atelier-4-dl-3

1 Atelier 4 : Réseaux de neurones récurrents (RNNs)

Réaliser par **CHAIMAA BOUABD**

Dans cet atelier, j'ai implémenter 3 modèle de Réseaux de neurones récurrents (RNNs) capable de sous-titrer une image d'entrée (Image Captioning).

- Model 1 : Xception avec LSTM
- Model 2 : VGG16 avec GRU
- Model 3 : VGG16 avec LSTM

1.1 1. Tutorial

[2]:
```
import numpy as np
from PIL import Image
import os
import string
import time
import progressbar
from pickle import dump
from pickle import load
import pandas as ps
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
import sys, time, os, warnings
import numpy as np
from collections import Counter
from nltk.tokenize import word_tokenize
warnings.filterwarnings("ignore")
from keras.applications.xception import Xception #to get pre-trained model␣
 ↪Xception
from keras.applications.xception import preprocess_input
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.text import Tokenizer #for text tokenization
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
#from keras.layers.merge import add
```

```
from keras.layers import add
from keras.models import Model, load_model
from keras.layers import Input, Dense#Keras to build our CNN and LSTM
from keras.layers import LSTM, Embedding, Dropout
from tqdm import tqdm_notebook as tqdm #to check loop progress
tqdm().pandas()
```

```
0it [00:00, ?it/s]
```

[3]:
```
import os
import urllib
from zipfile import ZipFile

# Create directories
data_dir = './data'
os.makedirs(data_dir, exist_ok=True)

# Download and extract Flickr_8k_Dataset
flickr_dataset_url = 'https://github.com/jbrownlee/Datasets/releases/download/
 ↪Flickr8k/Flickr8k_Dataset.zip'
flickr_dataset_zip = os.path.join(data_dir, 'Flickr8k_Dataset.zip')
urllib.request.urlretrieve(flickr_dataset_url, flickr_dataset_zip)

with ZipFile(flickr_dataset_zip, 'r') as zip_ref:
    zip_ref.extractall(data_dir)

# Download and extract Flickr_8k_text
flickr_text_url = 'https://github.com/jbrownlee/Datasets/releases/download/
 ↪Flickr8k/Flickr8k_text.zip'
flickr_text_zip = os.path.join(data_dir, 'Flickr8k_text.zip')
urllib.request.urlretrieve(flickr_text_url, flickr_text_zip)

with ZipFile(flickr_text_zip, 'r') as zip_ref:
    zip_ref.extractall(data_dir)
```

[4]:
```
# Check the contents of the data directory
os.listdir(data_dir)
```

[4]:
```
['Flickr_8k.trainImages.txt',
 'Flickr_8k.devImages.txt',
 'CrowdFlowerAnnotations.txt',
 'Flickr8k_Dataset.zip',
 'Flickr8k.token.txt',
 'Flickr_8k.testImages.txt',
 '__MACOSX',
 'Flicker8k_Dataset',
 'ExpertAnnotations.txt',
```

```
 'Flickr8k.lemma.token.txt',
 'Flickr8k_text.zip',
 'readme.txt']
```

[24]:
```
# Load the document file into memory
def load_fp(filename):
    # Open file to read
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text
```

[25]:
```
# Get all images with their captions
def img_capt(filename):
    file = load_fp(filename)  # Change load_doc to load_fp
    captions = file.split('\n')  # Fix 'n' to '\n'
    descriptions = {}

    for caption in captions[:-1]:
        img, caption = caption.split('\t')  # Fix 't' to '\t'
        if img[:-2] not in descriptions:
            descriptions[img[:-2]] = [caption]
        else:
            descriptions[img[:-2]].append(caption)

    return descriptions
```

[26]:
```
# Data cleaning function will convert all upper case alphabets to lowercase,␣
 ↪removing punctuations and words containing numbers
def txt_clean(captions):
    table = str.maketrans('', '', string.punctuation)
    for img, caps in captions.items():
        for i, img_caption in enumerate(caps):
            img_caption.replace("-", " ")
            descp = img_caption.split()
            # Uppercase to lowercase
            descp = [wrd.lower() for wrd in descp]
            # Remove punctuation from each token
            descp = [wrd.translate(table) for wrd in descp]
            # Remove hanging 's and a
            descp = [wrd for wrd in descp if (len(wrd) > 1)]
            # Remove words containing numbers
            descp = [wrd for wrd in descp if (wrd.isalpha())]
            # Converting back to string
            img_caption = ' '.join(descp)
            captions[img][i] = img_caption
```

```
    return captions
```

[27]:
```
def txt_vocab(descriptions):
    # To build vocab of all unique words
    vocab = set()
    for key in descriptions.keys():
        [vocab.update(d.split()) for d in descriptions[key]]

    return vocab
```

[28]:
```
# To save all descriptions in one file
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + '\t' + desc)  # Fix 't' to '\t'
    data = "\n".join(lines)  # Fix 'n' to '\n'
    file = open(filename, "w")
    file.write(data)
    file.close()

# Set these paths according to the project folder in your system
dataset_text = "data"
dataset_images = "data/Flicker8k_Dataset"

# To prepare our text data
filename = dataset_text + "/" + "Flickr8k.token.txt"

# Loading the file that contains all data and mapping them into descriptions␣
 ↪dictionary
descriptions = img_capt(filename)
print("Length of descriptions =", len(descriptions))

# Cleaning the descriptions
clean_descriptions = txt_clean(descriptions)

# To build vocabulary
vocabulary = txt_vocab(clean_descriptions)
print("Length of vocabulary =", len(vocabulary))

# Saving all descriptions in one file
save_descriptions(clean_descriptions, "descriptions.txt")
```

```
Length of descriptions = 8092
Length of vocabulary = 8763
```

1.1.1 Extract the Feature Vector

[ ]:
```
def extract_features(directory):
  model = Xception( include_top=False, pooling='avg' )
  features = {}
  for pic in tqdm(os.listdir(directory)):
            file = directory + "/" + pic
            image = Image.open(file)
            image = image.resize((299,299))
            image = np.expand_dims(image, axis=0)
           #image = preprocess_input(image)
            image = image/127.5
            image = image - 1.0
            feature = model.predict(image)
            features[pic] = feature
  return features
#2048 feature vector
features = extract_features(dataset_images)
dump(features, open("features.p","wb"))
#to directly load the features from the pickle file.
features = load(open("features.p","rb"))
```

1.1.2 Loading dataset for model training

[30]:
```
# Load the data
def load_photos(filename):
    file = load_fp(filename)
    photos = file.split("\n")[:-1]  # Change 'n' to '\n'
    return photos
```

[31]:
```
def load_clean_descriptions(filename, photos):
    # Loading clean_descriptions
    file = load_fp(filename)
    descriptions = {}

    for line in file.split("\n"):
        words = line.split()
        if len(words) < 1:
            continue
        image, image_caption = words[0], words[1:]

        if image in photos:
            if image not in descriptions:
                descriptions[image] = []
            desc = ' ' + " ".join(image_caption) + ' '
            descriptions[image].append(desc)
```

```
    return descriptions
```

[33]:
```
def load_features(photos):
    # Loading all features
    all_features = load(open("features.p", "rb"))  # You need to define 'load' function and import it
    # Selecting only needed features
    features = {k: all_features[k] for k in photos}
    return features

# Set these paths according to the project folder in your system
dataset_text = "./data"  # Adjust this path accordingly

filename_train = dataset_text + "/" + "Flickr_8k.trainImages.txt"
filename_test = dataset_text + "/" + "Flickr_8k.testImages.txt"
# Load train data
train_imgs = load_photos(filename_train)
train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)
train_features = load_features(train_imgs)
# Load test data
test_imgs = load_photos(filename_test)
test_descriptions = load_clean_descriptions("descriptions.txt", test_imgs)
test_features = load_features(test_imgs)
```

1.1.3 Tokenizing the Vocabulary

[34]:
```
# Convert dictionary to a clear list of descriptions
def dict_to_list(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

# Creating tokenizer class
# This will vectorize the text corpus
# Each integer will represent a token in the dictionary
from keras.preprocessing.text import Tokenizer

def create_tokenizer(descriptions):
    desc_list = dict_to_list(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(desc_list)
    return tokenizer

# Give each word an index and store that into the tokenizer.p pickle file
tokenizer = create_tokenizer(train_descriptions)
dump(tokenizer, open('tokenizer.p', 'wb'))
```

```
vocab_size = len(tokenizer.word_index) + 1
print("Vocabulary size:", vocab_size)

# Calculate the maximum length of descriptions to decide the model structure␣
 ↪parameters
def max_length(descriptions):
    desc_list = dict_to_list(descriptions)
    return max(len(d.split()) for d in desc_list)

max_length = max_length(descriptions)
print("Max length of description:", max_length)  # Fix 'Max_length' to␣
 ↪'print("Max length of description:", max_length)'
```

```
Vocabulary size: 7577
Max length of description: 32
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[1]:
```
#import pickle

# Example: Save a tokenizer object to a file
#tokenizer = ...  # Your tokenizer object
#with open('/tokenizer.p', 'wb') as f:
#    pickle.dump(tokenizer, f)

# Save descriptions to a text file
#descriptions = ...  # Your descriptions data
#with open('./descriptions.txt', 'w') as f:
#    for desc in descriptions:
#        f.write(desc + '\n')
```

[ ]:
```
#import pickle
# Save features to a pickle file
#features = ...  # Your features data
#with open('./features.p', 'wb') as f:
#    pickle.dump(features, f)
```

1.1.4 Create a Data generator

[35]:
```
# Data generator, used by model.fit_generator()
def data_generator(descriptions, features, tokenizer, max_length):
    while 1:
        for key, description_list in descriptions.items():
```

```
            # Retrieve photo features
            feature = features[key][0]
            inp_image, inp_seq, op_word = create_sequences(tokenizer,␣
 ↪max_length, description_list, feature)
            yield [[inp_image, inp_seq], op_word]

def create_sequences(tokenizer, max_length, desc_list, feature):
    x_1, x_2, y = list(), list(), list()
    # Move through each description for the image
    for desc in desc_list:
        # Encode the sequence
        seq = tokenizer.texts_to_sequences([desc])[0]
        # Divide one sequence into various X, y pairs
        for i in range(1, len(seq)):
            # Divide into input and output pair
            in_seq, out_seq = seq[:i], seq[i]
            # Pad input sequence
            in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
            # Encode output sequence
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            # Store
            x_1.append(feature)
            x_2.append(in_seq)
            y.append(out_seq)
    return np.array(x_1), np.array(x_2), np.array(y)

# To check the shape of the input and output for your model
[a, b], c = next(data_generator(train_descriptions, features, tokenizer,␣
 ↪max_length))
print(a.shape, b.shape, c.shape)
# ((47, 2048), (47, 32), (47, 7577))
```

```
(37, 2048) (37, 32) (37, 7577)
```

1.1.5 Define the CNN-RNN model

[48]:
```
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input, Dense, Dropout, Embedding, LSTM, add

# Define the captioning model
def define_model(vocab_size, max_length):
    # Features from the CNN model compressed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)
```

```
    # LSTM sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

    # Merging both models
    decoder1 = add([fe2, se3])
    decoder2 = Dense(256, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)

    # Merge it [image, seq] [word]
    model_tuto = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model_tuto.compile(loss='categorical_crossentropy', optimizer='adam',␣
 ↪metrics=['accuracy'])

    # Summarize model
    print(model_tuto.summary())
    plot_model(model_tuto, to_file='model.png', show_shapes=True)

    return model_tuto
```

1.1.6 Training the Image Caption Generator model

[49]:
```
# train our model
print('Dataset: ', len(train_imgs))
print('Descriptions: train=', len(train_descriptions))
print('Descriptions: test=', len(test_descriptions))
print('Photos: train=', len(train_features))
print('Photos: test=', len(test_features))
print('Vocabulary Size:', vocab_size)
print('Description Length: ', max_length)
model_tuto = define_model(vocab_size, max_length)
epochs = 10
steps_per_epoch = len(train_descriptions)
validation_steps = len(test_descriptions)
# creating a directory named models to save our models
#os.mkdir("models")
histories = []
for i in range(epochs):
   print("Epoch : ", i)
   generator_train = data_generator(train_descriptions, train_features,␣
 ↪tokenizer, max_length)
   generator_test = data_generator(test_descriptions, test_features, tokenizer,␣
 ↪max_length)
```

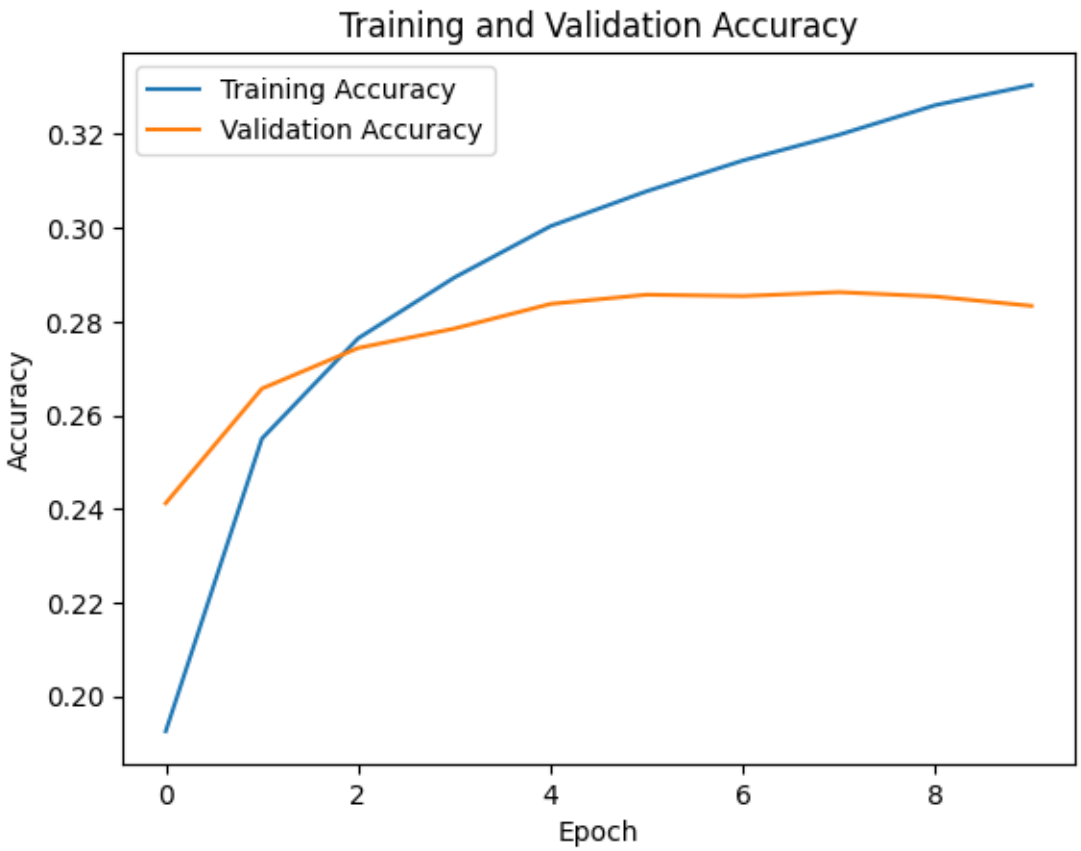
```
  hist = model_tuto.fit_generator(generator_train, epochs=1, steps_per_epoch=␣
↪steps_per_epoch, validation_data = generator_test, validation_steps =␣
↪validation_steps)
  histories.append(hist)
  model_tuto.save("models/model_" + str(i) + ".h5")
```

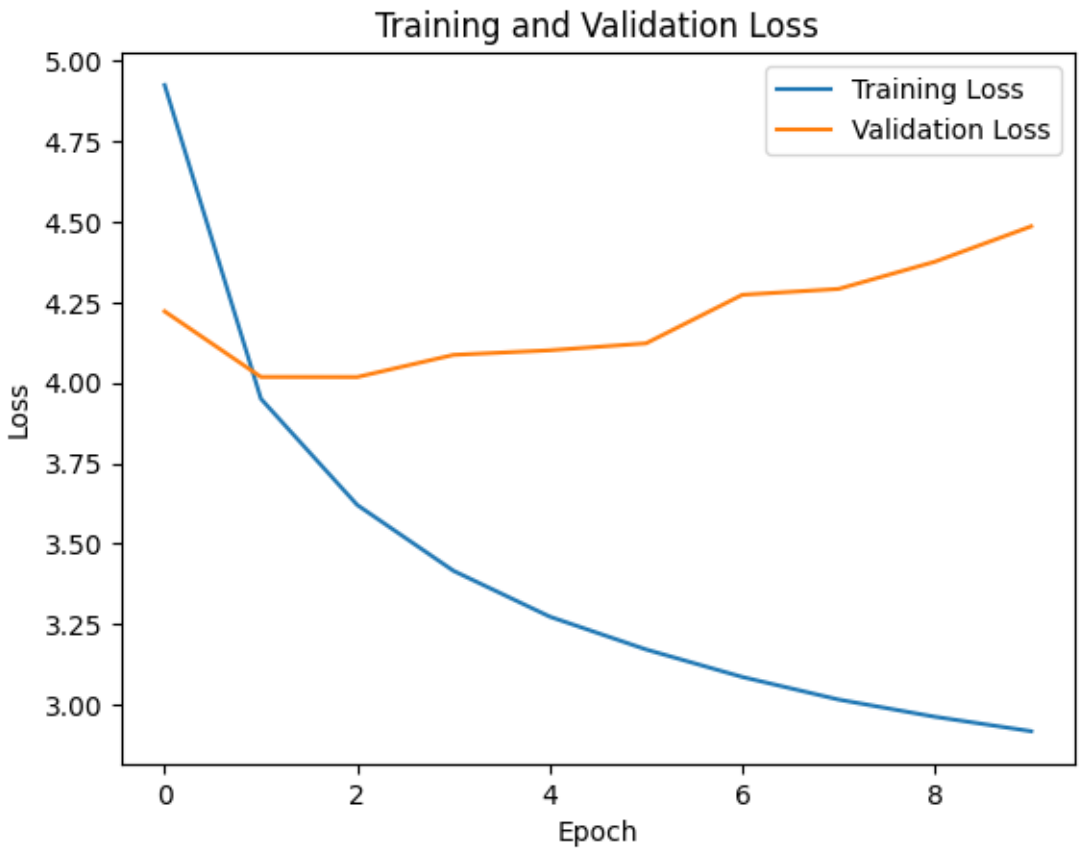
```
Dataset:  6000
Descriptions: train= 6000
Descriptions: test= 1000
Photos: train= 6000
Photos: test= 1000
Vocabulary Size: 7577
Description Length:  32
Model: "model_8"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================
==================
 input_18 (InputLayer)       [(None, 32)]                 0         []

 input_17 (InputLayer)       [(None, 2048)]               0         []

 embedding_7 (Embedding)     (None, 32, 256)              1939712
['input_18[0][0]']

 dropout_12 (Dropout)        (None, 2048)                 0
['input_17[0][0]']

 dropout_13 (Dropout)        (None, 32, 256)              0
['embedding_7[0][0]']

 dense_20 (Dense)            (None, 256)                  524544
['dropout_12[0][0]']

 lstm_6 (LSTM)               (None, 256)                  525312
['dropout_13[0][0]']

 add_19 (Add)                (None, 256)                  0
['dense_20[0][0]',
'lstm_6[0][0]']

 dense_21 (Dense)            (None, 256)                  65792
['add_19[0][0]']

 dense_22 (Dense)            (None, 7577)                 1947289
['dense_21[0][0]']
```

```
=================================================================================
=================
Total params: 5002649 (19.08 MB)
Trainable params: 5002649 (19.08 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________________________
_________________
None
Epoch :  0
6000/6000 [==============================] - 526s 87ms/step - loss: 4.9241 -
accuracy: 0.1925 - val_loss: 4.2214 - val_accuracy: 0.2412
Epoch :  1
6000/6000 [==============================] - 508s 85ms/step - loss: 3.9499 -
accuracy: 0.2550 - val_loss: 4.0177 - val_accuracy: 0.2657
Epoch :  2
6000/6000 [==============================] - 506s 84ms/step - loss: 3.6203 -
accuracy: 0.2764 - val_loss: 4.0174 - val_accuracy: 0.2743
Epoch :  3
6000/6000 [==============================] - 506s 84ms/step - loss: 3.4157 -
accuracy: 0.2894 - val_loss: 4.0862 - val_accuracy: 0.2785
Epoch :  4
6000/6000 [==============================] - 507s 85ms/step - loss: 3.2734 -
accuracy: 0.3004 - val_loss: 4.1004 - val_accuracy: 0.2837
Epoch :  5
6000/6000 [==============================] - 501s 83ms/step - loss: 3.1715 -
accuracy: 0.3078 - val_loss: 4.1221 - val_accuracy: 0.2857
Epoch :  6
6000/6000 [==============================] - 502s 84ms/step - loss: 3.0861 -
accuracy: 0.3144 - val_loss: 4.2728 - val_accuracy: 0.2854
Epoch :  7
6000/6000 [==============================] - 502s 84ms/step - loss: 3.0159 -
accuracy: 0.3199 - val_loss: 4.2911 - val_accuracy: 0.2863
Epoch :  8
6000/6000 [==============================] - 506s 84ms/step - loss: 2.9625 -
accuracy: 0.3262 - val_loss: 4.3756 - val_accuracy: 0.2853
Epoch :  9
6000/6000 [==============================] - 505s 84ms/step - loss: 2.9177 -
accuracy: 0.3305 - val_loss: 4.4854 - val_accuracy: 0.2833
```

[50]:
```python
import matplotlib.pyplot as plt

# Function to plot training and validation loss
def plot_loss(history,history_val):
  plt.plot(history, label='Training Loss')
  plt.plot(history_val, label='Validation Loss')
  plt.title('Training and Validation Loss')
```

```
  plt.xlabel('Epoch')
  plt.ylabel('Loss')
  plt.legend()
  plt.show()

# Function to plot training and validation accuracy
def plot_accuracy(history,history_val):
  plt.plot(history, label='Training Accuracy')
  plt.plot(history_val, label='Validation Accuracy')
  plt.title('Training and Validation Accuracy')
  plt.xlabel('Epoch')
  plt.ylabel('Accuracy')
  plt.legend()
  plt.show()

losses = []
accuracies = []
val_losses = []
val_accuracies = []

# Plot the loss and accuracy for each epoch
for i, history in enumerate(histories):

  losses.append(histories[i].history['loss'])
  accuracies.append(histories [i].history['accuracy'])
  val_losses.append(histories [i].history['val_loss'])
  val_accuracies.append(histories[i].history['val_accuracy'])

plot_accuracy(accuracies, val_accuracies)
plot_loss(losses, val_losses)
```


Training and Validation Accuracy
Training Accuracy
Validation Accuracy
Accuracy
0.32
0.30
0.28
0.26
0.24
0.22
0.20
0
2
4
6
8
Epoch



1.1.7 save the model to GOOGLE Drive

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[51]:
```
import shutil

# Replace 'your_folder_path' with the actual path of the folder
folder_path = './models'

# Create a zip archive
shutil.make_archive('/content/tuto_models', 'zip', folder_path)

# Copy the zip archive to Google Drive
shutil.copy('/content/tuto_models.zip', '/content/drive/My Drive/tuto_models.
 ↪zip')
```

```
# Remove the original zip file in /content (optional)
#shutil.rmtree('/content/upload_folder')

# Print a message indicating the completion
print("Folder tuto_models uploaded to Google Drive.")
```

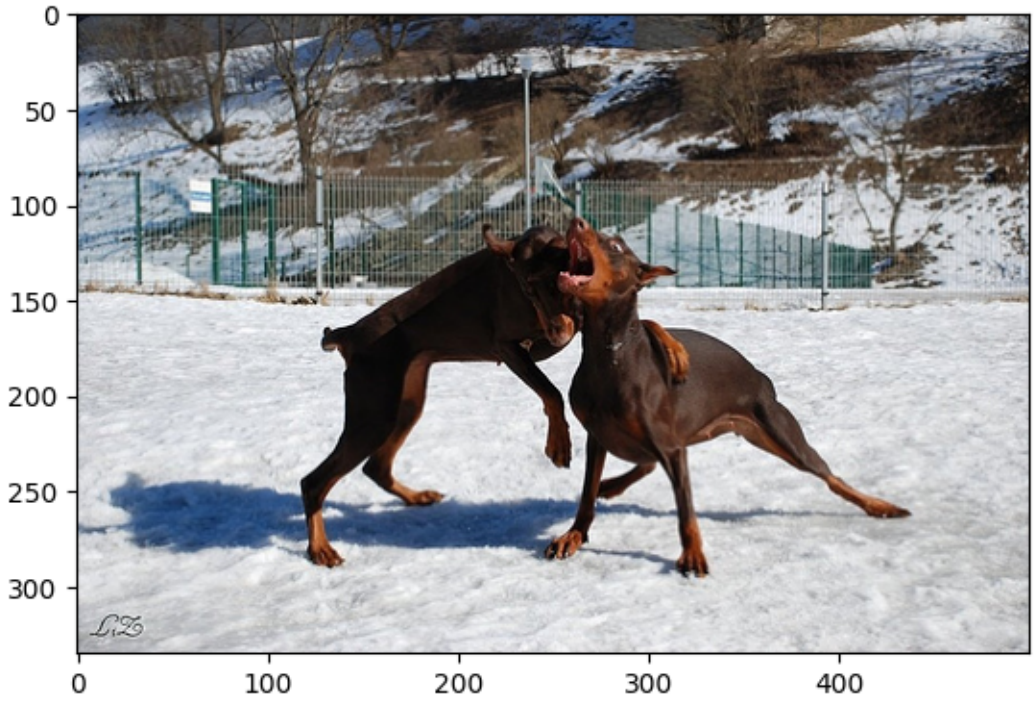
```
Folder tuto_models uploaded to Google Drive.
```

1.1.8 Testing the Image Caption Generator model

[55]:
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
from keras.models import load_model
from keras.applications.xception import Xception
from keras.applications.xception import preprocess_input
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from pickle import load

# Define functions for feature extraction and caption generation
def extract_features(filename, model):
    try:
        image = Image.open(filename)
    except:
        print("ERROR: Can't open image! Ensure that the image path and␣
 ↪extension are correct")
        return None

    image = image.resize((299, 299))
    image = np.array(image)

    # For 4 channels images, convert them into 3 channels
    if image.shape[2] == 4:
        image = image[..., :3]

    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)
    feature = model.predict(image)
    return feature

def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```

```
def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'start'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        pred = model.predict([photo, sequence], verbose=0)
        pred = np.argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'end':
            break
    return in_text

# Set the maximum sequence length
max_length = 32

# Load the tokenizer
tokenizer = load(open("tokenizer.p", "rb"))

# Load the trained model
model = load_model('models/model_9.h5')

# Load Xception model for feature extraction
xception_model = Xception(include_top=False, pooling="avg")

# Provide the image path directly (replace 'your_image_path.jpg' with the␣
 ↪actual path)
img_path = '/content/data/Flicker8k_Dataset/3385593926_d3e9c21170.jpg'
# Extract features and generate caption
photo = extract_features(img_path, xception_model)
if photo is not None:
    description = generate_desc(model, tokenizer, photo, max_length)
    print(description)

    # Display the image
    img = Image.open(img_path)
    plt.imshow(img)
    plt.show()
```

```
1/1 [==============================] - 1s 711ms/step
start of doberman is running through the grass with stick in its mouth and black
dog in the background with the water behind it and black dog in the background
and brown dog
```



2 2. Changer l'architecture CNN utilisée dans le tutoriel par l' architectures VGGNet16 pré-entraîné sur ImageNet, et ré-entraîner le modèle avec GRU.

[56]:
```
dir_Flickr_text = "/content/data/Flickr8k.token.txt"
dir_Flickr_jpg = "/content/data/Flicker8k_Dataset"

jpgs = os.listdir(dir_Flickr_jpg)
print("The number of jpg flies in Flicker30k: {}".format(len(jpgs)))
```

```
The number of jpg flies in Flicker30k: 8091
```

[57]:
```
## loading as dataframe
def load_csv(directory):
    desc=dict()
    text = pd.read_csv(directory,␣
 ↪delimiter='|',header=None,names=["filename","index","caption"])
    text = text.iloc[1:,:]
    df_new = text[text.iloc[:,2].notnull()]
    print(df_new.iloc[:5,:])
    return df_new
```

[58]:
```
import pandas as pd
file = open(dir_Flickr_text,'r')
text = file.read()
file.close()

datatxt = []
for line in text.split('\n'):
    col = line.split('\t')
    if len(col) == 1:
        continue
    w = col[0].split("#")
    datatxt.append(w + [col[1].lower()])

df_txt = pd.DataFrame(datatxt,columns=["filename","index","caption"])


uni_filenames = np.unique(df_txt.filename.values)
print("The number of unique file names : {}".format(len(uni_filenames)))
print("The distribution of the number of captions for each image:")
Counter(Counter(df_txt.filename.values).values())
```

```
The number of unique file names : 8092
The distribution of the number of captions for each image:
```

[58]:
```
Counter({5: 8092})
```

[59]:
```
from keras.preprocessing.image import load_img, img_to_array

npic = 5
npix = 224
target_size = (npix,npix,3)

count = 1
fig = plt.figure(figsize=(10,20))
for jpgfnm in uni_filenames[:npic]:
    filename = dir_Flickr_jpg + '/' + jpgfnm
    captions = list(df_txt["caption"].loc[df_txt["filename"]==jpgfnm].values)
    image_load = load_img(filename, target_size=target_size)

    ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
    ax.imshow(image_load)
    count += 1

    ax = fig.add_subplot(npic,2,count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
```

```
    ax.set_ylim(0,len(captions))
    for i, caption in enumerate(captions):
        ax.text(0,i,caption,fontsize=20)
    count += 1
plt.show()
```

a little girl in a pink dress going into a wooden cabin .
a little girl climbing the stairs to her playhouse .
a little girl climbing into a wooden playhouse .
a girl going into a wooden building .
a child in a pink dress is climbing up a set of stairs in an entry way .

two dogs on pavement moving toward each other .
two dogs of different breeds looking at each other on the road .
a black dog and a white dog with brown spots are staring at each other in the street .
a black dog and a tri-colored dog playing with each other on the road .
a black dog and a spotted dog are fighting

young girl with pigtails painting outside in the grass .
there is a girl with pigtails sitting in front of a rainbow painting .
a small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
a little girl is sitting in front of a large painted rainbow .
a little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .

man laying on bench holding leash of dog sitting on ground
a shirtless man lies on a park bench with his dog .
a man sleeping on a bench outside with a white and black dog sitting next to him .
a man lays on the bench to which a white dog is also tied .
a man lays on a bench while his dog sits by him .

the man with pierced ears is wearing glasses and an orange hat .
a man with glasses is wearing a beer can crocheted hat .
a man with gauges and glasses is wearing a blitz hat .
a man wears an orange hat and glasses .
a man in an orange hat starring at something .

[60]:
```
def df_word(df_txt):
    vocabulary = []
    for i in range(len(df_txt)):
        temp=df_txt.iloc[i,2]
        vocabulary.extend(temp.split())
    print('Vocabulary Size: %d' % len(set(vocabulary)))
    ct = Counter(vocabulary)
    dfword = pd.DataFrame({"word":list(ct.keys()),"count":list(ct.values())})
    dfword = dfword.sort_values("count",ascending=False)
```

```
    dfword = dfword.reset_index()[["word","count"]]
    return(dfword)
dfword = df_word(df_txt)
dfword.head(3)
```

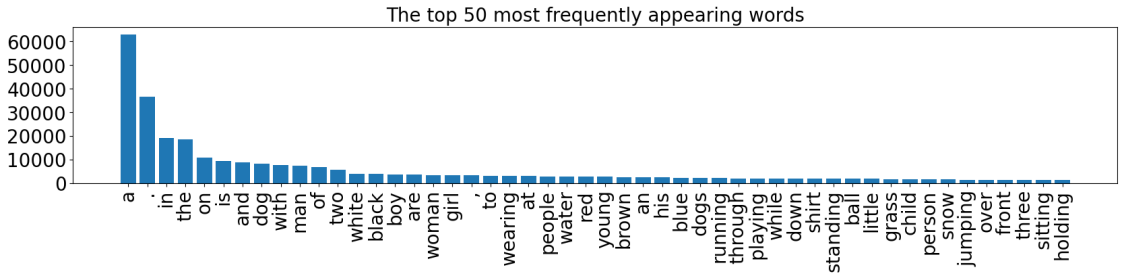
```
Vocabulary Size: 8918
```

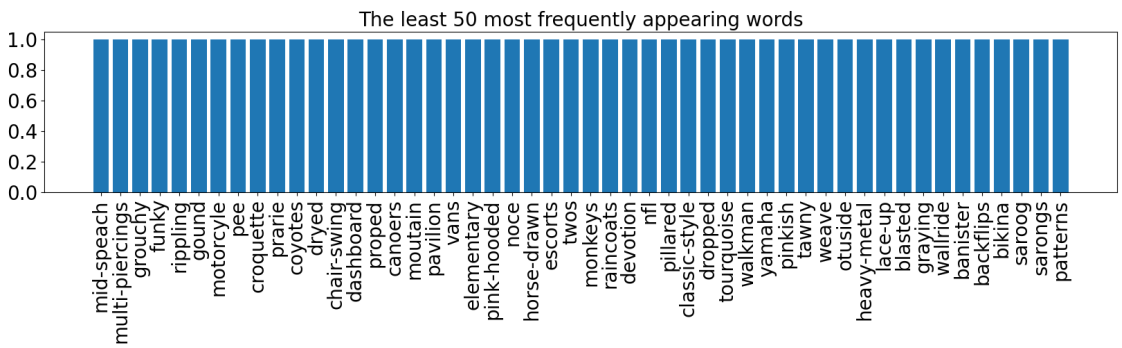
[60]:

	word	count
0	a	62989
1	.	36581
2	in	18975

[61]:

```
topn = 50

def plthist(dfsub, title="The top 50 most frequently appearing words"):
    plt.figure(figsize=(20,3))
    plt.bar(dfsub.index,dfsub["count"])
    plt.yticks(fontsize=20)
    plt.xticks(dfsub.index,dfsub["word"],rotation=90,fontsize=20)
    plt.title(title,fontsize=20)
    plt.show()

plthist(dfword.iloc[:topn,:],
        title="The top 50 most frequently appearing words")
plthist(dfword.iloc[-topn:,:],
        title="The least 50 most frequently appearing words")
```





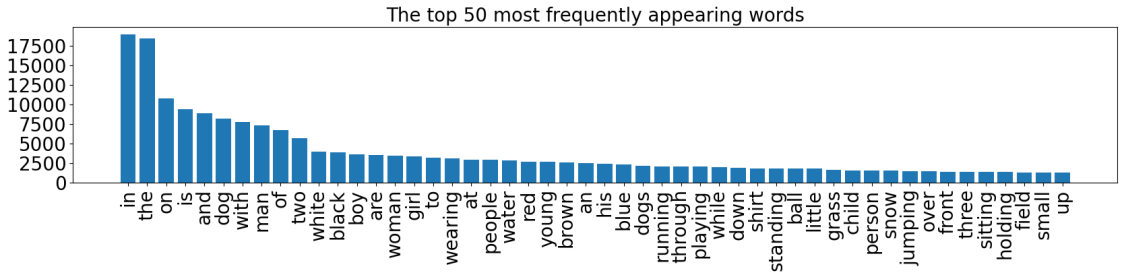
[62]:
```
import string
def remove_punctuation(text_original):
    text_no_punctuation = text_original.translate(str.maketrans('','',string.
 ↪punctuation))
    return(text_no_punctuation)

def remove_single_character(text):
    text_len_more_than1 = ""
    for word in text.split():
        if len(word) > 1:
            text_len_more_than1 += " " + word
    return(text_len_more_than1)

def remove_numeric(text,printTF=False):
    text_no_numeric = ""
    for word in text.split():
        isalpha = word.isalpha()
        if printTF:
            print("    {:10} : {:}".format(word,isalpha))
        if isalpha:
            text_no_numeric += " " + word
    return(text_no_numeric)
```

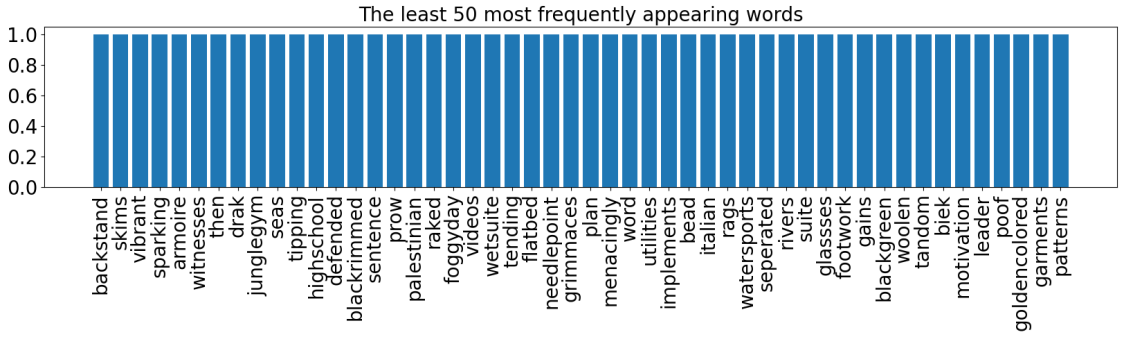
[63]:
```
def text_clean(text_original):
    text = remove_punctuation(text_original)
    text = remove_single_character(text)
    text = remove_numeric(text)
    return(text)

with progressbar.ProgressBar(max_value=len(df_txt.caption.values)) as bar:
    for i, caption in enumerate(df_txt.caption.values):
        newcaption = text_clean(caption)
        df_txt["caption"].iloc[i] = newcaption
        bar.update(i)
```

```
100% (40460 of 40460) |####################| Elapsed Time: 0:00:16 Time:  0:00:16
```

[64]:
```
dfword = df_word(df_txt)
plthist(dfword.iloc[:topn,:],
        title="The top 50 most frequently appearing words")
plthist(dfword.iloc[-topn:,:],
        title="The least 50 most frequently appearing words")
```

```
Vocabulary Size: 8763
```





[65]:
```
from copy import copy
def add_start_end_seq_token(captions):
    caps = []
    for txt in captions:
        txt = 'startseq ' + txt + ' endseq'
        caps.append(txt)
    return(caps)
df_txt0 = copy(df_txt)
df_txt0["caption"] = add_start_end_seq_token(df_txt["caption"])
df_txt0.head(5)
del df_txt
```

[66]:
```
from keras.applications import VGG16

modelvgg = VGG16(include_top=True,weights='imagenet')
modelvgg.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_22 (InputLayer)       [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792
```

```
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 1000)              4097000

=================================================================
Total params: 138357544 (527.79 MB)
Trainable params: 138357544 (527.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[67]:
```
from keras import models
modelvgg.layers.pop()
modelvgg = models.Model(inputs=modelvgg.inputs, outputs=modelvgg.layers[-1].
 ↪output)
## show the deep learning model
modelvgg.summary()
```

```
Model: "model_9"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_22 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
```

```
 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 fc1 (Dense)                 (None, 4096)              102764544

 fc2 (Dense)                 (None, 4096)              16781312

 predictions (Dense)         (None, 1000)              4097000

=================================================================
Total params: 138357544 (527.79 MB)
Trainable params: 138357544 (527.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

[ ]:
```
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import preprocess_input
from collections import OrderedDict

images = OrderedDict()
npix = 224
target_size = (npix,npix,3)
with progressbar.ProgressBar(max_value=len(jpgs)) as bar:
    for i,name in enumerate(jpgs):
        # load an image from file
        filename = dir_Flickr_jpg + '/' + name
        image = load_img(filename, target_size=target_size)
        # convert the image pixels to a numpy array
        image = img_to_array(image)
        nimage = preprocess_input(image)
        y_pred = modelvgg.predict(nimage.reshape( (1,) + nimage.shape[:3]))
        images[name] = y_pred.flatten()
        bar.update(i)
    #print(i,filename)
```

[69]:
```
dimages, keepindex = [],[]
nd=(df_txt0["index"].values)
b = [(int(i)==0) for i in nd]
#for i in nd:
 #   print(int(i)==0)
#df_txt0 = df_txt0.loc[b,: ]
df_txt0 = df_txt0.loc[df_txt0["index"].values == "0",: ]

for i, fnm in enumerate(df_txt0.filename):
    if fnm in images.keys():
        dimages.append(images[fnm])
```

```
        keepindex.append(i)

fnames = df_txt0["filename"].iloc[keepindex].values
dcaptions = df_txt0["caption"].iloc[keepindex].values
dimages = np.array(dimages)
print(df_txt0["index"][:5])
```

```
0     0
5     0
10    0
15    0
20    0
Name: index, dtype: object
```

[70]:
```
from keras.preprocessing.text import Tokenizer
## the maximum number of words in dictionary
count_words=22000
#nb_words = 31782
tokenizer = Tokenizer(num_words=8000)
tokenizer.fit_on_texts(dcaptions)
vocab_size = len(tokenizer.word_index) + 1
print("vocabulary size : {}".format(vocab_size))
dtexts = tokenizer.texts_to_sequences(dcaptions)
print(dtexts[:5])
```

```
vocabulary size : 4476
[[1, 38, 3, 66, 144, 7, 124, 52, 406, 9, 367, 3, 24, 2351, 522, 2], [1, 12, 8,
5, 752, 8, 17, 368, 2], [1, 48, 15, 170, 3, 584, 101, 3, 41, 9, 551, 1198, 11,
55, 213, 3, 1076, 2], [1, 10, 621, 6, 150, 27, 23, 8, 101, 46, 112, 2], [1, 10,
3, 24, 82, 96, 1199, 19, 162, 2]]
```

[71]:
```
prop_test, prop_val = 0.2, 0.2

N = len(dtexts)
Ntest, Nval = int(N*prop_test), int(N*prop_val)

def split_test_val_train(dtexts,Ntest,Nval):
    return(dtexts[:Ntest],
           dtexts[Ntest:Ntest+Nval],
           dtexts[Ntest+Nval:])

dt_test,  dt_val, dt_train   = split_test_val_train(dtexts,Ntest,Nval)
di_test,  di_val, di_train   = split_test_val_train(dimages,Ntest,Nval)
fnm_test,fnm_val, fnm_train  = split_test_val_train(fnames,Ntest,Nval)
```

[72]:
```
maxlen = np.max([len(text) for text in dtexts])
```

[73]:
```
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

def preprocessing(dtexts,dimages):
    N = len(dtexts)
    print("# captions/images = {}".format(N))

    assert(N==len(dimages))
    Xtext, Ximage, ytext = [],[],[]
    for text,image in zip(dtexts,dimages):

        for i in range(1,len(text)):
            in_text, out_text = text[:i], text[i]
            in_text = pad_sequences([in_text],maxlen=maxlen).flatten()
            out_text = to_categorical(out_text,num_classes = vocab_size)

            Xtext.append(in_text)
            Ximage.append(image)
            ytext.append(out_text)

    Xtext  = np.array(Xtext)
    Ximage = np.array(Ximage)
    ytext  = np.array(ytext)
    print(" {} {} {}".format(Xtext.shape,Ximage.shape,ytext.shape))
    return(Xtext,Ximage,ytext)


Xtext_train, Ximage_train, ytext_train = preprocessing(dt_train,di_train)
Xtext_val,   Ximage_val,   ytext_val   = preprocessing(dt_val,di_val)
# pre-processing is not necessary for testing data
#Xtext_test,  Ximage_test,  ytext_test  = preprocessing(dt_test,di_test)
```

```
# captions/images = 4855
 (49631, 30) (49631, 1000) (49631, 4476)
# captions/images = 1618
 (16353, 30) (16353, 1000) (16353, 4476)
```

[74]:
```
from keras import layers
from keras import models

print(vocab_size)

## Image feature
dim_embedding = 64

input_image = layers.Input(shape=(Ximage_train.shape[1],))
fimage = layers.Dense(256, activation='relu', name="ImageFeature")(input_image)
```

```
## Sequence model
input_txt = layers.Input(shape=(maxlen,))
ftxt = layers.Embedding(vocab_size, dim_embedding, mask_zero=True)(input_txt)
ftxt = layers.GRU(256, name="CaptionFeature")(ftxt)  # Change LSTM to GRU

## Combined model for decoder
decoder = layers.add([ftxt, fimage])
decoder = layers.Dense(256, activation='relu')(decoder)
output = layers.Dense(vocab_size, activation='softmax')(decoder)

model_vgg_gru = models.Model(inputs=[input_image, input_txt], outputs=output)
model_vgg_gru.compile(loss='categorical_crossentropy', optimizer='adam')

print(model_vgg_gru.summary())
```

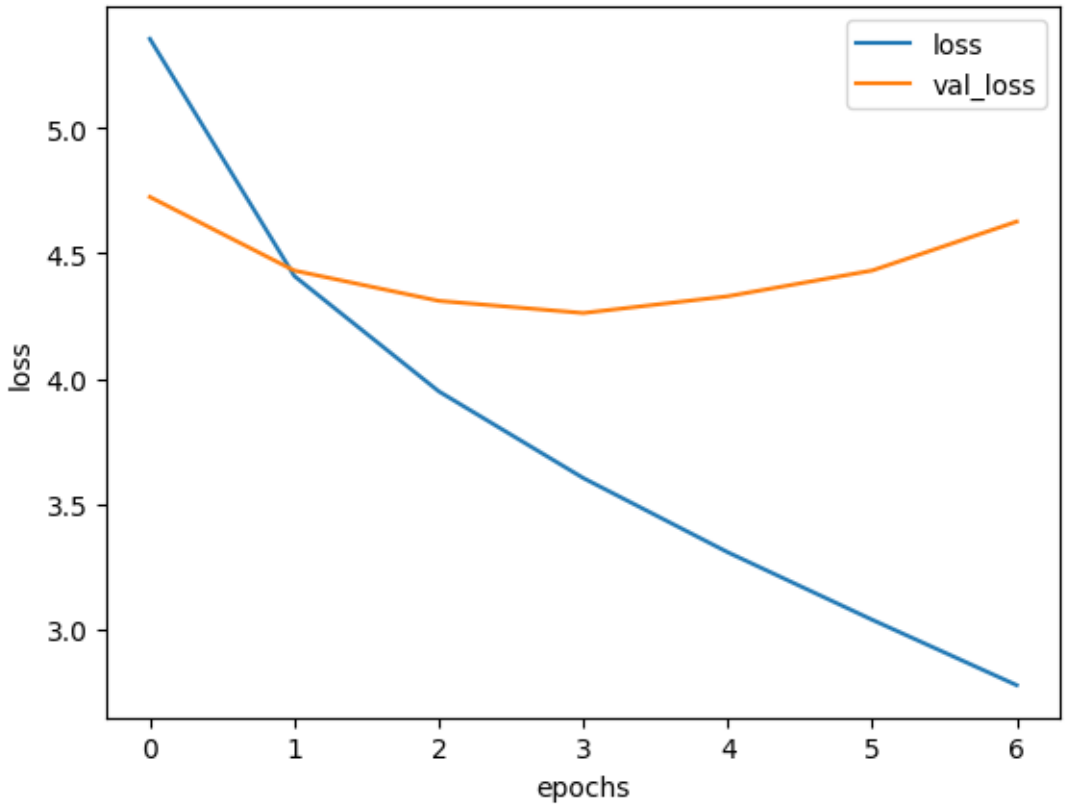
```
4476
Model: "model_10"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================
==================
 input_24 (InputLayer)       [(None, 30)]              0         []

 embedding_8 (Embedding)     (None, 30, 64)            286464
['input_24[0][0]']

 input_23 (InputLayer)       [(None, 1000)]            0         []

 CaptionFeature (GRU)        (None, 256)               247296
['embedding_8[0][0]']

 ImageFeature (Dense)        (None, 256)               256256
['input_23[0][0]']

 add_56 (Add)                (None, 256)               0
['CaptionFeature[0][0]',
'ImageFeature[0][0]']

 dense_23 (Dense)            (None, 256)               65792
['add_56[0][0]']

 dense_24 (Dense)            (None, 4476)              1150332
['dense_23[0][0]']

==================================================================================
```

```
==================
Total params: 2006140 (7.65 MB)
Trainable params: 2006140 (7.65 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________________________
_________________
None
```

[75]:
```
start = time.time()
#checkpoint_path = "training_1/cp.ckpt"
#checkpoint_dir = os.path.dirname(checkpoint_path)

# Create checkpoint callback
#cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
 #                                                 save_weights_only=True,
  #                                                verbose=2)

hist_vgg_gru = model_vgg_gru.fit([Ximage_train, Xtext_train], ytext_train,
                  epochs=7, verbose=2,
                  batch_size=64,
                  validation_data=([Ximage_val, Xtext_val], ytext_val))
                #callbacks = [cp_callback])
end = time.time()
print("TIME TOOK {:3.2f}MIN".format((end - start )/60))
```

```
Epoch 1/7
776/776 - 72s - loss: 5.3549 - val_loss: 4.7244 - 72s/epoch - 93ms/step
Epoch 2/7
776/776 - 58s - loss: 4.4089 - val_loss: 4.4307 - 58s/epoch - 74ms/step
Epoch 3/7
776/776 - 56s - loss: 3.9490 - val_loss: 4.3100 - 56s/epoch - 73ms/step
Epoch 4/7
776/776 - 57s - loss: 3.6033 - val_loss: 4.2615 - 57s/epoch - 73ms/step
Epoch 5/7
776/776 - 56s - loss: 3.3073 - val_loss: 4.3283 - 56s/epoch - 73ms/step
Epoch 6/7
776/776 - 56s - loss: 3.0381 - val_loss: 4.4311 - 56s/epoch - 72ms/step
Epoch 7/7
776/776 - 56s - loss: 2.7772 - val_loss: 4.6262 - 56s/epoch - 72ms/step
TIME TOOK 6.89MIN
```

[78]:
```
for label in ["loss","val_loss"]:
    plt.plot(hist_vgg_gru.history[label],label=label)
plt.legend()
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()
```



[79]:
```
index_word = dict([(index,word) for word, index in tokenizer.word_index.
 ↪items()])
def predict_caption(image):
    '''
    image.shape = (1,4462)
    '''

    in_text = 'startseq'

    for iword in range(maxlen):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence],maxlen)
        yhat = model_vgg_gru.predict([image,sequence],verbose=0)
        yhat = np.argmax(yhat)
        newword = index_word[yhat]
        in_text += " " + newword
        if newword == "endseq":
            break
    return(in_text)

npic = 5
```

```
npix = 224
target_size = (npix,npix,3)

count = 1
fig = plt.figure(figsize=(10,20))
for jpgfnm, image_feature in zip(fnm_test[:npic],di_test[:npic]):
    ## images
    filename = dir_Flickr_jpg + '/' + jpgfnm
    image_load = load_img(filename, target_size=target_size)
    ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
    ax.imshow(image_load)
    count += 1

    ## captions
    caption = predict_caption(image_feature.reshape(1,len(image_feature)))
    ax = fig.add_subplot(npic,2,count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
    ax.set_ylim(0,1)
    ax.text(0,0.5,caption,fontsize=20)
    count += 1

plt.show()
```

startseq boy in red shirt and black jacket is walking down the street endseq

startseq black and white dog is running through the grass endseq

startseq man in red shirt is surfing endseq

startseq black and white dog is walking on the grass endseq

startseq man in red shirt and black hat is holding her head endseq

[80]:
```
from nltk.translate.bleu_score import sentence_bleu
index_word = dict([(index,word) for word, index in tokenizer.word_index.
 ↪items()])


nkeep = 5
pred_good, pred_bad, bleus = [], [], []
count = 0
for jpgfnm, image_feature, tokenized_text in zip(fnm_test,di_test,dt_test):
    count += 1
```

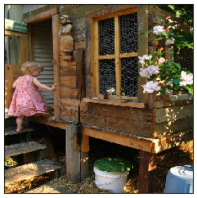
```
    if count % 200 == 0:
        print("  {:4.2f}% is done..".format(100*count/float(len(fnm_test))))

    caption_true = [ index_word[i] for i in tokenized_text ]
    caption_true = caption_true[1:-1] ## remove startreg, and endreg
    ## captions
    caption = predict_caption(image_feature.reshape(1,len(image_feature)))
    caption = caption.split()
    caption = caption[1:-1]## remove startreg, and endreg

    bleu = sentence_bleu([caption_true],caption)
    bleus.append(bleu)
    if bleu > 0.7 and len(pred_good) < nkeep:
        pred_good.append((bleu,jpgfnm,caption_true,caption))
    elif bleu < 0.3 and len(pred_bad) < nkeep:
        pred_bad.append((bleu,jpgfnm,caption_true,caption))
```

```
12.36% is done..
24.72% is done..
37.08% is done..
49.44% is done..
61.80% is done..
74.17% is done..
86.53% is done..
98.89% is done..
```

[81]:
```
def plot_images(pred_bad):
    def create_str(caption_true):
        strue = ""
        for s in caption_true:
            strue += " " + s
        return(strue)
    npix = 224
    target_size = (npix,npix,3)
    count = 1
    fig = plt.figure(figsize=(10,20))
    npic = len(pred_bad)
    for pb in pred_bad:
        bleu,jpgfnm,caption_true,caption = pb
        ## images
        filename = dir_Flickr_jpg + '/' + jpgfnm
        image_load = load_img(filename, target_size=target_size)
        ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
        ax.imshow(image_load)
        count += 1

        caption_true = create_str(caption_true)
```

```
        caption = create_str(caption)

        ax = fig.add_subplot(npic,2,count)
        plt.axis('off')
        ax.plot()
        ax.set_xlim(0,1)
        ax.set_ylim(0,1)
        ax.text(0,0.7,"true:" + caption_true,fontsize=20)
        ax.text(0,0.4,"pred:" + caption,fontsize=20)
        ax.text(0,0.1,"BLEU: {}".format(bleu),fontsize=20)
        count += 1
    plt.show()

print("Bad Caption")
plot_images(pred_bad)
print("Good Caption")
plot_images(pred_good)
```

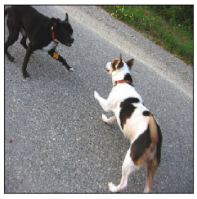
```
Bad Caption
```



true: child in pink dress is climbing up set of stairs in an entry way

pred: boy in red shirt and black jacket is walking down the street

BLEU: 9.853445011990208e-232



true: black dog and spotted dog are fighting

pred: black and white dog is running through the grass

BLEU: 1.384292958842266e-231

true: little girl covered in paint sits in front of painted rainbow with her hands in bowl

pred: man in red shirt is surfing

BLEU: 2.1986007635391227e-232

true: man lays on bench while his dog sits by him

pred: black and white dog is walking on the grass

BLEU: 1.1193096620723278e-231

true: man in an orange hat starring at something

pred: man in red shirt and black hat is holding her head

BLEU: 6.061838450024688e-155

Good Caption

true: black dog is running in the grass

pred: brown dog is running in the grass

BLEU: 0.8091067115702212

3 Changer l'architecture CNN utilisée dans le tutoriel par l' architectures VGGNet16 pré-entraîné sur ImageNet, et ré-entraîner le modèle avec LSTM.

[82]:
```
from keras import layers
from keras import models

print(vocab_size)

## Image feature
dim_embedding = 64

input_image = layers.Input(shape=(Ximage_train.shape[1],))
fimage = layers.Dense(256, activation='relu', name="ImageFeature")(input_image)

## Sequence model
input_txt = layers.Input(shape=(maxlen,))
ftxt = layers.Embedding(vocab_size, dim_embedding, mask_zero=True)(input_txt)
ftxt = layers.GRU(256, name="CaptionFeature")(ftxt)  # Change LSTM to GRU

## Combined model for decoder
decoder = layers.add([ftxt, fimage])
decoder = layers.Dense(256, activation='relu')(decoder)
output = layers.Dense(vocab_size, activation='softmax')(decoder)

model_vgg_lstm = models.Model(inputs=[input_image, input_txt], outputs=output)
model_vgg_lstm.compile(loss='categorical_crossentropy', optimizer='adam')

print(model_vgg_lstm.summary())
```

```
4476
Model: "model_11"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================
==================
 input_26 (InputLayer)       [(None, 30)]              0         []

 embedding_9 (Embedding)     (None, 30, 64)            286464
['input_26[0][0]']

 input_25 (InputLayer)       [(None, 1000)]            0         []
```

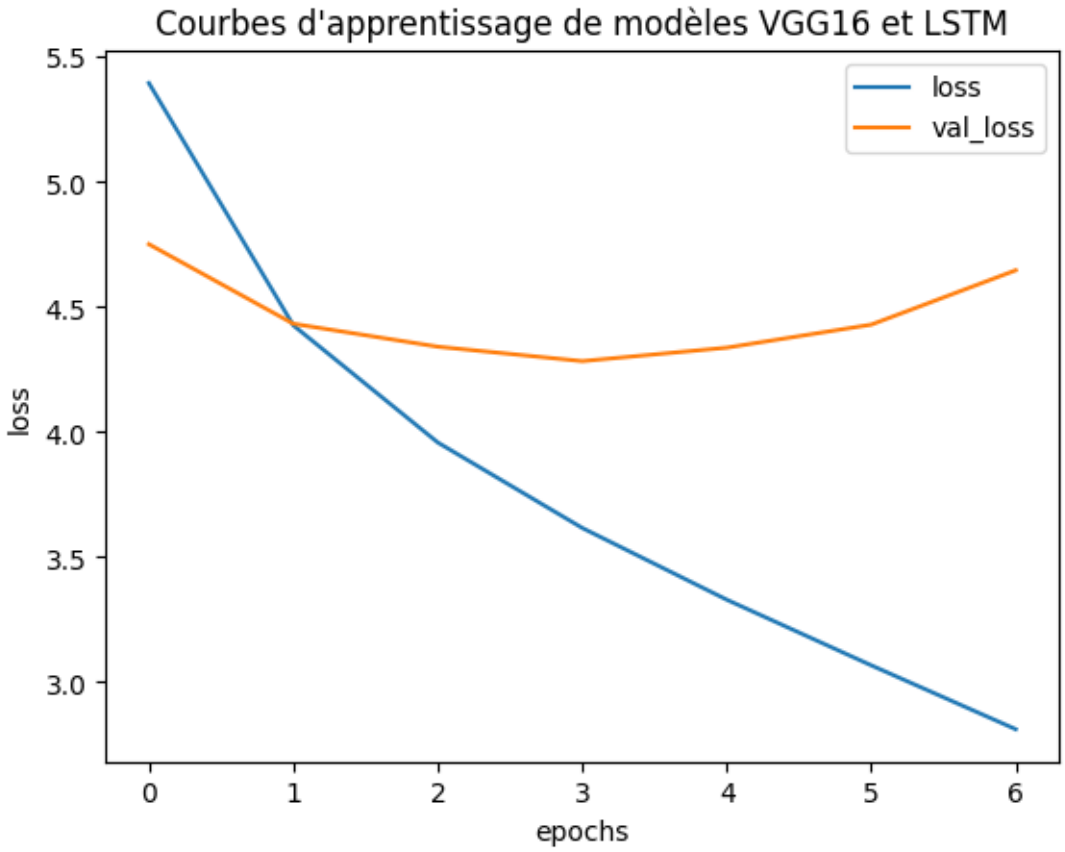
```
 CaptionFeature (GRU)        (None, 256)                 247296
['embedding_9[0][0]']

 ImageFeature (Dense)        (None, 256)                 256256
['input_25[0][0]']

 add_57 (Add)                (None, 256)                 0
['CaptionFeature[0][0]',
'ImageFeature[0][0]']

 dense_25 (Dense)            (None, 256)                 65792
['add_57[0][0]']

 dense_26 (Dense)            (None, 4476)                1150332
['dense_25[0][0]']

==================================================================================
==================
Total params: 2006140 (7.65 MB)
Trainable params: 2006140 (7.65 MB)
Non-trainable params: 0 (0.00 Byte)
----------------------------------------------------------------------------------
------------------
None
```

[83]:
```
start = time.time()
#checkpoint_path = "training_1/cp.ckpt"
#checkpoint_dir = os.path.dirname(checkpoint_path)

# Create checkpoint callback
#cp_callback = tf.keras.callbacks.ModelCheckpoint(checkpoint_path,
 #                                                 save_weights_only=True,
  #                                                verbose=2)

hist_vgg_lstm = model_vgg_lstm.fit([Ximage_train, Xtext_train], ytext_train,
                  epochs=7, verbose=2,
                  batch_size=64,
                  validation_data=([Ximage_val, Xtext_val], ytext_val))
                #callbacks = [cp_callback])
end = time.time()
print("TIME TOOK {:3.2f}MIN".format((end - start )/60))
```

```
Epoch 1/7
776/776 - 69s - loss: 5.3949 - val_loss: 4.7508 - 69s/epoch - 88ms/step
Epoch 2/7
776/776 - 56s - loss: 4.4264 - val_loss: 4.4328 - 56s/epoch - 72ms/step
Epoch 3/7
```

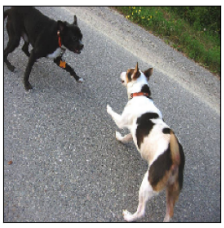
```
776/776 - 55s - loss: 3.9582 - val_loss: 4.3412 - 55s/epoch - 71ms/step
Epoch 4/7
776/776 - 55s - loss: 3.6166 - val_loss: 4.2835 - 55s/epoch - 71ms/step
Epoch 5/7
776/776 - 55s - loss: 3.3299 - val_loss: 4.3373 - 55s/epoch - 71ms/step
Epoch 6/7
776/776 - 55s - loss: 3.0676 - val_loss: 4.4294 - 55s/epoch - 71ms/step
Epoch 7/7
776/776 - 55s - loss: 2.8122 - val_loss: 4.6466 - 55s/epoch - 70ms/step
TIME TOOK 6.70MIN
```

[94]:
```
for label in ["loss","val_loss"]:
    plt.plot(hist_vgg_lstm.history[label],label=label)
plt.legend()
plt.title('Courbes d\'apprentissage de modèles VGG16 et LSTM')
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()
```



[88]:
```
index_word = dict([(index,word) for word, index in tokenizer.word_index.
 ↪items()])
def predict_caption(image):
    '''
    image.shape = (1,4462)
    '''

    in_text = 'startseq'

    for iword in range(maxlen):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence],maxlen)
        yhat = model_vgg_lstm.predict([image,sequence],verbose=0)
        yhat = np.argmax(yhat)
        newword = index_word[yhat]
        in_text += " " + newword
        if newword == "endseq":
            break
    return(in_text)

npic = 5
npix = 224
target_size = (npix,npix,3)

count = 1
fig = plt.figure(figsize=(10,20))
for jpgfnm, image_feature in zip(fnm_test[:npic],di_test[:npic]):
    ## images
    filename = dir_Flickr_jpg + '/' + jpgfnm
    image_load = load_img(filename, target_size=target_size)
    ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
    ax.imshow(image_load)
    count += 1

    ## captions
    caption = predict_caption(image_feature.reshape(1,len(image_feature)))
    ax = fig.add_subplot(npic,2,count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
    ax.set_ylim(0,1)
    ax.text(0,0.5,caption,fontsize=20)
    count += 1

plt.show()
```

startseq boy in blue shirt and blue shirt is jumping off of stairs endseq



startseq black and white dog is running in the snow endseq

startseq man in blue shirt is jumping off of the ground endseq

startseq black and white dog is running in the grass endseq

startseq man and woman are sitting in front of building endseq

[89]:
```
from nltk.translate.bleu_score import sentence_bleu
index_word = dict([(index,word) for word, index in tokenizer.word_index.
 ↪items()])


nkeep = 5
pred_good, pred_bad, bleus = [], [], []
count = 0
for jpgfnm, image_feature, tokenized_text in zip(fnm_test,di_test,dt_test):
```

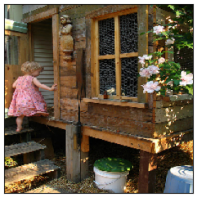
```
    count += 1
    if count % 200 == 0:
        print("  {:4.2f}% is done..".format(100*count/float(len(fnm_test))))

    caption_true = [ index_word[i] for i in tokenized_text ]
    caption_true = caption_true[1:-1] ## remove startreg, and endreg
    ## captions
    caption = predict_caption(image_feature.reshape(1,len(image_feature)))
    caption = caption.split()
    caption = caption[1:-1]## remove startreg, and endreg

    bleu = sentence_bleu([caption_true],caption)
    bleus.append(bleu)
    if bleu > 0.7 and len(pred_good) < nkeep:
        pred_good.append((bleu,jpgfnm,caption_true,caption))
    elif bleu < 0.3 and len(pred_bad) < nkeep:
        pred_bad.append((bleu,jpgfnm,caption_true,caption))
```

```
12.36% is done..
24.72% is done..
37.08% is done..
49.44% is done..
61.80% is done..
74.17% is done..
86.53% is done..
98.89% is done..
```

[90]:
```
def plot_images(pred_bad):
    def create_str(caption_true):
        strue = ""
        for s in caption_true:
            strue += " " + s
        return(strue)
    npix = 224
    target_size = (npix,npix,3)
    count = 1
    fig = plt.figure(figsize=(10,20))
    npic = len(pred_bad)
    for pb in pred_bad:
        bleu,jpgfnm,caption_true,caption = pb
        ## images
        filename = dir_Flickr_jpg + '/' + jpgfnm
        image_load = load_img(filename, target_size=target_size)
        ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
        ax.imshow(image_load)
        count += 1
```

```
        caption_true = create_str(caption_true)
        caption = create_str(caption)

        ax = fig.add_subplot(npic,2,count)
        plt.axis('off')
        ax.plot()
        ax.set_xlim(0,1)
        ax.set_ylim(0,1)
        ax.text(0,0.7,"true:" + caption_true,fontsize=20)
        ax.text(0,0.4,"pred:" + caption,fontsize=20)
        ax.text(0,0.1,"BLEU: {}".format(bleu),fontsize=20)
        count += 1
    plt.show()

print("Bad Caption")
plot_images(pred_bad)
print("Good Caption")
plot_images(pred_good)
```

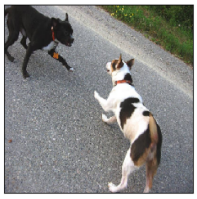
```
Bad Caption
```



true: child in pink dress is climbing up set of stairs in an entry way

pred: boy in blue shirt and blue shirt is jumping off of stairs

BLEU: 5.268188634815163e-155



true: black dog and spotted dog are fighting

pred: black and white dog is running in the snow

BLEU: 1.384292958842266e-231

true: little girl covered in paint sits in front of painted rainbow with her hands in bowl

pred: man in blue shirt is jumping off of the ground

BLEU: 6.686350417856737e-232

true: man lays on bench while his dog sits by him

pred: black and white dog is running in the grass

BLEU: 9.412234823955334e-232

true: man in an orange hat starring at something

pred: man and woman are sitting in front of building

BLEU: 1.2508498911928379e-231

Good Caption

true: black dog is running in the water

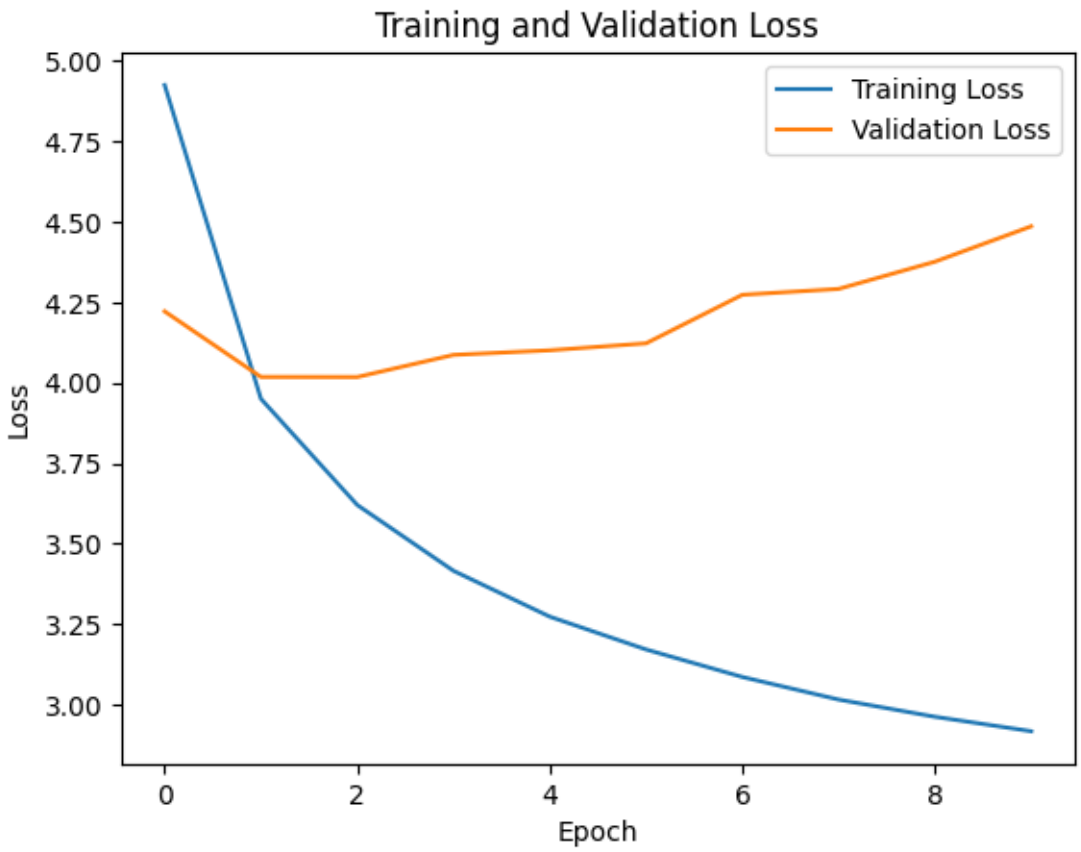
pred: black dog is running in the snow

BLEU: 0.8091067115702212

4 courbes d'apprentissage des modèles

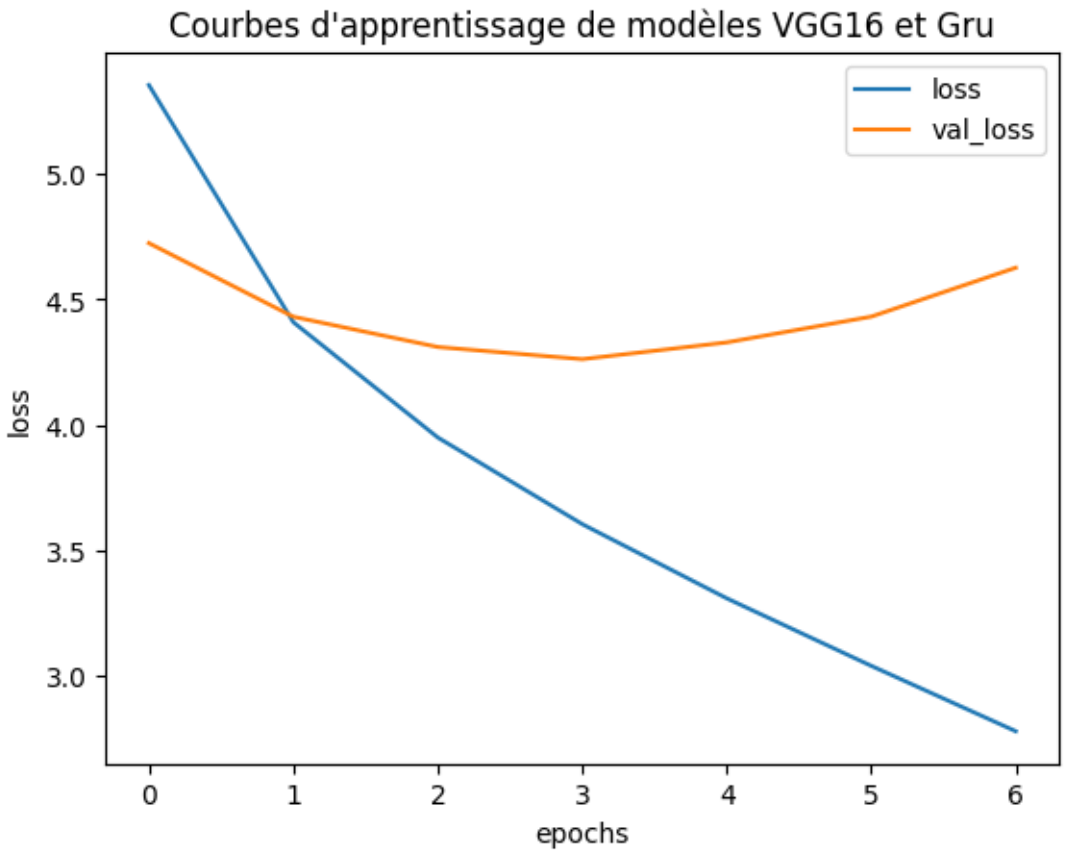
4.1 1. Tuto model

[103]:
```
import matplotlib.pyplot as plt

# Function to plot training and validation loss
def plot_loss(history,history_val):
  plt.plot(history, label='Training Loss')
  plt.plot(history_val, label='Validation Loss')
  plt.title('Training and Validation Loss')
  plt.xlabel('Epoch')
  plt.ylabel('Loss')
  plt.legend()
  plt.show()

losses = []
val_losses = []

# Plot the loss and accuracy for each epoch
for i, history in enumerate(histories):

  losses.append(histories[i].history['loss'])
  val_losses.append(histories [i].history['val_loss'])

plot_loss(losses, val_losses)
```



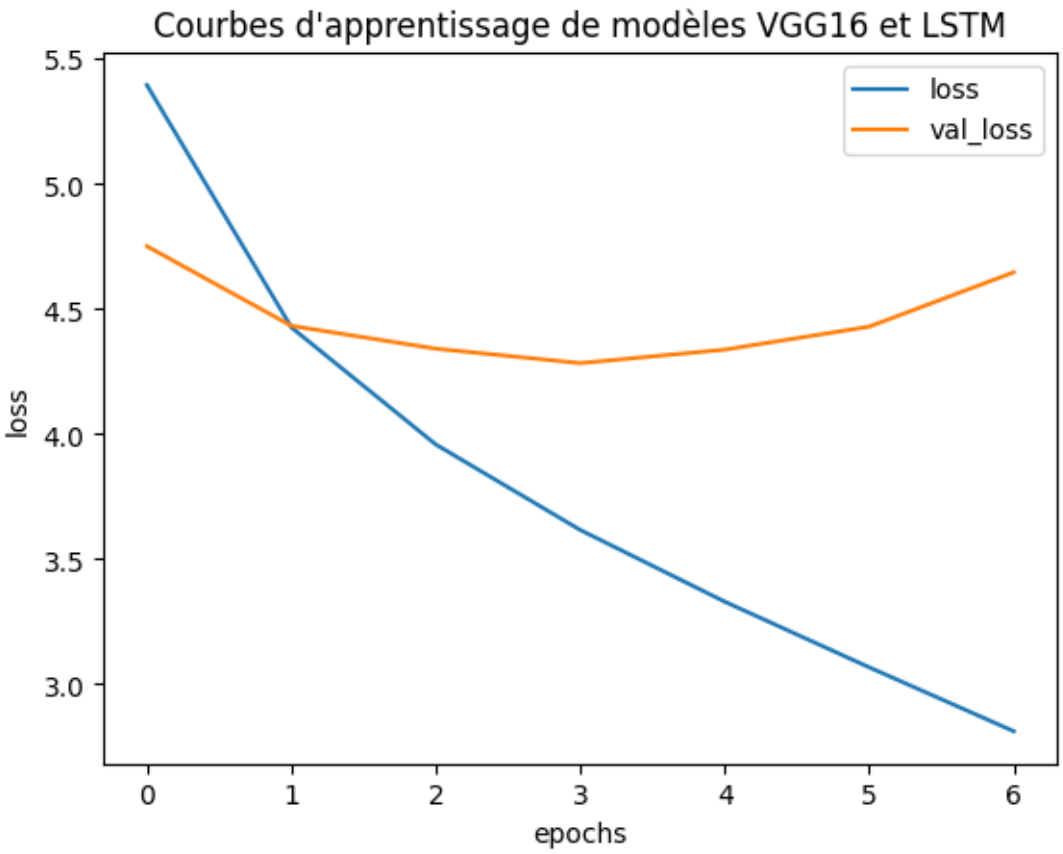
4.2 2. VGG16-GRU model

[105]:
```
for label in ["loss","val_loss"]:
    plt.plot(hist_vgg_gru.history[label],label=label)
plt.legend()
plt.title('Courbes d\'apprentissage de modèles VGG16 et Gru')
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()
```



4.3 3. VGG16-GRU model

[104]:
```
for label in ["loss","val_loss"]:
    plt.plot(hist_vgg_lstm.history[label],label=label)
plt.legend()
plt.title('Courbes d\'apprentissage de modèles VGG16 et LSTM')
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()
```


Courbes d'apprentissage de modèles VGG16 et LSTM
loss
val_loss
5.5
5.0
4.5
4.0
3.5
3.0
loss
0
1
2
3
4
5
6
epochs