Image Captioning

Avinash Kumar Mishra

Submitted for the Degree of Master of Science in

Data Science & Analytics

Department of Computer Science
Royal Holloway University of London
Egham, Surrey TW20 0EX, UK

December 11, 2021

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: 10120

Student Name: Avinash Kumar Mishra

Date of Submission: 07/12/2020

Signature: Avinash K Mishra

Acknowledgement

I would like to start by thanking my wonderful supervisor, Professor Nicola Paoletti, for his patience and insightful ideas that helped create and shape my thesis. Throughout this effort, I am grateful for his ongoing assistance and encouragement. I'd like to thank everyone at Royal Holloway University of London's Department of Computer Science for giving me with the knowledge and tools I needed to finish my project.

Abstract

Captioning the content of the Image is a basic requirement since the evolution of artificial intelligence in the data science field. This paper is focused on how to generate caption for an image. In this project we have used convolution networks which is joined with a machine translation technique to generate a sentence. The model is trained in Maximum Likelihood Estimation (MLE) of the word in a sentence given an training image. The Experiment is performed with generative model based on state of art model Inception V3. Inception V3 is being used as transfer learning to encode our train and test images, each image's captions is then trained using Long short-term memory(LSTM) with Glove word-embedding technique to convert text to vector representation. We have performed this experiment on dataset like Flickr8K, Flickr30K and COCO and it produces a significant results. Overall, Bleu score of uni-gram is $>$ .80 and for 2-gram it is .66. We have also used some random images from Internet to see our model performance or prediction capability, and it seems to be performing well. This shows that my model is not over fitting with the training dataset and is able to generalize on un-seen data, in other words images from different distribution. However we have also discussed some instances where my model has performed poorly and how we can mitigate this problem.

Contents

1 Introduction

A human can point out and describe an enormous amount of details about a visual scene with just a quick glance at it [9]. Using common languages and words to convey the substance of an image is a difficult task, but it could create great impact on everyone's life, for instance by assisting visually challenged people in better comprehending the content of their surroundings [34].

This task of generating natural languages with meaningful sentences of images and their region is one of the major focus of computer vision community. There has been many advances is visual recognition which has focused on labelling image and its region with a set of defined categories. The goal of this dissertation is to not only describe the objects but also express how different objects in a image can relate to each other. Additionally, the images encoding is being processed to generate natural language sentences through a language model.

Some approaches have been developed to address the challenge of describing images and its region. There are some previous attempts which are proposed to tie together different solutions of the above sub-problems. In this project I would attempt to build a model that would take an image as input, and is trained based on maximize the likelihood probability of each word to produce the natural language sentences where each word will come from the vocabulary build during the model, and then tries to describes the image and its property.

The motivation for building this technique comes from recent development in artificial intelligence, where the requirement is to transform a image and its region into a sentence. To build the model, the most elegant method of image representation, deep convolution neural network (CNN) is used. It has been demonstrated during the last few years that CNNs can provide a rich representation of an input image by embedding it in a fixed-length vector, which can then be used for a range of vision applications [19]. The model is also added with LSTM(RNN) technique for language generation. The end-to-end system is build by combining state-of-art networks for vision and language models. This experiment is explored on Flickr, Mococo dataset and some random internet images.

2 Background Research

Alexey Ivakhnenko and Lapa released the first generic, working learning method for supervised, deep, feedforward multilayer perceptrons in 1967 [16]. A deep network with eight layers trained by the group approach of data handling was reported in a 1971 paper [15]. Other deep learning working designs, particularly ones designed for computer vision, date back to Kunihiko Fukushima's Neocognitron, which was released in 1980 [10].

Geoff Hinton, Ruslan Salakhutdinov, Osindero, and Teh published papers in 2006 [12][13] showing how to effectively pre-train a multi-layered feedforward neural network one layer at a time, treating each layer as an unsupervised limited Boltzmann machine and fine-tuning it with supervised backpropagation. The studies discussed deep belief net learning[2][11].

According to Yann LeCun, the influence of deep learning in industry began in the early 2000s, when CNNs processed an estimated 10% to 20% of all checks signed in the United States. Deep learning for large-scale speech recognition was first used in industry about 2010 [24]. Since then we have seen many advances in Neural network area.

The new artificial intelligence (AI) boom has raised adequate awareness among Neural Networks in academia and industry. We have probably come across information that claims that some sort of AI/Neural Net system will eventually replace your current process. I am sure everyone have heard about Deep Neural Networks and Deep Learning accomplishment in Data Science realm.

Today Artificial Intelligence solves many problem in different areas. We will briefly discuss about few major areas and its impact in different fields.

Deep learning has been welcomed in every digital corner of our day-to-day lives since the arrival of cost-effective compute power and data storage. Popular virtual assistants like Alexa/Siri/Google Assistant, the suggestion to tag a friend in a Facebook photo uploaded, or a suggestion to add a new friend. Tesla's driverless cars, Snapchat's cat filter, Amazon and Netflix's movie/series recommendations are just a few examples of daily life digital products based on deep neural network. Without realising it, you may have already used a Deep Learning-based product.

Healthcare has used deep learning to identify cancer and diabetic retinopathy, Aviation has used it to optimise fleets, Oil & Gas has used it to anticipate machinery maintenance, Banking & Financial Services has used

it to detect fraud, retail and telecom has used it to predict customer churn, and so on. AI, as Andrew NG correctly stated, is the new electricity. In the same way that electricity changed everything, AI will change practically everything in the near future.

We must first comprehend the importance of this issue in real-world circumstances. Let's look at a couple situations when a solution to this problem could be quite valuable.

We can create a device for the blind that will guide them on the roads without requiring aid from others. This can be done by converting the scene to text and then the text to speech. Both of these Deep Learning applications are now well-known. To learn more about Nvidia Research's efforts to produce such a product, go to [7].

Additionally, Security cameras are now ubiquitous, but if we can provide appropriate knowledge in addition to watching the world, we can trigger alarms as soon as criminal activity is detected somewhere. This is likely to help minimise crime and/or accidents.

Self-driving automobiles, whenever he hear this term we relates things to TESLA. Automated driving is one of the most arduous difficulties, and describing the area around the car can help the self-driving system.

Because every image could be converted into a caption first, and then searches could be conducted based on the caption, automatic captioning has helped Google Image search become as good as Google Search.

From here I will deep dive on different aspect of Deep Learning I have explored and learned during my attempt to create caption generative model.

2.1 Deep Neural Networks

Deep Learning is a sub-field of machine learning in Artificial Intelligence that works with algorithms inspired by the biological structure and function of the brain to help computers with intelligence (See Fig. 1).

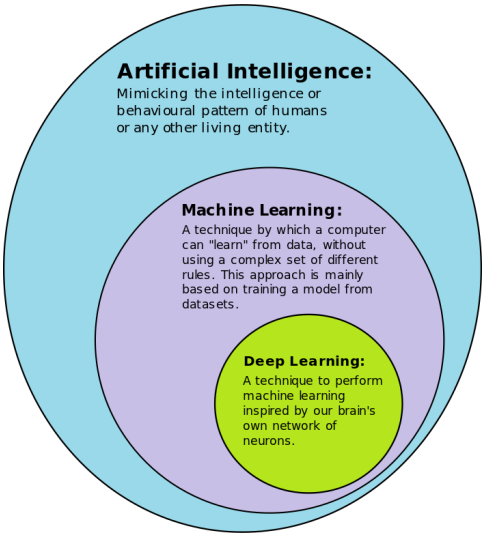


Figure 1: Deep learning is a subset of machine learning and machine learning is a subset of artificial intelligence [1]

Deep Neural Networks can be characterised as the property of intelligence incorporated into machines in its most general form. Because machines are frequently unintelligent, we must instil intelligence in them so that they can make judgments on their own. Consider an ATM machine that accepts your request to disburse the amount you desire using the correct mix of notes in the machine. Similarly, a washing machine that can choose the appropriate amount of water to consume, as well as the appropriate duration for soaking, washing, and spinning, i.e. making decisions when certain inputs are presented and therefore functioning smarter. Artificial Intelligence refers to the process of instilling intelligence into a machine in an artificial manner.

It's worth noting that this intelligence has been intentionally designed, as in a thorough set of if-else rules. The system's designer thoroughly considered all of the conceivable possibilities before creating a decision-making system based on rules by following a set rule path. What if we needed to give a machine intelligence without explicitly programming it, something that the machine could learn on its own? This is when we come into contact with Machine Learning [29].

The act of infusing intelligence into a system or machine without explicit programming is known as machine learning.

A system that learns from historical test results and student traits could predict whether a student will fail or pass a test as an example of machine learning. Instead of being programmed with a thorough list of all conceivable rules for determining whether a student will pass or fail, the system learns on its own based on patterns discovered in prior training data [29].

So, how does Deep Learning fit into this picture? Machine learning, while effective for a wide range of issues, falls short in some areas where humans excel. For example, categorising an image as a cat or a dog, or identifying audio recordings as male or female voices, and so on. Machine learning struggles with image, audio, and other sorts of unstructured input. When examining the reasons for this bad performance, an epiphany led to the notion of simulating the biological process of learning new things in the human brain, which is made up of billions of neurons connected and organised. On the other hand, neural networks had been a research focus for several years but had made only limited progress due to computational and data limitations [17]. When researchers combined machine learning and neural networks, the discipline of deep learning arose, which was defined by the development of deep neural networks, which are improvised neural networks with many more layers.

Each layers of deep learning learns to turn the data it receives into a more abstract and composite representation. The raw input may be a matrix of pixels in an image recognition application;The first representational layer can abstract pixels and encode edges; the second layer can compose and encode edge configurations; the third layer can encode a nose and eyes; and the fourth layer can recognise that the image comprises a face. Importantly, a deep learning process can figure out for itself which features belong in which level.

2.1.1 Convolutional Neural Network

Convolutional Neural Network was formally introduced as LeNet-5 in 1998. It was initially known for reading zip codes, digits. However the most popular Convolutional Networks in Computer Vision was the AlexNet. The AlexNet was submitted as part of ImageNet ILSVRC challenge in 2012 and significantly outperformed all other models. Since then Convolutional Neural Network is being widely used in image classification technique producing state of the art results when applied to different prediction problems across a variety of fields [25].

A simple Convolutional Neural Network is a sequence of layers, and every layer of a Network filters converts a 3-Dimesion of activations to an-

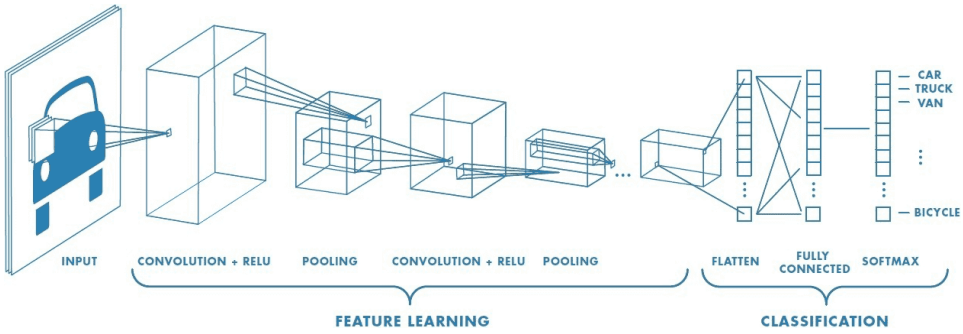


Figure 2: A simple Convolutional Neural Network [32]

other through a differentiable function. Generally Convolutional Neural Network architectures are made of three main types of layer-: Convolutional Layer with activation functions, Pooling Layer, and dense or Fully-Connected Layer (See Fig. 2).

The Convolutional Neural Network layer's parameters consist of a set of trainable filters/kernels. Every kernel has a tiny width and height dimension and covers the entire depth of the input image. For example, a typical filter on a first layer of a Convolutional network might have size 15x15x3 (i.e. 15 pixels width and height, and 3 because images have depth 3 known as the color channels). During the forward propagation, Image to be entered At a specific position, slide/convolve each filter over the width and height of the input volume and compute dot products between the filter and the input. Slide the filter across the width and height of the input volume to generate a 2-dimensional matrix containing the filter's responses at all spatial positions. On the first layer, the network will learn filters that activate when it sees a visual characteristic such as a vertical/horizontal edge of some orientation or a colour difference, and on subsequent levels, the network will learn filters that activate when it sees a whole object or shapes of a pattern. An entire set of filters in each Convolutional Layer produces a separate 2-dimensional matrix of activation map. These activation are stacked maps along the depth dimension to produce the resultant output volume.

let's consider an example, we take an input image of size $3 \times 100 \times 100$ in other words 3 colour images of size 100×100 pixels. And then it is convolve with 12 3×3 convolutional neurons which means we have 12 filters each of size $3 \times 3 \times 3$. While convolving depth of image and each convolution filter is always same to generate 2-dimensional output. And consider stride $= 1$ and no padding(stride and padding are explained later). This will give output

of $12 \times 98 \times 98$ using the formula below.

$$W_{out} = \frac{W_{old} - F + 2P}{S} + 1$$

W_{old} : Existing dimension (lets say 100 as one of side of the image)
F : Filter size(lets say 3 as one of side of the filter)
P : Padding
S : Stride
W_{old} : Output/size of one side of the new image matrix

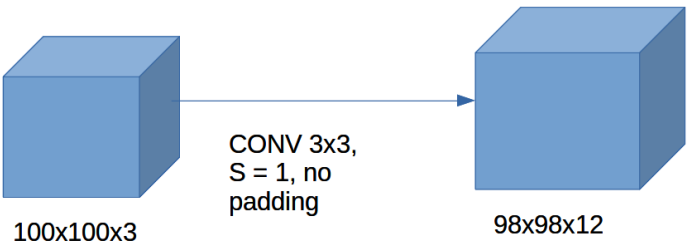


Figure 3: convolution operation

During this process we have $3 \times 3 \times 3 + 1 = 28$ trainable weights (the $+1$ is for bias) for each filter and as we have 12 convolutional filters. Thus we have $12 \times 28 = 336$ total trainable weights. These weight will be trained during backpropagation in Convolutional model.

In a Convolutional Neural Network a Pooling layer is periodically inserted between two adjacent Convolutional layers in a architecture. The purpose of this function is to gradually shrink the spatial dimension of the representation in order to reduce the number of parameters and computations in the network, as well as to prevent overfitting. Using the MAX operation, the Pooling Layer operates independently on each depth slice of the input and resizes it spatially. A pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, removing 75% of the activations, and is the most frequent variant. [8]. Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged. More generally, the pooling layer:

Accepts a volume of size $W_1 \times H_1 \times D_1$ Requires two hyperparameters: their spatial extent F, the stride S,
Produces a Image of size $W_2 \times H_2 \times D_2$ where:

$$
\begin{aligned}
W_2 &= \frac{W_1 - F}{S} + 1 \\
H_2 &= \frac{H_1 - F}{S} + 1 \\
D2 &= D1
\end{aligned}
$$

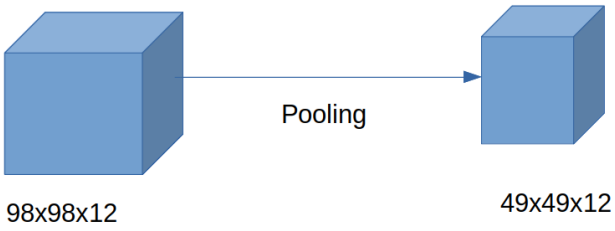


Figure 4: Pooling operation

Its mains purpose is to gradually shrink the representation's spatial size in order to reduce the number of parameters and computations in the network. Each feature map is treated separately by the pooling layer.

Introduces zero parameters since it computes a fixed function of the input For Pooling layers, it is not common to pad the input using zero-padding.

At the end it is connected to fully connected layer. The activations can hence be computed with a matrix multiplication followed by a bias offset to produce the final output.

2.1.2 Recurrent Neural Network

RNNs (Recurrent Neural Networks) are a sort of neural network design used to find patterns in a stream of data. Handwriting, genomes, text, and numerical time series are examples of such data that are frequently produced in industrial contexts.They can, however, be applied to images if they are

fragmented into a number of patches and handled as a sequence. On a higher level, RNNs are used in Language Modelling and Text Generation, Speech Recognition, Image Description Generation, and Video Tagging. The way information is transferred through the network distinguishes Recurrent Neural Networks from Feedforward Neural Networks, commonly known as Multi-Layer Perceptrons (MLPs). The RNN has cycles and transmits information back into itself, whereas Feedforward Networks transfer information through the network without cycles. This allows them to extend the functionality of Feedforward Networks to include earlier inputs, as well as the current input X_t. This distinction is depicted at a high level (See Fig. 5). It's worth noting that the option of having several hidden layers has been consolidated into a single Hidden Layer block H. Obviously, this block can be extended to include numerous hidden layers.

The process of passing information from the previous iteration to the hidden layer with the mathematical notation proposed in [24]. For that, we denote the hidden state and the input at time step t respecively as $H_t \in \mathbb{R}^{n*h}$ and $X_t \in \mathbb{R}^{n*h}$ where n is number of samples, d is the number of inputs of each sample and h is the number of hidden units. Further, we use a weight matrix $W_{xh} \in \mathbb{R}^{d*h}$, hidden-state-to-hidden-state matrix $W_{hh} \in \mathbb{R}^{h*h}$ and a bias parameter $b_h \in \mathbb{R}^{1*h}$. Lastly, all these informations get passed to a activation function ϕ which is usually a logistic sigmoid or tanh function to prepair the gradients for usage in backpropagation. Putting all these notations together yields the below 2 Equation as the hidden variable and as the output variable [33].

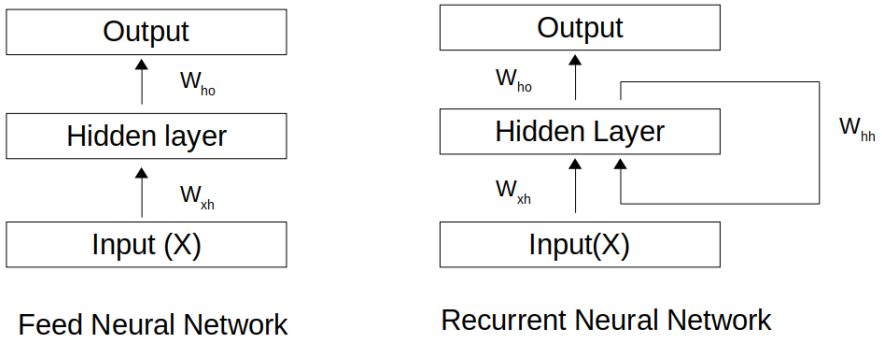


Figure 5: Differences between Feedforward NNs and Recurrent NNs

$$H_t = \phi_h(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$$

$$O_t = \phi_o(H_t W_{ho} + b_o)$$

Since H_t recursively includes H_{t-1} and this process occurs for every time step the RNN includes traces of all hidden states that preceded H_{t-1} as well as H_{t-1} itself. If we compare that notation for RNNs with similar notation for Feedforward Neural Networks we can clearly see the difference we described earlier [33]. Below Equations shows the computation for the hidden variable and output variable

$$H = \phi_h(X W_{xh} + b_h)$$

$$O = \phi_o(H W_{ho} + b_o)$$

The loops make recurrent neural networks very hard to understand. However there are not different than a normal neural network. A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Lets see what happens while unrolling the loop (See Fig. 6):

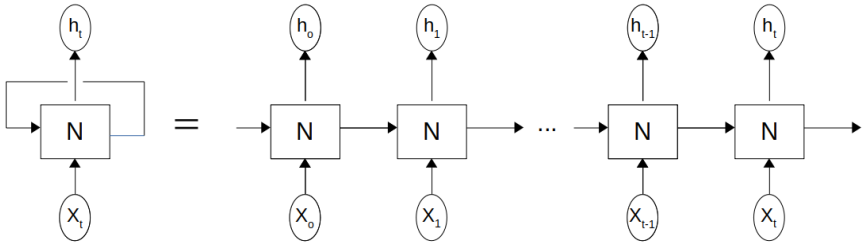


Figure 6: Recurrent Neural Network Detailed Structure

Recurrent neural networks are intricately tied to sequences and lists, as seen by their chain-like character. They're the most natural neural network architecture to employ for such data.

RNNs have had remarkable success in the previous several years when applied to a range of tasks, including speech recognition, language modelling, translation, image captioning, and so on. The list could go on and on. Andrej Karpathy's great blog post [20], The Unreasonable Effectiveness of

Recurrent Neural Networks, discusses the incredible feats that RNNs can do.

The usage of "LSTMs," a particularly specific type of recurrent neural network that performs far better than the regular Recurrent neural networks version for many tasks, is critical to these accomplishments. Almost all of the most fascinating recurrent neural network results are achieved with them. As in this paper we have used LSTMs thus we will discuss on this in the next section.

One of the allure of RNNs is the possibility of connecting earlier data to the current task, such as using previous video frames to inform comprehension of the current frame. RNNs would be immensely handy if they could accomplish this. Can they, however, do it? It is debatable.

To complete the following word in a sentence, we sometimes only need to glance at recent terms. Consider a language model that tries to predict the next word based on the previous ones. We don't need much more information to figure out what the last word in "birds soar in the sky" is – sky is a foregone conclusion. In circumstances where the distance between relevant information and the location where it's needed is short, RNNs can learn to use prior knowledge.

However, there are times when we require further context. Consider predicting the text's final word: "The computer that I had installed in the machine room on the fifth floor crashed." According to recent knowledge, the following word is most likely related to either the fifth floor or the machine room, but we need the context of computer from further back to narrow down which term. It's entirely feasible for the distance between relevant data and the point at which it's required to grow significantly.

Unfortunately, as the gap widens, RNNs lose their ability to learn to connect the dots. We are unable to learn the long-range dependencies between tokens because RNN suffers from vanishing gradients.

In theory, a special RNNs are capable of handling such "long-term dependencies." by using a memory unit with each cell. LSTMs were proposed as the special kind of RNN for handling long term dependency.

Note*- LSTMs discussion in this paper is highly inspired by post and research work shared by christopher olah [30]

2.1.3 LSTM - Long Short-Term Memory

Long Short Term Memory networks, or "LSTMs," are a type of RNN that can learn long-term dependencies. Hochreiter & Schmidhuber (1997) intro-

duced them, and numerous individuals (Felix Gers, Fred Cummins, Santiago Fernandez, Justin Bayer, Daan Wierstra, Julian Togelius, Faustino Gomez, Matteo Gagliolo, and Alex Graves) developed and popularised them in subsequent work. They are currently frequently utilised and function exceptionally effectively on a wide range of situations. LSTMs are specifically developed to prevent the problem of long-term dependency. They don't have to work hard to remember knowledge for lengthy periods of time; it's like second nature to them!

All recurrent neural networks are made up of a series of repeated neural network modules. This repeating module in ordinary RNNs will have a relatively simple structure, such as a single tanh layer (See Fig. 7).

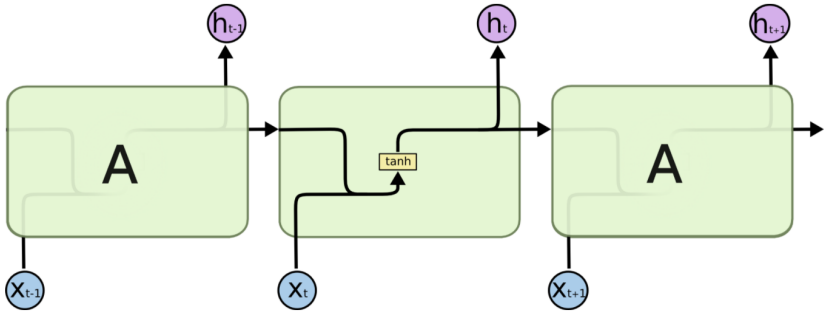


Figure 7: Repeating module in a standard RNN contains a single layer [30].

LSTMs have a chain-like structure as well, but the repeating module is different. Instead of a single neural network layer, there are four, each of which interacts in a unique way (See Fig. 8).

In the figure (See Fig. 9), Each line transmits a full vector from one node's output to the inputs of others. The pink circles denote pointwise operations, such as vector addition, and the yellow boxes denote learnt neural network layers. Concatenation occurs when lines merge, whereas forking occurs when a line's content is replicated and the copies are sent to various locations.

The cell state, the horizontal line going through the top of the diagram, is the key to LSTMs.The state of the cell is similar to that of a conveyor belt. With only a few tiny linear interactions, it flows straight down the entire chain. It's incredibly easy for data to simply travel along it unaltered.

The LSTM can delete or add information to the cell state, which is carefully controlled by structures called gates. Gates are a mechanism to selectively allow information to pass through. A sigmoid neural net layer plus a pointwise multiplication operation make them up (See Fig. 10).

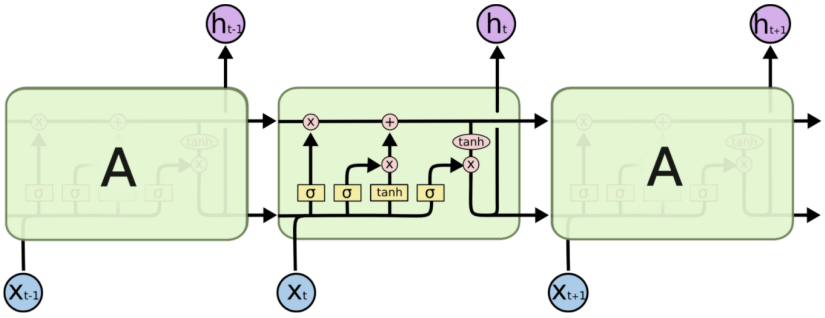


Figure 8: Repeating module in an LSTM contains four interacting layers. [30].

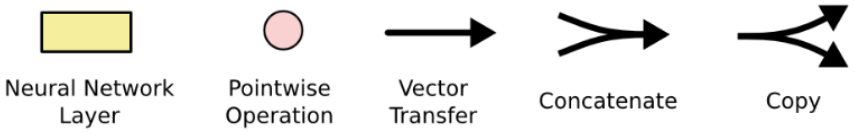


Figure 9: Repeating module in a LSTM contains four interacting layers [30].

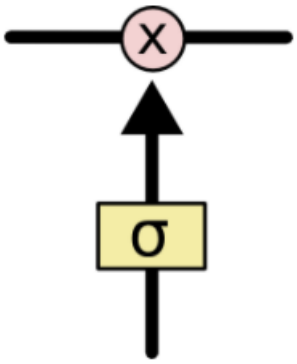


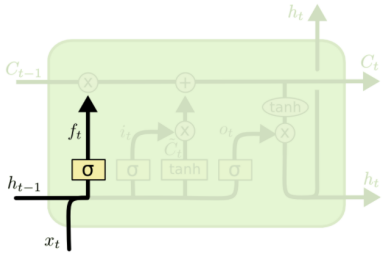
Figure 10: Sigmoid neural Net with pointwise multiplication [30].

The sigmoid layer produces values ranging from zero to one, indicating how much of each component should be allowed to pass. "Allow nothing through!" signifies a value of zero, while "let everything through!" means a

value of one. To safeguard and control the cell state, the LSTM contains three sigmoid gates.

The first stage in our LSTM is to decide which information from the cell state will be discarded. The "forget gate layer," a sigmoid layer, makes this judgement. It examines the values in h_{t-1} and x_t and returns a number between 0 and 1 for each number in the cell state C_{t-1}. A 1 indicates "totally keep this," while a 0 indicates "entirely discard this."

Let's return to our previous example of a language model attempting to anticipate the next word based on the preceding ones. In this case, the cell state may include the context of the current subject, allowing the appropriate word to be used. And when we observe a new subject, we desire to forget about the previous subject's related word.



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

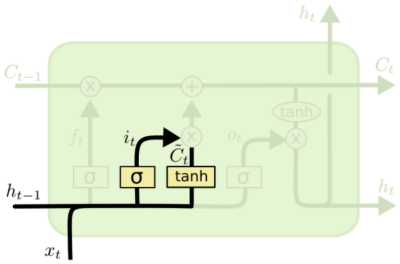
Figure 11: Forgot Gate [30].

The next stage is to figure out what new data we will store in the cell state. There are two components to this. The "input gate layer," a sigmoid layer, chooses which values we will update first. A tanh layer then generates a $\tilde{C}_t$ vector of new candidate values that could be added to the state. We will combine these two in the next step to make a state update.

In the case of our language model, we'd want to replace the old subject's gender with the new subject's gender in the cell state.

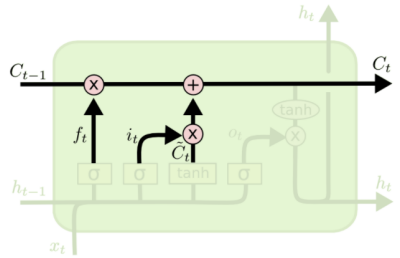
It's time to switch from the old cell state C_{t-1} to the new cell state C_t. We already know what to do because of the previous steps; now we just have to perform it.

We multiply the previous state by f_t, forgetting the items we had previously agreed to forget. Then we add $i_t \times \tilde{C}_t$ to the equation. This is the new set of candidate values, scaled by how much each state value was updated.In the instance of the language model, this is where we would remove the information about the old subject's information and replace it with the new information we decided on in the previous steps.



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

Figure 12: Input Gate: First Part[30].



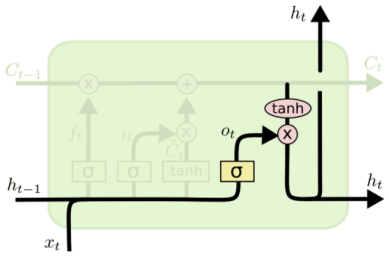
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 13: Input Gate: Second Part [30].

Finally, we must determine what we will produce. This output will be based on the state of our cells, but it will be filtered. First, we run a sigmoid layer to determine which aspects of the cell state will be output. The cell state is then passed through tanh (to force the values to be between -1 and 1) and multiplied by the output of the sigmoid gate, resulting in only the parts we choose to output.Because it just observed a subject, the language model might wish to output information relevant to a verb in case that's what comes next. It might, for example, output whether the subject is singular or plural so that we know how to conjugate a verb if that's what comes next.

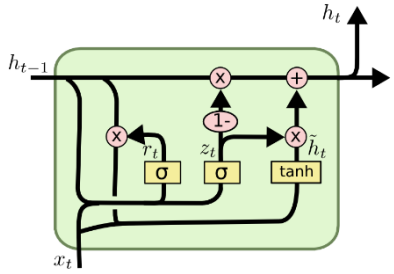
Long Short Term Memory Variants - Cho, et al. introduced the Gated Recurrent Unit, or GRU, as a somewhat more dramatic version on the LSTM (2014). It's a single "update gate" that combines the forget and input gates. It also modifies the cell state and hides the state, among other things. The resulting model is easier to understand than ordinary LSTM models, and it is gaining popularity (See Fig. 15).

LSTMs were a huge step forward in terms of what we could do with



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

Figure 14: Output Gate [30].



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 15: Output Gate [30].

RNNs. It's natural to think if there will be another major move forward. There is a next level, and it is all about paying attention. The aim is to let each stage of an RNN choose information to examine from a larger pool of data. If you are using an RNN to build a caption for a picture, it can choose a different part of the image to look at for each word it produces. In fact, Xu, et al. (2015) do just that, it could be a good place to start if you want to learn more about attention. There have been a number of pretty exciting outcomes employing attention, and it appears that there will be a lot more in the future. RNN research isn't only about attracting attention. Grid LSTMs, for example, by Kalchbrenner et al. (2015) appear to be quite promising. Work by Gregor, et al. (2015), Chung, et al. (2015), and Bayer & Osendorfer (2015) employing RNNs in generative models also appears to be quite interesting. For recurrent neural networks, the last few years have been thrilling, and the years ahead promise to be even more so.

2.1.4 Inception V3

Most state-of-the-art computer vision solutions for a wide range of tasks use convolutional networks as their foundation. Since 2014, very deep convolutional networks have been popular, with significant gains in a variety of evaluations. Although increased model size, high complexity and computational cost tend to translate to immediate quality gains for most tasks, computational efficiency and low parameter count are still enabling factors for a variety of use cases, including mobile vision and other deep neural netwrok techniques. Here we are exploring Inception V3 model which is one the ways to scale up networks in CNN which aims to utilize the added computation as efficiently as possible by suitably factorized convolutions and aggressive regularization.

In the development of CNN classifiers, the Inception network was a watershed moment. Prior to its creation, the majority of popular CNNs simply layered convolution layers higher and deeper in the hopes of improving performance.

On the other hand, the Inception network was intricate (heavily engineered). It employed numerous techniques to increase performance, both in terms of speed and precision. The network's ongoing evolution resulted in the production of multiple versions. The most widely used variations are as follows: Inception v1, Inception v2, and Inception v3 are the major three versions under the Inception umbrella.

We will briefly discuss about the architecture of Inception V3 and how it optimizes high filter convolution to a lower filter.

Convolutions with bigger spatial filters (e.g. 5x5 or 7x7) are disproportionately computationally demanding. A 5x5 convolution with n filters over a grid with m filters, for example, costs $25/9 = 2.78$ times as much to compute as a 3x3 convolution with the same number of filters. Of However, a 55 filter can catch relationships between signals between activations of units in earlier layers, so reducing the geometric size of the filters comes at a significant sacrifice in expressiveness. However, we can consider whether a multi-layer network with fewer parameters and the same input size and output depth could substitute a 5x5 convolution. Thus, Replacing the 5x5 convolution with two layers of 3x3 convolution is all it takes to slide this little network over the input activation grid [5]. (See Fig. 16)

In principle, we could even argue that any nxn convolution can be replaced by a 1xn convolution followed by a nx1 convolution, with the computational cost savings increasing drastically as n increases.(See Fig. 17) In reality, we've seen that using this factorization on early layers doesn't work

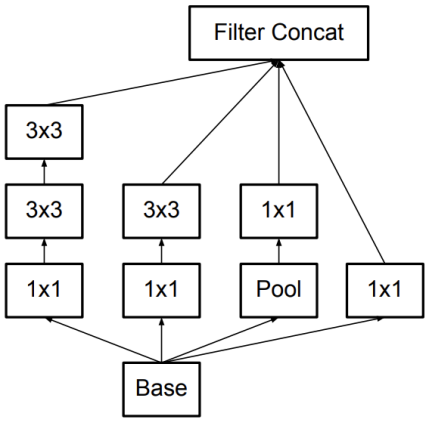


Figure 16: a) Inception modules where each 5×5 convolution is replaced by two 3×3 convolution [5]

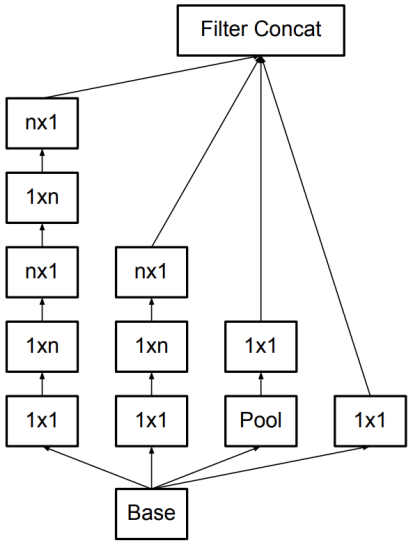


Figure 17: b) Inception modules after the factorization of the n $\times$ n convolutions. In our proposed architecture, we chose n = 7 for the 17 $\times$ 17 grid [5]

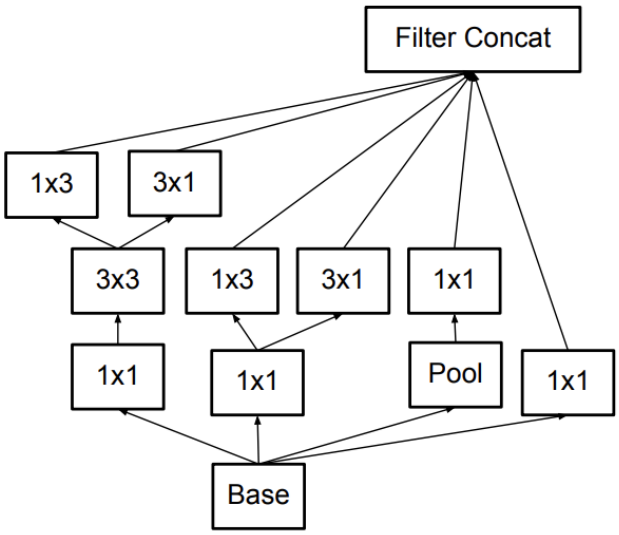


Figure 18: c) Inception modules with expanded the filter bank outputs. This architecture is used on the coarsest (8×8) grids to promote high dimensional representations [5]

well, but it does on medium grid sizes (m x m feature maps, where m is between 12 and 20). On that level, utilising 1x7 convolutions followed by 7x1 convolutions yields excellent results [5].

(See Fig. 19) shows the structure of Inception network. Note that the standard 7x7 convolution has been factorised into three 3x3 convolutions. Three typical inception modules at the 35x35 with 288 filters each for the Inception component of the network. Using the grid reduction approach, this is reduced to a 17x 17 grid with 768 filters. The factorised inception modules are then repeated five times, as seen in (See Fig. 16). The grid reduction approach reduces this to an 8x8 1280 grid.We have two Inception modules at the coarsest 8 x 8 level, as shown in figure (See Fig. 17), with a concatenated output filter bank size of 2048 for each tile [5].

type	patch size/stride	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	See Above Figure a)	35×35×288
5×Inception	See Above Figure b)	17×17×768
2×Inception	See Above Figure c)	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figure 19: The outline of the proposed network architecture. The output size of each module is the input size of the next one. Various reduction technique is being used [5]

2.1.5 Word Embeddings

A word embedding is a learnt text representation in which words with related meanings are represented similarly. One of the key breakthroughs of deep learning on challenging natural language processing problems may be this way to expressing words and documents. Individual words are represented as real-valued vectors in a predetermined vector space using word embeddings, which is a class of approaches. Because each word is mapped to a single vector and the vector values are acquired in a manner similar to that of a neural network, the technique is frequently grouped with deep learning. The word usage is used to learn the dispersed representation. This allows words that are used in similar ways to have similar representations, which captures their meaning more easily. This contrasts with the crisp but fragile representation in a bag of words approach, where individual words have various representations regardless of how they are used unless expressly handled.

Methods of word embedding from a corpus of text, learn a real-valued vector representation for a predetermined fixed sized vocabulary.

The learning process is either supervised, using document statistics, or unsupervised, employing a neural network model for some task, such as document classification [35].

Three strategies for learning a word embedding from text data is being discussed here. However we have implemented glove model in this paper.

Embedding Layer For want of a better term, an embedding layer is a word embedding learned in conjunction with a neural network model for a specific natural language processing task, such as language modelling or document categorization.

It necessitates the cleaning and preparation of document text so that each word can be encoded in a single pass. The model specifies the size of the vector space, which can be 50, 100, or 300 dimensions. Small random integers are used to start the vectors. The embedding layer is utilised on the front end of a neural network and is fitted with the Backpropagation method in a supervised manner [35].

Word2Vec Word2Vec is a statistical method for learning a solitary word embedding from a text corpus quickly and efficiently.

It was created by Tomas Mikolov, et al. at Google in 2013 as a reaction to the need to improve the efficiency of neural-network-based embedding

training, and it has since become the de facto standard for generating pre-trained word embedding.

In addition, the effort entailed analysing the learned vectors and experimenting with vector math on word representations. For example, removing the "man-ness" from "King" and replacing it with "women-ness" yields the word "Queen," which captures the parallel "king is to queen as man is to woman" [28].

GloVe GloVe, is an acronym for "Global Vectors for Word Representation." GloVe is a word vector learning approach that extends the word2vec method. Pennington et al. at Stanford developed it.

Previously, vector representations of words were created using matrix factorization techniques like Latent Semantic Analysis (LSA), which are good at using global text statistics but not as good at capturing meaning and demonstrating it on tasks like calculating analogies as learned methods like word2vec.

GloVe is a method for combining global statistics from matrix factorization techniques like LSA and local context-based learning from word2vec.

Instead of using a window to establish local context, GloVe uses data from the entire text corpus to create an explicit word-context or word co-occurrence matrix. As a result, the generated model developed may lead to improved word embeddings technique in general [18].

GloVe is a weighted least-squares model with a log-bilinear objective. The model's fundamental concept is based on the simple observation that ratios of word-word co-occurrence probabilities can encode some type of meaning. Consider the co-occurrence probability of the target words ice and steam with various vocabulary probing words. From a corpus of 6 billion words, here are some actual probabilities [18]:

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 20: Probability Ratio of different words in a 6 billion corpus[18]

Ice occurs more frequently with solids than with gases, as one might assume, but steam happens more frequently with gases than with solids. Both terms usually occur with their shared property water, while both

words appear infrequently with the unrelated word fashion. Noise from non-discriminative words like water and fashion cancels out only in the ratio of probabilities, thus large values (much greater than 1) correlate well with ice properties and small values (much less than 1) correlate well with steam properties. The probability ratio conveys some primitive form of meaning connected with the abstract concept of thermodynamic phase in this way [18].

3 Image Caption Implementation

3.1 Overview

What should be the caption for this?

Figure 21: Guess the Caption !!

The same image can be interpreted in a variety of ways like one can say "Kids are playing ball in front of a big elegant building,", or others can say "Two children are climbing on the wall of a white building while one child plays with a green ball," or simply "Three children play on a wall with a green ball."

All of these captions apply to this image, and there are likely to be more. But the point we are trying to make is that looking at an image and describing it in right language is so simple for us as people. And with this study, we are attempting to learn and build a strong model that can attempt to predict a caption for a given image.

Figure 22: Goal for this paper

3.2 Theory

We have used InceptionV3 which is one of state-of-the-art computer vision solutions based on optimized and regularized convolutional architecture for training images present in the provided Flickr Dataset. We are using transfer learning on InceptionV3. The ImageNet dataset was used to train Inception V3, which performs image classification on 1000 different classes of images. To enable transfer learning I have used automatic feature engineering technique.

All the images are first converted into a fixed length matrix of 299x299 size. During training, these converted images are then submitted to Inception model. Inception-V3 model on ImageNet Dataset and transfer-learning we are leveraging encoding technique to factorize each image to a fix-length vector. The process in pre-trained InceptionV3 model is used to provide a fixed length encoded vector representation of each images of the Flickr dataset. As a result, we simply remove the model's last softmax layer and

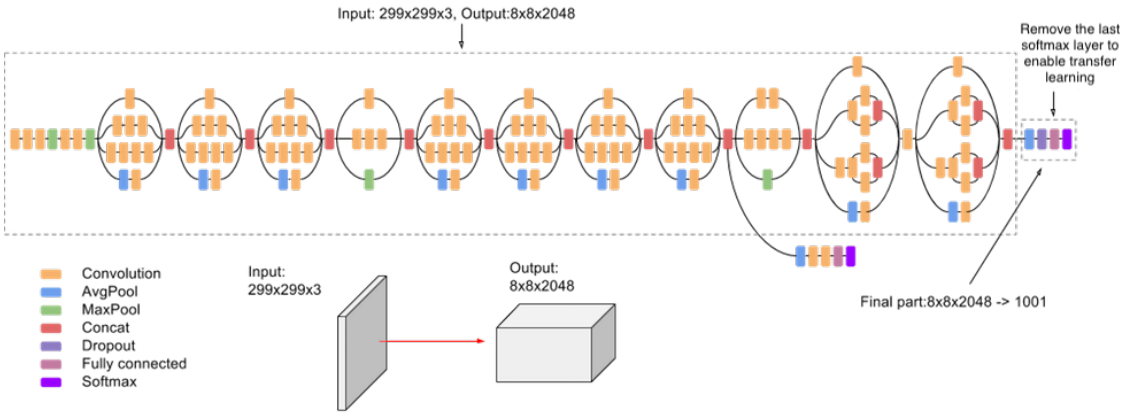


Figure 23: Transfer Learning of InceptionV3 by removing softmax layer [5] [31]

extract a 2048-length vector for each image (See Fig. 19 for detail).

We're using automated feature engineering, which attempts to assist data scientists with feature generation by creating hundreds or thousands of new features from a dataset automatically. It will allow us to concentrate on more important aspects of model building, such as producing reliable models. We have added $encode()$ & $pre\_process\_img\_inceptionv3()$ methods in $train\_inceptionV3\_model.py$ to encode images into acceptable format for Inception model. Further the model is dumped or saved in local system.

Images representation is used as independent variable and its corresponding caption is used as dependent variable. Thus caption is used to train the model and learn the prediction. However, whole sentence can not be encoded into a vector to make the prediction. I will use sequence-to-sequence of word prediction by using start-of-art model for language prediction : $LSTM$.

Now How to deal with captions, can we use some text/token representation?

Firstly, we have evaluated the max-length a caption can have, which found to be 37 words long. I am also putting a threshold on the words whose count in the whole caption dataset is less than 5 $word\_cnt\_threshold = 5$. Words with less than this threshold count is ignored. By this criteria we are getting 2984 unique words in the corpus. All filtered unique words are

mapped into collection to evaluate each word representation present in a caption which is fed into the model. As we have discussed I am using Glove Word Embedding to represent each token or word in the caption. Each word will have 200 dimension. All these implementation has been encapsulated into class WordEmbedding present in *caption_word_encoding.py*

3.3 Statistical procedure

We have tried to build a supervised model and to create datapoints we have to be fed each data while training the model. As we are using sequence of caption to build the model, we have adopted the most famous technique known as Generator Function.

Let suppose we have the following caption for a particular image (See Fig. 24)

Figure 24: Captions given below

Caption(s) given in the training dataset:
The dog is climbing out of the water with a stick
A brown dog climbing out of a pool with stick in his mouth

First we would convert the image using transfer learning of Inception V3 to an image of 2048 length. Additionally, we will build the vocabulary of captions to train our model. The vocabulary for the above two captions is $the, dog, is, climbing, out, of, water, with, a, stick, brown, pool, in, his, mouth$.

We will show in the below table how datapoint is mapped to the model. We have shown the example with the first caption [23].

Input1	Input2	Target
Image	the	dog
Image	the dog	is
Image	the dog is	climbing
Image	the dog is climbing	out
Image	the dog is climbing out	of
Image	the dog is climbing out of	a
Image	the dog is climbing out of a	pool
Image	the dog is climbing out of a pool	with
Image	the dog is climbing out of a pool with	a
Image	the dog is climbing out of a pool with a	stick

Considering one image and one of its caption we have generated 10 datapoints above. And we have around 40,455 captions, which means even if we assume that each caption is 7 token long, it will generate around 40455*7 = 283185 datapoints i.e., X.

Now we know each datapoint consist of Image and Word Encoding. Image Encoding vector length is 2048 and each word can have 37 caption mapped to 200 dimension using glove embedding technique. Thus each datapoint is made of vector of length around 2048+37*200. To fed this data into our model we have used data generator to provide us batch of data from datapoints generated during the procedure. A dropout technique is used to overcome any type of over-fitting and help it to generalize on variance between different captions.

3.4 Architecture

In this model summary we can see the following parameters distribution in different categories:

Total params: 2,422,449
Trainable params: 1,825,449
Non-trainable params: 597,000

As we are using glove embedding so we have to set trainable weight of the embedding layer to false. Thus we are getting 597,000 parameters as non-trainable weights.

We have tried with RMP Prop optimizer and Adam optimizer with different learning rate between $[10^{-4} - 10^{4}]$

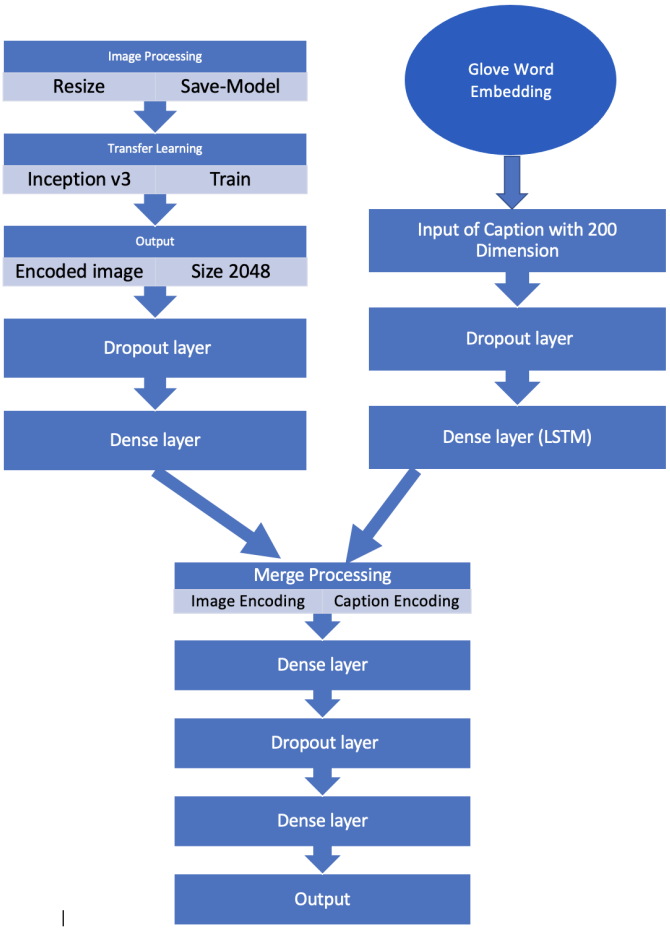


Figure 25: Model Architecture

4 Datasets

4.1 Flickr Dataset

The training dataset used for analysis in this paper was released by Ludicorp, a Vancouver-based company founded by Stewart Butterfield and Caterina Fake on February 10, 2004. Flickr Dataset ownership has changed several times and currently it is owned by SmugMug since April 20, 2018. The dataset is available on Kaggle and some university website like University of Illinois at Urbana-Champaign.

The Flickr dataset is a standard dataset for training models based on image description. This paper presents Flickr Entities, which augments the captions from Flickr with coreference chains of the same representation across different captions, and associating them bounding boxes. This dataset is originally developed by forming relation between images sharing common metadata from Flickr. Edges between images are formed from the same location, submitted to the same gallery, group, or set, images sharing common tags, images taken by friends, etc. The original images was collected from PASCAL, ImageCLEF, MIR, and NUS-wide [26].

The dataset used in the research paper is provided with a total of 8091 distinct images and each image is associated with multiple description. For the total of 8091 images of training records, we have 40455 distinct caption to build the model. This shows that for each image we have around 5 distinct description and thus it increases the training data point from 8091 to 40455. The dataset is publicly available at https://shannon.cs.illinois.edu/.

Figure 26: One of the random image picked from the Flickr dataset

Caption(s) provided:
1. A beagle is playing with a tennis ball.
2. A puppy dog is playing with a tennis ball.
3. A puppy is playing with a tennis ball on a well-kept path lined with groomed bushes.
4. A puppy plays with a tennis ball on a stone path.
5. A white-footed beagle plays with a tennis ball on a garden path .

4.2 Data Preparation

Once the training dataset is downloaded, we will get a captions.txt which contains captions of all the different images. The images are in standard jpg format. The captions.txt looks as follows:
Every Entity represents image Id and its corresponding caption.

1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg	A girl going into a wooden building .
1000268201_693b08cb0e.jpg	A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg	A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg	A little girl in a pink dress going into a wooden cabin .

Figure 27: View of dataset file

The data is run through preprocessing steps like tokenizing sentences into words, converting it to lower case and removing the common stopwords and any numeric character present in the sentence. A exception has been made to include 'a' as one the token.

After performing all the data processing and cleaning steps we have projected the frequency of top 30 words. X-axis represents the word and Y-axis represents the count of each word.

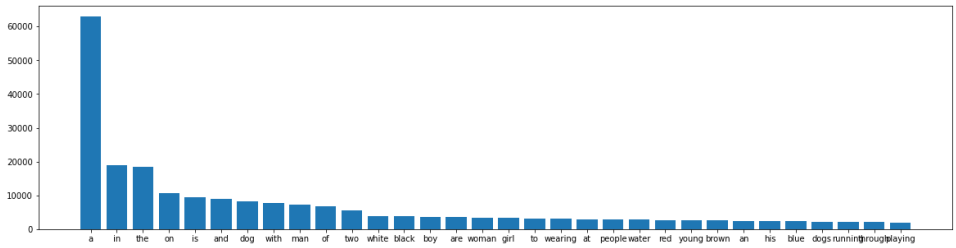


Figure 28: Frequency of top 32 words from the vocabulary

There is 8764 unique words in the data-set when calculated on 40455 sentences.

In this paper while performing Word cloud Analysis. Some of the words such as "dog","taxi","record" can be interpreted as most common words as they are present in dominantly on the cloud of words. We can see few words in smaller size in like "apartment", "cast" etc., which are quite relevant.

A start and end sequence is appended to each Caption sentence. I've included startseq, which is a start sequence token that will be put to the beginning of every caption, and endseq, which is an end sequence token that will be added to the end of every caption in the dataset.

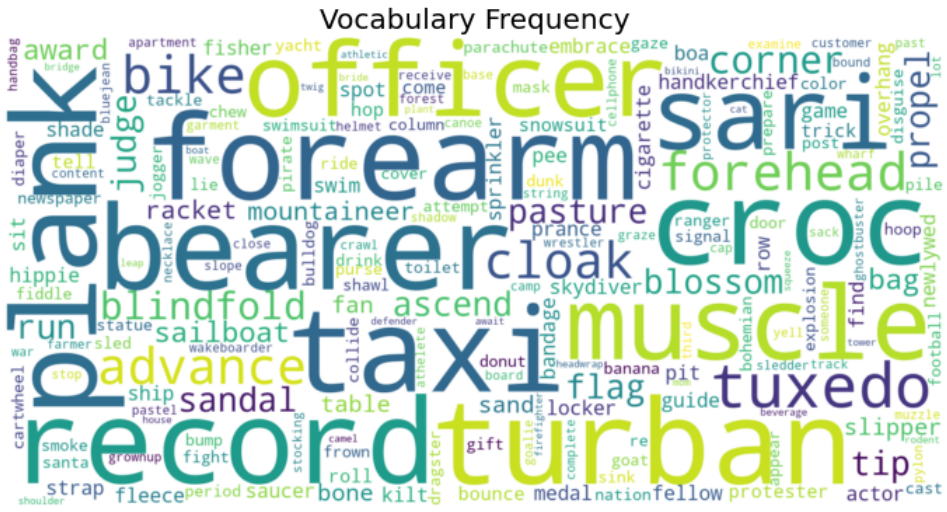


Figure 29: Word Cloud representation of the vocabulary

4.3 Data generation

Many of these terms will only appear a few of times, perhaps once or twice. We don't want all of the terms in our vocabulary because we're building a predictive model. We are just interested in words that are more likely to emerge or are widely used. This makes the model less prone to errors and more resistant to outliers. Thus we will only consider words which have appeared at-least 5 times or more (have also tried with 10, 20 frequency). All the data generated during the process in intermediate steps have been dumped in the system so that we don't have to perform the same steps over and over again in-case of system shutdown or loosing data from RAM. Glove Embedding is used to generate token representation. The dimension used is of size 200.

4.4 Test DataSet

Flick Images has been used to test the model. We are using 1000 images from the dataset to predict the caption of the images and then evaluated different metrics like bleu score to understand how the model scores. If the model is score is less than .5 then we can assume that model is guessing the image and have little knowledge about the pixels of the images.

5 Computational Experiments

In order to compare this model, we studies a series of tests which is used to measure the effectiveness of our Language based model, utilising a variety of metrics, data sources, and model architectures.

5.1 Evaluation Metrics

Although it's not always evident whether a caption is successful or not when presented with a picture, previous language-based models have offered a number of evaluation measures. The most reliable (though time-consuming) method is to have raters assign a subjective score to each description given the image. Following the principles established in [27], which asks graders to rate each generated phrase on a scale of 1 to 4, scientists have employed this technique in most language-based papers to underline that some of the automatic metrics do actually correlate with this subjective score.

An Amazon Mechanical Turk experiment is one of the most popular for this metric. Each image is rated by many workers in this method. The average level of worker agreement is calculated. They just average the scores and record the average as the score if there is a disagreement. Bootstrapping (resampling the findings with replacement and determining means/standard deviation over the resampled results) is used for variance analysis. It reports the percentage of scores that are more than or equal to a series of predetermined thresholds, similar to [27].

The BLEU score [22], which is a type of precision of word n-grams between generated and reference sentences, has been the most often used measure in the image description literature thus far. Despite some obvious flaws, this statistic has been proven to correspond well with human ratings.

We have implemented the BLEU score using the Python Natural Language Toolkit library, or NLTK, which one use to compare your generated text to a reference. It is measured by finding the "max number of times n-gram appeared in reference"

5.2 Random Internet Images

Figure 30: Experiment-1 [4]

Caption predicted : a young boy is swimming underwater in a pool

Figure 31: Experiment-2 [14]

Caption predicted : a white dog is jumping over a log

We can see that the model is able predict random images from internet and the captions generated is somewhat related or relevant with the images

provided to the model.

5.3 BLEU Metric

5.3.1 Overview

Machine translation reviews by humans are thorough yet costly. Human evaluations might take months to complete and need non-recyclable human labour. BLEU is a proposed technique for evaluating automatic machine translation that is quick, cheap, and language-independent, has a high correlation with human evaluation, and has a low marginal cost per run. A given source statement usually has a lot of "perfect" translations. Even when the same words are used, the translations may differ in word choice or order. Humans, on the other hand, can tell a good translation from a terrible one. Similarly, a BLEU algorithm's main responsibility is to compare the predicted sentence's n-grams to the reference translation's n-grams and tally the number of matches. Position has no bearing on these matches. The better the candidate translation, the more matches there are [22].

5.3.2 Statistical procedure

The well-known precision measure serves as the foundation of our metric. To calculate precision, we count the number of candidate translation words that appear in any reference translation and divide by the total number of words in the candidate translation. Unfortunately, machine translation systems have a tendency to over-generate reasonable words, resulting in unlikely but high-precision translations. Thus BLEU proposes a new modified n-gram precision technique. For each n-gram, it is calculated in several steps [22].

Let us consider an example with 1 candidate translations having 3 reference translation and understand how modified n-gram precision is calculated.

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.

To calculate modified precision score we will the perform the following steps: Count the maximum number of times a candidate n-gram appears in a single reference translation. It is referred as Count.
Count how many times a candidate n-gram appears in each reference sentence. We calculate Ref 1 count, Ref 2 count, and Ref 3 count since we have three reference translations.
In any reference count, take the maximum number of n-grams occurrences. It is referred as Max Ref Count.
Take the Count and Max Ref Count's lowest value. It's also called as Count clip since it divides each candidate word's total count by its maximum reference count.
Then sum all these clipped counts. To get the adjusted precision score, divide the clipped counts by the total (unclipped) number of candidate n-grams [22] [21].

Candidate Count	Count	Ref1 Count	Ref2 Count	Ref3 Count	Max Ref Count	Clip Count
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	2	0	0	2	2
the	3	1	4	4	4	3
military	1	1	1	0	1	1
always	1	0	1	1	1	1
obeys	0	0	0	0	0	0
commands	1	1	0	0	1	1
of	0	0	1	1	1	0
party	1	0	0	1	1	1
18						**17**

Clip Count for unigram

Candidate Count	Count	Ref1 Count	Ref2 Count	Ref3 Count	Max Ref Count	Clip Count
It is	1	1	1	1	1	1
is a	1	1	0	0	1	1
a guide	1	1	0	0	1	1
guide to	1	1	0	0	1	1
to action	1	1	0	0	1	1
action which	0	0	0	0	0	0
which ensures	0	0	0	0	0	0
ensures that	1	1	0	0	0	1
that the	1	1	0	0	1	1
the military	1	1	1	0	1	1
military always	0	0	0	0	0	0
always obeys	0	0	0	0	0	0
obeys the	0	0	0	0	0	0
the commands	0	0	0	0	0	0
commands of	0	0	0	0	0	0
of the	1	0	1	1	1	1
the party	1	0	0	1	1	1
17						**10**

Clip count for bigram

Figure 32: Modified Precision Score [21]

The modified precision score for the unigram is 17/18.

The modified precision score for bi-gram is 10/17.

To deal with short translations, BLUE applies a brevity penalty. Brevity Penalty is an exponential decay and its formulae is

$$p_n = \frac{\sum_{C \in \{Candidates\}} \sum_{n\text{-}gram \in C} Count_{clip}(n\text{-}gram)}{\sum_{C' \in \{Candidates\}} \sum_{n\text{-}gram' \in C'} Count(n\text{-}gram')}.$$

P is modified precision score

Figure 33: Modified Precision Score Formula [21]

$$\begin{aligned} BrevityPenalty &= 1, && if c > r \\ &= e^{1-r/c}, && if c <= r \end{aligned}$$

Finally, we evaluate BLEU as:

$$BLEU = BrevityPenalty \times \exp(\sum_{n=1}^{N} w_n \times \log_2 p_n)$$

Using n-grams up to length N and positive weights w_n summing to one, we first compute the geometric average of the adjusted n-gram precisions, p_n.

5.3.3 BLEU score on Test Data

The Python Natural Language Toolkit package, or NLTK, has a BLEU score implementation that we have used to compare generated captions to a reference of captions. It has been evaluated for uni-gram, 2-gram, 3-gram and 4-gram language model.

After performing the experiment on the test result, the bleu score decreases drastically as we move for higher-gram model. For the test data with 1000 random images, we got the Bleu score as follows:
Uni-gram - .86
2-gram - .66
3-gram - .36
4-gram - .2

```
sentence_bleu(reference, candidate, weights=(1, 0, 0, 0))
sentence_bleu(reference, candidate, weights=(0, 1, 0, 0))
sentence_bleu(reference, candidate, weights=(0, 0, 1, 0))
sentence_bleu(reference, candidate, weights=(0, 0, 0, 1))
```

Figure 34: BLEU Score Implementation

Figure 35: n-gram vs BLEU Score

5.4 Model Predicts Poorly

For some cases model can predict wrongly or poorly by identifying little or no information from the images. In those cases we can say our model could be either over-fitting or under-fitting the training dataset. Generally when we see over-fitting we try to apply more regularization to lower the degrees of freedom of the model. As we increase the constraint or decrease the epochs the model will find harder to over-fit. On the other hand in case of under-fit, we can reduce constraints or increase the complexity of the model to generalize better. But the most important criteria to reduce overfitting and underfitting simultaneously is by increasing the size of training data. By increasing the training data we are helping the model to learn on various data and generalize things with confidence.

5.4.1 Out-Of-Distribution Random image

Figure 36: Royal Holloway University of London [3]

Caption generated:

```
In [28]: prepare_test_model('test.txt')
Total words 8766 ::: and threashold words 2984
Considered vocabulary Size : 2984
Time taken in seconds = 0.25682806968688965
Caption predict : a group of people are standing on a bench in front of a building
```

Figure 37: caption : a group of people are standing on a bench in front of a building

The model has predicted : "a group of people are standing on a bench in front of a building".
As we can see clearly that model is generalizing royal holloway university with a normal building. It is a clear case of under-fitting as the model has not learned anything about this building. We can increase the training set and can add these images to help the model to predict and distinguish among different buildings.

5.4.2 Out-Of-Vocabulary Random image

Figure 38: Random Image from COCO Dataset [6]

Caption(s) given:
Two boaters are white water rafting through rough currents.
Two people in a small boat in a body of water.
There are people on a boat tube in the water.
Two people riding a raft through some waves.
Two people in a canoe in some rapids.

Caption generated:

```
In [29]: prepare_test_model('test.txt')
Total words 8766 ::: and threashold words 2984
Considered vocabulary Size : 2984
Time taken in seconds = 0.24413323402404785
Caption predict : a man is kayaking in the ocean
```

Figure 39: caption : a man is kayaking in the ocean

The model has predicted : "a man is kayaking in the ocean".

While training our model we have rarely used rafting i.e., rafting has been under-represented in the model. However kayaking is being used with multiple images and thus model is over-fitting with kayaking word. To mitigate this problem we have to fed more images of rafting into the model so that it can predict correctly about a rafting image.

Here we have candidate and reference captions. So we can try to find the BLEU score.

Sentence 1-gram: 0.06666666666666667

Sentence 2-gram: 2.2250738585072626e-308

Sentence 3-gram: 2.2250738585072626e-308

Sentence 4-gram: 2.2250738585072626e-308

BLEU score for uni-gram(1-gram) is very low which shows model has done a very bad prediction.

6 Conclusion

This paper offers a real-world picture captioning model build on a variety of figures and captions. In this paper we learned how we can use a CNN(Convolutional Neural Network) based architecture to encode image representation and a RNN(Recurrent neural network) based model to capture the context of a sentence. Image captioning is an example of these two field, in which we made the computer to learn and to interpret the visual information of an image by using one or more phrases. However the model performance and accuracy could be increased by using larger dataset like MOCOCO or Pascal but this will significantly increase the training time of the model and would also need significant higher amount of computer's CPU/GPU memory.

References

[1] Avimanyu Bandyopadhyay. *https://commons.wikimedia.org/wiki/File:AI-ML-DL.png.*

[2] Yoshua Bengio. *Practical recommendations for gradient-based training of deep architectures.* arxiv:1206.5533 edition, 2012.

[3] ©2020 Copyright by applyzones.com. *https://applyzones.com/royal-holloway-university-of-london-sch778.*

[4] © 2021 by The President and Fellows of Harvard College. Simplify your workout with lap swimming.

[5] Sergey Ioffe Jonathon Shlens Zbigniew Wojna Christian Szegedy, Vincent Vanhoucke. *Rethinking the Inception Architecture for Computer Vision*. arxiv:1512.00567 edition, 2015.

[6] COCO Consortium Copyright (c) 2015. *https://cocodataset.org/*.

[7] Copyright © 2021 NVIDIA Corporation. wearable-device-for-blind-visually-impaired.

[8] Stanford Edu. convolutional-networks.

[9] L. Fei-Fei, A. Iyer, C. Koch, and P. Perona. *What do we perceive in a glance of a real-world scene?* Journal of vision, New York, 7(1):10 edition, 2007.

[10] K. Fukushima. *Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position*. 193–202. doi:10.1007/bf00344251 edition, 1980.

[11] G. E. Hinton. *Learning multiple layers of representation*. At the wayback machine, trends in cognitive sciences edition, 2018.

[12] Geoffrey E Hinton. *Learning multiple layers of representation*. Issn 1364-6613 edition, 2007.

[13] Osindero S. Teh Y. W. Hinton, Geoffrey E. *A Fast Learning Algorithm for Deep Belief Nets*. Neural computation. edition, 2006.

[14] GOLLYKIM / GETTY IMAGES. Why it's important to let your dog sniff, unrushed, during walks.

[15] Alexey Ivakhnenko. *Polynomial theory of complex systems*. Ieee transactions on systems, man and cybernetics. smc-1 (4): 364–378. doi:10.1109/tsmc.1971.43083207 edition, 1971.

[16] Lapa V. G Ivakhnenko, A. G. *Cybernetics and Forecasting Techniques*. Isbn 978-0-444-00020-0. edition, 1967.

[17] Moolayil J. An introduction to deep learning and keras. in: Learn keras for deep neural networks.

[18] Christopher D. Manning Jeffrey Pennington, Richard Socher. *GloVe: Global Vectors for Word Representation*. Computer science department, stanford university, stanford, ca 94305 edition, 2014.

[19] A. Karpathy, A. Joulin, and L. Fei-Fei. *Deep fragment embeddings for bidirectional image sentence mapping.* Nips edition, 2014.

[20] Andrej karpathy. The unreasonable effectiveness of recurrent neural networks.

[21] Renu Khandelwal. *https://towardsdatascience.com/bleu-bilingual-evaluation-understudy-2b4eab9bcfd1.* 2020.

[22] Todd Ward Kishore Papineni, Salim Roukos and Wei-Jing Zhu. *BLEU: A method for automatic evaluation of machine translation.* Ibm t. j. watson research center edition, 2002.

[23] Harshall Lamba. *https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8.* 2018.

[24] Yann LeCun. *Slides on Deep Learning Online.* At the wayback machine edition, 2016.

[25] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. *Gradient-Based Learning Applied to Document Recognition.* Proceedings of the ieee edition, 1998.

[26] Jure Leskovec. *https://cs.stanford.edu/ jure/.*

[27] P. Young M. Hodosh and J. Hockenmaier. *Framing image de- scription as a ranking task: Data, models and evaluation metrics.* Jair, vol. 47, edition, 2013.

[28] Tomas; et al. Mikolov. *Efficient Estimation of Word Representations in Vector Space.* arxiv:1301.3781 edition, 2013.

[29] Jojo John Moolayil. *https://towardsdatascience.com/a-laymans-guide-to-deep-neural-networks-ddcea24847fb.*

[30] Christopher Olah. *Understanding LSTM Networks.*

[31] Facebook AI Research. *https://paperswithcode.com/method/inception-v3.*

[32] Sumit Saha. *https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.*

[33] Robin M. Schmidt. *Recurrent Neural Networks (RNNs): A gentle Introduction and Overview*. arxiv:1912.05911v1 [cs.lg] edition, 2019.

[34] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. *Show and Tell: A Neural Image Caption Generator*. In arxiv:1411.4555 edition, 2015.

[35] Graeme Hirst Yoav Goldberg. *Neural Network Methods in Natural Language Processing*. Morgan claypool publishers edition, 2017.

List of Figures

7 Appendix

7.1 Professional Issues

In this paper, we have proposed a model which is based on CNN and LSTM architecture to generate caption for the images fed into them. Image clas-

sification methods benefit from convolutional neural networks (CNN) since they can learn highly abstract features and work with fewer parameters. Overfitting, explosive gradients, and class imbalance are the most common problems encountered while using CNN to train the model. The model's performance may be harmed as a result of these flaws. Because CNNs are essentially a black box models, it is impossible to determine exactly what a given CNN has learnt for a given Images. Deep learning is still developing strategies to better grasp and comprehend its trainable parameters. Advance CNN Learning approaches have grown through time as a result of many experiments and the subsequent development of foundation math to support the evidence. On the other hand, different gating mechanisms in an LSTM, are meant to allow it to recall information over long distances by learning which sections of the input sequence are more significant, which is why the contrast is so perplexing. However, in a complicated prediction assignment when you are attempting to build anything complex (such as a sentence), you must be able to pay attention to different portions of the inputs (be it a sentence, an image, or whatever) as you begin to generate that complex output. All RNNs, including LSTMs, is a parametric model because it has a fixed amount of memory to encode the inputs. The LSTM algorithm combines the complete meaning of the text and encodes everything significant about it in one or more hidden layers of fixed size. As a result, we would lose being able to 'attend' to the parts of the sentence that are pertinent to the current part. With a standard LSTM, all we have is one or more vectors that encode the full sentence all at once, but the neural attention mechanism allows us to achieve this. Traditionally, attention methods have been used to generate complicated outputs such as captions for images, sentence translations, and other complex generative tasks that need a sequence of output prediction steps rather than merely predicting a single label or class.

This research explicitly provides a less theoretical example of a possibly harmful concept. We proposed a caption predictor that would predict captions based on training photos provided to the model. When a test image that is extremely distinct from any training samples is encountered, the problem can emerge. This might easily happen if there aren't enough photos available during training. When this happens, the model is likely to produce a random prediction, leading to the algorithm producing a meaningless caption. In this study, we take specific steps to minimise this issue by computing Bleu Score during testing to assess the model's prediction skills. However, we may still run into this problem because it is extremely difficult to collect all conceivable photos.

When constructing crucial captioning systems, it's extremely important to think about the worst-case scenarios in terms of model performance. This includes incorporating safeguards to ensure that the model never produces harmful predictions. It also means that a lot of effort should be put towards developing adversarial training instances in order to figure out where the model might go wrong. It should not be utilised for inference in circumstances where these examples are expected to occur until enough empirical work has been done to demonstrate that the model has been adjusted to learn appropriate answers for these examples.

Assessing model performance on noisy cases would be an additional investigation that would be valuable in gaining a deeper understanding of model uncertainty. This appears to be a method of ensuring that the probability scores are correct. It should be noted that as more noise is supplied, the probability scores decrease. This translates to the model's predictions becoming less certain. Experiments to prove that this is the case would be a useful addition to this research. If it isn't, a more reliable approach for computing metrics, such as METEOR or CIDER, should be tried.

7.2 How to Use my project

See Fig. 40 shows the directory structure of my project submission. The project paper has been added in the repository as report.pdf. Codes are segregated based on the module and task it is performing. All functionality and the result can be found in .py files. There are 8 python files each correspond to different task required to build and test the model.

This project is compatible and build with Python 3.9.1, tensorflow 2.7.0. It is tested on Mac(Mojave and Big Sur), Windows(10) and Linux(Mint) systems. Please also download other dependent libraries to test and generate the results, like matplotlib, nltk(Natural Language Toolkit) and InceptionV3 from keras.

The code to pre-process the data/caption and generate figures for different vocabularies count can be found in *load_data.py* and *pre_process_data.py*. Then the mapping is stored in the local using python file *save_training_images.py*. All the images to be used during training is encoded into 2048 size vector with the help of InceptionV3 model is present in file *train_inceptionV3_model.py*. The Word Embedding technique used to present word into vectorization form is encapsulated in WordEmbedding class present in python file *caption_word_encoding.py*. The model proposed in this paper is prepared and written in *Image_Caption* class in *model_build.py* file. A helper file *load_pre_processed_data.py* is used to

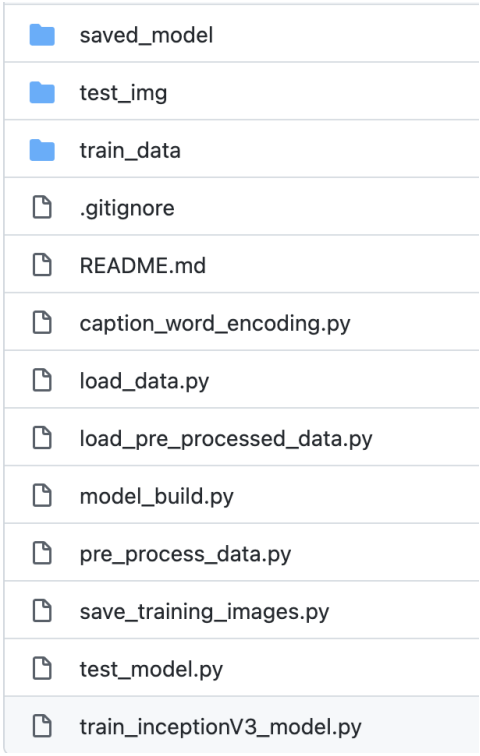


Figure 40: Repository Structure

generate captions with start and end padding. And then at the end to predict any image we can execute *test_model.py*.

Folder *train_data/∗* contains the training data used to build this model. It consist of *caption.txt* which comprises of 2 columns "image name" and and its corresponding caption and an images folder which contains all the images used to train the model. The picture (See Fig. 41) shows the structure. Other files like *images_name.txt* and *descriptions.txt* are generated as the pre-processing steps and thus saved in *train_data* folder.

Add a glove embedding file of 200 dimension (See Fig. 42). It is required to convert captions words into vector form. It can be downloaded from this url : *https* : *//www.kaggle.com/incorpes/glove*6*b*200*d*

7.2.1 Build with scratch InceptionV3 Encoding

If we are training the model from beginning we have to first encode the images using transfer learning on inception V3 and save the encoding in a

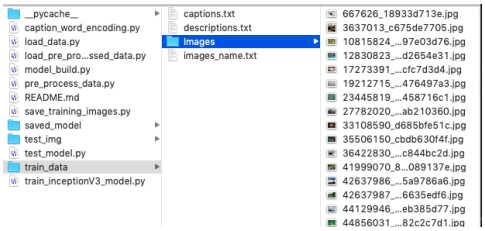


Figure 41: Train Data Structure

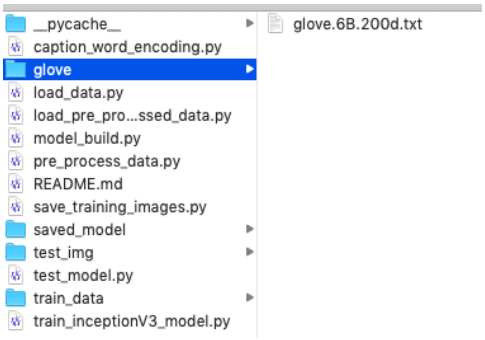


Figure 42: Golve Folder Structure

file named *train_data/encoded_flickr_training_images.pkl*. It will be used from next time to reduce the training time. Put all the training images and captions into *train_data* folder as shown in Figure (See Fig. 41). And then run the *test_model.py* with this section opened for building (See Fig. 43)

```
#To re-train the model either by building again or load
#if {Build from scratch}

training_images()
model = build_the_model()

#else
#model = get_saved_model()
```

Figure 43: Build From Scratch with InceptionV3 Encoding

7.2.2 Build using saved InceptionV3 image encoding

I have uploaded the image encoding file here *https://drive.google.com/file/d/1E_zTNqBUmARQbnhZ4pI7tJBsIkH2TNom/view?usp=sharing*, Please download it and save it in the train folder, it has the encoding trained on flickr dataset with 8091 images. (The file is bigger than

50MB so could not be uploaded to github). Then run the *test_model.py* with the other section opened for model preparation (See Fig. 44)

```
#To re-train the model either by building again or loading the exi
#if

#training_images()
#model = build_the_model()

#else{Build using saved InceptionV3 image encoding}
model = get_saved_model()
```

Figure 44: Build From Saved InceptionV3 Encoding

7.2.3 Saved Model

In both cases our optimized architecture/model is saved in *saved_model* folder (See Fig. 45), which is actually the trained model presented in this paper. It has been uploaded into github so that we can test the model and generate the prediction in less time if we have downloaded the saved "InceptionV3 image encoding" file as mentioned in the above section.

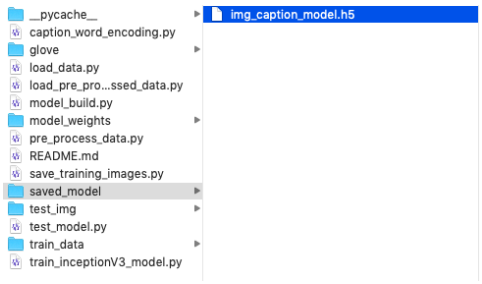


Figure 45: Saved Model Directory

7.2.4 Execution Step

To execute and predict the image run the file *test_model.py* as mentioned above and then execute the method *prepare_test_model*('*test.txt*'). The parameter should be the file name present in the *test_img* folder which contains the image file names to be considered for prediction (See Fig. 47).

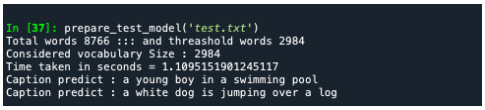


Figure 46: Execution Command and Response

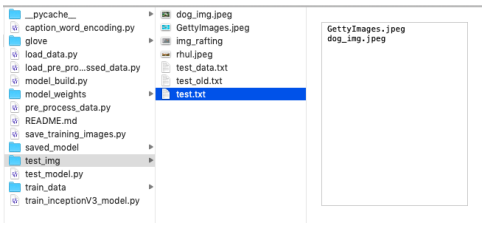


Figure 47: Execution Directory Structure

7.3 Self Assessment

I found the endeavour to be quite fulfilling. Over the course of the project, I worked on it for at least 600 hours. I learned a lot of knowledge about different Image encoding technique by using transfer-learning on Inception Models. Additionally I have also got opportunity to understand and explore image-to-text translation model and the mechanism behind it with the help of word embedding. Also explored different data-sets and metrics to analyze a language model. My greatest strength, I believe, was that I worked on the project every day, so I never felt overwhelmed. I want to continue working to enhance this model by applying it on different popular datatset like COCO and PASCAL for training and then analyzing it using metrics like METEOR and CIDER.