École Centrale Lyon

Informatique
Projet Option
Rapport

Image Caption Generation

Élèves :
Reda Ayassi
Martin Perez
Mehdi Elion
Amine Chiban

Enseignant :
Liming CHEN

8 avril 2019

Table des matières

1 Introduction **2**
1.1 Article de référence . . . 2
1.2 Objectifs . . . 2

2 Modèle Théorique **3**
2.1 Architecture globale . . . 3
2.2 Encoder . . . 3
2.3 Decoder . . . 4
2.4 Mécanisme d'attention . . . 6
2.4.1 Soft . . . 6
2.4.2 Hard . . . 7
2.5 Word Embeddings . . . 8

3 Méthodologie **8**
3.1 Training . . . 8
3.1.1 Data . . . 8
3.1.2 Fonction de coût . . . 8
3.1.3 Protocole d'apprentissage . . . 9
3.2 Évaluation . . . 9

4 Implémentation **10**
4.1 Pre-processing des données . . . 10
4.2 L'encodeur . . . 10
4.3 Le décodeur . . . 10
4.4 L'entraînement et l'évaluation . . . 11
4.5 La prédiction avec le beam search . . . 11

5 Résultats **12**
5.1 Expérimentations . . . 12
5.1.1 Taille des mini-batch . . . 12
5.1.2 Taille du modèle . . . 12
5.1.3 Régularisation . . . 13
5.1.4 Teacher forcing . . . 14
5.2 Visualisation de l'attention . . . 14
5.3 Évaluation . . . 16
5.3.1 Évaluation qualitative . . . 16
5.3.2 Évaluation quantitative . . . 16

6 Conclusion et perspectives **17**
6.1 Perspectives . . . 17
6.2 Conclusion . . . 17

1 Introduction

De nos jours, le nombre d'applications potentielles de l'apprentissage profond est immense. C'est l'une des raisons pour laquelle cette méthode d'apprentissage a pu s'imposer dans le monde de l'apprentissage automatique. Plus spécifiquement dans les applications impliquant le traitement d'image, cet outil a permis une amélioration remarquable, notamment pour les applications de biométrie faciale, le diagnostic de cancer à partir des radiographies dans le domaine de la médecine, ou encore les voitures autonomes pour la détection d'obstacles et de panneaux de signalisation.

Durant ce projet, nous nous sommes consacrés à l'utilisation de ces réseaux de neurones pour développer une application qui génère des descriptions pour des images qu'on lui donne en entrée. Cette application doit non seulement déterminer l'ensemble du contenu (objets, être vivants...) d'une image, mais aussi exprimer les relations entre les éléments de cet ensemble. C'est un grand défi pour les algorithmes d'apprentissage, dans la mesure où, cette application est sensée synthétiser toutes les informations visuelles importantes d'une image à travers un langage, comme est capable de le faire l'être humain.

1.1 Article de référence

Notre étude s'est appuyée sur un article de recherche « *Show, Attend and Tell : Neural Image Caption Generation with Visual Attention*» dont la dernière version a été publiée en 2016. Combinant les dernières avancées du domaine de traitement d'images, notamment les réseaux de neurones convolutif (CNN) et les réseaux de neurones récurrents (RNN), sur des bases de données volumineuses dédiées à la classification d'images, le modèle de l'article permet de générer des descriptions se rapprochant encore plus des descriptions faites par un être humain. La particularité de ce modèle réside dans l'ajout du mécanisme d'attention au RNN, qui fait référence à la capacité humaine à repérer les zones d'une image les plus intéressantes, pour générer les mots les plus représentatifs.

Cet article nous a été très bénéfique, dans la mesure où il nous a servi de référence et nous a permis de mieux nous orienter dans nos recherches. Nous nous sommes basés à peu près sur le même pipeline général, tout en développant ses différentes parties de manière indépendante ou en s'inspirant de certains codes retrouvés sur internet.

1.2 Objectifs

Dans un premier temps, nous avions définis des objetifs intermédiaires afin d'avoir tous les pré requis nécessaires pour débuter la réalisation :

- La familiarisation avec les réseaux de neurones à convolution (CNN)
- Le développement du CNN
- La familiarisation avec les réseaux de neurones récurrents
- L'étude du cas LSTM
- L'étude des deux approches d'attention
- Le développement d'un LSTM
- L'intégration des parties

Les objectifs finaux de ce projet sont les suivants :

- Développement d'une application permettant de générer une description pour une image, en s'inspirant des principes de l'article.

- Apporter des améliorations à chacune des parties du pipeline pour obtenir de meilleurs résultats.

2 Modèle Théorique

Pour générer la description d'une image, cette dernière passe par un ensemble de composantes intervenantes dans l'architecture globale. Dans cette partie est consacrée à l'étude théorique de ces composantes.Dans un premier temps, nous allons présenter une vue générale du pipeline, puis nous le détaillerons.

2.1 Architecture globale

Sur la figure suivante, vous pouvez voir l'enchaînement des traitements que subit une image. Le CNN, est ce qui nous permet d'extraire les caractéristiques de l'image, celles-ci seront ensuite combinées avec l'attention à chaque itération, pour l'introduire dans le RNN.

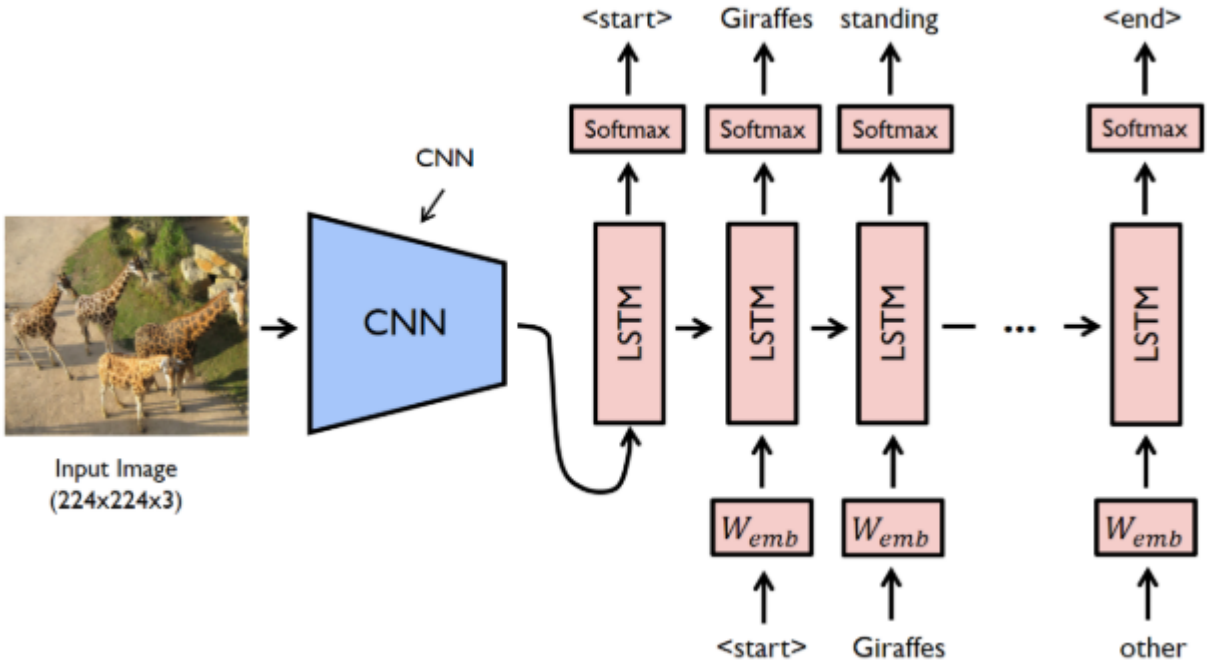


FIGURE 1 – Overview du pipeline

2.2 Encoder

Un encodeur permet de prendre une image brute, et d'en extraire les caractéristiques les plus pertinentes, et nous les retourne sous une certaine représentation matricielle. Les réseaux de neurones convolutionnels sont très performants et sont beaucoup utilisés pour ce genre d'application. Il existe plusieurs architectures de CNN (LeNet, GoogLeNet...), dans notre étude nous avons décidé d'utiliser VGG16.

VGG16 est un CNN qui permet de faire la classification d'images parmi 1000 catégories. Une image en entrée est représentée dans un espace à trois dimensions RGB, suivant

ce format 224x224x3. Cette image subit plusieurs transformations à travers le VGG16, pour en ressortir sous le format 14x14x512 représentant ses caractéristiques.

FIGURE 2 – Architecture du VGG16

— 3x3 Conv 64 : convolution par 64 filtres de taille 3x3.
— pool/2 : fonction max pooling, pour une fenêtre de 2x2. Remplacer une fenêtre 2x2 par la valeur du max et donc devient 1x1.
— FC 4096 : Couche de réseau de neurones simple avec 4096 noeuds

Comme nous le montre la figure précedente, le « 16 » du VGG16 représente le nombre de couche de notre CNN Une série de 5 bloc (couches de convolution et max pooling) pour l'extraction de features puis 3 couches pour la classification. Pour notre modèle, la partie du CNN qui nous intéresse est l'extraction des features, qui est faite par les 5 premiers blocs. On va donc devoir implémenter que cette partie après que le VGG entier ait appris à classifier.

Nous aurons donc le résultat suivant :

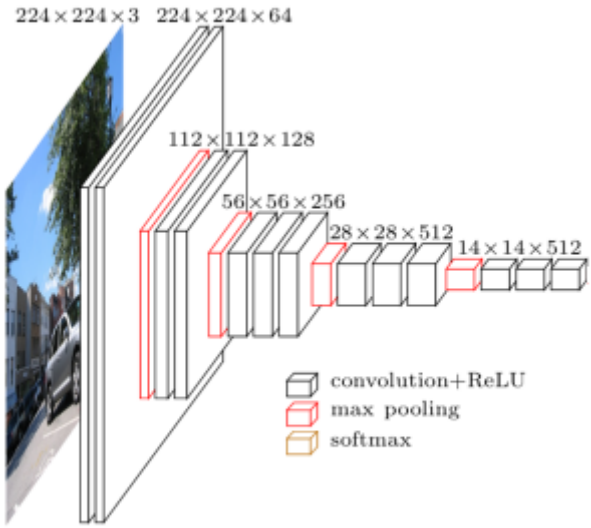


FIGURE 3 – Traitement de l'image dans l'encodeur après apprentissage

2.3 Decoder

Les réseaux récurrents (ou RNN pour Recurrent Neural Networks) sont des réseaux de neurones dans lesquels l'information peut se propager dans les deux sens, y compris des couches profondes aux premières couches. En cela, ils sont plus proches du vrai fonctionnement du système nerveux, qui n'est pas à sens unique.

Un LSTM fait partie de cette catégorie de réseau de neurones, sa particularité réside dans le fait qu'il garde en mémoire toutes les informations qu'il avait vu et prédit avant chaque itération. Cet aspect est très important car les RNN simples ne pouvait pas garder ces informations pour des longues séquences et finissaient par prédire des outputs ayant déjà été prédits. La figure suivante explique ce qui se passe dans une cellule LSTM.

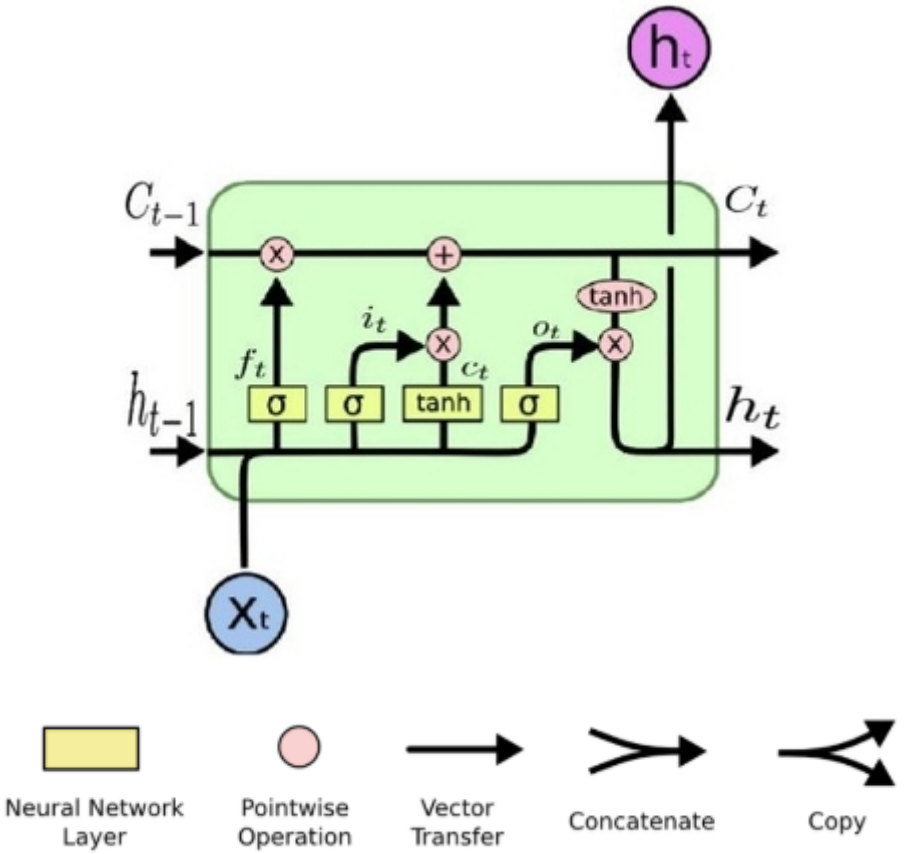


FIGURE 4 – Cellule LSTM

- X_t : Features de l'image (+attention)
- f_t : Contient l'information à supprimer de la mémoire (qui n'est plus nécessaire)
- i_t : Contient l'information à ajouter à la mémoire (importante)
- o_t : Utile pour le calcul de la Cell state
- c_t : Mémoire de la cellule à l'instant t
- h_t : Mot prédit

Le calcul de ces variable se fait de la manière suivante :

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$
$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * \tanh(c_t)$$

Avec W : les poids internes du LSTM, que ce derniers apprend.

2.4 Mécanisme d'attention

En regardant une image, l'être humain a tendance à se focaliser sur certaines zones qu'il considère importantes. Et pour fournir une description, il se base instinctivement sur ces zones. Afin d'imiter ce fonctionnement, un mécanisme d'attention sera utilisé pour orienter le RNN (LSTM) pendant ses prédictions.

Le mécanisme d'attention consiste à focaliser le décodeur sur certaines parties de l'image, ou plutôt sur certaines features issues de l'encodeur (que l'on associe alors à certaines régions de l'image). Concrètement, cela revient, à chaque itération du LSTM, à pondérer les features par des poids d'attention que l'on calcule en fonction de l'état caché précédent du LSTM et des features elles-mêmes. Le schéma ci dessous permet d'illustrer ce principe.

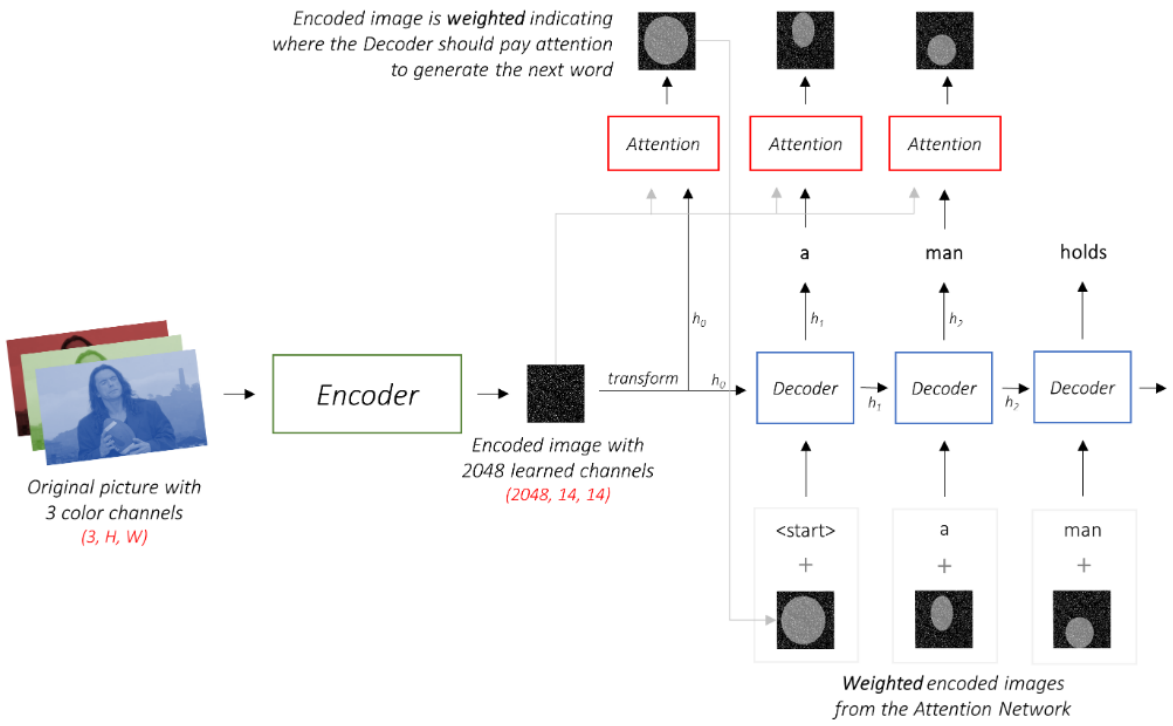


FIGURE 5 – Architecture générale avec mécanisme d'attention

L'article de référence mentionne deux types d'attention : la "soft" attention, qui découle d'une approche déterministe, et la "hard" attention, qui découle d'une approche stochastique. Le principe de ces deux approches sont expliqués dans les paragraphes suivants.

2.4.1 Soft

L'idée générale de la "soft" attention est de générer, à chaque itération t du LSTM, des "attention features" $e_{t,i}$ à partir du hidden state précédent h_{t-1} et des features a_i. Pour ce faire, on utilise deux perceptrons multicouches $\mathcal{W}_{hh}$ et $\mathcal{W}_{ih}$ suivis d'une tangente hyperbolique.

$$e_{t,i} = tanh(\mathcal{W}_{hh} h_{t-1} + \mathcal{W}_{ih} a_i) \tag{1}$$

Une fois que l'on dispose des ces "attention features", on les utilise pour calculer les poids d'attention $\alpha_{t,i}$. Pour ce faire, on leur applique un $softmax$ afin d'obtenir des valeurs comprises entre 0 et 1.

$$\alpha_{t,i} = \frac{exp(e_{t,i})}{\sum_{k=1}^{L}(exp(e_{t,k})} \in [0,1] \tag{2}$$

Les informations z_t que l'on passe au décodeur à l'itération t s'expriment alors ainsi :

$$z_t = \sum_i \alpha_{t,i} a_i \tag{3}$$

Puisque toutes ces opérations sont différentiables, on peut procéder à une descente de gradient classique sur une fonction de perte. Dans notre cas, nous avons utilisé une fonction de perte de type "Cross Entropy" avec une descente de gradient selon l'algorithme "Adaptive Momentum".

2.4.2 Hard

Dans cette approche stochastique, au lieu de pondérer les features par les coefficients $\alpha_{t,i} \in [0,1]$, on décide d'interpréter ces coefficients comme des probabilités de tenir compte, ou non, des features. Autrement dit, on construit une variable aléatoire dite $s_t = \{s_{t,i}\}$ qui prend ses valeurs dans $\{0,1\}$ et qui suit une distribution de multinoulli paramétrée par les $\{\alpha_{t,i}\}$, ce qui signifie que chaque variable aléatoire $s_{t,i}$ suit une loi de Bernoulli de probabilité $\alpha_{t,i}$.

$$s_t \hookrightarrow Multinoulli(\{\alpha_{t,i}\}) \tag{4}$$

$$p(s_{t,i} = 1 | s_{j<t}, a) = \alpha_{t,i} \tag{5}$$

Étant donné que l'on perd la propriété de différentiabilité à cause de cette variable aléatoire, on ne peut plus procéder comme précédemment. Ainsi, plutôt que de considérer une fonction de perte, on va considérer une fonction objectif qu'il va falloir maximiser (en pratique, on minimise l'opposé de la fonction objectif). Cette fonction L_s est un minorant de la log-vraisemblance marginale $\log p(y|a)$ où y représente la légende que l'on génère.

$$L_s = \sum_s p(s|a) \log p(y|s,a) \tag{6}$$

$$L_s \leq \log p(y|a) \tag{7}$$

Ne pouvant pas calculer un gradient par rétro-propagation, l'approche la plus répandue dans ce cas consiste à approximer une valeur du gradient par un échantillonnage de type Monte-Carlo.

$$\frac{\partial L_s}{\partial W} \approx \frac{1}{N} \sum_{n=1}^{N} \left[\frac{\partial \log p(y|s^n, a)}{\partial W} + \log p(y|s^n, a) \frac{\partial \log p(s^n|a)}{\partial W} \right] \tag{8}$$

Les informations que l'on passe au décodeur s'expriment alors ainsi :

$$z_t = \sum_i s_{t,i} a_i \tag{9}$$

2.5 Word Embeddings

Les "word embeddings" sont des vecteurs permettant de représenter chaque mot par ses coordonnées dans un espace de représentation abstrait de dimension d.

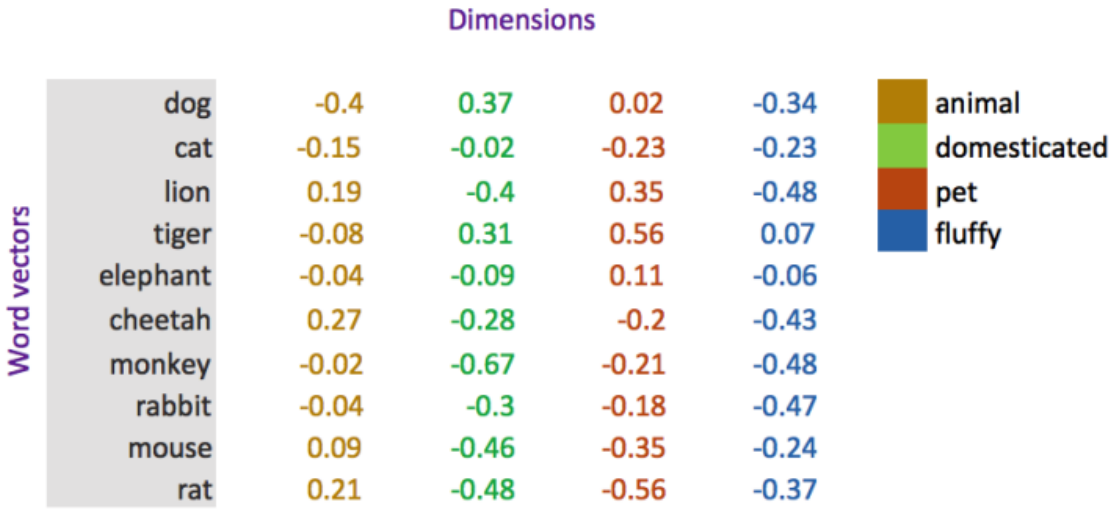


FIGURE 6 – Exemple de word embeddings

Sur l'exemple ci-dessous, les dimensions des word embeddings ont une signification intelligible mais ce n'est pas nécessairement la cas. Dans notre cas, on fixe une dimension pour les word embeddings et on les inclut dans le processus de retro-propagation de la phase d'apprentissage. On obtient alors des word embeddings abstraits dont on ne comprend pas nécessairement le sens mais qui assurent de bonnes performances.

3 Méthodologie

3.1 Training

3.1.1 Data

Pour le dataset Flickr8k, il existe un split standardisé avec 6000 images de pour l'apprentissage, 1000 images de validation et 1000 images de test, avec 5 légendes par image. Les images sont pré traitées par l'encoder (VGG16) et les images déjà encodées sont stockées avant l'apprentissage, ce qui permet de nettement accélérer l'entraînement du modèle.

3.1.2 Fonction de coût

Pour l'apprentissage, le modèle est entraîné par mini-batch tirés parmi les images encodées, et le décodeur est entraîné end-to-end par back propagation avec l'algorithme Adam. Pour prédire une légende et ainsi calculer un coût, il existe deux façons de procéder :

- cas nominal : au moment de prédire le n-ième mot de la légende d'une image, le $(n-1)$-ième mot prédit est passé en entré du LSTM du décodeur,
- cas teacher-forcing : au moment de prédire le n-ième mot de la légende d'une image, le $(n-1)$-ième mot de la légende de référence est passée en entré du LSTM du décodeur.

L'ajout de teacher forcing permet en particulier d'accélérer l'apprentissage et donne potentiellement de meilleurs résultats : à chaque epoch, un tirage aléatoire permet de calculer la fonction de coût avec ou sans teacher forcing, et l'on peut utiliser plus ou moins fréquemment de teacher forcing avec un paramètre teaching forcing ratio.

Pour la fonction de coût ou loss function, la cross-entropie est utilisée pour chaque mot de la légende prédite, puis sommée sur tous les mots. Notons N la taille maximale d'une légende, n la taille de la légende de référence et k la taille de la légende prédite (définie par l'instant où le décodeur prédit le token de fin de phrase <end>). Dans le cas de teacher-forcing, la cross-entropie est calculée pour chaque indice de mot allant de 1 à n.

Dans le cas sans teacher-forcing, on peut calculer la loss en prenant tous les mots d'indice allant de 1 à n, ou de 1 à $\max(k, n)$ ou encore de 1 à N en rembourrant les légendes avec un token de rembourrage (<pad>) jusqu'à N. En pratique, en calculant la cross-entropy sur les mots d'indices 1 à n ou 1 à $\max(k, n)$, le modèle après apprentissage a tendance à ne pas prédire le token <end>. En revanche, en calculant la fonction de coût jusqu'à l'indice n (ce qui incluant les mots $< pad >$), la taille des légendes est largement pénalisée et le modèle apprend rapidement à prédire le token $< end >$ suivi d'une série de padding $< pad >$.

3.1.3 Protocole d'apprentissage

L'apprentissage est paramétré par plusieurs variables : le taux d'apprentissage ou learning rate, le taux de teacher forcing, le taux de dropout du décodeur et la taille des mini-batch.

L'apprentissage du décodeur est babysitté : l'entraînement consiste en une suite de séquences d'epoch à paramètres constants ; et, après chaque séquence, la courbe d'apprentissage et les BLEU score(s) sur les datasets d'apprentissage et de validation. En fonction de ces données, les paramètres d'apprentissage sont ajustés.

- si le BLEU score sur validation stagne : on diminue le learning rate,
- si le BLEU score de validation diminue ou stagne depuis un moment, l'apprentissage du modèle est arrêté et on reprend le modèle avant la dernière séquence d'apprentissage,
- éventuellement, les autres paramètres d'apprentissage sont ajustés.

3.2 Évaluation

Une fois que l'on a généré une légende à l'aide, nous avons besoin d'une métrique permettant de quantifier la similarité entre cette légende générée et la ou les légendes de référence. Pour ce faire, nous utilisons le BLEU score qui consiste en une mesure de similarité entre phrases basée sur des comparaisons entre séquences de mots de taille n (d'ordinaire n peut valoir 1, 2 ou 3).

4 Implémentation

Par rapport à l'implementation du modèle, nou nous somme basé sur le langage PYTHON ainsi que la bibliothèque de machine learning PYTORCH. Notre but était d'implémenter une solution sous forme d'un programme en ligne de commande, qui perd d'exécuter les différentes taches du pipeline séparément, et de manière configurable. Ainsi nous avons séparé la solution en plusieurs partie, qui sont présentés dans les points suivants, en plus des différents choix d'implémentation que nous avons pris.

4.1 Pre-processing des données

La première étape consiste à traiter les données bruts issues des data sets flickr et COCO, afin de les préparer à l'entrée du modèle. Cette transformation consiste généralement à :

- Curation des données texte. C'est à dire le nettoyage, filtrage et tokenisation des description d'images.
- Génération de la wordmap, qui sert de dictionnaire pour mapper les mots à des entiers.
- Stockage des fichier traités pour faciliter l'exécution. Nous avons utilisé par exemple le format de fichier h5py pour compresser les images.

4.2 L'encodeur

L'encoder correspond à l'implémentation du CNN du modèle. Nous avons repris le modèle du VGG16 déjà entraîné, qui est disponible dans PYTORCH, et nous l'avons modifier afin de convenir à nos besoins.

Nous avons par contre choisi d'encoder toutes les images du data set préalablement, puis de les stocké sous forme de fichiers source traités. Cela nous a permis de gagné du temps dans les phases d'entraînement et d'expérimentation, car le modèle ne comporte désormais que le décodeur et le modèle d'attention, donc il s'exécute plus rapidement. Puisque nous avions décidé de ne pas entraîner l'encodeur en même temps avec le modèle, ce choix n'a pas impacter les résultats du modèle.

4.3 Le décodeur

Nous avons essayer d'implémenter le décodeur le plus fidèlement que possible par rapport aux modèle de référence. L'encodeur consiste en plusieurs modules, dont le LSTM et le modèle d'attention, et est décrit par les caractéristiques suivantes :

- Input : Une batch de [Image,captions,Taille captions]
- Output : Un ensemble de scores permettant de prédire les mots de la description, ainsi que les coefficients du modèle d'attention à chaque itération.
- Paramètres : Taille du batch, learning rate, taille des différents modules, Taux de teacher forcing

En plus de ces paramètres, le décodeur nécessite un nombre important de choix concernant sa structure interne. Nous avons ainsi expérimentés avec différentes version du modèle en jouant sur les points suivants :

- Embedding : permet de codifier les mots des description sous format vectoriel. Nous avons choisi d'entraîner le modèle d'Embedding en même temps que le décodeur.

- Régularisation : Nous avons opté pour une régularisation Dropout, mais nous avons expérimentés par rapport à l'emplacement du Dropout.
- Fonction de cout et optimiseur : La fonction de coût utilisée est la CrossEntropy. Deux optimiseur ont été testés, ADAM, et RMSPROP.
- Initialisation du LSTM : Les deux options testés sont l'initialisation à zéro, et l'initialisation avec une couche linéaire.
- Le teacher forcing : Le décodeur utilise un pourcentage de teacher forcing paramétré. C'est à dire qu'avec 30% de forcing, le décodeur à une probabilité de 0.3 d'activer le forcing à chaque itération.

4.4 L'entraînement et l'évaluation

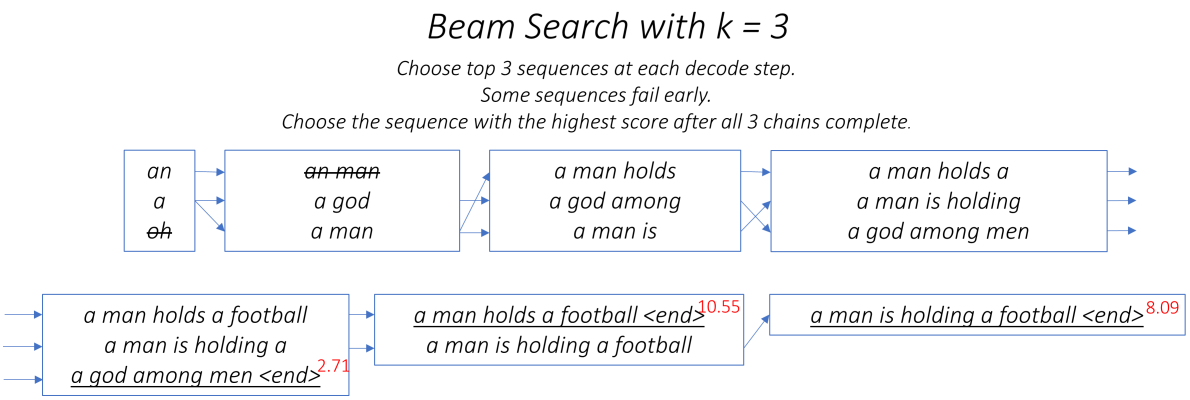
Un script d'entraînement permet de lancer la tache avec les paramètres souhaiter afin de faciliter les expérimentations. Les modèles issue de l'entraînement sont sauvegardés sous forme de check points, pour être utilisés ultérieurement pour d'autre taches.

L'entraînement utilise le corpus 'TRAIN' pour changer les poids du modèle, et le corpus 'VAL' afin de valider les performances du modèle. La validation sert principalement à changer le learning rate pour améliorer l'entraînement.

Le modèle est ensuite évalué sur le corpus 'TEST', afin de noter les benchmark du modèles entraînés.

4.5 La prédiction avec le beam search

Enfin, nous avons implémenter une méthode de prédiction qui repose sur l'algorithme du beam search afin d'améliorer les descriptions prédites. Cet algorithmes permet d'explorer un nombre k de possibilités des prédiction des mots, au lieu de sélectionner tout simplement celui avec le meilleur score. Nous n'avons pas incorporés cette prédiction dans l'évaluation des modèles.



5 Résultats

5.1 Expérimentations

L'optimisation des performances du modèle (BLEU score) nous a conduit à réaliser un certain nombre d'expérimentations sur les paramètres de l'apprentissage et du modèle.

5.1.1 Taille des mini-batch

Après avoir entraîné un modèle et constaté que la loss ne dépassait pas un certain seuil, l'idée a été de diminuer la taille des mini-batch de sorte que les paramètres du modèle soit actualisé pour pratiquement toutes les images et ainsi overfit les données d'apprentissage. En diminuant la taille des batch de 16 à 8 on constate en fait que la training loss oscille grandement et ne diminue en moyenne pas. Avec des batch de taille plus grande comme 32 ou 48 en revanche, la loss oscille beaucoup moins et diminue davantage.

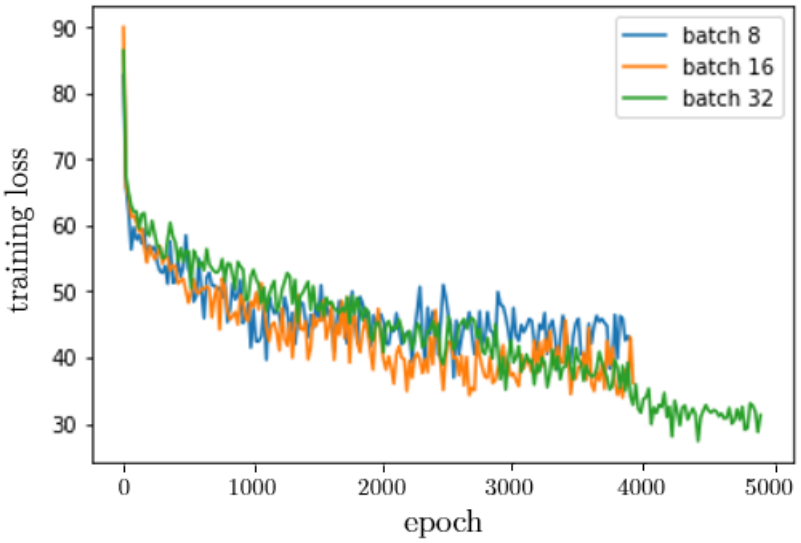


FIGURE 7 – Training loss en fonction de la taille des mini-batch

Taille des mini-batch	BLEU-1 (training set)	BLEU-1 (test set)
8	42.1	40.6
16	44.2	41.6
32	48.2	44.7

5.1.2 Taille du modèle

En dehors du mécanisme d'attention, plusieurs paramètres du modèle sont variables : les hidden state et context vector du LSTM sont initialisés avec un Multi-Layer-Perceptron dont on peu moduler les dimensions, la taille du Word Embedding ou encore le nombre de dimensions des vecteurs du LSTM que l'on se propose d'étudier :

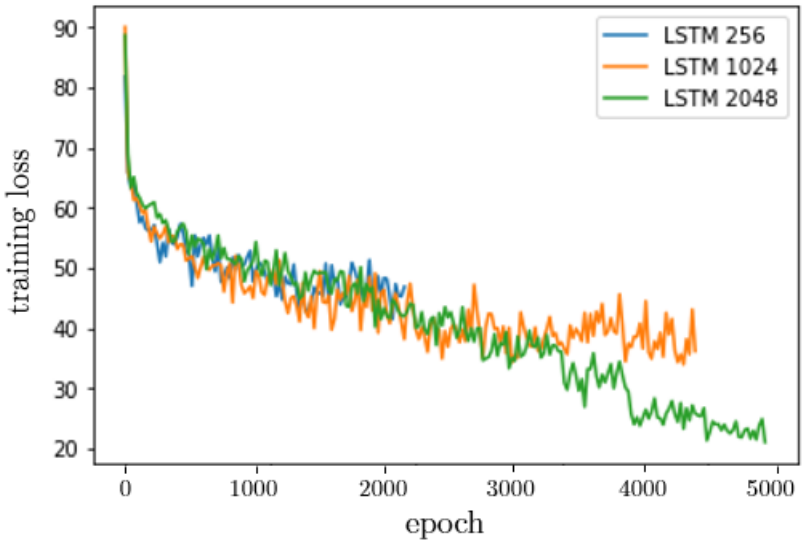


FIGURE 8 – Training loss en fonction des dimensions des vecteurs du LSTMCell

On remarque que la training loss converge rapidement vers un minimum local pour 256 dimensions, mais limite les capacités du modèle à apprendre sur les données. Avec 1024 dimensions, la loss diminue plus lentement mais prend des valeurs plus basses, tandis que pour des vecteurs de taille 2048 et en l'absence de régularisation, la loss diminue largement et n'a pas encore convergé au bout de 5000 epochs. Le modèle overfit alors le modèle, à la vue des différences de BLEU score entre les dataset de training et celui de validation :

Dimensions du LSTM hidden state	BLEU-1 (training set)	BLEU-1 (test set)
256	42.2	41.6
1024	50.2	45.9
2048	84.3	45.4

5.1.3 Régularisation

Avec un nombre de dimensions de 2048 pour le LSTM, le modèle overfit les données de Flickr8k. Un moyen de remédier à cela est d'introduire de la régularisation, par exemple avec du Dropout sur les réseaux de neuronnes. Nous avons optés pour ajouter du dropout sur le hidden state et le context vector du LSTM, ce qui donne les résultats suivants :

Taux de dropout	BLEU-1 (training set)	BLEU-1 (test set)
0.0	84.3	45.4
0.1	67.7	46.0
0.3	49.2	44.6

Moins il y a de dropout et plus il y a de différence entre les BLEU scores sur les données d'apprentissage et sur les données de validation. Le BLEU-1 score sur les données d'apprentissage est du même ordre qu'avec dropout ; néanmoins, au moment de prédire des légendes sur les données de validation ou de test, on note que, qualitativement, les prédictions n'ont pas toujours de sens pour le modèle entraîné sans dropout.

5.1.4 Teacher forcing

Plusieurs entraînement de modèle avec différents taux de teacher forcing donnent les résultats suivants :

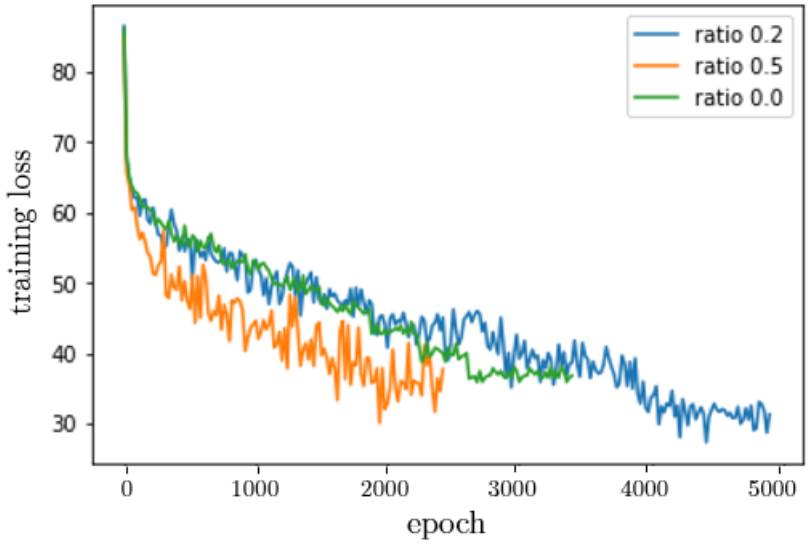


FIGURE 9 – Training loss en fonction du taux de teacher forcing

Pour un taux de teacher forcing relativement élevé (0.5), la convergence est rapide mais les résultats ne sont très bons : au moment d'évaluer le BLEU-score, il n'y a plus de teacher forcing et le BLEU score est relativement faible, même sur les données d'apprentissage (tableau ci-dessous). Lorsqu'il n'y a pas du tout de teacher forcing, la convergence est plus lente et les résultats ne sont pas bons non plus, aussi bien quantitativement que qualitativement : le modèle traduit souvent des suites de mots qui n'ont pas de sens. Cet effet peut être notamment attribué à l'entraînement du word embedding, qui s'entraîne en pratique mieux avec du teacher forcing.

Finalement, un taux de teacher forcing intermédiaire comme 0.2 donne les meilleurs résultats :

Taux de teacher forcing	BLEU-1 (training set)	BLEU-1 (test set)
0.0	46.5	38.5
0.2	67.7	46.0
0.5	52.4	42.7

5.2 Visualisation de l'attention

Les poids d'attentions étant une carte, il est possible d'afficher les poids d'attention à chaque mot prédit et de les superposer à l'image d'origine. Par exemple, sur une image au hasard des données de test on obtient les résultats suivants :

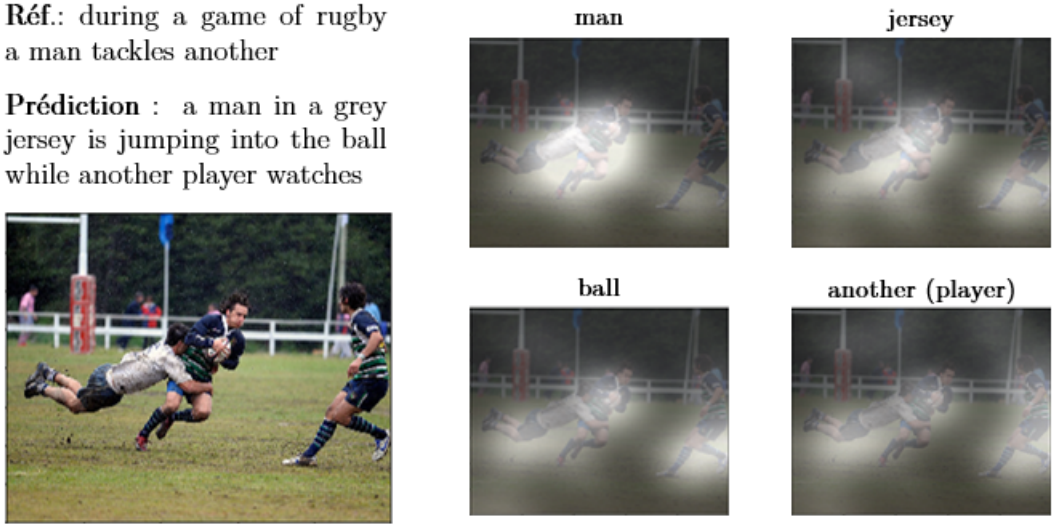


FIGURE 10 – Visualisation de l'attention (image de test 518)

On remarque le mécanisme d'attention arrive bien à capter les éléments de l'image dont il émet un mot (le joueur, le maillot, le ballon). Cependant, le mécanisme d'attention ne se concentre pas seulement sur l'élément : au moment de prédire le mot "ball" par exemple, l'attention est portée sur le ballon mais également sur les joueurs.

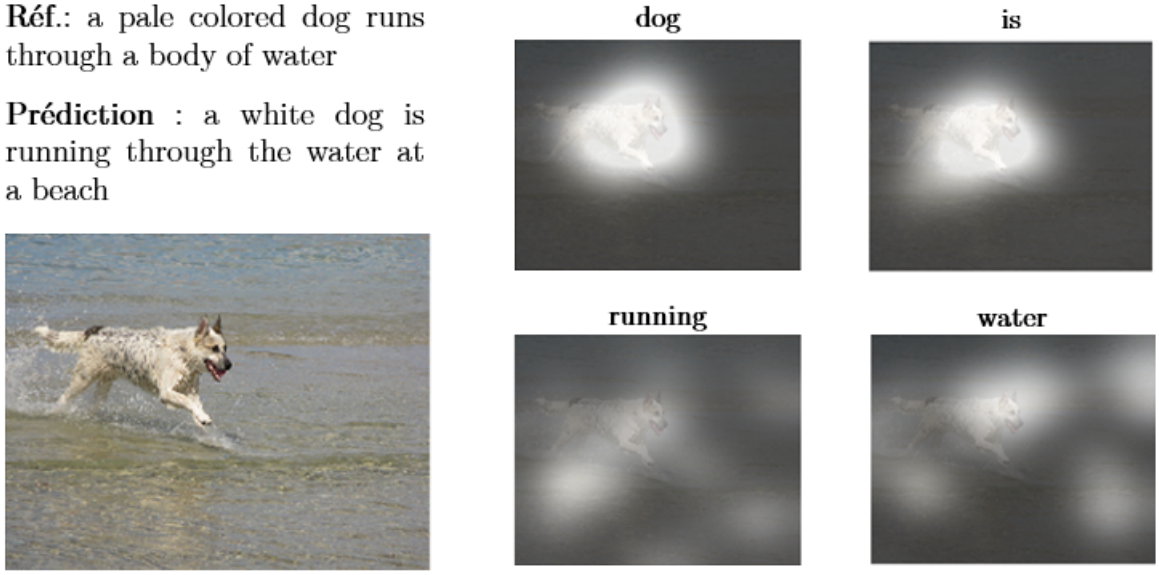


FIGURE 11 – Visualisation de l'attention (image de test 440)

Sur cette deuxième image, on voit que le chien est bien cerné par l'attention, et que l'attention est pratiquement inchangée pour le verbe être relatif au chien. Au moment de prédire "water", l'attention est répartie sur la surface de l'eau ; mais au moment de prédire le mot "running", il est difficile d'interpréter les poids d'attention.

De manière générale, le modèle a parfois du mal à capter ou non le mouvement : par exemple, dans le dataset Flickr8k, il y a de nombreuses scène avec un chien, très souvent qui coure ou saute. Il n'est pas rare que le modèle prédise qu'un chien coure sur une image de test alors que celui-ci est immobile comme sur l'exemple ci-dessous :

FIGURE 12 – Mauvaise prédiction de l'action de l'image

5.3 Évaluation

5.3.1 Évaluation qualitative

Sur la figure ci-dessous, deux exemples de prédictions sur des images de test sont données :

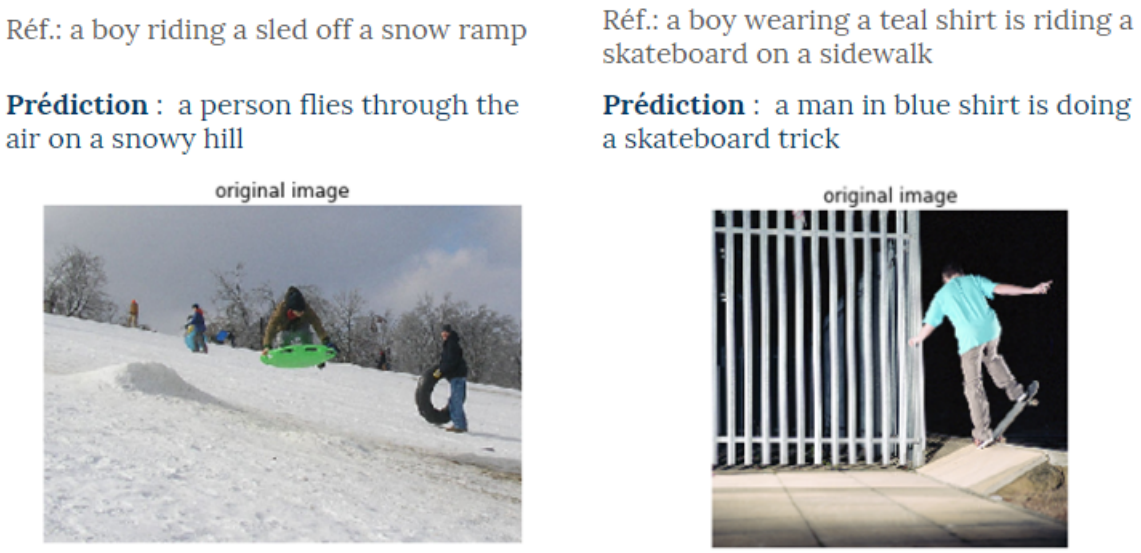


FIGURE 13 – Exemple de prédictions d'images de test

Sur ces deux exemples, les prédictions sont pertinentes : elles capturent l'objet saillant de l'image, une partie du contexte (mouvement) et certains détails. Le modèle est de manière général assez pertinent sur des images du dataset Flickr8k, où les images de test sont dans le même esprit que les images d'apprentissage.

5.3.2 Évaluation quantitative

Plusieurs modèles ont été entraînés dans le but d'obtenir le modèle le plus performant vis-à-vis du BLEU score. Le modèle retenu pour la prédiction de légendes correspond à celui dont la moyenne des BLEU-1, BLEU-2, BLEU-3 et BLEU-4 scores la plus élevée, avec les résultats suivants :

Modèle	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Soft (référence)	67.0	44.8	29.9	19.5
Soft (nôtre)	46.9	28.6	18.5	12.1

Les résultats ne sont pas au niveau de ceux de l'article de référence, mais restent tout de même acceptables. En particulier, les BLEU scores sont plus élevés avec le modèles d'attention que sans modèle d'attention, avec typiquement 35 à 40 de BLEU-1 score sans incorporer le modèle d'attention.

6 Conclusion et perspectives

6.1 Perspectives

- Expérimentations avec le data set COCO

Étant donnée la taille énorme du data set COCO, ce qui implique un temps d'entraînement considérable, nous n'avons pas eu la possibilité d'optimiser le modèle sur ce data set. Ainsi, le point d'amélioration principale serait d'effectuer plus de test sur COCO, car les performances des modèles de Deep learning, s'améliorent considérablement avec plus de données.

- Entraînement du VGG

Comme expliqué au niveau de la partie VGG, nous avons utilisé un VGG16 pré-entraîné sur un dataset de classification. L'idée sera de lancer l'apprentissage sur tout le pipeline y compris le VGG. D'après nos recherches, cette façon de faire permet au système de mieux apprendre, mais bien évidement le processus sera très coûteux.

6.2 Conclusion

En conclusion, nous avons le plaisir de constater qu'on est arrivé à développer une solution modulaire, et facilement configurable afin d'exécuter les différentes taches d'un pipeline "Machine learning". Ainsi nous avons pu expérimenter considérablement avec les différents paramètres du modèle, ce qui nous a permis de comprendre plus profondément l'effet de chaque élément du modèle sur les performances finales.

Ainsi la solution que nous avons développé peut être facilement reprise par d'autre personnes, et peut être étendue pour améliorer ses performances.

En outre, le résultat finale a été assez satisfaisant aussi. Car même si nos benchmarks n'étaient pas identique à celle de l'article de référence, notre modèle finale permet quand même de trouver des résultats assez satisfaisants.

Réf.: during a game of rugby a man tackles another

Prédiction : a man in a grey jersey is jumping into the ball while another player watches

original image

Poids d'attention

man

jersey

ball

another (player)