NATIONAL INSTITUTE OF TECHNOLOGY DELHI

Department of Computer Science and Engineering

Industry Internship Training (AI/ML Domain)

Tagging documents for Search Optimization

Submitted by	Under the guidance of
Krishal Prasad	Ms. Sushmita Chauhan
231210061	(Big Data Solution Architect, NIC Delhi)

ABSTRACT

This internship project, conducted at the National Informatics Centre (NIC) Delhi, focuses on optimizing document searching in **GovDrive** by leveraging advanced computer vision and machine learning techniques. The proposed system integrates **YOLOv8-small** for object detection and **EasyOCR** for multilingual text extraction, enabling the generation of accurate and context-specific metadata tags. Comparative analysis of **PaddleOCR**, **EasyOCR**, and **TesseractOCR** was performed to identify the most suitable OCR method for varied document types, with EasyOCR emerging as the optimal choice for this application. Future work could explore multimodal AI models like **LLaVA** to unify detection, recognition, and tagging in a single intelligent framework.

Keywords: *Object Detection, YOLOv8, Optical Character Recognition, EasyOCR, Metadata Tagging, GovDrive, Computer Vision*

DECLARATION

I hereby declare that this Internship Report is an authentic record of my own work carried out at the National Informatics Centre, Delhi as a requirement of one semester internship for the award of degree of B. Tech (Computer Science & Engineering) at National Institute of Technology Delhi, under the guidance of Ms. Sushmita Chauhan, in the months of June to August, 2025.

Date: 04/07/2024

Krishal Prasad

(231210061)

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

Ms Sushmita Chauhan

(Big Data Solution Architect, NIC Delhi)

ACKNOWLEDGEMENT

It gives me immense pleasure to write this internship report for the duration of my work at the National Informatics Centre, Delhi. I would like to take this opportunity to thank my Industry Mentor, Ms Sushmita Chauhan, for her continuous guidance in the field of Artificial Intelligence and Computer Vision. She has always been approachable and provided me with the right direction whenever I needed it. I am also grateful to the entire Development team for making my internship experience a valuable one. I learnt a lot from their expertise and experience, and I am grateful for their unwavering support.

I would also like to express my gratitude to Ms Rishika, Ms Dipanshi and Ms Alka, who guided me throughout the course of my internship at NIC, Delhi. The collaboration with other interns like Mr Nakul and Ms Rishika helped me understand the power of teamwork and the importance of having a well bonded team.

INDEX

INTRODUCTION

Research Domain

I have been working at National Informatics Center, Delhi for the past two months(June 10th to August 10th) in the domain of AI/ML. NIC is a government organisation that aims at providing secure and efficient tech support to the government and its subsidiaries. One particular service provided by it is the **GovDrive**, which is an online drive-like service used to store and share data or files across all government organisations.

GovDrive allows users to search for documents based on keywords that might appear in certain documents as text or media. However, searching every document for the presence of a particular set of keywords takes a very long time as it requires reading every document word by word while trying to to check if the word is in the set of keywords provided. In order to optimize this searching process we develop a list of **tags** that can be appended to the metadata of the document which would reduce the search time significantly.

Besides the search time optimization, tagging would also bring in some degree of space optimization. Searching would require the processor to load chunks of document onto the RAM in order to process it. A large document cannot be loaded at once, however, the metadata even for large files, occupies much less space than the whole document, thus optimizing the use of available memory.

Core Components

Computer Vision: Understanding and interpreting the visual information using Computer Science and Artificial Intelligence(AI).

Object Detection: Computer Vision task that aims at finding and locating the objects present in an image.

Optical Character Recognition: Computer Vision task that aims at reading texts from images.

Metadata: Data that defines the information stored in a file.

Tagging: Attaching labels or keywords to facilitate categorisation, searching and retrieval.

METHODOLOGY

Based on the previous discussion, we shall now discuss what methodology should we adopt to achieve our intended goal, i.e. generating tags and adding them to the document metadata. Here, I will share the basic idea and an algorithmic approach for the same. Later under the heading of 'Implementation', some code fragments are attached.

1. **Extracting labels:** Read the image uploaded by the user and using an Object detection model, determine the object(s) present in the image.
2. **Flagging image:** Based on the contents of the image it is flagged as having text or not.
3. **Extracting text:** If the image has text, it is passed through an OCR module to extract the text content.
4. **Dictionary of words:** Combine all the labels detected in the first step and the text content into a dictionary with two keys - labels and text.
5. **Generating tags:** Pass the dictionary generated in step 4 to an LLM module and generate domain specific tags and add them to the metadata of the document.

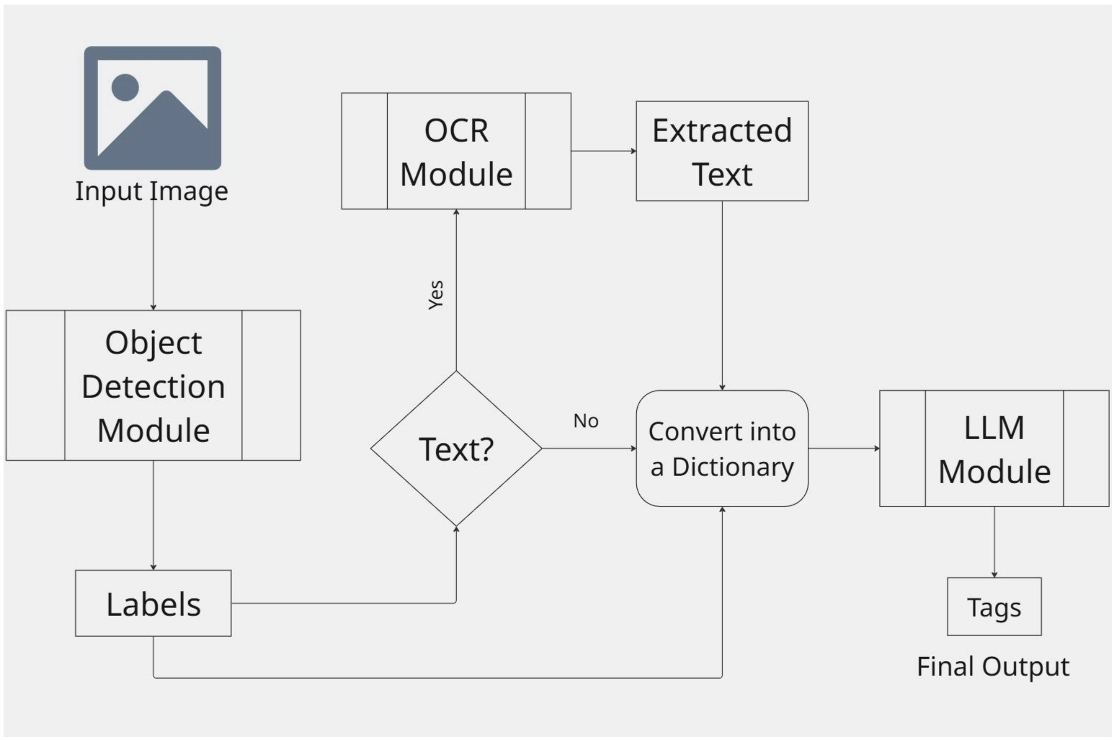


Fig: Generating tags based on image data

OBJECT DETECTION

What is object detection?

Object detection is a computer vision technique that enables machines to locate and identify multiple objects within an image or video frame. Unlike image classification which gives a single label to the whole image, object detection determines the location and the labels of multiple objects in an image.

Two key components of object detection module are:-

1. Object Localization - It refers to locating the object in the image and determining its position using bounding boxes.
2. Object Classification - Labelling of the detected object is known as object classification.

There are multiple object detection techniques available, however, for my project I have considered a deep learning based approach. Deep learning based approaches can be broadly classified into two categories:

1. Two stage detection - RCNN, Faster RCNN, Masked RCNN
2. One stage detection - YOLO

YOLO

You Only Look Once(YOLO) is a deep learning based object detection algorithm. Unlike the classical RCNN, it has the power to detect objects in an image in a single pass by treating detection as a single regression problem. Since its initial release in 2016, YOLO has gone through numerous enhancements, preserving its core principle. For the purpose of my project I have used the YOLOv8 model for its modern architecture and streamlined deployment. It is available in multiple size variants like nano, small, large etc, each setting a trade-off between speed and accuracy. YOLOv8-small (**yolov8s.pt**) version was chosen for its fast inference speed, moderate computational demands and balanced accuracy.

YOLO Annotations

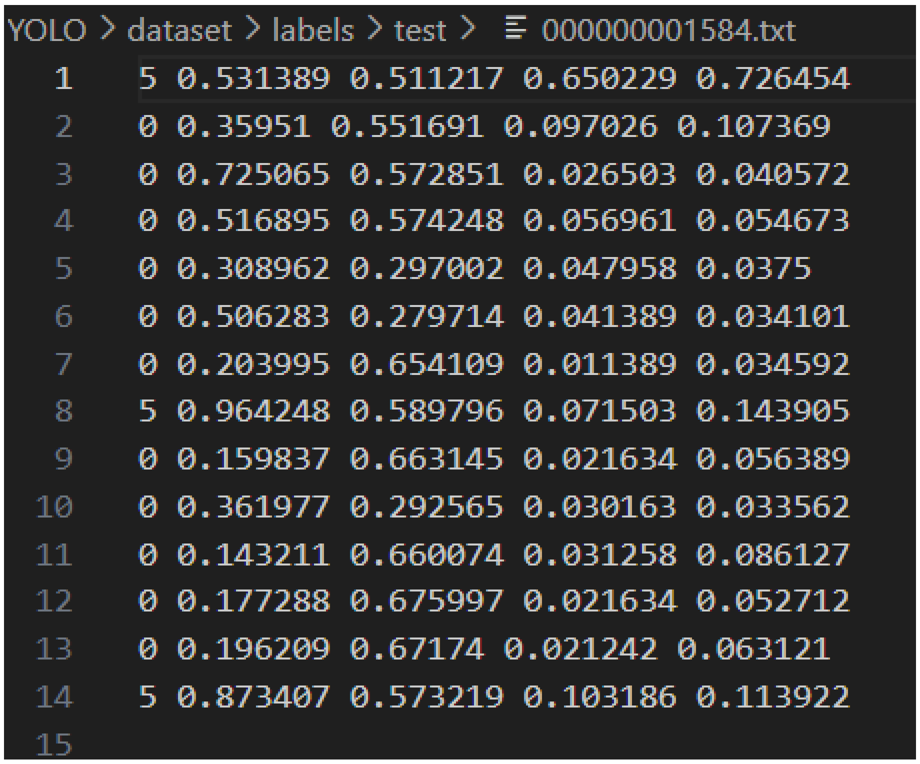
Unlike most ML/DL models, YOLO accepts ground truth annotations as a text (.txt) file instead of a JSON (.json) file. The name of the annotation file should be the same as the image file, therefore for each image a separate annotation (text) file should be available.

The syntax of the annotation is defined in the YOLO format:

1. Each line in a text file corresponds to a different object in the same image.
2. All the classes are given unique class ids which are 0-based integer indexes.
3. The YOLO bounding box doesn't account for maximum and minimum x and y coordinates of the object.
4. The bounding box contains the coordinates of the centre of the object and the total height and width of the object.
5. The bounding box values are all normalised with respect to the dimensions of the image, i.e. the center coordinates, width and height of the object all lie between 0 to 1.

A general YOLO annotation file would like:

<class_id> <x_centre> <y_centre> <width> <height>


YOLO > dataset > labels > test > 000000001584.txt

```
5 0.531389 0.511217 0.650229 0.726454
0 0.35951 0.551691 0.097026 0.107369
0 0.725065 0.572851 0.026503 0.040572
0 0.516895 0.574248 0.056961 0.054673
0 0.308962 0.297002 0.047958 0.0375
0 0.506283 0.279714 0.041389 0.034101
0 0.203995 0.654109 0.011389 0.034592
5 0.964248 0.589796 0.071503 0.143905
0 0.159837 0.663145 0.021634 0.056389
0 0.361977 0.292565 0.030163 0.033562
0 0.143211 0.660074 0.031258 0.086127
0 0.177288 0.675997 0.021634 0.052712
0 0.196209 0.67174 0.021242 0.063121
5 0.873407 0.573219 0.103186 0.113922
```

Fig: YOLO annotations file for an image file named '000000001584.png'

OPTICAL CHARACTER RECOGNITION

What is OCR?

Optical Character Recognition(OCR) is a computer vision task that aims at extracting text from images, scanned documents or printed materials. It enables the computer to “read” text from visual data.

The working of OCR can easily be explained in five simple steps:

1. Image Acquisition: Capture the source image through camera, screenshot or a scanner.
2. Preprocessing: Adjusting brightness, denoising, correcting skew and increasing contrast
3. Text Detection: The system detects the presence of the text and determines the bounding box coordinates.
4. Character Recognition: With the help of pattern recognition or machine learning algorithms, each character is compared to the known symbols to determine the corresponding text.
5. Post-processing: The determined text is corrected and formatted using a dictionary or natural language processing(NLP) to fix errors.

PaddleOCR

Developed by Baidu as a part of the **PaddlePaddle framework**, PaddleOCR is an open source, deep-learning based OCR system. It provides end-to-end OCR capabilities - from detection to recognition, supporting multiple languages making it suitable for a wide range of real-world applications.

However, as PaddleOCR is built on top of the PaddlePaddle framework, it needs the paddlepaddle library to perform its functions. Apart from Tensorflow/Keras for object detection, I had to import PaddlePaddle in order to use the PaddleOCR module. All requirements and dependencies for PaddleOCR are listed below:-

1. Architecture - x86_64(Intel/AMD 64-bit), ARM64 is not supported
2. Python - version 3.8 or higher
3. GPU - requires CUDA or cuDNN
4. PaddlePaddle - version 3 or higher
5. Numpy - version ≥ 1.22 and < 1.24

EasyOCR

EasyOCR is an open-source OCR library developed by Jaided AI. It was built using **PyTorch**. It provides a simple and efficient method to detect and recognise text from visual data with minimal setup. It supports more than 80 languages making it suitable for multilingual OCR applications.

EasyOCR is platform-agnostic beyond its PyTorch dependency. It demands only the PyTorch module and no specific deep learning platform is required. All requirements and dependencies for EasyOCR are listed below:-

1. Architecture - x86_64 (Intel/AMD 64-bit) and ARM64 (including Raspberry Pi)
2. Python - version 3.6 to 3.10
3. PyTorch - version 2.3.1
4. Numpy - version 1.25.2
5. EasyOCR - version 1.7.1

Comparison between OCR Techniques

Here, I am presenting a comparative analysis of the OCR techniques that I reviewed for this project.

Feature	PaddleOCR	EasyOCR	TesseractOCR
Ease of Integration	Easiest - a single Reader.readtext() call is needed	More complex - PaddlePaddle environment setup is required	Simple to run via CLI/wrappers but limited layout handling without extra tools
Accuracy	High	High	Moderate
Language support	80+ languages pretrained	80+ languages + training/fine tuning pipeline	Many languages but some need manual download
Hardware & speed	Fast on CPU, very fast on GPU	Fast on GPU	Optimized for CPU, slower for complex layouts
Training/Fine tuning	Possible with PyTorch	Complete training pipelines provided	Limited

IMPLEMENTATION

After all the discussion in the previous sections, now I will share the code fragments for every task that was carried out for this project. Some code snippets will be followed by the corresponding output that was generated. All the code provided is written in Python3 on VS Code and some on Google Colab. For YOLOv8 fine tuning, Google Colab's GPU was used.

1. YOLOv8 Fine Tuning

The model fine tuning was carried out on the Google Colab's GPU for about 82 minutes running for **50 epochs** with a **batch size of 16**. My google drive was mounted and so the results after every 5 epochs were getting saved at the address: 'MyDrive/yolo_training/train/'. The values of the last and the best epochs were saved as 'last.pt' and 'best.pt' respectively.

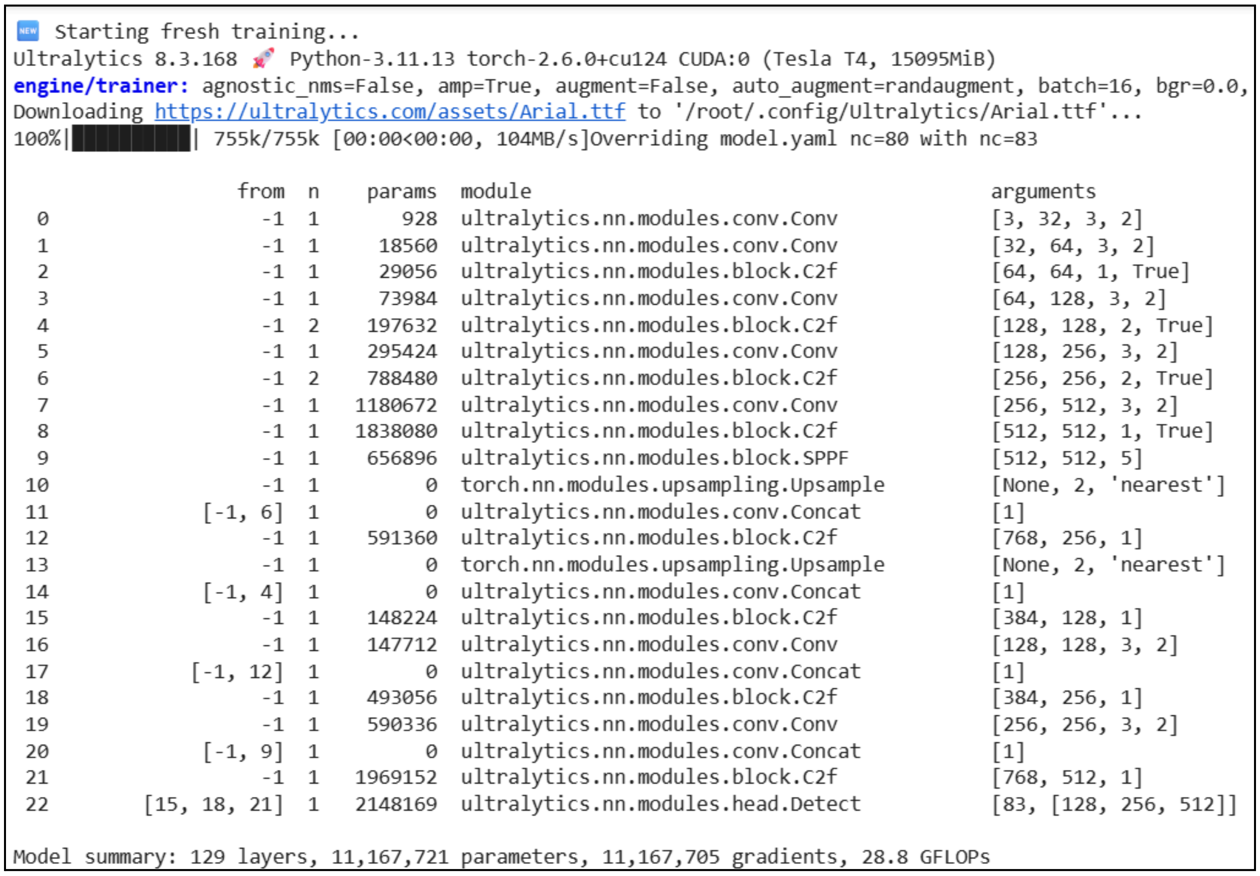
```
# Import packages
from ultralytics import YOLO
import os

# Define path variables
project_dir = "/content/drive/MyDrive/yolo_training"
model_name = "yolov8s.pt"
yaml_path = "/content/dataset.yaml"

# Download the yolov8s.pt model
model = YOLO('yolov8s.pt')
resume_path = f"{project_dir}/train/weights/last.pt"

# Start model fine-tuning
if os.path.exists(resume_path):
    print("✅ Resuming from last checkpoint...")
    model = YOLO(resume_path)
    model.train(resume=True)
else:
    print("🆕 Starting fresh training...")
    model.train(data=yaml_path,
                project=project_dir,
                epochs=50,
                imgsz=640,
                batch=16,
                device=0,
                name="train",
                save=True,
                save_period=5)
```

Output:

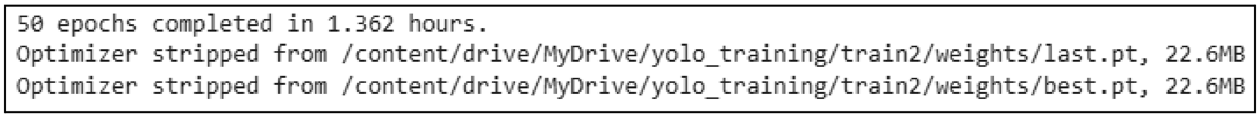


```
🆕 Starting fresh training...
Ultralytics 8.3.168 🚀 Python-3.11.13 torch-2.6.0+cu124 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer: agnostic_nms=False, amp=True, augment=False, auto_augment=randaugment, batch=16, bgr=0.0,
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/Ultralytics/Arial.ttf'...
100%|██████████| 755k/755k [00:00<00:00, 104MB/s]Overriding model.yaml nc=80 with nc=83

                   from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                  -1  1     29056  ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
  3                  -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  4                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  5                  -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  6                  -1  2    788480  ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
  7                  -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  8                  -1  1   1838080  ultralytics.nn.modules.block.C2f             [512, 512, 1, True]
  9                  -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                  -1  1    591360  ultralytics.nn.modules.block.C2f             [768, 256, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                  -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 16                  -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                  -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 19                  -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                  -1  1   1969152  ultralytics.nn.modules.block.C2f             [768, 512, 1]
 22        [15, 18, 21]  1   2148169  ultralytics.nn.modules.head.Detect           [83, [128, 256, 512]]

Model summary: 129 layers, 11,167,721 parameters, 11,167,705 gradients, 28.8 GFLOPs
```

Fig: Architectural summary of YOLOv8 model



```
50 epochs completed in 1.362 hours.
Optimizer stripped from /content/drive/MyDrive/yolo_training/train2/weights/last.pt, 22.6MB
Optimizer stripped from /content/drive/MyDrive/yolo_training/train2/weights/best.pt, 22.6MB
```

Fig: Post training summary of YOLOv8.pt model

2. PaddleOCR

In the implementation of PaddleOCR, I made it as a python script so that I can access the image to text function. PaddleOCR class takes in a **‘lang’** parameter which is a list of languages that the user would like to detect and a **‘use_textline_orientation’** parameter which is a boolean value that enables the algorithm to analyse the complex structures of the document. Multiple other modifications are also available.

```
# Import packages
from paddleocr import PaddleOCR
from PIL import Image

# Image resizing function
def resize_image(image_path, max_size=4000):
    img = Image.open(image_path)
    if max(img.size) > max_size:
        print("Pre-resizing image...")
        img.thumbnail((max_size, max_size))

        idx = image_path.find('.')
        resized_path = image_path[:idx] + "_resized" + image_path[idx:]

        img.save(resized_path)
        return resized_path
    return image_path

# Text from image function using PaddleOCR
def extract_text_from_image(image_path: str) -> str:

    ocr = PaddleOCR(use_textline_orientation=False, lang='en')

    try:
        print("\nGathering results...\n")
        results = ocr.predict(image_path)[0]
        print("\nResults gathered...\n")
        text = ' '.join(results['rec_texts'])
        return text if text.strip() else "[No text detected]"

    except Exception as e:
        return f"[OCR error: {e}]"

if __name__ == '__main__':
    path = input("Enter image path: ").strip().strip('"')
    path = resize_image(path)
    output = extract_text_from_image(path)
    print(f"\nRecognized Text:\n{output}")
```

3. EasyOCR

I present here the implementation of EasyOCR on a tax invoice document. I used **OpenCV** for image handling and **Pyplot** from **Matplotlib** for visualization of the results.

```
# Import packages
import easyocr
import cv2
from matplotlib import pyplot as plt

# Define image path
image_path = "tax_invoice.png"

# Create EasyOCR reader object and read the text
reader = easyocr.Reader(['en'], gpu=False)
result = reader.readtext(image_path)

# Plot the detection
for detection in result:
    top_left = tuple([int(val) for val in detection[0][0]])
    bottom_right = tuple([int(val) for val in detection[0][2]])
    text = detection[1]
    font = cv2.FONT_HERSHEY_SIMPLEX
    img = cv2.rectangle(img, top_left, bottom_right, (255, 0, 0), 2)
    img = cv2.putText(img, text, top_left, font, 2, (0, 0, 0), 2, cv2.LINE_AA)

plt.figure(figsize=(10, 10))
plt.imshow(img)
plt.show()
```

Output:

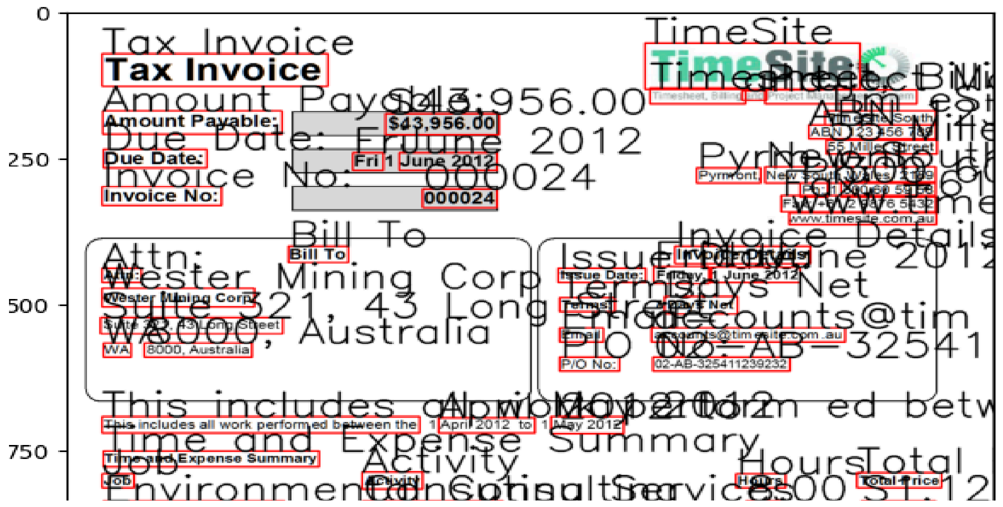


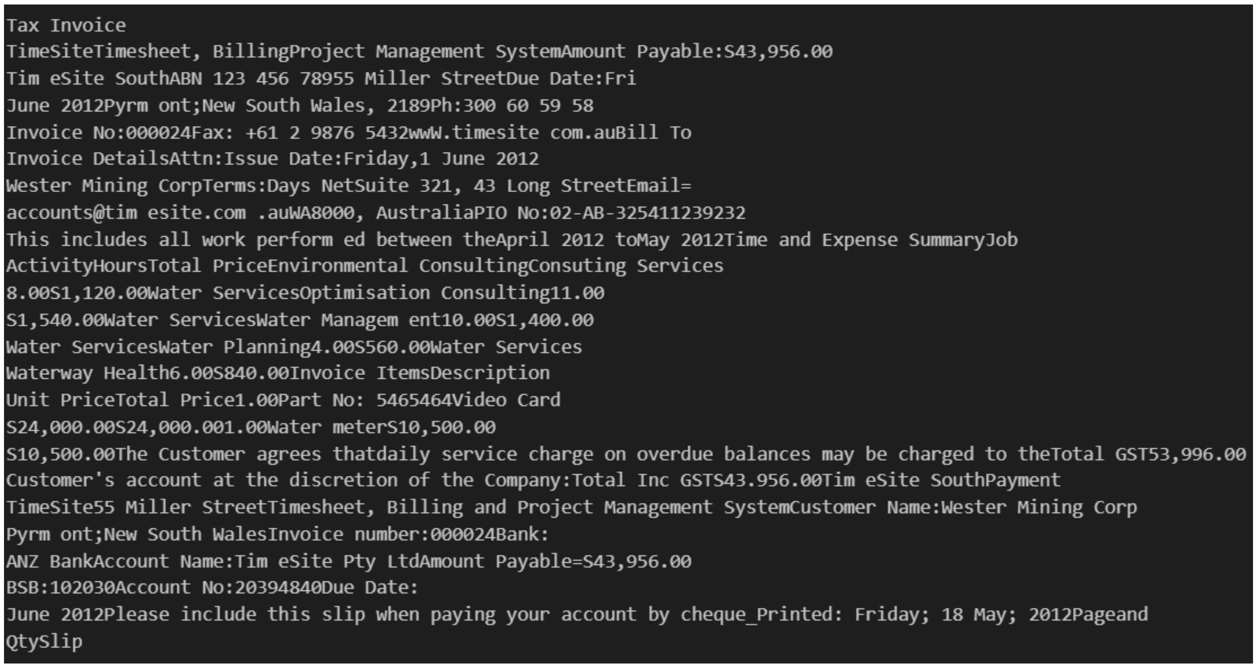
Fig: EasyOCR detection results plotted using Matplotlib.pyplot

Here, I will combine all of the detection results into a single string so that it can be passed to the LLM module for tag generation.

```
# Get all the extracted text
ctx = []
for i in range(len(result)):
    ctx.append(result[i][1])

ctx = ' '.join(ctx)
print(ctx)
```

Output:



```
Tax Invoice
TimeSiteTimesheet, BillingProject Management SystemAmount Payable:S43,956.00
Tim eSite SouthABN 123 456 78955 Miller StreetDue Date:Fri
June 2012Pyrm ont;New South Wales, 2189Ph:300 60 59 58
Invoice No:000024Fax: +61 2 9876 5432wwW.timesite com.auBill To
Invoice DetailsAttn:Issue Date:Friday,1 June 2012
Wester Mining CorpTerms:Days NetSuite 321, 43 Long StreetEmail=
accounts@tim esite.com .auWA8000, AustraliaPIO No:02-AB-325411239232
This includes all work perform ed between theApril 2012 toMay 2012Time and Expense SummaryJob
ActivityHoursTotal PriceEnvironmental ConsultingConsuting Services
8.00S1,120.00Water ServicesOptimisation Consulting11.00
S1,540.00Water ServicesWater Managem ent10.00S1,400.00
Water ServicesWater Planning4.00S560.00Water Services
Waterway Health6.00S840.00Invoice ItemsDescription
Unit PriceTotal Price1.00Part No: 5465464Video Card
S24,000.00S24,000.001.00Water meterS10,500.00
S10,500.00The Customer agrees thatdaily service charge on overdue balances may be charged to theTotal GST53,996.00
Customer's account at the discretion of the Company:Total Inc GSTS43.956.00Tim eSite SouthPayment
TimeSite55 Miller StreetTimesheet, Billing and Project Management SystemCustomer Name:Wester Mining Corp
Pyrm ont;New South WalesInvoice number:000024Bank:
ANZ BankAccount Name:Tim eSite Pty LtdAmount Payable=S43,956.00
BSB:102030Account No:20394840Due Date:
June 2012Please include this slip when paying your account by cheque_Printed: Friday; 18 May; 2012Pageand
QtySlip
```

Fig: EasyOCR detection results as a single string

4. Final Integration

Finally, I am presenting the overall pipeline that I designed by combining YOLOv8 for object detection and EasyOCR for text extraction. I have imported the **“text_ocr.py”** file which I created while implementing EasyOCR and I also used the fine-tuned yolov8-small variant. This piece of code is also written like a python library so that the **“get_text()”** function can be called by another program.

```
# Import packages
from ultralytics import YOLO
import text_ocr

# Define YOLO model
model_name = 'yolo_fine_tuned.pt'

# Define the pipeline
def get_text(image_path: str) -> str:
    yolo = YOLO(model_name)
    results = yolo.predict(image_path, conf=0.25)

    object_names = []
    if results and results[0].boxes is not None:
        for box in results[0].boxes.data.tolist():
            cls_id = int(box[5])
            object_names.append(results[0].names[cls_id])

    object_names = set(object_names)
    if 'text' in object_names:
        text = text_ocr.extract_text_from_image(image_path)
        object_names.discard('text')

    return {‘objects’: ' '.join(object_names), ‘extracted_text’: text}

if __name__ == "__main__":
    path = input("Enter the image path: ")
    text = get_text(image_path=path)
    print(text)
```

CONCLUSION

This project successfully demonstrated the integration of object detection and optical character recognition (OCR) techniques to optimize search by adding specific tags to the metadata of the documents. By employing **YOLOv8-small** for fast and accurate object detection and **EasyOCR** for text extraction, the system achieved a balanced trade-off between speed, accuracy, and computational efficiency.

Among the OCR methods evaluated — **PaddleOCR**, **EasyOCR**, and **TesseractOCR** — EasyOCR was found to be the most suitable choice for the project's requirements, with the ease of integration into a web-based pipeline. PaddleOCR proved to be a powerful alternative for enterprise-grade deployment with advanced layout parsing, while Tesseract remained optimal for clean, scanned documents in CPU-only environments.

The final integrated pipeline efficiently detects objects, extracts text when present, and generates relevant tags to be appended to the document's metadata. This significantly reduces search times and improves memory utilization on the **GovDrive** platform.

Looking ahead, the rapid advancement of multimodal large language models such as **LLaVA** presents an opportunity to unify the entire workflow — object detection, text extraction, and tag generation — within a single smart system. While the proposed pipeline (YOLO + OCR) still offers superior accuracy for dense document text, future deployments could leverage models like LLaVA for more flexible, context-aware, and conversational document understanding, thereby reducing integration complexity and streamlining the processing pipeline.