REPORT
Odilbek Umaraliev
Task 2

Task 1: Report on Scrapers

This report aims to provide an overview of the scrapers developed to retrieve cloth images information from Ralph Lauren (https://www.ralphlauren.nl/) website. Two different types of scraping were performed, each of them targeting a specific type of information. The first scraper retrieves the product images from website, the second one retrieves a list of proxy IP addresses.

Product Details Scraper
The first scraper was developed to retrieve product details from website. It uses the BeautifulSoup and Selenium library to parse the HTML content of the website and extract the relevant information. The product details are stored in a dictionary and the final list of dictionaries is saved as a JSON file.

Proxy Scraper
This Scraper was totally optional. The scraper was developed to retrieve a list of proxy IP addresses. It uses the requests library to make HTTP requests to a GitHub repository that contains a list of proxy IP addresses. The list is then parsed using BeautifulSoup and the proxy IP addresses are stored in a list. A validation step is performed to check if the proxy IP addresses are working, and only the working ones are saved in a text file.

Warnings and Issues
Scraping website was not easy. Because each time when sending a request to website more than 2 or 3 times, the website was blocking my ip. I have used user-agent, undetected chrome driver and different proxies. However, even these methods is not reliable and stable. First I scraped all the data using undetected chrome driver yesterday, but it gave me an error today. So, in this case I changed my driver to Firefox and it worked. Make sure that each time change ip address and try again. Trying to use different ip addresses and proxies will be helpful as well. I also add Proxy scraper, because there is some proxies are stable for 'https' requests. In case if you cant scrape with different ip addresses, use this approach.

Code Explanation
Both scrapers belong to class-based code structure. Codes are well documented and well commented. Everyone who reads this code with comments can easily understand.

Approach
I use following approach:

1 - 32 of 141 items

1. Getting all number of images from given website
2. I use them by dividing them 32 and to use it on link
   a. Example:
      https://www.ralphlauren.nl/en/men/clothing/hoodies-sweatshirts/10204?sw1=s

w-cache-me&webcat=men%7Cclothing%7Cmen-clothing-hoodies-sweatshirts&start=64&sz=32

  b. In this link there is start which is increments each time by 32 and sz. So, I used some logic to divide number of items and get each link.
  c. I choose this approach because when I use selenium to click "View All" link and JS code to scroll down the page stopped loading and I was not able to get all data that I want. So, the approach I used perfectly worked in this case.
3. Retrieving info (Product name, brand name, weared img link and not weared img link) from each link with brand name "Polo Ralph Lauren". Remember that each link has about 32 items.
4. All scraped data was downloaded and saved into data folder and was divided by two (Weared and Not Weared).

Task 2: Report on Segmentation

!!! I HIGHLY RECOMMEND TO OPEN THIS NOTEBOOK FILE ON COLAB

Applied Steps

I have divided this task in several steps

1. Getting list of Not Weared Images from folder
2. Selecting Random Images to visualize them
3. Visualizing Random Images
4. Getting masks of randomly taken images and download them
   a. This step allows to save original image and mask of that image in black and white format
5. Getting list of original and masks names that we saved in 4th step to use them in final function which is ***def scrape_images(url)***

   # saves the images in particular folders
6. This is final step of this task. The function saves the original image with a blue color background image. It is possible to iterate for each random taken image. but for simplicity I use only one image as an example.

In this project the pretrained model and baseline code were very helpful to achieve the goal. The model predicts mask itself not background of it. So I could easily change the sign of '>' to '<'. By changing this sign actually we are looking for background of an image.