

,

Mastère spécialisé Expert en science des données

Projet fil rouge

pour obtenir le diplôme délivré par

Institut national des sciences appliquées

présentée et soutenue par

Assia Dehiles, Florian Dargel et Eliot Farges

le 01/07/2022

Outil de reconnaissance d'ingrédients de cuisine sur une image avec création de recettes associées

Tuteur INSA : Gilles Gasso
INSA Rouen

Table des matières

1	Introduction	5
2	Cahier des charges	6
2.1	Descriptif	6
2.2	Répartition des tâches	6
3	État de l'art	7
3.1	Reconnaissance des ingrédients	7
3.1.1	CNN	9
3.1.2	Transfert learning and fine tuning	11
3.1.3	VGG16 : Architecture proposée pour le transfert learning	12
3.1.4	Modèle de détection : Yolo v3	13
3.2	Génération de recette	16
3.2.1	RNN	16
3.2.2	LSTM	17
3.3	Transformers	18
3.3.1	Architecture encodeur decodeur	19
3.3.2	Mécanisme d'attention	20
3.3.3	GPT2	25
3.3.4	L'architecture	25
3.4	Analyse et clustering des recettes	27
3.4.1	Introduction	27
3.4.2	Text mining	27
3.4.3	Les étapes du traitement de textes	27
3.4.4	Le clustering de documents	28
3.4.5	Algorithme de clustering	30
4	Reconnaissance des ingrédients	32
4.1	Objectif	32
4.2	Préparation de la base de données	32
4.2.1	Base de données kaggle	32
4.2.2	Fusion des images	33
4.3	Modélisation	36
4.3.1	Modèle de classification - multiclass	36
4.3.2	Modèles de détection	44
4.4	Bonus : données supplémentaires	49
4.4.1	Open images dataset V6	49
4.4.2	Scraping sur Google images	51
4.5	Conclusion pour la partie reconnaissance des ingrédients	51
5	Génération de recette	52
5.1	Objectifs	52
5.2	Problème de type Seq to Seq avec utilisation d'un RNN	53
5.2.1	Présentation des données	53
5.2.2	Préparation de la donnée	54

5.2.3	Création et entrainement du modèle	57
5.3	Génération de texte conditionné avec fine-tuning d'un modèle existant	62
5.3.1	Methode	63
5.3.2	Résultats	63
6	Analyse de text et clustering	65
6.1	Objectifs	65
6.2	Les données	65
6.3	Préparation des données	65
6.3.1	Nettoyage et prétraitement	65
6.3.2	Tokenization	66
6.3.3	Normalisation du lexique - Le Stemming	67
6.3.4	Lematization	67
6.3.5	Le POS-TAGGING	67
6.4	Analyse de la composition de recettes	68
6.5	Le clustering	69
6.5.1	Extraction de features - Vectorisation tf-idf	69
6.5.2	K-means	70
6.5.3	Conclusions de la classification	74
7	Conclusion du projet	76
7.1	Pistes d'amélioration	76

Table des figures

1	A simple CNN architecture	9
2	une analyse du processus de convolution	10
3	Transfert learning architecture en utilisant VGG16	13
4	Boîtes englobantes avec cotes a priori et emplacement.	14
5	Architecture Darknet-53	16
6	Architecture RNN	17
7	Différence RNN et réseaux classiques	17
8	Noyau LSTM et GRU	18
9	Architecture transformers	19
10	Architecture encodeur décodeur 1	20
11	Architecture encodeur décodeur 2	21
12	Un exemple du mécanisme d'attention qui suit les dépendances à longue distance dans l'attention autonome de l'encodeur dans la couche 5 sur 6. De nombreuses têtes d'attention s'occupent d'une dépendance à distance du verbe 'making', complétant la phrase 'rendre... plus difficile'. Les attentions sont montrées ici uniquement pour le mot 'faire'. Les différentes couleurs représentent les différentes têtes.	23
13	Exemple d'image : une pomme	32
14	Remplacement de l'image	33
15	Fusion des images	33
16	Fusion des aliments	34
17	l'arborescence des dossiers	35
18	Données d'entraînement équilibrées : 100 images par classe	36
19	Architecture du réseau	37
20	Représentation graphique : L'entraînement du modèle	37
21	Représentation graphique : Train and validation accuracy Train and validation loss	38
22	Évaluation du modèle sur la validation et le test	39
23	Prédiction sur une image de test	39
24	Architecture du VGG16	40
25	Modification de la dernière couche	40
26	Entrainement du modèle VGG16	41
27	Représentation graphique : Train and validation accuracy Train and validation loss	41
28	Evalution du modèle VGG16 sur le jeu de donnée test et validation	42
29	Prédiction sur une image du jeu de données test	42
30	La probabilité d'appartenir à la classe	43
31	Une image de voiture	44
32	Détection avec le modèle Yolov3	47
33	Les informations sur le modèle ssdlite_mobilnetv2	48
34	Détections des ingrédients avec le modèle ssdlite_mobilnet_v2	48
35	Comparaison entre ssdlite_mobilenet_V2 et Yolo v3	49
36	Open Images Dataset V6 + Extensions	50
37	Les ingrédients : Open Images Dataset V6	50

38	RNN au niveau des caractères	53
39	Appercu du jeu de données	54
40	Tokenization des recettes	56
41	Présentation du modèle LSTM 1	58
42	Présentation du modèle LSTM 2	58
43	Entrainement du modèle	59
44	Dataframe 'ingrédients'	66
45	Liste et fréquences des 10 tokens les plus fréquents	66
46	Résultat du POS-Tagging	68
47	Fréquences des tokens dans les recettes	68
48	Matrice des données pondérées par tf-idf	69
49	Inertie par nombre de clusters	71
50	Silhouette pour 17 clusters	72
51	Descriptif des 17 clusters	73
52	Dendrogramme de la classification hiérarchique des recettes	74

1 Introduction

Nous sommes trois étudiants en mastère spécialisé expert en science des données à l'INSA de Rouen. Pendant notre année scolaire, nous avons 80 heures pour mener un projet dont nous avons eu l'idée. Un tuteur nous guide dans l'avancée de notre projet.

Nous mettons en application nos connaissances, et découvrons d'autres méthodes et technologies non vues en cours. Nous nous familiarisons avec la gestion de projet, laquelle nous risquons fortement de retrouver en entreprise. Ce rapport est le compte rendu de ce projet.

2 Cahier des charges

2.1 Descriptif

On se met dans une situation où un individu a des ingrédients en sa possession et il souhaite réaliser une nouvelle recette inexistante. Ceci est intéressant, car une nouvelle recette peut nous donner un plat qui n'a jamais été cuisiné et peut être délicieux. Pour cela on va construire un modèle qui prend en entrée une image qui contient les ingrédients et qui nous donne en sortie les ingrédients présents sur l'image qui nous servira après pour générer une nouvelle recette que la personne pourra cuisiner. On a aussi pour but d'étudier les compositions possibles entre les différents ingrédients qui constituent une recette et de regarder les similarités entre les recettes.

2.2 Répartition des tâches

Nous avons extrait trois grands axes de travail autour des trois composants principaux :

- Création de la base, détection ou classification d'aliments (Florian)
- La génération de recette (Eliot)
- Une partie d'analyse statistique des recettes (Assia)

Voici le catalogue des tâches que nous nous sommes fixées à réaliser :

Catalogue des tâches	
Missions	Réalisations
Création de la base et détection ou classification d'aliments	
Capturer des images	✓
Nettoyage et fusion des images	✓
Modèles de classification	✓
Modèles de détection	✓
Génération de recette	
Recherche d'un bon dataset	✓
Bien formuler le problème pour trouver des modèles adaptés	✓
Nettoyage des données	✓
Mise en œuvre des modèles (LSTM et fine tuning GPT2)	✓
Analyse et clustering des recettes	
Nettoyage des données	✓
Formatages et construction des datasets	✓
Nettoyage et extraction de features	✓
Analyse exploratoire des données	✓
Test Kmeans	✓
Test algorithmes supplémentaires et comparaison	✓

3 État de l'art

3.1 Reconnaissance des ingrédients

L'avènement de la technologie d'apprentissage profond a stimulé de nombreux domaines scientifiques en fournissant des méthodes et des techniques avancées pour une meilleure pré-diction et reconnaissance des objets, à partir d'images ou de vidéos. En informatique, et plus particulièrement en vision par ordinateur et en intelligence artificielle, la classification d'images est une tâche essentielle, avec de nombreuses avancées récentes provenant de la reconnaissance d'objets avec des approches d'apprentissage profond (Krizhevsky, Sutskever, et Hinton, 2012 ; Simonyan et Zisserman, 2015 ; He, Zhang, Ren, et Sun, 2016 ; Szegedy, et al., 2015 ; Szegedy, Vanhoucke, Ioffe, Shlens, et Wojna, 2016 ; Szegedy, Ioffe, Vanhoucke, et Alemi, 2017).

La nourriture, qui constitue une partie importante de la vie quotidienne, constitue un défi particulier dans le domaine de la classification d'images, en raison de sa complexité visuelle, ainsi que de la complexité sémantique découlant de la variation du mélange de divers ingrédients pratiqué par les communautés régionales.

Il a été prouvé que ce défi présente de nombreux aspects complexes (Weiqing, Shuqiang, Linhu, Yong, et Ramesh, 2019 ; Mezgec et Koroušić, 2017). L'abondance d'images de nourriture fournies par les réseaux sociaux, les sites de partage de photos dédiés, les applications mobiles et les puissants moteurs de recherche (Mezgec et Koroušić, 2017) est considéré comme un moyen facile de développer de nouveaux ensembles de données à des fins scientifiques. Selon la communauté scientifique, la reconnaissance automatique (classification) des plats n'aiderait pas seulement les gens à organiser sans effort leurs énormes collections de photos et à rendre leur contenu plus accessible, mais elle permettrait également d'estimer et de suivre les habitudes alimentaires quotidiennes et l'apport calorique et de planifier des stratégies nutritionnelles même en dehors d'un environnement clinique contraignant (Mezgec Koroušić, 2017). Malgré les dizaines d'applications, d'algorithmes et de systèmes disponibles, le problème de la reconnaissance de plats (aliments) et leurs ingrédients n'a pas été entièrement traité par les communautés de l'apprentissage automatique et de la vision par ordinateur (Weiqing, Shuqiang, Linhu, Yong, et Ramesh, 2019). Cela est dû à l'absence de disposition distinctive et spatiale des images d'aliments, que l'on trouve généralement dans les images représentant des scènes ou des objets. Par exemple, une image d'une scène extérieure peut être typiquement décomposée en (a) un lieu au sol, (b) un horizon, (c) une forêt, et (d) le ciel. De tels motifs ne peuvent être trouvés dans les images d'aliments. Les ingrédients alimentaires, comme ceux que l'on trouve dans une salade, sont des mélanges qui se présentent fréquemment sous des formes et des tailles différentes, dépendant beaucoup des pratiques régionales et des habitudes culturelles. Il convient de souligner que la nature des plats est souvent définie par les différentes couleurs, formes et textures des divers ingrédients (Ciocca, Napoletano, et Schettini, 2018). Néanmoins, le type de caractéristiques qui décrivent les aliments dans la plupart des cas sont facilement reconnaissables par l'homme sur une seule image, quelles que soient les variations géométriques des ingrédients. Par conséquent, on peut considérer que la reconnaissance des aliments est un problème de classification spécifique et très complexe qui exige le développement de modèles capables d'exploiter les informations locales (caractéristiques) dans les images, ainsi que des modèles complexes de plus haut niveau. La reconnaissance des ali-

ments reste un problème difficile, problème qui suscite l'intérêt de la communauté scientifique (Ciocca, Napoletano, et Schettini, 2018 ; He, Zhang, Ren, Sun, 2016).

La reconnaissance des aliments s'est épanouie grâce à l'amélioration des performances de calcul et aux progrès de la vision par ordinateur et de l'apprentissage automatique au cours de la dernière décennie. En 2012, Matsuda et al. ont obtenu des taux d'accuracy de 55,8 % pour les images d'aliments à articles multiples et de 68,9 % pour les images d'aliments à article unique en utilisant deux méthodes différentes (Matsuda, Hoashi et Yanai, 2012). La première méthode était basée sur le modèle de parties déformables de Felzenszwalb et la seconde était basée sur la fusion de caractéristiques. En 2014, l'un des premiers travaux employant l'apprentissage profond a été développé (Kawano et Yanai, 2014b). Les chercheurs ont obtenu une précision de 72,26 % avec un modèle pré-entraîné (apprentissage par transfert) similaire à AlexNet (Krizhevsky, Sutskever et Hinton, 2012). À la même période, Bossard et al. ont obtenu un résultat de classification de 50,76 % avec l'utilisation de la forêt aléatoire sur le jeu de données Food101 (Bossard, Guillaumin, Van Gool, 2014). Les chercheurs ont noté que la méthode de la forêt aléatoire ne peut pas surpasser les approches d'apprentissage profond. Des conclusions similaires ont été tirées par Kagaya et al. qui ont rapporté un résultat de classification de 73,70 % en utilisant des réseaux de neurones convolutifs (CNN) profonds (Kagaya, Aizawa, et Ogawa, 2014). L'année suivante, les CNN profonds, en combinaison avec l'apprentissage par transfert, ont atteint 78,77 % (Yanai Kawano 2015). Christodoulidis et al. ont présenté une nouvelle méthode sur les CNN profonds atteignant un accuracy de 84,90 % sur un jeu de données personnalisé (Christodoulidis, Anthimopoulos, Mougiakakou, 2015). En 2016, Singla et al. ont adopté l'architecture GoogLeNet (Szegedy, et al., 2015) et, avec un modèle pré-entraîné, ils ont atteint un accuracy de 83,60 % (Singla, Yuan et Ebrahimi, 2016). Lui et al. ont développé DeepFood en 2016 et ont obtenu des résultats similaires en utilisant des techniques de convolution optimisées dans une version modifiée de l'Inception (Liu, et al., 2016). Les chercheurs ont obtenu un accuracy de 76,30 % sur le jeu de données UEC-Food100 un de 54,70 % sur le jeu de données UEC-Food256 et un accuracy de 77,40 % sur le jeu de données Food-101. Hassannejad et al. ont obtenu un accuracy de 81,45 % sur le jeu de données UEC-Food100, 76,17 % sur le jeu de données UEC-Food256 et 88,28 % sur le jeu de données Food-101 (Hassannejad, et al., 2016), en utilisant l'architecture de reconnaissance d'images de Google nommée Inception V3 (Szegedy, Vanhoucke, Ioffe, Shlens, et Wojna, 2016).

En 2017, Ciocca et al. ont présenté une nouvelle méthode qui combinait des techniques de segmentation et de reconnaissance basées sur des CNN profonds sur un nouveau jeu de données atteignant un accuracy de 78,30 % (Ciocca, Napoletano, et Schettini, 2017a). Mezgec et al. (2017) ont adopté une modification du populaire AlexNet et ont introduit le NutriNet. le NutriNet, qui a été entraîné avec des images acquises à l'aide de moteurs de recherche sur le Web. Ils ont atteint une performance de classification de 86,72 % sur 520 classes d'aliments et de boissons. Le NutriNet utilise moins de paramètres par rapport à la structure originale de l'AlexNet. En 2018, Ciocca et al. ont présenté un nouveau jeu de données qui a été acquis à partir de la fusion d'autres jeux de données (Ciocca, Napoletano, Schettini, 2018). Ils ont testé plusieurs architectures populaires sur le jeu de données, notamment un réseau résiduel (He, Zhang, Ren, Sun, 2016) avec 50 couches, en définissant ce dernier comme l'architecture de référence. Apparemment, de nombreux groupes de recherche ont travaillé activement sur le thème de la reconnaissance des aliments, en utilisant une variété de méthodes et de techniques.

Globalement, les méthodes les plus efficaces sont les variantes des approches d'apprentissage profond.

3.1.1 CNN

Les systèmes de reconnaissance des aliments les plus récents sont développés à partir d'architectures de réseaux de neurones convolutifs profonds (Ciocca, Napoletano, et Schettini, 2017b ; Ciocca, Napoletano, Schettini, CNN-based features for retrieval and classification of food images, 2018 ; Horiguchi, Amano, Ogawa et Aizawa, 2018 ; Yu, Anzawa, Amano, Ogawa et Aizawa, 2018 ; Weiqing, Shuqiang, Linhu, Yong et Ramesh, 2019). Ainsi, cette section présente quelques techniques et méthodes "incontournables" pour le développement et l'utilisation d'un modèle CNN efficace de reconnaissance des aliments.

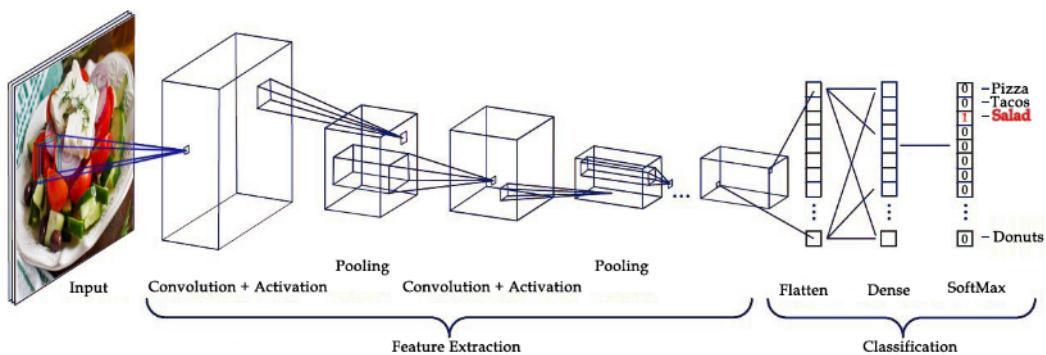


Figure 1. A simple CNN architecture

FIGURE 1 – A simple CNN architecture

Un CNN est un réseau neuronal multicouche doté d'une architecture unique conçue pour extraire des caractéristiques de plus en plus complexes des données à chaque couche, afin de déterminer correctement la sortie. Les CNN sont bien adaptés aux tâches perceptuelles. La figure 1 illustre une architecture CNN simple avec deux couches de convolution et deux couches de mise en commun au niveau de l'extraction des caractéristiques du réseau. Le niveau de classification du réseau (couches supérieures) est composé de trois couches, (a) une couche d'aplatissement, qui transforme les cartes de caractéristiques multiples, à savoir un tableau multidimensionnel (tenseur) de la dernière couche de convolution en un tableau unidimensionnel, (b) une couche dense et (c) une couche de prédiction. La sortie de la couche de prédiction est, dans la plupart des cas, codée à un coup, ce qui signifie que la sortie est une représentation vectorielle binaire des classes catégorielles. La cellule de ce vecteur qui est déclenché (est vraie ou "1") montre la prédiction du CNN, ainsi, chaque classe est représentée par un vecteur de "0" et un seul "1", dont la position détermine la classe.

Un CNN est principalement utilisé lorsqu'il existe un ensemble de données non structurées (par exemple, des images) et que le modèle doit en extraire des informations. Par exemple, si la tâche consiste à prédire la légende d'une image.

Le fonctionnement et l'apprentissage d'un CNN sont résumés par les points suivants :

- Le CNN reçoit une image couleur (par exemple une image de salade) sous la forme d'une

matrice 3D de pixels (trois matrices d'images d'intensité 2D).

- Pendant l'apprentissage, les couches cachées permettent au CNN d'identifier des caractéristiques uniques en formant des filtres appropriés qui extraient l'information des images.
- Lorsque le processus d'apprentissage du réseau converge, le CNN est alors capable de fournir une prédition concernant la classe à laquelle une image appartient.
- L'entrée d'un CNN est généralement un tenseur. Un tenseur est un tableau multidimensionnel contenant des données pour la validation de la formation du CNN. Une entrée de tenseur peut être représentée par (i, h, w, c) , où : i est le nombre d'images, h est la hauteur de l'image, w est la largeur de l'image et c est le nombre de canaux dans l'image. Pour exemple, $(64, 229, 229, 3)$ représente un lot de 64 images de 229×229 pixels et 3 canaux de couleur (RVB).
- Un noyau (également appelé filtre et détecteur de caractéristiques) est un petit tableau à valeur fixe. La convolution de ce tableau avec le tenseur d'entrée révèle les caractéristiques visuelles (telles que les bords). Le noyau se déplace avec des pas et une direction spécifiques pour couvrir la surface entière de chaque image. Ces étapes sont appelées strides. Un pas indique le nombre de pixels que le noyau déplace après chaque opération.
- La carte de caractéristiques de sortie est le groupe de caractéristiques extraites par le filtre (noyau) à partir de l'entrée et est pratiquement une forme transformée de la carte de caractéristiques d'entrée.

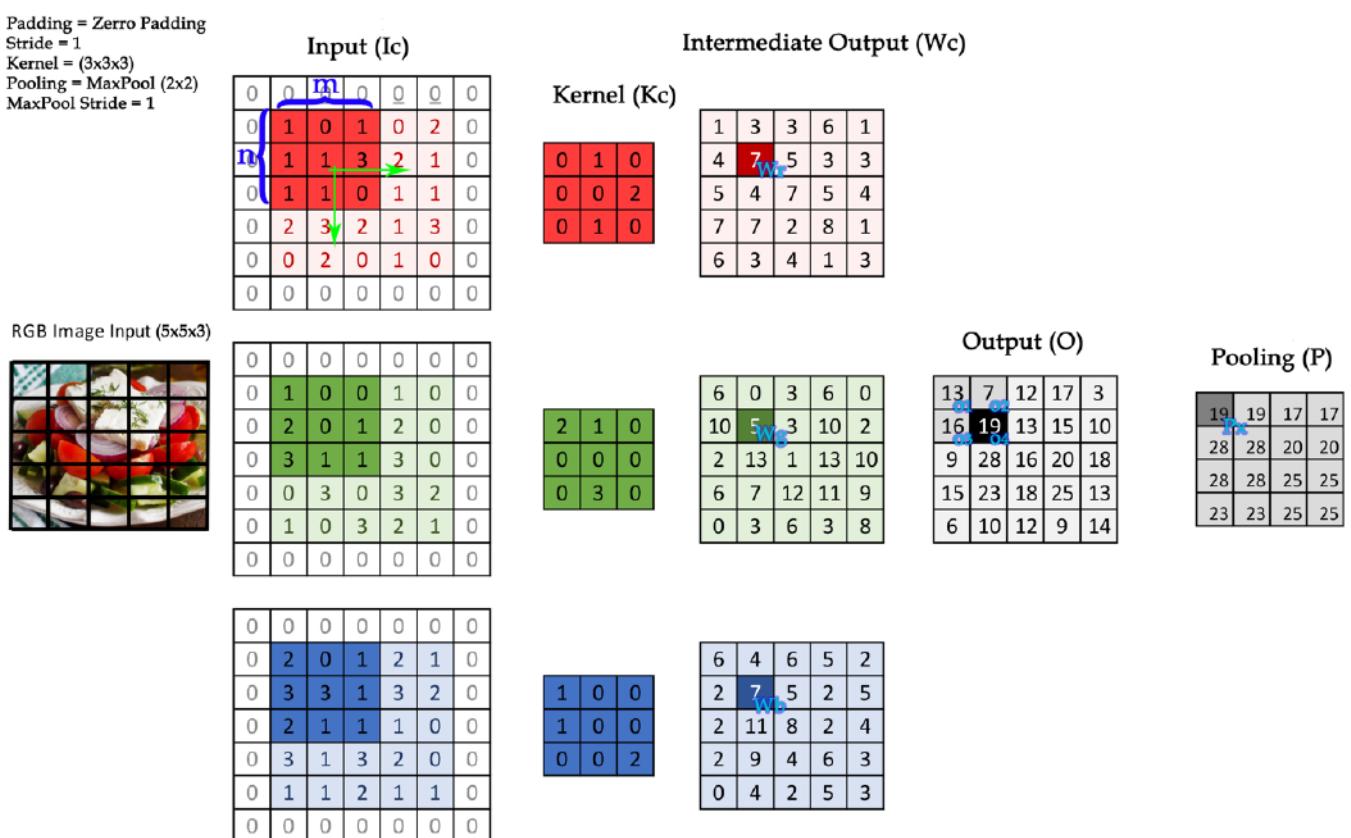


FIGURE 2 – une analyse du processus de convolution
[Kiourt et al. (2020)]

La figure 2 illustre le processus complet (une seule étape) pour une image RVB, sur $5 \times 5 \times$

3 blocs d'images. Pour chaque canal de couleur du tenseur d'entrée, une sortie intermédiaire (\mathbf{W}_c) est calculé par :

$$\mathbf{W}_c = \mathbf{K}_c * \mathbf{I}_{c(m,n)}$$

où $*$ est le produit de convolution, \mathbf{K}_c le noyau de convolution et $\mathbf{I}_{c(m,n)}$ est le bloc d'images d'entrée. La carte de caractéristiques finale \mathbf{O} est calculée en additionnant les sorties intermédiaires (\mathbf{W}_c) de chaque canal :

$$O = \sum_c \mathbf{W}_c$$

Afin de réduire la taille des cartes de caractéristiques, la méthode Pooling est couramment utilisée. Pooling réduit la taille d'une carte de caractéristiques en utilisant certaines fonctions pour résumer les sous-régions. Les fonctions typiques utilisées pour cette tâche sont le maximum (max pooling) ou la moyenne (average pooling) d'une sous-région (ou sous-carte) x de la carte de caractéristiques. Dans l'exemple de la figure, on utilise le max pooling, qui s'exprime comme suit :

$$\mathbf{O}_x = \begin{bmatrix} o_{1,1} & o_{1,2} \\ o_{2,1} & o_{2,2} \end{bmatrix}, \quad \mathbf{P}_x = \max_{i,j=1,2} o_x^{i,j} \quad (1)$$

où $o_x^{i,j}$ désigne le i-ème, j-ème élément de la sous-région \mathbf{O}_x . En général, la fenêtre de Pooling (sous-région de la carte des caractéristiques) se déplace avec un stride spécifique.

3.1.2 Transfert learning and fine tunining

Une voie alternative dans les approches de reconnaissance alimentaire basées sur l'apprentissage profond repose sur l'utilisation de modèles populaires d'apprentissage profond pré-entraînés, en mettant en œuvre la technique d'apprentissage par transfert. L'apprentissage par transfert dans la théorie de l'apprentissage profond est une technique populaire d'apprentissage automatique, dans laquelle un modèle développé et formé pour une tâche spécifique est réutilisé pour une autre tâche similaire avec un certain paramétrage (réglage fin). Une définition intuitive de l'apprentissage par transfert a été fournie par Torrey et Shavlink en (2009) : " L'apprentissage par transfert est l'amélioration de l'apprentissage d'une nouvelle tâche par le transfert des connaissances d'une tâche connexe déjà apprise ". Fine tuning est le processus dans lequel un modèle pré-entraîné est utilisé sans ses couches supérieures, qui sont remplacées par de nouvelles couches, plus appropriées à la nouvelle tâche (ensemble de données) et seules les couches convolutionnelles doivent être conservées avec leurs poids. De nouvelles couches supérieures doivent être ajoutées en fonction des besoins de la nouvelle tâche (nouveau jeu de données), et le nombre de sorties de la couche de prédiction doit être associé au nombre de nouvelles classes. Il est courant d'effectuer une recombinaison appropriée des couches supérieures, une reconfiguration des paramètres et de nombreuses répétitions afin d'obtenir une performance souhaitable.

En raison de la facilité d'utilisation de l'apprentissage par transfert, de nos jours, très peu de personnes développent et entraînent de nouveaux modèles (architectures) CNN (architectures) à partir de zéro. Cela est dû au fait qu'il est difficile de disposer d'un ensemble

de données de taille suffisante pour la tâche spécifique. De plus, comme nous l'avons déjà mentionné, le développement d'un nouveau modèle est complexe et prend beaucoup de temps. Il est donc plus pratique de réutiliser un modèle qui a été entraîné sur un très grand ensemble de données, comme ImageNet (Zeiler, 2013) qui contient plus de 14 millions d'images couvrant plus de 20 000 catégories. Notez également qu'ImageNet contient un grand nombre de catégories d'images représentant des aliments ou des articles comestibles en général. Par conséquent, un modèle pré-entraîné sur le jeu de données ImageNet devrait être assez efficace et facile à ajuster pour améliorer les performances des applications de reconnaissance des aliments.

3.1.3 VGG16 : Architecture proposée pour le transfert learning

VGG16 est considéré comme l'architecture d'apprentissage profond la plus courante. Il est possible d'avoir un CNN profond avec apprentissage par transfert utilisant le modèle VGG16 sans la couche supérieure. D'après l'étude de Jasman Pardede.al (Implementation of Transfer Learning Using VGG16 on Fruit Ripeness Detection), la couche supérieure du VGG16 a été remplacée par l'ajout d'un bloc de perceptron multicouches (MLP). VGG16 a été entraînés sur plus d'un million d'images provenant de la base de données ImageNet (ici les poids d'Imagenet sont utilisés). La pondération du modèle VGG16 utilisée pour l'extraction des caractéristiques des données sur la maturité des fruits...

Les résultats de l'extraction de caractéristiques du modèle VGG16 alimentent le bloc MLP. Ainsi, les caractéristiques extraites utilisées dans le transfert learning utilisent les caractéristiques extraites du VGG16, tandis que le classificateur de l'architecture proposée utilise le bloc MLP avec le classificateur à activation softmax. Les blocs MLP contiennent une couche d'aplatissement (flatten layer), une couche dense (dense layer) et une couche de régularisation. Les régularisations utilisées pour obtenir les meilleures performances de classification du système (en réduisant le sur ajustement) sont le Dropout, le Batch, la normalisation et les régularisations du noyau. La fonction d'activation utilisée dans le bloc MLP est l'activation ReLu. La couche de sortie du bloc MLP utilise la fonction d'activation softmax avec 8 (huit) classes. L'architecture du système utilisé pour la classification de la maturité des fruits est présentée dans la figure ci-dessous.

Les étapes de la formation du modèle sont les suivantes :

- Générer des données d'image de formation en utilisant le processus d'augmentation des données. Alors que la validation et le test des données n'utilisent pas le processus d'augmentation des données
- Extraction de caractéristiques pour chaque image (formation, validation et test) en utilisant l'extraction de caractéristiques du modèle VGG16 avec le poids d'ImageNet.
- Transfère des résultats de l'extraction des caractéristiques du modèle VGG16 au régulateur (Dropout, Batch Normalization et noyau de régularisation).
- Transfère des résultats de régulateur vers la couche d'aplatissement (flatten layer).
- Transfère des résultats de la couche Flatten à la couche Dense et utilisation l'activation ReLu.
- Transfère les résultats de la couche dense à la couche dense en utilisant la fonction d'activation softmax avec 8 classes (pomme mûre, mangue mûre, orange mûre, tomate mûre, pomme non mûre, mangue non mûre, orange non mûre, tomate non mûre)

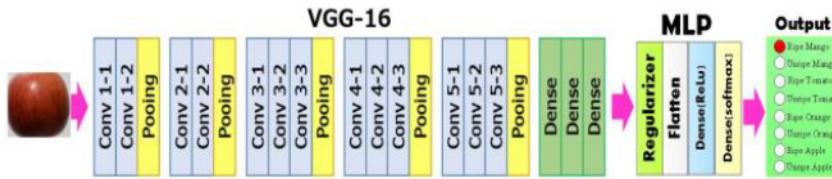


FIGURE 3 – Transfert learning architecture en utilisant VGG16

Dans cette étude, les chercheurs ont comparé les performances des régularisations afin de réduire l’overfitting. Les régularisations utilisées dans l’architecture d’apprentissage par transfert VGG16 proposée sont le Dropout, la normalisation par lots et le noyau de régularisation. La meilleure accuracy du système de l’architecture d’apprentissage par transfert VGG16 avec Dropout, Batch Normalization, et Regularizer kernel est de 0.90, 0.84, et 0.76,

D’après les résultats expérimentaux obtenus, la meilleure valeur d’accuracy pour l’architecture utilisant le régulateur Dropout 0,5 est de 90%, avec une valeur moyenne de précision de 0,90. Le rappel moyen et la F-mesure à Dropout 0,5 sont respectivement de 0,90 et 0,90.

3.1.4 Modèle de détection : Yolo v3

À l’époque, YOLO 9000 était l’algorithme le plus rapide et l’un des plus précis. Cependant, quelques années plus tard, il n’est plus le plus précis, des algorithmes comme RetinaNet et SSD le surpassant en termes de précision. Cependant, il était toujours l’un des plus rapides.

Mais cette vitesse a été échangée contre une augmentation de la précision dans la version 3 de YOLO. Alors que la version précédente tournait à 45 FPS(Frame Per Second : définit la vitesse à laquelle votre modèle de détection d’objets traite votre vidéo et génère le résultat souhaité) sur un Titan X, la version actuelle tourne à environ 30 FPS. Cela est dû à l’augmentation de la complexité de l’architecture sous-jacente appelée Darknet.

Boites englobantes

À l’instar de YOLO9000, le système prédit des boîtes englobantes en utilisant des groupes de dimensions comme boîtes d’englobantes [J. Redmon and A. Farhadi. (2017)]. Le réseau prédit 4 coordonnées pour chaque boîte englobante, t_x, t_y, t_w, t_h . Si la cellule est décalée du coin supérieur gauche de l’image (c_x, c_y) et la boîte englobante antérieure a une largeur et une hauteur p_w, p_h , alors les prédictions correspondent à :

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned}$$

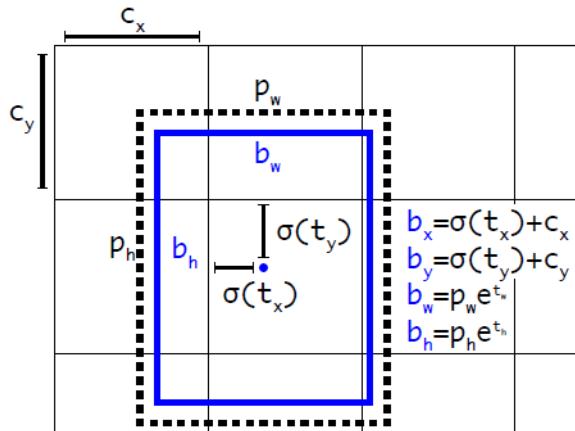


FIGURE 4 – Boîtes englobantes avec cotes a priori et emplacement.

Nous prédisons la largeur et la hauteur de la boîte sous forme de décalages à partir des centres de gravité des clusters. Nous prédisons les coordonnées du centre de boîte relative à l'emplacement de l'application du filtre à l'aide d'un sigmoïde fonction. Cette figure est un auto-plagiat flagrant de [J. Redmon and A. Farhadi(2017)]

Pendant la formation, nous utilisons la perte de la somme des erreurs quadratiques. Si la vérité terrain pour une certaine prédiction de coordonnées est \hat{t}_* , notre gradient est la valeur de vérité terrain (calculée à partir de la boîte de vérité terrain) moins notre prédiction : $\hat{t}_* - t_*$.

YOLOv3 prédit un score d'objet pour chaque boîte de délimitation en utilisant une régression logistique. Ce score doit être égal à 1 si la boîte englobante chevauche un objet de vérité terrain plus que tout autre antécédent de boîte englobante. Si l'antériorité de la boîte englobante n'est pas la meilleure, mais qu'elle chevauche un objet de vérité terrain de plus d'un certain seuil, nous ignorons la prédiction, en suivant les règles suivantes [S. Ren, K. He, R. Girshick, and J. Sun (2015)]. On utilise le seuil de .5. Contrairement à [S. Ren, K. He, R. Girshick, and J. Sun (2015)], le système n'attribue qu'un seul antécédent de boîte englobante pour chaque objet de vérité terrain. Si un antécédent de boîte englobante n'est pas attribué à un objet de vérité terrain, il ne subit aucune perte pour les prédictions de coordonnées ou de classe, mais uniquement pour l'objet.

Classe prédite

Chaque boîte prédit les classes que la boîte englobante peut contenir en utilisant la classification multilabel. Le softmax n'est pas utilisé, cela n'est pas nécessaire pour de bonnes performances, à la place, ce qui est utilisé est simplement des classificateurs logistiques indépendants. Durant formation, la perte d'entropie croisée binaire est utilisé pour la classe prédiction.

Cette formulation aide lorsque nous passons à des domaines comme l'ensemble de données Open Images [I. Krasin, T. Duerig, N. Alldrin et al. (2017)]. Dans cet ensemble de données, il existe de nombreuses étiquettes qui se chevauchent (c'est-à-dire pomme et fruit). L'utilisation d'un softmax impose l'hypothèse que chaque boîte a exactement une classe, ce qui n'est souvent pas le cas. Une approche multi-label modélise mieux les données.

Prédictions à travers les échelles

YOLOv3 prédit les boîtes à 3 échelles différentes. Le système extrait les caractéristiques de ces échelles en utilisant un concept similaire pour présenter des réseaux pyramidaux[T.-Y. Lin, P. Dollar, R. Girshick et al. (2017)]. À partir de notre extracteur de caractéristiques de base nous ajoutons plusieurs couches convolutionnelles. La dernière de ces couches prédit un tenseur 3-d encodant la boîte englobante, l'objet et les prédictions de classe. Dans leurs expériences avec COCO [T.-Y. Lin, M. Maire, S. Belongie et al. (2014)] ils prédisent 3 boîtes à chaque échelle, le tenseur est donc de $N \times N[3 * (4 + 1 + 80)]$ pour les 4 décalages de boîtes englobantes, 1 prédiction d'objet, et 80 prédictions de classe.

Ensuite, ils prennent la carte des caractéristiques des deux couches précédentes et l'échantillonnent par deux. Ils prennent également une carte de caractéristiques d'une partie antérieure dans le réseau et ils la fusionnent avec leurs caractéristiques sur-échantillonées en utilisant la concaténation. Cette méthode leur permet d'obtenir plus informations sémantiques plus significatives à partir des caractéristiques suréchantillonées et des informations plus fines à partir de la carte de caractéristiques précédente. Ils ajoutent ensuite quelques couches convolutives supplémentaires pour traiter cette carte de caractéristiques combinée, et finalement prédire un tenseur similaire, bien qu'il soit maintenant deux fois plus grand.

Ils effectuent la même conception une fois de plus pour prédire les cases pour l'échelle finale. Ainsi, les prédictions pour la 3e échelle bénéficient de tous les calculs antérieurs, ainsi que des caractéristiques à grain fin des premières étapes du réseau.

Ils utilisent toujours le clustering k-means pour déterminer les boîte englobante. Ils ont en quelque sorte choisi 9 groupes et 3 échelles de manière arbitraire, puis ils ont divisé les clusters de manière égale entre les échelles. Sur le jeu de données COCO, les 9 clusters étaient les suivants : (10×13) , (16×30) , (33×23) , (30×61) , (62×45) , (59×119) , (116×90) , (156×198) , (373×326) .

Les caractéristiques extraites

J. Redmon and A. Farhadi utilisent un nouveau réseau pour effectuer l'extraction de caractéristiques.

Le nouveau réseau est une approche hybride entre le réseau utilisé dans YOLOv2, Darknet-19, et ce nouveau réseau résiduel. Le réseau utilise successivement 3×3 et 1×1 couche convulsive, mais a maintenant des raccourcis de connexion et il est beaucoup plus grand. Il y a 53 couches de convolutions, ils appellent le réseau Darknet-53.

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32×1	
1x	Convolutional	64×3	
1x	Residual		128×128
2x	Convolutional	$128 \times 3 / 2$	64×64
2x	Convolutional	64×1	
2x	Convolutional	128×3	
2x	Residual		64×64
8x	Convolutional	$256 \times 3 / 2$	32×32
8x	Convolutional	128×1	
8x	Convolutional	256×3	
8x	Residual		32×32
8x	Convolutional	$512 \times 3 / 2$	16×16
8x	Convolutional	256×1	
8x	Convolutional	512×3	
8x	Residual		16×16
4x	Convolutional	$1024 \times 3 / 2$	8×8
4x	Convolutional	512×1	
4x	Convolutional	1024×3	
4x	Residual		8×8
	Avgpool	Global	
	Connected	1000	
	Softmax		

Table 1. Darknet-53.

FIGURE 5 – Architecture Darknet-53

Ce nouveau réseau est beaucoup plus puissant que Darknet-19 [J. Redmon and A. Farhadi (2017)]. Darknet-53 est meilleur que ResNet-101 [K. He, X. Zhang, S. Ren, and J. Sun (2016)] et $1.5 \times$ plus rapide. Darknet-53 a des performances similaires à ResNet-152 [K. He, X. Zhang, S. Ren, and J. Sun (2016)] et est 2 fois plus rapide.

3.2 Génération de recette

3.2.1 RNN

le réseau neuronal récurrent (RNN) est une classe de réseaux neuronaux profonds, le plus souvent appliquée à des données basées sur des séquences comme la parole, la voix, le texte ou la musique. Ils sont utilisés pour la traduction automatique, la reconnaissance vocale, la synthèse vocale, etc. La principale caractéristique des RNN est qu'ils ont une mémoire interne dans laquelle un certain contexte pour la séquence peut être stocké. Par exemple, si le premier mot de la séquence était "He", le RNN pourrait suggérer le mot suivant "speaks" au lieu de "speak" (pour former une phrase "He speaks"), car la connaissance préalable du premier mot "He" se trouve déjà dans la mémoire interne.

Les RNN utilisent les sorties précédentes comme entrées supplémentaires et sont parfaitement adaptés au traitement de données séquentielles. Généralement, elles se présentent sous la forme suivante :

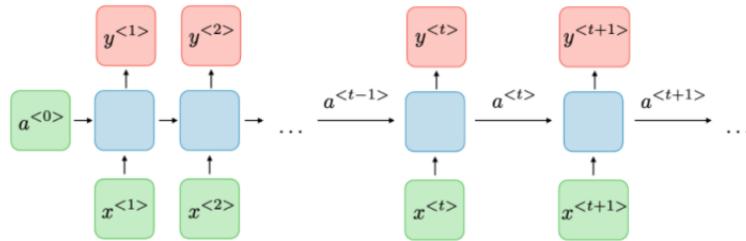


FIGURE 6 – Architecture RNN

À chaque instant t , le passage vers l'avant (forward pass) est modélisé par les équations suivantes :

$$(1) a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$(2) y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

Où x_t et a_t sont respectivement le vecteur d'entrée du réseau et le vecteur d'activation à l'instant t , $g1$, $g2$ sont des fonctions d'activation, les W et b sont respectivement les poids et les biais à apprendre durant l'entraînement du réseau.

La valeur de sortie à l'instant t $y^{<t>}$ est calculée par l'équation (2) en fonction de la valeur d'activation $a^{<t>}$ calculée par l'équation (1).

Nous constatons clairement l'aspect récurrent dans ces calculs (le calcul à l'instant t est à base de l'information apportée de l'instant $t-1$, elle-même calculée à partir de l'information apportée de $t-2$ etc.), contrairement à un réseau de neurones classique ANN (Artificial Neural Network) où la sortie dépend uniquement des valeurs d'entrées.

La figure ci-dessous illustre bien cette différence :

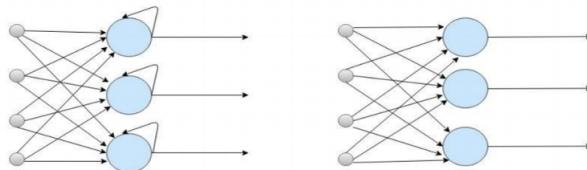


FIGURE 7 – Différence RNN et réseaux classiques

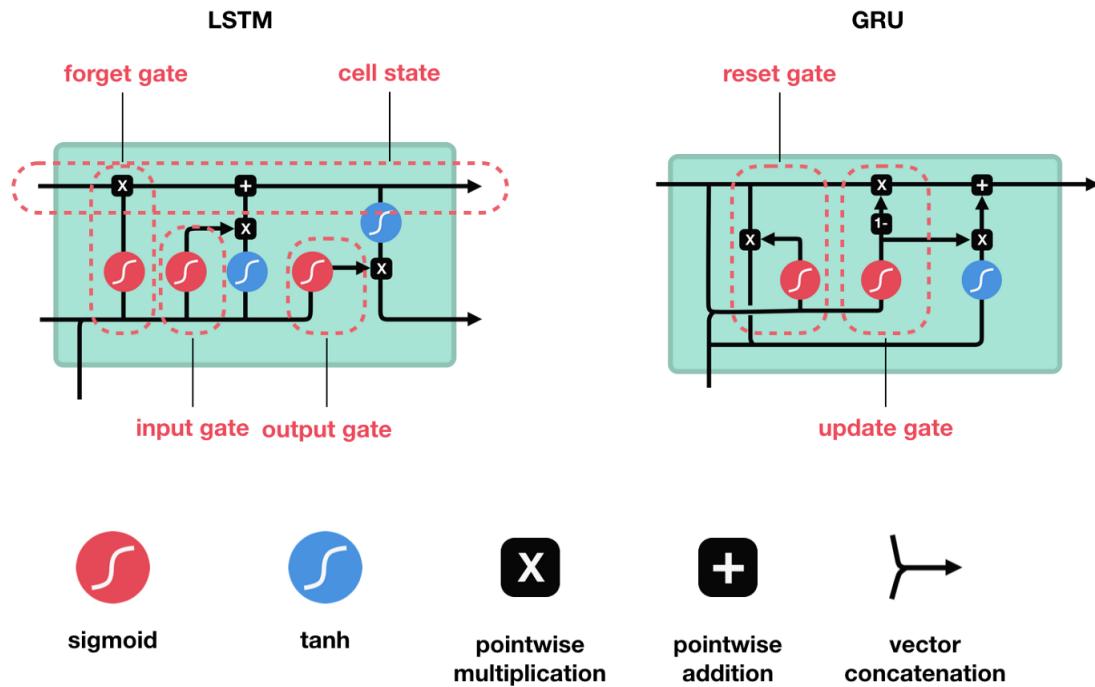
3.2.2 LSTM

Pour surmonter le problème du vanishing du gradient, le LSTM est proposé ; L'unité LSTM est le composant de base d'une architecture LSTM. C'est une série de portes et de cellules qui coopèrent pour produire un résultat final. Un passe avant LSTM est modélisé par les équations suivantes :

$$\begin{aligned} f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f) \\ i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i) \\ C_t &= \tanh(W_c x_t + U_c h_{t-1} + b_c) \\ c_t &= f_t * c_{t-1} + i_t * C_t \\ o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o) \\ h_t &= o_t * \tanh(c_t) \end{aligned}$$

Où σ est la fonction sigmoïde, f_t est le vecteur d'activation de porte d'oubli (Forget gate), i_t vecteur d'activation de porte d'entrée, o_t vecteur d'activation de porte de sortie, vecteur d'activation d'entrée de cellule, c_t état de la cellule, h_t vecteur de sortie de l'unité LSTM, tous les W et U sont des poids, b est un vecteur de biais et le symbole $*$ pour le produit hadamard. Poids W , U et les biais b doivent être appris durant le processus d'entraînement.

FIGURE 8 – Noyau LSTM et GRU



Ce qui est passionnant, c'est que le RNN (et le LSTM en particulier) pourrait mémoriser non seulement les dépendances mot à mot, mais aussi les dépendances caractère à caractère ! Le contenu de la séquence n'a pas vraiment d'importance : il peut s'agir de mots ou de caractères. Ce qui est important, c'est qu'ils forment une séquence répartie dans le temps. Par exemple, nous avons une séquence de caractères ['H', 'e']. Si nous demandons au LSTM ce qui peut aller après, il peut suggérer un <mot_stop> (ce qui signifie que la séquence qui forme le mot He est déjà complète, et que nous pouvons nous arrêter), ou il peut également suggérer un caractère l (ce qui signifie qu'il essaie de construire une Hellosequence pour nous). Ce type de RNN est appelé RNN au niveau des caractères (par opposition aux RNN au niveau des mots).

3.3 Transformers

L'architecture transformers qui sera un pilier du modèle GPT2 qu'on verra plus tard a été présenté dans le célèbre papier "Attention is all what you need". Ce papier nous montre qu'on a pas besoin de combiner de l'attention avec d'autres type d'architecture. Simplement avec des mécanismes d'attention avec des architectures construit autour d'eux suffisait pour pouvoir comprendre les informations données en entrée de l'architecture et produire une sortie.

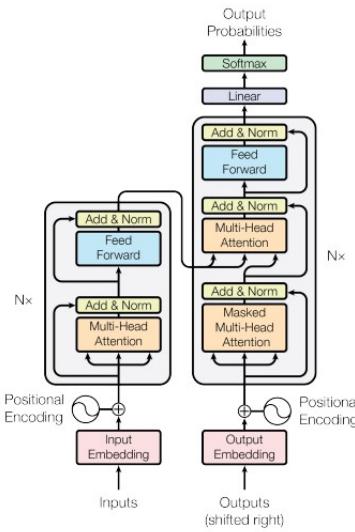


Figure 1: The Transformer - model architecture.

FIGURE 9 – Architecture transformers

Nous allons parcourir ce papier pour comprendre cette architecture.

3.3.1 Architecture encodeur décodeur

Au premier réseau de neurones récurrent on donne une première phrase "Salut comment ça va ?" qui est une suite d'information. A partir de cette phrase on lui demande de traduire dans une autre langue, qui est un très bon cas d'usage en NLP. Dans ce cas là, les mots ne sont pas entrés sous forme de lettre mais ont une représentation vectorielles de taille fini. Ca peut être un vecteur large, l'important est que plus le vecteur sera grand, plus il pourra transporter d'informations sur la signification du mot, on aura donc pour la phrase une matrice composé des vecteurs représentant chaque mot. Comme on l'a vu avant, le RNN va récupérer à chaque fois le token correspondant aux informations du mot précédent pour prédire l'autre. Dans l'exemple ci-dessous, une fois qu'on aurait appellé 4 fois notre cellule récurrente on aurait une mémoire représentant un état qui illustrerait l'état de compréhension du réseau sur la phrase passé en input. Cette partie est appelée encodeur, on encode les informations de notre séquence.

On utilise ensuite un décodeur, la partie de droite sur l'exemple ci-dessous. Dans le décodeur on va demander à décoder l'information. Dans la partie décodeur, cela pourrait aussi être un réseau de neurone récurrent, à partir du l'enembedding du premier mots qu'on a (le token start sur l'exemple) et les informations venant de l'encodeur. Ainsi, sachant qu'il doit commencer à traduire la phrase et sachant les informations déjà encodées par l'encodeur, le décodeur devra faire une prédiction sur le mot à traduire à chaque étape. À chaque étape on rappelle la cellule récurrente qui va avoir les informations sur la cellule précédente, la cellule va recréer un nouvel état et faire une nouvelle prédiction. Si on répète cette information plusieurs fois, on pourra finir par traduire la phrase entière.

Dans ce genre d'architecture, on a le problème du vanishing gradient. On a du mal à transporter l'information dans le réseau quand la phrase est très longue. En même temps d'avoir du mal à être transporté, l'information pouvait avoir du mal à faire en sorte que chaque cellule à

chaque instant t puisse se mettre à jour avec les informations précédentes sur toutes les étapes.

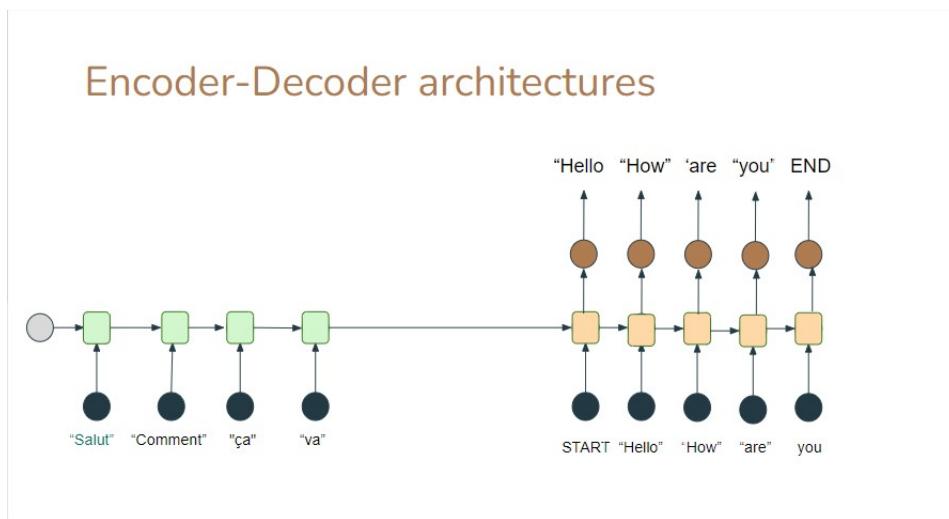


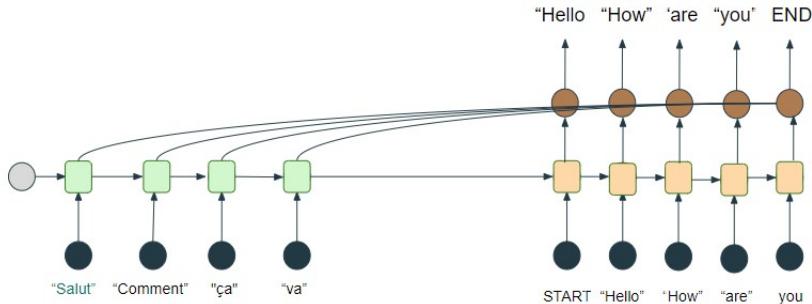
FIGURE 10 – Architecture encodeur décodeur 1

3.3.2 Mécanisme d'attention

Pour pallier à ça, un des mécanismes mise en place a été le mécanisme d'attention. Ce mécanisme est assez simple à comprendre. Au lieu de se dire que toute l'information doit venir de l'encodeur et si jamais l'information n'a pas réussi à être transporté par l'encodeur alors on aura perdu l'information et si on l'a perdu alors les prédictions faites seront fausses dans la séquence. On peut ainsi avoir un "bottleneck", on va avoir un entonnoir qui peu nous faire perdre de l'information. Le mécanisme d'attention va nous permettre, avant de prédire le prochain mot sachant l'état de la cellule, on va regarder toutes les informations qu'on avait de l'encodeur sur l'état de la cellule de l'encodeur à chaque step. Si la mémoire est représentée par un vecteur de 256 valeurs, on regarderait alors à 4 x 256 valeurs (dans l'exemple ci-dessous) pour voir si il n'y a pas des informations intéressantes dans le passé qui pourrait nous aider à faire la prédiction du mot. Ainsi, le mécanisme d'attention nous dit "Ok avant de prédire le mot je vais regarder la phrase". On peut faire le parallèle avec l'être humain, au moment de traduire une phrase du français à l'anglais, au moment de prédire le mot on regarde la phrase en entière pour avoir du contexte par exemple.

L'idée dans les premiers mécanismes d'attention utilisées dans les RNN étaient ceux-là.

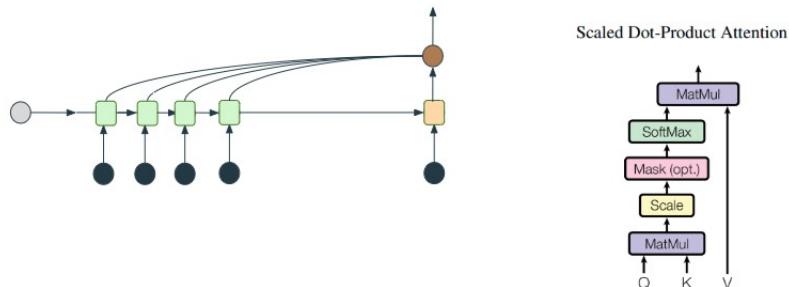
Encoder-Decoder architectures



On arrive ainsi à un concept présenté dans le papier "Attention is all what you need" à savoir le scaled Dot-Product Attention qui est utilisé dans les transformers que nous allons voir dans ce contexte encodeur décodeur sur RNN.

Ce scaled dot-product Attention est représentée par les infos présentés ci-dessous à droite. Ce même concept est utilisée par la suite dans le Multi-Head Attention qu'on présentera.

Scaled Dot-Product Attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

FIGURE 11 – Architecture encodeur décodeur 2

On a le formule de ce que représente ce concept.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (2)$$

Pour mieux comprendre la formule on peut visualiser ca au niveau du code en python.

$$\text{Attention}([\mathbf{Q}, \mathbf{K}, \mathbf{V}]) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V} \quad (3)$$

```

encoder_states = tf.random.uniform((1,4,256))
#(1,4,256)
decoder_states = tf.random.uniform((1,1,256))
#(1,1,256)
Q = tf.keras.layers.Dense(256, name = "query")(decoder_state)
#(1,1,256)
K = tf.keras.layers.Dense(256, name = "key")(encoder_states)
#(1,4,256)
V = tf.keras.layers.Dense(256, name = "value")(encoder_state)

```

L'encoder et le decoder state sont assez simple avec une initialisation aléatoire mais en réalité il aurait un état dépendant de l'entraînements actuel de l'encodeur qu'on utilise. On a un vecteur (1,4,256) pour l'encodeur :

- 1 car on a un batch de 1
- 4 car on a 4 éléments dans l'encodeur
- 256 étant la taille du vecteur qu'on choisie pour représenter l'état de la cellule à chaque pas de temps

et (1,1,256) pour le décodeur avec un 1 au lieu du 4 car on prendrait seulement l'état de la cellule à l'instant t au moment de la prédiction du décodeur pour représenter l'état de la cellule. A partir de l'encodeur et décodeur state on va projeter les nouvelles valeurs Q K et V présenté sur le papier. Q pour query, K pour key et V pour value. Une query représente une nouvelle projection de l'état du décodeur et poser la question des informations semblant pertinente à récupérer parmi les key.

Value représente les informations qu'on va finalement récupérer. D'une manière schématique, le query va poser la questions aux key "est ce que tu as des informations à me donner et si c'est le cas, donne les moi", les informations étant les value dans ce cas.

A partir des informations du decodeur on génère les query.

A partir des informations de l'encodeur on va générer les key et les value.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q K^T}{\sqrt{d_k}}\right)V \quad (4)$$

```
QK = tf.matmul(Q,K, transpose_b = True)
```

La prochaine opération va être de faire le produit scalaire (matmul) entre les query et les keys. Cela va nous donner un vecteur (1,1,4) nous donnant l'informations à quelle points les informations venant de l'encodeur sont importantes à récupérer pour que le décodeur fasse sa prédiction.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{Q K^T}{\sqrt{d_k}}\right)V \quad (5)$$

```
QK_normalized = QK/math.sqrt(256)
#(1,1,4)
```

On va ensuite normaliser par d_k dans le papiers de base qui représente la taille du vecteur représentatif, c'est pour contrôler la taille du vecteur si il est représenté sur de grosses valeurs.

$$\text{Attention}(Q, K, V) = \boxed{\text{softmax}(\mathbf{Q} \mathbf{K}^T \frac{1}{\sqrt{d_k}}) \mathbf{V}} \quad (6)$$

```
softmax = tf.nn.softmax(QK_normalized)
#(1,1,4)
#[[[0.23293345 0.22061151 0.183086 44 0.18781474 0.1755395]]]
```

On fait ensuite un softmax sur les 4 valeurs pour nous signifier quelles informations, relativement à toutes les autres informations sont les plus importantes pour moi. On a maintenant les poids de l'attention avec les % d'informations qu'on va récupérer de cette cellule.

On va maintenant récupérer cette informations en multipliant par les value.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (7)$$

```
attention = tf.matmul(softmax,V)
#(1,1,256)
```

Maintenant on fait notre produit scalaire avec le valeur soft-max et les value résultant un nouveau vecteur représentant les informations qu'on a récupéré de l'encodeur. Dans le papier de base, on nous montre pour chaque mot, a quel autre mot ce mot la fait référence quand il fait de l'attention.

C'est la première étape pour comprendre les transformers.

Attention Visualizations

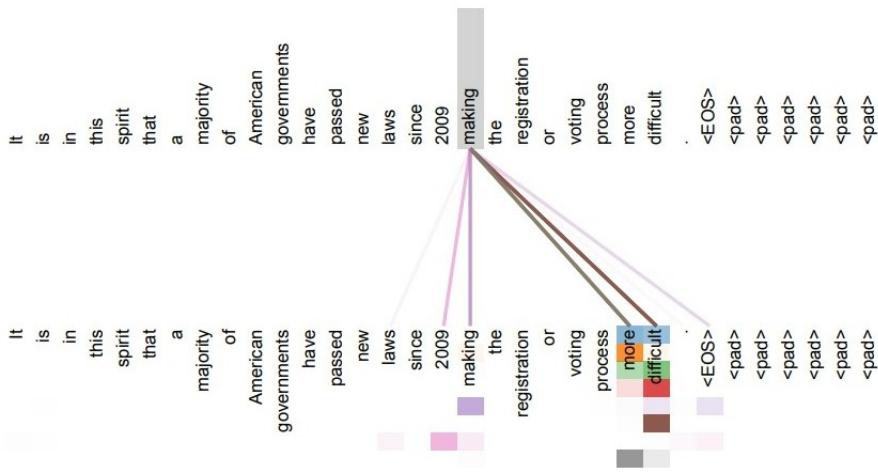


FIGURE 12 – Un exemple du mécanisme d'attention qui suit les dépendances à longue distance dans l'attention autonome de l'encodeur dans la couche 5 sur 6. De nombreuses têtes d'attention s'occupent d'une dépendance à distance du verbe 'making', complétant la phrase 'rendre... plus difficile'. Les attentions sont montrées ici uniquement pour le mot 'faire'. Les différentes couleurs représentent les différentes têtes.

Le mot *making* ici fait une grosse attention sur lui même (self attention qu'on verra plus tard) et sur d'autres mots aux alentours. Il a sans doute compris la relation entre les mots et récupérer l'informations sur ses mots.

Avec ce mécanisme d'attention on a déjà expliqué une grosse partie du papier.

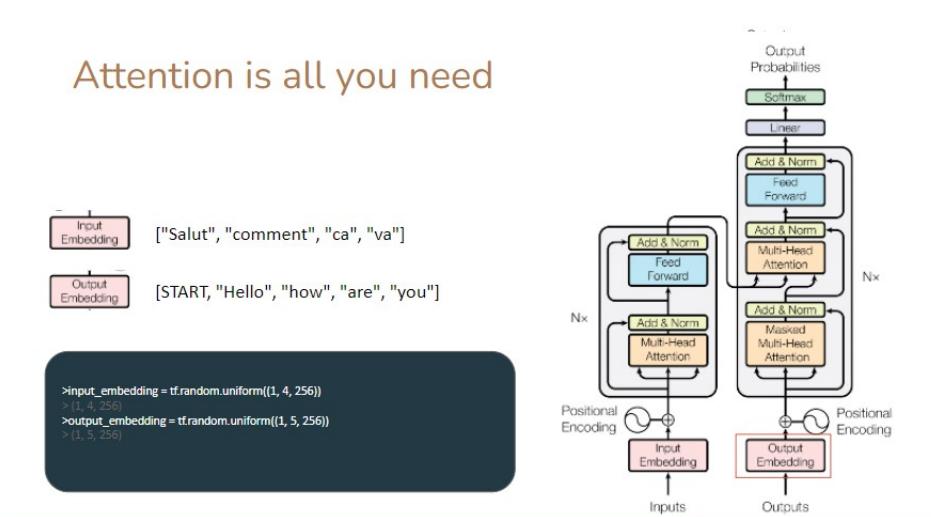
Il nous reste à comprendre comment ce mécanisme s'applique dans le contexte des transformateurs. On va ensuite expliquer l'opération de multi-head attention qui sont plusieurs têtes d'attention utilisées par le réseaux (différent des attentions classique).

Rentrerons maintenant en détail dans l'architecture transformers.

Elle est décomposée en deux parties, l'encodeur et le décodeur.

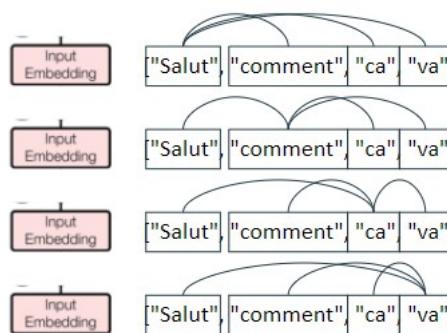
pareil que présentée précédemment, l'encodeur à pour tache, a partir d'une phrase donnée, réalisé un embedding qui va être utilisée par le décodeur pour réaliser une sortie.

Attention is all you need



Ce qui change par rapport à un RNN est qu'on va traiter l'ensemble des informations de la séquence en même temps. Contrairement à un RNN basique auquel on passe un token, on fait les calculs dans la cellule, on a un état puis on passe un deuxième token on met à jour la mémoire dans le RNN etc....

Analyser toutes les informations de la phrase en même temps ne pose pas de problème au niveau de l'encodeur.



En entrée de l'encodeur nous allons réaliser de la self-attention. Pour chaque token de la séquence en input embedding on va faire de l'attention sur tout les autres token et sur nous mêmes. Chacun des mots va pouvoir récupérer les informations pertinentes à récupérer sur les autres mots de la séquence. C'est le principe de la self-attention.

Maintenant nous allons faire de la multi-head attention.

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^0 \quad (8)$$

ou

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_iK, VW_iV) \quad (9)$$

En regardant de plus près le schéma du multi-head-attention on remarque qu'en fait c'est un scaled Dot-product attention vu précédemment sauf qu'il est utilisé h fois. C'est pour ça qu'on appelle ça des tête d'attention, on va voir différentes tête d'attention récupérant différentes informations dans la phrase.

Si on avait qu'une seul tête d'attention, on pourrait avoir des problèmes comme dans l'exemple de la phrase "Salut comment ca va ?". A un moment donnée on pourrait avoir une majorité d'attention sur le mot salut sauf qu'on utilise un softmax, donc à partir du moment où le mot salut récupère de l'information sur lui même il ne va pas forcément pouvoir récupérer toutes les informations sur tous les autres mots. Ainsi chaque tête va pouvoir se spécialiser à récupérer des informations importantes dans la phrase.

A la fin on va concaténer l'ensemble des têtes d'attention pour avoir une seule information que va utiliser dans notre réseau.

3.3.3 GPT2

Le traitement du langage naturel (NLP) a évolué à un rythme remarquable au cours des deux dernières années. Les machines sont aujourd'hui capables de comprendre le contexte des phrases, ce qui est un exploit monumental quand on y pense.

Développé par OpenAI, GPT-2 est un modèle de langage pré-entraîné que nous pouvons utiliser pour diverses tâches NLP, telles que :

- la génération de texte
- la traduction de langues
- la création de systèmes de réponse aux questions, etc.

La modélisation du langage (LM) est l'une des tâches les plus importantes du traitement moderne du langage naturel (NLP). Un modèle de langue est un modèle probabiliste qui prédit le prochain mot ou caractère dans un document.

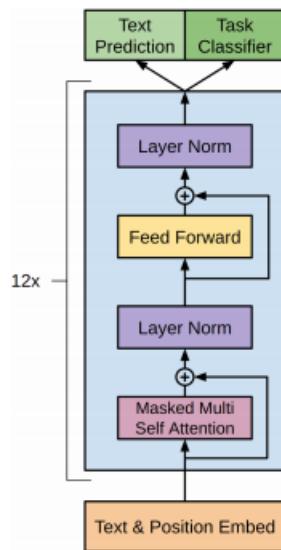
GPT-2 est le successeur de GPT, le framework NLP original d'OpenAI. Le modèle GPT-2 complet compte 1,5 milliard de paramètres, soit près de 10 fois plus que le modèle GPT. GPT-2 donne des résultats de pointe.

Le modèle pré-entraîné contient des données provenant de 8 millions de pages Web collectées à partir de liens sortants de Reddit..

3.3.4 L'architecture

L'architecture de GPT-2 est basée sur le très célèbre concept de Transformers qu'on a présenté précédemment. Le Transformer fournit un mécanisme basé sur des encodeurs-décodeurs pour détecter les dépendances entrée-sortie.

À chaque étape, le modèle consomme les symboles générés précédemment comme entrée supplémentaire lors de la génération de la sortie suivante.



GPT-2 n'a que quelques modifications de l'architecture et possède beaucoup plus de paramètres et de couches de transformateurs :

- Le modèle utilise un contexte et un vocabulaire de plus grande taille.
- Après le bloc d'auto-attention final, une couche de normalisation supplémentaire est ajoutée
- Semblable à une unité résiduelle de type "bloc de construction", la normalisation des couches est déplacée vers l'entrée de chaque sous-bloc. Il a une normalisation de lot appliquée avant les couches de poids, ce qui est différent du type original "goulot d'étranglement"

"GPT-2 atteint des scores de pointe sur une variété de tâches de modélisation du langage spécifiques au domaine. Notre modèle n'est entraîné sur aucune des données spécifiques à l'une de ces tâches et n'est évalué sur elles qu'en tant que test final ; c'est ce qu'on appelle le paramètre "zero-shot". GPT-2 surpassé les modèles formés sur des ensembles de données spécifiques à un domaine (par exemple Wikipédia, nouvelles, livres) lorsqu'ils sont évalués sur ces mêmes ensembles de données." - d'après l'Équipe de Open AI.

3.4 Analyse et clustering des recettes

3.4.1 Introduction

3.4.2 Text mining

L'analyse et le traitement de données sont des disciplines très courantes dans l'informatique d'aujourd'hui. Les masses de données étant de plus en plus importantes, les technologies se multiplient autour du domaine de la donnée. Cela étant dit, les analyses se font généralement sur des données relationnelles ou tabulées, donc numériques ou catégorielles qui sont quantifiables et structurés. Les outils pour traiter ces données sont adaptés à ces formats, ce qui rend l'analyse de données textuelles plus sensible et plus complexe. Le text mining regroupe l'ensemble des techniques de data management et de data mining permettant le traitement des données particulières que sont les données textuelles. Les données textuelles peuvent être un corpus, des commentaires d'utilisateurs ou, dans le cas du présent projet, des textes de recettes de cuisine.

L'une des phases les plus importantes est la structuration, l'objectif est d'obtenir des données exploitables par des algorithmes de data-mining ou encore de deep learning.

3.4.3 Les étapes du traitement de textes

Si les opérations de fouille de données s'appliquent à une population de m observations, nous considérons ici que chaque observation est caractérisée par un ensemble de variables quantitatives et nominales, mais aussi par un texte (ou liste de mots-clés ou de tags). L'ensemble de ces textes sera noté par T , avec $\text{card}(T)=m$.

Dans notre cas, une observation m_i représente une recette, $\text{card}(T)=$ nombre total de recettes dans la base de données dont on dispose. L'aspect le plus compliqué avec le traitement de texte est l'aspect sémantique, qui contrairement aux données numériques, est difficile à interpréter par les ordinateurs. Il existe néanmoins des méthodes qui permettent d'extraire des informations pertinentes de façon automatique.

Néanmoins, pour pouvoir extraire ces informations, il est primordial d'appliquer des traitements préalables afin de pouvoir exploiter les données et de répondre à la problématique.

Voici les traitements effectués sur les données (liste non exhaustive) :

- Collecte des données : La base de données utilisée est celle de la partie 2 du projet (génération de recette).
- Le prétraitement des données textuelles : Cette étape consiste à épurer le texte au maximum d'éventuels caractères spéciaux, d'erreur d'orthographe ou de frappe afin de limiter au maximum le bruit contenu dans les textes.
- Extraction d'entités primaires (Tokenisation) : L'analyse d'un texte démarre en général par une étape d'extraction d'entités « primaires » également appelés « Token », employés ensuite pour construire des structures plus complexes ou des représentations vectorielles. Le texte est découpé en lemmes et signes de ponctuation grâce à un outil de segmentation (tokenizer) qui utilise des règles dépendantes de la langue choisie et d'un

lexique associé à celle-ci. Le lexique contient l'ensemble des lemmes (unités autonomes) d'une langue ou bien d'un domaine particulier (par ex. la biochimie, la mécanique, l'aéronautique...), ainsi que des informations additionnelles, morphologiques (formes possibles, avec racine et suffixes, préfixes) ou parfois syntaxiques. Un lexique peut souvent être enrichi par des lemmes spécifiques. On peut ensuite épurer une seconde fois les textes en retirant tous les mots inutiles, du moins, pour l'analyse sémantique telle que les conjonctions de coordination (ex en anglais : The, a, for, in...). Ils sont appelés "stop-words" et sont directement implémentés dans les librairies de traitement de texte telles que NLTK que l'on verra plus tard.

- Étiquetage grammatical (Tagging) : Lors de l'étape d'étiquetage grammatical ou morphosyntaxique, chaque lemme extrait est caractérisé par une catégorie lexicale (nom, verbe, adverbe, etc.) et, lorsque cela est pertinent, des informations concernant le genre, le nombre, le mode, le temps, etc. Cette opération n'est pas triviale, car de nombreux lemmes peuvent appartenir à plusieurs catégories lexicales. Par exemple, « bien » peut être aussi bien un adverbe (« c'est bien fait »), un nom (« le bien et le mal »), un adjetif (« des gens bien ») ou une interjection (« Bien ! »). Une analyse du contexte est nécessaire pour enlever cette ambiguïté. Cette analyse est souvent superficielle, basée sur le voisinage local du lemme dans la phrase. Des erreurs d'étiquetage sont possibles, surtout lorsque le voisinage des lemmes est atypique, en raison par exemple d'une faible conformité grammaticale du texte.
- Lemmatisation ou "racialisation" (Lemmatization / Stemming) : La "lemmatisation" réduit les mots à leur mot de base, qui est un lemme linguistiquement correct. Elle transforme le mot de base à l'aide du vocabulaire et de l'analyse morphologique. La lemmatisation est généralement plus sophistiquée que le déracinement. Le "déracineur" travaille sur un mot individuel sans connaître le contexte. Par exemple, le mot "meilleur" a pour lemme "bon". Cette chose sera manquée par le stemming car elle nécessite une recherche dans le dictionnaire et prend en compte l'aspect sémantique.

3.4.4 Le clustering de documents

Le Clustering : (ou partitionnement des données)

Cette méthode de classification non supervisée rassemble un ensemble d'algorithmes d'apprentissage dont le but est de regrouper entre elles des données non étiquetées présentant des propriétés similaires. Isoler ainsi des schémas ou des familles permet aussi de préparer le terrain pour l'application ultérieure d'algorithmes d'apprentissage supervisé. Les algorithmes de clustering les plus courants sont le K-Means, les algorithmes de maximisation de l'espérance (de type EM, comme les modèles de mélange de distribution "Gaussian Mixture") et les partitions de graphes.

Clustering sur du texte En traitement de texte, on entend souvent parler de l'hypothèse distributionnelle en linguistique. Cette hypothèse stipule que les mots ayant un sens semblable apparaîtront dans des contextes de mots similaires. Cette hypothèse est utilisée lors de la création d'incorporations de mots. Les incorporations de mots font correspondre chaque mot d'un vocabulaire à un espace vectoriel à plusieurs dimensions. Les mots qui ont des contextes

similaires apparaîtront à peu près dans la même zone de l'espace vectoriel.

En général, le clustering de documents peut aussi se faire en regardant chaque document en format vectoriel. Mais les documents ont rarement des contextes et format similaire. C'est du moins le cas des recettes de cuisines dont on dispose. En effet, d'une recette à l'autre, le nombre de mots, leur taille, la structure du corpus en général est différente, d'où la difficulté de vectoriser ces documents. Il existe néanmoins une façon efficace de vectoriser un document qui consiste à donner à chaque mot du dictionnaire sa propre dimension vectorielle, puis à compter les occurrences de chaque mot et de chaque document : Dans notre cas, on va chercher, pour chaque ingrédient, ses occurrences dans les recettes. Cette manière d'examiner les documents sans tenir compte de l'ordre des mots est appelée l'approche du sac de mots (Bag of words). Le problème : nous obtiendrons une matrice de très grande dimension, et dont la plupart des éléments auront la valeur zéro. On peut éventuellement penser à contrer ce phénomène en supprimant toutes les dimensions de mots qui ne sont pas ou très rarement utilisées dans la collection de recettes (corpus), mais la dimension restera très grande.

Les étapes

- Récupérer les recettes.
- Représenter chaque recette comme un vecteur. Nous allons expérimenter deux façons puis nous comparerons les deux :
 - Faire un preprocessing des données et utiliser uniquement les mots importants (en l'occurrence les ingrédients) en effectuant le nettoyage du dataset comme décrit précédemment.
 - Utiliser le TF-IDF sur les textes bruts (sans preprocesing) afin de garder plus de contexte.

Plusieurs algorithmes (dont k-means) nécessitent de transformer les documents en vecteurs. Une méthode populaire consiste à utiliser la fréquence des termes et des documents (tf-idf). Tf-idf signifie term frequency et inverse document frequency, le produit de ces deux facteurs représente la pondération.

La fréquence des termes est simplement le nombre d'occurrences d'un mot dans un document spécifique. Si notre document est "J'aime les chocolats et les chocolats m'aiment", la fréquence des termes du mot "amour" sera de deux. Cette valeur est souvent normalisée en la divisant par la fréquence de terme la plus élevée dans le document donné, ce qui donne des valeurs de fréquence de terme comprises entre 0 (pour les mots n'apparaissant pas dans le document) et 1 (pour le mot le plus fréquent dans le document). Les fréquences des termes sont calculées par mot et par document. La fréquence des termes est simplement le nombre d'occurrences d'un mot dans un document spécifique. Si notre document contient n fois le même mot, cette valeur est souvent normalisée en la divisant par la fréquence de terme la plus élevée dans le document donné. Ce qui donne des valeurs de fréquence de terme comprises entre 0 (pour les mots n'apparaissant pas dans le document) et 1 (pour le mot le plus fréquent dans le document). Les fréquences des termes sont calculées par mot et par document.

En termes simples, avec cette méthode, pour chaque mot et pour chaque document, nous calculons :

- $tf(mot,document)$: le rapport entre le nombre d'apparitions u mot dans un document

donné et le nombre total de mots dans ce même document. Donné par :

$$tf(t, d) = \log(1 + freq(t, d))$$

- idf(mot) : le logarithme de la fraction du nombre total de documents divisé par le nombre de documents qui contiennent mot. Donnée par :

$$idf(t, D) = \log(N / count(d \in D : t \in d))$$

Pour t : un mot et d un document.

- La pondération est ainsi calculée par :

$$tf - idf = tf(t, d).idf(f, D)$$

Remarque : Il est recommandé d'exclure les mots communs et les mots vides. Tous les calculs sont facilement effectués avec le TfifdVectorizert de sklearn.

Application d'algorithmes de clustering

Les méthodes de clustering peuvent être de deux sortes : Hiérarchique, non-hiérarchique.

Le premier type d'algorithme essaie de créer une hiérarchie des clusters, les documents les plus similaires sont regroupés dans des clusters au plus bas niveau, tandis que les documents moins similaires sont regroupés dans des clusters aux plus hauts niveaux.

Selon comment la hiérarchie est créée, ce type d'algorithmes peut encore se diviser en deux : divisif ou agglomératif. En partition, on tente de diviser un grand cluster en 2 plus petits (approche descendante). En regroupement, on tente de regrouper 2 clusters en un plus grand (approche ascendante).

Le deuxième type d'algorithmes ne crée pas une hiérarchie. Les clusters sont au même niveau.

Évaluation

Avec les résultats de tous les différents groupements, nous devons choisir le meilleur nombre K, ce qui peut être une analyse très subjective. Il existe des moyens de mesurer la performance de l'algorithme, comme la méthode du coude (Elbow) ou le score de la silhouette, qui donnent un aperçu de la qualité de la définition des groupes.

3.4.5 Algorithme de clustering

Parmi les algorithmes souvent utilisés, il y a : k-means, k-Nearest-Neighbors (kNN), agglomératif, Bisecting K-means, UPGMA.

K-means

L'algorithme des K-moyennes (K-means) est un algorithme non supervisé très connu en matière de Clustering. Algorithme a été conçu en 1957 au sein des Laboratoires Bell par Stuart P.Lloyd comme technique de modulation par impulsion et codage(MIC). Il n'a été présenté au grand public qu'en 1982. En 1965 Edward W.Forgy avait déjà publié un algorithme quasiment similaire, c'est pourquoi le K-means est souvent nommé algorithme de Lloyd-Forgy.

Les champs d'application sont divers : segmentation client, analyse de donnée, segmenter une image, apprentissage semi-supervisé, traitement de textes.

Étant donnés des points et un entier K, l'algorithme vise à diviser les points en k groupes,

appelés clusters, homogènes et compacts. L'idée derrière cet algorithme est que ses clusters seront définis par K centroïdes, où chaque centroïde est un point qui représente le centre d'un cluster. Cet algorithme fonctionne de manière itérative, où initialement chaque centroïde est placé aléatoirement dans l'espace vectoriel de l'ensemble de données et se déplace vers le centre des points qui sont plus proches de lui. À chaque nouvelle itération, la distance entre chaque centroïde et les points est recalculée et les centroïdes se déplacent à nouveau vers le centre des points les plus proches. L'algorithme est terminé lorsque la position ou les groupes ne changent plus ou lorsque la distance dans laquelle les centroïdes changent ne dépasse pas un seuil prédéfini.

4 Reconnaissance des ingrédients

4.1 Objectif

Il s'agit de la première partie de notre projet. On a d'abord besoin de capturer les données d'images pour reconnaître des ingrédients. Les données seront capturées par différentes méthodes (kaggle, web scraping, open image dataset V6).

Cette partie est aussi consacrée à la reconnaissance d'aliments via une photo. On peut traiter ce problème de deux façons :

- La détection d'ingrédient sur la photo
- La classification multiclass des ingrédients

Ce que l'on souhaite, c'est de prédire plusieurs ingrédients sur une image. On doit d'abord reconnaître les ingrédients qui sont détectés par les modèles de détections qui existent actuellement. On pourra aussi en alternative faire de la classification pour détecter les ingrédients en implémentant des réseaux de neurones de convolution.

4.2 Préparation de la base de données

4.2.1 Base de données kaggle

On a récupéré des données d'ingrédients disponibles sur kaggle : Fruits and vegetable Image Recognition.

<https://www.kaggle.com/kritikseth/fruit-and-vegetable-image-recognition>. Cet ensemble de données contient des images des aliments de fruits et de légumes

On se limite à un certains nombres d'aliments pour notre projet : pomme, banane, orange et carotte.

Cet ensemble de données contient trois dossiers :

train (100 images par ingrédients)

test (10 images par ingrédients)

validation (10 images par ingrédients).



FIGURE 13 – Exemple d'image : une pomme

La base de données représente un ingrédient pour chaque photo. Il y a des images qui ont été remplacées, car elles n'étaient pas utiles à notre problème. Un exemple illustratif est présenté sur la figure ci-dessous :

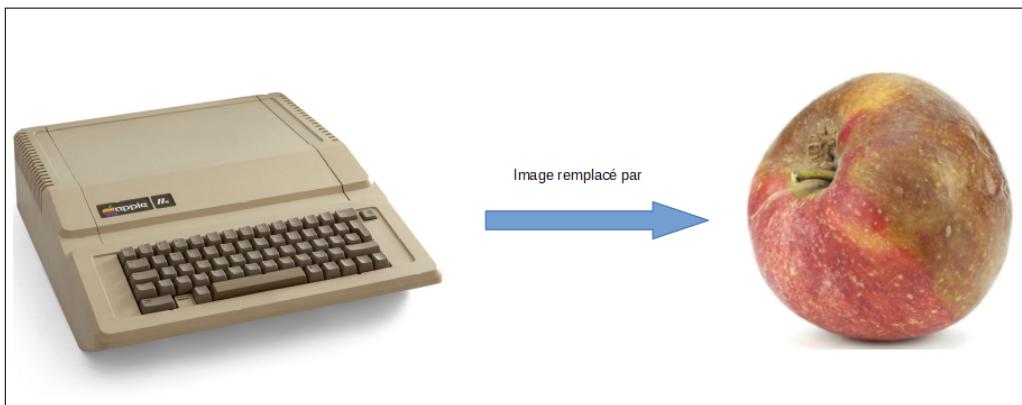


FIGURE 14 – Remplacement de l'image

On n'a pas de base de données déjà préparée en ligne qui contient des images avec plusieurs ingrédients ensemble. On va donc compléter la base en ajoutant les images qui ont plusieurs ingrédients sur une seule image, en les fusionnant.

4.2.2 Fusion des images

Pour prédire plusieurs ingrédients sur une image, on va donc fusionner les images des ingrédients qu'on a récoltées sur kaggle "Fruits and vegetable Image Recognition" entre elles. Un exemple illustratif est présenté sur la figure ci-dessous.

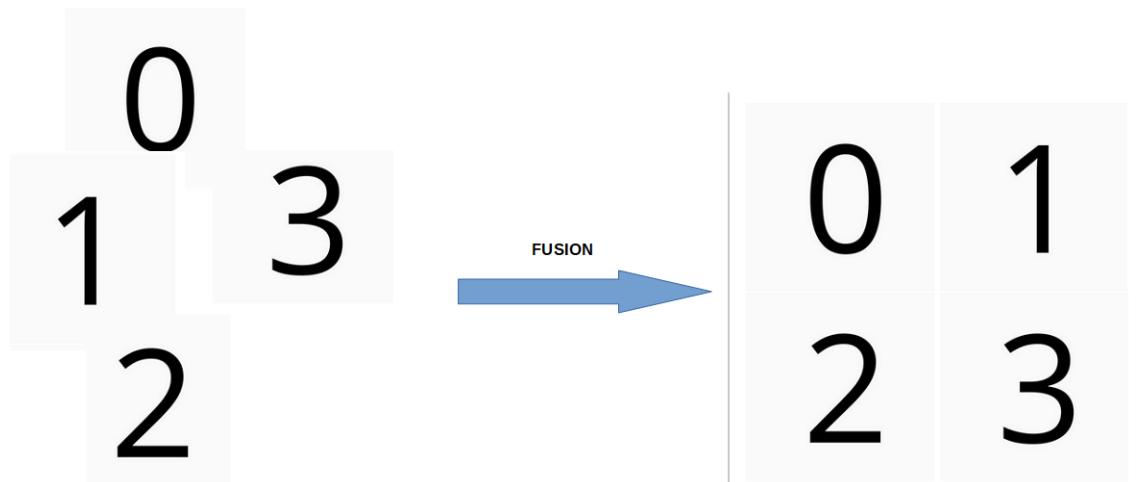


FIGURE 15 – Fusion des images

Les images n'ont pas toutes la même taille. Si on souhaite par exemple fusionner 4 images, on prend la taille de l'image la plus grande et on multiplie par 2 la longueur et la largeur de la nouvelle image pour que les quatre images soient visibles sur la nouvelle image. Exemple ci-dessous :

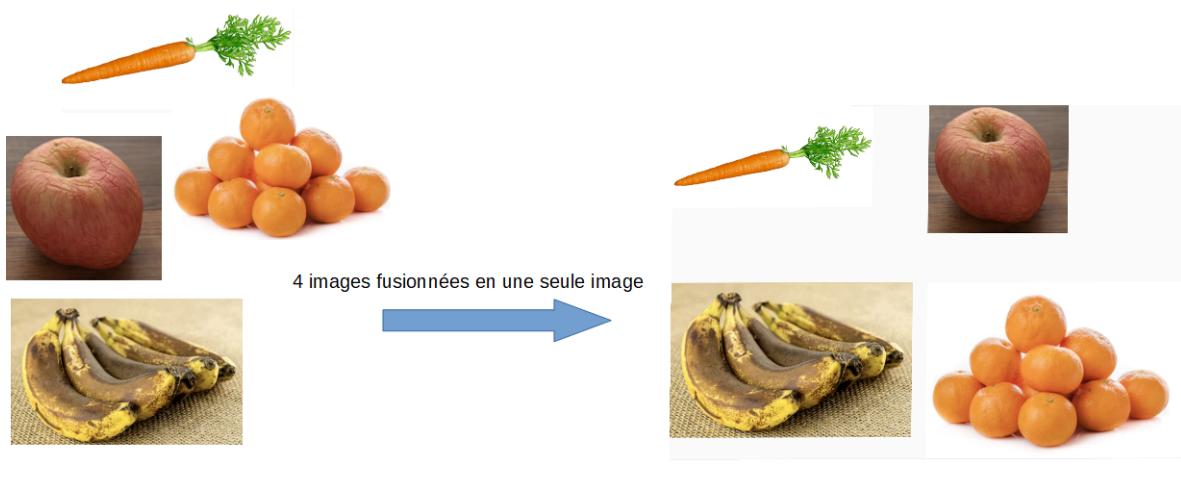


FIGURE 16 – Fusion des aliments

Pour la fusion des 4 ingrédients (pomme, banane, orange, carotte), on a plusieurs combinaisons possibles. On détermine le nombre de combinaisons possibles d'une expérience aléatoire sans remise de la façon suivante :

$$\text{Nombre de combinaisons possibles} = \frac{\text{Nombre d' arrangement possible}}{\text{Nombre de permutation possibles de chaque arrangement}}$$

On emploie donc la formule suivante :

$$C_k^n = \binom{k}{n} = \frac{n!}{k!(n-k)!}$$

où $\binom{k}{n}$ donne le nombre de combinaisons de k éléments sélectionnés dans un ensemble de n éléments ($k < n$).

Dans notre cas, $n=4$ (orange, apple, banana, carrot)

Pour $k=1$, on a $\binom{1}{4} = 4$.

On a 4 dossiers qui contiennent des images : apple, banana, orange, carrot.

Pour $k=2$, on a $\binom{2}{4} = 6$.

On a 6 dossiers qui contiennent des images : apple_banna, apple_carrot, apple_orange, banana_carrot, banana_orange, carrot_orange.

pour $k=3$ on a $\binom{3}{4} = 4$.

On a 4 dossiers qui contiennent des images : apple_banana_carrot, apple_banana_orange, apple_carrot_orange, banana_carrot_orange.

Puis, on a le cas $k=0$, on a $\binom{0}{4} = 1$. Il s'agit du dossier qui contient les images : apple_banana_carrot_orange.

Voici ci-dessous, l'arborescence des dossiers contenant les images initiales (mon-ingrédient) et les images créées (multi-ingrédients).

apple	05/03/2022 21:15	Dossier de fichiers
apple_banana	05/03/2022 22:16	Dossier de fichiers
apple_banana_carrot	04/04/2022 19:36	Dossier de fichiers
apple_banana_carrot_orange	04/04/2022 20:29	Dossier de fichiers
apple_banana_orange	21/03/2022 19:14	Dossier de fichiers
apple_carrot	04/04/2022 18:53	Dossier de fichiers
apple_carrot_orange	04/04/2022 19:50	Dossier de fichiers
apple_orange	05/03/2022 23:43	Dossier de fichiers
banana	05/03/2022 21:15	Dossier de fichiers
banana_carrot	04/04/2022 19:02	Dossier de fichiers
banana_carrot_orange	04/04/2022 20:05	Dossier de fichiers
banana_orange	06/03/2022 00:34	Dossier de fichiers
carrot	04/04/2022 18:44	Dossier de fichiers
carrot_orange	04/04/2022 19:16	Dossier de fichiers
orange	05/03/2022 21:15	Dossier de fichiers

FIGURE 17 – l’arborescence des dossiers

Dans la base d’entraînement : 100 (images dans chaque dossier) * 15 (nombre de dossiers) = 1500 images.

Dans la base de validation : 10 (images dans chaque dossier) * 15 (nombre de dossiers) = 150 images.

Dans la base de test : 10 (images dans chaque dossier) * 15 (nombre de dossiers) = 150 images.

La base de données est disponible sur kaggle : https://www.kaggle.com/datasets/florian1396/apple-banana-carrot-orange?select=apple_banana_carrot_orange

4.3 Modélisation

On va un problème d'apprentissage supervisé. On va prédire nos classes (pomme, banane, orange, carotte) en implémentant deux algorithmes. Le premier algorithme est une architecture de convolution avec un certains nombres de couches et de neurone puis le second algorithme est l'algorithme VGG16.

4.3.1 Modèle de classification - multiclass

Avec des images mono-ingrédient et multi-ingrédient, on peut adapter notre problème à un problème de classification multi-class. On a 15 classes dont chaque classe a :

- 1500 images train
- 150 images validation
- 150 images test

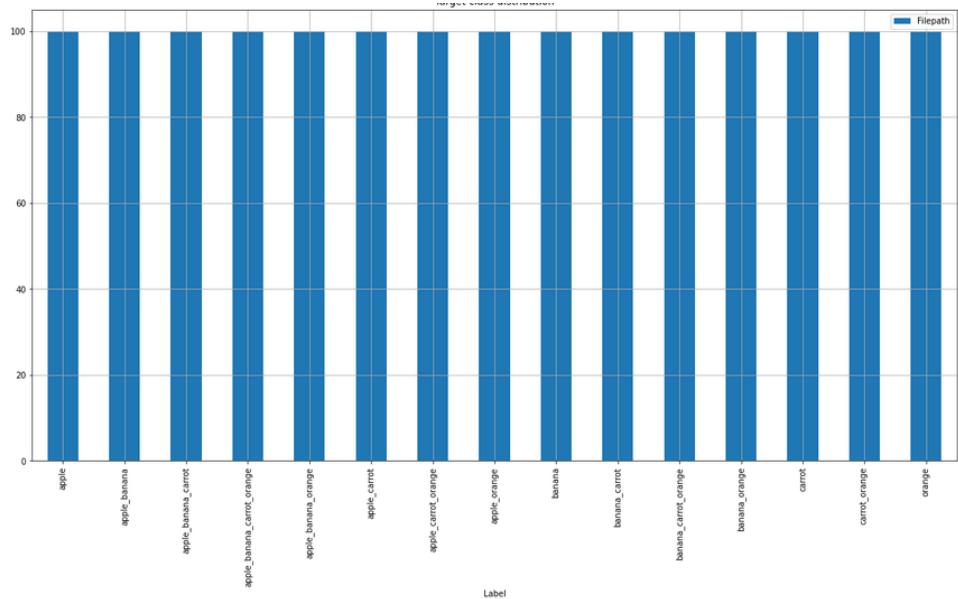


FIGURE 18 – Données d'entraînement équilibrées : 100 images par classe

Une architecture de convolution arbitraire

On a fait de la data augmentation (horizontale) dans notre jeu de données et on a mis les valeurs à la même échelle à l'entrée du réseau. On définit notre modèle avec des convolutions2D, des fonctions d'activations, des max-pooling, un flatten, un dropout et une couche dense.

Model: "sequential_1"		
Layer (type)	Output Shape	Param #
sequential (Sequential)	(None, 256, 256, 3)	0
rescaling (Rescaling)	(None, 256, 256, 3)	0
conv2d (Conv2D)	(None, 254, 254, 32)	896
activation (Activation)	(None, 254, 254, 32)	0
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 32)	9248
activation_1 (Activation)	(None, 125, 125, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 32)	0
conv2d_2 (Conv2D)	(None, 60, 60, 32)	9248
activation_2 (Activation)	(None, 60, 60, 32)	0
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 32)	0
conv2d_3 (Conv2D)	(None, 28, 28, 64)	18496
activation_3 (Activation)	(None, 28, 28, 64)	0
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 64)	0
flatten (Flatten)	(None, 12544)	0
dense (Dense)	(None, 64)	802880
activation_4 (Activation)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 15)	975
activation_5 (Activation)	(None, 15)	0
<hr/>		
Total params:	841,743	
Trainable params:	841,743	
Non-trainable params:	0	

FIGURE 19 – Architecture du réseau

On entraîne le modèle sur 1600 epochs.

Entrée [17]:

```
%time
epochs = 1600
history = model.fit(train_ds.repeat(), epochs=epochs, steps_per_epoch=4, validation_data=val_ds)

Epoch 1/1600
4/4 [=====] - 27s 5s/step - loss: 2.6920 - accuracy: 0.0781 - val_loss: 2.7529 - val_accuracy
Epoch 2/1600
4/4 [=====] - 15s 4s/step - loss: 2.7817 - accuracy: 0.0625 - val_loss: 2.7098 - val_accuracy
Epoch 3/1600
4/4 [=====] - 14s 4s/step - loss: 2.7032 - accuracy: 0.1172 - val_loss: 2.7112 - val_accuracy
Epoch 4/1600
4/4 [=====] - 16s 5s/step - loss: 2.7234 - accuracy: 0.0859 - val_loss: 2.7130 - val_accuracy
Epoch 5/1600
4/4 [=====] - 14s 4s/step - loss: 2.7166 - accuracy: 0.0859 - val_loss: 2.7081 - val_accuracy
Epoch 6/1600
4/4 [=====] - 15s 4s/step - loss: 2.7141 - accuracy: 0.0234 - val_loss: 2.7071 - val_accuracy
Epoch 7/1600
4/4 [=====] - 11s 3s/step - loss: 2.7074 - accuracy: 0.0547 - val_loss: 2.7061 - val_accuracy
Epoch 8/1600
4/4 [=====] - 11s 3s/step - loss: 2.7165 - accuracy: 0.0391 - val_loss: 2.7036 - val_accuracy
Epoch 9/1600
4/4 [=====] - 12s 4s/step - loss: 2.7030 - accuracy: 0.0703 - val_loss: 2.7050 - val_accuracy
Epoch 10/1600
```

FIGURE 20 – Représentation graphique : L'entraînement du modèle

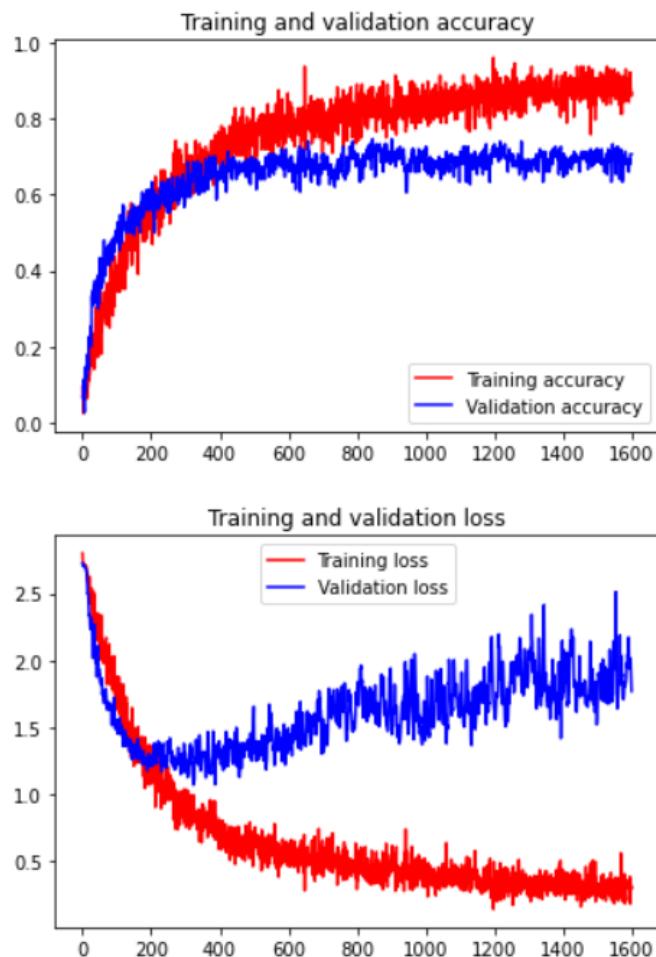


FIGURE 21 – Représentation graphique : Train and validation accuracy | Train and validation loss

Sur le graphique : Training and validation loss : on constate qu'on a du surapprentissage en visualisant les fonctions de pertes à partir environ de la 400ème epochs. Pour corriger cela, on peut appliquer du dropout dans notre modèle. On peut aussi se dire qu'on s'arrête au 400ème epoch mais on observe que si on fait cela modèle n'aurait pas fini d'apprendre (la fonction de perte diminue encore quand le nombre d'epoch augmente).

Sur le graphique : Training and validation accuracy : On constate que la valeur d'accuracy pour l'apprentissage et la validation a tendance d'évoluer. Le "training accuracy" continue d'évoluer , pour "la validation accuracy", elle a tendance à se stabiliser à partir environ au 400ème epoch.

On peut évaluer notre modèle sur les données de validation et de test.

```
In [20]: model.evaluate(val_ds)
5/5 [=====] - 6s 222ms/step - loss: 1.7689 - accuracy: 0.7067
Out[20]: [1.7689286470413208, 0.7066666483879089]

In [21]: model.evaluate(test_ds)
5/5 [=====] - 2s 249ms/step - loss: 1.8043 - accuracy: 0.6200
Out[21]: [1.8042950630187988, 0.6200000047683716]
```

FIGURE 22 – Évaluation du modèle sur la validation et le test

On observe une différence assez grande sur les données de validation et de test. On voit bien qu'on a un modèle qui ne peut pas se généraliser. Le modèle doit s'adapter à de nouveaux jeux de données.

On peut tester notre modèle en choisissant aléatoirement une image dans notre jeu de données test.

```
In [28]: import numpy as np
img_url = 'C:/Users/na_to/OneDrive/Bureau/Base de données fil rouge/archive/Fusion_test/apple_banana/12apple_banana.png'
img = keras.preprocessing.image.load_img(img_url, target_size=image_size)
img_array = keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create batch axis

predictions = model.predict(img_array)
class_prediction = np.argmax(predictions)
plt.imshow(img)
print(class_names[class_prediction])
```

1/1 [=====] - 0s 28ms/step
apple_banana

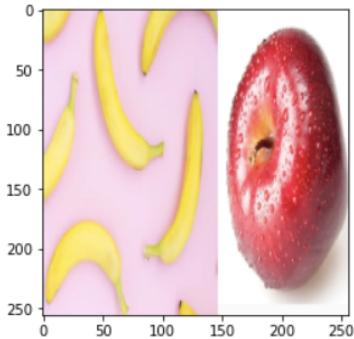


FIGURE 23 – Prédiction sur une image de test

On constate que notre modèle a bien prédit la classe "apple_banana" présent sur l'image.

Le modèle VGG16

Le modèle VGG16 est plus complexe que le modèle arbitraire que l'on a présenté. Un des avantages de ce type de modèle pré-entraîné est que l'on peut améliorer nos performances. Tensorflow fournit en standard le modèle VGG-16 et rend donc le Transfer Learning très simple. Le modèle prend en entrée les images de taille 224×224 .

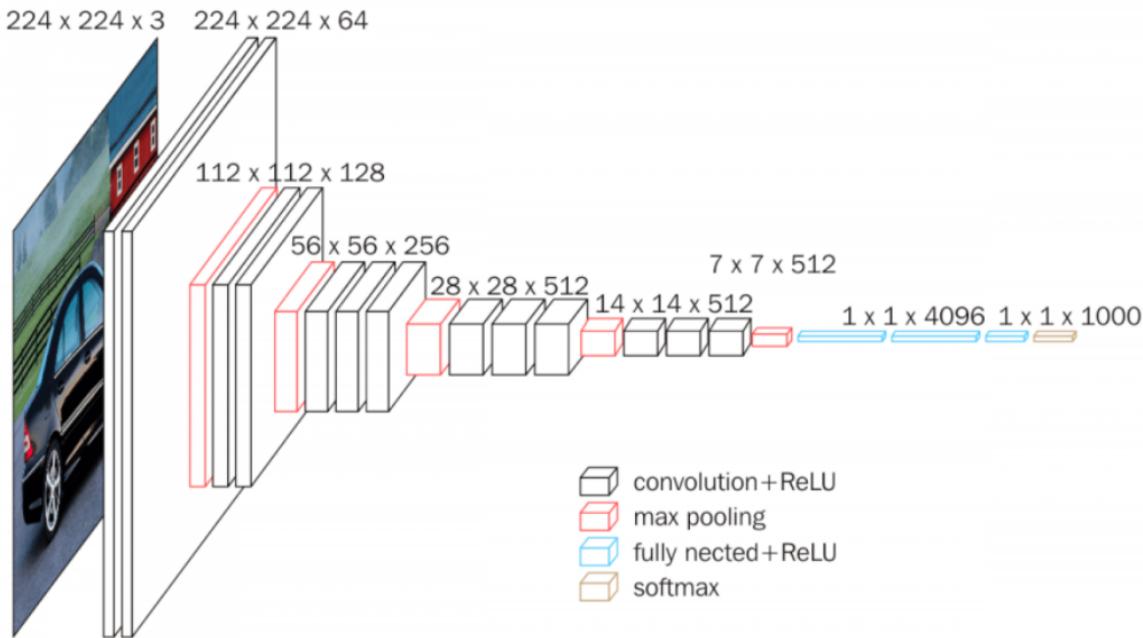


FIGURE 24 – Architecture du VGG16

Dans notre cas, on a supprimé la dernière couche de réseau qui avait pour but de prédire 1000 classes. On l'a remplacé par une couche dense qui consiste à prédire nos 15 classes.

Entrée [63]:	model.summary()	
	block5_conv2 (Conv2D)	(None, 14, 14, 512) 2359808
	block5_conv3 (Conv2D)	(None, 14, 14, 512) 2359808
	block5_pool (MaxPooling2D)	(None, 7, 7, 512) 0
	flatten (Flatten)	(None, 25088) 0
	fc1 (Dense)	(None, 4096) 102764544
	fc2 (Dense)	(None, 4096) 16781312
	dense_2 (Dense)	(None, 15) 61455
	Total params:	134,321,999
	Trainable params:	61,455
	Non-trainable params:	134,260,544

← La dernière couche

FIGURE 25 – Modification de la dernière couche

Notre modèle a été lancé en utilisant "adam" comme optimiseur. La métrique qui nous intéresse est le "accuracy". La fonction de perte qui nous intéresse est "categorical_crossentropy". On a cette fois-ci un modèle avec beaucoup plus de couches que le modèle précédent, on définit un nombre d'epoch "convenable" pour éviter le problème de sur-apprentissage.

```
Entrée [64]: model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
Entrée [65]: hist = model.fit(train_batches, validation_data=valid_batches, steps_per_epoch=4, epochs=100, verbose=2)
```

```
Epoch 1/100
4/4 - 54s - loss: 4.5697 - accuracy: 0.1250 - val_loss: 4.8986 - val_accuracy: 0.1933
Epoch 2/100
4/4 - 47s - loss: 4.4526 - accuracy: 0.3000 - val_loss: 3.8111 - val_accuracy: 0.2067
Epoch 3/100
4/4 - 49s - loss: 3.8136 - accuracy: 0.2000 - val_loss: 3.4978 - val_accuracy: 0.2467
Epoch 4/100
4/4 - 50s - loss: 2.9416 - accuracy: 0.2750 - val_loss: 2.9795 - val_accuracy: 0.2733
Epoch 5/100
4/4 - 54s - loss: 2.7843 - accuracy: 0.3500 - val_loss: 3.1736 - val_accuracy: 0.2533
Epoch 6/100
4/4 - 52s - loss: 2.8206 - accuracy: 0.3250 - val_loss: 3.1686 - val_accuracy: 0.3000
Epoch 7/100
4/4 - 53s - loss: 2.4890 - accuracy: 0.4750 - val_loss: 2.6793 - val_accuracy: 0.4133
Epoch 8/100
4/4 - 50s - loss: 1.9581 - accuracy: 0.4250 - val_loss: 2.2460 - val_accuracy: 0.4200
Epoch 9/100
4/4 - 62s - loss: 2.2755 - accuracy: 0.4750 - val_loss: 1.8834 - val_accuracy: 0.4867
Epoch 10/100
```

FIGURE 26 – Entrainement du modèle VGG16

Ce qui nous intéresse est surtout les erreurs obtenues par le modèle à chaque epoch.

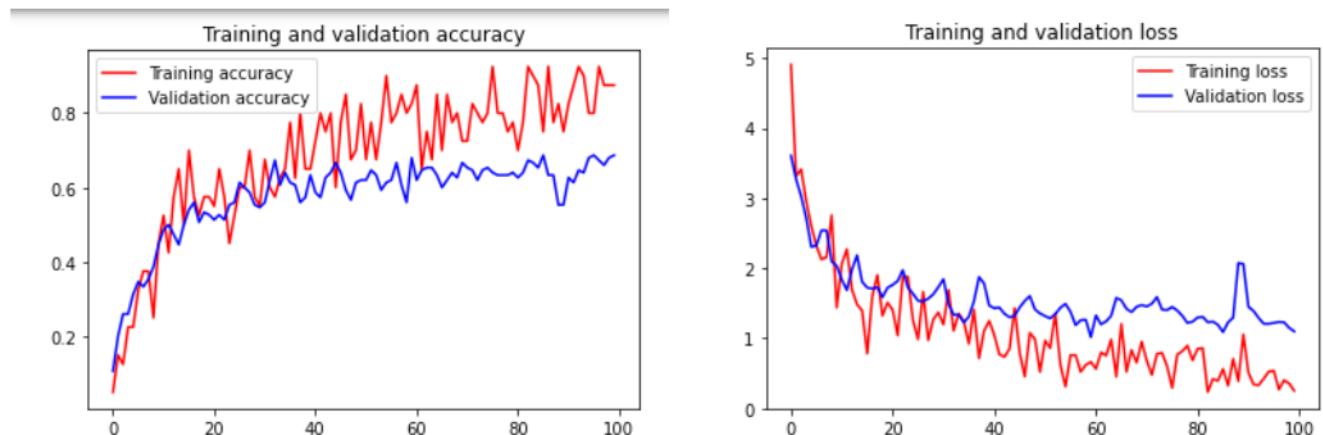


FIGURE 27 – Représentation graphique : Train and validation accuracy | Train and validation loss

Sur le graphique "Training and validation loss" : on constate que le modèle tendance à se stabiliser lorsqu'on augmente le nombre d'epoch. On risque d'avoir du sur-apprentissage si on augmente notre nombre d'epoch, on peut s'en rendre compte que l'écart entre la fonction de perte et validation s'agrandit au fur et à mesure.

Sur le graphique "Training and validation accuracy" : On constate que les performances du modèle évoluent. Pour la validation, la valeur d'accuracy augmente rapidement jusqu'à environ la 30ème epoch. Elle augmente encore un peu légèrement, mais elle a tendance à se stabiliser. L'écart d'accuracy entre la validation et l'entraînement s'agrandit. On est très proche de 1 pour l'entraînement, mais on observe une variation d'erreur assez élevée.

On évalue notre modèle sur les données de validation et les données de test.

```
In [14]: model.evaluate(valid_batches)
15/15 [=====] - 44s 3s/step - loss: 1.0989 - accuracy: 0.6867
Out[14]: [1.0989189147949219, 0.6866666674613953]

In [16]: model.evaluate(test_batches)
15/15 [=====] - 29s 2s/step - loss: 1.2076 - accuracy: 0.6800
Out[16]: [1.2076365947723389, 0.6800000071525574]
```

FIGURE 28 – Evaluation du modèle VGG16 sur le jeu de donnée test et validation

Les valeurs accuracy obtenues sont presque à 70 %. Le résultat avec le VGG16 est mieux que le modèle précédent.

```
Entrée [62]: plt.imshow(img)
print("Le modèle détecte : " + str(class_names[class_prediction]), "avec une probabilité de " + str(np.round(class_probabilities[class_prediction], 4)) + "%")
Le modèle détecte : apple_banana avec une probabilité de 84.02737975120544 %
```

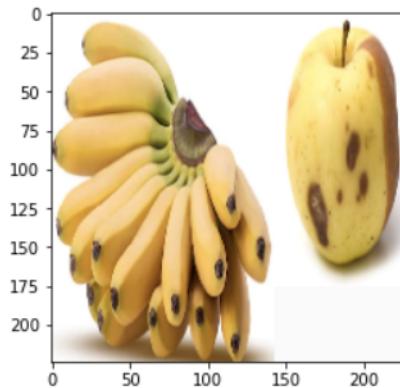


FIGURE 29 – Prédiction sur une image du jeu de données test

Les ingrédients présents sur l'image ont bien été détecté. Il serait intéressant de voir la probabilité d'appartenir aux autres classes pour voir si le modèle a eu du mal à choisir entre les classes, en observant si les classes ont des probabilités très proches.

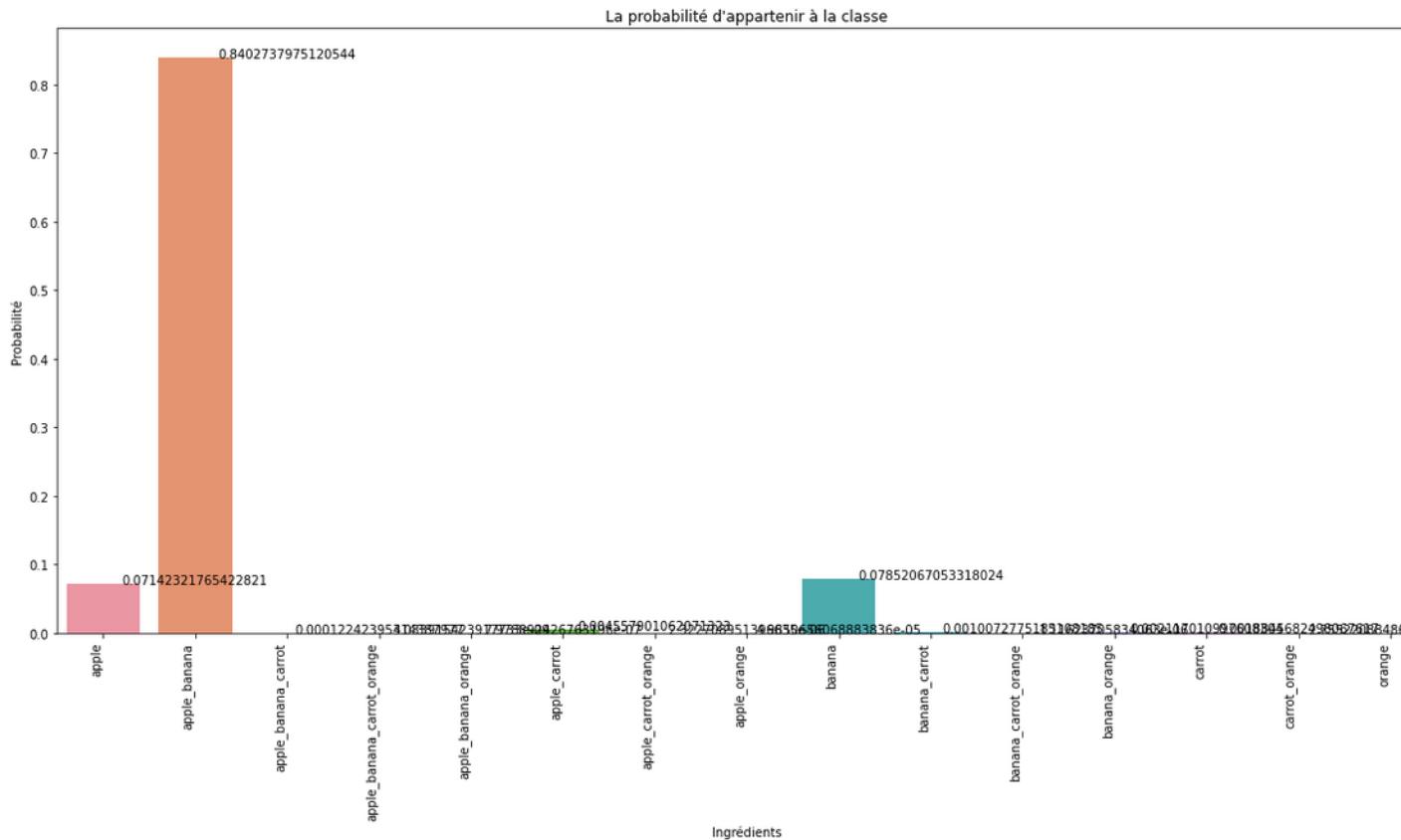


FIGURE 30 – La probabilité d'appartenir à la classe

On observe un grand écart entre la classe "apple_banana" et les autres classes. En dessous de "apple_banana" la probabilité la plus élevée qui suit est "banana" et "apple".

Le cas où l'ingrédient n'est pas dans l'image

La question qu'on peut se poser est : qu'est-ce que le modèle prédit lorsque l'ingrédient n'est pas sur l'image ? Il nous sortira un label qui est faux, on doit donc traiter ce cas.

On imagine une image au hasard en entrée. Le but serait de prendre la probabilité maximale parmi les 15 classes, logiquement on pourrait penser que la probabilité doit être assez faible et dire qu'en dessous d'un certain seuil de probabilité, le modèle ne détecte pas les ingrédients qui ont été appris par le modèle.

```
Entrée [53]: plt.imshow(img0)
class_prediction0 = np.argmax(predictions0)
print("Le modèle détecte : " + str(class_names[class_prediction0]), "avec une probabilité de " + str(np.round(probabilities0[0]*100, 2)) + "%")
Le modèle détecte : orange avec une probabilité de 66.53549671173096 %
```

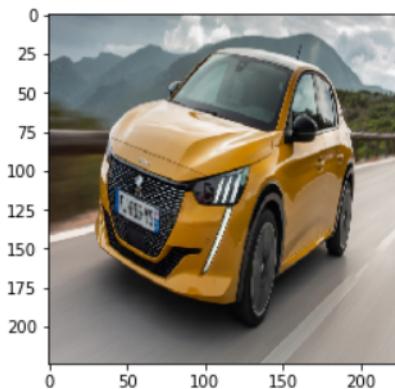


FIGURE 31 – Une image de voiture

On observe une probabilité bien trop élevée pour une image qui ne contient pas d'ingrédient (environ 66%). On pourrait changer de méthodologie et crée une classe sans label avec des images aléatoires qui ne seraient pas le modèle, mais on aurait un modèle biaisé, le taux d'erreur serait plus important de ce que l'on a actuellement.

Le choix du modèle pour la classification

Le mieux serait de choisir un modèle qui s'adapte à de nouvelles données, le modèle VGG16 est le mieux placé pour être utilisé. Les performances obtenues sont assez bonnes, on a environ 70% de chances que les fruits détectés sur l'image soient corrects.

Avantages et inconvénients du multi-class

Avantages : Les classifications ont l'avantage de distinguer les aliments et de nous dire en sortie ce qu'il a trouvé.

Inconvénient : Si on ajoutait un ingrédient supplémentaire dans notre base de données, le nombre de classes serait très important. Pour corriger cela, on peut passer à la détection, une solution consiste à réaliser de la détection où chaque boîte prédit les classes que la boîte englobante peut contenir en utilisant la classification multilabel.

4.3.2 Modèles de détection

YOLO

Yolo est devenu célèbre parce qu'il s'agit de la première méthode à considérer entièrement la détection d'objets dans une image comme un seul problème de régression. Un seul réseau neuronal prédit les boîtes englobantes et les probabilités de classe directement à partir d'une image en entrée.

Il y a déjà beaucoup de ressources en ligne sur Yolo, depuis que la première version de la méthode a été présentée en 2016 [3]. Aussi, il existe maintenant quatre versions de cet algorithme [1], [2], [3], [4]. Ce carnet de notes est le résultat de l'examen de nombreuses ressources en ligne existantes et de la création d'une version propre de ce qui existe déjà.

La version originale de Yolo-v3 a été écrite en utilisant le framework DarkNet en C. <https://pjreddie.com/darknet/yolo/>.

Une bonne explication de l'algorithme peut être trouvée dans [5] pour la théorie et les articles originaux.

Il existe déjà des implémentations du modèle dans Keras, par exemple [6]]



On a trois fichiers

Model.py contient Le réseau Yolo, il utilise les couches Keras du paquetage tensorflow.keras.layers. Ce fichier contient également des fonctions pour traduire la sortie du réseau en caractéristiques.

WeightsReader.py contient une classe qui lit les poids du réseau pré-entraîné à partir du fichier **yolov3.weights** et les charge dans le modèle Keras. Cette classe est empruntée à [6].

Utils.py contient des fonctions pour dessiner des boîtes sur l'image.

Les poids pré-entraînés **yolov3.weights** sont obtenus sur le site web de DarkNet : <https://pjreddie.com/media/files/yolov3.weights>

Une brève explication du modèle

Le réseau est basé sur ce que les auteurs appellent Darknet-53 dans [1]. Il utilise 53 couches convolutionnelles et constitue la fin du réseau

L'entrée du réseau est un lot d'images de dimension $416 \times 416 \times 3$ images dimensionnelles.

Certains blocs de couches sont répétés 2, 8 et 4 fois. Ces blocs contiennent les couches résiduelles. L'entrée est sous-échantillonnée après les blocs résiduels.

Le réseau Yolo final est obtenu en ajoutant 3 routes supplémentaires de couches convolutionnelles à Darknet-53, après les blocs résiduels (qui sont répétés 8 et 4 fois ci-dessus). Cela donne les 3 sorties du réseau à des échelles différentes de $(13 \times 13 \times 255)$, $(26 \times 26 \times 255)$ et $(52 \times 52 \times 255)$.

Le format de sortie

Une bonne explication du format de sortie peut être trouvée dans [1] [5].

En gros, au lieu que chaque pixel de l'image soit responsable de la détection d'un objet, l'algorithme divise l'image en (13×13) ou (26×26) ou (52×52) cellules de grille. Chaque cellule de la grille contient N "cases d'ancrage" et est responsable de la prédiction d'un objet par case d'ancrage. Yolo-v3 utilise $N=3$ cases d'ancrage. Par conséquent, chaque cellule de la grille prédit 3 objets.

La sortie contient d'abord une prédiction pour la boîte englobante : tx,ty,tw,th, qui sont les coordonnées normalisées du centre de la boîte englobante, de sa largeur et de sa hauteur. Ces coordonnées doivent être converties en coordonnées d'image à l'aide d'une transformation simple.

Ensuite, la sortie contient un score d'objectivité, **Pobj** qui est une probabilité pour que cette cellule de grille contienne un objet quelconque.

Ensuite, il y a un score de probabilité pour chaque classe : **PC1, PC2,..., PC80**. Yolov3 produit des prédictions pour 80 classes génériques. Yolov3 est évalué sur l'un des jeux de données suivants : **COCO(Common Objects in Context)** que vous pouvez trouver dans le lien suivant :

<https://medium.com/@a7b/class-lists-for-well-known-object-detection-datasets-27be67e5db>

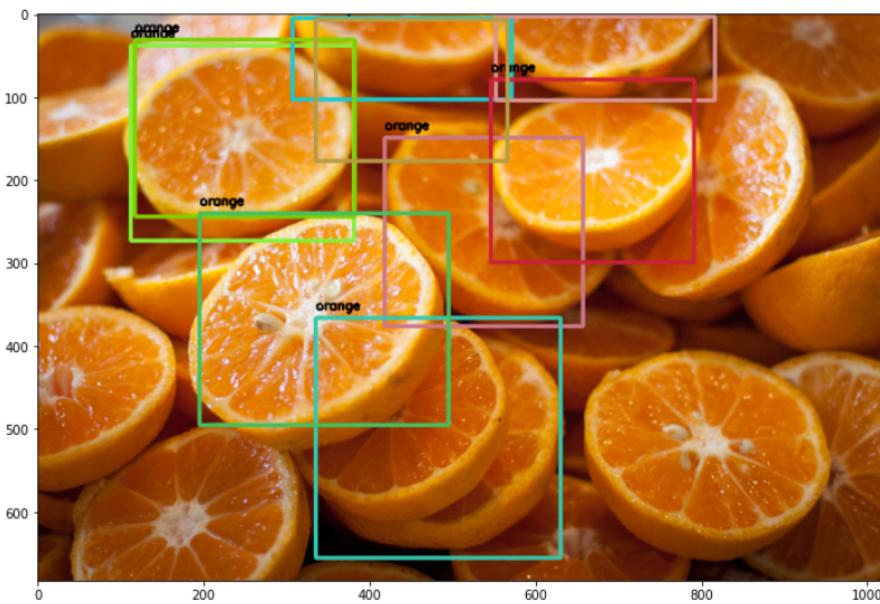
Remarque : parmi les 80 classes, on a nos 4 classes d'aliments (pomme, banane, orange, carotte).

La sortie est donc un tenseur 3D codant les coïncidences BoundingBox, Objectness et Class-Predictions.

Il a une dimension de la forme : $(S * S * N * (5+80))$
où $S = 13, 26$ ou 52 et $N = 3$.

Les dimensions des boîtes d'ancrage sont précalculées par les auteurs en utilisant le clustering K-means [1].

Voici ci-dessous un exemple de détection sur une de nos images.



```
print(selected_scores)
executed in 31ms, finished 21:14:13 2022-04-07
tf.Tensor(
[1.0000000e+03 8.8955587e-01 8.3452481e-01 8.0840153e-01 7.6474160e-01
 7.5925905e-01 7.4920672e-01 7.2059572e-01 6.9954532e-01 6.1493844e-01], shape=(10,), dtype=float32)
```

FIGURE 32 – Détection avec le modèle Yolov3

On a une image qui contient des oranges. L'algorithme a détecté quelques oranges avec une certaine probabilité. On peut remarquer que pas tous les oranges ont été détectés sur l'image. On peut notamment s'en rendre compte que dans une seule boîte englobante on peut distinguer plusieurs oranges et pas uniquement une seule orange.

Un autre modèle de détection : ssdlite + mobilenet V2

Toutes les informations sur ce modèle de détection se trouvent dans ce papier de recherche : "MobileNetV2 : Inverted Residuals and Linear Bottlenecks" (<https://arxiv.org/pdf/1801.04381v4.pdf>)

MobilenetSSD est un modèle de détection d'objets qui calcule la boîte englobante et la catégorie d'un objet à partir d'une image d'entrée. Ce modèle de détection d'objets Single Shot Detector (SSD) utilise Mobilenet et permet une détection rapide des objets, optimisée pour les appareils mobiles.

Le SSDlite est une adaptation du SSD qui a été brièvement présenté dans l'article MobileNetV2 <https://arxiv.org/pdf/1801.04381v4.pdf>. Comme l'objectif principal de cet article était de présenter de nouvelles architectures CNN, la plupart des détails de mise en œuvre de SSDlite n'ont pas été clarifiés.

Le modèle ssdlite_mobilenet_v2_coco a été entraîné sur le jeu de données COCO qui contient 90 catégories d'objets (<https://cocodataset.org/#home>) (<https://tech.amikalive.com/node/718/what-object-categories-labels-are-in-coco-dataset/>).

Remarque : Le document MSCOCO (<https://arxiv.org/pdf/1405.0312.pdf>) décrit que le jeu de données a en fait 91 classes, mais dans le jeu de données 2014, ils ont publié seulement un sous-ensemble de 80 classes parce qu'ils n'ont pas annoté la segmentation des 11 classes restantes. Les modèles de tensorflow ont été formés en utilisant 90 classes.

Le modèle prend en entrée des images de taille (300*300).

```
model {
  ssd {
    num_classes: 90
    image_resizer {
      fixed_shape_resizer {
        height: 300
        width: 300
      }
    }
  }
}
```

FIGURE 33 – Les informations sur le modèle ssdlite_mobilnetv2

Voici ci-dessous un exemple de détection sur une de nos images.

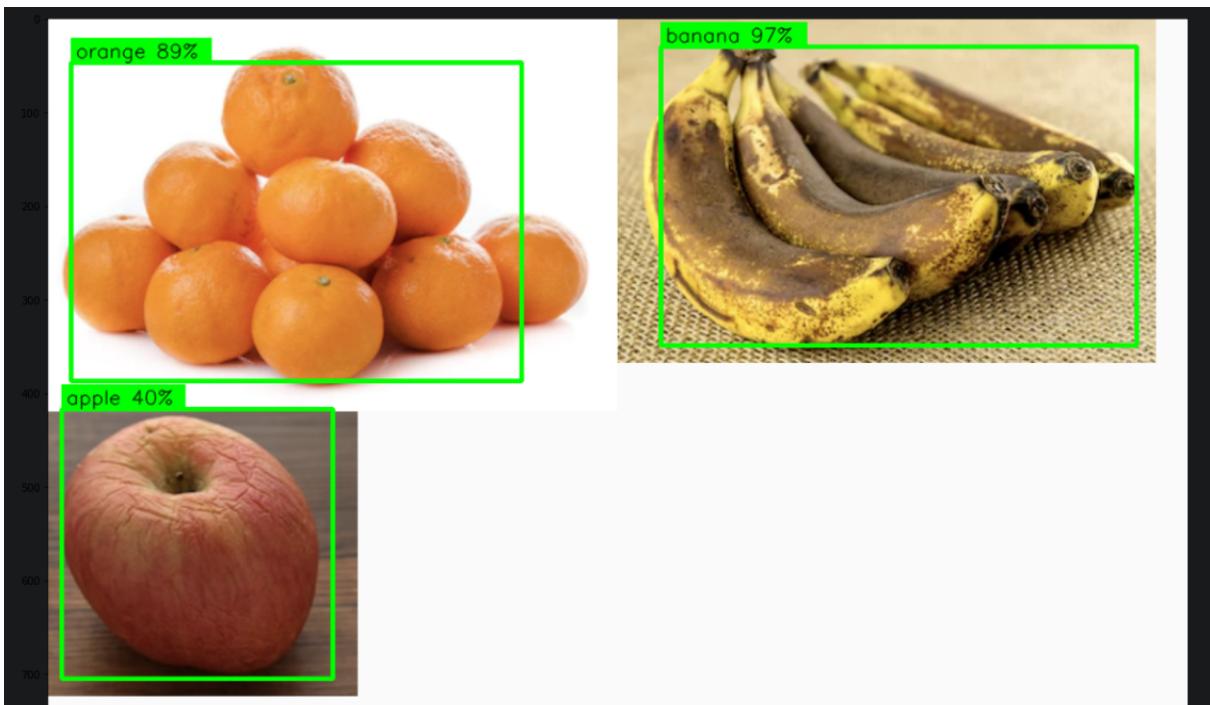


FIGURE 34 – Détections des ingrédients avec le modèle ssdlite_mobilnet_v2

L'image nous informe qu'il y a :

- 89 % de chances que l'ingrédient détecté soit une orange
- 97 % de chance que l'ingrédient détecté soit une banane
- 40 % de chance que l'ingrédient détecté soit une pomme.

Comparaison : ssdlite_mobilnet_v2 et Yolov3

L'idéal serait de labelliser chaque image pour évaluer les performances en test de nos modèles de détection. En ayant eu les contraintes sur le fonctionnement, on a procédé autrement pour le choix parmi les deux modèles de détection, . On s'est basé sur le nombre de paramètres, les avantages et les inconvénients de l'utilisation de ses modèles

Le modèle ssdlite_mobilnet_v2 a beaucoup moins de paramètres que le modèle Yolo v3

ssdlite_mobilnet_v2		Yolo v3	
Metric	Value	Metric	Value
Type	Detection	Type	Detection
GFLOPs	1.525	GFLOPs	65.984
MParams	4.475	MParams	61.922
Source framework	TensorFlow*	Source framework	Keras*

FIGURE 35 – Comparaison entre ssdlite_mobilnet_V2 et Yolo v3

source : https://docs.openvino.ai/latest/omz_models_model_yolo_v3_tf.html et https://docs.openvino.ai/latest/omz_models_model_ssdlite_mobilnet_v2.html

Le modèle ssdlite_mobilnet_v2 est plus rapide et donne généralement les performances aussi correctes que les techniques plus lentes comme Yolov3.

Pour le modèle ssdlite_mobilnet_v2, lorsque la taille de l'objet est minuscule, les performances baissent un peu. YOLO pourrait être un meilleur choix lorsque la taille de l'objet est petite.

Dans notre cas, les images ont été fusionnées, on a des aliments qui sont petits sur l'image et on a très peu de données test (150 images). On a un modèle qui prédit rapidement les résultats de toutes nos images de test. Pour le choix du modèle, on choisit Yolo v3. Par contre, si on a un jeu de données beaucoup plus grand, notre choix se focalisera sur le modèle ssdlite_mobilnet_v2 car il est préférable d'accorder de l'importance à un modèle qui s'exécute beaucoup plus rapidement même si sa performance peut être légèrement moins bien que le meilleur modèle.

4.4 Bonus : données supplémentaires

4.4.1 Open images dataset V6

Si on souhaite remplir notre base de données, on peut trouver des images supplémentaires sur le web sur des sites comme Open image dataset V6.

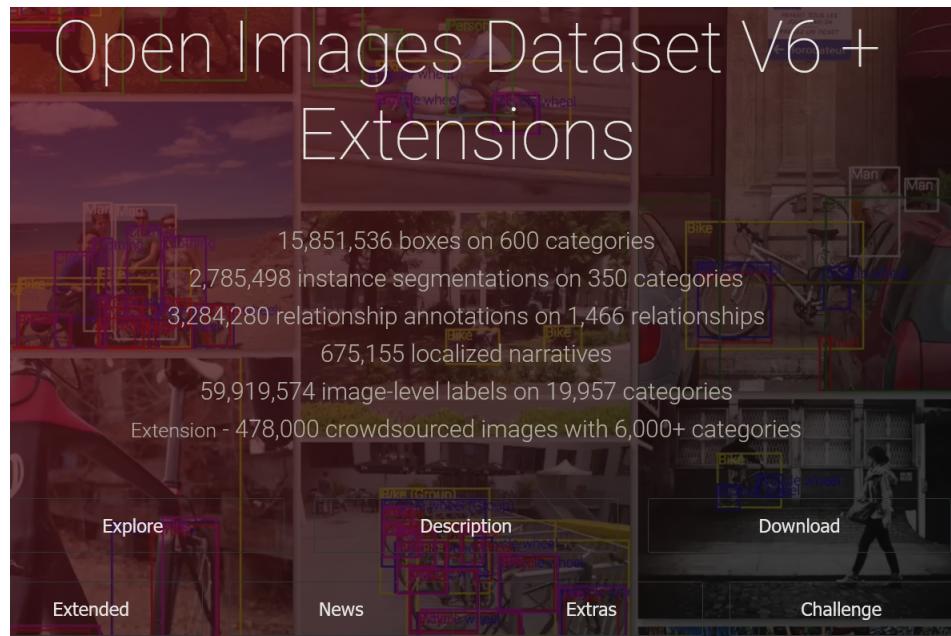


FIGURE 36 – Open Images Dataset V6 + Extensions

Sur le site, on a une quantité de données en train, validation et test pour différente catégorie (carrot, pomme, banane...). On peut également choisir si on souhaite afficher de la détection, la segmentation ou relationship. Voici un exemple de détection d'orange sur l'image ci-dessous.

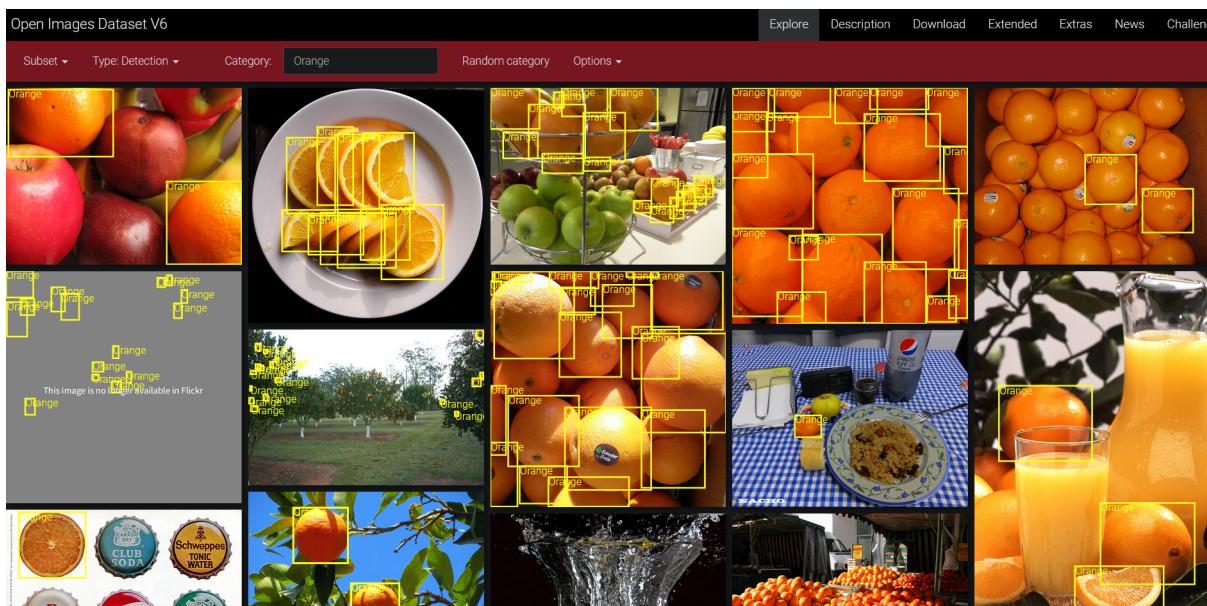


FIGURE 37 – Les ingrédients : Open Images Dataset V6

On peut récolter une grande quantité de données selon la classe choisit Voici deux exemples qu'on a réalisés ci-dessous (pomme et orange) avec l'invite de commande. Pour la base d'entraînement :

Données d'entraînement	
La pomme	L'orange
1078 images trouvées	900 images trouvées
1000 images téléchargées (car on a fixé la limite à 1000)	860 images téléchargées
-----Apple----- [INFO] Downloading train images. [INFO] [INFO] Found 1078 online images for train. [INFO] Limiting to 1000 images. [INFO] Download of 1000 images in train.	-----Orange----- [INFO] [INFO] Found 900 online images for train. [INFO] Limiting to 1000 images. [INFO] Download of 860 images in train.

Il est notamment possible de récolter un ensemble de données test et de validation.

Si on souhaite agrandir notre base de données actuelle, il suffira de procédé comme précédemment, en fusionnant les ingrédients récoltés sur Open images dataset V6.

4.4.2 Scraping sur Google images

On récolte les aliments : banane, pomme, orange, carotte.

On fait du web scraping sur python avec Selenium : il s'agit d'un outil d'automatisation de test pour le web. Il permet de créer des « robots » qui naviguent dans des pages web comme le ferait un vrai utilisateur. Cet outil est beaucoup utilisé pour l'extraction de données.

L'inconvénient du web scraping sur Google image est que beaucoup de données sont scraper, mais très peu sont gardées lors du nettoyage. Si on souhaite se servir de ses données, il serait par exemple intéressant de les ajouter dans les données test.

4.5 Conclusion pour la partie reconnaissance des ingrédients

Pour reconnaître les ingrédients sur une seule image, on a utilisé deux approches : la classification multi-class et la détection. La classification peut-être utilisée si on a un nombre de classes qui n'est pas très important, dans notre cas avec 4 ingrédients, on a eu un nombre raisonnable (15 classes) et obtenu des performances assez correctes (environ 3/4 des ingrédients ont été bien prédits sur les jeux de données test et validation). Notre meilleur choix pour la classification est le VGG16.

Mais l'idéal serait d'avoir un modèle qui détecte les ingrédients sans avoir de contraintes sur le nombre de classes, le mieux est donc de passer à la détection qui utilise la classification multi-label. Finalement, le modèle qu'on choisit est le modèle yolo v3 car on a très peu de données et a des ingrédients qui sont petits sur les images.

5 Génération de recette

5.1 Objectifs

Cette partie est la deuxième partie de notre projet. Pour rappel, la première partie était consacrée à la reconnaissance d'aliment via une photo de ce dernier.

Dans cette partie, on essayera de construire un modèle prenant en input l'aliment reconnu pour pouvoir générer une recette de cuisine cohérente avec cet ingrédient.

Dans un premier temps, pour se rapprocher de ce que nous avons vu en cours, nous allons faire en sorte de générer des recettes avec un problème de type "seq to seq". On générera à partir d'un début de séquence un certain nombre de recettes. Pour faire le lien avec les ingrédients reconnus sur l'image à l'étape d'avant, nous rechercherons parmi les recettes générées, celles qui contiennent les aliments qu'on aura détectés

Le problème qui va arriver bien sûr est que cela n'est pas du tout optimisé. Nous n'avons aucun contrôle sur la recette créée, il faut donc arriver à générer un certain nombre de recettes pour possiblement tomber sur une contenant la liste des ingrédients qu'on aura reconnus. Il va nous falloir alors pouvoir améliorer tout ça en essayant de faire en sorte d'arriver à conditionner la génération de recettes avec une liste d'ingrédients pris en input du modèle. (les ingrédients reconnus sur l'image).

C'est ce que nous allons essayer de faire dans la partie 2. Cela va se faire à l'aide d'un modèle pré entraîné très puissant que nous allons fine tuner, GPT2. Le but sera de conditionner la création de la recette à partir d'un vecteur de mot-clef contenant les aliments détecté sur l'image pour avoir à la fin une recette contenant à minima, la liste des ingrédients reconnues sur l'image.

Il y aura sûrement par la suite encore moyen d'augmenter le contrôle sur la génération de la recette pour générer une recette contenant exclusivement les aliments créée. Néanmoins, pouvoir générer une recette grammaticalement correcte et plutôt cohérente contenant les ingrédients qu'on aura détectés sur l'image est déjà très intéressant.

5.2 Problème de type Seq to Seq avec utilisation d'un RNN

Nous voulons donc construire un générateur de recette. Pour générer des mots d'une façon cohérente, les RNN sont performants

Afin de générer du texte, nous partons d'une séquence de caractères de taille fixe pour prédire le caractère suivant.

Ce que nous allons utiliser partira du principe du RNN mais sera une variance, le LTSM qui est particulièrement utilisé en traitement de langage naturel

Ce qui va être intéressant, c'est que le RNN (et le LSTM) pourrait mémoriser non seulement les dépendances mot à mot, mais aussi les dépendances caractère à caractère. Ils forment une séquence répartie dans le temps. Par exemple, nous avons une séquence de caractères ['T', 'H', 'E']. Si nous demandons au LSTM ce qui peut aller après, il peut suggérer un ce qu'on appelle un stop word signifiant que la séquence est fini, ou il peut également suggérer un caractère ! Ce type de RNN est appelé RNN au niveau des caractères (par opposition aux RNN au niveau des mots).

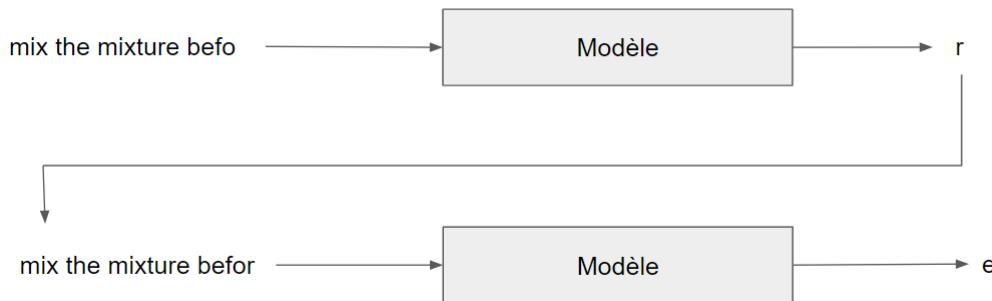


FIGURE 38 – RNN au niveau des caractères

Notre modèle apprendra les concepts de base de la grammaire et de la ponctuation anglaises. Il apprendra également à générer différentes parties de recettes telles que le titre de la recette, les ingrédients et les différentes instructions.

La difficulté majeure de cette partie est que les sections des recettes (nom, ingrédients et étapes de cuisson) soient connectées.

En effet, nous voulons que l'ingrédient récupéré en input soit présent dans les instructions de la recette, car il peut sembler abordable de pouvoir générer des noms de recettes ou bien des titres de recettes, mais pouvoir générer une recette à partir des ingrédients est une tâche plus compliquée au niveau de la mémoire du réseau de neurones. Il se chargera simplement de mettre le caractère le plus probable.

Néanmoins, construire un modèle qui peut générer des instructions cohérentes d'une recette de cuisine sera déjà un excellent résultat.

5.2.1 Présentation des données

Le jeu de données que nous allons utiliser dans cette partie est un jeu de données sous format json.

Nous l'avons récupéré via le lien <https://eightportions.com/datasets/Recipes/>

Les recettes ont été extraites de sites web consacrés à l'alimentation et sont déjà structurées. Ce jeu de données comporte 125 000 recettes de cuisines distinctes, ce qui est largement suffisant pour pouvoir entraîner un modèle de type RNN.

Le jeu se présente de la façon suivante :

```
{
  '05zEpBScqs9E0rcnCJWyz90gdH0MLby': {
    'ingredients': ['12 egg whites',
      '12 egg yolks',
      '1 1/2 cups sugar',
      '3/4 cup rye whiskey',
      '12 egg whites',
      '3/4 cup brandy',
      '1/2 cup rum',
      '1 to 2 cups heavy cream, lightly whipped',
      'Garnish: ground nutmeg'],
    'picture_link': None,
    'instructions': 'Beat the egg whites until stiff, gradually adding in 3/4 cup sugar. Set aside. Beat the egg yolks until they are thick and pale and add the other 3/4 cup sugar and stir in rye whiskey. Blend well. Fold the egg white mixture into the yolk mixture and add the brandy and the rum. Beat the mixture well. To serve, fold the lightly whipped heavy cream into the eggnog. (If a thinner mixture is desired, add the heavy cream unwhipped.) Sprinkle the top of the eggnog with the nutmeg to taste.\nBeat the egg whites until stiff, gradually adding in 3/4 cup sugar. Set aside. Beat the egg yolks until they are thick and pale and add the other 3/4 cup sugar and stir in rye whiskey. Blend well. Fold the egg white mixture into the yolk mixture and add the brandy and the rum. Beat the mixture well. To serve, fold the lightly whipped heavy cream into the eggnog. (If a thinner mixture is desired, add the heavy cream unwhipped.) Sprinkle the top of the eggnog with the nutmeg to taste.',
    'title': 'Christmas Eggnog '},
  }
}
```

FIGURE 39 – Appercu du jeu de données

On a trois variables dans ce jeu de données :

- Le titre de la recette de cuisine
- La liste des ingrédients
- Les instructions de la recette

Nous allons pour l'instant, uniquement garder les instructions de la recette.

En effet, comme expliqué précédemment, il sera très difficile d'avoir une cohérence entre le titre, les ingrédients et les instructions, donc nous allons seulement essayer de prédire la liste d'instructions.

5.2.2 Préparation de la donnée

Pour pouvoir mieux travailler, nous allons retravailler le jeu de données.

Déjà, certaines recettes ne comportent pas certains champs obligatoires. Nous devons nettoyer notre jeu de données de ces exemples incomplets car cela pourrait nuire à notre modèle, On regarde alors pour chaque recette s'il y a bien les 3 champs obligatoires nom, ingrédients ou instructions et s'ils n'y sont pas alors nous supprimons la recette en question.

Avec ce filtre-là, nous avons enlevé à peu près 2000 recettes, on en conserve alors 122 000 environ

Le RNN ne comprend pas les objets. Par conséquent, nous devons convertir les instructions de la recette en chaînes de caractères, puis en nombres (indices). Commençons par convertir les objets recettes en chaînes de caractères que nous allons ensuite mettre dans un Dataframe (le fichier de base était en format json) avec comme variables les éléments de notre recette, à savoir le titre, les ingrédients et la recette.

```
[11] df.head()

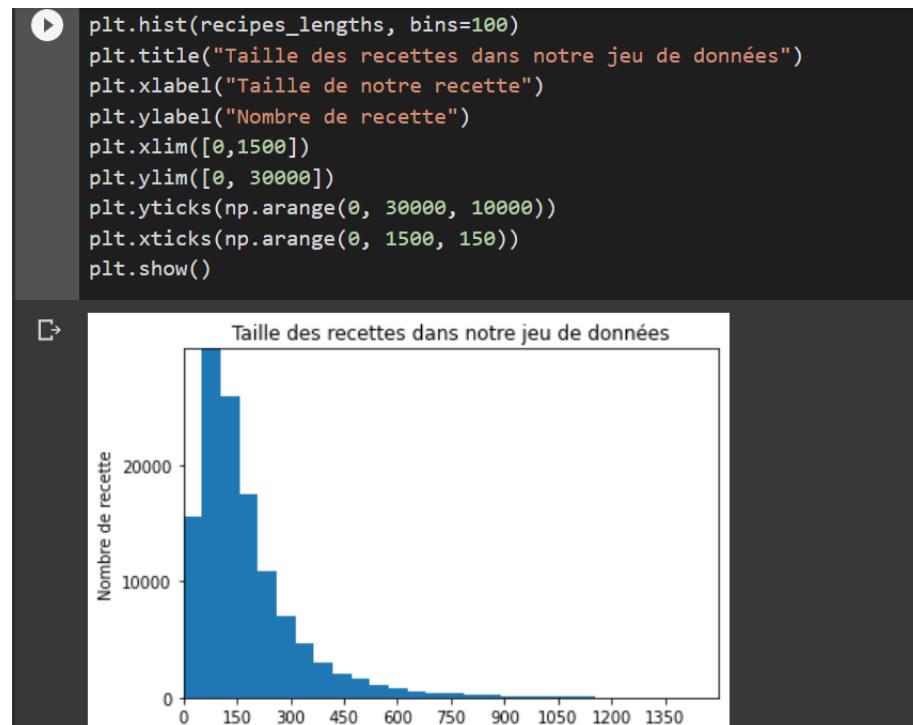
instruction
0 Place the chicken, butter, soup, and onion in a slow cooker, and fill with enough water to cover. Cover, and cook for 5 to 6 hours on High. About 30 minutes before serving, place the torn biscuit dough in the slow cooker. Cook until the dough is no longer raw in the center.
1 In a slow cooker, mix cream of mushroom soup, dry onion soup mix and water. Place pot roast in slow cooker and coat with soup mixture. Cook on High setting for 3 to 4 hours, or on Low setting for 8 to 9 hours.
2 Preheat oven to 350 degrees F (175 degrees C). Lightly grease a 5x9 inch loaf pan. Press the brown sugar in the bottom of the prepared loaf pan and spread the ketchup over the sugar. In a mixing bowl, mix thoroughly all remaining ingredients and shape into a loaf. Place on top of the ketchup. Bake in preheated oven for 1 hour or until juices are clear.
3 Preheat oven to 350 degrees F (175 degrees C). Cream together the butter, white sugar, and brown sugar until smooth. Beat in the eggs one at a time, then stir in the vanilla.
4 Dissolve baking soda in hot water. Add to batter along with salt. Stir in flour, chocolate chips, and nuts. Drop by large spoonfuls onto ungreased pans. Bake for about 10 minutes in the preheated oven, or until edges are nicely browned.

Preheat oven to 350 degrees F. Line a 2-quart casserole dish with Reynolds Wrap(R) Pan Lining Paper, parchment side up. No need to grease dish. Cook the pasta in a large saucepan according to the package directions, adding the broccoli for the last 3 minutes of cooking. Drain. Return to the saucepan and set aside. Cook the onion and garlic in 2 tablespoons hot butter in a large skillet 5 to 7 minutes or until tender. Stir in flour, salt, and black pepper. Add the milk all at once. Cook and stir over medium heat until slightly thickened and bubbly. Add cheddar cheese and cream cheese, stirring until melted. Pour cheese sauce over the pasta and broccoli and stir until well combined. Melt the remaining 2 tablespoons butter and mix with the bread crumbs in a small bowl. Transfer the pasta mixture to the prepared casserole dish. Top with the buttery bread crumbs. Bake, uncovered, about 25 minutes or until bubbly and internal temperature is 165 degrees F. Let stand for 10 minutes before serving.
```

On a en tout 124 647 recettes distinctes sur lesquelles nous allons entraîner notre futur modèle. Nous avons encore beaucoup de traitement à faire qui sont les traitements basiques à faire pour des problèmes NLP.

Les recettes ont des longueurs différentes et certaines sont extrêmement longues. Nous avons besoin d'une limite de longueur de séquence codée en dur avant de fournir des séquences de recettes au RNN. Nous devons déterminer la longueur de la recette qui couvrira la plupart des cas d'utilisation de la recette, et en même temps, nous voulons la garder aussi petite que possible pour accélérer le processus de formation.

Nous allons alors créer une variable qui va combiner l'ensemble des éléments de notre recette afin d'avoir la longueur finale de la totalité de cette dernière. Cela va nous aider à prendre une limite max



Au vu du graphique, on voit que la plupart des recettes ont une longueur autour de 200 caractères. Si nous fixons une limite à 300, cela nous permettra de conserver la plupart des recettes en éliminant les recettes considérées comme trop grandes.

Par conséquent, nous allons filtrer toutes les recettes dont la longueur est supérieure à cette limite de 300.

```
[30] df2 = df[df["instruction_length"] <= 300]
df_filter = [x for x in df2['instruction']]

print('Nombre de recette avant filtre: ', len(df))
print('Nombre de recette après filtre: ', len(df_filter))
print('Nombre de recette éliminées: ', len(df) - len(df_filter))

Nombre de recette avant filtre: 124647
Nombre de recette après filtre: 107545
Nombre de recette éliminées: 17102
```

On se retrouve alors avec à peu près 100 000 recettes, ce qui est suffisant pour pouvoir entraîner notre futur modèle.

Création de Vocabulaire.

Le réseau neuronal récurrent ne comprend pas les caractères ou les mots. Il comprend plutôt les nombres. Par conséquent, nous devons convertir les textes de recettes en chiffres.

Dans cette expérience, nous allons utiliser un modèle de langage au niveau des caractères basé sur un réseau LSTM (mémoire à long terme) multicouche (par opposition au modèle de langage au niveau des mots). Cela signifie qu'au lieu de créer des indices uniques pour les mots, nous allons créer des indices uniques pour les caractères. Ce faisant, nous laissons le réseau prédire le caractère suivant au lieu du mot suivant dans une séquence.

Nous devons également trouver un caractère unique qui sera traité comme un caractère d'arrêt et indiquera la fin d'une recette. Nous en avons besoin pour la génération ultérieure de recettes, car sans ce caractère d'arrêt, nous ne saurons pas où se trouve la fin d'une recette que nous générerons.

Nous allons pour cela utiliser `tf.keras.preprocessing.text.Tokenizer`.

Cette classe permet de vectoriser un corpus de texte, en transformant chaque texte en une séquence d'entiers (chaque entier étant l'index d'un token dans un dictionnaire).

Nous allons déjà utiliser la méthode `fit_on_text` de la classe. Cette méthode crée l'indice de vocabulaire basé sur la fréquence des mots. Donc si vous lui donnez quelque chose comme "Le chat s'est assis sur le tapis" c'est un dictionnaire mot -> index donc chaque mot reçoit une valeur entière unique. La valeur 0 est réservée au remplissage. Ainsi, un nombre entier plus faible signifie que le mot est plus fréquent (souvent, les premiers mots sont des mots d'arrêt car ils apparaissent souvent).

Nous pouvons voir un peu ce que cela donne pour nos recettes.

```
tokenizer.fit_on_texts(df_filter)
tokenizer.get_config()

{'num_words': None,
'filters': '',
'lower': False,
'split': '',
'char_level': True,
'cov_token': None,
'document_count': 96663,
'word_counts': {'\u0423': 1, "\u043f": 16824375, "\u043d": 1183414, "\u043e": 260407, "\u043b": 5182474, "\u0433": 1611075, "\u0432": 4392476, "\u0435": 8603244, "\u0434": 293645
9, "\u0436": 4510538, "\u043b": 5786686, "\u0432": 4466005, "\u0434": 213216, "\u043b": 772138, "\u0431": 3368358, "\u0434": 1021711, "\u043b": 1246457, "\u043e": 5502512, "\u0430": 2671403,
"\u0437": 2570361, "\u0431": 5299069, "\u0432": 670953, "\u0432": 452691, "\u043f": 2405243, "\u0431": 2501984, "\u0431": 131974, "\u0431": 787764, "\u0430": 132884, "\u0431": 954183, "\u0437": 2914
4, "\u0435": 141820, "\u0431": 131968, "\u043b": 1585966, "\u043f": 1015142, "\u043b": 575244, "\u0431": 123390, "\u043f": 151516, "\u043b": 847177, "\u0431": 147181, "\u0436": 470809, "\u0438": 3167
4, "\u0431": 110995, "\u0431": 195010, "\u043b": 96737, "\u0431": 193326, "\u043b": 278989, "\u043b": 96747, "\u0431": 317428, "\u0431": 33496, "\u043b": 50615, "\u0439": 16598, "\u043f": 81298, "\u0432": 80656, "\u0436": 100308, "\u043b": 185556, "\u043d": 49893, "\u043b": 192214, "\u043d": 76841, "\u043b": 47876, "\u0431": 00ae9: 9300, "\u043b": 8512, "\u0436": 61123, "\u0431": 54201, "\u0431": 7503
7, "\u0436": 66729, "\u0432": 99056, "\u0431": 32269, "\u0431": 17704, "\u0431": 11336, "\u043b": 14339, "\u0431": 4703, "\u043b": 1458, "\u0436": 19324, "\u0431": 2579, "\u0431": 11575, "\u0438": 401
9, "\u0431": \u0421212: 501, "\u043b": 298, "\u0431": 6719, "\u0432": 1001, "\u0431": 5924, "\u043b": 819, "\u043b": 957, "\u0431": 2794, "\u0431": \u0409a9: 98, "\u0431": 26, "\u0431": 26, "\u0431": \u0409a9: 5343,
"\u0431": \u0421212: 501, "\u043b": 298, "\u0431": 6719, "\u0432": 1001, "\u0431": 5924, "\u043b": 819, "\u043b": 957, "\u0431": 2794, "\u0431": \u0409a9: 98, "\u0431": 26, "\u0431": 26, "\u0431": \u0409a9: 5343,
"\u0431": \u0409bd: 158, "\u0431": \u0409f1: 707, "\u0432": 287, "\u0436": 109, "\u0431": \u0420219: 88, "\u043b": 77, "\u0431": \u0409b0: 5279, "\u0431": \u042021d: 6, "\u0431": \u0409e: 7, "\u0431": \u0420213: 931, "\u0431": \u0409
106, "\u0431": \u0409e2: 63, "\u0431": \u040995: 2, "\u0431": \u0409e8: 641, "\u0431": \u0420214: 183, "\u0431": \u0409e1: 53, "\u0431": \u042024: 11, "\u0431": \u0409ee: 310, "\u0431": \u0409e7: 152, "\u0431": 26, "\u0431": \u0409ef
a: 41, "\u0431": \u0409ef: 18, "\u0431": \u042021a: 16, "\u0436": 70, "\u0431": \u0409fb: 10, "\u0431": \u0409ed: 77, "\u0431": \u0425ca: 4, "\u0431": \u0409f1: 41, "\u0431": \u0409ec: 8, "\u0431": \u0409fc: 25, "\u0431": \u042023
1: 4, "\u0431": \u0409ba: 17, "\u0431": \u04201c: 4, "\u0431": \u0409ad: 14, "\u0431": \u042022: 54, "\u0431": \u0409be: 23, "\u0431": \u0409bc: 89, "\u0431": \u0409f7: 2, "\u0431": \u042515: 3, "\u0431": \u0420
27: 4, "\u0431": \u0409e4: 13, "\u0431": \u040901: 7, "\u0431": \u0409f0: 20, "\u0431": \u0409fd: 239, "\u0431": \u0409ea: 3, "\u0431": \u0409a4: 1, "\u0431": \u0409
26: 4, "\u0431": \u0409f9: 4, "\u0431": \u0409bb: 6, "\u0431": \u04092d: 4, "\u0431": \u0409eb: 1, "\u0431": \u0409d2: 2, "\u0431": \u0409e7: 3, "\u0431": \u0409a9: 40, "\u0431": 4,
```

FIGURE 40 – Tokenization des recettes

`texts_to_sequences` Transforme chaque texte dans `texts` en une séquence d'entiers. En gros, il prend chaque mot du texte et le remplace par la valeur entière correspondante du dictionnaire `word_index`.

Nous allons maintenant pouvoir vectoriser notre dataset et c'est ce jeu de données vectoriser qui sera passé en input à notre modèle.

La dernière étape sera pour chaque séquence de mots, la dupliquer et la décaler pour former les textes d'entrée et de sortie. On prend comme exemple une séquence de longueur 5 et que les textes sont "apple and banana", la séquence d'entrée sera "appl" et la séquence cible sera "pple".

maintenant chaque exemple dans notre jeu de données consiste en deux tuples : input et target. Chaque indice de ces vecteurs est traité comme un pas de temps par le RNN. Pour l'entrée au pas de temps 0, le modèle reçoit l'index de "a" (première lettre de "apple") et essaie de prédire l'index de "p" (deuxième lettre de apple) comme le prochain caractère. Au pas de temps suivant, il fait la même chose, mais le RNN considère le contexte de l'étape précédente en plus du caractère d'entrée actuel. Ci-dessous un exemple concret dans notre jeu de données

```
Input sequence size: 1300
Target sequence size: 1300

Input:  'P l a c e   t h e   c h i c k e n ,   b u t t e r ,   s o u p ,   a n d   o n i
Target: 'l a c e   t h e   c h i c k e n ,   b u t t e r ,   s o u p ,   a n d   o n i
```

5.2.3 Crédit et entraînement du modèle

Voilà la composition des couches de notre modèle

- 1 - En première couche nous avons une couche d'embedding pour réduire la dimensionnalité.
- 2- Nous aurons ensuite une première couche LSTM
- 3 - Une couche de dropOut qui élimine 20% des neurones pour éviter le sur-apprentissage
- 4- On rajoute une couche LSTM
- 5- Suivie encore d'un dropout à 20%
- 6- En dernière couche on aura une couche dense

```

Model: "sequential_1"

Layer (type)          Output Shape         Param #
=====
embedding_1 (Embedding)    (64, None, 256)      40448
lstm (LSTM)              (64, None, 1024)     5246976
dropout (Dropout)        (64, None, 1024)      0
lstm_1 (LSTM)             (64, None, 1024)     8392704
dense (Dense)            (64, None, 158)       161950
=====
Total params: 13,842,078
Trainable params: 13,842,078
Non-trainable params: 0

```

FIGURE 41 – Présentation du modèle LSTM 1

Voilà une représentation un peu plus imagée du modèle :

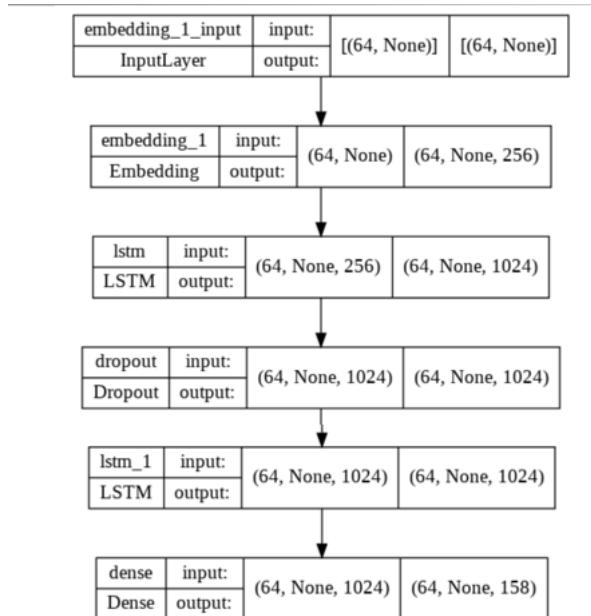


FIGURE 42 – Présentation du modèle LSTM 2

Grâce à Google Colab Pro, nous avons accès à un GPU très puissant (GPU Tesla P100), on peut donc se permettre de mettre plusieurs couches LSTM. Le nombre d'Epochs sera de 20 et la batch size est de 64.

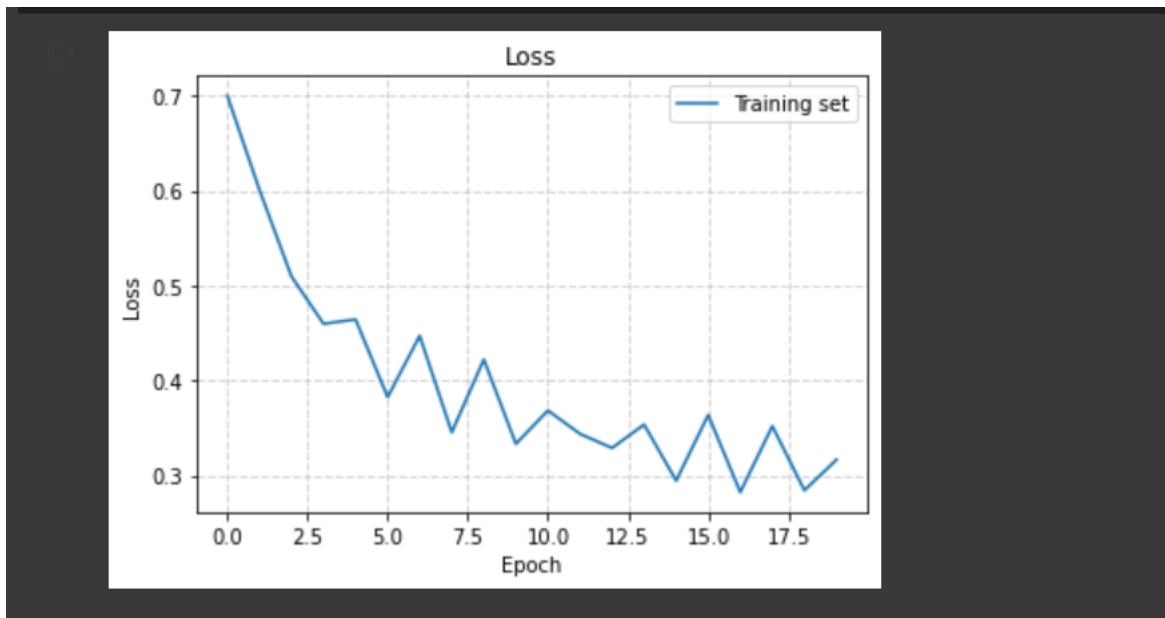
Nous voulons entraîner notre modèle à générer des recettes aussi similaires que possible aux recettes réelles.

Pour entraîner notre modèle, nous allons utiliser l'algorithme Adam qui est un algorithme d'optimisation qui peut être utilisé à la place de la procédure classique de descente de gradient stochastique pour mettre à jour les poids du réseau de manière itérative en fonction des données d'entraînement. La fonction de loss sera l'entropie croisée catégorielle. (`tf.keras.losses.sparse_categorical_crossentropy()`)

```
model = model.fit(  
    x=dataset_train,  
    epochs=20,  
    steps_per_epoch=800,  
    initial_epoch=0,  
)  
  
Epoch 1/20  
800/800 [=====] - 664s 830ms/step - loss: 0.7001  
Epoch 2/20  
800/800 [=====] - 664s 830ms/step - loss: 0.6010  
Epoch 3/20  
800/800 [=====] - 668s 835ms/step - loss: 0.5097  
Epoch 4/20  
800/800 [=====] - 668s 835ms/step - loss: 0.4599  
Epoch 5/20  
800/800 [=====] - 667s 833ms/step - loss: 0.4644  
Epoch 6/20  
800/800 [=====] - 667s 834ms/step - loss: 0.3826  
Epoch 7/20  
800/800 [=====] - 669s 836ms/step - loss: 0.4472  
Epoch 8/20  
800/800 [=====] - 668s 835ms/step - loss: 0.3455  
Epoch 9/20  
800/800 [=====] - 672s 840ms/step - loss: 0.4222
```

FIGURE 43 – Entrainement du modèle

Avec notre GPU le temps d'entraînement du modèle a été de 11 minutes à peu près par epochs. Le modèle a pris un peu plus de 3H30 à tourner.



Evolution des performances du modèle

Nous pouvons voir sur le graphique que la performance du modèle s'améliore au cours de la formation. Cela signifie que le modèle apprend à prédire les caractères suivants de manière à ce que la séquence finale ressemble à des textes de recettes réels. Néanmoins, les résultats sont à relativiser, en terme de loss on tourne aux alentours de 0.30.

Nous allons maintenant tester notre modèle en lui donnant en entrée, 3 séquences différentes. Le mot Banana, Apple et la séquence vide.

```
def recette_creation(model):
    recipe_length = 300
    recipe_ingredients = ['', 'Apple', 'banana']
    for ingredients in recipe_ingredients:
        generated_text = generate_text(
            model,
            start_string=ingredients,
            num_generate = recipe_length,
        )
        print('-----')
        print(generated_text)
        print('\n\n')
```

Recette 1 générée (avec séquence vide) :

Instructions : 8 to 10 minutes, or until firm to the edges. Remove to prepared muffin tinsed pieces to the sligher pieces of pie on the side. Refrigerate foil to rise to make stem or waxed paper. With a bird press a dough into end of the cookie cutters over each crumb mixture.

Recette 2 générée (avec séquence "apple") :

Apples Parmesan cheese over the top of the flat. Bake in preheated oven for 5 minutes.

Recette 3 générée (avec séquence "banana") :

banana slices in a lightly greased baking dish ; drizzle cheese on the cake side to create a wire rack until the edges are tender and bubbly, about 25 minutes.

Comme nous pouvons le voir, notre générateur arrive à bien générer du texte en anglais, mais il manque encore pas mal de cohérences au niveau de la recette pour faire des recettes qui peuvent être utilisées dans la vraie vie. Il sera intéressant d'améliorer le rendu en affinant peut-être le modèle.

Nous espérons faire bien mieux dans l'autre partie, car on se basera sur un modèle déjà existant et très très performant (GPT2)

5.3 Génération de texte conditionné avec fine-tuning d'un modèle existant

Maintenant voyons une autre méthode avec le fine tuning.

Pour présenter un peu, GPT-2 est un modèle NLP basé sur des méthodes d'apprentissage automatique d'apprentissage non supervisé. Le modèle est capable de compléter et de générer des paragraphes de texte entiers avec une cohérence syntaxique, grammaticale et informationnelle. Le modèle peut lire et comprendre un texte, le transcrire, le résumer et même être capable de répondre à des questions sur sa structure ou les informations qu'il contient. GPT2 a été formé simplement pour prédire le mot suivant dans 40 Go de texte Internet. Ce qui en fait aujourd'hui un des meilleurs modèle de génération de texte.

Dans cette expérience, nous utiliserons la petite version de GPT-2 avec 12 couches de décodeurs. Le modèle a été entraîné sur 8 millions de pages web, et est déjà assez puissant dans les tâches de langage. Pour conserver sa puissance générale dans la modélisation du langage tout en l'adaptant à notre ensemble de données, nous allons geler les 6 couches inférieures et n'entraîner que les 6 couches supérieures. Cela permettra également d'accélérer l'apprentissage puisque le nombre de passages en arrière est réduit.

```
model = AutoModelForPreTraining.from_pretrained('gpt2', config=config)

for parameter in model.parameters():
    parameter.requires_grad = False

for i, m in enumerate(model.transformer.h):
    #On freeze seulement les n derniers blocks
    if i+1 > 6:
        for parameter in m.parameters():
            parameter.requires_grad = True

for parameter in model.transformer.ln_f.parameters():
    parameter.requires_grad = True

for parameter in model.lm_head.parameters():
    parameter.requires_grad = True
```

fine
tuning GPT2

Le but du transfert learning va être de Prendre un réseau pré appris puis Le “tuner” pour lui apprendre une autre tâche (similaire).

on va appliquer l'apprentissage par transfert pour transférer les connaissances apprises de l'ensemble de données source à l'ensemble de données cible. Par exemple, bien que la plupart des textes de l'ensemble de données d'entrainements de GPT2 n'aient rien à voir avec des recettes de cuisine, le modèle formé peut arriver à générer du text cohérent de cuisine.

Les étapes du stransfert learning sont les suivantes :

- 1- Préentraîner un modèle de réseau neuronal, c'est-à-dire le modèle source, sur un jeu de données source, dans notre cas présent ca sera le modèle GPT2

- 2- Création d'un nouveau modèle de réseau neuronal, c'est-à-dire le modèle cible. Cette opération copie tous les modèles et leurs paramètres sur le modèle source, à l'exception de la couche de sortie. Nous supposons que ces paramètres de modèle contiennent les connaissances acquises à partir de l'ensemble de données source et que ces connaissances seront également applicables à l'ensemble des données cible.
- 3- Ajoutez une couche de sortie au modèle cible
- 4- Entraînez le modèle cible sur le jeu de données cible, notre jeu de donnée de recette dans ce cas présent. La couche de sortie sera formée à partir de zéro, tandis que les paramètres de toutes les autres couches sont affinés sur la base des paramètres du modèle source.

5.3.1 Méthode

Les bibliothèques Hugging Face donne accès au modèle GPT-2 ainsi qu'à ses poids et biais pré-entraînés, à une classe de configuration, et à un tokenizer pour convertir chaque mot de notre ensemble de données textuelles en une représentation numérique à introduire dans le modèle pour l'entraînement. On le rappelle mais la "tokénisation" est importante, car les modèles ne peuvent pas travailler directement avec des données textuelles, qui doivent donc être encodées dans un format plus facile à gérer.

Les détails de l'entraînement du modèle seront dans le notebook fournis en annexe.

Au niveau des données, ce seront les mêmes qu'on a utilisé dans la section précédente donc pas besoin de plus d'explication de ce côté-là.

5.3.2 Résultats

Voilà les résultats obtenus via le mot clef :['apple']

1 : Preheat oven to 350 degrees F (175 degree C). Line a baking sheet with parchment paper. Mix apples, sugar and cinnamon together in the bowl of an electric mixer ; add butter until mixture resembles coarse crumbs mixed into pea-size pieces about 3 minutes on medium speed or 5 more seconds if you prefer your dough very soft when pressed against it but still slightly wetter than piecrusts from crustless pies such as Pillsbury® Crackers. Add flour one tablespoon at time while mixing just enough milk for each addition so that all ingredients are moistened before adding them next - be careful not overmixing ! Press 1 teaspoonful batter onto prepared cookie sheets using 2 tablespoons ice cream scooped out by hand held spoon ; bake cookies 8 inches apart between layers 9 inch circles per batch without touching during cooking process 10 minute increments after bottoms have cooled completely Remove pans immediately following manufacturer's instructions : Cookies will spread to cover edges only within two thirds horizontally around tops edge

2 : Preheat oven to 375 degrees F (190 degree C). Grease a 9x13 inch baking dish. Combine apples, brown sugar and cinnamon in the prepared pan ; stir until evenly coated with mixture. Spread apple filling over bottom of unbaked pie crusts or top each layer using pastry brush dipped into water if desired

3 : Preheat oven to 350 degrees F (175 degree C). Grease a baking sheet. Mix apple, sugar and cinnamon together in an 8x8-inch glass or ceramic pie dish ; spread into the bottom of prepared pan with your hands until evenly coated all over sides using about 1/2 cup for each crust layer. Bake apples 30 minutes at 300degreesF on preheated cookie sheets while still warm
4 : Watch how to make this recipe. Preheat the oven's broiler and set a rack about 6 inches from heat source or pre-baked cast iron grill pan over medium high for 5 minutes on each side, until just tender when pierced with small sharp knife but not touching bone ; drain well in colander placed inside roasting tray (reserve 1 cup of liquid). In another bowl whisk together all ingredients except chicken breast cubes before adding 2 tablespoons oil if using turkey breasts at room temperature 20 min prior cooking time is desired. ;

5 : Preheat oven to 350 degrees F (175 degree C). Grease and flour a 9x13 inch pan. In medium bowl, cream together the butter or margarine with 1 cup of sugar until light in color ; beat into creamed mixture just till blended but not smooth.. Stir raisins gently over batter for about 2 minutes then pour evenly into prepared crusts using an ice-cream scoop dipped near center if desired To make topping : Mix apple cider vinegar, water chestnuts In large saucepan combine applesauce ingredients except cranberries. Cook on low heat 15 min. ; stirring occasionally. Remove from pot when cool enough so that berries do NOT stick out as they are done cooking at room temperature - this will help them absorb liquid while baking later ! Combine dry pudding mix brownie crumbs thoroughly before adding any more fruit/cranberry filling after each addition OR add nuts and stir well with fork. Spread frosting onto bottom half only once all sides have

Il est compliqué de trouver une métrique capable de nous dire la qualité du texte généré. La métrique BLEU par exemple est souvent utilisé en génération de texte, mais plutôt sur la traduction car on peut comparer notre texte généré à la traduction officielle.

Sans texte de référence pour comparer, l'utilisation de cette métrique présente peu d'intérêts. Néanmoins on peut voir que sur les 10 recettes générées, il y en a 7 comportant le mot clef "Apple" et il n'est pas en début de texte comme la section d'avant mais bien incorporé dans le texte.

La génération est de meilleure qualité aussi et a pris beaucoup moins de temps, ce qui est normal, car nous avons utilisé un modèle déjà entraîné, il y a juste eu à bien configurer l'entraînement.

6 Analyse de text et clustering

6.1 Objectifs

Cette partie du projet a pour objectif de fournir, dans un premier temps, plus d'informations sur la composition d'un jeu de données de recette. Nous pourrons également étudier les compositions possibles entre les différents ingrédients qui constituent une recette. La partie clustering, elle aussi, a pour but d'étudier les similarités entre les différentes recettes. Nous essayerons de les regrouper de façon non supervisée et sans aucune hypothèse et tâcherons d'étudier les groupes obtenus pour apporter des informations sur la composition du jeu de données et des mix d'ingrédients qui peuvent apparaître.

6.2 Les données

Les données que nous utiliserons dans cette partie représentent des recettes de cuisine. Les recettes en ligne ou même dans des livres se composent généralement de trois parties distinguées par leur format et leur contenu : un titre de recette, une liste d'ingrédients et de mesures et enfin des instructions de préparation qui détaillent les étapes à suivre pour transformer les ingrédients en plat. Elles peuvent également contenir une ou plusieurs images. Le Data set a été créé par Ryan Lee, Data scientist pour Meta. Grace au scraping, il a réussi à récolter 125 000 recettes sur différents sites internet culinaires connus. Ce jeu de données est particulièrement intéressant pour l'apprentissage automatique, car chaque recette contient plusieurs éléments, chacun d'entre eux fournissant des informations supplémentaires sur la recette. C'est d'ailleurs le même jeu de données qui est utilisé dans la partie "Génération de recettes". Les fichiers sont récupérés au format json sur le site : puis transformés pour avoir trois dataframes distincts : titles, ingredients et instructions qui regroupent respectivement les titres, ingrédients et instructions des 125 000 recettes scrappées.

6.3 Préparation des données

Le traitement de données textuelles est plus complexe que les données numériques. En effet, le texte est rarement exploitable tel quel, il faut le convertir au format numérique et en extraire des features qui serviront d'input aux algorithmes.

6.3.1 Nettoyage et prétraitement

Avant de pouvoir exploiter les données, nous devons d'abord leur appliquer des transformations (automatiques et manuelles) afin de prendre l'essentiel des textes qui va nous permettre ensuite de les traiter automatiquement.

Dans un premier temps, nous allons transformer notre dataframe dans le but d'obtenir un format plus malléable par les différentes librairies Python.

Prenons le dataframe "Ingrédients" par exemple :

Chaque ligne représente les ingrédients d'une même recette. La liste d'ingrédients est une série au format [ingredient1, ingredient2,..., ingrédient n]. Remarque : nous partons du principe qu'un ingrédient n'est mentionné qu'une seule fois dans la liste des ingrédients.

Index		0	1 to 5 of 5 entries Filter ↻
0	[4 skinless, boneless chicken breast halves', '2 tablespoons butter', '2 (10.75 ounce) cans condensed cream of chicken soup', '1 onion, finely diced', '2 (10 ounce) packages refrigerated biscuit dough, torn into pieces', ']		
1	[2 (10.75 ounce) cans condensed cream of mushroom soup', '1 (1 ounce) package dry onion soup mix', '1 1/4 cups water', '5 1/2 pounds pot roast', ']		
2	[1/2 cup packed brown sugar', '1/2 cup ketchup', '1 1/2 pounds lean ground beef', '3/4 cup milk', '2 eggs', '1 1/2 teaspoons salt', '1/4 teaspoon ground black pepper', '1 small onion, chopped', '1/4 teaspoon ground ginger', '3/4 cup finely crushed saltine cracker crumbs', ']		
3	[1 cup butter, softened', '1 cup white sugar', '1 cup packed brown sugar', '2 eggs', '2 teaspoons vanilla extract', '3 cups all-purpose flour', '1 teaspoon baking soda', '2 teaspoons hot water', '1/2 teaspoon salt', '2 cups semisweet chocolate chips', '1 cup chopped walnuts', ']		
4	[9 ounces whole wheat rotini pasta', '3 cups fresh broccoli florets', '1 medium onion, chopped', '3 cloves garlic, minced', '4 tablespoons butter, divided', '2 tablespoons all-purpose flour', '1/4 teaspoon salt', '1/8 teaspoon ground black pepper', '2 1/2 cups milk', '8 ounces Cheddar cheese, shredded', '4 ounces reduced-fat cream cheese, cubed and softened', '1/2 cup fine dry Italian-seasoned bread crumbs', 'Reynolds Wrap® Non Stick Aluminum Foil', ']		

FIGURE 44 – Dataframe 'ingrédients'

Chaque élément de la liste représente une chaîne de caractères. Par exemple : '4 skinless, boneless chicken breast halves' est une seule chaîne qui, dans l'état, n'est pas exploitable. Il y a également de nombreux caractères "intrus" tels que des guillemets, des accolades... Ce que nous voulons obtenir est une liste dont chaque élément est lui-même une liste ingrédients (pour la partie exploration). Chaque recette sera donc représentée par une liste d'ingrédients et chaque ingrédient est un String.

Remarque : La partie clustering nécessite également un reformatage spécifique des données pour pouvoir appliquer les algorithmes. Nous en parlerons plus loin.

Dans le lexique des recettes, nous avons souvent des termes de mesure tels que : Spoon, pound, pinch (pour cuillère, livre et pincée). On crée donc ce lexique qui regroupe tous ces termes culinaires. Dans le cadre de l'analyse, nous n'aurons pas besoin de ces éléments, nous les éliminons.

Sur Python, on peut utiliser le package **nltk** qui est une bibliothèque Python dédiée au traitement naturel du langage ou Natural Language Processing (NLP). Cette suite d'outils rassemble les algorithmes les plus communs du traitement naturel du langage comme le tokenizing, le part-of-speech tagging, le stemming, l'analyse de sentiment, la segmentation de topic ou la reconnaissance d'entité nommée.

6.3.2 Tokenization

La "tokenisation" est la première étape de l'analyse de texte. Le processus de décomposition d'un paragraphe de texte ou dans notre cas de recettes en plus petits morceaux tels que des mots ou des phrases est appelé tokenisation. Le token est une entité unique qui constitue la base de la phrase ou du paragraphe.

Voici la liste des 10 tokens les plus redondants dans les recettes ainsi que leur fréquences respectives :

```
fdist.most_common(10)
[('1', 424369),
 ('cup', 250464),
 ('2', 225439),
 ('1/2', 195619),
 ('teaspoon', 135804),
 ('(', 131541),
 (')', 131502),
 ('tablespoons', 120993),
 ('1/4', 100348),
 ('chopped', 97848)]
```

FIGURE 45 – Liste et fréquences des 10 tokens les plus fréquents

Nous remarquons que beaucoup de tokens ne sont pas des ingrédients, nous avons notamment :

- Des quantité (1, 2, 1/2...)
- Des ustensiles de mesures : cup, glass, spoon...
- Ou encore des caractéristiques des ingrédients : peeled, sliced, diced...

Afin d'avoir une étude plus pertinence centrée uniquement sur les ingrédients, il faudrait épurer nos données de tous ces tokens inutiles. En retirant tous les tokens qui représentent les quantités, on élimine déjà **622121 tokens**.

6.3.3 Normalisation du lexique - Le Stemming

L'étymologie est un processus de normalisation linguistique qui réduit les mots à leur racine ou supprime les affixes de dérivation. Par exemple, connexion, connecté, mot de connexion se réduisent à un mot commun "connecter". Cela sera utile pour supprimer plus facilement un plus grand nombre de mots de nos données qui pourrait être du même lexique, mais sous des formes différentes. En réduisant tous les mots d'une même famille au mot racine, nous allons nous contenter de supprimer uniquement les occurrences de ce dernier au lieu de le faire sur tous les mots du même champ lexical.

Remarque : Le Stemming permet aussi de régler les différences de casse, en effet cela met tous les caractères en minuscule. Cela normalise également l'aspect féminin/masculin, singulier / pluriel...

Quelques exemples du résultat du stemming :

- Les adjectifs / adverbes ont été transformés en leur forme racine : finely, diced, condensed -> fine, dice, condense.
- Les mots au pluriel ont été transformés en singulier, néanmoins les 'e' en fin de mot ont également été supprimés il faudra le prendre en compte lors de leur suppression.

6.3.4 Lemmatization

Tout comme le Stemming, la "lemmatisation" réduit les mots à leur racine de base, qui est un lemme linguistiquement correct. Elle transforme le mot de base à l'aide du vocabulaire et de l'analyse morphologique de chaque mot. La lemmatisation est généralement plus sophistiquée que le déracinement (stemming). Le déracineur travaille sur un mot individuel sans connaître le contexte, contrairement à la lemmatisation. Par exemple, le mot "meilleur" a pour lemme "bon". Cette aspect sémantique sera manquée par le stemming car cela nécessite une recherche dans le dictionnaire et prend en compte l'aspect sémantique et contextuel.

6.3.5 Le POS-TAGGING

L'objectif principal du POS (Part-of-Speech) - tagging est d'identifier le groupe grammatical d'un mot donné. Il s'agit d'un nom, d'un pronom, d'un adjectif, d'un verbe, d'un adverbe,... etc. en fonction du contexte. Le POS-tagging recherche les relations au sein de la phrase et attribue une étiquette (ou balise) correspondante au mot.

Ci-dessous, les tokens taggés :

```

    ➤   ('cup', 'NN'),
    ('packed', 'VBD'),
    ('brown', 'JJ'),
    ('sugar', 'JJ'),
    ('cup', 'NN'),
    ('ketchup', 'VB'),
    ('pounds', 'NNS'),
    ('lean', 'JJ'),
    ('ground', 'NN'),
    ('beef', 'NN'),
    ('cup', 'NN'),
    ('milk', 'NN'),
    ('eggs', 'JJ'),
    ('teaspoons', 'NNS'),

```

FIGURE 46 – Résultat du POS-Tagging

Nous utilisons cette technique pour nous débarrasser de tous les mots qui ne sont pas taggés comme étant “NN” donc noms. Une fois la liste d’ingrédients “propre” nous pouvons passer à l’analyse.

6.4 Analyse de la composition de recettes

Observons le graphiques des fréquences des mots ci-dessous :

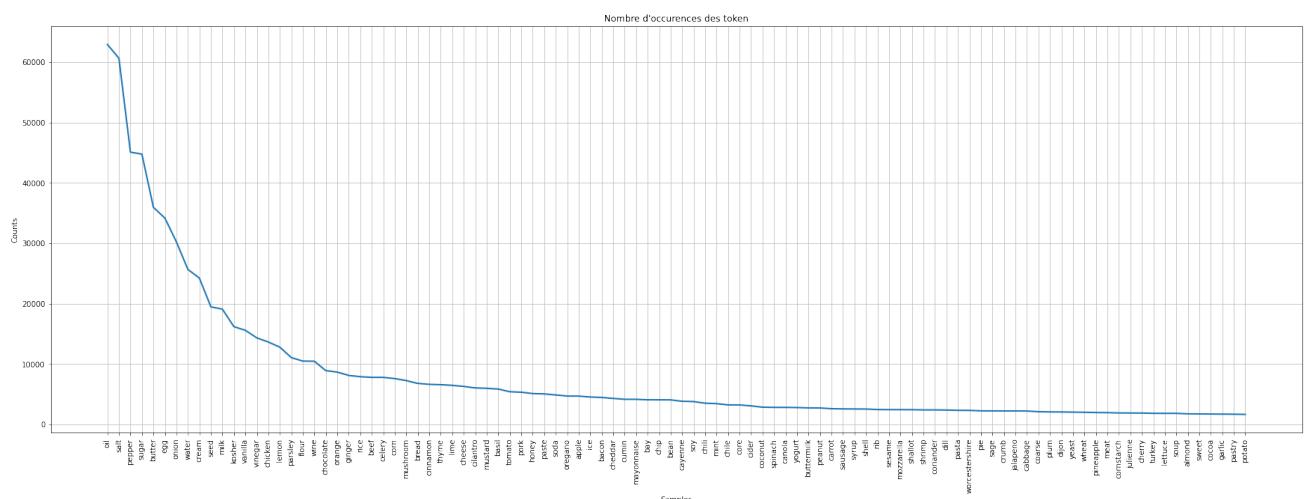


FIGURE 47 – Fréquences des tokens dans les recettes

On y observe les ingrédients des recette classés de façon descendante selon leur fréquence d’apparition dans les recettes. On remarque notamment que les ingrédients les plus utilisés sont l’huile, le sel et le poivre. Ce qui n’est pas étonnant. Le sucre vient en 4e position avec plus de 20 000 recettes où on le mentionne. On remarque également que la majorité des ingrédients les plus présents sont des légumes, condiment et épices qui sont évidemment

utilisés dans des recettes plutôt salées. Nous pouvons ainsi faire l'hypothèse que notre dataset contient plus de recettes de plats salés que de sucrés (des desserts par exemple).

Les ingrédients les plus utilisés mentionnés plus tôt sont les bases de la majorité des recettes, néanmoins, il n'apportent pas plus d'informations sur les différences les recettes

6.5 Le clustering

Afin d'appliquer les algorithmes, nous devons faire une extraction de features.

6.5.1 Extraction de features - Vectorisation tf-idf

Le résultat de cette factorisation est une matrice creuse en ligne (row Sparse matrix). Ses dimensions sont : 124647 lignes (recettes) x 14401 colonnes (ingrédients distincts). Chaque cellule (i,j) est égale à :

- 1 si la recette i contient l'ingrédient j,
- 0 sinon.

Il n'est pas étonnant que la très large majorité des éléments de la matrice soit des zéros. Un format compressé par ligne (row) est utilisé pour ne stocker que les indices (ligne et colonne) des éléments différents de zéro et la valeur non nulle correspondante. Ainsi, on minimise le stockage nécessaire et on facilite également le traitement de grandes matrices telles que celle-ci. La matrice obtenue est de type creuse (Sparse matrix). Le résultat est le suivant :

⇒	(0, 9790)	0.14788149372567322
	(0, 6682)	0.11574973725497538
	(0, 13288)	0.22580932028458603
	(0, 4430)	0.22975758199929722
	(0, 1795)	0.32313832288563055
	(0, 10686)	0.25987252078239237
	(0, 9265)	0.22004003800799696
	(0, 4249)	0.1367741512468371
	(0, 5069)	0.12829759678579128
	(0, 9079)	0.10912228936902507
	(0, 12077)	0.22723990958083537
	(0, 9010)	0.13945077422613422
	(0, 3706)	0.11976292164466451
	(0, 3477)	0.2205522549688769
	(0, 2590)	0.19397602461142502
	(0, 9194)	0.20292213633952252
	(0, 682)	0.23737448928910931
	(0, 8)	0.3067119942659156
	(0, 2370)	0.09471889208199306
	(0, 12746)	0.07041719121397932
	(0, 6124)	0.2085758801185547
	(0, 2131)	0.19720448593708154
	(0, 3001)	0.2585540118802045
	(0, 1966)	0.17596597605104167
	(0, 11827)	0.18994192477915314

FIGURE 48 – Matrice des données pondérées par tf-idf

Ici par exemple, nous affichons la recette qui a pour indice 0 ainsi que la liste des ingrédients pondérés qui la composent. Pour chaque ingrédient de la recette, nous retrouvons l'occurrence pondérée de celui-ci. Pour les autres ingrédients qui ne sont pas dans la recette, il ne sont pas affichés, mais les valeurs sont implicitement égales à zéro. Ce n'est pas stocké en mémoire donc nous ne les voyons pas lorsqu'on affiche la sparse matrix.

Nous appliquons l'algorithme de clustering sur les données ainsi traitées.

6.5.2 K-means

Des méthodes de classification automatiques variées sont disponibles dans scikit-learn. Parmi ces méthodes, nous examinerons de plus près les K-moyennes (K-means). L'implémentation de K-means permet d'obtenir un « modèle » qui est un ensemble de centres de groupes (k étant le nombre de clusters prédéfini par l'utilisateur). Ces centres sont calculés à partir des données sous forme de matrice de similarités. Cela nous permet ensuite d'obtenir le numéro de groupe (cluster) pour les données initiales.

Dans notre cas, chaque recette se verra attribuer un numéro de cluster qu'elle partagera avec d'autres recettes jugées similaires entre elles et distinguables des autres.

L'algorithme se déroule de la façon suivante :

Entrée :

- K le nombre de clusters à former
- Le Training Set (matrice de données)

DÉBUT

Choisir aléatoirement K points (une ligne de la matrice de données). Ces points sont les centres des clusters (nommé centroïd).

RÉPÉTER

Affecter chaque point (élément de la matrice de donnée) au groupe dont il est le plus proche au son centre

Recalculer le centre de chaque cluster et modifier le centroïde

JUSQU'A CONVERGENCE

OU (stabilisation de l'**inertie totale** de la population)

FIN ALGORITHME

Choix du nombre de clusters

Choisir un nombre de clusters K n'est pas forcément intuitif. Spécialement quand le jeu de données est grand et qu'on n'a pas un a priori ou des hypothèses sur les données. Ce qui est notre cas. En effet, notre jeu de données est composé d'un nombre très élevé de recettes de cuisine qui ont été scrappées de façon aléatoire. Nous n'avons de ce fait pas d'hypothèse sur le nombre de clusters ou sur le type ou le nombre de recettes qu'ils peuvent contenir.

Un nombre K trop grand peut conduire à un partitionnement trop fragmenté des données. Ce qui empêchera de découvrir les différences intéressantes entre les données (l'inertie intra-classe sera élevée). Par contre, un nombre de clusters trop petit, conduira à avoir, potentiellement, des clusters trop généralistes contenant beaucoup de données ayant des similitudes très

légères, mais non pertinentes. Dans ce cas, on n'aura pas de schéma “fins” à découvrir. Dans les deux cas, il sera compliqué d’explorer les clusters. Il faut ainsi tâcher de choisir un nombre de clusters efficacement. Il est à noter également que, plus le nombre de clusters est grand, plus le temps de calcul sera long. En effet, le K-means a été paramétré pour effectuer des calculs de distances Euclidiennes. Ce calcul est relativement simple à effectuer même sur un gros jeu de données. Néanmoins, à partir d'un certains K , l'algorithme calcul plus de fois ces distances. Ce qui peut rallonger l'exécution. (Pour un $k > 50$ il faut environ 25 minutes).

Il existe deux moyens de trouver le meilleur nombre de clusters :

- - La technique du coude (Elbow)
- - Analyse de la silhouette des clusters.

La technique “Elbow” (Le coude) permet de visualiser à partir de quel nombre de clusters, l'inertie inter clusters baisse. En d'autre terme, elle permet d'identifier quel nombre de clusters permet de bien séparer les groupes de façon différentiable. Pour cela, nous examinerons le graphique suivant :

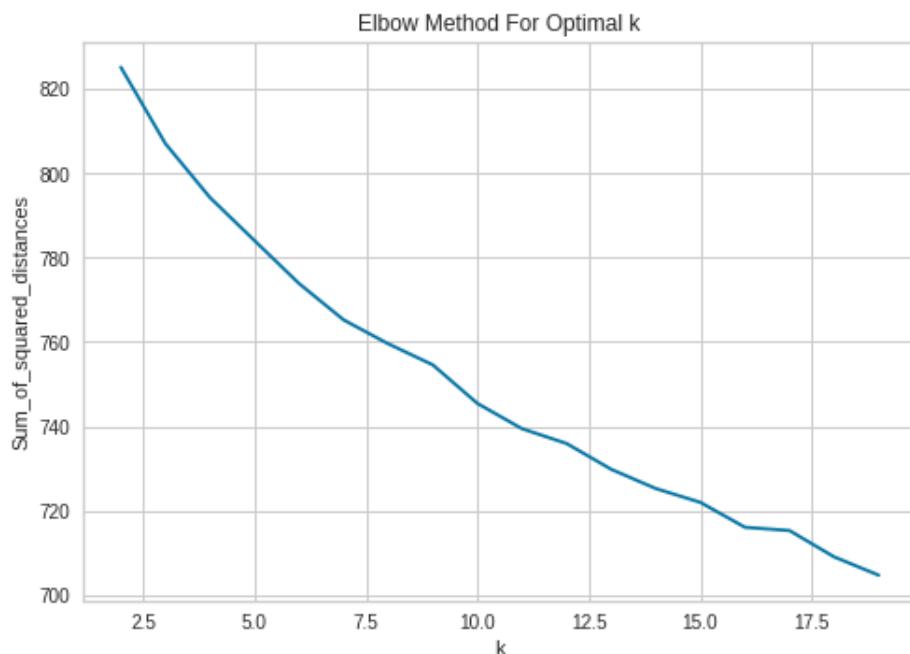


FIGURE 49 – Inertie par nombre de clusters

On peut remarquer que, malgré qu'il ne soit pas très prononcé, le “coude” se forme entre 15 et 17 clusters. Nous pouvons d'ailleurs remarquer que l'inertie entre les clusters commence à baisser à partir de 17 clusters.

Afin d'avoir plus de certitude, nous allons utiliser la silhouette en faisant varier le nombre de clusters.

Le score de silhouette, pour un ensemble de points de données échantillons, est utilisé pour mesurer la densité et la bonne séparation des clusters.

Le score de silhouette prend en compte la distance intra-groupe entre l'échantillon et les autres points de données dans le même groupe et la distance intergroupe entre l'échantillon

et le groupe le plus proche. Le score de silhouette se situe dans la plage [-1, 1]. Un score de silhouette de 1 signifie que les clusters sont très denses et bien séparés. Un score de 0 signifie que les clusters se chevauchent. Un score inférieur à 0 signifie que les données appartenant aux clusters peuvent être fausses/incorrectes.

Les aspects à surveiller dans les diagrammes de silhouette sont les scores de cluster inférieurs au score de silhouette moyen, les grandes fluctuations dans la taille des clusters, ainsi que l'épaisseur du diagramme de silhouette.

Nous avons examiné les silhouettes pour un nombre de clusters variant entre 2 et 50 clusters. Il est malheureusement difficile de trouver un nombre optimal de clusters dont les silhouettes sont toutes positives et fluctuent de façon plus ou moins similaire, mais proche. En effet, le meilleur nombre de clusters que nous avons est 17 clusters (ce qui confirme l'hypothèse faite sur le coude. Néanmoins, comme le montre le graphique suivant, certains clusters ne sont pas fiables. En classification non supervisée, il arrive souvent qu'il y ait certains clusters qui regroupent les données qui n'ont pas pu être directement associées à un groupe spécifique (elle ne sont pas suffisamment similaires avec les autres données).

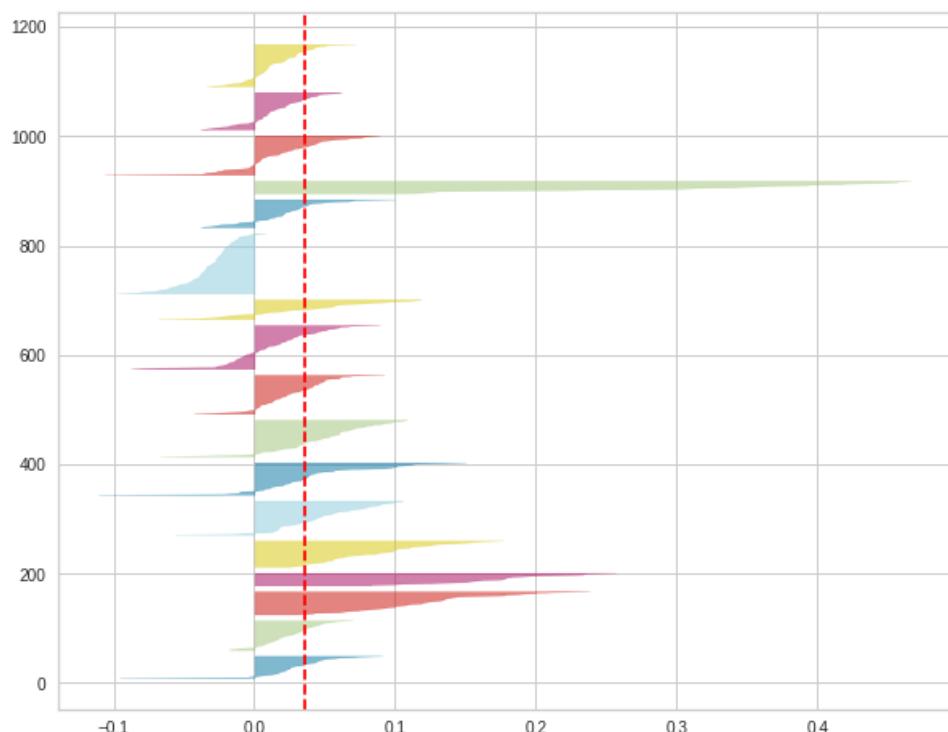


FIGURE 50 – Silhouette pour 17 clusters

Analyse des clusters résultants

Nous pouvons observer que certains clusters sont bien définis, analysons-les plus en détail. L'algorithme K-means reste très simple en théorie, néanmoins il arrive souvent à donner de bons résultats. Dans notre cas, nous avons d'abord appliqué le K-means sur le dataframe regroupant les titres des recettes. Néanmoins, les résultats n'ont pas été concluant. En effet, les titres peuvent paraître porteurs d'information concernant la recette en question pour

l'être humain. Néanmoins, dans le cadre d'un modèle de machine learning, il n'y a pas assez de matière pour distinguer les différences et les similitudes entre les données. Le dataframe instructions quant à lui est beaucoup plus riche en informations. Nous lui appliquons donc, indépendamment des titres des recettes, les transformations vues précédemment.

Les 17 clusters obtenus, regroupent un nombre différent de recettes. Le tableau suivant montre les proportions des différents clusters.

N° Cluster	Nombre de recettes	Proportion des données	Caractéristiques identifiées
0	4174	3%	Boissons, cocktails, jus
1	3256	3%	Recettes à base de viande de porc
2	4235	3%	Recettes avec des produits dérivés de porc
3	3654	3%	Cookies, biscuits
4	15055	12%	Recette amérique latine : Burritos, chili, guacamole
5	2095	2%	Recette de porc rôti, barbecue américain
6	3398	3%	Recettes italiennes : Pâtes, Pizza, fromage et jambon italiens
7	5700	5%	Recette à base de volaille : Poulet, dinde
8	5876	5%	mitigées
9	6125	5%	mitigées
10	5274	4%	Gâteaux, gâteau
11	4505	4%	Boisson fraîches/Glacées : Smoothie, glace, milkshake
12	19728	16%	mitigées
13	10745	9%	Cake et tarte
14	14445	12%	mitigées
15	7489	6%	Recette pour appétit
16	8893	7%	mitigées

FIGURE 51 – Descriptif des 17 clusters

Prenons pour exemple, le cluster N°0 : En analysant son contenu, on remarque que les recettes sont pour la plupart des recettes de boisson : On y retrouve notamment des boissons alcoolisées (cocktails) tels que Bloody Mary, Cosmopolitan, Mojito. Nous retrouvons par ailleurs des jus, limonades ou encore des smoothie, aux différents fruits, mais en plus petite quantité. On peut d'ailleurs l'observer dans le nuage de mots suivant :

Un autre exemple, le cluster N° 2 contient majoritairement des recettes contenant des produits à base de porc : des lardons, du bacon, du jambon... Ce qui est intéressant, c'est que certaines recettes ne font pas directement référence au porc, mais bien à ses produits dérivés. L'algorithme arrive à faire la distinction entre une recette à base de lardons (tel que la tartiflette) ou encore des salades ou des hamburgers contenant du bacon et une recette qui va contenir de la viande de porc comme du ragoût de porc.

Par exemple : Un sandwich au jambon aura plus de chance d'être classé dans le cluster N°2, alors qu'un sandwich à la viande de porc sera plutôt dans le cluster N°1.

Clustering hiérarchique Dans le clustering hiérarchique, les clusters sont combinés de manière itérative et hiérarchique, pour finalement aboutir sur une racine, qui regroupe toutes les données.

Agglomerative Hierarchical Clustering

Cette technique permet d'attribuer chaque point à un cluster individuel. Supposons qu'il y ait 4 points de données. Nous allons affecter chacun de ces points à un cluster et nous aurons donc 4 clusters au départ. Ensuite, à chaque itération, nous fusionnons la paire de clusters la plus proche et répétons cette étape jusqu'à ce qu'il ne reste qu'un seul cluster (la racine).

Nombre de clusters Le dendrogramme, permet de visualiser la hiérarchie en partant de

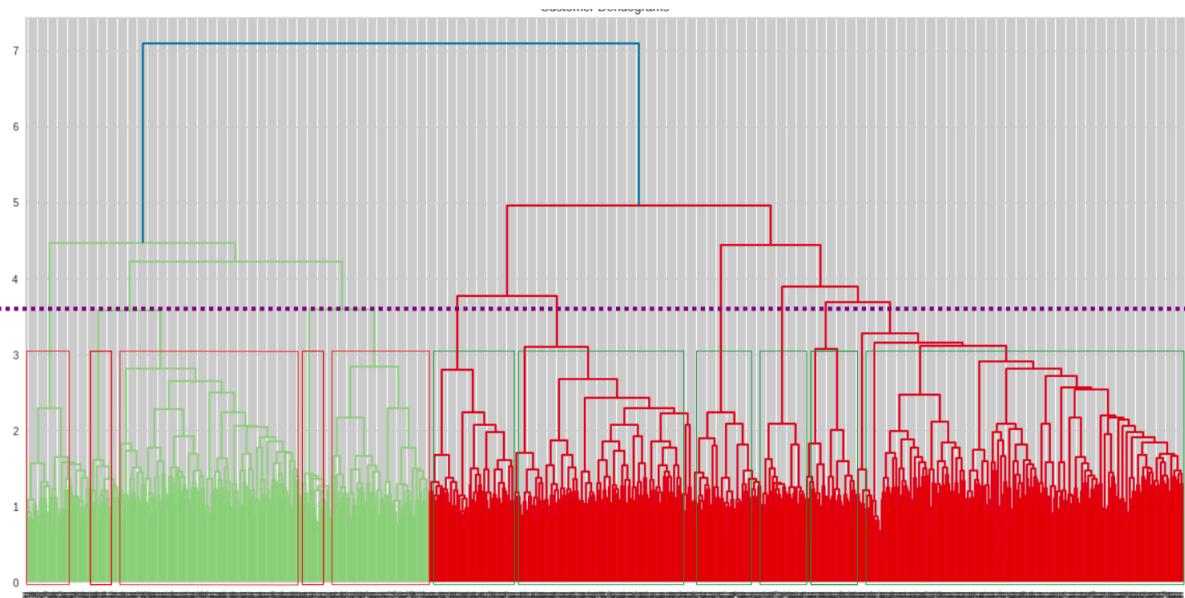


FIGURE 52 – Dendrogramme de la classification hiérarchique des recettes

chaque point jusqu'à parvenir à la racine.

Sur le graphique, on peut déterminer la profondeur à partir de laquelle on juge intéressant de découper les cluster. Il faut bien évidemment choisir un nombre de clusters qui découpe de façon distinguable les groupe. Nos allons choisir 10 clusters.

En analysant le contenu des clusters, on remarque certaines similitudes avec K-means. En effet, on retrouve certaines recettes regroupées ensemble dans les deux modèles. Néanmoins, la classification hiérarchique, en plus d'être beaucoup plus lente, suggère un nombre de clusters plus petit que K-means. Les résultats sont également moins bons. On distingue moins facilement les différences et les similitudes entre les recette d'un même cluster, ce qui implique qu'il est compliqué de trouver du sens dans cette classification.

Nous en concluons donc que K-means est meilleur.

6.5.3 Conclusions de la classification

L'objectif de cette partie était d'expérimenter une classification automatique sur des données textuelles et de voir si les méthodes appliquées pourraient distinguer des différences et des similitudes entre des recettes de cuisine. Comme nous avons pu l'observer, l'algorithme K-means peut effectivement mieux répondre à cette problématique. L'application de K-means sur des données textuelles possède des avantages et des inconvénients, voici ce que l'on peut retenir de cette étude :

- Avantages

- Il est facile d'implémenter k-means et d'identifier des groupes de données inconnus à partir d'ensembles de données complexes. Les résultats sont présentés de manière rapide.
- K-means convient à un grand nombre d'ensembles de données et est calculé beaucoup plus rapidement que le plus petit. Il peut également produire des clusters plus élevés.
- Les résultats sont très faciles à interpréter. K-Means génère des descriptions de cluster sous une forme minimisée pour maximiser la compréhension des données. En

effet, comme nous l'avons vu, il est possible de trouver du sens à la façon dont le modèle effectue la classification.

- Enfin, le coût de calcul est relativement faible : Comparée à l'utilisation d'autres méthodes de classification, une technique de classification k-means est rapide et efficace en termes de coût de calcul, en effet sa complexité est $O(K * n * d)$.
- Inconvénients
 - K-means ne permet pas de développer un ensemble optimal de clusters et vous devez choisir les clusters avant pour des résultats effectifs.
 - Les résultats diffèrent d'une exécution à l'autre : En effet, en plus du changement des noms des clusters, K-MEANS peut classifier un même point dans de clusters différents lorsqu'on réexécute le modèle (fit) et ce même si les paramètres et le jeu de données sont inchangés.
 - Le principal inconvénient reste la définition du nombre de clusters. En effet, pour que la classification par K-means soit efficace, il faut spécifier le nombre de clusters (K) au début de l'algorithme, ce qui n'est pas toujours un exercice évident, surtout lorsqu'on a un grand jeu de données et aucune hypothèse sur les données.

7 Conclusion du projet

Pour conclure le projet, on a pu avoir une base de données adaptée à notre problème en fusionnant les images des ingrédients (pomme, banane, orange, carotte) entre elles. On a décidé de choisir le modèle Yolov3 pour détecter nos ingrédients en se basant sur les avantages de ce modèle. Après la détection des aliments, les recettes ont été générées grâce à GTP2 qui est un modèle NLP basé sur des méthodes d'apprentissage automatique d'apprentissage non supervisé, parmi les recettes générées, on a obtenu quelques-unes qui contenaient l'ingrédient détecté. L'analyse de text est un outil puissant pour extraire de l'information des données. En effet, elle peut être utilisée, autant pour les classifier, que pour permettre de faire des hypothèses sur la composition d'un grand jeu de données et de pouvoir l'exploiter par la suite. Nous avons pu montrer, que malgré des résultats imparfaits, nous parvenons tout de même à analyser la composition du jeu de données des recettes et à trouver des similitudes entre celles-ci. La principale difficulté est d'arriver à obtenir un data set propre et exploitable avec le minimum de bruit dans les données. Il est donc primordial de ne pas négliger la partie de préparation des données, car celle-ci influe considérablement sur la qualité du clustering. Le modèle K-means, malgré sa simplicité sur le plan théorique, présente des résultats plutôt cohérents, qui contiennent un biais étant donné la complexité des données textuelles, mais présente néanmoins une forme de logique dans la classification et une facilité d'interprétation et d'analyse.

7.1 Pistes d'amélioration

Partie reconnaissance des ingrédients :

- Dans notre projet, on s'est limité à quatre ingrédients, mais on peut agrandir notre base de données et rajouter des ingrédients supplémentaires que l'on retrouve le plus dans les recettes de cuisine (œuf, beurre, sel, etc), de cette façon, on se rapprochera le plus à la réalité.
- Pour détecter les ingrédients sur une image, on utilise le modèle de détection Yolov3. On a d'autres modèles de détections qui sont aussi utilisés et qui sont connus par leurs performances (Faster R-CNN with Inception v2, RetinaNet with Resnet 50, etc).

Partie génération de recettes :

- Une des approches à l'état de l'art, en dehors des transformers est l'utilisation de GAN pour la génération de texte. [Yizhe Zhang 1 Zhe Gan 1 Kai Fan 1 Zhi Chen 1 Ricardo Henao 1 Dinghan Shen 1 Lawrence Carin 1]
- Une nouvelle version de GPT2 est sortie il y a peu de temps. Il s'agit de GPT3 qui est encore plus performant que ne l'était déjà GPT2, il possède 175 milliards de paramètres. L'accès au modèle est payant et n'a pas été rendu en open-source

Partie analyse descriptive des recettes :

- Une piste d'amélioration pour cette partie est d'utiliser les recettes d'instructions du dataset pour créer des séquences (suite d'instructions). On pourrait alors à partir de ses séquences faire de l'analyse descriptive pour voir les parties d'une recette allant souvent ensemble par exemple. Le clustering de séquence est très utilisée dans le milieu biologique avec le clustering de séquence d'ADN. L'état de l'art actuel sur cette partie étant de réaliser une représentation des séquences dans un espace vectoriel via utilisation

de réseaux de neurons apprenant la fonction de représentation. [SENSE : Siamese neural network for sequence embedding and alignment-free comparison (2019)]

Référence

- Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. 25th International Conference on Neural Information Processing Systems, (pp. 1097-1105). Lake Tahoe, Nevada.
- Simonyan, K., Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. 3rd IAPR Asian Conference on Pattern Recognition (ACPR), (pp. 730-734). Kuala Lumpur.
- He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep Residual Learning for Image Recognition. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (pp. 770-778). Las Vegas, NV.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2015). Going deeper with convolutions. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), (pp. 1-9). Boston, MA.
- Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z. (2016). Rethinking the inception architecture for computer vision. Proceedings of the IEEE conference on computer vision and pattern recognition, (pp. 2818-2826).
- Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. Thirty-First AAAI Conference on Artificial Intelligence.
- Weiqing, M., Shuqiang, J., Linhu, L., Yong, R., Ramesh, J. (2019). A Survey on Food Computing. ACM Computing Surveys.
- Mezgec, S., Koroušić, S. (2017). NutriNet : A Deep Learning Food and Drink Image Recognition System for Dietary Assessment. Nutrients, 9(7), 657.
- Ciocca, G., Napoletano, P., Schettini, R. (2018). CNN-based features for retrieval and classification of food images. Computer Vision and Image Understanding, 176-177, 70-77.
- Matsuda, Y., Hoashi, H., Yanai, K. (2012). Recognition of Multiple-Food Images by Detecting Candidate Regions. Proc. of IEEE International Conference on Multimedia and Expo (ICME).
- Kawano, Y., Yanai, K. (2014b). Food Image Recognition with Deep Convolutional Features. Proc. of ACM UbiComp Workshop on Cooking and Eating Activities (CEA), (pp. 589-593).
- Bossard, L., Guillaumin, M., Van Gool, L. (2014). ood-101 – Mining Discriminative Components with Random Forests. European Conference on Computer Vision (pp. 446-461). Springer, Cham.

Kagaya, H., Aizawa, K., Ogawa, M. (2014). Food Detection and Recognition using Convolutional Neural Network. 22nd ACM international conference on Multimedia, (pp. 1055-1088). Orlando, FL, USA.

Yanai, K., Kawano, Y. (2015). Food Image Recognition using Deep Convolutional Network with Pre-Training and Fine-Tuning. IEEE International Conference on Multimedia Expo Workshops, (pp. 1-6). Turin, Italy.

Christodoulidis, S., Anthimopoulos, M., Mougiakakou, S. (2015). Food recognition for dietary assessment using deep convolutional neural networks. International Conference on Image Analysis and Processing (pp. 458-465). Springer, Cham.

Singla, A., Yuan, L., Ebrahimi, T. (2016). Food/Non-Food Image Classification and Food Categorization using Pre-Trained GoogLeNet Model. 2nd International Workshop on Multimedia Assisted Dietary Management, (pp. 3-11). Amsterdam, The Netherlands.

Liu, C., Cao, Y., Luo, Y., Chen, G., Vokkarane, V., Ma, Y. (2016). DeepFood : Deep Learning-Based Food Image Recognition for Computer-Aided Dietary Assessment. 14th International Conference on Inclusive Smart Cities and Digital Health, (pp. 37-48). Wuhan, China.

Hassannejad, H., Matrella, G., Ciampolini, P., DeMunari, I., Mordonini, M., Cagnoni, S. (2016). Food Image Recognition using Very Deep Convolutional Networks. 2nd International Workshop on Multimedia Assisted Dietary Management, (pp. 41-49). Amsterdam, The Netherlands

Ciocca, G., Napoletano, P., Schettini, R. (2017a). Food Recognition : A New Dataset, Experiments, and Results. IEEE Journal of Biomedical and Health Informatics, 21(3), 588-598.

Ciocca, G., Napoletano, P., Schettini, R. (2017b). Learning CNN-based Features for Retrieval of Food Images. In New Trends in Image Analysis and Processing – ICIAP 2017 : ICIAP International Workshops, WBICV, SSPandBE, 3AS, RGBD, NIVAR, IWBAAS, and MADiMa 2017 (pp. 426-434). Catania, Italy : Springer International Publishing.

Horiguchi, S., Amano, S., Ogawa, M., Aizawa, K. (2018). Personalized Classifier for Food Image Recognition. IEEE Transactions on Multimedia, 20(10), 2836-2848.

Yu, Q., Anzawa, M., Amano, S., Ogawa, M., Aizawa, K. (2018). Food Image Recognition by Personalized Classifier. 25th IEEE International Conference on Image Processing, (pp. 171- 175). Athens.

Torrey, L., Shavlik, J. (2009). Transfer Learning. In E. Soria, J. Martin, R. Magdalena, M. Martinez, A. Serrano, Handbook of Research on Machine Learning Applications (pp. 242-264). IGI Global.

Torrey, L., Shavlik, J. (2009). Transfer Learning. In E. Soria, J. Martin, R. Magdalena, M. Martinez, A. Serrano, Handbook of Research on Machine Learning Applications (pp. 242-264). IGI Global.

Benhard Sitohang, Saiful Akbar, and Masayu Leylia Khodra (2021).Implementation of Transfer Learning Using VGG16 on Fruit Ripeness Detection

J. Redmon and A. Farhadi. Yolo9000 : Better, faster, stronger. In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 6517–6525. IEEE, 2017.

S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn : Towards real-time object detection with region proposal networks. arXiv preprint arXiv :1506.01497, 2015.

I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy. Openimages : A public dataset for large-scale multi-label and multi-class image classification. Dataset available from <https://github.com/openimages>, 2017.

T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2117–2125, 2017.

T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco : Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

Kiourt, C., Pavlidis, G. and Markantonatou, S., (2020), Deep learning approaches in food recognition

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. arXiv preprint arXiv :1601.06733, 2016.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. CoRR, abs/1406.1078, 2014.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert : Pre-training of deep bidirectional transformers for language understanding, 2018.

Yizhe Zhang 1 Zhe Gan 1 Kai Fan 1 Zhi Chen 1 Ricardo Henao 1 Dinghan Shen 1 Lawrence Carin 1, Adversarial Feature Matching for Text Generation

SENSE : Siamese neural network for sequence embedding and alignment-free comparison
Wei Zheng, Le Yang, Robert J Genco, Jean Wactawski-Wende, Michael Buck, Yijun Sun 1.
Joseph Redmon, Ali Farhadi YOLOv3 : An Incremental Improvement (April 2018)

2. Joseph Redmon, Ali Farhadi Yolo9000 : Better, Faster, Stronger (2016)

3. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi You Only Look Once : Unified, Real-Time Object Detection (2016)

4. Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao YOLOv4 : Optimal Speed and Accuracy of Object Detection (2020)

5. Rokas Balsys YOLO v3 theory explained (July 2019)

6. How to Perform Object Detection With YOLOv3 in Keras

7. DarkNet Github Repo

8. DarkNet Site for YOLO

9. Keras(TF backend) implementation of yolo v3 objects detection

11. Cours sur la génération de texte délivré par Clément Chatelain à l'INSA Rouen pour la partie RNN et LSTM

12. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>

https://cs.stanford.edu/~quocle/paragraph_vector.pdf

Similarity Measures for Text Document Clustering - Anna Huang

http://www.xavierdupre.fr/app/mlstatpy/helpsphinx/c_clus/kmeans.html

<https://delladata.fr/kmeans/>

<https://aclanthology.org/W15-1509.pdf>

<https://conservancy.umn.edu/bitstream/handle/11299/215421/00-034.pdf?sequence=1&isAllowed=y>