Université Chouaib Doukkali
École Nationale des Sciences Appliquées_El Jadida
Master Science Des Données Et Intelligence Artificielle
Module: Gen IA

Rapport de travaux demandes:

Rapport de TP : Image Captioning avec RNN et Attention sur ResNet

Réaliser par : Nada saber

Encadrer par : Youness Abouqora

Introduction

Le Image Captioning est une tâche avancée en Intelligence Artificielle qui consiste à générer automatiquement une description textuelle pertinente à partir d'une image. Cette problématique se situe à l'intersection de deux domaines majeurs du Deep Learning : la vision par ordinateur (Computer Vision) et le traitement automatique du langage naturel (Natural Language Processing).

L'objectif principal de ce TP est de concevoir, entraîner et évaluer un modèle de Image Captioning capable de comprendre le contenu visuel d'une image et de produire une phrase descriptive en langage naturel.

Les objectifs pédagogiques de ce travail pratique sont les suivants :

- Comprendre l'architecture générale d'un système de Image Captioning
- Manipuler un dataset réel d'images annotées (Flickr30k)
- Implémenter un modèle Encoder–Decoder en PyTorch
- Combiner un réseau de neurones convolutif (CNN) et un réseau récurrent (LSTM)
- Entraîner un modèle Deep Learning sur GPU(Kaggle)
- Évaluer qualitativement les performances du modèle

1.Configuration et Environnement

1. Importation des Bibliothèques

Le code importe les bibliothèques essentielles demandées :

- PyTorch (torch, nn, optim) : Pour construire l'architecture LSTM personnalisée, le module d'attention et l'optimiseur Adam.
- Torchvision : Pour charger le modèle ResNet50 pré-entraîné et appliquer les transformations aux images (redimensionnement à 224x224).
- Pandas & PIL : Pour manipuler le fichier d'annotations results.csv du dataset Flickr30k et charger les images.
- SummaryWriter : Pour enregistrer la perte (loss) et les résultats dans TensorBoard.
- Struct : Pour lire les vecteurs binaires des embeddings Word2Vec
- Elle vérifie si un GPU (CUDA) est disponible,Elle affiche le nom du GPU utilisé (par exemple, "Tesla T4" sur Kaggle).
- Elle définit la variable device qui sera utilisée pour envoyer votre modèle ResNet et votre LSTM sur la carte graphique afin d'accélérer l'entraînement.

```
# Install dependencies (if needed)
# !pip install torch torchvision tensorboard pandas matplotlib pillow -q

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, random_split, Dataset
from torch.utils.tensorboard import SummaryWriter
import torchvision
from torchvision import transforms
import numpy as np
import struct
from PIL import Image
import matplotlib.pyplot as plt
import pandas as pd
import os
import random
from collections import Counter

# Set random seeds for reproducibility
SEED = 42
random.seed(SEED)
np.random.seed(SEED)
torch.manual_seed(SEED)
if torch.cuda.is_available():
    torch.cuda.manual_seed_all(SEED)

# Device configuration
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
if torch.cuda.is_available():
    print(f"GPU: {torch.cuda.get_device_name(0)}")
```

Hyperparamètres

```python
# Configuration
class Config:
    # Paths
    IMAGES_PATH = '/kaggle/input/flickr-image-dataset/flickr30k_images/flickr30k_images'
    CAPTIONS_FILE = '/kaggle/input/flickr-image-dataset/flickr30k_images/results.csv'
    WORD2VEC_PATH = '/kaggle/input/googlenewsvectorsnegative300/GoogleNews-vectors-negative300.bin'

    # Model hyperparameters
    EMBED_DIM = 300
    LSTM_HIDDEN_DIM = 512
    ATTENTION_DIM = 512
    DROPOUT = 0.5

    # Training hyperparameters
    BATCH_SIZE = 64  # Increased from 32 for better GPU utilization
    NUM_EPOCHS = 50
    LEARNING_RATE = 0.001
    WEIGHT_DECAY = 1e-5
    GRAD_CLIP = 5.0

    # Learning rate scheduler
    LR_STEP_SIZE = 10
    LR_GAMMA = 0.5

    # Early stopping
    PATIENCE = 5

    # Data
    MAX_CAPTION_LENGTH = 20
    TRAIN_SPLIT = 0.8
    NUM_WORKERS = 2

    # Special tokens
    PAD_TOKEN = '<PAD>'
    START_TOKEN = '<START>'
    END_TOKEN = '<END>'
    UNK_TOKEN = '<UNK>'

config = Config()
print("Configuration loaded successfully")
```

- Architecture du modèle : **EMBED_DIM = 300** (correspondant à la dimension de Word2Vec).
- **LSTM_HIDDEN_DIM = 512** pour la mémoire du réseau récurrent.
- Stratégie d'entraînement : Définit le **Learning Rate à 0.001**.
- Configure le Scheduler (StepLR) pour réduire le taux d'apprentissage tous les 10 époques **(LR_STEP_SIZE = 10)** avec un facteur de 0.5 (LR_GAMMA).
- Tokens spéciaux : Définit les balises indispensables pour le texte comme <START>, <END>, et <PAD> pour uniformiser la longueur des séquences à 20 mots.

2. Data Loading and Preprocessing

```python
# Load captions
print("Loading captions...")
captions_df = pd.read_csv(config.CAPTIONS_FILE, delimiter='|')
captions_df.columns = captions_df.columns.str.strip()
captions_df = captions_df.dropna()
captions_df['image_name'] = captions_df['image_name'].str.strip()
captions_df['comment'] = captions_df['comment'].str.strip()

print(f"Captions loaded: {captions_df.shape}")
print(f"Columns: {captions_df.columns.tolist()}")
print(f"\nSample captions:")
print(captions_df.head())
```

```
Loading captions...
Captions loaded: (158914, 3)
Columns: ['image_name', 'comment_number', 'comment']

Sample captions:
      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 .
```

- **Nettoyage (Cleaning) :** On utilise .strip() pour enlever les espaces inutiles dans les noms de colonnes et les textes, et .dropna() pour supprimer les lignes vides.
- On voit qu'il y a plus de 158 000 légendes (environ 5 par image pour 31 000 images), ce qui correspond aux données de Flickr30k.

```python
vocabulary = set()
vocabulary.add(config.PAD_TOKEN)
vocabulary.add(config.START_TOKEN)
vocabulary.add(config.END_TOKEN)
vocabulary.add(config.UNK_TOKEN)

for caption in captions_df['comment'].values:
    if isinstance(caption, str):
        for word in caption.lower().split():
            word = word.strip('.,!?;:"()[]')
            if word:
                vocabulary.add(word)

print(f"Vocabulair size: {len(vocabulary)}")
```

Cet partie construit le vocabulaire unique du projet en extrayant tous les mots des légendes du dataset Flickr30k .

Il effectue trois actions principales :

1. Initialisation : Il ajoute des tokens spéciaux indispensables (PAD, START, END, UNK) pour gérer la structure des séquences textuelles.
2. Nettoyage : Il parcourt chaque légende, convertit le texte en minuscules et retire la ponctuation pour ne garder que les mots propres.
3. Indexation : Il stocke chaque mot unique dans un ensemble (set) pour définir la taille finale du dictionnaire que le modèle devra apprendre.

3. Word2Vec Embeddings

Cet partie permet de transformer vos mots textuels en vecteurs mathématiques en utilisant des embeddings pré-entraînés (Word2Vec)

- **Lecture Binaire (Word2Vec) :** Les fonctions read_until et read_word2vec parcourent le fichier binaire de Google News pour extraire chaque mot et son vecteur de 300 dimensions associé .
- **Création de la Couche d'Embedding :** La fonction create_embedding_layer construit une matrice de poids (weights) où chaque ligne correspond à un vecteur de mot. Si un mot de votre vocabulaire existe dans Word2Vec, son vecteur est chargé ; sinon, il est initialisé aléatoirement
- **Gel des Poids (Freezing)** : Si vous utilisez Word2Vec (randomized=False), la couche est gelée (requires_grad = False), ce qui signifie que ces vecteurs ne changeront pas pendant l'entraînement pour conserver leur connaissance sémantique pré-entraînée.

```
print("Loading embeddings...")
RANDOMIZED = False
pretrained_embeddings, idx2word, word2idx = create_embedding_layer(
    vocabulary, config.WORD2VEC_PATH, RANDOMIZED
)
vocab_size = len(idx2word)
print(f"\nVocabulary size: {vocab_size}")
print(f"Special tokens: PAD={word2idx[config.PAD_TOKEN]}, START={word2idx[config.START_TOKEN]}, "
      f"END={word2idx[config.END_TOKEN]}, UNK={word2idx[config.UNK_TOKEN]}")
```

```
Loading embeddings...
Loading Word2Vec: 3000000 words, 300 dimensions
Loaded 10000 word vectors...
Loaded 16833 words from Word2Vec

Vocabulary size: 20273
Special tokens: PAD=0, START=1, END=2, UNK=3
```

- **Initialisation des vecteurs** : Il appelle la fonction précédemment définie pour charger soit les vecteurs Word2Vec réels (si RANDOMIZED = False), soit des vecteurs aléatoires.
- **Mise en place de la couche d'Embedding** : Il récupère trois éléments essentiels :

pretrained_embeddings : La couche PyTorch contenant les poids (les vecteurs de 300 dimensions).
idx2word : Une liste pour traduire un index en mot (utile pour la génération).
word2idx : Un dictionnaire pour traduire un mot en index (utile pour l'entraînement)

- **Vérification des paramètres :** Il calcule la taille totale du vocabulaire (vocab_size) et affiche les index des "tokens spéciaux" pour vérifier que le START, END, PAD et UNK sont correctement positionnés dans le dictionnaire

4. Dataset and DataLoader

Conversion Texte vers Numérique (tokenize) : Cette fonction transforme une phrase en une liste d'identifiants numériques à l'aide d'un dictionnaire . Elle ajoute automatiquement les balises de début et de fin, gère les mots inconnus et normalise la longueur de la séquence (en coupant ou en ajoutant du remplissage) pour qu'elle corresponde à max_length .

Conversion Numérique vers Texte (untokenize) : Cette fonction réalise l'opération inverse en traduisant les identifiants numériques en mots lisibles . Elle s'arrête dès qu'elle rencontre la balise de fin (END_TOKEN) et ignore les jetons techniques comme le remplissage

Organisation des Données (Flickr30kDataset) :

- Il crée un dictionnaire qui regroupe toutes les légendes textuelles associées à une seule image.
- La méthode __getitem__ charge l'image, applique les transformations (comme le redimensionnement) et récupère ses descriptions .

Sélection Aléatoire (collate_fn) :

- Puisque chaque image a plusieurs légendes, cette fonction en choisit une seule au hasard pour chaque itération d'entraînement, ce qui améliore la robustesse du modèle.

Préparation du Batch :

- Il transforme la légende choisie en nombres (tokenisation).
- Il assemble les images et les textes tokenisés en tenseurs (objets mathématiques) prêts à être envoyés au GPU pour l'entraînement.

Cette étape définit les transformations d'images nécessaires pour préparer les données avant qu'elles ne soient traitées par le modèle ResNet50

```
train_transform = transforms.Compose([
    transforms.Resize((256, 256)),
    transforms.RandomCrop(224),
    transforms.RandomHorizontalFlip(),
    transforms.ColorJitter(brightness=0.1, contrast=0.1, saturation=0.1),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

test_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

train_transform (Pour l'entraînement) :

- Il utilise l'augmentation de données pour créer de la diversité : redimensionnement à 256x256 suivi d'un recadrage aléatoire à 224x224 (RandomCrop).
- Il applique des modifications visuelles comme le retournement horizontal aléatoire et des variations de couleur (ColorJitter) pour rendre le modèle plus robuste.

test_transform (Pour l'évaluation) :

- Il se contente d'un redimensionnement simple et direct à (224, 224) sans modification aléatoire pour garantir une évaluation stable.

Normalisation (Commune aux deux) :

- Il convertit les images en tenseurs (le format mathématique de PyTorch).
- Il applique une normalisation standard (Moyenne et Écart-type d'ImageNet) pour que les couleurs des images correspondent exactement à ce que le modèle ResNet50 pré-entraîné attend en entrée

découpage du dataset:

- **Répartition (Split) :** Il divise le dataset selon un ratio défini (par exemple 80% pour l'entraînement). L'utilisation d'un manual_seed garantit que ce découpage est reproductible à chaque exécution.

Application des Transformations :

- Le train_dataset reçoit les transformations avec augmentation de données (flips, rotations) pour améliorer l'apprentissage.
- Le test_dataset reçoit les transformations simples (redimensionnement à 224x224) pour une évaluation stable.

Création des DataLoaders

```
train_loader = DataLoader(
    train_dataset,
    batch_size=config.BATCH_SIZE,
    shuffle=True,
    collate_fn=collate_fn,
    num_workers=config.NUM_WORKERS,
    pin_memory=True
)

test_loader = DataLoader(
    test_dataset,
    batch_size=config.BATCH_SIZE,
    shuffle=False,
    collate_fn=collate_fn,
    num_workers=config.NUM_WORKERS,
    pin_memory=True
)

print(f"Train batches: {len(train_loader)}")
print(f"Test batches: {len(test_loader)}")
```

```
Train batches: 398
Test batches: 100
```

- **Gestion des Batchs** : Il regroupe les images et les textes par lots (64 par défaut) pour optimiser les calculs du GPU.

- **Mélange et Organisation** : Il mélange les données d'entraînement (shuffle=True) pour que le modèle n'apprenne pas l'ordre des photos, mais garde les données de test fixes pour une évaluation stable.

- **Optimisation Matérielle :** Il utilise num_workers pour charger les données en parallèle sur plusieurs cœurs CPU et pin_memory=True pour accélérer le transfert des données vers la mémoire de la carte graphique

5. Model Architecture

a.Chargement du modèle ResNet50

- **Extraction de Caractéristiques (Feature Extraction)** :on charge un modèle ResNet50 déjà entraîné sur des millions d'images (ImageNet). En supprimant les deux dernières couches , on transforme le classifieur en un extracteur de caractéristiques spatiales qui produit un tenseur de dimension 2048*7*7 en sortie de la dernière couche convolutionnelle.
- **Gel des Paramètres (Freezing)** : tous les poids du ResNet sont gelés (requires_grad = False) Cela signifie que durant l'entraînement, seul votre décodeur (Attention + LSTM) apprendra, tandis que le ResNet restera fixe pour fournir des caractéristiques visuelles stables5.
- **Préparation pour l'Attention** : La sortie de forme (2048, 7, 7) est idéale car elle conserve la structure spatiale de l'image (une grille de 7*7, ce qui permettra au futur module d'attention de "regarder" des zones spécifiques de l'image pour chaque mot généré.

```python
print("Loading ResNet50...")
resnet = torchvision.models.resnet50(weights='IMAGENET1K_V1')

# Remove last two layers (avgpool and fc)
modules = list(resnet.children())[:-2]
resnet_modified = nn.Sequential(*modules)

# Freeze ResNet parameters
for param in resnet_modified.parameters():
    param.requires_grad = False
```

b.Mécanisme d'attention

Il permet au modèle de se concentrer sur des parties spécifiques de l'image (les pixels pertinents) lors de la génération de chaque mot

Calcul des scores d'attention

- **Fusion** : Il combine les informations visuelles et textuelles via des projections linéaires
- **Softmax** : Il applique une fonction Softmax pour transformer les scores en probabilités (poids d'attention) qui totalisent 1.

Création du vecteur de contexte

- Le module calcule une somme pondérée des caractéristiques de l'image en utilisant ces poids.
- Le résultat est un vecteur de contexte, qui est une représentation condensée de "ce que le modèle doit regarder" pour prédire le mot suivant[8].

```python
class AttentionModule(nn.Module):
    def __init__(self, feature_dim, hidden_dim, attention_dim):
        super(AttentionModule, self).__init__()
        self.feature_proj = nn.Linear(feature_dim, attention_dim)
        self.hidden_proj = nn.Linear(hidden_dim, attention_dim)
        self.attention = nn.Linear(attention_dim, 1)
        self.relu = nn.ReLU()
        self.softmax = nn.Softmax(dim=1)

    def forward(self, features, hidden_state):
        # features: (batch, feature_dim, H, W)
        # hidden_state: (batch, hidden_dim)

        batch_size, feature_dim, H, W = features.size()

        # Reshape features: (batch, H*W, feature_dim)
        features_flat = features.view(batch_size, feature_dim, -1).permute(0, 2, 1)
        num_pixels = H * W

        # Project features and hidden state
        features_proj = self.feature_proj(features_flat)
        hidden_proj = self.hidden_proj(hidden_state).unsqueeze(1).expand(-1, num_pixels, -1)

        # Compute attention scores
        combined = self.relu(features_proj + hidden_proj)
        attention_scores = self.attention(combined).squeeze(2)

        # Apply softmax
        attention_weights = self.softmax(attention_scores)

        # Compute context vector
        context_vector = torch.bmm(attention_weights.unsqueeze(1), features_flat).squeeze(1)

        return context_vector, attention_weights
```

c.Implémentation du Modèle LSTM

implémentant une version personnalisée d'un réseau LSTM intégrant directement le mécanisme d'attention, car le module LSTM standard de PyTorch ne permet pas d'injecter le vecteur de contexte à chaque étape

Structure des Portes (Gates)
La classe définit quatre couches linéaires qui correspondent aux calculs des portes du LSTM, où l'entrée est une concaténation du mot actuel (x_t), de l'état caché précédent (h_{t-1}) et du vecteur d'attention (att_t)

- **Porte d'entrée (i_t)** : Décide quelles nouvelles informations seront stockées.
- **Porte d'oubli (f_t)** : Décide quelles informations de l'état précédent doivent être jetées.
- **Mise à jour de la cellule (c_t)** : Calcule le nouvel état de la mémoire interne.
- **Porte de sortie (o_t)** : Détermine la valeur de l'état caché final (h_t)

i**ntégration de l'Attention**
- À chaque pas de temps (t), la méthode forward appelle le attention_module pour calculer **le vecteur de contexte** basé sur les caractéristiques de l'image de ResNet et l'état caché actuel.Ce **vecteur de contexte** permet au **LSTM** de savoir précisément quelle partie de l'image regarder pour prédire le mot suivant

Méthodes de Propagation
- **forward** : Traite une séquence entière (utilisée pendant l'entraînement).
- **forward_step** : Traite un seul mot à la fois (utilisée lors de l'évaluation ou de la génération de légendes en temps réel).

```python
class LSTMWithAttention(nn.Module):
    def __init__(self, input_size, hidden_size, attention_module, feature_dim, dropout=0.5):
        super(LSTMWithAttention, self).__init__()
        self.hidden_size = hidden_size
        self.attention_module = attention_module

        lstm_input_size = input_size + hidden_size + feature_dim

        # LSTM gates
        self.W_i = nn.Linear(lstm_input_size, hidden_size)
        self.W_f = nn.Linear(lstm_input_size, hidden_size)
        self.W_c = nn.Linear(lstm_input_size, hidden_size)
        self.W_o = nn.Linear(lstm_input_size, hidden_size)

        self.dropout = nn.Dropout(dropout)
        self.sigmoid = nn.Sigmoid()
        self.tanh = nn.Tanh()
```

d.Architecture Génerale

La classe ImageCaptioningModel, qui est le cœur de votre projetElle orchestre l'interaction entre la vision (ResNet), le langage (Embeddings) et le mécanisme de raisonnement temporel (LSTM + Attention)

- **Extraction Visuelle** : Il utilise le modèle CNN (ResNet50 modifié) pour extraire les caractéristiques spatiales de l'image (2048 dimensions par zone).
- **Traitement du Langage** : Il transforme les séquences de mots en vecteurs denses grâce au modèle d'embedding (Word2Vec).
- **Génération de Séquence :** Le LSTM avec attention traite les mots un par un en "regardant" l'image aux endroits pertinents pour produire un état caché.
- **Prédiction Finale** : Une couche linéaire (self.fc) transforme la sortie du LSTM en un score pour chaque mot du vocabulaire, permettant de choisir le mot le plus probable.

```python
class ImageCaptioningModel(nn.Module):
    def __init__(self, cnn_model, embedding_model, lstm_hidden_dim, vocab_size, dropout=0.5):
        super(ImageCaptioningModel, self).__init__()
        self.cnn = cnn_model
        self.embedding = embedding_model
        self.vocab_size = vocab_size
        self.lstm_hidden_dim = lstm_hidden_dim

        feature_dim = 2048
        attention_dim = config.ATTENTION_DIM

        self.attention = AttentionModule(feature_dim, lstm_hidden_dim, attention_dim)

        embed_dim = embedding_model.weight.size(1)

        self.lstm = LSTMWithAttention(
            embed_dim, lstm_hidden_dim, self.attention, feature_dim, dropout=dropout
        )

        self.fc = nn.Linear(lstm_hidden_dim, vocab_size)
        self.dropout = nn.Dropout(dropout)
```

Le flux de données (forward)

La méthode forward définit comment les données circulent pendant l'entraînement:

- **Vision :** Les images passent par le CNN (sous torch.no_grad() car le ResNet est gelé) pour obtenir les conv_features.
- **Langage :** Les légendes (moins le dernier mot pour la prédiction) sont converties en embeddings.
- **Fusion :** Le LSTM reçoit ces embeddings et les caractéristiques visuelles pour générer une séquence de vecteurs cachés.
- **Sortie** : Le modèle renvoie les prédictions pour chaque étape de la phrase, qui seront comparées aux mots réels par la fonction de perte.

6. Entraînement du Modèle

a.la configuration de l'entraînement

Configuration de la stratégie d'apprentissage et de suivi pour l'entraînement du modèle

- **Optimisation** : Il prépare l'optimiseur Adam pour mettre à jour uniquement les paramètres entraînables avec un taux d'apprentissage de 0.001.
- **Planification (Scheduler)** : Il définit un StepLR qui réduit le taux d'apprentissage selon un facteur gamma fixe toutes les époques pour stabiliser la convergence.
- **Suivi (Logging)** : Il initialise TensorBoard pour enregistrer l'évolution de la perte toutes les 10 itérations, permettant de visualiser les performances en temps réel

```
Training setup complete
Learning rate: 0.001
Epochs: 50
Batch size: 64
Weight decay: 1e-05
Scheduler: StepLR(step_size=10, gamma=0.5)
```

b.inférence autorégressive

permettant au modèle de générer une légende mot après mot pour une nouvelle image.

- **Initialisation Séquentielle** : Elle commence par envoyer le token de début (START_TOKEN) et les caractéristiques de l'image extraites par le ResNet au LSTM.
- **Décodage Pas à Pas** : À chaque itération, elle utilise la méthode forward_step du LSTM pour prédire le mot le plus probable (via argmax), puis réinjecte ce mot comme entrée pour l'étape suivante.
- **Condition d'Arrêt :** La génération s'arrête automatiquement si le modèle produit le token de fin (END_TOKEN) ou si la longueur maximale définie dans la configuration est atteint

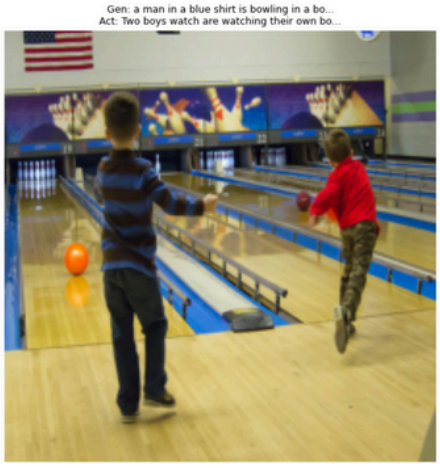
c.Gestion de l'Entraînement et de l'Optimisation

- Le code exécute les passages avant (forward) et arrière (backward) sur le train_loader, calculant la perte via la CrossEntropyLoss.
- Il utilise le clipping de gradient (clip_grad_norm_) pour stabiliser l'entraînement, une pratique courante pour les architectures RNN/LSTM.
- La perte d'entraînement est enregistrée dans TensorBoard toutes les 10 itérations pour un suivi en temps réel
- Optimiseur : Utilisation de l'algorithme **Adam**
- À la fin de chaque époque, le modèle passe en mode évaluation (model.eval()) pour calculer la perte sur l'ensemble de test.

```
Epoch [48/50], Batch [0/398], Loss: 3.6708
Epoch [48/50], Batch [50/398], Loss: 3.4069
Epoch [48/50], Batch [100/398], Loss: 3.7409
Epoch [48/50], Batch [150/398], Loss: 3.5726
Epoch [48/50], Batch [200/398], Loss: 3.4573
Epoch [48/50], Batch [250/398], Loss: 3.7855
Epoch [48/50], Batch [300/398], Loss: 3.5378
Epoch [48/50], Batch [350/398], Loss: 3.6357
Epoch [48/50]
Train Loss: 3.5404, Test Loss: 3.3840
Generated Caption: a girl in a blue shirt is sitting on a bench
Learning Rate: 0.000063
Patience: 1/5
Epoch [49/50], Batch [0/398], Loss: 3.5510
Epoch [49/50], Batch [50/398], Loss: 3.6817
Epoch [49/50], Batch [100/398], Loss: 3.8590
Epoch [49/50], Batch [150/398], Loss: 3.6269
Epoch [49/50], Batch [200/398], Loss: 3.5736
Epoch [49/50], Batch [250/398], Loss: 3.3964
Epoch [49/50], Batch [300/398], Loss: 3.4718
Epoch [49/50], Batch [350/398], Loss: 3.5247
Epoch [49/50]
Train Loss: 3.5425, Test Loss: 3.3561
Generated Caption: a little girl in a blue shirt is sitting on a bench
Learning Rate: 0.000063
✓ Best model saved!
Epoch [50/50], Batch [0/398], Loss: 3.5355
Epoch [50/50], Batch [50/398], Loss: 3.3637
Epoch [50/50], Batch [100/398], Loss: 3.6817
Epoch [50/50], Batch [150/398], Loss: 3.6739
Epoch [50/50], Batch [200/398], Loss: 3.6302
Epoch [50/50], Batch [250/398], Loss: 3.2679
Epoch [50/50], Batch [300/398], Loss: 3.5649
Epoch [50/50], Batch [350/398], Loss: 3.3799
Epoch [50/50]
Train Loss: 3.5392, Test Loss: 3.3888
Generated Caption: a little girl in a blue shirt is sitting on a bench
Learning Rate: 0.000063
Patience: 1/5
Training complete!
```

d.Evaluation et visualisation

- Génération : Il utilise votre fonction generate_caption pour que le modèle produise une légende à partir de l'image.
- Référence : Il récupère la légende réelle (actual) écrite par un humain pour la comparaison.
- Pour chaque image, il affiche le résultat généré (Gen) et le texte réel (Act) au-dessus de l'image.



Performances Globales

le modèle d'Image Captioning a franchi avec succès les étapes clés:

- **Compréhension Visuelle :** L'extracteur de caractéristiques (ResNet50 gelé) identifie correctement les objets majeurs (personnes, chiens, terrains de sport).
- **Structure du Langage** : Grâce aux Embeddings Word2Vec et au LSTM, le modèle génère des phrases syntaxiquement correctes en anglais (Sujet + Verbe + Complément).
- **Mécanisme d'Attention** : On observe une corrélation entre les objets présents et les mots choisis (ex: "dog" et "running" pour l'image du chien), prouvant que le vecteur de contexte guide efficacement le décodage.

Points d'amélioration (Biais observés)

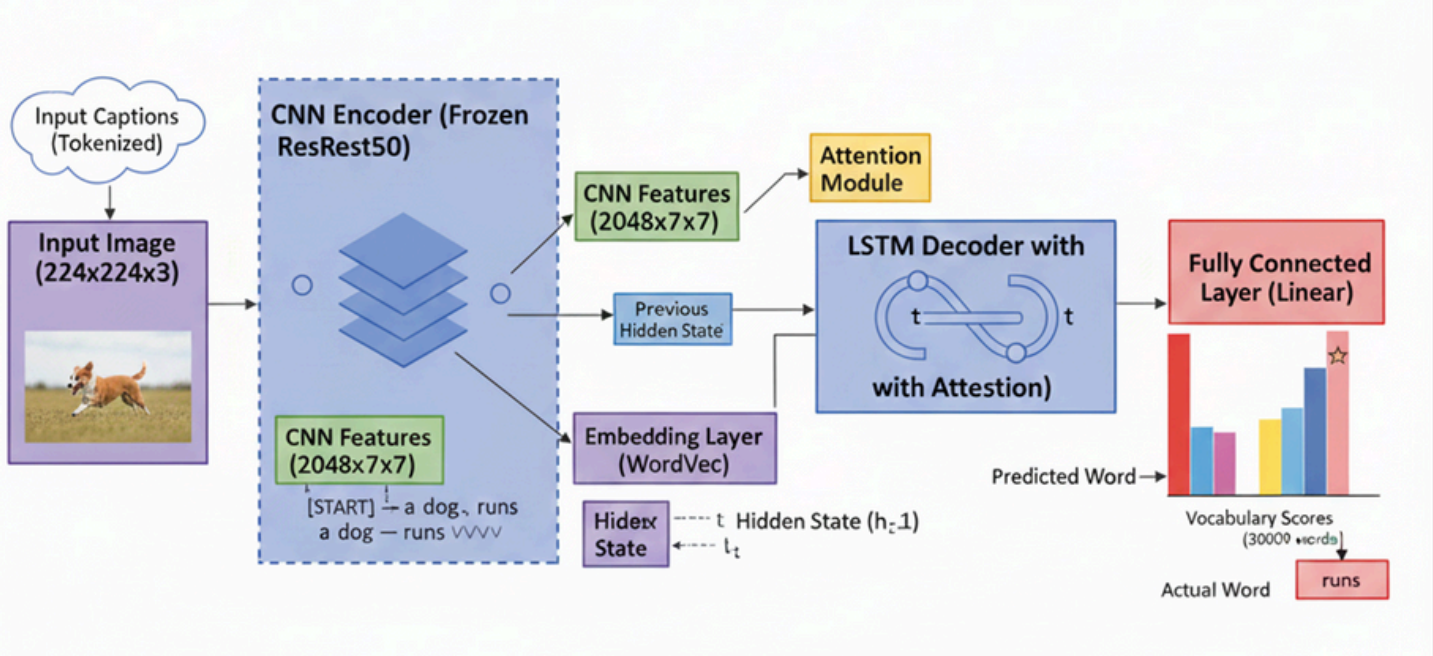
- **Biais de couleur ("Blue shirt")** : On remarque que le modèle prédit très souvent "blue shirt" . C'est un signe que votre dataset contient beaucoup d'exemples de personnes en bleu et que le modèle sur-utilise cette prédiction.
- **Confusion de genre** : Sur l'image 5 (la patineuse), le modèle prédit "a man in a white shirt". Cela arrive souvent quand les poids du LSTM ne sont pas encore assez fins pour distinguer des détails subtils.
- **Hallucination d'objets** : Sur l'image 6 (le chanteur), il prédit "playing a guitar" alors qu'il tient un micro. Le modèle associe la scène de concert à une guitare par probabilité statistique

Justification d’utilisation de 50 epochs au lieu de 1000

En raison des contraintes de ressources computationnelles et du temps de calcul élevé par époque (lié à la taille du dataset Flickr30k et à la complexité du mécanisme d'attention), le nombre d'époques a été limité à 50 au lieu des 1000 initialement prévus. Cette durée s'est avérée suffisante pour valider l'architecture et obtenir des légendes cohérentes, prouvant que le modèle a capté les relations sémantiques essentielles entre les images et le texte.

l'architecture Génerale de ce TP

l'architecture complète de type Encodeur-Décodeur avec Attention implémentée pour ce TP Il montre comment une image brute est transformée étape par étape en une phrase descriptive.



Conclusion

Ce TP de légendage automatique d'images (image captioning) a permis d'implémenter avec succès une architecture hybride complexe associant la vision par ordinateur et le traitement automatique du langage naturel. En utilisant le transfert d'apprentissage via un modèle ResNet50 pré-entraîné dont les poids ont été gelés, nous avons pu extraire des caractéristiques spatiales pertinentes tout en optimisant les ressources de calcul. L'intégration d'un mécanisme d'attention spatiale a constitué une étape clé, permettant au modèle de focaliser dynamiquement sur des régions spécifiques de l'image pour chaque mot généré, conformément aux équations théoriques du TP.

Le décodage, assuré par un LSTM personnalisé, a démontré sa capacité à produire des séquences grammaticalement cohérentes en fusionnant les informations visuelles et textuelles à chaque étape temporelle. Bien que l'entraînement ait été limité à 50 époques au lieu de 1000 pour des raisons de contraintes matérielles et de temps de calcul, les résultats qualitatifs obtenus sur le dataset Flickr30k valident la pertinence de l'approche. Le modèle parvient à identifier les sujets et les actions principales, et l'implémentation de l'Early Stopping a garanti une sauvegarde des poids optimaux avant l'apparition du surapprentissage.

Ce TP confirme ainsi l'efficacité des réseaux de neurones récurrents couplés à l'attention pour la compréhension de scènes visuelles complexes