CSC420
Project 4
Scalable Recognition with a Vocabulary Tree

Devanshu Singhvi

Introduction:

This project aims to identify the book covers of an image of book(s) taken in some oblique angle, from a database that contains around 100 book cover images scraped from *Amazon*. When given an image of some books, localising each individual book in the image and attempting to find the matching cover from the database seems to be the logical approach, but directly taking the features and descriptors of each book and comparing it against the entire database is very calculation heavy and complex, taking a lot of time and space.

Instead, the idea behind my approach for this is based on Nister and Stewenius' *Scalable Recognition with a Vocabulary Tree* and uses a vocabulary tree, which is a hierarchical structure for indexing visual features (descriptors of the image found by SIFT in this project). We will see details about how this tree is built, how the features are quantified and how the querying of an image of a book is done in this report.

Once the vocabulary tree is built, we can see how similar two different images are by querying the tree with their descriptors and finding the number of common clusters between them. It is explained in detail in the following sections. Although, before the images are compared, it is important to isolate/localise the books from the images.

After we have the top *k* matches to the queried image in the database, it can be filtered by applying RANSAC to find the best homography transformation between the queries image and each potential match given by the vocabulary tree. Finding the image comparison that has the highest total number of inliers, which are the number of consistent keypoints between the compared images, reduces the top *k* matches to just one, which should be the actual cover of the book.

These are the base, general descriptions of the steps I followed to obtain the covers of books in an image. More details of each step are listed in the following sections.

1. Localising Books from Query Images:

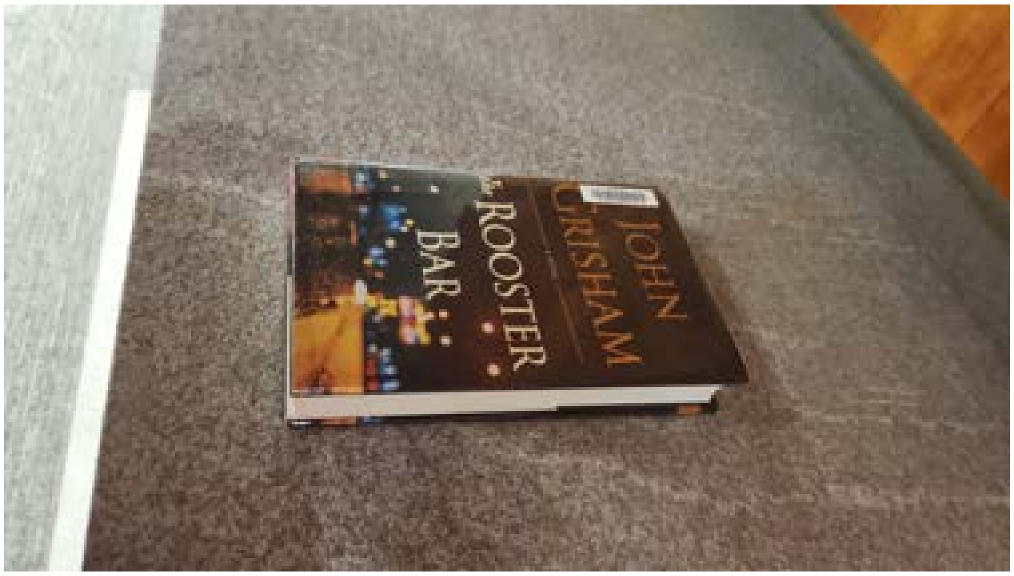
Since the database for book covers I am using is already preprocessed and localised, only the query images which may contain more than one book have to be processed to somehow obtain each individual book in the image.

I first tried to use corner detection and edge detection, but neither of which gave satisfactory results and instead I ended up using the YOLOv8 (You Only Look Once) library with a built-in pretrained model to crop out different books from one singular image due to mainly the speed and accuracy with which it could recognise various objects since the model was trained on a general object detection dataset.

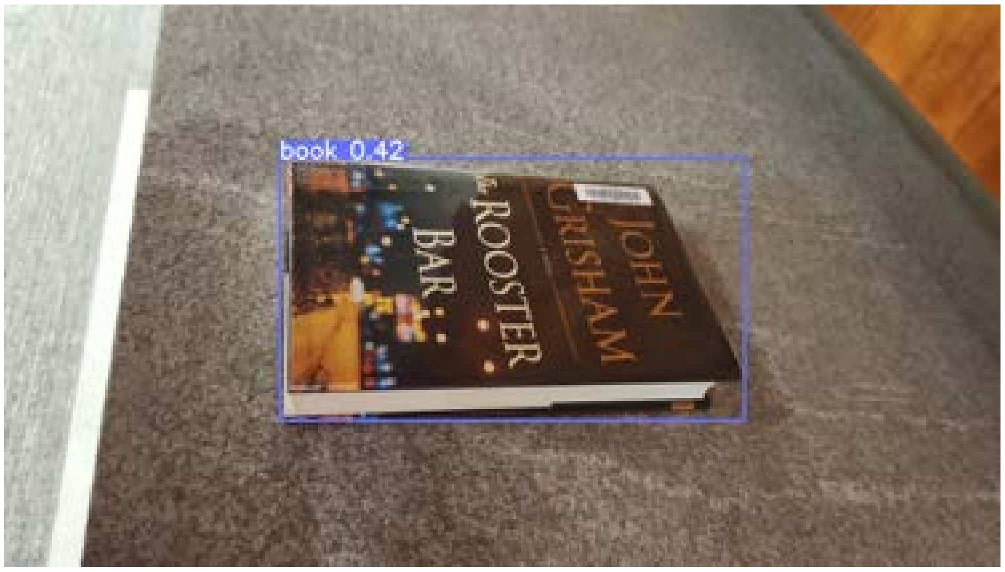
After running the YOLOv8 model on the input image, we obtained bounding boxes around the detected books. These bounding boxes indicated the regions where books are located within the image. We can use these to crop out individual book regions from the original image, resulting in cropped images containing only one book each.

Take a look at the following examples of this below. Since I did not train this model on a custom dataset of book covers, it tends to make mistakes sometimes.

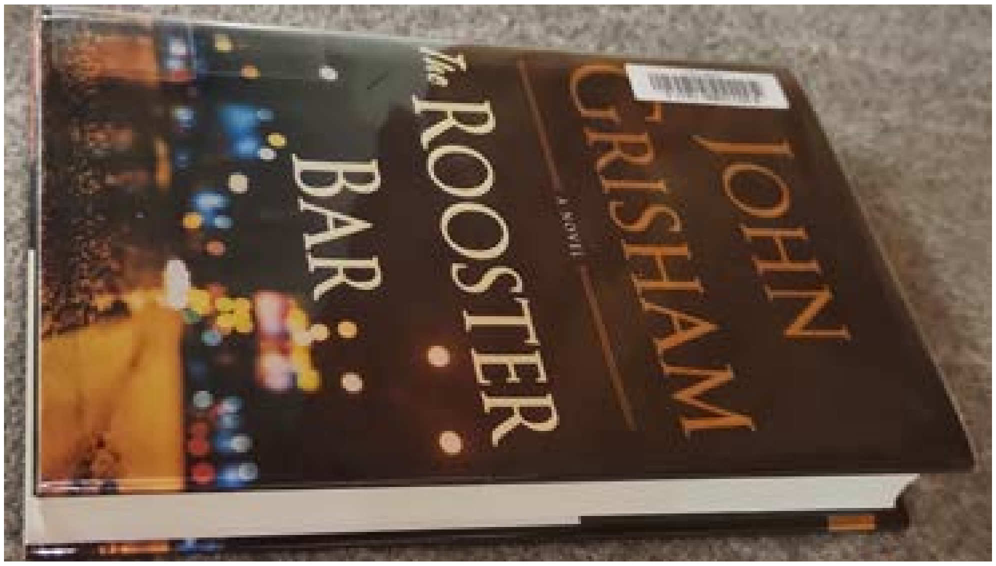
Original Image:



After running the YOLO model on the image:



Cropped image:



2. Vocabulary Tree:

The main concept and idea behind a vocabulary tree is fantastically explained in [1]. The vocabulary tree is a hierarchical structure built using hierarchical k-means clustering on certain features (SIFT descriptors in this case). The process of building it begins by extracting local features,i.e., the SIFT descriptors from every image in a dataset of images. These represent distinct visual patterns in each image. Then, the descriptors are clustered hierarchically using k-means clustering, which organises the descriptors in a tree structure. Each level of the tree represents more detail than the last in the visual features.

The build_tree method in the VocabularyTree class implements this. It starts with a root node and iteratively creates child nodes corresponding to each cluster. The KMeans algorithm is utilised to perform clustering, and the resulting clusters form the children of the current node. This process continues until either the maximum depth is reached or the number of descriptors in a cluster is insufficient for further partitioning.

The tree seems to be most accurate when k=10 and L=6 after hyperparameter tuning. As shown in [1],

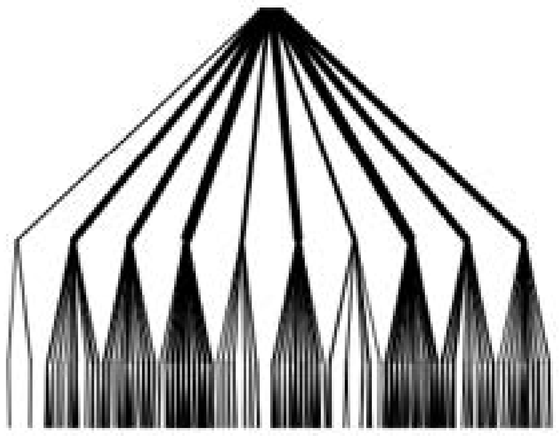


Figure 3. Three levels of a vocabulary tree with branch factor 10 populated to represent an image with 400 features.

Initialization:

- The VocabularyTree class is initialised with parameters k and L, where k represents the number of clusters (or centroids) for each node, and L represents the depth or number of levels of the tree.
- Various attributes that keep track of the tree, descriptors of images in dataset, weights, etc are initialised.

Fitting the Tree:

- The fit method is called to build the tree based on a set of input images.
- The num_imgs attribute is set to the total number of input images.
- The feature_extraction method is called to extract keypoints and descriptors from each input image.
- The build_tree method is invoked to construct the hierarchical structure of the tree.

Building Each Node:

- The build_tree method recursively constructs each node of the tree.
- For each node:
    - If the number of descriptors associated with the node is less than the specified threshold or if the maximum depth is reached, the node is a leaf node.
    - Otherwise, k-means clustering is performed on the descriptors associated with the node to partition them into k clusters.
    - Each cluster represents a child node, and the process is recursively applied to build the subtree rooted at each child node.
    - Leaf nodes are assigned an index and associated with the images containing their descriptors.
    - The *TF-IDF* weights for the leaf nodes are calculated based on the frequency of occurrence of descriptors across all images. More explained in the next section.

Database Vector Computation:

- After building the tree, the _compute_database_vectors method computes the database vectors for each image based on the *TF-IDF* weights and the frequency of descriptors associated with each leaf node.
- m_i represents the number of descriptor vectors of the database image i, and w_i represents the weight of the corresponding node in the vocabulary tree. The database vector d_i is computed as the element-wise product of m_i and w_i for each image i.

Query Vector Computation:

- The query vector q_i is calculated as the weighted combination of nodes in the vocabulary tree that are closest to the query descriptors. For each descriptor in the query image the closest node in the tree is identified.
- The weight of each node (w_i) is multiplied by the number of occurrences of descriptors associated with that node in the query image (n_i). The resulting weighted values are summed up to form the query vector q_i

Closest Node:

- Starting from the root of the tree, it traverses down the tree by selecting the child node with the minimum Euclidean distance between its cluster centre and the descriptor. If the current node is a leaf node, the method returns the current node. This process continues until a leaf node is reached.

Computing Scores:

- It extracts the descriptors of the query image using the feature_extraction method. Then, it computes the query vector q_i and obtains the list of nodes corresponding to the query descriptors using the _compute_query_vector method. Next, it normalises the query vector. For each node in the list of nodes, it collects the images associated with that node and calculates the database vector d_i for each image. It normalises each database vector and computes the L1 distance between the normalised database vector and the normalised query vector. Finally, it sorts the computed scores and returns the sorted lists of target images (targets) and their corresponding scores.

2.1 Vocabulary Tree: Weights Analysis

During the construction of the vocabulary tree, the weights of leaf nodes are calculated and given by the formula:

$$w_i = \ln \frac{N}{N_i},$$

where N is the number of images in the database and N_i is the number of images in the database with at least one descriptor vector path through node i.

The weights are calculated in a *TF-IDF (Term Frequency Inverse Document Frequency) s*cheme. The leaf nodes in the vocabulary tree are associated with explicitly stored file names. The images in these files have a feature represented by the visual word at the leaf.

Here, Term Frequency (TF) represents the frequency of occurrence of a term (leaf node) within an image and Inverse Document Frequency (IDF) measures the rarity of the term across all images.

Within the build_tree method, when a leaf node is identified:

- The number of descriptors associated with each image for the leaf node is accumulated in the num_descriptors dictionary.
- The weight is computed as the logarithm of the ratio num_imgs / len(images), where num_imgs is the total number of images and len(images) is the number of images containing descriptors associated with the leaf node.

3. Homography estimation with RANSAC

From the score computation, we can get the top 10 closest matches and perform feature mapping for each feature in the query image and find the closest feature in a potential final match image.

Once these are found, using RANSAC with 1000 iterations to find the best homography that has the highest number of inliers for each potential match to the query image and the highest inlier-holding image should be the book cover for that query image.

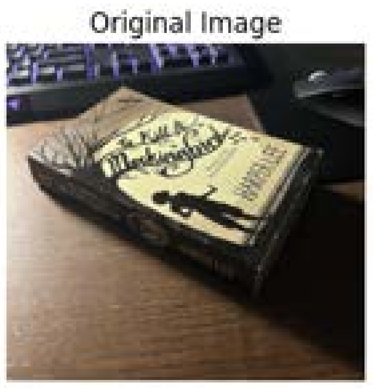
4. Challenges and Problems

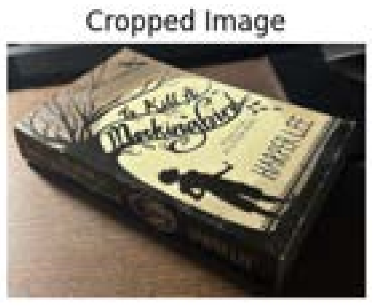
The biggest issue was coming up with ideas to make the VocabularyTree setup and functions not be super heavy on memory and time complexity. The paper by Nister and Stewenius did not go in depth on how to implement the tree given space and time complexities and I had to run many tests to keep it as efficient as I could.

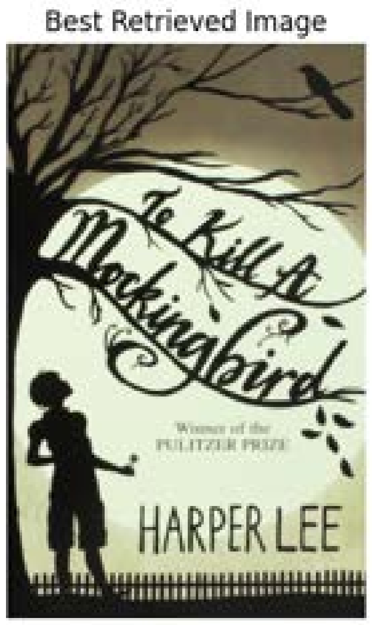
Another issue is that feature matching is a very slow task and can take up to a couple minutes to run in my implementation.

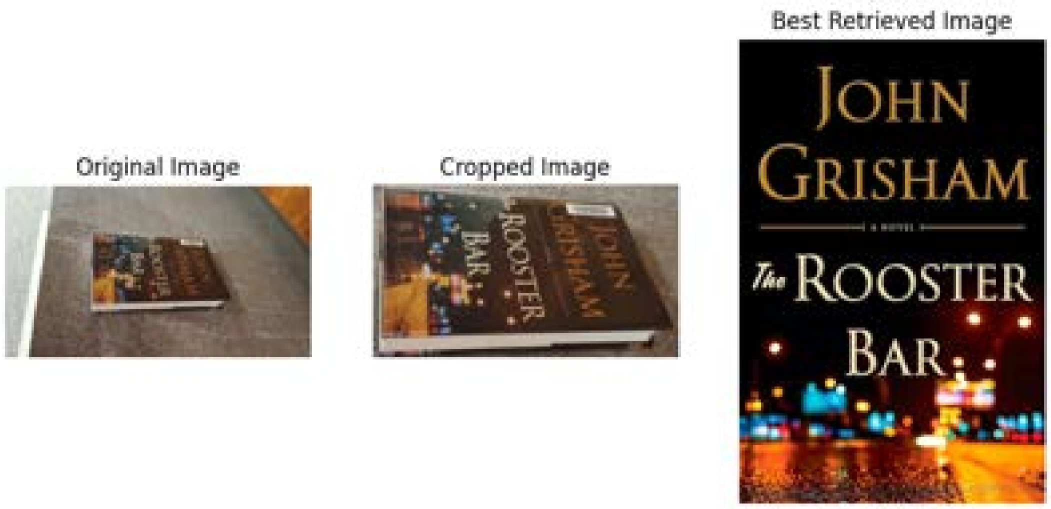
5. Testing:

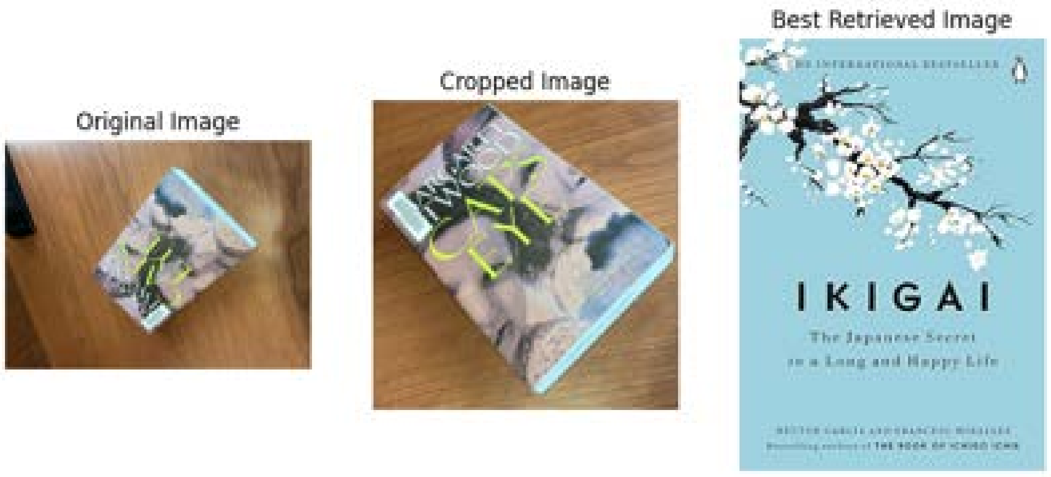
Note that these were run with k=10 and L=6.

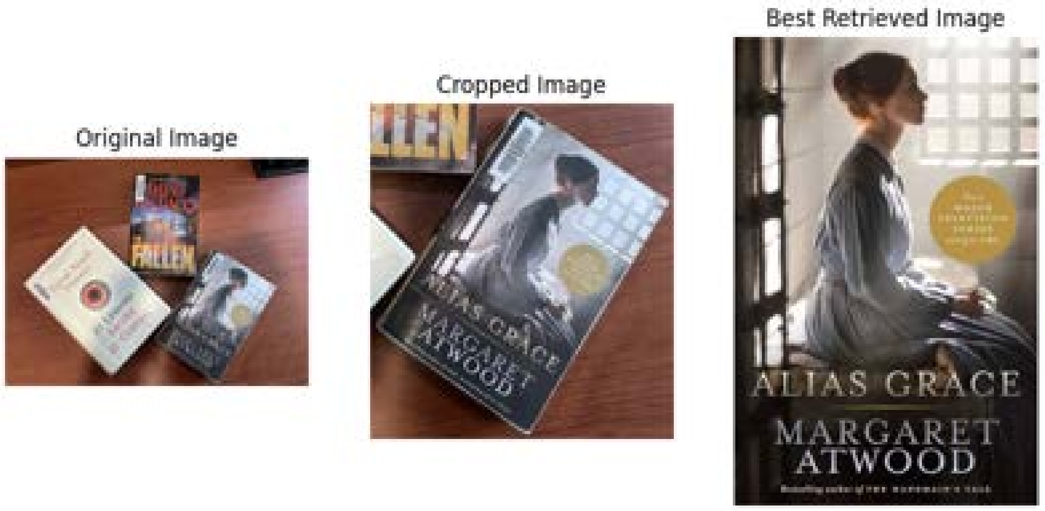


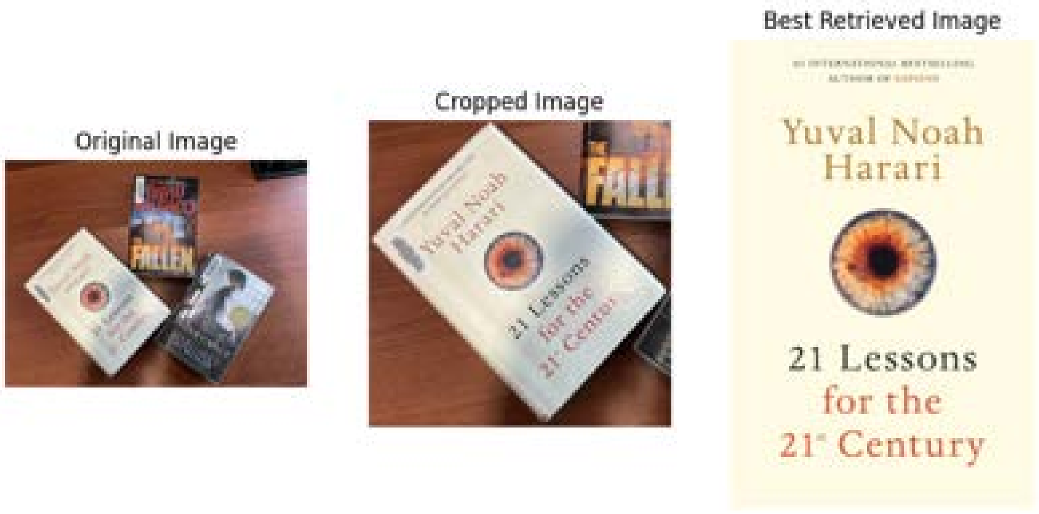


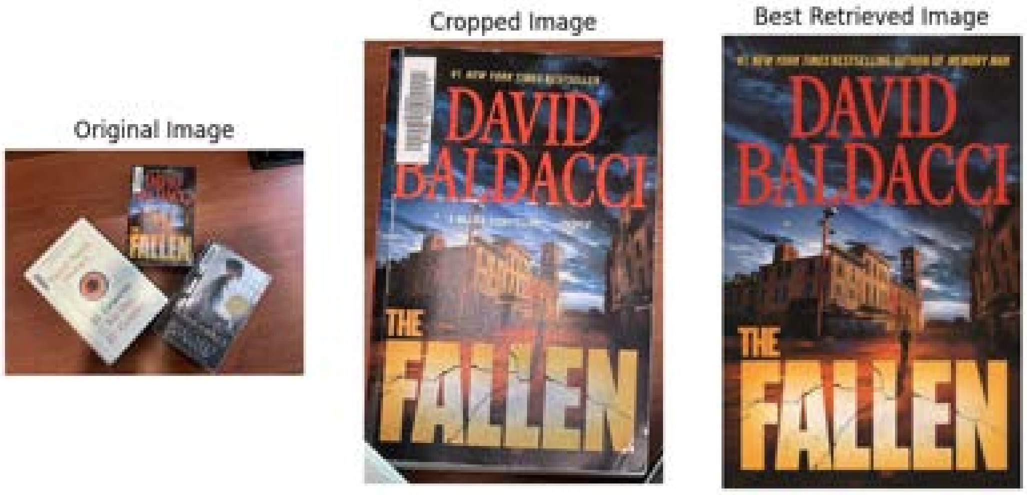


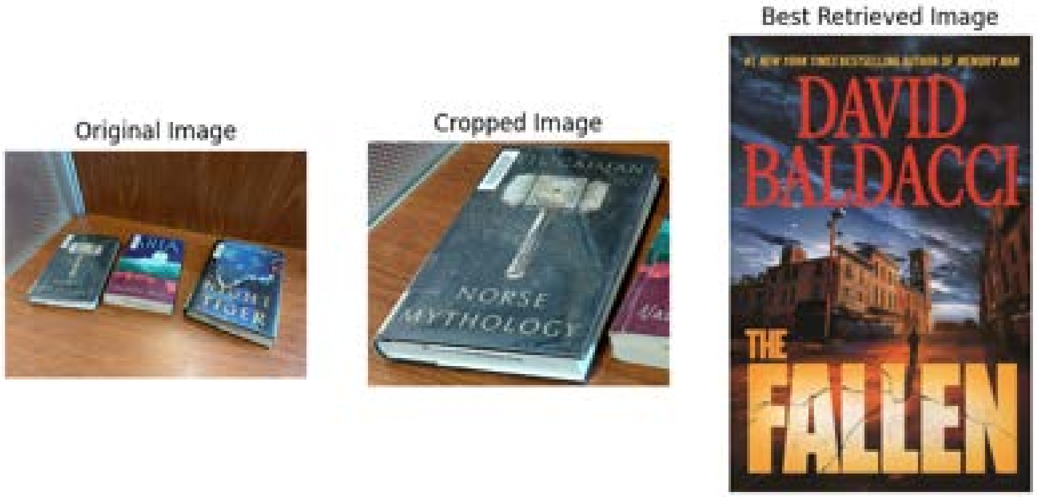

Original Image
Cropped Image
Best Retrieved Image
JOHN GRISHAM
A NOVEL
The ROOSTER BAR


Original Image
Cropped Image
Best Retrieved Image
THE INTERNATIONAL BESTSELLER
IKIGAI
The Japanese Secret to a Long and Happy Life


Original Image
Cropped Image
Best Retrieved Image
ALIAS GRACE
MARGARET ATWOOD

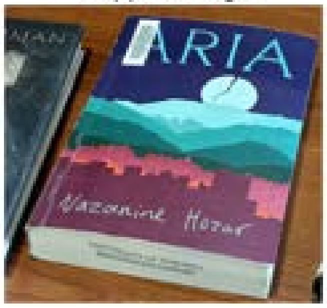

Original Image
Cropped Image
Best Retrieved Image
Yuval Noah Harari
21 Lessons for the 21st Century


Original Image
Cropped Image
Best Retrieved Image
DAVID BALDACCI
THE FALLEN

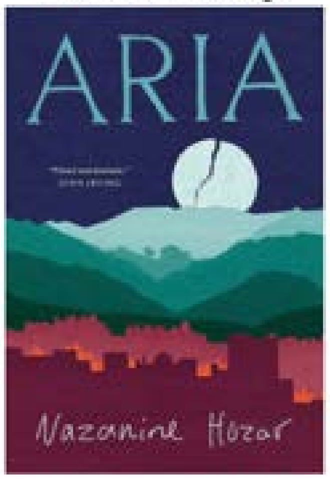

Original Image
Cropped Image
Best Retrieved Image
NORSE MYTHOLOGY
DAVID BALDACCI
THE FALLEN

Original Image

Cropped Image

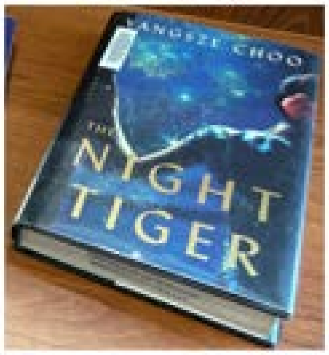


Best Retrieved Image

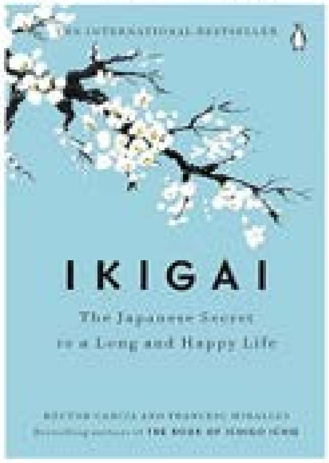


Original Image

Cropped Image



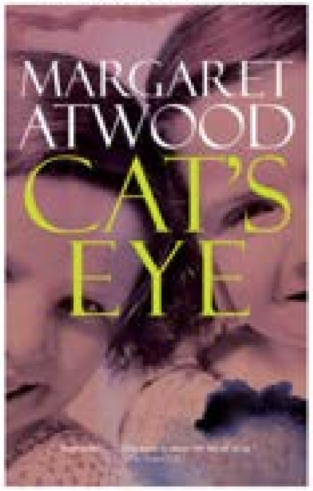
Best Retrieved Image



Original Image

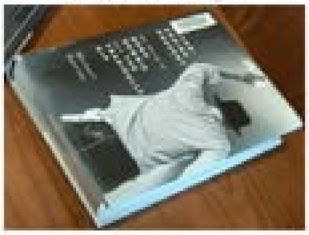
Cropped Image

Best Retrieved Image

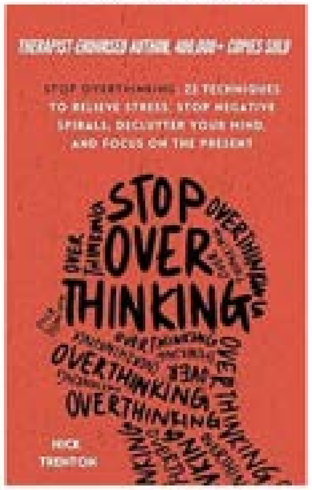


Original Image

Cropped Image

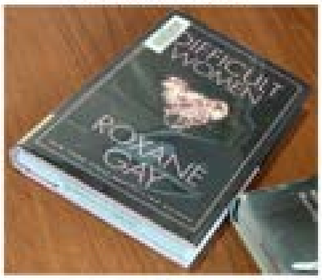


Best Retrieved Image

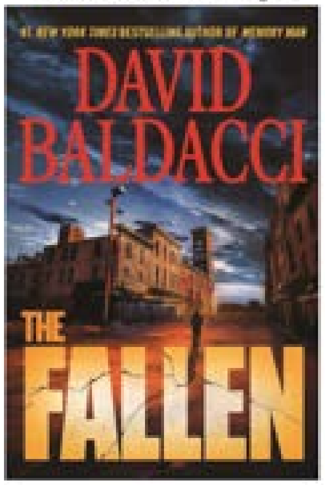

THERAPIST-ENDORSED AUTHOR, 400,000+ COPIES SOLD
STOP OVERTHINKING: 23 TECHNIQUES TO RELIEVE STRESS, STOP NEGATIVE SPIRALS, DECLUTTER YOUR MIND, AND FOCUS ON THE PRESENT
STOP OVERTHINKING
NICK TRENTON

Original Image

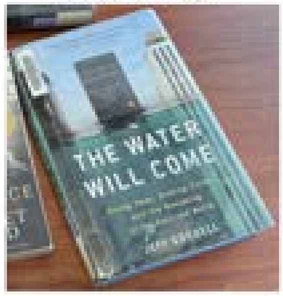
Cropped Image



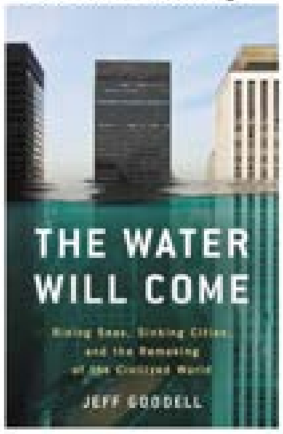
Best Retrieved Image



Original Image

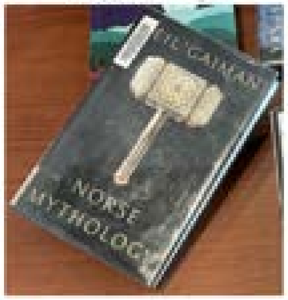


Cropped Image

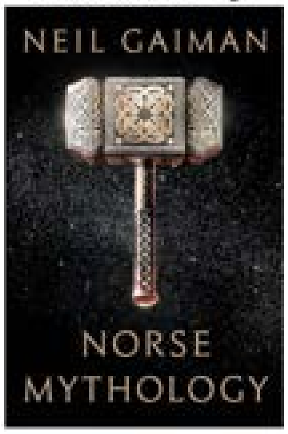


Best Retrieved Image

Original Image

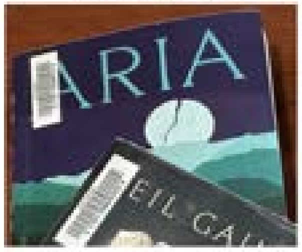


Cropped Image

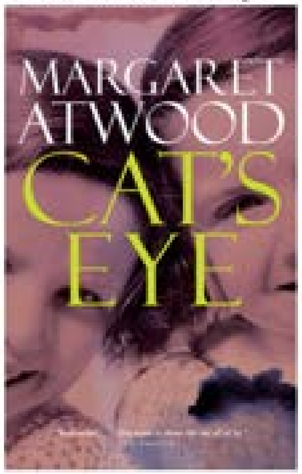


Best Retrieved Image

Original Image



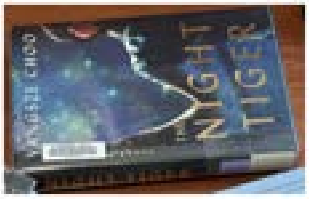
Cropped Image



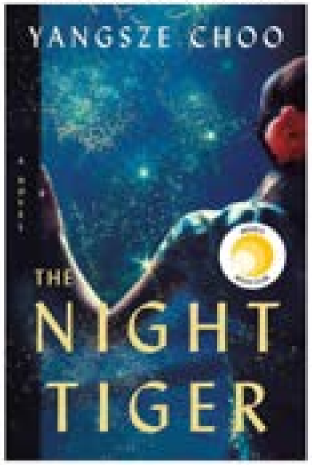
Best Retrieved Image

Original Image

Cropped Image

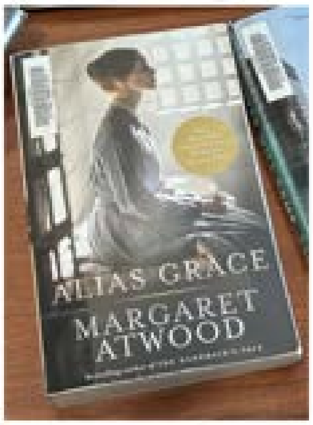


Best Retrieved Image

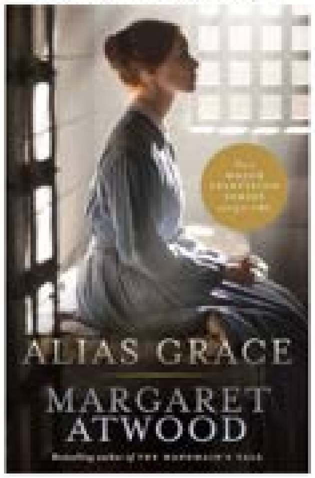


Original Image

Cropped Image

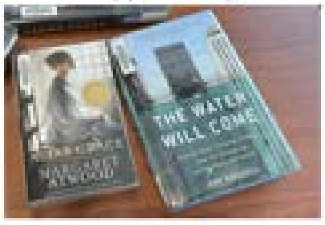


Best Retrieved Image

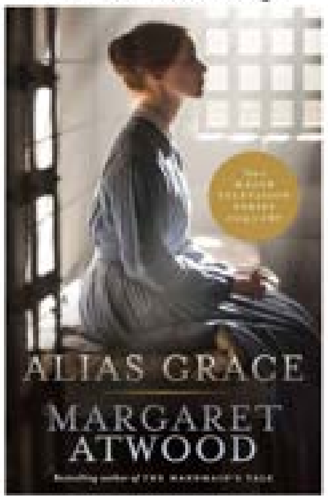


Original Image

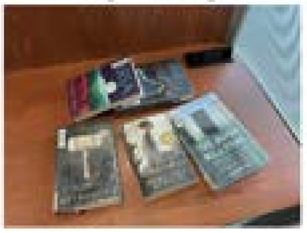
Cropped Image



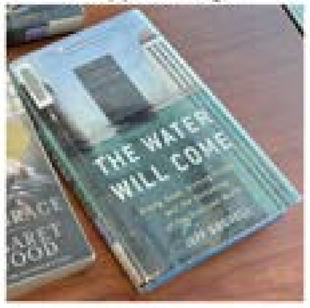
Best Retrieved Image



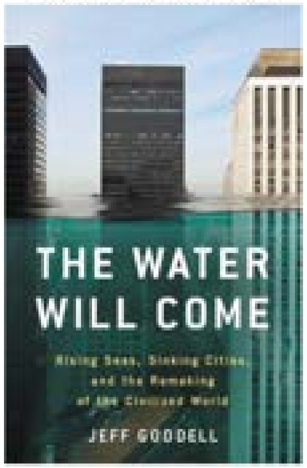
Original Image



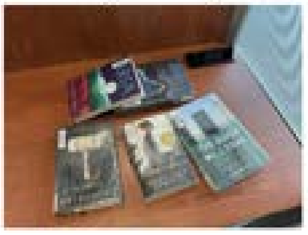
Cropped Image



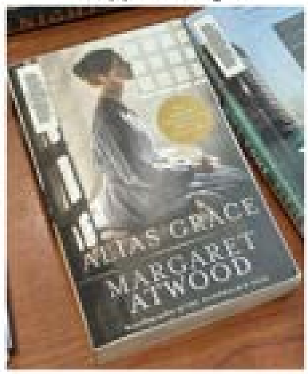
Best Retrieved Image



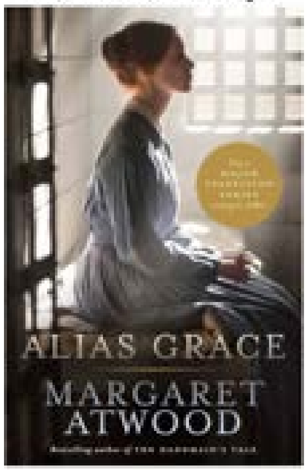
Original Image



Cropped Image

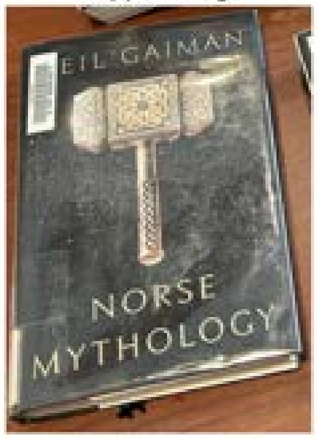


Best Retrieved Image

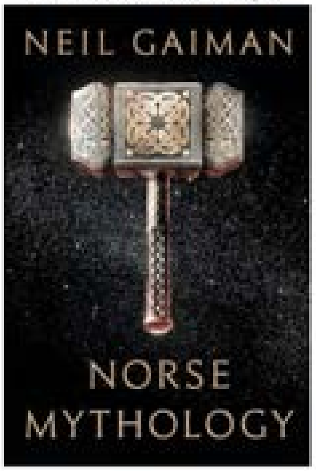


Original Image

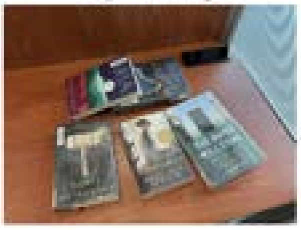
Cropped Image



Best Retrieved Image

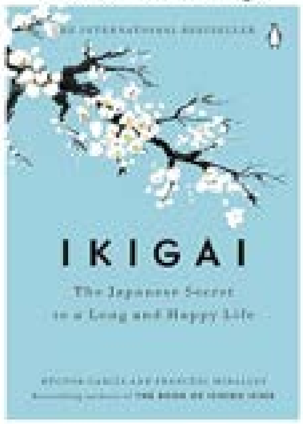


Original Image



Cropped Image

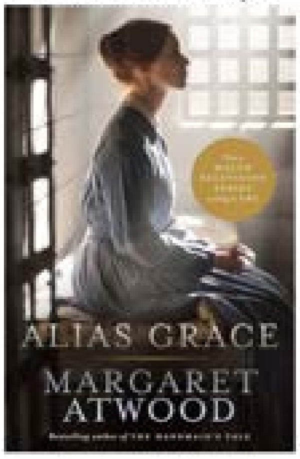
Best Retrieved Image



Original Image

Cropped Image

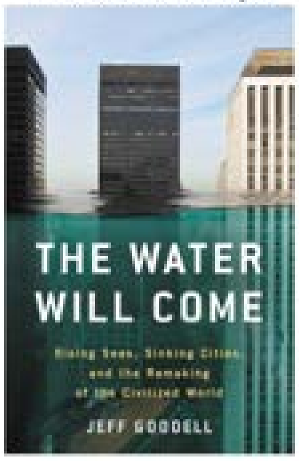
Best Retrieved Image



Original Image

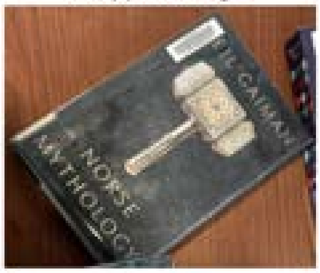
Cropped Image

Best Retrieved Image

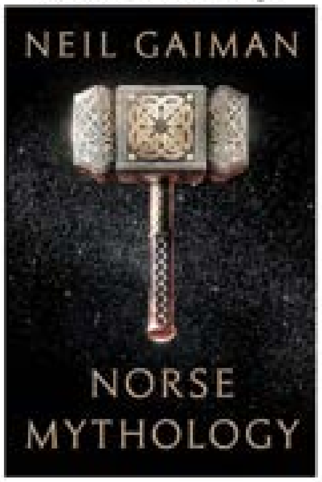


Original Image

Cropped Image

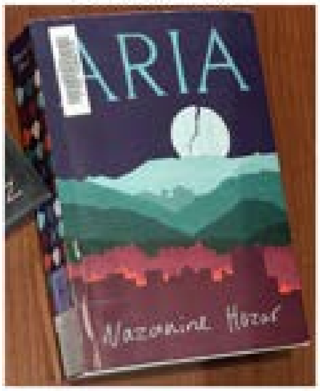


Best Retrieved Image

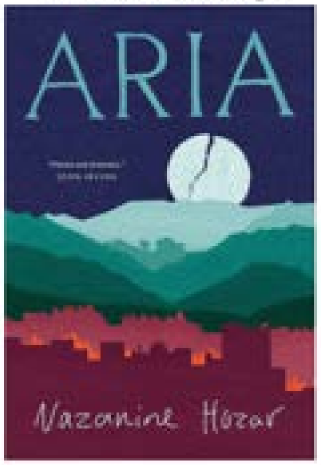


Original Image

Cropped Image

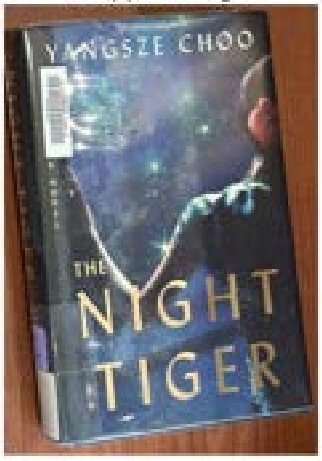


Best Retrieved Image

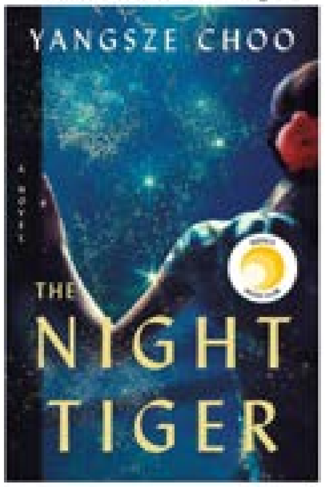


Original Image

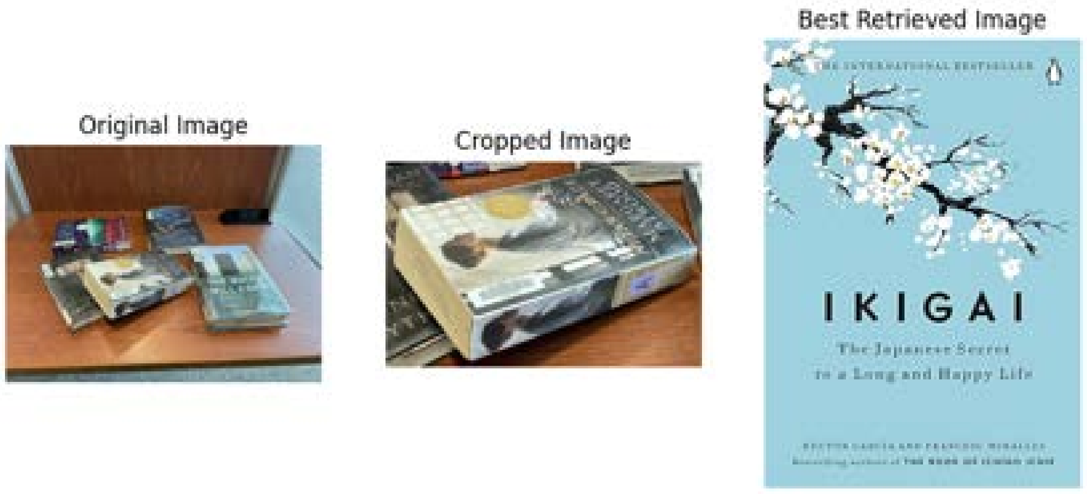
Cropped Image

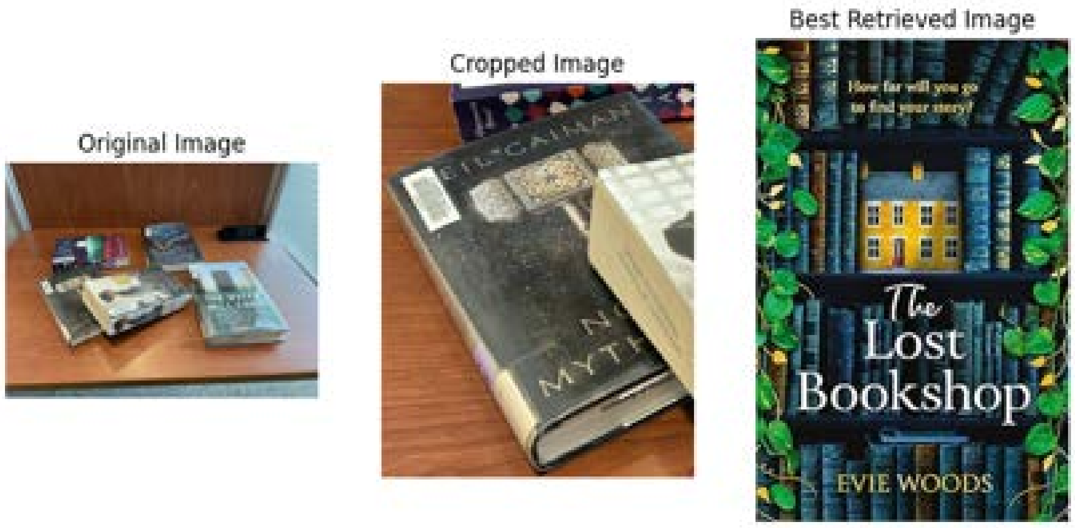


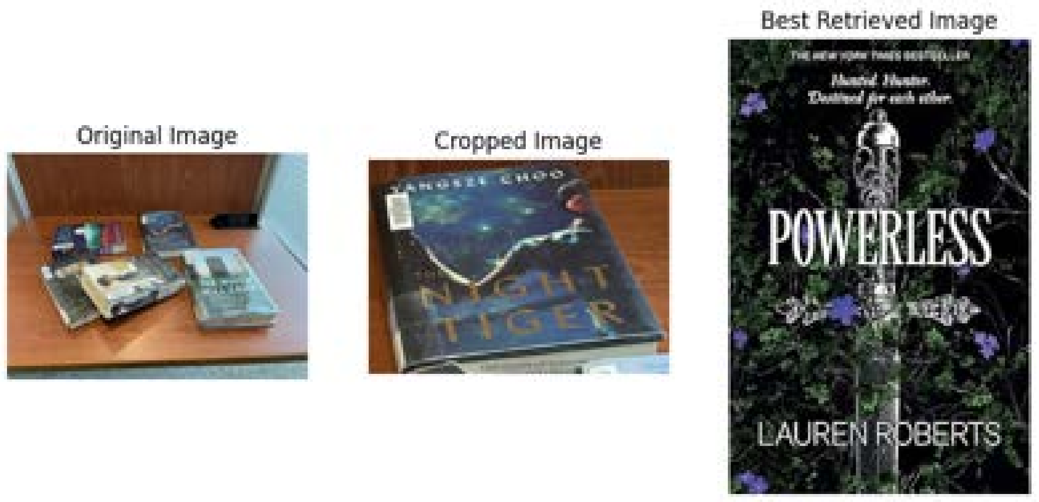
Best Retrieved Image

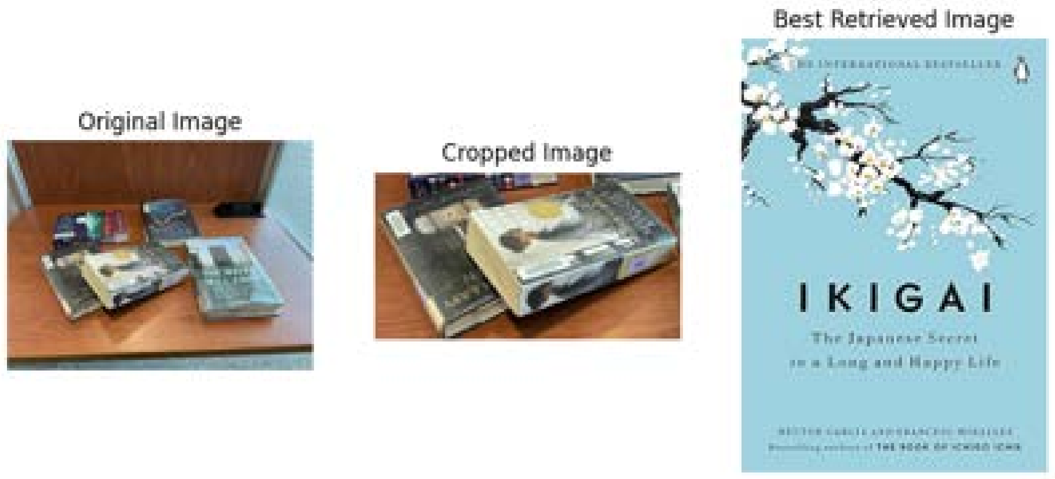


Original Image
Cropped Image
Best Retrieved Image
THE WATER WILL COME
IKIGAI
The Japanese Secret to a Long and Happy Life


Original Image
Cropped Image
Best Retrieved Image
IKIGAI
The Japanese Secret to a Long and Happy Life


Original Image
Cropped Image
Best Retrieved Image
How far will you go to find your story?
The Lost Bookshop
EVIE WOODS





6. Results

Overall, a lot of the predictions are correct. For the ones that aren't, it seems to be a glare/reflection issue since I tried testing it with those exact same books under different lighting and similar angles, which can also be seen in some of the images above, that sometimes the book cover is accurately found and sometimes it is not.

Another problem is that yolov8 doesn't always crop the image properly and sometimes there's another book or another object that gets counted as a book.

7. References

[1] David Nister and Henrik Stewenius "Scalable Recognition with a Vocabulary Tree", CVPR 2006
[2] YOLOv8 https://github.com/ultralytics/ultralytics