Dissertation on

"Deep Learning based System for Image Retrieval"

Submitted in partial fulfillment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

Submitted by:

Parashara Ramesh	**01FB15ECS202**
Shahid Ikram	**01FB15ECS274**
Sumanth S Rao	**01FB15ECS314**

Under the guidance of

Internal Guide
Prof. Chitra G M
Assistant Professor,
PES University

January – May 2019

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

‘Deep Learning based System for Image Retrieval’

is a bonafide work carried out by

Parashara Ramesh	**01FB15ECS202**
Shahid Ikram	**01FB15ECS274**
Sumanth S Rao	**01FB15ECS314**

In partial fulfilment for the completion of eighth semester project work in the Program of Study Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period Jan. 2019 – May. 2019. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 8th semester academic requirements in respect of project work.

Signature Prof. Chitra G M Assistant Professor	Signature Dr. Shylaja S S Chairperson	Signature Dr. B K Keshavan Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1. _____	_____
2. _____	_____

DECLARATION

We hereby declare that the project entitled **"Deep Learning based System for Image Retrieval"** has been carried out by us under the guidance of Prof. Chitra G M and submitted in partial fulfillment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester January – May 2019. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

01FB15ECS202	**Parashara Ramesh**
01FB15ECS274	**Shahid Ikram**
01FB15ECS314	**Sumanth S Rao**

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without mention of people who made it possible, because ‘Success is the abstract of hard work and perseverance, but steadfast of all is encouragement and guidance. Therefore, we acknowledge all those whose guidance and encouragement served as a beacon light and crowned our efforts with success.

We express our sincere gratitude to our guide Prof. Chitra G M, Professor, Department of Computer Science and Engineering, PES University for her able guidance, continual source of encouragement, valuable suggestions and indispensable assistance throughout this project.

We also express our deepest thanks to Mr. P. N. Anantharaman, AI consultant, JN Research who in spite of being extraordinarily busy, took time to hear and guide us whenever we asked for, throughout the project.

We especially thank the project coordinator, Prof. Preet Kanwal, Assistant Professor, Department of Computer Science and Engineering, PES University and Prof.Sangeeta V I, Assistant Professor, Department of Computer Science and Engineering, PES University for putting in time and effort into organizing our evaluations and making sure they went smoothly. They were ever willing to support and address all our concerns and queries throughout the course of this project.

Our earnest gratitude to our Chairperson, Dr. Shylaja S S, Department of Computer Science and Engineering, PES University for giving us this wonderful opportunity to explore, understand and tackle this exciting problem and for furnishing all the resources required for the completion of the project.

We have immense pleasure in expressing our gratitude to Dr. B K Keshavan, Dean of Faculty, for providing us a convivial environment and framework throughout the progress of this project. We express our heartfelt thanks to the Vice Chancellor, Dr. K N Balasubramanya Murthy for providing all the facilities for the successful completion of this project.

We would also like to thank Prof. D Jawahar, Pro Chancellor, PES University for his moral support towards completing our project work. With due respect, we would also like to thank

our Chancellor, Dr. M R Doreswamy for providing us a platform for development and innovation that has occurred for fulfillment of this research work.

Parashara Ramesh,
Shahid Ikram,
Sumanth S Rao

ABSTRACT

In the modern era, an enormous amount of digital pictures, from personal photos to medical images, is produced and stored every day. It is more and more common to have thousands of photos sitting in our smart phones; however, what comes with the convenience of recording unforgettable moments is the pain of searching for a specific picture or frame.

How nice it would be to be able to find the desired image just by typing one or few words to describe it? In this context, automated caption image retrieval is becoming an increasingly attracting feature, comparable to text search.

In this project, we consider the task of content-based image retrieval and propose effective neural network-based solutions for that. Specifically, the input to our algorithm is a collection of raw images in which the user would like to search, and a query sentence meant to describe the desired image. The output of the algorithm would be a list of top images that we think are relevant to the query sentence.

In particular, we obtain a representation of the sentence that will be properly align with the corresponding image features in a shared high dimensional space. The images are found based on nearest neighborhood search in that shared space.

TABLE OF CONTENTS

Chapter No.	Chapter and Section Name	Page No.
1.	Introduction	1
	1.1 Prologue	1
	1.1.1 Existing Solutions to the Problem	1
	1.1.2 Google's image retrieval system - Tag based	1
	1.1.3 Content Based Image Retrieval	2
	1.2 Overview	3
	1.3 Scope	3
	1.3.1 The basic Scope	3
	1.3.2 Coverage of the project	4
	1.3.3 The highlights of the solution	4
	1.3.4 Scope of the components involved in the model	5
	1.3.5 Limitations	5
	1.4 Objective	5
	1.5 Outcomes	6
2.	Research Background	7
	2.1 Literature Survey	7
	2.1.1 Scaling up to large vocabulary image annotation. Jason Weston, Samy Bengio and Nicolas Usunier , 2013	7
	2.1.2 Show, Attend and Tell	8
	2.1.3 Look, Imagine and Match: Improving Textual-Visual Cross-Modal Retrieval with Generative Models - Jiuxiang Gu, Jianfei Cai, Shafiq Joty, Li Niu, Gang Wang	10
	2.1.4 CNN features are also great at unsupervised classification - Joris Gu´erin, Olivier Gibaru, St´ephane Thiery, and Eric Nyiri	13
3.	Methodology	17
	3.1 Proposed Approach	17

	3.2 High Level System Architecture	17
	3.3 Experiments conducted	21
	3.4 Training Procedure	22
	3.5 A deeper dive into models	23
	3.5.1 Baseline models	23
	3.5.1.1 Resnet + GRU +fasttext	24
	3.5.1.2 Inception + GRU +fasttext	26
	3.5.2 Bidirectional model	26
	3.5.2.1 Inception + Bidirectional LSTM +fasttext	27
	3.5.3 Self attention models	28
	3.5.3.1 Self Attention (Inception) + Bidirectional LSTM + fasttext (Self attention only for Image)	29
	3.5.3.2 Inception + Self Attention (fasttext) (Self Attention only on Caption side)	31
	3.5.3.3 Self Attention (Inception) + Self Attention (fasttext) (Self attention to both caption as well as image)	33
4.	Environment Requirements	35
	4.1 Hardware Requirements	35
	4.2 Software Requirements	37
	4.3 Data Requirements	37
5.	Demonstration of Outcome	39
6.	Proposed Approach	42
	6.1 Research Approach	42
	6.2 Constraints and Limitations	44
	6.3 Word Cloud Visualization	44
	6.4 Boundary Conditions	46
7.	Final Exploration Chapters	47
	7.1 Cross Attention	47
	7.2 Elmo	48

	7.3 Using Resnet embeddings	49
	7.4 Addition of clustering	50
8.	Results	55
	8.1 Metrics used in the project	55
	8.1.1 Recommendation system workflow	55
	8.1.2 Mean Rank metric	56
	8.1.3 Recall@k metric	58
	8.2 Results from various models developed	60
	8.3 Inferences obtained from the results table	62
	8.4 Looking at some example queries	64
	8.4.1 A group of people eating pizza	64
	8.4.2 Surfing in the beach	66
	8.4.3 A bench in the park on a calm evening	68
9.	Conclusion	70
10.	Future Work	71
	10.1 Integration of clustering with the prediction algorithm	71
	10.2 Cross attention implementation	72
	10.3 Implementing multihead transformer stack	72
	10.4 Research on more advanced recent embedding models	73
	10.5 Shipping the model as a product - Strategy	74
11.	References	75
	11.1 Literature Survey Papers	75
	11.2 Blogs	75
	11.3 Code snippets	75
12.	Appendix	76
	12.1 Code Snippets	76

LIST OF TABLES

Table No.	Title	Page No.
1	A summary of the Space and time complexity taken when trying out different algorithms	8
2	Various metrics tabulated across different CNN stacks to measure the performance of clustering	15
3	A sample 5x5 test similarity score matrix	57
4	Metrics computed for all the successful models developed	60

LIST OF FIGURES

Figure No.	Title	Page No.
1	An example of content based image retrieval given a descriptive caption	2
2	A summary of the attention technique used for image captioning	9
3	A visual representation of the attention weights when the model focuses on a particular caption	9
4	A visual explanation of the cross modular component workflow	11
5	The model architecture used	12
6	Training time model of DLSIR	18
7	Inferencing time model architecture	20
8	Model summary for the resnet baseline model	24
9	A snapshot of the model training	25
10	model summary for the inception baseline model	26
11	Model graph of the bidirectional model	27
12	A snapshot of the bidirectional model training	28
13	Model summary of the image self-attention model	29
14	A snapshot of the image self-attention model training	30
15	: Model graph summary for the caption self-attention model	31
16	A snapshot of the caption self-attention model training	32
17	Model graph summary for the double self-attention model	33
18	A snapshot of the double self-attention mode training	34
19	A look at the hardware powering google colab notebooks	36

20	A look at how to configure the runtime settings in colab to use GPU/TPU as the hardware accelerator	36
21	A single datapoint from the dataset	37
22	Web application homepage	39
23	An example caption	40
24	A snapshot of the GUI developed	40
25	Flask application server starting	41
26	A brief overview of the inferencing time model	43
27	a visual word cloud representation of the test dataset	46
28	a summary of the cross attention process	47
29	Issues with the model tending towards 0 accuracy and 0 loss on the 3rd epoch itself	50
30	Cluster 1 - seems to be capturing the concepts of road, vehicles	51
31	Cluster 2 - seems to be capturing concept of things	52
32	Cluster 3 - seems to be capturing the concept of people	52
33	Cluster 4 - appears to be capturing the concept of food	53
34	Cluster 5 - appears to be capturing the concept of animals	53
35	Cluster 6 - appears to have captured the concept of sports and games	54
36	A typical recommender system workflow	55
37	caption 1- Base Model	64
38	caption 1 – Caption Self Attention model	64
39	caption 1 – Image Self Attention model	65
40	caption 1 – Caption Image Self Attention	65
41	caption 2 - Base Model	66
42	caption 2 – Caption Self Attention model	66
43	caption 2 – Image Self Attention model	67
44	caption 2 – Caption Image Self Attention	67
45	caption 3 - Base Model	68
46	caption 3 – Caption Self Attention model	68
47	caption 3 – Image Self Attention model	69
48	caption 3 – Caption Image Self Attention	69

ACRONYMS & ABBREVIATIONS

- CNN: This model is called as convolutional neural networks and are typically used to represent abstract features of the image
- RNN: This model is called as recurrent neural networks and are typically used in representing abstract features of sequences such as sentences
- Self-Attention: Also known as intra-attention, is an attention mechanism relating different positions of a single sequence in order to compute a representation of the same sequence
- GRU: This model is called as the Gated Recurrent Unit and is a specific type of RNN model
- LSTM: This model is called as the Long Short Term Memory and is a specific type of RNN model
- Bidirectional: This is a layer which can be wrapped over any type of RNN to make it a bidirectional RNN which can capture both forward and backward context in a sequence.
- Positional Embedding: This layer is used as a part of the Multi Head Self attention and is used in encoding the positions of a sequence in order for it to be processed by the Multi headed self-attention layer.
- Inception: This is a deep CNN model which was developed by google
- Resnet: This is a deep CNN model which was developed by microsoft
- Transformer networks: This model is presented in the “attention is all you need” paper where this model is presented as an encoder-decoder model but with stacked multi headed self attention in both the encoder and the decoder along with stacked multi headed cross attention between the encoder decoder model.
- elmo: This is deep neural word embedding model developed by OpenAI for NLP tasks
- bert: This is a deep Bidirectional Transformer word embedding model developed by google for solving standard NLP tasks
- fasttext: This is a deep neural word embedding model developed by facebook which also takes into account the n-gram nature of words.
- word2vec:This is a deep neural word embedding model which can be used to give out feature vectors describing a word token
- glove: This is a deep neural word embedding model which can be used to predict a n-dimensional feature vector for a given word.

- Cross-Attention: This technique is used commonly in Neural Machine translation problems, where the encoder and decoder learn to attend on only a few relevant captions from either side in order to make a prediction by learning hidden abstractions which are encapsulated as attention weights.
- wordcloud: This technique is commonly used in visualising the frequencies of words across a large dataset in a very intuitive manner
- keras: This is a high level machine learning framework based off python
- tensorflow: This is a lower level machine learning framework based off python
- GPU: This is called as the "Graphic Processing Unit" and this is typically used to speeden up the training of machine learning models due to the number of parallel cores GPU's have which give it an edge over training it against CPU's
- TPU: This hardware developed by google is also known as the "Tensorflow Processing Unit" which can be used to speeden up the training process and the model inferencing process by over 2 times when dealing with tensorflow based models.
- L2 normalisation: A normalisation technique which reduces the effect of overfitting on a model. Here the errors are firstly taken a difference of and then squared.
- Clustering: This is an unsupervised based learning algorithm which can group together several Data points based off the vector representations of them into meaningful relevant groups where all data points in a group represents a common concept
- GAN: This model is called as the "Generative Adversarial Network" which consists of two adversarial networks called as the "Generator" and "Discriminator". When trained together this model reaches an equilibrium point where the "Generator" can generate data which is very similar to the underlying dataset used which the "Discriminator" cannot distinguish that easily.
- SVM: These models are also known as "Support Vector Models" which are supervised learning models which can be used both for regressive and predictive analysis of data.

1 INTRODUCTION

1.1 Prologue

1.1.1 Existing Solutions to the Problem

The issue of picture recovery can be seen dependent on a great deal of justification for relating a picture with a specific setting or portrayal.

It depends on what we are taking a gander at in the picture and need not really be what the real pictorial portrayal delineates, as on account of sign sheets or emblematic pictures. Henceforth it's critical to settle on the basic connection first before jumping into the arrangement.

There are a huge number of methodologies and edges to taking a gander at this.

- Label based explanations for the picture.
- Setting based comment for the picture.
- Portrayal based picture connection

These conceivable outcomes offer ascent to numerous arrangements which may fit a specific setting.

1.1.2 Google's image retrieval system - Tag based

Google utilizes human explained picture labels and connects the equivalent with rank in their database and the pictures are positioned dependent on the importance of labels.

Since the labels are human and setting based, there is a lot of space for mistake to sneak in and degenerate the model outcomes.

There has been a ton of reports of such wrong outcomes prompting contentions also.

The label based framework additionally requires impressive exertion to approve and create.

To permit watchword look, the framework develops a file from picture labels allotted by the clients. The tag-based picture recovery approach, albeit truly versatile, has some genuine downsides because of the issues of tag spamming and subjectivity in labeling. In this paper, we propose a methodology for improving the tag-based picture recovery by misusing a few procedures in substance based picture recovery (CBIR).Users are permitted to perform inquiry by catchphrases with shading as well as tone choice. The shading file is likewise utilized for improving positioning of query items by means of the client pertinence criticism

1.1.3 Content Based Image Retrieval

Fig 1: An example of content based image retrieval given a descriptive caption

Content-based picture recovery otherwise called question by picture content (QBIC) and substance based visual data recovery (CBVIR) is the utilization of PC vision methods to the picture recovery issue, that is, the issue of hunting down advanced pictures in substantial databases. Content-based picture recovery is against conventional idea based methodologies.

"Content-based" implies that the pursuit breaks down the substance of the picture instead of the metadata, for example, watchwords, labels, or portrayals related with the picture. The expression "content" in this setting may allude to hues, shapes, surfaces, or whatever other data that can be gotten from the picture itself. CBIR is attractive on the grounds that looks through that depend absolutely on metadata are subject to explanation quality and fulfillment.

Having people physically explain pictures by entering watchwords or metadata in an extensive database can be tedious and may not catch the catchphrases wanted to portray the picture. The assessment of the viability of catchphrase picture seek is abstract and has not been well-characterized. In a similar respect, CBIR frameworks have comparable difficulties in characterizing success. "Keywords additionally limit the extent of questions to the arrangement of foreordained criteria." and, "having been set up" are less solid than utilizing the substance itself.

1.2 Overview

This report contains a reasonable portrayal of the different parts and their degree in the task titled "Profound Learning Based framework for Image Retrieval". The record is created so that the initial three areas including Objective and Scope obviously characterize the true objective and the issue space the venture is gone for settling.

We additionally attempt to build up clear limits with respect to the degree to which the proposed arrangement is legitimate. The later areas clarify in detail every one of the parts associated with the venture, the proposed design of the segments included and results, and writing study done to set up the baselines and set the goals.

The area on procedure covers all the plan choices produced using an engineering outlook on how each model was manufactured and prepared.

We have likewise featured a portion of the fizzled tests and gave a few bits of knowledge on why they fizzled.

The segment on results speaks inside and out about the two measurements we have actualized which are commonly utilized in recommender frameworks to be specific the "Recall@k" and the "Mean rank" metric which has been utilized to classify the execution of each model worked, alongside playing out a similar examination of what every measurement naturally implies while displaying a portion of the expectations made by these models on some test inquiries.

The archive plainly sets the establishment of the issue explanation and is gone for anybody attempting to comprehend the issue and perceive how what we propose is one of a kind contrasted with the counterparts in a similar area.

1.3 Scope

1.3.1 The basic Scope

The Image retrieval model being proposed is a proof of concept that the model can be used on any image data to retrieve them using their description.

The successful validation of the model will lead to the proof of the concept and hence the model can be used to train any arbitrary image data and can be utilized to retrieve the images.

Here, we evaluate the baseline model for Image retrieval and also build the model with an additional attention layer and analyze the performance improvements.

1.3.2 Coverage of the project

The project covers all the cases mentioned in the objective, the evaluative dataset we are using to train our model is the official Microsoft COCO dataset which has 16GB worth of image-caption pairs.

The model and its variations will be trained and tested only on the mentioned dataset.

For training purposes, the dataset used is capped to 2 lakh images along with their corresponding captions for those images. Whereas for testing we will be performing the model inferencing on only 6000 images from which images are retrieved.

1.3.3 The highlights of the solution

- Better Captioning using attention
  - The focus will be on incorporating attention into the baseline model to see if there will be any improvements.
- Usage of various word embedding models
  - The solution involves experimenting with various word embeddings and evaluating them for the caption dataset.
- Usage of various CNN models
  - The solution involves experimenting with multiple convolutional neural network models via transfer learning and evaluating which model gives out the best results in the long run.
- Building on top of a state of the art base model
  - The base models being used for this model are best in class and hence provide the best representation for both the image and the captions.
  - E.g. In most of our models, the images are represented with the feature vectors derived from Google's Inception CNN model whereas the captions are represented with the feature vectors derived from Facebook's Fasttext word embedding model.
  - Both of the above models have been used time and again to achieve a state of the performance in the field of computer vision and natural language processing respectively

1.3.4 Scope of the components involved in the model

- The text embedding generation component consists usage of pre-trained word embedding and experimenting with various dimensions and available options with the same.
- The image representation generation component involves usage of pre-trained model weights from a deep model and building on top of the same for enhanced efficiency. It involves experimenting on various available base models and their performance to pick the best representation.
- These two representations would be combined and the similarity score will between the two modes, this acts as the evaluation metric for the model and for fine-tuning the learning of the commonality as well.
- Attention mechanism will then be built on top of this set-up to pass a fine-grained representation for the image and text models so that they can learn better.
- A clustering approach will also be evaluated on the image dataset to constrain the search space and evaluate the time required for prediction.

1.3.5 Limitations

The model being built is trained and tested on the MS COCO dataset, although the dataset is fairly generic and contains samples of all kinds of images that are encountered in real life if one needs to use the idea for a specific problem which has its own images in its domain.

This limitation is due to the vast diversity of domains of images present in the real world data and the puniness of the machine learning models to imbibe all the possible representations due to space and time constraints.

Hence, the model is being presented as a proof of concept and this applied to any dataset would yield similar results.

1.4 Objective

The objective of our project can be summarized with the following points:

- Develop a novel objective function which can be used to train a model to understand correlations between multimodal representations of data. In this case, the correlation between the space of sentences/captions and the space of images

- Build an encoder-decoder based architecture and experiment with multiple model architectures and hyperparameters. The various experiments we have conducted as a part of this project are along these lines:
    - Experimenting with various word embeddings
    - Experimenting with various CNN architectures
- Incorporating the base model with various forms of attention in multiple different ways and evaluating the result of this new model.
- Perform a comparative analysis of all the models developed as a part of the various experiments conducted, and tabulate the results using custom metrics.

1.5 Outcomes

The outcomes of this project are fivefold:

1. Show proof of concept of deep learning based image retrieval as opposed to classical tag frequency based image retrieval systems.
2. Show the performance of the model using evaluation metrics typically used in recommender systems such as:

a. Recall @K (refer 8.1.3)

b. Mean Rank (refer 8.1.2)

3. Validate the effectiveness of the application of self-attention on multimodal machine learning models.
4. Model's adaptability to any new user-specific image dataset on fine tuning.
5. Build a simple, intuitive web application which demonstrates the results obtained from various models which were developed:
    - Base Model (refer 3.5.1)
    - Image Self Attention Model(refer 3.5.3.1)
    - Caption Self Attention Model(refer 3.5.3.2)
    - Double Self Attention Model(refer 3.5.3.3)

2 RESEARCH BACKGROUND

2.1 Literature Survey

2.1.1 Scaling up to large vocabulary image annotation. Jason Weston, Samy Bengio and Nicolas Usunier , 2013

What does the paper explain

They created a special ranking loss function (called WARP loss) to learn

a) a classifier for ranking tags given an image and

b) embeddings for both the images and the tags, in the same shared vector space.

What is the claim of the paper?

- State-of-the-art results for precision @1 and @10 against baselines in image classification tasks.
- Much faster and consumes much less memory than alternatives.
- Ensemble models using different types of image features perform even better.
- Directly optimizes Precision @k
- Apparently calculates some score for each (instance,label) pair.
- Supervised approach to learning embeddings.

Summary of the paper.

- Here a firmly performing technique that scales to expansive datasets by all the while figuring out how to streamline accuracy at the highest point of the positioned rundown of explanations for a given picture and learning a low-dimensional joint installing space for the two pictures and comments.
- Low dimensional embeddings are quick at testing time due to quick calculations for positioning explanations.
- Rank every comment for a picture and retrive the best, additionally train with negative comments.
- Become familiar with a joint circulation of both picture and content and guide them straightly.

- Twist misfortune is joined which changes over the position of an explanation to misfortune utilizing an Indicator, which is a hyper parameter that is set, which can be tuned to get precision@k or to gauge the most elevated positioned ones.
- SGD utilizing a triplet to ideally combine which incorporates uniform inspecting of the negative example.
- Key thought is the estimate of rank utilizing uniform examining trap, which affixes up the SGD

Results

Algorithm	Time Complexity	Test Time and Memory Usage for Web-data	
		Time	Space
k-NN	$\mathcal{O}(n \cdot d_{\bar{\phi}})$	3913 d	27 GB
Approx. k-NN	$\mathcal{O}((p + n/2^p) \cdot d_{\bar{\phi}})$	*variable*	27 GB
One-vs-Rest	$\mathcal{O}(Y \cdot d_{\bar{\phi}})$	19 d	8.2 GB
PAMIR^{IA}	$\mathcal{O}(Y \cdot d_{\bar{\phi}})$	19 d	8.2 GB
WSABIE	$\mathcal{O}((Y + d_{\bar{\phi}}) \cdot D)$	6.5 d	82 MB

Table1: A summary of the Space and time complexity taken when trying out different algorithms

Inference

The sampling trick results in much faster convergence and hence uses much less time and memory. Makes large scale training and annotation possible for massive online datasets.

What are the takeaways from this paper

The joint feature representation of image and text is the core of the idea of multi-modal machine learning domain, hence this paper proves that joint low dimensional vectorisation of the modes result in enhanced performance.

2.1.2 Show, Attend and Tell

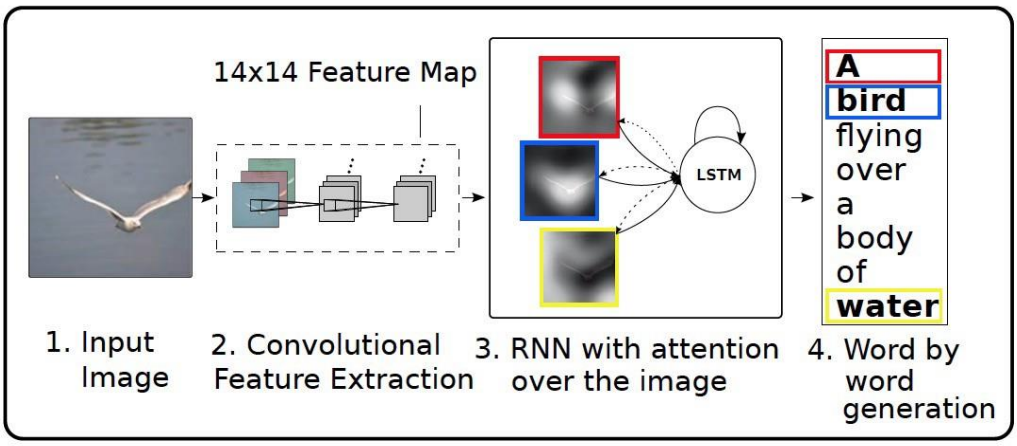

Fig 2: A summary of the attention technique used for image captioning.

The authors use an attention mechanism in image caption generation, allowing the decoder RNN focus on specific parts of the image. In order find the correspondence between words and image patches, the RNN uses a lower convolutional layer as its input (before pooling). The authors propose both a "hard" attention (trained using sampling methods) and "soft" attention (trained end-to-end) mechanism, and show qualitatively that the decoder focuses on sensible regions while generating text, adding an additional layer of interpretability to the model. The attention-based models achieve state-of-the art on Flickr8k, Flickr30 and MS Coco.

(a) A man and a woman playing frisbee in a field.

Fig 3: A visual representation of the attention weights when the model focuses on a particular caption

Key Points

- To find image correspondence use lower convolutional layers to attend to.
- Two attention mechanisms: Soft and hard. Depending on evaluation metric (BLEU vs. METERO) one or the other performs better.
- Largest data set (MS COCO) takes 3 days to train on Titan Black GPU. Oxford VGG.
- Soft attention is same as for seq2seq models.
- Attention weights are visualized by upsampling and applying a Gaussian

All these small components are like bedrock for most prevalent image captioning systems nowadays. There are many papers for image captioning with most of them working around these central ideas and between those papers, most of the changes are more or less in these areas:

1. How the word vectors are learned.(ex. Glove is another methodology to generate word vectors)
2. Back propagation to be done on image and word2vec layers or not.
3. LSTM (long short term memory) used in place of RNN which takes care of vanishing gradient problem in RNN.
4. Stochastic vs soft attention: Here we have discussed soft attention mechanism wherein different location vectors are aggregated on the basis of their individual probabilities. In the stochastic mechanism, a single location is sampled on the basis of probability distribution and vector from just that location is used in the RNN unit.

2.1.3 Look, Imagine and Match: Improving Textual-Visual Cross-Modal Retrieval with Generative Models - Jiuxiang Gu, Jianfei Cai, Shafiq Joty, Li Niu, Gang Wang

Literary visual cross-modular recovery has been a hot research point in both PC vision and regular language handling networks. Learning the appropriate portrayals for multi-modular information is the most significant thing for the cross-modular recovery execution. In the present picture content recovery approaches picture content sets are implanted into a solitary component vectors in a typical authentic space. In this paper the creators have proposed to consolidate generative procedures into the cross-modular element installing, through which

they had the capacity to learn the worldwide conceptual highlights as well as the neighborhood grounded highlights.

Despite the fact that the element portrayals in the educated regular portrayal space have been effectively used to depict abnormal state semantic ideas of multi-modular information, they are not adequate to recover pictures with nitty gritty neighborhood similitude (e.g., spatial format) or sentences with word-level closeness. Interestingly, as people, we can relate a printed (resp. picture) question to applicable pictures (resp. writings) all the more precisely, on the off chance that we give more consideration to the better subtleties of the pictures (resp. writings). As it were, on the off chance that we can ground the portrayal of one methodology to the items in the other methodology, we can become familiar with a superior mapping.

Roused by this idea, in this paper they have proposed to consolidate generative models into printed visual component installing for cross-modular recovery. Specifically, notwithstanding the regular cross-modular element installing at the worldwide semantic dimension, they likewise present an extra cross-modular element inserting at the neighborhood level, which is grounded by two generative models: picture to-content and content to-picture.

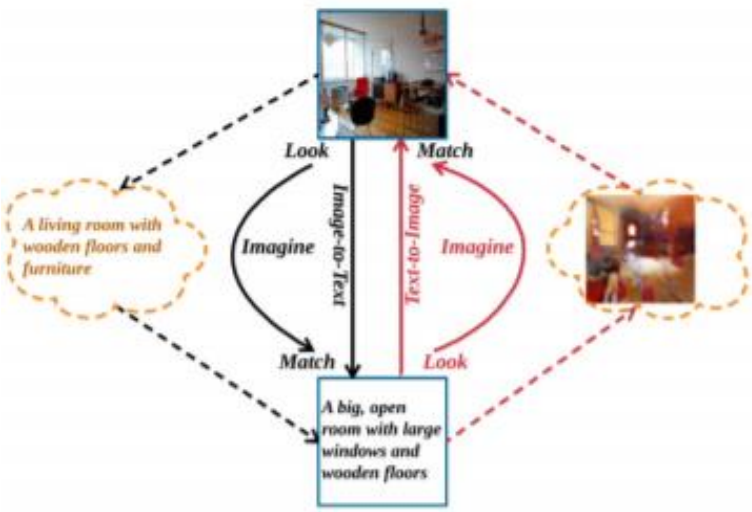

Fig 4: A visual explanation of the cross modular component workflow

The figure here represents the idea of their proposed cross-modular component inserting with generative models at abnormal state, which incorporates three learning steps: look, envision, and match.

Given a question in picture or content, we first take a gander at the inquiry to remove a conceptual portrayal. At that cpoint, we envision what the objective thing (content or picture) in the other methodology should resemble, and get a progressively concrete grounded portrayal. We achieve this by soliciting the portrayal from one methodology (to be evaluated) to create the thing in the other methodology, and contrasting the produced things and best quality levels. From that point onward, we coordinate the correct picture content sets utilizing the importance score which is determined dependent on a mix of grounded and unique portrayals

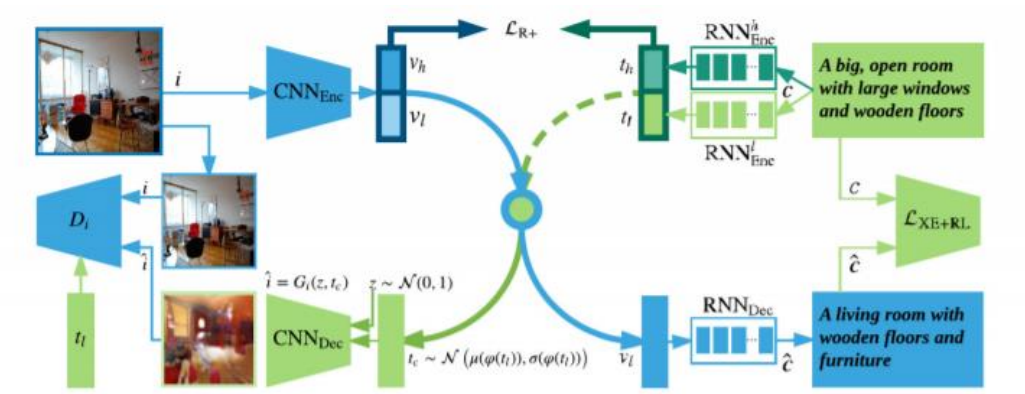

Fig 5: The model architecture used

The proposed generative cross-modular learning system (GXN). The whole structure comprises of three preparing ways: cross-modular element implanting (the whole upper part), picture to-content generative component learning (the blue way), and content to-picture generative antagonistic element learning (the green way). It incorporates six systems: two sentence encoders RNNh Enc (dull green) and RNNl Enc (light green), one picture encoder CNNEnc (blue), one sentence decoder RNNDec, one picture decoder CNNDec and one discriminator Di.

The primary way is like the current cross-modular component implanting that maps diverse methodology highlights into a typical space. In any case, the distinction here is that we utilize two parts of highlight installing, i.e., making the implanted visual element and the printed highlight nearer. The whole first preparing way for the most part incorporates one picture

encoder and two sentence encoders, one for the abnormal state dynamic highlights and one for the low dimension highlights!

The second preparing way (the blue way) is to create a sentence from the installed generative visual component. It comprises of the picture encoder and a sentence indicator. With a legitimate misfortune against ground-truth sentences, the grounded highlight will be balanced by means of back spread.

The third preparing way (the green way) is to produce a picture from the printed highlight tl. Here we receive the generative antagonistic model, which involves a generator/decoder and a discriminator.

Cross-modular recovery results on MSCOCO 1K picture test set (strong numbers are the best outcomes)

2.1.4 CNN features are also great at unsupervised classification - Joris Gu´erin, Olivier Gibaru, St´ephane Thiery, and Eric Nyiri

This paper aims at providing insight on the transferability of deep CNN features to unsupervised problems. They have studied the impact of different pretrained CNN feature extractors on the problem of image set clustering for object classification as well as fine-grained classification. They propose a rather straightforward pipeline combining deep-feature extraction using a CNN pretrained on ImageNet and a classic clustering algorithm to classify sets of images. They then compared this approach to state-of-the-art algorithms in image-clustering and provides better results. These results strengthen the belief that supervised training of deep CNN on large datasets, with a large variability of classes, extracts better features than most carefully designed engineering approaches, even for unsupervised tasks. They have also validated their approach on a robotic application, consisting in sorting and storing objects smartly based on clustering.

As of today, deep convolutional neural networks (CNN) are the method of choice for supervised image classification. Performances of CNN are highly correlated to the amount of labeled training data available. Nevertheless, even when few labels are available, several recent

papers have shown that CNN can still outperform any other approach by transferring knowledge learned on large datasets (Transfer Learning). In particular, it has also been shown that extracting features from OverFeat pretrained on ImageNet, and training a simple Support Vector Machine (SVM) classifier on these features to fit the new dataset provides better results than many more complex approaches for supervised classification. These results demonstrate that a CNN, trained on a large and versatile dataset, learns information about object characteristics that is generic enough to transfer to objects that are not in the original dataset.

The goal of this paper is to answer the following question:

How good are features extracted with a CNN pretrained on a large dataset, for unsupervised image classification tasks?

To do so they followed the approach consisting in applying classic clustering methods to features extracted from the final layers of a CNN and comparing it with state-of-the-art image set clustering algorithms on several public datasets. The intuition behind such approach for unsupervised object classification is that, as it works with SVM, the CNN must project data in a feature space where they are somehow linearly separable. Thus, simple clustering algorithms such as K-means might be working well. However, this study is interesting as the performance of such simple clustering algorithms often depends on the notion of distance between points, on which we remain uncertain.

What do you mean by Image clustering?

Given a set of unlabeled images, the image-set clustering problem consists in finding subsets of images based on their content: two images representing the same object should be clustered together and separated from images representing other objects.

The first successful methods focused on feature selection and used sophisticated algorithms to deal with complex features.

For instance,

- Representing images by Gaussian Mixture Models fitted to the pixels and clusters the set using the Information Bottleneck (IB) method.
- Using features resulting from image joint segmentation and sequential IB for clustering.

- Using Bags of Features with local representations (SIFT, SURF) to define commonality measures used for agglomerative clustering.

Recently, image-set clustering algorithms have shifted towards using deep features. For instance: using deep auto-encoders combined with ensemble clustering to generate feature representations suitable for clustering.

Experiments they conducted:

- Chose various CNN base stacks to obtain the image feature representations.
- Chose different layer outputs from the CNN stack for feature extraction
- Chose various clustering algorithms
- Computed the metric NMI score for performance comparison

Results they obtained

		KM	MBKM	AP	MS	AC	DBS [1]	Bi
Inception V3	mixed9	0.108 *374*	0.105 *7.3*	0.219 *12.7*	0.153 *16281*	0.110 *525*	0 *138*	0.110 *577*
	mixed10	0.468 *609*	0.401 *5.1*	0.442 *8.5*	0.039 *12126*	0.595 *525*	0 *119*	0.595 *567*
	avg˙pool	0.674 *6.3*	0.661 *0.2*	0.621 *7.7*	0.024 *230*	0.686 *8.5*	0 *1.8*	0.686 *8.9*
Resnet 50	avg˙pool	0.6748 *7.0*	0.641 *0.1*	0.587 *4.6*	0.043 *197*	0.640 *8.0*	0 *1.9*	0.640 *8.9*
VGG 16	block4˙pool	0.218 *278*	0.085 *3.6*	0.133 *6.0*	0.124 *10010*	0.277 *391*	0 *82.8*	0.277 *436*
	block5˙pool	0.488 *78*	0.048 *1.1*	0.262 *9.3*	0.194 *2325*	0.530 *99*	0 *21*	0.530 *107*
	fc1	0.606 *17*	0.458 *0.2*	0.421 *4.8*	0.187 *365*	0.617 *17*	0 *3.8*	0.617 *19*
	fc2	0.661 *16*	0.611 *0.2*	0.551 *4.3*	0.085 *373*	0.673 *15.9*	0 *3.8*	0.673 *19.7*
VGG 19	block4˙pool	0.203 *220*	0.139 *3.7*	0.124 *6.3*	0.135 *10298*	0.234 *388*	0 *83*	0.234 *435*
	block5˙pool	0.522 *74*	0.321 *0.9*	0.250 *9.3*	0.198 *2353*	0.540 *97*	0 *20*	0.540 *106*
	fc1	0.607 *17*	0.471 *0.2*	0.449 *9.3*	0.188 *365*	0.628 *17*	0 *3.9*	0.628 *18*
	fc2	0.672 *15*	0.615 *0.2*	0.557 *5.6*	0.083 *391*	0.671 *17*	0 *3.9*	0.671 *18*
Xception	block13˙pool	0.376 *410*	0.264 *4.8*	0.351 *10*	0.044 *9677*	0.473 *403*	0 *87*	0.473 *444*
	block14˙act	0.574 *935*	0.428 *10*	0.584 *10*	0.071 *24809*	0.634 *820*	0 *180*	0.634 *901*
	avg˙pool	0.692 *7.1*	0.636 *0.1*	0.636 *4.9*	0.052 *201*	**0.726** ***8.5***	0 *5.5*	0.726 *9.1*

Table 2: Various metrics tabulated across different CNN stacks to measure the performance of clustering

NMI scores (in black) and time in seconds (in blue, italics) on Pascal VOC2007-SL test set using different CNN, different output layers and different clustering algorithms. (Layers names are the same as in the Keras implementations).

Conclusion:

For unsupervised classification tasks, features extracted from deep CNN trained on large and diverse datasets, combined with classic clustering algorithms, can compete with more sophisticated and tuned image-set clustering methods. The fairly simple and naive pipeline proposed outperforms the best results reported in recent work, which raises the question of which research direction should be chosen to reach generic knowledge. Are efforts spent in developing image representation extractors more useful than simply building larger and more diverse datasets?

3 METHODOLOGY

3.1 Proposed Approach

We propose to manufacture an encoder-decoder based model which can become familiar with the concealed mapping between the two distinct modalities of pictures and content effectively.

Past answers for this issue additionally utilize a comparable encoder-decoder based design where the subtitle is encoded into some portrayal which the decoder at that point interprets and radiates some sort of expectation. There are additionally a few arrangements proposed which fuses some Natural Language Processing based "Grammatical form" models in relationship with Computer Vision based item recognition to make sense of a mapping between the articles in the sentence to the items found in the picture.

Our methodology basically rotates around the use of encoder-decoder based designs without the association of any scene comprehension or article identification models or NLP based methodologies.

In this task we have explored different avenues regarding a few variations of best in class segments which when stuck together makes up our DLSIR model. We have additionally given taken a stab at trying different things with a few kinds/variations of Recurrent systems , word installing models, convolutional neural system stacks for exchange learning and furthermore fused the idea of self-consideration into our models.

3.2 High Level System Architecture

As stated above our model primarily revolves around the encoder-decoder architectural design pattern.

The figure below shows a high level abstract model architecture where each component several implementations. This way we have carried out around 6 experiments by trying several implementations for a specific component.

Each experiment carried out, has been designed in such a way so as to overcome the limitations of previously carried out experiments by incrementally replacing some existing components with more state of the art components which serve the same functionality.

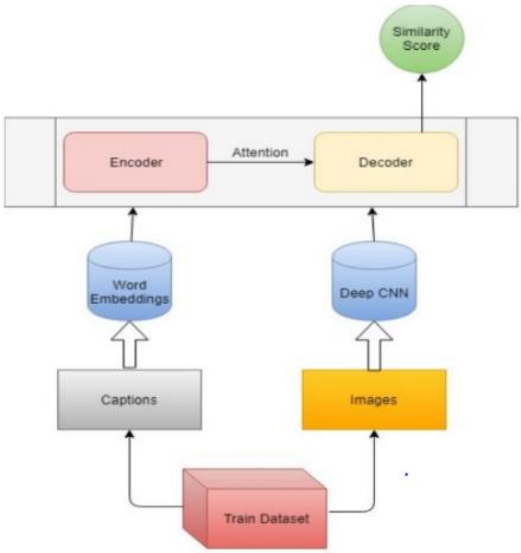

Fig 6: Training time model of DLSIR

This diagram encapsulates a high level model architecture used at the time of training the model.

The various components involved in this diagram are:

1. **Train Dataset:**

- The MS Coco dataset is used to train this model.
- It consists of 16 GB of images with each image associated with 5 captions each.(more about the dataset in section 4.3)

2. **Word Embeddings**:

- During training, each caption is split into tokens and each token is passed through a pre-trained neural word embedding model
- Some of the popular word embedding neural models include word2vec, fasttext, elmo, bert.
- In this project we have extensively used fasttext as our primary word embedding model after experimenting with elmo, word2vec and glove

3. Deep CNN Stack

- During training, each image is fed through a deep pre-trained convolutional stack for extracting the image's feature vector for feeding into the decoder.
- Some of the already existing pre-trained models available in the open source are Inception, VGG, Resnet, Alexnet etc.
- In this project we have extensively used Inception along with Resnet

4. **Encoder:**

- This is a Recurrent Neural Network which is fed the word embedding vectors of the caption during training time which in turn is used to generate a caption embedding.
- In this project we have tried experimenting with various types of RNN's such as GRU, LSTM
- We have also used the bidirectional layer as a wrapper to the already existing RNN in one of our models.
- In one of our models we have also tried using "Multihead self-attention" layer as the encoder instead of using RNN's

● **Decoder**:

- This neural network is primarily built on top of the Deep CNN model.

- Here we take the image feature representations from this CNN model and then pass it through a custom neural network to get embedding of the image, which is used along with the caption embedding in order to arrive at the similarity score.

Together the encoder and decoder when trained together learn a mapping between how images and captions are correlated.

For training this model, we have used a custom loss function called as “contrastive loss” (refer section 3.4).

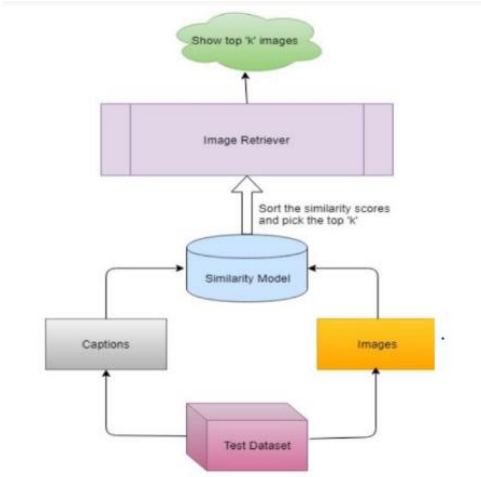

Fig 7: Inferencing time model architecture

As mentioned in the figure above the various components involved in the inferencing tie model are enumerated as follows:

- **Test Dataset:**
  - This is a subset of the MS coco dataset which will be used for validating/testing the model built at training time.
  - In our project we are using a test dataset of size 5000 to make predictions and retrieve images from.

- **Similarity Model:**
    - This is the model built at training time which after proper and sufficient training can give out a similarity score in the range of 0-1 which correspond to how well does an image suit the given caption.
    - In our case the closer this score is towards 0,the more similar the caption-image pair is.
- **Image Retriever:**
    - This component uses both the similarity model and the test dataset and runs the given caption at test time against all the images in the test dataset and sorts it based on the similarity score.
    - It then retrieves the top 'k' images with the highest similarity score and displays it to the user.
    - In our case we have decided to fix 'k' as 10 images.

3.3 Experiments conducted

As a part of this project we have tried out a multitude of experiments where each new model developed had some minor architectural differences which gave it a slight edge over its previous model but the overall arching encoder-decoder architecture still remains the same.

Listed below are all the experiments we conducted so far:

1. Baseline models
    - Resnet + GRU + fasttext
    - Inception + GRU + fastext
2. Baseline with bidirectional
    - Inception + Bidirectional(LSTM) + fastext
3. Self attention models

- SelfAttention(Inception) + Bidirectional(LSTM) + fastext
- Inception + SelfAttention(fastext)
- SelfAttention(inception) + SelfAttention(fastext)

In addition to this there were also several failed experiments which we could not implement completely due to lack of time.(refer section 7 for detailed explanations of these experiments)

1. Usage of openai's elmo as a word embedding

2. Implementation of a resnet bidirectional model

3. Implementing cross attention on top of the double self attention model

4. Usage of clustering at inferencing time to reduce the search time.

3.4 Training Procedure

The "Order-embeddings of images and language" paper proposes a new loss function for text to text retrieval and caption image retrieval based on the idea of hypernyms/higher abstractions of an entity thereby generalising well. (refer references section)

It is this loss function we have also implemented as a part of training our model.

Firstly before defining the loss function we have to define the similarity function between captions and images.

S(c,i) is defined as the similarity score between the caption "c" and the image "i",

$$S(c,i) = -\|f_i(i) - f_c(c)\|_2^2 \quad (1)$$

Here, $f_i(i)$ & $f_c(c)$ represents the embedding functions for images and captions respectively. There is a negative sign since we would like larger S to indicate more similarity.

Here this function is basically called as L2 normalization.

Our loss function can be defined as minimizing the following equation.

$$\sum_{(c,i)}\left(\sum_{c'} max\{0, \alpha - S(c,i) + S(c',i)\} + \sum_{i'} max\{0, \alpha - S(c,i) + S(c,i')\}\right) \quad (2)$$

This loss function is written in this manner for the purpose of contrasting correct and incorrect matches, we introduce negative examples that comes handy in the same training batch.

Where (c, i) is the true caption-image pair, c' and i' refer to incorrect captions and images for the selected pair. Therefore, the cost function enforces positive (i.e. correct) examples to have zero penalty and negative (i.e. incorrect) examples to have penalty greater than a margin α.

When training, the embedding functions $f_i(i)$ & $f_c(c)$ both return a 1024 dimensional vector, and to indicate to the model that a particular image is related to a particular caption we pass along a 1024 dimensional vector full of zeroes.

After training the model using this objective function the weights of the embedding model of both the caption (i.e. encoder) and that of the image (i.e. decoder) will be aligned in such a way that for a corresponding caption - image pair the similarity score $S(c,i) \approx 0$ and for any other non-matching caption-image pair the similarity is close to 1.

3.5 A deeper dive into models

3.5.1 Baseline models

To start off with the baseline models implemented here are the simplest encoder-decoder models one can start off with.

But instead of starting from scratch we decided that we wanted to use transfer learning in both the image representations and the word embeddings as training them from scratch would probably have needed an even bigger dataset with weeks of training.

Therefore, for these two baseline models we decided to choose the “fasttext” neural word embedding developed by facebook over other word embedding models such as word2vec, glove ,elmo etc because the way fasttext is trained depends a lot on n-gram models of the word. Which implies that even for out of vocabulary words the model provides an excellent embedding representation because of the n-gram nature of the fasttext model.

And on the image side we decided to go with two of the most famous deep CNN stacks which are inception and resnet as both models have achieved state of the art performance on the image net dataset.

3.5.1.1 Resnet + GRU + fastttext

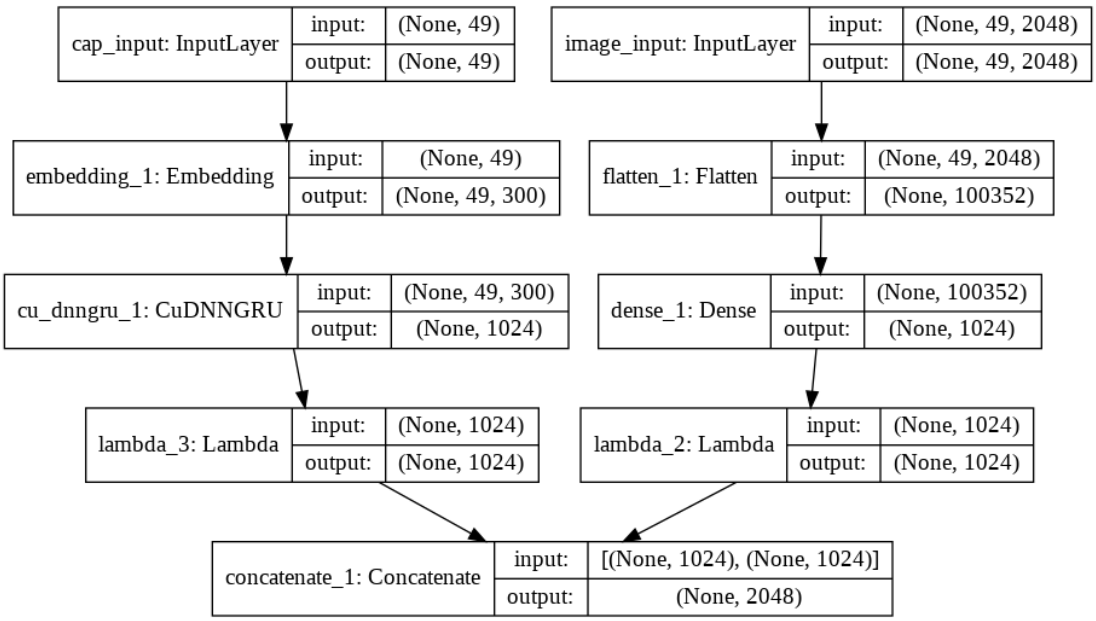

Fig 8: model summary for the renet baseline model

Number of trainable parameters used in this model are 109378344

```
1600/1600 [==============================] - 347s 217ms/step - loss: 24.0000
Epoch 2/20
1600/1600 [==============================] - 342s 214ms/step - loss: 21.3642
Epoch 3/20
1600/1600 [==============================] - 340s 212ms/step - loss: 19.4707
Epoch 4/20
1600/1600 [==============================] - 342s 213ms/step - loss: 18.5013
Epoch 5/20
1600/1600 [==============================] - 341s 213ms/step - loss: 17.6475
Epoch 6/20
1600/1600 [==============================] - 343s 214ms/step - loss: 16.1790
Epoch 7/20
1600/1600 [==============================] - 341s 213ms/step - loss: 16.8897
Epoch 8/20
1600/1600 [==============================] - 342s 213ms/step - loss: 18.6140
Epoch 9/20
1600/1600 [==============================] - 341s 213ms/step - loss: 14.9255
Epoch 10/20
1600/1600 [==============================] - 341s 213ms/step - loss: 14.0434
Epoch 11/20
1600/1600 [==============================] - 341s 213ms/step - loss: 13.6477
Epoch 12/20
1600/1600 [==============================] - 341s 213ms/step - loss: 13.5099
Epoch 13/20
1600/1600 [==============================] - 342s 214ms/step - loss: 14.5875
Epoch 14/20
1600/1600 [==============================] - 342s 214ms/step - loss: 13.2440
Epoch 15/20
1600/1600 [==============================] - 344s 215ms/step - loss: 13.1516
Epoch 16/20
1600/1600 [==============================] - 348s 218ms/step - loss: 13.0653
Epoch 17/20
1600/1600 [==============================] - 345s 216ms/step - loss: 12.9951
Epoch 18/20
1600/1600 [==============================] - 344s 215ms/step - loss: 12.9044
Epoch 19/20
1600/1600 [==============================] - 345s 216ms/step - loss: 12.8173
Epoch 20/20
1600/1600 [==============================] - 344s 215ms/step - loss: 12.4662
```

Fig 9: A snapshot of the model training

3.5.1.2 Inception + GRU + fasttext

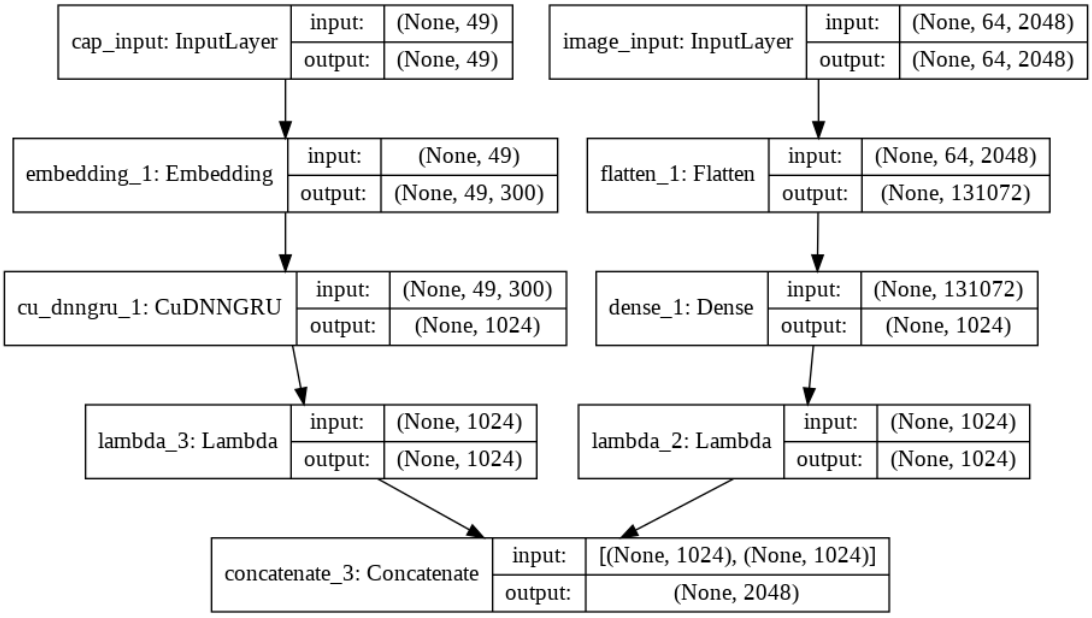

Fig 10: model summary for the inception baseline model

Number of trainable parameters used in this model are 140835624

3.5.2 Bidirectional model

Since the base model was performing decently well we decided to go ahead with improving the feature representation of the caption to begin with to see if there would be any improvements and we did observe a lot of improvement in terms of the loss decrease as well as the recall@k values computed.

Here we also chose to replace the GRU model with an LSTM model due to the fact that LSTM models have two hidden cell states within them as opposed to the one which GRU provides.

This is quite understandable as a bidirectional sequence model will run the inputs in two ways, one from past to future and one from future to past and what differs this approach from unidirectional is that in the LSTM that runs backwards we preserve information from the

future and using the two hidden states combined we can in any point in time preserve information from **both past and future** and hence it understands the context better.

3.5.2.1 Inception + Bidirectional LSTM + fasttext

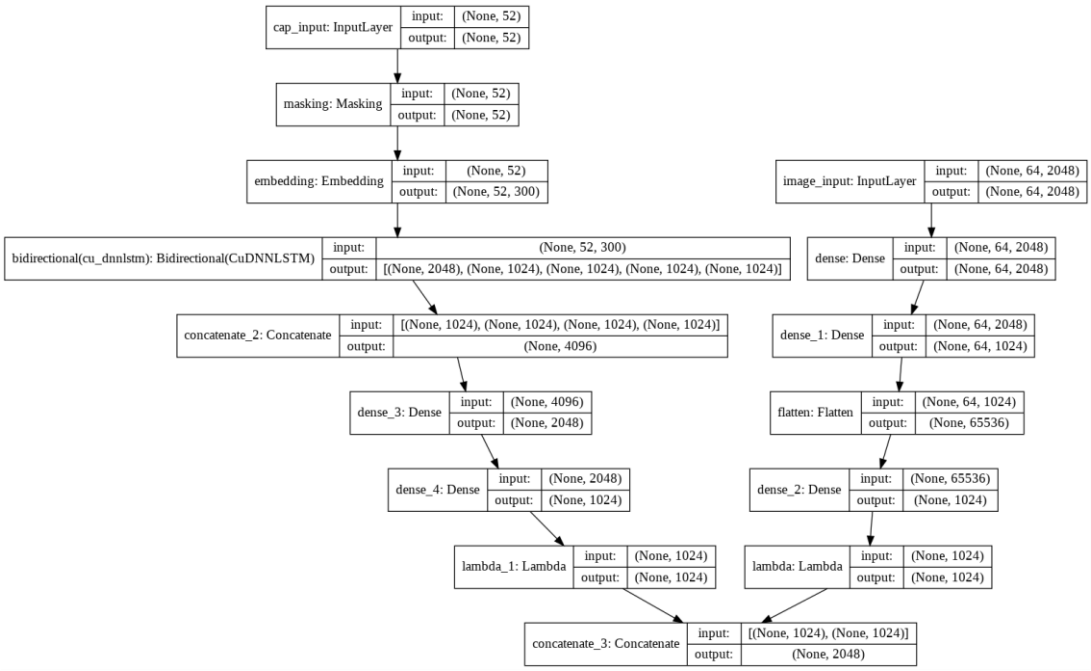

Fig 11: Model graph of the bidirectional model

Number of trainable parameters used in this model are 101958240

```
6250/6250 [==============================] - 1638s 262ms/step - loss: 68.7780
Epoch 2/10
6250/6250 [==============================] - 1618s 259ms/step - loss: 20.5580
Epoch 3/10
6250/6250 [==============================] - 1615s 258ms/step - loss: 13.0719
Epoch 4/10
6250/6250 [==============================] - 1621s 259ms/step - loss: 10.5892
Epoch 5/10
6250/6250 [==============================] - 1615s 258ms/step - loss: 9.2930
Epoch 6/10
6250/6250 [==============================] - 1623s 260ms/step - loss: 8.2322
Epoch 7/10
6250/6250 [==============================] - 1612s 258ms/step - loss: 7.4321
Epoch 8/10
6250/6250 [==============================] - 1609s 257ms/step - loss: 6.9332
Epoch 9/10
6250/6250 [==============================] - 1621s 259ms/step - loss: 6.4731
Epoch 10/10
6250/6250 [==============================] - 1615s 258ms/step - loss: 6.0723
```

Fig 12: A snapshot of the bidirectional model training

3.5.3 Self attention models

Now that the results improved by improving the caption representation in the latent space, we focused on improving the image feature representation as well. We decided to add a layer of self-attention on the image side.

What good is self-attention going to do?

> While convolutional filters are good at exploring spatial locality information, the receptive fields may not be large enough to cover larger structures. We can increase the filter size or the depth of the deep network but this will make the training complex.

Alternatively, we can apply the attention concept. Now we pay attention to only particular feature maps. Now we have a larger receptive field and the context is more focused and more relevant.

It has already been shown in the paper "Attention is all you need" that non-recurrent architectures (self-attentional) have outperformed RNNs in neural machine translation. Self-attentional networks can connect distant words via shorter network paths than

RNNs, and it has been speculated that this improves their ability to model long-range dependencies.

As a result, to the model configuration that had given the best results so far, Self-attention was employed:

- just to the image representation
- just to caption representation
- to both caption and image representations

3.5.3.1 Self Attention (Inception) + Bidirectional LSTM + fasttext (Self attention only for Image)

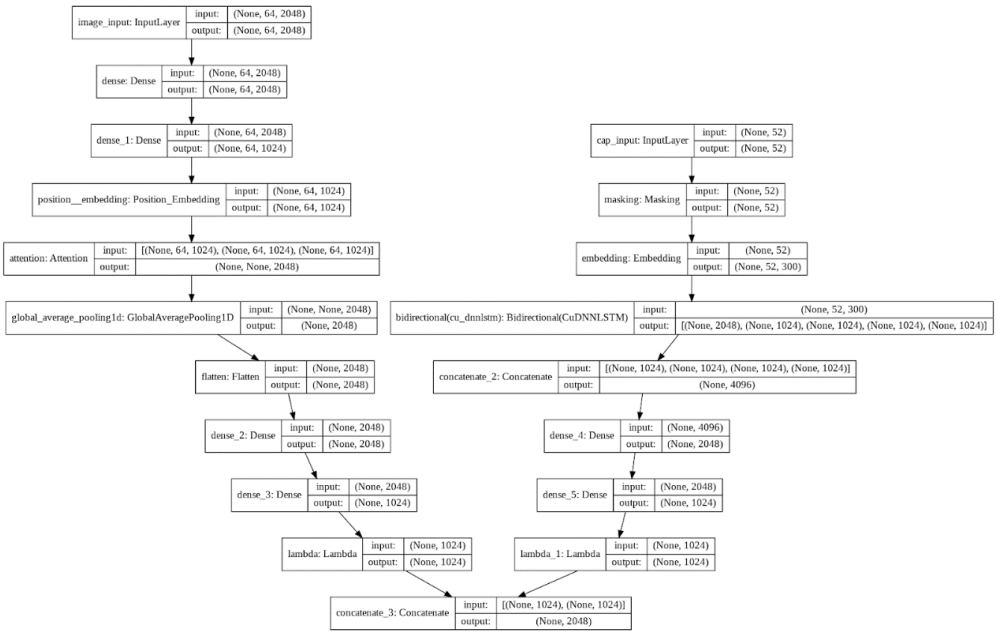

Fig 13: Model summary of the image self-attention model

Number of trainable parameters used in this model are 47434336

```
Use tf.cast instead.
2500/2500 [==============================] - 575s 230ms/step - loss: 17.8214
Epoch 2/15
2500/2500 [==============================] - 562s 225ms/step - loss: 11.1742
Epoch 3/15
2500/2500 [==============================] - 561s 225ms/step - loss: 7.9108
Epoch 4/15
2500/2500 [==============================] - 560s 224ms/step - loss: 6.3639
Epoch 5/15
2500/2500 [==============================] - 562s 225ms/step - loss: 5.0303
Epoch 6/15
2500/2500 [==============================] - 567s 227ms/step - loss: 4.2283
Epoch 7/15
2500/2500 [==============================] - 571s 228ms/step - loss: 3.7750
Epoch 8/15
2500/2500 [==============================] - 567s 227ms/step - loss: 3.3626
Epoch 9/15
2500/2500 [==============================] - 568s 227ms/step - loss: 2.8742
Epoch 10/15
2500/2500 [==============================] - 575s 230ms/step - loss: 2.5064
Epoch 11/15
2500/2500 [==============================] - 569s 228ms/step - loss: 2.2297
Epoch 12/15
2500/2500 [==============================] - 576s 230ms/step - loss: 2.0048
Epoch 13/15
2500/2500 [==============================] - 578s 231ms/step - loss: 1.8039
Epoch 14/15
2500/2500 [==============================] - 577s 231ms/step - loss: 1.7139
Epoch 15/15
2500/2500 [==============================] - 576s 230ms/step - loss: 1.5861
```

Fig 14: A snapshot of the image self-attention model training

3.5.3.2 Inception + Self Attention (fasttext) (Self Attention only on Caption side)

Fig 15: Model graph summary for the caption self-attention model

Number of trainable parameters used in this model are 95907424

```
6250/6250 [==============================] - 2175s 348ms/step - loss: 25.3237
Epoch 2/15
6250/6250 [==============================] - 2170s 347ms/step - loss: 11.7750
Epoch 3/15
6250/6250 [==============================] - 2169s 347ms/step - loss: 8.8722
Epoch 4/15
6250/6250 [==============================] - 2175s 348ms/step - loss: 6.4907
Epoch 5/15
6250/6250 [==============================] - 2180s 349ms/step - loss: 5.7408
Epoch 6/15
6250/6250 [==============================] - 2202s 352ms/step - loss: 5.5868
Epoch 7/15
6250/6250 [==============================] - 2183s 349ms/step - loss: 5.0216
Epoch 8/15
6250/6250 [==============================] - 2186s 350ms/step - loss: 3.8681
Epoch 9/15
6250/6250 [==============================] - 2192s 351ms/step - loss: 3.4563
Epoch 10/15
6250/6250 [==============================] - 2196s 351ms/step - loss: 2.9339
Epoch 11/15
6250/6250 [==============================] - 2188s 350ms/step - loss: 2.6284
Epoch 12/15
6250/6250 [==============================] - 2191s 351ms/step - loss: 2.7209
Epoch 13/15
6250/6250 [==============================] - 2195s 351ms/step - loss: 2.3625
Epoch 14/15
6250/6250 [==============================] - 2193s 351ms/step - loss: 2.0914
Epoch 15/15
6250/6250 [==============================] - 2190s 350ms/step - loss: 1.9274
```

Fig 16: A snapshot of the caption self-attention model training

3.5.3.3 Self Attention (Inception) + Self Attention (fasttext) (Self attention to both caption as well as image)

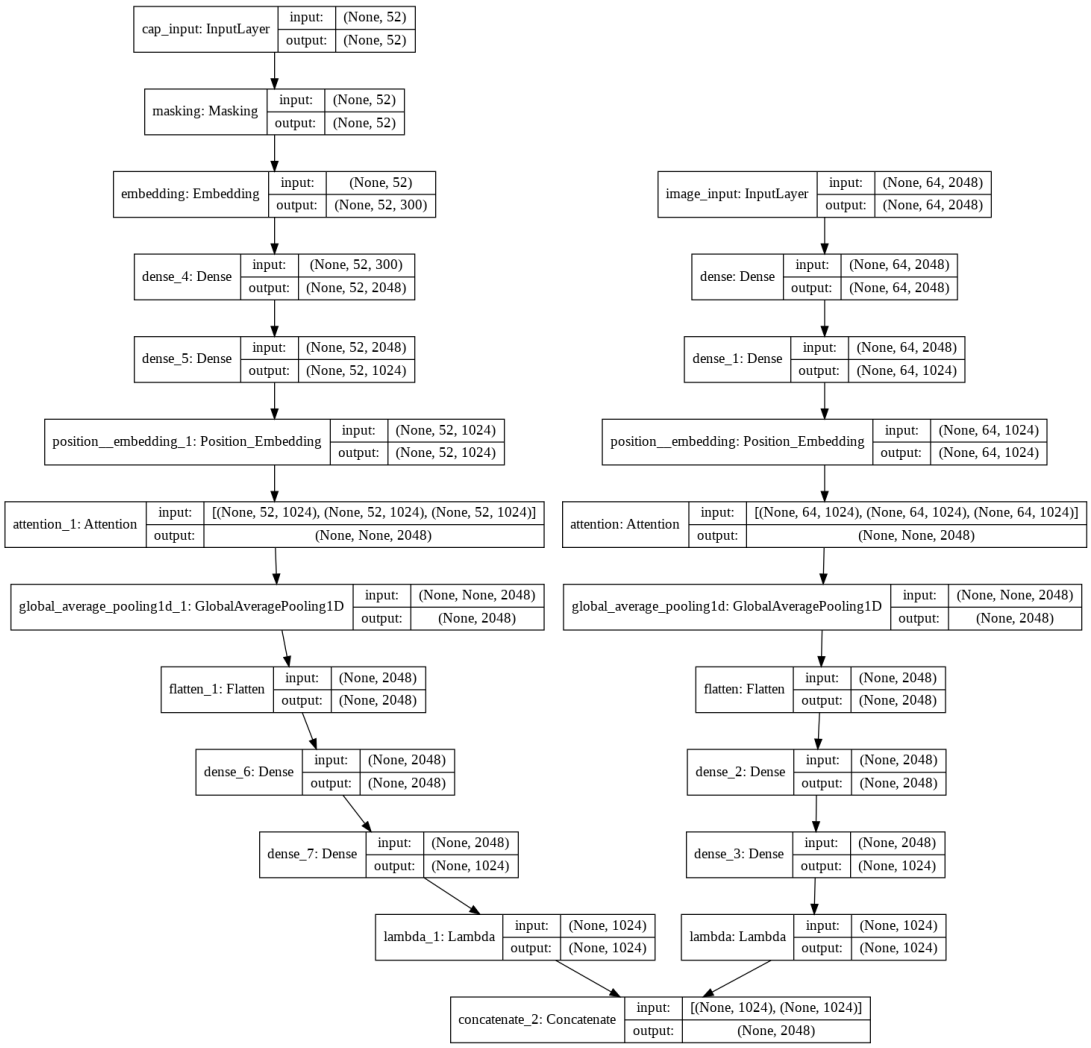

Fig 17: Model graph summary for the double self-attention model

Number of trainable parameters used in this model are 41383520

```
6250/6250 [==============================] - 2233s 357ms/step - loss: 45.5562
Epoch 2/15
6250/6250 [==============================] - 2230s 357ms/step - loss: 21.0302
Epoch 3/15
6250/6250 [==============================] - 2240s 358ms/step - loss: 16.9048
Epoch 4/15
6250/6250 [==============================] - 2228s 356ms/step - loss: 14.3237
Epoch 5/15
6250/6250 [==============================] - 2231s 357ms/step - loss: 12.2029
Epoch 6/15
6250/6250 [==============================] - 2244s 359ms/step - loss: 10.9709
Epoch 7/15
6250/6250 [==============================] - 2246s 359ms/step - loss: 10.3345
Epoch 8/15
6250/6250 [==============================] - 2247s 359ms/step - loss: 9.9459
Epoch 9/15
6250/6250 [==============================] - 2249s 360ms/step - loss: 9.2689
Epoch 10/15
6250/6250 [==============================] - 2242s 359ms/step - loss: 9.2912
Epoch 11/15
6250/6250 [==============================] - 2246s 359ms/step - loss: 9.4230
Epoch 12/15
6250/6250 [==============================] - 2243s 359ms/step - loss: 8.9960
Epoch 13/15
6250/6250 [==============================] - 2246s 359ms/step - loss: 8.7346
Epoch 14/15
6250/6250 [==============================] - 2251s 360ms/step - loss: 8.3924
Epoch 15/15
6250/6250 [==============================] - 2245s 359ms/step - loss: 8.1398
```

Fig 18: A snapshot of the double self-attention mode training

NOTE:

Each of the aforementioned model was trained with 2 Lakh images from the COCO dataset for 15 epochs in batch sizes of 32.

For the models with self-attention we implemented a multihead attention with 8 heads each head having 256 units.

4 ENVIRONMENT

The following configurations with respect to the hardware, software and data were necessary to complete the project:

4.1 Hardware Requirements:

Typically machine learning problems involve a lot of tensor operations which take up enormous amounts of time and space to compute while performing both forward propagation of tensors and the backward propagation of errors.

Because of this inherent nature of training complex machine learning models, training the model even on a CPU with multiple parallel hyperthreaded cores also proves to be time-consuming.

Since training our models on a large dataset takes longer time we have chosen to use “Google Colab” as our primary environment due to the following reasons:

- It offers a “Tesla K80 GPU” for free with 12GB RAM and 360GB of disk space.
- It offers a free TPU as well for very fast model inferencing
- Ability to store model weights in our google drive filesystem after training of the model
- Ability to download large datasets at tremendous speeds because of the cloud based architecture it provides.
- Allows upto 12 hours of continuous model training on the cloud server, with options to reconnect to the server in case of network failure
- The notebook already comes preloaded with all necessary deep learning frameworks
- Ability to install any third party modules via pip
- Ability to share a notebook with multiple authors via email.

After training the multiple models and saving the weights of each model in our drives, we have used these model weights in creating a simple web GUI which can be run on any computer with a browser installed.

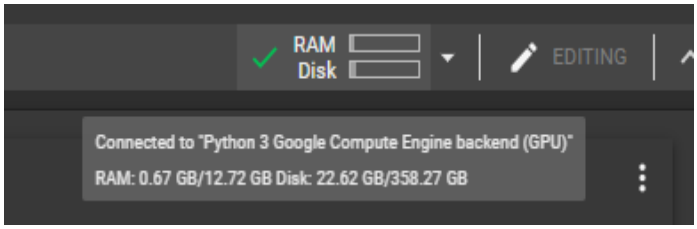

Fig 19: A look at the hardware powering google colab notebooks

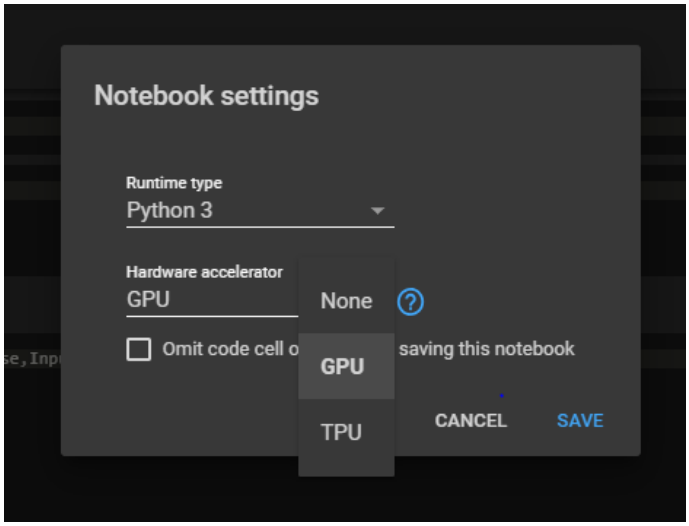

Fig 20: A look at how to configure the runtime settings in colab to use GPU/TPU as the hardware accelerator

4.2 Software Requirements

Our solution was primarily built on python by using the following modules:

1. Machine Learning Frameworks
    - tensorflow.keras : used for building, compiling and executing the model graph
    - tensorflow 1.13.1 with eager execution enabled : used various efficient data generator functions for efficient training of the model by continuously yielding batches of data
2. Data Processing and cleaning modules
    - numpy : used for representing the dataset features in an matrix like form for model training
    - pandas : used for performing a preliminary visualization of the dataset features

<u>3. Web App modules</u>

a. Backend:
    - flask : used in creating a server which exposes endpoints which give out the model predictions

b. Frontend:
    - HTML5: used in defining the structure of our simple web application
    - CSS3 : used in provide styling to the structure of our web application
    - JavaScript : used in providing very simple frontend functionality to our web application

4.3 Data Requirements

The dataset used in this project is the MS-COCO dataset which is a large-scale object detection, segmentation, and captioning dataset.

- Three teddy bears laying in bed under the covers.
- A group of stuffed animals sitting next to each other in bed.
- A white beige and brown baby bear under a beige white comforter.
- A trio of teddy bears bundled up on a bed.
- Three stuffed animals lay in a bed cuddled together.

Fig 21: A single datapoint from the dataset

The reason why picked COCO over other datasets are because of the following features:

1. Object segmentation
2. Recognition in context
3. Superpixel stuff segmentation
4. 330K images (>200K labeled)
5. 1.5 million object instances
6. 80 object categories
7. 91 stuff categories
8. 5 captions per image
9. 250,000 people with keypoints

For our project we have only used around 2 lakh image-caption pairs for training from a dataset of size 13GB.

This capping of the dataset was done primarily in order to perform multiple experiments as training a single model with large data also correspondingly needs a larger number of epochs to train over as well.

5 DEMONSTRATION OF OUTCOMES

We have built a Web application for image search. From the complete MS COCO dataset containing around 4 lakh images and their corresponding captions, we set aside a test data containing around 80k images and we randomly sample 6k images and their corresponding captions for the demonstration purposes.

Just to give an idea of the kind of images we are searching among we created a word-cloud representation of the captions corresponding to the demo test data. This word-cloud will be displayed to the user just to provide assistance while crafting the search query.

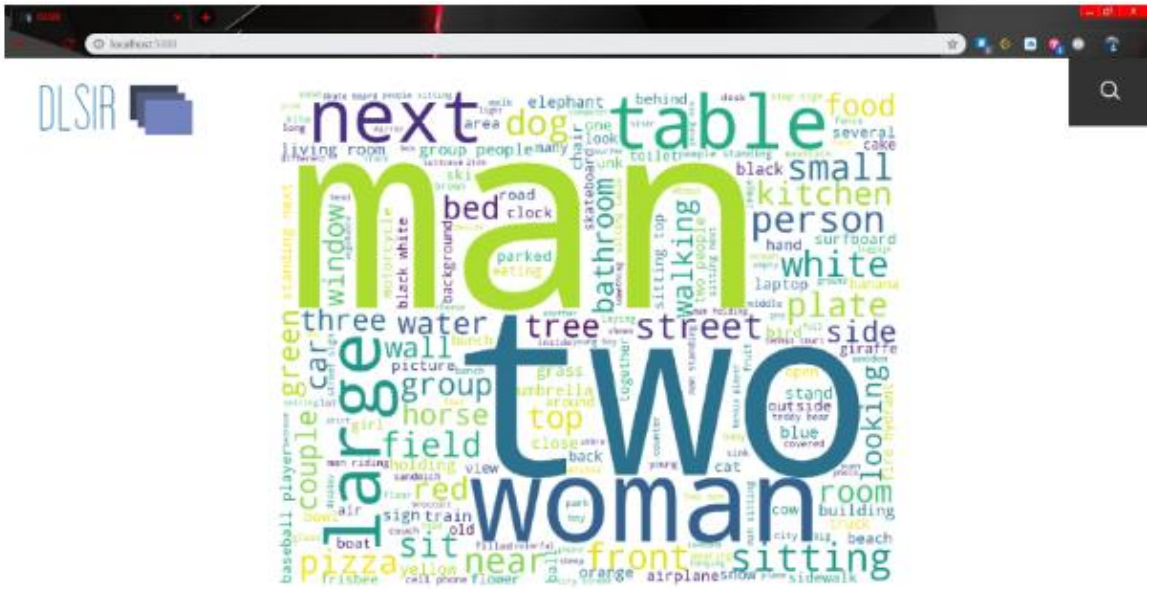

Fig 22: Web application homepage

On clicking the search icon on the top right, the user will be prompted to enter a query caption

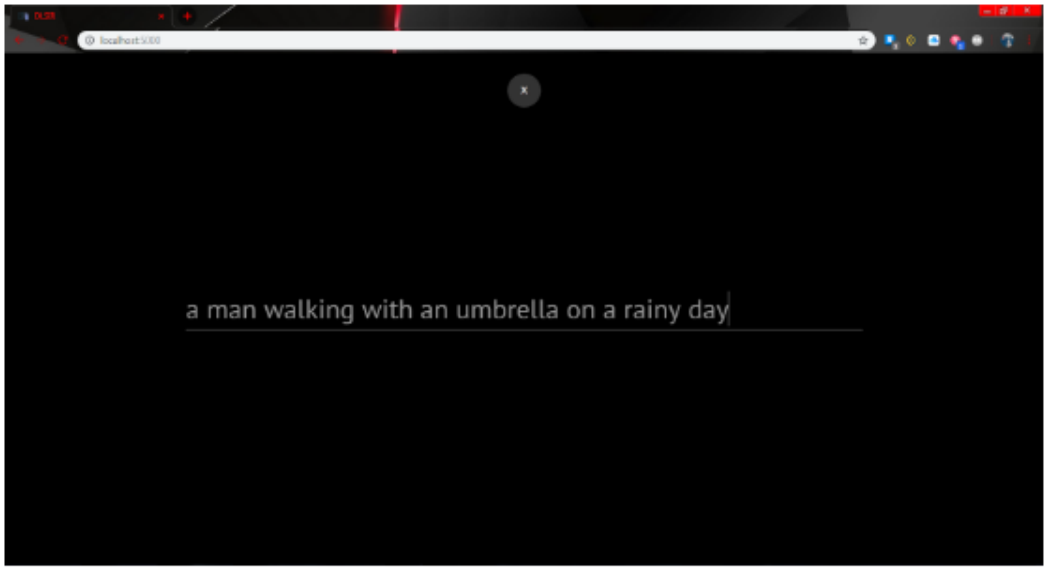

Fig 23: An example caption

On hitting top 9 images retrieved by the base model will be displayed with the scores assigned to the image being visible on hovering.

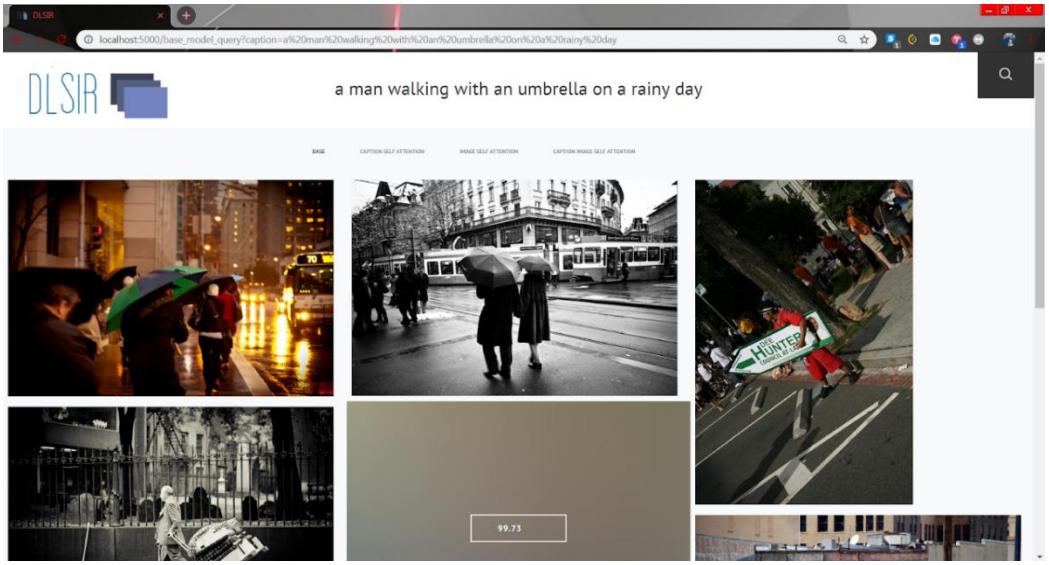

Fig 24: A snapshot from the GUI developed

We have also provided options to select from the various different models we experimented with and check out what kinds of images do those retrieve and what's the kind of score it provides to the retrieved images.

In the backend we have loaded up all the models ready for querying

```
(base) C:\Users\sikram\Documents\DLSIR\workspace\DLSIR_Local>python predictions.
py
C:\ProgramData\Anaconda3\lib\site-packages\smart_open\ssh.py:34: UserWarning: pa
ramiko missing, opening SSH/SCP/SFTP paths will be disabled.  `pip install param
iko` to suppress
  warnings.warn('paramiko missing, opening SSH/SCP/SFTP paths will be disabled.
 `pip install paramiko` to suppress')
--------------------------- Loading Test data ---------------------------
---------------------- Preparing Embedding Matrix ----------------------
Size of embedding matrix =  (24008, 300)
------------------------- Loading Base model -------------------------
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\py
thon\ops\resource_variable_ops.py:435: colocate_with (from tensorflow.python.fra
mework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
2019-04-26 00:02:11.466441: I tensorflow/core/platform/cpu_feature_guard.cc:141]
 Your CPU supports instructions that this TensorFlow binary was not compiled to
use: AVX2
2019-04-26 00:02:11.576442: W tensorflow/core/framework/allocator.cc:124] Alloca
tion of 268435456 exceeds 10% of system memory.
2019-04-26 00:02:16.513585: W tensorflow/core/framework/allocator.cc:124] Alloca
tion of 268435456 exceeds 10% of system memory.
2019-04-26 00:02:18.231672: W tensorflow/core/framework/allocator.cc:124] Alloca
tion of 268435456 exceeds 10% of system memory.
Loaded model from disk
---------------- Loading Caption Self Attention model ----------------
2019-04-26 00:02:19.081685: W tensorflow/core/framework/allocator.cc:124] Alloca
tion of 268435456 exceeds 10% of system memory.
2019-04-26 00:02:23.245812: W tensorflow/core/framework/allocator.cc:124] Alloca
tion of 268435456 exceeds 10% of system memory.
Loaded model from disk
----------------- Loading Image Self Attention model -----------------
Loaded model from disk
-------------- Loading Caption_Image Self Attention Model --------------
Loaded model from disk
------------------------- Base Model Prediction -------------------------
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorflow\py
thon\keras\utils\losses_utils.py:170: to_float (from tensorflow.python.ops.math_
ops) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
```

Fig 25: Flask application server starting

6 PROPOSED APPROACH

6.1 Research Approach

We experimented on the different approaches viable for each major component of the model and studied the effect on the overall performance of the model.

Image Representation

The image representation was obtained through two different baseline models and the results were noted.

● ResNet50 - The model with the base as resnet50, performed poorly and did not scale well with the data we have. The loss values saturated after some point of training. The loss values actually saturated towards zero and started giving Nan results. This behavior of the model was quite unexpected and unpredictable, therefore we abandoned this model

● Inception - The inception as baseline model performed very well and was able to scale well with the data and the loss values converged very well.

Summary - The inception model suits better for the image domain we have than resnet, hence we choose to use Inception as baseline in the model.

Text Representation

We evaluated with different embedding models and with different dimensioned vectors.

● Fasttext - This model proved the best among the other approaches as this also involves using character level n-gram embeddings

● word2vec and Glove - The results were not as good as the ones obtained by fasttext embeddings

● summary - We choose to use fasttext over the other two embedding models.

We also explored the necessity of applying a “Masking” layer for the Recurrent Neural Network which showed promising results

The encoder-decoder model

- vanilla GRU
    - This model took a long time to converge but we eventually reached a loss value of 9 after training for over 20 epochs

- vanilla LSTM
    - The performance of this model is almost the same as GRU except for the fact that LSTM also gives out 2 different hidden states which supposedly hold more context information of the sentence when compared to GRU's
    - bidirectional LSTM
    - This model converged extremely quickly and we reached loss values of 2 within the first 15 epochs suggesting that taking context from both the forward and backward direction proves to be better for the model as a whole.
    - Another reason why this model outshines all other forms of recurrent neural networks experimented on thus far is due to the fact that LSTM contributes 2 hidden cell states and by encapsulating the LSTM with a Bidirectional layer contributes 4 hidden cell states in both the forward and backward direction thereby giving more context to the model for learning the weights.

Testing Approach

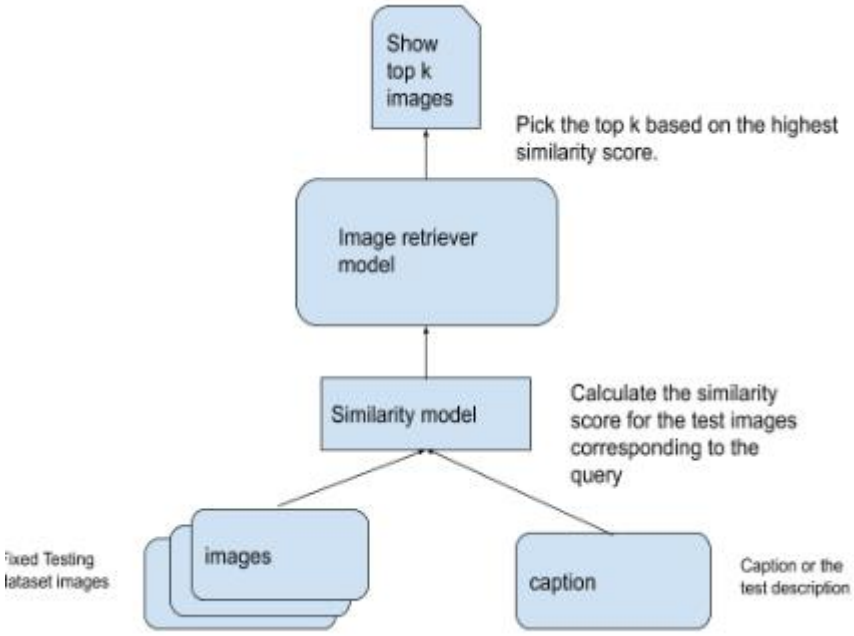

Fig 26: A brief overview of the inferencing time model

The specifications in testing are:

- The image dataset for testing needs all the images to be on the memory, hence we are limiting the test images to 6000.

- The captions therefore are to be provided such that, the image lies in the dataset of 6000 that has been picked.
- In order to facilitate this, we have made a word cloud of the caption for all the 6000 images picked in the test dataset.
- The query can be constructed based on the prominent words in the word cloud.
- Testing for various models have been benchmarked based on the below facts:
- Use the same test query for all the four models.
- Compare the results for the same.
- Compare recall@k, mean rank and loss values for each one of the models
- Tabulate the results for each model
- Make sure the same 6000 images are picked for each model by using a common seed for the random function.

6.2 Constraints and Limitations

- Since we will be employing a fairly complex architecture the resolution of the image will be limited. We will be experimenting with various state of the art CNN base stacks and each of them will require the input image to be within a fixed size.
- Since we will be using GRU or LSTMs to understand the caption while associating it with the respective image, the caption length can't be too long. We know that GRU,LSTMs suffer from this inherent problem of vanishing Gradient which is the basis for this constraint. We look forward to eliminating this constraint by encompassing the concept of attention.
- While retrieving the top images based on the users query, the more the pool of images to search from, the more the time required. We wish to reduce this retrieval time. This problem would be tackled using "clustering". Since the images will be encoded in such a way that the concepts captured in the encoded form will well relate to the caption, we can perform clustering on this encoded form and final search for relevant images from a specific cluster. This would speeden up the retrieval process.

6.3 Word Cloud Visualization

A perception strategy that shows how much of the time words show up in a given group of content, by making the extent of each word relative to its recurrence. Every one of the words

are then orchestrated in a bunch or haze of words. On the other hand, the words can likewise be masterminded in any arrangement: level lines, segments or inside a shape.

Word Clouds can likewise be utilized to show words that have meta-information allocated to them. For instance, in a Word Cloud with all the World's nations' names, the populace could be allotted to each name to decide its size.

Shading utilized on Word Clouds is normally negligible and is basically tasteful, however it tends to be utilized to classify words or to show another information variable.

Normally, Word Clouds are utilized on sites or web journals to delineate watchword or label use. Word Clouds can likewise be utilized to look at two changed collections of content together.

In spite of the fact that being straightforward and straightforward, Word Clouds have some real imperfections:

1. Long words are emphasised over short words.
2. Words whose letters contain many ascenders and descenders may receive more attention.
3. They're not great for analytical accuracy, so used more for aesthetic reasons instead.

   Using an open source python module called "wordcloud" we were able to create a word cloud from the fixed test dataset of size 6000 across all models.

   Given below is the image of all the words present in 6000 captions corresponding to the 6000 images we fixed as a test dataset.

man
two
woman
next
table
large
food
dog
elephant
behind
area
living room
group people
several
cake
small
kitchen
person
bed
road
clock
window
bathroom
walking
white
plate
three
water
tree
street
side
wall
group
horse
top
field
red
sit
near
front
sitting
pizza
room
building
truck
beach
cow
boat
sign
train
old
bowl
air
couch
frisbee
flower
orange
airplane
snow
sidewalk
green
car
couple
baseball player
standing next
motorcycle
black white
background
parked
eating
skateboard
toilet
black
hand
surfboard
laptop
banana
bench
picture
grass
umbrella
around
close
back
holding
view
sandwich
man riding
girl
shirt
smiling
lot
street sign
open
stand
outside
teddy bear
blue
covered
cat
sink
young
counter
zebra
animal
fire hydrant
giraffe
wooden
tennis court
bird
full
middle
guy
man holding
man standing
tennis player
fruit
baby
oven
photo
city
big
park
boy
two men
man sitting
someone
wearing
hanging
phone
sheep
city street
plane
cell phone
yellow
filled
colorful
broccoli
floor
display
glass
bedroom
vase
skate board
people sitting
walk
light
computer
skier
young man
desk
stop sign
fence
face
mountain
people standing
suitcase item
track
different
box
mirror
kite
long
pink
head
vegetable
brown
ski
road
clock
beside
donut
image
ocean
empty
luggage
ground
another
laying
shown
inside
young boy
together
cheese
something
sitting table
sitting top
two people
sitting next
unk
look
one
chair
many
surfer
computer
skier
young man

7 FINAL EXPLORATION CHAPTERS

In our experiments of trying out various configurations and architectural designs there were some models which were left incomplete due to time constraints and there were also experiments which were tried out which did not yield results as expected.

This section will touch upon some of the exploratory experiments carried out which could still be tried out as a part of future work.

7.1 Cross Attention

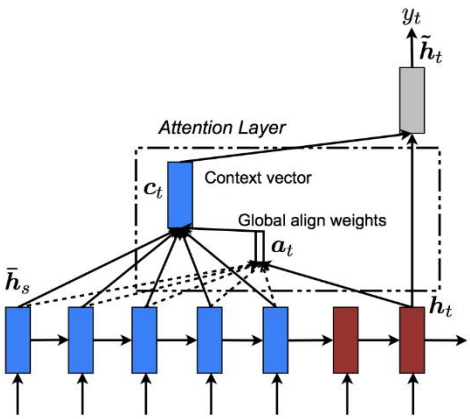

Fig 28: a summary of the cross attention process

The reason this experiment was not pursued further for this project was because of two reasons:

1. Issues in integration of this model with the self-attention model

Since cross attention is a complex mechanism it has to be written a keras layer for the model to compile the runtime graph. But not many good implementations where available for cross attention as a keras layer which was compatible with the latest tensorflow version which we were operating in.

Another issue was that it was hard to debug the error messages given by the model once we tried to integrate the cross attention layer on top of self-attention.

2. Implementing self-attention

Looking at the results given by transformer model from the “Attention is all you need” paper. It was quite clear that “Multi head self-attention” could learn a lot of finer and hidden abstractions and representations of the data by just implementing attention on itself as opposed to trying to attend on cross features.

We believed that if self-attention was implemented as a layer on top of either the encoder (which dealt with captions) or on top of the decoder(which dealt with images) there could be more better representations of the caption embeddings and the image embeddings learnt by just picking up the context from attention weights which focused on itself.

At a high level this would mean that the decoder would instead focus only on a few relevant group of pixels (like objects, or the color of the background etc.) as opposed to having a feature representation of the entire image. Similarly an argument can also be drawn towards the space of captions where self-attention implemented on the encoder side would mean that the model learns to focus only a few key words (like nouns and verbs for example) as opposed to the overall semantic structure of the sentence.

This way the representations/embeddings generated from implementing self-attention are more compact and much more meaningful.

We still believe that a combination of cross attention and self-attention is the right way to proceed because cross attention has worked really well in the field of image-captioning (ours is the inverse problem of image retrieval but the same principles apply) therefore this also counts as future work in our project in order to boost the model's performance even beyond what we currently have now.

7.2 Elmo

Implementation

There are 3 approaches to use ELMo pre-trained model.

They are:

- Weighted sum of the 3 layers with word embeddings
- Weighted sum of the 3 layers without word embeddings
- Fixed mean-pooling without word embeddings

Since we were using eager execution of tensorflow and ELMO takes the whole set of sentences and trains the model, we were unable to incorporate the same in the model.

It was just evaluated as a separate component independent of the model.

Caveats of ELMO embedding

- ELMo is quite **time consuming to compute vectors**. Per authors suggested, you can precompute token offline and lookup it during online prediction to reduce time spending.

7.3 Using Resnet embeddings

Here we had attempted to build a model which used the pretrained weights of the famed "Resnet-50" model as a part of our decoder while we were incorporating the bidirectional model into the architecture.

This model on training with 2 lakh images immediately converged to a loss of "0" within the first few epochs itself which resulted in "NAN" predictions by the model.

These "NAN" errors typically happen when the model has overfitted itself to the training data such that it has nothing more to learn from it or it could also be because of the activation functions resulting in an undefined computation on forward propagation and backward propagation such as 0/0 etc

This behaviour was unexplainable as that either meant that the model was overfitting or underfitting which was clearly not possible given the size of the data we were training with and the number of learnable parameters in the model.

Since this bug was not fixable even on repeated changes to the architecture we had decided to instead go ahead with using the "Inception" CNN stack as a pretrained model from which image features were extracted. Since this gave better results than the base model we decided to go with "Inception" as our CNN stack from which various other models was trained on from then onwards.

On doing a bit deeper digging as to the root cause of this error,(which many people face when training models which involve resnet) we realised that it has more to do with the optimal hyperparameter tuning.

For example a higher learning rate for a model typically tends to converge faster and since our default learning rate is a bit high to start off with this could be one of the reasons why our model failed.

Another reason could be the choice of the optimizer chosen which in our case the famed “adam optimizer” was.

But since this experiment required tremendous amount of time in debugging unexplainable issues we decided to skip it in order to try other CNN models.

```
Epoch 3/15
10/10 [==============================] - 30s - loss: nan - acc: 0.0000e+00 - val_loss: nan - val_acc: 0.0000e+00
```

Fig 29: Issues with the model tending towards 0 accuracy and 0 loss on the 3rd epoch itself

7.4 Addition of clustering

The current scope of clustering that the algorithm can identify and cluster the given set of images, if we can incorporate the prediction function to predict from only a set of related cluster which could possible contain the image fulfilling the description, it would boost the model efficiency.

The ideal way the clustering should be implemented is to make it a dynamic clustering algorithm which can directly group images into multiple clusters without having to initialise a hyperparameter to set the number of clusters.

Another idea we have in mind currently is to cluster the images based the image embedding features extracted from the decoder CNN model. These embeddings also inherently capture the caption embeddings as well because of the way the loss values are back propagated. This way images which “look” similar and have similar captions “defining” them will most probably be grouped in a single cluster.

<u>These are the caveats in the clustering approach</u>

- Usage of text and image embeddings together, since text data contains some of the keywords and image correspondingly contains those features, clustering would work the best when both of these are passed, viz., concatenate both the features.

- Another factor that might come into play with trickier situations is the general vs specific case of query.
    - Specific case - The query passed has a specific meaning and hence can be localised to a cluster and hence the prediction would look for the image in that cluster.
    - General case - The query is general to localise for one cluster and hence might require sequential search through multiple clusters

Sample Clustering output:

We ran the agglomerative clustering algorithm on the image features obtained from the trained model for all images in the test set.

Although the determination of the number of clusters must be dynamic, currently we fixed provided the number of clusters to be 6. The clusters obtained were then analysed. The observations were quite promising showing that clustering is actually a right approach.

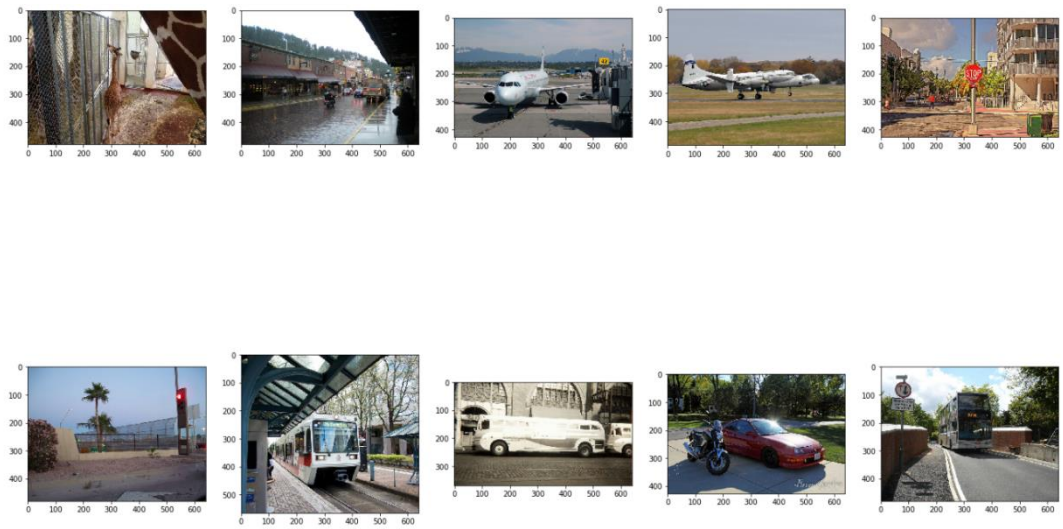

Fig 30: Cluster 1 - seems to be capturing the concepts of road, vehicles

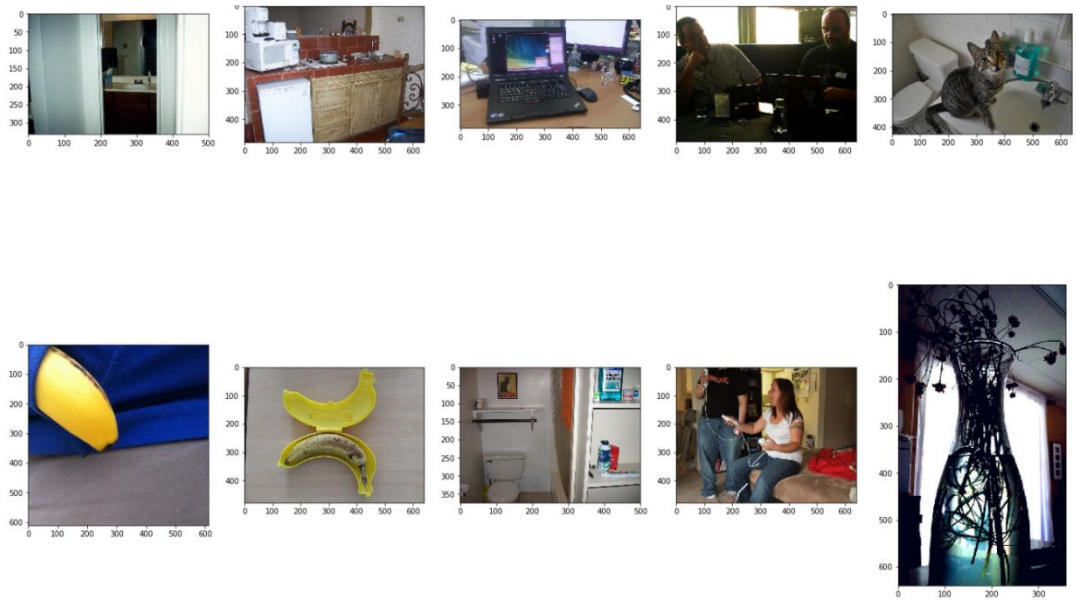

Fig 31: Cluster 2 - seems to be capturing concept of things

Fig 32: Cluster 3 - seems to be capturing the concept of people

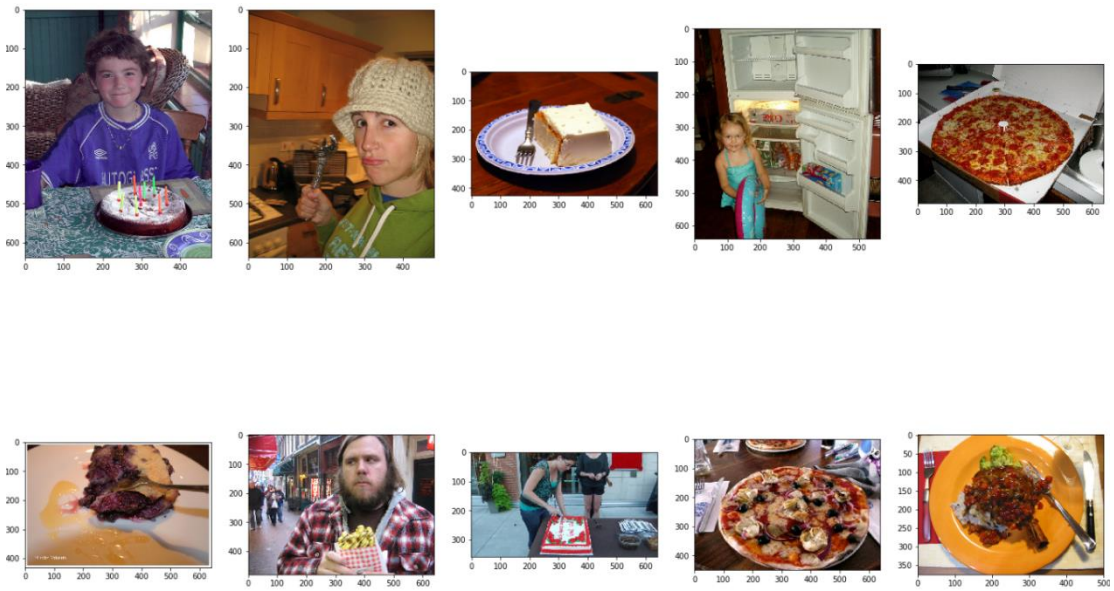

Fig 33: Cluster 4 - appears to be capturing the concept of food

Fig 34: Cluster 5 - appears to be capturing the concept of animals

Fig 35: Cluster 6 - appears to have captured the concept of sports and games

8 RESULTS

This section will firstly go over the definition of two metrics used typically in evaluating the performance of recommender systems which are namely:

1. Recall@K
2. mean rank

This will be followed by a tabular comparison of these aforementioned against all models we were successfully able to train the dataset with.

In order to make more sense of this table of numbers an intuitive explanation of what these numbers signify will be given in the section following the table.

To conclude this section we will be showcasing various predictions made by all our successful models by including several screenshots followed by some interesting observations made on what kind of images are being retrieved.

8.1 Metrics used in the project

Typically precision and recall are the classical evaluation metrics used in binary classification algorithms and used in information retrieval tasks.

These metrics have been translated to better suit the evaluation of "recommender systems". In order to understand the reason why we chose these metrics it is essential to first glance at a high level workflow of how recommender systems work.

8.1.1 Recommendation system workflow

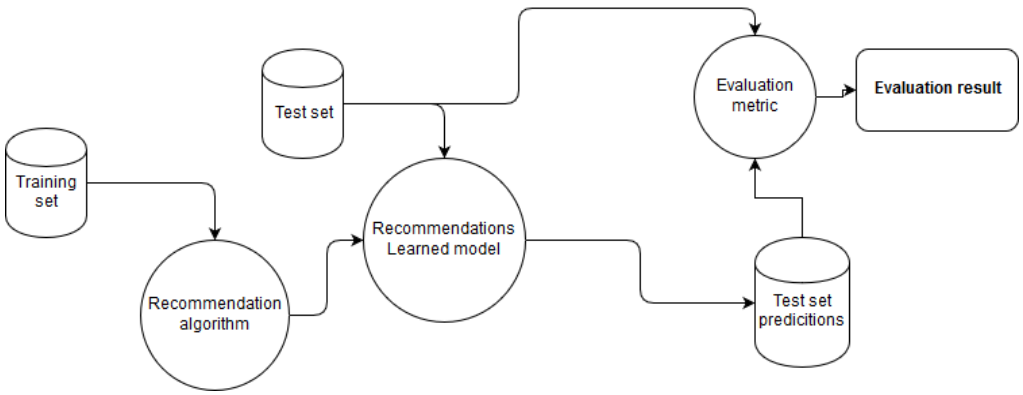

Fig 36: A typical recommender system workflow

In this diagram we can observe that, firstly a training set is fed to a recommendation algorithm which produces a recommendation model that can used in generating new predictions. To evaluate the model a separate test dataset is fed to the learned model where the predictions are generated for each user-item pair.

Predictions with known labels are then used as an input to the evaluation algorithm to produce evaluation results.

Although a variety of evaluation metrics exist we realized that many of the recommender system metrics might not apply for the problem of image retrieval given a caption.

For example, Precision@k is defined as *"the proportion of recommended items in the top-k set that are relevant"*. But we chose to not implement this because there is no way to tell how the predictions are relevant with respect to the user as the space of images retrieved from the the input query/caption can't be measured without considering a "*minimum threshold*" beyond which relevant predictions are cut off based on just looking at the similarity score computed.

Since there is no logical and intuitive reasoning for selecting how much this threshold should be we decided to instead go with another metric called as "*mean rank*" which gives a more intuitive understanding of how the model "*ranks*" various images with respect to another for a given rank.

Another metric we decided to implement is the popular Recall@k metric which is commonly used in recommender systems to provide a percentage of relevant information are found in the top-k predictions.

8.1.2 Mean Rank metric

In order to understand this metric we have to firstly define what "rank" means.

Assuming there are N images in the test dataset which has N corresponding captions lifted directly from the dataset. We firstly compute an NxN matrix (called test_errors) where, test_errrors[i][j] corresponds to the similarity score computed by our model for the ith image and the jth caption.

TEST_SCORES	image_1	image_2	image_3	image_4	image_5
caption_1	0.432	0.412	0.02	0.12	0.55
caption_2	0.82	0.001	0.124	0.422	0.243
caption_3	0.432	0.432	0.23	0.432	0.432
caption_4	0.432	0.432	0.432	0.003	0.432
caption_5	0.432	0.432	0.432	0.432	0.32

Table 3: A sample 5x5 test similarity score matrix

In the table above we can see the "test_errors" matrix being constructed for a test dataset size of N = 5.

Each cell in this table represents a similarity score between the range of 0 and 1, where the closer the score is towards 0 higher is the similarity.

Here the principal diagonal scores are highlighted in red because these numbers represent what score the model predicts for a corresponding caption and image pair.

Ideally we would expect the values in red to be the most similar i.e. most closest to zero but due to the probabilistic nature of machine learning models it may always not be the case.

Here, rank is defined as "the position/order of the test_errors[i][i] in the row containing test_errors[i]. for i∈[1,N]".

i.e., rank(i) = Rank of the value test_errors[i][i] in a sorted list of the $i^{th\ row}$ We define a rank of "1" as the cell with the highest similarity , therefore we would expect the principal diagonal elements to always have a rank of "1" but that might always not be the case.

So taking the example of the second row for the second caption, the corresponding cell value of the second image is the lowest therefore we can conclude that for the second caption across all captions in the test dataset, its rank is "1".

Whereas if take a look at the first row for the predictions of the first caption across all images, we can see that the cell marked in red (i.e. the principal diagonal element) is not the lowest score in that row. In fact it is the 4th lowest score therefore we can say that the rank of the second caption out of five ranks is "4".

Taking an average/mean of all the ranks across all rows/captions gives a good intuitive understanding as to how the model rates its own predictions.

Therefore, mean rank can be defined as.

$$meanrank = \frac{\sum_{i=1}^{N} rank(i)}{N} \quad (3)$$

Similarly, the mean rank across all columns could also be calculated which would intuitively explain how the model rates its predictions for a single image across all test captions.

This way both the mean rank for images as well as mean rank for captions can be obtained, where mean rank of images assigns ranks to all images for a single caption (i.e., one row) and takes the mean of all ranks assigned across all rows, whereas mean rank of captions assigns ranks to all captions for a single image (i.e., one column) and then takes the mean of all ranks assigned across all columns.

8.1.3 Recall@K metric

Recall@k can defined as "*the proportion of relevant items found in the top-k recommendations*".

For example let us suppose that we computed recall at 10 and found it is 40% in our top-10 recommendation system. This means that 40% of the total number of the relevant items appear in the top-k results.

Mathematically recall@k is defined as follows:

$$Recall@k = \frac{No\ of\ \ recommended\ items@top\ k\ that\ are\ relevant}{Total\ number\ of\ \ relevant\ items} \quad (4)$$

In the context of our project the way we find out which items are relevant is by using the "rank" metric as explained in section 8.1.2.

Therefore, using the "rank" metric " Recall@k" can be redefined as:

$$Recall@k = \frac{Number\ of\ \ predictions\ where\ the\ rank\ is\ lesser\ than\ k}{Total\ number\ of\ \ predictions\ in\ the\ test\ dataset} \quad (5)$$

For example let us assume that our test dataset is of size “N”. After computing the “test_errors” matrix of dimensions NxN (as discussed in section 8.1.2) we count the number of rows where that particular row’s rank is lesser than that of “k” among all the “N” rows.

Assuming “M” of those rows had a rank of lesser than “k”,(where M<=N) the Recall@K can be rethought of as the percentage of rows which have a rank lesser than or equal to that of k.
i.e

$$Recall@k = \frac{M}{N} \times 100 \qquad (6)$$

By formulating the metric in this manner one can see that the higher this percentage is, the better the model is in giving out retrieval predictions which implies that the model can generalise well with any new dataset as the test dataset is a mutually exclusive dataset from the training dataset.

For all our experiments we decided to implement the Recall@10 metric because for an application which has to throw out the top 10 images given a search query from the test dataset, this number needed to be as high as possible.

Since this metric uses rank internally it is possible to get both Recall@k for images as well as captions as there are two ways of finding ranks (refer the above section).

This was Recall@k for images refer to the percentage of image predictions whose “*image_rank*” is less than k across all predictions and the Recall@k for captions refer to the percentage of caption predictions whose “*caption_rank*” is less than k across all predictions.

8.2 Results from various models developed

Models	Recall@10 (caption)	Mean Rank (caption)	Recall@10 (image)	Mean Rank (image)
Baseline1 [resnet + GRU + fastext]	0.8613782	955.4006	0.70112179	888.9565304
Baseline2 [inception + GRU + fastext]	1.0016025	862.44831	1.08173076	777.8826121
Bidirectional [inception + Bidirectional(LSTM) + fastext]	9.7355769	175.64042	9.41506410	174.0995594
SA1 [SA(inception)+ Bidirectional(LSTM) + fastext]	12.960737	176.88421	12.8405448	179.7253605
SA2 [inception + SA(fastext)]	42.773437	38.964355	43.9941406	33.75537109
DSA [SA(inception) + SA(fastext)]	11.358173	170.31911	10.356570	184.9172676

Table 4: Metrics computed for all the successful models developed

In this table we can see on the left hand side is an enumeration of all the models which we were able to train successfully and the other columns are the metric values (“Recall@10” and “Mean rank”) which we have computed across all the models for both the captions and the images side of the encoder-decoder architecture.(refer sections 8.1.2 and 8.1.3 to understand the meaning of both image and caption side)

In the models column, each model has been named in the following format:

MODEL_NAME [configuration]

where the configuration refers to layers used in the model architecture.

Here the models labelled as SA1, SA2 and DSA refer to models which are different experiments conducted by using the "Self Attention" layer found in "transformer networks" (refer the references section).

1. SA1: Self Attention Model 1 where we used the multi head self-attention layer as a part of the decoder side. i.e on the images.
2. SA2: Self Attention Model 2 where we used the multi head self-attention layer as a part of the encode side. i.e on the captions
3. DSA: Double Self Attention Model where we used the multihead self-attention layer in both the encoder and the decoder side of the model architecture.

Some of the components listed in the configurations of each model are defined below:

- **Inception**: this is the CNN stack used by google trained on imagenet dataset used in the decoder part of the architecture
- **GRU**: this is also called as the "Gated Recurrent Unit" which are typically used as RNN units in the encoder part of the architecture
- **LSTM**: this is also called as the "Long Short Term Unit" which are also typically used as RNN units in the encoder part of the architecture, this can be thought of as a variant to GRU.
- **Bidirectional**: this is a layer which converts the RNN on the encoder side into bidirectional RNN which can capture more context from both the forward and the backward representations of a caption.
- **SA (Self Attention):** this layer implements the "Positional embedding" and the "Multi head self-attention" layer found in "transformer networks".

For calculating these metrics we firstly fixed a set of 5000 images as the test dataset.

After all models were sufficiently trained on 2 lakh images ,we then used the model to predict the "test_scores" for a matrix of size 5000 x 5000 after which these metric evaluation functions were run.

8.3 Inferences obtained from the results table

Here we can see that across 6 models the best model by far was model which incorporated multihead self-attention on the captions by neglecting any usage of bidirectional models or Recurrent Neural Networks.

This is probably because of the fact that "attention is all we need" paper talks about the fact the Attention models triumph over Recurrent Models because these models overcome the "vanishing gradient" problem which RNN's typically struggle with.

For this particular model we can see that both the Recall@10 metrics for both the captions and the image side are around the 42-43% which implies that around 2500 predictions out of the 5000 predictions made in the test dataset have a rank value of less than 10. This in turn implies that our model has been trained very well and can generalise across a new unseen test dataset very well.

Ideally we would want the model to always predict that image which best suits the caption and has a rank of 1 among all other 5000 predictions, but the fact that around half of the predictions have ranks of less than 10 implies that our model has not only learnt the correct correlation between the space of captions and images but it has also learnt how to generalise and pick other similar looking images as well which nearly suit that caption.

In addition to this, the mean ranks for both the captions and the images are around 30s from over 5000 ranks. Which implies that while over half of the predictions have ranks lesser than 10 the other half amount of predictions have ranks in the range of 30-50 out of the 5000 possible ranks it could have assigned.

Therefore, one can say that the models performance is directly proportional to how high the recall percentage value is and how low the mean rank is.

Going by this logic we can see that models which implemented self-attention have performed better than the base model and the bidirectional model and the model which has the bidirectional implementation performs better than the base model and is almost comparable with the self-attention model implemented on images.

The reason the bidirectional model outperforms the base model is primarily due to two factors:

1. Usage of LSTM over GRU provided the model with two hidden cell states as opposed to one.
2. Usage of the Bidirectional layer gave the model over 4 hidden cell states, and generally models which have more context tend to learn finer abstractions of the captions and hence resulted in better performance.

The primary reasons the models with self-attention performer better than the model with the bidirectional model is because of:

1. Multi Head Self attention trumps the recurrent /recursive implementation of RNN as this provides more context for the model to learn from while not suffering from the "vanishing gradient" problem faced by RNNs

But among the models which implement self-attention, the model which implemented self-attention on the captions outperformed the other two self-attention models, this behavior was unexpected but the reasons this could have happened are:

1. The hyper parameters needed in the Self attention layer vary for the space of captions vs the space of images.
2. Since images contain more spatial and temporal data the hyper parameters such as "number of heads" and "number of attention units" should have been increased for the self-attention layer applied on the images.

We believe that once the right hyper parameters are configured for both the self-attention layer on the caption side along with the self-attention layer on the image side the model will have enough context on what to "attend" to and what to ignore from both the encoder side and the decoder side.

We also believe that incorporating Bahadanau's cross attention (refer literature survey) on top of this double self-attention model, the model can also learn the correlations between objects

In the caption and the actual objects in the image as now the attention weights from this method depend on both the encoder's representation of the caption and the decoder's representation of the image. i.e., cross attention.

8.4 Looking at some example queries

8.4.1 A group of people eating pizza

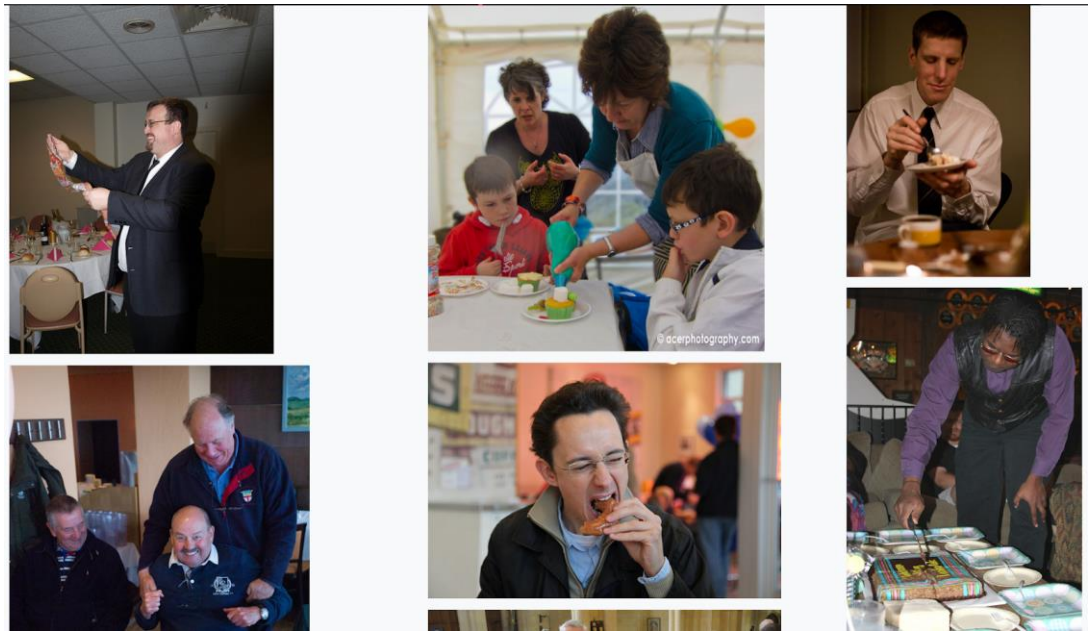

Fig. 37: caption 1- Base Model

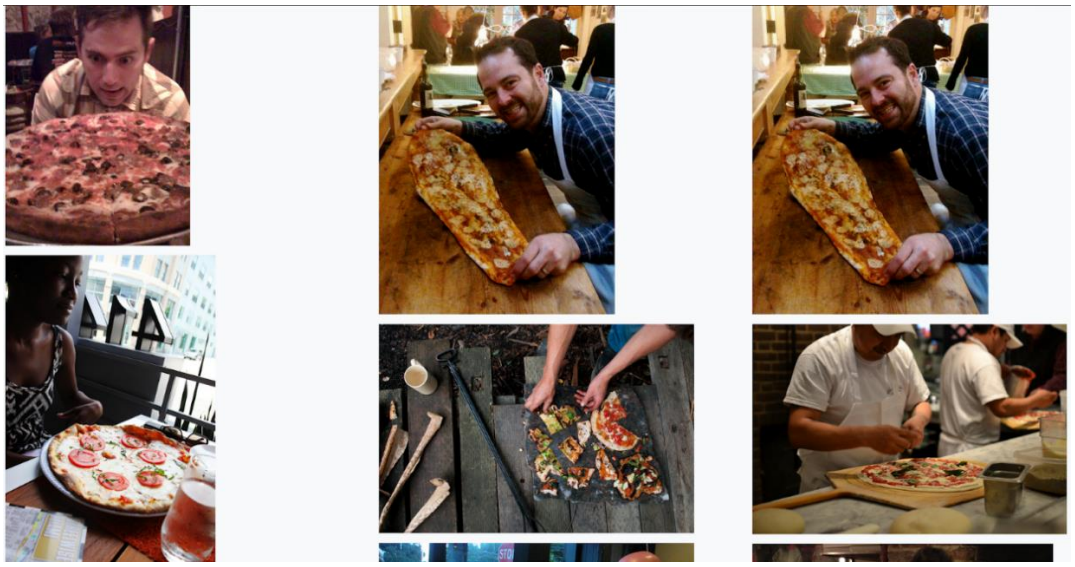

Fig. 38: caption 1 – Caption Self Attention model

Fig. 39: caption 1 – Image Self Attention model

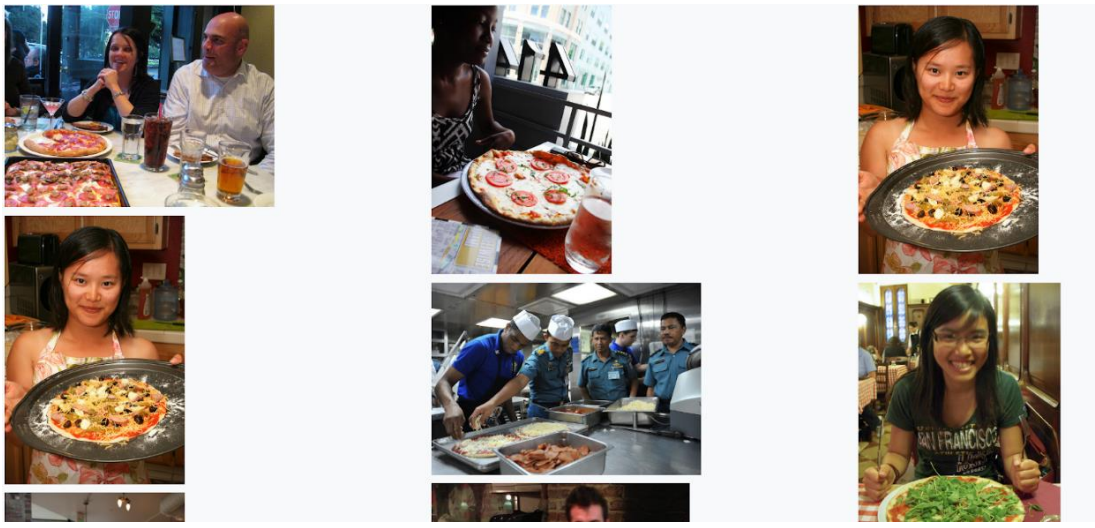

Fig. 40: caption 1 – Caption Image Self Attention

8.4.2. Surfing in the beach

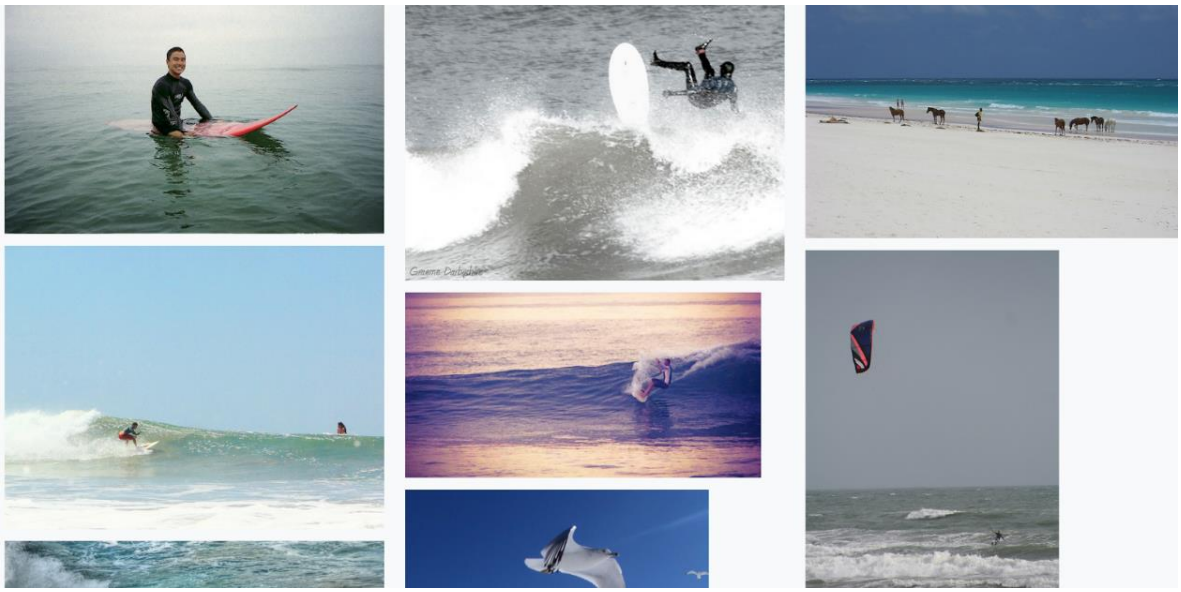

Fig. 41: caption 2 - Base Model

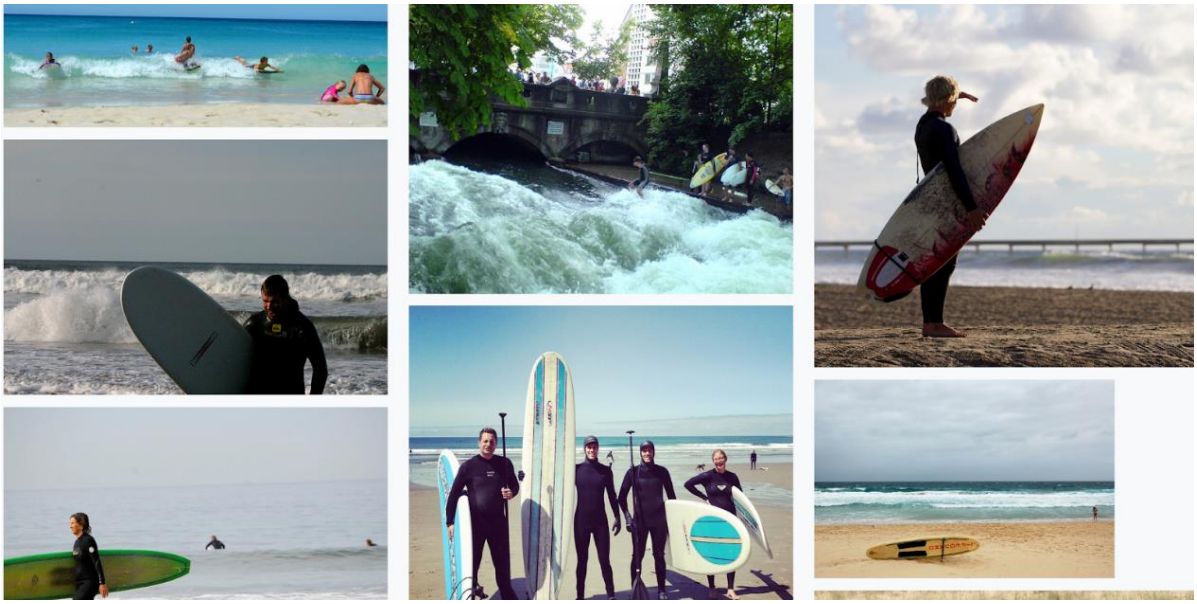

Fig. 42: caption 2 – Caption Self Attention model

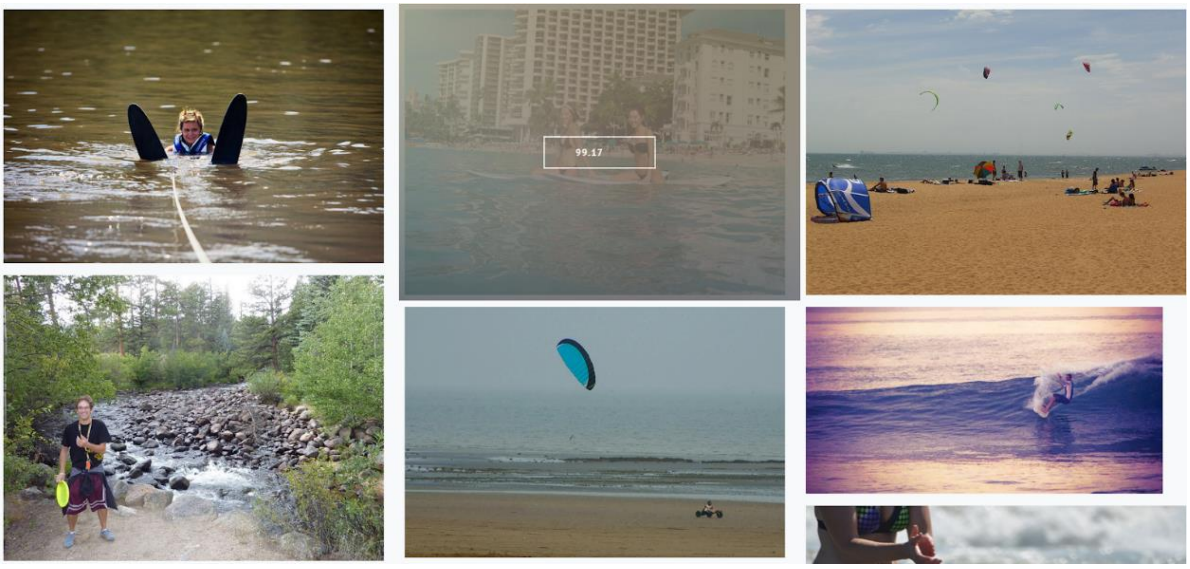

Fig 43: caption 2 – Image Self Attention model

Fig 44: caption 2 – Caption-Image Self Attention model

8.4.3. A bench in the park on a calm evening

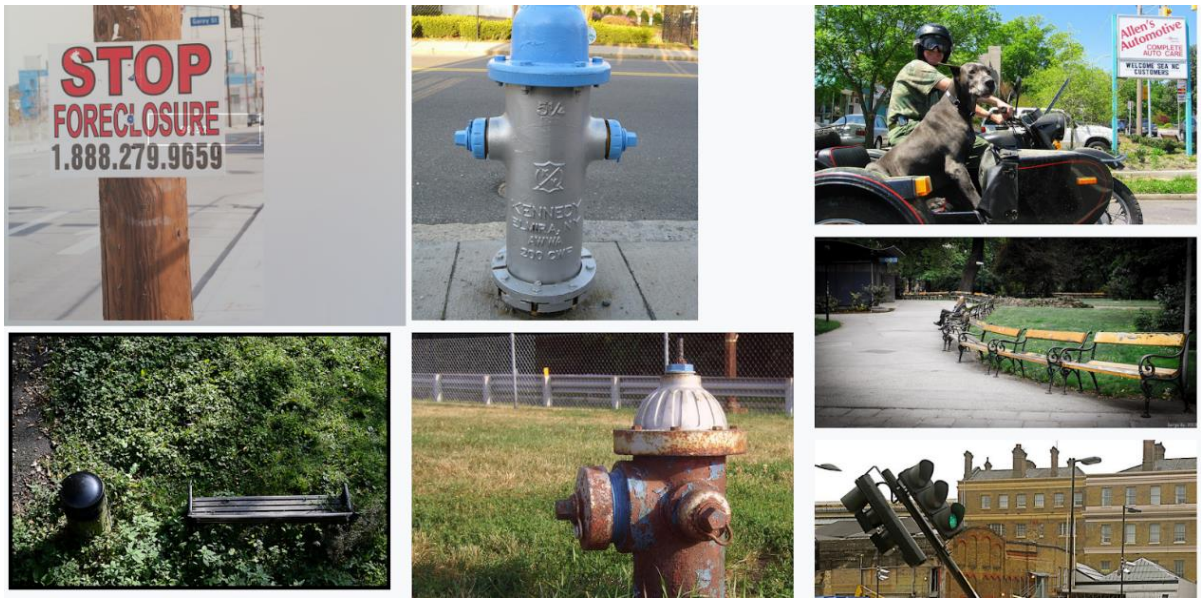

Fig 45: caption 3 – Base Model

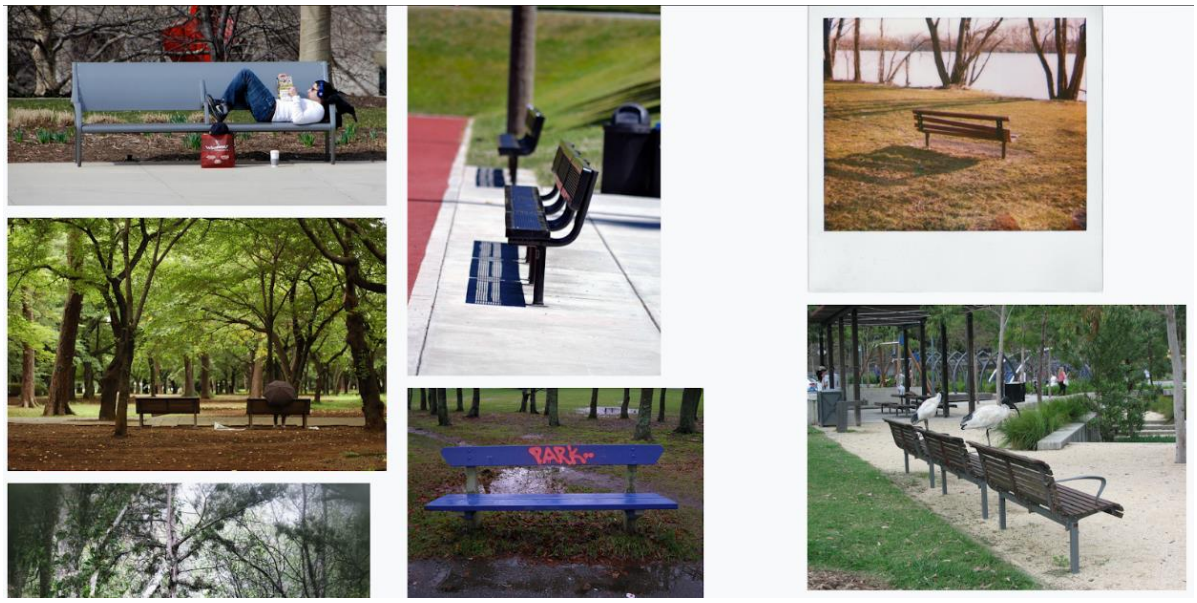

Fig 46: caption 3 – Caption Self Attention model

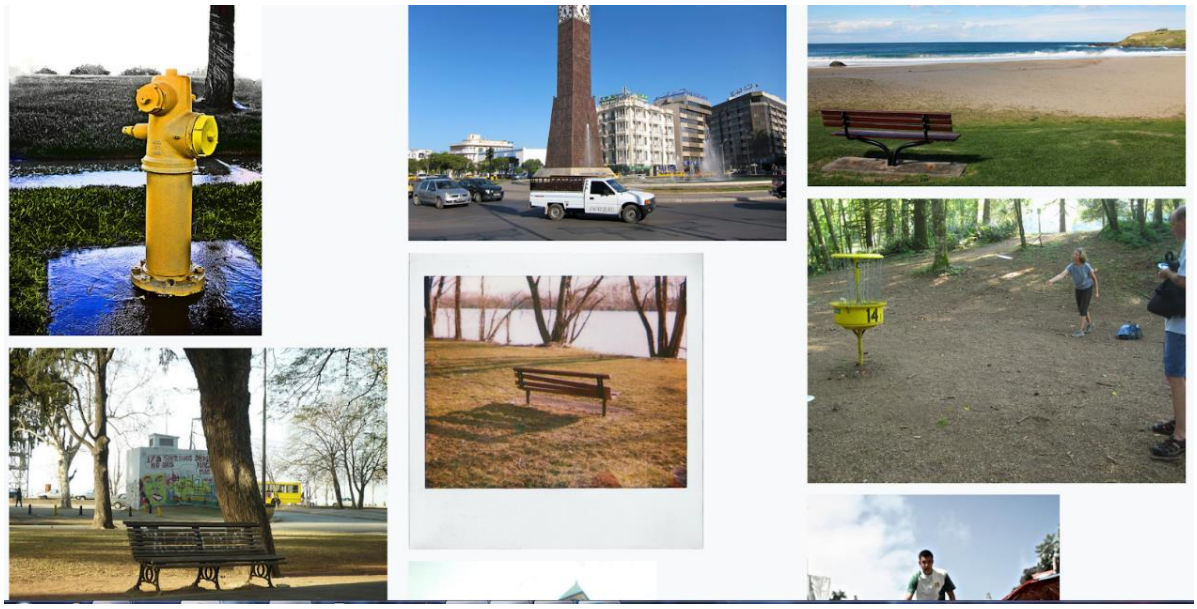

Fig. 47: caption 3 – Image Self Attention model

Caption-Image Self Attention model

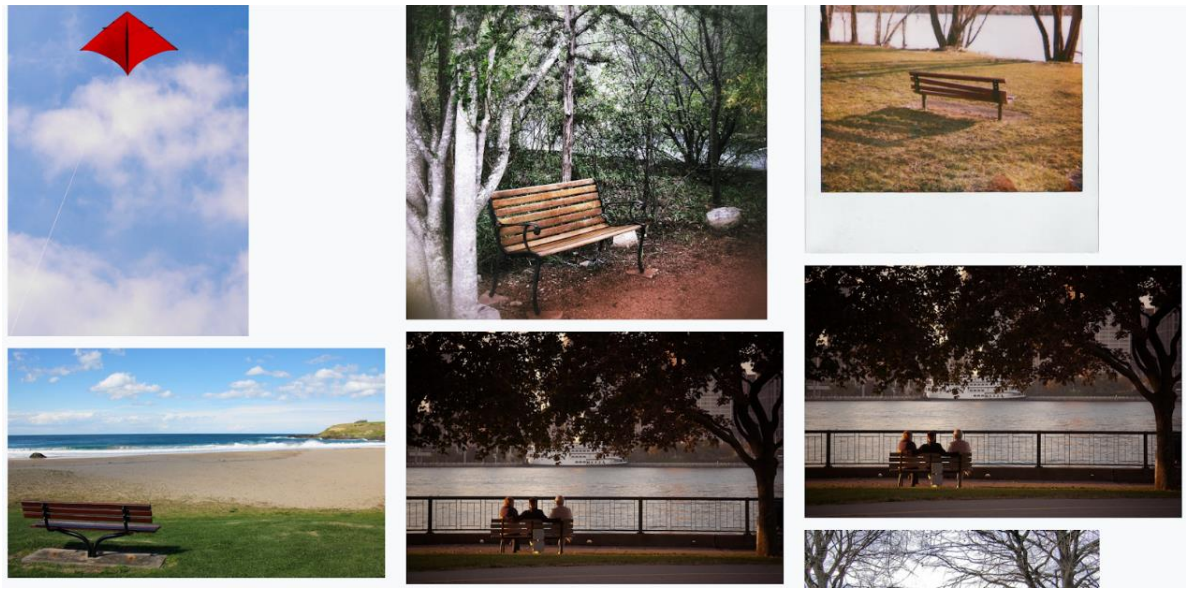

Fig. 48: caption 3 – Caption-Image Self Attention model

9 CONCLUSION

With the above work, we have expanded the horizon for research in multimodal machine learning. The fact that two varied modes, with structurally diverse dimensions can be combined and brought together on a common ground to process and extract the inherent relationship which is hard to perceive for human comprehension.

The model is a proof of how the deep learning algorithm we are using tries to learn and create its internal representation of the inputs to correlate and retrieve the right facts while it is being used for prediction. It essentially creates an internal map of image features in a convenient way so it can retrieve the images later on based on text features.

Our model, which provides a proof for the above mentioned concept was successful on most of the common scenarios that occur in reality. We saw that model brings together description and images in unique ways based on various levels of commonality. We observed the images being linked to a description based on its features, objects and also based on the background, colour, domain, action, complexity etc.,

The fact that the model with self-attention model is performing consistently validates the idea and provides a wide scope for research along the same lines. Applying cross attention between the word and image features and attending the most relevant features for the word level predictions will improve the model performance by multiple folds and Clustering approach we followed to narrow down the search space was also successful and if incorporated with the model would prove to be a state-of-the-art solution for the domain.

10 FUTURE WORK

10.1 Integration of clustering with the prediction algorithm

The bunching that has been done includes bunching the pictures into discrete gatherings, where each gathering speaks to a lot of related pictures.

The group could speak to

- A typical space of pictures
- A gathering of between related pictures
- bunch dependent on shading or shape conveyance
- in light of a typical example and so on.,

These groups speaks to an obliged set of pictures and consequently a littler hunt space, if the forecast calculation could channel the entire dataset dependent on the subtitle highlights, it would decrease the pursuit space by an extensive factor.

The present extent of grouping that the calculation can distinguish and bunch the given arrangement of pictures, on the off chance that we can consolidate the expectation capacity to foresee from just a lot of related bunch which could conceivable contain the picture satisfying the portrayal, it would support the model productivity.

The perfect way the bunching ought to be executed is to make it a dynamic grouping calculation which can straightforwardly assemble pictures into different bunches without initialising a hyper parameter to set the quantity of groups.

Another thought we have as a main priority at present is to bunch the pictures based the picture installing highlights removed from the decoder CNN model. These embeddings inalienably catch the subtitle embeddings also in view of the manner in which the misfortune esteems are back spread. Along these lines pictures which "look" comparative and have comparable inscriptions "characterizing" them will most presumably be assembled in a solitary group.

10.2 Cross attention implementation

The cross consideration is to make a walk towards going to differentially to fundamental picture locale and words with each other as setting for understanding the image content equivalence. This introduces a novel Stacked Cross Attention that enables thought with setting from both picture and sentence in two stages. In the proposed ImageText definition, given an image and a sentence, it at first deals with words in the sentence concerning each image region, and takes a gander at each image area to the went to information from the sentence to pick the essentialness of the image areas (for instance referenced in the sentence or not). Also, in the proposed Text-Image definition, it at first deals with picture locale in regards to each word and after that gives basically thought to each word.

The cross consideration approach essentially limits the picture to a specific zone while we take a gander at each word in the portrayal. This fundamentally lessens the region of center for the picture in a unique sense.
With cross consideration, the picture can be envisioned as a mass of highlights and thus parceled into that set and the model can concentrate on just the important element while anticipating a specific word.

10.3 Implementing multihead transformer stack

The Transformer plays out a little, consistent number of steps (picked observationally). In each progression, it applies a self-consideration component which straightforwardly models connections between all words in a sentence, paying little heed to their particular position. In the precedent, "I touched base at the bank in the wake of intersection the waterway", to discover that "bank" alludes to the shore of a stream and not a money related establishment, the Transformer can figure out how to quickly take care of "stream" and settle on this choice in a solitary advance. Indeed, consider English-French interpretation model we watch precisely this behavior. More explicitly, to figure the following portrayal for a given word - "bank" for instance - the Transformer looks at it to each other word in the sentence. The aftereffect of these correlations is a consideration score for each other word in the sentence. These consideration scores decide how much every one of different words ought to add to the following portrayal of "bank". In the model, the disambiguating "stream" could get a high

consideration score when figuring another portrayal for "bank". The consideration scores are then utilized as loads for a weighted normal of every one of words' portrayals which is nourished into a completely associated system to produce another portrayal for "bank", mirroring that the sentence is discussing a waterway bank.

The decoder works also, yet produces single word at once, from left to right. It goes to not exclusively to the next recently produced words, yet in addition to the last portrayals created by the encoder.

10.4 Research on more advanced recent embedding models

Bert embedding approach

BERT is a model that broke a few records for how well models can deal with language-based errands. Not long after the arrival of the paper depicting the model, the group likewise publicly released the code of the model, and made accessible for download adaptations of the model that were at that point pre-prepared on enormous datasets. This is an earth shattering advancement since it empowers anybody constructing an AI model including language handling to utilize this powerhouse as a promptly accessible segment – sparing the time, vitality, learning, and assets that would have gone to preparing a language-preparing model without any preparation.

The ELMO embeddings

In the event that we're utilizing this GloVe portrayal, at that point "stick" would be spoken to by this vector regardless of what the setting was. "Hold up a moment" said various NLP scientists, "stick"" has different implications relying upon where it's utilized. Why not give it an inserting dependent on the setting it's utilized in – to both catch the word significance in that setting just as other logical data?" Thus, contextualized word-embeddings were conceived.

Rather than utilizing a fixed installing for each word, ELMo takes a gander at the whole sentence before doling out each word in it an inserting. It utilizes a bi-directional LSTM prepared on a particular errand to have the capacity to make those embeddings.

ELMo gave a huge advance towards pre-preparing with regards to NLP. The ELMo LSTM would be prepared on a huge dataset in the language of our dataset, and afterward we can utilize it as a part in different models that need to deal with language

What's ELMo's secret?

ELMo picked up its language understanding from being prepared to foresee the following word in a grouping of words - an assignment called Language Modeling. This is helpful in light of the fact that we have tremendous measures of content information that such a model can gain from without requiring labels. We can see the shrouded condition of each unrolled-LSTM step cresting out from behind ELMo's head. Those prove to be useful in the inserting process after this pre-preparing is done.ELMo really goes above and beyond and trains a bi-directional LSTM – with the goal that its language model doesn't just have a feeling of the following word, yet additionally the past word.

ELMo thinks of the contextualized installing through gathering the shrouded states (and starting inserting) with a particular goal in mind (link pursued by weighted summation).

10.5 Shipping the model as a product – Strategy

The end product can be visualised as a set of API calls to train, fit and predict using the model we have defined. It can also have the flexibility to change some hyper-parameters as that might suit the user's requirements.

The final product will have the following endpoints:

- Model training function
- Image loading module
- Obtaining the predictions
- Storing the prediction

The benefit of this approach include

- Ability of the user to train on his specific domain or confidential data and yet not expose them to public.
- Ability of the user to train based on his data and fine tune the model.
- Make simple API calls to obtain prediction, without having to load or set up the entire model in his local set up.

11 REFERENCES

(APA format followed for referencing)

11.1 Literature Survey Papers

1. Weston, J., Bengio, S., & Usunier, N. (2011, June). Wsabie: Scaling up to large vocabulary image annotation. In *Twenty-Second International Joint Conference on Artificial Intelligence*.
2. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., ... & Bengio, Y. (2015, June). Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning* (pp. 2048-2057).
3. Gu, J., Cai, J., Joty, S. R., Niu, L., & Wang, G. (2018). Look, imagine and match: Improving textual-visual cross-modal retrieval with generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 7181-7189).
4. Guérin, J., Gibaru, O., Thiery, S., & Nyiri, E. (2017). Cnn features are also great at unsupervised classification. *arXiv preprint arXiv:1707.01700*.
5. Vendrov, I., Kiros, R., Fidler, S., & Urtasun, R. (2015). Order-embeddings of images and language. *arXiv preprint arXiv:1511.06361*.
6. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*(pp. 5998-6008)

11.2 Blogs

1. http://jalammar.github.io/illustrated-word2vec/
2. http://jalammar.github.io/illustrated-bert/
3. http://jalammar.github.io/illustrated-transformer/
4. https://towardsdatascience.com/light-on-math-ml-attention-with-keras-dc8dbc1fad39

11.3 Code snippets

1. https://www.kaggle.com/fareise/multi-head-self-attention-for-text-classification
2. https://github.com/thushv89/attention_keras
3. https://colab.research.google.com/github/tensorflow/tensorflow/blob/master/tensorflow/contrib/eager/python/examples/generative_examples/image_captioning_with_attention.ipynb

12 APPENDIX

12.1 CODE SNIPPETS

*** Contrastive loss function**

```
def contrastive_loss(labels, predict):
    """For a minibatch of sentence and image embeddings,
compute the pairwise contrastive loss, refer the original
report to understand this better"""
    global model_options
    margin = model_config['margin']
    predict = tf.identity(predict)
    s, im = tf.split(predict, [model_config['output_dim'],
model_config['output_dim']], axis = -1)
    im2 = tf.expand_dims(im,axis=0)
    s2 = tf.expand_dims(s,axis=1)
    errors = tf.reduce_sum(tf.pow(tf.subtract(im2, s2), 2),
2)
    diagonal = tf.diag_part(errors)
    cost_s = tf.maximum(tf.constant(0, dtype=tf.float32),
tf.add(tf.subtract(margin, errors), diagonal))
    diagonal2 = tf.reshape(diagonal, [-1, 1])
    cost_im = tf.maximum(tf.constant(0, dtype=tf.float32),
tf.add(tf.subtract(margin, errors), diagonal2))
    cost_tot = tf.add(cost_s, cost_im)
    zeroes = tf.zeros((model_config['batch_size'],),
tf.float32)
    cost_tot = tf.linalg.set_diag(cost_tot, zeroes)

    return tf.reduce_sum(cost_tot)
```

* Position Embedding Layer

```
class Position_Embedding(Layer):

    def __init__(self, size=None, mode='sum', **kwargs):
        self.size = size
        self.mode = mode
        super(Position_Embedding, self).__init__(**kwargs)

    def call(self, x):
        if (self.size == None) or (self.mode == 'sum'):
            self.size = int(x.shape[-1])
        batch_size,seq_len = K.shape(x)[0],K.shape(x)[1]
        position_j = 1. / K.pow(10000., \
                                 2 * K.arange(self.size / 2,
dtype='float32' \
                                ) / self.size)
        position_j = K.expand_dims(position_j, 0)
        position_i = tf.math.cumsum(K.ones_like(x[:,:,0]),
1)-1
        position_i = K.expand_dims(position_i, 2)
        position_ij = K.dot(position_i, position_j)
        position_ij = K.concatenate([K.cos(position_ij),
K.sin(position_ij)], 2)
        if self.mode == 'sum':
            return position_ij + x
        elif self.mode == 'concat':
            return K.concatenate([position_ij, x], 2)

    def compute_output_shape(self, input_shape):
        if self.mode == 'sum':
            return input_shape
        elif self.mode == 'concat':
            return (input_shape[0], input_shape[1],
input_shape[2]+self.size)
```

*** Self-Attention Layer**

```
class Attention(Layer):

    def __init__(self, nb_head = 8, size_per_head = 256,
**kwargs):
        self.nb_head = nb_head
        self.size_per_head = size_per_head
        self.output_dim = nb_head*size_per_head
        super(Attention, self).__init__(**kwargs)

    def build(self, input_shape):
#         print("input shape - ", input_shape)
        self.WQ = self.add_weight(name='WQ',

shape=(input_shape[0].as_list()[-1], self.output_dim),

initializer='glorot_uniform',
                                  trainable=True)
        self.WK = self.add_weight(name='WK',

shape=(input_shape[1].as_list()[-1], self.output_dim),

initializer='glorot_uniform',
                                  trainable=True)
        self.WV = self.add_weight(name='WV',

shape=(input_shape[2].as_list()[-1], self.output_dim),

initializer='glorot_uniform',
                                  trainable=True)
        super(Attention, self).build(input_shape)

    def Mask(self, inputs, seq_len, mode='mul'):
        if seq_len == None:
            return inputs
        else:
```

```
            mask = K.one_hot(seq_len[:,0],
K.shape(inputs)[1])
            mask = 1 - tf.math.cumsum(mask, 1)
            for _ in range(len(inputs.shape)-2):
                mask = K.expand_dims(mask, 2)
            if mode == 'mul':
                return inputs * mask
            if mode == 'add':
                return inputs - (1 - mask) * 1e12

    def call(self, x):
#         print("in call!")
#         print("x =", x)
        if len(x) == 3:
            Q_seq,K_seq,V_seq = x
            Q_len,V_len = None,None
        elif len(x) == 5:
            Q_seq,K_seq,V_seq,Q_len,V_len = x
        Q_seq = K.dot(Q_seq, self.WQ)
        Q_seq = K.reshape(Q_seq, (-1, K.shape(Q_seq)[1],
self.nb_head, self.size_per_head))
#         print("Q_seq1 =", Q_seq)
        Q_seq = K.permute_dimensions(Q_seq, (0,2,1,3))
#         print("Q_seq2 =", Q_seq)
        K_seq = K.dot(K_seq, self.WK)
        K_seq = K.reshape(K_seq, (-1, K.shape(K_seq)[1],
self.nb_head, self.size_per_head))
        K_seq = K.permute_dimensions(K_seq, (0,2,1,3))
        V_seq = K.dot(V_seq, self.WV)
        V_seq = K.reshape(V_seq, (-1, K.shape(V_seq)[1],
self.nb_head, self.size_per_head))
        V_seq = K.permute_dimensions(V_seq, (0,2,1,3))
        A = K.batch_dot(Q_seq, K_seq, axes=[3,3]) /
self.size_per_head**0.5
        A = K.permute_dimensions(A, (0,3,2,1))
        A = self.Mask(A, V_len, 'add')
        A = K.permute_dimensions(A, (0,3,2,1))
        A = K.softmax(A)
```

```
        O_seq = K.batch_dot(A, V_seq, axes=[3,2])
        O_seq = K.permute_dimensions(O_seq, (0,2,1,3))
        O_seq = K.reshape(O_seq, (-1, K.shape(O_seq)[1],
self.output_dim))
        O_seq = self.Mask(O_seq, Q_len, 'mul')
#         print("\n\n\n\n", O_seq)
        return O_seq

    def compute_output_shape(self, input_shape):
        return (input_shape[0].as_list()[0],
input_shape[0].as_list()[1], self.output_dim)
```