Final Project Report: Comic Scraper (December 2019)

Chase Brunette-Pak, Casey Lemon, Noah Branch, and Sai Manikonda

I. Objective

The main goal of the project is to help users, who are comic readers to hop into a series without knowing enough background knowledge of the characters and their relationships within a select series. It is designed to search for a specific character in a comic series and create a biography for that character based on their overall dialogue in the series and behavior. Also, it is programmed to scrape the data from a comic website and to recognize, analyze all the available data from the images through different machine learning techniques and open-source Python libraries that most fit the requirements of the project. The application is built using different programming languages and to get familiar with machine learning algorithms.

II. Data

To help achieve the goal of the project, there needed to be a sufficient dataset to work from. Although no previous research was done for any existing datasets that suited the project needs, the team opted for building a personal dataset, mostly to be able to gain experience in actually sourcing and creating a dataset. Gathering the data was done via existing web repositories of comic books and their series and the separate pages of each comic. The specific web scraper built for this project was only programmed to function correctly with one online repository (https://readcomicsonline.ru/), but it could easily be adjusted and modified to be able to source data from other repositories. When the scraper runs, it waits for the user to enter a specific URL to scrape from which would be the URL to a certain comic book series. The scraper would then visit each issue in the given series and grab and save the image each page in the issue. The data was organized in a hierarchical format in order to separate out the series, each issue, and then each page.

Moreover, the scraper was built in certain specific manners so that whatever network it was being run on would not be IP banned from the repository the project was utilizing. The two main aspects of the program were the processing of only one URL at a time and random time delays. Only having the user of the scraper specify one URL would prevent the scraper from visiting every comic book and every issue belonging to the repository. Also, it allows the user to focus the dataset on only certain characters or comics. Also, random time delays were implemented into the scraper. The random time delays were also used to slightly slow down the process and hopefully prevent overloading the repository's network. The random time delays would occur between visiting each issue and between each page in each issue. The scraper would pause for a random duration of time, usually between half a second to three seconds.

Once the data was collected it was manually labelled by us using the 'Label IMG' tool. Our dataset was a collection of eight DC comic series, with over 20 comic issues that were spliced for training data for our objection detection model. Our model sought to analyze an entire comic, however due to certain limitations and issues we faced we had only been able to detect dialogue between characters within a comic. To create it we had used Google's TensorFlow as our main library to help train the model. We had started our training locally on a laptop without a GPU. This drastically increased our training time due to our CPU limitation and was one of our biggest issues of the entire project.

A solution we found to decrease our training time was using Google's online Collab's to train the model because they gave the option of using a free GPU in their settings to help train models like ours. However, when trying to transfer our code and data to the application, we were not able to compile our code due to certain dependencies. We attempted to troubleshoot this issue, however due to time constraints we went with continuing training our model locally with our GPU limitation.

To help train our model we had initially used the 'ssd_mobilenet_v1_coco' model to help start the training process. The reasoning we had for using this specific model was due to the capability of running our processes with our limitations. While training however, we had to reduce our batch size to image per training which in turn had expanded the amount training time again. After developing our first model we knew our amount of training data was insufficient and/or are model we built upon was lacking.

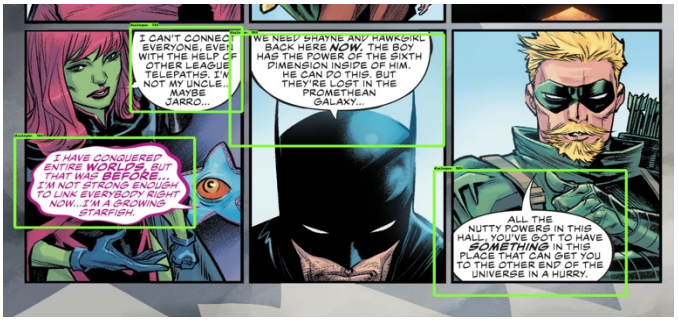
To improve upon our model, we had decided to go with the 'ssd_mobilenet_v2_coco' model to help increase our training time. We found this out by reading the documentation between to two and discovering that training with the newer model trained objected detection models more quickly than the previous versions. We had also increased our data set and relabeled the data to help better train our model. Once the retraining had completed, we had a loss of roughly 5-8% (92-95% accuracy) which we found to be reasonable as the previous iteration had a 12-14% loss (86-88% accuracy). We would continue training; however, the deadline had been drawing close and we decided to leave it as is to help work on other parts of the project such as the image analysis.

As previously stated, our model had been only able to detect dialogue within a comic. This limited us in what we were able to analyze through the comics. However, we were able to use the model to crop the images that would then be sent to another program. This program would scrape any form of text from a given image or images in the comic issue using Google's Tesseract-OCR and PyTesseract. Tesseract-OCR is an optical character recognition engine that recognizes characters in images and is the most-accurate open-source engine. The engine can recognize over 100 languages and can be trained to learn a new language. In addition, PyTesseract is wrapper that is created around this system for Python. After scraping the data from the images, the program will be able to detect certain moods or tones that were being portrayed in the image using an open-source, Python library, Text Blob.

Text Blob can process textual data and has a consistent API that can perform many Natural Language Processing (NLP) tasks such as noun-phrase extraction, language translation, sentiment analysis, classifier and more. For this project, we decided to perform sentiment analysis on the scraped text to analyze the overall tone of the dialogue. The sentiment analyzer returns a tuple of the form polarity, subjectivity as the parameters. Polarity has the range of [-1.0, 1.0] and subjectivity of [0.0, 1.0]. So, the program either returns the overall tone as neutral, positive, or negative of the dialogue. From there we were able to start detecting whether certain characters liked or disliked one another based on their text. For instance, Batman would have an aggressive tone towards the Joker.

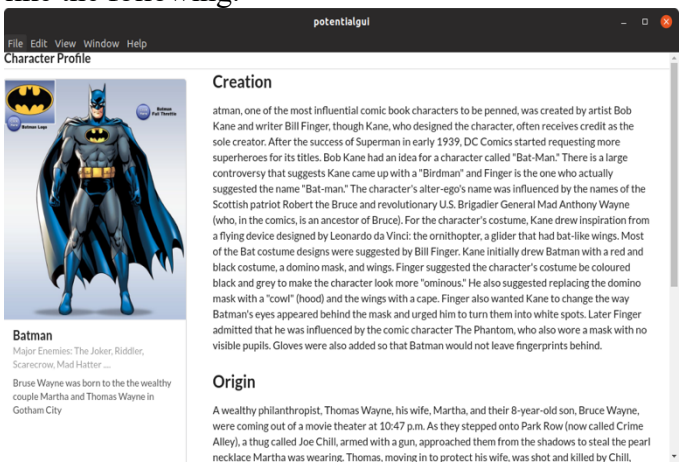
This all is expanded in our Graphical User Interface. In the interface the data gathered would be represented in a presentable way to the user. For instance, the user can search a certain series and character and the user interface would pull up relevant images and information scraped throughout the series. Based on the users searches we also added a table for recent searches so the users could go back and see what they've previously searched. In the future we could turn that table into more relevant searches table as the user may want to see more information relating to a specific character.

III. Results



Above we can see what happens once we detect dialogue and those boxes would essentially be sent so Text Blob an actually process the words from the images. It wasn't always accurate but for the most part it was able to pull data. Our detection module had an average loss at around 5%.

The Final project would have looked something like the following.



As stated above, we can see how the user would search for data and how it would be displayed.

IV. Primary Issues

The main issue we had since we did not fully complete what we intended, was getting the data represented in the Graphic user interface. Because we could not get the data represented in order to present in the GUI, we have templated out what the represented Data should look like.

Another issue was simply merge conflicts with work on the GUI and adding dependencies. Certain commits would try to commit my node_modules folder which is too big to commit and GitHub was not reading out .gitignore file So we had to find a way around that.

V. Future Work

Moving forward, there are a few aspects of the project that need to be visited again or improved on. The team came to a consensus that the model could be retrained so that it was more accurate. Regarding the model, it could also be improved as right now it is able to detect speech of characters within the data, however it would be ideal to be able to detect the actual images and whether or not there are different characters present or if specific actions are occurring on the page. The model could be improved upon by being able to detect whether or not a villain or hero is present or speaking or be able to detect if there is fighting occurring.

Also, the actual gathering of the data could be vastly improved upon. The current state of the scraping of data is sufficient for initial stages of the project, however it is slow, time consuming, and limited in scope. The first aspect of the scraper that could be modified would be to add the ability to target different repositories instead of just one. Also, while one comic at a time and random time delays in the scraper are sufficient, they are very time consuming and not efficient, especially when trying to grab large sets of data from one source. So, changing the functionality of one comic at a time to being able to process multiple at once would vastly improve the performance of the scraper and reduce the time it takes to gather data by a great amount. To improve upon the efficiency further, instead of simply using a random time delay, it would be ideal to utilize proxies for the program. By utilizing proxies, it would not matter whether or not data source or repository had banned an IP since the program would be able to simply grab and use another one and continue working.

Lastly, implementing the GUI to be fully functional with the models, character descriptions, and more would be necessary moving forward. In its current state, the GUI does not function completely with all of the aspects that were decided at the beginning of the project. Also, it would be ideal for the GUI to display our data instead of external sources. Utilizing external sources was necessary mostly for development of the GUI, however it was also needed since the models and the gathered data was not ready or complete by the time the GUI was implemented.

VI. Organization Chart

Casey Lemon: Data Scraper

He was responsible for creating the data scraper and develop an efficient way to gather data for the

project. This involved researching different web scraping libraries for python and different ways to scraping data from websites in large volumes. Moreover, the scraper needed to be efficient and be able to grab data in a timely manner.

Chase Brunette-Pak: Object Detection Model

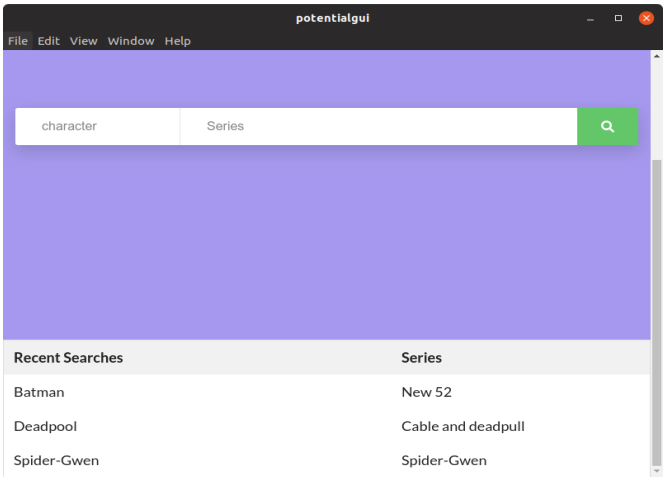
He was responsible for setting up the custom dataset for DC comics, deciding on pre-mode, training the object detection model, and improving upon the model until the deadline.

Sai Manikonda: Text Analysis

She was responsible for researching about the python libraries that are required for the text analysis of the project. She implemented a scraper to scrape the data from the images and performing sentiment analysis on them. It is programmed using Python's open source libraries, Google's Tesseract OCR, Jupyter Notebook, VS code, Git, and GitHub.

Noah Branch: Graphical User-Interface

Responsible for the GUI, and for wrapping everything up and displaying it to the user.



VII. References

Comic Repository: https://readcomicsonline.ru/

Google Collab: https://colab.research.google.com/notebooks/welcome.ipynb

Google Tesseract-OCR: https://opensource.google/projects/tesseract

Label IMG: https://github.com/tzutalin/labelImg

Text Blob: https://textblob.readthedocs.io/en/dev/