Book Cover Quality Checker

The Book Cover Quality Checker is a Python-based application designed to automatically monitor a specified Google Drive folder for new book cover files (images or PDFs), process them with OCR and image quality assessments, and then provide feedback through Airtable records and email notifications. The application verifies that the text on a book cover does not overlap the designated award badge area, checks for safe margins, and assesses image quality (DPI and pixelation). It is especially useful for publishers who require a quick review of submitted cover designs before they are published.

Overview

- **Monitoring:** Continuously polls a given Google Drive folder for new files at specified intervals.
- **OCR & Text Detection:** Uses EasyOCR to extract text from images and PDFs.
- **Badge & Margin Checking:** Ensures that the award badge at the bottom-right corner is not invaded by text and that all text stays within a safe margin.
- **Image Quality Assessment:** Evaluates the image quality by calculating the effective DPI and checking for pixelation.
- **Reporting & Feedback:** Generates an overall assessment and selects an email template based on issues detected. It also logs the processed file details in Airtable.
- **Email Notification:** Sends feedback emails to authors if issues are found.
- **Google Drive & Airtable Integration:** Supports automatic file download and data insertion for seamless workflow automation.

---

1. **Set Up Google Drive and Airtable Credentials**
    - **Google Drive:**
        - Create a Google Cloud project and enable the Drive API.
        - Configure OAuth2.0 credentials and download the credentials file.
        - Rename the downloaded file and place it where your code can access it.
        - On the first run, the application will prompt you to authenticate via the browser.
    - **Airtable:**

- Sign up for Airtable and create a base with the appropriate table.
- Retrieve your API key, Base ID, and Table Name.
- Update the configuration section in main.py with these values:
- AIRTABLE_API_KEY = "YOUR_AIRTABLE_API_KEY"
- AIRTABLE_BASE_ID = "YOUR_AIRTABLE_BASE_ID"
- AIRTABLE_TABLE_NAME = "YOUR_TABLE_NAME"

2. **Configure Email Credentials**

In main.py, update the email address and password (or app-specific password when using Gmail):

EMAIL_ADDRESS = "amranadaf@gmail.com"

EMAIL_PASSWORD = "YOUR_EMAIL_PASSWORD"

Usage Examples

Run the script

To start monitoring new files in the specified Google Drive folder, simply execute:

python main.py

The application will:

- Authenticate with Google Drive.
- Poll the folder every defined interval (default 60 seconds) for new book cover files.
- Process any new file it finds, log its information, insert data into Airtable, and send an email with feedback.

Test on a Single File Locally

(This was used for testing purpose of the entire workflow)

For quick testing of a single image file:

1. Open main.py.
2. Uncomment the testing lines at the bottom of the file:
3. if __name__ == "__main__":
4. # For testing a single image locally:
5. result = process_book_cover("path/to/your/test_image.png")

6. print(result)
7.
8. # For automated Google Drive monitoring:
9. # main()
10. Run the script:
11. python main.py

Configuration / Customization

Configurable Parameters

- **Google Drive Folder and Polling:**
    - FOLDER_ID: Your target Google Drive folder ID.
    - POLL_INTERVAL: Time interval between checks (in seconds).
    - DOWNLOAD_DIR: Local directory to save downloaded files.
- **Airtable Settings:**
    - AIRTABLE_API_KEY: Your Airtable API key.
    - AIRTABLE_BASE_ID: The base ID of your Airtable.
    - AIRTABLE_TABLE_NAME: The table name where records will be inserted.
- **Expected Badge Text:**
    - EXPECTED_TEXT1 and EXPECTED_TEXT2: Text values used to verify acceptable overlap with the award badge.
- **Email Settings:**
    - EMAIL_ADDRESS and EMAIL_PASSWORD: Credentials for sending email notifications.
    - Default recipient is set in the send_email() function but can be modified as needed.
- **Image Quality & Safe Margin Settings:**
    - The safe margin is calculated based on an actual DPI value (here set at 100 DPI). Adjustments may be needed based on your image source.

Feel free to modify these parameters to suit your integration needs.

Dependencies

- Python 3.6+
- OpenCV (opencv-python)
- EasyOCR
- NumPy
- pdf2image
- PyAirtable
- PyDrive
- Pillow (for pdf2image)
- Standard Python libraries: os, time, datetime, difflib, re, smtplib, email

---

DOCUMENTED CODE FOR REFERENCE

```
```python
import cv2
import easyocr
import difflib
import numpy as np
import re
import os
import time
from datetime import datetime
from pdf2image import convert_from_path
from pyairtable import Api
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# ==================== CONFIGURATION ====================
FOLDER_ID = "1V_K4reMFymFRytjrtMFmOO7QkN-E5dMf"  # Your Google Drive folder ID
POLL_INTERVAL = 60  # Check every 60 seconds
DOWNLOAD_DIR = "./downloads"
os.makedirs(DOWNLOAD_DIR, exist_ok=True)

# Airtable setup
AIRTABLE_API_KEY = "#"
AIRTABLE_BASE_ID = "app4SYI2WfSXar9nP"
AIRTABLE_TABLE_NAME = "tblCECcfMV07ksUxM"

# Initialize Airtable API
api = Api(AIRTABLE_API_KEY)
table = api.table(AIRTABLE_BASE_ID, AIRTABLE_TABLE_NAME)

# Expected badge text
EXPECTED_TEXT1 = "Winner of the 21st Century Emily Dickinson Award".lower()
EXPECTED_TEXT2 = "Award".lower()

# Track processed files
PROCESSED_FILES_LOG = "processed_files.txt"

def load_processed_files():
    """Load previously processed file IDs from disk"""
    if os.path.exists(PROCESSED_FILES_LOG):
        with open(PROCESSED_FILES_LOG, 'r') as f:
            return set(line.strip() for line in f if line.strip())
    return set()
```
```

```
def save_processed_file(file_id):
    """Save a processed file ID to disk"""
    with open(PROCESSED_FILES_LOG, 'a') as f:
        f.write(f"{file_id}\n")

processed_files = load_processed_files()
print(f"Loaded {len(processed_files)} previously processed files")

# Initialize OCR reader once (expensive operation)
print("Initializing OCR reader...")

reader = easyocr.Reader(['en'], gpu=True)
print("✓ OCR reader ready\n")

# ==================== HELPER FUNCTIONS ====================

def draw_bounding_boxes(image, detections, threshold=0.25):
    """Draw bounding boxes around detected text"""
    for bbox, text, score in detections:
        if score > threshold:
            cv2.rectangle(image, tuple(map(int, bbox[0])), tuple(map(int,
bbox[2])), (0, 255, 0), 5)
            cv2.putText(image, text, tuple(map(int, bbox[0])),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 0.65, (255, 0, 0), 2)

def find_overlap_text_dual(text_detections, badge_coords, expected_text1,
expected_text2, similarity_threshold=0.8,
                           tolerance=3):
    """Check if any text overlaps with badge area (except expected badge
text)"""
    badge_x1, badge_y1, badge_x2, badge_y2 = badge_coords
    safe_margin_flag_foverlap = False
    overlap_flag = False
    overlap_text = ""

    for bbox, text, score in text_detections:
        xs = [pt[0] for pt in bbox]
        ys = [pt[1] for pt in bbox]

        if any((badge_x1 - tolerance) <= x <= (badge_x2 + tolerance) and
               (badge_y1 - tolerance) <= y <= (badge_y2 + tolerance) for x, y
in zip(xs, ys)):

            similarity1 = difflib.SequenceMatcher(None, text.lower(),
expected_text1.lower()).ratio()
            similarity2 = difflib.SequenceMatcher(None, text.lower(),
expected_text2.lower()).ratio()
            print(f"  Detected: '{text}', similarity1: {similarity1:.2f},
similarity2: {similarity2:.2f}")

            if similarity1 < similarity_threshold and similarity2 <
similarity_threshold:
                print(f"  ⚠ Overlapping text found: '{text}'")
                safe_margin_flag_foverlap = True
                overlap_flag = True
```

```
                    overlap_text = text

        if not overlap_flag:
            print("  ✓ No unwanted overlaps detected.")

        return safe_margin_flag_foverlap, overlap_flag, overlap_text

    def comprehensive_image_quality(image_region, expected_width_inches=5,
    expected_height_inches=8, actual_dpi=100):
        """Complete image quality assessment including DPI and pixelation"""
        height_px, width_px = image_region.shape[:2]

        # Calculate DPI
        dpi_h = width_px / expected_width_inches
        dpi_v = height_px / expected_height_inches
        avg_dpi = (dpi_h + dpi_v) / 2

        # DPI Assessment
        if avg_dpi >= 300:
            dpi_status = "✓ EXCELLENT - Print Ready"
            dpi_score = 100
        elif avg_dpi >= 200:
            dpi_status = "⚠ ACCEPTABLE - May show quality loss"
            dpi_score = 60
        elif avg_dpi >= 150:
            dpi_status = "⚠ POOR - Visible pixelation expected"
            dpi_score = 30
        else:
            dpi_status = "❌ REJECTED - Not suitable for printing"
            dpi_score = 0

        # Pixelation check
        gray = cv2.cvtColor(image_region, cv2.COLOR_BGR2GRAY).astype(np.float32)
        block_size = 8

        vertical_diffs = []
        for i in range(block_size, width_px, block_size):
            vertical_diffs.append(np.mean(np.abs(gray[:, i] - gray[:, i - 1])))

        horizontal_diffs = []
        for j in range(block_size, height_px, block_size):
            horizontal_diffs.append(np.mean(np.abs(gray[j, :] - gray[j - 1, :])))

        blockiness = (np.mean(vertical_diffs) + np.mean(horizontal_diffs)) / 2

        if blockiness > 15:
            pixel_status = "Highly Pixelated"
        elif blockiness > 4:
            pixel_status = "Moderately Pixelated"
        else:
            pixel_status = "Not Pixelated"

        return {
            'dimensions': f"{width_px}x{height_px} pixels",
```

```
            'dpi': round(avg_dpi, 1),
            'dpi_status': dpi_status,
            'blockiness_score': round(blockiness, 2),
            'pixelation_status': pixel_status,
            'dpi_score': dpi_score
        }

    def pdf_to_image(pdf_path, dpi=300):
        """Converts the first page of a PDF to an OpenCV image"""
        pages = convert_from_path(pdf_path, dpi=dpi)
        first_page = pages[0]
        img = np.array(first_page)
        img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
        return img

    def extract_book_id(file_reference):
        """Extracts ISBN (Book ID) from filename"""
        filename = os.path.basename(file_reference)
        match = re.search(r"(\d+)_", filename)
        if match:
            return int(match.group(1))
        return None

    def pick_email_template(overlap_flag, safe_margin_flagged, dpi_status):
        """Select appropriate email template based on issues"""

        email_templates = {
            "badge_only": "Hello Amra,\n\n⚠ Your book cover has text overlapping the award badge area.\nPlease move the text above the badge to resolve the issue.\n\nThank you!",
            "margin_only": "Hello Amra,\n\n⚠ Some text is outside the safe margin zone.\nPlease adjust your text to stay within the safe margins (3mm on sides).\n\nThank you!",
            "quality_only": "Hello Amra,\n\n❌ Your cover image quality is too low for print.\nPlease upload a higher resolution version (≥300 DPI).\n\nThank you!",
            "badge_and_margin": "Hello Amra,\n\n⚠ Your cover has text overlapping the badge and outside safe margins.\nPlease fix both issues.\n\nThank you!",
            "badge_and_quality": "Hello Amra,\n\n⚠ Your cover text overlaps the badge and image quality is too low.\nPlease fix both issues.\n\nThank you!",
            "margin_and_quality": "Hello Amra,\n\n⚠ Text outside safe margins and image quality too low.\nPlease fix both issues.\n\nThank you!",
            "all_three_issues": "Hello Amra,\n\n⚠ Your cover has three issues: badge overlap, unsafe margins, and low quality.\nPlease correct all issues.\n\nThank you!",
            "pass": "Hello Amra,\n\n✓ Your cover passed all checks and is ready for publishing!\n\nThank you!"
        }

        dpi_issue = dpi_status != "✓ EXCELLENT - Print Ready"

        if overlap_flag and safe_margin_flagged and dpi_issue:
            return email_templates["all_three_issues"]
        elif overlap_flag and safe_margin_flagged:
            return email_templates["badge_and_margin"]
```

```
        elif overlap_flag and dpi_issue:
            return email_templates["badge_and_quality"]
        elif safe_margin_flagged and dpi_issue:
            return email_templates["margin_and_quality"]
        elif overlap_flag:
            return email_templates["badge_only"]
        elif safe_margin_flagged:
            return email_templates["margin_only"]
        elif dpi_issue:
            return email_templates["quality_only"]
        else:
            return email_templates["pass"]

    # ==================== MAIN PROCESSING FUNCTION ====================

    def process_book_cover(image_path):
        """Process a single book cover image and return results"""

        print(f"\n{'=' * 70}")
        print(f"PROCESSING: {os.path.basename(image_path)}")
        print(f"{'=' * 70}")

        # Load image (PDF or image file)
        file_ext = os.path.splitext(image_path)[1].lower()

        if file_ext == ".pdf":
            img = pdf_to_image(image_path)
        elif file_ext in [".png", ".jpg", ".jpeg"]:
            img = cv2.imread(image_path)
        else:
            print(f"❌ Unsupported file format: {file_ext}")
            return None

        if img is None:
            print(f"❌ Error loading image: {image_path}")
            return None

        height, width = img.shape[:2]
        print(f"Image dimensions: {width}x{height} pixels")

        # -------------------- OCR Detection --------------------
        print("\n[1/4] Running OCR detection...")
        text_detections = reader.readtext(img)
        print(f"  Detected {len(text_detections)} text regions")

        # -------------------- Badge Area Detection --------------------
        print("\n[2/4] Checking badge area overlap...")
        bottom_strip_height = 106

        badge_x1 = width // 2
        badge_x2 = width
        badge_y1 = height - bottom_strip_height
        badge_y2 = height
        badge_coords = badge_x1, badge_y1, badge_x2, badge_y2
```

```
    safe_margin_flag_foverlap, overlap_flag, overlap_text =
find_overlap_text_dual(
        text_detections, badge_coords, EXPECTED_TEXT1, EXPECTED_TEXT2
    )

    # -------------------- Extract Right Half --------------------
    print("\n[3/4] Analyzing image quality (right half)...")
    right_half = img[0:height, width // 2:width]

    quality_results = comprehensive_image_quality(right_half, actual_dpi=100)

    print(f"  Dimensions: {quality_results['dimensions']}")
    print(f"  Estimated DPI: {quality_results['dpi']}")
    print(f"  DPI Status: {quality_results['dpi_status']}")
    print(f"  Pixelation Score: {quality_results['blockiness_score']}")
    print(f"  Pixelation Status: {quality_results['pixelation_status']}")

    # -------------------- Safe Margin Check --------------------
    print("\n[4/4] Checking safe margins (3mm)...")
    text_detections_right = reader.readtext(right_half)

    actual_dpi = 100
    safe_margin_px = (3 / 25.4) * actual_dpi
    height_px, width_px = right_half.shape[:2]

    safe_margin_flagged = False
    unsafe_texts = []

    for bbox, text, score in text_detections_right:
        xs = [pt[0] for pt in bbox]
        if any(x < safe_margin_px or x > width_px - safe_margin_px for x in
xs):
            print(f"  ⚠ Text '{text}' is in unsafe margin")
            safe_margin_flagged = True
            unsafe_texts.append(text)

    if not safe_margin_flagged:
        print("  ✓ All text within safe margins")

    safe_margin_message = "✓ All text within safe margins" if not
safe_margin_flagged else f"⚠ Unsafe margins: {', '.join(unsafe_texts)}"

    # Apply overlap flag to margin status
    if safe_margin_flag_foverlap:
        safe_margin_flagged = True

    # -------------------- Overall Assessment --------------------
    if overlap_flag:
        overall_assessment = "Review Needed"
    elif quality_results['dpi_status'] != "✓ EXCELLENT - Print Ready":
        overall_assessment = "Review Needed"
    elif safe_margin_flagged:
        overall_assessment = "Review Needed"
    else:
```

```
            overall_assessment = "Pass"

        # -------------------- Email Template --------------------
        email_text = pick_email_template(overlap_flag, safe_margin_flagged, quality_results['dpi_status'])

        # -------------------- Prepare Data --------------------
        file_name = os.path.basename(image_path)
        book_id = extract_book_id(image_path)
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        result = {
            "File Name": file_name,
            "Book ID": book_id,
            "Timestamp": timestamp,
            "Author Email": "amranadaf@gmail.com",  # TODO: Extract from filename or metadata
            "Author Name": "Amra",  # TODO: Extract from filename or metadata
            "Overlap Flag": overlap_flag,
            "Overlap Text": overlap_text if overlap_flag else "",
            "Safe Margin Flag": safe_margin_flagged,
            "Safe Margin Message": safe_margin_message,
            "DPI of Image": quality_results['dpi'],
            "Pixelation Score": quality_results['blockiness_score'],
            "Pixelation Status": quality_results['pixelation_status'],
            "DPI Status": quality_results['dpi_status'],
            "Overall Assessment": overall_assessment,
            "Email Text": email_text
        }

        # -------------------- Final Report --------------------
        print(f"\n{'=' * 70}")
        print("FINAL ASSESSMENT")
        print(f"{'=' * 70}")
        print(f"Status: {overall_assessment}")
        print(f"DPI: {quality_results['dpi']}")
        print(f"Pixelation: {quality_results['pixelation_status']}")
        print(f"Badge Overlap: {'Yes' if overlap_flag else 'No'}")
        print(f"Safe Margins: {'No' if safe_margin_flagged else 'Yes'}")
        print(f"{'=' * 70}\n")

        return result

    def insert_to_airtable(result: dict):
        """Insert record into Airtable"""
        try:
            record = table.create(result)
            print("✓ Record created:", record['id'])
            return record['id']
        except Exception as e:
            print("❌ Error uploading to Airtable:", e)
            return None

    #----------------------------function for sending mail--------------------
    EMAIL_ADDRESS = "amranadaf@gmail.com"
```

```
EMAIL_PASSWORD = "#"  # Use app password if using Gmail

def send_email(email_text, recipient_email='amraspacestars@gmail.com', subject="Book Cover Feedback"):
    """Send email to author"""
    try:
        # Compose email
        msg = MIMEMultipart()
        msg['From'] = EMAIL_ADDRESS
        msg['To'] = recipient_email
        msg['Subject'] = subject
        msg.attach(MIMEText(email_text, 'plain'))

        # Connect to Gmail SMTP server
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()
        server.login(EMAIL_ADDRESS, EMAIL_PASSWORD)
        server.send_message(msg)
        server.quit()

        print(f"✓ Email sent to {recipient_email}")
        return True
    except Exception as e:
        print(f"❌ Failed to send email to {recipient_email}: {e}")
        return False

def authenticate_google_drive():
    """Authenticate with Google Drive"""
    print("Authenticating with Google Drive...")

    gauth = GoogleAuth()
    gauth.LoadCredentialsFile("mycreds.txt")

    if gauth.credentials is None:
        print("First time authentication - browser will open...")
        gauth.LocalWebserverAuth()
    elif gauth.access_token_expired:
        print("Refreshing expired credentials...")
        gauth.Refresh()
    else:
        print("Using saved credentials...")
        gauth.Authorize()

    gauth.SaveCredentialsFile("mycreds.txt")
    drive = GoogleDrive(gauth)
    print("✓ Authenticated successfully\n")

    return drive

# ==================== MAIN MONITORING LOOP ====================

def main():
    """Main function to monitor Google Drive and process new files"""

```

```
    # Authenticate with Google Drive
    drive = authenticate_google_drive()

    print(f"Monitoring folder ID: {FOLDER_ID}")
    print(f"Checking every {POLL_INTERVAL} seconds")
    print(f"Press Ctrl+C to stop\n")
    print("=" * 70)

    while True:
        try:
            # List all files in the folder
            file_list = drive.ListFile({
                'q': f"'{FOLDER_ID}' in parents and trashed=false"
            }).GetList()

            for file in file_list:
                file_id = file['id']
                file_title = file['title']

                # Check if it's a new image/PDF file
                if file_id not in processed_files and
file_title.lower().endswith(('.png', '.jpg', '.jpeg', '.pdf')):
                    print(f"\n🔍 NEW FILE DETECTED: {file_title}")

                    # Download file
                    local_path = os.path.join(DOWNLOAD_DIR, file_title)
                    file.GetContentFile(local_path)
                    print(f"📥 Downloaded to: {local_path}")

                    # Process the image
                    result = process_book_cover(local_path)

                    if result:
                        # Insert into Airtable
                        insert_to_airtable(result)  # Uncomment when ready

                        # TODO: Send email notification
                        send_email(result['Email Text'], result['Author
Email'])

                        print(f"✓ Completed processing: {file_title}")

                    # Mark as processed
                    processed_files.add(file_id)
                    save_processed_file(file_id)

            # Wait before next poll
            print(f"\n⏳ Waiting {POLL_INTERVAL} seconds before next
check...")
            time.sleep(POLL_INTERVAL)

        except KeyboardInterrupt:
            print("\n\n⚠️ Stopping monitor...")
            break
        except Exception as e:
```

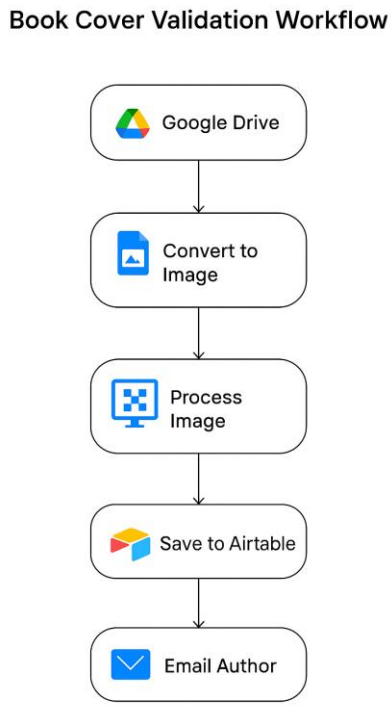
```
            print(f"⚠ Error: {e}")
            time.sleep(POLL_INTERVAL)

# ==================== ENTRY POINT ====================

if __name__ == "__main__":
    # For testing a single image locally:
    # result = process_book_cover("test_image.png")
    # print(result)

    # For automated Google Drive monitoring:
    main()
```

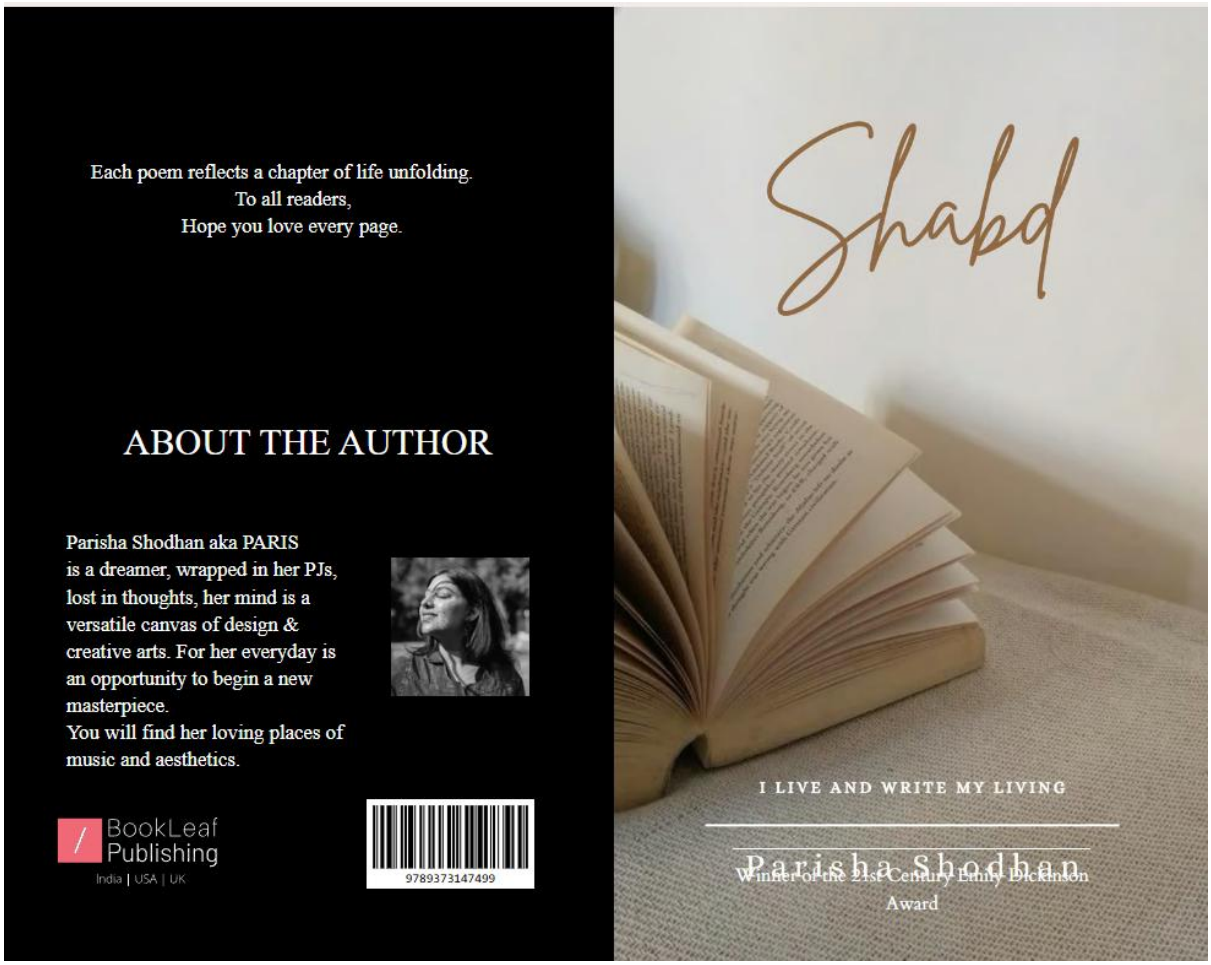
This code has the entire cycle of:



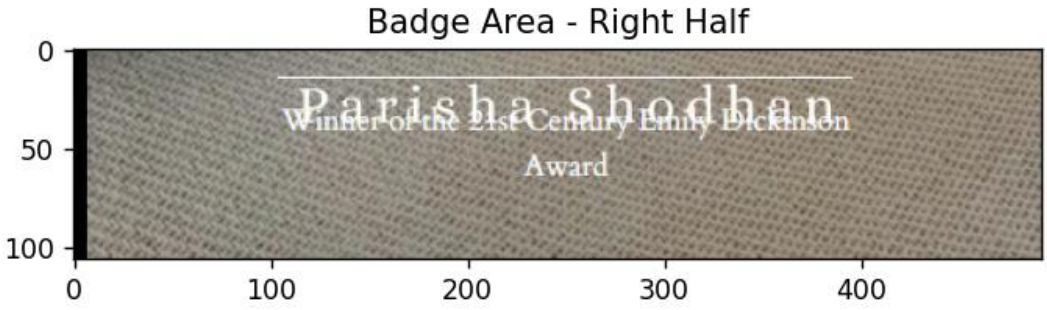
INPUT and OUTPUT examples:

All images given as examples for testing in the challenge:

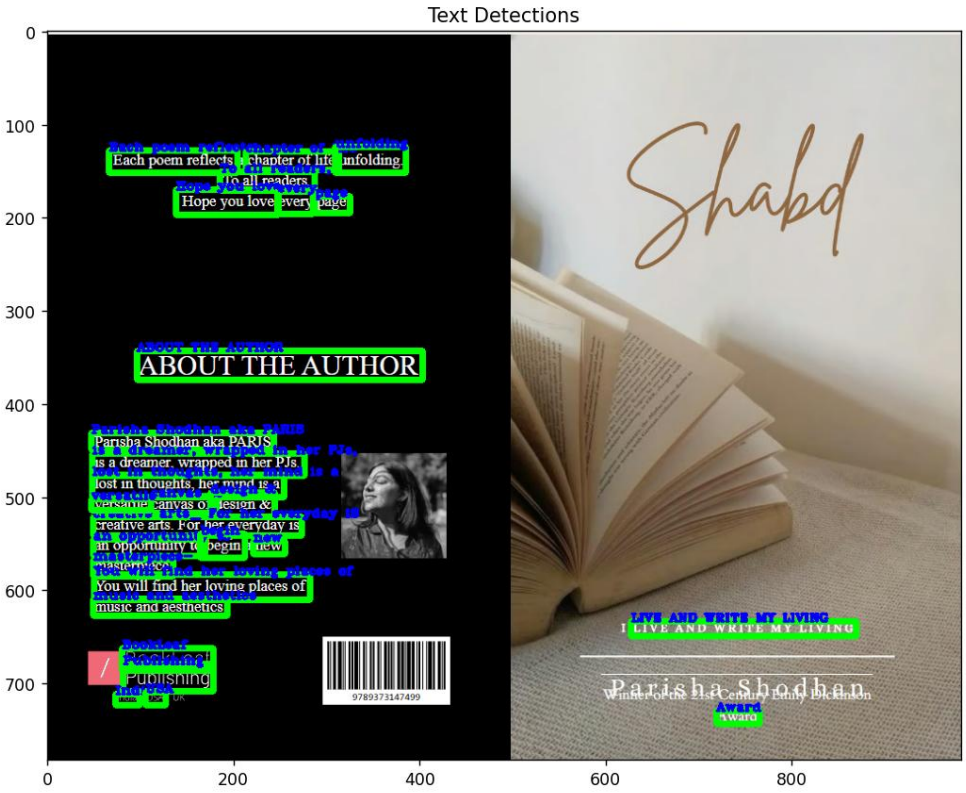
A) Test image1 : img(28).png



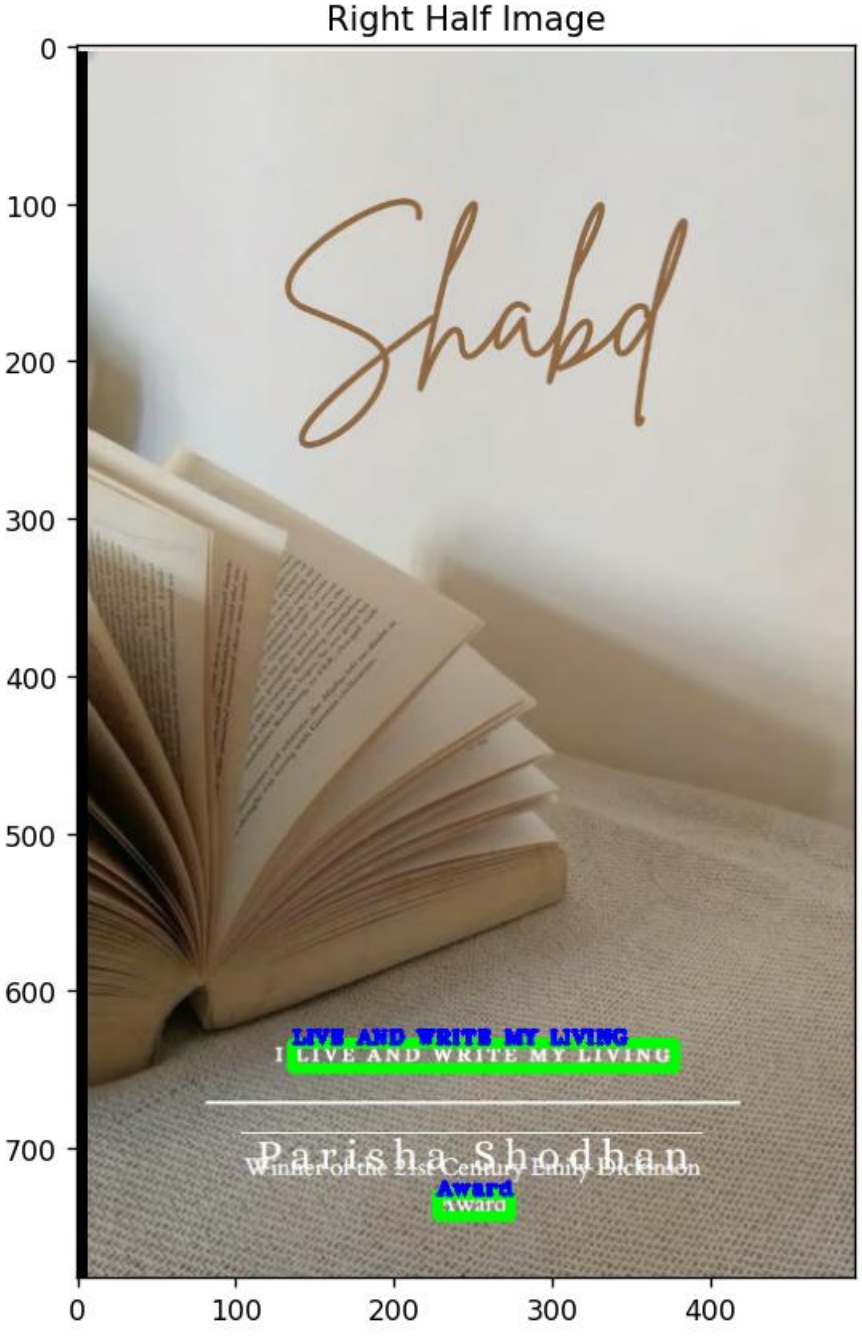
We get the badge region



Detecting texts found in the entire cover using easyOCR



We extract the right half of the image for detection

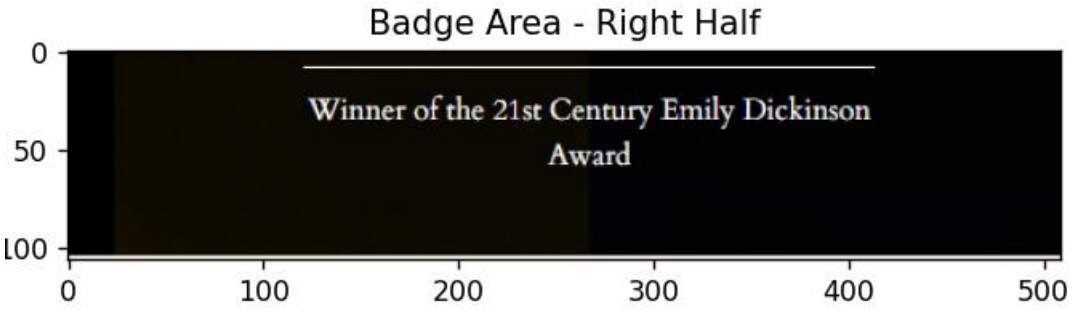

Right Half Image
Shabd
LIVE AND WRITE MY LIVING
I LIVE AND WRITE MY LIVING
Parisha Shodhan
Award
0
100
200
300
400
500
600
700

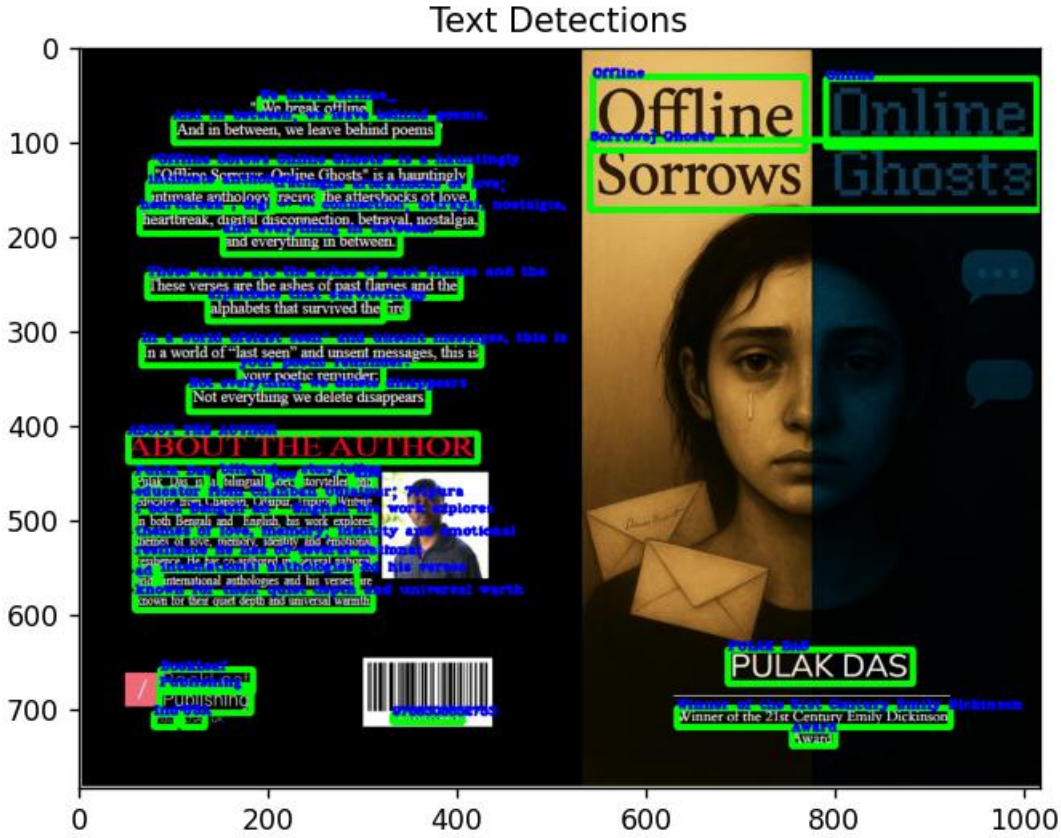
Output:

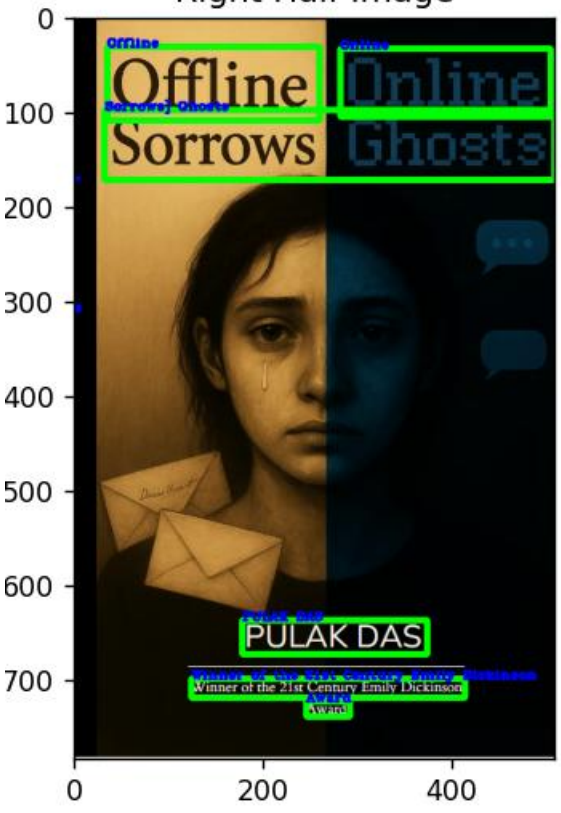
```
C:\Users\AAHILAHMED\PycharmProjects\pythonProject4\venv\Scripts\python.exe "C:\Users\AAHILAHMED\PycharmProjects\pythonProject4\Henge challenge.py"
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is much faster with a GPU.
Text detections: [[[[66, 128], [204, 128], [204, 152], [66, 152]], 'Each poem reflects', 0.8144779864751068), ([[213, 129], [313, 129], [313, 149], [213, 149]], 'chapter of lif
Pixelation score: 4.77
→ Moderately pixelated
Detected: 'Waazoik hsaecSh 04l be.h', similarity1: 0.06, similarity2: 0.14
⚠ Overlapping text found: 'Waazoik hsaecSh 04l be.h'
Detected: 'Award', similarity1: 0.19, similarity2: 1.00

Process finished with exit code 0
```

B) Test image: image(31).png



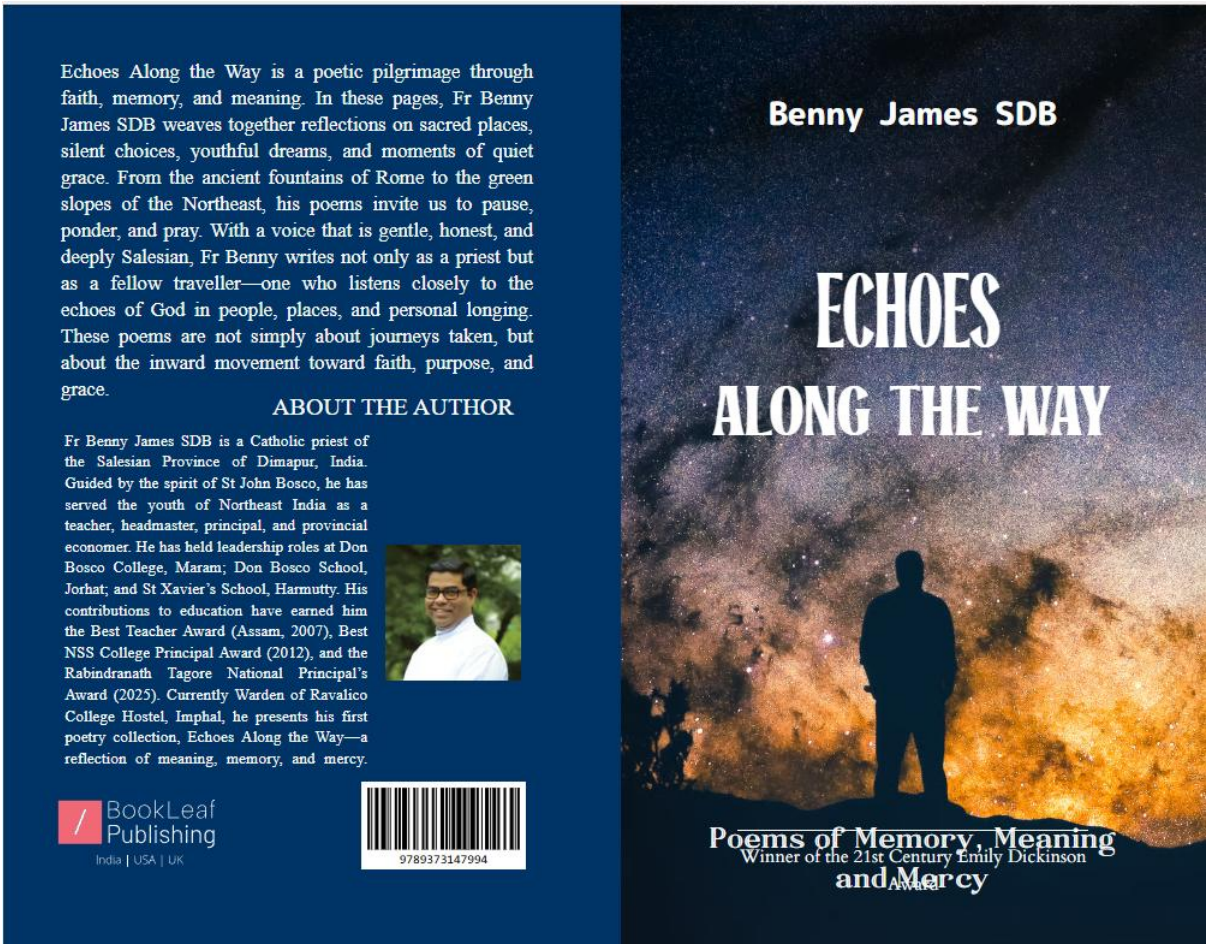

Text Detections
"We break offline.
And in between, we leave behind poems."
heartbreak, digital disconnection, betrayal, nostalgia,
and everything in between.
These verses are the ashes of past flames and the
alphabets that survived the fire.
Not everything we delete disappears.
ABOUT THE AUTHOR
Offline
Sorrows
Online
Ghosts
PULAK DAS
Winner of the 21st Century Emily Dickinson
Award
0
100
200
300
400
500
600
700
0
200
400
600
800
1000

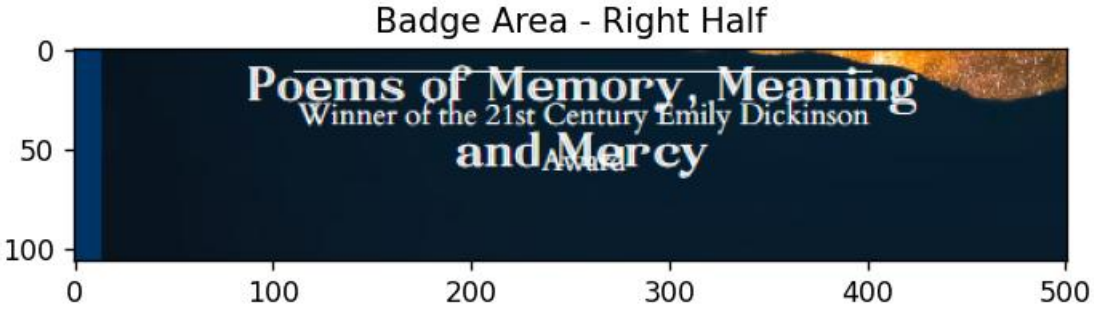

Right Half Image
Offline
Sorrows
Online
Ghosts
PULAK DAS
Winner of the 21st Century Emily Dickinson
Award
0
100
200
300
400
500
600
700
0
200
400

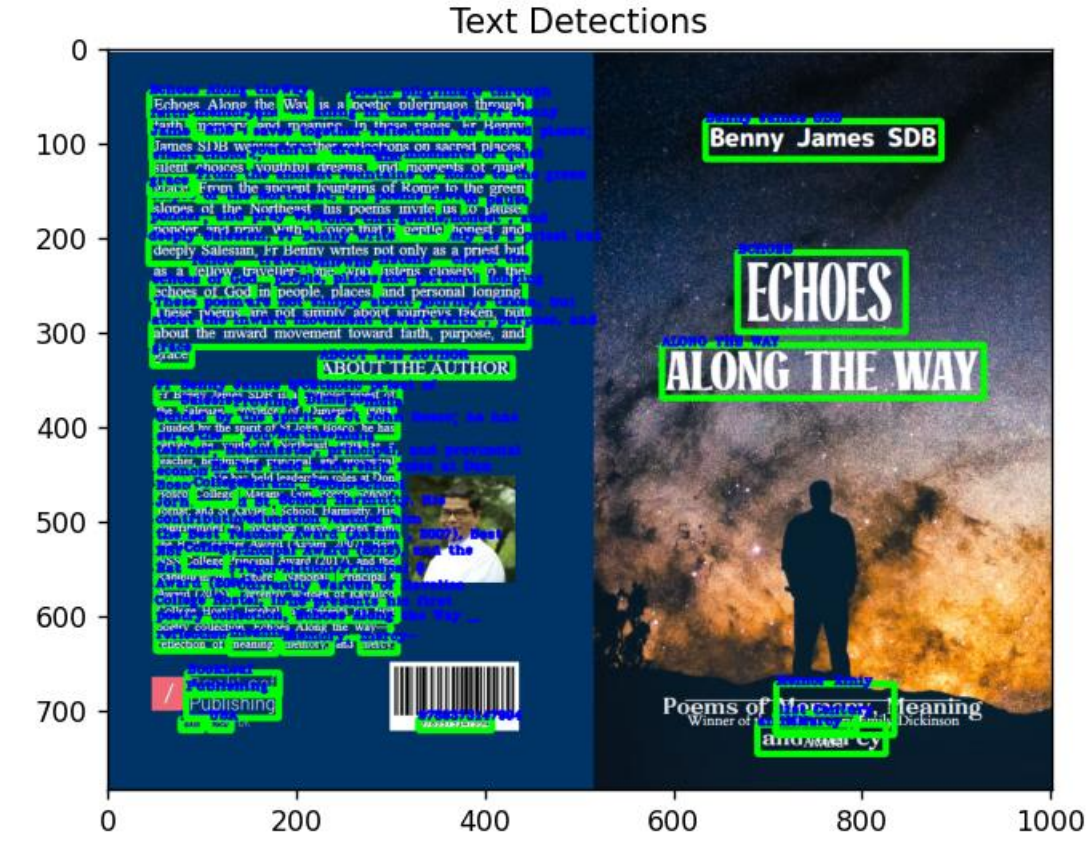
Output:

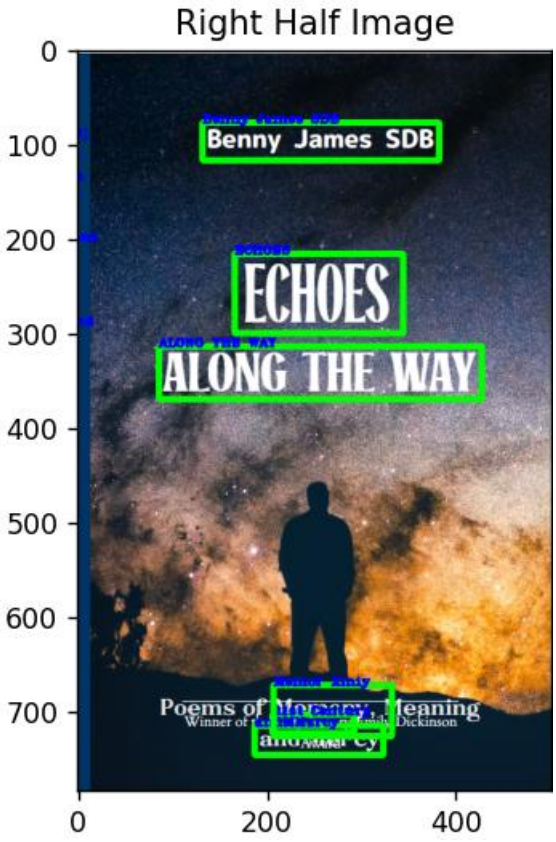
```
C:\Users\AAHILAHMED\PycharmProjects\pythonProject4\venv\Scripts\python.exe "C:
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is m
Text detections: [([[191, 53], [307, 53], [307, 73], [191, 73]], 'We break off
Pixelation score: 6.17
→ Moderately pixelated
Detected: 'Winner of the 21st Century Emily Dickinson', similarity1: 0.93, sim
Detected: 'Award', similarity1: 0.19, similarity2: 1.00
✅ No unwanted overlaps detected.
```

C)Test image: img(32).png




Badge Area - Right Half
Poems of Memory, Meaning
Winner of the 21st Century Emily Dickinson
and Mercy
Award


Text Detections
Benny James SDB
ECHOES
ALONG THE WAY
ABOUT THE AUTHOR
Publishing

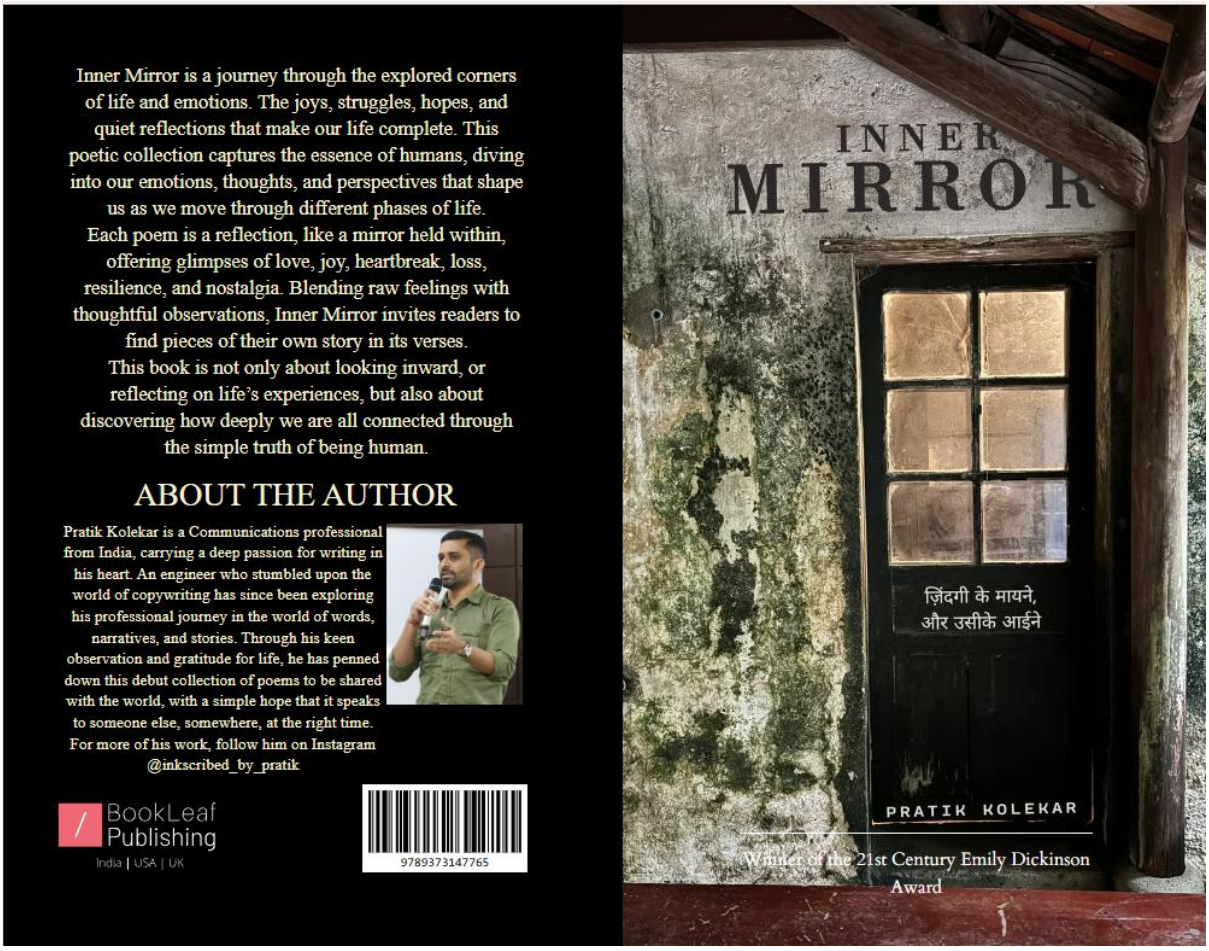


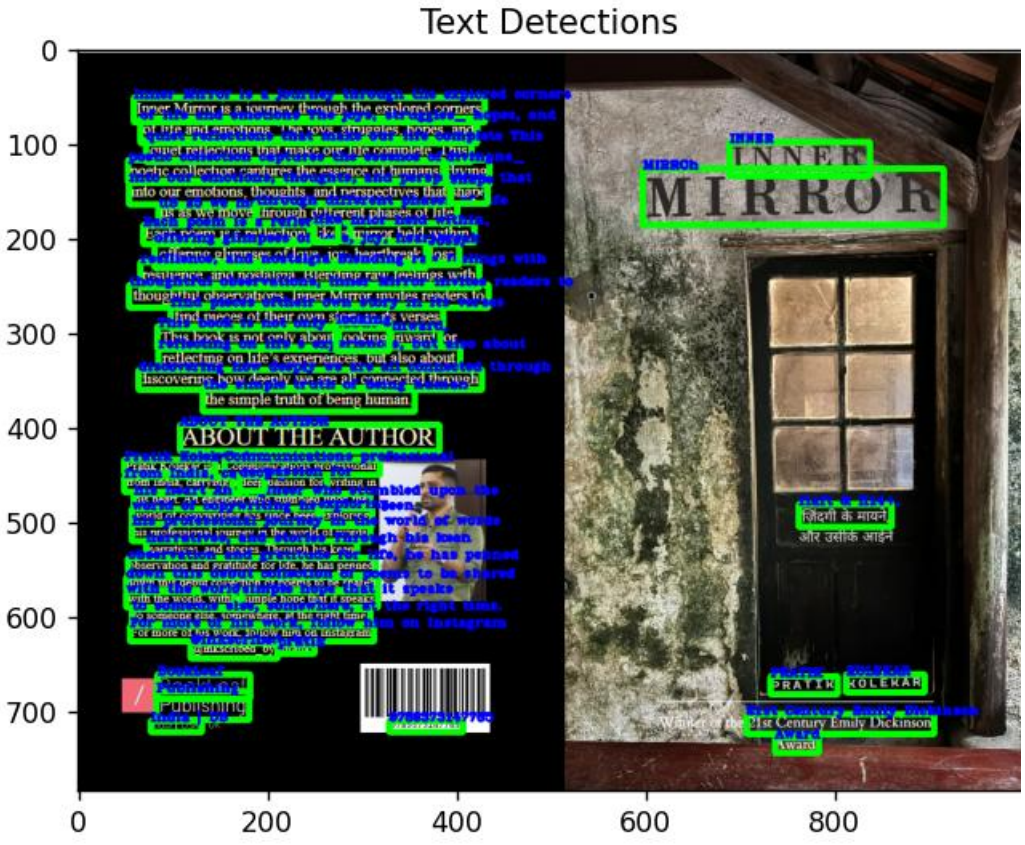
Output:

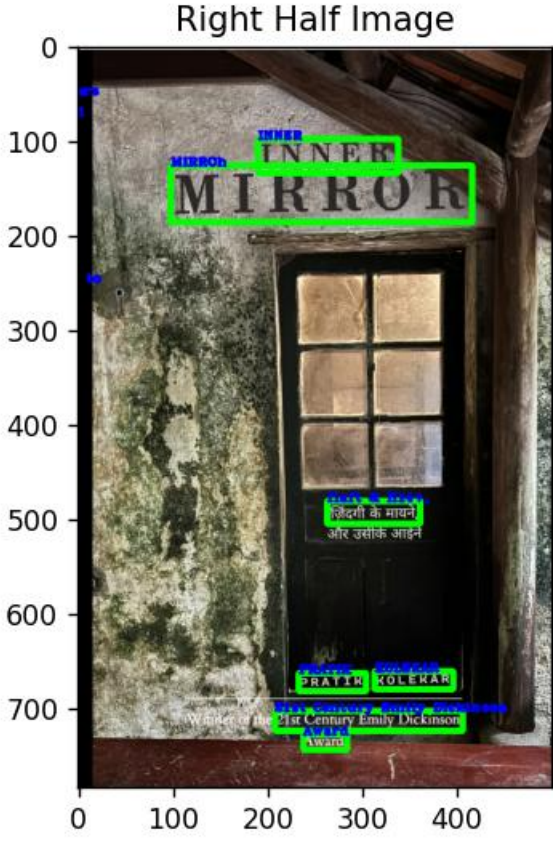
```
C:\Users\AAHILAHMED\PycharmProjects\pythonProject4\venv\Scripts\pytho
Neither CUDA nor MPS are available - defaulting to CPU. Note: This mo
Text detections: [([[44, 46], [182, 46], [182, 72], [44, 72]], 'Echoe
Pixelation score: 12.75
→ Moderately pixelated
Detected: 'PoemsoPfe !', similarity1: 0.20, similarity2: 0.00
⚠ Overlapping text found: 'PoemsoPfe !'
Detected: '21st Century', similarity1: 0.40, similarity2: 0.12
⚠ Overlapping text found: '21st Century'
Detected: 'Ieaning', similarity1: 0.07, similarity2: 0.17
⚠ Overlapping text found: 'Ieaning'
Detected: 'andMMarcy', similarity1: 0.11, similarity2: 0.43
⚠ Overlapping text found: 'andMMarcy'
Detected: 'Memor Xmiy', similarity1: 0.14, similarity2: 0.13
⚠ Overlapping text found: 'Memor Xmiy'

Process finished with exit code 0
```

D) Test image 4: img(34).png




Text Detections
Inner Mirror is a journey through the explored corners
into our emotions, thoughts, and perspectives that
the simple truth of being human
ABOUT THE AUTHOR
INNER
MIRROR
PRATIK
KOLEKAR
0
100
200
300
400
500
600
700
0
200
400
600
800


Right Half Image
INNER
MIRROR
PRATIK
KOLEKAR
0
100
200
300
400
500
600
700
0
100
200
300
400

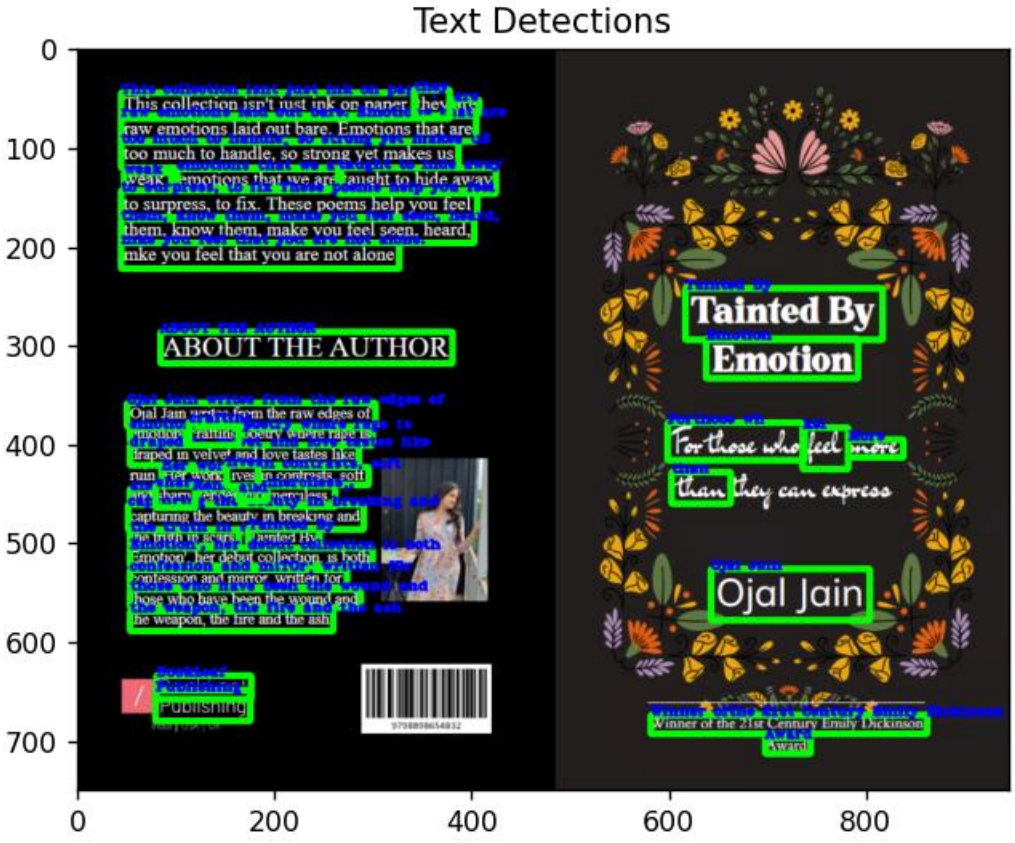
Output:

```
C:\Users\AAHILAHMED\PycharmProjects\pythonProject4\venv\Scripts\pytho
Neither CUDA nor MPS are available - defaulting to CPU. Note: This mo
Text detections: [([[59, 50], [430, 50], [430, 76], [59, 76]], 'Inner
Pixelation score: 15.39
→ Highly pixelated
Detected: 'PRATIK', similarity1: 0.07, similarity2: 0.18
⚠ Overlapping text found: 'PRATIK'
Detected: 'KOLEKAR', similarity1: 0.15, similarity2: 0.33
⚠ Overlapping text found: 'KOLEKAR'
Detected: '21st Century Emily Dickinson', similarity1: 0.74, similari
⚠ Overlapping text found: '21st Century Emily Dickinson'
Detected: 'Award', similarity1: 0.19, similarity2: 1.00

Process finished with exit code 0
```

E)Test image 5: img(37).png


Badge Area - Right Half
Winner of the 21st Century Emily Dickinson
Award


Text Detections
This collection isn't just ink on paper, they are
raw emotions laid out bare. Emotions that are
too much to handle, so strong yet makes us
weak, emotions that we are taught to hide away
to surpress, to fix. These poems help you feel
them, know them, make you feel seen, heard,
mke you feel that you are not alone
ABOUT THE AUTHOR
Ojal Jain writes from the raw edges of
emotion, crafting poetry where raw is
draped in velvet and love tastes like
capturing the beauty in breaking and
emotion, her debut collection, is both
confession and mirror, written for
those who have been the wound and
he weapon, the fire and the ash
Publishing
Tainted By
Emotion
For those who feel more
than they can express
Ojal Jain
Winner of the 21st Century Emily Dickinson
Award

Output:

```
C:\Users\AAHILAHMED\PycharmProjects\pythonProject4\venv\Scripts\python.exe
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module
Text detections: [([[44, 44], [342, 44], [342, 70], [44, 70]], 'This colle
Pixelation score: 6.40
→ Moderately pixelated
Detected: 'Winner ofthe 21st Century Emily Dickinson', similarity1: 0.92,
Detected: 'Award', similarity1: 0.19, similarity2: 1.00
✅ No unwanted overlaps detected.

Process finished with exit code 0
```

F) Test image 6: img(35).png

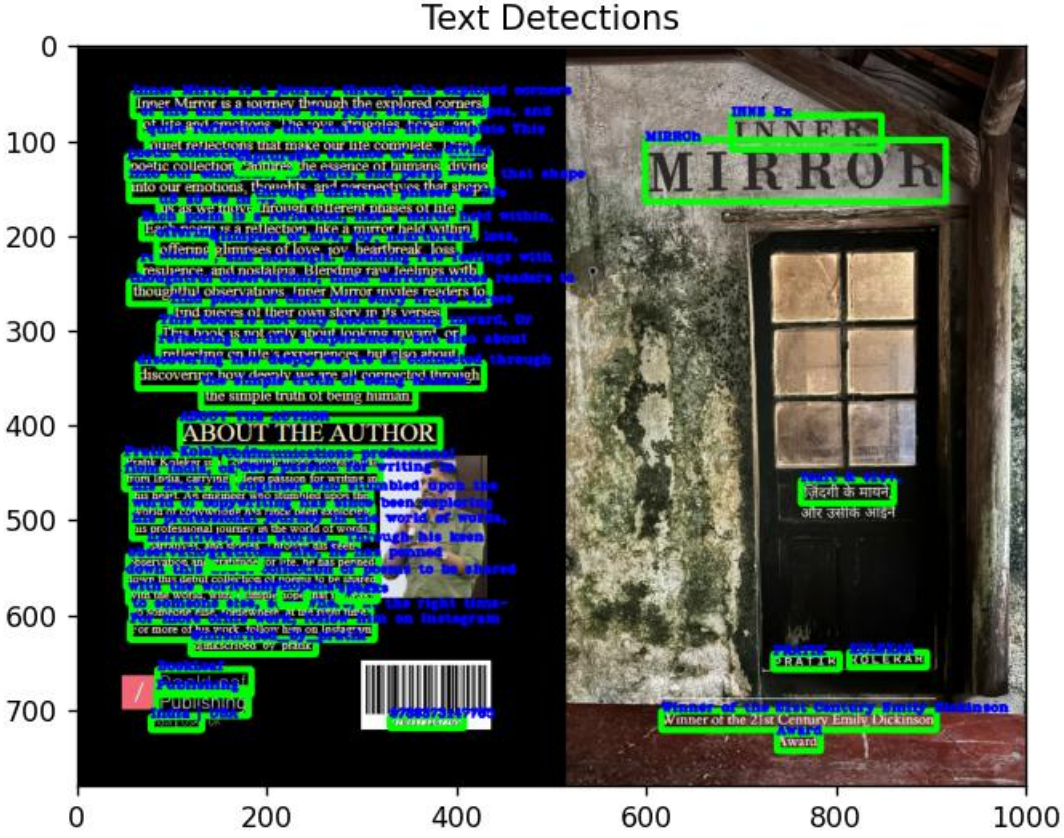
Inner Mirror is a journey through the explored corners of life and emotions. The joys, struggles, hopes, and quiet reflections that make our life complete. This poetic collection captures the essence of humans, diving into our emotions, thoughts, and perspectives that shape us as we move through different phases of life.
Each poem is a reflection, like a mirror held within, offering glimpses of love, joy, heartbreak, loss, resilience, and nostalgia. Blending raw feelings with thoughtful observations, Inner Mirror invites readers to find pieces of their own story in its verses.
This book is not only about looking inward, or reflecting on life's experiences, but also about discovering how deeply we are all connected through the simple truth of being human.

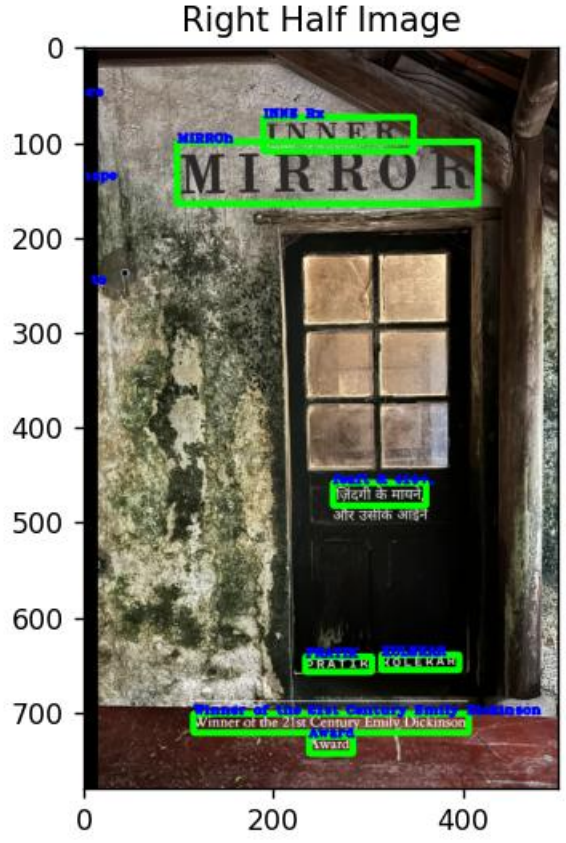
ABOUT THE AUTHOR

Pratik Kolekar is a Communications professional from India, carrying a deep passion for writing in his heart. An engineer who stumbled upon the world of copywriting has since been exploring his professional journey in the world of words, narratives, and stories. Through his keen observation and gratitude for life, he has penned down this debut collection of poems to be shared with the world, with a simple hope that it speaks to someone else, somewhere, at the right time.
For more of his work, follow him on Instagram @inkscribed_by_pratik

BookLeaf Publishing
India | USA | UK

9789373147765


Text Detections
Inner Mirror is a journey through the explored corners
MIRROR
ABOUT THE AUTHOR
Winner of the 21st Century Emily Dickinson
Award


Right Half Image
MIRROR
Winner of the 21st Century Emily Dickinson
Award

Output:

```
C:\Users\AAHILAHMED\PycharmProjects\pythonProject4\venv\Scripts\python.exe
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module
Text detections: [([[58, 50], [430, 50], [430, 74], [58, 74]], 'Inner Mirr
Pixelation score: 15.05
→ Highly pixelated
Detected: 'Winner of the 21st Century Emily Dickinson', similarity1: 0.93,
Detected: 'Award', similarity1: 0.19, similarity2: 1.00
✅ No unwanted overlaps detected.

Process finished with exit code 0
```

G) Test image 7: dawn-9856375.jpg (High-quality and resolution image)


Badge Area - Right Half
0
100
0
500
1000
1500
2000
2500


Text Detections
0
500
1000
1500
2000
2500
3000
3500
0
1000
2000
3000
4000
5000

Output:

```
C:\Users\AAHILAHMED\PycharmProjects\pythonProject4\venv\Scripts\py
Neither CUDA nor MPS are available - defaulting to CPU. Note: This
Text detections: []
Pixelation score: 3.27
→ Not pixelated
✅ No unwanted overlaps detected.

Process finished with exit code 0
```

Book Cover Validator - User Guide & Testing Documentation

Overview

This Gradio-based web application validates uploaded book cover images against specific print-ready criteria, including badge text overlaps, safe margins for text placement, and overall image quality (DPI, pixelation, and blockiness). It's designed for quick, automated checks to ensure covers meet publishing standards before printing.

The app processes images using OCR (via EasyOCR), computer vision (OpenCV), and quality metrics. Results are displayed in a clear Markdown report.

Key Features:

- Badge Overlap Detection Checks for unwanted text in the designated badge region (bottom-right corner).

- Safe Margin Check: Ensures no text falls within 3mm of the left/right edges on the right half of the cover.

- Image Quality Assessment: Evaluates DPI, pixelation, and blockiness for print suitability.

- Final Decision: "Pass" or "Review Needed" based on all checks.

UI Walkthrough

Step 1: Upload an Image

- Input: Click the "Upload Book Cover (JPG/PNG)" area to select an image file.

  - Supported Formats: JPG, PNG (images only—for now, to simplify testing and reduce memory usage during deployment).

  - Recommendations: Use high-resolution images (ideally 300 DPI, i.e approx. 1000 pixel x 2000 pixel wide and across). Ensure the cover is upright and text is legible.

- Preview: The uploaded image will appear below the upload area for confirmation.

Step 2: Run Validation

- Action: Click the "Validate Cover" button (orange primary button) beneath the image preview.

- Processing Time: Expect 10-60 seconds for results.

  - Initial load: EasyOCR model initializes once (may take ~10-20 seconds on first run; subsequent validations are faster).

  - Analysis: OCR scanning, overlap/margin checks, and quality metrics.

- Status: Watch for the report to update in the right panel.

Step 3: View Results

- Output: The "Validation Report" section (right column) displays a formatted Markdown summary.

  - Structure:

    - Overall Decision: Bold "Pass" (✅ all checks clear) or "Review Needed" (⚠️ / ❌ issues found).

    - Badge Overlap:

      - ✅ "No unwanted text overlaps" if clear.

      - ⚠️ "Overlap Detected: `[detected_text]`" if issue (lists all unexpected texts; may include gibberish if pixels merge visually).

    - Safe Margins:

      - ✅ "All text within safe margins" if clear.

      - ⚠️ "Margin Violations: `[text1], [text2]`" if text too close to edges (only lists violating texts; margins are 3mm from left/right on the right half).

    - Image Quality:

      - DPI: Value + status (e.g., " ✅ EXCELLENT - Print Ready" for ≥300 DPI).

      - Pixelation: Status (e.g., "Not Pixelated") + score (0-100).

      - Blockiness Score: Numeric value (lower = better; <4 ideal).

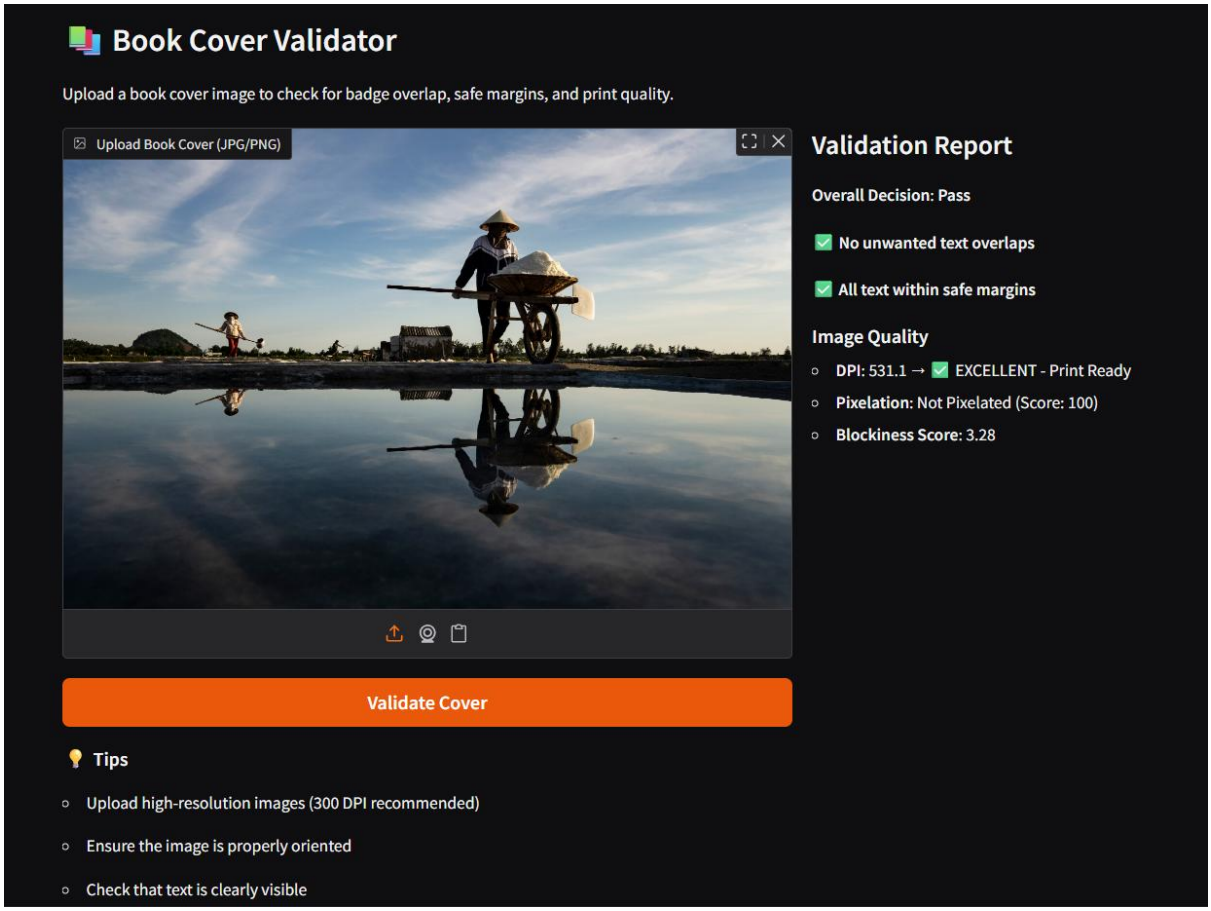
      - Overall Assessment: " ✅ PASS - Suitable for printing" or similar.

- **Notes on Edge Cases:**

  - Badge Overlap Impact: If overlap is detected, safe margins are auto-flagged as violated (for holistic review), but no duplicate margin violation texts appear details are in the overlap section.

  - Gibberish Text: In badge overlaps, merged pixels (e.g., from direct overlap) may OCR as nonsense strings. This is expected and flags the issue accurately.

Graphical representation of test UI



Public Link for testing:

https://huggingface.co/spaces/amranad/book-cover-validator

Thank you for reading the documentation till the end,

Happy testing!