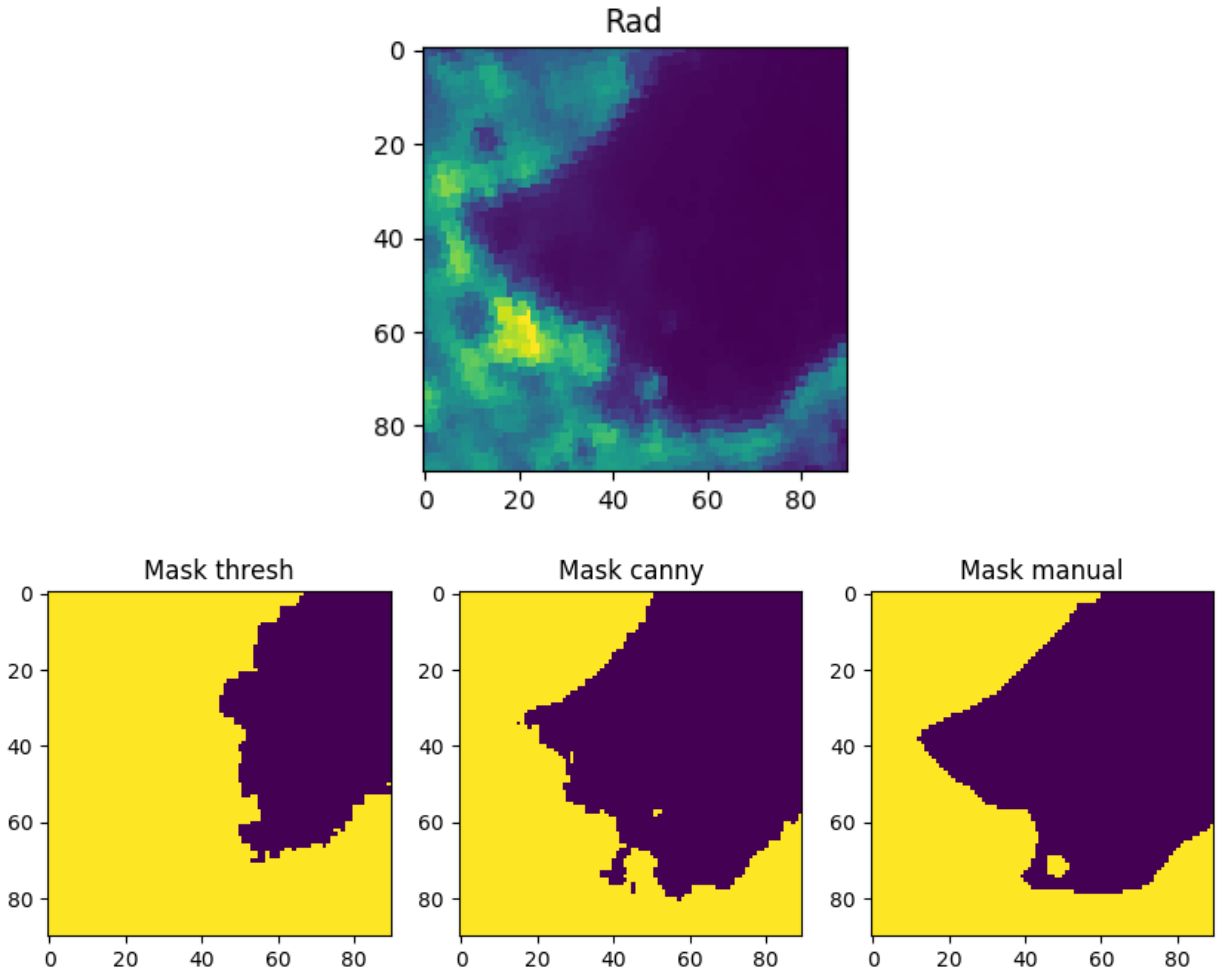


FIGURE 6 – Zoom sur la zone problématique pour les modèles physiques.

Cependant, chaque image nécessitant environ une vingtaine de minutes pour être annotée à la main, il devient impensable d'annoter de cette façon les 500 images de la vidéo, et c'est ici que le Deep Learning entre en jeu. Étant donné qu'il est assez simple d'annoter le front de la flamme pour un oeil averti, on peut espérer qu'un CNN puisse faire de même, pourvu qu'il dispose de suffisamment de données. C'est précisément ce facteur, le manque de données, qui risque de nous poser problèmes.

4.5.2 Construction de la base d'apprentissage

Pour construire une première base de données, des annotations manuelles ont dû être faites sur quelques images. Le modèle canny est utilisée comme support sur lequel on vient directement modifier les zones incorrectes pour que le travail sur les annotations soit plus rapide.
En une journée, 12 champs complets de taille 400×400 pixels ont été annotés de cette

façon, ce qui nous a permis d'obtenir une première base de données pour entraîner notre modèle.
La principale difficulté présente ici, comme nous travaillons avec une très petite base de données, sera **l'overfitting** [5] : lorsqu'un modèle arrête de généraliser et commence à apprendre des particularités trop spécifiques du jeu d'apprentissage. Trois méthodes seront utilisées pour y remédier :

— réduire le nombre de paramètres entraînables du réseau,
— la régularisation (Voir chapitre 4.5.3),
— et enfin, l'**augmentation de données** [17].

L'augmentation de données consiste à effectuer des transformations sur notre jeu d'apprentissage pour que le modèle ne revoit pas exactement la même image à chaque fois et permet ainsi de l'empêcher de mémoriser toute notre base de données.
Parmi ces méthodes, des opérations de rotation et de miroir sont appliquées sur toutes les images de notre jeu d'entraînement.
Un **random cropping** de taille 64×64 est ensuite appliqué sur ces images, permettant d'une part d'augmenter grandement la taille de la base de données, mais également d'extraire des sous images très différentes les unes des autres sur chaque champ tout en ayant suffisamment d'informations dans chaque bloc pour pouvoir détecter les zones brûlées ou non.
Ces opérations peuvent être effectuées sans altérer l'apprentissage car les propriétés des images de notre base de données sont invariantes par translation et par rotation.
Ces données, après avoir subi les opérations vues précédemment, sont traitées par le réseau par groupes de 32 : c'est ce que l'on appelle le **Batch Size**.

4.5.3 Implémentation du U-net

Comme montré sur la Fig. 19, l'allure générale de notre modèle ressemble à celle d'un auto-encodeur complètement convolutionnel avec une partie encodeur et une partie décodeur.
La partie encodeur est composée de plusieurs blocs séparés par des couches de **MaxPooling** permettant de réduire la dimension des images. Chacun de ces blocs contient deux couches de convolution suivies d'une **Batch Normalization** [10], et d'une fonction d'activation **ReLu**. Une **L2 Regularization** est appliquée sur les poids de chacune de ces couches de convolution, ce qui va les empêcher de prendre des valeurs trop élevées via une pénalisation directement sur la fonction de coût totale, et va permettre au réseau de pouvoir mieux généraliser.
La partie décodeur réalise les opérations inverses de la partie encodeur, avec des opérations d'**UpSampling** entre chaque blocs qui permettent d'obtenir les mêmes dimensions en entrée et en sortie du réseau.
Des **skip-connections**, qui ont fait leurs preuves pour permettre de récupérer des informations spatiales qui auraient été perdues dans la phase d'encodage, sont ajoutées pour connecter chaque blocs de même niveau des parties encodeur et décodeur.
Enfin, le nombre de blocs et le nombre de filtres par blocs montré sur la Fig. 19 ont

été réduits par rapport au U-net dont nous nous sommes inspirés afin de lutter contre l'**overfitting**, et notre modèle contient alors $472,257$ paramètres entraînables.

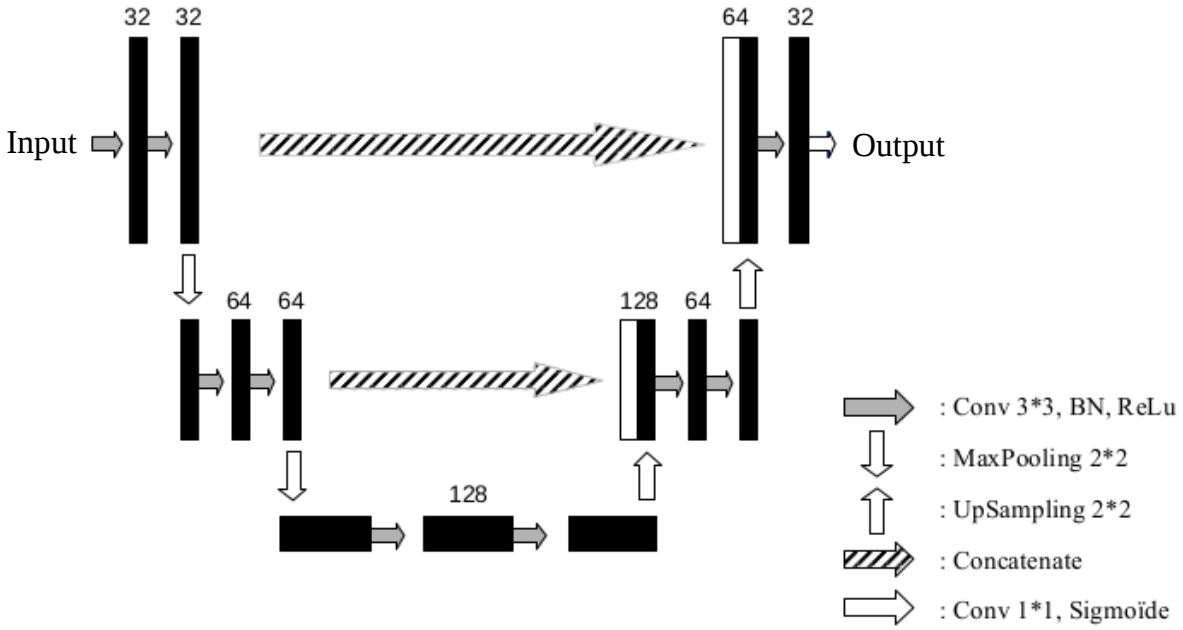


FIGURE 7 – Notre CNN inspiré d'un U-net.

4.5.4 La phase d'apprentissage

Une fois notre modèle implémenté et notre base de données opérationnelle, il faut encore standardiser nos données. En effet, les valeurs de radiance sont très élevées au niveau du front de flamme avec des pics allant jusqu'à 500 environ alors que ces valeurs sont d'une dizaine environ dans les zones froides et notre modèle aurait alors du mal à traiter correctement des entrées avec de si grandes différences de valeurs. Nous standardisons donc toutes nos données à l'aide de la moyenne et de l'écart type obtenu sur notre jeu d'entraînement.
L'apprentissage de notre réseau peut alors commencer. De nombreux paramètres ont dû être adaptés pour obtenir de meilleurs résultats : la valeur de la **L2 Regularization**, le nombre de blocs du réseau, le nombre de filtres par couches...
La plupart ont été adaptés de manière empirique, mais certains problèmes peuvent être identifiés directement à la vue des résultats de nos premières prédictions.
Un des premiers modèles que nous avons essayé d'entraîner avait des difficultés à converger et voici le résultat que l'on obtenait lorsque l'on utilisait notre modèle entraîné pour faire une prédiction sur une image du jeu de test qu'il n'avait jamais vu jusque là, avec un **Learning Rate** initial de 0.001 multiplié par 0.8 toutes les 10 **Epochs** :

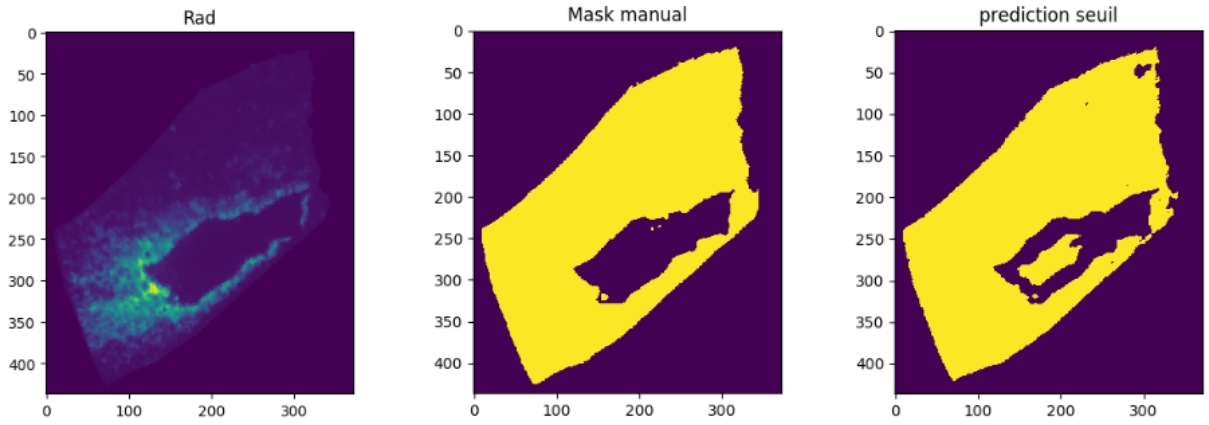


FIGURE 8 – Prédiction du réseau entraîné sur des cubes de 16×16.

Sur la prédiction Fig. 8, le réseau semble avoir du mal à différencier les zones au centre encore intactes de celle en haut à droite qui ont déjà brûlées mais ont eu le temps de refroidir, et qui présentent toute deux des valeurs de radiance très faibles. Sur ce premier modèle, nous réalisions un random cropping avec des tailles de champs de 16×16, or ces champs étaient de dimensions trop faibles pour que des informations telles que la présence du front de flamme ou un gradient de température plus élevée dans une certaine direction puisse permettre au réseau de comprendre dans laquelle des deux situations il se trouvait. Augmenter la taille de ces random cropping, en choisissant maintenant des blocs de taille 64×64 a permit de réduire ce problème, mais il restait encore présent et d'autres solutions ont du être mises en place pour le corriger complètement.

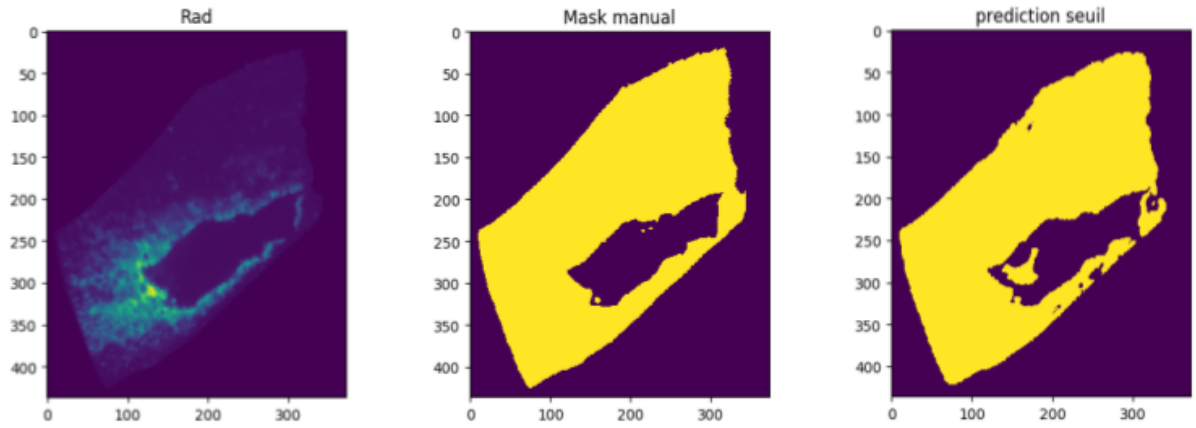


FIGURE 9 – Prédiction du réseau entraîné avec des cubes de 64×64.

Une première piste explorée a été d'essayer d'utiliser des informations temporelles pour permettre au réseau de comprendre dans quel sens se déplace le front de flamme.

Pour fournir ces informations au réseau, la technique choisie dans le cas présent consiste à ajouter d'autres **channels** comportant la radiance de la parcelle pour quelques instants avant et après l'instant t de l'image que nous voulons annoter, cf Fig. 10.

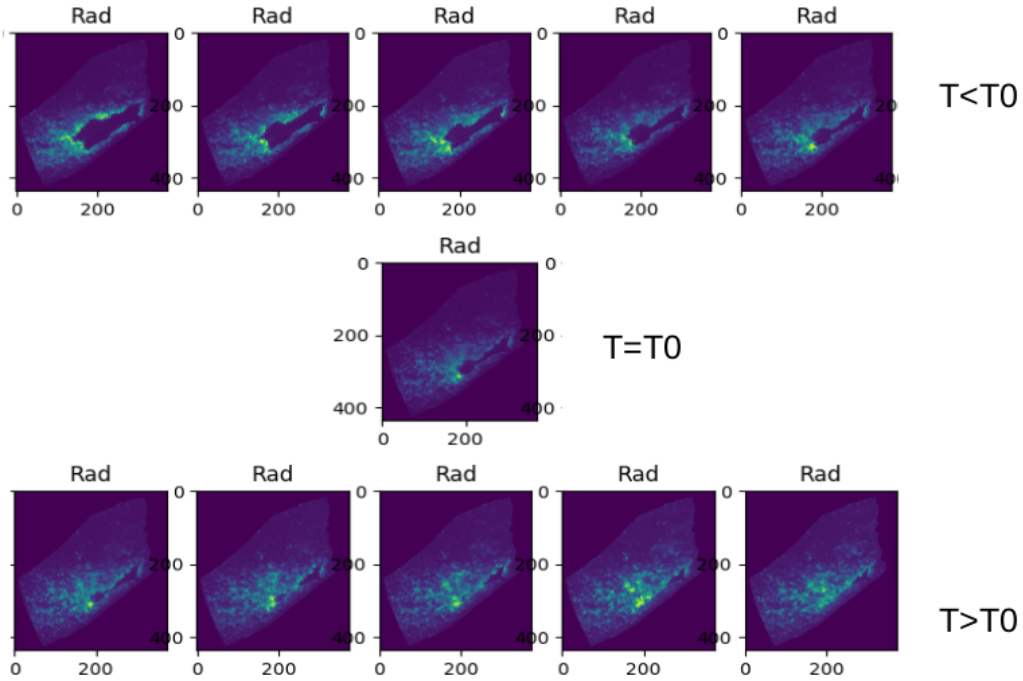


FIGURE 10 – Ajout de channels.

Cependant, cette piste ne s'est pas montrée très fructueuse. Les images que nous possédons sont extraite d'une vidéo prise depuis un hélicoptère avant d'être géoréférencée, et il y a donc un décalage de quelques pixels sur certaines zones dû à des erreurs de parallaxe à cause du fait que la position de l'hélicoptère et donc de la caméra infrarouge ne soit pas parfaitement statique, ce qui a rendu l'utilisation de cette méthode beaucoup moins efficace. Un seule channel sera alors utilisé par la suite.
Enfin, pour continuer à améliorer les performances de notre réseau, avoir une base de données plus grande s'est avéré indispensable mais nous ne pouvions nous permettre de passer une vingtaine de minutes pour annoter chaque image en partant de zéro.
De l'**Active Learning** a alors été mis en place : nous avons utilisé notre réseau pour faire une prédiction sur 25 nouvelles images que nous avons ensuite corrigées à la main avant de les introduire dans notre base de données initiale.
Ce procédé a permis de tripler notre base de données en environ une heure seulement, alors que réaliser cette annotation à partir de zéro aurait requis une dizaine de fois plus de temps.
Nous avons ensuite pu relancer l'apprentissage avec les corrections vues précédemment et une base de données trois fois plus grande. La courbe d'apprentissage a alors eu cette allure :

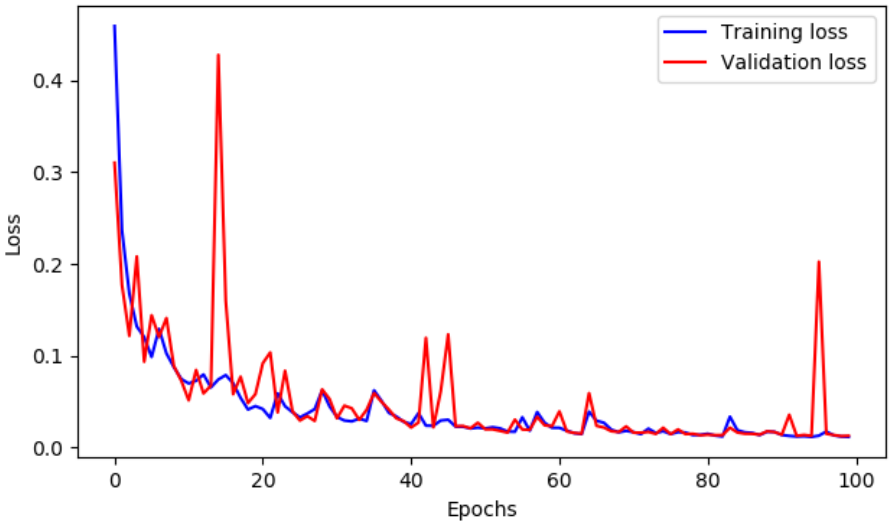


FIGURE 11 – Courbe d'apprentissage sur le jeu d'entraînement et de validation.

Le courbe en bleu montre la **loss** sur le jeu d'entraînement, et celle en rouge qui nous intéresse le plus représente la loss sur le jeu de validation, que le réseau n'a jamais vu et pour lequel il n'a donc pas pu adapter ses poids. L'écart entre ces deux courbes est très important car il permet de voir si le réseau ne commence pas à overfitter et continue bien à généraliser.
Dans le cas présent, la loss sur le jeu de validation est toujours assez proche de celle du jeu d'entraînement, et nous avons donc réussi à éviter l'overfitting malgré notre jeu de données qui reste très faible même après les traitements effectuées précédemment. Après avoir entraîné notre réseau pendant 100 Epochs, voici le résultat que l'on obtient :

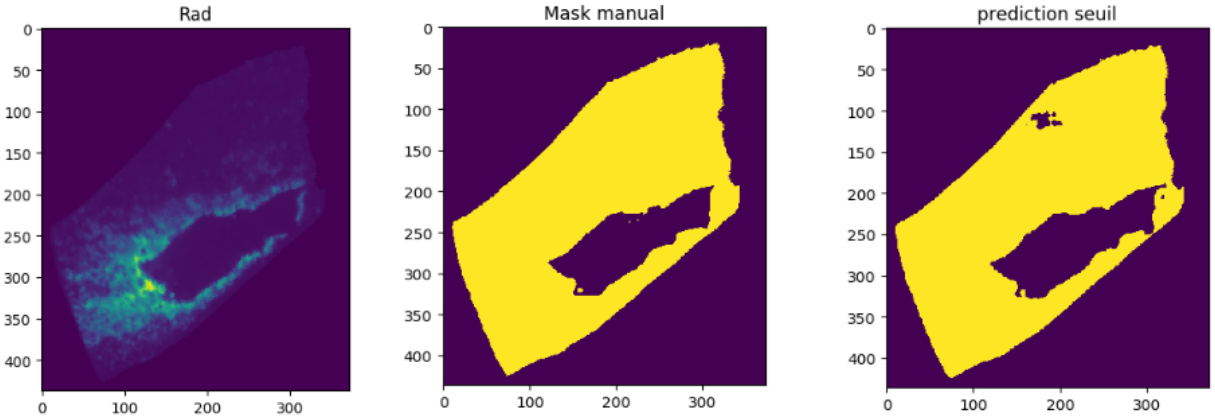


FIGURE 12 – Prédiction finale.

Les prédictions du réseau sont très satisfaisantes, et ont permis d'annoter correctement toutes les images de notre base de données.

5 Modèle de combustion en Machine Learning

5.1 Introduction

Connaître précisément le taux de réaction aux échelles non résolues pour un écoulement turbulent est un problème de notoriété publique qui pose problème aux chercheurs dans le domaine de la combustion depuis longtemps. L'évaluation de la surface de la flamme due au plissement de la flamme aux échelles non résolues est au cœur de tous les modèles basés sur la surface de la flamme au cours des 50 dernières années pour les modèles LES [1, 12, 16]. L'ouverture d'une nouvelle voie pour évaluer cette quantité serait une percée pour les modèles de combustion turbulente.

Les simulations que je vais présenter dans cette partie permettant d'obtenir la DNS n'ont pas été réalisées par moi même mais par Corentin Lapeyre, mon maître de stage au CERFACS. De plus, il m'aura grandement aidé particulièrement sur l'aspect théorique, pour mettre en place le filtrage permettant d'obtenir la LES à partir de cette DNS.

5.1.1 Les modèles de combustion de la littérature

L'estimation de la surface de flamme complètement résolue consiste à mettre en place un modèle qui prendrait certains paramètres tels que le champ passif filtré $\overline{c}$ en entrée et permettrait d'obtenir le champ $\overline{\Sigma}$.

Un des premiers modèles de combustion à être mis en place est le modèle de Gouldin en 1989 [9]. L'approche fractale faite par ce modèle suggère que le relation entre ces deux paramètres est de la forme :

$$\overline{\Sigma} = \left(\frac{\Delta}{\eta_c}\right)^{D_f-2} |\nabla \overline{c}| \tag{2}$$

où D_f est la dimension fractale de la surface de flamme, et η_c est l'échelle de la fréquence de coupure à partir de laquelle la flamme n'est plus plissée.

Des études plus récentes basées sur les interactions flamme / vortex et des analyses multi-fractales comme le modèle de Charlette en 2002 [2] suggèrent une forme différente :

$$\overline{\Sigma} = \left(1 + \min\left[\frac{\Delta}{\delta_L^0}, \Gamma_\Delta\left(\frac{\Delta}{\delta_L^0} - 1, \frac{u'_\Delta}{S_L^0}, Re_\Delta\right)\frac{u'_\Delta}{S_L^0}\right]\right)^\beta |\nabla \overline{c}| \tag{3}$$

5.1.2 L'approche Machine Learning

D'un point de vue Machine Learning, ces modèles correspondent tous deux à une régression linéaire permettant de prédire $\overline{\Sigma}$ avec différentes variables en entrée : $(\overline{c}, \eta_c, \Delta/\delta_L^0, u'_\Delta/S_L^0)$...

Mais ces deux équations (2) and (3) sont complètement locales, ces fonctions sont de la forme :

$$\overline{\Sigma} = f(\overline{c}, \mathbf{u}, ...) \tag{4}$$
$$f : \mathbb{R}^k \mapsto \mathbb{R} \tag{5}$$

où k est le nombre de variables choisies en entrée du modèle.

Une approche Machine Learning permettrait quant à elle d'extraire des informations topologiques de la turbulence, les fonctions seraient alors de la forme :

$$\overline{\Sigma}(X) = f(\overline{c}(X)),\ X \in \mathbb{R}^{n^3} \tag{6}$$
$$f : \mathbb{R}^{n^3} \mapsto \mathbb{R}^{n^3}$$

où n serait maintenant plus grand que 1, et où $X \in \mathbb{R}^{n^3}$ serait un cube de taille $n \times n \times n$

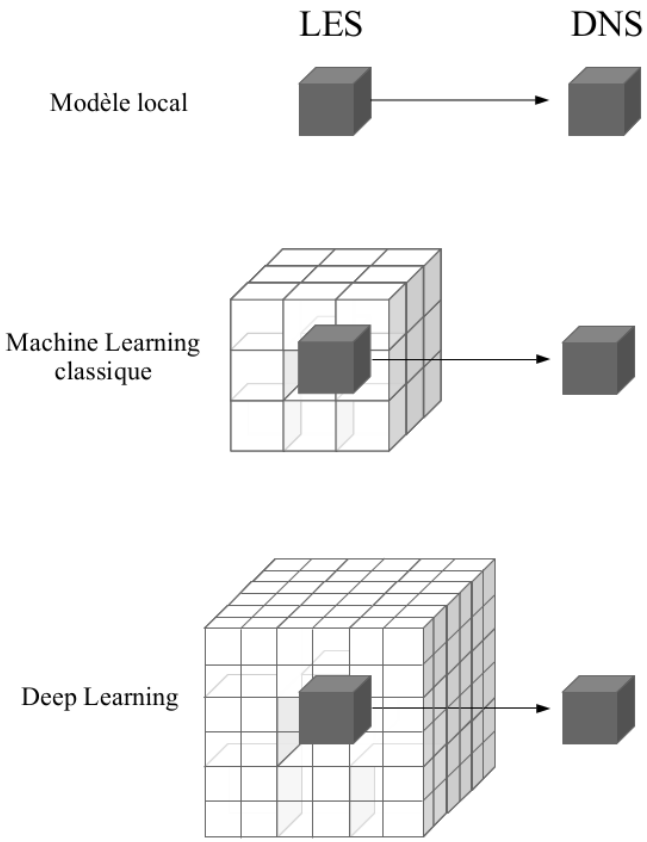


FIGURE 13 – Dimension des entrées et sorties des modèles pour chaque approches.

Différents modèles de Machine Learning Classique seront mis en place avec des cubes de petites taille ($n = 3$), mais nous allons également explorer le potentiel du Deep Learning avec des modèle neuronaux prenant en entrée des cubes de tailles plus élevées $n = 8 - 32$ permettant d'extraire bien plus d'informations topologiques. Ces différentes approches seront ensuite comparées sur leur précision et leur rapidité

5.2 Création de la base d'apprentissage

La stratégie choisie pour entraîner notre réseau est la suivante : une fois notre DNS obtenu sur le maillage initial, le champ passif de la DNS c sera filtré pour obtenir $\overline{c}$ qui sera l'entrée du réseau, ce que l'on aurait obtenu dans le cas d'une LES, et $\overline{\Sigma}^+$ sur le même maillage plus grossier (d'un facteur 8 dans le cas de la (Fig. 14)) qui sera la valeur cible de notre réseau

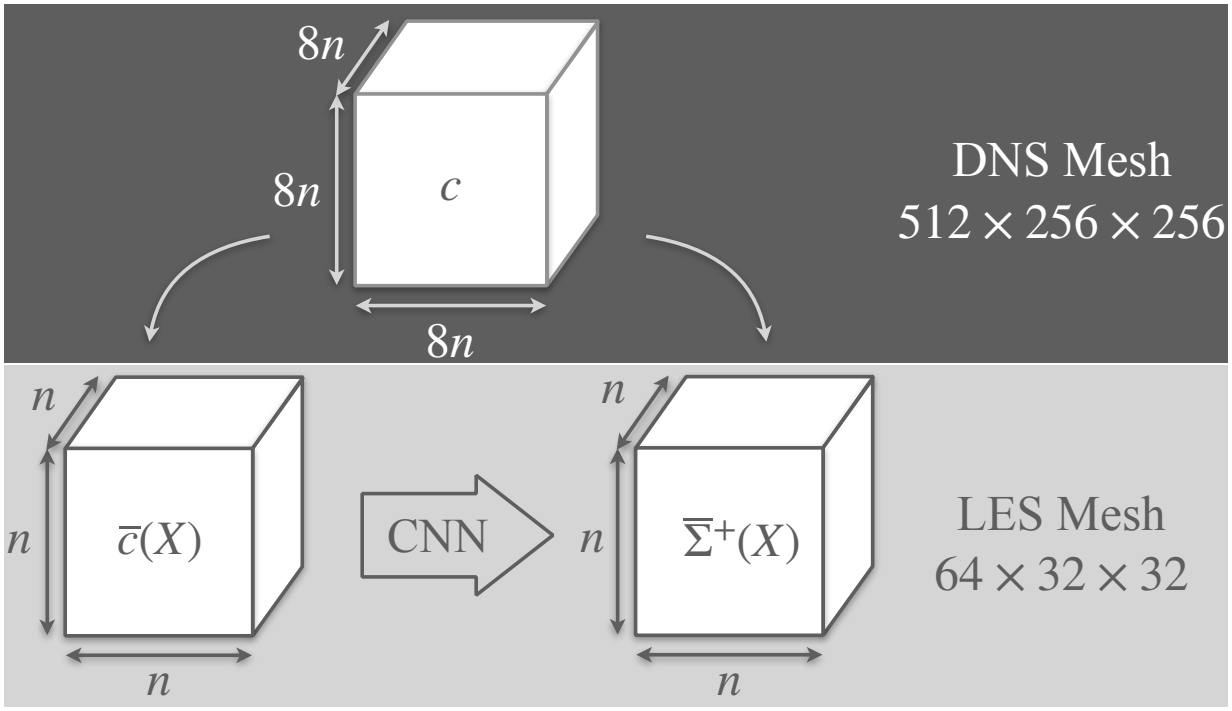
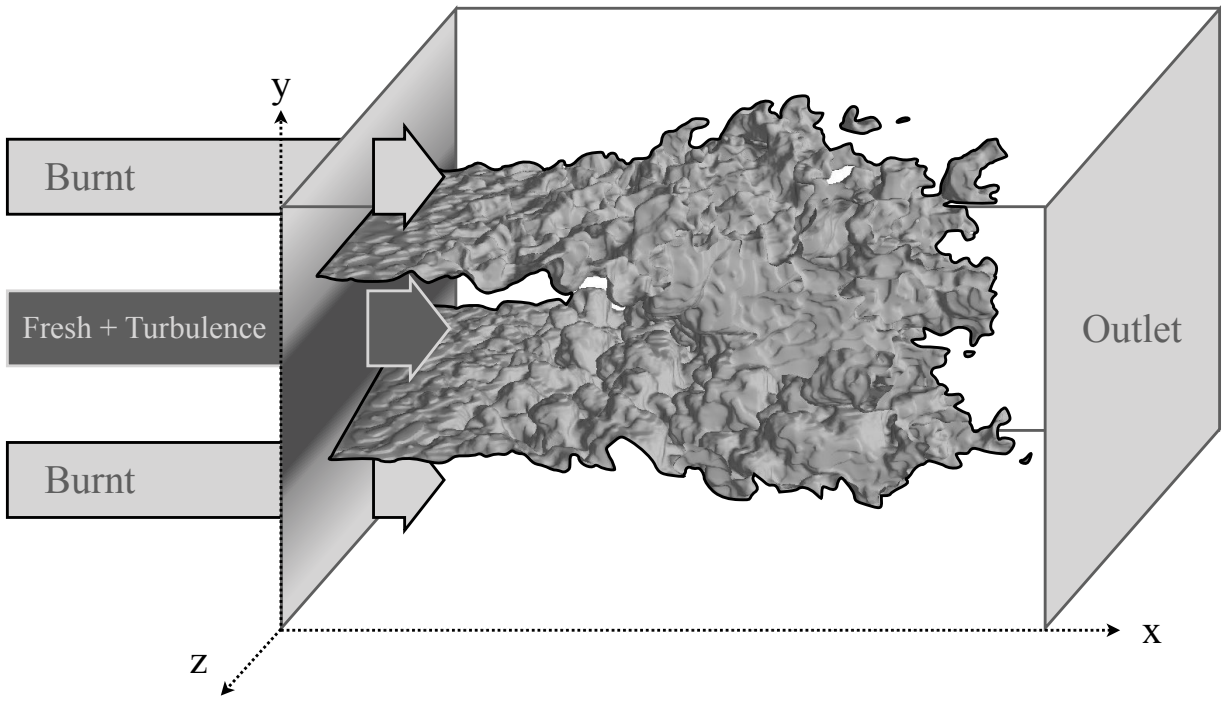


FIGURE 14 – Le champ DNS c est filtré pour obtenir $\overline{c}$ et $\overline{\Sigma}^+$ sur le maillage LES

5.2.1 Réalisation de la DNS

Deux DNS d'un brûleur méthane-air présentées en (Fig. 15) ont été lancées pour construire notre base de données.
L'écoulement central est composé d'un mélange stoechiométrique de méthane et d'air. Des gaz brûlés sont injectés de part et d'autre, avec une température identique à celle des gaz frais après leur combustion totale. Une turbulence est injectée dans les gaz frais seulement,



]

FIGURE 15 – Isosurface à $T = 1600$ K de la DNS 1. Les conditions aux limites sont périodiques suivant y et z.

avec un taux de turbulence de 5% et 10% respectivement pour les DNS 1 et 2. Ces deux taux de turbulences assez différents permettent d'introduire un peu plus de variété dans la base d'apprentissage.
Sur chacune de ces simulations, des captures instantanées du champ sont sauvegardées toutes les 0.2ms. Ce temps correspond au temps minimal nécessaire pour que deux instantanés soient suffisamment décorrélés entre eux. Ces deux simulations ont tourné pendant 14 ms, les 4 premières ms étant inexploitables le temps que la flamme se mette en place. Nous avons alors pu extraire 50 instantanés de chaque DNS, et donc constitué une base de données de 100 champs 3D de taille $512 \times 256 \times 256$ $180,000$ heures CPU ont été utilisées sur un cluster équipé de 20 nœuds de calcul Broadwell de 2.6 Ghz pour effectuer ces calculs, et le code AVBP explicite et compressible a été utilisé pour résoudre les équations de Navier-Stokes en 3D avec une chimie simplifiée sur des maillages non structurées.[19, 20]

5.2.2 Filtrage de la DNS

Pour obtenir la LES, un filtrage spatial est utilisé afin d'enlever les échelles non résolues du spectre de turbulence de la DNS. Pour chaque quantité d'intérêt Q d'un champ d'écoulement bien résolu, appliquer un filtre spatial passe-bas F_Δ avec largeur Δ donne :

$$\overline{Q(\mathbf{x},t)} = \int_{\mathcal{V}} F_\Delta(\mathbf{x}-\mathbf{x}')Q(\mathbf{x}',t)\mathbf{x}' \tag{7}$$

où $\overline{Q(\mathbf{x},t)}$ est l'opération de filtrage appliquée à $Q(\mathbf{x},t)$. Nous limiterons notre étude à des cas de prémélange parfait, où la variable de progrès c de l'écoulement est défini comme :

$$c = \frac{T - T_{min}}{T_{max} - T_{min}} \tag{8}$$

c varie donc entre 0 dans la zone des gaz frais, et 1 dans la zone des gaz brûlés.

En filtrant 7, et en utilisant une G-équation de propagation [11], on obtient [12] :

$$\frac{\partial \overline{\rho}\tilde{c}}{\partial t} + \nabla\cdot(\overline{\rho}\tilde{\mathbf{u}}\tilde{c}) + \nabla\cdot(\widetilde{\overline{\rho}\mathbf{u}c} - \overline{\rho}\tilde{\mathbf{u}}\tilde{c}) = \rho_u S_L|\overline{\nabla c}| \tag{9}$$

Où le terme de droite contient les termes de diffusion et de réaction filtrés en une seule isosurface c de vitesse de déplacement, assimilée à la vitesse de flamme laminaire S_L, et où ρ_u est la densité des gaz frais. $|\overline{\nabla c}|$ est la densité de surface de flamme, aussi noté $\overline{\Sigma}$
Cette surface de flamme ne peut pas être obtenu parfaitement, en effet lorsque l'on filtre c, la surface de flamme décroît, ce qui amène une isosurface c plus faible.
Une méthode populaire pour modéliser $\overline{\Sigma}$ consiste à introduire un *facteur de plissement* Ξ qui est le rapport entre les flammes complètement et partiellement résolues. Le terme de droite de l'équation. 9 peut alors s'écrire comme :

$$\rho_u S_L|\overline{\nabla c}| = \rho_u S_L \Xi|\nabla\overline{c}| \tag{10}$$

$$\text{où} \quad \Xi = \frac{\overline{\Sigma}}{|\nabla\overline{c}|} = \frac{|\overline{\nabla c}|}{|\nabla\overline{c}|} \tag{11}$$

où $|\overline{\nabla c}|$ est la densité de flamme DNS, et $|\nabla\overline{c}|$ la densité de flamme LES.

Pour effectuer le filtrage de la DNS, un filtre gaussien est utilisé et sa largeur *width* est défini comme le rapport du max des gradients DNS et LES $|\nabla c|$, $d$ans la zone laminaire i.e. :

$$\Delta = \frac{\max|\nabla c|}{\max|\nabla\overline{c}|}x \tag{12}$$

La fonction de filtrage en résultant est alors de la forme :

$$F_\Delta(n) = \begin{cases} e^{-\frac{1}{2}(\frac{n}{\sigma})^2} & \text{if } n \in [1,N] \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

Et est ensuite normalisée par la somme $\sum_{n\in[0,N]} F_\Delta(n)$. Ici, $\sigma = 26$ and $N = 31$ sont les paramètres de ce filtrage, qui ont été optimisés pour obtenir un $\Delta \approx 8dx$.

5.2.3 Normalisation de la base de données

Une fois le filtrage réalisé, $\overline{c}$ sera l'entrée et $\overline{\Sigma}^+$ la sortie de nos modèles. Il nous reste cependant à normaliser ces valeurs. En effet lorsque l'on fait du Machine Learning, normaliser les données permet de donner un impact similaire à chacune des variables.
$\overline{c}$ est déjà normalisé par construction du fait que ce champ prenne des valeurs allant de 0 dans les gaz frais à 1 dans les gaz brûlés, mais $\overline{\Sigma}^+$ prend des valeurs très élevées dans la zone de flamme ce qui poserait problème pour les différents modèles que l'on veut mettre en place si on laissait ses valeurs telles quelles.
Cependant, la normalisation que l'on souhaite mettre en place doit pouvoir être reproductible, même lorsque l'on disposera uniquement de la LES, quand on voudra appliquer notre modèle une fois entraîné. La méthode que nous avons choisie qui répond à toutes ces contraintes est de diviser $\overline{\Sigma}$ par la valeur maximale de $\overline{\Sigma}$ lorsque la flamme est laminaire. La valeur cible de notre réseau est alors :

$$\overline{\Sigma}^+ = \frac{\overline{\Sigma}}{\overline{\Sigma}_{lam}^{max}} \tag{14}$$

et varie entre 1 dans les zones où la flamme est très peu plissée aux petites échelles jusqu'à environ 3 sur quelques zones très plissées.

5.3 Estimation en Machine Learning

Les modèles physiques mis en place dans la littérature utilisent différentes variables en chaque point ($\overline{c}$, mais aussi la densité, la vorticité...). Nous utiliserons dans nos modèles uniquement le champ passif $\overline{c}$.

5.3.1 Limites des modèles algébriques

Un modèle algébrique tel que le modèle de Gouldin par exemple (2) se ramène à une régression linéaire, et il est donc intéressant de voir si ce genre de modèle à une chance de pouvoir réussir à faire correspondre correctement la densité d'une flamme LES à celle de la flamme DNS correspondante.

Si pour un champ 3D donné, on place sur un graphe pour chaque mailles de ce champ la valeur de densité de flamme LES en abscisse, et la densité de flamme DNS en ordonné, voilà ce que l'on obtient :

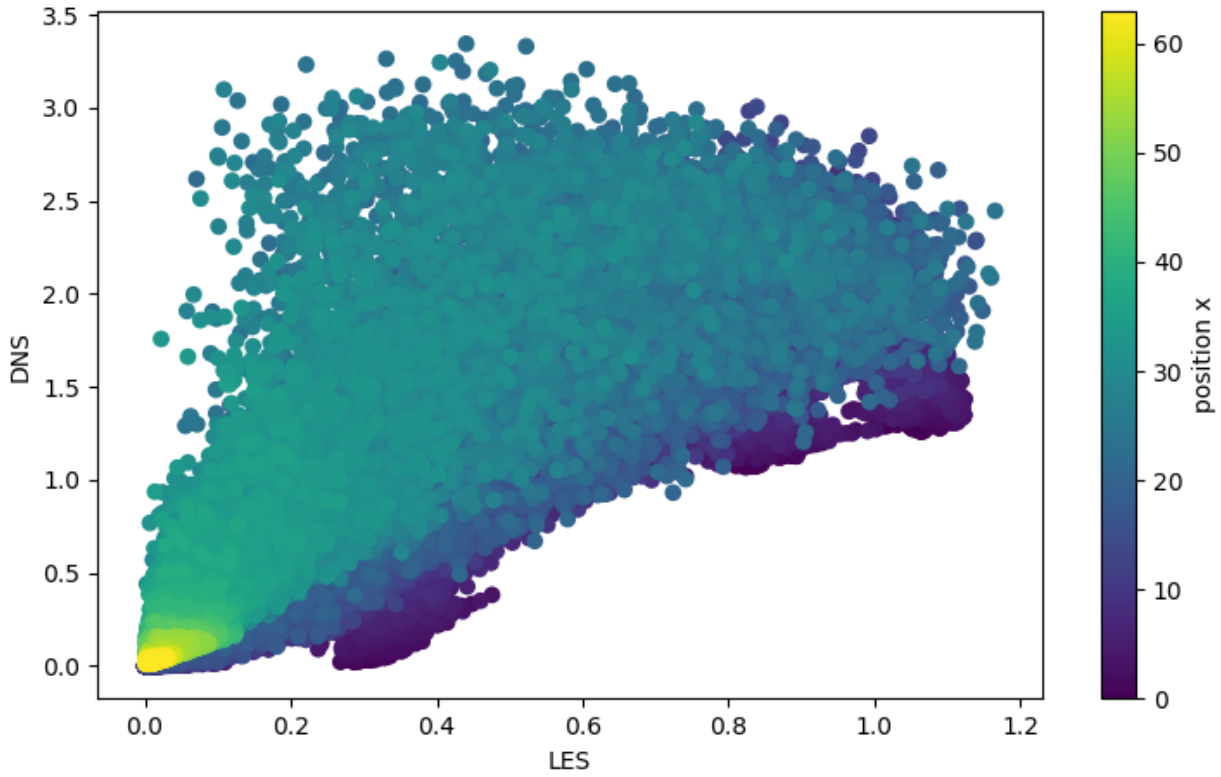


FIGURE 16 – Scatter-plot de l'épaississement nécessaire au passage LES->DNS pixel par pixel coloré par la position axiale x.

Sur le graphe 16, on voit que le nuage de points est bien trop étendu pour qu'une simple régression linéaire puisse être un bon modèle pour faire correspondre la densité de flamme LES à celle DNS. Ainsi, même si des modèles comme le modèle de Gouldin utilisent en

pratique d'autres variables en plus de $\overline{c}$ telles que la densité, leur permettant d'extraire plus d'informations et ainsi de faire de meilleures prédictions, nous venons de voir qu'une régression linéaire ne pouvait pas nous permettre d'espérer obtenir de bons résultats en effectuant uniquement une analyse locale point à point.

C'est là que les modèles de Deep Learning avec des successions de nombreuses couches de convolution entrent en jeu, car on sait qu'ils peuvent se montrer très efficaces pour ce genre de tâche, où les informations topologiques sont très importantes.
Mais avant de passer directement à de gros modèles de Deep Learning qui peuvent être très coûteux et longs à entraîner, il convient de voir si l'on ne peut pas obtenir des résultats semblables avec des méthode de Machine Learning classiques.

5.3.2 L'approche Machine Learning Classique

Comme montré en (Fig. 13), une approche Machine Learning pourrait, à la différence des approches algébriques utilisées jusque-là dans la plupart des modèles physiques, prendre en entrée du modèle bien plus de paramètres spatiaux et donc obtenir beaucoup plus d'informations topologiques sur la flamme, telles que sa courbure ou son orientation.

Pour entraîner différents modèles de Machine Learning classique, il faut déjà que l'on se constitue une base d'apprentissage à partir des 100 champs DNS et LES dont nous disposons.
Pour chaque maille de ces 100 champs dont nous disposons, nous constituons un couple entrée/sortie composé pour la sortie de la valeur de la densité de flamme DNS sur cette maille et en entrée des valeurs de cette maille de tous ces voisins sur le maillage 3D (soit 27 variables). On peut ainsi extraire environs 2000 "couples" par champs 3D, et en utilisant une répartition $80\% - 20\%$ pour nos jeux d'entraînement et de validation, on obtient alors environ $160,000$ données d'apprentissage et 40000 pour la validation.

On peut alors entraîner différents modèles de Machine Learning classiques sur ce jeu de données. Voici les différentes méthodes que nous avons mises en place, et les résultats que l'on a obtenu avec une MSE comme fonction de coût :

	MSE	Durée en secondes	Nombre de paramètres
Gouldin [1]	0.0951	0.010	1
Decision Tree	0.00904	0.0121	27
SVR	0.00796	8.0347	27
Nearest Neighbor	0.00690	6.821	27
Random Forest	0.00422	0.109	27

5.4 Estimation en Deep Learning

Les résultats vus précédemment sont très encourageants sur l'apport d'informations topologiques pour améliorer la qualité de notre prédiction ; le fait de connaître la valeur de la densité de flamme LES sur tous les voisins directs d'un point a permis de grandement réduire l'erreur de prédiction pour ce point, et on peut donc espérer pouvoir faire encore mieux en fournissant les valeurs sur encore plus de points alentours à notre réseau.

Cependant, si l'on souhaite encore augmenter la taille de nos entrées, des réseaux de neurones convolutifs seront plus adaptés que des méthodes de machine learning classique, de part leur capacité à traiter des quantités énormes de données et leur grande performance sur des problèmes de vision par ordinateurs.

5.4.1 L'architecture de notre réseau

Nous avons donc choisi de réutiliser un réseau de type U-net comme celui que nous avions mis en place au chapitre 4.5.3), mais quelques modifications ont dû être effectuées pour adapter ce modèle à notre cas :

- Les entrées de notre modèle sont maintenant en 3 dimensions, les couches de convolution, de MaxPooling et d'UpSampling, ont donc du être modifiées en conséquence.
- Le modèle dont nous nous sommes inspirés permettait de faire de la segmentation alors que nous voulons faire de la régression. La couche d'activation finale qui était une sigmoïde a donc été remplacée par une activation ReLu, permettant d'avoir une sortie continue tout en empêchant le réseau de prédire des valeurs négatives.

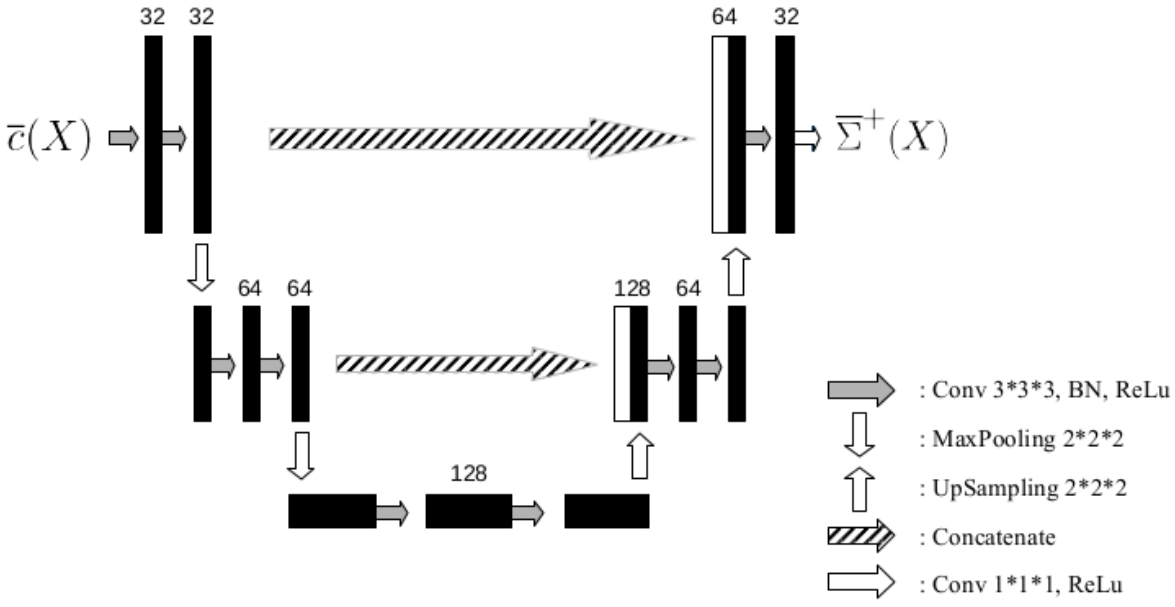


FIGURE 17 – Notre modèle inspiré d'un U-net.

Nous appelons par la suite ce réseau un CNN (pour Convolutional Neural Network).

5.4.2 L'entraînement du réseau

La base de données obtenue comme présenté en 5.2 à partir de deux DNS est splittée en 3 parties :

— le jeu d'entraînement, utilisé pour optimiser les poids du réseau (70% des données),
— e jeu de validation, utilisé pour évaluer le réseau durant l'apprentissage sur un jeu de données que le réseau n'a pas encore vu, ce qui permet de détecter que le réseau est encore en train de généraliser et ne commence pas à overfitter (20% des données),
— le jeu de test, utilisé uniquement à posteriori, pour évaluer les performances du réseau une fois l'entraînement terminé (10% des données).

Une fois cette répartition faite, de l'augmentation de données est appliquée sur le jeu d'entraînement : un **random cropping** de taille $16 \times 16 \times 16$ ainsi que des rotations et des inversions sont appliquées.
Or, la base de données est maintenant beaucoup trop grande pour tenir en mémoire et un générateur est alors nécessaire. Il permet de ne charger qu'un **Batch** en mémoire à la fois et d'appliquer ces opérations d'augmentation de données en temps réel.
100 **Batchs** sont traités par le réseau durant chaque **Epochs**, avec un **Learning Rate** initial de 0.01, réduit de 20% toutes les 10 **Epochs**.

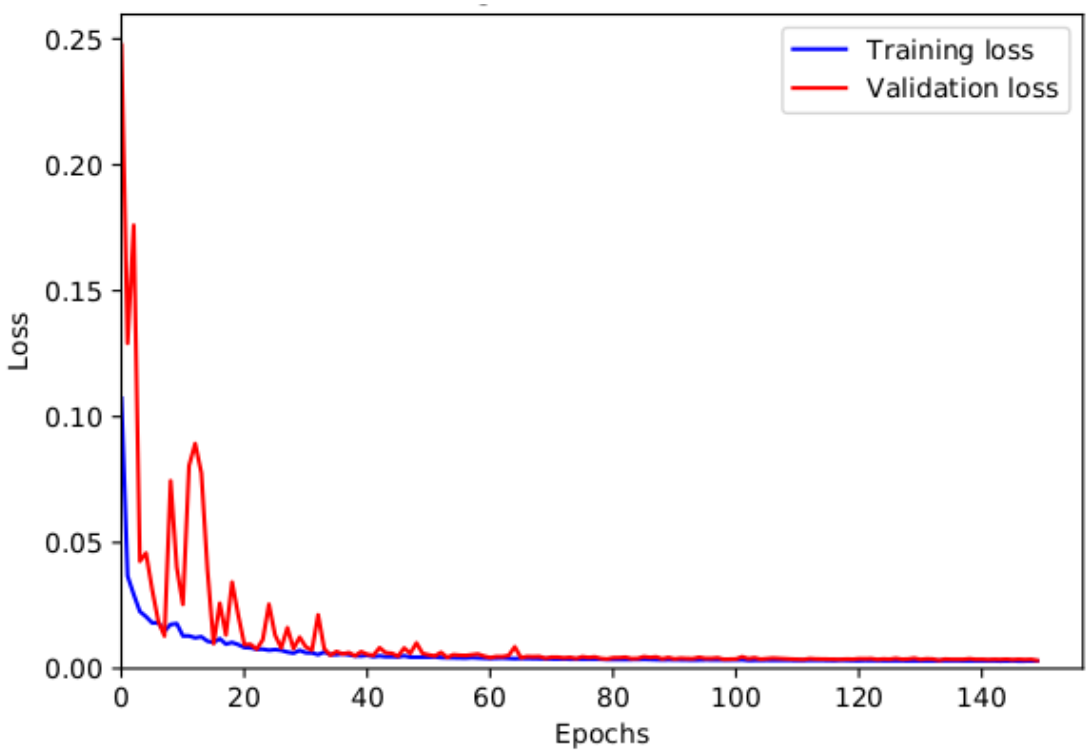


FIGURE 18 – Courbe de **loss** sur le jeu d'entraînement et de validation.

140 **Epochs** seront nécessaires pour que le réseau converge, ce qui prend environ 30 minutes sur un GPU Nvidia Tesla V100.

5.4.3 Résultats

Voici les résultats que l'on obtient lorsque l'on réalise une inférence sur un des champs du jeu de test une fois le modèle entraîné.

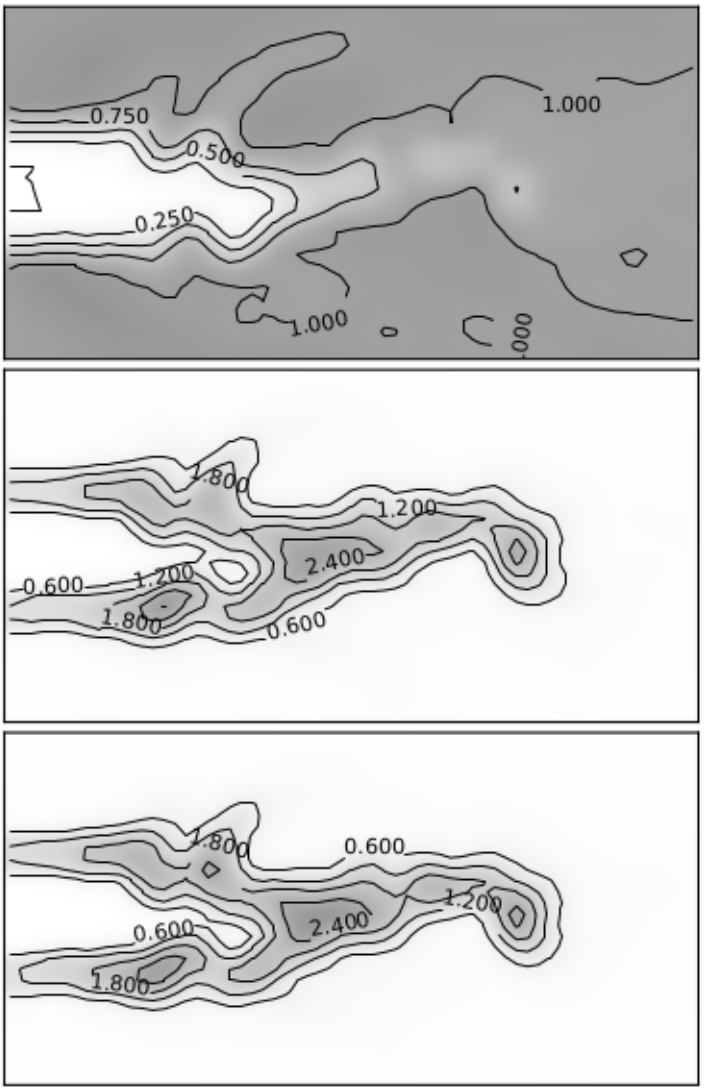


FIGURE 19 – En haut : le champ passif filtré $\overline{c}$ qui est l'entrée du réseau, au centre : la densité de flamme DNS qui est la target, et en bas : notre prédiction.

La prédiction à l'air très bonne, mais il est assez difficile de juger visuellement de la qualité de la reconstruction de façon objective. Une des principales métrique qui vient à l'esprit est alors la MSE, que nous utilisons d'ailleurs comme loss pour entraîner notre modèle, et si l'on compare cette MSE sur les différentes reconstructions, avec un modèle algébrique (Gouldin), avec un modèle de Machine Learning classique et avec notre modèle de Deep Learning, voici les résultats que l'on obtient :

	MSE	Durée en secondes	Nombre de paramètres
Gouldin	0.0951	0.010	1
Decision Tree	0.00904	0.0121	27
SVR	0.00796	8.0347	27
Nearest Neighbor	0.00690	6.821	27
Random Forest	0.00422	0.109	27
CNN on CPU	0.00191	2.213	16^3
CNN on GPU	0.00191	0.012	16^3

D'après la MSE, notre CNN est plus performant que des méthodes de Machine Learning Classique, qui l'étaient elles-mêmes bien plus que des méthodes algébriques telles que le modèle de Gouldin. Le temps nécessaire pour faire une inférence est également très important, notamment si l'on veut coupler notre CNN avec un modèle de combustion.
Pour ne pas devenir le goulot d'étranglement et ralentir tout le modèle, le temps d'exécution d'une inférence doit rester de l'ordre de grandeur du centième de seconde. Dans le cadre du Machine Learning Classique, seule la méthode d'arbres de décisions est exécutée suffisamment rapidement, et c'est donc cette méthode qui sera adoptée si l'on dispose uniquement de CPU.
Un autre gros avantage du CNN par rapport à une approche de Machine Learning Classique est le fait de pouvoir être exécuté sur un GPU. Dans le cas présent, pouvoir exécuter l'inférence de notre CNN sur GPU nous permet d'avoir un speed-up de 200 environ, et ainsi de respecter l'ordre de grandeur souhaité pour la durée d'exécution, tout en ayant une prédiction de bien meilleure qualité.
Ainsi, si l'on dispose d'une architecture hybride avec un GPU, notre CNN sera la méthode la plus performante dans tous les domaines.

Mais si une métrique telle que la MSE permet facilement de comparer différents modèles et nous indique que le CNN est la méthode la plus efficace parmi celles que nous avons mises en place en Machine Learning, elle nous donne cependant assez peu d'informations sur les zones problématiques. Nous avons alors choisi de comparer les surfaces de flammes intégrées par tranches y-z pour chaque prédictions, pour avoir une meilleure vision des choses.
Deux des méthodes algébriques utilisées dans la littérature que nous avons vues précédemment dans la partie 5.1.1 ont été implémentées pour que nous puissions avoir une comparaison pour notre modèle.
Si l'on compare, sur une image de notre jeu de test, les surfaces de flammes par tranches prédites par notre CNN (U-net prediction) et celles obtenues à l'aide de modèles algébriques (Gouldin à différents beta et Charlette), voici ce que l'on obtient :
LES est la surface de flamme sous résolue et DNS la target parfaite.

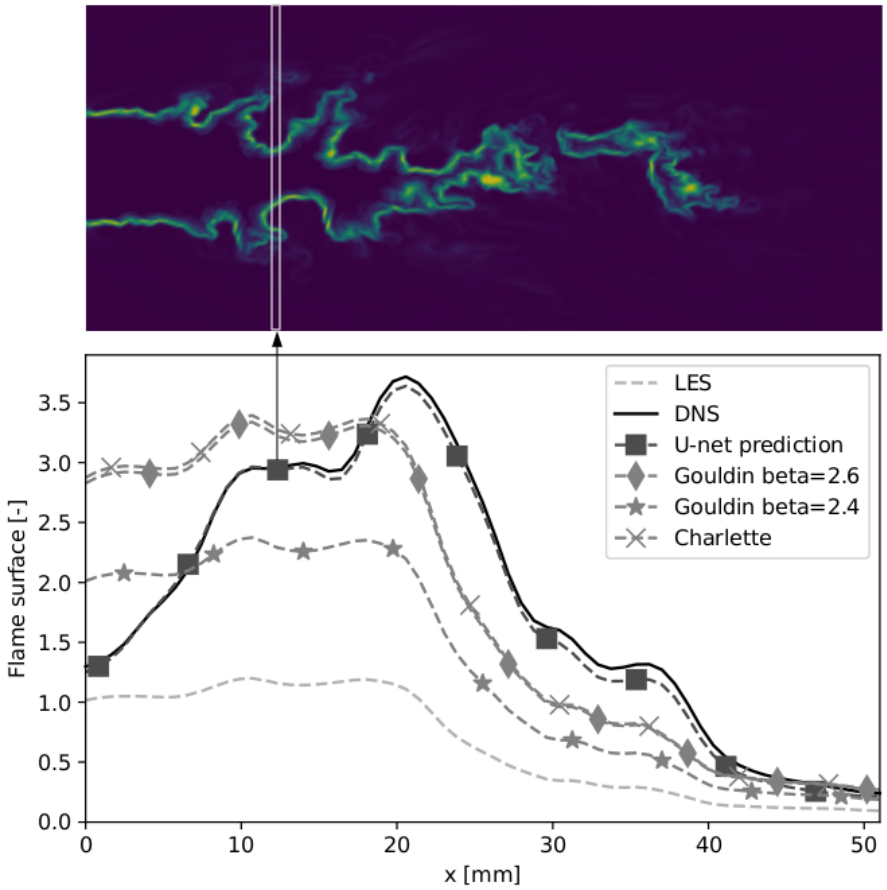


FIGURE 20 – Surface de flamme totale intégrée sur une tranche y-z suivant x.

A l'entrée, où la turbulence est présente mais n'a pas encore plissé le front de flamme, les modèles classiques de turbulence, comme le modèle de Charlette par exemple, surestiment de manière significative la surface de la flamme.
Ceci est dû au fait que le front de flamme n'est pas encore plissé par les structures turbulentes comme on peut le voir sur la partie gauche de la Fig. 20, où la flamme est encore presque laminaire. Dans ce cas, la turbulence et les mouvements de flamme n'ont pas encore atteint d'équilibre, or cet état est nécessaire dans la dérivation des modèles algébriques que nous avons effectuée.
Plus en aval, les modèles classiques de turbulence sous estiment grandement la surface de la flamme. Ainsi, la surface de flamme totale est plutôt bien reconstruite par ces modèles : la surface de flamme totale surestimée à l'entrée compense assez bien celle sous estimée plus loin en aval, mais elle est très mal localisée.

De son coté, notre CNN arrive à reconnaître, grâce à la topologie de l'écoulement, lorsque la flamme est froissée ou non et peut ainsi bien mieux s'adapter à ces différents cas. Il arrive ainsi à prédire une bonne surface de flamme totale, et surpasse grandement les modèles algébriques existants sur sa capacité à comprendre l'épaississement nécessaire localement.

5.4.4 Validation et limites de nos modèles.

De mi-juin à mi-juillet durant la dernière partie de mon stage, s'est déroulé le CTR Summer Program à Stanford, où notre projet concernant la réalisation d'un modèle de combustion assisté par du Deep Learning a été accepté. L'équipe participant a ce projet était composée de Corentin Lapeyre, Antony Misdariis et moi même, sous la supervision de Thierry Poinsot. Corentin Lapeyre était sur place à Stanford pendant que nous le soutenions depuis le CERFACS.
Le Summer Program regroupait de nombreux projets sur des thématiques similaires et aura été une excellente opportunité pour valider nos modèles. Au cours de la première présentation effectuée par Corentin, certains doutes ont été émis sur la capacité de généralisation de notre modèle. En effet, même si la prédiction finale de notre réseau était effectuée sur notre jeu de test qui n'avait jamais été vu par le réseau durant son apprentissage, le jeu de données d'entraînement provenait d'une simulation similaire et l'on peut alors se demander si les excellentes performances du réseau n'étaient pas dûes à cette grande proximité entre les données du jeu d'entraînement et celles du jeu de test.
Le filtrage vu en 5.2 aura été reproduit sur deux nouvelles simulations dont une qui aura été mise à notre disposition par une équipe de Princeton.
Pour la première des deux simulations, la chimie utilisée et la configuration globale de la simulation était assez semblable à celle utilisée pour notre jeu d'entraînement, mais avec une pulsation périodique supplémentaire qui était ajoutée en entrée. Dans ce cas, le réseau n'a eu aucune difficulté à s'adapter et présente une prédiction d'aussi bonne qualité que sur l'ancien jeu de test.

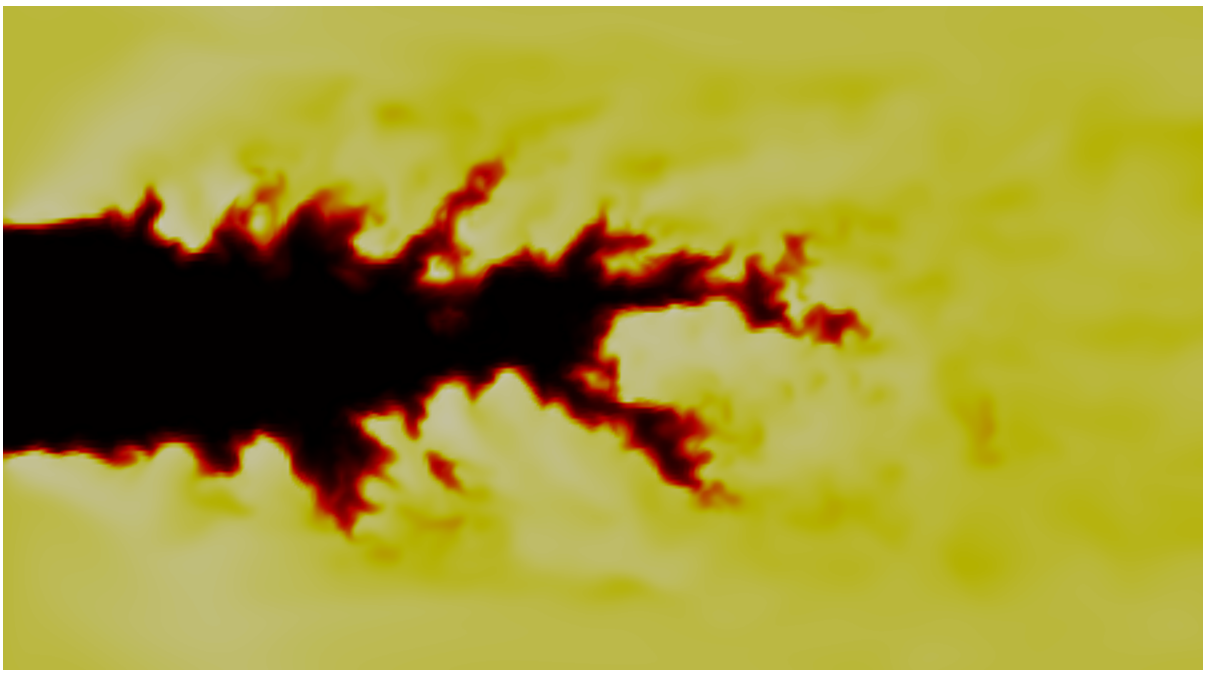


FIGURE 21 – Image "non pulsée" extraite du jeu d'entraînement.

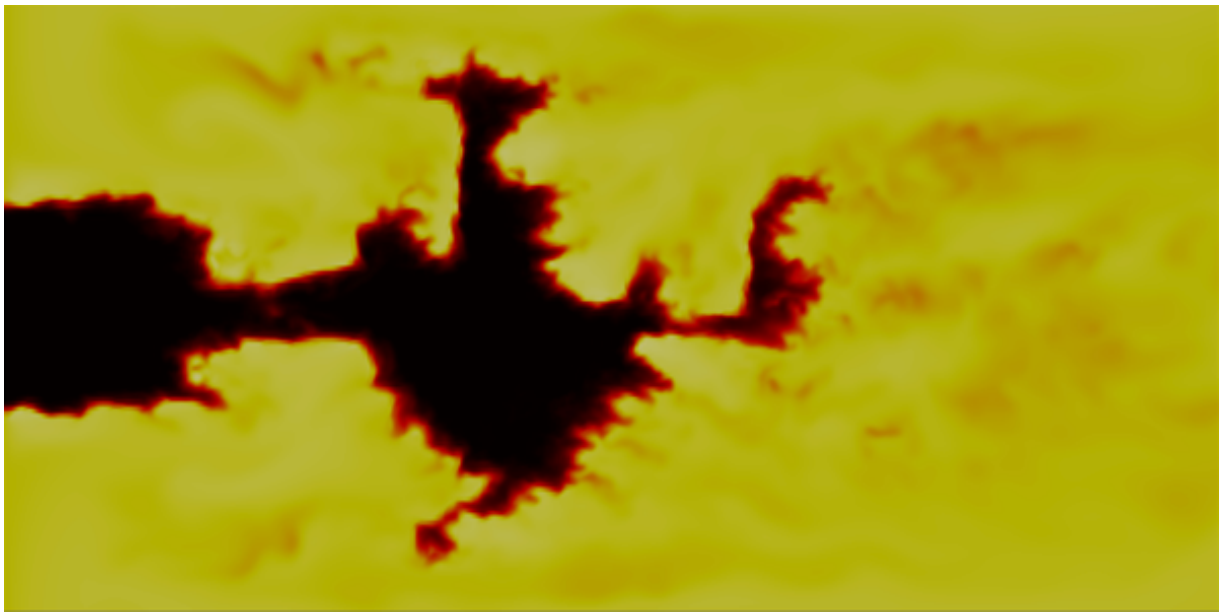


FIGURE 22 – Image "pulsée" extraite du jeu de test.

Cependant pour la simulation de l'équipe de Princeton, la chimie utilisée n'était plus la même, ce qui change grandement l'allure et l'épaisseur de la flamme. Dans ces nouvelles conditions, très différentes des données sur lesquelles le réseau avait été entraîné, les prédictions ont été de piètre qualité.
Plusieurs solutions pourraient permettre d'aider à la généralisation, en intégrant différentes configurations dans la base de données d'apprentissage et en ajoutant de nouveaux **channels** en entrée du réseau pour permettre au réseau de différencier chaque cas par exemple. Cela fait partie des points qu'il reste à perfectionner et à poursuivre pour cette étude.

6 Conclusion

Au cours de ce stage, nous avons exploré le potentiel du Deep Learning en tant qu'outil pour différentes tâches. Dans un premier temps, un CNN inspiré d'un U-net aura été mis en place pour détecter le front de flamme d'un feu de forêt alors que cette tâche présentait de grosses difficultés pour les modèles physiques existants, tout en minimisant le plus possible l'effort d'annotation humaine nécessaire à la création d'une base de données.
Ce même modèle, au vu de ses très bons résultats, aura ensuite été adapté pour faire de la régression en 3 dimensions, afin d'être utilisé pour prédire la surface totale d'une flamme complètement résolue à partir d'une flamme sous résolue sur un maillage LES.
Des modèles algébriques issues de la littérature ont été implémentés pour être comparés à notre CNN, et il a ainsi été montré que notre réseau obtenait de bien meilleurs résultats pour cette tâche.
Des travaux plus approfondis sont encore nécessaires, afin de rendre notre modèle encore plus généralisable et plus robuste, mais nous pensons que ce travail qui est la première application d'un CNN sur un problème de modélisation de combustion turbulente, a démontré de suffisamment bons résultats pour ouvrir la voie à l'utilisation de modèles de Deep Learning pour assister des simulations de mécanique des fluides.

7 Annexe

Prise en compte de l'impact environnemental

La bonne performance des modèles mis en place permet de remplacer des modèles déjà existants, permettant d'estimer le plissement sous résolu d'une flamme, qui sont beaucoup plus coûteux en calcul.
Cela aboutit donc à une réduction énergétique pour effectuer ces opérations.
En effet, les calculs nécessaires actuellement pour effectuer cette tâche sont effectués sur des supercalculateurs qui sont assez gourmand en énergie, par exemple pour les deux supercalulateurs actuellement en place au CERFACS :

- Nemo a une consommation énergétique de 98KW pour une puissance crête de 280 Tflops,
- Kraken, le supercalculateur opérationnel depuis début 2018, a lui une consommation énergétique de 47KW pour une puissance crête de 320 Tflops.

Or, la méthode que nous proposons nécessite des besoins en calcul bien plus faible sur le long terme.
Certes, l'entraînement du modèle est assez coûteux (il nécessite plusieurs heures de calculs sur un GPU NVIDIA TESLA V100), mais une fois entraîné, chaque inférence est bien moins gourmande en énergie que les méthodes traditionnelles.

Respect du planning prévisionnel

Le planning que nous avons mis en place au cours du pré-rapport de stage a été respecté, et le stage a pu être effectué dans les délais.
Toutefois, des choix ont dû être faits sur les différentes méthodes à implémenter afin de suivre ce planning, et les **GANs** par exemple ont été laissés de côté au profit des modèles convolutionnels de type **U-net**.

Références

[1] M. Boger et al. "Direct Numerical Simulation analysis of flame surface density concept for Large Eddy Simulation of turbulent premixed combustion". In : *27th Symp. (Int.) on Combustion*. Boulder : The Combustion Institute, Pittsburgh, 1998, p. 917–927.

[2] F. Charlette, D. Veynante et C. Meneveau. "A power-law wrinkling model for LES of premixed turbulent combustion : Part I - non-dynamic formulation and initial tests". In : 131 (2002), p. 159–180.

[3] Francois Chollet. *Deep learning with python*. Manning Publications Co., 2017.

[4] J. Deardorff. "A numerical study of three-dimensional turbulent channel flow at large Reynolds numbers". In : 41 (1970), p. 453–480.

[5] elitedatascience. *Overfitting in Machine Learning*. 2017. url : `https://elitedatascience.com/overfitting-in-machine-learning`.

[6] P. Givi. "Model-free simulations of turbulent reactive flows". In : 15 (1989), p. 1–107.

[7] Ian Goodfellow et al. *Deep learning*. T. 1. MIT press Cambridge, 2016.

[8] Ian Goodfellow et al. "Generative adversarial nets". In : *Advances in neural information processing systems*. 2014, p. 2672–2680.

[9] F. Gouldin, K. Bray et J. Y. Chen. "Chemical closure model for fractal flamelets". In : 77 (1989), p. 241.

[10] Sergey Ioffe et Christian Szegedy. "Batch normalization : Accelerating deep network training by reducing internal covariate shift". In : *arXiv preprint arXiv :1502.03167* (2015).

[11] A. R. Kerstein, W. Ashurst et F. A. Williams. "Field equation for interface propagation in an unsteady homogeneous flow field". In : *Phys. Rev. A* 37.7 (1988), p. 2728–2731.

[12] R. Knikker, D. Veynante et C. Meneveau. "A dynamic flame surface density model for large eddy simulation of turbulent premixed combustion". In : 16 (2004), p. L91–L94.

[13] U. Piomelli et J. R. Chasnov. "Large eddy simulations : theory and applications". In : *Turbulence and Transition Modelling*. Sous la dir. de H. Hallback et al. Kluwer Academic Publishers, 1996, p. 269–336.

[14] H. Pitsch et N. Peters. "Unsteady flamelet modeling of turbulent hydrogen-air diffusion flames". In : *27th Symp. (Int.) on Combustion*. The Combustion Institute, Pittsburgh, 1998, p. 1057–1064.

[15] T. Poinsot, S. Candel et A. Trouvé. "Application of direct numerical simulation to premixed turbulent combustion". In : 21 (1996), p. 531–576.

[16] T. Poinsot et D. Veynante. *Theoretical and Numerical Combustion*. Third Edition (www.cerfacs.fr/elearning), 2011.

[17] Bharath Raj. *Data augmentation*. 2017. URL : https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced.

[18] Olaf Ronneberger, Philipp Fischer et Thomas Brox. "U-net : Convolutional networks for biomedical image segmentation". In : *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, p. 234–241.

[19] T. Schønfeld et M. Rudgyard. "Steady and Unsteady Flows Simulations Using the Hybrid Flow Solver AVBP". In : 37.11 (1999), p. 1378–1385.

[20] L. Selle et al. "Compressible Large-Eddy Simulation of turbulent combustion in complex geometry on unstructured meshes". In : 137.4 (2004), p. 489–505.