CS 5984 Big Data Text Summarization
Electric Thesis and Dissertation Summarization

Ashish Baghudana, Stephen Lasky, Guangchen Li, Beichen Liu

{*ashishb, slasky, guang15, bcliu430*}*@vt.edu*

Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

Instructor: Dr. Edward A. Fox

December 15, 2018

Contents

List of Figures

List of Tables

Abstract

Automatic text summarization is the task of creating accurate and succinct summaries of text documents. These documents can vary from newspaper articles to more academic content such as theses and dissertations. The two domains differ significantly in sentence structure and vocabulary, as well as in the length of the documents, with theses and dissertations being more verbose and using a very specialized vocabulary.

Summarization techniques are broadly classified into extractive and abstractive styles - the former where salient sentences are extracted from the text without any modification and the latter where sentences are modified and paraphrased. Recent developments in neural networks, language modeling, and machine translation have spurred research into abstractive text summarization. Models developed recently are generally trained on news articles, specifically CNN and DailyMail, both of which have more readily available summaries available through public datasets.

In this project, we apply recent deep-learning techniques of text summarization to produce summaries of electronic theses and dissertations from VTechWorks, Virginia Tech's online repository of scholarly work. We overcome the challenge posed by different vocabularies by creating a dataset of pre-print articles from ArXiv and training summarization models on these documents. The ArXiv collection consists of approximately 4500 articles, each of which has an abstract and the corresponding full text. For the purposes of training summarization models, we consider the abstract as the summary of the document. We split this dataset into a train, test, and validation set of 3155, 707, and 680 documents respectively. We also prepare gold standard summaries from chapters of electronic thesis and dissertations. Subsequently, we train pointer generator networks on the ArXiv dataset and evaluate the trained models using ROUGE scores. The ROUGE scores are reported on both the test split of the ArXiv dataset, as well as for the gold standard summaries. While the ROUGE scores do not indicate state-of-the-art performance, we do not find any equivalent work in summarization of academic content to compare against.

Chapter 1

Introduction

Summarization is the task of creating a succinct and accurate summary of a large document or a set of documents. Manual text summarization is regularly used in the field of journalism where reporters need to create headlines, paraphrase interviews, and condense a large amount of information into a few sentences. However, summarization is not restricted to journalism and books. Even in the academic world, we often create summaries of journal and conference papers, theses, and dissertations to highlight the key contributions of research. To aid and substitute for manual summarization, computer scientists and linguists have worked on automatic text summarization since late 1950s [17, 2, 5, 7] with varying degrees of success.

The two main approaches to automatic text summarization are extractive and abstractive. Extractive summarization is a technique that *extracts* or selects salient sentences from the document. This generally involves ranking each sentence according to how important it is in the document [20, 9]. Abstractive summarization, on the other hand, paraphrases sentences from the document to create a summary as a human would. State-of-the-art implementations of abstractive summarization are neural network-based techniques [21, 26, 6].

In this project, we explore abstractive text summarization for electronic theses and dissertations (ETDs) from VTechWorks[1], Virginia Tech's online repository of research and scholarly works from students, faculty, and staff. Due to a lacuna of appropriate datasets in summarization for academic content, we create our own dataset of conference articles from ArXiv, an online repository of preprints. We use existing abstractive text summarization methods, and retrain them on our datasets to produce summaries of chapters from ETDs. We evaluate the summaries qualitatively by inspection, and quantitatively using the ROUGE metric. To the best of our knowledge, we have not seen any published work on summarization of such long documents.

The report is divided into multiple chapters. Chapter 2 is a Literature Survey, explaining basic architectures for abstractive text summarization and reviewing important research papers listed therein. Chapter 3 highlights the challenges with summarization,

[1]https://vtechworks.lib.vt.edu/

especially with respect to PDF parsing, dataset creation, and training the summarization methods. Chapter 4 talks about the different experiments we ran with each of the models, and discusses the hyperparameters used so as to enable replication of our results. Chapter 5 explains quantitative metrics used in the summarization domain. This chapter also includes gold standard summaries for our theses and dissertations. Chapter 6 includes ROUGE precision, recall, and F1-scores for all our models, as well as the evaluation done on our own gold standard summaries. Chapters 7 and 8 include the user's manual and developer's manual, respectively.

Chapter 2

Literature Review

As part of our implementation of this project, we have studied existing works including abstractive, extractive, and hybrid summarization approaches. In this chapter, we provide an overview of existing machine summarization techniques and their corresponding datasets. Section 2.1 provides a brief introduction to extractive and abstractive summarization, two major categories of automatic summarization. We focus primarily on abstractive summarization. The first technique we cover is a sequence-to-sequence (seq2seq) model, in Section 2.2. We subsequently focus our attention on two state-of-the-art models, one that uses pointer generator networks (Section 2.3), and another that rewrites reinforce-selected sentences. Section 2.5 will talk about public datasets that are widely used in this field.

2.1 Extractive and Abstractive Summarization

Methods of automatic summarization can loosely fit into two major categories: extractive and abstractive. An extractive summarization system directly selects content from the source collection without applying any modification. It can either create a "tag collection" of an article by extracting keywords or generate a short paragraph summary by selecting complete sentences. Unlike extractive summarization systems, abstractive summarization systems can paraphrase the original content. In this sense, abstractive summarization is more similar to how human perform summarization tasks in the real world. For this reason, our project focuses on the abstractive method of summarization.

Implementing abstractive summarization requires sophisticated natural language processing technology. Early works include parsing and summarizing simple messages by using expert systems based on symbolic rules [19]. This type of approach does not scale very well on larger problems due to the complexity and ambiguity of natural languages.

With the polynomial increase of computing power [23], mid-1980s interest in neural networks saw a dramatic revival in the 21st century. Most cutting-edge approaches are based on artificial neural networks and deep learning.

2.2 Seq2Seq Models

In the recent past, deep learning techniques that map one input sequence to another have become popular. Such models, called sequence-to-sequence (or *seq2seq*) have found success in machine translation tasks and been implemented for commercial applications like Google Translate [30].

Summarization tasks draw several parallels from machine translation tasks wherein they map a *full text* document to a *summary*. However, summarization introduces two additional levels of complexity. Firstly, the length of sequences in the input and output can be vastly different. Secondly, the output is expected to be *lossy*. Some information from the input sequence is likely to be lost. These two properties contrast strongly with machine translation tasks where one generally expects a one-to-one word level alignment in the two sequences [3].

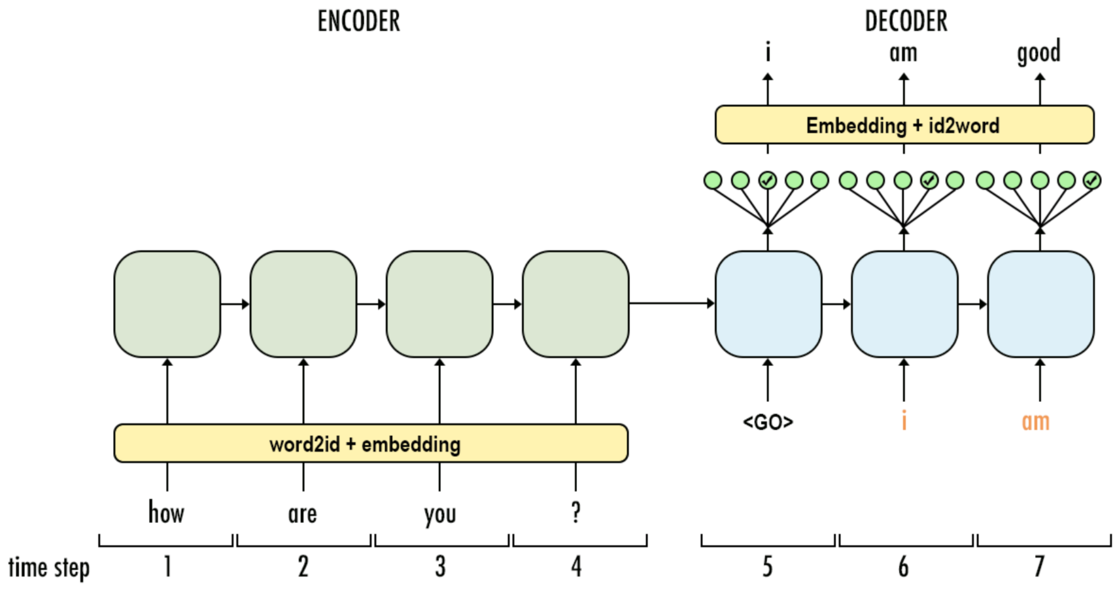


Figure 2.1: The image shows a cartoon representation of a sequence-to-sequence architecture. The encoder is generally a long-short term memory (LSTM) unit or a gated-recurrent unit (GRU). The encoder produces a single vector, commonly called a *sentence* vector. This vector is fed to the decoder, which produces tokens using a softmax layer up till a fixed length or a special end of document (`<EOD>`) token. The figure is taken from `https://medium.com/@Aj.Cheng/seq2seq-18a0730d1d77`.

A baseline model that is regularly used in abstractive summarization is an RNN-based (Recurrent Neural Network) seq2seq model with an attention mechanism [21]. The encoder consists of a bidirectional gated recurrent unit (GRU) RNN while the decoder consists of a unidirectional GRU-RNN. The hidden state size is consistent across the encoder and decoder. The attention mechanism acts over the source hidden states. On the output side, a softmax layer is used to select words from the target vocabulary. This model is largely the same as the one for a Neural Machine Translation (NMT) task. However, the authors in [21] make modifications on vocabulary generation to help the model converge faster. They use a large-vocabulary trick (LVT) [12] to restrict the vocabulary of each mini-batch to the vocabulary of the source documents in that batch and concatenate this with the most frequent words across the corpus up till a certain count.

2.2.1 Attention Mechanism

Discussion of the encoder-decoder model is incomplete without an explanation of the attention mechanism used for the decoder. Attention mechanisms are loosely based on the visual attention mechanism in humans. These help us focus on important parts of an image instead of scanning it pixel by pixel. The idea was first introduced in computer vision [8, 14], but has found applications in natural language processing [31, 29].

As shown in Figure 2.1, the encoder produces a single sentence vector that must be decoded to produce the output sequence. For long sequences, both LSTMs and GRUs fail to capture long-range dependencies between the beginning and the end of the sequence. Attention mechanisms alleviate this by considering not only the last hidden state, but instead a weighted combination of all the hidden states of the encoder. This forms the context vector for each token that the decoder decodes. This is illustrated in Figure 2.2.

2.2.2 Enhancements to the Seq2Seq Model

Nallapati *et al.* [21] enhanced the vanilla neural machine translation model in several orthogonal ways to test their applicability to summarization tasks. Some of these are detailed in this section.

Capturing Keywords using Feature-rich Encoder

As an additional input to the encoder, the authors create look-up embedding matrices for linguistic features such as part-of-speech tags, named entities, and TF-IDF statistics of the words. Continuous variables such as TF-IDF are binned to create categorical variables. The target side decoder is left unmodified.

Modeling Unseen Words using Switching Generator-Pointer

Documents in the test set may often have words that are not present in the source vocabulary. Since the vocabulary is fixed during decoding, the most common technique used is to emit a placeholder word `<UNK>`. Instead of emitting `<UNK>`, a possible modification is to point to a location in the source document. The model uses a switching mechanism to either generate a word from the vocabulary or to point to a word in the original document. Such a model is further explored in [26].

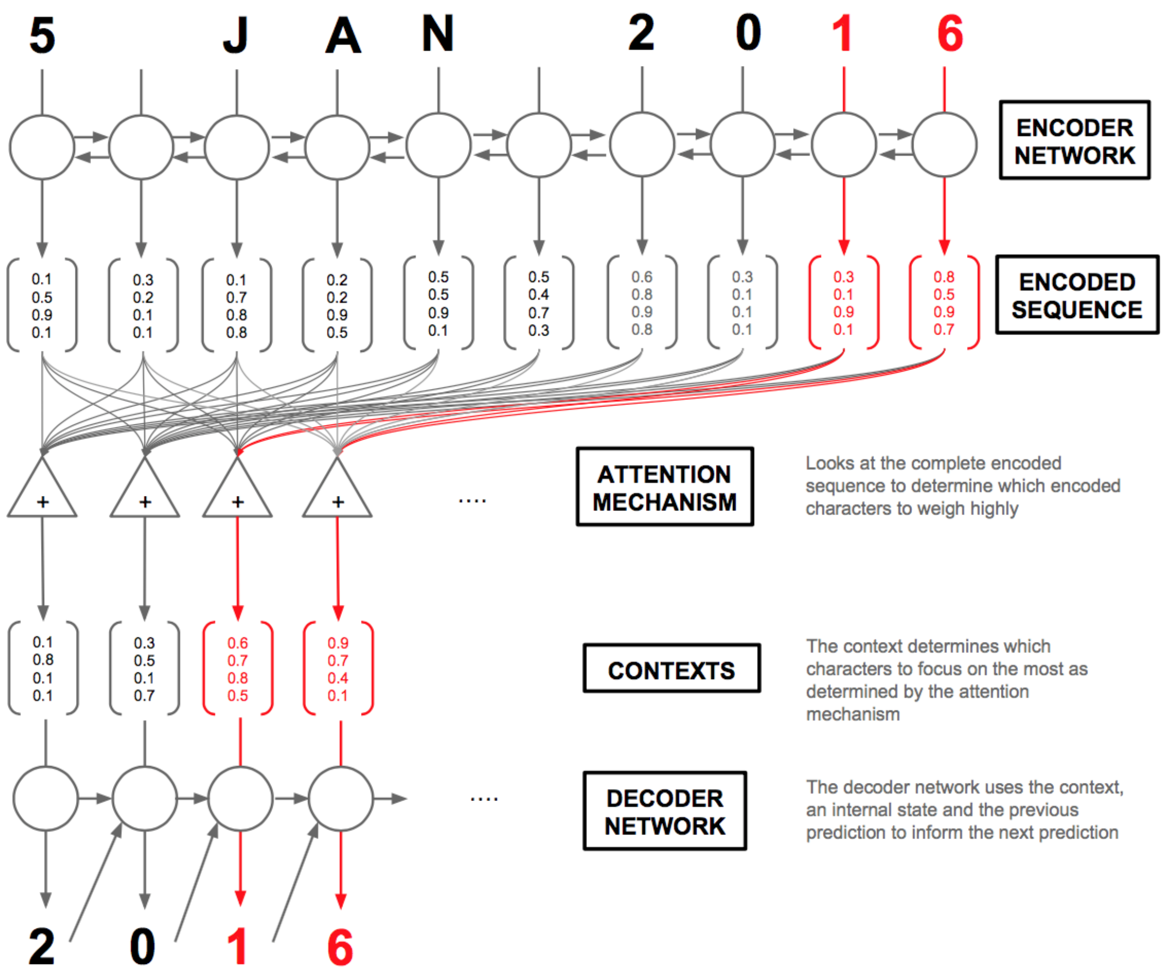

5
J
A
N
2
0
1
6
ENCODER NETWORK
ENCODED SEQUENCE
ATTENTION MECHANISM
Looks at the complete encoded sequence to determine which encoded characters to weigh highly
CONTEXTS
The context determines which characters to focus on the most as determined by the attention mechanism
DECODER NETWORK
The decoder network uses the context, an internal state and the previous prediction to inform the next prediction
2
0
1
6

Figure 2.2: Attention mechanisms help the decoder of the recurrent neural network "attend" to different parts of the input sequence selectively. They achieve this by scanning through the entire encoded sequence and creating context vectors that help weight different parts of the input sequence. These context vectors are used while generating tokens in the decoding state. The image is sourced from [1].

Hierarchical Document Structure with Hierarchical Attention

Even with an attention mechanism, encoder-decoder models do not generate good summaries when the source documents are very long. This can be tackled by using a hierarchical attention model using two RNNs on the source sequence. The first RNN operates at the sentence level, while the second RNN operates at the word level. The weights of the context vector at the word levels are modified based on the first RNN's attention.

Beam Search Decoding

Another technique to improve the generated summaries is to enhance the decoder using a heuristic search technique called beam search[13]. In beam search, we keep track of the best k hypotheses while generating each token. The number k is called the beam size. Each hypothesis is scored by an external function that decides on the quality of the summary. After generating the `<EOD>` token, the hypothesis with the best token is returned.

2.3 Summarization with Pointer-Generator Networks and Coverage

See *et al.* build on the seq2seq model discussed in [21] and enhance it in two orthogonal ways[26]. These two methods are *pointer-generator networks* and *coverage*.

Pointer-Generator Networks

Firstly, they use a probabilistic pointer-generator network that can both copy words from the source document via pointing, and generate words from a fixed vocabulary. The pointing mechanism involves calculating three parameters. The first, called *attention distribution* (denoted by a^t) is a probability distribution over the source words that tells the decoder where to look to produce the next token. The second, called the *context vector* (denoted by h_t^*, is the weighted sum of the encoder hidden states with the attention vector. The context vector is concatenated with the decoder hidden state to produce a probability distribution over the vocabulary, denoted by P_{vocab}. The third, called the *generation probability* (denoted by P_{gen}) is calculated by applying the *sigmoid* function over the attention distribution (a^t), the context vector (h_t^*), and the decoder input at the given timestep t (denoted by x_t). For each document, the extended vocabulary includes all words in the fixed vocabulary and the words in the source document. With the pointer-generator network, the decoder is able to produce out-of-vocabulary (OOV) words that a baseline *seq2seq* will be incapable of.

Coverage

See *et al.* incorporate findings from neural machine translation and include the concept of coverage to avoid repetitions of text in *seq2seq* models. The coverage mechanisms used is adapted from [28] and introduces a coverage vector c^t. Intuitively, the coverage vector measures the degree to which a given word has received *attention* so far. The vector is updated at each timestep t such that the coverage of the document is maximized.

The authors found it necessary to add a coverage loss to prevent the model from learning to attend to only one part of the document that maximizes the coverage vector.

2.4 Fast Abstractive Summarization using Reinforce-Selected Sentence Rewriting

Fast Abstractive Summarization with Reinforce-Selected Rewriting (FastAbsRL) is an abstractive-extractive hybrid model that achieves state-of-the-art results on the CNN/DailyMail dataset on all ROUGE and METEOR metrics. At a high level, the model runs by extractively selecting salient sentences, and then abstractively rewriting (compressing) them in what the authors call a "human-inspired coarse-to-fine" approach. More specifically, the extractor uses a Convolutional Neural Network (CNN) for sentence representation, which is then fed into a bidirectional LSTM-RNN which captures long-range semantic dependency between sentences. Finally, a Pointer Network is trained to extract sentences recurrently on the above output. The abstractor then compresses and paraphrases the extractor output using the standard encoder-aligner-decoder with a copy mechanism to utilize out-of-vocabulary words.

Random initialization for end-to-end training resulted in the extractor pulling irrelevant sentences, so the extractor and abstractor were trained separately using maximum-likelihood objectives. The extractor is trained as a classification problem, with the abstractor being trained as a usual sequence-to-sequence model. The full model is trained by encouraging good sentence selection from the extractor via a good ROUGE match from the abstracted sentence, and then discouraged whenever the extractor selects an irrelevant sentence. The model learns how many sentences to extract, by extracting while there are still remaining ground-truth summary sentences, and then stopping by optimizing a global ROUGE.

2.5 Datasets

Deep learning based abstractive text summarization is a very new research area with limited approaches and datasets. Of the published work that we have read, we find several commonly used datasets. The following subsections will provide a quick review of these datasets.

2.5.1 DUC 2003 and DUC 2004

DUC stands for "Document Understanding Conferences". DUC 2003 and DUC 2004 are two datasets used in the DUC workshops of 2003 and 2004. According to the DUC website[22], each dataset consists of the following parts:

1. Original Documents
2. Summaries, results, etc.
    (a) manually created summaries

(b) automatically created baseline summaries

(c) submitted summaries created by the participating groups' systems

(d) tables with the evaluation results

(e) additional supporting data and software

The most useful parts for our project are original documents, manually created summaries, and automatically created baseline summaries. We could utilize original documents and manually created summaries as our training dataset. Automatically created baseline summaries can help us during the model evaluation phase.

2.5.2 English Gigaword

The English Gigaword dataset is a comprehensive archive of newswire text data in English that has been acquired by the LDC (Linguistic Data Consortium) from 1994 to 2002. Unlike DUC 2003 and DUC 2004, this dataset does not directly contain summaries of original documents. Therefore, it creates challenge for researchers hoping to utilize it in machine summarization.

One solution of this problem includes using the first paragraph as source and the headline as target. This strategy is intuitive for this dataset—journalists are expected to provide as much information as possible in the beginning of an article, and the headline of an article generally includes the same piece of information in a shorter form. Here is an example of source-target pair:

- **Source:** Tributes pour in for late British Labour Party leader
- **Target:** Tributes poured in from around the world Thursday to the late Labour Party leader John Smith, who died earlier from a massive heart attack aged 55.

2.5.3 Large Scale Chinese Short Text Summarization Dataset (LCSTS)

LCSTS is a large corpus of Chinese short text summarization data constructed from the Chinese microblogging website Sina Weibo [11]. It consists of more than 2 million Chinese short texts along with summaries provided by their author. Additionally, it contains 10,666 entries with manually tagged relevance score, which make it extremely useful in model evaluation. A typical entry in this corpus looks like the following example (all texts are translated from Chinese):

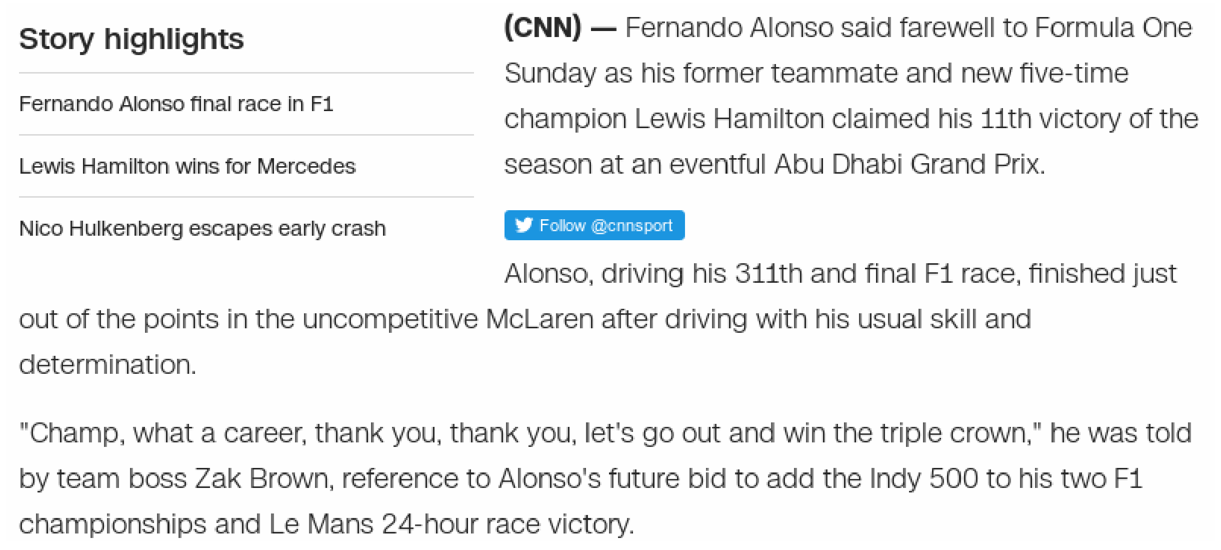
- **Short Text:** Mingzhong Chen, the Chief Secretary of the Water Devision of the Ministry of Water Resources, revealed today at a press conference, according to the just completed assessment of water resources management system, some provinces are closed to the red line indicator, some provinces are over the red line indicator. In some places over the red line. It will enforce regional approval restrictions on some water projects, implement strictly water resources assessment and the approval of water licensing.

- **Summary:** Some provinces exceeds the red line indicator of annual water using, some water project will be limited approved.
- **Human Score:** 4

2.5.4 CNN-Dailymail Dataset

The CNN-Dailymail dataset contains over 93,000 different CNN news articles where each article is stored in a separate `.story` file. The general structure of a `.story` file is the story itself followed by several "highlights" summarizing the main points of the story.

One important characteristic of this dataset is that its summaries have high compression ratio as compared to the datasets mentioned before. Unlike the previous datasets, where the summaries are created out of a few sentences of original content, highlights in this dataset are summarized from long stories consisting of multiple paragraphs.


Story highlights

Fernando Alonso final race in F1

Lewis Hamilton wins for Mercedes

Nico Hulkenberg escapes early crash

(CNN) — Fernando Alonso said farewell to Formula One Sunday as his former teammate and new five-time champion Lewis Hamilton claimed his 11th victory of the season at an eventful Abu Dhabi Grand Prix.

Alonso, driving his 311th and final F1 race, finished just out of the points in the uncompetitive McLaren after driving with his usual skill and determination.

"Champ, what a career, thank you, thank you, let's go out and win the triple crown," he was told by team boss Zak Brown, reference to Alonso's future bid to add the Indy 500 to his two F1 championships and Le Mans 24-hour race victory.

Figure 2.3: Example of a piece of CNN News with Highlights [27]

2.5.5 Cornell Newsroom Summarization Dataset

The Cornell Newsroom summarization dataset is a large scale dataset for training as well as evaluating machine summarization systems. It consists of over 1.3 million articles and summaries created by authors and editors from 38 different major publications.

This dataset is widely adopted by recent state-of-the-art researches, including the famous pointer generator paper [26] and text rank paper [20].

Chapter 3

Challenges

Abstractive Text Summarization is a relatively new research area with limited tried and tested approaches. This was exacerbated by the lack of appropriate datasets. Furthermore, our group focused on electronic theses and dissertations (ETDs) which are primarily in a PDF format. Extraction of text from PDF can be challenging as PDF is human-friendly but not machine-friendly. In this chapter, we briefly describe each challenge and our approaches to solve them.

3.1 PDF Parsing of ETDs

Since the goal of the project was to generate summaries for each chapter of an ETD, our first task was to parse each PDF document and generate a corresponding *full text* document. Our dataset consisted of approxmimately 30,000 ETDs with 13,071 disserations and 17,890 thesis. This occupied over 335GB on disk. The scale of data that we were handling made even a copy operation several hours long.

PDF parsing is an inexact method of obtaining raw text from a PDF document. We tried several PDF to text conversion tools[1,2], however we found them ineffective for academic content (equations, captions, and figures), headers and footers, and references.

We instead turned our attention to PDF-parsing tools that were designed specifically for parsing academic content. Two promising tools are Grobid[3] [24] and ScienceParse[4]. Both these tools use an underlying machine learning model that helps them parse data in more structured formats than simpler PDF2text tools. However, Grobid and ScienceParse are trained on academic papers in the double-column IEEE/ACM format, which makes them unsuitable for ETDs, which tend to be single column. We used both tools to convert ETDs to raw text.

[1] `https://github.com/jalan/pdftotext`

[2] `https://github.com/pdfminer/pdfminer.six`

[3] `https://github.com/kermitt2/grobid`

[4] `https://github.com/allenai/science-parse`

3.1.1 Grobid

Grobid is easy to set up, available as both a Java server or a Docker container. It returns structured XML for each document. It can be configured to accept either single or batches of documents as an input. However, parsing using a machine learning model is slow; it took over three days to convert all ETDs to text. Moreover, Grobid's text extraction algorithm consistently failed on ETDs, with over 70% of the documents returned with either an error or an empty text body.

3.1.2 ScienceParse

ScienceParse is built on the open source library `allennlp`[5] from AllenAI. Similar to Grobid, ScienceParse uses a machine learning model to extract text from each document. While there is no research paper associated with this library, however, the Github documentation suggests that it uses the same labeled data to train its machine learning model as Grobid. The method returns a JSON file for each document. We find that ScienceParse works better than Grobid for ETDs and can grab raw text from PDFs more accurately. However, ScienceParse is difficult to set up, and the model has difficulty differentiating between different sections of a paper. The model often considers the heading of each page (from the header) as a section title. Finally, We post-process the JSON file and extract the relevant values from the key-value representation. This forms the the raw text input for summarization tasks.

3.2 Dataset Collection and Preprocessing

After reviewing several models in the published literature, we found that the CNN-Dailymail dataset is most regularly used for training. As described before, the CNN-Dailymail dataset consists of approximately 270,000 articles with a mean sentence length of 37. However, the models we trained do not take in an input of plain text files. The dataset is converted into a binary format and a vocab file that differs for each method.

For the method described in See *et al.* [26], we processed the data using code available online on Github[6]. This obtains the CNN-Dailymail dataset from the links present in the `url_lists` directory in the root folder and uses Stanford CoreNLP [18] to tokenize the text. However, the code is specific only to the CNN-Dailymail dataset and cannot be easily extended to custom datasets or formats. Each "story" in the dataset is specified in the format given below. The first part of the document is the full text of the article. This is followed by `@highlight` annotations, which mark each sentence of the summary. The sentences are concatenated together to form the full summary.

[5]https://github.com/allenai/allennlp
[6]https://github.com/becxer/cnn-dailymail/

Format of the CNN-Dailymail Dataset

```
By
Emma Glanfield

Eric Craggs, 68, of Stockton, County Durham, pictured arriving at
court, is accused of asking for the 'Laserstar' device to be
fitted to his car in 2009

...

@highlight

Eric Craggs, 68, accused of asking for 'Laserstar' device to be
fitted to car

@highlight

Device interferes with lasers in speed guns and stops reading
being taken
```

The method described by Chen *et al.* [6] is trained on the same dataset. However, it preprocesses the dataset differently and converts each document to a JSON object. The preprocessing scripts for this are available on Github[7] as well.

Given the paucity of datasets with academic content in the summarization domain, we resorted to collecting our own dataset. A good first-degree approximation to electronic theses and dissertations involves papers from academic journals and conferences. Therefore, we collected 4,542 articles from the electronic pre-print repository called ArXiv[8]. ArXiv hosts papers from various domains including physics, mathematics, biology, computer science, and economics. We used a dataset[9] from Kaggle of nearly 42,000 ArXiv links and abstracts to seed our collection. We downloaded a 10% subsample of the Kaggle dataset to parse from ScienceParse and train our own models. As described above, the preprocessing scripts for training summarization models do not work with custom datasets. Therefore, we developed our own scripts that process custom datasets. Our work is publicly available at https://github.com/ashishbaghudana/pointer_generator_data. The format for the dataset in specified in Chapter 7.

[7] https://github.com/ChenRocks/cnn-dailymail

[8] https://arxiv.org/

[9] https://www.kaggle.com/neelshah18/arxivdataset

3.3 Pointer-Generator Networks

One of the primary models we wished to train was the Summarization using Pointer-Generator Networks described in [26]. The code for this model is publicly available on Github[10] from the original author. The code is written in TensorFlow and Python 2. However, the repository is no longer maintained and users have complained of version incompatibilities. Furthermore, none of the team members were familiar with TensorFlow's APIs. Therefore, we decided to employ a re-implementation of this model by another user in PyTorch[11]. We further adapted this code to make it more streamlined. The result is available publicly at `https://github.com/ashishbaghudana/pointer_summarizer`.

A significant problem with the original implementation is that it can only be used statically and requires a lot of preprocessing to create a binary file. We add our own scripts that make the process of testing more interactive.

We provide more details of the training methods in Chapter 7, interactive scripts in Chapter 8, and the experiments with this model in Chapter 4.

3.4 Fast Abstractive Summarization using Reinforce-Selected Sentence Rewriting

There were several challenges associated with utilizing the Fast Abstractive Summarization using Reinforce-Selected Sentence Rewriting model (FastAbsRL), both theoretically and practically. The challenges can be summarized more concisely as follows:

1. FastAbsRL builds a vocabulary from the training dataset which it then draws from during the abstraction phase to create original sentences. The size of the Arxiv dataset is substantially smaller (more than one order of magnitude) than the CNN/DailyMail dataset, on which the paper is based. As such, training FastAbsRL on only the Arxiv dataset would give the abstractor a substantially smaller vocabulary pool, and would intuitively lead to poor results. Additionally, the immense loss in data points from the substantially smaller set would cause a significant loss in learned sentence structure.

2. The Arxiv dataset includes academic papers which regularly introduce new, unforeseen terms. Although the abstractor can handle unforeseen terms by attempting to add them using the "copy mechanism", the heterogenous vocabulary of academic papers might degrade the quality of the summarization. In fact, the robustness of the abstractor demotivates training on anything smaller than the CNN/DailyMail dataset.

[10] `https://github.com/abisee/pointer-generator`

[11] `https://github.com/atulkum/pointer_summarizer`

3. ETDs and ETD chapters can be substantially longer than news articles for which the FastAbsRL is tailored. FastAbsRL was specifically designed to handle varying-lengths, but there also remains a significant domain dependency problem.

4. FastAbsRL has a very complex pipeline for training and testing. FastAbsRL also appears to be highly fault intolerant as numerous bugs were encountered during training and testing. These ended up posing serious, time-consuming challenges throughout the project.

Chapter 4

Experiments

Our team ran experiments with two different summarization models with two datasets and a baseline *seq2seq* model. We describe the hyperparameters we chose for each run in order to allow others to replicate our results as closely as possible.

We ran all of our experiments on Cascades[1]. Each node on Cascades has an nVidia V100 GPU with 12GB of RAM.

4.1 *seq2seq* Baseline Model

PGNs when run without pointing and coverage default to baseline *seq2seq* models. The settings used are given in Table 4.1.

Parameter	**Setting**
Optimizer	Adagrad
Adagrad Initial Accumulation	0.1
Learning Rate	0.15
Vocabulary Size	50,000
Hidden State Dimensions	256
Word Embedding Dimensions	200
Encoding Steps	400
Decoding Steps	100
Training Steps	500,000

Table 4.1: Settings used to train a baseline *seq2seq* model.

As expected, we observed that the default *seq2seq* model performs poorly on both the CNN-Dailymail and the ArXiv dataset. We consistently found repeated words in the generated summary. While the experiments themselves were instructive in understanding

[1] https://www.arc.vt.edu/computing/cascades/

how encoder-decoder models work, we chose not to pursue this approach because of its flaws with summarization tasks.

4.2 Pointer-Generator Networks

PGNs come in two flavors – one with only the pointing mechanism switched on, and another with both the pointing mechanism and coverage. It is easier to train with only the pointing mechanism, as convergence is simpler. We copy over settings from the baseline *seq2seq* model. We saved the model at every 5000*th* step and monitored the loss function to ensure the model was converging.

CNN-Dailymail Dataset

We repeated the experiments from the original paper [21] as there was no pre-trained model available on PyTorch. There was no change to the settings from the paper, except to reduce the number of steps from 600,000 to 500,000. Additionally, we were successfully able to use both pointing and coverage for this dataset. Training took 3 days and 7 hours. While we did not calculate ROUGE scores on the CNN/DailyMail dataset, we found the model to work quite well with news articles.

ArXiv Dataset

We replaced the original dataset with the ArXiv collection mentioned previously in 3. While this is a smaller dataset in terms of the number of articles, the length of each article and summary makes the training process longer. We increased the number of encoding steps to 8,000 to account for the increased length of the source document. Similarly, we increased the number of decoding steps to 200 for the summary. We were successfully able to train a model with just the pointing mechanism. However, when we trained with coverage switched on, we noticed the losses turned to `NaN` (not a number). This is likely because the coverage mechanism also uses a learning rate. We have yet to run a parameter sweep to find the optimal value for the coverage LR. Training took 3 days and 8 hours.

Chapter 5

Evaluation

Evaluation has long been of interest to researchers of automatic summarization systems. This task could be challenging and expensive to the extent that humans may be involved in the evaluation process. A widely accepted method is to utilize manually created summaries as gold standard summaries. Subsequent evaluations are done by machine, following certain metric(s). This chapter will introduce commonly used metrics and how they are applied in the evaluation of our summarization systems.

5.1 ROUGE

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation. It was developed by Chin-Yew Lin of the University of Southern California in 2004. ROUGE includes four different measurements: ROUGE-N, ROUGE-L, ROUGE-W, and ROUGE-S. They work by comparing an automatically produced summary (called **system summary**) against **reference summaries** (typically manually created gold standard summaries). The following subsections will cover ROUGE-N and ROUGE-L, the two most commonly used measurements in ROUGE, as well as the definition of several basic concepts.

5.1.1 Recall and Precision

Recall in the context of ROUGE means how much of the words in the system summary and reference summary are overlapping. Let W_o denote the number of overlapping words, W_r denote the total number of words in the reference summary and R denote the recall score. Recall can be calculated by:

$$R = \frac{W_o}{W_r}$$

However, recall is not a good measurement of the quality of a summary, since machine generated summaries can be extremely long and thus cover every word in the reference summary. Therefore, we need another measurement, precision, to calculate how much of the system summary is relevant. Let W_s denote the total number of words in the system summary, and P denote the precision score, where precision can be calculated by the following formula:

$$P = \frac{W_o}{W_s}$$

5.1.2 ROUGE-N

ROUGE-N is essentially n-gram recall between reference and system summary. ROUGE-N is calculated as follows [16]:

$$\text{ROUGE-N} = \frac{\sum\limits_{S \in ReferenceSummaries} \sum\limits_{gram_n \in S} Count_{match}(gram_n)}{\sum\limits_{S \in ReferenceSummaries} \sum\limits_{gram_n \in S} Count(gram_n)}$$

where n is the order of the n-gram, $gram_n$, and $Count_{match}(gram_n)$ is the maximum number of n-grams occurring both in a system summary and a set of reference summaries.

In practice, ROUGE-1 is often used in conjunction with ROUGE-2. Unlike ROUGE-1, ROUGE-2 can show the fluency of summary in the sense that system summary with higher ROUGE-2 score follows the word ordering of the reference summary more precisely.

5.1.3 ROUGE-L

ROUGE-L take the longest common subsequence (LCS) into account. ROUGE-L use LCS-based F-measure to estimate the similarity of two summaries [16]. Assume we have two summaries, reference summary X of length m and system summary Y of length n. The ROUGE-L score (denoted F_{lcs} in the following formulas) can be calculated as follows [16]:

$$R_{lcs} = \frac{LCS(X,Y)}{m}$$

$$P_{lcs} = \frac{LCS(X,Y)}{n}$$

$$F_{lcs} = \frac{(1+\beta^2)R_{lcs}P_{lcs}}{R_{lcs} + \beta^2 P_{lcs}}$$

Here R_{lcs} is LCS recall score, P_{lcs} is LCS precision score, $LCS(X,Y)$ is the length of the LCS between X and Y, and β is P_{lcs}/R_{lcs}.

5.2 METEOR

METEOR stands for Metric for Evaluation of Translation with Explicit ORdering. This metric is generally used in machine translation systems. However, due to the similarity between translation and summarization tasks, it can also be adapted to automatic summarization systems.

Like ROUGE, METEOR also utilize precision P and recall R in evaluation (see section 5.1.1 for details). Precision and recall are combined to calculate the harmonic mean F_{mean}:

$$F_{mean} = \frac{10PR}{R + 9P}$$

where P is weighted 9 times of R in the original paper [4]. To take longer matches into account, METEOR also introduces a penalty factor $Penalty$. In order to calculate $Penalty$, we need to group unigrams in system summary and reference summary into the fewest number of $chunk$s, where unigrams in each $chunk$ are adjacent in both system summary and reference summary. $Penalty$ is calculated as follows:

$$Penalty = 0.5(\frac{\#chunks}{\#unigrams\_matched})$$

Finally, the METEOR score $Score$ can be computed by the following formula:

$$Score = F_{mean} * (1 - Penalty)$$

5.3 Gold Standard ETD Summaries

To evaluate models using the metrics we introduced before, we must write our own golden standard summaries for the ETD dataset. Due to the length and complexity of a common academic thesis, even for an experienced human it is very hard to generate a concise summary of the whole thesis. Common summary models also do not have large compression ratio to support extremely long articles. Therefore, we decide to divide and conquer, that is, create chapterwise summaries instead. We include our gold standard summaries at the end of this report; see appendix A.

ETD number	Team Member
17355	Baghudana
17355	Liu
17347	Lasky
17405	Baghudana
17772	Li

Table 5.1: ETDs and Corresponding Team Members

Chapter 6

Results and Discussion

In this section, we present the results of three models – the baseline *seq2seq* model, PGNs (with and without coverage), and FastAbsRl. For our qualitative results, we generate summaries of a chapter from a gold standard master's thesis and compare these against the gold standard summaries. Through this Section, we compare our results against a gold standard summary of Chapter 1 of the master's thesis 17355 (Appendix A). This chapter was summarized by Ashish Baghudana and is reproduced verbatim below. For quantitative evaluation, we use the ROUGE metrics. We report ROUGE-1, ROUGE-2, and ROUGE-L (described in Section 5.1) values on our ArXiv dataset on a test split of 407 documents.

Gold Standard Summary

The last fifty years have seen a change in how and where Americans live and shop. Fewer people shop at Big Box stores such as Wal-Mart. As a result, several large sites and buildings have been left unoccupied. Communities have looked at bringing in new retailers and businesses into these buildings, as well as new types of development. However, new development often comes at the cost razing and reconstructing these sites. Instead of demolishing these buildings, they can be repurposed efficiently to avoid loss of investment.

6.1 Baseline seq2seq Model

Qualitative Results

The summary generated for the gold standard chapter is:

last fifty years Americans have evolved in both how and where we live , and in how we shop for the things we need . Today we look at the things we need . Today we look at the things we need . Today we look at the things we need . Today we look at the things we need . Today we look at the things we need . Today we look at the

Quantitative Results (ArXiv)

The baseline *seq2seq* had the lowest ROUGE scores amongst the three models that we trained. The summaries were poor, with several repetitions of words in the summary. The ROUGE scores on the ArXiv test dataset are given in Table 6.1.

Table 6.1: ROUGE Scores for Baseline *seq2seq* model

Metric	Precision	Recall	F1-Score
ROUGE-1	0.2476	0.1178	0.1518
ROUGE-2	0.0214	0.0083	0.0112
ROUGE-L	0.2235	0.1041	0.1112

6.2 Pointer Generator Networks

We trained two models of Pointer Generator Networks (PGNs) – one with disabling coverage and the other after enabling coverage. Results from both these models are presented in the sections below.

6.2.1 Without Coverage

Qualitative Results

The summary generated for the gold standard summary is reproduced below.

last fifty years Americans have evolved in both how and where we live , and in how we shop for the things we need . Today we look at the ubiquitous " Big Box " store from the past and see a way of shopping that fewer and fewer of us use regularly the While Realty division currently offers some 490 buildings and pieces of land for sale the

Quantitative Results (ArXiv)

Table 6.2: ROUGE Scores for Pointer Generator Networks without Coverage

Metric	Precision	Recall	F1-Score
ROUGE-1	0.2325	0.2152	0.2150
ROUGE-2	0.0423	0.0380	0.0379
ROUGE-L	0.2087	0.1932	0.1827

6.2.2 With Coverage

Qualitative Results

The summary generated for the gold standard summary is reproduced below.

Past few years and leaving communities wondering what to do with these large , imposing buildings the Wal-Mart Realty division currently offers some 490 buildings and pieces of land for sale the embodied energy these buildings already have invested into them from their construction . If we look for a new use without having to demolish.

Quantitative Results (ArXiv)

Table 6.3: ROUGE Scores for Pointer Generator Networks with Coverage

Metric	Precision	Recall	F1-Score
ROUGE-1	0.2452	0.2216	0.2237
ROUGE-2	0.0431	0.0380	0.0382
ROUGE-L	0.2184	0.1979	0.1886

6.2.3 Quantitative Results on Gold Standard Data

Aside from evaluating our models on the test split of the ArXiv dataset, we also evaluate our best performing model against the gold standard ETD dataset that was manually summarized by our team members. This consists of 8 ETDs divided into overall 28 chapters. We use the script provided by the GTA for the course to calculate ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-SU4 scores. These results are presented in Table 6.4 and Table 6.5.

Table 6.4: ROUGE (Paragraph) Scores for Gold Standard Data

Metric	Score
ROUGE-1	0.2308
ROUGE-2	0.0400
ROUGE-L	0.1538
ROUGE-SU4	0.0643

Table 6.5: ROUGE (Sentence) Scores for Gold Standard Data

Metric	Score
Max ROUGE-1	0.3333
Max ROUGE-2	0.1428

6.3 Discussion

The performance of summarization models improves from the baseline *seq2seq* to PGNs. In the baseline *seq2seq* model, we saw several repeated words and sentences. This is possibly due to the model getting stuck in a local minimum. Additionally, we see several unknown tokens in this model because of the model's inability to deal with out-of-vocabulary words. PGNs with coverage performs marginally better than that without coverage. However, none of the models produce good sentence structure. We hypothesize that complicated sentence structures in theses and dissertations poses a challenge to the summarization model in understanding the linguistics of a sentence. CNN-Dailymail, due to its simpler content and style, generates better sentences. Quantitatively, we notice that performance on the ArXiv dataset falls below the numbers reported by See, Liu, and Manning [26]. They report ROUGE-1, ROUGE-2, and ROUGE-L F1-scores of 0.3953, 0.1728, and 0.3638 respectively. Our best performing model trained on the ArXiv dataset has F1-Scores of 0.2237, 0.0382, and 0.1886 for ROUGE-1, ROUGE-2, and ROUGE-L respectively, when evaluated on the test split in the ArXiv dataset. This compares with the results we generated on the gold standard data (*i.e.* ETDs that were summarized by our team) with roughly the same F1-Scores.

Chapter 7

User's Manual

For this project, we have used various tools and development kits to assist us to set up a baseline, and to implement our own design. In this section, we briefly talk about the setups we have used, and how to run our implementations. The following sections describe: 1) how to gather ETDs and other papers from arXiv and pre-process data; 2) how to set up and run Grobid; 3) how to set up and run ScienceParse; 4) how to run summarization with PGNs; 5) how to run FastAbsRL. The first three are running and tested on `Ubuntu 16.04.5 LTS`. The last two are run and tested on the ARC clusters (Cascades and Huckleberry) from Virginia Tech.

7.1 How to Pre-Process Data

Our team is mainly working with Theses, Dissertations, and arXiv papers for training models.

7.1.1 arXiv Papers

arXiv papers work well with Grobid. We developed a script named `arxiv_fetching.py` to download all the academic papers. The data and metadata can be found online[1].

```
Usage:
python3 arxiv_fetching.py -s 0 -e 9 -t ~/arxiv_pdfs/ \
-m arxivData.json
```

Here, we have 4 arguments needed to parse with the program.

- **-s** starting index, default value is 0.

[1]https://www.kaggle.com/neelshah18/arxivdataset

- **-e** ending index, default value is 41000.
- **-t** target directory to store PDFs, default value is *./arxiv_pdfs/*
- **-m** metadata location, default value is *../resources/arxivData.json*

With this program, you can download all the arXiv papers in a JSON file.

7.1.2 Electronic Theses and Dissertations

We provide a tar file containing all the ETDs for training purposes, and to get familiar with our systems. To use the tar ball, just simply type `tar xf ETD_text.tar`

7.1.3 Pre-process Text

7.2 Grobid

GROBID (or Grobid) means GeneRation Of BIbliographic Data [10]. We grabbed the most important part that is used in our design, and provides instructions to get it work. Here we provide an installation guide for a server installation. To install Grobid, follow the procedure:

```
git clone https://github.com/kermitt2/grobid.git
./gradlew clean install
```

After this you can simply run `./gradlew clean install test` to test the installation. To run the server, use command `./gradlew run`. This will launch the server on localhost with port **8070**.

Note: the Gradle process will hang at 88%; this is normal because the web service is run sharing the same JVM as Gradle.

On the client side, we provide a *full_text_extract.py* script that can convert the PDFs to XML format and .txt formats. This is running with `pipenv`.
To install dependencies, follow the steps provided below:

```
## This is tested on Python 3.6.6. Any Python 3.6.y should work
## without problems.
## But Python 3.7.y will not work with the Grobid client!
## install pipenv
pip3 install pipenv
## install dependency
pipenv install lxml bs4 requests
```

After this you should be able to use our script to pre-process the PDFs. Our script usage is `pipenv run python full_text_extract.py --pdf-file $input --output $output`. Another easy approach is to put your PDF's directories into a text file, and use our `pdf_to_xml.sh` to process all the files you want to process.

7.3 ScienceParse

The science parser is another approach we use to pre-process the ETDs, which works pretty well. Simpler than Grobid, we can just use an HTTP request to get a JSON response of a processed file.

```
curl -v -H "Content-type: application/pdf" --data-binary @paper.pdf \
"http://scienceparse.allenai.org/v1"
# here paper.pdf is your pdf path with name 'paper'
```

You will then get a response with a json format response. You can easily use python3's json module to load the json file and process it.

7.4 Pointer Generator Networks

The original implementation of PGNs is in Tensorflow 1.2.1 and is available at https://github.com/abisee/pointer-generator. However, our team was more familiar with PyTorch and chose to fork off of a PyTorch implementation available at https://github.com/atulkum/pointer_summarizer. Our own fork resides at https://github.com/ashishbaghudana/pointer_summarizer.

The subsections below provide a walkthrough on 1) how to pre-process data for PGNs; 2) how to train a model; 3) how to test a model; and 4) how to run the model interactively.

7.4.1 Pre-process Data for PGNs

Training PGNs requires the input of the dataset as a binary file, which has already been lowercased and tokenized, and a vocabulary file. We create these as part of a pre-processing step. The code for pre-processing is available at https://github.com/ashishbaghudana/pointer_generator_data. The main file in this repository is `make_datafiles.py`. This file expects two inputs – path to a directory containing full text files and a path to a directory containing summary files. We specify our datasets in the format specified below.

```
dataset/
├── fulltext/
│   ├── 0.text
│   ├── 1.text
│   └── ...
└── summary/
    ├── 0.summary
    ├── 1.summary
    └── ...
```

The pre-processing script depends on Stanford's CoreNLP PTBTokenizer. CoreNLP is distributed as a `jar` file and can be downloaded from https://stanfordnlp.github.io/CoreNLP/. The `jar` file also has to exported to the `CLASSPATH` variable. We provide a script `setup_stanford_corenlp.sh` that can automatically set this up. The pre-requisites can be set up using the scripts given below.

```
source ./setup_stanford_corenlp.sh
echo $CLASSPATH
```

After installing the prerequisites, run `python make_datafiles.py -f [fulltext] -s [summary] -o [output]`.

```
$ python make_datafiles.py -h
usage: Preprocess dataset for Pointer Generator Networks [-h]
                                                          -f FULLTEXT
                                                          -s SUMMARY
                                                          -o OUTPUT

optional arguments:
  -h, --help            show this help message and exit
  -f FULLTEXT, --fulltext FULLTEXT
                        Path to directory containing full text
                        documents
  -s SUMMARY, --summary SUMMARY
                        Path to directory containing summaries
  -o OUTPUT, --output OUTPUT
                        Path to directory to contain the .bin files
```

7.4.2 Training PGNs

Once the data has been pre-processed, we can train models using the repository https://github.com/ashishbaghudana/pointer_summarizer. This codebase is compatible **only** with Python 2. The main file for training is `training_ptr_gen/train.py`. The path to the dataset is provided in `training_ptr_gen/data_util/config.py`. These can be modified to point to the output directory of the pre-processing scripts.

```
root_dir = os.path.expanduser("../dataset/news")

train_data_path = os.path.join(root_dir, "finished_files/chunked/train_*")
eval_data_path = os.path.join(root_dir, "finished_files/val.bin")
decode_data_path = os.path.join(root_dir, "finished_files/test.bin")
vocab_path = os.path.join(root_dir, "finished_files/vocab")
log_root = os.path.join(root_dir, "log")
```

We performed all of our training on Cascades (ARC Cluster). Running jobs on the ARC cluster is done via the PBS Job Scheduler. The job scheduler needs information about the resources requested, the queue ID, and the allocation name. We specify these in `start_train.sh`.

The job can be run on Cascades with the command `qsub start_train.sh`. If not using the PBS job scheduler, a model can be trained simply by invoking `python training_ptr_gen/train.py` from the command line.

7.4.3 Testing PGNs

Similar to the training scripts, we have written a script `start_test.sh` that can be invoked to test the model trained in the previous step. The `start_test.sh` script is compatible with Cascades and already has the PBS declaratives. To test on a machine without the PBS job scheduler, invoke `python training_ptr_gen/test.py` from the command line.

7.4.4 Interactive Mode

Finally, we also provide a way to use a trained model interactively such that the user can feed in a paragraph of text in the command line and get an abstractive summary from the model. This can be run as follows:

```
python training_ptr_gen/interactive.py [path/to/model]
```

Chapter 8

Developer's Manual

For this project, we have used various tools and development kits to assist us to set up a baseline, and to implement our own design. In this section, we briefly talk about the setups we have used, and how to run our implementations. The following sections are including 1) how to gather ETDs and other papers from arXiv and pre-process data; 2) how to set up and run grobid; 3) how to set up and run ScienceParse; 4) how to run fast_abs_rl[6]. The first three are running and tested on `Ubuntu 16.04.5 LTS`. The last one is run and tested on the ARC cluster from Virginia Tech.

8.1 How to pre-process data

Our team are mainly working with Dissertations, arXiv papers for training model.

8.1.1 arXiv Papers

arXiv papers are pretty good for Grobid. we developed a script named `arxiv_fetching.py` to download all the academic papers. This work and meta data can be found online[1].

```
Usage:
python3 arxiv_fetching.py -s 0 -e 9 -t ~/arxiv_pdfs/ -m arxivData.json
```

Here, we have 4 arguments needs to parse to the program.

- **-s** starting index, default value is 0.
- **-e** ending index, default value is 41000.
- **-t** target directory to store PDFs, default value is *./arxiv_pdfs/*

[1] https://www.kaggle.com/neelshah18/arxivdataset

- **-m** metadata location, default value is *../resources/arxivData.json*

With this program, you can download all the arxiv papers in the json file.

8.1.2 Electronic Theses and Dissertations

We will provide a tar file contains all the ETDs for training purposes and get familiar with our systems. To use the tar ball, just simply type `tar xf ETD_text.tar`. This can also be found on hadoop server at path
`/home/public/cs4984_cs5984_f18/unlabeled/data/ETD`

8.1.3 Pre-process Text

8.2 Grobid

GROBID (or Grobid) means GeneRation Of BIbliographic Data[10]. We grabbed the most important part that are used in our design, and provided manuals for that part. Here we provide installation guide for a server installation.

To install Grobid, follow the procedure:

```
$ git clone https://github.com/kermitt2/grobid.git
$ ./gradlew clean install
```

After this you can simply run `./gradlew clean install test` to test the installation. To run the server, use command `./gradlew run`. This will launch the server on localhost with port **8070**.

Note: the Gradle process will hang at 88%, this is normal because the web service is ran sharing the same JVM as Gradle.

On the client side, we provide a *full_text_extract.py* script that can convert the PDFs to xml format and txt format. This is running with `pipenv`.
To install dependencies, follow the steps provided below:

```
## This is tested on Python 3.6.6. Any Python 3.6.y should work
## without problems. But Python 3.7.y will not work with Grobid client!
## install pipenv
$ pip3 install pipenv
## install dependency
$ pipenv install lxml bs4 requests
```

After this you should be able to use our script to pre-process the PDFs. Our script Usage is `pipenv run python full_text_extract.py --pdf-file $input --output $output`. Another easy approach is to put your PDF's directories into a text file, and use our `pdf\_to\_xml.sh` to process all the files you want to process.

8.3 ScienceParse

The ScienceParse is another approach we use to pre-process the ETDs, which works pretty well. Simpler than Grobid, we can just use HTTP request to get a json response of processed file.

```
$ curl -v -H "Content-type: application/pdf" --data-binary @paper.pdf \
"http://scienceparse.allenai.org/v1"
# here paper.pdf is your pdf path with name 'paper'
```

You will then get a response with a json format response. You can easily use python3's json module to load the json file and process it.

8.4 Pointer Generator Networks

The main developer settings for training PGNs resides in the configuration file `training_ptr_gen/data_util/config.py`. We provide a snippet of these configuration parameters below and explain the important ones.

```
# Hyperparameters
hidden_dim = 256
emb_dim = 200
batch_size = 8
max_enc_steps = 400
max_dec_steps = 100
beam_size = 4
min_dec_steps = 35
vocab_size = 100000

lr = 0.15
adagrad_init_acc = 0.1
rand_unif_init_mag = 0.02
trunc_norm_init_std = 1e-4
max_grad_norm = 2.0

```

```
pointer_gen = True
is_coverage = False
cov_loss_wt = 1.0

eps = 1e-12
max_iterations = 100000

use_gpu = True

lr_coverage = 0.15

# Logging
log_level = 'DEBUG'
log_file = os.path.join(log_root, '{}_{}.log')
```

1. `hidden_dim`: The number of hidden units for both the encoder and decoder of the model. A general rule-of-thumb is that as we increase the number of hidden units, we also increase accuracy. However, increasing the hidden units also increases the training and testing time.

2. `emb_dim`: Dimensions for the word embeddings. Typically, this value is set no larger than 300.

3. `batch_size`: Number of datapoints per batch.

4. `max_enc_steps`: The number of words that will be encoded from each input data point. This will vary based on the average length of the full text document. We set this to 8000 for the arXiv dataset and 400 for the CNN/Dailymail dataset.

5. `max_dec_steps`: The number of words that will be decoded for each summary. This will vary based on the average length of the summary document. We set this to 200 for the arXiv dataset and 100 for the CNN/Dailymail dataset. The ratio of the `max_enc_steps` and `max_dec_steps` defines the compression ratio.

6. `beam_size`: Number of possible hypotheses to explore when decoding.

7. `vocab_size`: Trim the vocabulary if it exceeds this value.

8. `lr`: Learning rate for the Adagrad optimizer.

9. `adagrad_init_acc`: Initial accummulation value for the Adagrad optimizer.

10. `pointer_gen`: Enable or disable the pointing mechanism in the model.

11. `coverage:` Enable or disable the coverage mechanism in the model. If the model is run with `pointer_gen = False` and `coverage = False`, it is equivalent to running a basic Seq2Seq with Attention model.

As a developer, it becomes important to perform several experiments by changing these values to obtain the best model. We find that by setting `coverage = True`, the training sometimes does not converge, and the loss becomes `NaN`. Solutions to this are discussed more at https://github.com/abisee/pointer-generator#help-ive-got-nans.

Chapter 9

Lessons Learned

Automatic text summarization is still a very nascent research area with several problems and challenges. The field began as an extension of machine translation and therefore, most architectures extend *seq2seq* models. We find that these models produce mediocre results. One of the primary issues with *seq2seq* models is the repetition of words that occurs because the model fails to converge. Pointer generator networks (PGNs) and Fast Abstractive Summarization using Reinforce-Selected Sentences (FastAbsRL) are improvements over *seq2seq* models, especially since they use pointing mechanisms, coverage, and a hybrid extractive-abstractive approach to improve summarization. These models however take a long time to train and are very resource intensive.

In the training process, we find that cross-entropy or negative log likelihood loss are ineffective metrics for measuring how well the model is learning. While the loss decreases substantially over the course of the training period, we still do not see effective sentence formation in the summaries. This is a potential area of improvement in text summarization using deep learning. We also notice that the number of encoding and decoding steps need to be tweaked according to the dataset being used. In the CNN-Dailymail dataset, the articles were roughly 400 words long and the summaries were approximately 100 words long. In contrast, the articles in the ArXiv dataset were on an average 6000 words long and abstracts were about 200 words long. This required the number of encoding steps to be increased so as to capture the essence of the full article. However, even with the increase in encoding steps, training is not very effective, as it is difficult to condense information from large texts into relatively small vectors. Finally, we also find that the ROUGE metric is not indicative of real world performance, especially because summaries can be expressed in several different ways, with different sentence phrasing and words. A different way of evaluating summaries might be through the use of Paragraph2Vec [15] and calculating the cosine similarity between the gold standard summary and the generated text.

At last, in order to process the ETDs and find proper dataset, we tried different tools, including using Grobid[10] and Science Parse[25], as well as various sets of data, such as arXiv papers. We have learned how to set up the models and to use proper tools for pre-processing the text.

Chapter 10

Acknowledgements

We would like to thank Dr. Edward A. Fox for giving us the opportunity to work on exciting and challenging problems in automatic text summarization. We greatly benefited from his advice, comments, and guidance in framing the problem of summarizing ETDs using a dataset from ArXiv. We used the ARC clusters provided for the course to their full extent for training multiple models. This was an integral part of our trial and error-based learning. We also want to thank the GTA, Mr. Liquing Li, for his support and patience throughout the course. We used his assistance several times while running our scripts on the cluster. We also acknowledge the NSF grant **IIS-1619028** that made datasets and resources accessible to us.

Appendix A

Gold Standard Summary

ETD 17355 Baghudana

Chapter 1

With increasing pressure on freshwater resources, seawater desalination is becoming an important research area to meet the water needs of the world. Membrane distillation is a promising process which could be extensively used for desalination in the future because of its versatility and resistance to fouling. Of many membrane distillation techniques, air gap membrane distillation is shown to be the most energy efficient. Air-cooled systems for AGMD are compared against water-cooled systems. To make the desalination process sustainable, solar energy, concentrated with optical waveguides, can be used to meet the thermal needs of the cooling system, thus reducing cost.

Chapter 2

Experimental setups for optimization and parametric studies include the AGMD module and superhydrophobic surface. An air-cooled air gap membrane distillation system works well, with lower energy requirements, due to its modular design. The conductivity of the support mesh is an important factor in flux values, with copper mesh giving good yield. Furthermore, increasing the air gap reduces the flux value. A hydrophobic surface with small air gap in series configuration, which reduces the temperature drop of the saline water, works best and results in about three times more yield when compared to a single pass water-cooled system.

Chapter 3

There is an analytical closed form solution for radial waveguides that can be used for solar thermal desalination. Also discussed are simulation for ray trace analysis, loss mechanisms associated with waveguides, and a cost model for waveguide economics. A parametric study considered net thermal power from the waveguide to the receiver, collection efficiency, and aperture area requirements for the system. Feasible design and operational envelopes of the radial planar waveguide concentrator-receiver system are reported based on the structural and thermal constraints, like grating size, waveguide thickness and radius, and receiver radius. The results from this analytical model are backed up by TracePro simulations.

Chapter 4

Air-cooled AGMD systems with modular design have lower energy requirements than water-cooled counterparts while still giving comparable performance. Conductivity of the support mesh had a significant effect on the yield of freshwater. Copper meshes performed the best followed by aluminum and steel. Desalination can be made more sustainable by using a solar energy concentration system. An analytical model for radial waveguide based solar concentration, to make the process sustainable, was developed. A parametric study considered net thermal power delivered to the receiver from the waveguide, collection efficiency, and aperture area requirement. A cost model was developed to minimize levelized cost of power (LCOP). Future studies, including of manufacturing, could further improve yields and validate use in natural environments.

ETD 17355 Liu

Chapter 1

The landscape of hardware and software has changed in the last decades: the hard disk drives and DRAM has been supplanted by SSDs, and the cloud-based storage services and application workloads expand a lot. This thesis argues that storage tiering is an effective technique for balancing operational or deployment costs and performance in modern storage systems. In order to solve the draw back of PCM which is low write-endurance, a tired cache design named THMCACHE has beed designed and implemented. On the other side, to solve the software side issue, a Cloud Analytics Storage Tiering solution that enables cloud tenants can use to reduce monetary cost and improve performance of analytics workloads is also designed and implemented.

Chapter 2

thmCache is a tiered cache design that combines low-endurance persistent memory devices with hish-endurance PMEM devices. thmCache caches write-intensive blocks in the high-endurance rite to write as much as possible to reduce the write operation in the low-endurance tier. The AccuSim cache simulater has been extended to support a hybrid PMEM architecture and it includes an implemntation of thmCache. Results show that write-intensive workloads has a 75% reduction in PCM write using DRAM with only 15% of the PCM size. This will overcome the drawbacks of persistent memory devices.

Chapter 3

Cloud Analytics Storage Tiering is a solution that cloud tenants can use to reduce maintaince cost while having a good performance. This idea provides storage tiering support for data analytics tools in the cloud. It can perform offline workload profiling to predict job performance model for different tasks, and combine the models with workload specifications. CAST also has a optimization model called CAST++ in which reuse pattrns and across-jobs interdependencies common in realistic analytics workloads. Tests show that CAST++ has a 1.21x performance gain and reduce the cost by 51%.

Chapter 4

Big Data analytics platforms becomes more popular with Cloud object stores due to the simplicity of management of large blocks of data at a large scale. Hard disk drives is the most popular way because of the cost. But some big data applications need to have strict performance, HDDs are not a good choice. Faster storages such as SSDs are desirable but expensive. So A tired object store contains fast and slow storage devices is a desiable choice. This approach allows both a service provider and its tenants to engage in a pricing game. This is a result of win-win situation.

ETD 17347 Lasky

Chapter 1

With the automotive industry rapidly pushing towards fully autonomous vehicles, an Autonomous Vehicle Research Platform (AVRP) would be very useful for carrying out research on self-driving cars. A multitude of successful AVRPs have been deployed with varying degrees of success utilizing a variety of sensors (including ultrasonics), driving mechanisms, and computer hardware. In 2007, 35 teams competed in the DARPA autonomous driving challenges, with CMU, the winning team, using HDR cameras, LIDAR, and radar on the competition vehicle. Virginia Tech also competed, and used LIDAR and cameras for perception with the vehicle completing the 60-mile course in about 6 hours. Faculty at Virginia tech were interviewed to get a better understanding of what features researches would like to see in an AVRP, which is the foundation of this thesis. The goal of this thesis is to provide operational specifications for the development of a level 4 capable AVRP, which are capable of being fully autonomous within a specified operational design domain (ODD), but are not expected to be fully autonomous outside of ODD.

Chapter 2

The interdisciplinary design needs for an Autonomous Vehicle Research Platform (AVRP) include vehicle communication, perception, and vehicle adapting to different environments. Vehicle communication will require a radio transmitter and transceiver; path following and object recognition will require cameras and sufficient processing hardware. Adaptive Cruise Control (ACC) and Cooperative Adaptive Cruise Control (CACC) also will require the use of radar and communication. Additionally, monitoring and collecting data will require access to the AVRP communication buses. The basic autonomous platform will require sensing hardware (LIDAR, radar, ultrasonic, cameras), GPS/INS (inertial navigation system), inertial measurement unit (IMU), computers for perception processing, computers for route planning and obstacle avoidance, data storage, drive by wire (DBW) system for vehicle input, and power and communication bus(es). A research platform will use a two-layer setup, hooks to connect additional computers, hooks to access communication buses, universal mounting racks, and appropriate power outlets.

Chapter 3

The design of the Autonomous Vehicle Research Platform (AVRP) includes about drive by wire (DBW), as it allows the AVRP to be fully functional in autonomous mode without any external manual controlling. For navigation, the highest accuracies can be achieved by using a combination of GPS and dead reckoning. Sensing is critical in an AVRP, including cameras, radar, LIDAR, acoustic (ultrasonic), wheel speed, and steering angle sensors. From a systems perspective, the AVRP needs to be a two-layer system (admin, user) such that custom software may be mounted and unmounted on the user side without compromising the integrity of the AVRP. An electric or hybrid vehicle is preferable such that there is sufficient electrical power (through the power bus) to run the base AVRP and all attached hardware. Finally, universal mounting racks on the front, top, and rear of the vehicle are essential to allow researchers to easily install and uninstall hardware.

Chapter 4

The base sensor suite allows for the Autonomous Vehicle Research Platform (AVRP) to perform the following capabilities without any user added sensors or hardware: text, sign, and signal recognition, Adaptive Cruise Control (ACC), blind spot detection, lane following, path following, obstacle detection, obstacle avoidance, emergency braking and stop-and-go-traffic. AVRPs require validation and testing, and can be broken up into three pieces: software, hardware and overall system validation. Additional validation should include point-to-point testing, unprivileged admin access and running simulations.

Chapter 5

An Autonomous Vehicle Research Platform (AVRP) must be developed on a need-driven basis which can be ascertained by interviewing potential future users. The base specs for an AVRP include a multilayer user system, ports to connect additional computers and devices, separate communication buses, universal mounting racks, an adequate sensor suite and an appropriate power source. The operational specifications are not limited simply to automobiles, which demonstrates the scalability and applicability of the AVRP. To extend the focus on hardware, future work on software specifications is desirable. For hardware, the testing of various models and brands of sensors should be carried out.

ETD 17405 Baghudana

Chapter 1

The last fifty years have seen a change in how and where Americans live and shop.

Fewer people shop at Big Box stores such as Wal-Mart. As a result, several large sites and buildings have been left unoccupied. Communities have looked at bringing in new retailers and businesses into these buildings, as well as new types of development. However, new development often comes at the cost razing and reconstructing these sites. Instead of demolishing these buildings, they can be repurposed efficiently to avoid loss of investment.

Chapter 2

Several unused Big Box stores have been successfully converted to public facilities such as libraries and health clinics. However, architects often have difficulty designing around existing column grids and structures, increasing time and cost of construction. This can be simplified by using repetitive construction techniques, such as modulated and pre-fabircated construction. This technique has been applied to create single family homes, multi-story hotels, and commercial buildings. The Potomac Yard Center in norther Alexandria, Virginia provides an ideal setting for exploring reconstruction without demolition. The area houses retail space and three to four-story townhouses well connected by the Washington Metro. However, the architectural designs and materials are mostly conventional, and create a sense of cost-cutting in the neighborhood. Though the community has asked for reforms to the construction plans, the new plans will not come into force until 2019. Is it possible to deliver the same square footage with shorter wait times by using the existing built environment?

Chapter 3

Word diagramming, sketching, 3D computer and hand modeling, and phasing diagrams helped understand how to use existing infrastructure to speed up reconstruction. Big box stores, with large parking spaces, are generally car-centric. This was changed to pedestrian-friendly spaces by introducing on-street parking, street-facing retail, and street trees. New buildings built in the out lots and parking lots can utilize a modified slab on grade foundation built on top of existing asphalt, thereby saving large amounts of asphalt and concrete. Residential townhouse type units in different sizes also add depth to the site. Residential construction can be pre-fabricated off-site in different colors to differentiate types of units, as well as add color and life to the streets. Addition of street lights also creates a welcoming street environment. All these modifications can be done without changing the existing beam structure.

Chapter 4

The ease of assembly of pre-manufactured components makes it suitable for constructions at abandoned sites such as Big Box retail stores, as well as, the Potomac Yard Center. Pre-fabricated components also allow for further planned development while at the same time re-use existing construction. It will help to strengthen and build up the community along with phase one and foster the neighborhoods desire to undertake phase two in the projected twenty to twenty five years it may take to reach that point of construction.

ETD 17772 Li

Chapter 1

It will be nice to be able to prove that a program is bug free in critical systems. USIMPL, the theorem-proving aspect of Orca, which is being developed within Virginia Tech's Systems Software Research Group, builds upon Isabelle/UTP (from Unifying Theories of Programming by Hoare and He) and the Simpl language (by Schirmer). The result is a theorem-based language including additional features such as algebraic laws of programming and a forward verification condition generator.

Chapter 2

USIMPL builds upon formal methods, Isabelle, UTP, Isabelle/UTP, and the Simpl language, as well as some other related works. Formal methods can be generalized as the usage of logical inference rules to derive well-formed conclusions in sound mathematical frameworks, applied to the domains of hardware and software development. Isabelle is designed as a successor to the HOL series of ITPS. Types, functions, syntax translations, proofs, and locales of Isabelle involved. The book Unifying Theories of Programming (UTP) was written to provide denotational semantics for a generalized nondeterministic imperative programming language expressed in a common setting. Simpl is essentially an extension of the IMP language presented by Winskel. Many design decisions were made when implementing UTP in Isabelle.

Chapter 3

The basic Isabelle/UTP system is extended by adding program state features, new algebraic laws, and scoping rules. The USIMPL VCG (verification condition generator) can generate the Strongest Postcondition (SP). Hoare rules are used to generate the verification conditions (VCs).

Chapter 4

Real world applications of the USIMPL VCG include using the USIMPL VCG to prove the correctness of insertion sort, which is a relatively simple sorting algorithm, and quicksort, which is more complex.

Chapter 5

USIMPL's contributions include extension to the Isabelle/UTP implementation of Hoare and He's UTP in the proof assistant Isabelle with features of the Simpl language, as well as development of additional algebraic laws for Isabelle/UTP language constructs. A helper-library is needed for beyond auxiliary lemmas. For loops and other control flow structures with complex behavior, and development of invariants and assertions to continue the flow of proving, can be difficult and tedious. To connect USIMPL to the wider world, more automated methods are required. Another possible approach is to directly integrate pre-/postconditions and invariants into the language with which programs are written. Future work will include scoping support, recursive function calls, a heap-style memory model, and more testing features.

ETD 14603 Lasky

Chapter 1

Computer vision is a branch of artificial intelligence (AI) that is concerned with visual recognition. This thesis is concerned with Object proposals, which aid in locating and classifying objects in images. The primary contributions of this thesis include a new data-driven approach for generating object proposals, a MATLAB library for simplifying interfacing with object proposal algorithms and identifying and correcting biases in object proposal evaluation protocols. Chapter 2 reviews related works, chapter 3 discusses the novel data-driven object proposal approach, chapter 4 introduces and demonstrates the object proposal library, and chapter 5 discusses and empirically demonstrates the "gameability" of modern evaluation metrics.

Chapter 2

Object detection is concerned with detecting the location and type of objects that are contained in a n image (if any). The process is two-fold: a set of regions which likely contain objects are proposed (using either sliding window or region proposal) and then the set of regions are classified. The sliding window method for region proposal is a brute-force solution for selecting regions in an image, but is typically no longer used as of 2015 due to it's computational infeasability with O(n^4) complexity. A new class of techniques called object proposal techniques aim to remedy the computational infeasibility of sliding window. Object proposals are broadly categorized into two categories: window scoring involves selecting a subset from the set of all windows within a frame based on some scoring metric, and segment-based that involves over-segmenting an image and then converting the segments into bounding boxes. The dataset typically evaluated on is the PASCAL VOC detection test set, with the most popular metrics being Recall @ IOU Threshold, Area under the recall Curve (AUC), Volume Under Surface (VUS), Average Best Overlap (ABO), and Average Recall (AR).

Chapter 3

This chapter introduces a novel non-parametric, data-driven approach for generating object proposals. This new approach is neither window-scoring nor segmentation based, but instead being data-driven. The system generates object proposals by first finding the k-nearest neighbors via the DeCAF and GIST feature spaces from a database annotated with bounding boxes. The bounding boxes of the retrieved images are then transferred over to the query image by either establishing dense pixel-wise correspondence using SIFT-flow or by warping the neighbors and their annotations to be mapped onto the query image. The authors experiment using label transfer with and without sift, using DeCAF or Gist features among varying values of k between 5 and 100. While the results are not the best-performing among the most popular algorithms (like selective search), they are also not the worst. This chapter introduced a novel approach for regional proposals, with the Label Transfer DeCAF with no Sift flow (LT DeCAF no SF) being the best. Compared to other popular region proposal algorithms, the author's algorithm performed average at best, but was conducive to a segway into new ideas.

Chapter 4

This chapter provides an overview of the object proposals library which is a GitHub repository for object proposal algorithms. It is common in the vision research community for authors to make their algorithms open-source, which is a great service but leads to disparate implementations of the algorithms that are absent of a common formatting. More specifically, the proposals may specify different coordinate systems. This thesis proposes an easy-to-use Object Proposal Library which generates proposals using all existing object proposal algorithms, standardizes the formatting and evaluates the proposals on the following metrics: recall at specific threshold, recall at specific number of proposals., area under recall curves and average best overlap. See batra-mlp-lab/object-proposals.

Chapter 5

Object proposals can have two different interpretations, one which only detects the existence of objects, without classification, and the other improving object detection pipelines such as 20 PASCAL categories. The idea of this chapter is that current evaluation protocols are only suitable for detection proposals and is a biased protocol for category-indecent object proposals. Given datasets like PASCAL emit certain object categories, these evaluation metrics fail to measure the objects being detected that weren't originally annotated in the ground truth. This raises cause for concern because it allows the metrics to be "gameable" or susceptible to both intentional and unintentional manipulation. One glaring flaw is that PASCAL is only partially annotated allows models to be tuned (intentionally or unintentionally) to only 20 categories. None of the currently proposal methods seem to be biased, but we should still be wary of overfitting as a community to a specific set of object classes.

Chapter 6

Object proposal algorithms have become standard in object detection pipelines, specifically due to their computational efficiency gain over the sliding-window approach. Chapter 2 summarized various object proposal algorithms, chapter 3 proposed a new approach for generating object proposals, chapter 4 provided an overview of the Object Proposals Library introduced in the thesis. Chapter 5 did several things, including: reporting the bias and "gameability" of evaluation protocols, emphasized the different interpretations of object proposals, demonstrated gameability via a simple experiment, conducted thorough evaluation of existing object proposal methods, and introduced densely-annotated the PASCAL VOC 2010 to remedy evaluation protocol concerns. Future work might include better distance metrics for Chapter 3, and if the analysis in this thesis can be extended to other kinds of proposals, such as patio-temporal, RGBD, etc.

References

[1] Zafarali Ahmed. *How to Visualize Your Recurrent Neural Network with Attention in Keras*. https://medium.com/datalogue/attention-in-keras-1892773a4f22. (Accessed on 11/18/2018).

[2] Richard Alterman and Lawrence A. Bookman. "Some computational experiments in summarization". In: *Discourse Processes* 13.2 (1990), pp. 143–174.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[4] Satanjeev Banerjee and Alon Lavie. "METEOR: An automatic metric for MT evaluation with improved correlation with human judgments". In: *Proceedings of the ACL workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 2005, pp. 65–72.

[5] John B. Black and Robert Wilensky. "An evaluation of story grammars". In: *Cognitive science* 3.3 (1979), pp. 213–229.

[6] Y.-C. Chen and M. Bansal. "Fast Abstractive Summarization with Reinforce-Selected Sentence Rewriting". In: *ArXiv e-prints* (May 2018). arXiv: 1805.11080 [cs.CL].

[7] Richard Edward Cullingford. *Script application: computer understanding of newspaper stories.* Tech. rep. Yale University, New Haven, Connecticut, Dept. of Computer Science, 1978.

[8] Misha Denil et al. "Learning Where to Attend with Deep Architectures for Image Tracking". In: *Neural Computation* 24.8 (2012), pp. 2151–2184.

[9] Günes Erkan and Dragomir R Radev. "Lexrank: Graph-based lexical centrality as salience in text summarization". In: *Journal of artificial intelligence research* 22 (2004), pp. 457–479.

[10] *Grobid*. https://github.com/kermitt2/grobid. [Online; accessed 29-October-2018]. 2008 — 2018.

[11] Baotian Hu, Qingcai Chen, and Fangze Zhu. "LCSTS: A Large Scale Chinese Short Text Summarization Dataset". In: *CoRR* abs/1506.05865 (2015). arXiv: 1506.05865. URL: http://arxiv.org/abs/1506.05865.

[12] Sébastien Jean et al. "On Using Very Large Target Vocabulary for Neural Machine Translation". In: *arXiv preprint arXiv:1412.2007* (2014).

[13] Philipp Koehn. "Pharaoh: a beam search decoder for phrase-based statistical machine translation models". In: *Conference of the Association for Machine Translation in the Americas*. Springer. 2004, pp. 115–124.

[14] Hugo Larochelle and Geoffrey E Hinton. "Learning to Combine Foveal Glimpses with a Third-order Boltzmann Machine". In: *Advances in Neural Information Processing Systems*. 2010, pp. 1243–1251.

[15] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *International Conference on Machine Learning*. 2014, pp. 1188–1196.

[16] Chin-Yew Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text Summarization Branches Out* (2004).

[17] Hans Peter Luhn. "The automatic creation of literature abstracts". In: *IBM Journal of research and development* 2.2 (1958), pp. 159–165.

[18] Christopher Manning et al. "The Stanford CoreNLP natural language processing toolkit". In: *Proceedings of 52nd annual meeting of the Association for Computational Linguistics: system demonstrations*. 2014, pp. 55–60.

[19] Elaine Marsh, Henry Hamburger, and Ralph Grishman. "A Production Rule System for Message Summarization." In: *AAAI*. 1984, pp. 243–246.

[20] Rada Mihalcea and Paul Tarau. "Textrank: Bringing order into text". In: *Proceedings of the 2004 conference on empirical methods in natural language processing*. 2004.

[21] Ramesh Nallapati et al. "Abstractive Text Summarization using Sequence-to-Sequence RNNs and Beyond". In: *arXiv preprint arXiv:1602.06023* (2016).

[22] NIST. *Past Data*. https://www-nlpir.nist.gov/projects/duc/data.html. [Online; accessed 24-November-2018]. 2014.

[23] I Present. "Cramming More Components onto Integrated Circuits". In: *Readings in computer architecture* 56 (2000).

[24] Laurent Romary and Patrice Lopez. "Grobid - information extraction from scientific publications". In: *ERCIM News* 100 (2015).

[25] *Science Parse*. https://github.com/allenai/science-parse. [Online; accessed 29-October-2018].

[26] A. See, P. J. Liu, and C. D. Manning. "Get To The Point: Summarization with Pointer-Generator Networks". In: *ArXiv e-prints* (Apr. 2017). arXiv: 1704.04368 [cs.CL].

[27] CNN Sport. *Abu Dhabi GP: Fernando Alonso farewell as Lewis Hamilton caps triumphant season with 11th win.* https://edition.cnn.com/2018/11/25/motorsport/abu-dhabi-gp-alonso-hamilton/index.html. [Online; accessed 24-November-2018]. 2018.

[28] Zhaopeng Tu et al. "Modeling coverage for neural machine translation". In: *arXiv preprint arXiv:1601.04811* (2016).

[29] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. 2017, pp. 5998–6008.

[30] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *CoRR* abs/1609.08144 (2016). URL: http://arxiv.org/abs/1609.08144.

[31] Kelvin Xu et al. "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention". In: *International Conference on Machine Learning*. 2015, pp. 2048–2057.