Stefania Zanetta S4463362

Computational Vision: Image Captioning Project

Abstract

Image captioning is a challenging task in computer vision and natural language processing, aiming to generate coherent and contextually relevant descriptions for images. In this study, we explore and compare the effectiveness of three distinct image captioning models: the Convolutional Neural Network with Recurrent Neural Network (CNN-RNN), a Pre-trained model with Recurrent Neural Network (VGG16-RNN) and a Trasformer-based model.

We conducted experiments on the Flicker8k dataset with their corresponding captions to evaluate the models caption generation capabilities.

Our results reveal valuable insights into the strengths and weaknesses of each model. We further evaluate these models on metrics such as BLEU to provide quantitative comparisons. Our findings contribute to a better understanding of the trade-offs between different image captioning architectures and offer insights into their potential applications.

Introduction

Image captioning is the task of describing the visual content of an image in natural language, employing a visual understanding system and a language model capable of generating meaningful and syntactically correct sentences.

In its standard configuration, the task is an image-to-sequence problem whose inputs are pixels. These inputs are encoded as one or multiple feature vectors in the visual encoding step, which prepares the input for a second generative step, called the language model. This produces a sequence of words or subwords decoded according to a given vocabulary.

Image captioning task is currently based on the usage of deep learning-based generative models. In these few years, the research community has improved model design considerably: from the first deep learning-based proposals adopting Recurrent Neural Networks (RNNs) fed with global image descriptors, methods have been enriched with attentive approaches and reinforcement learning up to the breakthroughs of Transformers and self-attention and single-stream BERT-like approaches.

Several domain-specific proposals and variants of the task have also been investigated to accommodate for different user needs and descriptions styles. According to “Framing image description as a ranking task: Data, models and evaluation metrics” and “Vision to Language: Methods, Metrics and Datasets” papers: image captions can be perceptual, when focusing on low-level visual attributes; non-visual, when reporting implicit and contextual information; conceptual, when describing the actual visual content.

Dataset

The Dataset we used is Flicker8k, as the name suggest it contains 8000 images that covers a wide range of subjects and scenes from the Flickr website.

Each of the 8,000 images in the dataset is paired with five different textual captions. These captions describe the content of the image in natural language. The captions are typically short and concise, providing brief but informative descriptions of what is happening in the image.

Flicker also provides larger datasats, such as Flickr30k, since training models on larger datasets requires more computational power and time, we decided to use the smaller dataset, but the notebook provided should work also on the larger dataset.

Image captioning models

As explained in the "From Show to Tell: A Survey on Deep Learning-based Image Captioning" paper the image captioning task can be divided in two main components:

- Visual encoding, meaning the process of extracting meaningful features or representations from an input image.
- Language models, responsible for generating descriptive text based on those visual features.

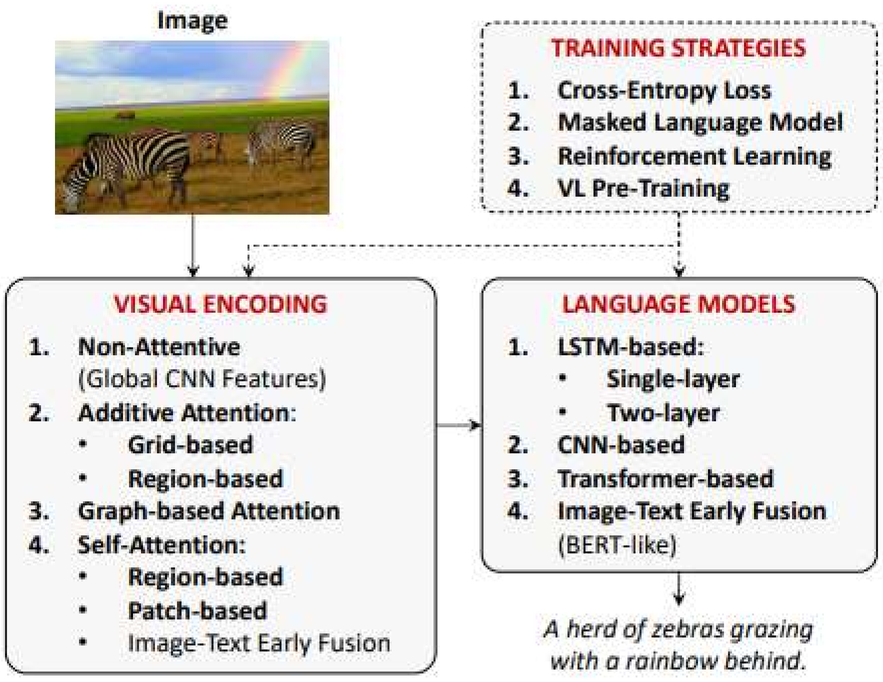


Figure 1

As we can see from Fig. 1, the current approaches for visual encoding can be classified as belonging to four main categories: 1. non-attentive methods based on global CNN features; 2. additive attentive methods that embed the visual content using either grids or regions; 3. graph-based methods adding visual relationships between visual regions; and 4. self-attentive methods that employ Transformer-based paradigms, either by using region-based, patch-based, or image-text early fusion solutions.

In our experiment we will only see non-attentive methods (CNN and a pretrained model VGG16) and a self-attentive method (Swin Trasformer).

While the main language modelling strategies applied to image captioning can be categorized as: 1. LSTM-based approaches, which can be either single-layer or two-layer; 2. CNN-based methods that constitute a first attempt in surpassing the fully recurrent paradigm; 3. Transformer-based fully-attentive approaches; 4. Image-text early-fusion (BERT-like) strategies that directly connect the visual and textual inputs.

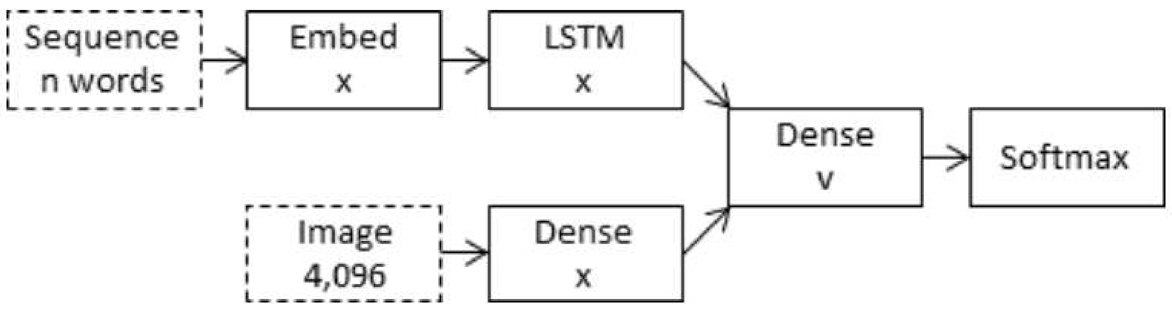
In our experiment we will only see LSTM-based approaches (single layer LSTM) and Transformer-based (GPT-2).

First Model (VGG16-RNN)

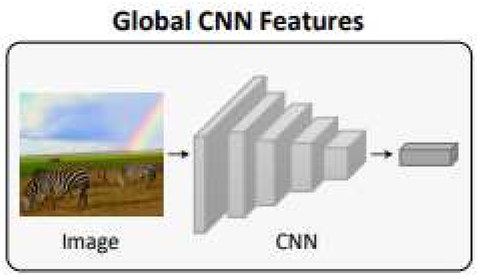
As previously mentioned, the first model we present has: a non-attentive method (as a visual encoder) and a single layer LSTM-based approach (as a language model).

First presented in the paper "Where to put the Image in an Image Caption Generator" and later expanded in "What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?" this is the merge architecture. In merge architectures involve combining image and linguistic information, but the combination is delayed until a vectorized representation of the caption's prefix has been obtained.

(a) The merge architecture.



This separates the concern of modelling the image input, the text input and the combining and interpretation of the encoded inputs. It is common to use a pre-trained model for encoding the image, but similarly, this architecture also permits a pre-trained language model to be used to encode the caption text input. In our case we will use a pre-trained model for encoding the image (VGG16) but not for the language model.



(a)

VGG16 is a convolutional neural network (CNN) architecture that was developed by the Visual Geometry Group (VGG) at the University of Oxford. VGG16 is one of several variants of the VGG network architecture, with "16" referring to the total number of weight layers in the network. The VGG16 model was originally trained on a large dataset known as ImageNet.

The language model expects input sequences with a pre-defined length which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by a single LSTM layer with 256 memory units.

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

There are multiple ways to combine the two encoded inputs, such as concatenation, multiplication, and as we used in our experiment addition as it was shown by Marc Tanti, et al. (in the paper “What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?”) to work better.

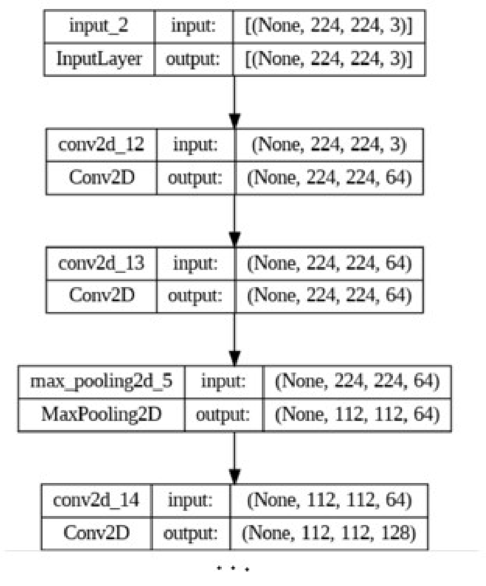
```python
# decoder model
decoder1 = tf.keras.layers.add([fe2, se3])
decoder2 = tf.keras.layers.Dense(256, activation='relu')(decoder1)
outputs = tf.keras.layers.Dense(vocab_size, activation='softmax')(decoder2)
```

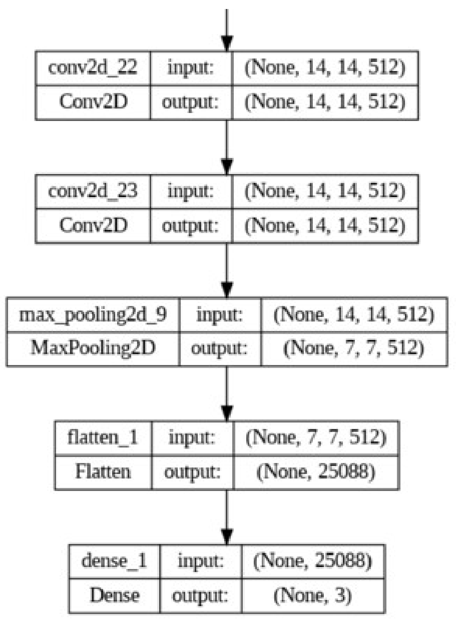
Second Model (CNN-RNN)

Just like the first model, the second model presents: a non-attentive method (as a visual encoder) and a single layer LSTM-based approach (as a language model).

The second model also presents a merge architecture, with the only difference being the visual encoder.

In this case we decided not to use a pre-trained model, but a convolutional neural network built from scratch.
For the CNN we decided to take inspiration from the structure of the VGG16 and built the following model:


input_2 | input: | [(None, 224, 224, 3)]
InputLayer | output: | [(None, 224, 224, 3)]
conv2d_12 | input: | (None, 224, 224, 3)
Conv2D | output: | (None, 224, 224, 64)
conv2d_13 | input: | (None, 224, 224, 64)
Conv2D | output: | (None, 224, 224, 64)
max_pooling2d_5 | input: | (None, 224, 224, 64)
MaxPooling2D | output: | (None, 112, 112, 64)
conv2d_14 | input: | (None, 112, 112, 64)
Conv2D | output: | (None, 112, 112, 128)
. . .


conv2d_22 | input: | (None, 14, 14, 512)
Conv2D | output: | (None, 14, 14, 512)
conv2d_23 | input: | (None, 14, 14, 512)
Conv2D | output: | (None, 14, 14, 512)
max_pooling2d_9 | input: | (None, 14, 14, 512)
MaxPooling2D | output: | (None, 7, 7, 512)
flatten_1 | input: | (None, 7, 7, 512)
Flatten | output: | (None, 25088)
dense_1 | input: | (None, 25088)
Dense | output: | (None, 3)

To avoid having to reshape once again our images we’ll keep our input like the first model (224, 224, 3). Our model has a total of 12 convolutional layers with a stride of 1. All pairs of convolutional layers are followed by max-pooling layers with a stride of 2, in total we have 5 max-pooling layers (with the one last being after 4 convolutional layers instead of 2). After the last max-pooling layer, there's a flatten layer that converts the 3D feature maps into a 1D vector followed by a fully connected dense layer with 3 units.

The language model is the same as the first model, an Embedding layer that uses a mask followed by a single LSTM layer with 256 memory units. Also, the decoder model that combines the two inputs is the same. The only change notable is that instead of max_length, we put 3 as our input shape of our image features since the shape of the output our CNN.

Third Model (Transformer)

The last model we present has: a self-attentive method (as a visual encoder) and Transformer-based approach (as a language model).

Transformers have been increasingly used in image captioning tasks due to their effectiveness in handling sequential data, which is essential for generating captions. While Transformers were initially developed for natural language processing (NLP) tasks, they have been adapted and extended for multimodal tasks like image captioning. Here's how Transformers are used in image captioning:

- Encoder: The image is initially processed to extract visual features then are passed through an encoder module. The encoder's role is to encode the visual information into a format that the decoder can work with.
- Decoder: The decoder is a transformer-based model that generates the caption text. It takes as input the encoded visual features from the encoder, as well as a special start token indicating the beginning of the caption. The decoder generates words one by one, attending to both the visual features and the previously generated words to predict the next word in the caption.

We will use Vision Encoder-Decoder architecture models, where the encoder is the Swin Trasformer, and the decoder is the language model GPT2.

First introduced in the paper titled "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows" by Ze Liu, et al., we will use, like previously anticipated, the Swin Trasformer as our encoder. The ResNet and ViT (Vision Transformer) architectures could have been also used but Swin Transformer has achieved state-of-the-art performance on several image classification benchmarks, surpassing previous models.

For our decoder we use Generative Pre-trained Transformer 2 (GPT-2). Developed by OpenAI, it is a neural network-based model, utilizes self-attention mechanisms to capture long-range dependencies in the input text. GPT-2 has been trained on a massive corpus of text data, collected from the internet. It has approximately 1.5 billion parameters, which allows it to capture and learn intricate language patterns and structures.

To load the pre-trained weights of both models and combine them together in a single model, we use the VisionEncoderDecoderModel class from the trasformers library and the from_encoder_decoder_pretrained() method that expects the name of both models.

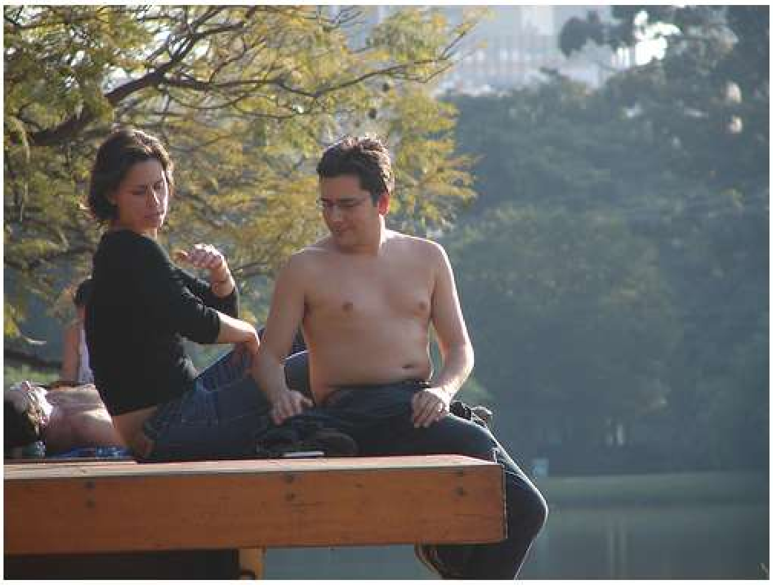
```
# the encoder model that process the image and return the image features
encoder_model = "microsoft/swin-base-patch4-window7-224-in22k"
# the decoder model that process the image features and generate the caption
decoder_model = "gpt2"
# load the model
model3 = VisionEncoderDecoderModel.from_encoder_decoder_pretrained(
    encoder_model, decoder_model
).to(device)
```

Results

In this section we present the results of the models previously presented. The first two models were trained for 20 epochs, whereas the third Transformer-based model was trained over a span of 2 epochs.

Generating captions from an image we observe the following results:

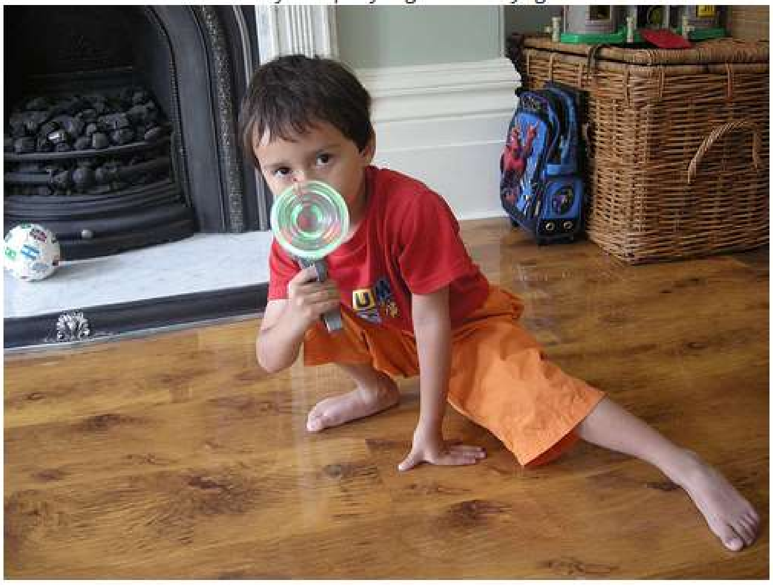
```
First model:  startseq two children are playing on gaurd rail endseq
Second model:  startseq two boys are playing on gaurd rail endseq
Third model:  man and woman sit on bench
```



As we can see the third model gives the better results. In contrast, the first two models, while generating somewhat similar captions, don’t describe well the scene presented in the image.

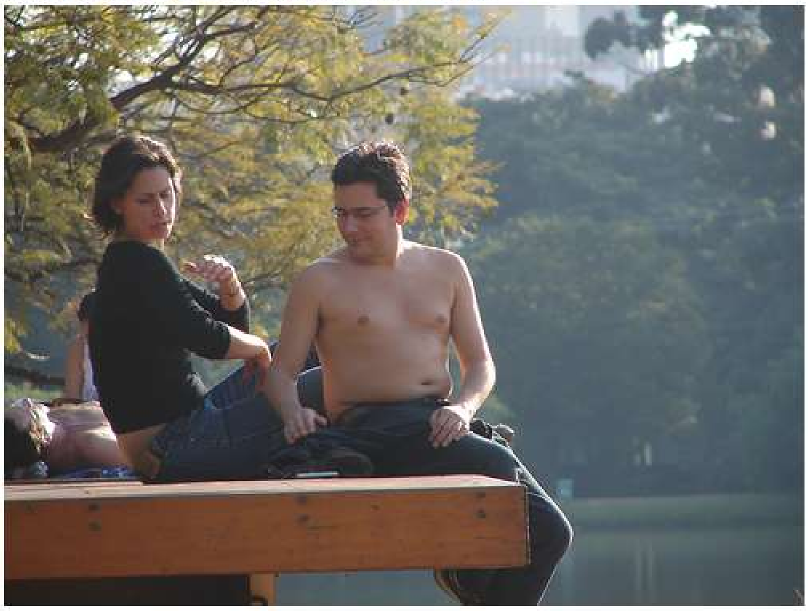
Upon generating captions from other images, we can notice a pattern:

```
First model:  startseq two children are playing on gaurd rail endseq
Second model:  startseq two boys are playing on gaurd rail endseq
Third model:  little boy is playing with toy gun
```

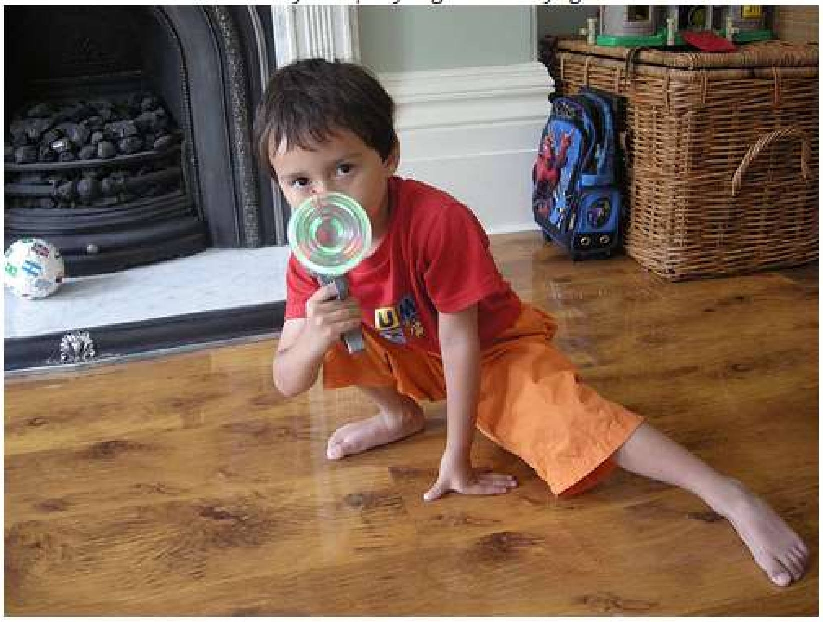


The first two models frequently generate identical captions for distinct images. If we force them to generate new captions with different words, we get the following results:

```
First model:  startseq young girl is jumping on bed near the ocean and she holds bottle endseq
Second model:  startseq man is climbing rock cliff with the sun beams endseq
Third model:  man and woman sit on bench
```



```
First model:  startseq young girl is blowing bubbles under an enclosed store range with person
Second model:  startseq man is climbing rock cliff with the sun beams endseq
Third model:  little boy is playing with toy gun
```



The first model, while not entirely successful, does provide some relevant descriptive elements (e.g., the word "ocean"). Meanwhile the second model seem to get stuck once again in predicting the same sentence for different images.

This discrepancy can be attributed to the difference in number of features extracted from the image, since the VGG16 model (4096 features) gives in output more feature than our CNN (3 features). This gives captions in input to the LSTM more weight in the merge architecture. Since the work of the LSTM is to predict the next word given an input, keeping in mind that the start of a sequence is always the same “startseq”, and the image input is just composed of 3 features, the output seems to converge in the on repeated predictions.

The same results are reflected in the BLEU metrics scores. The BLEU metric scores a caption on a scale of 0 to 1, the closer to 1 the more overlap there is with their original captions and thus, the better output is deemed to be.

The first two models have similar results with a score of 0.3, VGG16 model being closer to 0.38 while the CNN being closer to 0.36. In contrast, the transformer-based model has better results ranging from 0.6 to 0.8.

While being the better model overall, it is worth noting that that the transformer-based model, takes more time to train compared to the first two simpler models. For 20 epochs the first two model require approximately two hours, while the third model an hour and a half for just 2 epochs.

Conclusion

The results obtained from generating captions for images unveiled interesting results.

As expected, the third model, based on the Transformer architecture, consistently outperformed the simpler two models. Notably, it demonstrated a higher level of accuracy in describing the images. While initially designed for natural language processing, transformer-based architectures like Swin Transformer gained popularity for image classification and even outperformed CNN-based models. Furthermore, the incorporation of the GPT-2 language model, known to be highly effective in various NLP tasks, within a vision encoder-decoder architecture renders the third model's superiority a foreseeable outcome.

In contrast, the other two merge architecture models produced worse results, indicating a limitation to provide contextually relevant descriptions. However, it is worth noting that these models could potentially be improved by increasing their training dataset with additional captions and by limiting the themes of the provided images.

Of the three models, the merge architecture model with a CNN not pretrained yielded the worst results. Nevertheless, this model can surly be improved by incrementing the number of features extracted from the pictures, which are an important aspect, if not the most important, for this kind of models.