Image Captioning Using Attention Mechanism

Sanjay Singh

Dec-2019 to Sep-2020

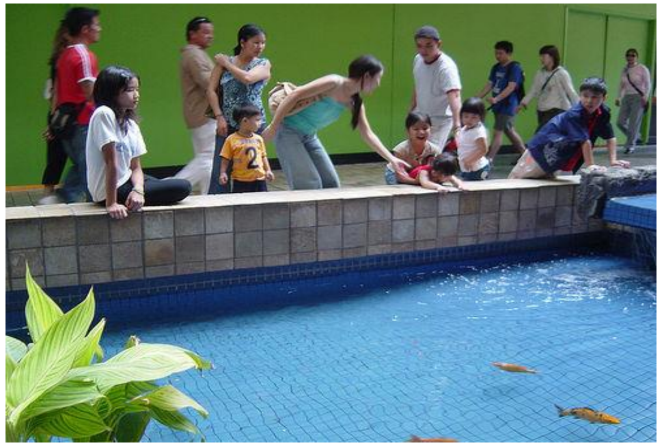
Jio Platforms Ltd. – Navi Mumbai (M.H), India

san.singhsanjay@gmail.com

Introduction

- Automated Image Captioning (or simply Image Captioning) can be defined as generating a textual description for a given image.
- This problem was well researched by Andrej Karpathy in his PhD at Stanford University.
- Deep Learning has achieved state-of-art result in Image Captioning.
- In this project, the advanced solution (i.e., Attention Mechanism or to be more specific, Visual Attention Mechanism) for Image Captioning is implemented. A classical solution for Image Captioning is Encoder/Decoder based
- Attention Mechanism is also quite useful in Neural Machine Translation, i.e., translating text from one natural language to another.
- Following is an example of Image Captioning:

1. Children sit and watch the fish moving in the pond.
2. people stare at the orange fish.
3. Several people are standing near a fish pond.
4. Some children watching fish in a pool.
5. There are several people and children looking into water with a blue tiled floor and gold fish.



Introduction…

- Some other sample images are:

Introduction…

- Following are some of the applications of Image Captioning:
   1. Self Driving Cars
   2. Aid to blind people: It can guide blind people by generating text for the scene in front and speaking it by using TTS (Text to Speech) systems.
   3. CCTV cameras are everywhere, but along with viewing the world, it can generate relevant captions, then we can raise alarms as any malicious activity take place.
   4. Image Captioning can make Google Image Search better.

Dataset

- Flickr8k dataset is used here. Following is the link of this dataset:
  https://www.kaggle.com/adityajn105/flickr8k
- Flickr8k has 8,091 images with caption.txt file containing five captions for each image. All these five captions are written by different people. Thus,
  8,091 Images x 5 Captions = 40,455 Image-Captions
- Size of this dataset: 1.04 GB
- A training file and testing file containing name of images to be used in training and testing is downloaded from the Internet (source is missing).
- Other than Flickr8k dataset, some other datasets for Image Captioning are:
  1. Flickr30k: It contains 30,000 images
  2. MS-COCO: It contains 1,80,000 images. This is the largest dataset for Image Captioning.
- We will work on Flickr8k dataset as this is sufficient to learn the implementation of Automatic Image Captioning using Attention Mechanism.

Technology

- In this project, advanced solution (i.e., Attention Mechanism or Visual Attention Mechanism) for Image Captioning is implemented.
- A naive approach is also there, called as Classical Encoder/Decoder based approach. This advanced solution is also Encoder/Decoder based, however it has an additional layer for Attention Mechanism.
- Attention Mechanism is also quite useful for Neural Machine Translation (i.e., translating text from one natural language to another natural language). To be more specific, Attention Mechanism for Image Captioning is called as Visual Attention Mechanism.
- This project is at the intersection of two technologies:
    1. Computer Vision (CV): To understand the content of a given image.
    2. Natural Language Generation (NLG): NLG transforms data into plain English text.
- Applications of Natural Language Generation (NLG):
    1. Freeform text generation: User provides an input, like a phrase, sentence or paragraph and the NLG model generates continuation of this input as output. For instance, Google Smart Compose predicts a phrase following a word input in Gmail.
    2. Question Answering: This is a system that can answer questions posed by humans. These systems can be open-ended or close-ended (domain specific).

Technology…

3. Summarization: Summarization reduces the amount of information while capturing the most important details in a narrative. This is of two types:
   i. Extractive Summarization: It takes the most important phrases or sentences from the given text and stitches them together to form a summarized narrative.
   ii. Abstractive Summarization: This is equivalent of a human writing a summary in his / her own words. For instance, headline generation, abstract for journals / whitepaper / etc.
4. Image Captioning

- How NLG is different from NLP: NLP is focussed on deriving analytic insights from textual data. Whereas, NLG is used to synthesize textual content by combining analytic output with contextualized narratives. In short, NLP reads while NLG writes.

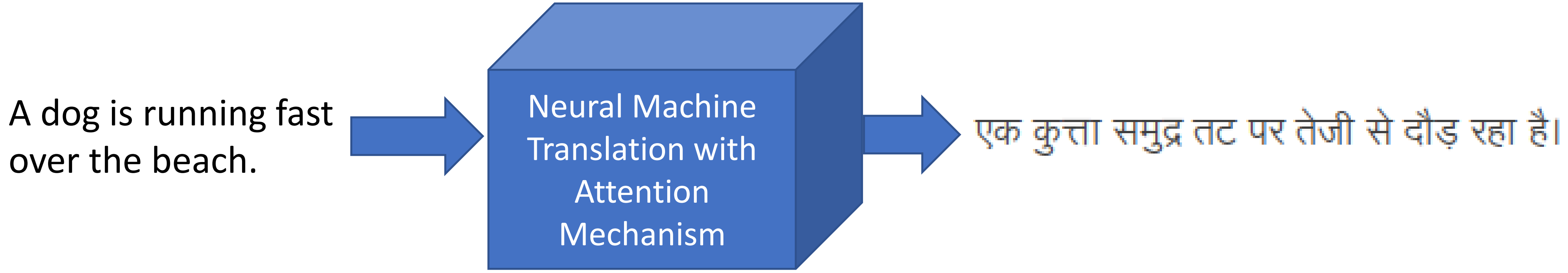
Attention Mechanism

- Attention Mechanism is one of the most influential ideas in the Deep Learning.
- Initially, this idea was designed for Neural Machine Translation. However, today it can be used in various problems, like Image Captioning, etc.
- As we know, there is a classical Encoder / Decoder based solution for Image Captioning. The drawback with that classical approach is:

  Since each word of the caption would be defining a part of the image, thus by considering the whole representation of image to generate the next word of caption which will be describing a part of the image would not be efficient, especially for long captions or descriptions.
- Attention mechanism is a complex cognitive ability that human possess. When people receive information, they consciously ignore some of the secondary information. This ability of self selection is called Attention.
- Attention Mechanism allows the neural networks to have the ability to focus on its subset of inputs to select specific features.
- Neural Network architecture with Attention Mechanism for Image Captioning is also an Encoder / Decoder like classical Encoder / Decoder solution but with an additional layer, called as Attention Mechanism.

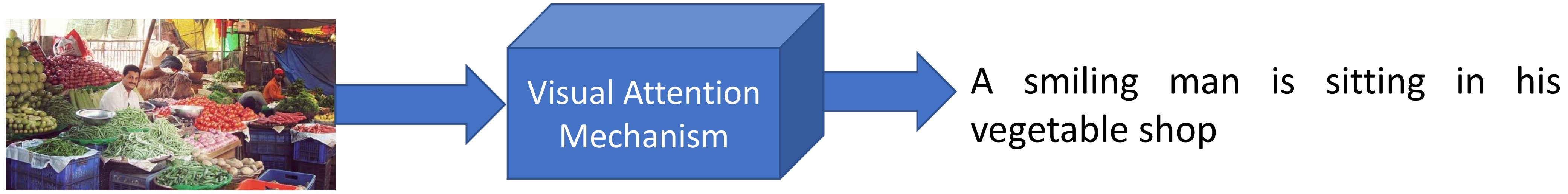
Attention Mechanism…

- Attention Mechanism was actually developed for Neural Machine Translation (i.e., translating text from one natural language to another). For this, Attention Mechanism gets text in one natural language and translate it into another natural language, as depicted below:


A dog is running fast over the beach.
Neural Machine Translation with Attention Mechanism
एक कुत्ता समुद्र तट पर तेजी से दौड़ रहा है।

Thus, in case of Neural Machine Translation, input and output, both are text.

- Whereas, in case of Image Captioning, input is an image and output is text. That's why Attention Mechanism was modified to take an image as input. This modified version of Attention Mechanism is known as Visual Attention Mechanism because it takes an image as input.

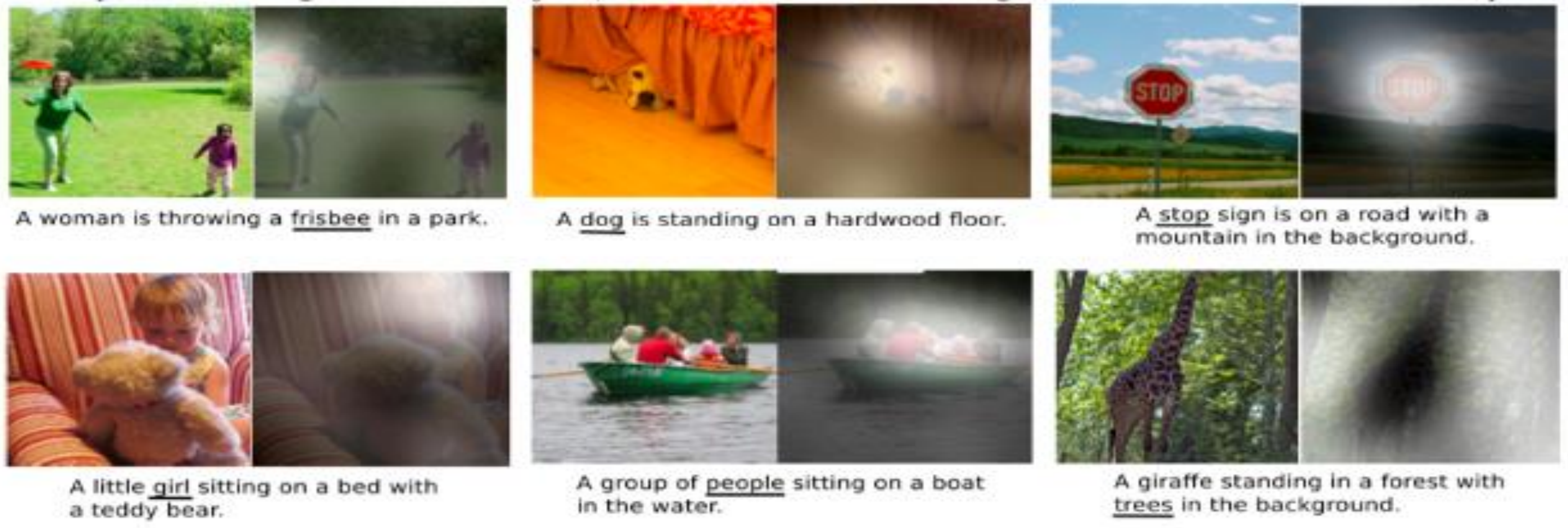

Visual Attention Mechanism
A smiling man is sitting in his vegetable shop

Attention Mechanism…

- With Attention Mechanism for Image Captioning, the image is first divided into n parts and we compute representation of each part (representation of each part is denoted by $h_1, h_2, h_3, \ldots, h_n$) by a CNN (Convolutional Neural Network).

  When the RNN (Recurrent Neural Network) is generating a new word, the Attention Mechanism focuses on the relevant part of the image. So, the decoder uses the specific parts of input image while generating a new (or next) word. Following is the example:

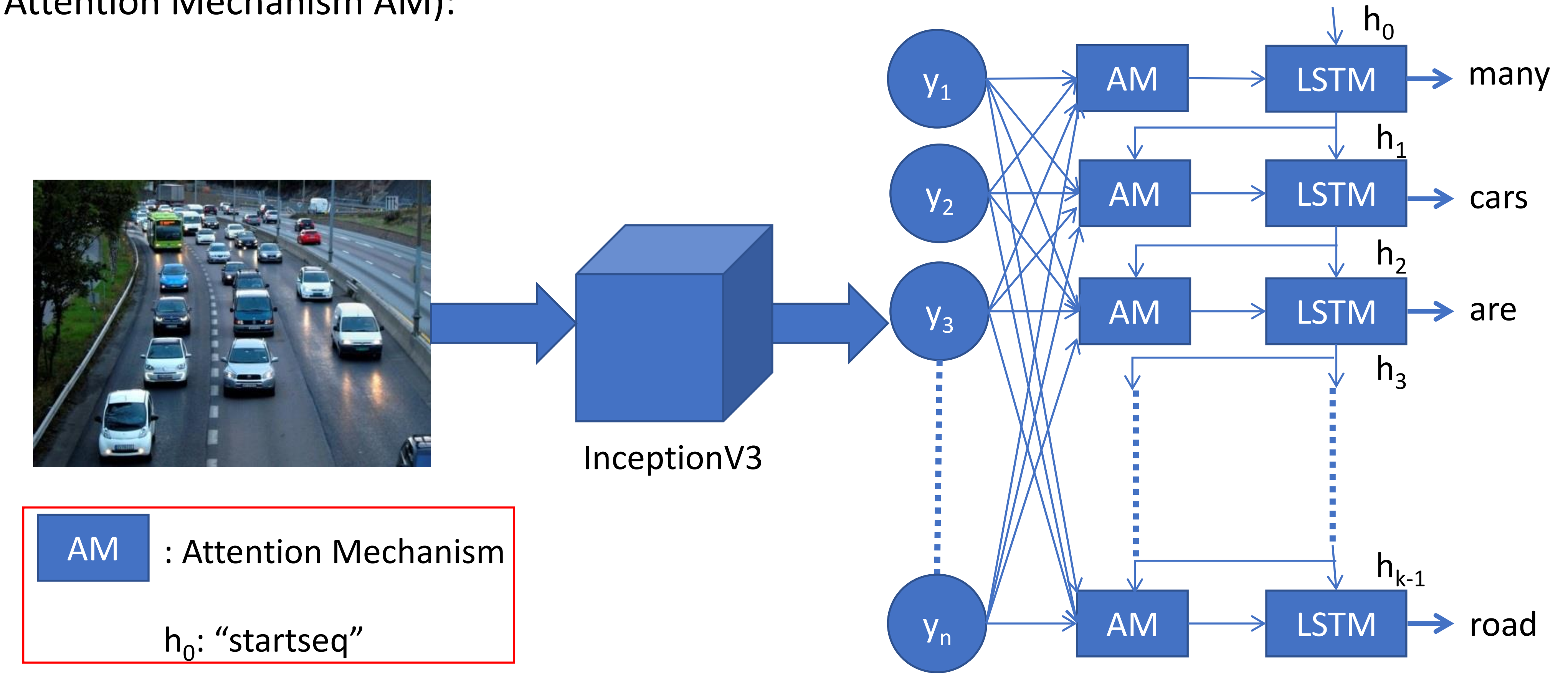
Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



A woman is throwing a frisbee in a park.
A dog is standing on a hardwood floor.
A stop sign is on a road with a mountain in the background.
A little girl sitting on a bed with a teddy bear.
A group of people sitting on a boat in the water.
A giraffe standing in a forest with trees in the background.

Attention Mechanism…

- Following is the architecture of our Neural Network with Attention Mechanism for Image Captioning (similar to Classical Encoder / Decoder architecture but with an additional layer of Attention Mechanism AM):



Attention Mechanism…

- Following is how this advanced solution works:

  If network has predicted i words, then the hidden state of LSTM would be h_i. This h_i would be passed to the AM (i.e., Attention Mechanism) which will select the relevant part of the image by using h_i as context and pass this relevant part of image (say, z_i) to LSTM to predict the next word which will make the hidden state of LSTM as h_{i+1}.
- There are two types of Attention Mechanism:
  1. Global Attention Mechanism (aka Luong's Attention): Attention is placed on all source position.
  2. Local Attention Mechanism (aka Bahdanau's Attention): Attention is placed only on a few source positions.
- Both the types of Attention Mechanisms differ from the classical Encoder / Decoder architecture only in the decoding phase due to presence of AM (i.e., Attention Mechanism layer).
- Both Global and Local Attention Mechanism differ in the way that they compute context vector (aka thought vector, i.e., the output of Encoder), can be represented by c(t).

Attention Mechanism…

- Global Attention (i.e., Luong's Attention) takes into consideration all encoder hidden states to derive the context vector c(t).

  This can be further simplified as in case of Neural Machine Translation, Global Attention focuses on all source side words to derive all target words. Similarly in case of Image Captioning, Global Attention considers all divided parts of input image to generate the caption (or textual description) for image.

  Thus, it is computationally very expensive and is impractical when translating for long sentences.
- Therefore, we will see implementation of only Local Attention (i.e., Bahdanau's Attention).
- Local Attention (or Bahdanau's Attention):

  Suppose, we have divided our input image into an input sequence of 5 parts. Now, before we start decoding, we first need to encode the input sequence into a set of internal states (h_1, h_2, h_3, h_4, h_5).

  Now, the next word in the output sequence is dependent on the current state of the decoder as well as on the hidden state of the encoder. Thus, at each time step, we consider these two things and follow the below steps:

Attention Mechanism…

We want our decoder to pay more attention to the states h_1 and h_2 (assume) while paying less attention to the remaining states of the encoder. For this reason, we train a feed forward neural network which will learn to identify relevant encoder states by generating a high score for the states for which attention is to be paid while low score for the states which are to be ignored.

Let s_1, s_2, s_3, s_4, s_5 be the scores generated for the states h_1, h_2, h_3, h_4, h_5. Since, we want to pay attention to h_1 and h_2 (assume) and ignore (h_3, h_4, h_5), thus s_1 and s_2 will be high while (s_3, s_4, s_5) are relatively low.

Once these scores are generated, we apply a softmax on these scores to produce the attention weights (e_1, e_2, e_3, e_4, e_5).

The advantage of applying softmax is as below:

1. All the weights lie between 0 and 1, i.e., (e_1, e_2, e_3, e_4, e_5) ϵ [0, 1].
2. All the weights sum to 1, i.e., $e_1 + e_2 + e_3 + e_4 + e_5 = 1$.

Suppose, following are the values of attention weights:

$e_1 = 0.75$ $e_2 = 0.2$ $e_3 = 0.02$ $e_4 = 0.02$ $e_5 = 0.01$

Thus, due to large values of e_1 and e_2, attention will be on h_1 and h_2, others (i.e., h_3, h_4, h_5) will be ignored due to small values of (e_3, e_4, e_5).

Attention Mechanism…

Now, we will compute the context vector (or thought vector) which will be used by the decoder in order to predict (or generate) the next word in the sequence:

$$context\ vector, c(t) = e_1 * h_1 + e_2 * h_2 + e_3 * h_3 + e_4 * h_4 + e_5 * h_5$$

Due to high values of e_1 and e_2, context vector (or thought vector) will have more information from the states h_1 and h_2, and relatively less information from the states h_3, h_4 and h_5.

Finally, the decoder will use the below two inputs to generate the next word in the sequence:

1. The context vector (or thought vector), c(t).
2. The output word generated from the previous time step.

Note that for the first time step, since there is no output from the previous time step, thus we use a special <startseq> token for this purpose.

The decoder then generates the next word in the sequence and along with the output, the decoder will also generate an internal hidden state, let's call it as d_1.

In order to generate the next word, the decoder will repeat the same procedure.

At the end, the decoder will either output <endseq> token or it would have generated (or predicted) words equal to maximum caption length, then we will stop the generation process.

Evaluation Metric

- Evaluating NLG system is a much more complicated task. There are following four evaluation metrics for evaluating a NLG system:
  1. Bilingual Evaluation Understudy (BLEU Score)
  2. Recall Oriented Understudy for Gisting Evaluation (ROUGE)
  3. Metric for Evaluation for Translation with Explicit Ordering (METEOR)
  4. Consensus based Image Descriptive Evaluation (CIDEr)
- Since above metrics differ mostly in terms of the way Precision and Recall (i.e., Sensitivity) calculated, thus we will first see how to calculate Precision and Recall (or Sensitivity) in NLG.
- In general,

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} = \frac{No.of\ correctly\ predicted\ positives}{Total\ no.of\ predicted\ positives}$$

$$Recall\ (or\ Sensitivity) = \frac{True\ Positive}{True\ Positive + False\ Negative} = \frac{No.of\ correctly\ predicted\ positives}{Total\ no.of\ actual\ positives}$$

Evaluation Metric…

- In NLG, predicted (or generated) text is called as Candidate text and the actual text is called as Reference text.
- Following is the definition of Precision and Recall (or Sensitivity) in NLG:

$$Precision = \frac{No. of\ words\ in\ Candidate\ matched\ with\ Reference}{Total\ no. of\ words\ in\ Candidate}$$

$$Recall\ (or\ Sensitivity) = \frac{No. of\ words\ in\ Candidate\ matched\ with\ Reference}{Total\ no. of\ words\ in\ Reference}$$

- Consider the following example:

  Reference: "I work on machine learning"

  Candidate A: "I work"

  Candidate B: "He works on machine learning"

$$Precision\ of\ Candidate\ A = \frac{2}{2} = 100\%$$

$$Recall\ (or\ Sensitivity) of\ Candidate\ A = \frac{2}{5} = 40\%$$

$$Precision\ of\ Candidate\ B = \frac{3}{5} = 60\%$$

$$Recall\ (or\ Sensitivity) of\ Candidate\ B = \frac{3}{5} = 60\%$$

Evaluation Metric…

- All previous calculations are done by using unigrams (i.e., no. of words, n = 1). These calculation can also be done by using bigrams (n = 2), trigrams (n = 3) and so on.
- Consider the following example:

  Reference: "I work on machine learning"
  Candidate A: "He works on machine learning"
  Candidate B: "He works on on machine machine learning learning"

  In case of unigram (i.e., n = 1):

$$Precision\ of\ Candidate\ A = \frac{3}{5} = 60\% \qquad Recall\ (or\ Sensitivity)\ of\ Candidate\ A = \frac{3}{5} = 60\%$$

$$Precision\ of\ Candidate\ B = \frac{6}{8} = 75\% \qquad Recall\ (or\ Sensitivity)\ of\ Candidate\ B = \frac{6}{5} = 120\%$$

- There is a modified n-gram scheme in which we match candidate's n-grams only as many times as they are present in any of reference text. Thus, "on", "machine" and "learning of Candidate B will get match only once in unigram (i.e., n = 1).

$$Precision\ of\ Candidate\ A = \frac{3}{5} = 60\% \qquad Recall\ (or\ Sensitivity)\ of\ Candidate\ A = \frac{3}{5} = 60\%$$

$$Precision\ of\ Candidate\ B = \frac{3}{8} = 37.5\% \qquad Recall\ (or\ Sensitivity)\ of\ Candidate\ B = \frac{3}{5} = 60\%$$

Evaluation Metric…

- To include all the n-gram precision scores (i.e., precision calculated by using unigram, bigram, trigram, etc.) in our final precision, we take their geometric mean. This is done because it has been found that precision decreases exponentially with n and we would require logarithmic averaging to represent all values fairly.

$$Precision = \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \qquad where, w_n = \frac{1}{n}$$

- Best Match Length: The problem with recall (or sensitivity) is that there may be many reference texts. So it is difficult to calculate the sensitivity of the candidate w.r.t a general reference. However, it is intuitive to think that a longer candidate text is more likely to contain a larger fraction of some reference than a shorter candidate.

  Therefore, we can introduce recall by just penalizing brevity (meaning: the state of being short or quick) in candidate texts. This is done by adding a multiplicative factor, called as Brevity Penalty (BP) with the modified n-gram precision as follows:

$$BP = \begin{cases} 1, & if\ c > r \\ \exp\left(1 - \frac{r}{c}\right), & otherwise \end{cases}$$

Evaluation Metric…

Where,

“c”: Total length of candidate translation corpus

“r”: The effective reference length of corpus, i.e., average length of all references

As the candidate length decreases, the ration $\frac{r}{c}$ increases, and the BP decreases exponentially.

Following is the formula of BLEU Score:

$$BLEU\ Score = BP.(Modified\ n - gram\ precision)$$

BLEU Score ϵ [0, 1].

BLEU is used for:

1. Neural Machine Translation (or simply Machine Translation)
2. Image Captioning
3. Text Summarization
4. Speech Recognition

BLEU Score can be directly used from the “nltk” library of Python:

```
import nltk.translate.bleu_score as bleu
bleu_sc = bleu.sentence_bleu(reference, candidate) # for one reference text
bleu_sc = bleu.corpus_bleu(reference, candidate) # for multiple reference text
```

Evaluation Metric…

- We have seen how to calculate modified n-gram precision for one reference. However, practically we have multiple references. Thus, let us see how to calculate it for multiple references:

  Candidate 1: It is a guide to action which ensures that the military always obeys the commands of the party.

  Reference 1: It is a guide to action that ensures that the military will forever heed party commands.

  Reference 2: It is the guiding principle which guarantees the military forces always being under the command of the party.

  Reference 3: It is the practical guide for the army always to heed the directions of the party.

- We will calculate following:
  1. Count: Count the maximum number of times a candidate n-gram occurs in the candidate.
  2. Ref1 Count, Ref2 Count and Ref3 Count: For each reference sentence, count the number of times a candidate n-gram occurs.
  3. Max Ref Count: Take the maximum number of n-grams occurrences in reference count.
  4. Count Clip: Take the minimum number of Count and Max Ref Count.

     Count Clip = min(Count, Max Ref Count)
  5. Divide the Clipped Count by the total unclipped number of candidate n-grams to get the modified precision score (p_n).

Evaluation Metric…

Candidate n-gram	Count	Ref1 Count	Ref2 Count	Ref3 Count	Max Ref Count	Count Clip
“It”	1	1	1	1	1	1
“is”	1	1	1	1	1	1
“a”	1	1	0	0	1	1
“guide”	1	1	0	1	1	1
“to”	1	1	0	1	1	1
“action”	1	1	0	0	1	1
“which”	1	0	1	0	1	1
“ensures”	1	1	0	0	1	1
“that”	2 (1)	2	0	0	2	2 (1)
“the”	3	1	4	4	4	3
“military”	1	1	1	0	1	1
“always”	1	0	1	1	1	1
“obeys”	0 (1)	0	0	0	0	0
“commands”	1	1	0	0	1	1
“of”	0 (1)	0	1	1	1	0 (1)
“party”	1	0	0 (1)	1	1	1
18						17

Evaluation Metric…

Applying step 5 (calculating Modified Precision Score, p_n):

$$p_n = \frac{17}{18}$$

Modified n-gram Precision Score (p_n) captures:

1. Adequacy: A candidate using the same words as in the references tends to satisfy adequacy.
2. Fluency: The long n-gram matches between candidate and reference account for fluency.

$$Brevity\ Penalty, BP = \begin{cases} 1, & if\ c > r \\ exp\left(1 - \frac{r}{c}\right), & otherwise \end{cases}$$

Where,

"r": Count of words in reference. "c": Count of words in candidate.

$$BLEU = BP.\exp\left(\sum_{n=1}^{N} w_n \log p_n\right)$$

Where,

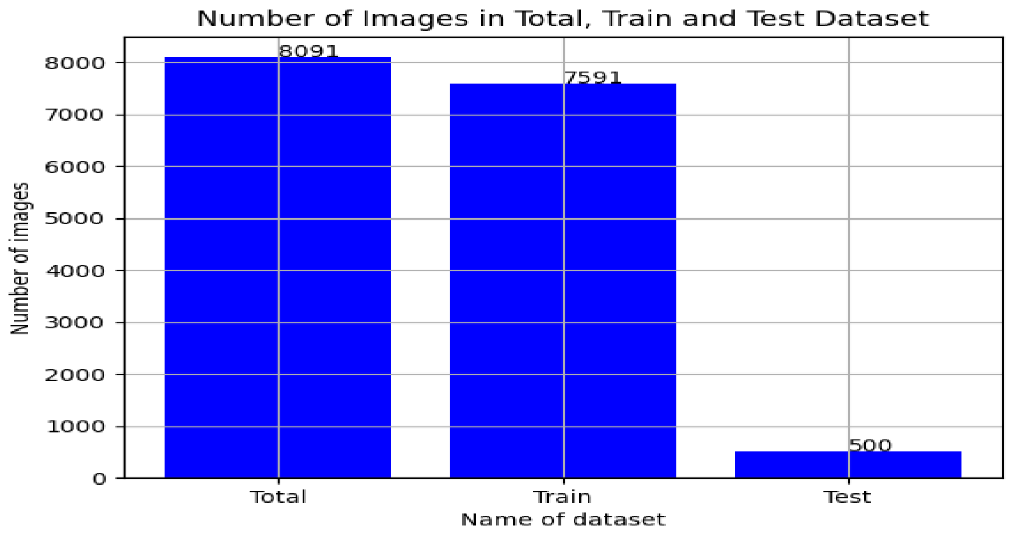
N: No. of n-grams, we usually use unigram, bigram, trigram, 4-gram.

$w_n = \frac{1}{N}, by\ default\ N = 4$ and p_n: Modified Precision Score

Scripts Execution Flow

1. Script “scripts/check_training_val_test.py” does following tasks:
   1. Verifies that the name of images given in training and testing .txt files (i.e., Flickr_8k.trainImages.txt and Flickr_8k.testImages.txt) are in captions.txt file or not.
   2. It also creates a file Flickr8k.valImages.txt which has name of images that are not in training and not in testing .txt file (i.e., Flickr_8k.trainImages.txt and Flickr_8k.testImages.txt). These images can be used for validation purpose, as the name of file suggests.
   3. At last, this script generates and saves following plot which summarizes the number of images in the entire dataset; training, validation and testing subsets.
2. Script “scripts/segregate_train_val_test.py” does following tasks:
   1. This script reads image filenames from Flickr_8.trainImages.txt, Flickr_8k.valImages.txt and Flickr_8k.testImages.txt; extracts these filenames and their captions (5 captions per file) from captions.txt;
   2. Since we need more data to train Attention Mechanism in order to get acceptable performance, the complete training data along with complete validation and half of testing data is combined to use for training. Only remaining half of test data is used for testing.
   3. Then, this script generated two csv files: train_image_caption.csv and test_image_caption.csv having two columns: ‘image’ and ‘caption’. ‘image’ column has image names and ‘caption’ columns has all 5 captions of corresponding image separated by ‘<>’.
   4. At last, this script generated a plot showing below:

Scripts Execution Flow…



3. Script "scripts/preprocessing_train.py" does following tasks:
   - i. It reads train_image_caption.csv file. Extracts image filenames and their five captions (joint by "<>").
   - ii. It takes out each caption of each image file and perform these operations: converts into lower case, removes all special characters, removes all single characters (like 'a', 's', etc.) and removes numerical figures (such as '1', '2', etc.). This part is well known as Data Cleaning or Data Pre-processing.
   - iii. Then, it puts "<startseq>" and "<endseq>" before and after each processed caption, join these processed captions by "#" and save it with its image filename in file train_image_caption_processed.csv. "<startseq>" and "<endseq>" are special tokens.
   - iv. Along with this, it also saves following files: max_caption_length.txt, vocabulary.txt & WordFreq.csv.

Scripts Execution Flow…

4. Script “scripts/preprocessing_test.py” does following tasks:
   i. This script reads test_image_caption.csv file. Extracts image filenames & their five captions (joint by “<>”).
   ii. It takes out each caption of each image file and perform these operations: converts into lower case, removes all special characters, removes all single characters (like ‘a’, ‘s’, etc.) and removes numerical figures (such as ‘1’, ‘2’, etc.). This part is well known as Data Cleaning or Data Pre-processing. This entire processing will be done by referring vocabulary.txt file as words not in vocabulary will be deleted from captions.
   iii. Then, it joins processed captions by “#” and save them with their image filename in file test_image_caption_processed.csv. “<startseq>” and “<endseq>” special tokens are not required for test data because we have to directly match with generated captions which will not have “<startseq>” and “<endseq>” tokens.
5. Script “scripts/gen_image_features.csv” does following tasks:
   i. This script loads each image which is in train_image_caption_processed.csv, resize it for the pre-trained model to generate bottleneck features (here, we have used InceptionV3, discussed in later slides).
   ii. Then, it passes each of these images through our chosen pre-trained model (here it is InceptionV3) and generates bottleneck feature of dimension (8, 8, 2048). Then, script is reshaping it to (64, 2048) and saving it as a npy file. Script has done this task in batches of 64 images. In this manner, it has saved 7,591 npy files (since we have 7591 file names in train_image_caption_processed.csv file).

Scripts Execution Flow…

6. Script “scripts/training.py” or “scripts/training.ipynb” does following tasks:
   i. This script is accessing following paths:
      a) “images_npy_path”: This path has 7591 npy files for each of 7591 images. Each of these npy files have (64 x 2048) dimensional bottleneck feature generated by using IncpetionV3 model in earlier steps.
      b) “img_caption_csv_path”: This is the path of file “train_image_caption_processed.csv” file. This file contains name of 7591 images in one column and their 5 captions in another column. So, the dimension of this file (7591, 2). All captions are cleaned (or processed) in earlier steps.
      c) “vocabulary_path”: This is the path of “vocabulary.txt” file. This file has all unique words of our captions.
      d) “max_caption_len_path”: This is the path of file “max_caption_length.txt” file. This file has the maximum length of any caption found.
      e) “checkpoint_path”: This is the path where this script will save the trained model.
   ii. The first thing that this script does is loading data of train_image_caption_processed.csv file. This file has path of all 7591 images and their 5 captions.
   iii. After this, script will load data of “vocabulary.txt” file. Along with that, script has also created “wordtoix” (“word-to-index”) dictionary type variable.

Scripts Execution Flow…

iv. Then, script is loading "max_caption_length.txt" file. This file has the length of longest caption, i.e., maximum length of any caption. Script will extract this value from this file and store it in "max_caption_len" variable (an int type variable).

v. Now, script will replace all captions by their indices by using "wordtoix" (i.e., word-to-index) dictionary type variable. In this way, all textual captions will become a numerical vectors. For instance:
   Caption: "one boy is walking on street"
   Equivalent numerical vector: [4928, 880, 3813, 8136, 4925, 7192]
   All these indices are taken from vocabulary.txt file. These indices are actually (line number – 1) of these words in vocabulary.txt file.

vi. Since it is the need of algorithm to have all numerical vectors of captions to be equal, thus we will pad zeros in all these numerical vectors to make them equal in length. For this, we will grow the length of each numerical vector by padding zeros until their length is lesser than max_caption_len (an int type variable explained above). Thus, suppose the value of max_caption_len is 10, then we have to add 4 zeros in the above example, i.e.,:
   Caption: "one boy is walking on street"
   Equivalent numerical vector: [4928, 880, 3813, 8136, 4925, 7192], length: 6
   Padded numerical vector: [4928, 880, 3813, 8136, 4925, 7192, 0, 0, 0, 0], length: 10

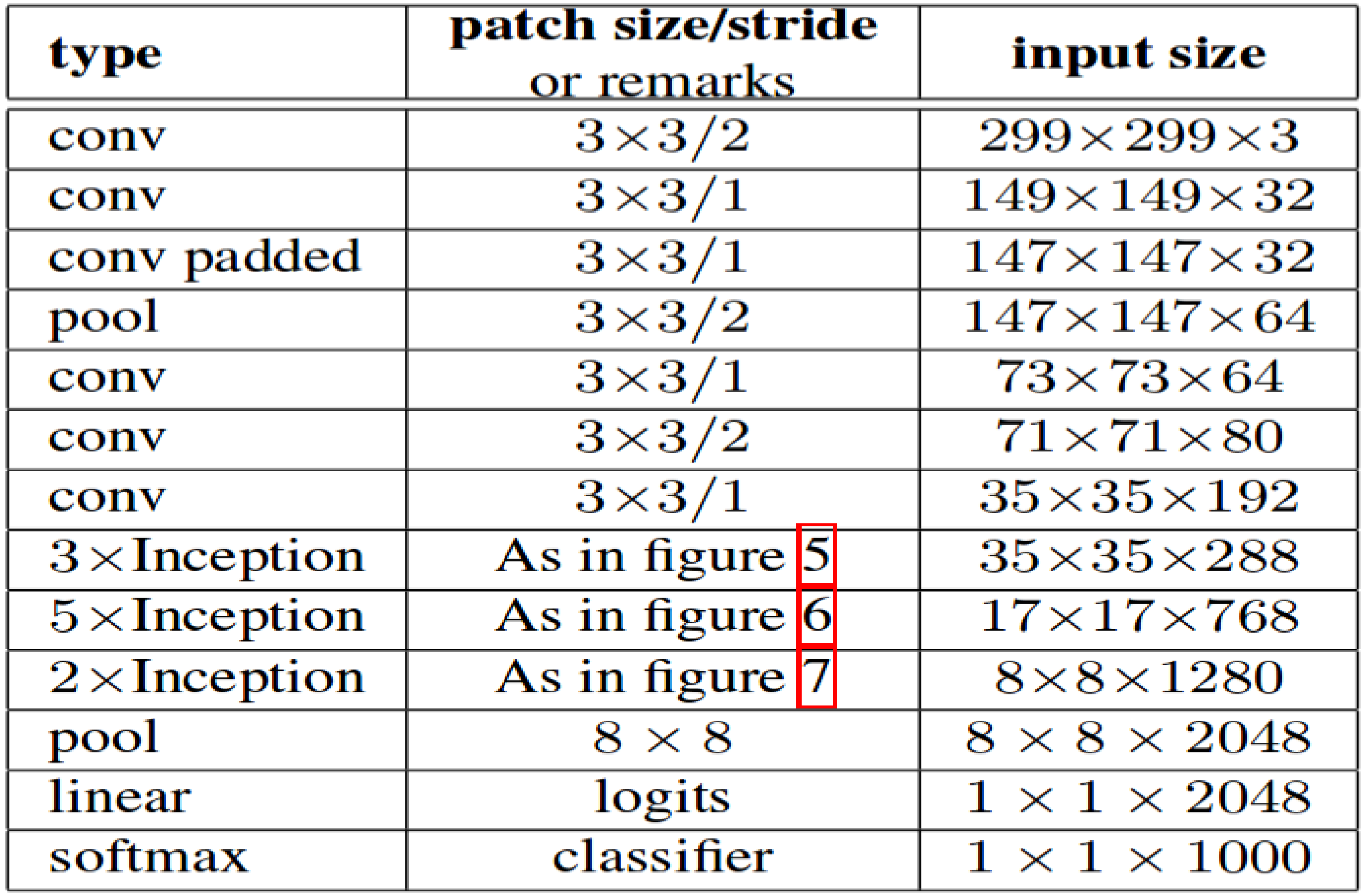
Scripts Execution Flow…

vii. After this, script has created a dataset variable which will pass data in batches during training runtime. This will prevent “MemoryOverflow” error.
viii. Script has defined “Adam” optimizer
ix. Then, it has defined “SparseCategoricalCrossentropy” as the loss function.
x. After this, script has created object of class “CNN_Encoder” and “RNN_Decoder” and called them “encoder” and “decoder”.
xi. Script has created checkpoint object to save the entire network after (or during) training.
xii. Finally, script will start training in a for-loop.

7. After completion of training, one can run any of the following scripts to see the performance of the model:
   i. Script “scripts/inference_on_single_instance.py”: This script outputs the caption for a single input image. User has to change the path of image to be tested in the script. This path is mentioned at the top of the script. Thus, user can change this path and the script will pass this image through the model and the model will generate the caption for the input image on the terminal.
   ii. Script “scripts/inference_on_bulk_data.py”: This script outputs captions for a bulk data, i.e., more than one test image. User has to change the path “test_images_path” variable to make it to point to their test data. Script will pass this test data through model, generate captions and save all generated captions with their image name in a file “test_data_predicted_captions.csv” file under “./output/generated_captions/” directory.

Pre-Trained Models

- We have used only one pre-trained model to generate bottleneck features of images. This model is InceptionV3:

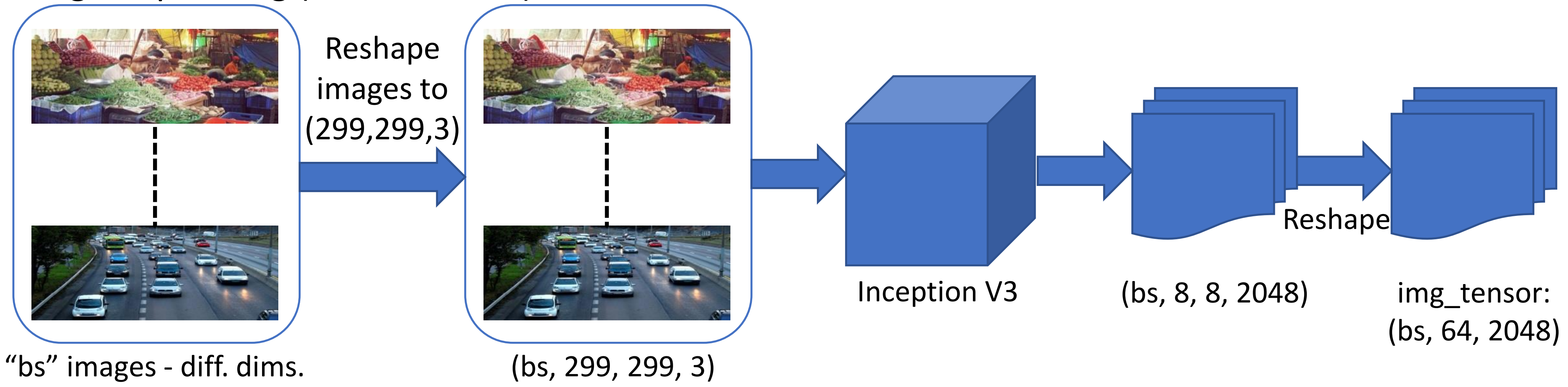


type	patch size/stride or remarks	input size
conv	$3\times3/2$	$299\times299\times3$
conv	$3\times3/1$	$149\times149\times32$
conv padded	$3\times3/1$	$147\times147\times32$
pool	$3\times3/2$	$147\times147\times64$
conv	$3\times3/1$	$73\times73\times64$
conv	$3\times3/2$	$71\times71\times80$
conv	$3\times3/1$	$35\times35\times192$
$3\times$Inception	As in figure $\boxed{5}$	$35\times35\times288$
$5\times$Inception	As in figure $\boxed{6}$	$17\times17\times768$
$2\times$Inception	As in figure $\boxed{7}$	$8\times8\times1280$
pool	8×8	$8\times8\times2048$
linear	logits	$1\times1\times2048$
softmax	classifier	$1\times1\times1000$

Output of this layer is taken as bottleneck feature

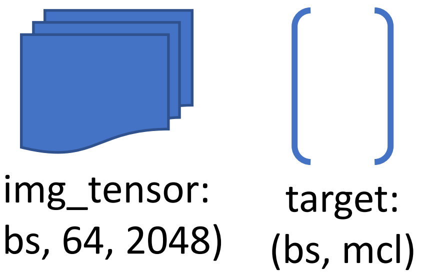
Training & Neural Networks Specific

- Neural Network Framework: Tensorflow (version: 2.4.1)
- Other training / neural – network related details:
  - Epochs: 20
  - Batch Size: 64 (i.e., 64 image-caption pairs)
  - Loss Function: "SparseCategoricalCrossentropy"
  - Optimization Function: Adam
- Pre-trained model used: InceptionV3 to generate (8 x 8 x 2048) dimensional bottleneck feature. These (8 x 8 x 2048) bottleneck features are reshaped to (64 x 2048) dimension and then saved.
- Following is the snapshot of the entire Neural Network with Visual Attention Mechanism for Image Captioning (bs: batch size):


Reshape images to (299,299,3)
"bs" images - diff. dims.
(bs, 299, 299, 3)
Inception V3
(bs, 8, 8, 2048)
Reshape
img_tensor: (bs, 64, 2048)

Training & Neural Networks Specific…

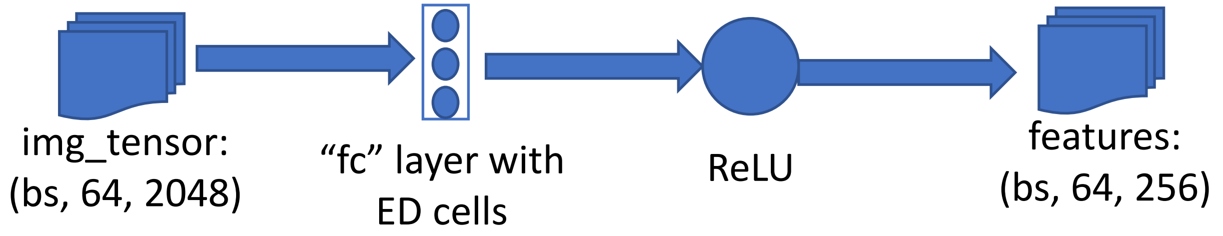
Input:



- Here, bs is "batch size" and mcl is "maximum caption length".
- "target" has indices (int values) of words of captions, padded with zeros to make these vectors equal to max_caption_length.

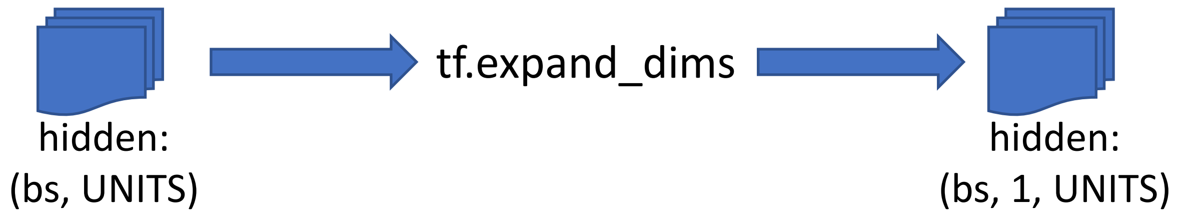
bs: "BATCH_SIZE"; mcl (maximum caption length)= 34; UNITS= 512; EMBEDDING_DIM (ED)= 256, vs (vocabulary size)= 8511;

1. Reset RNN_Decoder state to zeros, called it "hidden", its dimension will be (bs, UNITS).
2. Here, input to RNN_Decoder is called as "decoder_input". The first input to RNN_Decoder will be "<startseq>". It will be a numerical vector having index of "<startseq>" for all instances in batch, thus its dimension will be (bs, 1).
3. Pass img_tensor to Encoder for encoding the meaningful features generated by InceptionV3 to be suitable for image captioning. Following it is depicted:



Training & Neural Networks Specific…

4. Run a for-loop from 1 to mcl (maximum caption length):



$$\textit{attention\_hidden\_layer} = \textit{tf.nn.tanh}(\underbrace{\underbrace{\underbrace{W1(\textit{features})}_{(bs,\ 64,\ 256)}}_{(bs,\ 64,\ 512)} + \underbrace{\underbrace{W2(\textit{hidden\_with\_time\_axis})}_{(bs,\ 1,\ UNITS)}}_{(bs,\ 64,\ 512)}}_{(bs,\ 64,\ 512)})$$

score = V(attention_hidden_layer); shape of *score*: (bs, 64, 1)
attention_weights = tf.nn.softmax(score); shape of *attention_weights*: (bs, 64, 1)
*context_vector = attention_weights * features;* (bs, 64, 1) * (bs, 64, 256) → (bs, 64, 256)

Training & Neural Networks Specific…

context_vector = tf.reduce_sum(context_vector); (bs, 64, 256) → (bs, 256)

decoder_input → *embeddings* → *decoder_input*
(bs, 1) (vs, ED)=(8511, 256) (bs, 1, 256)

x = tf.concat(context_vector, decoder_input); concat((bs, 256), (bs, 1, 256)): (bs, 1, 512)

$$\underset{(bs,\ 1,\ 512)}{x} \rightarrow \underset{(UNITS = 512))}{GRU} \rightarrow \begin{cases} output: (bs,\ 1,\ 512) \\ state: (bs,\ 512) \end{cases}$$

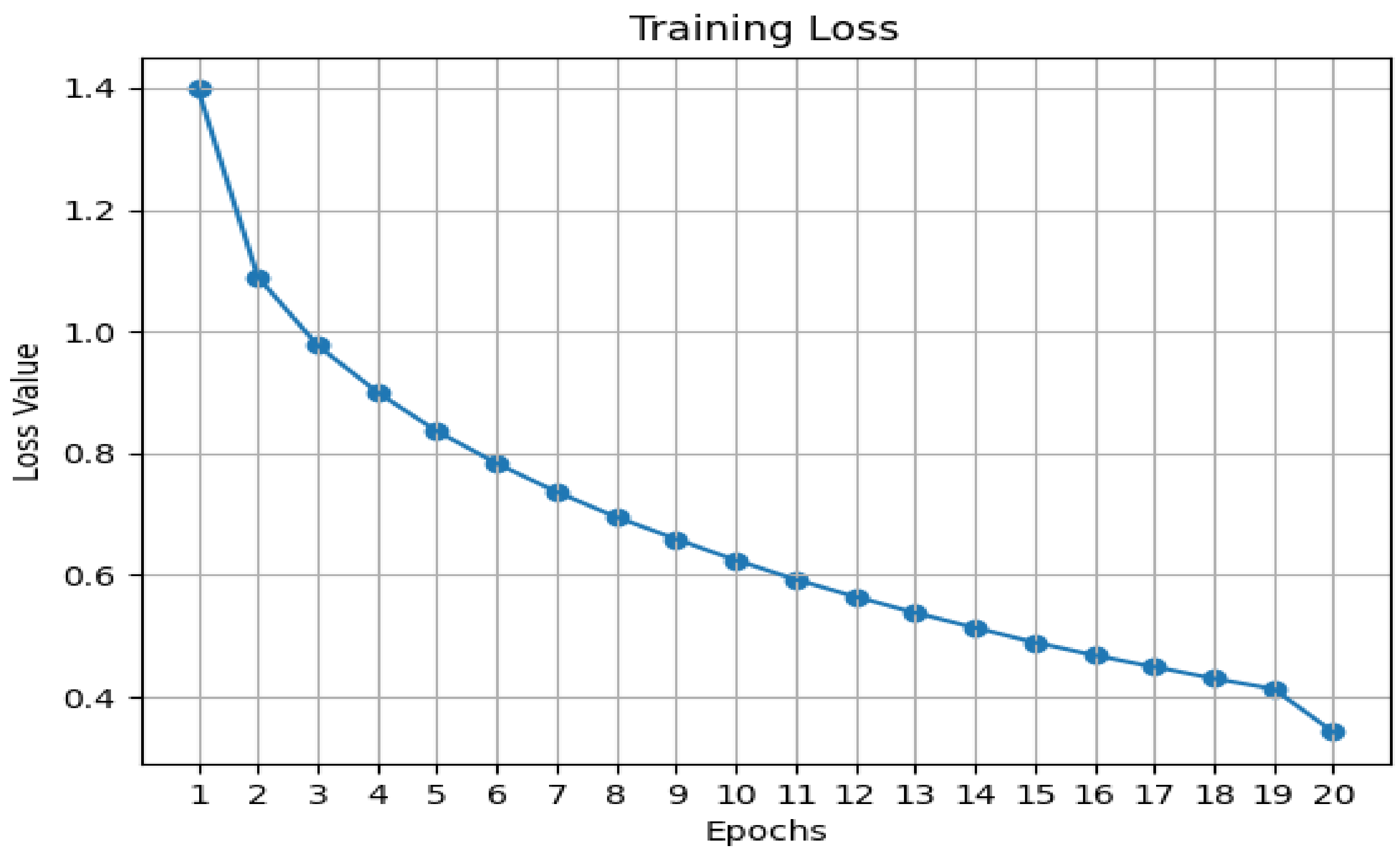
$$\underset{(bs,\ 1,\ 512)}{x} \rightarrow \underset{(UNITS = 512)}{fc1} \rightarrow \underset{(bs,\ 1,\ 512)}{x} \rightarrow \text{RESHAPE} \rightarrow \underset{(bs,\ 512)}{x}$$

$$\underset{(bs,\ 512)}{x} \rightarrow \underset{(vs = 8511)}{fc2} \rightarrow \underset{(bs,\ vs) = (bs,\ 8511)}{x}$$

Now, ‘x’ is prediction made by model and state is “hidden”.
Update: dec_input = numerical vector for next word from “target”
Iterate again over step 4 until for-loop terminates & then over all training instances in the same manner.

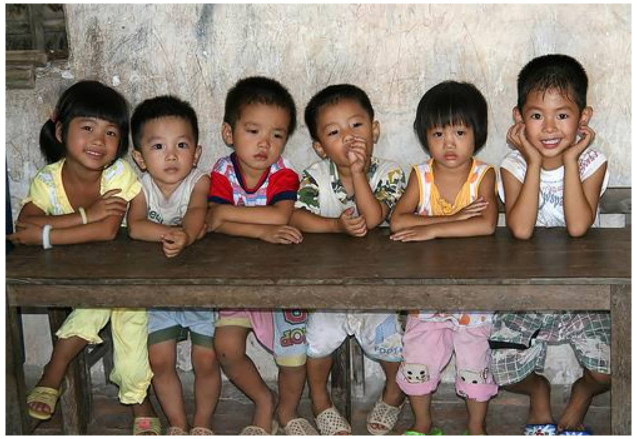
Training & Neural Networks Specific…

- Following is the plot of training loss:

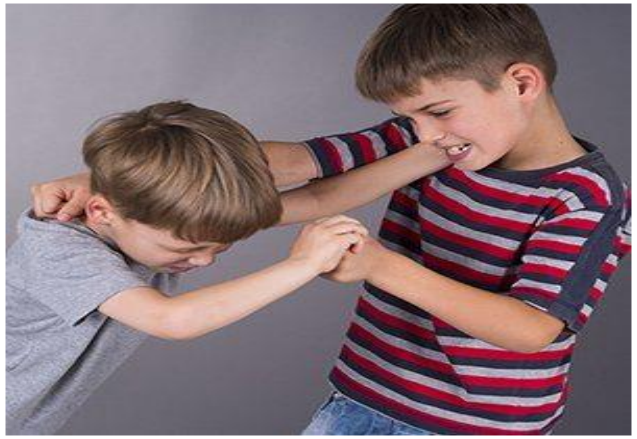


Result

- Following are some results obtained from model:

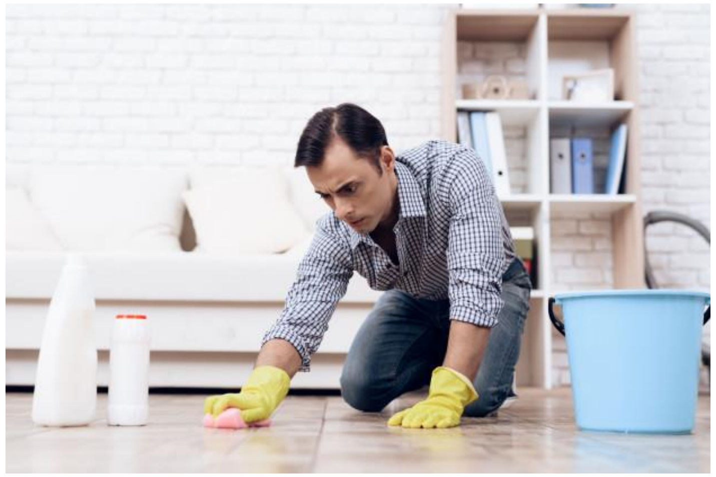


six children sitting at their picture

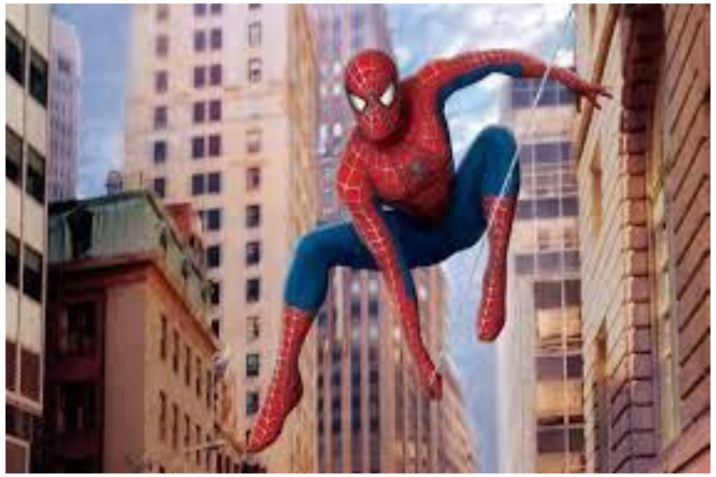


father watching baby on the man

Result…



boy is sitting on the water



man with backpack and another on the railing in the background

Result…

- Trained model is tested on half of test images, i.e., 500 test images. Each of these 500 test images has 5 captions written by different people.
- Model generated a caption for each image.
- Since, it has become a kind of document where each reference has five texts and candidate has only one text, thus corpus bleu score is used to measure the performance.
- Obtained Corpus BLEU Score is:

0.029157590163286912

Conclusion

- Since model is trained for less number of epochs and also on insufficient data, thus the BLEU Score is quite less on validation images. However, if we see model's performance on a single image, then it seems that the model has learnt something during training and it verifies the technique used here for Image Captioning.

Future Work

- In this neural network code, we are developing embeddings for each word of vocabulary during training. In order to improve the performance (or bleu score) of model, we can try word embeddings of any pre-trained model, such as GloVe (i.e., Global Vector) model.
- Training on huge dataset, such MSCOCO, the largest open source dataset for Image Captioning, in order to improve BLEU Score of the model.

Template

- Template

Thank You