VR PART 2 Mini Project Report

Team Id : 12

Team members

Ch. Sivani(IMT2019020)
G. Sri Harsha(IMT2019030)
M. Dhanush(IMT2019049)

Problem 1 :

Design a CNN-LSTM system (preferably in pytorch) that can perform image captioning [System 1] based on the following details:

- free to decide the CNN and LSTM architecture that best suits your case. The objective is to achieve a good BLEU score on the test set of Flickr8K data .
- Use the Flickr8K data for training and testing the model.

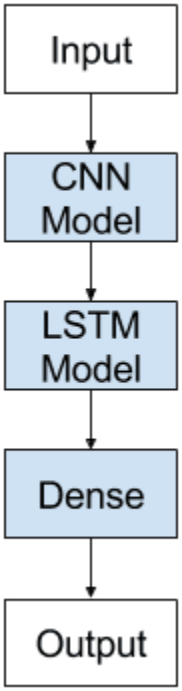
Introduction :

Image Captioning :
The technique of creating captions for a picture based on the items and behaviors in the image and generating textual description of an image is known as image captioning.
It recognises the context of an image and describes it in a natural language like English using Computer Vision and Natural Language Processing techniques. The data will be in the following format: (image, captions).
The encoder-decoder architecture used to create these captions uses an abstract picture feature vector as an input to encoders. Object identification models, Convolutional Neural Networks (CNN), and attention-based Recurrent Neural Networks have all contributed significantly to the improvement of picture caption outcomes.

The typical Vanilla LSTM cannot easily simulate input with spatial structure, such as pictures. The CNN Long Short-Term Memory Network, or CNN LSTM, is an LSTM architecture intended primarily for sequence prediction issues using spatial inputs such as images or videos.

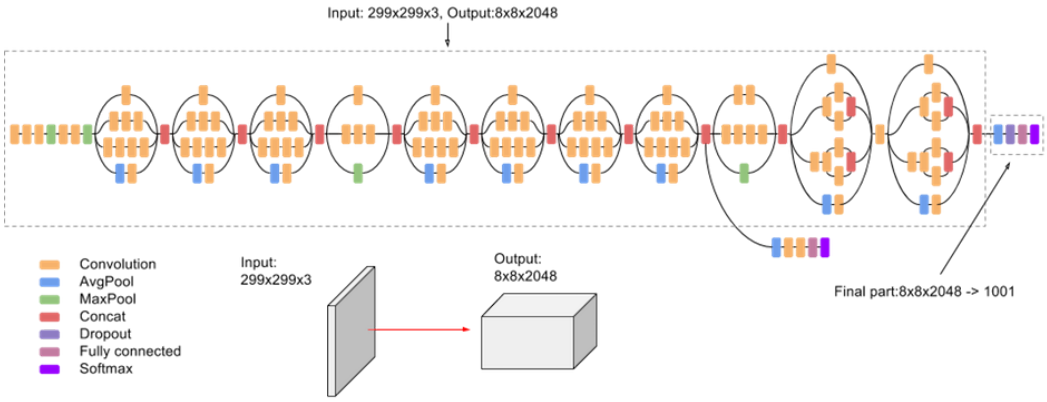
Convolutional Neural Network (CNN) layers for feature extraction on input data are paired with LSTMs to facilitate sequence prediction in the CNN LSTM architecture.

CNN LSTMs were created to solve visual time series prediction problems and to generate textual descriptions from image sequences(e.g. videos).



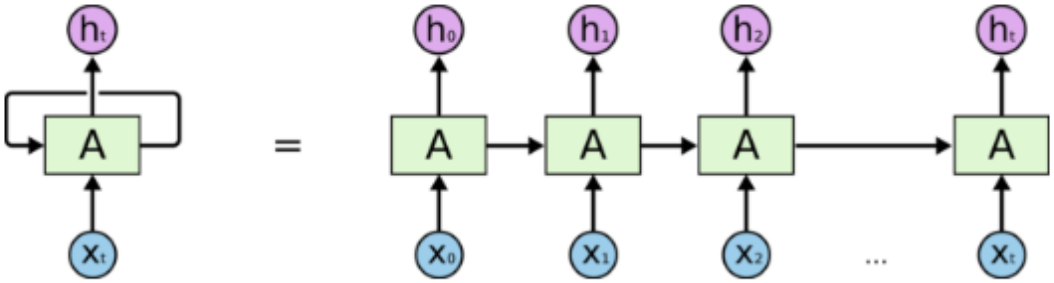
CNN : CNN is a deep learning technique that is widely applied to image analysis. Unlike older methods, which require hard engineering of image features, CNN allows us to provide input as is and let the picture learn the features/characteristics on its own.

Multiple layers of artificial neurons make up CNNs. The output of one CNN layer is sent to the next. The first layer of a CNN usually recognises basic features like horizontal and vertical edges. It starts to recognise higher-level elements like objects and faces as you advance further into the layer. Convolutional, pooling, and fully-connected CNN layers are available.



RNN : A feedforward neural network with an internal memory is known as a recurrent neural network. RNN is recurrent in nature since it executes the same function for each data input, and the current input's outcome is dependent on the previous computation.
The output is replicated and transmitted back into the recurrent network when it is created. It evaluates the current input as well as the output it has learned from the prior input when making a decision. RNNs, unlike feedforward neural networks, may process sequences of inputs using their internal state (memory).

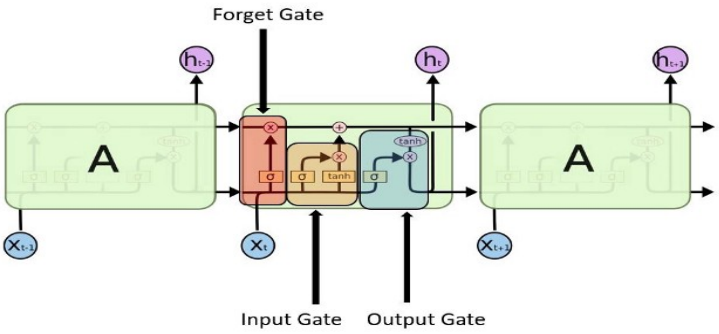
As a result, activities like unsegmented, connected handwriting recognition or speech recognition are possible. All of the inputs in other neural networks are independent of one another. However, in an RNN, all of the inputs are connected.



An unrolled recurrent neural network.

Long Short Term Memory (LSTM) : Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags

of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present:



1. Input gate — discover which value from input should be used to modify the memory. The Sigmoid function decides which values to let through 0,1. and the tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1.

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

2. Forget gate — discover what details to be discarded from the block. It is decided by the sigmoid function. it looks at the previous state(ht-1) and the content input(Xt) and outputs a number between 0(omit this)and 1(keep this)for each number in the cell state Ct−1.

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

3. Output gate — the input and the memory of the block is used to decide the output. The Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed, deciding their

level of importance ranging from-1 to 1 and multiplied with output of Sigmoid.

$$o_t = \sigma\left(W_o\ \left[h_{t-1}, x_t\right]\ +\ b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

About Dataset :

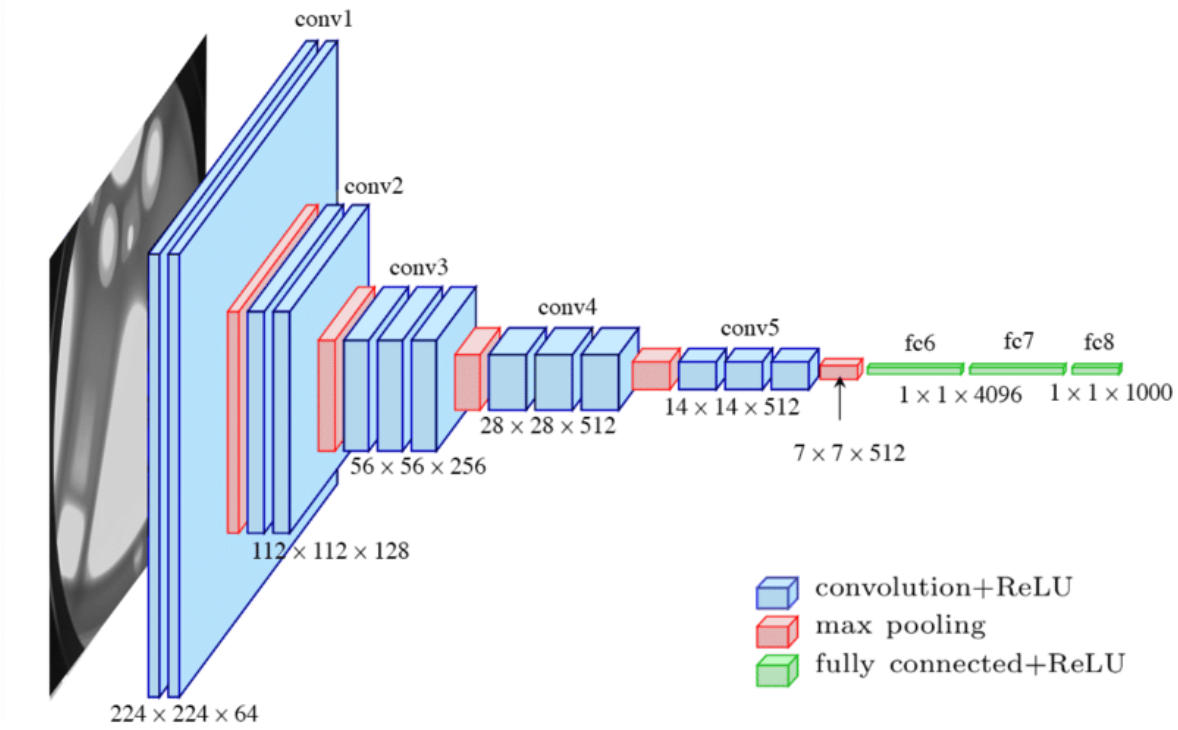
A new benchmark collection for sentence-based image description and search, consisting of 8,000 images that are each paired with five different captions which provide clear descriptions of the salient entities and events. The images were chosen from six different Flickr groups, and tend not to contain any well-known people or locations, but were manually selected to depict a variety of scenes and situations.

- Training Set — 6000 images
- Val Set — 1000 images
- Test Set — 1000 images

Image feature generation :

The first step is to develop rich features out of photographs that are representative of the content. For this task, it is ideal to use pre-trained deep CNNs, which have been trained on huge datasets like ImageNet and can create outstanding features. We used PyTorch's VGG-16, which is a VGGNet with 16 layers.

Architecture


conv1
conv2
conv3
conv4
conv5
fc6
fc7
fc8
224 × 224 × 64
112 × 112 × 128
56 × 56 × 256
28 × 28 × 512
14 × 14 × 512
7 × 7 × 512
1 × 1 × 4096
1 × 1 × 1000
convolution+ReLU
max pooling
fully connected+ReLU

Model

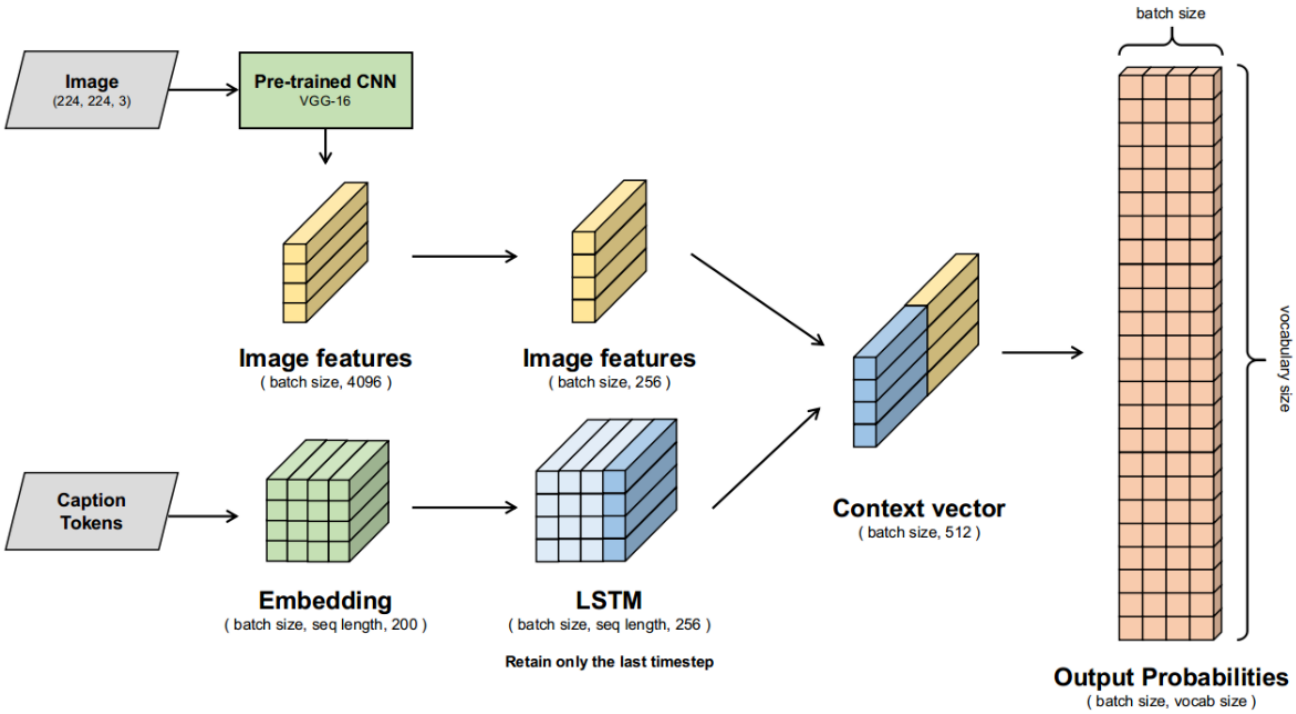


Image
(224, 224, 3)
Pre-trained CNN
VGG-16
Image features
(batch size, 4096)
Image features
(batch size, 256)
Caption Tokens
Embedding
(batch size, seq length, 200)
LSTM
(batch size, seq length, 256)
Retain only the last timestep
Context vector
(batch size, 512)
batch size
vocabulary size
Output Probabilities
(batch size, vocab size)

The model's last layer is a completely connected layer that maps to 1000 units. This is preserved for classification purposes (the ImageNet dataset includes 1000 classes) but isn't required by us. Only the model's penultimate layer will be kept, giving us a feature vector with 4096 dimensions for each image.

The form of all images passing through the network must be the same. To accomplish this, we use **transform.compose** to create a transformation pipeline. Compose is a torch function. The following operations are carried out by this pipeline.

- The image is resized to (256, 256, 3).
- Crops the image to only keep the size of the center (224, 224, 3). This implies that the image's edges do not contain any useful information, which is reasonable in most cases.
- This PIL image is converted to a torch tensor.
- Each of the image's three channels (RGB) is normalized with averages (0.485, 0.456, 0.406) and standard deviations (0.229, 0.224, 0.225). We'll use these numbers because they were determined to yield the best results in the ILSVRC (a huge computer vision competition).

Caption Processing :

There are five captions for each image. The first step may be to organize this data in a more user-friendly manner. A lookup table with the filename as the key and a list of 5 captions for that file as the value should suffice. We put together a dictionary. We break each sentence on the tab delimiter "t" to separate the keys from the sentences. We'll also delete the number tags from the filenames (#0, #1, etc.).

Tokenization :

Generating a vocabulary of all the words we have (bag of words) and assigning an integer to each word.

Preprocessing :

Some components of caption statements are meaningless. Uppercase alphabets, punctuation, and digits are examples (at least for our purposes). First, we'll remove these entities from each statement. Then, at the beginning and conclusion of each sentence, we'll add two unique tokens to signify the start and finish of the sentence (startseq and endseq respectively).

These tell the model when to predict and when to stop. After that, we use the Tokenizer class from the keras.preprocessing.text submodule to tokenize all of the lines. When given a batch of inputs, neural networks expect them to be of a fixed size. Because captions might vary in length, we must pad the end of each sentence with special tokens (typically zeros) to ensure that all inputs have the same structure.

Global Vectors is what GloVe stands for. Consider these vectors to be a word lookup table. We go to this "learned" database, request the vector associated with a word, and replace the word's integer token with it. GloVe is a fantastic set of pre-trained vectors that have been learned on massive corpora like Common Crawl and Wikipedia. These vectors are excellent at capturing word co-occurrences, and we'll use them again here.

Encoder :

A pre-trained Convolutional Neural Network model called MobileNet v2 is employed as an encoder. It has an inverted residual structure, with the input and output of the residual block being thin bottleneck layers, as opposed to typical residual models, which employ extended representations in the input. It uses lightweight depth-wise convolutions to filter features in the intermediate expansion layer.

Non-linearities in the thin layers were also eliminated to maintain the representational power. The compressed output gives a summary of the image's usefulness. As the model builds smaller and smaller representations of the original image, each consecutive representation becomes more "learned," with an increasing

number of channels. The final encoding has a resolution of 14x14 pixels and 1280 channels. To resize the encoding to a fixed size, the AdaptiveAvgPool2d layer is used.

Decoder :

To generate a caption word per word, a Recurrent Neural Network with a stack of LSTM layers and an Attention Mechanism is utilized as a Decoder. The Attention network is made up of only linear layers and a few activations. Separate linear layers transform both the encoded image (flattened to N, 14 * 14, 1280) and the hidden state (output) from the Decoder to the same dimension, namely the Attention size.
The encoded image is used to initialize the hidden and cell states of the LSTM utilizing two independent linear layers. After that, they've added and ReLU is turned on. A third linear layer transforms the result to a dimension of 1, after which the softmax is employed to obtain the weights alpha.

The previous concealed state's sigmoid activated linear transform The attention-weighted encoding is processed using a decoder filter or gate to emphasize the items in the image. The LSTMCell is concatenated with this filtered attention-weighted encoding and the embedding of the prior word to form the new hidden state.

A linear layer transforms this new hidden state into scores for each word in the lexicon, which are then saved. At different times in the sequence, the model examines different sections of the encoded image. Instead of using the basic average, the weighted average over all pixels is employed, with the weights of the relevant pixels being higher.
To generate the next word, this weighted representation of the image can be concatenated with the previously generated word at each step.

The output of the Decoder is transformed into a score for each word in the vocabulary using a linear layer. The best caption is found using beam search. Rather than greedily choosing the most likely next step as the sequence is formed, it expands all possible next steps and preserves the k most likely, where k is a

parameter that regulates the number of beams or simultaneous searches across the sequence of probabilities.

Cross-entropy loss was utilized to train the model using Maximum Likelihood Estimation. Words from the supplied sentence are fed at various points during training, with the expected output being the next word in the sentence.

As a result, the training protocol looks for the factors that best explain text data. During inference, the output of the current time step is used as the input for the next one instead of the true words. The losses are not computed in the padded sections.

The model's performance on the validation set is evaluated using the Automated Bilingual Evaluation Understudy (BLEU) evaluation metric. A generated caption is compared to a reference caption with this function (s). For each generated caption, all available captions for an image are considered as reference captions.

Hyperparameters :

Epochs : 15 to 50
Learning rate : 0.0003
Optimizers used : Adam and RMSprop

Experiments done to improve the performance :

1. Tuning the Encoder: Because the convolutional blocks must have learnt something very basic to image processing, such as identifying lines, edges, and curves, the MobileNet-v2 model tends to take more computational resources to update the weights of the two convolutional blocks, it is not fine-tuned.
2. Increasing Epochs
3. Increasing LSTM layers: More LSTM layers were added to provide for greater model complexity and access to more levels of abstraction of input observations over time, however this complicated model specifications and raised computing costs dramatically. As a result, there is only one LSTM layer in the model.

4. Teacher Forcing-based Training: This is a technique for accelerating convergence. It's difficult to train a model from the start when the dataset is tiny, and convergence can take a long time. Ground truth is provided as the next input rather than the previous output at each stage of the decoder in this type of training. The model will converge faster as a result of this. Different teacher forcing ratios have been tried.
   The model was able to train appropriately and converge faster with a teacher forcing ratio of 0.5. The disadvantage is that the inference must be written individually, as we did.

Results :

CNN model	**activation function**	**Epochs**	**loss**	**BLEU-score (training)**	**BLEU-score (testing)**
Alexnet	ReLU	20	0.8537	BLEU1:0.3104 BLEU2:0.1495 BLEU3:0.0624 BLEU4:0.0163	BLEU1:0.3097 BLEU2:0.1456 BLEU3:0.0581 BLEU4:0.0137
Alexnet	ReLU	30	0.7895	BLEU1:0.3230 BLEU2:0.1621 BLEU3:0.0750 BLEU4:0.0289	BLEU1:0.3223 BLEU2:0.1582 BLEU3:0.0707 BLEU4:0.0263
Alexnet	ReLU	35	0.7462	BLEU1:0.3338 BLEU2:0.1729 BLEU3:0.0858 BLEU4:0.0397	BLEU1:0.3331 BLEU2:0.1690 BLEU3:0.0815 BLEU4:0.0371
Alexnet	Tanh	20	0.9851	BLEU1:0.3883 BLEU2:0.2223 BLEU3:0.1312 BLEU4:0.0776	BLEU1:0.3558 BLEU2:0.1815 BLEU3:0.0845 BLEU4:0.0340
Alexnet	Tanh	30	0.8722	BLEU1:0.4009 BLEU2:0.2349	BLEU1:0.3684 BLEU2:0.1941

				BLEU3:0.1438 BLEU4:0.0902	BLEU3:0.0971 BLEU4:0.0466
Alexnet	Tanh	35	0.8362	BLEU1:0.4117 BLEU2:0.2457 BLEU3:0.1546 BLEU4:0.1010	BLEU1:0.3792 BLEU2:0.2049 BLEU3:0.1079 BLEU4:0.0574

Prediction : A black and white dog is running on a beach.

Prediction : A woman with white shirt is playing in the street.

Problem 2:

The LSTM-based image captioning can ‘blindly’ learn the structure of the language and predict meaningful sentences even without learning much insight into the content of the image. This is termed the “language bias” of the system.
Design a training experiment with Flickr8K data to assess the language bias of your CNN-LSTM system [System 1 Modified]. Provide objective and subjective analysis/comparison of the results of System 1 and System 1 Modified.

Introduction:

Although LSTM-based image captioning predicts a meaningful sentence, it may not be related to the image, i.e. may not provide an understanding of the image's content, and this is known as language bias.

Analysis:

Noun Bias: Extract all nouns from both the actual and predicted captions for the test image. If a noun appears in the predicted caption but not in the actual phrase, the model has learned that term from the corpus.
We'll see a few examples of the training experiment with Flickr8K data to analyze our CNN-LSTM system's linguistic bias.

Results:

We can observe that while training a dataset there are many predictions that occurred with the “two dogs” as the sentence and some of them are like men are in a black shirt. For some of the images, it detected fault i.e it is not related to the context. Here, we found language bias.

Actual Image Caption: A brown dog is running with a yellow toy on the ground.

Predicted Image Caption: Two white dogs are playing with each other over grass.

Subjective comparison: This image says that many dogs playing on the ground holding a yellow toy in training. Hence LSTM tries to complete the sentence with the phrase “yellow ball” even though that image doesn’t contain it.

Actual Image Caption: Two boys are playing with a ball on a seashore.

Predicted Image Caption: A man with a shirt is jumping on brown soil.

Subjective comparison: This image says that men are jumping with a shirt on brown soil in training. Hence LSTM tries to complete the sentence with the phrase “with a shirt” even though that image doesn’t contain it.

Actual Image Caption: A brown dog is running over grass.

Predicted Image Caption: A dog is running with a green ball over grass.

Subjective comparison: This image says that dogs holding a green ball in training. Hence LSTM tries to complete the sentence with the phrase “green ball” even though that image doesn’t contain it.

Solution for language bias:

- We need to improve the training set's quality, make it more balanced in terms of sentences used, and avoid the use of too many repeated phrases in the training set to help resolve the issue.

- By changing hyperparameters we may resolve the issue of language bias.

References :

Flickr8K data :
https://drive.google.com/drive/folders/1RQ5qHm0aVFqWDG9VBiSnXlNPl5T15Wf_?usp=sharing

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning

https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html

https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

https://github.com/NishantPrabhu/Image-Captioning-with-PyTorch-and-Keras

https://medium.com/@raman.shinde15/image-captioning-with-flickr8k-dataset-bleu-4bcba0b52926