Image Caption Generator

Introduction:

Using deep learning techniques, today we can solve many tasks which were previously known to be unsolvable or very complex to be framed for computer systems. With today's computing power and deep learning frameworks available, tasks such as object detection, sentiment analysis, natural language translation and many more are possible.

In in this project, I have tried to solve one very interesting problem i.e., generating captions for the input image. We humans tend to identify and provide suitable description of any image just by looking at the image, but this trivial looking task is very complex to handle for any computing device but using deep learning frameworks such CNN (Convolutional Neural Network) and LSTM-RNN (Long Short-Term Memory – Recurrent Neural Network) we can frame this task to be solvable.

For this project I have used **CNN and LSTM-RNN** with most widely used **Flicker8k dataset** containing **8091 images** in total with **captions in separate .txt file.**

For this task I have used a high-performance computing device available on Kaggle.

Train-images = 7000

Test-images = 1091

Max length in available captions(MDL) = 31

Literature Review:

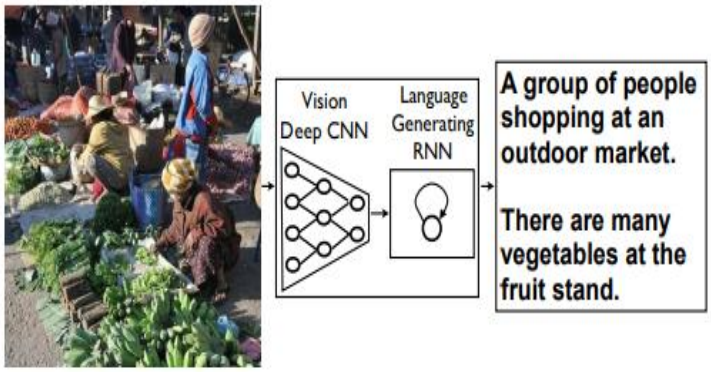
Paper -1:

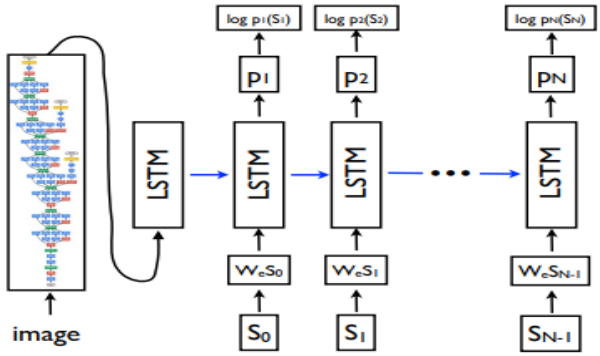
https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Vinyals_Show_and_Tell_2015_CVPR_paper.html

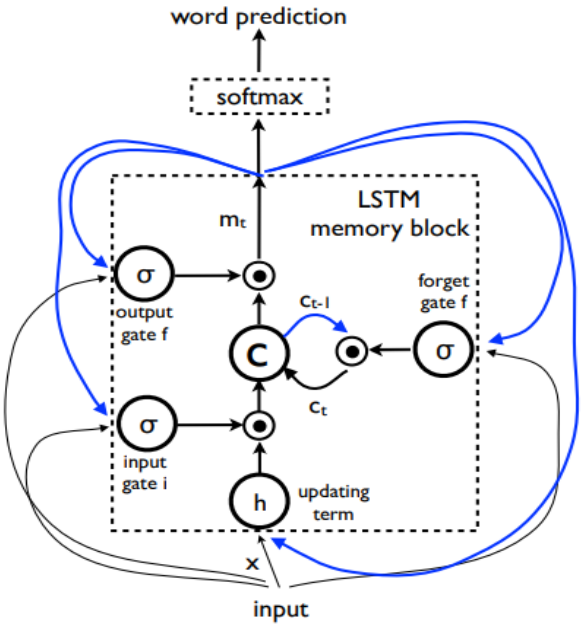
Title: *"A Neural Image Caption Generator"*

Abstract:

Using CNN architecture to fetch a feature vector from input image and then using LSTM-RNN on datasets ranging from small to large datasets.







Approach	PASCAL (xfer)	Flickr 30k	Flickr 8k	SBU
Im2Text [24]				11
TreeTalk [18]				19
BabyTalk [16]	25			
Tri5Sem [11]			48	
m-RNN [21]		55	58	
MNLM [14][5]		56	51	
SOTA	25	56	58	19
NIC	**59**	**66**	**63**	**28**
Human	69	68	70	

Table 2. BLEU-1 scores. We only report previous work results when available. SOTA stands for the current state-of-the-art.

Given By :

Oriol Vinyals	Alexander Toshev	Samy Bengio	Dumitru Erhan
Google	Google	Google	Google
vinyals@google.com	toshev@google.com	bengio@google.com	dumitru@google.com

Year: 2015

Paper -2:

https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3368837

Title: *"Visual Image Caption Generator Using Deep Learning"*

Abstract:

Used VGG(Visual Geometric Group) 16 layer pre-trained network for feature extraction. VGG16 is available in tensorflow.keras.application package, the extracted features then go through dropout layer with 0.5 dropout and then a dense layer of 256 nodes. Extracted features has input size of (4096), dropout(0.5), dense(256).

For training a caption generation model, it used a LSTM layer. First we get tokenized captions as input then a embedding layer with shape(mdl, 256) which will provide mapping of word -> vec then a dropout(0.5) layer after that a LSTM layer accepting input in (34, 256) and providing a 256 dimension output.

Output from both models then added and passed through two dense layer one which output 256 dimensional vector and another shrink it down to mdl.

Mdl(max. description length) here is 34.

Results:

BLEU -1 (1.0, 0, 0, 0)

BLEU -2 (0.5, 0.5, 0, 0)

BLEU -3 (0.33, 0.33, 0.33, 0)

BLEU -4 (0.25, 0.25, 0.25, 0.25)

Given By:

Grishma Sharma
Asst. Professor of Department Of Computer Engineering
K.J Somaiya College Of Engineering, Mumbai
neelammotwani@somaiya.edu

Priyanka Kalena
Department Of Computer Engineering
K.J Somaiya College Of Engineering, Mumbai
kalenapriyanka@gmail.com

Nishi Malde
Department Of Computer Engineering
K.J Somaiya College Of Engineering, Mumbai
nshmalde97@gmail.com

Aromal Nair
Department Of Computer Engineering
K.J Somaiya College Of Engineering, Mumbai
aromaln31197@gmail.com

Saurabh Parkar
Department Of Computer Engineering
K.J Somaiya College Of Engineering, Mumbai
saurabh.parkar@somaiya.edu

Year : 2019

Paper -3:

http://ijcrt.org/papers/IJCRT_196552.pdf

Title: "*Image Caption Generator Using CNN and LSTM*"

Abstract:

This uses a Xception model which is pre-trained imagenet model in keras and extension of inceptionV3 model. Here the Xception model is used to get output features vectors then using LSTM layers to generate captions.

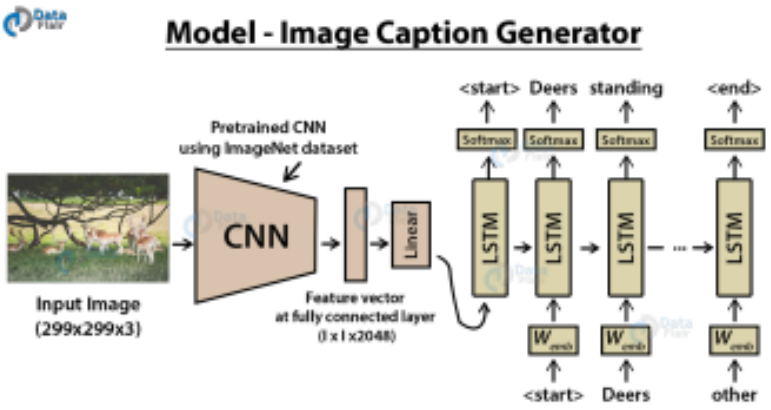


Fig.7 CNN-LSTM model

Given By:

Swarnim Tripathi
swarnim0711@gmail.com
Galgotias University,India

Ravi Sharma
ravi.sharma@galgotiasuniversity.edu.in
Galgotias University,India

Paper -4:

https://arxiv.org/abs/1308.0850v5

Title : "*Generating Sequences With Recurrent Neural Networks*"

Abstract:

This demonstrates how a deep recurrent neural network works in predicting a sequence and also proposes LSTM framework.

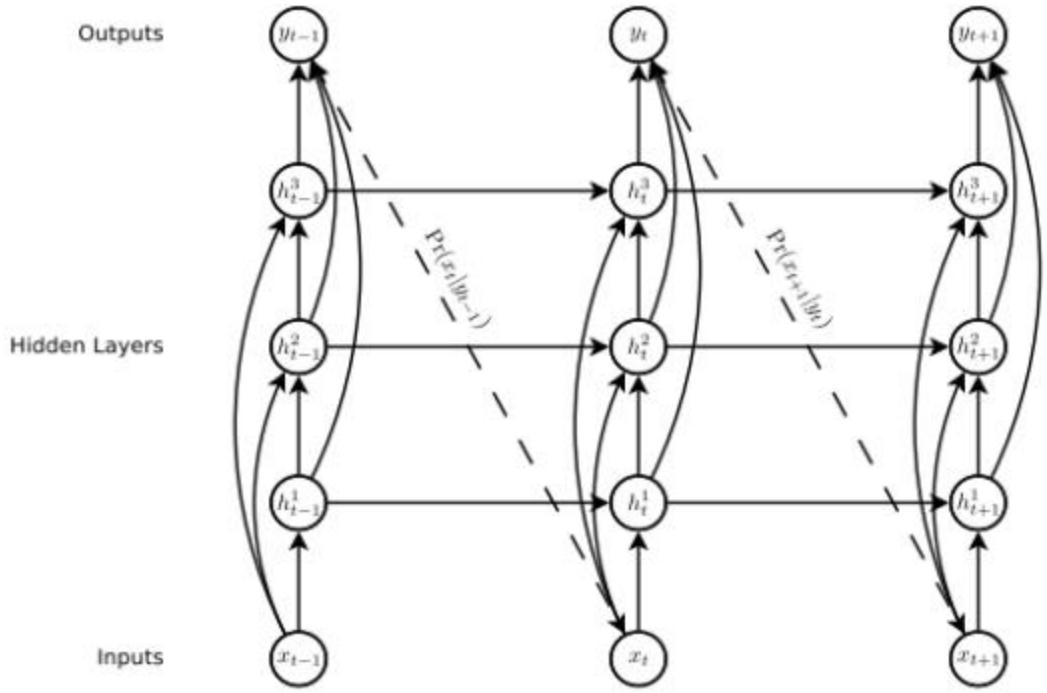
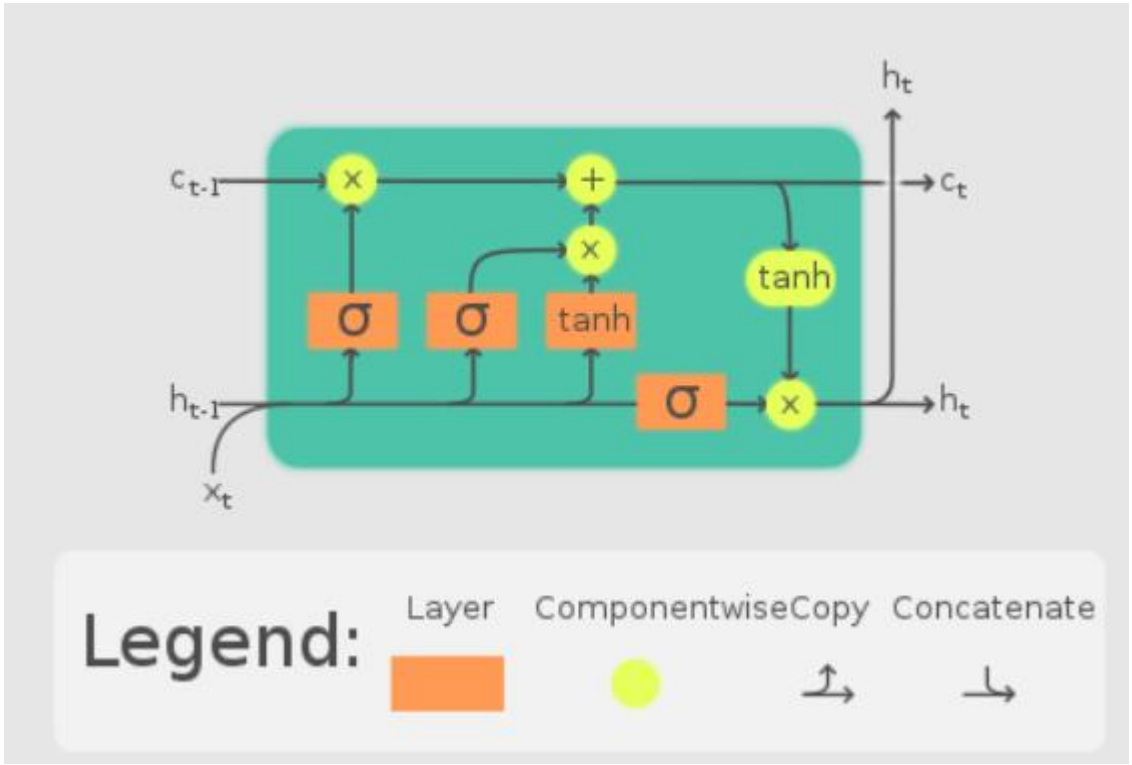


Figure 1: **Deep recurrent neural network prediction architecture.** The circles represent network layers, the solid lines represent weighted connections and the dashed lines represent predictions.



Given By:

Generating Sequences With Recurrent Neural Networks

Alex Graves
Department of Computer Science
University of Toronto
graves@cs.toronto.edu

Year: 2013

Paper -5:

http://www.aclweb.org/anthology/P02-1040.pdf

Title: *"BLEU: a Method for Automatic Evaluation of Machine Translation"*

Abstract:

This paper suggested a metric for evaluating the predictions made by automatic machine translation systems known as "Bilingual Evaluation Understudy Score".

According to BLEU:

"The primary programming task for a BLEU implementor is to compare n grams of the candidate with the n-grams of the reference translation and

count the number of matches. These matches are position-independent.

The more the matches, the better the candidate translation is."

Given By:

Kishore Papineni, Salim Roukos, Todd Ward, and **Wei-Jing Zhu**
IBM T. J. Watson Research Center
Yorktown Heights, NY 10598, USA
{papineni,roukos,toddward,weijing}@us.ibm.com

Year : 2002

Dataset used:

Name	Flicker8k
Total Images	8091
Total Captions	8091
Training Split	7000
Testing Split	1091
source	Kaggle

Name	GLOVE
Total words	6 billion
Dimension	200
source	Kaggle

Flicker8k dataset is most widely used for object detection purposes and in most of the image captioning research paper that

I seen so far, It also have a extension where 30k images are provided for training large image-nets but that would require very high computing power to be executed.

Glove is global vector for word representation which is unsupervised algorithm developed by Stanford researchers for word embedding. Embedding is important as each token must be represented in vectorized form.

For embedding we can train our own Embedding layer but then we have large number of parameters to train on and likely to give sub-par results if not trained properly, so to save time and resource I have used pre-trained Glove embeddings.

Name of glove file is *"glove.6B.200d.txt"* . We can use glove algorithm to find embeddings in our dataset if pre-trained ones are not providing better results but that would be very rare case as pre-trained one contains very vast library of English words. Training GLOVE locally also come with the requirement very large dataset and computing power.

For inputting data to model, there are is typical format:

X1, X2, y :

X1 => Image feature map

X2 => Input sequence

Y => Output sequence

x1(feature vector)	x2(Text sequence)	y(word to predict)
feature	start,	two
feature	start, two	dogs
feature	start, two, dogs	drink
feature	start, two, dogs, drink	water
feature	start, two, dogs, drink, water	end

As in the image, we need to segregate the caption in such a way that model has to predict every other word in sequence given previous word,a type of greedy search.

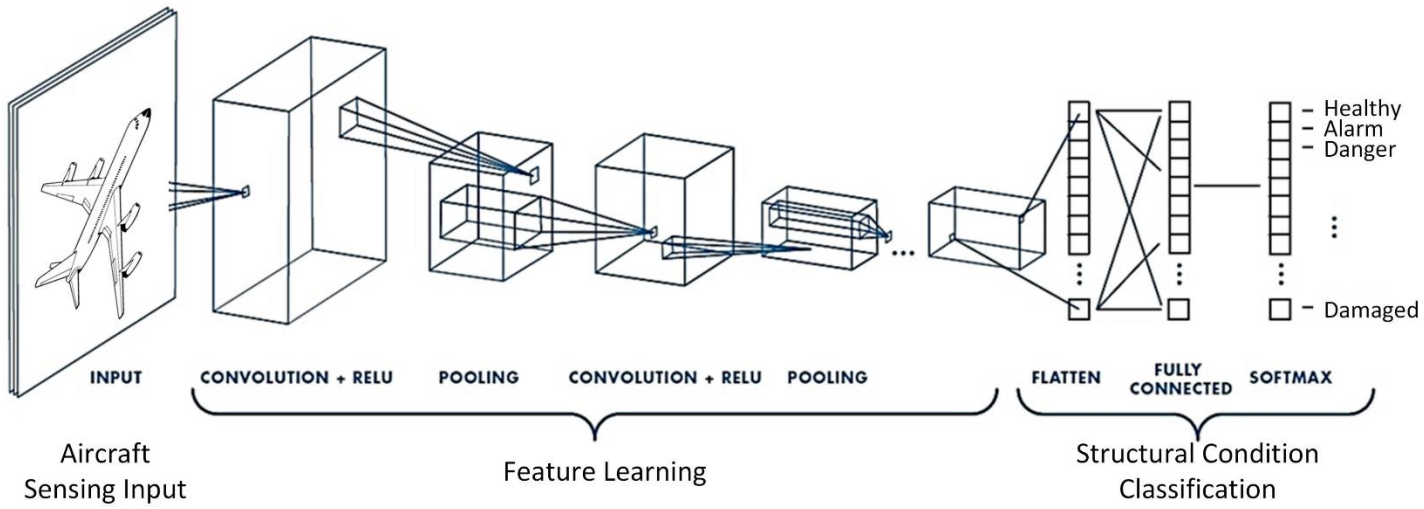
Proposed Model Design:

Understanding about CNN:

A convolution neural network (CNN) is basically used to extract features from images and then image classification. CNNs are specialized in the way that they apply several filters that convolve around the image to extract fine details hidden in image.

Filters in image can be of varying functions such as a filter to detect all vertical edges in image, a filter to detect diagonal edges, horizontal edges and many more.

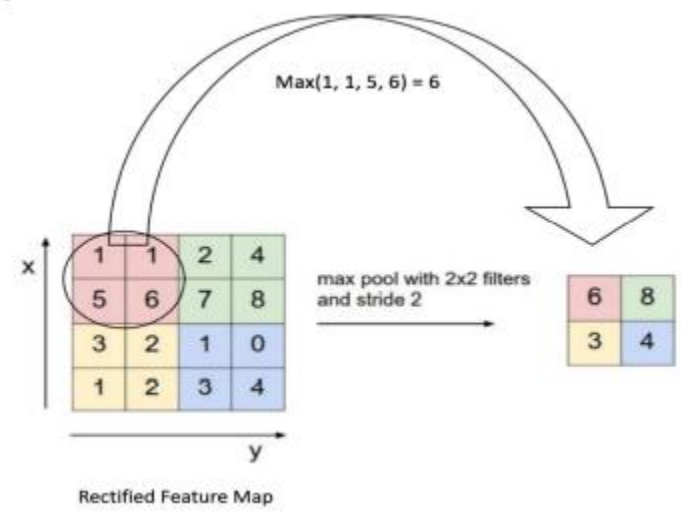
Example:



As in image we can see that input image is being convolved by a small filter of size say 3x3, then many of such filters will be applied and then RELU activation layer will be applied, function of RELU layer is to introduce a non-linearity.

Pooling layer is used to down sample extracted feature maps in the form of filters by applying small filters over each and using either max or average from filter.

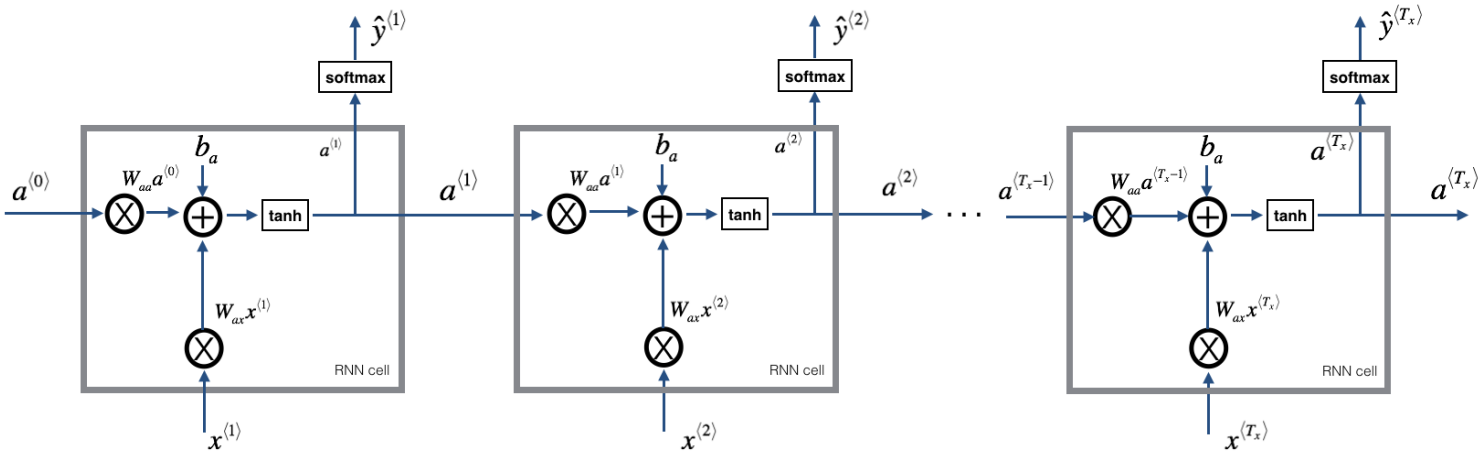
Example of pooling:



We can stack many more CNN and pooling layers together to get deep feature extraction model.

Understanding about LSTM:

Recurrent Neural Nets (RNN) are a form of sequence generating model in deep neural nets. A RNN looks something like this



Here x(1) , x(2) – x(T) are input sequence and yhat(1), yhat(2) – yhat(T) are generated seequences, Wa and Wx are weights applied to previous output sequence and current input sequence.

As we can observe we have reccurence relation here i.e

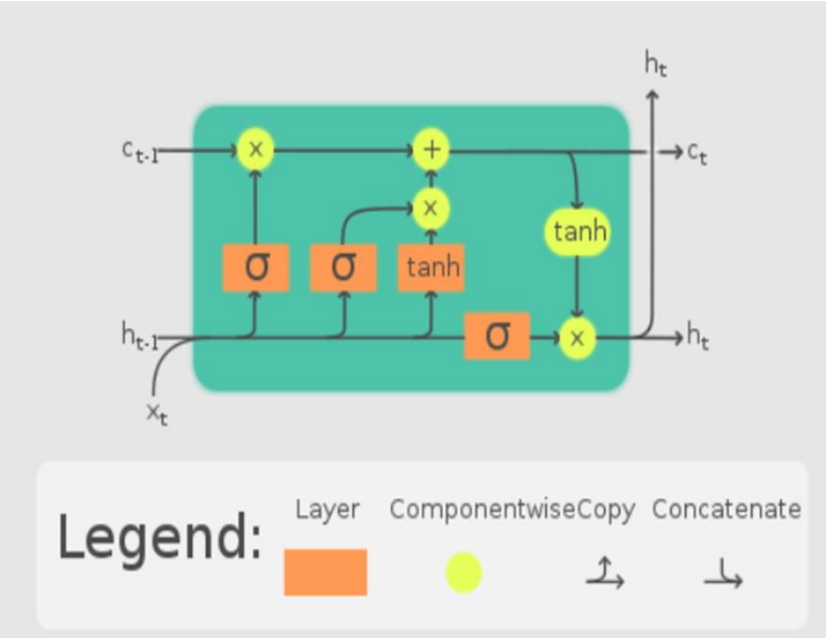
Yhat(t) = softmax(tanh(Wa(a(t-1) +Wa(x(t)) + ba)))

ba is bias added.

This picture shows have feedforward with time is executed in RNN but to update the weights namely Wa and Wx we need backpropogate with time and that's we use gradients to optimize these weights.

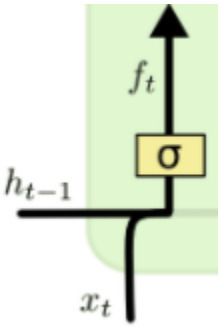
One common problem with rnn is that as length sequence grows the RNN scales with it and with large RNN the number of gradient operations in backpropagation grows and we are left with Vanishing gradient problem.

LSTM are form of gated RNNs which are used to solve Vanishing gradient problem in RNN. A typical cell structure in LSTM is:



It has 4 gates:

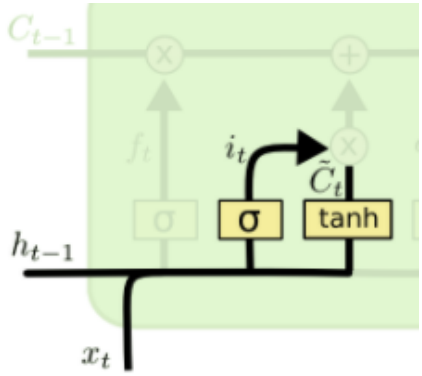
Forget gate:



This gate forget any previous input information that is no longer in

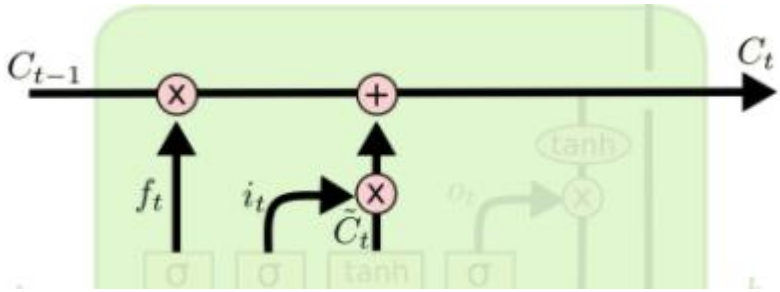
Context with current input sequence.

Update gate:



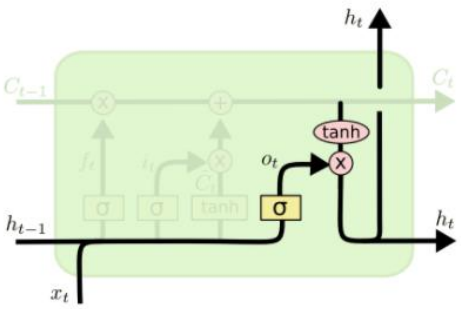
This gate is used to add new information received from current input into the model.

Memory gate:



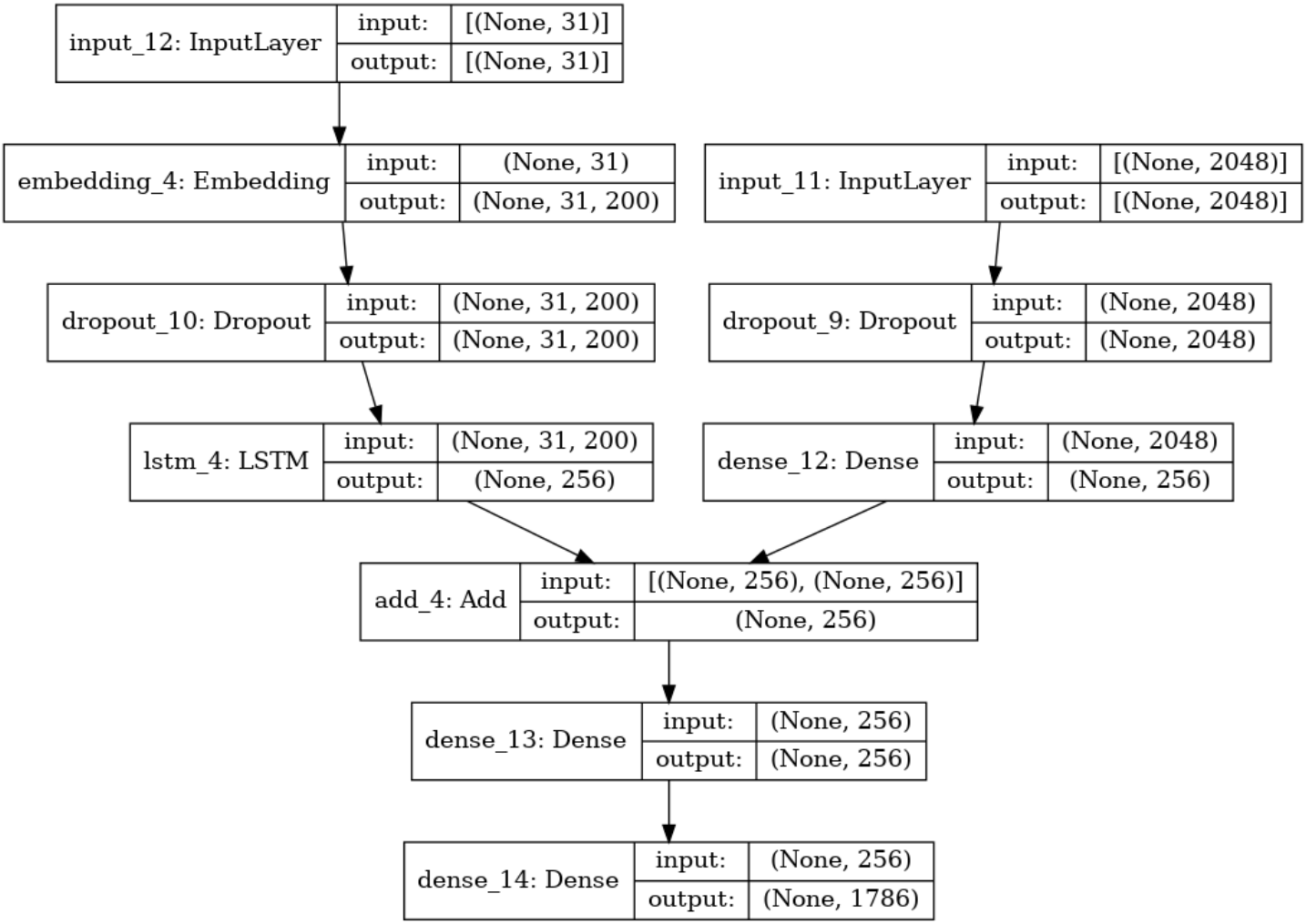
This is where old unwanted information gets rejected and new information get added to the language model.

Output gate:



Output the new generated sequence using all the changes made in the model

Model Design:



CNN model I used here is inception_v3 available in tensorflow.keras.application.inception_v3. Model accepts input image of size (299, 299, 3).

Using concept of transfer learning, here I am using this model for feature extraction so in my model I will prefer not to engage in fully connected layer in the inceptionV3 model.

Output will be taken from model.layers[-2].output which will output of 2048 dimensional vector representing features of input image.

For captions I first extracted vocabulary of unique words available in caption corpus of images and then after filtering vocabulary using threshold filter the task of tokenization is done.

For tokenization, a simple mapping of each word in vocab to an integer number is done.

For each tokenized word GLOVE word embedding is used to find it's corresponding vector in "*glove.6b.200d.txt* "file. An embedding matrix is generated using GLOVE of size (vocab_size, 200).

Embedding layer parameters will be set to embedding matrix and then it will freeze as we don't need to train it.

Using add function and dense layers we can get out features and encoded sequences mapped to final generated text.

After add function, there are 2 dense layers:

Dense layer 1: Nodes=256, activation=RELU

Dense layer 2: Nodes=vocab_size, activation=softmax

Loss used in model is "*categorical_crossentropy"* and optimizer used is "*adam"*.

Training:

```
Model: "model_4"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_9 (InputLayer)            [(None, 31)]         0
__________________________________________________________________________________________________
input_8 (InputLayer)            [(None, 2048)]       0
__________________________________________________________________________________________________
embedding_3 (Embedding)         (None, 31, 200)      357200      input_9[0][0]
__________________________________________________________________________________________________
dropout_6 (Dropout)             (None, 2048)         0           input_8[0][0]
__________________________________________________________________________________________________
dropout_7 (Dropout)             (None, 31, 200)      0           embedding_3[0][0]
__________________________________________________________________________________________________
dense_9 (Dense)                 (None, 256)          524544      dropout_6[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   (None, 256)          467968      dropout_7[0][0]
__________________________________________________________________________________________________
add_3 (Add)                     (None, 256)          0           dense_9[0][0]
                                                                 lstm_3[0][0]
__________________________________________________________________________________________________
dense_10 (Dense)                (None, 256)          65792       add_3[0][0]
__________________________________________________________________________________________________
dense_11 (Dense)                (None, 1786)         459002      dense_10[0][0]
==================================================================================================
Total params: 1,874,506
Trainable params: 1,517,306
Non-trainable params: 357,200
__________________________________________________________________________________________________
..

 7000/7000 [==============================] - 623s 89ms/step - loss: 4.0297
    1/7000 [..............................] - ETA: 9:23 - loss: 3.7998
/opt/conda/lib/python3.7/site-packages/keras/utils/generic_utils.py:497: CustomM
m mask layer must be passed to the custom_objects argument.
  category=CustomMaskWarning)
 7000/7000 [==============================] - 617s 88ms/step - loss: 3.4090
 7000/7000 [==============================] - 620s 89ms/step - loss: 3.2403
 7000/7000 [==============================] - 620s 89ms/step - loss: 3.1552
 7000/7000 [==============================] - 614s 88ms/step - loss: 3.1010
 7000/7000 [==============================] - 608s 87ms/step - loss: 3.0616
 7000/7000 [==============================] - 609s 87ms/step - loss: 3.0366
 7000/7000 [==============================] - 613s 88ms/step - loss: 3.0171
 7000/7000 [==============================] - 609s 87ms/step - loss: 3.0014
 7000/7000 [==============================] - 607s 87ms/step - loss: 2.9940
 7000/7000 [==============================] - 613s 88ms/step - loss: 2.9843
 7000/7000 [==============================] - 619s 88ms/step - loss: 2.9809
 7000/7000 [==============================] - 618s 88ms/step - loss: 2.9703
 7000/7000 [==============================] - 606s 87ms/step - loss: 2.9718
 7000/7000 [==============================] - 608s 87ms/step - loss: 2.9681
 7000/7000 [==============================] - 604s 86ms/step - loss: 2.9652
 7000/7000 [==============================] - 611s 87ms/step - loss: 2.9627
 7000/7000 [==============================] - 613s 88ms/step - loss: 2.9615
 7000/7000 [==============================] - 614s 88ms/step - loss: 2.9648
```

Results:

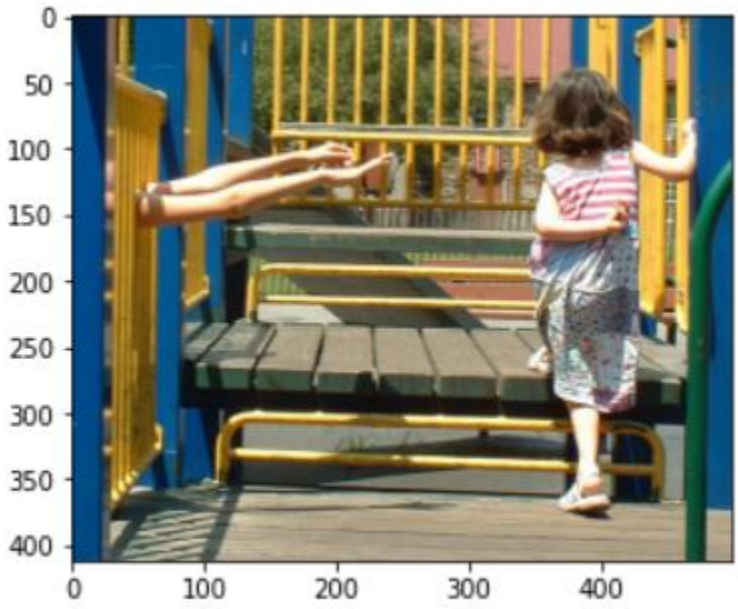
Xception + LSTM:

```
Dataset: 1091
Descriptions: test=1091
Photos: test=1091
BLEU-1: 0.461176
BLEU-2: 0.183539
BLEU-3: 0.104814
BLEU-4: 0.034413
```

Inception + LSTM:

```
{'BLEU-1': 0.4195933456561922,
 'BLEU-2': 0.22323135649222947,
 'BLEU-3': 0.14684539739415145,
 'BLEU-4': 0.06508638206111438}
```

Example:



two children are playing on trampoline

Conclusion:

Overall model perform not bad on test set but performance on general examples shows that more training and data can improve performance by a lot but that comes at expense of high-computing devices available for long hours which is hard to manage.