Web Application for Generating Image Descriptions Using Natural Language Processing and Computer Vision Techniques

Adam Zucker
Department of Computer Science
Yale University
adam.zucker@yale.edu

Dragomir Radev
Department of Computer Science
Yale University
dragomir.radev@yale.edu

Lawrence Staib
Department of Electrical Engineering
Yale University
lawrence.staib@yale.edu

Abstract

This project involved the creation of a functional, deployable web application that automatically generates descriptions or "captions" for uploaded images. It consists of two parts, the web application that hosts and interfaces with the captioning mechanism and the computational engine that actually handles the image description generation. The web application interface for the caption generation uses the Flask web framework (Python) for the back-end, ReactJS (JavaScript) for the front-end interactions, and ReactBootstrap (CSS) for the user interface styling. The caption generation engine combines some existing CV and NLP engines, namely TensorFlow's im2txt algorithm and Clarifai's Image Recognition API in a novel way to generate captions for images.

1 Introduction

Throughout history, as society has become more technologically advanced, people have been constantly seeking to automate more and more tasks to improve general quality of life. The advent of modern computers, in particular, has opened endless possibilities, as such complex analytical machines can perform far more nuanced tasks than their "naive" purely electro-mechanical machinery counterparts can. It thus comes as no surprise that fields of artificial intelligence and machine learning have seen unbelievable investment and advancement over the past few decades, some standouts being IBM's Deep Blue and Watson, Apple's Siri, and Google's AlphaGo.

One interesting, more recent area of research in the field of AI is computer vision, in which a computer can extract information and understanding from visual data, namely images. There are many practical motivations for wanting to create a system that can accomplish such a task, from those that pertain to the aforementioned automation to creating assistive tools for the visually impaired.

In particular, this project explores how a computer can express what it "sees" in an image using natural language, much as a human could do.

The end product of this work is an image caption generation engine built partly upon existing algorithms, packaged into a custom-built web app. The project is thus split into two parts: the website and the caption generator.

2 Web Application

As is standard with all web development projects, the needs of this app dictated both its front-end and back-end design choices. The focus of this project was to be the caption generation engine, so the app was designed to be minimalistic, with the image uploader and captioned image components front and center. Besides these two components, the only other major component of the app was a document viewer, in which a user could view this report as well as the project's proposal and abstract.

2.1 UI Overview

The three components described above were conceived as two separate views: the uploader view, which contained either the uploader or a captioned image (depending on whether or not an image was uploaded) and a document viewer view, which

would display the documents in PDF form within a pane. But rather than using two separate webpages for these two views, they were combined into a single-page view that would toggle between the different views based on interactions with a navigation bar at the top of page.

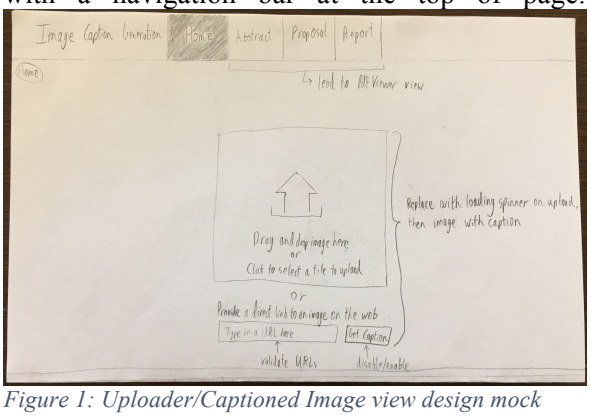

Figure 1: Uploader/Captioned Image view design mock

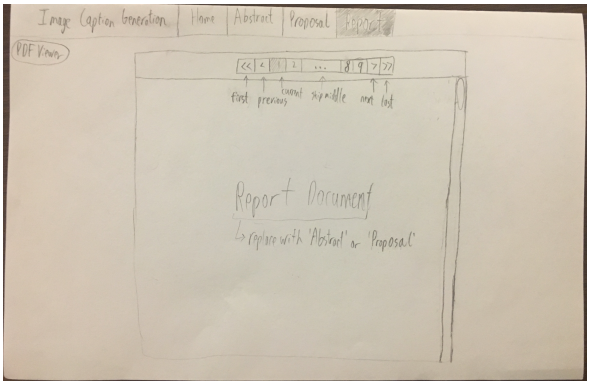

Figure 2: Document Viewer view design mock

Because this app is relatively small, it was deemed feasible to place both views within a single view, to make a single-page, single-URL web app. This choice had the benefit of making the app easier to interact with (no full page reloads), but as a result, required a more methodical front-end engineering design.

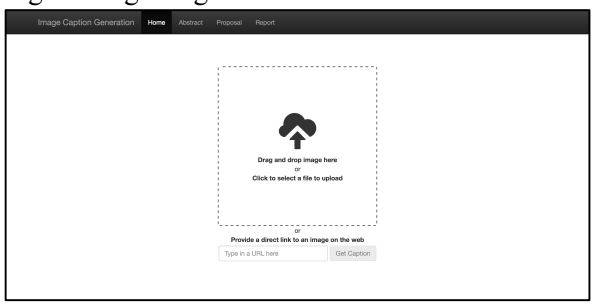

Figure 3: Uploader view

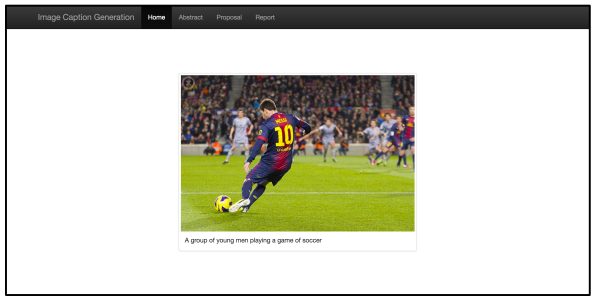

Figure 4: Captioned Image view

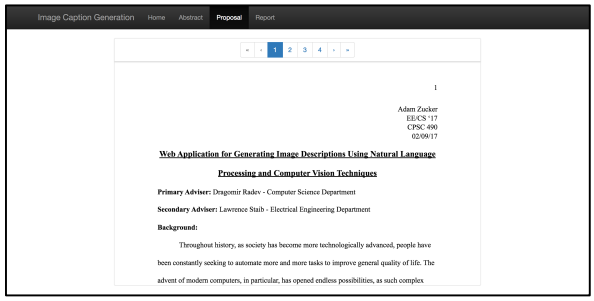

Figure 5: Document Viewer view

2.2 Design Choices: Front-end

The highly dynamic nature of this app required a front-end design optimized for handling highly dynamic views. As such, the front-end code for the app was built with the ReactJS framework, which is optimized for exactly this kind of quick re-rendering.

ReactJS eschews the traditional use of large HTML templates files in favor of modularized JavaScript objects that are compiled and rendered into HTML elements on the fly. So, what would have been a single, large complicated, pre-defined DOM hierarchy was replaced with a greatly modularized React Component hierarchy.

As for individual element styling, ReactBootstrap, which is a version of the popular Bootstrap front-end framework ported to ReactJS, was used in combination with some custom CSS. Using ReactBootstrap greatly sped up the process of beautifying rudimentary UI components (e.g. buttons, text input forms, the navigation bar, etc.), as these components all have pre-packaged CSS styling defined in the Bootstrap libraries.

2.3 Design Choices: Back-end

In stark contrast to the rendering of the website, the process of generating a caption for an image is extremely computationally intensive. As a result, the captioning process would have to be run on a server computer with a lot of computing power, which means images uploaded to the webpage client-side would have to be sent over HTTP to the back-end of the web app for processing.

So, the first conclusion reached in choosing how to structure the back-end of the web app was that it should use a framework written in a language that can easily interact with hefty machine learning libraries. Python (Python3 was used) is the optimal choice for this as the desired deep-learning library TensorFlow, as well as many web frameworks, are built with it.

The decision for which of the numerous Python-built back-end frameworks to use boiled down to which one was the most lightweight: the web app has only a single page, so a complicated routing system is unnecessary, and uploaded images are not stored long-term, so neither an ORM nor even a database system is needed. The Flask framework was thus chosen, as it is a highly configurable framework that ships without these features that have been deemed extraneous to this project.

3 Caption Generation Engine

As mentioned in the discussion of the back-end design for the web app, the caption generation mechanism requires the use of deep learning libraries. This is because most of the recent breakthroughs in CV and NLP related tasks involve the training and use of complex neural networks, which require highly specialized and optimized scaffolding code to run efficiently. Google's TensorFlow deep learning library was used for this purpose. The core algorithm used is the TensorFlow im2txt algorithm, which implements the *Show and Tell* Model (Vinyals et al., 2016).

3.1 *Show and Tell* Model

The *Show and Tell* Model was the winning model of the MSCOCO Image Captioning Challenge competition in 2015. The challenge for competitors was to come up with a system that can automatically generate descriptions for images using a large training set of pre-segmented and pre-captioned images called the COCO (Common Objects in Context) dataset (Lin et al., 2014).

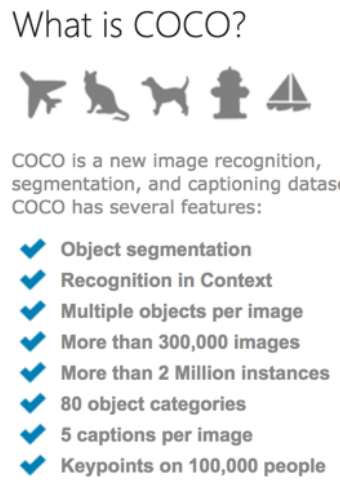

Figure 6: COCO Dataset overview (Lin et al., 2014)

Figure 7: Sample COCO captioned and segmented Image (Lin et al., 2014)

The *Show and Tell* model uses a hybrid neural network called an encoder-decoder network. The encoder half of the system is a convolutional neural network that extracts the features from the provided image and encodes them into vectors. These vectors are then run through the decoder half of the system, which is a series of recurrent neural networks called long-short term memory (LSTM) neural networks. The LSTM networks decode the feature vectors into the words of the caption (the s_i terms in Figure 8) probabilistically, one word at a time.

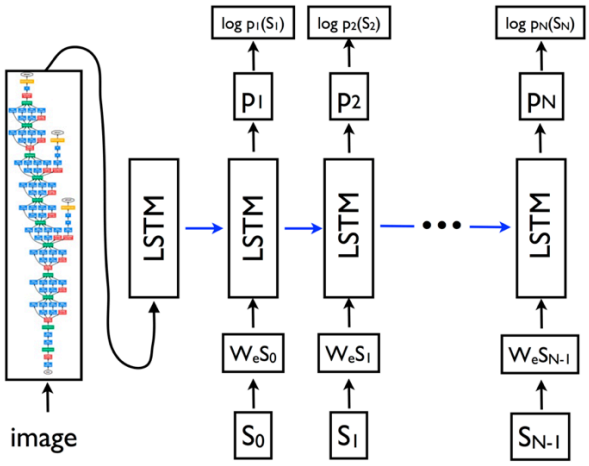

Figure 8: Show and Tell encoder-decoder network (Vinyals et al., 2016)

3.2 Captioning Mechanism

The captioning mechanism used in this project os built off of TensorFlow's implementation of the *Show and Tell* model, im2txt. For the im2txt algorithm to work, its neural network must first be trained on the MSCOCO data set. There are two required phases of training, an initial phase, which performs 1 million steps, and a fine-tuning phase, which performs an additional 1-3 million steps. When the model is fully trained, captions can then be generated for an image via a shell script.

```
INFO:tensorflow:Successfully loaded checkpoint: model.ckpt-2000000
Captions for image tmp1r618gb8:
  0) a woman sitting at a table with a plate of food . (p=0.005372)
  1) a man sitting at a table with a plate of food . (p=0.000980)
  2) a woman sitting at a table with a plate of food (p=0.000828)
```

Figure 9: Sample output from im2txt shell script

For this project, a model that went through 2 million steps of training (initial training plus 1 million fine-tuning) was used to caption the images that were received from the web app. The captioning process works as follows:

a) Flask receives and image through HTTP and sends it to the captioning engine
b) The captioning engine uses the Python Image Library to open and save the image in a temporary file
c) The captioning engine runs the im2txt captioning shell script on the saved image file in a forked process
d) When it completes, the im2txt shell script's stdout is piped back into Python
e) The temporary file containing the image is closed and deleted
f) The captioning engine selects the most highly rated caption and sanitizes it
g) The sanitized caption is sent back to client via HTTP for rendering

Figure 10: Sample captioned image A

Figure 11: Sample captioned image B

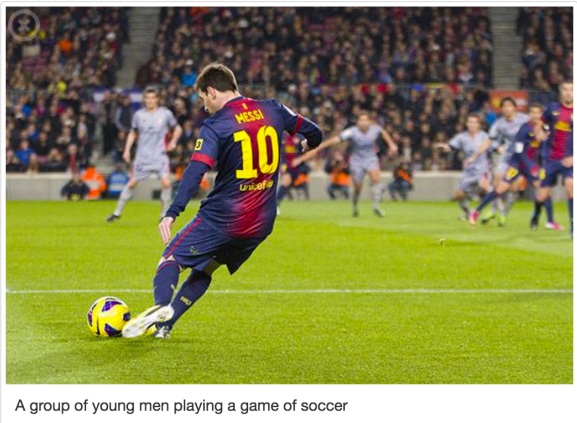

Figure 12: Sample captioned image C

4 Algorithm Analysis and Experimentation

The im2txt algorithm on the whole yields some excitingly accurate descriptions, but they are by no means perfect. This section of the report attempts to pinpoint some of the issues and limitations the algorithm encounters and then provide potential ways to improve it.

4.1 Misidentification

The most common problem im2txt runs into is the misidentification of features. In Figure 13, for example, the algorithm seems to believe the person and/or rocks at the bottom constitute a boat, which is clearly not the case.

Figure 13: Caption with a misidentified feature

In this particular example, along with many others, it is plausible to see how a computer could misconstrue one object for another based on the shapes of the items (e.g. the man appears suspended on the water, much like a boat would be).

Unfortunately, regardless of this plausibility, this issue is probably the most difficult to correct, as it lies within the encoding part of the model, which is almost entirely determined by fine-grained details within a neural network. The best fix for this issue would probably be more training, but even that is not guaranteed to fix all problems (and could actually make things worse through overfitting).

4.2 Hypernym-Hyponym Problem

Another common but perhaps more readily fixable issue is that of the identification of "specialized objects." This is readily apparent in Figure 14, where the algorithm identifies "fruit" in the bowl, but not "strawberries" (the general term: "fruit" is a hypernym of "strawberry", which is in turn a hyponym of "fruit")

Figure 14: Caption with hypernym/hyponym problem

This issue can perhaps be viewed as a special case of the general misidentification problem, but the key defining characteristic of this problem is that it lies with the dataset. It is wholly feasible that the dataset, despite having many pictures of fruit, has no pictures of strawberries. Thus, the algorithm has no way of identifying a strawberry as a "strawberry," because it cannot possibly identify something it has never encountered before (in fact it is probably a testament to the success of the algorithm that it can even identify the strawberry as a "fruit").

4.3 Possible Fix for Hypernym-Hyponym Problem

In this project, an additional layer of caption processing was experimentally added to the captioning engine to tackle the problem of specialized objects. Instead of deleting the temporary file containing the image immediately after sanitizing the im2txt output, the image captioning engine leveraged a third party image analysis Python API provided by Clarifai to improve the caption.

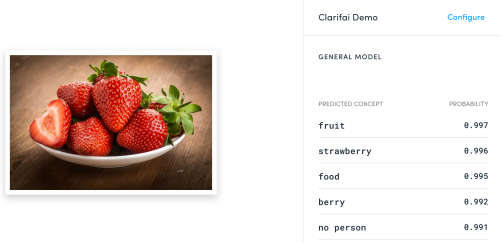

Figure 15: Web UI for Clarifai API

Clarifai's API identifies specific objects, which it calls "concepts", in images, without necessarily constructing a holistic description for the image. The algorithm used this list of identified objects to attempt to replace non-specialized hypernyms with specialized hyponyms: the post processing algorithm obtained hyponyms for each of the

nouns in the sentence using the Python nltk.wordnet library and cross referenced Clarifai's returned list of concepts for these hyponyms. The hyponyms of highest probability in the concept list then replaced its corresponding hypernym in the caption.

Figure 16: Caption post-processed to alleviate hypernym/hyponym problem

In Figure 16, for example, the word "berry" has replaced the word "fruit" as according to nltk.wordnet, fruit has a hyponym "berry."

This post-processing layer did not fully solve the problem as hoped: "berry" is still not as specific as "strawberry". Additionally, this error shows the limitations of the nltk.wordnet library ("strawberry" was not returned as a hyponym of "fruit"; perhaps another recursive level of post processing could replace "berry" with "strawberry"). Nonetheless, barring the grammatical agreement issue in the caption, such a change could be considered an improvement upon the first caption.

However, this fix creates new issues besides grammatical agreement ones in the form of false positives.

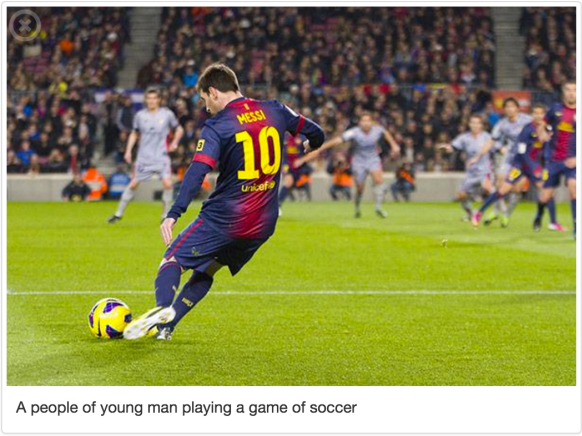

Figure 17: Post-processed caption with erroneous hyponym replacement

In Figure 17, the original, accurate captions of "a group of young men playing a game of soccer" has been obfuscated with needless hyponym replacement. As a result of a proliferation of captions like these, this post processing was ultimately disabled in the final product.

5 Conclusion

This project has presented a user-friendly web application that can automatically generate captions for images. Additionally, experimentation with the tool has provided some insights into some of the issues the current deep learning approach faces, and some possible ways of fixing them. In truth, the latter proved to be quite a difficult proposition; truly improving the system (beyond adding more data) would probably require more precise refinement within the actual neural networks that implement the algorithm. Computer vision and natural language processing remain extremely active areas of research, so there is definitely reason for optimism that such algorithms will improve. If nothing more, this project has been a fun attempt at tackling a challenging problem.

References

Oriol Vinyals, Alexander Toshev, Samy Bengio: "Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge", 2016, IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: PP, Issue: 99, July 2016), arXiv:1609.06647

Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick: "Microsoft COCO: Common Objects in Context", arXiv:1405.0312, 2014