University of Southampton

Faculty of Engineering and Physical Sciences

Electronics and Computer Science

Artificial Intelligence for Web Accessibility

by

Shaunak Sen

January 2020

Supervisor: Prof. Mike Wald
Second Examiner: Prof. Timothy J Norman

A dissertation submitted in partial fulfilment of the degree of MSC Data Science

University of Southampton

ABSTRACT

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES
ELECTRONICS AND COMPUTER SCIENCE

Master of Science

by Shaunak Sen

This project aims to explore how Artificial Intelligence (AI) techniques like Deep Learning and Natural Language Processing (NLP) can be applied to improve web accessibility. Primarily, we have considered two tasks - automatically captioning images and detecting if the hyperlink text is contextual or not. Various neural network architectures and NLP techniques have been applied to both these tasks. For the first task, a lot of work has also been done on applying word embedding techniques to improve evaluation systems of image captioning models. Also, object detection and image-text similarity have been considered as possible extensions to the current task. For the second task, the major contribution of this project is extending an existing dataset to suit the aim. Also, analysis has been done on how existing pre-trained embeddings compare with embeddings trained using the generated dataset.

Acknowledgements

I would like to express my sincere gratitude to my supervisor prof Mike Wald for motivating and guiding me through the project. I would like to thank the professors of this college for the modules that they taught on the relevant subjects which provided me with the foundation knowledge required to complete a project in this field. I would also like to thank prof Tim Norman for his feedback during the demonstration.

This project would not have been possible without the open-source tools, and the excellent quality of reading materials that are available in the fields of AI. I would especially like to thank Jason Brownlee's blog Machine Learning Mastery and Andy Thomas's blog Adventures in Machine Learning for the excellent tutorials and posts. I have cited these tools and tutorials in my work wherever applicable.

I also would like to thank my parents, friends, and classmates who have supported and motivated me during this phase. Especially I would like to thank my partner, Manisha for her constant support and advice.

Statement of Originality

- I have read and understood the ECS Academic Integrity information and the University's Academic Integrity Guidance for Students.
- I am aware that failure to act in accordance with the Regulations Governing Academic Integrity may lead to the imposition of penalties which, for the most serious cases, may include termination of programme.
- I consent to the University copying and distributing any or all of my work in any form and using third parties (who may be based outside the EU/EEA) to verify whether my work contains plagiarised material, and for quality assurance purposes.

You must change the statements in the boxes if you do not agree with them.

We expect you to acknowledge all sources of information (e.g. ideas, algorithms, data) using citations. You must also put quotation marks around any sections of text that you have copied without paraphrasing. If any figures or tables have been taken or modified from another source, you must explain this in the caption and cite the original source.

I have acknowledged all sources, and identified any content taken from elsewhere.

If you have used any code (e.g. open-source code), reference designs, or similar resources that have been produced by anyone else, you must list them in the box below. In the report, you must explain what was used and how it relates to the work you have done.

I have not used any resources produced by anyone else.

You can consult with module teaching staff/demonstrators, but you should not show anyone else your work (this includes uploading your work to publicly-accessible repositories e.g. Github, unless expressly permitted by the module leader), or help them to do theirs. For individual assignments, we expect you to work on your own. For group assignments, we expect that you work only with your allocated group. You must get permission in writing from the module teaching staff before you seek outside assistance, e.g. a proofreading service, and declare it here.

I did all the work myself, or with my allocated group, and have not helped anyone else.

We expect that you have not fabricated, modified or distorted any data, evidence, references, experimental results, or other material used or presented in the report. You must clearly describe your experiments and how the results were obtained, and include all data, source code and/or designs (either in the report, or submitted as a separate file) so that your results could be reproduced.

The material in the report is genuine, and I have included all my data/code/designs.

We expect that you have not previously submitted any part of this work for another assessment. You must get permission in writing from the module teaching staff before re-using any of your previously submitted work for this assessment.

I have not submitted any part of this work for another assessment.

If your work involved research/studies (including surveys) on human participants, their cells or data, or on animals, you must have been granted ethical approval before the work was carried out, and any experiments must have followed these requirements. You must give details of this in the report, and list the ethical approval reference number(s) in the box below.

My work did not involve human participants, their cells or data, or animals.

ECS Statement of Originality Template, updated August 2018, Alex Weddell aiofficer@ecs.soton.ac.uk

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction and motivation

The impact of Artificial Intelligence (AI) has already taken huge proportions and is affecting billions of human lives. AI holds enormous potential and can assist humans and complement traditional applications in a wide variety of tasks. This deviation from traditional programming in which programmers specified complex rules and logic to enable a computer to solve a problem towards a more data-driven deep learning-based approach in which we train the computer with lots of examples and it figures out an estimate of the underlying function, much like how young children process information and learn from them has been possible due to the explosion of big data. As the availability of relevant data increases the performance of these models also increases as shown the figure 1.1.

Also, AI has been used to assist humans in basic sensory functions like captioning speech from videos, natural language understanding enabled bots that can interact and converse with humans, and computer vision applications that can view and translate the world around us. In less than 25 years, computers went from manipulating 0 and 1 digits to utilizing neural network and deep learning technology to enable rapid progress in fields like computer vision and natural language understanding (29).

FIGURE 1.1: Performance of Deep Learning models with data (55)

Web accessibility is an important concern with the rapidly increasing number of internet users and online content in this modern digital age. Internet usage penetration rate has grown over 1000 % in the past decade and currently, over 58% of the world, i.e. over four and a half billion people are connected to the internet (20). The World Health Organization (WHO) estimates that about 15% of the worlds population lives with some form of disability (35), and this accounts for over 600 million internet users. It is a challenge for them to consume internet content at the same speed and efficiency as any other person. In this scenario, technology can be used to assist them and aid them and improve their online experience. Recently Equadex partnered with Microsoft to create an application that uses AI technologies like computer vision, speech to text, text translation, and natural language understanding to help autistic children better interact and communicate with their environment (43). This is an example in which AI technology can not only assist disabled people but enable them with new ways to interact with the world that was not possible a few years back. Recently, Microsoft launched its app, Seeing AI which leverages the camera of a smartphone and uses deep learning and computer vision algorithms to look at objects in the surrounding of a person and describes them (38).

However, the application of this technology in the field of web accessibility, tools that leverage the scope of AI to improve web accessibility for the blind has been limited. Most popular screen readers like NVDA, ORCA, Apple VoiceOver, and ChromeVox use a simple text-to-speech mechanism to read out the website for a blind person and help navigate them (33). The idea of this project will be to explore how AI technologies can augment these tools and improve the experience of disabled people. This has been a largely unexplored field and the scope of research in this field is huge, and if correctly applied it can help millions of people in their experience on the internet.

1.1 Project aim

The primary aim of this project is to explore how Artificial Intelligence technologies can be applied in a manner to help make Web content more accessible, primarily for people with disabilities and apply them in this regard.

The secondary aim of this project is to optimize and evaluate the algorithms created, build upon them and explore how they can be extended to other tasks in this field.

1.2 Objectives and methodology

The first objective in this project would be to explore how AI technologies can be applied to the task of automatically captioning of images with a motivation to help

visually impaired people better understand what the image is trying to convey. This task will involve:

1. Research and experiment on appropriate datasets suitable for this task
2. Research and experiment on different model architectures and optimize the chosen model
3. Evaluate the model using standard metrics for related tasks
4. Apply Natural Language Processing (NLP) techniques to better quantify model performance
5. Explore how automatic captioning can be extended to other related tasks that would improve the web experience of visually impaired people

The next objective would be to detect if hyperlink texts are contextual to with the source page text and target page text. Often, on many websites, we find hyperlinks occurring randomly, unrelated to the current topic being discussed. My aim in this project is to find out if the hyperlink text, the text surrounding the hyperlink and the text (and title) of the link it points to are in close context or not. This will help detect if the hyperlink is contextual and relevant to the topic being discussed. This task will involve:

1. Research and experiment on appropriate datasets suitable for this task
2. Extend available datasets, if needed, for this task
3. Build word embedding based networks to learn word associations and develop algorithms for detecting the similarity between hyperlink text, the source text, and the target link text.
4. Apply and extend the NLP techniques in the previous task for this problem
5. Compare results with pre-trained models on publicly available datasets and understand the scope of creating separate text corpus for such scenarios

While developing the project and making design choices, the scope of deploying the proposed solutions to the web has been largely taken into account. The algorithms have been optimized to train and provide results faster, and both time and space complexities have also been considered while writing the code.

Risk category	Risk	Likelihood	Impact (days)	Risk measure	Contingency plan
Data	Correct dataset not found	0.5	7	3.5	Search for alternate dataset; ask assistance from supervisor
	Data inconsistency - missing values, out of range values, spelling mistakes etc.	0.8	2	1.6	Correct dataset. Use text pre processing methids and software like OpenRefine
	Data incomplete - dataset not large/diverse enough for task	0.7	7	4.9	Try to extend dataset by scraping; ask assistance from supervisor
Software	Library/framework does not support a particular function	0.4	3	1.2	Try looking for alternative function, else implement it from scratch
	Library/framework update causing code to be depricated	0.4	5	2	Modify code to work with current update
	Library/Framework out of service	0.2	20	4	Reach out to the developers; look for alternate library; ask assistance from supervisor
Cloud platform	Cloud platform - RAM not sufficient	0.5	10	5	Configure AWS server/university deep learning resources
	Cloud platform - RAM upgrade not working	0.3	10	3	Try to optimize code; Configure AWS server/university deep learning resources
	Cloud platform - GPU support disabled	0.5	10	5	Configure AWS server/university deep learning resources
API	API - request limit exceeded	0.6	4	2.4	Run code in multiple instances from multiple IP addresses
	API depriciated	0.3	7	2.1	Modify code to work with current update; look for alternate API
	API response not in JSON or XML or any known standard	0.3	5	1.5	Search for libraries to support standard; write custom parser for the data format

FIGURE 1.2: Risk Analysis

1.3 Risk Analysis

Deep Learning based applications and algorithms are computationally expensive. Also the models have to be trained and optimized experimentally, which is a time-consuming process. So slight errors can cause a big loss in time, and possible risks should be kept in mind.

In developing these applications, errors are bound to occur and they will take time to fix. However, it does help to think about some possible risks and possible contingency plans to overcome the situation quickly and effectively. Keeping this is mind, a risk analysis plan was developed.

For every risk, we assign a corresponding probability or likelihood (`P`) of it occurring. Then, we also assign the impact the days will have. Since this project involves no cost, the impact (`I`) is the time (in days) approximately needed to fix the risk. The Risk measure (`R`) can be calculated as $R = I \times P$ (9). This has been shown in Figure 1.2

Based on the severity of the Risk Measure value (green means Risk Measure is low and red means high) and the type of risk, we can assign appropriate contingency plans. Because risk measure incorporates both probability and impact, we should handle risks with a high value of Risk Measure with utmost caution. This is why for these risks associated with high-risk measure, asking for assistance from the supervisor is a good contingency plan.

Chapter 2

Preliminaries and research

As discussed previously in section 1.2, the primary aim for this project will be to develop an automatic image captioning system. A lot of work has been done in this field and this project is about understanding and exploring how these deep neural network architectures can be applied to the problem. This chapter discusses some of the key technologies that will be used for this project and also provides an overview of how past contributions and research can be leveraged for the development.

2.1 The role of Convolutional Neural Networks (CNNs)

The ImageNet competition (42) comprises a large database (25) containing around 14 million images. These images are grouped into classes resembling the semantic hierarchy of WordNet (34). The competition promotes developing computer vision algorithms primarily for the tasks of image classification and object detection.

CNNs have gained huge popularity ever since AlexNet was developed. AlexNet (22) used a comparatively small deep neural network (five convolution layers) and 3 fully connected layers to get state-of-the-art results in the ImageNet Large Scale Visual Recognition Challenge competition in 2010.

2.1.1 The working of a basic CNN

The working of a basic CNN can be explained through the diagram shown in figure 2.1. The basic steps involved in such a neural network are described below:

1. Each convolution layer comprises of a number of image filters (kernels) that span over the image multiplying the corresponding image pixel values to the filter weights. This operation is called convolution.

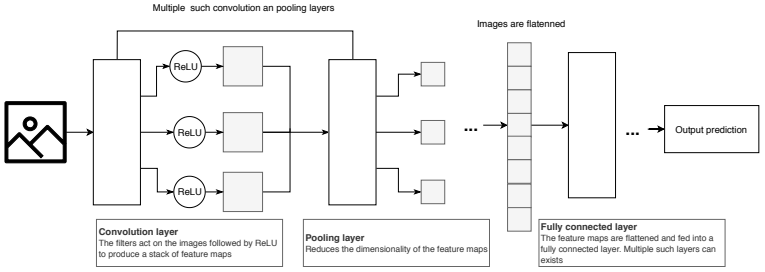

FIGURE 2.1: Basic CNN architecture

2. By setting the filter weights appropriately the network learns patterns in the image like edges, borders, shapes, colors, texture, etc.

3. The convolution is generally followed by an activation function that helps achieve non-linearity, popular ones being Rectified Linear Unit (ReLU) or hyperbolic tangent (tanh) (22)

4. This convolving operation of the filters results in a set of new images called activation or feature maps. Thus, every image, after a convolution operation becomes a stack of feature maps.

5. The next step is usually to apply a Pooling operation to reduce the dimensionality of the feature maps for the remaining part of the network, while still preserving the pattern. For example, AlexNet (22) uses a Max Pooling operation, which effectively reduces the dimensionality of the feature maps by half.

6. To prevent overfitting, AlexNet and many popular CNN architectures use a technique called Dropout (15), which basically ignores the contribution of each hidden neuron with a certain pre-set probability.

7. Multiple such Convolution and Pooling layers are stacked together. The output of the last layer is flattened and fed into a fully connected (dense) layer.

There may be multiple such layers. The final output from this layer is generally a K dimensional vector, K being the number of classes to predict. These K output variables represent the corresponding probability of the image falling in that particular category and are generally accompanied by a softmax activation function. The network learns through the backpropagation algorithm (21) - after every iteration, the weights of the filters, as well as the fully connected layers, are updated to reduce the loss of the network.

2.1.2 Transfer Learning

Transfer Learning can be used for the task of captioning images. In transfer learning (66), we use a network, or part of a network that has been trained on a related task

(like image classification) and use it for a similar task. By doing this we hope that the network has learned some general features about the data it was originally trained on, and these learned features can be applied to the task we are aiming to solve.

Oxford Visual Geometry Group or VGG model won the ImageNet competition described above in 2014 (46). Since then it has been used as a transfer learning CNN model for a wide variety of applications. What makes it a good model for our use case is:

1. The network is fairly simple to understand and implement
2. Since the network is only 16 or 19 layers deep, the number of trainable parameters is quite manageable
3. The network relies on minimal pre-processing. The images are normalized by mean centering each pixel value. Other than that, no pre-processing is necessary
4. The network is sophisticated enough to perform classification with high accuracy and also simple enough to generalize well to other datasets and problems

The role of the CNN model used in the Image Captioning task is to basically understand the features that distinguish each object. The final layer of the VGG model is a fully-connected layer with 1000 classes. We remove this layer as for the purpose of our problem, we are not interested in the final classification. We look one layer back. This layer is a fully-connected layer with 4096 neurons. This is the layer that extracts the high-level features from the image. We use the output of this layer for the captioning model

2.1.3 Transfer Learning in Word Embeddings

Neural network architectures have become popular in Natural Language Processing tasks. (31) uses a method to learn efficient word embeddings in vector space. Each word is represented in vector space by a dense vector of a pre-fixed dimensionality. Also the model is trained in a manner that similar words occur in close proximity in vector space.

Pre-trained architectures and word embeddings exist (12) in this regard, which can be used for the contextual hyperlink detection task as discussed in section 1.2.

2.2 The role of Recurrent Neural Networks (RNN)

Recurrent Neural Networks (RNNs) have achieved outstanding results in sequence to sequence-based modeling tasks (45). The task of captioning an image can also be viewed

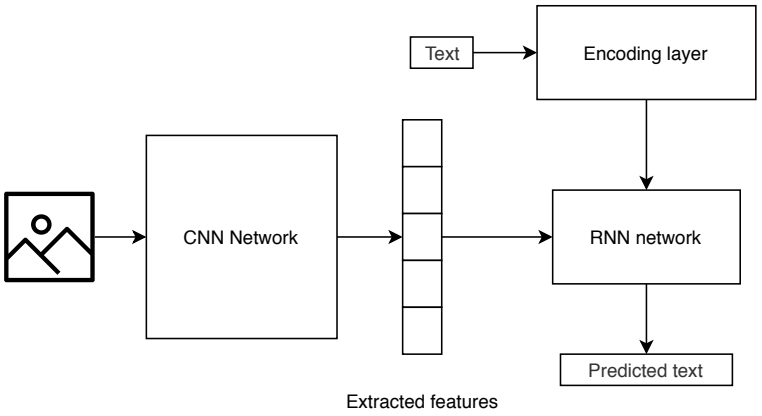

FIGURE 2.2: Proposed solution - overview

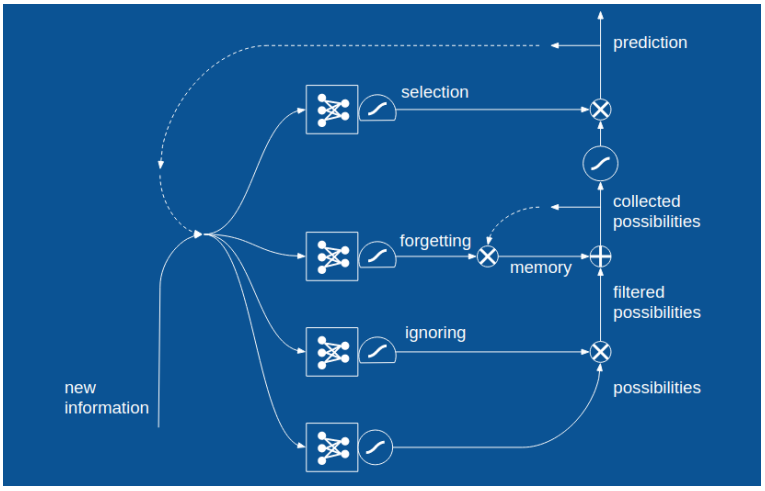

FIGURE 2.3: Basic architecture of LSTM(41)

as such. Given a set of features from an image we want to translate (caption) these features to text that would represent the corresponding captions. We discussed already that the output of our CNN model will be a list of features that characterize the image. The idea is to feed these features along with part of the training caption into the RNN model and have it predict the next word in the sequence. This is described in the figure 2.2:

The problem with vanilla recurrent neural networks is that they look back only limited time steps. They doe not have a feature to retain items in memory. Basically the problem is that it has very short term memory (1). The solution is to use a new type of RNN called Long short-term memory (LSTM) (16).

The key part of LSTM is the addition of a memory branch as shown in 2.3. This enables the network to store information that is potentially useful. The network learns what information to store and what to forget through another neural network that learns this through the training data. The selection part is like a filter that filters the relevant predictions from the collected possibilities. The last part is like an attention mechanism

that filters the possible possibilities so that the memory is not overloaded (41). This entire system can sufficiently overcome the problems of RNN discussed earlier and will be an essential part of this project.

Chapter 3

Automatic Image Captioning System

3.1 The Problem

The World Wide Web Consortium (W3C) is an organization responsible for developing and maintaining web standards such as HTML, CSS, etc. (57). The Web Content Accessibility Guidelines (WCAG) is developed through the W3C process and it aims to create and maintain a single set of guidelines and recommendations for individuals, organizations, and governments internationally to follow to make web content more accessible and inclusive, especially for people with disabilities (58; 64).

Guideline H37 (56) of the WCAG focuses on the proper use of alternative (alt) texts for images to help visually impaired people understand the message the image is trying to convey. Often developers fail to provide the above-mentioned alt texts and even if they do, the text does not really convey the message of the image.

Automatic Image captioning is a challenging task because it combines the workings of both CNNs and RNNs together. The CNN must understand the high-level features of the image and the RNN must translate these features into relevant captions as discussed in subsection 2.1.2.

3.2 The Dataset

There are several options for a dataset of images accompanied by their corresponding captions. Some of these are Flickr8k (17), Microsoft COCO: Common objects in context (MSCOCO) (27), Conceptual captions dataset by Google (44). I have used the Flickr8k dataset for this task because of the following reasons:

1. The dataset is relatively small so that the model can be trained using reasonable RAM (less than 30GB). As mentioned in section 1.3, there are risks associated with RAM over-utilization and RAM demand should be taken into account
2. The dataset is representative and contains a wide variety of images
3. Each image has five caption texts that were collected via crowdsourcing
4. The dataset contains over eight thousand images split into training (six thousand) validation (one thousand) and testing (one thousand) parts
5. There exists multiple copies of the dataset, available for downloading, so we have options in case the primary link does not work. As dicussed in section 1.3, this is a good option to have

3.3 Data Cleaning and pre-processing

In this task, we are dealing with both image data as well as textual data, which has been crowdsourced (17). Data cleaning and preprocessing is very important for the performance of the deep learning model.

The pre-processing steps vary for images and text. The pre-processing for the images involves:

1. Resize the images to dimensions: 224x224x3 - 224 is the image height and width (in pixels). 3 denotes the number of color channels (RGB)
2. Normalize the images by mean centering them. The mean RGB value was subtracted from each pixel value of the image

For natural language processing based tasks it is a good practice to clean text data and create a text cleaning pipeline using tools like python nltk (2; 6). The steps in the text cleaning pipeline suitable for our task include:

1. Tokenize the captions into separate words
2. Case Normalization - Convert all words to lowercase
3. Remove punctuations from the words
4. Remove non-alphanumeric characters from the words

For more traditional machine learning tasks additional steps like stemming and lemmatization need to be carried out, but because our model is going to have an embedding layer, it does not make sense to perform additional preprocessing (6).

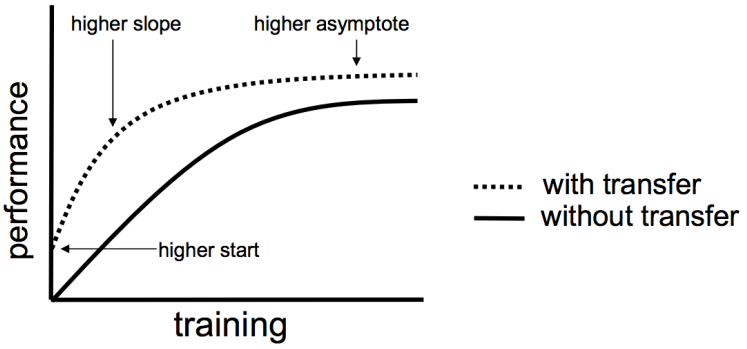

FIGURE 3.1: Performance adavantage of transfer learning

3.4 Model for Image Classification

As previously mentioned in section 2.1, we use CNN generally for image classification. In this task, we strip away the last layer of the CNN model. This is because we are only interested in the high-level features that the CNN learns from the image, and not on the final classification. These features can be fed into the RNN along with part of the corresponding text of the caption. The features and the text together are used to predict the next text in the caption as shown in Figure 2.2.

For the purpose of this task, I initially tried training my own CNN models, but the results were not good. Then, I used Transfer learning, where we re-use a model that has already been developed for a certain task for a related but unidentical task (5). We can use a pre-trained network for recognizing and classifying images and use it to get the high-level features of the images. Transfer learning helps reduce running time as the model does not have to be trained from scratch. Figure 3.1 also suggests how transfer learning can lead to better performance, as the model has a better start as it does not need to be trained from scratch. The rate of increase in performance (or decrease in loss) is also higher and these two factors help the model reach a better final performance score (5).

Various options, like AlexNet (22), VGG networks(46), Resnet (13) and GoogleNet (49) exist for using Transfer Learning for Image Classification. For this project, I have experimented with the following architectures:

- AlexNet
- VGG 16 - a variation of VGG network with 16 layers
- VGG 19 - a variation of VGG network with 19 layers

Some observations after running the experiments:

1. The performance was highest for VGG 19, then VGG 16 and least AlexNet, which is to be expected, considering the fact that deeper the network, better the performance, in general

2. The performance improvement for VGG 19 over VGG 16 was 0.7% on the test dataset. But the training time was considerably larger (23 minutes longer on 25 GB RAM)

Thus, VGG 16 was selected as the final model and it was used it to extract the photo features. Each feature has a dimensionality of **4096**.

3.4.1 Optimization of the VGG-16 model

Running each image through the entire VGG network takes a long time and had scope for optimization. The process was:

```
Assign a unique id to each image in the dataset
Create a dictionary of form {   id   : [..., features,    ]}
For each image
    Run the image through the VGG network except for the last layer
    Extract the features for that image
    Store the image id as the key and the list of features as the value in
   the dictionary
Return the formed dictionary
```

Listing 3.1: Optimization for photo features

Once the dictionary is formed, we can easily look up the corresponding features for an image using the id of the image

Python supports pickle files for easily storing python objects, however, they do not scale well for big data (beyond 1GB) and are often filled with garbage values. So, Hierarchical Data Format (HDF5) file (61) was used for storing the computed features. This optimization process reduced the training time of the final network from 50 minutes to almost 20 minutes.

At the end of this step, **we have computed the high-level features of an image.** An important point to note here is that the choice of stripping away exactly one layer from the model is experimental. As discussed in the prev section 2.1, as we go deeper into the CNN, it learns more complicated features. One can argue that it might be a good idea to explore the results after stripping away the last two layers so that slightly less specific features are taken into account. I tried this process, however, the performance of the model by stripping away 2 layers reduced dramatically and the network became under fitted. That means **by stripping away 2 layers the network could not understand the complexities of the image well enough to associate the features with the captions**.

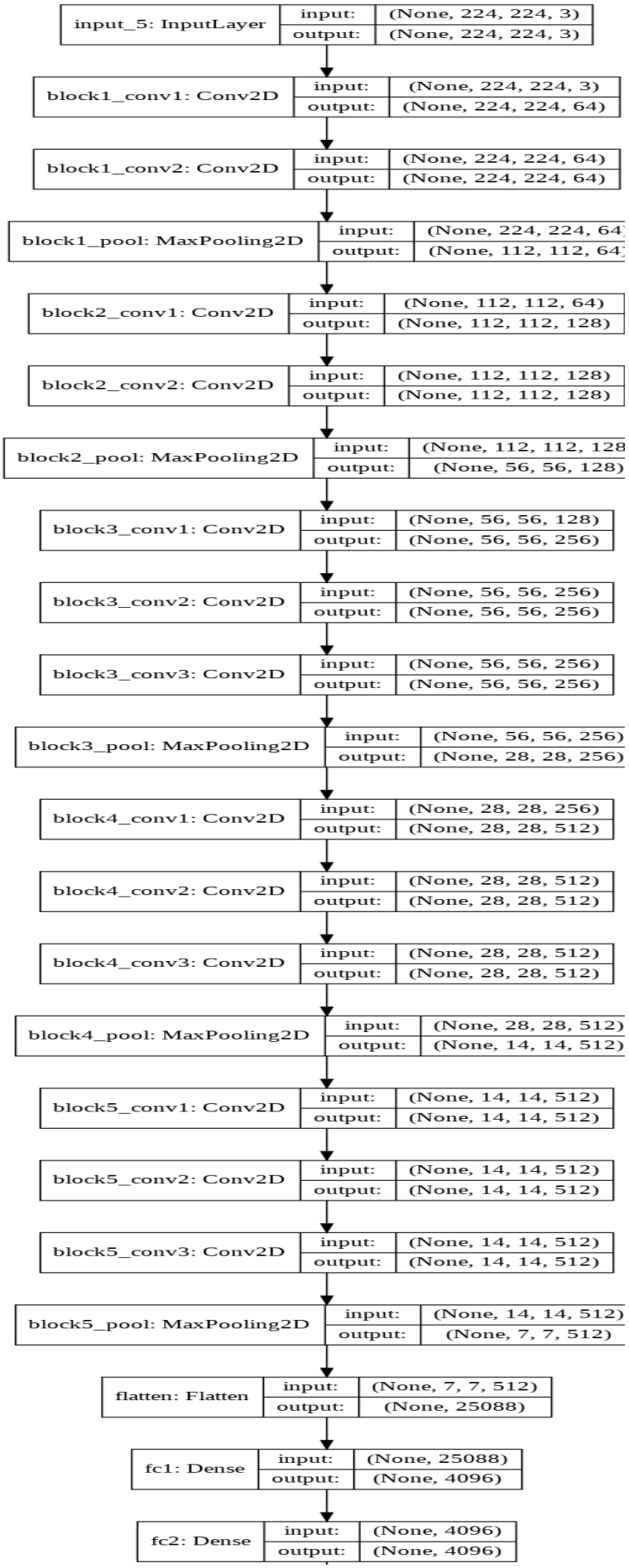

FIGURE 3.2: VGG 16 architecture; last layer removed

3.5 Creating the training set

The CNN model has already identified the high-level features of the image. The next step is to associate these features with the caption. As discussed in section 3.3 we have already run the text preprocessing pipeline on the captions. Also, we have assigned a unique id to each image. So the data for some an image with id_1 is of the form:

```
id_1   : [caption text 1, caption text 2, caption text 3, caption text
 4, caption text 5]
id_2   : [...]
```

Listing 3.2: Data schema for each image id

The next step is to create the dataset that will combine the image features and the captions together which we can then feed into our RNN model. While generating captions we have to set a limit for the model to stop predicting the next word in the caption. We do this by appending two special tokens `startseq` and `endseq` to the beginning and end of each caption respectively. These tokens tell the system when to start and when to stop predicting the sequence of words. This process is visualized in Figure 3.3 and outlined as:

```
Initialize lists X1, X2 and y
For each image
    Lookup the image features learned by the VGG model using the image id
    Lookup the image captions  using the image id
    For index i to length of words in the caption
        Append the photo features in the list X1
        Append the first i words in the caption in the list X2
        Append (i+1)the word in the caption in the list y
        Repeat until we append the   endseq   token in y
Return X1, X2 and y
```

Listing 3.3: Alogorithm to create training dataset

`X1, X2 and y` are now our training lists. The model should receive the pair `[X1, X2]` and predict `y`. `[X1, X2]` are like the photo features and the corresponding tokens from the caption combined and `y` is the next token that the model should learn to predict.The same algorithm is used to create the corresponding test sequences.

Figure 3.3 represents `X2 and y` as words for readability and understanding purpose. However neural networks cannot understand text features. We have to encode these numbers in some form. The paper (31) discusses the benefits of using word embeddings. So we convert the input text into a one-hot vector and then feed them into an Embedding layer, which is built into Keras (8). This layer basically converts these sparse one-hot vectors into a dense vector representation by embedding them into a lower dimension.

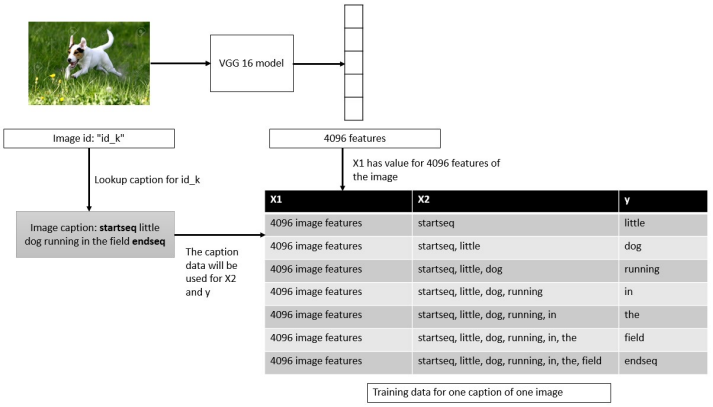

FIGURE 3.3: Process to create training data

Dataset statistics	**Value**
Total images in dataset	8091
Average words per caption	9
Length of training set	6000
Len of testing set	1000
Vocabulary size	8763
Maximum words in a caption	34
Minimum words in a caption	3
Length of training set after creating sequences (X1,X2,y)	306404
Length of test set after creating sequences (X1,X2,y)	50903

TABLE 3.1: Dataset statistics

The choice of how many dimensions to use for embedding is arbitrary, and through experimentation it was found that **300** dimensions gave the best results.

Table 3.1 displays some statistics about the dataset and also after creating the training sequences `X1, X2 and y` and the corresponding test sequences.

3.6 Model for Image Captioning

(65) discussed the approach of using CNNs for extracting higher-level features from an image. Instead of extracting the features from the last fully connected layer, it extracts the features from the last convolutional layer. As mentioned in subsection 3.4.1, I also experimented with this approach of successively looking back into the VGG network so that the RNN network can better understand the features. The network discussed in (65) follows an encoder-decoder based architecture. (50) also follows a similar approach but considers a variety of possible architectures in this space.

As mentioned in (65), the task of automatically generating captions from an image is very similar to the task of machine translation. Encoder-decoder based architectures have

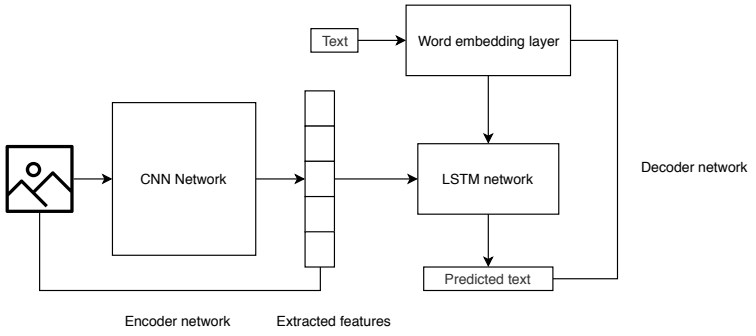

FIGURE 3.4: Modified architecture

achieved excellent results in machine translation (7). Encoder-decoder based models generally consist of an encoder RNN, which maps the input sequence to a vector of fixed length, and the decoder maps this representation to a target sequence. This architecture, when incorporated with attention-based networks like LSTM (16) achieve state-of-the-art results in machine translation tasks. Also, the models are very interpretable and quite simple compared to other complex models for similar tasks.

The choice of LSTM for the model was both inspired by the attention mechanism in-built into the network that proved important for the image captioning task (65). Also, LSTMs have proven to be very successful in machine translation tasks like English to French translation (48) and the model was found to be consistent even for long input sequences. This is to be expected as discussed in section 2.2, as the primary advantage of LSTMs over vanilla RNNs is the addition of the memory sub-network.

Keeping all these factors in mind, we can modify the initial architecture as discussed in section 2.2 and Figure 2.2 to the one depicted in Figure 3.4:

Now that we have fixed the type of architecture and the type of RNN to be used we can focus on how our model will understand the features of the image together with that of the text from the captions. The work of (50) discusses various architectures regarding how the image features are incorporated in the model. The primary difference between the architectures is the input to the LSTM network. These are discussed in details below:

1. **Inject architecture**: In this architecture, the image features and embedded vector of the word sequence are provided as input to the LSTM network. Thus in this type of architecture, the LSTM learns the visual features as well as the textual features together (3).

2. **Merge architecture**: In this, the RNN part of the network only deals with the text. The input to the LSTM is just the embedded vector of the word sequence. The features learned by the LSTM is then merged with the image features and inputted to a fully connected layer.

Parameter	**Experimental values**	**Final value chosen**	**Parameter optimized (Yes/No)**
LSTM layer neurons	128,256,512,1024	512	Yes
Dense layer neurons	256,512,1024	512 and 256	Yes
Dropout rate	0, 0.3, 0.5	0.3	Yes
Embedding dimension	100, 200, 300, 500	300	Yes
Optimizer	Stochastic Gradient Descent, Adam, RMSProp	Adam	Yes
Learning rate	0.01 till 0.5; randomly sampled	0.03	No (RAM constraint)
Number of epochs	10, 20, 50, 75	50	No (RAM constraint)

TABLE 3.2: Model parameters

The experiments by (50) prove that the merge model is better in the following ways:

1. Merge model needs almost half the number of neurons than the inject architecture. This is because the merge model uses RNN for training on only the words and not on the images. From this, we can conclude that the number of learnable parameters in the merge architecture will also be much lesser. Thus the training time will be much shorter. This is an essential factor for our project, as the training time needs to be taken into account for a model that is deployed on the web. (50) suggests that the optimal number of neurons for the LSTM network in the merge model was 128 vs 512 for the inject model.

2. Inject architecture, because of its need for larger LSTM networks needs much more memory to train. For our project, we will be training on 6000 images and each image will have 5 captions. As shown in table 3.1, the length of the training set is 306404. Google collaboratory is used for training the model which provides 25GB of RAM and the merge model is beneficial over the inject model for this purpose. section 1.3 also discusses possible issues regarding RAM and GPU over-utilization, and these risks have been considered

Based on the architectures defined in the paper, experiments were conducted based on a number of parameters as shown in table 3.2. Some parameters like the number of epochs and the learning rate of the optimizer could not be optimized because of the RAM constraint of Google Collaboratory. The final model architecture is shown in Figure 3.5

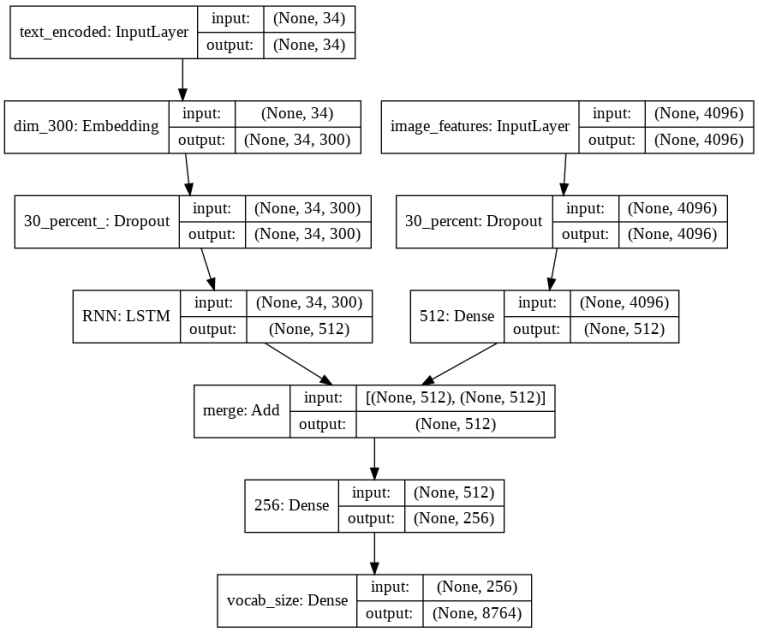

FIGURE 3.5: Final model

3.7 Generating the captions

Now, we have our final model which has been trained on 6000 images and their corresponding captions. We can generate the captions on the test set (1000 images) and evaluate the results. To generate the captions we use the following algorithm:

```
For each image in the test set
    Lookup the image features learned by the VGG model using the image id
    Set initial input sequence as   startseq
    While length of generated sequence is less than or equal to 34 (maximum
     length of a caption)
        One hot encode the input sequence
        Pass the image features and the input sequence into the model
        The output prediction is a vector of length equal to the vocabulary
     size, containing the probability distribution of the words
        Get the index of the maximum element of the vector
        Get the word for the corresponding index
        If the word is   endseq
        Break out of the loop
        Append the word to the input sequence
    The input sequence holds the caption for that particular image
```

LISTING 3.4: Generating the captions

At this stage, we have the captions for all the 1000 images in the test set.

3.8 Evaluating the model

(50) mentions a variety of metrics to evaluate the quality of the generated captions. Initially, the metric Bilingual Evaluation Understudy Score (BLEU) (36) was used for evaluating the generated captions against the real captions in the test dataset. As discussed in (4), BLEU offers some advantages that apply to this project like:

1. Simple to understand
2. Easy to implement - nltk (14) in python has an implementation of BLEU
3. It can be used to compare the performance of our model against the model described in (65) and (50)
4. It correlates highly with human evaluation
5. The score is calculated irrespective of the order of the words
6. A cumulative BLEU score can be calculated based on N-gram (62) matches

The metric ranges from 0 to 1; 1 being a perfect match. We can calculate cumulative BLEU score for N-grams. For example, while considering the BLEU-2 score we see the percentage of matching 2-grams in the real and generated caption. Using this metric, our final model Figure 3.5 has the following scores:

- BLEU-1: 0.535031
- BLEU-2: 0.282928
- BLEU-3: 0.196293
- BLEU-4: 0.091624

3.8.1 Sample results

Some examples of the captions generated by our model are shown in Figure 3.6. It can be seen that there are cases when the model gets the caption correct (green), partially correct (yellow) and completely incorrect (red).

FIGURE 3.6: Few captions generated by the model

Source caption	**Generated caption**	**BLEU-1**	**BLEU-2**	**BLEU-3**	**BLEU-4**
puppies running in the ground	two dogs playing in the field	0.33	0.2	0	0
man riding bicycle in maroon	man in red shirt riding bike	0.5	0	0	0
bird sits in leafless tree	little bird sitting on a branch	0.17	0	0	0
child in black wetsuit is in the waves on surfboard	man in blue is in the water	0.37	0.22	0.13	0

TABLE 3.3: Model parameters

3.8.2 BLEU metrics - The problem

Even though the BLEU metrics for our model look promising a closer inspection revealed a problem.

The BLEU scores between the real and generated captions as shown in table 3.3 are very low although the captions are quite close to the real ones. The reason why BLEU score fails to capture this is because it tries to match every word exactly considering various N-grams. It fails to understand the words in their context and synonyms. Other metrics discussed in (65) and (50) like CIDEr (54), METEOR (24), and ROUGE-L (26) also fail to address this issue.

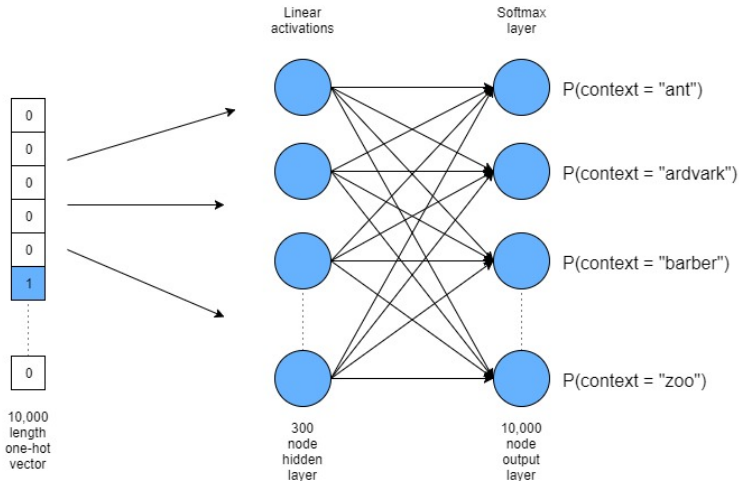

FIGURE 3.7: Working of a basic word embedding model (52)

3.8.3 A Proposed Solution

It is clear from the above discussion that we would require a more representative method of evaluating the generated captions. One possible solution is to use a word embedding technique like word2vec (31). The motivation behind using word embeddings for this project is:

1. When words are trained by deep learning algorithms, they should be represented in a manner which can capture the context in some manner
2. Detecting phrases which have close context is essential not only for evaluating the current model but also for developing the second part of this project as discussed in section 1.2
3. By embedding the words in vector space we can apply metrics like Cosine similarity between them which can give us a better idea about the similarity than BLEU scores or Euclidean similarity. Cosine similarity is insensitive to the relative sizes of the documents (28).

3.8.4 Working of a word embedding model - word2vec

For this project, we use the famous word embedding technique - word2vec (31). Word2vec is a shallow neural network that learns the association between the words (52). From Figure 3.7, we can see that the model takes as input the one-hot encoded form of the words. The size of this input vector is the same as our vocabulary size. Then there is a hidden layer, the size of which determines the vector size in which all the words will be embedded. The final output layer again is the same size as the vocabulary, but usually has a softmax (63) activation function, which outputs a probability distribution of all

possible words in the vocabulary. By training this network on a large corpus of words, it gradually learns to maximize the probability of words which are in close proximity to it. An important assumption that this model makes is that words in close proximity are often similar or contextual. Finally, after the network has learned the associations, we can extract the hidden (embedding) dimension, and words which have similar context will represent similar vector space embeddings.

3.8.5 Word Movers Distance

Now that we have a brief idea about how word2vec works, we can go back to our problem of understanding if the generated captions are similar to the original captions or not. Basically, we have to compare two sentences, and in the field of natural language processing, sentences are often referred to as documents. So we need a metric that can give us a similarity score based on the distances between the documents, incorporating the features of word2vec.

Word Movers Distance (WMD) (23) is such a metric. WMD utilized word2vec embeddings. The words and hence, the documents can be represented in vector space. WMD computes the cumulative distance between two documents in vector space as the minimum distance the cloud of points for one document will need to travel to merge with the cloud of points for the above document. Because it uses word2vec embeddings, similar words will be close together in vector space, and, as a result, similar documents will have less WMD between them. The motivation for using WMD for our use case of identifying the similarity between captions are:

1. Can leverage pre-trained embeddings of the word2vec model
2. Can capture the semantic similarity between documents that other metrics like TF-IDF (23; 37) fail to do
3. The algorithm is hyperparameter free - so a lot of time and memory for hyperparameter optimization can be saved. As discussed in section 1.3, RAM management is a crucial factor
4. The algorithm can be easily implemented using the gensim package in python (39)

3.8.6 Implementing the proposed solution

To implement WMD, we use pre-trained word2vec embeddings from the Google News dataset (12). These are a set of pre-trained vectors, each representing a unique word, which have been extracted from Google news data. There are about 100 billion unique words, and the vector (embedding) size is 300. It is a safe assumption to consider that

Source caption	**Generated caption**	**WMD**	**BLEU-1**	**BLEU-2**	**BLEU-3**	**BLEU-4**
puppies running in the ground	two dogs playing in the field	0.98	0.33	0.2	0	0
man riding bicycle in maroon	man in red shirt riding bike	1.04	0.5	0	0	0
bird sits in leafless tree	little bird sitting on a branch	1.06	0.17	0	0	0
child in black wetsuit is in the waves on surfboard	man in blue is in the water	0.88	0.37	0.22	0.13	0

TABLE 3.4: WMD and BLEU scores for sample captions

all of the words in our dataset is a part of this massive 100 billion words dataset, so we do not have to train our own word2vec model for this. Another important catch while computing WMD is it considers Euclidean distance (60), and not Cosine Similarity (28). So if two documents have different lengths the Euclidean distance will be large, so we normalize the embedded vectors so that they have the same lengths.

The implementation of the WMD distance on our test dataset is as follows:

```
Normalize the word2vec embedded vectors
For each real and generated caption in the test dataset
    Convert all words to lowercase
    Remove all stopwords
    Compute the WMD similarity
    Store this similarity score in a list
Compute the mean of the similarity scores
```

LISTING 3.5: WMD implementation

The average WMD Distance score is **1.17**, which suggests that the generated captions are quite close to the original ones. Through experimentation, we can see that if the captions are not similar the WMD score is generally over **1.25**, and often over **1.5**.

3.8.7 Results

In table 3.3, we had observed few samples of real and generated captions for which the N-gram BLEU scores were very less, even though the captions were quite similar. We compute the WMD Distance metric between these captions using pre-trained Google News word embeddings (12), and the results are summarized in table 3.4. It is clear from the above results that applying WMD metric to the captions generate better results.

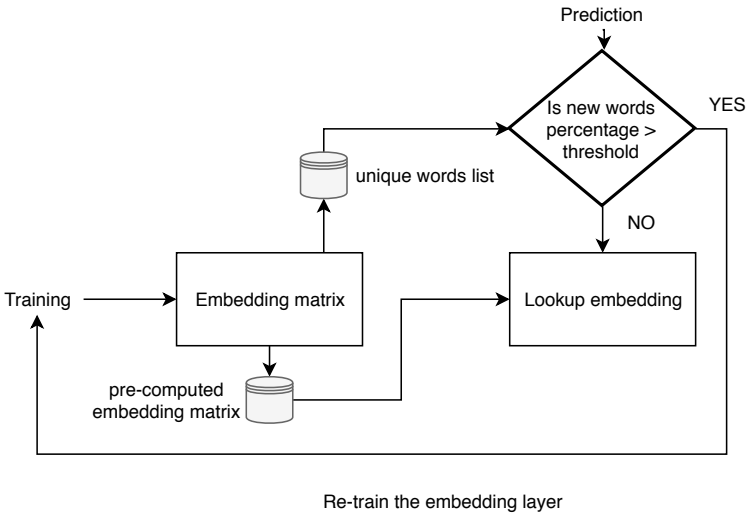

FIGURE 3.8: Optimizing word embeddings

3.9 Some optimizations for deploying

As discussed in subsection 3.4.1, we optimized the CNN part of the model by pre-computing and storing the image features. When we deploy the model to the web, we need to consider the running time of the model. For that, we should store all the variables, data structures and weights of the models which have been trained and optimized on disk so that while predicting, the application can just read from these files and run the data through the model to get the predictions. We should not have to re-create the datasets or re-train the model every time. Most importantly, the creation of the training dataset, as discussed in section 3.5 takes a long time (30 mins on 25GB RAM) and should be stored on disk.

3.9.1 Optimizing word embeddings

As discussed in section 3.6, our model has an embedding layer of fixed dimensionality which learns dense vector space embeddings of words. Once trained on the whole dataset, **8763** words are learned by the model. This is a significant number and it does not make sense to precompute the embeddings every time. So, the embedding matrix once learned can be stored as a numpy array of dimensionality `(vocabulary_size, embedding_size)` and then be stored on disk as a pickle, which the model can refer to while training. This significantly reduced the training time of the model (by almost 10 minutes on a batch of 1000 new images for 20 epochs). Also, to make this approach work, we keep a list of words in our vocabulary. If newer data comes in and the percentage of words that are out-of-vocabulary is beyond a particular threshold, we re-train the embedding. This process is summarized in Figure 3.8.

3.9.2 Using model checkpoints

Also, because the model will be deployed on the cloud, we may need to re-train the model as we receive new data. So we should always ensure that we are using the best model for the predictions. An easy way to do this is to monitor the loss via callbacks. Callbacks allow us to monitor important statistics of a model like loss and accuracy while training [29]. Using callbacks we can create checkpoints of the model The way we do this is:

```
Set the format of the model file as: model-ep{epoch:03d}-loss{loss:.3f}-
    val_loss{val_loss:.3f}.h5
Create a new ModelCheckpoint instance
Set the appropriate parameters:
    monitor =   val_loss
    save_best_only = True: Only save the model with the lowest validation
    score
    mode =   min   : overwrite the current file if the model has the
    minimum validation loss
Set the checkpoint as a callback while training the model
```

Listing 3.6: Model checkpoint implementation

Thus, only the model with the lowest validation loss will be stored on disk. If the current model has a lower validation loss, the model on disk will be replaced by the current one.

3.9.3 File management

Table 3.5 shows a possible configuration of the files that have to be maintained on disk, their types and refresh rates. Generally files of sizes less than 1GB can be maintained as pickle files. Files larger than that have to be stored in HDF (HD5) format.

3.10 Extensions

Now that we have developed a system for automatically captioning images and we have also applied metrics to test the quality of the generated captions keeping in mind the complexities of natural language. Additionally, we explore some additional extended features that we can provide users to improve their experiences on the web.

3.10.1 Object detection

Image captions provide a visually impaired user with an overview of what the image is trying to convey. However, what we noticed was that the captions generated were often vague, like children playing in the park instead of three children playing with football

Filename	File type	File description	Refresh
X1	Hierarchical Data Format (HD5)	Training data - photo features	When the number of training data points increase significantly and the newer data points improve the performance of the model
X2	Hierarchical Data Format (HD5)	Training data - the sequence of captions to be fed into the model	When the number of training data points increase significantly and the newer data points improve the performance of the model
y	Hierarchical Data Format (HD5) or Python pickle	Training data - the token that the model should predict	When the number of training data points increase significantly and the newer data points improve the performance of the model
features	Hierarchical Data Format (HD5) or Python pickle	Features for the images	When the number of training data points increase significantly and the newer data points improve the performance of the model
embedding_matrix	Hierarchical Data Format (HD5) or Python pickle	Embedding vectors for each word in the vocabulary	Whenever the number of words out of vocabulary crosses a particular threshold
best_model	Hierarchical Data Format (HD5)	The model with the lowest validation loss	Whenever a model achieves a lesser validation loss

Table 3.5: File management

in the park. One way to do this would be to incorporate an attention mechanism (65) in our model. However, this would not solve the problem perfectly as the results in (65) and (50) are quite comparable.

A solution is to use object detection. As mentioned, image captioning provides a general overview of what the image is trying to convey. The motivation behind this is that to get a detailed understanding of what is going on in the image, users can toggle through the objects in the image and the system will read out what that object is.

There are many object detection libraries and APIs available, but we use Microsoft Azure Vision API (30) for this purpose, due to the following reasons:

1. The API returns the objects detected as well as the coordinates of the bounding boxes
2. The API returns a confidence level of the detection

3. The API returns parent objects for any detected object. For e.g bicycle helmet - helmet - headwear. bicycle helmet is the detected object here.

4. The API returns the data in JSON (JavaScript Object Notation) format which is easy to interpret using JavaScript and Python

A sample response from the API, when the model has detected the object **dog** is shown below:

```
{
    "url": url for the image,
    "response": {
        "objects": [{
          "rectangle": {
            "x": 154,
            "y": 23,
            "w": 102,
            "h": 122
          },
          "object": "dog",
          "confidence": 0.845,
          "parent": {
            "object": "mammal",
            "confidence": 0.849,
            "parent": {
              "object": "animal",
              "confidence": 0.906
            }
          }
        }],
        "requestId": unique request id,
        "metadata": {
          "width": 356,
          "height": 277,
          "format": "Png"
        }
      }
},
```

Listing 3.7: Sample response from Azure Vision API

Some of the features that this application should have are the following:

1. The image should have a generic caption which will be the output of the image captioning model

2. If the user wants, they can explore more. On the click of a button, the object detection API should be executed

3. The bounding boxes for each object in the image should be drawn

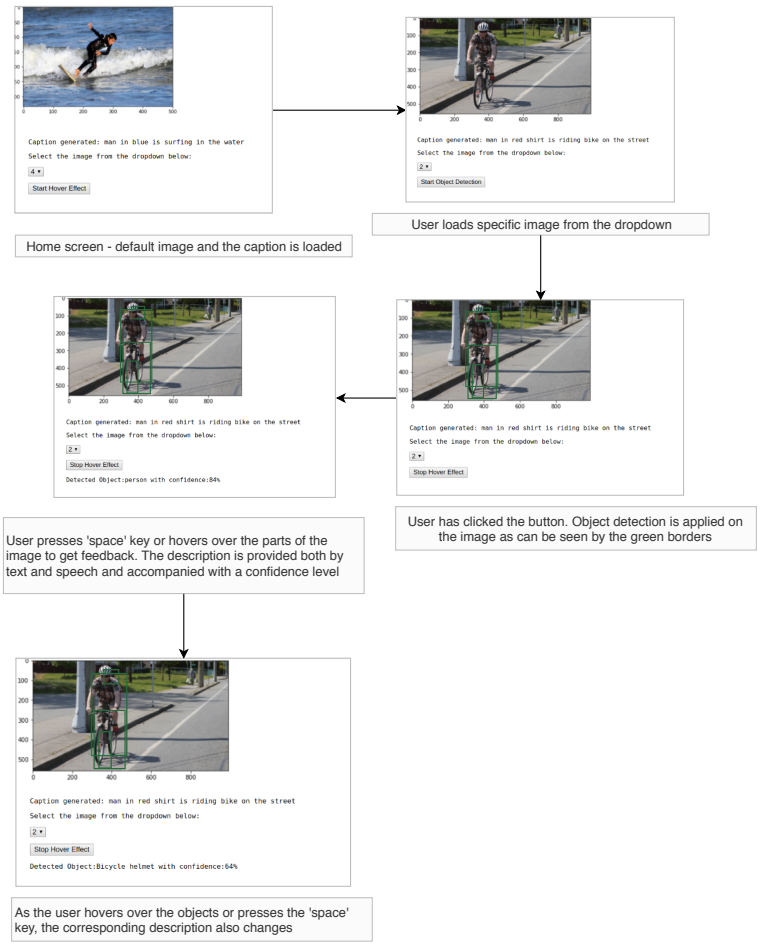

FIGURE 3.9: Object Detection App - User interaction

4. User can hover over these objects and the information should be provided via text and speech

5. User can also toggle through the objects by pressing a specific key (completely blind users will not know where to hover on the image)

Keeping these features in mind, a demonstration application was built. The link to this application is: `https://codepen.io/shaunak1105/full/dybZEXa`

Figure 3.9 shows how users can interact with the app. The information about the objects detected is provided both by text and speech. The bounding boxes are also overlaid on the image. Users can choose to hover over the objects or toggle through them by pressing the `space` key.

3.10.2 Relevance of image on a web page

Often, we come across web sites which are cluttered with images. These images may be present for the purpose of advertisements and they do not convey any real meaning

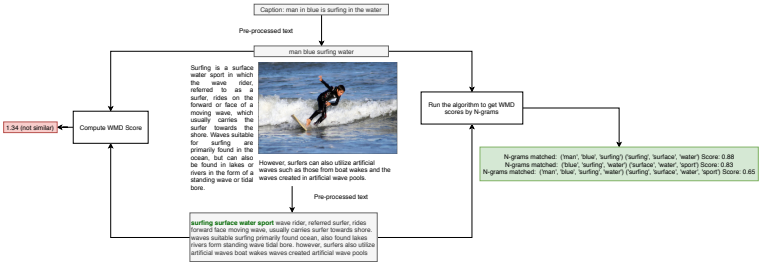

FIGURE 3.10: Visualizing the algorithm for Relevance of image in a web page

to the topic being discussed. These images are thus confusing and distracting and for someone who is using screen readers, the experience will be worse.

The motivation behind this extension is to detect these irrelevant images from the text surrounding it and automatically flag these images so that they can be ignored by the screen readers.

We have already built an image captioning system and evaluated metrics for testing caption-to-caption similarity using WMD. To extract the text surrounding the image, we can consider a window of `w` words around the image tag. Extracting such a data will be discussed in detail in subsection 4.3.2. So we have the generated caption, the words surrounding the image and we want to apply WMD to detect of the text and image caption are in context to each other or not.

We initially pre-process both the caption text and the surrounding text as in section 3.3. Then we computed the WMD score. however, we were not getting proper results (the scores were always higher than 1.25).

The caption text does not have a fixed length of words. So we cannot directly consider it as a document and compute WMD between the texts. Additionally, only a part of the surrounding text might be discussing the caption. So if we simply compute the WMD between the caption and surrounding text, it will return a high value, but that does not mean that they are not similar.

However, at least one part of the surrounding text must be discussing the caption. We can detect this by splitting both the surrounding text and the caption into N-grams (62). For example, if the caption has 5 words, we can consider 2-gram sequences, 3-gram sequences, 4-gram sequences, and 5-gram sequences between the caption and the surrounding text and then compute WMD similarity. The intuition is that at least one of these N-grams between the caption and the surrounding text should have a close match i.e have less WMD score.

The algorithm to do this is described below:

This process is visualized in Figure 3.10. It is clear for this scenario, simply computing the WMD score between the caption and surrounding text resulted in a high score of

```
Preprocess both the caption text and surrounding text
Compute length of caption
For i in range of 2 to length of caption:
    Create i-grams of caption text
    Create i-grams of surrounding text
    For each such i-gram pair
        Compute WMD between caption i-gram and surrounding i-gram
            If WMD < 1.15
                The caption is similar
                    Return true
            Else
                continue
Return false - the caption is not similar
```

LISTING 3.8: Algorithm for Relevance of image in a web page

1.34. However using the algorithm discussed, we get a much lower score and we can also visually see which parts of the text received a close match (shown in green in Figure 3.10). By removal of stopwords, we have ensured that common words are not taken into the formation of N-grams.

Chapter 4

Contextual link text detection

4.1 The problem

As previously discussed in section 3.1, there exists a set of guidelines called Web Content Accessibility Guidelines (WCAG), which aims to create and maintain a single set of guidelines and recommendations for individuals, organizations, and governments internationally to follow to make web content more accessible and inclusive, especially for people with disabilities (58; 64).

Guideline G91 (19) of the WCAG focuses on hyperlinks, specifically providing link text that describes the purpose of the link. The aim of this guideline is, that from the link text, users should get an idea about what the link is talking about. This project deals with a modification of this particular problem.

People with reading disabilities, like dyslexia, cannot process large amounts of texts (32). Often, on many websites, we find hyperlinks occurring randomly, unrelated to the current topic being discussed. My aim in this project is to find out if the hyperlink text, the text surrounding the hyperlink and the text (and title) of the link it points to are in close context or not. This will help detect if the hyperlink is contextual and relevant to the topic being discussed.

4.2 Proposed solution

The problem can be tackled in multiple approaches using Natural Language Processing and Deep Learning techniques. One way to see this is as a topic modeling task; we detect the topic in the source link, the target link, and the hyperlink text and try to match if they are similar or not. However, having applied word embedding techniques

FIGURE 4.1: Links to Wikipedia as entity labels (47)

like word2vec (31) and document-based similarity metrics like WMD (23) in the subsection 3.8.5, I wanted to explore how these solutions could be extended for the current problem.

As we discussed in subsection 3.8.4, how word2vec (31) works is by learning the associations between words that occur together in a large text corpus. It assumes that words that occur together in this corpus have a close context, i.e. are similar. For our image captioning task, we used a pre-trained word embedding of Google News (12) because detecting if two captions were similar or not was a generic task, for which pre-trained embedding was ideal. For the current task, it makes sense to consider a more specialized dataset, that can capture the associations between source and target hyperlinks. So, using this specialized text corpus, the aim is to train a specialized word2vec model for this task and evaluate the results. By comparing the results with those obtained using pre-trained embeddings, we can also understand the need for using specialized datasets for natural language processing tasks in this domain.

The overall process of the proposed solution is thus:

1. Explore a suitable dataset that captures hyperlink information
2. Build a text corpus
3. Build and train a word2vec model on this corpus
4. Evaluate the results
5. Compare the results with those obtained using pre-trained word embeddings

4.3 Building the dataset

4.3.1 Initial dataset

The Wikilinks dataset (47) provides the URLs of web pages and each URL of a web page has the following:

1. The link texts of all URLs in that page
2. The target link to which the particular URL points to

This is shown in Figure 4.1

The number of URLs in total in this dataset is **10,893,248**, the number of link texts is **40,323,863** and the number of unique target links is **2,933,659** (many links point to the same target). An important factor to note here is that all the target links are Wikipedia links, so the model we develop will have this bias (18). The data schema looks like:

```
URL url1
MENTION mention_1 target_link_1
MENTION mention_2 target_link_2

MENTION mention_k target_link_k


URL url2
MENTION mention_1 target_link_1
MENTION mention_2 target_link_2

MENTION mention_m target_link_m
```

Listing 4.1: Data schema

As we can see each URL has a number of mentions (links and link texts) and a number of target links. The first URL has `k` mentions and target links, while the second one has `m`. From this initial dataset, the plan is to extend it to build a text corpus on which we can train a deep learning model.

4.3.2 Extending the dataset

The idea is to consider a sliding window of size `w` of words around the source link and to extract the words in this window. For the target link, we extract the title and the first `w` words. This will enable us to build a corpus of link texts and the corresponding target link texts. The process is summarized in Figure 4.2

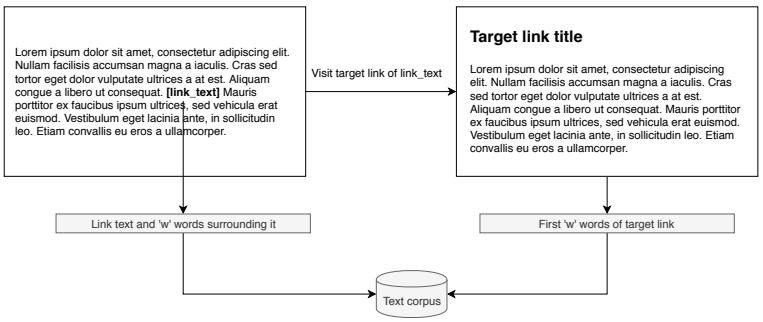

FIGURE 4.2: Process to extend the dataset

4.3.2.1 Read the file for valid URLs

To read the valid URLs from file we do the following:

```
For each URL
    Check if the URL starts with    http   . If yes, go to next step. Else,
     consider URL invalid
    Match the URL with the regular expression: (http|https)://\w.*
    Check if the URL ends with    .pdf   or    .doc   or    .docx   - We
     consider these URLs invalid
    Remove the invalid URLs
Return filtered list of URLs
```

LISTING 4.2: Red the file for valid URLs

The regular expression will match `http` or `https` and then look for a word character `[a-zA-Z0-9_]`. This is because URLs starting with punctuations are invalid.

4.3.2.2 Create a list of all link texts, source URLs and target URLs

Now that we have detected all the valid URLs in the file, we want to create a single consolidated list of the link text source link and target link. This list has the form:

```
[
[link_text1, source_link1, target_link1],
[link_text2, source_link2, target_link2],
...
]
```

LISTING 4.3: List of link texts, source links and target links

We use the following process to create this list:

```
Create an empty list
For each line in the file
    If line starts with   URL
    Extract the source link
    While the next line starts with  MENTION
        Extract the link text
        Extract the target link
        Append [link_text, source_link, target_link] to the list
    Else, proceed with the next line
The list contains the required data
```

LISTING 4.4: Process to organize source and target data

```
Initialize a list final_corpus = []
For i = 0 to length of source_dataset
    Initialize a list corpus = []
    Scrape data for ith source_link
    Get the link from the scraped data which points to the ith target_link
    Get the parent tag of the link
    For words in the parent tag
        Set a counter = 0
        Read words until we find the ith link_text and increase counter
    word by word
        Append the words to the corpus list
        If link_text is found and counter > window_size break out of the
    loop
    Scrape the data for the ith target link
    Get the heading (h1 tag with id: firstHeading) and store it in the
    corpus list
    Get all the paragraphs of the page
    For each paragraph
        Set a counter = 0
        For words in the paragraph
            Append the words to the corpus list and increment counter
            If counter > window_size break out of the loop
        If counter < window_size
            Continue with the next paragraph
    Append corpus to final_corpus
The list: final_corpus contains all the required data
```

LISTING 4.5: Algorithm to extend the dataset

4.3.2.3 Scrape the data of the source URL and target URL using the sliding window

We have already created a list of all link texts, source URLs and target URLs. Let us name this list as `source_dataset`. Now we have to create a text corpus by extracting a window of words around the source URL and the target URL. We use Beautiful-Soup Python library (40) for scraping purpose. Also, we maintain 3 separate lists: `link_texts`, `source_links`, `target_links` for utility. The algorithm is described in Listing 4.5.

Before adding the words to the corpus, we also run a text preprocessing pipeline on it to remove any data discrepancies. The pipeline is exactly similar to the one we used in

```
[
[words for source_link1, link_text1, words for target_link1],
[words for source_link2, link_text2, words for target_link2],
...
]
```

Listing 4.6: Final corpus data schema

Dataset statistics	Value
Number of words	3806212
URLs scraped	21058
Vocabulary size	38435

Table 4.1: Statistics of the extended dataset

section 3.3

All the words for each `source_link`, `link_text` and `target_link` combination is now stored in the list `final_corpus`. The data in `final_corpus` schema is shown in Listing 4.6.

This schema is consistent with the one described in Figure 4.2.

4.3.2.4 Store the data in the disk

The above section discusses the method of scraping out the relevant data. But as already mentioned, the number of URLs to scrape is massive and the code cannot simply be run in a single iteration. This is why it was necessary to divide the work into batches and use multiple computing instances to run the code. Google collaboratory (11) allows us to create multiple instances of our code and run them parallelly on multiple CPUs. Another advantage of using this software is that it allows us to mount Google Drive into the development environment so that we can easily save and load the datasets generated without having to clunk our local systems.

To parallelize the process, we set a batch size of **1000**, i.e when 1000 URLs are scraped, we dump the data into a new pickle file in Google Drive. Also, we maintain 2 new variables `lower_limit` and `upper_limit` to keep track of how much data has been scraped. The filenames that are dumped to drive follow the syntax: `corpus_lower_limit_to_upper_limit.pickle`. We can use pickles here instead of HD5 files (61) as we are only considering a batch size of 1000 and the file sizes are not generally greater than 500MB. Some statistics of the extended dataset are in table 4.1.

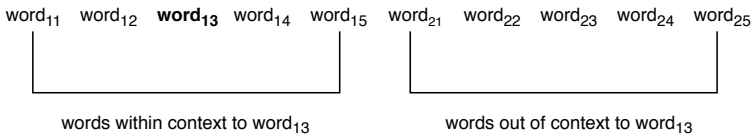

FIGURE 4.3: Words within and out of context example

4.4 Building the model

4.4.1 Negative sampling

Now that we have a specialized corpus for our problem, we can create our model. We consider a variation of the word2vec architecture (31) called negative sampling. The idea here is to sample words randomly from the dataset. These words can either lie within a specified window of words (context word) or not (negative samples). Consider the Figure 4.3. In this figure `Word_ij represents Word number j for link i`

If we consider `word_13` as the target and a window size of 2 (2 words to the left and right), the words within the context and the words out of context will be as shown above. So, if we sample `(word_11, word_13)` they are a contextual pair, so they should be similar. If we sample `(word_11 and word_23)`, they are negative samples and are not similar.

4.4.2 Creating the training dataset

There is a key difference in our corpus and the corpus like Google News (12), which are usually used to train word embedding networks. Usually, the corpus contains a list of sentences, and the sentences usually are considered together as one single corpus. However, the scheme of our final corpus as discussed in subsubsection 4.3.2.3 is:

```
[
[words for source_link1, link_text1, words for target_link1],
[words for source_link2, link_text2, words for target_link2],
...
]
```

LISTING 4.7: Final corpus data schema

We can think of the above schema in a simplified manner as:

```
[
[words associated with link1],
[words associated with link2],
...
]
```

LISTING 4.8: Final corpus data schema simplified

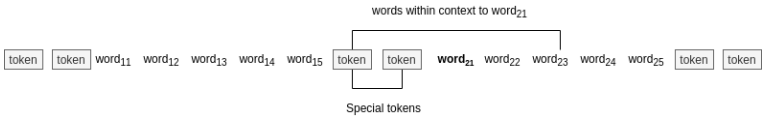

FIGURE 4.4: Words within and out of context example with tokens

Words that are associated with link1 should not be associated with link2. In Figure 4.3, if we consider a fixed window size of 2, `word_15` will be associated with `word_21`, which is actually wrong. `word_15` is the last word for the first link, while `word_21` is the first word of the second link. These links have no relationship between them, and so these words should not be associated together.

One solution is adding special tokens (the word: `specialtoken`) between these links. If the window size is `w`, we add `w` number of `specialtoken` tokens between the last word of one link and the first word of the next link. Refer to Figure 4.4.

This system is for handling the edge words corresponding to each link. Now, `word_15` will no longer be associated with `word_21`. This is a simple solution, but with the following drawbacks:

1. It increases the space complexity
2. The words might learn association with the `specialtoken` word, which does not make sense

The space complexity is not really a concerning factor, as we are saving the pickle files remotely, and not on our local systems. The second problem, however, deserves closer attention. One solution is to sample words based on the frequency by which they occur in the corpus. Keras (8) offers a method by which we can generate a word rank-based probabilistic sampling table. These probabilities are inversely related to the frequency of word; so words with higher frequency will be sampled less. So the model will sample `specialtoken` fewer number of times and this would avoid words being frequently linked to this token.

Now that we have solved this challenge, we can proceed with creating the final training set of our model. Given the dataset, we sample target words according to the sampling table. Also for each target word, we sample context words or out-of-context words using the sampling table. Thus, we create a list of word pairs and each word pair is in context (label of 1) or out-of-context (label of 0). Only two words within the window size `w` will be considered in context. Some examples considering Figure 4.4 are shown in table 4.2

Word pair (X)	**Label (y)**
word_11, word_12	1
word_13, word_22	0
word_24, word_22	1
word_22, word_11	0

TABLE 4.2: Sample word pairs

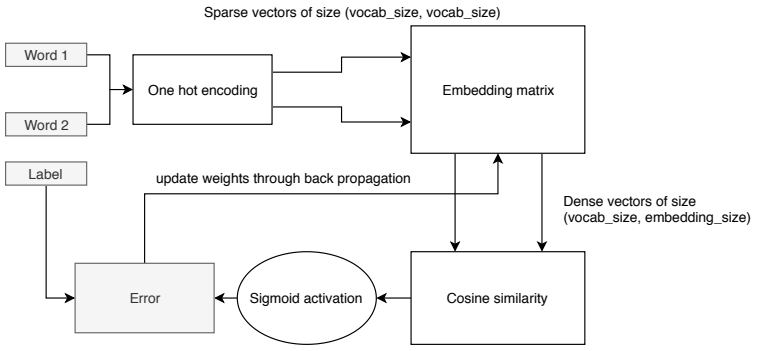

FIGURE 4.5: Negative Sampling Model Architecture

4.4.3 The model architecture

We have created our training set as pairs of words and their corresponding labels. The next step is to create the model architecture.

The architecture of the model (51) is shown in Figure 4.5

The model receives a pair of words as input. This pair has an associated label with the (1 or 0). These words are one-hot encoded into a sparse vector of size equal to the vocabulary size of our corpus. These vectors then pass to the Embedding layer which is basically a matrix of dimensions `(vocabulary_size, embedding_size)`. The embedding size considered is 256. The embedding size is generally optimized through experimentation. Using the embedding matrix we look up the corresponding dense 256 length dense vector representations of the target and context (or out-of-context) word.

At the next step, we compute the cosine similarity (28) between these two vectors. If the two vectors are similar, they will have a higher similarity score. We then use the sigmoid activation function to smoothen out the similarity score between 0 and 1 (59).

After obtaining the normalized cosine similarity score (between 0 and 1) we compute the cross-entropy loss (10). This loss is then used in backpropagation and tuning the weights of the embedding matrix.

We train the model like this for many epochs and monitor the cross-entropy loss. After the loss has converged we stop training the model. At this point, we can extract the embedding matrix out and save it on the disk. The embedding matrix contains the

Word	Top 5 most similar words
happy	glad, alive, joy, funny, smile
india	indian, India, jammu, maharashtra, hindustan
mumbai	kolkata, hyderabad, madras, chennai, pune
england	wales, essex, scotland, yorkshire, london
obama	barack, romney, sinclair, President Obama, Barack Obama
football	basketball, uefa, coach, hockey, soccer
google	youtube, blogs, doubleclick, facebook, internet

TABLE 4.3: Some results of the learned embeddings

learned vector representations of all words in our vocabulary. These are our learned embeddings.

4.4.4 Learned embeddings results

Using these embeddings we can plot the words in space using dimensionality reduction techniques like t-SNE (53) and also find the most similar words to a given word using Cosine Similarity metric (28). The table 4.3 shows some results of most similar words to some random words, and the words which the model has learned to be in context seem correct.

4.5 Evaluating the model

As discussed previously, the output of the model is a matrix of learned vector embeddings. Using these embeddings, we can now detect the similarity between the source link data, source link text, and target hyperlinks and determine if they are contextual or not.

We have already discussed in subsection 3.8.5, that Word Movers Distance (WMD) (23) is a metric that can give us a similarity score based on the distances between the documents, incorporating the features of word2vec (31).

We assume for the remaining part of the document that the source link words be referred to as **source text**, the words in the hyperlink be referred to as **source link text** and the target link words be referred to as **target text**

4.5.1 Evaluating similarity between the source text and target text

```
Normalize the word2vec embedded vectors
For each source and target text in the test dataset
    Convert all words to lowercase
    Remove all stopwords
```

Corpus used	**Corresponding source and target (Yes/No)**	**Mean of WMD**	**Median of WMD**
Generated corpus	Yes	1.08	1.10
Google News	Yes	1.11	1.13
Generated corpus	No	1.23	1.24
Google News	No	1.22	1.23

TABLE 4.4: Results for similarity between source text and target text

```
    Compute the WMD similarity
    Store this similarity score in a list
Compute the mean of the similarity scores
```

LISTING 4.9: Evaluating similarity between the source text and target text

This algorithm is exactly the same as the one discussed in subsection 3.8.6, and thus promotes a lot of code re-usability. This algorithm can be used because the source and target text will have comparable lengths and can be considered as two documents and Word Movers Distance can thus be directly computed on them.

Using this algorithm we can figure out, that given a source text, does it make sense for that text to contain a link that points to the target text or not. This is not directly linked to the WCAG guideline G91 (19), but if the source text and target text are not related, then the hyperlink should not ideally exist and can be ignored by screen readers. That way, information clutter can be reduced and a much smoother user experience can be provided.

First, we run the model to find WMD between the source text and the corresponding target text. The WMD score should ideally be low in this case. Then, we also run the model to find WMD between the source text but not the corresponding target text, but some other randomly sampled target texts. So, we would expect the WMD score to be higher. We perform the above process for the model trained on our text corpus and also on the word2vec model trained on the Google news dataset (12). The results are shown in table 4.4

We have considered both the mean and median scores as, for some documents the WMD scores may be too high, and we do not want these outliers to have a huge impact on the analysis. Medians are less sensitive to outliers.

An important fact to note here is that we are dealing with an **unsupervised learning task** and we do not have a fixed, established metric to compare the two approaches used. But we can definitely conclude from the above table that the results on the generated corpus look promising. There exists a significant difference in both the mean and median scores for our method when we are evaluating for corresponding source and target texts. This suggests that the model is good at detecting the similar source and target texts for

Corpus used	Corresponding source and target (Yes/No)	Mean of WMD	Median of WMD
Generated corpus	Yes	1.21	1.25
Google News	Yes	1.27	1.32
Generated corpus	No	1.43	1.42
Google News	No	1.45	1.42

TABLE 4.5: Results for similarity between source link text and source text

hyperlinks. However when we feed the model random target texts for source texts the average WMD scores increase, as expected, but there is not much difference from that of the model pre-trained on Google News dataset.

4.5.2 Evaluating similarity between the source link text and target text/source text

To evaluate the similarity between the source link text and target text/source text, we use the same algorithm as discussed in the previous section. However, since the link texts are generally shorter than the source or target texts, we expect the WMD score to be higher.

From the table 4.5, we can observe that the average WMD scores are higher, as expected. There exists a significant difference in both the mean and median scores for our method when we are evaluating for corresponding link texts and source texts. This suggests that the model is good at detecting the similar link texts and source texts. However when we feed the model random target texts for source texts the average WMD scores increase, as expected, but the pre-trained model on Google News dataset performs slightly better.

Chapter 5

Conclusion

5.1 Reflection

For the Automatic Image Captioning task, we have created a model for automatically captioning images based on research and experimentation. Additionally, word embedding based similarity metrics have been applied to evaluate the performance of the model. Through this, we have tried to capture not only word-to-word matches but context and synonyms in the real and generated captions. This metric has been extended and a method has been developed to detect image-text-similarity in web pages. Extensions like object detection and how it can be applied in web accessibility domain has also been addressed. Through this project, a wide variety of AI techniques have been explored to help improve the experience on the web for the disabled.

For the Contextual hyperlink detection task, we created our own corpus and the performance of a word2vec model on that corpus was compared against that of pre-trained embeddings. The results for the model created on the generated dataset look promising. An important fact to note is that the pre-trained model we compare it to has been trained on a much larger corpus. As we extend our dataset more and more, it should capture further complexities and the performance should increase. However, while the models have been compared using WMD score, we cannot say that our model is better than the pre-trained one, because this is an unsupervised task, and there are no true labels in the dataset for us to compare with. But, one can definitely see that there is future scope of applying DL and NLP techniques to this problem, and also there is a scope of developing further specialized datasets for this purpose.

AI and web accessibility has largely been an unrelated field. With this project, I sincerely hope to motivate others to apply these technologies in this field, and help people experience the web better.

5.2 Further Research

One of the motivations for doing this project has been to set a good foundation of applying certain AI techniques in the domain of web accessibility, so that it can encourage future work. Certain possible modifications and extensions to this project are discussed below:

1. The Automatic Image Captioning module can be made to output a confidence score for the generated caption for better user experience
2. Like WMD, other methods to compute document similarity like Doc2Vec, which is an extension of word2vec (31) can be considered and compared with the current system
3. All the techniques discussed can be integrated together into a complete web solution

Bibliography

[1] Y. Bengio, P. Simard, and P. Frasconi, *Learning Long-Term Dependencies with Gradient Descent is Difficult*, IEEE Transactions on Neural Networks, 5 (1994), pp. 157–166.

[2] S. Bird, L. Edward, and K. Ewan, *Natural Language Processing with Python*, O'Reilly Media Inc., 2009.

[3] J. Brownlee, *Caption generation with the inject and merge encoder-decoder models*, 2017.

[4] ——, *A gentle introduction to calculating the bleu score for text in python*, 2017.

[5] ——, *A gentle introduction to transfer learning for deep learning*, 2017.

[6] ——, *How to clean text for machine learning with python*, 2017.

[7] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using RNN encoder-decoder for statistical machine translation*, EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference, (2014), pp. 1724–1734.

[8] F. Chollet et al., *Keras*. `https://keras.io`, 2015.

[9] V. Dumbrav, *Using Probability Impact Matrix in Analysis and Risk Assessment Projects*, tech. rep., 2013.

[10] B. Fortuner, M. Viana, and B. Kowshik, *Loss Functions ML Cheatsheet documentation*, 2018.

[11] Google, *Welcome to Colaboratory!*

[12] Google Code Archive, *word2vec*, 2013.

[13] K. He, X. Zhang, S. Ren, and J. Sun, *Deep Residual Learning for Image Recognition*, (2015).

[14] R. Hermawan, *Natural language processing with python*, vol. 1, O'Reilly Media Inc, 2011.

[15] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, *Improving neural networks by preventing co-adaptation of feature detectors*, tech. rep., 2012.

[16] S. Hochreiter and J. Urgen Schmidhuber, *Lstm*, Neural Computation, 9 (1997), pp. 1735–1780.

[17] M. Hodosh, P. Young, J. Hockenmaier, Micah Hodosh, Peter Young, Julia Hockenmaier, M. Hodosh, P. Young, and J. Hockenmaier, *Framing Image Description as a Ranking Task : Data , Models and Evaluation Metrics (Extended Abstract)* , Tech. Rep. Ijcai, 2015.

[18] IBM, *AI and bias - IBM Research*, 2018.

[19] I. Information and R. Techniques, *G91: Providing link text that describes the purpose of a link*, 2018.

[20] IWorld Internet Users Statistics and 2016 World Population Stats, *World Internet Users Statistics and 2016 World Population Stats*, 2016.

[21] B. KAMGAR-PARSI and B. KAMGAR-PARSI, *Hopfield Model and Optimization Problems*, Neural Networks for Perception, (1992), pp. 94–110.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, *ImageNet classification with deep convolutional neural networks*, tech. rep., 2012.

[23] M. J. Kusner, Y. Sun, N. I. Kolkin, and K. Q. Weinberger, *From word embeddings to document distances*, tech. rep., 2015.

[24] A. Lavie and A. Agarwal, *METEOR: An Automatic Metric for MT Evaluation with Improved Correlation with Human Judgments*, tech. rep., 2007.

[25] L.-J. Li, K. Li, F. F. Li, J. Deng, W. Dong, R. Socher, and L. Fei-Fei, *ImageNet: a Large-Scale Hierarchical Image Database Shrimp Project View project hybrid intrusion detction systems View project ImageNet: A Large-Scale Hierarchical Image Database*, in 2009 IEEE Conference on Computer Vision and Pattern Recognition, IEEE, jun 2009, pp. 248–255.

[26] C. Y. Lin, *Rouge: A package for automatic evaluation of summaries*, Tech. Rep. 1, 2004.

[27] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, *Microsoft COCO: Common objects in context*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 8693 LNCS (2014), pp. 740–755.

[28] Machine Learning Plus, *Cosine similarity understanding the math and how it works (with python codes)*, 2019.

[29] S. Makridakis, *The forthcoming Artificial Intelligence (AI) revolution: Its impact on society and firms*, Futures, 90 (2017), pp. 46–60.

[30] Microsoft Azure Documentation, *Detect common objects in images*, 2019.

[31] T. Mikolov, K. Chen, G. Corrado, and J. Dean, *Efficient Estimation of Word Representations in Vector Space*, (2013).

[32] National Institute of Neurological Disorders and Stroke, *Dyslexia information page*, 2019.

[33] J. Oldman, *10 Free Screen Readers For Blind Or Visually Impaired Users*, 2012.

[34] P. Oram, *WordNet: An electronic lexical database. Christiane Fellbaum (Ed.). Cambridge, MA: MIT Press, 1998. Pp. 423.* , Applied Psycholinguistics, 22 (2001), pp. 131–134.

[35] Oranización Mundial de la Salud, *WHO — World report on disability*, Who, (2016).

[36] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, *BLEU: a Method for Automatic Evaluation of Machine Translation*, tech. rep., 2002.

[37] J. Ramos, *Using TF-IDF to Determine Word Relevance in Document Queries*, Tech. Rep. 4, 2003.

[38] H. Reese, *Microsoft's new AI app to assist the blind could be a 'game changer' in accessibility - TechRepublic*.

[39] R. Řehůřek and P. Sojka, *Software Framework for Topic Modelling with Large Corpora*, in Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Valletta, Malta, May 2010, ELRA, pp. 45–50. `http://is.muni.cz/publication/884893/en`.

[40] L. Richardson, *Beautiful Soup Documentation*, 2016.

[41] B. Rohrer, *How LSTM and RNN work*, 2017.

[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*, Tech. Rep. 3, 2015.

[43] Sébastien Pertus, *How Equadex used Cognitive Services to help people with language disorders — Microsoft Technical Case Studies*, 2017.

[44] P. Sharma, N. Ding, S. Goodman, and R. Soricut, *Conceptual captions: A cleaned, hypernymed, image alt-text dataset for automatic image captioning*, tech. rep., 2018.

[45] A. Sherstinsky, *Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network*, tech. rep., 2018.

[46] K. Simonyan and A. Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*, tech. rep., 2014.

[47] S. Singh, A. Subramanya, F. Pereira, and A. McCallum, *Wikilinks: A large-scale cross-document coreference corpus labeled via links to Wikipedia*, tech. rep., 2012.

[48] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, Tech. Rep. January, 2014.

[49] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, *Going Deeper with Convolutions*, tech. rep.

[50] M. Tanti, A. Gatt, and K. P. Camilleri, *Where to put the image in an image caption generator*, Tech. Rep. 3, 2018.

[51] A. Thomas, *Recurrent neural networks and LSTM tutorial in Python and TensorFlow - Adventures in Machine Learning*, 2017.

[52] ——, *Word2vec word embedding tutorial in python and tensorflow*, 2019.

[53] L. Van Der Maaten and G. Hinton, *Visualizing Data using t-SNE*, tech. rep., 2008.

[54] R. Vedantam, C. L. Zitnick, and D. Parikh, *CIDEr: Consensus-based image description evaluation*, tech. rep., 2015.

[55] G. von Zitzewitz, *Survey of neural networks in autonomous driving*, (2017).

[56] W3C, *H37: Using alt attributes on img elements — Techniques for WCAG 2.0*, 2008.

[57] ——, *How WAI Develops Accessibility Standards through the W3C Process: Milestones and Opportunities to Contribute — Web Accessibility Initiative (WAI) — W3C*, 2019.

[58] W3cdevs, *Web Content Accessibility Guidelines (WCAG) Overview — Web Accessibility Initiative (WAI) — W3C*, 2019.

[59] Weisstein, Eric W, *"sigmoid function." from mathworld–a wolfram web resource.*

[60] Wikipedia contributors, *Euclidean distance — Wikipedia, the free encyclopedia*, 2019. [Online; accessed 4-September-2019].

[61] ——, *Hierarchical data format — Wikipedia, the free encyclopedia*, 2019. [Online; accessed 4-September-2019].

[62] ——, *N-gram — Wikipedia, the free encyclopedia*, 2019. [Online; accessed 5-September-2019].

[63] ——, *Softmax function — Wikipedia, the free encyclopedia*, 2019. [Online; accessed 4-September-2019].

[64] ——, *Web content accessibility guidelines — Wikipedia, the free encyclopedia*, 2019. [Online; accessed 4-September-2019].

[65] K. Xu, J. L. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, *Show, attend and tell: Neural image caption generation with visual attention*, tech. rep., 2015.

[66] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, *How transferable are features in deep neural networks?*, Advances in Neural Information Processing Systems, 4 (2014), pp. 3320–3328.