International Journal of Recent Technology and Engineering (IJRTE)
ISSN: 2277-3878 (Online), Volume-11 Issue-1, May 2022

Image Captioning using Convolutional Neural Networks and Long Short Term Memory Cells

Hitoishi Das

Abstract: ***This paper discusses an efficient approach to captioning a given image using a combination of Convolutional Neural Network (CNN) and Recurrent Neural Networks (RNN) with Long Short Term Memory Cells (LSTM). Image captioning is a realm of deep learning and computer vision which deals with generating relevant captions for a given input image. The research in this area includes the hyperparameter tuning of Convolutional Neural Networks and Recurrent Neural Networks to generate captions which are as accurate as possible. The basic outline of the process includes giving an image as input to the CNN which outputs a feature map. This feature map is passed as input to the RNN which outputs a sentence describing the image. The research in image captioning is relevant because this method demonstrates the true power of the encoder-decoder network made up of Convolutional Neural Network and Recurrent Neural Network and potentially will open many pathways for further interesting research on different types of neural networks.***

Keywords: Captioning, Deep Learning, Encoders, Decoders, Convolutional Neural Networks, Recurrent Neural Networks, Computer Vision

I. INTRODUCTION

Deep neural networks have come a long way in establishing new boundaries and benchmarks in the realm of deep learning. Topics like image recognition and facial recognition to self-driving cars, deep learning and neural networks have come a long way in not only smashing known boundaries of science and technology but also solve problems which exist in the society.

The research on deep learning will never end because the potential and opportunities presented are limitless. One of the mostly used types of neural network is the Convolutional Neural Network [1]. These neural networks are broadly used for classification purposes, where given an input, the network classifies the input into one of the classes based on probabilistic calculations.

Generally, convolutional neural networks take in an image as input or any other object which does not consist of any sequence or pattern which can influence the classification task. The input image is then passed through something called as filters and then finally the image classified into one of the multiple classes where the image is associated with separate probabilities with each class. The training process of a convolutional neural network includes kernel convolution [1]. Kernel convolution is the process of passing a small matrix of numbers over the input image and transform the values of the image based on the convolution created, these small matrices are called convolutional filters.

The basic mathematical equation involved when passing a convolutional filter over an image is like so: -

$$G[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k] \quad (1)$$

Where $G[m,n]$ represents the resultant 2D feature map of the image. Literals h represents the kernel and f represents the input image. These filters generate feature maps, and these maps are passed through multiple layers of the convolutional neural network.

Coming to the second type of neural networks which I propose to use, these neural networks are called Recurrent Neural Networks [4]. The main difference between a convolutional neural network and a recurrent neural network is the fact that while convolutional neural networks do not store information of earlier time steps, recurrent neural networks rely on information persisted from previous time steps to classify a sequence or a pattern and generate a similar sequence or a pattern, broadly put, any RNN relies on previous time steps and the current state of a RNN can be expressed as follows: -

$$\bar{s}_t = \varphi(\bar{x}_t W_x + \bar{s}_T W_s) \quad (2)$$

where $T = t - 1$. Here W_x represents the weight matrix of the input connected with the state layer and W_s represents the weight matrix connecting the state from previous timestamps to the state in the following timesteps. Image captioning in its most basic sense refers to generating appropriate and relevant captions to a given input image. The approach involves the use of two of the most important components in the realm of deep learning, namely convolutional neural networks, and recurrent neural networks. The Convolutional Neural Network is trained on the CoCo Dataset [6]. The CoCo dataset consists of image-caption pairs which later helps the Convolutional Neural Network to output a vector which is then encoded and fed into a Recurrent Neural Network. The training happens in the form of an encoder decoder pattern. The Convolutional Neural Network assumes the role of the encoder and the Recurrent Neural Network with Long Short Term Memory(LSTM) cells [3] takes upon the role of the decoder. The encoder generates a latent space vector for an image given as input to the encoder and the generated latent space vector for a given image is passed to

Manuscript received on 03 January 2022
Revised Manuscript received on 18 April 2022
Manuscript published on 30 May 2022
* Correspondence Author
Hitoishi Das*, Department of Computer Science and Engineering, ICFAI Foundation for Higher Education, Hyderabad (Telangana), India. Email: hitoishi.das@gmail.com

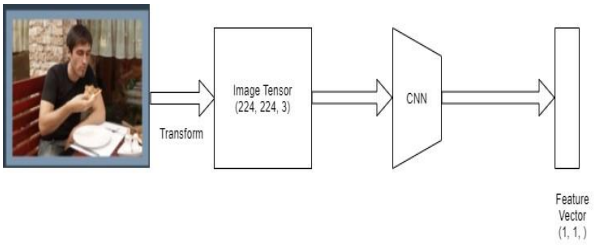

Fig. 1. Flow diagram of encoders

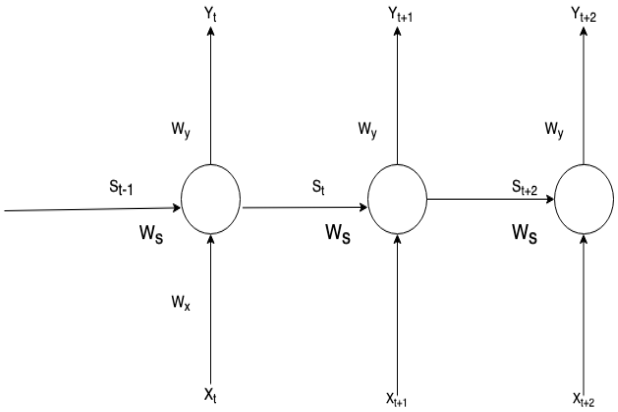

Fig. 2. Unfolded view of a Recurrent Neural Network

the decoder (the Recurrent Neural Network with Long Short Term Memory cells) as an encoding which needs to be mapped into words put together into a sentence. The encoder outputs the latent feature vector or feature map by passing the image through multiple layers, with each layer having convolutional filters for the same. The Recurrent Neural Network processes the feature vector by storing spatial information of the image in every timestep and then producing a sequence of words by connecting all the previous timesteps as per Equation 2

II. LITERATURE SURVEY

A. Recurrent Neural Networks

As we saw earlier, any Recurrent Neural Network, in a nutshell, can be represented mathematically like so:-

$$\bar{s}_t = \varphi(\bar{x}_t W_x + \bar{s}_T W_s) \quad (3)$$

The above equation represents the current state, which depends on the current input and the previous states. φ is an activation function which is generally a *tanh* function. To represent the current output of a RNN mathematically we have

$$\bar{y}_t = \sigma(\bar{s}_t W_y) \quad (4)$$

Where σ represents the activation function used to get the output. Apart from mathematical representations of the RNN, there are two main types of pictorial representations for any RNN, namely the folded and the unfolded model. The unfolded model gives us a brilliant insight into what happens to a RNN under the hood.

In the preceding model (Fig. 2) of the unfolded model of the RNN, *s* represents the state of the RNN in each timestep ($t-1$ represents one timestep before the current state). The

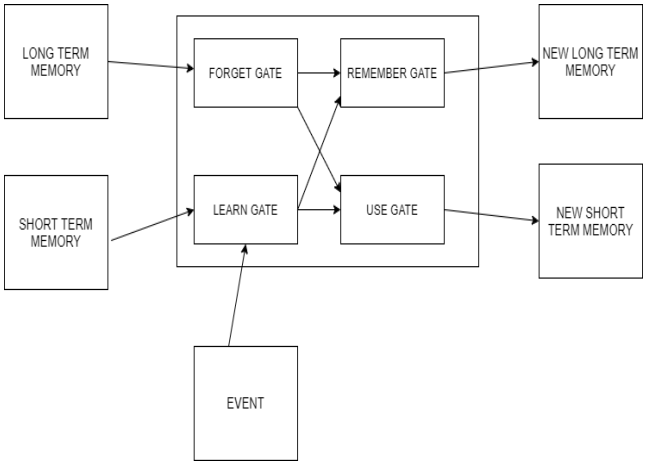

Fig. 3. A typical gated representation of a LSTM in action.

notation W_y represents the weight matrix connecting the states to the outputs at each timestep. The notation W_s refers to the weight matrix which connects the input *x* to the states *s* for every timestep. Each step in the previous timestep is passed on to the current state and the flow direction continues towards newer timesteps thus helping the RNN achieve considerable number of patterns in sequential data. The gradient calculation in the training process of the RNN is as follows: -

$$\frac{\delta E_N}{\delta W_y} = \frac{\delta E_N}{\delta \bar{y}_N} \frac{\delta \bar{y}_N}{\delta \bar{s}_i} \frac{\delta \bar{s}_i}{\delta W_y} \quad (5)$$

This gradient calculation over multiple timesteps produces a major problem for RNNs, which will be discussed in the next section

B. Long Short Term Memory Cells

When the gradient calculation as per *Equation 5* goes on for more than roughly 10 timesteps, the term

$$\frac{\delta E_N}{\delta W_y} \quad (6)$$

approaches zero and we are presented with a classical problem in neural networks called as the vanishing gradient problem [2]. There have been many approaches to counter this but the most effective way so far is LSTM or Long Short Term Memory Cells [3] [4]. The main fighting point for the existence of LSTM cells in the realm of recurrent neural networks is the fact that the core functionality of these cells is" latching" on to data in the states in the hidden layer of the RNN and stemming the gradient decay so that the gradient term does not approach zero easily. Each state in the hidden layer of the RNN has the LSTM embedded in it thus effectively countering the problem of vanishing gradients at least on paper. Every LSTM cell primarily consists of four gates, namely Forget Gate, Learn Gate, Remember Gate and Use Gate. Arbitrarily speaking, the architecture of a typical LSTM cell can be represented diagrammatically as shown in figure 3. Referring to the gated representation of a LSTM (Figure 3), the Forget Gate selectively remembers data from the Long Term Memory accumulated over previous timesteps. The Learn Gate learns everything it can from the Short Term

Fig. 4. Example of an image-caption instance from the CoCo dataset.

Memory accumulated over previous timesteps and combines the learned knowledge with the events occurring in the context of the Short Term Memory. The knowledge gained by the Forget Gate and the Learn Gate together are combined into the Remember Gate which is the new Long Term Memory. Similarly, the Learn Gate and the Forget Gate together are combined to form the Use Gate which is the new Short Term Memory Mathematically, the architecture of a LSTM is never fixed and is always arbitrary. Just like many other things in the realm of machine learning, an architecture for a LSTM may exist simply because it works. One such architecture is the Gated Recurrent Unit or the GRU [5].

III. ABOUT THE DATASET

The dataset used for this research is the Microsoft CoCo dataset. The dataset is chiefly comprised of image-caption pairs where each image is paired with five captions describing the image (refer Figure 4). The latest revision of the dataset, which was in 2017, contains 123,287 images and 886,284 instances. All the images in the dataset are divided into 91 categories of images, where one image can come under multiple categories based on the objects found in the image. The images are automatically categorized into categories by using image segmentation [6] techniques.

IV. PROPOSED METHODOLOGY

The following sections and subsections attempt to guide the reader through the exact steps that were undertaken to achieve the desired results. The steps can be reproduced as is and will get the desired results in a quite straight-forward manner. The following diagram represents the entire process in a nutshell.

A. Preparing the Data

Before we can dive into the training process, we need to undertake some preliminary measures to make sure we do not run into any unwarranted errors during the training process. The first thing to do is initialize the dataset for instance annotations and also, we would need to annotate the captions. Annotations basically mean providing more information over a particular instance of the dataset and, we store all the instance and caption annotations in the memory.

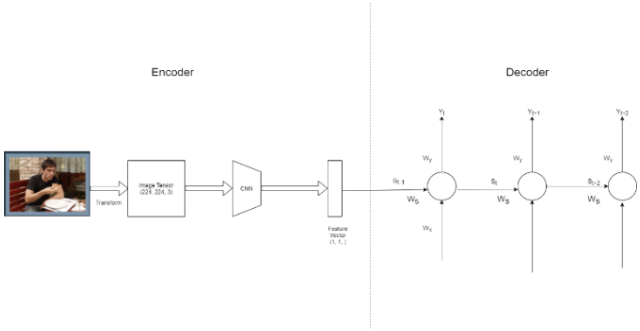

Fig. 5. The proposed architecture in its entirety

B. The Encoder

According to the diagram in Figure 5, we need an encoder and a decoder for the process. As discussed, our encoder will be a Convolutional Neural Network which takes in an input image and outputs a feature vector for the given image which describes the spatial content of the image. To obtain the feature vector with the spatial information of the image, the final fully connected layer of the network is removed. Then the feature vector is passed through an untrained linear layer which outputs an embedded vector as per the input of the decoder (Recurrent Neural Network) The feature vector for each layer of the neural network is calculated as per Equation 1. The neural network used for this research is the Resnet-152 architecture [7]. The last fully connected layer of the neural network is decoupled, and it is replaced by an untrained linear layer depending on the embedding size to be input into the decoder (Figure 5). The embedding size chosen for the neural network is 256.

C. The Decoder

Referring to the diagram in Figure 5, the second part of the plan is a decoder. The decoder is nothing but a recurrent neural network with LSTM cells embedded in each unit of the hidden layer of the RNN. The gradient calculations of the Recurrent Neural Network are done as per Equation 2. The way the RNN is structured is, the RNN takes in as input a feature vector of length equaling the embedding size of 256. Then, hidden layers are stacked upon the input layer of the RNN. The number of hidden layers used in the research is 512. Finally, the output from the hidden layers is passed through a linear fully connected layer which outputs a vector of size denoted by vocabulary size, which in turn denotes the number of words used in the process of sentence formation. The linear layer is run through SoftMax [8] activation. The LSTM architecture used is the one denoted by Figure 3.

D. The Training

Before implementing the training part, it is important to go over the required parameters. This realm of deep learning, which deals with the generation of objects (captions in this context) based upon given set of inputs requires a lot more variables than the ones needed in a classification pipeline.

So, it would be a good approach and a good idea to walk the reader through the variables needed during the training process. The variables, whose knowledge is good to have been as follows: -

1. Batch Size: - The size of every training batch which will be fed into the LSTM. It is the number of image-caption pairs which will be used to amend weights of the model after each training iteration. The batch size of this research is 64.
2. Vocab Threshold: - The minimum number of occurrences needed for a word to be included in the vocabulary which will be used when generating captions. A larger threshold will mean a smaller vocabulary and a smaller threshold would mean a large vocabulary with rare words. The value of this variable selected was 4.
3. Embed Size: - The dimensionality of the image and word embeddings. The embed size selected was 256.
4. Hidden Size: - The number of features in the hidden state of the RNN Decoder. Number of hidden layers in the research are 512.
5. Number of epochs: - As the name suggests, it is the number of epochs needed to train the model. For this paper, I have gone with a value of 3 epochs to achieve a satisfactory result. But the reader is more than welcome to try out different values of number of epochs to try and improve the result. My choice was limited to 3 due to lack of a proper GPU.
6. Save Every: - This variable is used to determine after how many epochs does the model save the weights. Since my epochs were only 3, I went with 1.

The choice of the optimizer for the training process was the Adam optimizer [9]. The trainable parameters for this architecture are bundled in such a way that the last embedding layer of the RNN is clubbed with the trainable parameters of the decoder, which created a magnanimous list of trainable parameters. The training process was carried out in such a manner that each epoch was divided into approximately 6470 steps. That made it easier to visualize the behavior of the training loss with each step since the number of epochs was only 3 due to hardware constraints. The training loss is plotted in Figure 6. The lowest loss achieved was 1.77 and the lowest perplexity of 6.03. Perplexity in this context defines the similarity between the original probability distribution and the estimated probability distribution [10]. Perplexity is defined mathematically like so: -

$$PPL(T) = \frac{1}{\sqrt[n]{m(T)}} \tag{7}$$

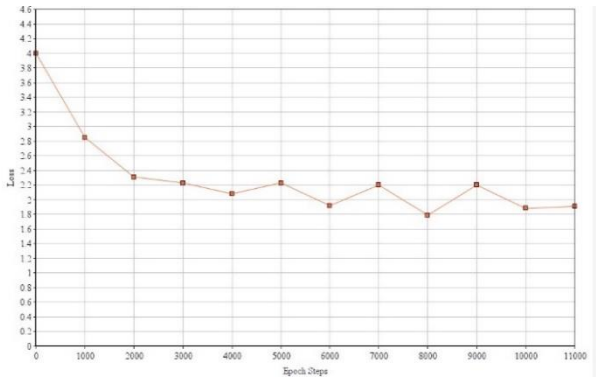

Fig. 6. The training loss vs epoch steps graph. Each epoch represents roughly 7000 steps, so step number 8000 signifies the 1000th step of the 2nd epoch

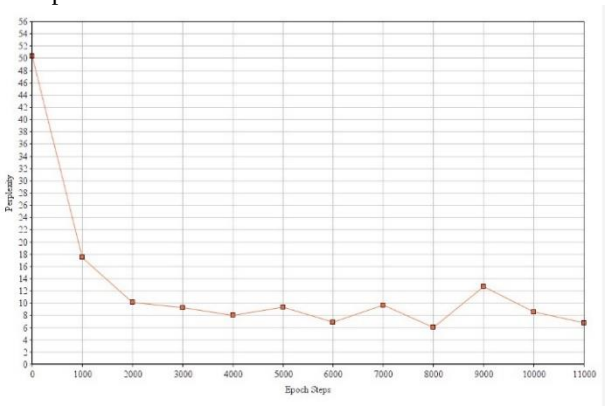

Fig. 7. Behavior of the perplexity against epoch steps. As in Figure 6, each epoch represents approximately 7000 steps, so step 8000 signifies the 1000th step of the 2nd epoch

Here, *n* is the number of words in the test set *T* and *m* is the geometric average probability on test set *T* [10].
Denoted in figure 7 is the behavior of the changing perplexity with each epoch step

E. The Results

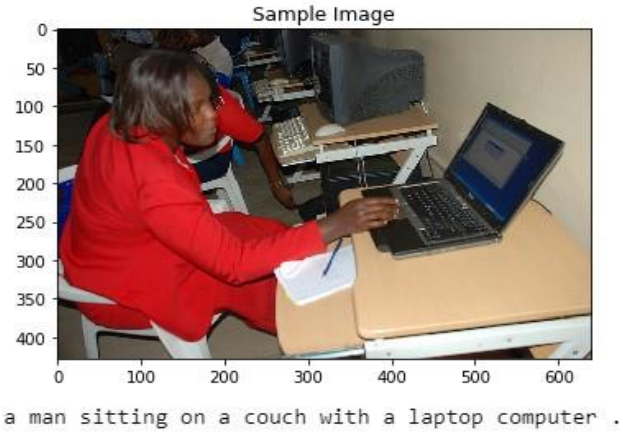

Fig. 8. The generated caption below the input image

After an intense round of training 3 epochs, the lowest loss which was achieved was

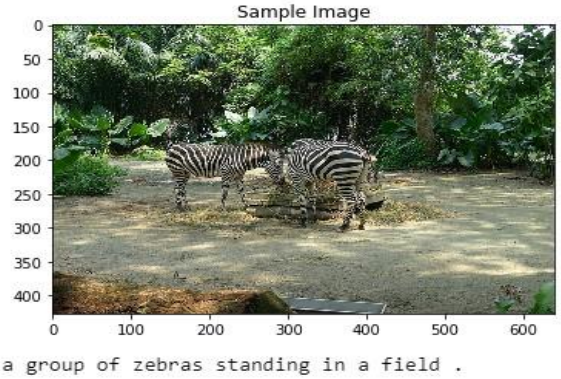

Fig. 9. The generated caption below the input image

Retrieval Number: 100.1/ijrte.E67410110522
DOI: 10.35940/ijrte.E6741.0511122
Journal Website: www.ijrte.org

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)

1.77 and here are some generated captions for given input images. The decoder generates an array of whole numbers for every given image. The numbers are mapped to their respective words in the vocabulary which was generated at the beginning of training. The flow is same as Figure 5. An example of a generated caption can be seen in figures 8 and 9.

V. CONCLUSION

The proposed architecture can successfully generate captions for any given input image in quite a satisfactory manner. A dataset which is as huge and magnanimous as the CoCo dataset needs a lot of time to be trained upon and the highest available batch size as per the hardware available was 64. Nevertheless, the proposed methodology achieves what it was supposed to achieve. The training process starts with each image of each batch of training being transformed into a 224 x 224 x 3 image tensor, and the generated image tensor is then fed into the Convolutional Neural Network (CNN), which used the Resnet-152 architecture in our case and generated a feature vector which matches the dimensionality of the decoder (RNN) which follows. In order to achieve the aforementioned feature vector, the last layer of the CNN is chopped off. Then, the generated feature vector, which contains the spatial information of the image in the form of a numerated vector is then passed to the first layer of the Recurrent Neural Network (RNN), which is equipped with Long Short Term Memory (LSTM) cells, which aids in tacking major problems like the vanishing gradient problem. The RNN trains on the captions of the given image and remembers the pattern in which every word of the caption is formulated. The caption which is generated after the training is satisfactory and is successfully able to capture the gist of the given input image. The advantage of using LSTM cells, which aid in tackling major problems like the vanishing gradient problem. The RNN trains on the captions of the given image and remembers the pattern in which every word of the caption is formulated. The caption which is generated after the training is satisfactory and is successfully able to capture the gist of the given input image. The advantage of using LSTM cells in the RNN really paid dividends as the calculated gradient had a stemmed decay which training which made sure the gradient never reached a value which might be deemed as negligible mathematically. This avoided unwanted circumstances during the training process. The vocabulary threshold of 4 was perfect as far as the scope of the research is concerned and it generated a near perfect vocabulary. To conclude, the results of the proposed methodology and the associated research and literature survey are satisfactory.

REFERENCES

1. O'Shea, Keiron Nash, Ryan. (2015). An Introduction to Convolutional Neural Networks. ArXiv e-prints.
2. Hochreiter, Sepp. (1998). The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems. 6. 107-116. 10.1142/S0218488598000094.
3. Sepp Hochreiter, Jurgen Schmidhuber (1997) Long Short Term Memory¨
4. Alex Sherstinsky (2018) Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network
5. Rajib Rana (2016). Gated Recurrent Unit (GRU) for Emotion Classification from Noisy Speech. ArXiv e-prints
6. Tsung-Yi Lin and Michael Maire and Serge Belongie and Lubomir Bourdev and Ross Girshick and James Hays and Pietro Perona and Deva Ramanan and C. Lawrence Zitnick and Piotr Dollar (2014). Microsoft´COCO: Common Objects in Context. ArXiv e-prints
7. Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun (2015). Deep Residual Learning for Image Recognition. ArXiv e-prints
8. Gao, Bolin and Pavel, Lacra. (2017). On the Properties of the Softmax Function with Application in Game Theory and Reinforcement Learning.
9. Diederik P. Kingma and Jimmy Ba (2014). Adam: A method for stochastic optimisation. ArXiv e-prints
10. Kushal Arora and Anand Rangarajan (2016). Contrastive Entropy: A new evaluation metric for unnormalized language models

AUTHORS PROFILE

Hitoishi Das, is a graduate of the ICFAI Foundation for Higher Education from the Department of Computer Science and Engineering in Hyderabad, India. He is a merit scholarship recipient in his university. He is currently working as a Software Engineer at Technovert Solutions Pvt. Ltd in Hyderabad