Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
polytech.univ-tours.fr

Projet Recherche & Développement

2019-2020

Interprétabilité des IA : application à l'analyse de séquences/trajectoires

Tuteur académique
Nicolas Ragot

Étudiant
Romain Hérault (DI5)

12 avril 2020

Liste des intervenants

Nom	Email	Qualité
Romain Hérault	romain.herault-2@etu.univ-tours.fr	Étudiant DI5
Nicolas Ragot	nicolas.ragot@univ-tours.fr	Tuteur académique, Département Informatique

Avertissement

Ce document a été rédigé par Romain Hérault susnommé l'auteur.

L'Ecole Polytechnique de l'Université François Rabelais de Tours est représentée par Nicolas Ragot susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnait assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnait qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document

Romain Hérault, *Interprétabilité des IA : application à l'analyse de séquences/trajectoires*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2019-2020.

```
@mastersthesis{
    author={Hérault, Romain},
    title={Interprétabilité des IA : application à l'analyse de séquences/trajectoires},
    type={Projet Recherche \& Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2019-2020}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document polytech.cls V2.2.5 (2017/05/02)

Table des matières

Table des figures

5 Mise en oeuvre

D Gestion de projet

G Guide du développeur

Liste des tableaux

1 Introduction

1 Acteurs, enjeux et contexte

Acteurs

Les acteurs de ce projet sont Nicolas Ragot, qui en est l'encadrant et qui fait office de MOA ainsi que Romain Hérault qui constitue la MOE.

Enjeux et contexte

Dans l'analyse prédicitve, on utilise des IA pour extraire des informations à partir de données de manière à pouvoir prédire des tendances futures ou à classer ces données. Ces logiciels peuvent avoir beaucoup d'utilisations, comme par exemple classifier une image ou un texte, diagnostiquer une maladie en se basant sur les symptômes, suggérer un produit correspondant à un client, ou encore traduire un texte. L'intérêt croissant pour ces technologies ainsi que leur efficacité fait que ce type d'algorithme est de plus en plus utilisé dans des domaines parfois critiques. Or ces algorithmes ont pour la plupart un inconvénient, leur manque d'interprétabilité.

L'interprétabilité d'un modèle est son habilité à expliquer ou fournir une signification dans des domaines compréhensibles par des humains sur la façon dont il obtient ses résultats.

De nos jours, la plupart des algorithmes de machine learning ne sont pas interprétables, ce qui fait qu'il est impossible d'expliquer pourquoi ils ont les résultats qu'ils obtiennent. Par exemple, lorsque l'on utilise un réseau de neurones pour classifier des images de chiens et de chats, il est impossible de dire si l'IA s'est basée sur l'animal présent sur la photo, sur le fond, ou encore sur l'éclairage pour prendre sa décision. Dans ce cas de classification d'images, tant que le résultat est bon, le manque d'interprétabilité n'est pas gênant, mais il existe beaucoup de domaines où cela peut s'avérer critique. Imaginons une banque dont l'algorithme choisi si ses clients ont le droit ou non d'avoir un crédit. Sans modèle interprétable, il est impossible pour cette banque d'expliquer au client pourquoi il n'y a pas eu droit. Une autre utilisation des modèles interprétables est de pouvoir voir, si jamais les prédictions sont fausses, pourquoi la machine fait des erreurs, et de corriger ces biais.

Le domaine de l'interprétabilité des IA est donc un domaine encore nouveau, mais dont l'intérêt est croissant. Le but de ce PRD est de faire une veille des méthodes existantes, d'en prendre en main une, et de pouvoir l'appliquer sur un exemple concret de projet du laboratoire utilisant des données spatio-temporelles pour lesquelles il n'existe que peut de travaux en interprétabilité.

2 Objectifs

Le PRD contient plusieurs objectifs. La première partie consiste à faire une veille technologique des différentes méthodes pour rendre un modèle interprétable. Il existe de nombreuses méthodes qui s'appliquent de plusieurs manières, et sur des données différentes. Il s'agira ensuite de choisir une de ces méthodes et de la prendre en main de manière à pouvoir l'appliquer sur un classificateur utilisé par les chercheurs du laboratoire. Enfin une troisième partie pourra s'appliquer si la précédente est finie, il s'agirait soit d'essayer d'améliorer le modèle interprétable choisi, de le comparer à une autre approche, ou bien de l'expérimenter avec d'autres données.

3 Hypothèses

Après avoir fait un état de l'art (voir Chapitre 3), nous avons choisis de nous concentrer sur une méthode appelée LIME [9]. A partir de ce stade, le projet devient une étude de faisabilité qui a pour but de voir s'il est possible d'appliquer cette méthode à des données d'un type différent. S'il s'avère qu'il est trop compliqué de mettre en place cette méthode, alors il faudra repasser par une phase d'état de l'art pour essayer de trouver une autre méthode plus facile à mettre en œuvre.

4 Bases méthodologiques

Pour la gestion de projet, ce sera plutôt un modèle en cascade qui sera adopté (voir Figure 1). Ce modèle défini différentes phases qui s'enchaînent séquentiellement. Il est possible de revenir sur la phase précédente lorsque l'on doit y apporter des modifications ou pour corriger les erreurs.

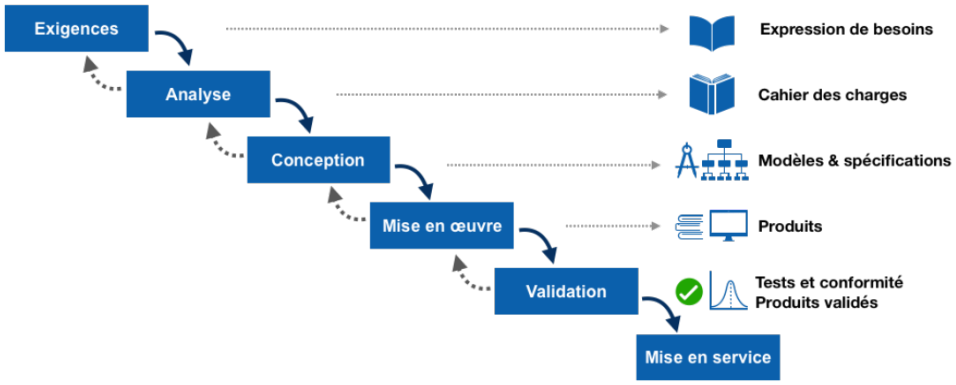


Figure 1 – *Exemple d'un modèle en cascade avec rétroaction*

Les trois premières étapes de ce modèle correspondent au semestre 9 et les trois dernières au semestre 10. Le projet est planifié avec un diagramme de Gantt créé avec Time Performance. La gestion des tâches et des restes à faire est aussi effectuée avec ce logiciel. Le code du projet sera hébergé sur GitHub pour des besoins de versionning. Les règles de programmations suivront la convention PEP8 pour le langage Python.

2 Description générale

1 Environnement du projet

Ce PRD n'est pas la suite d'un autre projet, il s'agit d'une nouvelle problématique. En revanche, il pourra s'appuyer sur des éléments déjà existants pour faciliter son développement. Le programme comporte en effet plusieurs éléments qui viendront d'autres projets :

— La base de données ainsi que le modèle prédictif viendront d'un autre projet du laboratoire,
— La méthode utilisée pour l'interprétation du modèle sera choisie parmi les études. Il y aura probablement une librairie développée par des chercheurs qui sera utilisée.

Le but du projet sera de trouver un moyen d'assembler ces éléments pour que l'on puisse interpréter un modèle. D'après une étude catégorisant les méthodes d'interprétabilité (voir [3]), deux cas sont possibles :

— Soit on modifie le modèle de manière à le rendre directement interprétable. Le modèle obtenu sera alors un modèle transparent. Le désavantage de cette façon de faire est qu'il n'est utilisable que pour un seul modèle, et donc si l'on veut l'essayer sur un autre modèle, il faut tout recommencer. De plus elle nécessite d'avoir un accès total au modèle, ce qui n'est pas toujours le cas.
— Soit on crée un nouveau modèle interprétable, se rapprochant du modèle de décision original. Cette problématique est appelée interprétation de boite noire, et consiste à créer un nouveau modèle en ne se basant que sur les résultats que l'on a pu recueillir en testant le modèle avec différentes données. Elle a l'avantage de pouvoir être utilisée sur plusieurs boites noires différentes, du moment que l'algorithme créant le modèle interprétable peut s'y appliquer. C'est préférablement cette méthode qui est à choisir.

Voici des diagrammes de composants pour ces deux cas (modèle transparent (voir Figure 1) et interprétation de boite noire (voir Figure 2)).

1.1 Les données

Les données utilisées en entrée sont des vidéos montrant les mouvements de différents liquides lorsqu'ils sont dans une machine simulant le comportement d'un piston de moteur. Ces vidéos permettent aux chercheurs de comparer les différentes propriétés des liquides lorsqu'ils sont

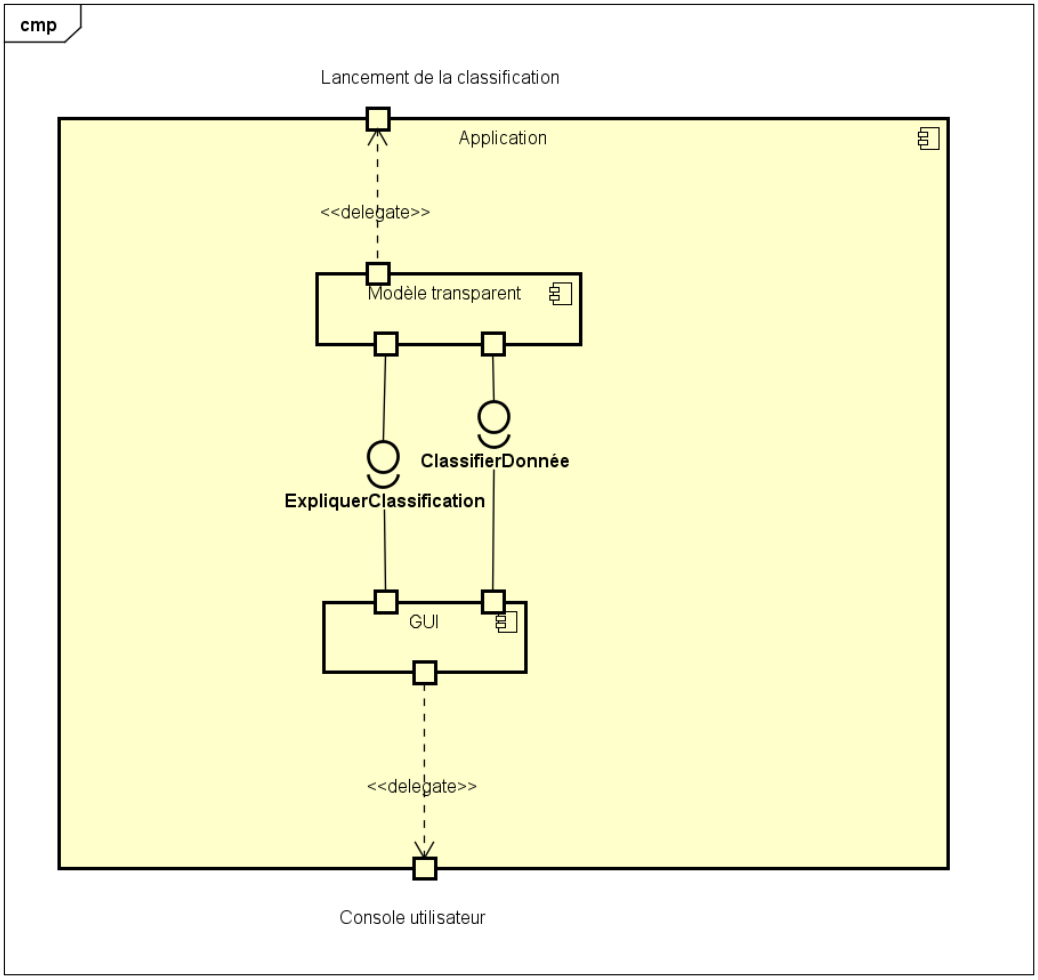


Figure 1 – *Diagramme de composant représentant la problématique du modèle transparent*

soumis aux mêmes conditions que dans les moteurs. Grâce à ces vidéos, on peut déterminer plusieurs caractéristiques, comme par exemple la manière dont ils chauffent, leur vitesse etc...

1.2 Le classificateur

Le classificateur que l'on veut rendre interprétable est un modèle prédictif fournit par le laboratoire. Il s'agit d'un réseau de neurones convolutifs spatio-temporel, appelé P3D ResNet ([6] et [8]). Celui-ci prend en entrée des vidéos, que l'on peut interpréter comme des séquences d'images. Le classificateur prend donc en entrée les vidéos dont nous avons parlé au paragraphe précédent, et fait une estimation de la vitesse de rotation du liquide. Le modèle que l'on va utiliser à l'avantage d'être déjà entraîné et on sait qu'il obtient des performances acceptables.

L'intérêt de rendre ce classificateur interprétable est de pouvoir savoir sur quoi il se base pour établir la vitesse, si c'est grâce aux pixels qui redescendent verticalement, à ceux qui au contraire remontent, ou à ceux qui montrent des perturbations par exemple. De plus, les chercheurs se sont rendu compte que l'on obtenait des meilleurs résultats en faisant ressortir les pixels les plus lumineux. Avec un modèle interprétable, on pourrait vérifier si le classificateur s'appuie bien sur ces pixels.

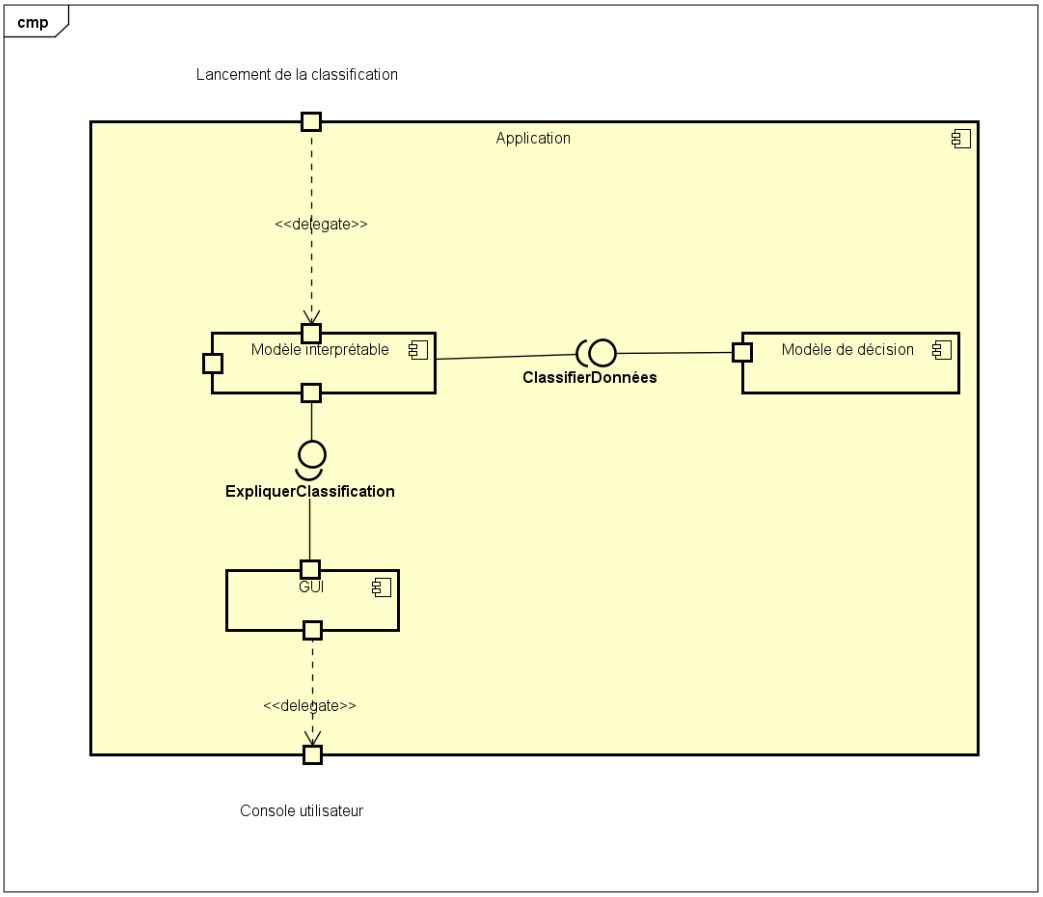


Figure 2 – *Diagramme de composant représentant la problématique de l'interprétation de boite noire*

1.3 La méthode

La méthode choisie s'appelle LIME [9] (voir état de l'art Section 4.1 (Chapitre 3)). Le principe de cette méthode est de générer un voisinage des données d'entrée et tester chacun des voisins avec le classificateur, permettant de faire ressortir les attributs ayant le plus contribués à la décision, ou au contraire ceux ayant impacté négativement la décision.

Elle a l'avantage d'être agnostique, c'est-à-dire que l'on peut l'appliquer sur n'importe quel modèle de classificateur, et de pouvoir s'appliquer sur n'importe quel type de données. De plus, le code de la méthode est disponible avec une documentation et des tutoriaux pour appliquer la méthode sur du texte, des tableaux ou des images. La plus grosse partie de ce PRD va être d'essayer d'implémenter la méthode LIME pour des vidéos.

Cette méthode peut être découpée en 4 grandes partie.

Classificateur

Elle a d'abord besoin d'un classificateur, il est indispensable étant donné que l'on veut expliquer comment il prend ses décisions. Le classificateur est directement utilisé par LIME pour classer chacun des voisins et voir quels sont les résultats.

Segmentation

Ensuite il lui faut un algorithme de segmentation adapté aux données que l'on veut traiter. Cet algorithme est utilisé pour créer les voisinages. En effet, chaque voisin étant une modification

des données d'entrée de base, il s'agit de les découper d'une certaine manière pour n'en garder que certaines parties.

Génération de l'explication

La troisième partie s'occupe de générer tous les voisins de la donnée initiale avec l'algorithme de segmentation et de passer chacun de ces voisins dans le classificateur. Les résultats du classificateur pour chacun de ces voisins nous donne des informations sur lesquels ont des impacts négatifs ou positifs sur la classification, ce qui nous permet de retrouver quelles sont les parties importantes des données d'entrée.

Affichage du résultat

La dernière partie est une fonction d'affichage qui va nous permettre de rendre les résultats générés par la partie précédente interprétables. Par exemple en retournant des images ou une vidéo où les zones importantes sont repassées en vert.

2 Caractéristiques des utilisateurs

Etant donné que l'objectif du PRD est de se faire la main sur une méthode d'interprétabilité et de voir comment tout cela fonctionne, on peut considérer que les personnes utilisant le projet seront des chercheurs dans l'analyse d'image, comme Mr.Ragot par exemple. Le projet est exploratoire et ne vise pas à avoir une application utilisable par un utilisateur lambda, ainsi les principaux utilisateurs connaîtront en détails comment utiliser l'application.

3 Fonctionnalités du système

Les fonctions utilisateurs de l'application ne sont pas nombreuses. Un utilisateur doit pouvoir choisir les données qu'il veut tester par le modèle (en l'occurrence des séquences d'images) et lancer le modèle. Un fois la donnée classée, il doit avoir une interface lui fournissant le résultat de la classification et une explication sur comment la boite noire a obtenu ce résultat (voir Figure 3). Il est probable que le choix de la séquence en entrée se fasse directement au moment du lancement de la classification par le biais d'une option dans une ligne de commande. L'affichage des résultat sera aussi fait de la manière la plus simple en indiquant un dossier ou un fichier dans lequel l'explication sera générée.

4 Structure générale du système

Nous avons vu que l'application est composée de plusieurs éléments existants et que l'enjeu du PRD est d'essayer de les faire fonctionner ensemble. Nous pouvons déjà préciser la structure interne de l'application en se basant sur le fonctionnement de la méthode LIME. En effet, toute l'application sera en fait la mise en œuvre de la méthode LIME avec un classificateur et des données particulières, donc la structure est celle utilisée dans la méthode LIME (voir Figure 4 et Figure 5). Le principe étant de faire passer un voisinage d'une donnée d'entrée dans un classificateur pour voir quels attributs sont importants, il nous faut les fonctions suivantes.

Un classificateur et une fonction permettant de l'utiliser

Le classificateur est celui que l'on veut rendre interprétable, mais pour pouvoir l'utiliser dans LIME, il faut créer une fonction faisant toutes les manipulations nécessaires pour qu'il soit utilisable directement en passant des données (séquences) à cette fonction.

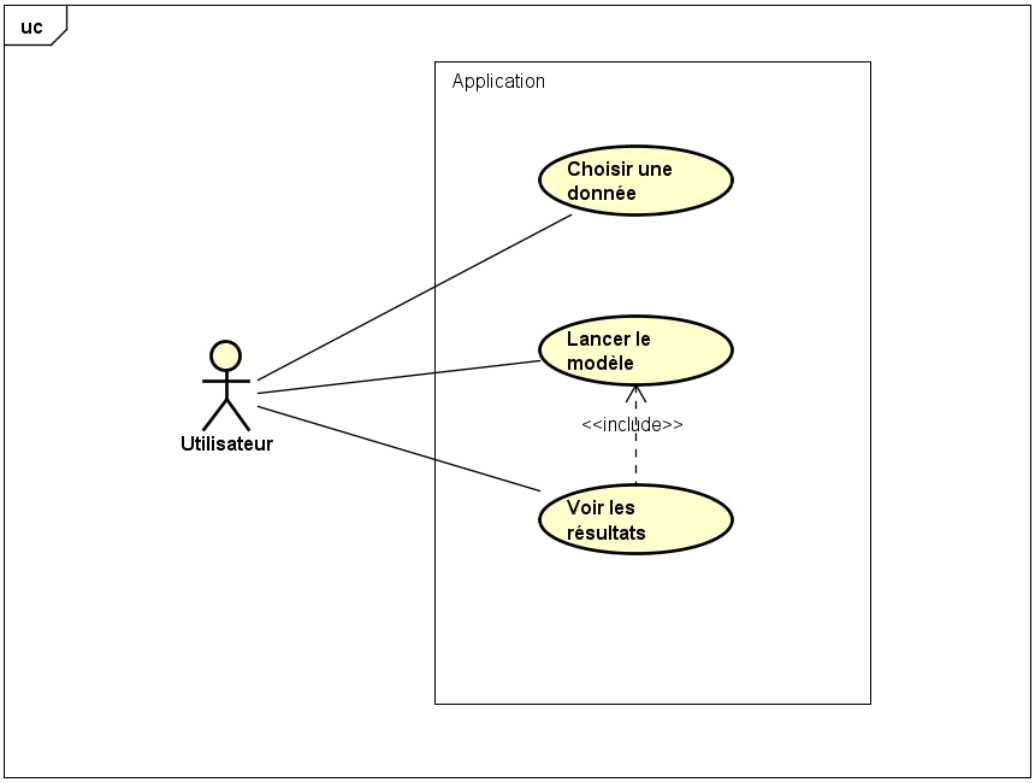


Figure 3 – *Diagramme des cas d'utilisation*

Une fonction de segmentation

La fonction de segmentation est la fonction dans laquelle sera implémentée l'algorithme de segmentation. C'est elle qui permettra de découper notre séquence en entrée en différents segments pour pouvoir plus tard générer les voisinages.

Une fonction d'explication (« explainer »)

Ce composant s'occupe d'utiliser la fonction du classificateur ainsi que la fonction de segmentation pour générer les voisins et passer chacun d'entre eux dans le classificateur, permettant de générer une explication. Cette explication contient les informations sur quels segments sont les plus importants, mais n'est pas interprétable directement.

Une fonction d'affichage

Une fois l'explication générée, il faut la rendre interprétable pour un utilisateur. Pour cela il faut la convertir en images ou vidéos et les enregistrer dans un dossier accessible à l'utilisateur.

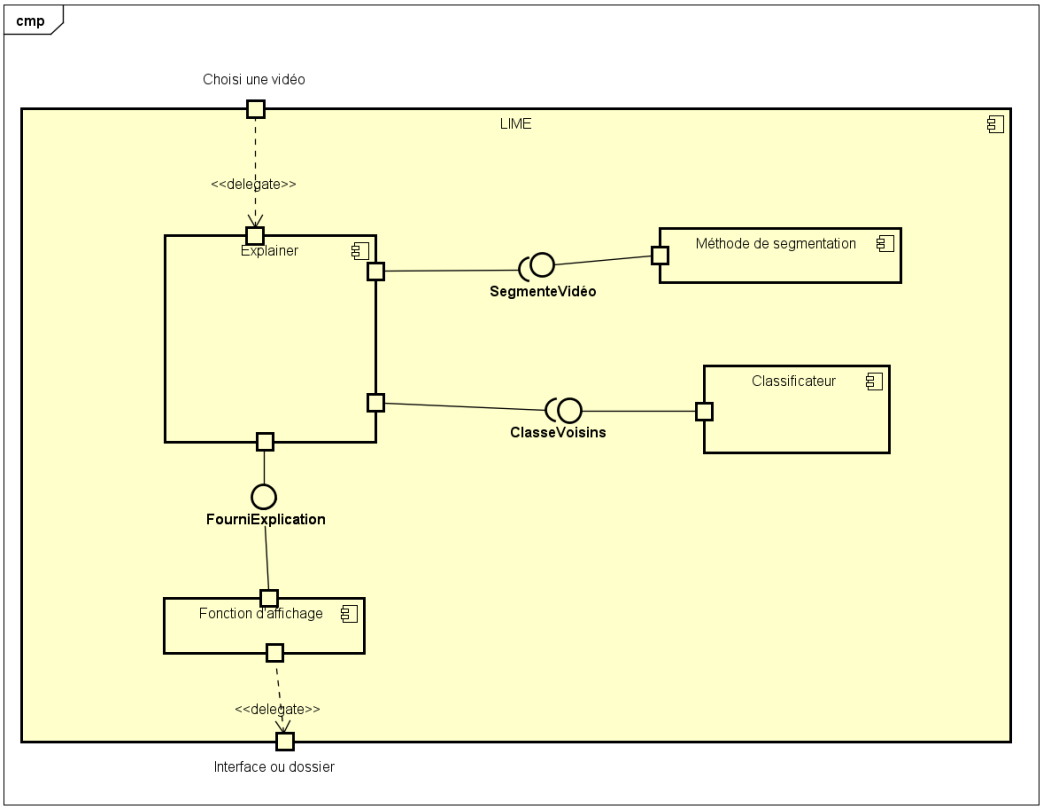


Figure 4 – *Diagramme de composants de la méthode LIME*

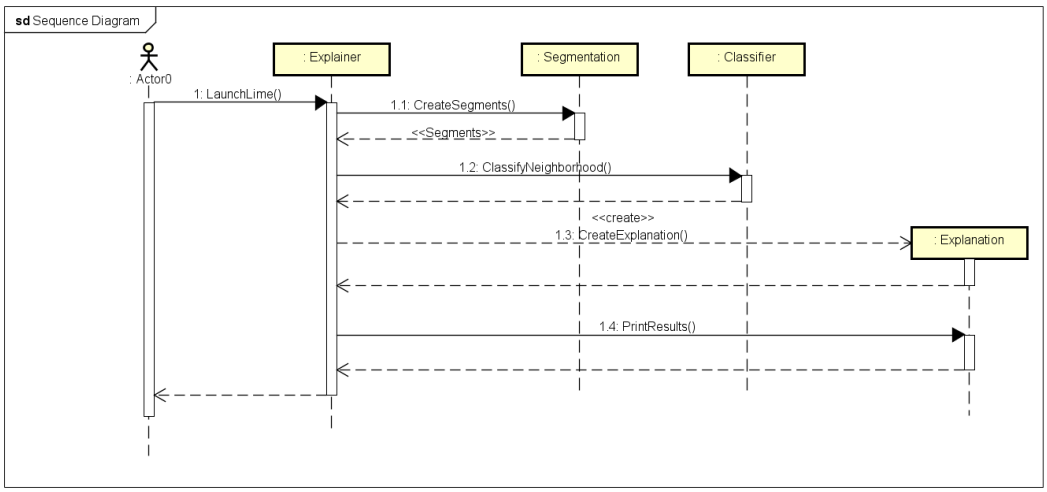


Figure 5 – *Diagramme de séquence de la méthode LIME*

3 Etat de l'art / veille

Nous allons maintenant procéder à l'état de l'art concernant l'interprétabilité des IA. Nous allons nous concentrer sur ce qu'est l'interprétabilité des IA, comment classifier les différentes méthodes d'interprétation et voir plus en détail quelques méthodes. Ces éléments vont alors nous permettre de choisir la bonne méthode à utiliser pour notre projet.

1 Interprétabilité des IA

Les informations de cette section et des deux sections suivantes (Section 2 et Section 3) viennent en grande partie de l'étude [3].

Comme nous l'avons vu plus tôt dans le contexte du projet, l'utilisation des IA est devenue de plus en plus commune. Ces IA permettent de faire des prédictions à partir des données qui leur sont fournies, et se révèlent souvent assez efficaces, d'où une augmentation de leur fréquence. Nous avons vu que la façon dont ces IA parvenaient à une solution restait dans le meilleur des cas floue. Il est en effet très difficile de savoir pourquoi un classificateur à pris tel décision, ou quelles sont les parties des données d'entrée qui l'ont poussé à choisir ce résultat. Cela peut se révéler problématique lorsque ces IA prennent des décisions sensibles, comme dans la médecine ou les assurances par exemple, mais aussi lorsqu'elles font des erreurs, puisqu'il est difficile, voir impossible de savoir pourquoi elles se trompent. Comme on ne connaît pas les mécanismes internes de ces IA, elles sont parfois surnommées des « boites noires ». Tout l'objectif de l'interprétabilité des IA va être de trouver des méthodes ou des mécanismes qui vont nous permettre d'avoir des informations sur la prise de décisions de ces boites noires.

Exemples

Voici plusieurs exemples démontrant l'intérêt de l'interprétabilité dans ce domaine.

Un algorithme calculant le score COMPAS, qui mesure le risque de récidive de crime d'une personne s'est révélé avoir un fort biais ethnique. D'après ce score, les personnes de couleur noire avaient deux fois plus de chances de récidiver que les personnes de couleur blanche.

De la même manière, une étude à Priceton a montré que les textes et le web étaient biaisés, en démontrant que les noms associés aux personnes de couleurs noires étaient significativement associés à plus de termes négatifs que positifs, comparés aux personnes de couleurs blanches. Ainsi, les modèles entraînés sur ces données se retrouvent eux aussi biaisés.

Un autre exemple concernant Amazon cette fois, où un algorithme permettant de décider des zones où la livraison gratuite en 1 jour ouvrable excluait systématiquement les quartiers contenant des minorités, même si tous leurs voisinages participaient au programme.

On peut aussi trouver d'autres cas dans les banques américaines, où, en prenant 3 d'entre elles, les scores attribués aux clients pour établir des crédits diffèrent d'au moins 50 points pour 29% d'entre eux.

L'armée américaine a aussi été victime de ces biais. Ainsi, un programme destiné à détecter si un char est ennemi ou allié avait de très mauvaises performances une fois utilisé sur le terrain à cause de la couverture nuageuse. En effet, sur les données d'entraînement, les chars alliés étaient présentés lors d'un beau ciel dégagé alors que les chars ennemis l'étaient sous un ciel nuageux.

Un autre cas similaire pour un algorithme permettant de détecter des huskys ou des loups, qui lui se basait essentiellement sur la présence de neige en arrière-plan pour parvenir au résultat.

Plus généralement, les réseaux de neurones profonds qui sont très utilisés de nos jours peuvent parvenir à des résultats complètement aberrants en ne modifiant que quelques données bien choisies. Par exemple il est possible de tromper un CNN (classificateur d'image) en ne modifiant que quelques pixels, de manière complètement invisible à l'œil nu, et d'arriver à des résultats complètement différents de ce qui est vraisemblablement représenté sur l'image.

Définitions

On peut définir l'interprétabilité comme étant l'habilité à expliquer ou fournir une signification dans des termes compréhensibles par des humains. Ainsi, une explication d'une boite noire sera une interface entre un humain et l'algorithme que prend la décision et qui représente précisément la prise de décision tout en étant compréhensible par un humain.

2 Construire un modèle interprétable

2.1 Dimensions

Pour rendre une IA interprétable, et pour définir nos explications, il faut prendre en compte plusieurs dimensions. La première question a se poser, est si l'on veut expliquer notre classificateur complètement et quelle est sa logique, ou si l'on ne veut expliquer que comment il parvient à un résultat en particulier. Le premier cas est une interprétation globale, alors que le deuxième est une interprétation locale. L'interprétation globale, qui donc permet d'expliquer la logique du fonctionnement du classificateur, peut consister par exemple à recréer un autre modèle basé sur le premier qui serait lui interprétable. L'interprétation locale consiste plutôt à fournir des éléments permettant de décider pourquoi l'algorithme à pris une décision pour une instance donnée.

Pour fournir une explication, il faut aussi choisir la complexité de celle-ci. Pour cela, on a besoin de connaître l'utilisateur, et notamment son niveau d'expertise. Si c'est un expert dans le domaine, on peut se permettre de mettre des explications plus complètes et plus complexes, alors que si c'est une personne n'ayant que peu de compétences dans le domaine, il vaut mieux privilégier les explications simples et courtes.

Il faut aussi prendre en compte le temps que l'utilisateur a à passer sur l'interprétation du résultat. Si l'utilisateur doit approuver le résultat en peu de temps (un désastre imminent par exemple), la complexité de l'explication doit être différente de s'il n'a pas de limitation temporelle.

2.2 Desiderata d'un modèle interprétable

Voici les pré-requis nécessaires aux modèles interprétables :

— Interprétabilité : le modèle ou ses prédictions doivent être compréhensibles par un humain (elle est souvent mesurée avec la complexité du modèle en termes de taille,
— Précision : le modèle est capable de prédire correctement des nouvelles instances,
— Fidélité : Si le modèle interprétable n'utilise pas directement le modèle de décision à interpréter (mais en est une approximation par exemple), le modèle interprétable doit l'imiter correctement. Cela veut dire que le résultat pour une instance donné du modèle interprétable doit être le même que celui du modèle qu'il interprète pour la même instance.

Ces prérequis sont liés à l'interprétabilité du modèle, mais il y en a d'autres qui sont liés aux modèles issus du machine-learning et de datamining. :

— Equité : permet de protéger les groupes contre la discrimination,
— Confidentialité : s'assurer que l'on ne révèle pas de données sensibles,
— Monotonicité : par exemple, augmenter une valeur numérique en entrée doit modifier manière monotone la probabilité que le résultat appartienne à une classe,
— Ergonomie : le modèle doit être facile à utiliser, l'utilisateur aura plus confiance en quelque chose qu'il arrive à utiliser facilement,
— Fiabilité, robustesse, causalité, scalabilité, généralité : principes importants que l'on retrouve régulièrement.

3 Classification de méthodes

On peut classifier les différentes méthodes d'interprétabilité selon plusieurs éléments :

— le type de problème rencontré,
— le type d'explication fournie,
— le type de modèle de la boite noire,
— le type de données utilisées en entrée du modèle.

3.1 Type de problème rencontré

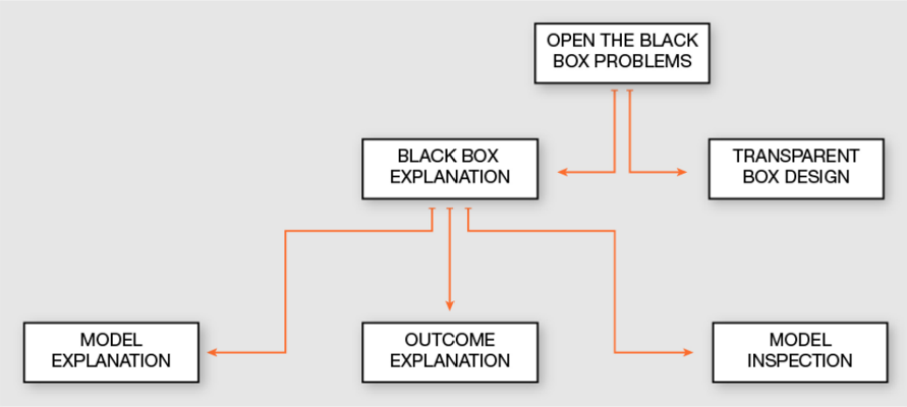


Figure 1 – *Les différents types de problèmes*

Le type de problème rencontré peut se classer selon 2 catégories principales, dont une peut être séparée de nouveau en trois sous-catégories (voir Figure 1). Les deux catégories principales sont le « transparent box design » et la « black-box explanation ».

3.1.1 Transparent Box Design

Le transparent box design consiste à, dès la création du modèle, faire en sorte qu'il soit interprétable. Ainsi, les méthodes utilisées pour les rendre interprétable ne peuvent s'appliquer qu'au modèle utilisé, et il est impossible de refaire la même chose si l'on change de modèle. Le modèle est donc transparent et complètement explicable. Ce type de problème est à utiliser pour des situations spécifiques où le besoin d'avoir un modèle transparent est vital. Ce n'est pas le type de méthode que l'on souhaite appliquer dans notre cas puisqu'il est impossible de l'appliquer à un cas différent de celui pour lequel il a été initialement construit.

3.1.2 Black-box Explanation

La deuxième catégorie principale, que l'on peut traduire par « explication de boite noire », correspond aux méthodes que l'on peut appliquer aux boites noires sans connaître leurs mécanismes internes, voir même sans connaître les données avec lesquelles elles ont été entraînées. On essaie donc de reconstruire le modèle seulement en se basant sur ses résultats, ce qui fait que cette méthode est une sorte de « reverse engineering ». L'utilisation de ces méthodes repose donc sur des essais de la boite noire avec différentes données dans le but d'en deviner son fonctionnement, puis de pouvoir l'expliquer. Cette catégorie se divise en trois autres :

— Explication du modèle
— Explication d'une instance
— Inspection du modèle

Explication du modèle

L'explication du modèle revient à trouver une explication globale du modèle et permet de comprendre comment le modèle fonctionne dans sa globalité. Ainsi, on peut expliquer comment sont formés les différents résultats. Par exemple, cela peut être un arbre de règles de décision montrant toutes les possibilités et les résultats de la boite noire.

Explication de l'instance

L'explication d'une instance revient à se placer dans une logique locale, c'est-à-dire d'expliquer comment on arrive à un résultat pour une instance bien précise. Cela peut être par exemple un masque sur l'image que l'on a en entrée montrant les zones de l'image qui ont été importantes lors de la décision.

Inspection du modèle

L'inspection du modèle est différente des deux cas précédents dans le sens où l'on cherche à fournir une représentation des propriétés spécifiques au modèle ou à ses prédictions. Cela peut être par exemple la sensibilité du modèle à changer le résultat lorsque l'on modifie certaines parties des données d'entrée, ou bien de mettre en valeur les parties du modèle qui sont responsables des décisions, comme par exemple mettre en valeur les nœuds ayant les plus contribués à la décision pour un CNN.

3.2 Type d'explication fournie

Une méthode d'interprétation doit pouvoir permettre d'expliquer une boite noire de manière à ce que l'on puisse comprendre son fonctionnement. Pour cela, il est possible de fournir les informations nécessaires à sa compréhension de différentes façons.

Règles de décision

Les règles de décision permettent une représentation textuelle du classificateur. Il s'agit simplement d'un ensemble de règles, ce qui est facilement interprétable pour un humain. Cette forme de classificateur fait qu'il est interprétable à la fois globalement car il explique le modèle dans son ensemble, et localement car il peut expliquer une solution en particulier. On peut aussi l'utiliser dans le cas d'un design transparent.

Arbre de décisions

Les arbres de décisions sont un ensemble de décisions que l'on peut représenter graphiquement sous la forme d'arbres. Chaque feuille de l'arbre correspond à une décision et il suffit donc de parcourir l'arbre en suivant les règles pour avoir le résultat de notre instance. Cette forme est elle aussi facilement interprétable pour un humain, et fonctionne aussi bien globalement que localement. Il existe aussi des méthodes pour transformer un arbre de décision en un ensemble de règles de décisions.

Features Importance

Les features importance consistent à renvoyer les poids et les magnitudes des attributs de la boite noire. Cette méthode peut aussi être utilisée globalement, en indiquant les attributs privilégiés dans le choix des décisions de la boite noire, ou localement, en montrant les attributs qui ont le plus ou le moins participé à une décision en particulier.

Saliency Maks

Un saliency mask est une façon efficace que montrer les parties importantes d'une donnée d'entrée qui ont influé sur le résultat. Il s'agit de mettre en valeur les parties utiles des données d'entrée. Cela est applicable pour des images ou pour du texte, en surlignant les portions qui ont un fort impact sur la prise de décision.

Sensivity analysis

Une explication basée sur l'analyse de la sensibilité d'un modèle consiste à voir quels sont les éléments qui, lorsqu'ils sont modifiés, changent le résultat. On étudie donc la sensibilité de la boite noire au changement du résultat.

Partial dependence plot

Le graphe de dépendance partielle est une représentation graphique permettant de mettre en relation les entrées et les sorties du modèle dans un espace vectoriel réduit.

Prototype selection

La sélection de prototype est une méthode d'interprétation consistant à retourner en même temps que la sortie un exemple similaire (appelé prototype) à l'enregistrement d'entrée, qui rend clair sur quels critères la boite noire s'est basée pour prendre sa décision.

Activation maximization

Cette méthode d'interprétation s'applique pour les réseaux neuronaux et consiste à observer les neurones fondamentaux activés pour des enregistrements particuliers de manière à trouver quels paternes sont responsables de l'activation de certains neurones dans certaines couches.

Tous ces différents types d'explication influent sur la manière dont les résultats sont présentés à l'utilisateur, certains étant plus explicites sur le fonctionnement de la boite noire que d'autres, certains étant plus visuels, mais ils ne peuvent pas tous s'appliquer sur les mêmes types de boites noires.

3.3 Type de modèle de la boite noire

La boite noire que l'on veut observer suit un modèle en particulier. Ces modèles peuvent être regroupés en catégories qui vont être interprétés différemment en fonction de leurs particularités.

Voici une liste des modèles que l'on retrouve les plus fréquemment dans les études traitant d'interprétabilité :

Réseaux de neurones (NN)

Les réseaux de neurones sont des réseaux de nœuds artificiels inspirés des réseaux de neurones biologiques. Ces nœuds (appelés neurones) sont organisés en plusieurs couches qui appliquent chacune des transformations différentes aux données d'entrée. Le signal passe donc par la couche d'entrée, ressort par la couche de sortie, en passant à travers un certain nombre de couches cachées entre les deux. Chaque neurone et connexion possède des poids qui varient au cours du processus d'entraînement, augmentant ou diminuant la force du signal.

Tree ensemble (TE)

Les méthodes ensemblistes utilisent plusieurs algorithmes d'apprentissage pour essayer d'améliorer le niveau de prédictions de chacun des algorithmes qu'elles combinent.

Support Vector Machine (SVM)

Un SVM est un classificateur qui se base sur des vecteurs particuliers (support vector) et des transformations de l'espace de représentation des données dans des dimensions plus grandes de manière à y trouver des séparations linéaires.

Réseaux de neurones profonds (DNN)

Les réseaux de neurones profonds sont des réseaux de neurones avec une multitude de couches, leur permettant de modéliser des relations complexes non linéaires. Il en existe de différentes catégories comme les réseaux de neurones récurrents (RNN), dans laquelle un type de modèle particulier appelé LSTM permet de traiter les séquences, ce qui est particulièrement efficace dans les domaines liés au langage. Il y a aussi les réseaux de neurones convolutifs (CNN) qui sont utilisés dans le traitement d'images. Il existe énormément de types différents de DNN et la seule différence qu'ils ont avec les NN est qu'ils sont plus profonds, permettant plus d'architectures différentes, et sont donc plus variés.

Modèles non-linéaires (NLM)

La fonction utilisée pour modéliser les observations est une combinaison non linéaire des paramètres du modèle et dépend d'une ou plusieurs variables indépendantes.

Il existe des approches qui sont agnostiques, c'est-à-dire qu'elles peuvent théoriquement s'appliquer à n'importe quel type de modèle de décision. Par exemple une méthode agnostique peut très bien fonctionner avec un réseau de neurones tout comme avec un SVM ou un tree ensemble.

3.4 Type de données utilisées en entrée du modèle

Dans la plupart des études sur le sujet, les principaux types des données utilisés en entrée du modèle sont :

Tableaux

Les données sont présentées sous forme de tableaux, dans lesquels chaque enregistrement partage le même jeu d'attributs et chaque attribut peut être numérique, décrire une catégorie ou être un booléen.

Images

Beaucoup de boites noires servent à catégoriser des images. Les images peuvent être directement utilisées par la boite noire ou peuvent être modifiées avant d'être utilisées (pre-processing).

Texte

Le texte est notamment utilisé dans la modélisation de langage. On utilise beaucoup de boites noires dont le rôle est de catégoriser un texte, pour par exemple reconnaître des spams ou classer des sujets.

Très peu de modèles interprétables existent pour d'autres types de données, comme des séquences, des données spatio-temporelles ou encore des réseaux complexes. En général, les méthodes utilisant ces données sont des transparent box design, et ne sont applicables qu'aux cas qu'elles traitent. En revanche, il existe des méthodes qui peuvent fonctionner avec n'importe quel type de données (souvent des méthodes qui sont déjà agnostiques au niveau du modèle de décision).

Le type de données a son importance pour déterminer quel modèle on peut utiliser, et donc quel type d'explication on doit fournir, il s'agit donc d'un paramètre important pour choisir la méthode de classification.

3.5 Résumé

Une fois que l'on a tous ces éléments, il est possible de catégoriser toutes les méthodes d'interprétabilité différentes et d'en choisir une plus simplement. Un tableau extrait de l'étude [3] et résumant un grand nombre de méthodes est disponible en annexe Section 4 (Annexe H).

Un diaporama résumant l'étude [3] est disponible Section 2 (Annexe H).

4 Méthodes détaillées

Voici quelques méthodes étudiées au cours de ce PRD. Pour la plupart d'entre-elles, un résumé plus détaillé sous forme de diaporama est disponible en annexe.

4.1 LIME (Local Interpretable Model-agnostic Explanations)

Table 1 – *Caractéristiques de LIME*

Année	2016
Type de problème	Explication d'une instance
Explication fournie	Features Importance
Modèle de la boite noire	Agnostique
Type de données	Toutes

4.1.1 Type de problème

LIME [9] est une méthode assez récente pour interpréter une IA. Le type de problème résolu par cette méthode est l'explication d'une instance, c'est-à-dire que cette méthode permet de comprendre pourquoi le classificateur a obtenu un résultat en particulier. Elle ne permet pas d'expliquer le modèle dans son intégralité, mais seulement des exemples.

4.1.2 Principe

Le principe de cette méthode est assez simple, il s'agit de créer un voisinage à partir des données d'entrée. Nous appellons voisinage un ensemble de répliques de ces données en entrée, mais dont chacune est légèrement modifiée par rapport à la donnée de base. Une fois que l'on a généré le voisinage, on fait passer chacun des voisins dans le classificateur et on regarde les résultats obtenus. Il suffit ensuite de faire le lien entre les résultats que l'on a eu est les données des voisins utilisés pour connaître quelles sont les parties des données d'entrée de base qui sont importantes.

Par exemple pour un modèle labélisant des images, la méthode va d'abord découper cette image en morceaux, puis créer plusieurs images qui seront des répliques de l'image d'entrée avec certains des morceaux masqués (en noir ou de couleur uniforme). Cet ensemble d'images forme le voisinage. Ensuite on fait passer chacun des voisins dans le classificateur pour obtenir une explication qui va regrouper l'image d'entrée, les « morceaux » de l'image et leur importance à chacun. Il suffit pour finir d'afficher l'image d'entrée en recouvrant en vert les morceaux qui ont le plus grand impact positif sur le résultat et en rouge ceux qui ont le plus grand impact négatif (voir Figure 2).

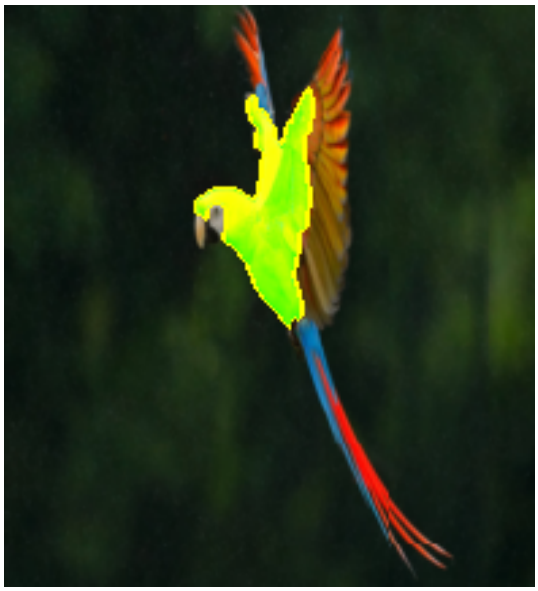


Figure 2 – *Exemple de résultat de LIME avec une image, les parties importantes de l'image pour le classificateur sont surlignées en vert*

4.1.3 Avantages

Théoriquement, LIME est model-agnostic, ce qui veut dire que l'on peut l'utiliser avec n'importe quel classificateur, et sur n'importe quel type de données d'entrée. Ce point est important pour

le choix de la méthode dans notre projet, car cela veut dire qu'il est théoriquement possible d'adapter la méthode avec des vidéos et le classificateur que nous avons de fourni par le laboratoire.

De plus, le code-source de la méthode est disponible sur github, ainsi que des exemples et des tutoriaux. Cependant le cas des vidéos ne sont pas implémentés, nous avons seulement des exemples avec des modèles classifiant des textes, des tableaux, ou des images.

Nous pouvons commencer par étudier cette méthode avec des images, car cela se rapproche le plus des vidéos.

4.2 ANCHORS

Table 2 – *Caractéristiques de ANCHORS*

Année	2018
Type de problème	Explication d'une instance
Explication fournie	Règles de décision
Modèle de la boite noire	Agnostique
Type de données	Toutes

4.2.1 Type de problème rencontré

Anchors [7] est une amélioration de LIME, le type de problème rencontré reste le même.

4.2.2 Principes et avantages par rapport à LIME

Tout comme LIME, Anchors est une méthode s'appliquant sur des instances en particulier et qui est model-agnositc. Anchors est une amélioration de LIME. L'objectif de cette méthode est d'améliorer l'interprétabilité de LIME. Il se base sur les mêmes concepts, mais au lieu de montrer les attributs des données d'entrées qui sont importantes (comme les mots ayant le plus de poids dans un texte), cette méthode créer des règles de décisions appelées anchors. Ces règles sont plus facilement interprétables pour un humain, ce qui fait la force de cet algorithme.

Sur la Figure 3, on cherche à déterminer si une phrase à une connotation positive ou négative. On peut voir que l'explication fournie par LIME n'est pas assez explicite, alors que les règles fournies par Anchors nous permettent de voir facilement que c'est l'association des mots « not » et « bad » qui donnent une connotation positive à la phrase par exemple.

Les anchors sont des règles de type « si-alors » qui sont construites de manière à ce que si l'on change dans les données d'entrée les attributs qui ne sont pas compris dans les anchors, le résultat ne change pas.

En termes de classification d'images (ce qui nous intéresse le plus dans notre cas), la méthode implémentée de anchors ne semble pas très différente de LIME. En effet il est difficile d'écrire des règles pour une image, alors une anchors sera les morceaux d'images les plus importants, de manière à ce que si l'on change complètement le reste de l'image, le résultat de la classification reste le même. Pour faire cela, au lieu de cacher simplement certains morceaux des images du voisinages, ils les ont remplacés par des images complètement différentes. Cela permet de s'assurer que les morceaux restants sont vraiment ceux sur lequel se concentre le classificateur.

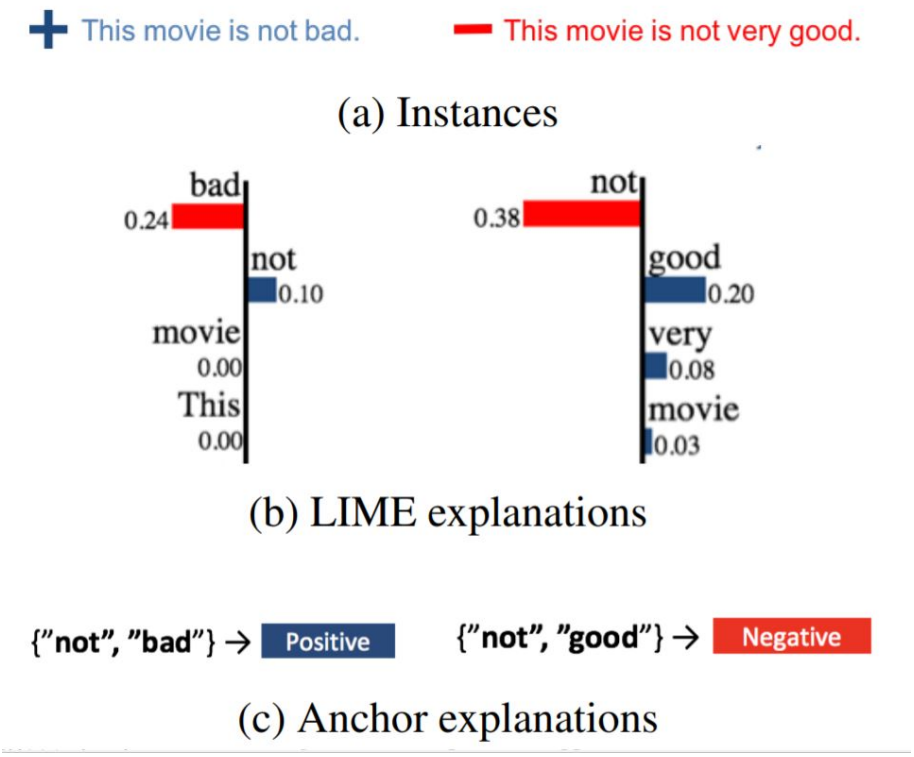


Figure 3 – *Anchors vs LIME*

Les tutoriaux de cette méthode ne concernent que les tableaux et les textes, et cette méthode reste plus difficile à implémenter que LIME, pour des bénéfices assez peu intéressants sur des images. Dans le cadre du PRD, LIME reste donc une meilleure option.

4.3 LORE (Local Rule-Based Explanations)

Table 3 – *Caractéristiques de LORE*

Année	2018
Type de problème	Explication d'une instance
Explication fournie	Règles de décision
Modèle de la boite noire	Agnostique
Type de données	Tableaux

4.3.1 Type de problème rencontré

LORE [2] est une autre variante de LIME, elle s'applique toujours au même type de problème (explication d'une instance), et est model-agnostic. En revanche, il ne peut que s'appliquer sur des données sous forme de tableaux.

4.3.2 Principe

La génération du voisinage avec LORE repose sur un algorithme génétique, et à partir de voisinage ainsi créé, on en extrait un arbre de décision montrant toutes les règles amenant à chaque instance du voisinage. Ensuite, on génère une explication en extrayant le chemin dans cet arbre nous emmenant à la solution initiale, ainsi que tous les chemins counterfactuals, qui sont les chemins où si l'on change un paramètre, le résultat devient différent. LORE ressemble donc à Anchors dans le sens où l'on obtient un ensemble de règle nous emmenant à notre solution, mais va plus loin en nous générant aussi les couterfactuals, qui sont les mêmes règles dont une propriété est changée et qui emmènent à des résultats différents.

Cette méthode est très difficilement exploitable avec des images (étant donné qu'elle est conçue pour traiter des tableaux), et donc il serait impossible de la faire fonctionner avec des vidéos dans le cadre de ce PRD. Le problème ne vient pas de la données d'entrée (car on peut considérer une image comme un tableau), mais de la façon dont on peut rendre les résultats interprétables.

4.4 Méthode globale reposant sur LORE et ANCHORS

Table 4 – *Caractéristiques de la méthode se reposant sur Lore et Anchors*

Année	2018
Type de problème	Explication du modèle
Explication fournie	Arbre de décision
Modèle de la boite noire	Agnostique
Type de données	Tableaux

4.4.1 Type de problème rencontré

Une autre méthode [5] reposant sur Anchors et LORE a été proposée. Elle permet, en partant d'explications locales, de généraliser les règles de manière à obtenir une explication globale du système. Le type de problème résolu est donc l'explication du modèle.

4.4.2 Principe

L'idée est de partir d'un jeu de données suffisamment large et varié (comme par exemple le jeu de données d'entraînement). Pour chacun des vecteurs de ce jeu de donnée, on utilise Anchors ou LORE de manière à générer un voisinage à ce point, et a partir de ce voisinage, de générer une explication locale qui est une ensemble de règles, exactement comme nous l'avons vu précédemment. Un fois que l'on a fait ceci, on se retrouve avec un ensemble d'explications locales. A ce stade, l'ensemble de ces règles peut servir d'explication globale du modèle, mais elles sont bien trop nombreuses pour être interprétables.

Il s'agit donc de fusionner les règles les plus proches de manière à en diminuer le nombre et obtenir des règles plus générales. Pour cela, une fonction calculant la distance entre deux règles ainsi qu'une fonction permettant de fusionner les règles sont utilisées. On réitère l'explication jusqu'à n'avoir plus qu'une seule règle très générale. On obtient donc un dendrogramme où les

feuilles sont les explications locales. La dernière étape est d'extraire le bon nombre d'explications du dendrogramme. Il faut en effet trouver un compromis entre la généralisation (si on prend des règles trop généralisées, elles ne seront plus représentatives du modèle) et la complexité (prendre un nombre trop élevé de règles augmente la complexité de l'explication globale, ce qui la rend difficile à interpréter).

Bien que cette méthode soit intéressante dans son fonctionnement, elle a été conçue pour des données sous forme de tableau et n'a pas été mise en œuvre pour d'autres types de données comme du texte ou des images. Il est donc difficile d'essayer de la mettre en place sur les vidéos car nous avons toujours le même problème qui est la difficulté de construire des règles interprétables avec des images.

Plus de détails sur l'étude [5] Section 1 (Annexe H).

4.5 Méthode basée sur les auto-encodeurs

Table 5 – *Caractéristiques de la méthode basée sur les auto-encodeurs*

Année	2018
Type de problème	Inspection du modèle
Explication fournie	Saliency Mask
Modèle de la boite noire	Agnostique
Type de données	Images

4.5.1 Type de problème rencontré

Une étude [4] propose une approche différente de l'interprétation. Cette méthode a pour but d'essayer d'établir la quantité d'informations contenue dans les données d'entrée et qui est utilisée par le classificateur. L'objectif est cette fois-ci non pas d'obtenir une explication pour une entrée en particulière (comme une image ou un texte), mais d'avoir un résultat plus général pour le classificateur. On répond alors à un problème de type Inspection du modèle.

4.5.2 Principe

Cette méthode ne s'applique qu'aux modèles utilisant des images, et donc le plus souvent des réseaux de neurones. Elle fonctionne en utilisant un auto-encodeur. Un auto-encodeur est un type de réseau de neurones particuliers. Il est composé de deux parties, une première série de couches de neurones se charge de réduire la dimension de l'espace des données d'entrées et une deuxième partie agrandi cette dimension de manière à revenir sur celle de départ. En général, les auto-encodeurs ont pour objectif d'arriver au résultat le plus proche possible de la donnée d'entrée.

Les auto-encodeurs permettent de réduire la quantité d'information pour ne garder que ce qui est important pour reconstruire l'image de départ. Ils peuvent être utilisés pour faire de la compression ou bien de la réduction du bruit.

Dans notre cas, l'auto-encodeur est utilisé différemment. On fait passer une image à travers l'auto-encodeur, puis on classe l'image obtenue avec le classificateur que l'on veut interpréter. Les résultats de la classification (positifs ou négatifs) sont alors utilisés par rétropropagation sur

l'auto-encodeur pour le paramétrer correctement. Ainsi, on essaye de faire en sorte que l'image modifiée par l'auto-encodeur soit facilement et correctement classifiable par le classificateur.

Une fois que l'on a entraîné notre auto-encodeur avec suffisamment d'images, on peut essayer de l'utiliser pour interpréter le comportement du classificateur. En effet, en faisant passer une image à travers l'auto-encodeur, elle va subir des modifications qui vont impacter positivement le classificateur que l'on veut tester. Certaines données de l'images vont être dégradées car elles ne sont pas importantes pour le classificateur, et d'autres vont être améliorées.

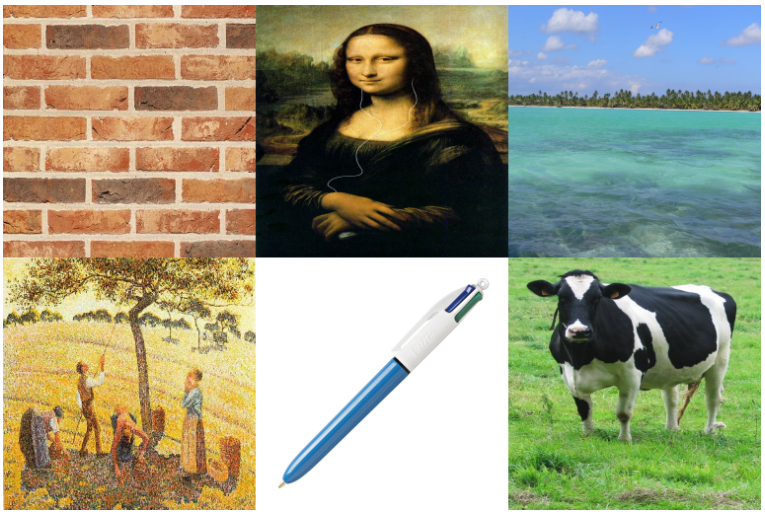
Cette méthode ne va pas modifier chaque image avec un pattern particulier, mais va modifier toutes les images selon des patterns similaires et qui sont caractéristiques du classificateur utilisé (comme on peut le voir Figure 4). Ainsi, on remarquera que toutes les images que l'on fait passer dans l'auto-encodeur vont obtenir un traitement similaire. En revanche, si l'on entraîne l'auto-encodeur sur un autre classificateur, ces motifs vont être différents.

Un autre avantage de cette méthode est qu'elle améliore les résultats du classificateur lorsque l'image en entrée contient du bruit. En effet une caractéristique des auto-encodeurs est de pouvoir réduire le bruit, alors les résultats de classification d'une image bruitée après être passée dans l'auto-encodeur sont améliorés.

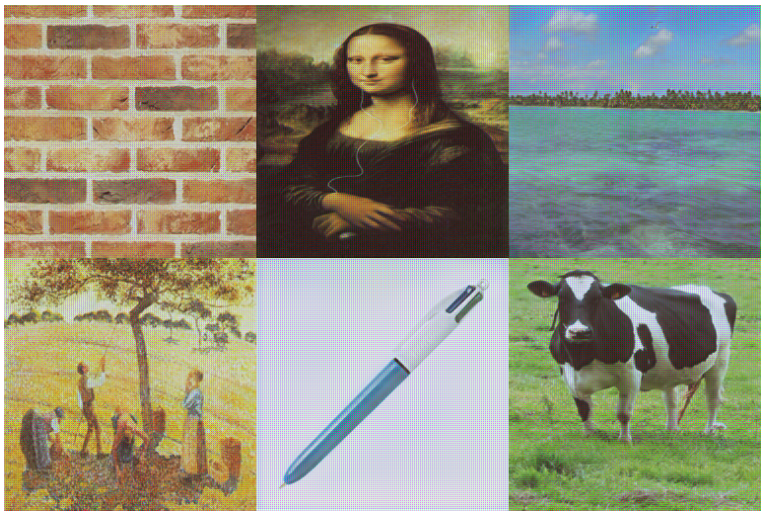
Cette étude permet de classer différents classificateur en fonction de la quantité de signal qu'ils utilisent dans les données d'entrée et de montrer l'existence de patterns propres à chaque classificateur. On peut en déduire aussi des caractéristiques propres à certains réseaux. Par exemple AlexNet (un réseau de neurones convolutifs) est un réseau qui va être très sensible aux structures locales car on peut voir grâce aux auto-encodeurs qu'il n'utilise qu'une part très réduite du signal d'entrée.

Cette étude permet d'avoir une approche différente de l'interprétabilité des IA, mais malheureusement, elle semble difficilement exploitable dans notre cas. En effet elle permettrait de voir quelle part du signal notre réseau de neurone utilise en général, mais ne donne pas d'information sur chacun des cas (i.e. sur chaque vidéo), comme on peut le voir sur les images ci-dessus. Par exemple on peut voir que pour certains réseau de neurones un quadrillage apparaît, pour d'autres cela va être des traitements comme la diminution du contraste. En aucun cas ces traitements ne concernent des images particulières, ils sont appliqués sur toutes. De plus cela nécessiterait d'utiliser un auto-encodeur fonctionnant sur des vidéos ce qui rend la tâche plus compliquée.

Plus de détail sur l'étude [4] Section 3 (Annexe H).



(a) *Images normales*



(b) *Images en sortie de l'auto-encodeur*

Figure 4 – *Comparaison avant/après de 6 images passées séparément à travers l'auto-encodeur*

4 Analyse et conception

Dans ce chapitre nous allons étudier plus en détail comment certains mécanismes sont implémentés dans la méthode LIME pour les images, et nous allons voir comment il serait possible de les adapter pour traiter des séquences.

1 Le classificateur

Nous allons voir dans cette partie comment il sera possible d'utiliser le classificateur qui nous a été fourni par le laboratoire.

Dans le cadre de ce PRD, il n'est pas nécessaire de rentrer en détails dans le fonctionnement du réseau. En effet, lors de ce PRD nous cherchons à implémenter un méthode d'interprétation qui peut être utilisable sur n'importe quel type de modèle sans se soucier de comment il fonctionne, il nous suffit juste de savoir l'utiliser. Nous allons cependant voir brièvement pourquoi on utilise ce réseau et quelles sont ses particularités.

1.0.1 Pseudo-3D Residual Networks (P3D ResNet)

En temps normal, lorsque l'on veut classifier des images, on utilise des réseaux de neurones convolutifs (CNN). Ces types de réseaux sont particulièrement efficaces dans leur domaines, mais sont conçus pour traiter des données 2D (des images). Il n'est pas trivial d'utiliser ces réseaux lorsque l'on veut classifier des vidéos, qui sont des données spatio-temporelles. On peut toujours fabriquer un CNN fonctionnant avec des données 3D, cela existe, comme les réseaux C3D, mais ils ont comme défauts de nécessiter énormément de temps de calcul pour leur entraînement et d'être très lourds. Un réseau C3D de 11 couches pèse environ 321 Mo alors qu'un CNN 2D de 152 couches ne pèse que 235Mo par exemple.

Le réseau P3D est un réseau conçu pour des données spatio-temporelles (comme des vidéos). Dans ce réseau, chaque bloc est une combinaison d'une couche convolutive 1x3x3 (comme un CNN) et d'une couche de 3x1x1 convolutions (qui se charge de la dimension temporelle) en parallèle ou en cascade.

Cette architecture permet d'avoir un réseau à la fois plus léger et plus performant que les autres méthodes existantes pour des vidéos. C'est un réseau de ce type que nous allons essayer d'expliquer.

1.0.2 Utilisation

En pratique pour utiliser ce type de réseau, il faut prendre en entrée une vidéo. Les données fournies par le laboratoire sont stockées sous une forme particulière. Chaque dossier représentant une vidéo est composé d'un certain nombre de dossiers numérotés et chacun de ces dossiers contient 30 images numérotées au format PNG. Le modèle prenant en entrée une vidéo, il faut assembler tous ces fichiers en un seul objet. Une classe appelée FluideDataset fournie par le laboratoire permet de faire toutes les transformations nécessaires.

Une fois les données chargées, il faut charger le modèle qui est lui aussi fourni dans une autre classe R2Plus1DClassifer. Il n'est pas nécessaire d'entraîner le modèle car le laboratoire nous a aussi fourni un fichier correspondant au modèle entraîné. Il suffit simplement de charger ce fichier avec torch et de l'intégrer au modèle.

Une fois cette étape effectuée, le modèle est directement utilisable en lui passant en entrée les bonnes données (que nous avons extraites précédemment).

2 LIME

2.1 La segmentation

La première étape dans l'implémentation de LIME est la segmentation des données d'entrée pour pouvoir par la suite générer le voisinage.

Pour obtenir de meilleurs résultats, la segmentation ne doit pas être aléatoire ou linéaire (même s'il serait possible d'envisager cela pour commencer). En effet, il est plus intéressant d'essayer de garder ensemble les zones qui se ressemblent.

Nous allons commencer par voir comment la segmentation est implémentée actuellement dans la méthode pour des images, nous allons ensuite proposer quelques pistes pour faire la même chose avec une vidéo.

Pour les images

Pour segmenter les images, LIME propose un wrapper qui va permettre d'utiliser au choix un algorithme de segmentation parmi trois (cf [WWW1]). Les algorithmes de segmentation sont fournis au travers de la bibliothèque skimage.segmentation. Les trois algorithmes sont les suivants :

— Felzenszwalb : Algorithme qui ne prend qu'un seul paramètre qui influence la taille des segments. La taille et le nombre des segments peuvent varier énormément en fonction du contraste local de l'image.
— Quickshift : Algorithme plus récent appartenant à la famille des algorithmes de recherche de mode local. L'avantage de cet algorithme est qu'il produit une segmentation sur plusieurs échelles en même temps. Il a deux paramètres principaux : Sigma contrôle la taille de l'approximation de la densité locale, max_dist choisis le niveau dans la segmentation hiérarchique. Il y a aussi le paramètre ratio permettant de régler le ratio entre la distance dans l'espace des couleurs et la distance spatiale.
— Slic : Algorithme assez similaire au Quickshift mais dont la méthode de clustering est plus simple. Le paramètre Compactness défini le ratio entre similarité de couleur et proximité (comme pour Quickshift) et le paramètre n_segments choisis le nombre de centres sur lesquels se basent les moyennes.

L'implémentation de base de la méthode pour une image utilise l'algorithme Quickshift. Cette implémentation retourne une image représentant les segments. Cette image est composées de tous les segments, chacun avec une couleur qui lui est propre (voir Figure 1).

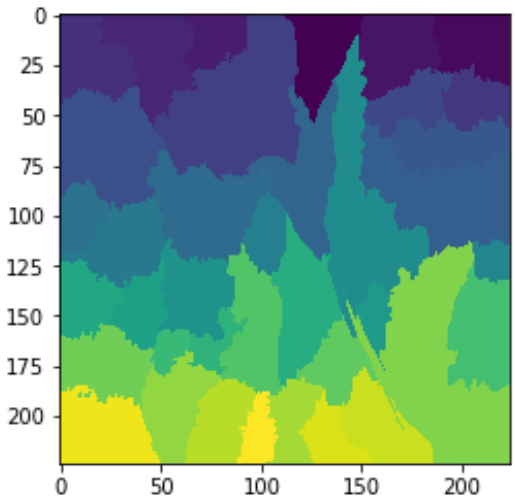


Figure 1 – *Segments renvoyés par Quickshift*

Il est possible de jouer sur les paramètres dans le code pour avoir des segments plus petits et donc plus nombreux, ce qui permet d'avoir des résultats plus précis dans notre méthode à condition que l'on génère un nombre de voisins plus grand (ce qui nécessite un temps de calcul plus long).

Pour les vidéos

Nous avons commencé à réfléchir à plusieurs façons d'effectuer la segmentation d'une vidéo, cependant une analyse plus poussée sera à effectuer au S10. Il faudra soit trouver un algorithme de segmentation existant pour une vidéo, soit adapter une méthode s'appliquant aux images. La difficulté de segmenter une vidéo est qu'il faudrait la découper aussi bien spatialement que temporellement.

Les pistes pour effectuer une segmentation sont les suivantes :

Une première piste sera de remplacer la segmentation colorimétrique par une segmentation temporelle. Au lieu d'avoir une image RGB, notre vidéo serait une ensemble d'images en niveau de gris. La segmentation colorimétrique se faisant en 3 dimensions (hauteur, largeur, couleurs), alors il serait assez simple de remplacer les couleurs par la dimension temporelle et faire la segmentation temporelle selon les dimensions (hauteur, largeur, temps).

Une autre piste plus simple serait simplement de diviser les vidéos en un ensemble d'un petit nombre de frames et de classer chacune de ces vidéos. Ici, on ne fait qu'une segmentation temporelle. Le résultat de Lime serait que l'on aurait lors de l'affichage une mise en valeur des moments de la vidéo les plus importants pour la prise de décision mais nous ne saurions pas quelles sont les zones de l'images qui ont le plus été utilisées par le classificateur.

Il existe aussi une méthode pour convertir une vidéo en une seule image. Cela consiste à créer une image dont les couleurs représentent la direction des pixels et l'intensité représente la magnitude du déplacement. On pourrait alors convertir nos morceaux de vidéos de cette manière là et appliquer une segmentation sur cette image. Il faudrait ensuite un algorithme qui nous permette de retranscrire nos images "compressées" en vidéos pour fournir les voisins ainsi créés au classificateur.

Appliquer la segmentation aux vidéos va nécessiter une nouvelle phase de recherche et une phase d'expérimentation. La façon dont la segmentation va être effectuée aura aussi un impact

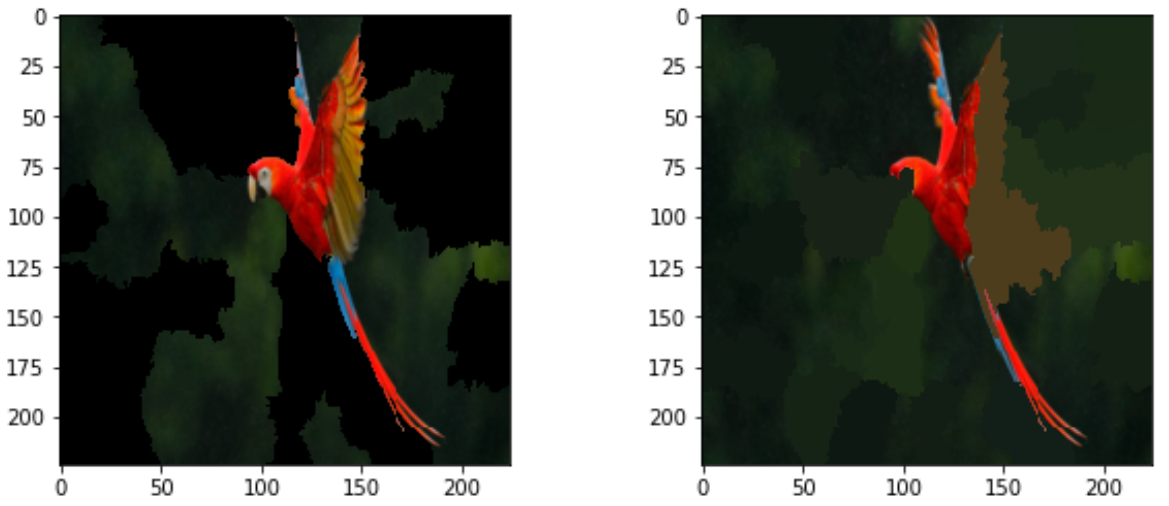
important sur la suite du développement. Plus la segmentation sera proche de la façon dont elle est faite pour les images et plus il sera simple de développer la suite du programme car il n'y aura pas beaucoup de modifications à faire. En revanche, si la segmentation est très différente et si les segments ne sont pas de la même nature que ceux utilisés par les images, il faudra procéder à des modifications majeures du reste du programme.

2.2 L'Explainer

L'Explainer est l'objet qui va nous permettre de générer une explication. Au travers de sa méthode explain_instance(), il s'occupe de générer les voisins et de les passer dans le classificateur de manière à générer l'Explanation.

Voici comment fonctionne cette fonction pour des images. Le principe devrait rester le même pour les vidéos.

Tout d'abord on utilise la fonction de segmentation pour récupérer les segments. Ces segments nous serviront plus tard pour générer les voisins. Chaque segments représentant une zone de l'image initiale, un voisin sera composé de segments où l'on a l'image initiale et de segments "vides". Il est possible de modifier la teneur des segments "vides" dans le code. Soit on laisse ces segments en noir (voir Figure 2a), soit on les remplace par la couleur moyenne des pixels de l'image initiale se trouvant à ce niveau (voir Figure 2b).



(a) *en remplaçant les "trous" par des pixels noirs*

(b) *en remplaçant les "trous" par la couleur moyenne des pixels cachés*

Figure 2 – *Exemple de voisins générés*

Pour faire cela, nous avons besoin de créer une image "truquée" ("fudged") qui va représenter tous les segments "vides". Si l'on est dans le cas où les segments vides sont noirs, alors l'image va être complètement noire, sinon, l'image va être similaire à celle représentant les segments, mais leur couleur sera la couleur moyenne des pixels correspondants à l'image initiale (voir Figure 3).

La partie suivante est la génération des voisins. Pour cela on génère un tableau 2D de la taille du nombre de segments x le nombre de voisins que l'on veut créer. On rempli ensuite aléatoirement ce tableau de 0 et de 1. Il n'y a que la première ligne de ce tableau qui ne sera composée que de 1 (pour représenter l'image initiale). On obtient alors un tableau data qui nous permet de savoir pour chaque voisin quels sont les segments que l'on veut afficher (correspondants aux 1 dans le tableau), et ceux que l'on veut cacher (correspondant aux 0).

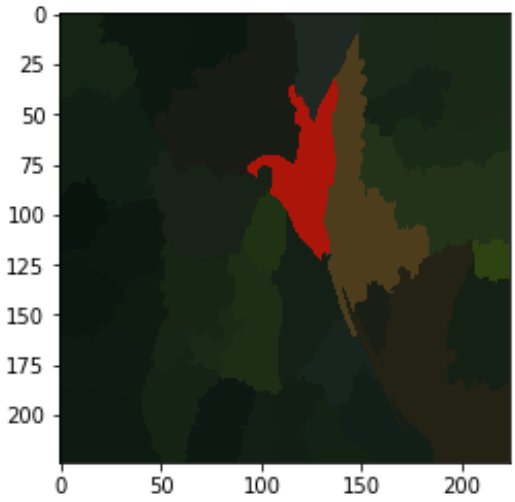


Figure 3 – *Image fudged avec la moyenne des couleurs*

Il suffit ensuite pour chacune des lignes de ce tableau data (chaque ligne correspond à un voisin) de créer une copie de l'image initiale et de remplacer toutes les parties où les segments représentés dans le tableau sont à 0 par les segments correspondant de l'image "truquée". On obtient alors un voisin.

On stocke ensuite ce voisin dans un tableau, et lorsque ce tableau est suffisamment rempli (un paramètre batch_size de la fonction permet de le choisir), on fait passer les voisins dans le classificateur pour en extraire les prédictions.

On vide alors le tableau puis on recommence le processus. Il est plus intéressant de limiter le nombre de voisins traités en même temps grâce à ce paramètre batch_size plutôt que de générer tous les voisins en amont car cela permet d'éviter de surcharger la mémoire vive.

Un fois que tous les voisins sont classifiés, on retourne les prédictions ainsi que le tableau data des segments par voisin.

On calcule les distances entre l'image initiale et chacun des voisins. Pour cela on calcule les distances entre les lignes du tableau data et sa première lignes (celle correspondant à l'image initiale).

Une fois que l'on a récupéré toutes ces données, à savoir le tableau data, les labels et les distances, on peut générer notre objet Explanation. Notre objet Explanation nous permet de regrouper toutes les informations pour fournir une explication d'un résultat. Il peut expliquer plusieurs labels en même temps. Par exemple, si l'on choisi dans les paramètres l'explication des deux premiers labels, alors notre Explanation contiendra toutes les informations nécessaires pour expliquer le résultat pour ces deux labels.

Pour faire cela, on initialise notre nouvel objet Explanation, et pour chacun des labels pour lesquels on veut l'explication, on appelle la fonction LIME explain_instance_with_data(). Cette fonction prend en paramètre le tableau data représentant les voisins, les prédictions pour ces voisins, le label pour lequel on veut l'explication, le nombre de features de l'Explanation (qui est le nombre de segments que l'on veut surligner). Cette fonction utilise une régression linéaire pour nous renvoyer tous les éléments permettant de compléter notre objet Explanation.

2.3 Création de l'image finale

Une fois que l'on a notre objet Explanation, il nous faut le convertir en une image avec un masque qui va nous permettre d'interpréter le résultat. L'image va être recouverte d'un masque

vert pour tous les segments qui vont contribuer de manière positive au résultat, et rouge pour ceux qui y contribuent de manière négative. De plus on ne veut afficher qu'un certain nombre paramétrable de segments surlignés (par exemple les 3 plus importants). Pour récupérer l'image et le masque, on utilise la fonction get_image_and_mask() de l'objet Explanation. Cette fonction prend en paramètre le label que l'on veut expliquer, le nombre de segments que l'on veut surligner (voir Figure 4), un boolean permettant de choisir si l'on veut juste afficher les segments ayant un impact positif (et non ceux ayant un impact négatif), et un autre booléen permettant de cacher les parties de l'image n'affectant pas le résultat (voir Figure 5).

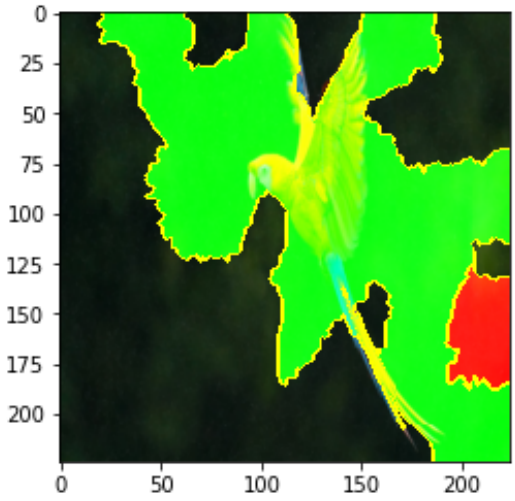


Figure 4 – *Image du résultat avec 10 segments surlignés*

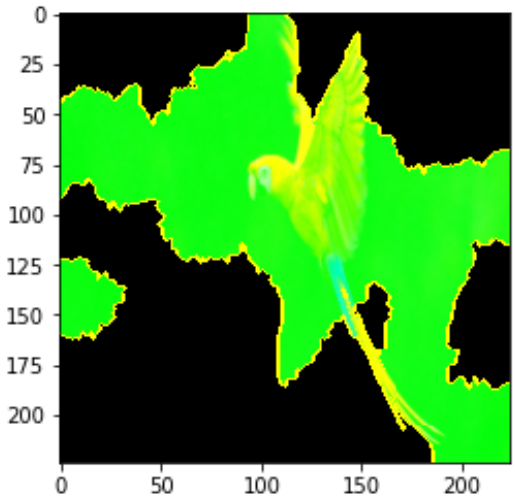


Figure 5 – *Image du résultat avec la fonction Hide activée*

2.4 Mise en oeuvre pour des vidéos

Pour mettre en oeuvre toutes ces fonctions de Lime avec des vidéos, une partie Analyse plus poussée sera nécessaire au semestre 10. Les parties Explainer et création de la vidéo finale pourront sans doute se faire de manière assez triviale, mais la partie segmentation sera sûrement plus compliquée. Il pourrait être intéressant de faire une segmentation linéaire ou aléatoire pour commencer afin de pouvoir faire l'Explainer et la création du résultat en premier, pour ensuite revenir à la partie segmentation.

5 Mise en oeuvre

1 Outils et librairies utilisés

Alors qu'il était initialement prévu de développer sur une machine locale, avec l'IDE Spyder, un problème s'est rapidement posé. Le classificateur utilisé a besoin d'une grande quantité de mémoire graphique pour fonctionner (plus de 6 Go). Nous avons donc changé d'environnement pour développer sur Google Colaboratory.

1.1 Outils

Google Colaboratory

Google Colaboratory (abrégé en Colab) est un environnement d'exécution à distance permettant d'exécuter du code en Python. Dans notre cas, cet environnement nous intéresse car il est possible d'utiliser des cartes graphiques disposant d'une grande quantité de mémoire vidéo (12 Go). Ce type d'environnement est idéal pour faire du Deep Learning par exemple.

Avantages

Google Colab a d'autres avantages, comme le fait d'être disponible à distance. Il est possible d'accéder à nos travaux et d'exécuter le code depuis n'importe quelle machine, sans ce soucier des contraintes matérielles ou logicielles. La seule condition pour pouvoir y accéder et d'avoir un navigateur et une connexion internet. Un autre avantage de Google Colab est qu'il permet de faire des appels systèmes facilement car on peut les intégrer directement à l'intérieur de notre code Python, et il est possible d'utiliser les variables Python dans les appels systèmes. Cela se révélera très utile pour la partie segmentation.

Inconvénients

Google Colab se présente sous une forme de Jupyter Notebook. Il s'agit d'un ensemble de cellules de code et de texte, qui peuvent être exécutées indépendamment. Ce format est très pratique pour présenter des exemple de code et faire des tutoriels, mais il n'est pas vraiment adapté au développement d'application. Il est par exemple compliqué de faire plusieurs fichiers et de naviguer entre eux. Si l'on veut séparer nos différentes classes, le mieux reste de les écrire dans des cellules séparées. Il faut cependant exécuter toutes les cellules une par une si l'on veut

exécuter notre code. De plus Google Colab ne dispose pas de la plupart des fonctionnalités présente sur d'autres IDE, comme l'indication d'erreurs ou de problème de formatage du code. Il est aussi difficile d'utiliser des outils de qualimétrie qui sont plus adaptés pour des fichiers Python classiques. Enfin, comme tout le code est disponible dans un même fichier, on passe une grande partie du temps de développement à naviguer dans le fichier là où l'on veut modifier du code.

Stockage

Comme nous travaillons sur une machine virtuelle de Google, l'environnement d'exécution est réinitialisé dès qu'on le quitte. Ainsi, même si il est possible de stocker temporairement des fichiers sur la machine de Google Colab, il faut un autre moyen de stockage si l'on veut garder des fichiers à long terme, comme par exemple les résultats de nos expériences. Pour ce faire, soit on peut les télécharger sur notre machine locale, soit on peut utiliser Google Drive. Je ne recommande pas la première méthode car elle est peu pratique (tout doit se faire manuellement) et il faut réuploader les fichiers si l'on veut les réutiliser. La meilleure méthode consiste donc à utiliser le cloud de Google. Pour cela, il suffit de disposer d'un compte Google Drive, et il est possible de le lier directement avec Google Colaboratory (voir Annexe E). Le Drive est ensuite directement considéré comme un espace disque de la machine virtuelle de Google Colab, ce qui permet de l'utiliser comme espace de stockage.

1.2 Librairies

De nombreuses librairies Python sont utilisées. Les plus importantes sont Numpy pour la gestion des images en tant que tableau, Torch et Torchvision pour le chargement et l'utilisation des classificateurs, ainsi que matplotlib pour la création des images finales. Toutes ces librairies sont déjà installées dans l'environnement de Google Colab.

Une autre librairie importante est Lime, qui est le composant essentiel de notre méthode. Cette librairie contient des fonctions qui permettent de faire fonctionner Lime de manière agnostique, c'est-à-dire que l'on peut les utiliser pour n'importe quel type de données ou classificateur, il suffit de leur passer les bonnes entrées. Il s'agit essentiellement des fonctions qui permettent de calculer les poids des différents segments en fonction des résultats de la classification des voisins. Cette librairie n'est pas installée de base dans Google Colab, un appel à l'installateur de librairie PyPi est donc effectué pour l'installer (l'utilisateur n'a rien a faire, cette commande est effectuée automatiquement en lançant le code).

Enfin, la dernière librairie importante est celle qui nous sert à faire la segmentation des vidéos. il s'agit d'une implémentation en C++ d'un algorithme de segmentation vidéo hiérarchique et basé sur des graphes (voir [1]). Le Github de l'implémentation est disponible ici [WWW2]. Comme cette implémentation est en C++, il a fallu faire des appels systèmes pour l'installer et l'utiliser, mais ce n'est pas un problème dans Google Colab car il est possible d'inclure les appels systèmes directement dans le code Python. Tout comme pour Lime, l'installation est donc automatique et l'utilisateur n'a rien d'autre à faire que d'exécuter le code.

2 Implémentation

2.1 Structure du programme

Le programme se comporte en 5 grandes parties, chacune située dans différentes cellules du Notebook :

— Classifier : Contient la classe nécessaire au chargement et à l'utilisation du modèle du classificateur,
— Preprocessing : Contient les méthodes préparant les données pour la classification et la segmentation,
— Segmentation : Contient les méthodes et appels systèmes permettant de segmenter les vidéos,
— LIME Method : Contient les classes permettant d'utiliser la méthode LIME pour interpréter une décision,
— Execution : Contient les méthodes pour exécuter tout le programme.

2.2 Fonctionnement

2.2.1 Fonctionnement théorique

Voici une description du fonctionnement simplifié du programme. On a en entrée notre vidéo et notre classificateur.

On va d'abord segmenter notre vidéos (découpage spatial et temporel), ce qui nous donne des segments.

Ensuite on va générer les voisins de notre vidéo. Un voisin est constitué de notre vidéo originale à laquelle on a caché certains segments de manière aléatoire. Les segments cachés sont des parties de la vidéo qui vont être noires. Ensuite, on va passer notre vidéo originale et tous les voisins au classificateur qui va alors établir des prédictions. Dans la pratique, ces deux dernières étapes ne se font pas l'une après l'autre, mais on va plutôt générer un certain nombre de voisins avant de les classifier et on recommence jusqu'à obtenir le bon nombre de voisins (cela permet de ne pas surcharger la mémoire).

Une fois que l'on a nos prédictions, la méthodes Lime va les utiliser calculer les poids des différents segments.

Enfin, il ne reste plus qu'à créer la vidéo de résultat et surligner de la bonne couleur les segments les plus importants.

2.2.2 Fonctionnement pratique et description des classes

Dans la pratique, ces différents étapes sont séparées dans des classes, et elle s'exécutent au sein de la classe LIME qui se charge d'appeler tous les éléments dans le bon ordre.

La classe LIME

A l'initialisation de cette classe, elle va créer le classificateur (objet Classifier) et charger les poids du modèle. Au lancement de sa fonction explain, elle va d'abord créer l'objet Segmentation, segmenter la vidéo et stocker les segments. Ensuite, la fonction va créer l'objet LimeVideoExplainer, qui va se charger de générer les voisins à l'aide des segments et les passer dans le classificateur. Le LimeVideoExplainer va ainsi générer un objet VideoExplanation contenant les données de résultat sur les poids des segments. Enfin, la fonction va utiliser ces résultats pour générer la vidéo de sortie.

La classe Classifier

La classe Classifier représente les objets qui vont s'occuper de contenir le modèle du classificateur, les poids du modèle chargés à partir d'un fichier, ainsi que la fonction permettant d'effectuer une prédiction.

La classe Segmentation

La classe segmentation représente un objet capable, à partir d'un dossier contenant les images ordonnées d'une vidéo, de les découper en morceaux appelés segments. Ce découpage est spatial et temporel, ce qui veut dire que chaque image est découpée en plusieurs segments, et qu'un segment peut se retrouver sur deux (ou plus) images consécutives.

La classe Segmentation fait appel à un programme en C++ pour effectuer le découpage. Ce programme utilise un algorithme de segmentation vidéo hiérarchique et basé sur des graphes (voir [1]), ce qui veut dire qu'il va générer plusieurs segmentation en fonction de la hiérarchie. Par exemple, si on règle ses paramètres de manière à avoir 50 hiérarchies de segmentation, alors, on va obtenir 50 segmentations différentes (une pour chaque niveau de hiérarchie). Les premières segmentations (niveau de hiérarchie faible) contiendront des segments petits mais nombreux, alors que les dernières segmentations (niveau de hiérarchie élevé) contiendront peu de segments mais ils seront plus gros. Nous stockons temporairement toutes ces segmentations, et la classe Segmentation contient une fonction get_segment() qui permet de récupérer la segmentation au niveau hiérarchique voulu. Cela permet de ne pas avoir a segmenter la vidéo plusieurs fois lors d'une même exécution et de gagner du temps, car la segmentation est assez longue.

La classe LimeVideoExplainer

C'est la classe qui s'occuper d'exécuter la méthode Lime. A partir du classificateur, de la vidéo en entrée et des segments, cette classe va générer tous les voisins et les classifier. Les résultats des prédictions sont alors stockés et prêt à être envoyés à une méthode de la librairie Lime qui s'appelle explain_instance_with_data(). A ce stade là, nos données sont agnostiques, c'est-à-dire qu'elles sont les mêmes quelque soit le type de donnée en entrée et le classificateur. En effet, tout ce que la méthode a besoin comme informations, ce sont les prédictions en fonction de chaque voisin, et quels segments chaque voisin possède. A partir de ces données (et de quelques unes supplémentaires qui sont calculées à la volée), cette fonction va nous donner les résultats de chaque segment. Toutes ces données sont alors stockées dans un objet VideoExplanation qui va nous servir à renvoyer le résultat.

La classe VideoExplanation

Cette classe permet de stocker les résultats de la méthode Lime. Elle contient une méthode get_images_and_masks() qui permet de récupérer les images de la vidéo traitée, avec les segments surlignés. Cette méthode contient des paramètres qui nous permettent de choisir le nombre de segments surlignés qui l'ont veut afficher, si l'on ne veut afficher que des segments positifs ou négatifs, et si l'on veut afficher que les segments qui ont au moins un poids minimal.

2.3 Utilisation pratique

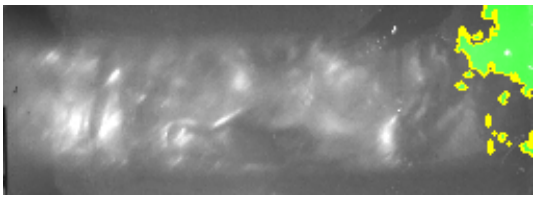
Pour une description détaillée de l'utilisation du programme, se référer au guide d'utilisation Annexe F.

3 Tests et Analyse des résultats

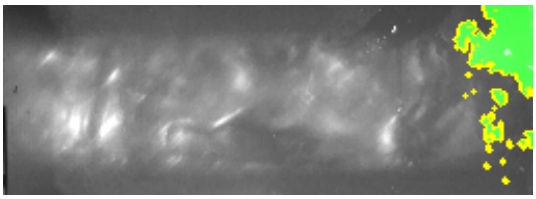
3.1 Test avec le premier classificateur

Le programme a d'abord été implémenté pour fonctionner avec le classificateur permettant d'estimer la vitesse de rotation de liquides dans un moteur.

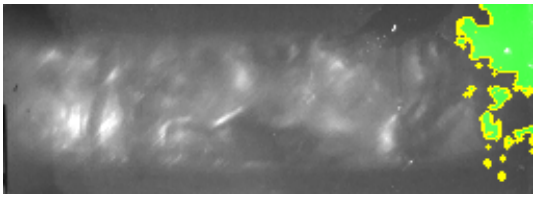
Voici un extrait de 4 images au milieu d'une vidéo du résultat que l'on a obtenue avec notre version de Lime (Figure 1). On n'a surligné que le premier segment.



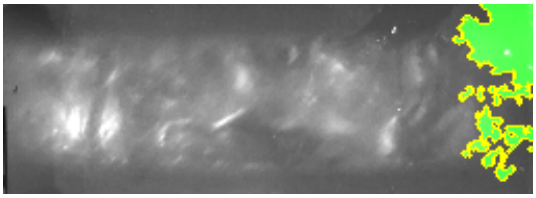
(a) *Image 16*



(b) *Image 17*



(c) *Image 18*



(d) *Image 19*

Figure 1 – *Extrait d'une vidéo de résultat d'un liquide tournant dans un moteur*

On peut voir sur ces images que c'est la partie droite qui a l'air importante pour la prise de décision. On obtient un résultat similaire sur beaucoup d'autres vidéos de liquide, avec la partie droite surlignée. Malheureusement, il est difficile d'exploiter ce résultat, car la vidéo en elle même n'est pas explicite. On ne peut pas dire par exemple si il y a un problème avec l'algorithme, les paramètres, la vidéo ou même le classificateur.

De plus il est difficile de savoir si ce segment est significativement plus important que les autres, ou non. Pour expliciter un peu plus nos résultats, nous avons du procéder à des petites améliorations de l'algorithme, et pour vérifier si le programme fonctionne correctement, nous allons le tester sur un autre classificateur avec lequel il est plus facile de distinguer les objets segmentés.

3.2 Amélioration de l'affichage de LIME

Fichier des poids

La première amélioration a été d'ajouter un fichier contenant les numéros de tous les segments ainsi que leur poids, le tout trié selon l'ordre décroissant des poids. Les poids des segments permettent de se rendre compte si ils ont beaucoup impacté la prise de décision (valeur absolue du poids élevée) ou s'ils ont peu impacté dans la prise de décision (valeur absolue du poids faible). Grâce à ce fichier, on peut aussi voir si tous les segments ont des poids similaires, ou si un certain nombre d'entre eux ressortent particulièrement, ce qui peut nous permettre de mieux ajuster les paramètres pour un futur essai, notamment le nombre de segments affichés sur la vidéo de résultat.

Couleurs et numéros sur les segments

Une autre amélioration apportées est la distinction des différents segments sur la vidéo de résultat par leur couleur et l'ajout de leur numéro sur la vidéo. Les segments affichés ayant des poids positifs plus grands seront plus verts que les segments dont les poids positifs sont plus petits, et les segments avec des poids négatifs plus petits sont plus rouges que ceux dont les poids négatifs sont plus proches de zéro. Les segments avec des poids plus importants sont donc plus "flashy". Cependant, comme la couleur ne suffit pas, le numéro du segment est directement

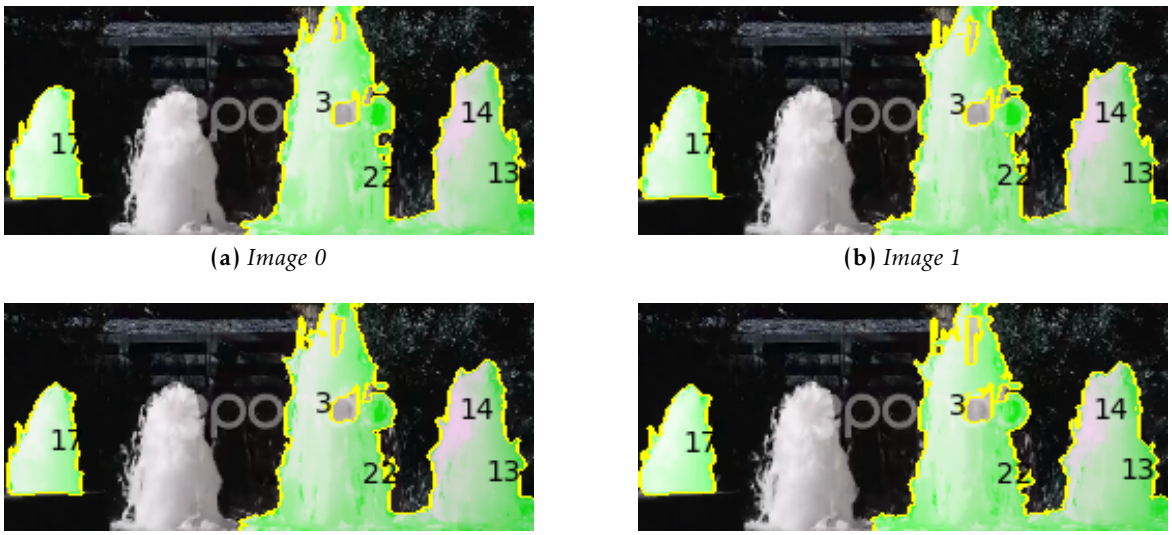
affiché sur l'image, à l'endroit où le segment est placé. Cela permet de le retrouver dans le fichier des poids et donc de retrouver son poids et son importance dans la prise de décision.

3.3 Implémentation d'un deuxième classificateur

Nous avons donc ajouté le code nécessaire pour utiliser un deuxième classificateur (pour connaître la démarche suivie, voir Section 3 (Annexe F)). Ce deuxième classificateur est capable de reconnaître des éléments présents sur une vidéo parmi ceux qu'il a appris. Ce sont principalement des éléments de paysages, de nature, ou d'objets en mouvement (fleurs, nuages, trafic routier, éoliennes etc...). Il est plus facile de vérifier si notre algorithme fonctionne sur un classificateur de ce type, car il suffit surtout de vérifier que l'objet prédit par le classificateur est surligné sur la vidéo de résultat de notre méthode.

Résultats

Nous avons fait les tests avec des vidéos de geysers, et la classe trouvée par le classificateur était effectivement "geyser". En toute logique, notre méthode devrait donc nous ressortir la vidéo avec les geysers de la vidéo surlignés en vert. C'est effectivement ce qu'il se passe lorsque l'on observe les résultats (voir Figure 2).



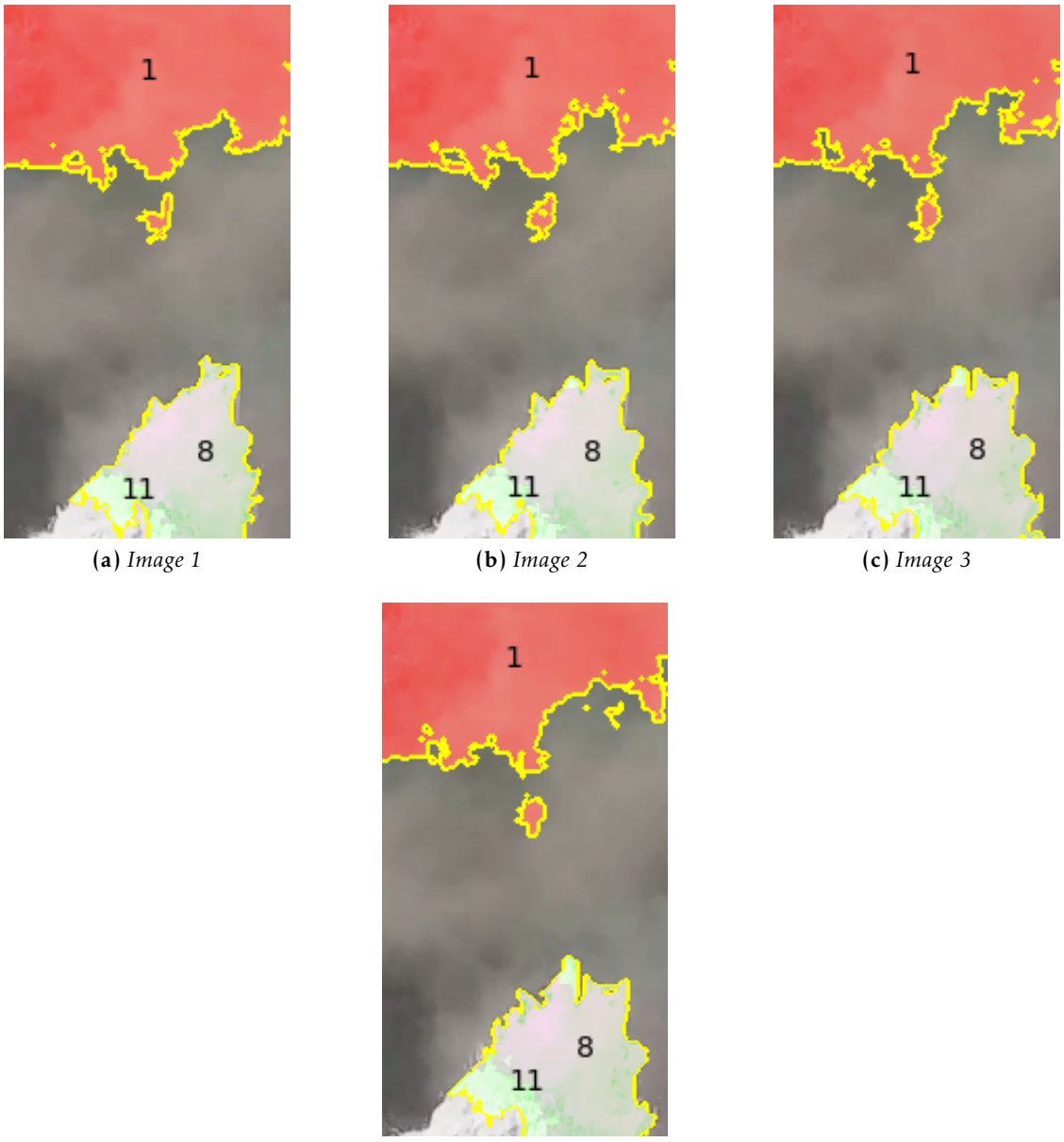
(a) *Image 0* **(b)** *Image 1*

(c) *Image 2* **(d)** *Image 3*

Figure 2 – *Extrait d'une vidéo de résultat de geysers*

Ces résultats nous confirment que notre méthode fonctionne correctement, on a bien les résultats espérés, et on obtient des résultats similaires en relançant la méthode avec différents paramètres. On peut de plus voir sur le fichier des poids que les segments surlignés ont des poids assez significatifs (supérieurs à 0.10 alors que les autres segments ont des poids inférieurs à 0.05). Sur la deuxième vidéo de geyser (Figure 3), on constate un grosse partie en rouge en haut de l'image. Dans le fichier des poids, c'est le deuxième segment le plus important (après le numéro 11 en vert sur le geyser). Cela peut vouloir dire que cette partie en haut de l'image a tendance à fortement influencer notre classificateur vers une autre décision.

Nous obtenons de bons résultats sur ces vidéos car les geysers sont faciles à distinguer et à segmenter, mais la méthode Lime que nous avons implémentée comporte aussi des limites.



(a) *Image 1* **(b)** *Image 2* **(c)** *Image 3*

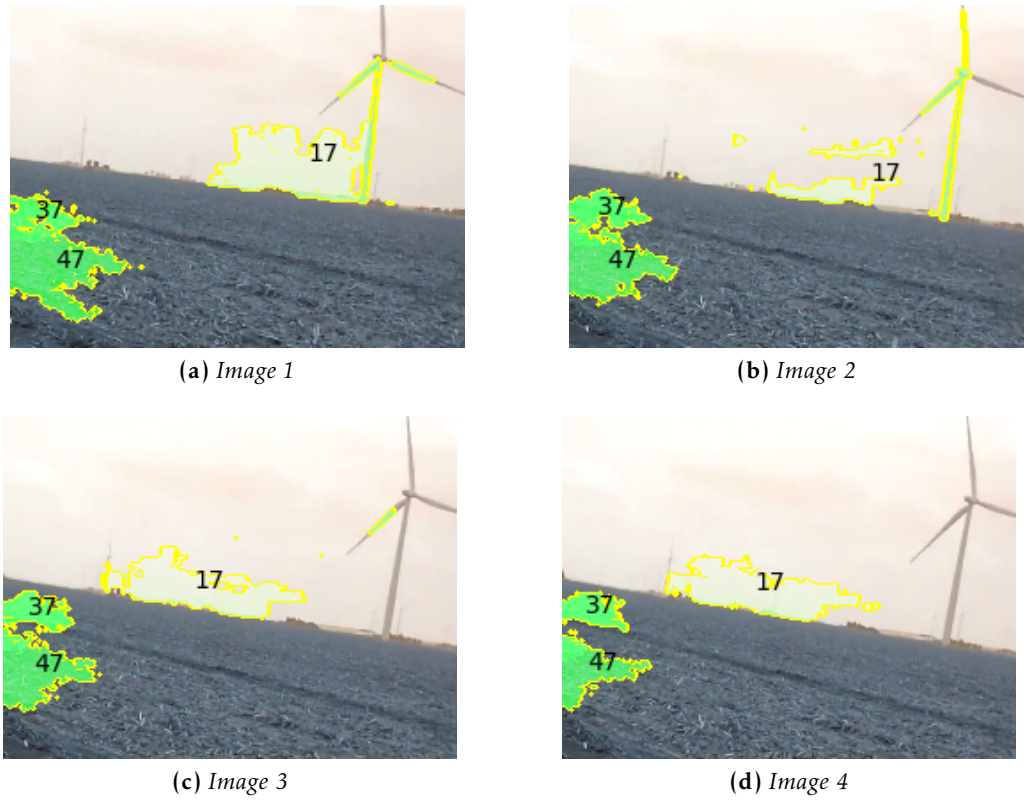
(d) *Image 4*

Figure 3 – *Extrait d'une vidéo de résultat d'un autre geyser*

4 Limites de la méthodes

Nous pouvons nous apercevoir des limites de cette méthode au travers de la vidéo Figure 4.

Cette vidéo représente des éoliennes, et c'est bien cette classe qui nous est donnée par le classificateur. En revanche il est plus difficile de voir la logique derrière l'attribution des segments colorés par Lime. En effet, on pourrait s'attendre à ce que ce soit uniquement les éoliennes qui soit surlignés, et même si c'est partiellement le cas, on a aussi d'autres éléments du paysage qui le sont, comme un morceau du champ au premier plan. Plus troublant, on observe, en regardant les poids, que le morceau numéro 37 (le champ) à un poids légèrement plus élevé que celui sur l'éolienne (le numéro 17). Les poids sont respectivement de 0.007 et de 0.006. De plus ces poids sont très faibles (cela peut s'expliquer avec la segmentation qui a généré de nombreux et petits segments), et il n'y a pas vraiment de segments avec des poids beaucoup plus élevés que les autres.



(a) *Image 1* **(b)** *Image 2*

(c) *Image 3* **(d)** *Image 4*

Figure 4 – *Extrait d'une vidéo de résultat d'un champ d'éoliennes*

Impact de la segmentation

Nous pensons que ces résultats moins probants sont dû à la segmentation. En effet, sur cette vidéo, l'algorithme de segmentation a du mal à séparer les objets représentés correctement. Cela est probablement dû à la taille des éléments importants (les éoliennes sont très fines et petites sur la vidéo, comparé aux geysers de l'exemple précédent). De plus les segments ne restent pas "fixés" aux objets qu'ils découpent dans les premières images. Par exemple, le segments 17 est sur l'éolienne au début de la vidéo, mais se déplace ensuite sur le ciel, en n'étant plus du tout sur l'éolienne, diminuant donc son importance dans le calcul des poids. Cela veut aussi dire que plusieurs segments différents peuvent se retrouver sur les mêmes objets selon le déroulement de la vidéo, ce qui contribue à brouiller les pistes pour la méthode LIME.

Association de segments

Un autre phénomène pouvant être handicapant pour LIME, c'est que la méthode n'utilise pas les associations de segments. C'est-à-dire qu'elle ne peut pas comprendre que c'est l'association de deux segments qui peut permettre au classificateur de trouver une bonne solution. Par exemple, supposons que l'on ait une vidéo d'une éolienne, mais qu'elle soit coupée en plusieurs segments de manière à ce que si l'on prenne chaque segment seul, cela ne ressemble pas à une éolienne. Alors avec ce genre de vidéo, LIME aura du mal à trouver de bons résultats, car c'est l'assemblage des segments recouvrant l'éolienne qui permet de prendre la bonne décision. C'est notamment pour cette raison qu'avait été conçue la méthode Anchors (voir [7]), qui prend en compte ce phénomène. A noter que ce problème dépend aussi de la segmentation. En effet si on a dès le départ une segmentation "parfaite", qui arrive à ne faire qu'un seul segment pour tout l'objet (ici un segment pour l'éolienne), alors ce problème peut-être résolu.

6 Bilan et conclusion

1 Voies d'amélioration

Il y a plusieurs axes pour améliorer la méthode actuellement implémentée. La première étape serait de perfectionner l'algorithme de segmentation pour qu'il soit plus fiable, qu'il sépare mieux les différents objets de la vidéo et qu'il arrive mieux à les "suivre". La seconde voie d'amélioration serait d'essayer d'implémenter la méthode Anchors, qui est elle-même une amélioration de Lime.

2 Point sur Fait et Reste à faire / retards

2.1 Semestre 9

Ce semestre 9 aura été consacré à faire la partie recherche, la conception et la rédaction de documents. Voici les étapes qui ont été réalisées.

Recherche

Pour la partie recherche, nous avons étudié plusieurs études ([5], [3], [4]). Cela nous a permit de recueillir le maximum d'informations sur ce qu'est l'interprétabilité des IA et de choisir une méthode d'interprétation.

Conception

La partie conception a été consacrée à la rédaction de documents tel que le cahier de spécifications et l'état de l'art (fait avec les informations obtenues grâce à l'étape précédente). Cette partie a aussi permis de faire une analyse plus approfondie de la méthode choisie (voir Section 4.1 (Chapitre 3)) et de tester son fonctionnement avec des images (voir Figure 1 (Annexe D)).

Reste à faire

Concernant le restes à faire du S9, nous sommes dans les temps, nous avons fait tout ce qui était prévu, il n'y a pas de retards.

2.2 Planning du S10

Le semestre 10 va être consacré à la mise en oeuvre de la méthode pour le classificateur du laboratoire. Plus précisement, il va falloir réimplémenter les 4 grandes fonctionnalités précisées dans le cahier de spécification (voir Annexe B), à savoir la mise en place du classificateur, de la segmentation de vidéo, de l'Explainer et de la méthode permettant de générer le résultat. Une partie test est aussi prévue à la fin du semestre (voir Figure 2 (Annexe D))..

2.3 Semestre 10

Ce semestre 10 aura été consacré à la mise en oeuvre de la méthode et à la rédaction de documents. Voici les étapes qui ont été réalisées.

Mise en oeuvre

Développement de la méthode LIME fonctionnant avec des vidéos à partir du code de la méthode fonctionnant sur des images. Recherche et intégration d'une méthode de segmentation vidéo ([WWW2]). Intégration de deux classificateurs différents.

Reste à faire

Il a été fait tout ce qu'il avait été prévu de faire durant ce PRD, il n'y a pas de retards.

3 Bilan sur la qualité

Concernant la qualité du code, toutes les conventions de nommages ont été respectées pour le code ajouté, cependant pour une partie de code repris (notamment de la méthode LIME), les conventions n'ont pas été modifiées pour garder une cohérence envers le code d'origine. Un documentation a été rédigée, selon les conventions Python, et des indications ont été rajoutées dans les blocs de texte du notebook. Des guides (d'installation, d'utilisation et de développement) ont été rédigés (voir Annexe E, Annexe F et Annexe G) pour permettre une reprise plus facile du projet.

4 Bilan sur la gestion de projet

La gestion du projet à l'aide du logiciel Time Performance s'est bien déroulée. Dans l'ensemble j'étais en avance sur les tâches de mon Gantt, certaines parties du développement étant plus rapides que prévues (notamment l'adaptation de LIME image aux vidéos). Cela m'a permis de passer plus de temps sur les tests, d'ajouter un deuxième classificateur et de procéder à quelques améliorations de l'affichage de la méthode.

Annexes

A Description des interfaces externes du logiciel

1 Interfaces matériel/logiciel

Etant donné que nous allons travailler avec des réseaux de neurones, il peut être intéressant d'utiliser un ordinateur doté d'une carte graphique de manière à accélérer certains traitements. Cependant, si le modèle est déjà entraîné et si la méthode d'interprétation ne l'utilise pas, un GPU peut ne pas être nécessaire.

2 Interfaces homme/machine

Le lancement du programme se fera en ligne de commande, donc il n'y a pas d'interface à ce niveau-là. Concernant l'affichage des résultats, étant donné que l'on utilise LIME sur des vidéos, la sortie sera donc une vidéo avec des masques (zones de couleurs) montrant les zones de la vidéo impactant le plus (positivement ou négativement) la prise de décision. Cette vidéo sera stockée dans un dossier. Le résultat devrait être similaire au résultat obtenu avec une image, comme sur la Figure 2 (Chapitre 3).

3 Interfaces logiciel/logiciel

Les interfaces logiciel dépendent de la façon dont nous allons utiliser LIME. Notre application sera en Python et nous allons utiliser les classes de la méthode LIME développées par les chercheurs. Il y a deux possibilités, soit nous allons utiliser ces classes à la manière d'une bibliothèque et simplement faire appel à elles dans notre code, soit nous allons modifier ces classes pour intégrer le traitement de séquences. Dans ces deux cas, il n'est pas nécessaire de faire des interfaces particulières étant donné que tout se fera en Python.

B Spécifications fonctionnelles

1 Définition de la fonction 1 : batch_predict

Identification de la fonction 1

La fonction batch_predict sert à mettre en œuvre le classificateur. Elle contient toutes les instructions nécessaires pour lancer les prédictions à partir de plusieurs jeux de données d'entrées et retourne les résultats des prédictions. Elle permet à LIME de classer chaque entrée générée dans le voisinage. Cette fonction est primordiale car sans elle il n'y a pas de classificateur à interpréter.

Description de la fonction 1

— Les entrées de cette fonction sont un tableau contenant une ou plusieurs vidéos. Une vidéo serait vraisemblablement constituée de plusieurs ensembles de 30 frames stockées dans des dossiers. Les sorties sont les prédictions pour chacune de ces vidéos. Une prédiction est un tableau contenant les probabilités que la vidéo appartienne à chacune des classes. On aurait donc un vecteur 2D contenant les probabilités d'appartenir à chaque classe.
— Cette fonction utilisera le modèle du classificateur qui sera au préalablement importé. L'appel au classificateur ne modifie pas les données passées en entrées.
— Les traitements associés à cette fonction sont d'abord d'effectuer, si nécessaire, des transformations sur les images des séquences pour qu'elles soient exploitables pour le classificateur (redimensionement et recadrage), et de lancer le classificateur avec les données modifiées pour en extraire les tableaux de probabilités.

2 Définition de la fonction 2 : segmentation_algorithm

Identification de la fonction 2

La fonction segmentation_algorithm est la fonction qui va s'occuper de découper les vidéos de manière à pouvoir plus tard générer les voisins. Une fois que l'on aura découpé les différents « morceaux » de la vidéo, une autre fonction s'occupera plus tard de générer les voisins à partir de ces morceaux. Cette fonction est primordiale car sans elle, il est impossible de générer le voisinage et donc d'utiliser la méthode LIME.

Description de la fonction 2

— Les entrées et les sorties de cette fonction ne sont pas encore connus étant donné que nous ne savons pas encore précisément comment nous allons procéder pour découper des vidéos (le découpage peut être spatial comme temporel). Les paramètres vont dépendre de la façon de découper la vidéo et permettront de modifier la façon dont elle sera découpée (plus finement par exemple).
— En sortie, nous auront une représentation de ces « segments » qui correspondront aux différentes portions des vidéos. Bien évidemment, l'ensemble de tous les segments devrait permettre de représenter la vidéo originale à la manière d'un puzzle.

3 Définition de la fonction 3 : explain_instance

Identification de la fonction 3

La fonction explain_instance a pour rôle de générer le voisinage et d'appeler le classificateur sur chacun des voisins générés. Il génère ensuite une explication à partir des données recueillies. Cette fonction est aussi primordiale car il s'agit du cœur de la méthode LIME.

Description de la fonction 3

— En entrée de cette fonction, nous avons besoin de la vidéo pour laquelle nous voulons générer une explication. Cette fonction va utiliser la fonction segmentation_algorithm qui va être utilisée pour générer le voisinage et la fonction batch_predict pour classer chacun des voisins. D'autres paramètres serviront à paramétrer la fonction, comme le nombre de voisins que l'on veut générer, et pour quels labels on veut générer l'explication (il est possible de générer l'explication pour plusieurs labels).
— La sortie de cette fonction est un objet Explanation, qui correspond à une explication de la prédiction pour les labels choisis. Cet objet Explanation contient les informations relatives aux prédictions dues à chacun des segments, mais n'est pas interprétable en tant que tel.
— Les traitements de cette fonction sont d'abord d'appeler la fonction de segmentation sur la vidéo passée en entrée de manière à récupérer tous les segments (i.e. morceaux de vidéos) et de construire le voisinage de la vidéo à partir de ces segments. Un voisin est donc un ensemble de segments incomplets de la vidéo et tous les "trous" dans la vidéo sont soit des pixels noirs (voir Figure 2a (Chapitre 4)), soit des couleurs unies correspondant à la moyenne de la couleur des pixels cachés (voir Figure 2b (Chapitre 4)). Ce paramètre se défini directement dans le code. Il faut ensuite appeler le classificateur sur chacun des voisins et recueillir les probabilités obtenues pour chaque label. Enfin à partir de ces probabilités et des segments utilisés par les voisins, il s'agit de trouver les poids de chaque segment pour chacun des labels. Une fois que l'on a ces informations, on peut créer un objet Explanation qui contient toutes ces données. Ces données ne sont cependant pas encore interprétables pour un être humain, il faut donc une étape supplémentaire pour en faire un affichage facilement interprétable, c'est le rôle de la fonction suivante.

4 Définition de la fonction 4 : print_result

Identification de la fonction 4

La fonction print_result permet d'extraire les données de l'Explanation de manière à pouvoir être interprétées par un humain. Concrètement, il va s'agir d'afficher la vidéo initiale en avec un masque montrant les zones les plus importantes dans la prise de décision. Cette fonctionnalité

est aussi primordiale, même si elle peut être implémentée après les autres. Cette fonctionnalité nous permettra aussi d'effectuer des tests (notamment des vérifications visuelles des résultats).

Description de la fonction 4

— La fonction print_result prendra en entrée un objet Explanation, un nombre de features qui correspond au nombre de segments que l'on veut afficher (par exemple les 5 segments les plus importants) et le label pour lequel on veut afficher l'explication. Normalement, elle ne devrait rien renvoyer en sortie étant donné qu'elle ne fait que de l'affichage ou de l'écriture de fichiers.
— Les traitements sont d'abord de récupérer la vidéo de base, les masques des segments et leurs poids pour le bon label. Ensuite, il s'agit d'assembler les bons masques correspondant aux segments voulus avec la vidéo. Enfin, il y a deux possibilités pour l'affichage, soit on affiche la vidéo directement dans la console Python (si c'est impossible, on peut afficher toutes les frames une par une), soit on enregistre tous cela dans des fichiers (un fichier par frame).

C Spécifications non fonctionnelles

1 Contraintes de développement et conception

1.1 Contraintes matérielles

Le modèle fourni par le laboratoire est un modèle entraîné, donc normalement, il n'y a pas besoin de GPU pour cette partie. En revanche, la méthode d'interprétation LIME nécessite d'utiliser le classificateur de nombreuse fois pour une même données d'entrée, un ordinateur performant peut être utile pour limiter le temps d'exécution. Si le modèle doit être réappris, alors l'utilisation d'un GPU s'avèrera indispensable.

1.2 Langage de développement

Etant donné que les bibliothèques utilisées pour faire du machine learning, le modèle de décision, et les méthodes d'interprétation sont écrite en Python, nous allons développer notre programme dans ce langage.

1.3 Logiciels et bibliothèques à utiliser

Le logiciel et les bibliothèques à utiliser seront donc le modèle de décision, le code de la méthode d'interprétation utilisée et les bibliothèques permettant de les faire fonctionner, comme par exemple PyTorch et TorchVision, qui sont des bibliothèques permettant de faire du deep learning en Python. Le modèle de décision est fourni par le laboratoire, c'est un doctorant qui a travaillé dessus et nous avons pu récupérer le code nécessaire pour le faire fonctionner. Il faudra cependant le lancer sur des échantillons de vidéos pour s'assurer qu'il fonctionne correctement. Quand au code de la méthode LIME, il est disponible sur GitHub et a déjà été testé avec des images (voir Chapitre 4).

1.4 Environnement

Puisque le développement est en Python, il peut être nécessaire de créer un environnement virtuel qui permet une gestion plus simple des packages importés. Pour cela, j'utilise Anaconda qui simplifie la gestion des packages. L'IDE utilisé sera probablement PyCharm ou Spyder.

1.5 Protocoles de communication

Etant donné qu'il n'y a pas vraiment de problématique concernant les communications, il n'y a pas de contrainte sur les protocoles de communication.

2 Contraintes de fonctionnement et d'exploitation

2.1 Performances

Normalement, l'utilisateur utilisera déjà un modèle interprétable « entraîné » qui est habituellement une étape longue. Le paramètre qui devrait le plus jouer sur le temps d'exécution de la méthode est donc le nombre de voisin généré. En effet il faut classifier chacun des voisins et s'il y en a beaucoup, l'application pourra mettre du temps pour s'exécuter. Dans notre cas, on peut considérer que l'utilisateur est patient, il peut par exemple lancer une interprétation la nuit et récupérer le résultat le lendemain.

2.2 Capacités

Du point de vue de l'environnement, il n'y a pas de contraintes temporelles particulières.

Notre modèle doit s'appliquer à des vidéos qui est un type de données assez lourd. De plus étant donné que nous générons de nombreux voisins, on peut se retrouver à traiter une quantité assez importante de données en terme de taille. Il faudra prendre en compte ce facteur dans le développement du programme pour ne pas avoir des problèmes de mémoire RAM.

2.3 Contrôlabilité

De messages seront affichés dans la console pour suivre l'exécution du programme et avertir l'utilisateur en cas de problème.

2.4 Sécurité

Il n'y a pas de contrainte liée à la sécurité étant donné qu'il n'y a qu'un seul type d'utilisateur.

2.5 Tests

La problématique des tests est délicate avec ce projet car une interprétation n'est pas vérifiable. On peut cependant imaginer plusieurs tests une fois l'application terminée pour s'assurer de son bon fonctionnement et pour tester l'efficacité de la méthode. Le premier test à faire est la confrontation visuelle qui permettra de s'assurer que les données obtenues sont cohérentes et qu'il n'y a pas de résultat aberrant. On pourra aussi s'assurer que pour des vidéos de même classe on obtient bien les même recouvrements (ou au moins on retrouve des similarités dans les recouvrements). On pourra aussi essayer d'établir un lien entre la performance du classificateur et les régions identifiées par la méthode. Pour cela, on pourra construire une matrice de confusion pour voir si les mêmes régions reviennent pour les classes fiables. On peut aussi prévoir des tests fonctionnels et unitaires avec un faux classificateur, ou bien avec des vidéos et un classificateur très simples.

3 Maintenance et évolution du système

Le but de ce PRD est de prendre en main une méthode permettant d'interpréter une boite noire. Lorsque celui-ci sera terminé, on peut imaginer plusieurs extensions, comme par exemple tester la méthode avec une autre boite noire et bien avec d'autres données. Il est donc important de prendre cela dans l'architecture de manière à pouvoir les changer facilement.

D Gestion de projet

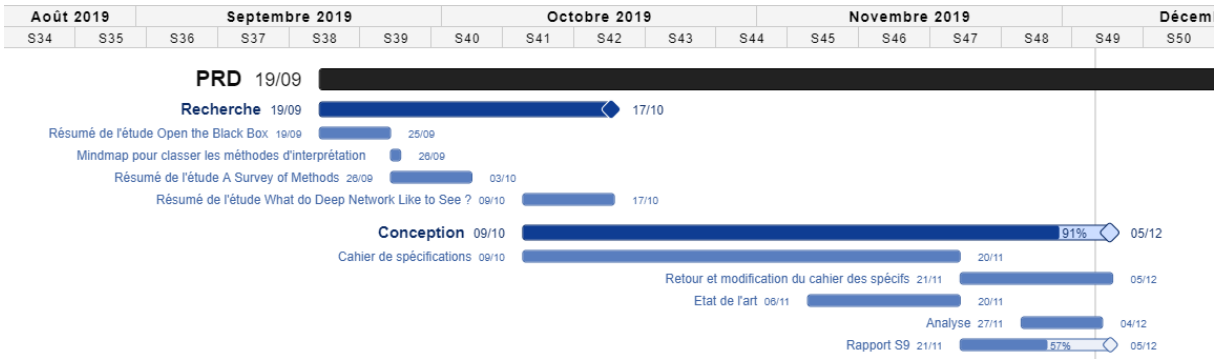


Figure 1 – *Diagramme de Gantt montrant le travail réalisé au S9*

Figure 2 – *Planning prévisionnel du S10*

Pour les bases méthodologiques et les outils utilisés, se référer à la Section 4 (Chapitre 1).

E Installation et déploiement

Pour ouvrir le projet, il faut utiliser Google Colaboratory. Pour ce faire, il faut se rendre sur le site https ://colab.research.google.com/ et importer le fichier .ipynb (importer -> choisir le fichier -> importer le fichier Lime_video.ipynb)

Avant d'exécuter une cellule, il faut vérifier que le GPU est activé en allant dans Modifier -> Paramètres du notebook et choisir GPU dans la catégorie Accélérateur matériel. Vous pouvez ensuite quitter ce menu et cliquer sur Se connecter pour se connecter à un environnement virtuel.

Par la suite il sera possible d'accéder au Notebook directement avec l'URL de celui-ci, il ne sera plus nécessaire de paramétrer l'exécution sur GPU. Les étapes qui suivent, en revanche, sont à faire à chaque fois que l'on se reconnecte au Notebook après l'avoir quitté, ou après avoir réinitialisé l'environnement d'exécution.

Si l'on veut utiliser le classificateur fonctionnant avec les vidéos de liquides, il faut importer le fichier R2Plus1D.py qui contient le modèle. Pour ce faire, il faut s'assurer d'avoir sélectionné l'onglet fichier à droite, puis de glisser-déposer le fichier dans la colonne à droite de l'écran.

Pour pouvoir avoir un espace de stockage qui ne soit pas temporaire, il faut utiliser un compte Google Drive, c'est au travers de celui que l'on va stocker nos vidéos d'entrée et nos résultats finaux. Il faut donc monter notre compte Drive qui va nous servir de disque pour stocker nos données. Pour cela, lancer l'exécution de la première cellule (bouton play en haut à gauche de la cellule). La cellule va afficher un lien qui permet de se connecter à son compte Google. Un fois connecté, un code est affiché, il faut alors le copier-coller dans le champ de la cellule et appuyer sur entrée. Cela va monter le compte Google Drive directement dans l'environnement.

Dernière étape, il faut installer les packages que l'ont aura besoin d'utiliser (torchivision et lime). Pour cela il suffit d'exécuter la deuxième cellule (package installation) et l'installation se fait automatiquement. Comme cette exécution réinstalle torchvision (il était déjà installé mais sous une mauvaise version), il faut ensuite cliquer sur "Exécution" en haut à droite puis "Redémarrer l'environnement d'exécution...". Cela permettra de prendre en compte les versions nouvellement installées.

Votre environnement est maintenant prêt à exécuter le programme.

F Manuel d'utilisation

1 Les différentes parties

Le programme se comporte en 5 grandes parties :

- Classifier : Contient la classe nécessaire au chargement et à l'utilisation du modèle du classificateur,
- Preprocessing : Contient les méthodes préparant les données pour la classification et la segmentation,
- Segmentation : Contient les méthodes et appels systèmes permettant de segmenter les vidéos,
- LIME Method : Contient les classes permettant d'utiliser la méthode LIME pour interpréter une décision,
- Execution : Contient les méthodes pour exécuter tout le programme.

2 Exécuter les exemples

Deux classificateurs différents ont été implémentés pour fonctionner avec la méthode LIME, le premier classificateur fonctionne avec des vidéos de liquide et estime leur vitesse de rotation. Le deuxième classificateur est capable de reconnaître quels objets ou éléments sont présents sur une vidéo (classificateur d'objets).

Pour utiliser le premier classificateur, il faut importer le fichier python contenant le modèle (comme précisé Annexe E).

2.1 Paramétrage des chemins

Avant d'exécuter les cellules, il faut aussi s'assurer d'avoir le fichier contenant les poids du modèle du classificateur, la vidéo d'entrée et un dossier pour contenir les résultats dans son compte Google Drive. Ensuite, naviguez dans le Notebook jusqu'à la section TEST Liquid video si vous souhaitez utiliser le classificateur de liquides ou bien la section Test Object Video si vous souhaitez utiliser l'autre classificateur. On peut voir que pour interpréter une vidéo, il faut utiliser deux fonctions. La première est un appel au constructeur de LIME qui va se

charger de créer et de charger le modèle de classificateur. Pour commencer, changer la valeur de model_path par le chemin menant au fichier contenant les poids du modèle dans votre compte Drive.

L'autre fonction appelée (la fonction explain()) permet de lancer toute les méthodes nécessaires pour obtenir notre vidéo interprétée. Les deux premiers paramètres sont respectivement le chemin du compte Drive vers la vidéo en entrée, et le chemin du compte Drive vers le dossier où l'on veut enregistrer nos résultats. Veuillez modifier ces chemins pour qu'ils correspondent aux votres.

2.2 Les différents paramètres

Avant d'exécuter le programme, voici un rappel des différents paramètres qui influent sur le résultat que l'on obtient avec LIME. La méthode LIME contient plus de paramètres, mais ils ont été simplifiés pour faciliter son utilisation. Tous ces paramètres sont les paramètres de la méthode explain vue précédemment. Voici les différents paramètres :

— segmentation_hierarchie : Il s'agit du niveau de segmentation. Un nombre petit impliquera des segments plus petits et plus nombreux, alors qu'un nombre plus grand entraînera des segments plus gros et moins nombreux. Les valeurs conseillées pour ce paramètre sont entre 20 et 40.
— number_features : Nombre de segments que l'on veut surligner. La méthode va forcément surligner les segments qui ont le plus de poids. Par exemple, si l'on choisit 5, alors la méthode va surligner sur notre vidéo les 5 segments qui impactent le plus notre classificateur.
— number_sample : Il s'agit du nombre de vidéos voisines que notre méthode va générer. Plus ce nombre sera grand et plus le temps d'exécution sera long, mais plus le résultat sera fiable. Il est recommander de mettre un nombre plus grand si l'on a de nombreux petits segments. Une valeur par défaut pour ce paramètres est de 1000 samples.

2.3 Première exécution

Une fois les paramètres correctement configurés, on peut passer à l'exécution du programme. Pour cela, retournez à la troisième cellule (la première de la catégorie Classifier) et exécutez la (bouton play en haut à gauche de la cellule) ainsi que toutes les suivantes jusqu'à la section Execution où vous exécuterez la cellule qui vous intéresse en fonction du classificateur utilisé. Il n'est pas nécessaire d'attendre la fin de l'exécution d'une cellule avant de lancer la suivante, elles s'exécuteront toutes dans l'ordre auquel vous les aurez activées.

L'exécution dans sa globalité du programme peut prendre quelques minutes. Une fois terminée, les résultats se trouvent dans le dossier de résultat sur votre compte Drive.

2.4 Exécutions suivantes

Si vous voulez modifier les paramètres, changer de vidéo ou autre, il n'est pas nécessaire de tout réexécuter si vous êtes toujours connecté au même environnement Google Colab. Vous avez juste à exécuter la cellule que vous avez modifiée.

En revanche, si vous quittez l'environnement d'exécution, en fermant la page web par exemple, il faudra réexécuter de nouveau toutes les cellules (après avoir suivi les étapes indiquées Annexe E).

Si vous exécutez plusieurs fois la méthode sur la même vidéo, le temps d'exécution devrait être un peu plus court car la segmentation est sauvegardée temporairement pendant la session.

Il faut aussi prendre en compte le fait que la méthode ne sauvegardera pas les résultats si ils existent déjà, c'est-à-dire si la méthode est exécutée sur une vidéo déjà traitée avec les mêmes paramètres.

3 Ajouter un nouveau classificateur

Il est possible d'utiliser le code de la méthode LIME sur un nouveau classificateur. Pour cela, il faut que le classificateur fonctionne avec des vidéos, et qu'il soit possible de convertir la vidéo que le classificateur prend en entrée en fichiers d'images (png,jpg, ect...), et de convertir une liste d'images sous forme de tableaux Numpy en un format utilisable en entrée du classificateur. Il faut aussi que le modèle du classificateur soit utilisable avec la librairie Torch.

Pour ajouter un nouveau classificateur, il faut tout d'abord écrire deux nouvelles fonctions. Pour vous aider, vous pouvez vous baser sur la partie preprocessing du programme qui contient ces deux fonctions pour les deux classificateurs déjà implémentés.

La première fonction (appelons la fonction A) va servir à convertir la vidéo en entrée en fichiers, où chaque fichier correspond à une image de la vidéo. L'équivalent de la fonction A pour le classificateur de liquides est la fonction store_video_in_one_folder(), et l'équivalent pour le classificateur d'objets est la fonction split_video_in_images(). Vous pouvez vous baser sur ces fonctions pour construire la fonction A. La fonction A doit prendre en entrée le chemin de la vidéo en entrée, et un nombre d'images n. Il faut qu'elle sépare la vidéo en plusieurs fichiers et qu'elle ne garde que les n premières images. La fonction A doit retourner le dossier où se trouve toutes les images créées.

La deuxième fonction (fonction B) va servir à convertir une liste d'images sous forme de tableau Numpy en un format utilisable directement par le classificateur. Les fonctions équivalentes pour les classificateurs de liquide et d'objets sont respectivement convert_images_to_clip() et convert_images_to_frame_block() La fonction B doit prendre en entrée une liste d'images où chacune de ces images est un tableau Numpy et retourner un objet directement exploitable par le classificateur.

Un fois que ces deux fonctions sont créées, vous pouvez retourner dans la partie Execution, rajouter une nouvelle cellule et créer un nouvel objet LIME en appelant son constructeur. Il faudra passer en paramètre le modèle du classificateur, le chemin vers le fichier contenant les poids du modèle, la fonction B, puis la fonction A, et indiquer si les images sont en RGB ou non avec un booléen.

Une fois l'objet créé, il est possible d'appeler la méthode explain, de la même manière qu'avec les exemples.

4 Interprétation des résultats

Les vidéos produites sont disponibles dans le compte Google Drive que vous avez passé en paramètre de la fonction explain. Chaque vidéo est stockée dans un dossier contenant des images et un fichier texte. Les images composent la vidéo et sont triées dans l'ordre. Les résultats sont visibles sur les images. Certaines d'entre elle auront des parties surlignées en vert ou en rouge. Chaque partie surlignée correspond à un segment de la vidéo qui a beaucoup influencé le classificateur. Les segments en vert sont ceux l'ayant influencé positivement (ce sont les segments qui l'ont poussés a prendre la décision qu'il a prise), et les segments en rouge sont ceux qui l'ont influencés négativement (ces segment l'ont plutôt poussés vers une autre décision).

Attention cependant pour les vidéo en RGB, les valeurs des pixels modifiées par un segments surlignés ne sont que sur la composante Green du pixel pour les segments verts et sur la composantes Red pour les segments rouges. Cela veut dire que sur des images avec des couleurs "fortes", les segments peuvent apparaître d'une autre couleur (comme turquoise par exemple, si le pixel est bleu à la base).

Le fichier texte contient tous les poids des segments en fonction de leur numéros. Il permet de se rendre compte des poids respectifs des segments, et de voir par exemple si un segment a particulièrement influencé la décision ou si tous les segments ont des poids similaires.

Chaque segment affiché sur les images possède un numéro qui permet de le retrouver dans le fichier texte et ainsi pouvoir connaître son poids dans la décision.

Le nom du dossier dans lequel sont stockées les images du résultat contient les différents paramètres utilisés, il prend la forme suivante :

nomDeLaVideo_classePredite_nombreDImages_hierarchieDeSegmentation _NombreDeSegmentsAffichés_NombreDeVoisinsGénérés

Exemple : maVideo.mp4_5_30_35_3_1000

G Guide du développeur

Ce petit guide décrit globalement la structure du programme et comment tous les modules fonctionnent entre eux.

Nous allons décrire plus en détail les grandes parties de programme présentées dans le tableau Section 1 (Annexe F).

Voici le fonctionnement simplifié du programme. On a notre vidéo et notre classificateur.

On va d'abord segmenter notre vidéo (découpage spatial et temporel), ce qui nous donne des segments.

Ensuite on va générer les voisins de notre vidéo. Un voisin est constitué de notre vidéo originale à laquelle on a caché certains segments de manière aléatoire. Les segments cachés sont des parties de la vidéo qui vont être noires. Ensuite, on va passer notre vidéo originale et tous les voisins au classificateur qui va établir des prédictions. Dans la pratique, ces deux dernières étapes ne se font pas l'une après l'autre, mais on va plutôt générer un certain nombre de voisins avant de les classifier et on recommence jusqu'à obtenir le bon nombre de voisins (cela permet de ne pas surcharger la mémoire).

Une fois que l'on a nos prédictions, la méthode Lime va les utiliser pour calculer les poids des différents segments.

Enfin, il ne reste plus qu'à créer la vidéo de résultat et surligner de la bonne couleur les segments les plus importants.

Dans la pratique, ces différents étapes sont séparées dans des classes, et elle s'exécutent au sein de la classe LIME qui se charge d'appeler tous les éléments dans le bon ordre.

A l'initialisation de cette classe, elle va créer le classificateur (objet Classifier) et charger les poids du modèle. Au lancement de sa fonction explain, elle va d'abord créer l'objet Segmentation, segmenter la vidéo et stocker les segments. Ensuite, la fonction va créer l'objet LimeVideoExplainer, qui va se charger de générer les voisins à l'aide des segments et les passer dans le classificateur. Le LimeVideoExplainer va ainsi générer un objet VideoExplanation contenant les données sur les poids des segments. Enfin, la fonction va utiliser ces résultats pour générer la vidéo de sortie.

Voici un pseudo-diagramme UML décrivant ces différentes classes.

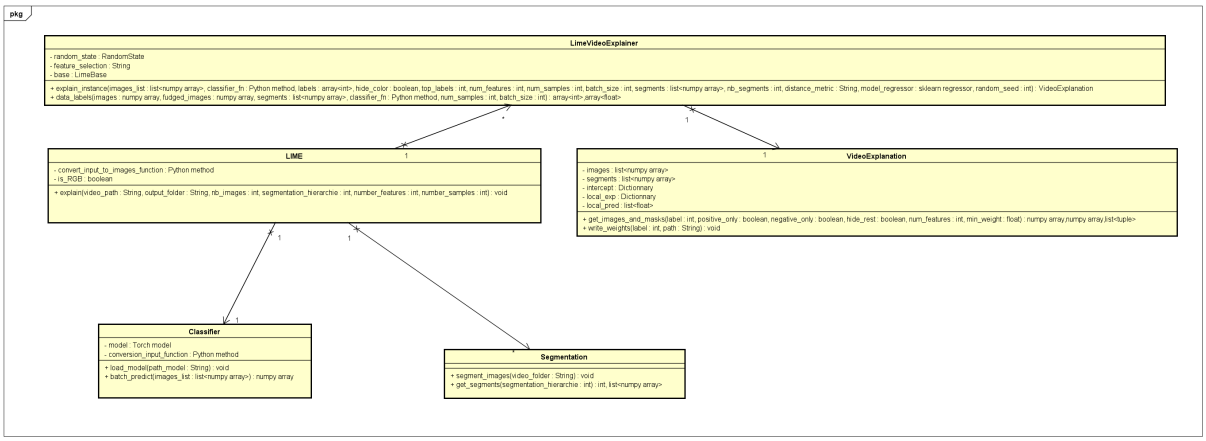


Figure 1 – *Planning prévisionnel du S10*

H Documents supplémentaires produits

1 Résumé de l'étude Open the Black Box Data Driven Explanation of Black Box Decision Systems [5]

Résumé d'étude

Open the Black Box Data-Driven Explanation of Black Box Decision Systems

1-Introduction

- Les algorithmes de Machine Learning sont des boites noires où le modèle de décision est incompréhensible ou secret.
- Manque de transparence mais aussi possibles biais cachés dans les algorithmes pouvant entrainer de mauvaises décisions.
- Besoin de technologies capable d'expliquer la logique de ces boites noires.
- Malgré l'intérêt croissant pour le ML interprétable, il n'existe pas de technologie largement déployable dans l'explication des boites noires.

2-Le problème de l’explication des boites noires

- 2 approches possibles :
  - L’interprétation par le design (eXplanation by Design (XbD)) : avec un jeu de données d’entraînement connu, comment développer un modèle de décision de machine learning en y intégrant les mécanismes d ’explication.
  - L’interprétation de boites noires (Black Box eXplanation (BBX)): comment reconstruire une explication du modèle à partir des enregistrements de décision en sortie d’une boite noire dont le modèle est inaccessible.

eXplanation by Design (XbD)

- Avec cette approche, le développeur du modèle de décision de machine learning doit fournir une explication de la logique du modèle.
- Le data scientist a le contrôle total sur le processus de création du modèle et le développement de la méthode d’explication est surtout une étape de validation de la qualité du modèle de sortie.

Black Box eXplanation (BBX):

- Problème plus difficile que le précédent.
- Le but est de trouver une interprétation d'une boite noire conçue par d'autres personnes.
- On ne connait pas le jeu de données servant à l'entrainement de la machine ni les mécanismes internes du modèle.

Méthodes existantes en XbD

Explication de la logique globale d'une boite noire :

- La boite noire est associée à un classifieur interprétable qui imite la boite noire.
  - Surtout conçu pour des modèles de machine learning spécifiques donc dépendant du modèle.
  - Le classifieur consiste souvent en un arbre de décision ou un jeu de règles de décision.
- Il existe quelques méthodes indépendantes du modèle.

Explication de la logique locale d'une boite noire :

- On cherche à expliquer le résultat pour une instance spécifique
- Certaines méthodes dépendent du modèle et visent à voir quelles portions des données d'entrée sont principalement responsables du résultat.
- D'autres méthodes sont indépendantes du modèle (ex : LIME), l'idée est de trouver une explication pour la décision *y* sur l'instance *x* en apprenant un modèle interprétable à partir de données générées aléatoirement au voisinage de *x*, et en les étiquetant avec la boite noire.

Etat de l'art actuel et résultat visé

- Aujourd'hui, l'état de l'art montre des résultats ad hoc, éparpillés et souvent liés à des modèles spécifiques.
- D'après l'étude, un framework d'explication de boites noires largement applicable devrait avoir les propriétés suivantes :
  - Indépendant du modèle, il doit pouvoir être appliqué à n'importe quel modèle de boite noire.
  - Basé sur la logique, les explications doivent être compréhensibles pour des hommes de différentes expertises, et basées sur le raisonnement.
  - A la fois local et global, doit pouvoir expliquer les cas individuellement ainsi que la logique globale du modèle de boite noire.
  - Haute fidélité, doit fournir une approximation fiable et précise du comportement de la boite noire.

3 Comment construire des explications qui ont du sens ?

- Création d'un framework à partir d'un problème XbD avec une boite noire *b* de haute qualité, sans overfitting, entraînée avec un jeu de données *D* composé d'exemples (x,y) où x est le vecteur des caractéristiques observées et y l'étiquette de classe.
- Classification binaire donc $y \in \{0, 1\}$.

Hypothèses de départ

- H1: Explications logiques. Le moyen de véhiculer les explications doit être proche d'un langage de raisonnement, il doit être logique.
- H2: Explications locales. La frontière de décision peut-être complexe sur toutes les données d'entraînement, mais au voisinage de chaque point spécifique, il y a une forte chance qu'elle soit simple et interprétable.
- H3: Composition d'explications. Il y a une forte chance que deux points similaires admettent une explication similaire, et qu'il soit possible d'assembler les explications similaires de manière à former une explication globale de la boite noire.

Etapes

- Les hypothèses suggèrent 2 étapes :
  - (Etape locale) Pour chaque point *(x,y')* du jeu de données D, on interroge la boite noire *b* sur un nombre d'exemples au voisinage de *(x,y')* de manière à pouvoir construire une explication *e(x,y')*. L'explication doit répondre à la question : Pourquoi *b* a assigné la classe *y'* à *x* ? Et possiblement à : Que doit-on changer dans *x* pour obtenir un résultat différent.
  - (Etape de composition) On considère l'ensemble de toutes les explications locales comme une première explication globale, puis on synthétise itérativement les explications similaires.

3.1 Règles d'explication

- Elles peuvent être classées des plus simples aux plus expressives :
  - Associations simples avec des caractéristiques nominales, ex : `CheckingBalance=low, SavingBalance=low → Credit=no,`
  - Règles de décision sur des caractéristiques de types génériques, ex : `CheckingBalance < 0, SavingBalance < 100 → Credit=no,`
  - Règles avec des contraintes entre les caractéristiques, ex : `CheckingBalance > 0, SavingBalance > 100, CreditBalance + SavingBalance < 200 → Credit=no,`
  - Règles avec des caractéristiques paramétriques, abstrayant un ensemble de contraintes entre plusieurs caractéristiques, ex : CreditBalance + `SavingBalance ≥ a, CreditBalance − SavingBalance ≥ 500−a, 200 ≥ a ≥ 300 → Credit=no.`

3.2 Explications locales

- Différentes méthodes existent pour les explications locales. Elles fonctionnent suivant les mêmes principes :
  - A partir d'un couple *(x,y')* d'une solution donnée, on génère un voisinage *N(x,y')*.
  - On construit un classifieur interprétable à partir de *N(x,y')*.
- Les trois méthodes proposées sont différentes sur la manière d'exécuter ces étapes.

3.2 Explications locales

- LIME : Génère le voisinage aléatoirement et utilise un modèle linéaire comme classifieur.
- Anchor : Génère le voisinage avec l'algorithme Bandit. L'explication est formée à partir de règles de manière à ce que si l'on change les caractéristiques qui ne sont pas dans ces règles, le résultat reste le même.
- LORE : Génère le voisinage grâce à un algorithme génétique, et utilise un arbre de décision en tant que classifieur. Il permet aussi de générer des règles contrefactuelles suggérant que le changement d'une caractéristique entraîne une décision différente.

3.3 Passer des explications locales à une globale

- Dans cette partie, on souhaite rassembler les explications locales en en explication globale.
- Pour cela, on réalise un dendrogramme, un arbre binaire décrivant les compositions de paires d'explications similaires en une seule plus générale.
- On utilise pour cela une fonction de distance $d(e,e') \in [0, 1]$ où 0 signifie que e et e' sont identiques et 1 qu'elles sont disjointes, ainsi qu'une fonction de fusion $m(e, e') = e''$ permettant de fusionner e et e' en une explication plus générale (mais minimale) et les contenant toutes les deux.
- On cherche alors les deux explications les plus proches, on les fusionne, et on ajoute un nœud au dendrogramme dont la hauteur correspond à la distance entre les deux explications. On recommence jusqu'à ce qu'il n'y ai plus qu'une explication.

3.3 Passer des explications locales à une globale

- Ensuite il faut exploiter le dendrogramme de manière à extraire une collection d'explications pouvant servir d'explication globale et qui couvre toutes les explications locales. Cette collection *E* extraite doit maximiser un score de qualité *q(E)* qui mesure la fidélité de la collection à imiter la boite noire, et le nombre et la taille de ses explications (c'est-à-dire sa complexité).
- L'objectif est de maximiser la fidélité en minimisant la complexité.
- Pour mesurer *q(E)*, on utilise le Critère d'information bayésien (BIC).

Résultats

- On utilise LORE et Anchor pour retrouver les explications locales, l'algorithme de distance entre les explications est basé sur la distance de Jaccard et on utilise BIC pour trouver l'explication globale optimale.
- On compare les résultats avec deux autres méthodes :
  - Classification globale pure : apprendre un arbre de décision sur tout le jeu de données d'entrainement.
  - La collection de l'ensemble des explications locales obtenues avec LORE et Anchor.
- On obtient une fidélité similaire au deux autres méthodes mais avec une complexité nettement inférieure (on passe d'environ 600 règles à 50).

4 Nouvelles pistes de recherche

- Nécessité de développer de nouveaux algorithmes de découverte de règles pour travailler avec des règles de plus haut niveau.
- Etendre l'approche au delà de la classification binaire, comme la classification de données ordinales et la prédiction de variables numériques.
- Améliorer la communication avec l'utilisateur, en rendant plus intelligibles les explications données.
- Fournir des interfaces visuelles et textuelles.

2 Résumé de l'étude A Survey of Methods for Explaining Black Box Models [3]

Résumé d'étude

A Survey of Methods for Explaining Black Box Models

1 - Introduction

- Rappel des problèmes liés au manque d'interprétabilité des boites noires.
- Différentes communautés ont étudié le problème, chacune d'une perspective différente et donnant un sens différent à l'interprétabilité.
- Besoin d'une organisation ou d'une classification des méthodologies proposées dans la littérature.
- Cette étude a donc pour objectif de classifier toutes ces méthodes selon différents critères.

2 – Pourquoi a-t-on besoin de modèles interprétables ?

Liste d'exemples de modèles biaisés ou faisant preuve de discrimination :

- Algorithme calculant le score COMPAS (risque de récidive de crimes) ayant un fort biais ethnique.
- Exclusion des quartiers contenant des minorités par un logiciel utilisé par Amazon pour calculer les zones où offrir une livraison en 1 jour gratuite.
- Différences d'au moins 50 points pour 29% des scores attribués à des mêmes clients par 3 grandes banques américaines.
- Logiciel militaire de reconnaissance de tanks biaisé par la couverture nuageuse.
- Logiciel de reconnaissance de husky ou de loup ne se basant que sur la présence de neige en arrière plan.
- Des altérations indétectables d'images peuvent conduire à des résultats aberrants pour des algorithmes de deep learning de reconnaissance d'images.

3 – Caractéristiques d'un modèle interprétable

- Interprétabilité : Habilité à expliquer ou fournir une signification dans des termes compréhensibles par des humains.
- Une « explication » est une interface entre un humain et l'algorithme qui prend la décision et qui représente précisément la prise de décision tout en étant compréhensible par un humain.

3.1 – Dimensions d'interprétabilité

Différentes dimensions à prendre en compte pour fournir une interprétation d'un modèle :

- Globale ou locale : modèle complétement interprétable ou interprétation d'une unique décision.
- Limitation du temps : l'utilisateur peut avoir du temps pour comprendre l'explication du modèle ou bien doit vérifier la décision rapidement.
- Nature de l'expertise de l'utilisateur : les utilisateurs peuvent avoir différents niveaux d'expertise.

3.2 Desiderata d'un modèle interprétable

Un modèle interprétable doit suivre une liste de prérequis :

- Interprétabilité : si le modèle ou ses prédictions sont compréhensibles par un humain (souvent mesuré avec la complexité du modèle interprétable en terme de taille),
- Précision : si le modèle est capable de prédire correctement des instances non-vues,
- Fidélité : si le modèle imite correctement la boite noire qu'il explique,

Ces prérequis sont directement liés à l'interprétabilité, mais il faut en appliquer des supplémentaire correspondants aux modèles issus du data-mining et du machine-learning.

Prérequis supplémentaires

Ces prérequis sont :

- Equité et confidentialité (liés aux aspects éthiques),
- Monotonicité,
- Ergonomie (l'utilisateur aura plus confiance en quelque chose facile à utiliser),
- Fiabilité, robustesse, causalité, scalabilité, généralité

3.3 Modèles interprétables reconnus

3 modèles interprétables sont reconnus pour être facilement compréhensibles et interprétables par des humains:

- Arbres de décision : représentation graphique, structure en graphe, chaque feuille correspond à une classe, peut-être linéarisé en un ensemble de règles,
- Règles de décision : représentation textuelle, ensemble de règles,
- Modèles linéaires : souvent utilisés pour visualiser l'importances des attributs (signe et magnitude) pour une prédiction donnée.

4 – Problèmes d'ouverture des boites noires

On peut séparer les problèmes en deux catégories de haut niveau, le reverse engineering et le interprétation par le design.

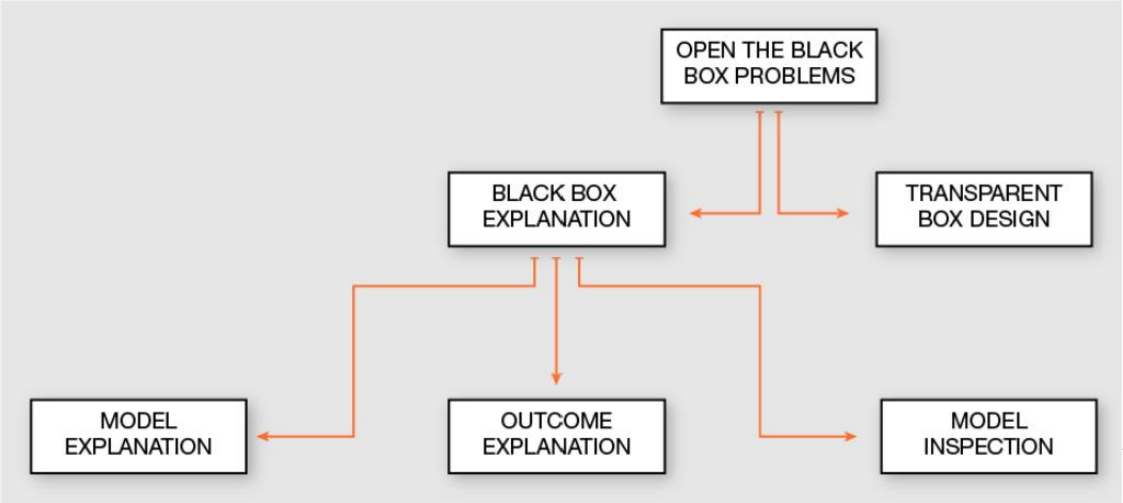
- Reverse engineering : on n'a accès qu'aux résultats et on essaye de reconstruire une interprétation a partir d'eux (appelé Black Box Explanation dans l'étude).
- Interprétation par le design : on construit le modèle de manière à ce qu'il soit directement interprétable (appelé Transparent Box Design).

Black Box Explanation

La partie Reverse engineering (ou Back Box Explanation) se divise en trois parties :

- Model Explanation : correspond à une interprétation globale de la boite noire à travers un modèle transparent et interprétable.
- Outcome explanation : correspond à une interprétation locale de la boite noire. Permet d'expliquer comment on a obtenu le résultat pour une instance.
- Model inspection : consiste à donner une représentation visuelle ou textuelle pour comprendre des propriétés spécifiques du modèle. Cela peut-être par exemple la sensibilité du modèle aux changements des attributs, ou bien l'identification des composants (ex : neurones pour les réseaux neuronaux) responsables d'une décision spécifique.

Open the black box problems



5 – Classification

La classification des méthodologies se base sur le type de problème rencontré et sur le type d'explication donné. La classification tient compte de quatre éléments :

- Le type de problème rencontré,
- le type d'explications fourni,
- le type de modèle de la boite noire,
- le type de données utilisées en entrée du modèle.

Les différents type d'explications

- Decision Tree : un des types les plus interprétables, pour les explications globales ou locales,
- Decision Rules : aussi une des techniques les plus facilement compréhensibles,
- Features Importance : consiste à renvoyer les poids et magnitudes des attributs utilisés dans la boite noire,
- Saliency Mask : consiste à surligner les aspects déterminants d'une image ou d'un texte en entrée,

Les différents type d'explications

- Sensitivity Analysis : étudie la relation entre l'incertitude d'un résultat et de son entrée.
- Partial Dependence Plot : graphique permettant de mettre en relation les entrées et les sorties dans un espace vectoriel réduit.
- Prototype Selection : consiste à retourner en même temps que la sortie un exemple similaire (prototype) à l'enregistrement d'entrée, rendant clair sur quels critères la prédiction a été effectuée.
- Activation Maximization : pour les réseaux neuronaux, consiste à observer les neurones fondamentaux activés pour des enregistrements particuliers de manière à trouver quels paternes sont responsables de l'activation de certains neurones dans certaines couches.

Les différents modèles de boites noires

- Neural Networks,
- Tree Ensemble,
- Support Vector Machine,
- Deep neural Networks,
- Non-Linear Models.

Classification pour les méthodes s'appliquant à tout type de modèle : Agnostic (AGN).

Les différents types de données d'entrée

- Tableau,
- Texte,
- Image.

Il n'existe que très peu de méthodes d'interprétation qui sont applicables à d'autres types de données d'entrée.

Pour résumé

Problème	Type d'explication	Boite noire	Type de données
Model Explanation	DT - Decision Tree	NN – Neural Networks	TAB – Tableau
Outcome Explanation	DR - Decision Rule	TE – Tree Ensemble	IMG – Image
Model Inspection	FI - Features Importance	SVM – Support Vector Machines	TXT – Texte
Transparent Design	SM - Saliency Mask	DNN – Deep Neural Networks	ANY – N'importe quel type de donnée
	SA - Sensivity Analysis	AGN – Agnostic Black Box	
	PDP - Partial Dependence Plot	NLM – Non Linear Models	
	AM – Activation Maximization		
	PS – Prototype Selection		

6, 7, 8, 9 – Présentation des différentes méthodes

Le reste de l'étude classe toutes les méthodes selon les critères vus précédemment et en fait en brève description.

Plutôt que de toutes les lister, voici un tableau répertoriant chaque méthode est diponible.

3 Résumé de l'étude What do Deep Networks Like to See ? [4]

Résumé de l'étude

What do Deep Networks Like to See?

1- Introduction

- Les diagnostiques des réseaux de neurones se basent en général sur des mesures prises à la fin de la chaîne de processus (mesure de l'erreur au cours du temps, top-K accuracy etc…) mais ne permettent pas de dire pourquoi le modèle a échoué.
- D'autres recherches ont été proposées pour mieux comprendre les transformations des données d'entrée en prédictions, en regardant par exemple les signaux internes du réseaux, mais cela sert plus à un but descriptif qu'explicatif.
- Cette étude propose d'étudier le réseau de neurones en se basant sur la quantité d'information des données d'entrées sur lesquelles le modèle s'appuie. C'est-à-dire essayer de mesurer la quantité de signal des données d'entrée que le modèle utilise.

1- Introduction

- Pour faire cela, les chercheurs ont utilisé un auto-encodeur pré-entrainé puis l'on « fine-tuné » en utilisant les gradients d'un classificateur d'image avec des paramètres fixes. Une fois l'auto-encodeur prêt, les prédictions se font en passant les images à travers l'auto-encodeur d'abord.
- Ce processus a deux effets :
  - D'abord les informations utiles de la donnée d'entrée sont préservées alors que les parties inutiles sont supprimées,
  - Ensuite, l'utilisation d'auto-encodeurs atténue des aspects distractifs de l'image, comme le bruit.
- Les chercheurs ont testé cette méthode sur 5 différents DCNN, donnant des résultats différents pour chaque classificateur.

3.1 Méthode

La méthode appliquée est en deux temps :

- 1) Entrainer les auto-encodeurs avec différents modèles,
- 2) Quantifier et comparer les images reconstruites par les auto-encodeurs à travers :
  - a) des changements de précision lorsque que l'on passe ces images modifiées dans un autre classificateur,
  - b) la quantité d'information contenue dans les images reconstruites.

3.1 Sélection de l'auto-encodeur

Besoin d'un auto-encodeur :

- Capable de capturer de manière fiable de larges quantités d'information contenues dans les données d'entrées,
- Où la mesure de validation dépend de la précision du classificateur, c'est-à-dire que l'on considère que l'auto-encodeur produit une bonne reconstruction si la précision du classificateur d'image ne change pas (ou est améliorée) par rapport à la vraie image d'entrée.

Les chercheurs ont choisis SegNet pour l'architecture de l'auto-encodeur, il s'agit d'un auto-encodeur convolutionnel conçu pour la segmentation sémantique d'image RGB.

3.2 Pré-training de l'auto-encodeur

- L'auto-encodeur est d'abord pré-entrainé pour minimiser les pertes lors de la reconstruction, cela permet d'éviter aux images transformées de modifier significativement la précision du classificateur.
- L'AE est entraîné avec la base d'image YFCC100m qui contient environ 100 millions d'images utilisables.

3.3 Fine-tuning de l'auto-encodeur

- Fine-tuning en utilisant les gradients provenant d'un classificateur pré-entraîné avec des paramètres fixes.
- En principe, l'auto-encodeur doit garder les parties de l'image importantes qui améliorent la reconnaissance et délaisser celles qui n'ont pas un impact positif.
- Les meilleurs résultats sont obtenus en ne modifiant seulement les poids appartenant au décodeur.
- On note que l'utilisation d'un AE fine-tuned avec le même classificateur entraîne une augmentation de la top-k accuracy jusqu'à presque 4 points, alors qu'utiliser l'AE qui est seulement pré-entraîné la diminue jusqu'à -4 points.

3.3.1 Représentations encodées des AE

En étudiant les images encodées qui sortent des AE, on peut voir les différences avec les images initiales.

Ces différences changent en fonction des classificateurs :

- VVG16 BN modifie le moins les images,
- AlexNet et ResNet-50 introduisent des artefacts sur tout l'espace d'entrée,
- ResNet-50 et Inception-v3 compressent la luminance (sombre plus clair et clair plus sombre)
- Inception-v3 diminue les bleus/jaunes-marrons,
- AlexNet produit des images avec une teinte rose.

3.3.1 Représentations encodées des AE

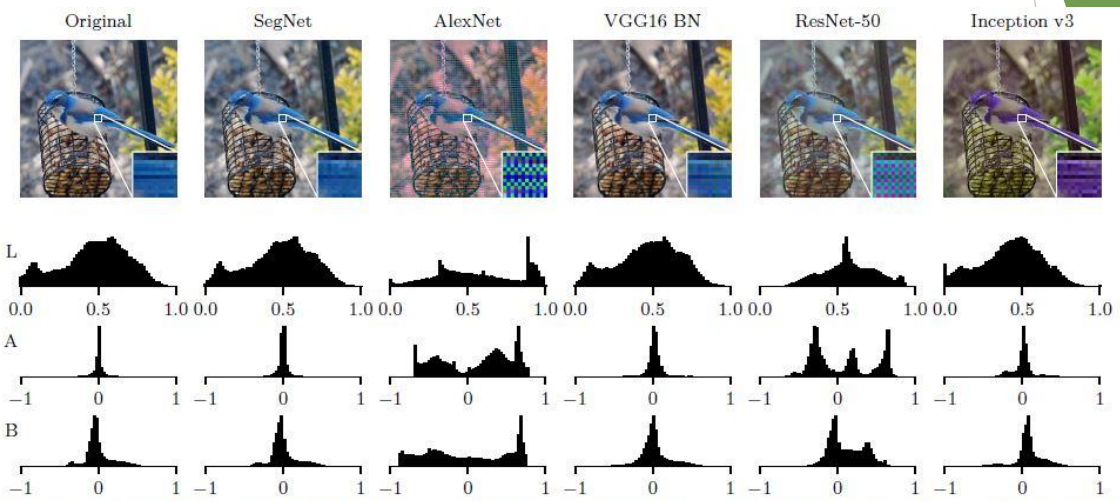
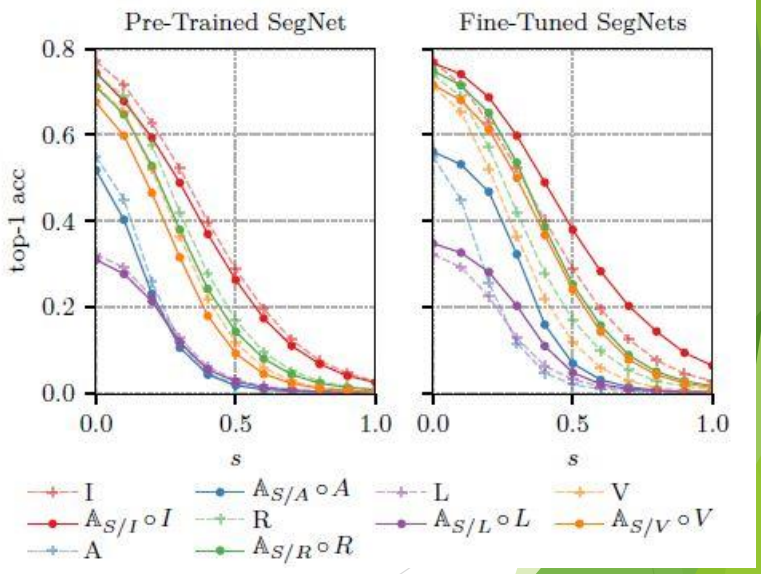


Figure 3. Example taken from ImageNet's validation set. Each image corresponds to a reconstruction produced by one of the fine-tuned AEs. Below each reconstruction, global histograms of the image in LAB space are shown. Zoomed portions of the image reconstructions are provided, showing the emergent patterns in more detail.

3.3.2 Résilience émergente au bruit

- Un conséquence du fine-tuning est une meilleure précision lorsque du bruit est présent sur l'image initiale.
- Lorsque l'on teste la classification avec des images initiales bruitées et avec un AE seulement pré-entraîné, on observe une baisse de la précision.
- Si l'on teste avec un AE fine-tuned, le résultat sera bien meilleur pour n'importe quel niveau de bruit, indiquant que l'AE permet d'avoir une représentation plus utile pour le classificateur.



3.4 Mesure de l'information grâce aux classificateurs

- Les chercheurs ont testé chaque classificateur avec chaque AE.
- Les résultats montrent que la précision de chaque mélange reste supérieur ou égal au minimum de la précision des deux modèles utilisés (celui pour le fine-tuning et celui pour la classification). Cela veut dire que les réseaux utilisent au moins le signal utilisé par le réseau le moins performant.
- L'analyse montre que tous les réseaux extraient les features en se basant sur un portion commune du signal d'entrée, même si elle peut n'être que 10% du signal.

3.5 Mesure de l'information grâce aux images reconstruites

- Mesure de la perte et de la préservation d'information dans les images reconstruites.
- Utilisation de la mesure de l'information mutuelle normalisée (nMI) en deux temps :
  - nMI intra-classe : on compare les reconstructions de chaque AE avec l'image originale (une forte correspondance est attendue),
  - nMI inter-class : on compare les reconstructions de deux images différentes (faible correspondance attendue).

3.5 Mesure de l'information grâce aux images reconstruites

- Il est possible de classer les classificateurs en fonction de la quantité de signal d'entrée qu'ils utilisent.
- Les résultat montrent l'existence d'un pattern où les différents réseaux utilisent plus ou moins d'information de l'entrée originale.
- On peut aussi supposer que la perte d'information est aussi utilisée par les classificateurs.

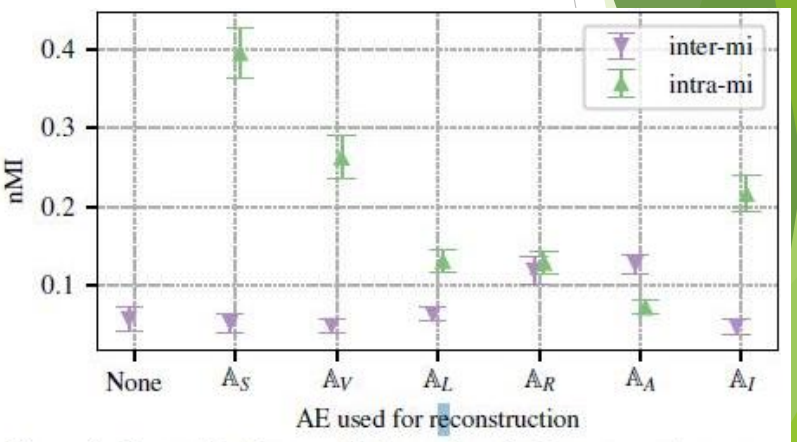


Figure 9. Normalized mutual information for input samples reconstructed from different AEs. *Intra-class* nMI measures information between reconstructions of the same sample

3.6 Compréhension des précédents travaux

On peut expliquer certains résultats d'études :

- Une étude à montré que AlexNet est sensible aux structures locales, ce qui peut-être une conséquence de la part réduite du signal d'entrée qu'il utilise,
- Une autre étude à montré qu'ajouter d'autres filtres au réseau AlexNet ne le rendait pas plus performant, suggérant que la capacité du réseau était déjà suffisante. Les nouveaux travaux permettent de proposer une nouvelle explication : le signal d'entrée est déjà complétement utilisé, c'est-à-dire que les couches plus profondes interprètent déjà tout le signal qui leur est fourni.

Résultats supplémentaires

- La part du signal d'entrée utilisée par les classificateurs est assez faible, par exemple ResNet peut utiliser moins de 10% de l'entrée originale. Cela veut dire que moins d'informations peuvent être utilisées pour faire une prédiction correcte, mais aussi que de petits changements sur des parties importantes peuvent changer le résultat.
- La quantité de signal d'entrée n'est pas corrélée avec le nombre de paramètres ou la performance.
- La partie améliorée du signal d'entrée suit des patterns généraux et distribués, c'est-à-dire que ces patterns ne dépendent pas du contenu spécifique à chaque image (par exemple l'assombrissement des parties claires ou l'augmentation de la distance entre les couleurs).

4 Tableau répertoriant une majorité des méthodes d'interprétation

Name	Ref	Authors	years	Explanator adopted	Black box model	Data used	Problem faced	Generalizable	Code	Notes
Anchors	[99]	Ribeiro	2018	DR	AGN	ANY	Outcome Explanation	Yes	Yes	
LORE	[37]	Guidotti	2018	DR	AGN	TAB	Outcome Explanation	Yes	Yes	
PALM	[58]	Krishnan	2017	DT	AGN	ANY	Model Explanation	Yes		Explanator agnostic
—	[121]	Tolomei	2017	FI	TE	TAB	Model Explanation			
—	[30]	Fong	2017	SM	DNN	IMG	Outcome Explanation			
DTD	[78]	Montavon	2017	SM	DNN	IMG	Outcome Explanation			
DeapLIFT	[107]	Shrikumar	2017	FI	DNN	ANY	Outcome Explanation		Yes	
—	[143]	Zintgraf	2017	SM	DNN	IMG	Outcome Explanation/Model Inspection		Yes	
IG	[115]	Sundararajan	2017	SA	DNN	ANY	Model Inspection			
IP	[108]	Shwartz	2017	AM	DNN	TAB	Model Inspection			
—	[95]	Radford	2017	AM	DNN	TXT	Model Inspection			
—	[143]	Zintgraf	2017	SM	DNN	IMG	Model Inspection		Yes	
1Rule	[75]	Malioutov	2017	DR	—	TAB	Transparent Design			
STA	[140]	Zhou	2016	DT	TE	TAB	Model Explanation			
—	[38]	Hara	2016	DT	TE	TAB	Model Explanation			
TSP	[117]	Tan	2016	DT	TE	TAB	Model Explanation			
MFI	[124]	Vidovic	2016	FI	AGN	TAB	Model Explanation/Outcome Explanation	Yes		
CAM	[139]	Zhou	2016	SM	DNN	IMG	Outcome Explanation		Yes	
Grad-CAM	[106]	Selvaraju	2016	SM	DNN	IMG	Outcome Explanation		Yes	
—	[113]	Sturm	2016	SM	DNN	IMG	Outcome Explanation			
VBP	[11]	Bojarski	2016	SM	DNN	IMG	Outcome Explanation/ Model Inspection			
—	[65]	Lei	2016	SM	DNN	TXT	Outcome Explanation			
LIME	[98]	Ribeiro	2016	FI	AGN	ANY	Outcome Explanation	Yes	Yes	
MES	[122]	Turner	2016	DR	AGN	ANY	Outcome Explanation	Yes		
—	[110]	Singh	2016	DT	AGN	TAB	Outcome Explanation	Yes		
QII	[24]	Datta	2016	SA	AGN	TAB	Model Inspection	Yes		
Prospector	[55]	Krause	2016	PDP	AGN	TAB	Model Inspection	Yes		
Auditing	[2]	Adler	2016	PDP	AGN	TAB	Model Inspection	Yes	Yes	
OPIA	[1]	Adebayo	2016	PDP	AGN	TAB	Model Inspection	Yes		
DGN-AM	[80]	Nguyen	2016	AM	DNN	IMG	Model Inspection		Yes	
—	[72]	Mahendran	2016	AM	DNN	IMG	Model Inspection		Yes	
VBP	[11]	Bojarski	2016	SM	DNN	IMG	Model Inspection			
TreeView	[119]	Thiagarajan	2016	DT	DNN	TAB	Model Inspection			
IDS	[61]	Lakkaraju	2016	DR	—	TAB	Transparent Design			
RuleSet	[130]	Wang	2016	DR	—	TAB	Transparent Design		Yes	
—	[134]	Xu	2015	SM	DNN	IMG	Outcome Explanation		Yes	
PWD	[7]	Bach	2015	SM	DNN	IMG	Outcome Explanation			
ICE	[35]	Goldstein	2015	PDP	AGN	TAB	Model Inspection	Yes	Yes	
—	[136]	Yosinski	2015	AM	DNN	IMG	Model Inspection			
FRL	[127]	Wang	2015	DR	—	TAB	Transparent Design		Yes	
BRL	[66]	Letham	2015	DR	—	TAB	Transparent Design			
TLBR	[114]	Su	2015	DR	—	TAB	Transparent Design			
OT-SpAMs	[128]	Wang	2015	DT	—	TAB	Transparent Design		Yes	
inTrees	[25]	Deng	2014	DR	TE	TAB	Model Explanation			
GoldenEye	[40]	Henelius	2014	FI	AGN	TAB	Model Explanation	Yes	Yes	
—	[39]	Haufe	2014	FI	NLM	TAB	Outcome Explanation			
—	[137]	Zeiler	2014	AM	DNN	IMG	Model Inspection		Yes	
—	[112]	Springenberg	2014	AM	DNN	IMG	Model Inspection			
BCM	[51]	Kim	2014	PS	—	ANY	Transparent Design			
—	[34]	Gibbons	2013	DT	TE	TAB	Model Explanation	Yes		
—	[70]	Lou	2013	FI	AGN	TAB	Model Explanation	Yes	Yes	
—	[109]	Simonian	2013	SM	DNN	IMG	Outcome Explanation			
CP	[64]	Landecker	2013	SM	NN	IMG	Outcome Explanation			
RxREN	[6]	Augasta	2012	DR	NN	TAB	Model Explanation			
VEC	[18]	Cortez	2011	SA	AGN	TAB	Model Inspection	Yes		
PS	[9]	Bien	2011	PS	—	ANY	Transparent Design			
—	[29]	Strumbelj	2010	FI	AGN	TAB	Outcome Explanation	Yes		
GDP	[8]	Baehrens	2010	SA	AGN	TAB	Model Inspection	Yes		
GPDT	[46]	Johansson	2009	DT	NN	TAB	Model Explanation	Yes		
FIRM	[142]	Zien	2009	FI	AGN	TAB	Model Explanation	Yes	Yes	
CDT	[104]	Schetinin	2007	DT	TE	TAB	Model Explanation			
POIMs	[111]	Sonnenburg	2007	FI	SVM	TAB	Model Explanation		Yes	
ExplainD	[89]	Poulin	2006	FI	SVM	TAB	Outcome Explanation			
—	[33]	Fung	2005	DR	SVM	TAB	Model Explanation			
VIN	[42]	Hooker	2004	PDP	AGN	TAB	Model Inspection	Yes		
G-REX	[44]	Johansson	2003	DR	NN	TAB	Model Explanation	Yes		
REFNE	[141]	Zhou	2003	DR	NN	TAB	Model Explanation	Yes		
CPAR	[135]	Yin	2003	DR	—	TAB	Transparent Design			
DecText	[12]	Boz	2002	DT	NN	TAB	Model Explanation	Yes		
SVM+P	[82]	Nunez	2002	DR	SVM	TAB	Model Explanation			
NID	[83]	Olden	2002	SA	NN	TAB	Model Inspection			
—	[57]	Krishnan	1999	DT	NN	TAB	Model Explanation	Yes		
TreeMetrics	[17]	Chipman	1998	DT	TE	TAB	Model Explanation			
CCM	[26]	Domingos	1998	DT	TE	TAB	Model Explanation	Yes		
Trepan	[22]	Craven	1996	DT	NN	TAB	Model Explanation	Yes		
ConjRules	[21]	Craven	1994	DR	NN	TAB	Model Explanation			

Webographie

[WWW1] *Comparison of segmentation and superpixel algorithms*. URL : https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_segmentations.html (visité le 27/11/2019).

[WWW2] *Implementation of the hierarchical graph-based video segmentation algorithm*. URL : https://github.com/davidstutz/hierarchical-graph-based-video-segmentation (visité le 26/03/2020).

Bibliographie

[1] Matthias Grundmann, Vivek Kwatra, Mei Han et Irfan Essa. « Efficient Hierarchical Graph-Based Video Segmentation ». In : (2010). url : http://cpl.cc.gatech.edu/projects/videosegmentation/.

[2] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Dino Pedreschi, Franco Turini et Fosca Giannotti. « Local Rule-Based Explanations of Black Box Decision Systems ». In : *arXiv e-prints*, arXiv :1805.10820 (mai 2018), arXiv :1805.10820. arXiv : 1805.10820 [cs.AI].

[3] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti et Dino Pedreschi. « A Survey of Methods for Explaining Black Box Models ». In : *ACM Comput. Surv.* 51.5 (août 2018), 93 :1-93 :42. issn : 0360-0300. doi : 10.1145/3236009. url : http://doi.acm.org/10.1145/3236009.

[4] Sebastian Palacio, Joachim Folz, Jörn Hees, Federico Raue, Damian Borth et Andreas Dengel. « What do Deep Networks Like to See ? » In : *arXiv e-prints*, arXiv :1803.08337 (mar. 2018), arXiv :1803.08337. arXiv : 1803.08337 [cs.CV].

[5] Dino Pedreschi, Fosca Giannotti, Riccardo Guidotti, Anna Monreale, Luca Pappalardo, Salvatore Ruggieri et Franco Turini. « Open the Black Box Data-Driven Explanation of Black Box Decision Systems ». In : *arXiv e-prints*, arXiv :1806.09936 (juin 2018), arXiv :1806.09936. arXiv : 1806.09936 [cs.AI].

[6] Zhaofan Qiu, Ting Yao et Tao Mei. « Learning Spatio-Temporal Representation with Pseudo-3D Residual Networks ». In : *arXiv e-prints*, arXiv :1711.10305 (nov. 2017), arXiv :1711.10305. arXiv : 1711.10305 [cs.CV].

[7] Marco Tulio Ribeiro, Sameer Singh et Carlos Guestrin. « Anchors : High-Precision Model-Agnostic Explanations ». In : *AAAI*. 2018.

[8] Du Tran, Heng Wang, Lorenzo Torresani, Jamie Ray, Yann LeCun et Manohar Paluri. « A Closer Look at Spatiotemporal Convolutions for Action Recognition ». In : *arXiv e-prints*, arXiv :1711.11248 (nov. 2017), arXiv :1711.11248. arXiv : 1711.11248 [cs.CV].

[9] Marco Tulio Ribeiro, Sameer Singh et Carlos Guestrin. « "Why Should I Trust You ?" : Explaining the Predictions of Any Classifier ». In : *arXiv e-prints*, arXiv :1602.04938 (fév. 2016), arXiv :1602.04938. arXiv : 1602.04938 [cs.LG].

Interprétabilité des IA : application à l'analyse de séquences/trajectoires

Résumé

Ce document est un rapport de PRD du semestre 10. Le sujet du projet concerne l'interprétabilité des IA, et présente des méthodes existantes pour permettre à des êtres humains de comprendre comment une IA prend ses décisions. Il présente aussi l'implémentation de la méthode LIME et comment elle a été modifiée pour être appliquée à des vidéos.

Mots-clés

Interprétabilité, IA, LIME, Réseaux de neurones

Abstract

This document is a report of a PRD of the 10th semester. It deals with the interpretability of the AI, and presents existing methods that help human being to understand how artificial intelligences take decisions. It also describes the implementation of the LIME method, and the modifications made to work with videos.

Keywords

Interpretability, AI, LIME, Neural networks

Tuteur académique
Nicolas Ragot

Étudiant
Romain Hérault (DI5)