DICGM: A Deep Learning based Novel Image Caption Generator Model

Shweta Taneja[1], Bhawna Suri[2] and Savneet Kaur3
shwetataneja@bpitindia.com, bhawnasuri@bpitindia.com, savneetkaur@yahoo.com
1,2 Department of Computer Science & Engineering, Bhagwan Parshuram Institute of Technology
Guru Gobind Singh Indraprastha University, Delhi.
3 Department of Information Technology, Guru Teg Bahadur Institute of Technology, Delhi, India.

Abstract-Image caption generator is a task that involves computer vision and natural language processing concepts to recognize the context of an image and describe them in a natural language like English. It is used in a variety of applications like describing images to People with Visual Impairment (PVI's), web development, describing real time videos etc. With the advancement in Deep learning techniques, availability of huge datasets and computer power, we can build models that can generate captions for an image.

In this work, we propose a fully automated approach that uses deep neural networks to build image caption generator model – DICGM. The image features are extracted from Xception which is a CNN model trained on the ImageNet dataset and then fed to LSTM model, which is responsible for generating the image captions. The main goal here is to put CNN-RNN together to create an automated image captioning model that takes in an image as input and outputs a sequence of text that describes the image. DICGM is tested using Flickr8k_Dataset. It contains a total of 8092 images in JPEG format with different shapes and sizes. Of which 6000 are used for training, 1000 for test and 1000 for development. DICGM works really well with a small number of photos. For production-level models, there is a need to train on datasets larger than 100,000 images which can produce better accuracy models. This proposal is a work in progress and we are convinced that with more extensive tuning and precise configuration the results will improve.

Keywords: Machine learning, Image caption generation, Image colorization, CNN, RNN, LSTM

I Introduction

For a long time, colorization has been a semi-automated process, relying on hints from the user. Levin et al.[1], for instance, proposed that neighboring pixels in space with similar intensities should have similar colors. Thus, in his work, hints are provided as rough inaccurate "scribbles" on a grayscale image, and the algorithm is able to generate Exploring Convolutional Neural Networks for Automatic Image. Several others have improved this algorithm further, including Huang et al [2] (by addressing color-bleeding issues) and Qu et al.[3] (by modifying the cost function to account for color continuity over similar textures in addition to similar intensities).

The objective of our research is to learn the concepts of a CNN and LSTM model and build a working model of Image caption generator by implementing them. In this work, we will be implementing the caption generator using ***CNN (Convolutional Neural Networks)*** and ***LSTM (Long short term memory)***. The image features will be extracted from Xception which is a CNN model trained on the imagenet dataset and then we feed the features into the LSTM model which will be responsible for generating the image captions.

There are two main architectures of an image captioning model. The first one is an image-based model which extracts the features of the image, and the other is a language-based model which translates the features and objects given by our image-based model to a natural sentence. In this, we will be using a pretrained CNN network that is trained on the ImageNet dataset. The images are transformed into a standard resolution of 224 X 224 X 3. This will make the input constant for the model for any given image.

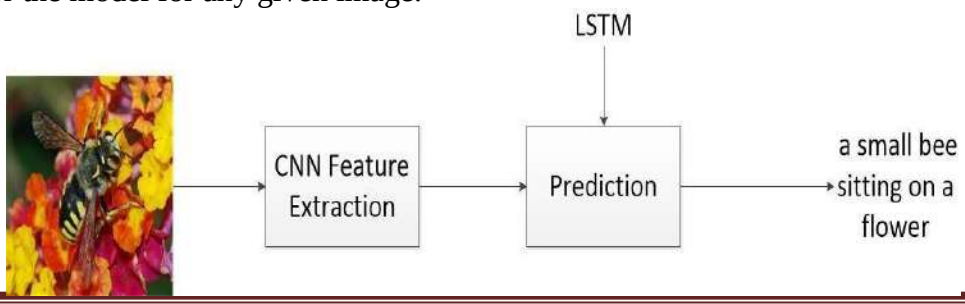

Figure 1 Image captioning model

The condensed feature vector is created from a convolutional neural network (CNN). In technical terms, this feature vector is called *embedding*, and the CNN model is referred to as an **encoder.** In the next stage, we will be using these embeddings from the CNN layer as input to the LSTM network, a **decoder**. In a sentence language model, LSTM is predicting the next word in a sentence. Given the initial embedding of the image, the LSTM is trained to predict the most probable next value of the sequence. Its just like showing a person a series of pictures and asking them to remember the details. And then later show them a new image which has similar content to the previous images and ask them to recall the content. This "recall" and "remember" job is done by our LSTM network. The symbols <start> and <stop> stoppers are used to signal the end of the caption. This way, the model learns from various instances of images and finally predicts the captions for unseen images. The main goal here is to put CNN-RNN together to create an automatic image captioning model that takes in an image as input and outputs a sequence of text that describes the image.

A captioning model relies on two main components, a CNN and an RNN. Captioning is all about merging the two to combine their most powerful attributes i.e. A **Convolutional Neural Network (ConvNet/CNN) which** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. Convolutional Neural networks are specialized deep neural networks which can process the data that has input shape like a 2D matrix. Images are easily represented as a 2D matrix and CNN is very useful in working with images. **Long short-term memory (LSTM)**is a type of RNN (**recurrent neural network**) which is well suited for sequence prediction problems. Based on the previous text, the next word can be predicted. It has proven itself effective from the traditional RNN by overcoming the limitations of RNN which had short term memory. LSTM can carry out relevant information throughout the processing of inputs and with a forget gate, it discards non-relevant information. An LSTM consists of three main components: a forget gate, input gate, and output gate. Each of these gates is responsible for altering updates to the cell's memory state. The **long-term memory** is usually called the **cell state**. The looping arrows indicate the recursive nature of the cell. This allows information from previous intervals to be stored within the LSTM cell. Cell state is modified by the forget gate placed below the cell state and also adjusted by the input modulation gate. From the equation, the previous cell state forgets by multiplying with the forget gate and adds new information through the output of the input gates. The **remember vector** is usually called the **forget gate**. The output of the forget gate tells the cell state which information to forget by multiplying 0 to a position in the matrix. If the output of the forget gate is 1, the information is kept in the cell state. From equation, the sigmoid function is applied to the weighted input/observation and previous hidden state. The save vector is usually called the input gate. These gates determine which information should enter the cell state / long-term memory. The important parts are the activation functions for each gate. The input gate is a sigmoid function and have a range of [0,1]. The **focus vector** is usually called the **output gate**. Out of all the possible values from the matrix, which should be moving forward to the next hidden state. The **working memory** is usually called the **hidden state**. What information should I take to the next sequence? This is analogous to the hidden state in RNN and HMM.

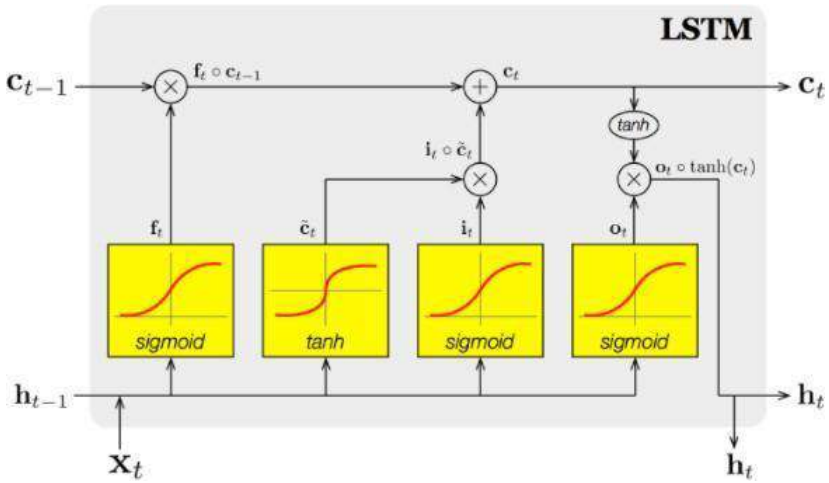

Figure 2 VIEW OF LSTM CELL

The first sigmoid activation function is the **forget gate**. Which information should be forgotten from the previous cell state (C_t-1). The second sigmoid and first tanh() activation function is our **input gate**. The last sigmoid is the **output gate** and highlights which information should be going to the next **hidden state**.

II State of Art

We review and describe the main categories of existing image captioning methods and they include template-based image captioning, retrieval-based image captioning, novel caption generation and Deep neural network-based image captioning.

2.1 Template-based image captioning

Template-based approaches have fixed templates with a number of blank slots to generate captions. In these approaches, different objects, attributes, actions are detected first and then the blank spaces in the templates are filled. For example, Farhadi et al. [4] use a triplet of scene elements to fill the template slots for generating image captions. Li et al. [5] extract the phrases related to detected objects, attributes and their relationships for this purpose. A Conditional Random Field (CRF) is adopted by Kulkarni et al. [6] to infer the objects, attributes, and prepositions before filling in the gaps. Template-based methods can generate grammatically correct captions. However, templates are predefined and cannot generate variable-length captions. Moreover, later on, parsing based language models have been introduced in image captioning [8,9,10,11] which are more powerful than fixed template-based methods.

2.2 Retrieval-based image captioning

One type of image captioning commonly used in early work is retrieval based. Here, the query image, retrieval-based methods produce a caption for it through retrieving one or a set of sentences from a pre-specified sentence pool. The generated caption can either be a sentence that has already existed, or a sentence composed from the retrieved ones. In [12], the ⟨object, action, scene⟩ meaning space to link images and sentences is established. Given a query image, they map it into the meaning space by solving a Markov Random Field, and use Lin similarity measure to determine the semantic distance between this image and each existing sentence. In [13], to caption an image global image descriptors are employed to retrieve a set of images from a web scale collection of captioned photographs. Then, they utilize semantic contents of the retrieved images to perform re-ranking and use the caption of the top image as the description of the query.

2.3 Novel caption generation.

Novel captions can be generated from both visual space and multimodal space. A general approach of this category is to analyze the visual content of the image first and then generate image captions from the visual content using a language model [9,10,11,12]. These methods can generate new captions for each image that are semantically more accurate than previous approaches. Most novel caption generation methods use deep machine learning based techniques. Therefore, deep learning based novel image caption generating methods are our main focus in this literature. Usually captions are generated for a whole scene in the image.

2.4 Deep neural network-based image captioning

Retrieval based and template-based image captioning methods are adopted mainly in early work. Due to great progress made in the field of deep learning [13] [14], recent work begins to rely on deep neural networks for automatic image captioning. Even though deep neural networks are now widely adopted for tackling the image captioning task, different methods may be based on different frameworks. Therefore, we classify deep neural network-based methods into subcategories on the basis of the main framework they use and discuss each subcategory respectively.

2.4.1 Encoder-Decoder Architecture-Based Image captioning

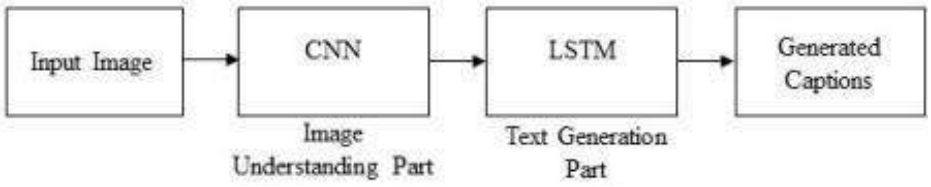

Figure 3: A block diagram of simple Encoder-Decoder architecture-based image captioning

The neural network-based image captioning methods work as just simple end to end manner. These methods are very similar to the encoder-decoder framework-based neural machine translation [15]. In this network, global image features are extracted from the hidden activations of CNN and then fed them into an LSTM to generate a sequence of words. A typical method of this category has the following general steps:
(1) A vanilla CNN is used to obtain the scene type, to detect the objects and their relationships.
(2) The output of Step 1 is used by a language model to convert them into words, combined phrases that produce an image caption. A simple block diagram of this category is given in Figure 3. Vinyals et al. [23] proposed a method called Neural Image Caption Generator (NIC). The method uses a CNN for image representations and an LSTM for generating image captions. This special CNN uses a novel method for batch normalization and the output of the last hidden layer of CNN is used as an input to the LSTM decoder. This LSTM is capable of keeping track of the objects that already have been described using text. NIC is trained based on maximum likelihood estimation. In generating image captions, image information is included to the initial state of an LSTM. The next words are generated based on the current time step and the previous hidden state. This process continues until it gets the end token of the sentence. Since image information is fed only at the beginning of the process, it may face vanishing gradient problems. The role of the words generated at the beginning is also becoming weaker and weaker. Therefore, LSTM is still facing challenges in generating long length sentences.

III Materials and Methods

3.1 Data set

We have used the following two datasets:

Flickr8k_Dataset: Contains a total of 8092 images in JPEG format with different shapes and sizes. Of which 6000 are used for training, 1000 for test and 1000 for development.

Flickr8k_text: Contains text files describing train_set ,test_set. Flickr8k.token.txt contains 5 captions for each image i.e. total 40460 captions.

This dataset contains 8000 images each with 5 captions (as we have already seen in the Introduction section that an image can have multiple captions, all being relevant simultaneously). The images do not contain any famous person or place so that the entire image can be learnt based on all the different objects in the image. It is small in size. So, the model can be trained easily on low-end laptops/desktops. Data is properly labelled, for each image 5 captions are provided and the dataset is available for free. Example: Each photo has 5 captions:

Figure4 sample data

```
['A blond horse and a blond girl in a black sweatshirt be
stare at a fire in a barrel.'
'A girl and her horse stand by a fire.'
'A girl holds a horse 's lead behind a fire.'
'A man, and girl and two horse be near a contain fire.'
'Two person and two horse watch a fire.']
```

These images are bifurcated as follows: Training Set — 6000 image and Test Set — 2000 images.

3.2 Proposed Approach

In our proposed approach, two main objectives are met- Colorizing Grayscale Videos And building an Image Caption Generator Model.

3.2.1 Method for Colorizing Grayscale Videos

- Phase one of the algorithm has five parts of action, as following:
- Accepting the colored source image.
- Converting that image to a grayscale image and extract the L (luminance) value out of given RGB image; using formula Gray(L) = 0.299 Red + 0.587 Green + 0.114Blue.
- Each pixel of the grayscale version is processed in which its luminance, average and the standard deviation of its 8-neighbours are recorded in a table called learning table. These three columns (the intensity of each pixel, the average of 8-neighborhood pixels, and the standard deviation of 8-neighborhood pixels) are applied as inputs to the 3 nodes neural network.
- The target data are prepared by recording the RGB values of each pixel of the colored version. So, there will be three inputs, and three targets.
- The whole dataset is divided to two subsets in a full random procedure, 90% for training set and 10 % for testing set.

Figure 5 Block diagram for the first phase.

Colorizing of the Target Grayscale Image

Once the neural network is trained it can be directly used to colorize the destination grayscale image. The following points explain the second phase of the technique .Accepting the gray scale image (destination image to be colorized).

- The luminance (L) of pixel values of the grayscale image will be determined and recorded as well as the average and standard deviation of its 8-neighborhood pixels.
- The variables (L) are applied as inputs to the trained ANN to produce three values which are the R, G, and B components of the colored version.
- The output RGB components are rearranged in three layered matrix to reconstruct a colored image.

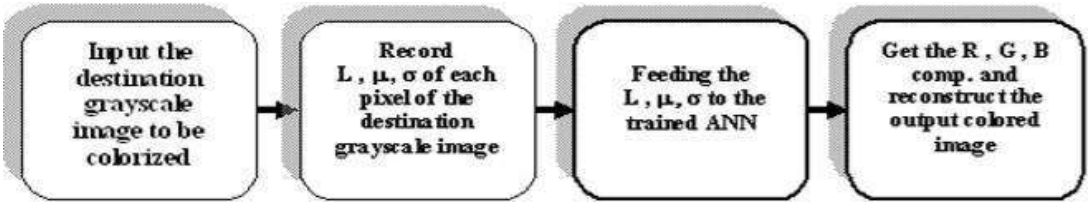

Figure 6 Block diagram for the second phase

3.2.2 Image Caption Generator Merged Model

So, to make our image caption generator model, we will be merging these architectures. It is also called the CNN-RNN model.

1. CNN is used for extracting features from the image. We will use the pre-trained model Exception.
2. LSTM will use the information from CNN to help generate a description of the image.

3.2.3 Proposed CNN-RNN Model

We'll be using a pre-trained network like VGG16 or Xception. Since we want a set of features that represents the spatial content in the image, **we're going to remove the final fully connected layer that classifies the image** and look at earlier layer that processes the spatial information in the image. So now CNN acts as a **feature extractor** that compresses the information in the original image into a smaller representation. Since it encodes the content of the image into a smaller feature vector hence, this **CNN is often called the encoder.**

When we process this feature vector and use it as an initial input to the following RNN, then it would be called decoder because RNN would decode the process feature vector and turn it into natural language.

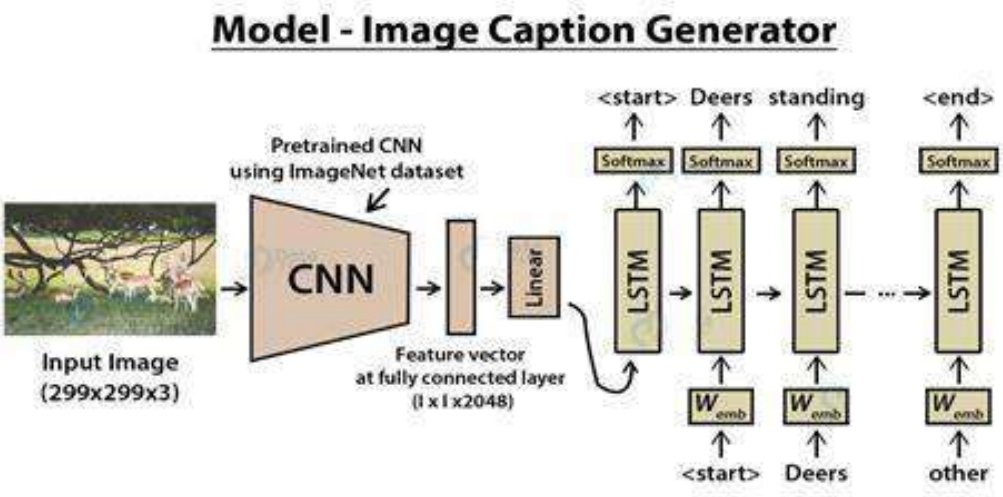

Figure 7 Combined Model-Image Caption Generator

Data pre-processing and cleaning is an important part of the whole model building process. Understanding the data helps us to build more accurate models.

Caption Preprocessing

Each image in the dataset is provided with 5 captions. For e.g image 1000268201_693b08cb0e.jpeg has captions

```
['A child in a pink dress is climbing up a set of stairs in
an entry way .',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']
```

Captions are read from Flickr8k.token.txt file and stored in dictionary k:v where k = image id and value =[list of caption].Since there are 5 captions for each image and we have preprocessed and encoded them in below format

"startseq " + caption + " endseq"

The reason behind startseq and endseq is,

startseq : Will act as our first word when feature extracted image vector is fed to decoder. It will kick-start the caption generation process.

enseq : This will tell the decoder when to stop. We will stop predicting word as soon as endseq appears or we have predicted all words from train dictionary whichever comes first.

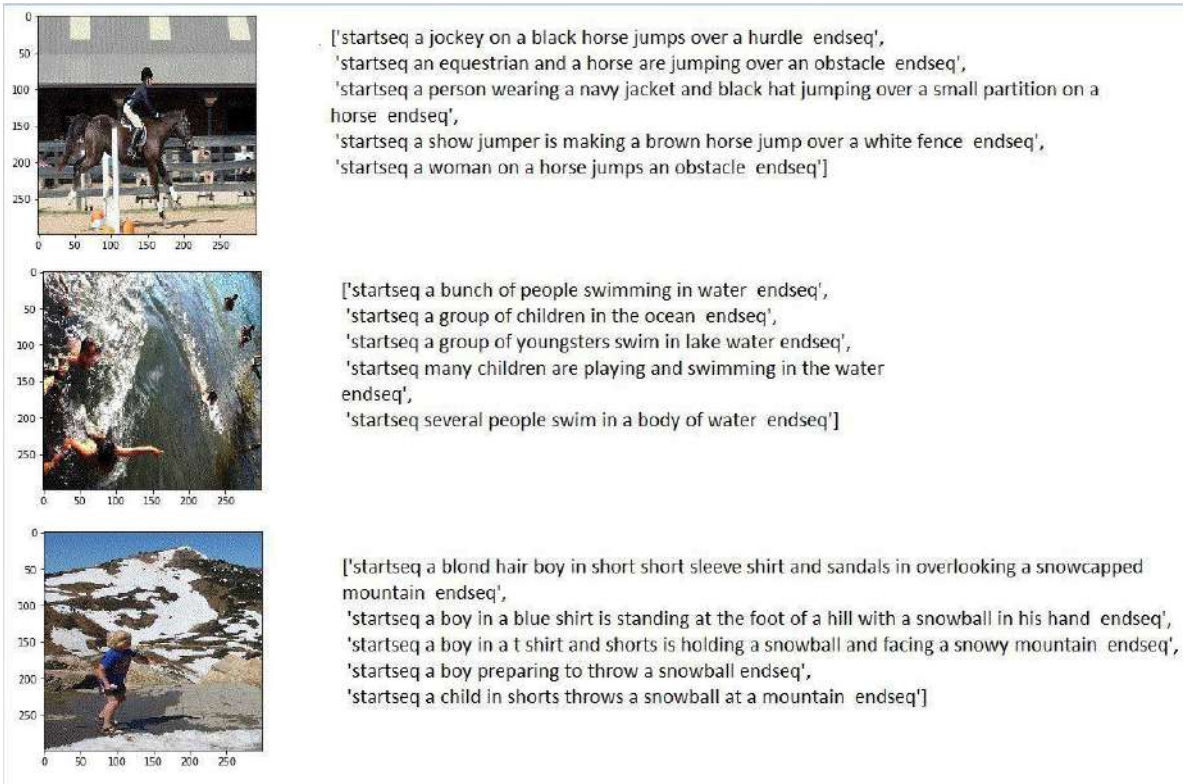

Figure 8 Captions after preprocessing

Tokenizing Captions

The RNN component of the captioning network is trained on the captions in the Flickr_8K dataset. We're aiming to train the RNN to predict the next word of a sentence based on previous words. For this we transform the captioned associated with the image into a list of tokenize words. This tokenization turns any strings into a list of integers. First, we iterate through all of the training captions and create a dictionary that maps all unique words to a numerical index. So, every word we come across, will have a corresponding integer value that we can find in this dictionary. The words in these dictionaries are referred to as our vocabulary. This list of tokens (in a caption) is then turned into a list of integers which come from our dictionary that maps each distinct word in the vocabulary to an integer value. It transforms each word in a caption into a vector of a desired consistent shape. After this embedding step, we're finally ready to train an RNN that can predict the most likely next word in a sentence.

We will define the deep learning model and fit it on the training dataset. This section is divided into the following parts:

1. Loading Data.
2. Defining the Model.
3. Fitting the Model.
4. Evaluate the model.

Loading Data

First, we must load the prepared photo and text data so that we can use it to fit the model.We are going to train the data on all of the photos and captions in the training dataset. While training, we are going to monitor the performance of the model on the development dataset and use that performance to decide when to save models to file.

The train and development dataset have been predefined in the *Flickr_8k.trainImages.txt* and *Flickr_8k.devImages.txt* files respectively, that both contain lists of photo file names. From these file names, we can extract the photo identifiers and use these identifiers to filter photos and descriptions for each set.

The model we will develop will generate a caption given a photo, and the caption will be generated one word at a time. The sequence of previously generated words will be provided as input. Therefore, we will need a '*first word*' to kick-off the generation process and a '*last word*' to signal the end of the caption.

We will use the strings '*startseq*' and '*endseq*' for this purpose. These tokens are added to the loaded descriptions as they are loaded. It is important to do this now before we encode the text so that the tokens are also encoded correctly.

Defining the Model

We will define a deep learning based on the "*merge-model*" described by Marc Tanti, et al. in their 2017 papers. We will describe the model in three parts:

- **Photo Feature Extractor**. This is a 16-layer VGG model pre-trained on the ImageNet dataset. We have pre-processed the photos with the VGG model (without the output layer) and will use the extracted features predicted by this model as input.
- **Sequence Processor**. This is a word embedding layer for handling the text input, followed by a Long Short-Term Memory (LSTM) recurrent neural network layer.
- **Decoder** (for lack of a better name). Both the feature extractor and sequence processor output a fixed-length vector. These are merged together and processed by a Dense layer to make a final prediction.

The Photo Feature Extractor model expects input photo features to be a vector of 4,096 elements. These are processed by a Dense layer to produce a 256-element representation of the photo.

The Sequence Processor model expects input sequences with a pre-defined length (34 words) which are fed into an Embedding layer that uses a mask to ignore padded values. This is followed by an LSTM layer with 256 memory units. Both the input models produce a 256-element vector. Further, both input models use regularization in the form of 50% dropout. This is to reduce overfitting the training dataset, as this model configuration learns very fast.

The Decoder model merges the vectors from both input models using an addition operation. This is then fed to a Dense 256 neuron layer and then to a final output Dense layer that makes a SoftMax prediction over the entire output vocabulary for the next word in the sequence.

Fitting the Model

When the skill of the model on the development dataset improves at the end of an epoch, we will save the whole model to file. At the end of the run, we can then use the saved model with the best skill on the training dataset as our final model. We can do this by defining a *ModelCheckpoint* in Keras and specifying it to monitor the minimum loss on the validation dataset and save the model to a file that has both the training and validation loss in the filename.

Evaluate Model

Once the model is fit, we can evaluate the skill of its predictions on the holdout test dataset.We will evaluate a model by generating descriptions for all photos in the test dataset and evaluating those predictions with a standard cost function.First, we need to be able to generate a description for a photo using a trained model. This involves passing in the start description token '*startseq*', generating one word, then calling the model recursively with generated words as input until the end of sequence token is reached '*endseq*' or the maximum description length is reached.

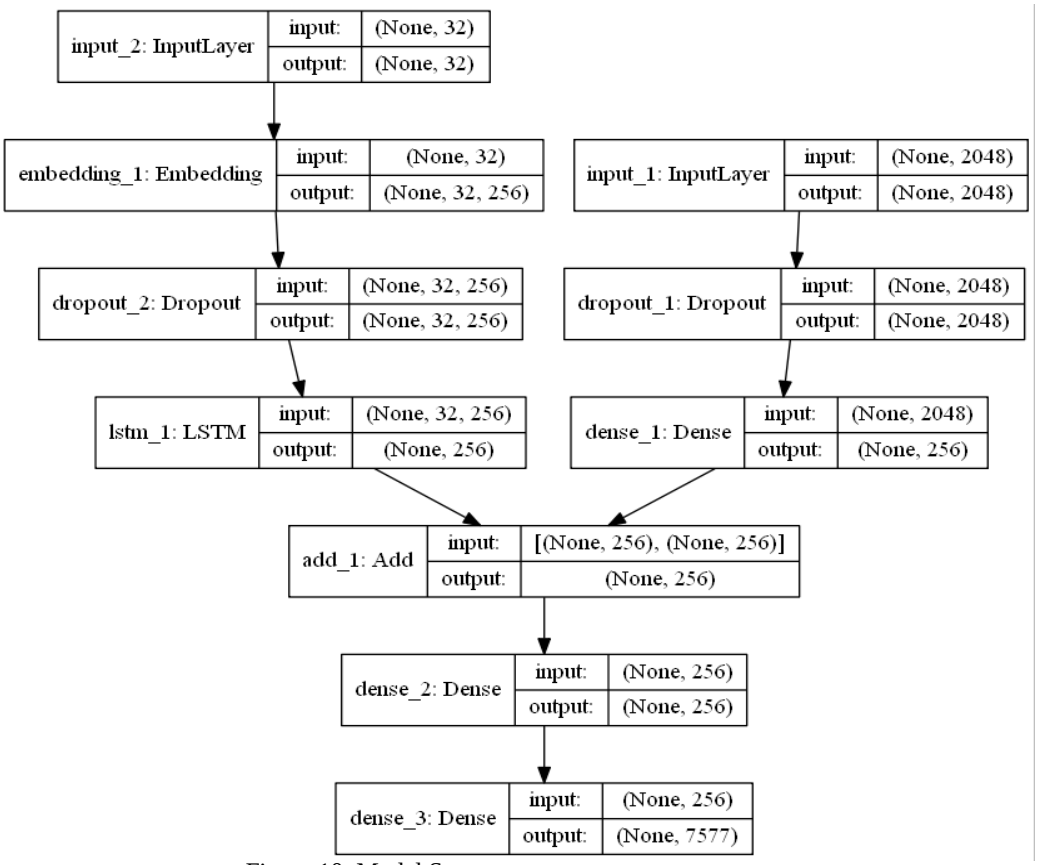

Figure 10 Model Summary

IV Results and Discussion

In this work, we have implemented a CNN-RNN model by building an image caption generator. Some key points to note are that our model depends on the data, so, it cannot predict the words that are out of its vocabulary. We used a small dataset consisting of 8000 images. For production-level models, we need to train on datasets larger than 100,000 images which can produce better accuracy models.

Caption : two girls are playing in the grass

Caption : man in yellow kayak is reflecting up river

Figure 11 Result for image captioning

Following colorization is performed on manually handpicked similar images from Unsplash dataset which is an ongoing project containing images of countryside, coasts, mountains, roads etc. One of these videos is converted into gray scale and other one is used to colorize first one. In this work, we figured out what is deep learning. we assembled and trained the CNN model to colorize black and white videos. We have tested that this model works really well with a small number of photos. For production-level models, we need to train on datasets larger than 100,000 images which can produce better accuracy models.

Figure 12 Results for Video Colorization

V CONCLUSION

In this overview, we have compiled all aspects of the image caption generation task, discussed the model framework proposed in recent years to solve the description task, focused on the algorithmic essence of different mechanisms. We summarize the large datasets and evaluation criteria commonly used in practice. Although image caption can be applied to image retrieval, video caption, and video movement and the variety of image caption systems are available today, experimental results show that this task still has better performance systems and improvement. It mainly faces the following three challenges: first, how to generate complete natural language sentences like a human being; second, how to make the generated sentence grammatically correct; and third, how to make the caption semantics as clear as possible and consistent with the given image content. In this work we design and develop a methodology, strongly based on neural networks, to colorize gray level videos. Our proposal consists on a series of steps to finally obtain a trained neural network which predicts the color of a gray level pixel. The methodology joins two main ideas from previous works: color reduction using vector quantization and using a group of neighboring pixels to predict the color of a single pixel. In our opinion the results are good and promising although the prediction scores obtained are below the 50%. To a human eye the images obtained show good results and have colors close to the intuition. As the prediction highly depends on the set of training images, the images more similar to the training set show better results than those more different.

References

1. Levin, A., Lischinski, D., & Weiss, Y. (2004). Colorization Using Optimization. ACM Transactions on Graphics (Tog), 23, 689-694.
2. Huang, Yi-Chin, et al. "An adaptive edge detection-based colorization algorithm and its applications." Proceedings
   of the 13th annual ACM international conference on Multimedia. ACM, 2005
3. Qu, Yingge, Tien-Tsin Wong, and Pheng-Ann Heng. "Manga colorization." ACM Transactions on Graphics (TOG). Vol.25. No. 3. ACM, 2006.
4. Ali Farhadi, Mohsen Hejrati, Mohammad Amin Sadeghi, Peter Young, Cyrus Rashtchian, Julia Hockenmaier, and David Forsyth. 2010. Every picture tells a story: Generating sentences from images. In European conference on computer vision. Springer, 15–29.
5. Siming Li, Girish Kulkarni, Tamara L Berg, Alexander C Berg, and Yejin Choi. 2011. Composing simple image descriptions using web-scale n-grams, 220–228.
6. Girish Kulkarni, Visruth Premraj, Sagnik Dhar, Siming Li, Yejin Choi, Alexander C Berg, and Tamara L Berg. 2011. Baby talk: Understanding and generating image descriptions. In Proceedings of the 24th CVPR. Citeseer.
7. .J. Curran, S. Clark, J. Bos, Linguistically motivated large-scale nlp with cc and boxer, in: Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions, pp. 33–36.

8. Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In International Conference on Machine Learning. 2048–2057.
9. V. Ordonez, G. Kulkarni, T. L. Berg., Im2text: Describing images using 1 million captioned photographs, in: Advances in Neural Information Processing Systems, 2011, pp. 1143–1151.
10. Ryan Kiros, Ruslan Salakhutdinov, and Richard S Zemel. 2014. Unifying visual- semantic embeddings with multimodal neural language models. In Workshop on Neural Information Processing Systems (NIPS)).
11. Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. 2015. Show, attend and tell: Neural image caption generation with visual attention. In International
12. Conference on Machine Learning. 2048–2057.
13. Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo. 2016. Image captioning with semantic attention. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 4651–4659.

14. Ting Yao, Yingwei Pan, Yehao Li, Zhaofan Qiu, and Tao Mei. 2017. Boosting image captioning with attributes. In IEEE International Conference on Computer Vision (ICCV). 4904–4912.

15. Y. Bengio, A. Courville, P. Vincent, Representation learning: A review and new perspectives, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (8) (2013) 1798–1828.