Unstructured Data Analysis
Final Project

Thomas Ribaroff
02154056
22 December 2022

Table of Contents

Introduction and Problem Statement

Outside of Machine Learning, films are my greatest passion. Over the early years of my childhood, my family collected a vast DVD collection, which now sits idly by while we use Apple TV, Netflix and other similar apps. This DVD collection was an integral part of my growing up. I wanted to explore how I might renew its' utility.

Using the tools I learned in the Image Data section of our course, I wanted to explore the beginnings of Computer Vision. Could I scan these DVD cases and manipulate the images to build a database of what films we owned? Perhaps this could save me searching the whole collection of DVDs when a film I would like to watch does not appear on Netflix.

After some preliminary exploration of the data, I pursued the following Problem Statements:

1 - Isolate individual text characters *using contours*

2 - Scan images of DVD cases and extract the text *using pre-trained OCR models*

3 - Improve the word recognition *using filtering and smoothing on noisy images*

Data Selection

I photographed a selection of our 25 DVD cases to practice these techniques on. Each image was saved as a JPEG, at approximately 2MB. I chose my 5 favourites for a total dataset size of 10MB. JPEG's are stored in a compressed format and can be read into a Python environment using various different libraries. In this analysis, we use OpenCV and Skimage at different times for convenience. When read into our environment, the images are stored as Numpy Arrays (3-D for colour, 2-D for grayscale), where each element represents pixel intensity.

Images were also chosen for their variation in noisiness, colour and font size. Some have clear text, while others are clouded with images superimposed on the backgrounds, making the analysis more complex.

Methods, Analysis and Results

Isolating Individual Text Characters

We begin by importing our first image, the cover of the 12 Angry Men DVD. Some of the text is clouded by superimposed images, but the title has a clear background. I use the **Contours** tool from the CV2 Library. Firstly, we preprocess the image, using a Gaussian Filter and thresholding, to make our letters stand out more clearly for the algorithm. Then, we find the Contours and search through them, picking the largest which represent our letters. We also have to ignore those contours which represent our letter interiors by referring to their hierarchy (see explanation below). By drawing bounding rectangles around them, we can easily save the letters as individual characters for future analysis (Fig 1). This is where we run into our first problem - the contours are not found in the same order in which we read letters (left to right on the image), thus saving them is not in a useful order.

Fig 1 - 12 Angry Men DVD Case, with the title bounded by rectangles in green

Contours in CV2 - an explanation

Contours are the continuous lines that make up the boundaries of objects in our image. They can also be the curves along which an object in an image is split.

Contour lines that are found inside other contours are called 'child' contours (as opposed to parent). These relationships are stored in a tree-like structure, so we can analyse how contours are related to each other.

Contours are found using the Suzuki algorithm, which scans a point on an image from all sides, finding changes in pixel intensity, and follows the edge of an object until it loops back to its starting point.

To solve the ordering issue, we can loop through the contours twice. Firstly, to save the centre of each bounding box in a list (which we then sort from left to right). Secondly, to save the bounding box images with indexes in the order in which they appear in our sorted list. I used this technique on the Poirot DVD cover (Fig 3) and the six letters of POIROT were saved in the exact order for us to easily use in future work (Fig 2).

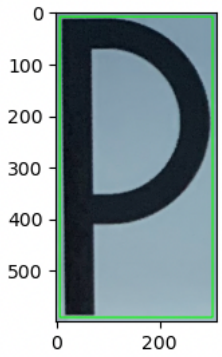

Fig 2 - Our first letter (saved with index 0, from our sorted list)

Fig 3 - Poirot DVD Case, with the title bounded by rectangles in green

Saving our characters as individual letters might seem unnecessary given that we could save titles as a whole. However, when we are struggling to uncover what the text in the image says, having individual characters proves more useful. Here, we can plug our individual characters into pre-trained CNNs to predict which letters they are, rather than trying to sort for complete text on a whole image, as shown below.

Extracting DVD Titles as Complete Strings

Next we explore the **pyOCR** library, which gives us tools to uncover text in our images and convert them to strings. We import the library, set the language to English and allow it to scan the image as a whole for text. When given the whole DVD case for 2001, it correctly identified the text of the title, but with numerous spelling errors.

A great alternative to this library is using the Tesseract library which is another open-source OCR engine. I chose not to use it on this analysis as it is more powerful

but more unwieldy for small projects. My code (see GitHub) could easily be adapted for use in this library.

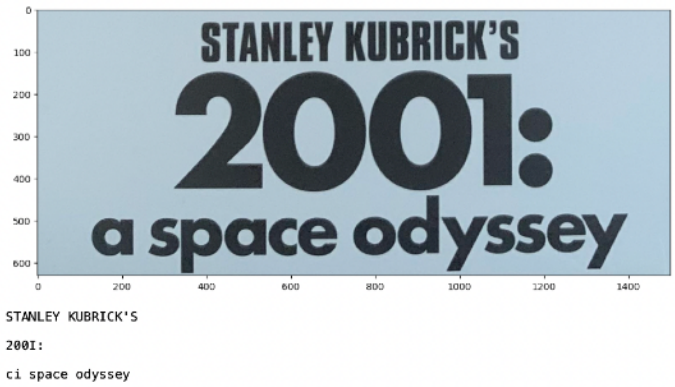

Fig 4 - Our 2001 DVD Cover, with the predicted text of its title using the pyOCR tool. Clearly a good but imperfect tool.

Image to String Tool in pyOCR

The image-to-string tool in pyOCR is an Optical Character Recognition. It uncovers text in images and converts it into usable strings.

The algorithm the pyOCR uses to 'see' the text is determined by the implementation, but it is limited to SVMs, Neural Networks and some Tree-Based Methods.

It is important to note that the pyOCR image to text tool performs some of its own pre-processing of images, like filtering and thresholding. However, this is done generally over all images and therefore isn't image specific. We can improve the performance of the tool with domain specific knowledge, like using more relevant filters.

Improving our Word Recognition

The font and the format of our images are making the job of our tool harder, as is our image noise. The pyOCR tool is built to work on clear, black and white images with high contrast, hence why the first attempt on the 2001 case was quite successful. However, on the Casablanca DVD case (Fig 5), it is not so successful. So, I followed the following preprocessing steps to improve the outcome of our analysis.

1 - Convert image to RBG

2 - Threshold image, to create a black backdrop with high contrast for letters

3 - Smoothed image noise using a Gaussian Filter

4 - Alternatively, smoothed imaged noise using a Bilateral Filter

The Bilateral Filter is highly effective at removing noise while maintaining edge sharpness. This is key if we want to preserve letter structure, so I expect that this filter might give us some modest improvement. This improvement is confirmed in Fig 7 below.

Fig 5 - An improvement using our preprocessing steps. The '1' was recognised as a number, rather than the letter 'I'.

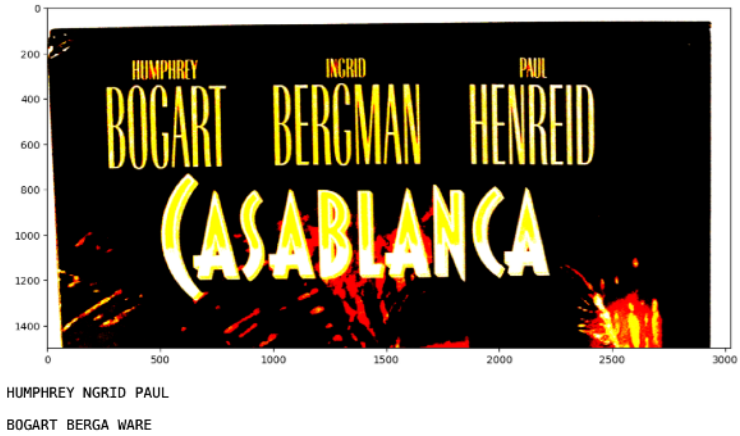

Fig 6 - We are able to uncover the names, but not the title on the Casablanca DVD

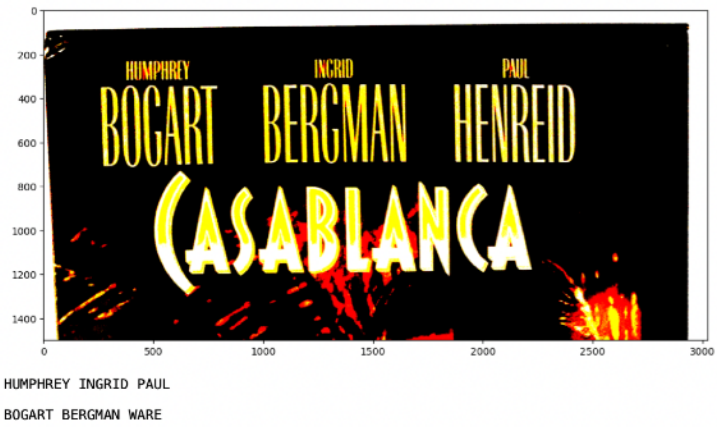

Fig 7 - A further improvement using Bilateral Filter instead of the Cauchy - fewer typos

Unfortunately, given that the font for "Casablanca" is so abnormal, our algorithm has trouble classifying it as text. This is where our contour letter isolation method could come in handy when building a pipeline to uncover text. If this first technique fails, we could revert to using the other one, which would treat these letters as individuals. This is a similar process that the EAST text detector from OpenCV follows - the

state of the art text detector deep-learning model. This project has been a great introduction to how a powerful Computer Vision product like EAST works. Exploring the intricacies of EAST, as well as that of Google's ViT-G/14, would be excellent candidates for further work.

Another problem we run into is text orientation. If the words are not in a straight line, then these tools are much less effective. Returning to our 12 Angry Men DVD cover, the image does not have perfectly straight text. We use CV2's perspective warp functionality to straighten these images. To straighten the images, we draw a box around the title that we want to appear as straight, and warp the image to fit into a perfect rectangle. The rest of the image follows the same geometric transformation.

Fig 8 - Before and after warping our image to have horizontal text
The results of the image to text tool are included

With the filtering, thresholding and straightening preprocessing steps built into our pipeline, we have much more promising results as expected.

Code and Data

Access to the code used for this project can be found in the following GitHub repository:

https://github.com/tomribaroff/UDA-FinalProject

References

1 - “Text-Based Image Segmentation Methodology” 2014 - Gupta, Patel, Dave, Goradia and Saurin

2 - https://docs.opencv.org/3.1.0/d3/d05/tutorial_py_table_of_contents_contours.html

3 - https://pyimagesearch.com/2014/05/05/building-pokedex-python-opencv-perspective-warping-step-5-6/

4 - https://people.csail.mit.edu/sparis/bf_course/course_notes.pdf

5 - https://pyimagesearch.com/2018/09/17/opencv-ocr-and-text-recognition-with-tesseract/

Independent Work Statement

I confirm that have worked on this project independently.