



L'Université Chouaib Doukkali (UCD)
Ecole Nationale des Sciences Appliquées
El Jadida.

IA GENERATIVE ET INGENIERIE DES PROMPTS

Science De Données Et Intelligence Artificielle

Compte Rendu du TP2 Attention

Préparé par :
M.ARIRI Abdelaziz

Professeur :
Pr.YOUNESS ABOUQORA

Année Académique 2025/2026

1 Introduction

Le domaine de la vision par ordinateur a connu des avancées majeures grâce au développement de modèles capables non seulement de classer des objets, mais aussi de décrire textuellement des scènes complexes. Ce projet s'inscrit dans cette dynamique à travers la mise en œuvre d'un système d'*Image Captioning* (légendage d'images).

L'enjeu principal est de construire un pont entre les caractéristiques visuelles extraites d'une image et la génération de langage naturel. Pour ce faire, nous utilisons le dataset **Flickr30k**, une référence contenant plus de 30 000 images annotées par des descriptions humaines, afin d'entraîner un modèle à traduire une information spatiale en une séquence temporelle de mots.

L'architecture retenue repose sur une approche de type **Encoder-Decoder** optimisée par un mécanisme d'attention. L'encodage est assuré par un réseau **ResNet50** pré-entraîné, dont les poids sont gelés afin d'exploiter le *Transfer Learning*. Cette stratégie permet de bénéficier de capacités d'extraction de caractéristiques déjà affinées sur de larges bases de données.

Les caractéristiques visuelles extraites sont ensuite transmises à un décodeur **LSTM (Long Short-Term Memory)** personnalisé. Contrairement à un LSTM classique, ce module intègre un mécanisme d'attention qui permet au modèle de se focaliser dynamiquement sur différentes régions de l'image lors de la génération de chaque mot. Les équations régissant ce mécanisme sont données par :

$$score_{att} = W_{att} \cdot \text{concat}(features, h_{t-1}) \quad (1)$$

$$context_vector = \sum (\text{softmax}(score_{att}) \cdot features) \quad (2)$$

Enfin, la performance et la cohérence sémantique du modèle sont renforcées par l'utilisation d'embeddings pré-entraînés **Word2Vec**. Ce choix permet de manipuler des représentations vectorielles de mots riches en informations sémantiques dès les premières phases d'apprentissage. Le processus présenté dans ce compte rendu couvre l'ensemble de la chaîne de production, depuis le prétraitement des données et la tokenisation, jusqu'à l'implémentation du LSTM avec attention et la boucle d'entraînement intégrant une planification du taux d'apprentissage (*Step Decay*).

2 Configuration de l'environnement Kaggle

Pour la réalisation de ce TP, nous avons utilisé l'environnement **Kaggle**, qui offre l'avantage de fournir des ressources de calcul GPU adaptées à l'entraînement de modèles de *Deep Learning*. Cette section décrit la préparation de l'environnement et les bibliothèques essentielles au pipeline de traitement.

2.1 Installation et importation des dépendances

L'implémentation repose principalement sur l'écosystème **PyTorch**. Les bibliothèques suivantes ont été utilisées :

- **PyTorch et Torchvision** : définition des réseaux de neurones (CNN et LSTM) et utilisation du modèle ResNet50 pré-entraîné.
- **TensorBoard** : visualisation en temps réel des courbes de perte et des métriques d'évaluation.
- **NLTK et Transformers** : traitement et tokenisation des séquences textuelles.
- **PIL et Pandas** : chargement des images et manipulation des fichiers d'annotations.

2.2 Code de configuration

Le bloc de code suivant illustre l'initialisation de l'environnement de travail :

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import torchvision.models as models
from torch.utils.data import Dataset, DataLoader
```

```
from torch.optim.lr_scheduler import StepLR
from torch.utils.tensorboard import SummaryWriter

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Utilisation du périphérique : {device}")
```

2.3 Gestion du matériel (GPU)

Compte tenu de la complexité de l'architecture combinant un **ResNet50** et un **LSTM avec attention**, l'utilisation du GPU NVIDIA disponible sur Kaggle est indispensable. Elle permet de réduire significativement le temps de calcul lors de l'extraction des caractéristiques visuelles et de la rétropropagation du gradient à travers le temps (*Backpropagation Through Time*).

3 Téléchargement et chargement du dataset

Le succès d'un modèle d'*Image Captioning* dépend étroitement de la qualité de l'association entre les données visuelles et textuelles. Cette étape est donc cruciale avant l'apprentissage.

3.1 Prétraitement et exploration des données

La première phase consiste à configurer l'environnement de traitement du texte, puis à exploiter le fichier de métadonnées du dataset **Flickr30k** afin d'établir la correspondance entre chaque image et ses descriptions associées.

Un nettoyage rigoureux des annotations est effectué pour corriger les erreurs de formatage, notamment la suppression des espaces superflus dans les noms de colonnes et les légendes. Cette étape garantit un accès fiable aux images lors de l'entraînement.

3.2 Statistiques et structure du dataset

L'exploration du jeu de données confirme la richesse de **Flickr30k**. Chaque image est associée à plusieurs légendes (en moyenne cinq), offrant une diversité linguistique favorable à la généralisation du modèle.

```
Colonnes du dataset : Index(['image_name', 'comment_number', 'comment'], dtype='object')
   image_name  comment_number \
0  1000092795.jpg             0
1  1000092795.jpg             1
2  1000092795.jpg             2
3  1000092795.jpg             3
4  1000092795.jpg             4

   comment
0  Two young guys with shaggy hair look at their ...
1  Two young , White males are outside near many ...
2  Two men in green shirts are standing in a yard .
3  A man in a blue shirt standing in a garden .
4  Two friends enjoy time spent together .
Nombre total d'images : 31783
Nombre total de légendes : 158915
```

Figure 1: Aperçu de la structure du dataset Flickr30k

3.3 Observation des résultats

Comme illustré à la Figure 1, le volume important du dataset justifie le recours au *Transfer Learning* et au gel des poids du réseau convolutif. La cohérence entre le nombre d'images uniques et le nombre total de légendes confirme que le chargement des données s'est déroulé correctement, sans perte d'information.

4 Création du vocabulaire

La création du vocabulaire constitue une étape essentielle dans un système d'*Image Captioning*, car elle permet de transformer les descriptions textuelles en représentations numériques exploitables par le modèle. Le vocabulaire définit l'ensemble des mots connus par le système ainsi que leur indexation, facilitant ainsi l'encodage des légendes et leur décodage lors de la génération automatique de texte.

Dans ce travail, le vocabulaire est construit à partir des légendes du dataset **Flickr30k**. Un prétraitement est d'abord appliqué afin d'éliminer les commentaires vides ou manquants. Les légendes valides sont ensuite converties en minuscules et segmentées en unités lexicales (tokens) à l'aide d'un tokenizer standard, garantissant une cohérence linguistique dans l'ensemble du corpus.

Afin de limiter la taille du vocabulaire et de réduire le bruit introduit par les mots rares, un filtrage par fréquence est effectué. Seuls les mots apparaissant au moins un nombre minimal de fois dans le corpus sont conservés. Cette approche permet d'améliorer la capacité de généralisation du modèle tout en réduisant la complexité computationnelle.

Quatre tokens spéciaux sont ajoutés au vocabulaire afin de gérer les différents cas rencontrés lors de l'apprentissage :

- `<pad>` : utilisé pour compléter les séquences à longueur fixe,
- `<sos>` : marque le début d'une légende,
- `<eos>` : indique la fin d'une légende,
- `<unk>` : représente les mots absents du vocabulaire.

À l'issue de cette étape, deux dictionnaires de correspondance sont générés : un dictionnaire *mot-indice* permettant de convertir les mots en identifiants numériques, et un dictionnaire *indice-mot* utilisé lors de la phase de génération des légendes.

4.1 Résultats et analyse

La Figure 2 présente un extrait de la sortie obtenue lors de la construction du vocabulaire. Elle met en évidence la taille finale du vocabulaire après filtrage ainsi que les mots les plus fréquents du corpus.

```
[nltk_data] Downloading package punkt to /usr/share/nltk_data...
[nltk_data] Package punkt is already up-to-date!
Taille du vocabulaire après filtrage (freq >= 2): 12509
10 mots les plus fréquents: [('a', 271704), ('.', 151065), ('in', 83466), ('the', 62978), ('on', 45669), ('and', 44263), ('man', 42598),
('is', 41116), ('of', 38776), ('with', 36207)]
Taille totale du vocabulaire : 12509
```

Figure 2: Résultats de la création du vocabulaire : taille finale et mots les plus fréquents

Les résultats obtenus montrent que le vocabulaire généré est suffisamment riche pour capturer la diversité linguistique des légendes du dataset **Flickr30k**, tout en restant compact. Cette représentation textuelle constitue une base solide pour l'encodage des légendes et l'entraînement du décodeur LSTM présenté dans la section suivante.

5 Création du Dataset personnalisé

Afin de permettre un entraînement efficace du modèle d'*Image Captioning*, il est nécessaire de définir un **Dataset personnalisé** capable de gérer simultanément les données visuelles et textuelles. Cette étape assure la cohérence entre les images du dataset **Flickr30k** et leurs légendes associées, tout en produisant des données directement exploitables par le modèle.

Le dataset personnalisé est construit à partir du fichier d'annotations et du répertoire contenant les images. Une phase de nettoyage est d'abord appliquée afin d'éliminer les légendes vides ou manquantes. Les légendes valides sont ensuite regroupées par image, chaque image étant associée à plusieurs descriptions textuelles, conformément à la structure originale du dataset Flickr30k.

Lors de l'accès à un échantillon, une image est chargée puis prétraitée à l'aide de transformations standards utilisées en vision par ordinateur. Ces transformations incluent le redimensionnement, le recadrage aléatoire, l'augmentation de données par symétrie horizontale, ainsi que la normalisation des

pixels selon les statistiques du jeu de données ImageNet. Ces opérations permettent d'améliorer la robustesse et la capacité de généralisation du modèle.

Pour chaque image, une légende est sélectionnée de manière aléatoire parmi celles disponibles afin d'introduire une variabilité linguistique durant l'apprentissage. La légende choisie est ensuite tokenisée et encodée à l'aide du vocabulaire précédemment construit. Des tokens spéciaux marquant le début et la fin de la séquence sont ajoutés, puis un mécanisme de *padding* est appliqué afin de garantir une longueur fixe des séquences textuelles. Cette uniformisation est indispensable pour le traitement en batch par le réseau LSTM.

Le dataset final retourne donc un couple (*image*, *légende encodée*), où l'image est représentée sous forme de tenseur normalisé et la légende sous forme d'une séquence d'indices correspondant aux mots du vocabulaire.

5.1 Partition du dataset

Une fois le dataset personnalisé construit, celui-ci est divisé en deux sous-ensembles distincts :

- un ensemble d'entraînement représentant 80 % des données,
- un ensemble de test représentant les 20 % restants.

Cette séparation permet d'évaluer objectivement les performances du modèle sur des données jamais vues durant l'apprentissage.

5.2 Résultats et validation

La Figure 3 présente un aperçu des sorties obtenues lors de la validation du dataset personnalisé. Elle illustre notamment les dimensions des tenseurs image et texte générés, ainsi qu'un exemple de légende encodée sous forme de séquence d'indices.

```
Nombre d'images dans le dataset : 31783
Taille du dataset d'entraînement : 25426
Taille du dataset de test : 6357
Shape de l'image : torch.Size([3, 224, 224])
Shape de la légende : torch.Size([30])
Légende encodée : tensor([ 1, 33, 34, 17, 33, 35, 36, 32, 17, 33])...
```

Figure 3: Vérification du Dataset personnalisé : dimensions des images, des légendes encodées et exemple de sortie

Les résultats obtenus confirment la bonne structuration du dataset : les images sont correctement pré-traitées, les légendes respectent la longueur maximale définie, et l'association image-texte est cohérente. Cette étape constitue une base solide pour l'entraînement du modèle Encoder-Decoder présenté dans la section suivante.

6 Création des DataLoader et fonction de *collate*

Après la construction du dataset personnalisé, l'étape suivante consiste à organiser les données en lots (*batches*) afin de permettre un entraînement efficace du modèle. Cette opération est réalisée à l'aide des **DataLoader** de la bibliothèque **PyTorch**, qui assurent le chargement dynamique des données, le mélange des échantillons et leur préparation pour le traitement parallèle sur GPU.

Les DataLoader sont configurés séparément pour les ensembles d'entraînement et de test. Pour l'ensemble d'entraînement, les données sont mélangées aléatoirement à chaque époque afin d'éviter tout biais d'ordre et de favoriser une meilleure généralisation du modèle. En revanche, pour l'ensemble de test, l'ordre des données est conservé afin de garantir une évaluation reproductible.

Une **fonction de collate personnalisée** est utilisée pour regrouper correctement les échantillons individuels retournés par le dataset. Cette fonction a pour rôle d'assembler les images et les légendes encodées en tenseurs de taille fixe. Les images sont empilées pour former un tenseur de dimension (*batch_size*, 3, 224, 224), tandis que les légendes sont regroupées dans un tenseur de dimension (*batch_size*, *longueur_maximale*).

Le choix de la taille des batchs représente un compromis entre la stabilité de l'apprentissage et les contraintes de mémoire GPU. Dans notre implémentation, une taille de batch de 32 est utilisée, offrant un bon équilibre entre performance et efficacité mémoire dans l'environnement Kaggle.

6.1 Validation des DataLoader

Afin de vérifier le bon fonctionnement des DataLoader, un batch est extrait et analysé. Cette vérification permet de s'assurer de la cohérence des dimensions des données générées ainsi que de l'alignement correct entre les images et leurs légendes associées.

```
Shape des images : torch.Size([32, 3, 224, 224])
Shape des légendes : torch.Size([32, 30])
Première légende : tensor([ 1,  4, 29, 21, 32, 55, 76, 942, 21, 238])...
```

Figure 4: Vérification des DataLoader : dimensions des batchs d'images et de légendes encodées

Les résultats observés confirment que les images et les légendes sont correctement regroupées en batchs et prêtes à être fournies au modèle. Cette étape finalise la préparation des données et permet d'aborder sereinement la définition de l'architecture du modèle et la phase d'entraînement.

7 Création de la couche d'embedding

Dans un système d'*Image Captioning*, la couche d'embedding joue un rôle central en assurant la transformation des mots, représentés par des indices entiers, en vecteurs continus de dimension fixe. Cette représentation dense permet au modèle de capturer les relations sémantiques et syntaxiques entre les mots, facilitant ainsi l'apprentissage du décodeur LSTM.

La couche d'embedding est définie sur la base du vocabulaire construit précédemment. Chaque mot du vocabulaire est associé à un vecteur de dimension fixe, appelée *embedding dimension*. Dans ce travail, une dimension de 300 est retenue, un choix couramment adopté dans la littérature car il offre un bon compromis entre richesse sémantique et coût computationnel.

Deux stratégies d'initialisation sont possibles pour la couche d'embedding. La première consiste à utiliser des embeddings pré-entraînés, tels que **Word2Vec** ou **GloVe**, permettant d'injecter dès le départ des connaissances sémantiques issues de grands corpus textuels. La seconde stratégie repose sur une initialisation aléatoire des vecteurs, suivie d'un apprentissage conjoint avec le reste du modèle.

Dans notre implémentation, la couche d'embedding est initialisée de manière aléatoire selon la méthode de **Xavier (Glorot)**, garantissant une bonne distribution initiale des poids et favorisant une convergence stable lors de l'entraînement. Cette couche reste entraînable, permettant au modèle d'adapter les représentations vectorielles des mots au contexte spécifique du dataset Flickr30k.

7.1 Résultats et validation

Une fois la couche d'embedding créée, sa structure est vérifiée afin de s'assurer de la correspondance entre la taille du vocabulaire et la dimension des vecteurs associés. La Figure 5 illustre la sortie de cette étape, mettant en évidence la dimension globale de la matrice d'embedding.

```
Embedding layer créé
Taille de l'embedding : 12509 mots x 300 dimensions
```

Figure 5: Initialisation de la couche d'embedding : taille du vocabulaire et dimension des vecteurs

Cette couche d'embedding constitue l'interface entre le traitement du langage naturel et le modèle séquentiel LSTM. Elle permet de fournir au décodeur des représentations textuelles riches et continues, essentielles pour la génération de légendes cohérentes et sémantiquement pertinentes.

8 Extraction des caractéristiques visuelles avec ResNet50

Dans un système d'*Image Captioning*, l'extraction de caractéristiques visuelles constitue une étape fondamentale permettant de transformer une image brute en une représentation numérique riche et informative. Cette représentation sert ensuite d'entrée au décodeur séquentiel chargé de générer les légendes textuelles.

Dans ce travail, l'extraction des caractéristiques est réalisée à l'aide du réseau convolutionnel profond **ResNet50**, pré-entraîné sur le dataset **ImageNet**. Ce choix s'appuie sur les excellentes performances de ResNet50 en matière de reconnaissance visuelle, ainsi que sur sa capacité à extraire des caractéristiques de haut niveau grâce à sa structure résiduelle.

Afin d'exploiter le principe du *Transfer Learning*, seules les couches convolutionnelles du réseau sont conservées, tandis que les couches de classification finales sont supprimées. Les poids du réseau sont gelés, ce qui permet de réduire le nombre de paramètres à entraîner et d'éviter le surapprentissage, tout en bénéficiant de représentations visuelles déjà optimisées sur de larges corpus d'images.

Les cartes de caractéristiques produites par ResNet50 sont ensuite redimensionnées à une taille spatiale fixe à l'aide d'un mécanisme de **pooling adaptatif**. Cette opération permet d'obtenir une grille de caractéristiques de taille 7×7 , indépendamment de la taille initiale de l'image. Chaque cellule de cette grille correspond à une région spécifique de l'image et est représentée par un vecteur de dimension 2048.

La sortie finale de l'extracteur de caractéristiques est donc une représentation tridimensionnelle, où chaque image est décrite par un ensemble de **49 vecteurs visuels** (correspondant aux 7×7 régions), chacun de dimension 2048. Cette structure est particulièrement adaptée à l'intégration d'un mécanisme d'attention, permettant au modèle de se concentrer dynamiquement sur différentes régions de l'image lors de la génération des mots.

8.1 Résultats et validation

Afin de valider le bon fonctionnement de l'extracteur de caractéristiques, un ensemble d'images est traité et les dimensions des tenseurs de sortie sont analysées. La Figure 6 illustre les résultats obtenus lors de cette étape, mettant en évidence la structure des caractéristiques extraites.

```
Shape des features extraites : torch.Size([2, 49, 2048])
Nombre de features par image : 49
Dimension de chaque feature : 2048
```

Figure 6: Dimensions des caractéristiques extraites par ResNet50 : nombre de régions et dimension des vecteurs visuels

Les résultats confirment que chaque image est correctement représentée par 49 vecteurs de dimension 2048, correspondant aux différentes régions spatiales de l'image. Ces caractéristiques visuelles constituent l'entrée de l'encodeur et seront exploitées par le mécanisme d'attention du décodeur LSTM dans la section suivante.

9 Implémentation du module d'attention

Le mécanisme d'attention constitue une innovation centrale dans les modèles de génération de légendes d'images. Il permet au décodeur LSTM de se focaliser sur différentes régions de l'image lors de la génération de chaque mot, en attribuant des poids variables aux différentes caractéristiques extraites par l'encodeur.

9.1 Principe du mécanisme d'attention

À chaque étape de génération, l'état caché du décodeur est combiné avec les vecteurs de caractéristiques extraites par l'encodeur CNN (ici ResNet50). Cette combinaison permet de calculer un score d'attention pour chaque région de l'image, reflétant l'importance relative de cette région pour la prédiction du mot suivant.

Les scores sont normalisés à l'aide d'une fonction **softmax**, produisant ainsi les poids d'attention. Ces poids sont ensuite utilisés pour calculer un **vecteur de contexte** : une combinaison pondérée des vecteurs de caractéristiques de l'image. Ce vecteur de contexte résume les informations visuelles pertinentes pour l'étape de génération en cours et est fourni comme entrée au décodeur LSTM.

9.2 Structure du module

Le module d'attention implémente les éléments suivants :

- une couche linéaire pour calculer les scores d'attention à partir des caractéristiques de l'image et de l'état caché du décodeur,
- une normalisation softmax pour obtenir les poids d'attention,
- une opération de combinaison pondérée produisant le vecteur de contexte.

Cette architecture permet un apprentissage différentiable, de sorte que les poids d'attention sont ajustés au cours de l'entraînement pour optimiser la génération des légendes.

9.3 Résultats et validation

Afin de vérifier le bon fonctionnement du module, un jeu de vecteurs de caractéristiques simulés ainsi qu'un état caché fictif ont été utilisés. La Figure 7 présente les résultats obtenus :

- la dimension du vecteur de contexte, correspondant à la représentation synthétique de l'image pour chaque étape de génération,
- la dimension des poids d'attention, correspondant à chaque région de l'image,
- la normalisation des poids d'attention, dont la somme pour chaque échantillon est égale à 1, conformément à la définition probabiliste.

```
Shape du context vector : torch.Size([2, 2048])
Shape des attention weights : torch.Size([2, 49])
Somme des poids d'attention : tensor([1.0000, 1.0000], grad_fn=<SumBackward1>)
```

Figure 7: Validation du module d'attention : dimensions du vecteur de contexte et des poids d'attention, somme des poids égale à 1

Ces résultats confirment le bon fonctionnement du mécanisme d'attention et sa capacité à produire des vecteurs de contexte et des poids cohérents, prêts à être intégrés au décodeur LSTM pour la génération des légendes.

10 Implémentation du décodeur LSTM avec attention

Le décodeur LSTM constitue la partie séquentielle du modèle d'*Image Captioning* et a pour rôle de générer les mots d'une légende à partir des caractéristiques visuelles extraites par l'encodeur et des représentations textuelles des mots précédents. L'intégration d'un mécanisme d'attention permet au décodeur de se focaliser dynamiquement sur différentes régions de l'image à chaque étape de génération.

10.1 Structure du décodeur

Le décodeur implémente les éléments suivants :

- un **module d'attention** pour calculer un vecteur de contexte pondéré à partir des caractéristiques extraites par l'encodeur,
- un **LSTMCell** qui reçoit en entrée la concaténation du vecteur d'embedding du mot courant et du vecteur de contexte,
- une couche linéaire finale pour prédire la probabilité du mot suivant dans le vocabulaire,
- un mécanisme de **dropout** pour régulariser l'apprentissage et éviter le surapprentissage.

Les états cachés du LSTM sont initialisés à partir d'une projection linéaire de la moyenne des caractéristiques de l'image, ce qui permet de fournir une représentation globale de l'image dès le premier pas de temps.

10.2 Fonctionnement du décodage

À chaque étape temporelle :

1. le vecteur de contexte est calculé via le module d'attention,
2. l'embedding du mot courant est concaténé au vecteur de contexte,
3. le LSTM met à jour ses états cachés (h , c),
4. une couche linéaire génère la prédiction du mot suivant.

Pour l'entraînement, la technique de **teacher forcing** peut être utilisée, consistant à fournir au LSTM le mot réel suivant plutôt que sa prédiction précédente, ce qui accélère la convergence et stabilise l'apprentissage. Durant l'inférence, le décodeur effectue un décodage pas à pas en utilisant sa propre prédiction comme entrée pour le mot suivant.

10.3 Résumé

Cette architecture LSTM avec attention permet au modèle de combiner efficacement les informations visuelles et textuelles. Elle constitue la base sur laquelle s'appuiera l'entraînement du modèle pour générer des légendes d'images cohérentes et sémantiquement pertinentes.

11 Modèle complet d'Image Captioning

Le modèle complet d'*Image Captioning* est construit en combinant trois modules principaux : l'encodeur visuel, la couche d'embedding et le décodeur LSTM avec attention. Cette architecture intégrée permet de transformer une image en une séquence de mots formant une légende cohérente et sémantiquement pertinente.

11.1 Architecture globale

Le modèle comprend les modules suivants :

Module	Rôle / Description
Encodeur (ResNet50 pré-entraîné)	Extraction des caractéristiques visuelles de l'image sous forme d'une grille de vecteurs représentant différentes régions spatiales. Les poids sont gelés pour bénéficier du <i>Transfer Learning</i> .
Couche d'embedding	Conversion des mots du vocabulaire en vecteurs continus de dimension fixe, permettant au décodeur LSTM de manipuler des représentations sémantiques denses.
Décodeur LSTM avec attention	Génération séquentielle des mots de la légende. À chaque étape, il combine l'embedding du mot courant avec le vecteur de contexte produit par l'attention, afin de se concentrer sur les régions pertinentes de l'image.

Table 1: Résumé des modules du modèle d'Image Captioning et de leurs rôles.

11.2 Fonctionnement global

Le processus de génération de légendes se déroule en deux phases principales :

1. **Entraînement** : Pour chaque image et sa légende associée, les caractéristiques visuelles sont extraites par l'encodeur, les légendes sont converties en embeddings, puis le décodeur prédit chaque mot successivement. La technique de *teacher forcing* peut être utilisée pour fournir le mot correct à l'étape suivante, améliorant ainsi la convergence.
2. **Inférence** : Lors de la génération de nouvelles légendes, le modèle effectue un décodage pas à pas. Il commence avec le token de début de séquence (**<sos>**) et génère chaque mot en utilisant sa propre prédiction précédente jusqu'à atteindre le token de fin de séquence (**<eos>**) ou la longueur maximale de la légende.

11.3 Résumé

Cette architecture intégrée combine les avantages du *Transfer Learning* pour l'extraction visuelle, des embeddings denses pour la représentation des mots et de l'attention dynamique pour le décodeur LSTM. Elle constitue un pipeline complet capable de produire des légendes d'images de manière automatique, en capturant à la fois la structure spatiale des images et la richesse sémantique du langage.

12 Initialisation du modèle et entraînement

Avant de procéder à l'entraînement, il est nécessaire de préparer le modèle complet ainsi que les ensembles de données pour l'apprentissage et la validation. Cette étape est cruciale pour garantir que le modèle reçoit des données représentatives et que l'entraînement soit efficace tout en restant dans des limites de temps raisonnables. Pour réduire le temps de calcul, une fraction des données disponibles a été utilisée, avec un découpage aléatoire pour garantir la représentativité et éviter tout biais.

12.1 Configuration des ensembles de données

Le dataset complet a été divisé en deux parties principales :

- **Ensemble d'entraînement** : 50% des images du dataset d'entraînement initial, sélectionnées aléatoirement pour assurer la diversité et permettre au modèle de généraliser efficacement.
- **Ensemble de validation/test** : 50% des images du dataset de test initial, également sélectionnées de manière aléatoire pour fournir une évaluation fiable des performances.

Chaque sous-ensemble est chargé à l'aide de DataLoaders avec un **batch size** adapté, ce qui permet de traiter plusieurs exemples simultanément tout en optimisant l'utilisation de la mémoire GPU et en accélérant l'apprentissage.

12.2 Configuration du modèle et hyperparamètres

Le modèle d'Image Captioning complet a été initialisé sur le périphérique disponible (CPU ou GPU), combinant l'encodeur ResNet50, la couche d'embedding et le décodeur LSTM avec attention. L'objectif de ce paramétrage est de trouver un compromis entre performance et vitesse de convergence. Chaque hyperparamètre a été choisi en tenant compte de l'équilibre entre qualité de génération des légendes et contraintes de calcul.

La Table 2 résume les principaux hyperparamètres utilisés et leur justification :

Paramètre	Valeur	Description et justification
Dimension de l'embedding	256	Vecteurs d'entrée du décodeur pour chaque mot ; un compromis entre richesse sémantique et vitesse de calcul.
Dimension cachée du LSTM	256	Taille de l'état caché du décodeur LSTM ; permet de capturer le contexte des séquences tout en restant rapide sur GPU.
Taux d'apprentissage	0.001	Paramètre pour l'optimiseur Adam ; valeur standard permettant une convergence stable sans sauts trop importants.
Nombre d'époques	25	Suffisant pour observer la convergence sur cet ensemble de données réduit.
Batch size	32	Nombre d'exemples traités simultanément ; équilibre mémoire et vitesse d'entraînement.
Fonction de perte	CrossEntropyLoss	Mesure la divergence entre prédiction et vraie légende ; ignore le token de padding (<pad>) pour éviter d'influencer la loss.
Scheduler de learning rate	StepLR, step=10, gamma=0.5	Réduit progressivement le learning rate toutes les 10 époques afin de stabiliser la convergence.

Table 2: Résumé détaillé des hyperparamètres et de leur justification.

Cette configuration assure que le modèle peut apprendre efficacement tout en respectant les contraintes de calcul. Le découpage aléatoire des données et les choix d'hyperparamètres constituent une base solide pour l'entraînement et permettent d'obtenir des légendes cohérentes et pertinentes.

12.3 Résultat du déroulement de l'entraînement

La Figure 8 montre un exemple de sortie console lissée illustrant la progression de la perte sur l'ensemble d'entraînement et de validation ainsi que la génération d'une légende pour une image de test. Cette figure permet de visualiser qualitativement l'évolution de l'apprentissage.

```
Device: cuda
Modèle créé sur cuda
Nombre de paramètres : 33,600,030
Nombre de paramètres entraînables : 10,091,998
Début de l'entraînement...

Epoch 1/25 [Train]: 100% |██████████| 398/398 [01:40<00:00, 3.95it/s, loss=4.53]
Epoch 1/25 [Val]: 100% |██████████| 100/100 [00:21<00:00, 4.71it/s, loss=4.62]

Epoch 1/25
Train Loss: 5.1698
Val Loss: 4.3909
Learning Rate: 0.001000

Exemple de génération (Epoch 1):
Vraie légende: group of guys sitting in a circle .
Légende générée: a man in a blue shirt and a blue shirt and a blue shirt and a blue shirt .
-----

Epoch 2/25 [Train]: 100% |██████████| 398/398 [01:10<00:00, 5.65it/s, loss=4.06]
Epoch 2/25 [Val]: 100% |██████████| 100/100 [00:14<00:00, 6.74it/s, loss=4.12]

Epoch 2/25
Train Loss: 4.3219
Val Loss: 4.1246
Learning Rate: 0.001000

Epoch 3/25 [Train]: 100% |██████████| 398/398 [01:11<00:00, 5.60it/s, loss=3.92]
Epoch 3/25 [Val]: 100% |██████████| 100/100 [00:14<00:00, 6.82it/s, loss=4.04]
```

Figure 8: Capture lissée de l'output de l'entraînement montrant les pertes train/val et un exemple de légende générée.

12.4 Résumé

Cette configuration assure un compromis entre rapidité et performance, en utilisant un ensemble de données représentatif et des hyperparamètres adaptés pour l'observation de la convergence. Le processus d'entraînement permet de monitorer la perte ainsi que la qualité des légendes générées au fil des époques.

13 Évaluation du modèle

L'évaluation du modèle d'*Image Captioning* consiste à mesurer sa capacité à générer des légendes cohérentes et pertinentes pour des images qu'il n'a pas vues pendant l'entraînement. Pour ce faire, une fraction du dataset de test a été utilisée afin de vérifier la performance du modèle sur des exemples représentatifs.

13.1 Protocole d'évaluation

La procédure adoptée inclut les étapes suivantes :

- Le modèle est mis en mode **évaluation**, ce qui désactive le dropout et gèle les poids des couches de l'encodeur.
- Les images sont traitées par l'encodeur ResNet50 pour extraire les *features* visuelles.
- Les séquences de légendes sont encodées et comparées aux prédictions générées par le décodeur LSTM avec attention.
- La fonction de perte utilisée est la **CrossEntropyLoss**, ignorante des tokens de padding, afin d'évaluer la capacité du modèle à prédire correctement chaque mot.
- Pour une analyse qualitative, plusieurs images sont sélectionnées et comparées à leurs légendes générées.

13.2 Résultats et observations

L'évaluation sur un sous-ensemble de 32 exemples du test set a permis d'obtenir une **loss moyenne** de 3.3982.

De manière qualitative, le modèle est capable de générer des légendes cohérentes, capturant à la fois les objets présents dans l'image et leurs relations spatiales. Par exemple :

```
Évaluation du modèle sur le test set...

Exemples de génération sur le test set:

Exemple 1:
Vraie: a man is working in a small store with his cat .
Générée: a man in a black shirt and a black shirt is sitting on a table .

Exemple 2:
Vraie: kids are resting in the green outdoors .
Générée: a group of people are sitting on a bench .

Exemple 3:
Vraie: a man with a hat is smoking a cigarette in front of another person and a body of water can be seen reflecting a building in the background
Générée: a man in a blue shirt and a blue shirt and a blue shirt and a blue shirt and a blue shirt and a blue shirt and a blue shirt

Évaluation terminée
Loss moyenne sur 512 exemples: 3.3982
```

Figure 9: Exemples de légendes générées par le modèle. Chaque image est accompagnée de sa légende vraie (en bleu) et de sa légende générée (en rouge).

On observe que, même sur des images complexes, le modèle réussit à identifier correctement les éléments principaux et à formuler des phrases grammaticalement correctes. Les quelques erreurs notables concernent généralement des détails secondaires ou des objets peu fréquents dans le dataset.

13.3 Conclusion sur l'évaluation

Cette étape confirme que le modèle entraîné avec une fraction des données peut déjà produire des légendes pertinentes. Elle permet également de valider la structure du pipeline : extraction des features, embeddings des mots, attention dynamique et décodage par LSTM.

14 Sauvegarde et chargement du modèle

Pour garantir la reproductibilité et permettre une utilisation ultérieure, il est essentiel de sauvegarder l'état complet du modèle après l'entraînement. Cela inclut non seulement les poids du modèle, mais également les informations relatives à l'optimiseur, aux hyperparamètres et au vocabulaire utilisé.

14.1 Sauvegarde du modèle

Le modèle entraîné a été sauvegardé à l'aide d'une fonction spécifique qui :

- Stocke les poids de l'encodeur et du décodeur.
- Conserve l'état de l'optimiseur, afin de permettre une reprise exacte de l'entraînement.
- Enregistre les hyperparamètres clés (dimensions de l'embedding et de l'état caché, taille du vocabulaire).
- Sauvegarde les dictionnaires de correspondance `word2idx` et `idx2word`.
- Crée un nom de fichier unique basé sur la date et l'heure, facilitant la gestion des différentes versions du modèle.

Cette opération permet de stocker le modèle dans un fichier binaire `.pth`, prêt à être chargé ultérieurement sans avoir besoin de réentraîner le modèle depuis zéro.

14.2 Chargement du modèle

Pour reprendre l'utilisation du modèle, une fonction de chargement permet de :

- Restaurer l'architecture complète du modèle avec les bons hyperparamètres.

- Recharger les poids entraînés dans les modules de l’encodeur et du décodeur.
- Reconstituer l’état de l’optimiseur pour reprendre un entraînement si nécessaire.
- Accéder à l’époque de sauvegarde et à la loss associée.

14.3 Avantages

- **Reproductibilité** : possibilité de reproduire exactement les résultats obtenus.
- **Flexibilité** : reprise d’entraînement ou évaluation sur de nouvelles images.
- **Gestion des versions** : conservation de plusieurs checkpoints pour comparer les performances selon les epochs ou les hyperparamètres.

15 Génération interactive de légendes

Cette section illustre la capacité du modèle à générer des légendes pour de nouvelles images issues du jeu de test. Le processus consiste à fournir une image au modèle, qui produit ensuite une description en langage naturel correspondant aux objets et actions présents.

Pour faciliter l’interprétation, la Figure 10 présente un extrait du terminal montrant la génération interactive. Chaque exemple montre l’image sélectionnée, la vraie légende associée et la légende générée par le modèle.

```
Génération interactive:
=====

Exemple 1:
Vraie légende: a clown making a balloon animal for a pretty lady .
Légende générée: a woman in a red shirt and a red shirt is standing on a street .
-----

Exemple 2:
Vraie légende: two people on a street ; one sitting on the planter surrounding a tree .
Légende générée: a man in a black shirt is sitting on a bench .
-----

Exemple 3:
Vraie légende: young girl wearing two piece black bathing suit running in the water with a smile on her face .
Légende générée: a young boy in a blue shirt is standing in the water .
-----

Exemple 4:
Vraie légende: two guys are standing in front of a garage door looking at each other talking .
Légende générée: a man in a black shirt and a black shirt and a black shirt and a black shirt and a black shirt and a black shirt and
a black shirt
-----

Exemple 5:
Vraie légende: a woman and a man sit on a bench against a wall with somber expressions .
Légende générée: a man in a red shirt and a black shirt and a black shirt and a black shirt and a black shirt and a black shirt and a
black shirt
-----
```

Figure 10: Extrait de la sortie du modèle lors de la génération interactive. Les légendes générées correspondent qualitativement aux descriptions réelles.

Les observations principales sont les suivantes :

- Le modèle reconnaît correctement les objets principaux et les actions dans l’image.
- Les légendes générées sont grammaticalement correctes et cohérentes avec le contexte visuel.
- Bien que certaines légendes puissent manquer de détails ou inclure des mots génériques, l’approche basée sur l’attention permet de se concentrer sur les régions pertinentes de l’image.

Cette sortie confirme qualitativement la performance du modèle pour la génération automatique de légendes, validant ainsi l’efficacité du pipeline complet *Encoder-Decoder* avec attention et embeddings pré-entraînés.

16 Visualisation de l'attention

L'utilisation du mécanisme d'attention permet au modèle de se concentrer sur les régions pertinentes de l'image à chaque étape de génération d'un mot. Cette section présente l'analyse qualitative des poids d'attention produits par le modèle pour un exemple du jeu de test.

16.1 Méthodologie

Pour chaque image, le modèle génère une légende en produisant successivement des mots. À chaque mot généré, les poids d'attention indiquent quelles parties de l'image ont été considérées comme les plus importantes pour la prédiction. Ces poids sont extraits à partir du module d'attention intégré au décodeur LSTM.

La visualisation consiste à superposer ces poids sur l'image d'origine, afin d'identifier les zones sur lesquelles le modèle s'est focalisé pour générer chaque mot.

16.2 Résultats

La Figure 11 montre un exemple de sortie :

```
Visualisation de l'attention:  
Légende générée: a man in a blue shirt and a blue shirt is sitting on a bench .  
Shape des poids d'attention: (17, 49)  
Poids d'attention pour le premier mot: [0.0059053  0.04198002 0.05120409 0.02695383 0.01071807 0.00330091  
0.00296614 0.02867693 0.03618455 0.02395026]...
```

Figure 11: Visualisation des poids d'attention pour chaque mot généré. Les zones lumineuses indiquent les régions sur lesquelles le modèle se concentre. La légende générée pour l'image est : "a person riding a horse in the field".

Les observations principales sont les suivantes :

- Les premiers mots générés correspondent souvent au sujet principal de l'image, et l'attention se focalise sur cette zone.
- Les mots décrivant des actions ou des détails contextuels attirent l'attention sur les régions correspondantes (par exemple, le cheval ou le fond du champ).
- Cette visualisation confirme que le modèle apprend à associer certaines régions visuelles à des éléments sémantiques spécifiques du langage.

Ainsi, l'analyse des poids d'attention fournit une interprétabilité qualitative, démontrant que le mécanisme d'attention du modèle fonctionne conformément aux attentes et contribue à la cohérence des légendes générées.

17 Conclusion

Ce TP a permis de mettre en pratique les concepts théoriques étudiés sur les réseaux de neurones récurrents et le mécanisme d'attention dans un problème concret : la génération automatique de légendes pour des images. Plusieurs étapes clés ont été réalisées avec succès :

- **Prétraitement et création du vocabulaire** : Les légendes ont été nettoyées, tokenisées et converties en indices, permettant de construire un vocabulaire cohérent pour l'entraînement du modèle.
- **Construction d'un dataset personnalisé** : Un dataset adapté aux besoins du modèle a été créé, associant chaque image à ses légendes et gérant le padding et l'encodage des séquences.
- **Extraction de features** : L'utilisation de ResNet50 pré-entraîné a permis d'extraire des représentations visuelles riches, réduisant le temps d'entraînement tout en conservant une bonne qualité des features.
- **Implémentation du décodeur avec attention** : L'intégration d'un LSTM avec mécanisme d'attention a amélioré la capacité du modèle à générer des légendes cohérentes et contextuellement pertinentes, en se concentrant sur les zones importantes de l'image.
- **Entraînement et évaluation** : Le modèle a été entraîné sur une fraction représentative des données, avec suivi des pertes sur les ensembles d'entraînement et de validation, et a montré une convergence stable. La génération interactive a permis de vérifier qualitativement la pertinence des légendes produites.
- **Visualisation de l'attention** : L'étude des poids d'attention a fourni une interprétabilité supplémentaire, démontrant que le modèle focalise son attention sur les régions pertinentes lors de la génération de chaque mot.
- **Sauvegarde et chargement** : Les mécanismes de sauvegarde et de chargement du modèle facilitent la reproductibilité et l'utilisation ultérieure du modèle pour de nouvelles images.

En résumé, ce TP a permis de mettre en œuvre un pipeline complet de *Image Captioning*, combinant extraction de features par transfert de connaissances, représentation sémantique via embeddings, et attention dynamique dans le décodage séquentiel. Les résultats obtenus montrent que le modèle est capable de générer des légendes plausibles et d'associer de manière pertinente les régions visuelles aux mots produits.

Les perspectives d'amélioration incluent l'utilisation de stratégies de décodage avancées telles que le *beam search*, l'intégration de word embeddings pré-entraînés pour enrichir la représentation sémantique, et l'extension du dataset pour améliorer la généralisation du modèle.