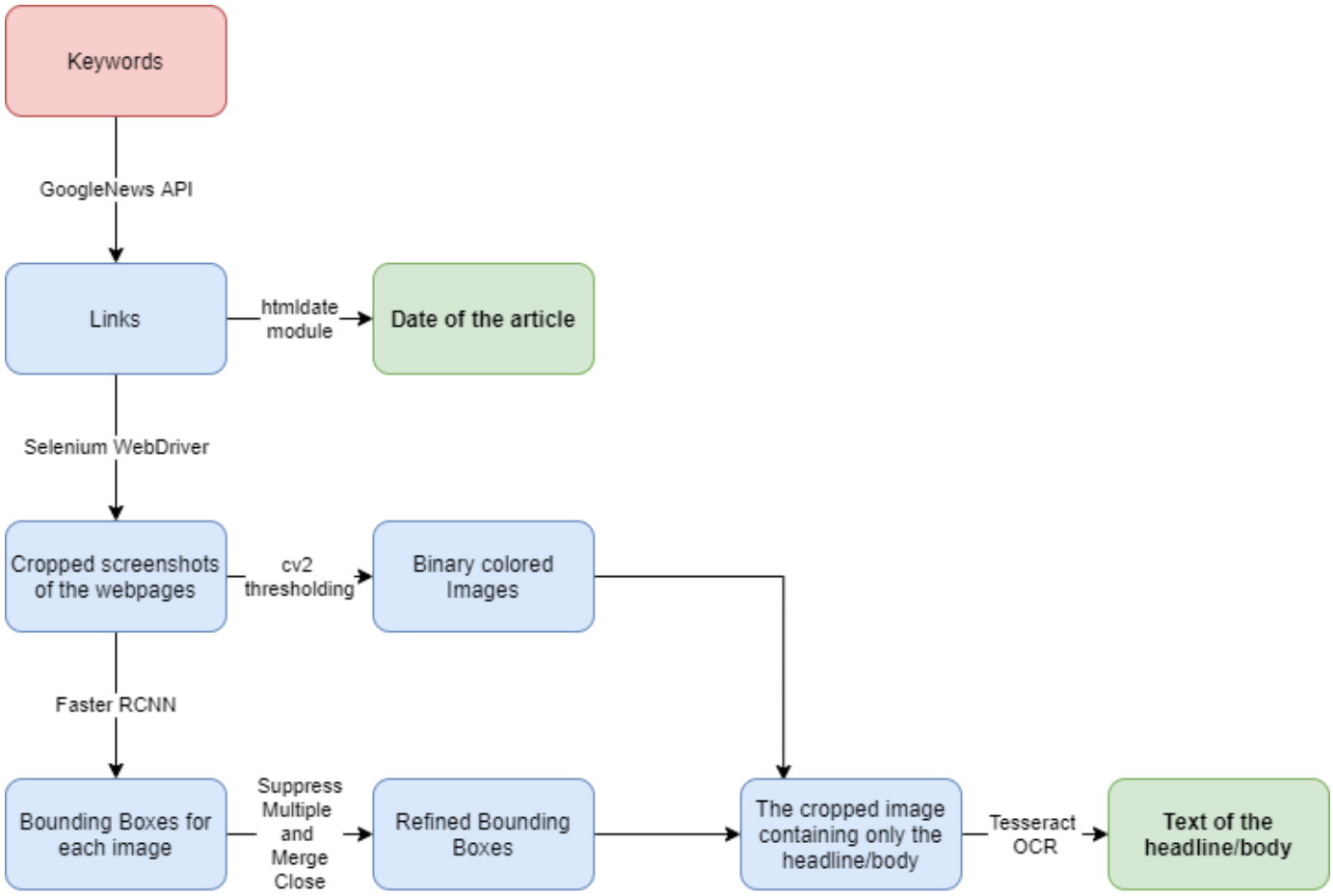
Headline and Body Detection in Web Articles

Hardik Siloiya

Abstract

This paper is the documentation of the project for detecting headlines and body text from images of articles which are acquired by the respective links of the articles. A convolutional object detection approach has been used for solving this problem which has produced accurate results.

The general pipeline which the program follows is given below:



Looking at the individual components:

Getting the links from the keyword:

Here, I have used a python module GoogleNews which fetches the links from a string which contains all the keywords. The time taken to fetch can vary depending upon the max number of links set in the function.

Screenshotting the page from the links:

For getting the screenshots from the link, a selenium chrome-driver is used. Thus for this to work the Chrome browser should be first installed on the system.

There are several issues with this method, most prominent being the presence of ads and popups which appear after the site load that cover/blur out the entire webpage. To deal with this a manual strategy of handpicking the elements of the close button of those popups and then getting their xpath. This xpath is then pasted into the code for the screenshot function where the selenium driver clicks on this button and removes the popup. An adblock extension (uBlock Origin) is also used, which prevents most sites from popping up ads and alerts. Then we get the height by scrolling the page through the driver, which gives us the height of the webpage and then we set the window size of the driver to be 1920 x 'height obtained' to capture the entire page.

The screenshot is then split into images of height 1500, to get accurate detections from the model. This is because we had trained our model on cropped images of size ~ 1920 x 1400. Thus this function returns the final list of cropped images.

Prediction step:

Now with our list of images obtained, for each image we get a prediction from the object detector model by using a Predictor object. Here the framework used is of Faster-RCNN which is implemented by the Detectron2 model made by Facebook. The weights used for training the model can be adjusted in the get_predictor function which returns the predictor object.

Optimizations:

Suppress multiple and merge_close are two functions which are used to remove the intersection between two boxes and merge two boxes which are very close respectively. Suppress multiple is used to remove the error caused wherein two boxes sometimes overlap and the same text is read by the OCR in both the boxes which leads to decrease in the accuracy.

Merge-close is used to merge two boxes which are very close and are similar in their shapes so that if a line in between the two boxes gets missed or if one of the boxes' edges has cut through some line then, we can get the line back. Merge Close has tunable parameters.

Post Processing:

After running the predictor on the images we get a set of bounding boxes. We can directly crop out the regions of these bounding boxes from the image and use the Tesseract OCR for getting the text but the color of the text in the image can be a hindrance, sometimes being very light in

color which often gets missed by the OCR. Thus, to improve accuracy, I have binarized the image using the threshold functions from the cv2 module. This changes all the colors in the image to black or white, and the threshold is set very high so that even the light text can be scaled down to 0 (black). This greatly improved the accuracy of the OCR model.

Heuristics deciding the headline:
The first headline captured in the list of images for a particular is set as the headline of the article. The subsequent headlines are ignored.

The date is fetched using a python module htmldate, which searches for the date heuristically in the html source code of the link. Thus, this requires only the link of the article.

Some results:
The final Faster-RCNN framework used, has been trained for 3200 epochs achieving a mean Average Precision (mAP) of 90.38% on the test data.

Metrics	2500 epochs	3200 epochs
mAP	86.95	90.38
Mean body similarity	71.33	70.56
Mean headline similarity	73.76	76.83

Results of the final model weights used in the App

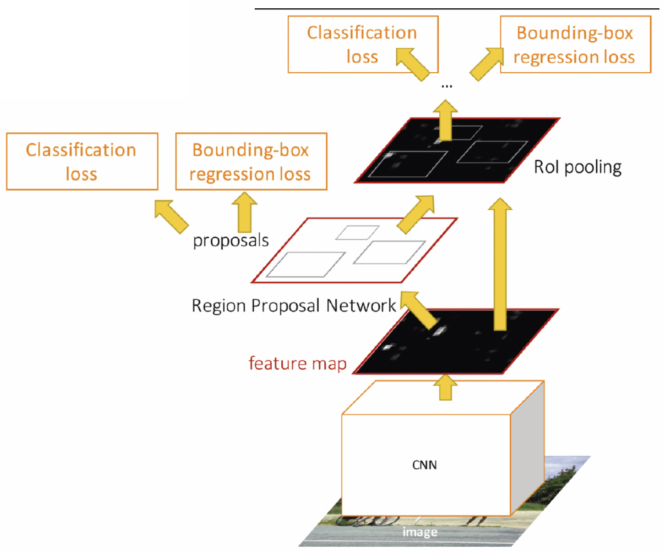
4 Fold Validation for Faster RCNN:

Sr. no	Headline (AP)	Body (AP)	Last training loss	mAP at IoU=0.5	mAP at IoU=0.5:0.95
1	56.872	75.170	0.086	82.82	66.02
2	58.959	75.001	0.123	84.99	66.98
3	57.792	74.502	0.133	84.78	66.14
4	62.451	73.896	0.123	85.95	68.17

Working Details of Faster RCNN:

Faster RCNN was implemented as an object detection framework for fast real time detection of objects in images. This network builds upon the working of its ancestors, the RCNN and Fast-RCNN. RCNN originally used selective search to get the regions of interest in the image and then passed these regions into a classifier which detected if it was an object of interest or not. Since the selective search part ran on CPU and was non-trainable, RCNN was quite slow in training and testing. Then, came the Fast-RCNN which improved the performance of RCNN by allowing resources to be shared among the classifications in each of the regions proposed by the selective search. This was done by projecting the region on the feature map of the image and using this projection for further computations in each of the regions of interest instead of running the entire classifier from start on the cropped region. But even in Fast RCNN the selective search could not be trained and the pipeline was split.

In Faster RCNN the selective search is replaced by the region proposal network, which performs a similar function to it and outputs the regions in the photo which contain an object. This is then followed by the later part of the Fast RCNN which classifies these boxes and refines them.



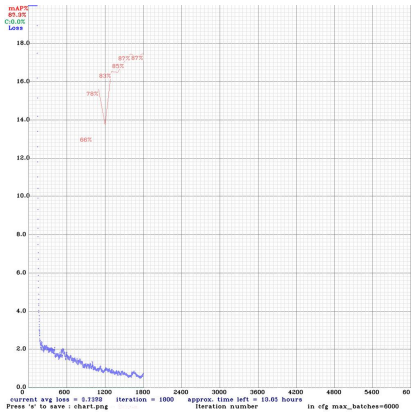
The pipeline followed by the Faster RCNN
(Note that the Region proposal network does not give the outputs of the bounding boxes directly, but the offsets to the pre-existing anchor boxes.)

Some previous approaches used:

YOLO Object detection Network

YOLO algorithm was used which gave some good results after training for an extended period of time but due to its nature of only going through the image once and predicting

on the go, it was outperformed by the Faster RCNN. Although it should be noted that YOLO is quite faster than the Faster RCNN model.



Yolo final mAP = 85.52% at 0.5 IoU

An alternative using Image Processing for headlines

Since the headlines always seemed to follow a structural pattern in the articles, a hardcoded image processing algorithm was tried. I first converted the image to binary coloring to help in the text detection. Then, I used the cv2 library (OpenCV) to process the image.

I used the find contour functions to get the same pixel intensities to be clubbed together into a line. Since our image is black and white here, with the background being white and text is black so all the alphabets will form the contours since they are a bunch of connected pixels and are black. Naturally, all the black parts in the image will form a contour.

So, now we need to remove the images.

Ads/Pictures usually occupy a large area in the image so I filtered them using a threshold (like remove all contours with area more than x) and most of the large images got filtered.

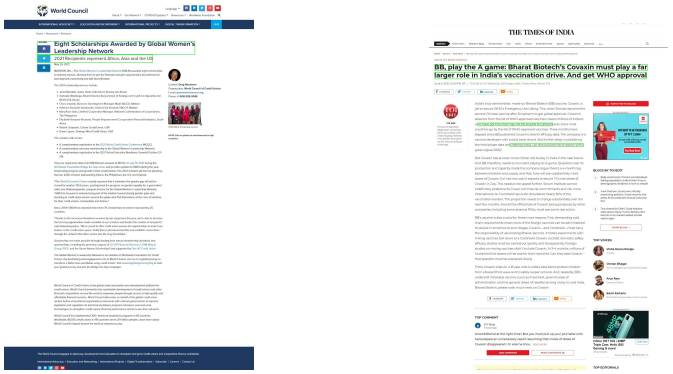
Now the body is also a hindrance so I used a height threshold to filter contours as Headline letters are larger than that in the body.

Now after all this, quite a bit of text in the body, ads , etc. is still not filtered so I could not apply an OCR directly.

A Run length smoothing algorithm (RLSA) was applied on the image to smoothen the letters and sort of merge them into a line.

This connected the words in the headline forming a larger contour of sorts. Now, to get rid of the small body contours which also got till here, I used a width filter on the result. The headlines will be covering a large width in the image while the body words will be scattered around and miss the threshold width for the entire contour.

So now, only the headline remains and we draw bounding boxes on the original image to obtain the required annotations or we could also use an OCR on the filtered headline image to directly get the text



Some results of the contour finding approach

Naturally, as there were a lot of hyperparameters and hard-coding involved here, the mAP of headline predictions was

quite low on the entire dataset with mAP@0.5 being only 9%.

Some interesting observations:

Since we use a CNN model for getting the bounding boxes, our network cannot understand what is actually inside the boxes it predicts due to the inability of the network to read. Hence, it was noticed that image sizes had a considerable effect on the accuracy of the predictions. Too large input images (ex: greater than 1800 pixels in height) can lead to very low accuracy predictions because the images get resized to match the input size of the network. Thus the resized image gets distorted. Similar case for very small cropping sizes where the correlated structural properties of body and headline are lost.

Conclusions:

Moving forward, the Faster RCNN seems to be giving quite good results but the minute details can often be missed because of some edges cutting through the lines. Also if the shape of the body gets distorted a lot by images and ads in between, then the prediction quality for the body also decreases. The strategy for improving the whole model can be either through getting better bounding boxes or better text.

- If we want better predictions from the Convolutional Net, the first step can be to gather better data for training the model.
- Till now we have trained it on normally structured articles with a few articles having exceptions.
- Training on articles having irregular body and headline shapes can provide some robustness to the model for identifying distorted bodies at test time.
- Increasing the dataset size can also be helpful.
- Decreasing the width size of the images for training upto 1366 and reducing the heights of cropped images to 1200 can be an option.

For Getting better text output, an NLP strategy can be employed for correcting errors and filling in missing parts which are missed by the Faster RCNN. The model to use here can be a **Generative Adversarial network** for Text Generation.

References:

- [1506.01497] Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (arxiv.org)
- chenyuntc/simple-faster-rcnn-pytorch: A simplified implementation of Faster R-CNN that replicate performance from origin paper (github.com)
- facebookresearch/detectron2: Detectron2 is FAIR's next-generation platform for object detection, segmentation and other visual recognition tasks. (github.com)
- R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms | by Rohith Gandhi | Towards Data Science
- Guide to build Faster RCNN in PyTorch | by AI@Scale Research Group | Medium
- Implementing RoI Pooling in TensorFlow + Keras | by Jaime Sevilla | xplore.ai | Medium
- The Selenium Browser Automation Project :: Documentation for Selenium
- htmldate · PyPI

Appendix

Data Collection and Annotation

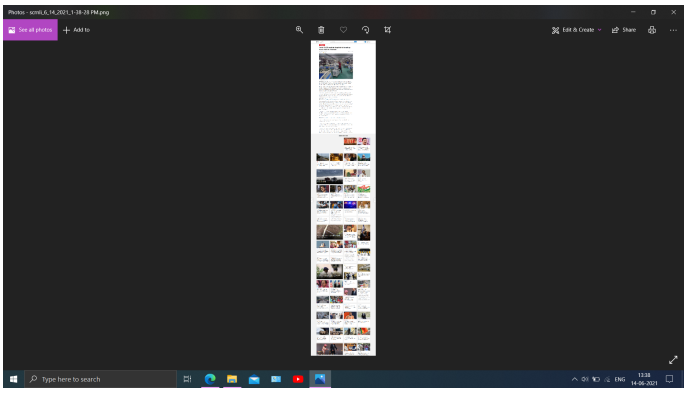
Steps for building the dataset for screenshots:

1) Download the chrome extension Scrnli for screenshotting the entire web page - [Scrnli Screenshot & Screen Video Recorder - Chrome Web Store (google.com)]
2) When on the news article site, click the extension and select capture the whole page. Then another tab should appear and select Download as Image in that.

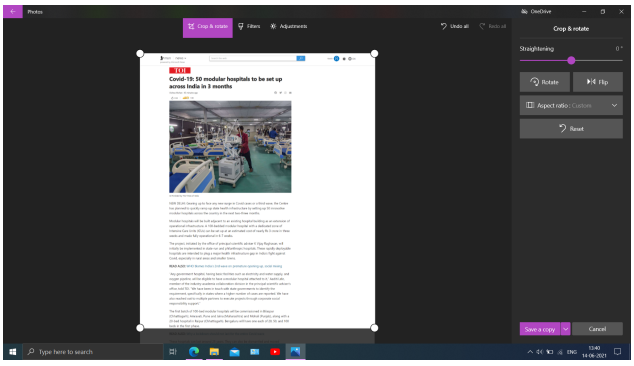
3) Follow this process to get images of all the webpages.

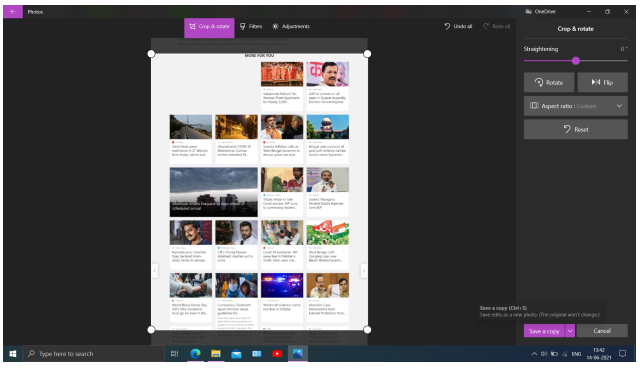
4) Now the screenshots taken directly from the extension are very large in resolution. So we need to crop them and split them into 3-4 parts.

5) Navigate to the images in the directory where they are downloaded
6) The images should look like this:



7) Now for each image crop it into 3-4 parts until each of the parts are readable by just opening it without zooming





8) You can also crop the parts which don't have any information and are filled with ads and images too.

9) Remove the original large image or move it to another folder after it has been split up into 2-4 smaller images.

10) Then you can continue the labelling process for all these split up images.

Steps for Labelling Images:

1) Install python Download Python | Python.org

2) Open command line and run the following: i) pip install PyQt5 ii)pip install lxml

3) Run the following cmd as well on the command line - pyrcc5 -o libs/resources.py resources.qrc

4) Go to this github site and download the code as zip (green button) : - tzutalin/labelImg: 🖍 LabelImg is a graphical image annotation tool and label object bounding boxes in images (github.com)

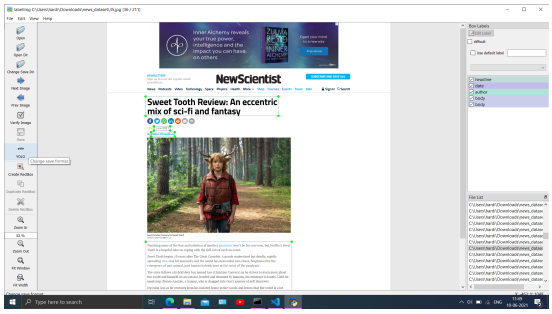
5) After downloading extract the zip file

6) Open up cmd again and navigate to where the folder is using cd commands (Usually - cd Downloads -> cd LabelImg-master

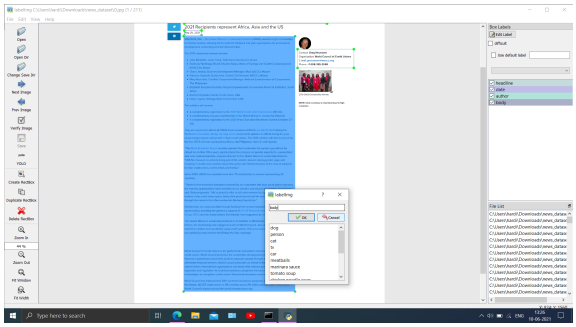
7) Now use python LabelImg.py to run the application

8) The application will pop up and you can navigate to the directory by selecting the open directory, and open the folder containing the photos.

9) (IMPORTANT) Then you must select the YOLO format option in the save formats on the left menu.



10) You can now begin drawing the boxes by clicking the create Rectbox button and draw the bounding boxes for the various objects and give it the appropriate label.



11) After finishing labelling the current image, hit CTRL - S to save the txt file for the bounding boxes (Make sure the file saved is a text file NOT a xml)