Deep Learning Final Project:

Implementation and analysis of an Encoder-Decoder Image Captioning architecture

---

Marcel Fernández Serrano
Iván Hernández Gómez
Alejandro Pastor Rubio

Index

1. Description of the problem

Image captioning is the process of generating a textual description for an image. It involves using machine learning and computer vision techniques to analyze the visual content of an image and produce a coherent, relevant, and often contextually accurate description. We can describe the main steps to procedure with this project:

1. Image Analysis: The system first analyzes the image to identify various elements and objects within it. This typically involves using convolutional neural networks (CNNs) to extract features from the image.

2. Feature Extraction: The visual features extracted from the image are converted into a format that can be processed by a language model. This is usually done by passing the features through a neural network to create a fixed-length vector representation of the image.

3. Sequence Generation: The vector representation of the image is then used as input to a sequence-generating model, often a type of recurrent neural network (RNN) like a Long Short-Term Memory (LSTM) network. The RNN generates a sequence of words that form a sentence describing the image.

4. Language Modeling: The language model component ensures that the generated sequence of words is grammatically correct and contextually appropriate. This involves training on large datasets of images and their corresponding captions.

5. End-to-End Training: Modern image captioning systems are typically trained end-to-end, meaning that the image analysis, feature extraction, and sequence generation components are trained simultaneously to optimize the overall performance of the system.

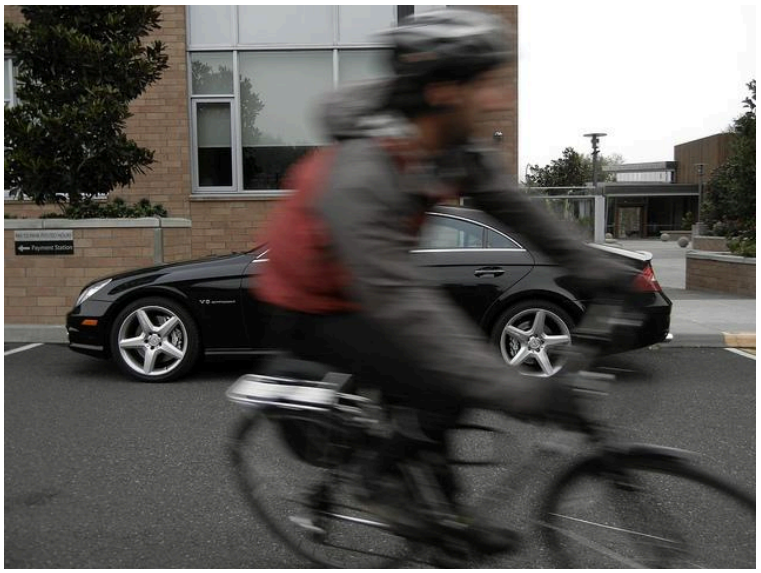
The overall goal of image captioning is to create a system that can automatically generate human-like descriptions of images, which can be useful for a variety of applications, including aiding visually impaired individuals, enhancing search engine capabilities, and improving image organization and retrieval systems.

2. Dataset: COCO

Due to the complexity of training an image captioning model, we required a dataset with a large number of image-caption pairs. Therefore, we used the COCO dataset, a large-scale resource for object detection, segmentation, and captioning, which contains over 80,000 training images, each with five human-generated captions. This is the overall structure of the dataset:

	Training	**Validation**	**Testing**
# Images	80,000	40,000	40,000
# Captions per image	5	5	0

Sample image and its five associated captions:



'A blurry bike rider zooms past a new Mercedes.'
'A blurry bicycle riders goes by a black car.'
'A man rides a bike past a black car in a parking lot.'
'A black car is near someone riding a bike.'
'A person on a bicycle is riding in front of a car.'

3. Architecture: Encoder - Decoder

So, now that we understand the general structure of this topic, we want to delve deeper. We have conducted extensive research to find various model architectures, and our objective is to analyze their structure, learn how they are implemented, assess their accuracy, test them with various examples, and compare their performance.

We have identified a common structure in various models: an encoder is used to read and extract the main features of the image, followed by a decoder to convert the feature vector into text. As described in several papers, it is not necessary to retrain the whole encoder, instead pretrained object classification models can be used, saving computational resources and providing a consistent starting point for the image captioning implementation. The output vector from this encoder serves as the input for the decoder.

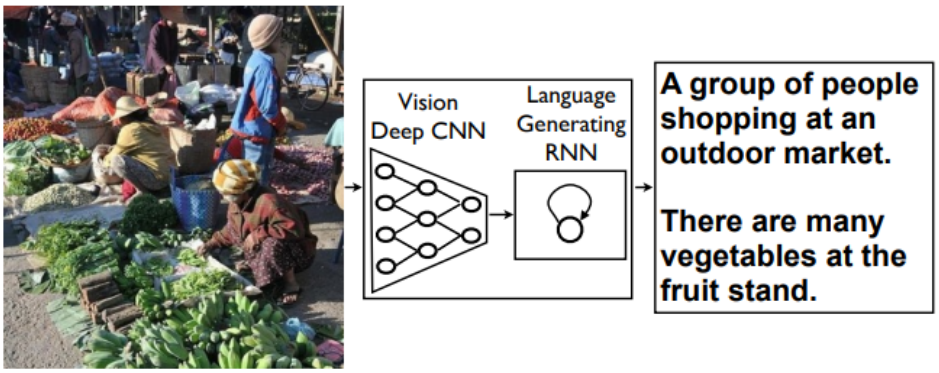
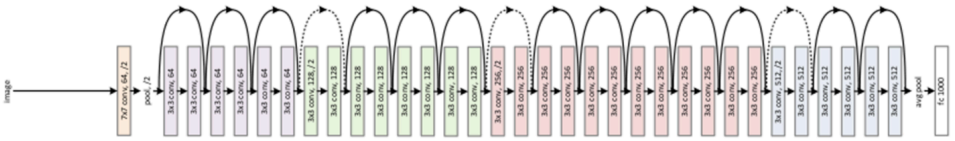


Image from the paper *Show and Tell: A Neural Image Caption Generator*

3.1. Encoder

The encoder of the image captioning model consists of a pretrained ResNet-152 CNN, trained for image classification purposes with the ImageNet dataset. This dataset has over 14 million images with more than 1000 different classes.



ResNet architecture

In order to adapt the encoder output to the image captioning task, we have retrained the last fully connected layer, setting the output size of the encoder to the embedding size of the words. We will delve further into this concept later.

Once we had defined the encoder architecture, we had to transform the input images in a way they could be fed into our CNN. To do so we applied two transformations: resize and normalization.

- **Image Resize:** The images are resized using Lanczos interpolation, so they meet the 224x224 ImageNet size.
- **Image Normalization:** The images are normalized using the mean and the standard deviation of the ImageNet dataset. These two measures are 3-dimensional tensors, each dimension corresponding to one channel of the RGB image.

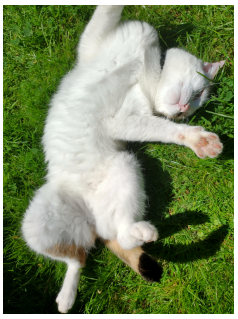
3.2. Decoder

The encoder of the image captioning model consists of a LSTM, which is in charge of generating raw logit vectors. These logit values are eventually converted to probabilities using a Softmax activation function. The final probability vectors contain the probabilities of all possible words in the vocabulary given the previous words in the sequence and the input image.

$$p(S_t|I, S_0, \ldots, S_{t-1})$$

Where S_i is the i-th word of the sequence and I represents the input image.

Example:

I =  $S_{0,\ldots,S_{t-1}}$ = <a>, <cat>, <laying>, <on>, <the>

Possible output of the next LSTM block (after a SoftMax activation):

[p("hello"|I, $S_{0,\ldots,S_{t-1}}$) = 0.01, p("grass"|I, $S_{0,\ldots,S_{t-1}}$) = 0.8, ... , p("flan"|I, $S_{0,\ldots,S_{t-1}}$) = 0.0003]

As happened previously, we need to apply some transformations to our captions during the training phase so they can be fed into our decoder model. These transformations consist of: word tokenization, word indexation, and embedding.

- **Word tokenization:** Each sentence is separated into words, each one representing a token. These tokens represent the unitary input for our LSTM block.
- **Word indexation:** Each word in the sentence is indexed into a positive integer. This one-to-one relationship is saved in a Vocabulary dictionary.
- **Embedding:** The indexed words are eventually passed through an embedder, which is trained with the rest of the model.

Additionally, we add an <<start>> and <<end>> token at the start and end of each sentence.

(Word tokenization): “This is an example” ⟶<<start>>, <this>, <is>, <an>, <example>,<<end>>

(Word indexation): <<start>>, <this>, <is>, <an>, <example>,<<end>> ⟶<0>, <7>, … , <40>

(Word embedding): <0>, <7>, … , <40>⟶ <[0’7, 0’1, …., 0.3]>, … , <[1’3, 0’5, …, 2’1]>

Now that we have defined how to transform the captions so they can be fed into our model, we have to define the way in which they will be fed to the LSTM alongside the image features during training, and how the LSTM will behave during inference when no caption is provided. Note that as of now, what we have is one feature vector which is the output of the encoder, and N+2 embedding vectors coming from the reference N-word caption. Notice that as mentioned earlier in the report, the image feature vector is forced to have the same dimensionality as the embedding vectors by modifying the last fully connected layer of the CNN.

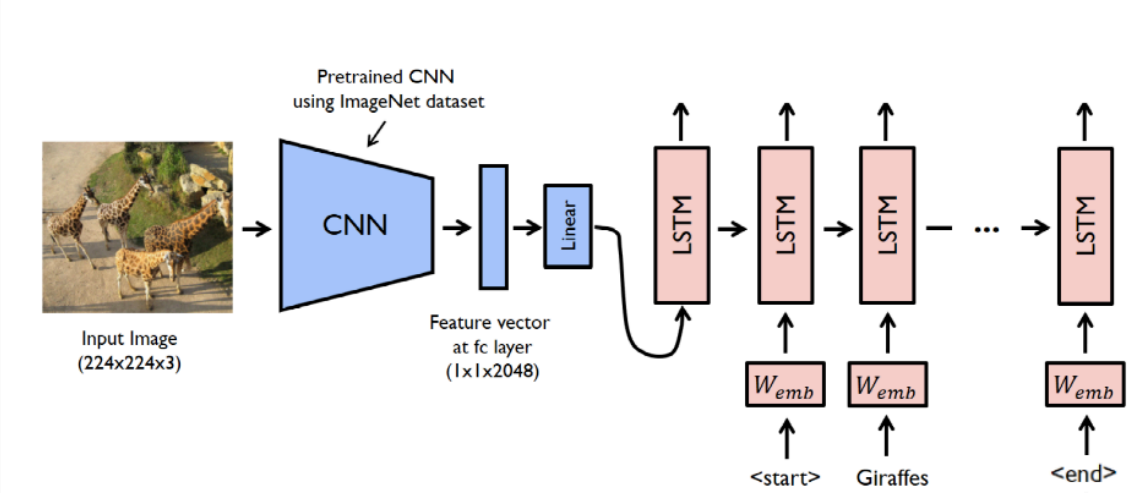
We have two types of situations in which we have to define the behavior of the LSTM:

- During training
- During evaluation

	Training	**Evaluation**
Image feature vector	Yes	Yes
Embedding vectors	Yes	No

3.2.1. LSTM during training

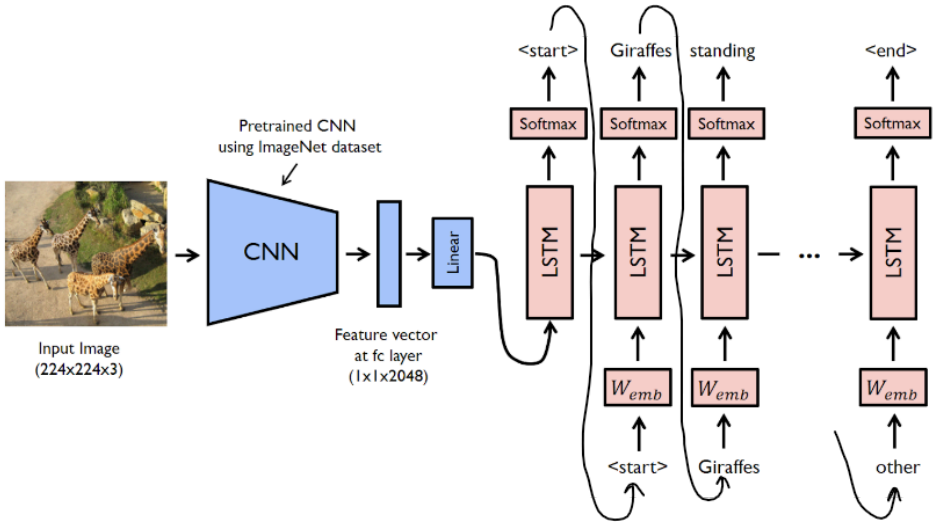
Our strategy during training is to concatenate the image feature vector with the embedding vectors, and use them as input for the LSTM. Notice that as a consequence, we will have N + 2 + 1 LSTM blocks, each one responsible for processing one input.



Additionally, during training we have padded our sequences to ensure all input sequences within each batch had the same length. This is enforced by PyTorch, which requires tensors in a batch to be of uniform size. This is used for efficient training, specifically to keep the computational graph during training low, otherwise a computational graph for each sequence length would need to be constructed.

3.2.2. LSTM during evaluation

Since we do not have embedding vectors, we have to generate those inputs for the LSTM in any way. The solution for that is sampling. As we will see later we have implemented two different ways of sampling, but they are both based on the same idea. Once the LSTM processes the image feature vector and the output is passed through a Softmax, we sample one word based on the probabilities described in this vector and use the sampled word as the input for the following block. This process is repeated until a maximum sequence length is reached or an <<end>> token is sampled.



The sampling strategies described later will be greedy search and beam search.

Overall, during training what we want is to obtain the parameters that maximize the likelihood of the reference caption given the input image.

$$\theta^{\star} = \arg\max_{\theta} \sum_{(I,S)} \log p(S|I;\theta)$$

On the other hand, during inference what we want is to obtain the sequence that maximizes the joint probability of words given the input image and the model parameters.

$$S^{*} = argmax \ \log p(S|I;\theta)$$

where

$$p(S|I;\theta) = \prod_{S} p(s_t|s_1,\ldots,s_{t-1}, I;\theta)$$

4. Training

The training has been conducted locally with the following environment specifications:

GPU	NVIDIA RTX A1000 Laptop
CPU	11th Gen Intel i7
RAM	16GB DDR4
Operating System	Ubuntu 22.04 LSTM

4.1. Training Loss

During training we have used the cross-entropy loss and Perplexity to evaluate the dissimilarity between the generated captions and the ground truth captions. Initially we trained two models, with the following settings:

Cross entropy loss, also known as log loss, is a measure of the difference between two probability distributions: the true labels and the predicted probabilities. It quantifies the performance of a classification model whose output is a probability value between 0 and 1.

The loss increases as the predicted probability diverges from the actual label. Perfect predictions result in a loss of zero, while larger deviations result in higher loss values. Cross entropy loss is widely used in classification problems, especially in neural networks, to guide the training process by minimizing the discrepancy between the true labels and the model's predictions.

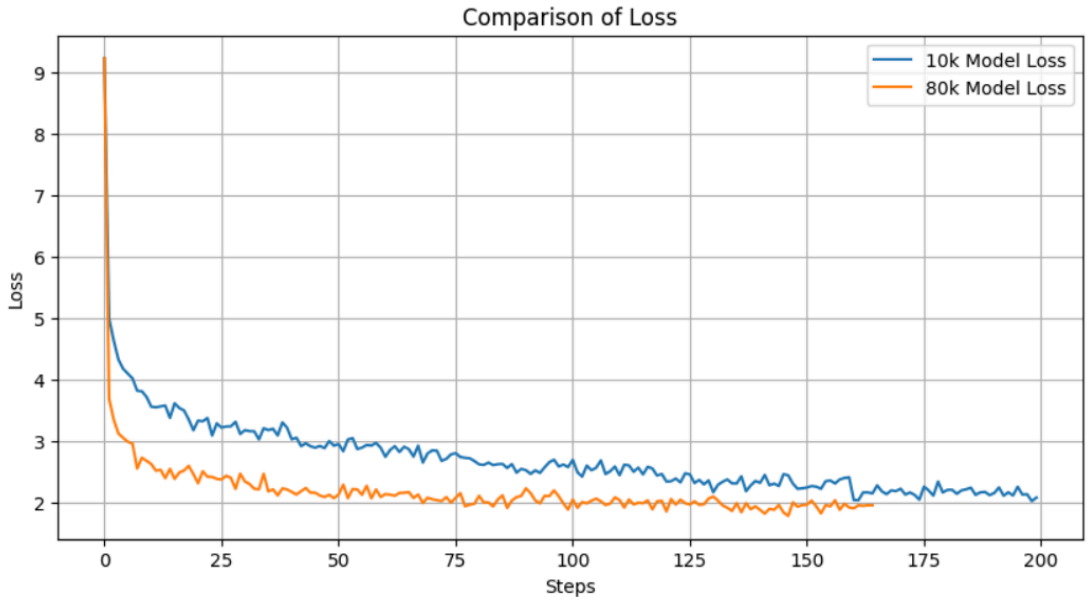
$$L = -\sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Perplexity is a metric used to evaluate language models, particularly in natural language processing. It measures how well a probability distribution or probability model predicts a sample. Lower perplexity indicates a better predictive model.
Perplexity essentially quantifies the model's uncertainty. For a given sequence, it reflects the model's ability to predict the next word. A model with low perplexity is more confident and accurate in its predictions, indicating better performance. In other words, perplexity measures how surprised the model is by the test data; the lower the perplexity, the better the model is at predicting the test data.

$$PP = \exp\left(-\frac{1}{N}\sum_{i=1}^{N} \log P(w_i)\right)$$

	Model 1	Model 2
# Training samples	10,000	80,000
# LSTM layers	1	1

Each step represents a batch.



We can observe that the loss decreases significantly at first, followed by a gradual decline after 10 steps. It appears that the loss converges around 125 steps. This pattern is similar for both models, but as expected, the model trained on 80,000 images performs better due to the increased amount of data, which helps in better understanding future inputs. The 80,000 images model converges faster than the 10,000 images model, but they ultimately converge to a similar loss. The main difference is that the 10,000 images model requires more steps to converge.

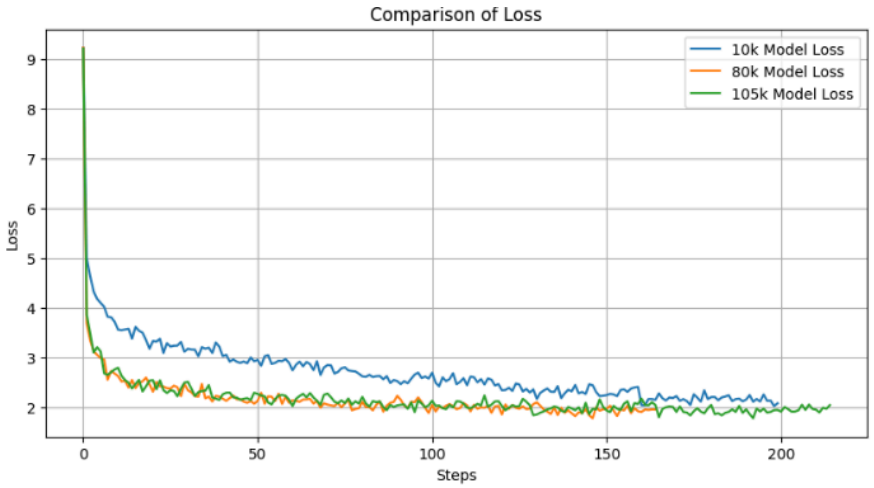
	Model 1	Model 2
Cross-Entropy Loss	2.0750	1.9495
Perplexity	7.9649	7.0250

4.2. Train & Validation redistribution

The original split of the Training / Validation / Test sets was 80K / 40K / 40K images respectively as mentioned earlier. Generally, is it enough to have a validation set of around 10%-20% of the size of the training data. For this reason, we have redistributed our data following a different split: Training = 105K / Validation = 15K / Test = 40K. We believed that training with more data would result in better generalization and consequently higher performance.

	Model 1	**Model 2**	**Model 3**
# Training samples	10,000	80,000	105,000
# LSTM layers	1	1	1

However, after completing the training and analyzing the results, we found that the model performed similarly to the original one, without any significant improvement. This lack of performance gain can be attributed to several factors:
Furthermore, using more data for training is computationally more expensive, leading to longer training times and increased resource consumption.



	Model 1	Model 2	Model 3
Cross-Entropy Loss	2.0750	1.9495	2.0393
Perplexity	7.9649	7.0250	7.6849

5. Model evaluation

We have used the BLEU score to measure the performance of the model in the validation set.

5.1. BLEU

BLEU (BiLingual Evaluation Understudy) is a method for evaluating the quality of text translations generated by deep learning models. In our case, since we are carrying out an image captioning task, it works by comparing the generated caption with five ground truth captions.

Here's how BLEU functions:

- **N-gram Precision**: Count the n-grams (sequences of words) in the generated translation that also appear in the reference translations. N-grams can be of different sizes, such as 1-grams (individual units), 2-grams (word pairs), 3-grams, etc. The precision of each n-gram is calculated as the number of matching n-grams divided by the total number of n-grams in the generated translation.
- **Brevity Penalty:** Apply a penalty for brevity to translations that are shorter than the reference translations. The penalty is calculated as the length of the generated translation divided by the length of the shortest reference translation.
- **Integration of N-gram Precision**: Take the geometric mean of the n-gram precisions. This provides a translation score that shows how well the generated translation matches the reference translations in terms of n-gram precision.
- **Finally BLEU Score**: Multiply the n-gram precision score by the brevity penalty to obtain the final BLEU score. This score is expressed between **0 and 1**, where 1 indicates a translation perfectly similar to the reference translations.

Small example to better understanding :

Comparing metrics for candidate "the the cat"

Model	Set of grams	Score
Unigram	"the", "the", "cat"	$\frac{1+1+1}{3}=1$
Grouped Unigram	"the"*2, "cat"*1	$\frac{1+1}{2+1}=\frac{2}{3}$
Bigram	"the the", "the cat"	$\frac{0+1}{2}=\frac{1}{2}$

BLEU scores have been shown to correlate well with human judgment of translation quality, making them a reliable metric for evaluating image captioning models.

Once we grasp this concept, we can apply it to our model to uncover insights. This will enable us to determine if there are areas where our model needs improvement or adjustments

We have executed our models using 40,000 images (the ones corresponding to the validation set), and this is the outcome.

Average bleu score: 0.20087021083219175 | Average theoretical score: 0.18581574292701428

After that, we can compare the results with some of the models we've based on, firstly on the paper Show and Tell they've designed a model with this results:

Metric	BLEU-4	METEOR	CIDER
NIC	27.7	23.7	85.5
Random	4.6	9.0	5.1
Nearest Neighbor	9.9	15.7	36.5
Human	21.7	25.2	85.4

Table 1. Scores on the MSCOCO development set.

Additionally one of the pretrained models from hugging face (BLIP) obtain the following results:

BLIP: Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation

Method	Pre-train #Images	NoCaps validation in-domain C	in-domain S	near-domain C	near-domain S	out-domain C	out-domain S	overall C	overall S	COCO Caption Karpathy test B@4	COCO Caption Karpathy test C
Enc-Dec (Changpinyo et al., 2021)	15M	92.6	12.5	88.3	12.1	94.5	11.9	90.2	12.1	-	110.9
VinVL† (Zhang et al., 2021)	5.7M	103.1	14.2	96.1	13.8	88.3	12.1	95.5	13.5	38.2	129.3
LEMON$_{base}$† (Hu et al., 2021)	12M	104.5	14.6	100.7	14.0	96.7	12.4	100.4	13.8	-	-
LEMON$_{base}$† (Hu et al., 2021)	200M	107.7	14.7	106.2	14.3	107.9	13.1	106.8	14.1	**40.3**	**133.3**
BLIP	14M	111.3	15.1	104.5	14.4	102.4	13.7	105.1	14.4	38.6	129.7
BLIP	129M	109.1	14.8	105.8	14.4	105.7	13.7	106.3	14.3	39.4	131.4
BLIP$_{CapFilt-L}$	129M	**111.8**	**14.9**	**108.6**	**14.8**	**111.5**	**14.2**	**109.6**	**14.7**	39.7	**133.3**
LEMON$_{large}$† (Hu et al., 2021)	200M	116.9	15.8	113.3	15.1	111.3	14.0	113.4	15.0	40.6	135.7
SimVLM$_{huge}$ (Wang et al., 2021)	1.8B	113.7	-	110.9	-	115.2	-	112.2	-	40.6	143.3
BLIP$_{ViT-L}$	129M	114.9	15.2	112.1	14.9	115.3	14.4	113.2	14.8	40.4	136.7

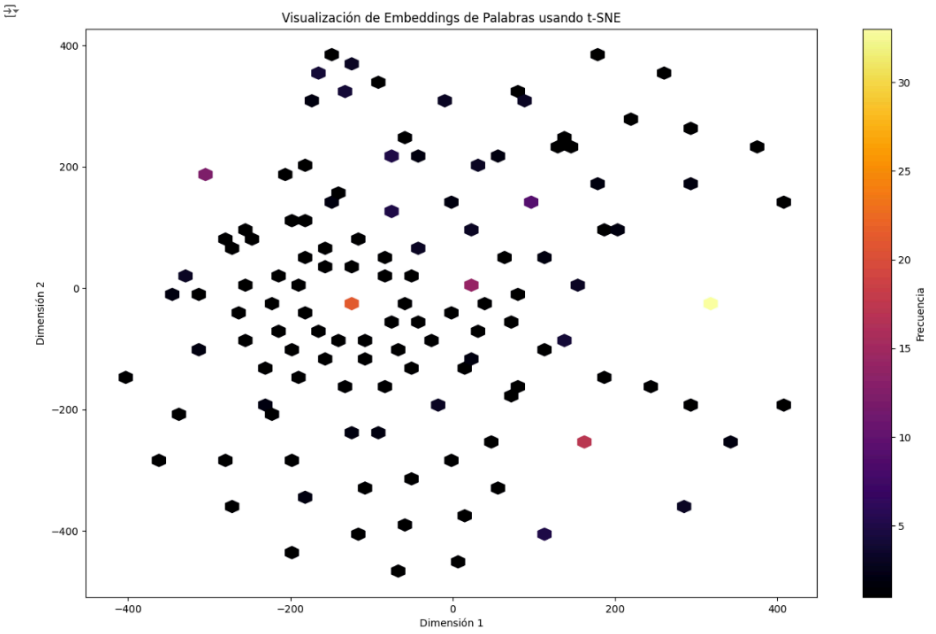
We have evaluated our models and calculated the 'loss' of our models to see how they were doing. Before anything, we set aside the trained model with 3k images since the results were very poor, but we kept it because initially it was very challenging to train the model due to the high computational cost. However, one of our colleagues had a GPU that helped us train the data a little bit faster, so we could train more images.

6. Embedding Analysis

We wanted to explore the embedding distribution to identify the relationship between the embeddings and the captions. To achieve this, we collected the embeddings from the decoder part. However, since they are in a high-dimensional space, we needed to reduce their dimensionality.

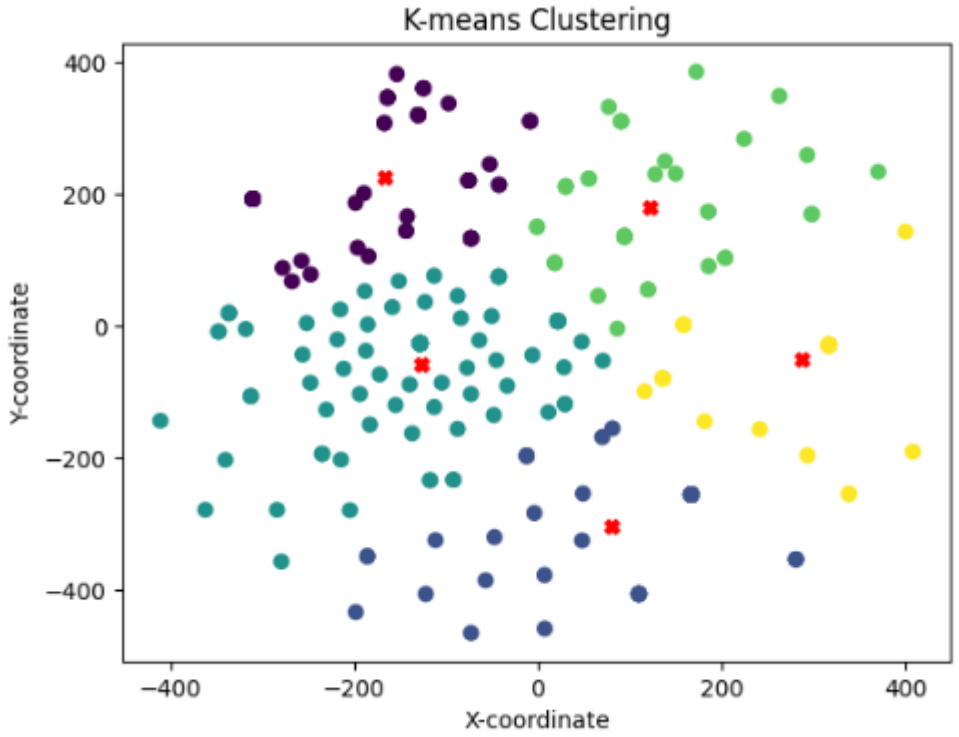
We considered various techniques for dimensionality reduction but ultimately chose **t-SNE,** (t-distributed Stochastic Neighbor Embedding) that is a nonlinear dimensionality reduction technique that is particularly well suited for visualizing high-dimensional data in a lower-dimensional space, typically two or three dimensions. It aims to learn a d-dimensional map that reflects the similarities between high-dimensional inputs by minimizing the Kullback–Leibler divergence between the probability distributions of the high-dimensional data and the lower-dimensional representation.This technique aligns well with our problem. We also considered PCA, but we opted for t-SNE to introduce some variation and learn new, distinct structures.

Once we have the embeddings in a lower-dimensional scale, we can plot them. Before proceeding, a brief reminder: each image has five captions. To begin with, we focused on just five images to understand how the values interact. So, imagine 5 images, each with 5 captions, and approximately 7-8 words per caption without repetitions. This yields around 150 points to plot. Here, you can see a plot where we represent each word and classify them by their frequency of appearance.



Once we had a representation, we considered performing classification to see if similar words are closer together. To achieve this, we applied k-means clustering to group the words. Here, we have a representation of the previous values.

Applying clustering with k = 5



Results of the grouped words

So we can look for the corresponding word captions of the plotted values, we've find out the following:

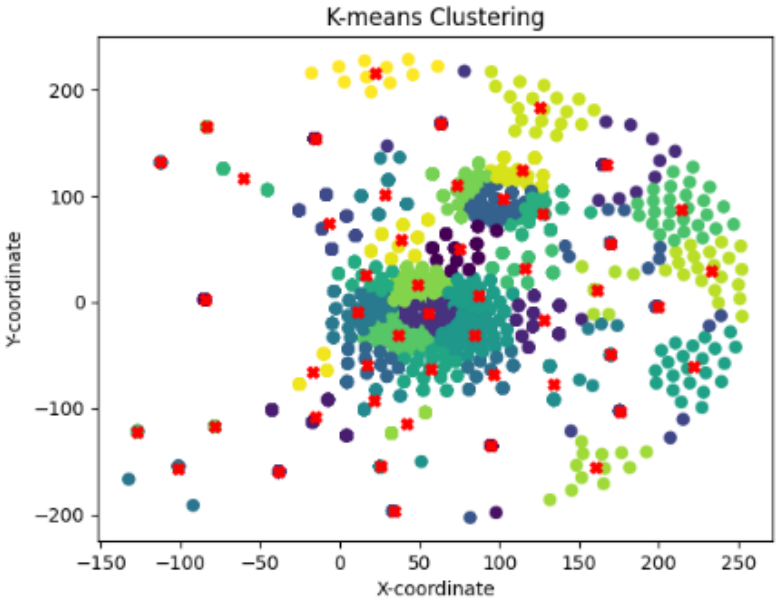
```
Cluster 1:
Words : ['two', 'boys', 'female', '.', 'on', ',', '<<end>>', 'men', 'is']
Cluster 2:
Words : ['jumping', 'frisbee', '<<padding>>']
Cluster 3:
Words : ['going', 'other', 'competition', 'for', 'lush', 'looking', '<<padding>>', 'the', 'residential', 'dress', 'open', 'ground', 'skateboards', 'street', 'green', 'ready', 'neighborhood', 'of', 'people', 'male', 'foot', 'woman', '
Cluster 4:
Words : ['in', 'boy', 'to', 'serve', 'player', 'field', 'catch', 'are', '<<padding>>', 'next', 'engaged', 'with']
Cluster 5:
Words : ['skateboard', 'tennis', '<<padding>>']
```

So, we can observe that similar words are clustered together. Focusing on the first cluster, we see that it contains :

- 'boys', 'female', and 'men'.

This implies that the embeddings are effectively distributing the values in a meaningful manner. Although dealing with a limited number of points, we can expect some errors and words may need to be reassigned to different clusters. However, we can start to form some ideas about this representation. We will add more images to refine the clusters and improve the ability to differentiate between them.

If we proceed in the same way with 50 images (6262 captions in this case) for example, we can see the following.



In this instance, we use more clusters to end up with smaller groups that share common relationships. This allows us to examine whether words with similar topics are closer together than others.

For example in this case :

```
Cluster 17:
Words : ['cabin', 'yellow', 'buses', 'television', 'mounted', 'zebras', 'four', 'blue', 'green', 'couch', 'black']
```

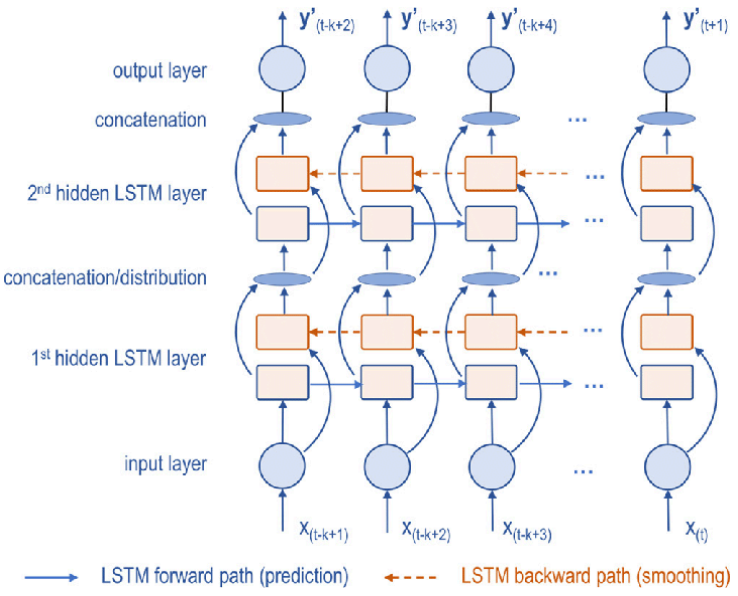
7. Improvements

7.1 Layers & Model complexity

Another improvement we explored was using additional LSTM layers. The intuition behind this is that more layers could potentially capture more complex patterns in the data, leading to better performance. However, we found that adding more LSTM layers increased the complexity of the model significantly.

This increased complexity did not translate into better results. In fact, the performance of the model with additional layers was not noticeably better than that of the simpler model with fewer layers. Moreover, the added complexity led to a new issue: overfitting. With more LSTM layers, the model started to perform well on the training data but poorly on the validation data, indicating that it was memorizing the training data rather than generalizing well to new, unseen data.

Therefore, while the idea of adding more LSTM layers seemed promising initially, it ultimately did not improve our model's performance and introduced overfitting. This suggests that a simpler model might be more effective for our specific task, as it strikes a better balance between complexity and generalization.



7.2 Beam Search

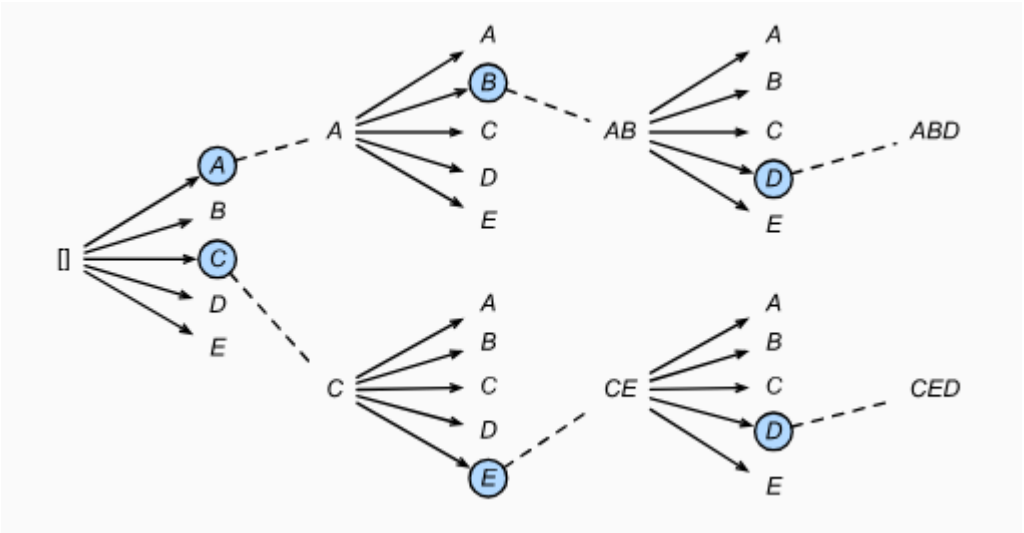
Greedy Decoding is a simple and fast decoding strategy where at each step, the model selects the token with the **highest probability** as the next token in the sequence. This process is repeated until a termination condition is met, such as reaching an end-of-sequence token. This decoding technique is what we have been using to do the output captions.

So we can summarize it in:

- Pros: Fast and computationally efficient.
- Cons: Often suboptimal because it makes decisions based only on immediate highest probabilities without considering future possibilities.

On the other hand, we have **Search Decoding** is a more sophisticated strategy that keeps **multiple candidate sequences** (beams) at each step instead of just one. It explores a fixed number of top sequences (determined by the beam width) and expands them by considering the probabilities of the next tokens. At each step, the beam search retains the top sequences with the highest cumulative probabilities. (we are checking the joint probabilities of each sequence)

- Pros**:** Generally produces better results than greedy decoding by considering multiple hypotheses and looking ahead.
- Cons: More computationally intensive and slower than greedy decoding.



Beam decoding visual representation

So we have modified the function *sample* from the decoder part and we could try to train again to see if the performance changes.

7.3 Data augmentation

For further improvements, we believe that applying additional transformations to create a larger dataset would enhance the model's performance. However, this is not feasible for us due to limited computational power.

Data augmentation is a powerful technique used in machine learning, particularly in training deep learning models. It involves creating new training data by applying various transformations to the existing data, thus effectively increasing the diversity and size of the dataset without actually collecting new data. This is particularly useful when dealing with limited datasets and helps to improve the model's generalization capabilities.

Our approach to data augmentation involves using the same captions for transformed images as the original ones. This can lead to problems because the model might learn to generate similar patterns in the captions. We observed this issue during our project, as the generated sentences were quite similar to those in the training dataset.

Therefore, the best solution might be to use a larger dataset. After conducting some research, we found other datasets, such as COCO-2017, which contain more images. This could be the path to follow.

7.4 Attention

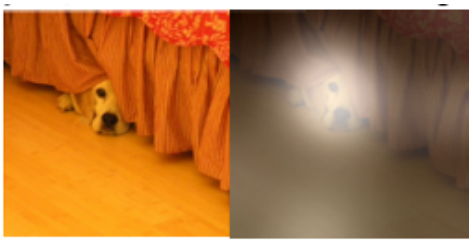
In the context of Image Captioning, "**Attention**" refers to a technique used to enhance the generation of image descriptions. The basic idea behind attention is that instead of processing the entire image at once to generate a description, the model focuses on specific parts of the image at different times during the generation of the descriptive text.

Imagine you're looking at an image and trying to describe it. Your attention would shift to different parts of the image based on what you're trying to convey at that moment. For example, if you're describing a beach scene, you might focus on the sea and waves when discussing the overall scene, but then shift your attention to a beach umbrella and chair when describing specific details.

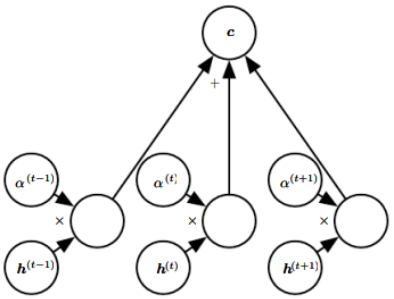
This can improve the coherence and relevance of the generated descriptions, as the model can focus on important details of the image at each step of text generation.

Image representation :

A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A context vector c is formed by taking a weighted average of feature vectors h(t) with weights α(t) . In some applications, the feature vectors h are hidden units of a neural network, but they may also be raw input to the model. The weights α(t) are produced by the model itself. They are usually values in the interval [0, 1] and are intended to concentrate around just one h(t) so that the weighted average approximates reading that one specific time step precisely. The weightsα(t) are usually produced by applying a softmax function to relevance scores emitted by another portion of the model. The attention mechanism is more expensive computationally than directly indexing the desired h(t) , but direct indexing cannot be trained with gradient descent. The attention mechanism based on weighted averages is a smooth, differentiable approximation that can be trained with existing optimization algorithms.

8. State-of-the-art models comparison

8.1 Gpt-2

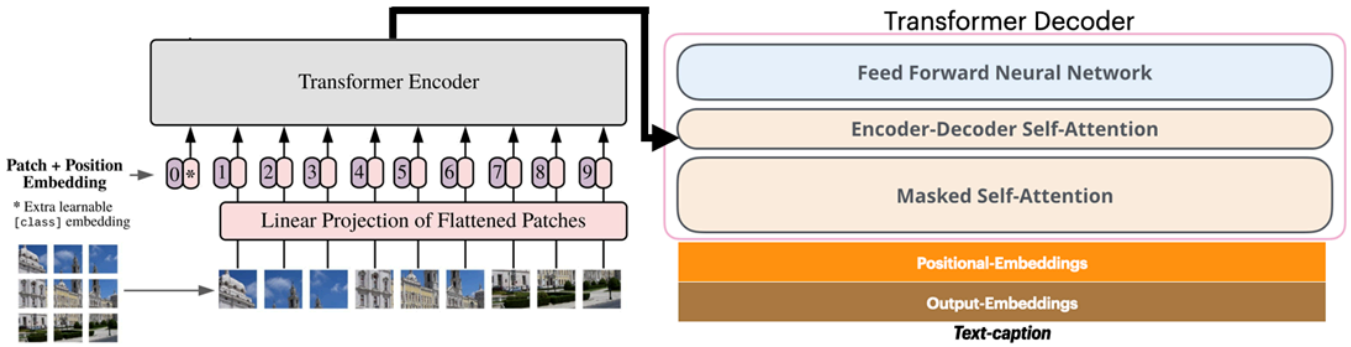
This model employs a ***Vision Encoder-Decoder*** architecture that combines two pre-existing models to perform the task of generating image captions.

Let's try to understand a bit more about this architecture:

On one hand, we have the Vision Transformer (ViT) as the **Encoder.** The vision encoding component of the model is the Vision Transformer (ViT). ViT is a transformer architecture designed specifically for image processing. Unlike traditional convolutional neural networks (CNNs), ViT splits the image into patches (like pieces of a puzzle) and treats them as tokens, similar to words in a sentence for text transformers. Each patch is transformed into a feature vector, and these vectors are passed through transformer layers to capture spatial and contextual relationships within the image.

On the other hand, GPT-2 as the **Decoder**, the text decoding component is the GPT-2 model. GPT-2 is an autoregressive language model that generates text sequences. Once the image has been encoded into a latent representation by the ViT, this representation is passed to the GPT-2 decoder, which generates a natural language description of the image.

For a better understanding here we have a visual representation of the architecture:



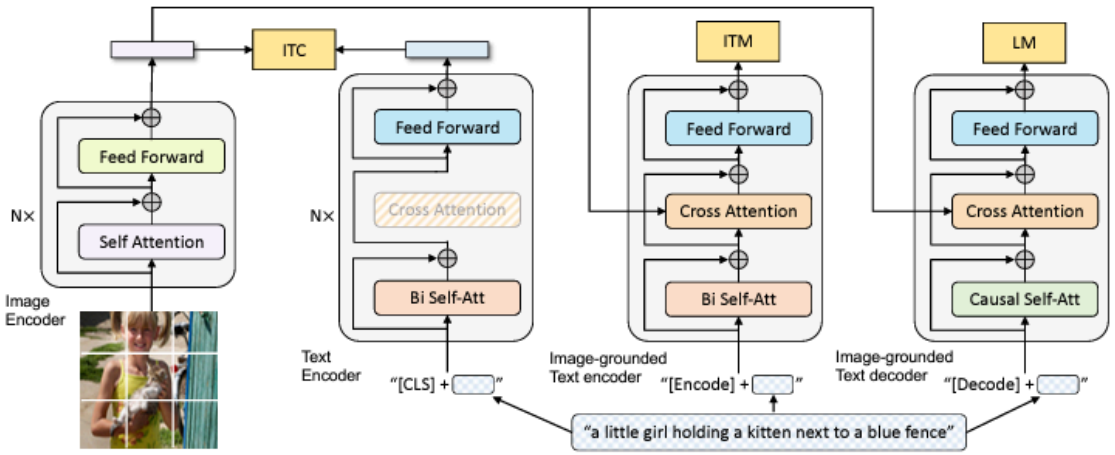
8.2 BLIP (Bootstrapping Language-Image Pre-training)

The architecture used in this code is from a model known as BLIP (Bootstrapping Language-Image Pre-training), which is designed for tasks involving vision and language, such as image captioning. They've used around 150 millions of images. Here's a detailed explanation of the components and the process:

BLIP Processor:
This component is responsible for preprocessing both the image and text inputs to make them suitable for the model. It tokenizes the text and processes the image into a format that the model can understand.

BLIP For Conditional Generation:
This is the core model that performs conditional and unconditional image captioning. It likely consists of a vision encoder (like a Vision Transformer) and a text decoder (like a GPT or BERT variant).

So basically, the BLIP model architecture utilizes a combination of a vision encoder and a text decoder to perform image captioning. The processor handles the preprocessing of images and text inputs, while the model itself generates captions, either conditionally (based on a text prompt) or unconditionally (based only on the image). This allows the model to flexibly generate detailed and contextually relevant descriptions for images.



Output Examples :

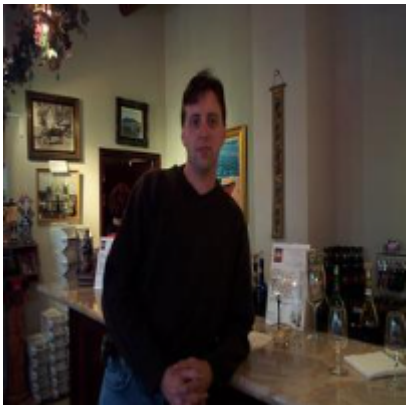
First Example:

- Our model (trained with 3k images) :
    - A group of people standing around a table
- Our model (trained with 10k images) :
    - A black and white cat is sitting on a bench
- Our model (trained with 80k images):
    - A dog sitting on a bench lookin out the windows
- Our model (trained with 105k images):
    - A dog is sitting in a window looking out of a window
- Chat gpt 2:
    - A dog is looking out the window of a fence
- BLIP :
    - A dog with a black nose and a black fence

Second example:

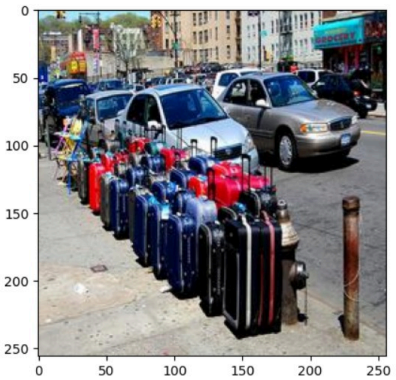
- Our model (trained with 3k images) :
    - “A white plate topped with a sandwich and fries”
- Our model (trained with 10k images) :
    - “A man is sitting on a table with a plate of food ”
- Our model (trained with 80k images):
    - “A glass of wine sitting on top of a table”
- Our model (trained with 105k images):
    - “A vase with a flowers sitting on a table
- Chat gpt -2 :
    - “ Three wine glasses sitting on top of a wooden table ”
- BLIP:
    - “Two glasses of wine on a table overlooking the ocean”

Thread Example (Complex) :



- Our model (trained with 3k images) :
  - “A man is sitting on a couch with a dog on the floor”
- Our model (trained with 10k images) :
  - “A man is holding a wii remote controller”
- Our model (trained with 80k images):
  - “A man and woman standing in a room”
- Our model (trained with 105k images):
  - “A man and woman standing in front of a cake”
- Chat gpt -2 :
  - “ A man standing in front of a counter in a kitchen ”
- BLIP :
  - “A man standing in a kitchen with a wine glass”

Fourth Example (Complex image):



- Our model (trained with 3k images) :
  - “A group of people standing around a table”
- Our model (trained with 10k images) :
  - “ A man riding a bike down a street next to a building ”
- Our model (trained with 80k images):
  - “ A man riding a bike with a dog on a leash”
- Our model (trained with 105k images):
  - “ A group of people standing around a parking lot”
- Chat gpt -2 :
  - “ A row of suitcases lined up on a sidewalk ”
- BLIP :
  - “A row of suitcases”

Once we have done a few examples, we can evaluate visually what users would actually see when using this model. If we look at the **first two** examples, they are quite "**easy**" to describe since there are not many details and the important part of the image is quite centered. Therefore, we can see that our model describes these types of images quite well, and the difference between our result and those of other models trained on many more images and hours is not that significant.

Although we can see that our model, when trained with a small number of images (3k-10k examples), generates an output that is completely different and nonsensical, giving us an understanding that the output will be incorrect for most images. However, when we train the model with the entire dataset, things improve.

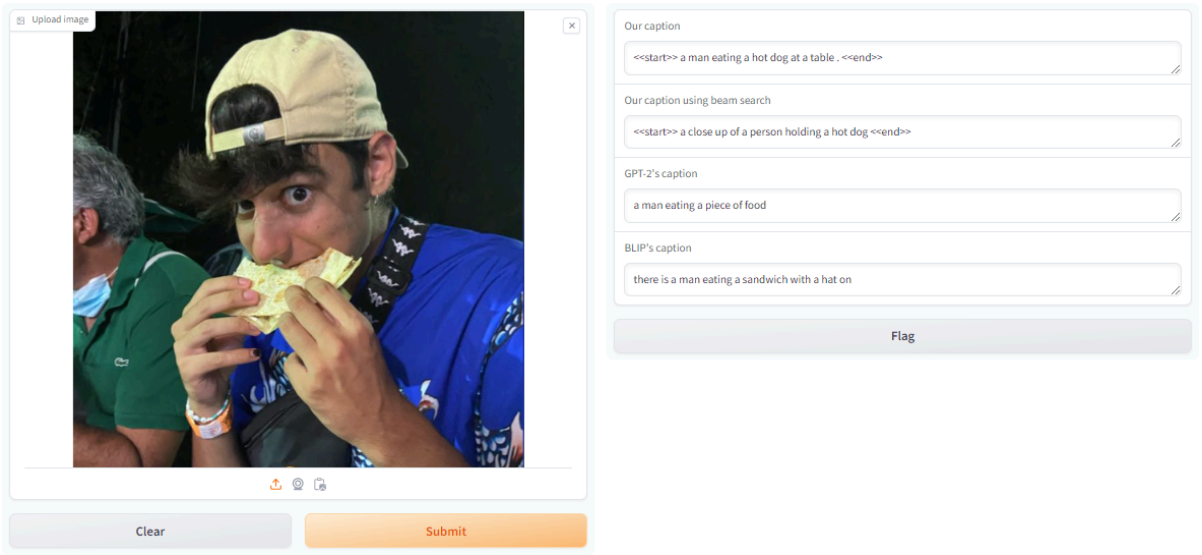
Now, if we look at examples 3-4, we can see that the images have a lot of details, and it is difficult for the model to distinguish which elements are main and which are not. In example 3, we can see how our model got a bit confused and put the words "A man and woman" when there is only one man, so we can see that in slightly more complex images, our model will have limitations. On the other hand, if we look at the models that are theoretically better, they are quite precise.

If we look at the last example we have provided, we can see that it has many details and is a somewhat unusual image. In this case, we can see that our model is completely wrong, while the other two models are closer to a good caption. However, some details about the cars could be added, as the results only mention the suitcases.

Personal Captions :

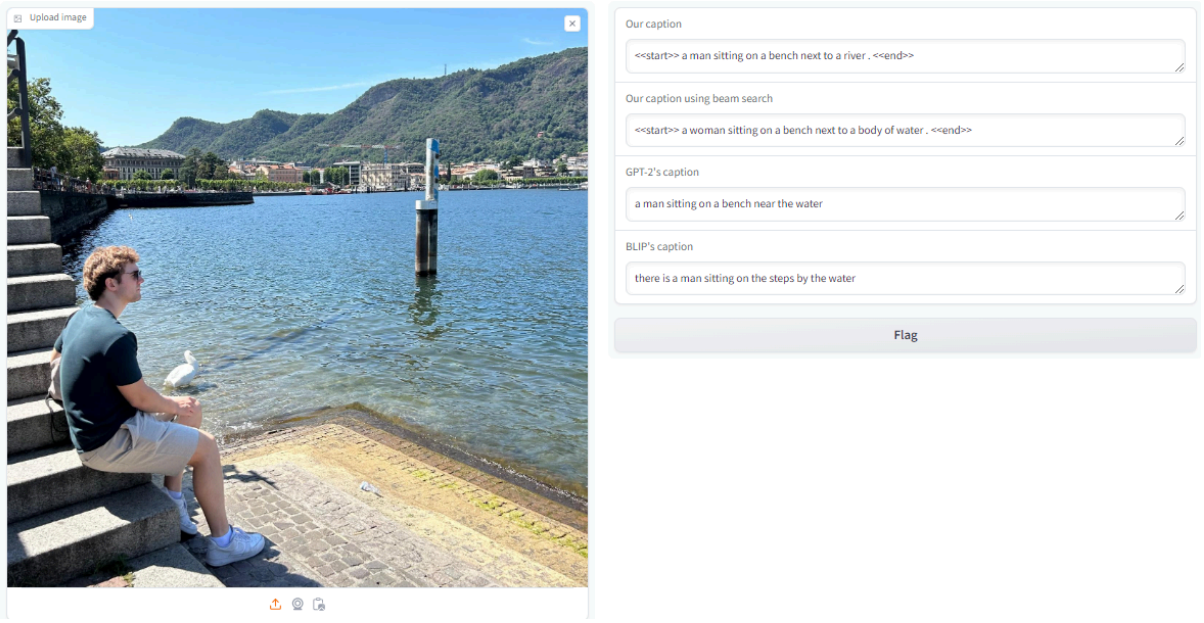
We use a Gradio interface that allows users to upload an image and receive captions generated by different models. It includes a single image upload component and four text boxes for displaying captions. The captions are produced by our best model (trained with 80k images) using greedy decoding and also one caption with beam search. Additionally, we compare our model with the pretrained models GPT-2 and BLIP.

Once an image is uploaded, the results will be displayed on the right-hand side of the screen, here we have some examples:

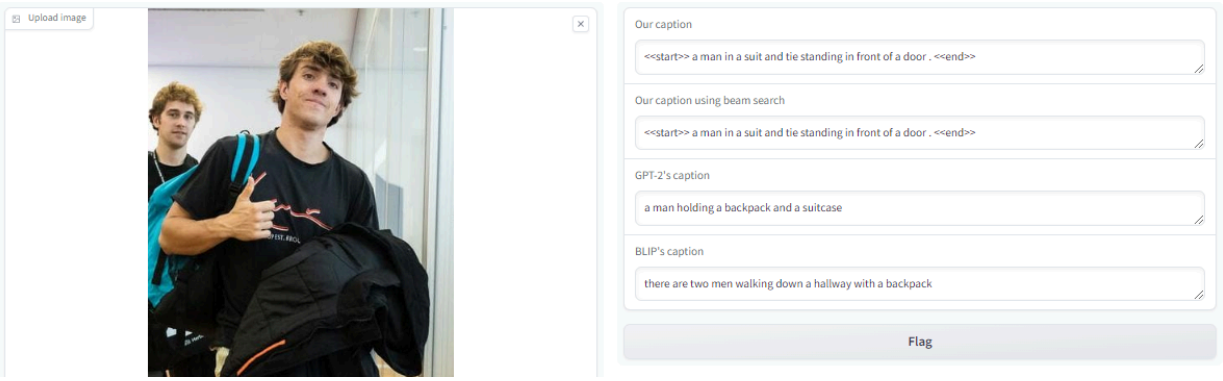


As we can see, the results are quite accurate. This is because the photo is relatively simple, and our model is able to effectively capture its essence. Although it's not the most advanced model, it's still able to provide a general description of the photo. The GPT caption is generating a general caption, which is sufficient. The BLIP model, on the other hand, is more specific, as it also mentions the fact that the person is wearing a hat.

In this example in concrete, we can see that the beam search gives us more details than the one using greedy decoding.



In this image, there are more details, but our model is able to extract the key feature that the person is in front of a body of water, specifically a lake. This is accurate even with the presence of many background details. The other two models also generate good captions, and all three captions correctly identify that the man is seated, which is a positive aspect overall.



In this final example, we can see that our model is slightly confused, as it mentions that the man is wearing a suit. This is likely due to the fact that the training dataset contains many images of men wearing suits, causing the model to default to this assumption when it encounters a photo with similar features. Despite this, the overall caption is still acceptable.