Automated Timetable Digitization using OCR and Image Processing

Avani Bhuva
Assistant Professor, Dept. of Computer Engineering
Mukesh Patel School of Technology Management and Engineering
Mumbai, India
avani.bhuva@nmims.edu

Anushka Batte
Dept. of Computer Engineering
Mukesh Patel School of Technology Management and Engineering
Mumbai, India
anushka.batte56@nmims.in

Bhushan Chavan
Dept. of Computer Engineering
Mukesh Patel School of Technology Management and Engineering
Mumbai, India
bhushan.chavan77@nmims.in

***Abstract*—Our college provides us with timetables that are scanned copies in a PDF format. We aim to build an app which serves as a student hub having multiple functionalities to aid the students at our college. One of its functionalities is timetable management where the user can upload a copy of their timetable, and it will be converted into a digital format automatically by the system without requiring any manual efforts from their end. For this purpose, we have used image inputs and applied preprocessing methodologies to them such as morphological closing, grayscale conversion, noise removal, and gaussian filtering, for image enhancement. After detecting individual cells of the table, the use of Mistral AI has been made for its basic OCR capabilities to retrieve text, followed by a conversion to a JSON format which can be stored in the database. This data can then be utilized by the app to provide timely notifications to the students with their classroom details. The results have been excellent with the OCR model, especially due to the preprocessing applied to the image. This is a unique approach since we have combined two fields to make the lives of our fellow students easier.**

Keywords—timetable extraction, image processing, text detection, OCR, database.

I. Introduction

In modern educational institutions, students and faculty often deal with complex timetables having scattered details such as timings, days, batches, classrooms, and subjects. Managing these timetables can be time-consuming and tedious, especially in the fast-paced education system that exists today. Timetable formats tend to vary across not only colleges, but even courses within the same university. The presence of merged cells, inconsistent layouts, and varied fonts makes the data harder to interpret.

Automating the extraction of key details from these timetables and storing data for easy access reduces these issues substantially. However, traditional optical character recognition (OCR) methods struggle with this type of data, giving low accuracy and meaningless text outputs.

Our goal in the future is to build a mobile application, which serves as a student hub for our college. One of its main features is having timetable management simplified where students can just upload a copy, and have that be the only work required from their end. Automated timetable digitization allows students to receive relevant schedule notifications directly on their devices, significantly reducing the time spent searching for and interpreting timetable information.

The motivation behind this work is to develop a system that can automatically extract structured timetable data from images, saving students and administrative staff time and effort. This work presents a pipeline that combines image preprocessing, table detection, and OCR to reliably extract structured timetable data, which can then be used in mobile applications for student notification and academic planning.

II. Literature Review

OCR has made a lot of progress in text detection, to the extent of being used for making a mixture of Japanese and mathematical formulas. To improve accuracy, a combination of two models – Tesseract and Mathpix were used in the study, to classify text into one of three categories: word, sentence, and formula. The model achieved varying accuracies – high for Mathpix sentences and formulas, standard Tesseract words and formulas, and Tesseract after learning words. [1]

In another study, documents were processed with two OCR models and then the text was classified using the Naive Bayes algorithm. The models work by creating small blobs for each character by using a grayscale filter, binarizing the image, and then inverting it, using AForge framework. The small blobs are also resized, and their average pixel value is calculated. Then, the Naive Bayes algorithm is used to classify text. Handwriting documents had a low accuracy of between 45% to 60%. In terms of Naive Bayes, the approximate accuracy was 53%. [2]

Another method involves combining OCR with large-language models (LLMs) such as GPT-4, Claude, and Mistral. Here, the document is first prepared by binarization and skew correction for improved accuracy. Then, for retrieving unprocessed textual information, OCR engines like Tesseract, EasyOCR, and Google Vision API are used. On the other hand, if it is a digital file, then layout-aware libraries or tools are used to interpret it, such as PyMudPDF, pdfplumber, and python-docx. These tools can retrieve tabular data, metadata, etc. Advanced LLMs are applied on the raw text to carry out focused information extraction based on input prompt. Output received in JSON format is stored as machine-accessible, structured data. The dataset used 3 sets of scanned documents and 2 sets of digital documents. In OCR, the highest accuracy was achieved by Google Vision API of ~94%, followed by DocTR with ~91% and lastly, Tesseract (v4) ~85%. For digital document parsing, a near-perfect accuracy of ~100% was received. However, authors of this paper suggest that using another deep learning driven handwriting dataset, improving assistance in case of multilingual assistance, and incorporating real time feedback can improve it more. [3]

Combining conventional OCR systems like Tesseract, EasyOCR, TrOCR, and PaddleOCR, with vision-language models (VLMs_) like Mistral, GotOCR, Pali Gemma, and LLaMa 3.2 Vision, has been evaluated for their respective performance. Google Paligemma-3B came out with the best accuracy of 91.22%. The lowest was that of Adept Fuyu8B at 9.12$. This shows how performance can vary across different architectures. The process essentially begins by having a book's image data as the input followed by model loading and image processing. OCR will load it and then validate it for quantity and format. It will extract the text and refine it through VLM. Future scope includes improving the accuracy of poorly performing models, deriving insights from high-performing models, and lastly using a standardized evaluation protocol for soft match tasks. [4]

Real-Time Table Structure Recognition (RTSR) is a lightweight, deep learning model which will extract the logical structure of tables in real time. The model employs a ResNet-18 with a Feature Pyramid Network (FPN) backbone and introduces a feature aggregation module (RESA) to enhance contextual understanding between rows and columns while maintaining high speed. Soft label gaussian heat maps are used for accurate separator alignment. RTSR achieved real-time performance (38.1 FPS average) and competitive accuracy compared to state-of-the-art methods across multiple benchmark datasets. [5]

This last study that we reviewed is highly relevant to our paper. 'Time-Table to Mobile Reminders' is an app designed to automate reminders of lectures without any hassle, using an input of the timetable as an image. The process includes using OCR with the help of OpenCV and Tesseract, identifying the text stored in an excel file, and classifying it into date, time, and lecture. Finally, as per this data, a reminder is set in the app. In this way, a notification is received on the student or faculty device a few minutes before the lecture is about to commence, saving them a lot of time. The method followed is simple – after receiving an input image, processing is applied to it. Then, using OCR, data is extracted and stored in the excel file, which is passed to the app for setting reminders accordingly. This is their proposed method, it has not actually been implemented, and that is what we wish to do in this study. [6]

III. Methodology

The pipeline we followed for our own studyconsists of 6 distinct stages which are discussed in detail below.

A. *Image Input and Preprocessing*

First, we take an input of the timetable in PDF or image format.

```
#Import libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt

#Image upload and PDF conversion (if required)
!pip install pdf2image
!apt-get install poppler-utils -y
from pdf2image import convert_from_path
from google.colab import files
import os

#Upload file (PDF or image)
uploaded = files.upload()
file_path = list(uploaded.keys())[0]

#Check if it's a PDF
if file_path.lower().endswith(".pdf"):
#Convert PDF pages to images (default 300 DPI)
pages = convert_from_path(file_path, dpi=300)
#Take first page (or loop for all pages)
img = np.array(pages[0])
#Convert RGB to BGR for OpenCV
img = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
else:
#Load image directly
img = cv2.imread(file_path)

#Display the image
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.figure(figsize=(10,8))
plt.imshow(img_rgb)
plt.title("Uploaded Timetable")
plt.axis('off')
plt.show()
```

PDFs are converted to jpg format. Then, the image is pre-processed using the following techniques in order to refine it and make text detection more accurate:

- Grayscale conversion
- Noise reduction
- Adaptive thresholding
- Morphological operations

```
#Preprocessing the image
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
#Reduce noise while preserving edges
blur = cv2.bilateralFilter(gray, d=9, sigmaColor=75, sigmaSpace=75)
#Adaptive thresholding
thresh = cv2.adaptiveThreshold(blur, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 15, 10)
#Kernel for morphology
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (2,2))
#Close small gaps in letters
morph = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel,
iterations=2)
#Closing to connect broken text regions
morph = cv2.dilate(morph, kernel, iterations=1)
morph = cv2.erode(morph, kernel, iterations=1)
#Display image
plt.figure(figsize=(10,8))
plt.imshow(morph, cmap='gray')
plt.title("Preprocessed Image")
plt.axis('off')
plt.show()
```

B. *Cell Detection*

This step involves locating the boundaries of the timetable, to classify the lectures as per their timings. This is an important step as it serves as a precursor to classifying the lectures into their various attributes for storing in the JSON format.

- Binarization of image
- Detection of horizontal and vertical lines
- Finding contours of each cell
- Drawing contours and storing cell coordinates

```
#Detecting table cells
#Binarize the image (inverse)
_, thresh_bin = cv2.threshold(gray, 180, 255,
cv2.THRESH_BINARY_INV)
#Step 1: Detect horizontal and vertical lines
#Horizontal lines (adjust width depending on table size)
horizontal_kernel = cv2.getStructuringElement(cv2.MORPH_RECT,
(40,1))
horizontal_lines = cv2.morphologyEx(thresh_bin, cv2.MORPH_OPEN,
horizontal_kernel, iterations=1)
#Vertical lines (adjust height depending on table size)
vertical_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1,40))
vertical_lines = cv2.morphologyEx(thresh_bin, cv2.MORPH_OPEN,
vertical_kernel, iterations=1)
#Combine horizontal and vertical lines to get grid
table_mask = cv2.add(horizontal_lines, vertical_lines)
#Step 2: Find contours of each cell
#Invert mask for contour detection
table_mask_inv = 255 - table_mask
contours, hierarchy = cv2.findContours(table_mask_inv,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
#Step 3: Draw contours and store cell coordinates
rgb = cv2.cvtColor(morph, cv2.COLOR_BGR2RGB)
img_cells = rgb.copy()
cell_boxes = [] #store each cell's bounding box (x,y,w,h)
for cnt in contours:
x, y, w, h = cv2.boundingRect(cnt)
if w > 30 and h > 20: #filter tiny boxes
cv2.rectangle(img_cells, (x,y), (x+w, y+h), (0,255,0), 2)
cell_boxes.append((x,y,w,h))

#Display detected cells
plt.figure(figsize=(10,8))
plt.imshow(cv2.cvtColor(img_cells, cv2.COLOR_BGR2RGB))
plt.title("Detected Individual Cells")
plt.axis('off')
plt.show()
```

C. *Cropping the image*

This is specifically done in our case since our college's timetables have a secondary table in the bottom half having details such as faculty names, full forms of labs and subjects, and certain locations where floor is not mentioned. However, when it comes to storing in the final database, this data is not as important as the main timetable content, and hence, is cropped out in order to avoid confusing the OCR model regarding what needs to be focused on.

```
#Cropping timetable to only keep the relevant information
cropped = img_cells[250:1650, 0:img.shape[1]] #after adjusting height
and width
saved_image = cv2.imwrite('final_image.jpg',cropped)

plt.figure(figsize=(10,8))
plt.imshow(cv2.cvtColor(cropped, cv2.COLOR_BGR2RGB))
plt.title("Cropped Timetable")
plt.axis("off")
plt.show()
```

D. *OCR Text Detection using Mistral AI*

Mistral AI is a French artificial intelligence startup which offers both open-source and proprietary LLMs. We use an API key to access their basic OCR text detection functionality in our work. The image is first encoded into a base64 string format, and then the model works on it.

```
#OCR Detection using Mistral AI
!pip install mistralai
import base64
import os
from mistralai import Mistral

def encode_image(image_path):
"""Encode the image to base64."""
try:
with open(image_path, "rb") as image_file:
return base64.b64encode(image_file.read()).decode('utf-8')
except FileNotFoundError:
print(f"Error: The file {image_path} was not found.")
return None
except Exception as e: #Added general exception handling
print(f"Error: {e}")
return None

#Path to your image
image_path = "/content/final_image.jpg"

#Getting the base64 string
base64_image = encode_image(image_path)

client = Mistral(api_key="INSERT_API_KEY_HERE")
ocr_response = client.ocr.process(
model="mistral-ocr-latest",
document={
"type": "image_url",
"image_url": f"data:image/jpeg;base64,{base64_image}"
},
include_image_base64=True
)

print("\n\n".join([f"Page {i+1}\n{ocr_response.pages[i].markdown}"
for i in range(len(ocr_response.pages))]))
```

E. *Storing into JSON format*

Lastly, the OCR response generated in the previous step is converted into JSON format and stored using entries for "Program", "Effective_From", and "Timetable" which is further segregated into timings and days.

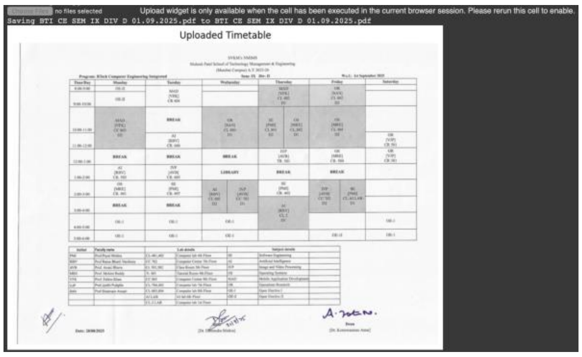
```
#Storing output to JSON file
import json
import re

markdown_output = ocr_response.pages[0].markdown

# Extract program and metadata
program_match = re.search(r'Program:(.*)', markdown_output)
wef_match = re.search(r'W\.e\.f\.: (.*)', markdown_output)
program_info = program_match.group(1).strip() if program_match else
""
wef_date = wef_match.group(1).strip() if wef_match else ""

#Extract table lines (lines that contain '|')
table_lines = [line.strip() for line in markdown_output.split("\n") if "|"
in line]
```

```
#Get headers (remove first and last '|' and strip)
headers = [h.strip() for h in table_lines[0].split("|")[1:-1]]

#Process table rows (skip header + divider)
data_rows = []
for line in table_lines[2:]:
cells = [c.strip() for c in line.split("|")[1:-1]]
if len(cells) == len(headers):
row = dict(zip(headers, cells))
data_rows.append(row)

#Final JSON structure
timetable_json = {
"Program": program_info,
"Effective_From": wef_date,
"Timetable": data_rows
}

#Save as JSON file
with open("/content/timetable.json", "w") as f:
json.dump(timetable_json, f, indent=4)

# Display structured JSON
print("\n=== Structured JSON Output ===")
print(json.dumps(timetable_json, indent=4))
```

IV. Results

This section shows the results and output of the implementation of the code written in the methodology section. Authors took an input image of a timetable to check if the model is functional and generating the correct output.

A. Image Input and Preprocessing



```
Choose Files no files selected    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving BTI CE SEM IX DIV D 01.09.2025.pdf to BTI CE SEM IX DIV D 01.09.2025.pdf
```

Fig. 1. Figure shows output of code snippet for taking image input and displaying it.

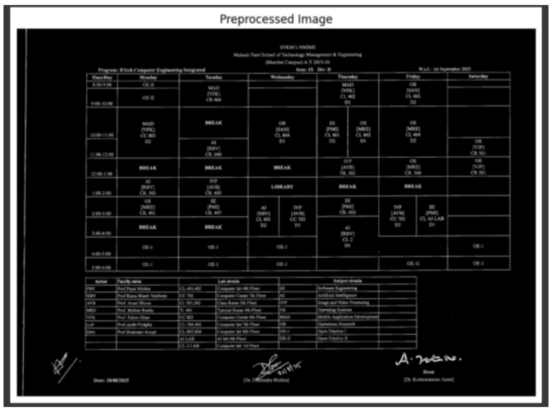


Fig. 2. Figure shows output of code snippet for applying image preprocessing techniques to the input image.

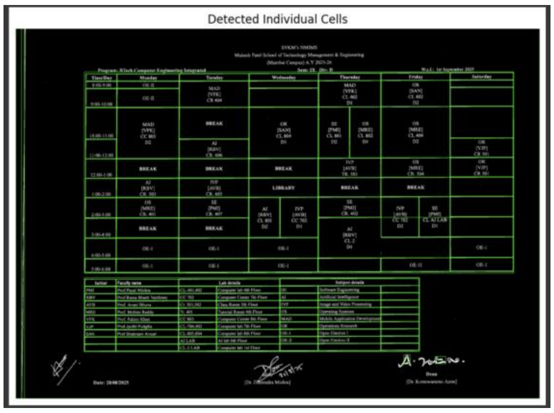


Fig. 3. Figure shows output of code snippet used for cell detection in the image.

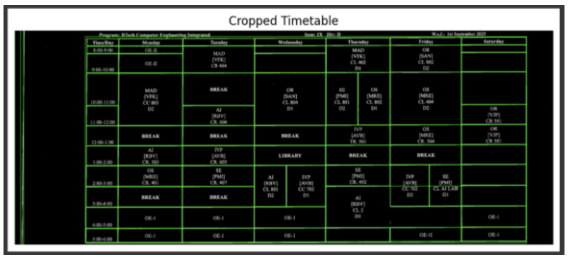
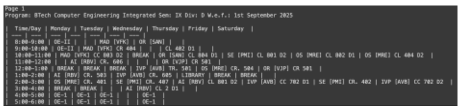
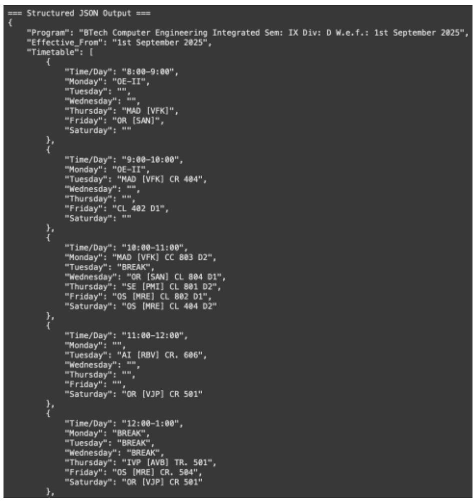


Fig. 4. Figure shows output of code snippet used for cropping the image to keep only the relevant data, so as to improve accuracy of text detection.



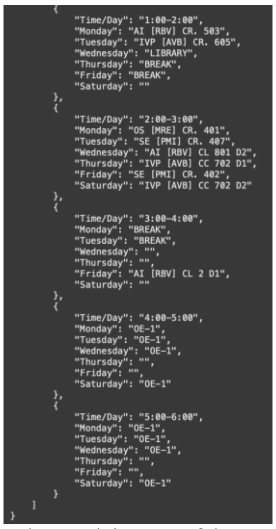
```
Page 1
Program: BTech Computer Engineering Integrated Sem: IX Div: D W.e.f.: 1st September 2025

| Time/Day | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday |
| --- | --- | --- | --- | --- | --- | --- |
| 8:00-9:00 | OE-II | | | MAD [VFK] | OR [SAN] | |
| 9:00-10:00 | OE-II | MAD [VFK] CR 404 | | | CL 402 D1 | |
| 10:00-11:00 | MAD [VFK] CC 803 D2 | BREAK | OR [SAN] CL 804 D1 | SE [PMI] CL 801 D2 | OS [MRE] CL 802 D1 | OS [MRE] CL 404 D2 |
| 11:00-12:00 | | AI [RBV] CR. 606 | | | | OR [VJP] CR 501 |
| 12:00-1:00 | BREAK | BREAK | BREAK | IVP [AVB] TR. 501 | OS [MRE] CR. 504 | OR [VJP] CR 501 |
| 1:00-2:00 | AI [RBV] CR. 503 | IVP [AVB] CR. 605 | LIBRARY | BREAK | BREAK | |
| 2:00-3:00 | OS [MRE] CR. 401 | SE [PMI] CR. 407 | AI [RBV] CL 801 D2 | IVP [AVB] CC 702 D1 | SE [PMI] CR. 402 | IVP [AVB] CC 702 D2 |
| 3:00-4:00 | BREAK | BREAK | | | AI [RBV] CL 2 D1 | |
| 4:00-5:00 | OE-1 | OE-1 | OE-1 | | | OE-1 |
| 5:00-6:00 | OE-1 | OE-1 | OE-1 | | | OE-1 |
```

Fig. 5. Figure shows output of code snippet where Mistral AI's OCR functionality is used to detect the text and print it out in a table format.



```
=== Structured JSON Output ===
{
    "Program": "BTech Computer Engineering Integrated Sem: IX Div: D W.e.f.: 1st September 2025",
    "Effective_From": "1st September 2025",
    "Timetable": [
        {
            "Time/Day": "8:00-9:00",
            "Monday": "OE-II",
            "Tuesday": "",
            "Wednesday": "",
            "Thursday": "MAD [VFK]",
            "Friday": "OR [SAN]",
            "Saturday": ""
        },
        {
            "Time/Day": "9:00-10:00",
            "Monday": "OE-II",
            "Tuesday": "MAD [VFK] CR 404",
            "Wednesday": "",
            "Thursday": "",
            "Friday": "CL 402 D1",
            "Saturday": ""
        },
        {
            "Time/Day": "10:00-11:00",
            "Monday": "MAD [VFK] CC 803 D2",
            "Tuesday": "BREAK",
            "Wednesday": "OR [SAN] CL 804 D1",
            "Thursday": "SE [PMI] CL 801 D2",
            "Friday": "OS [MRE] CL 802 D1",
            "Saturday": "OS [MRE] CL 404 D2"
        },
        {
            "Time/Day": "11:00-12:00",
            "Monday": "",
            "Tuesday": "AI [RBV] CR. 606",
            "Wednesday": "",
            "Thursday": "",
            "Friday": "",
            "Saturday": "OR [VJP] CR 501"
        },
        {
            "Time/Day": "12:00-1:00",
            "Monday": "BREAK",
            "Tuesday": "BREAK",
            "Wednesday": "BREAK",
            "Thursday": "IVP [AVB] TR. 501",
            "Friday": "OS [MRE] CR. 504",
            "Saturday": "OR [VJP] CR 501"
        },
```

Fig. 6.1. Figure shows part of the output generated to store the text in JSON format.



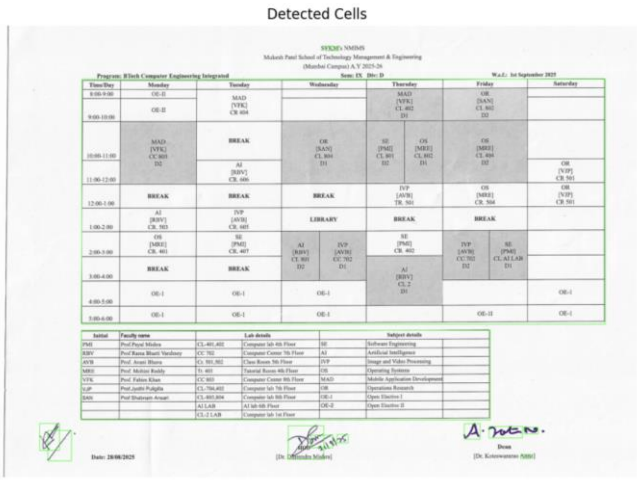
```
        {
            "Time/Day": "1:00-2:00",
            "Monday": "AI [RBV] CR. 503",
            "Tuesday": "IVP [AVB] CR. 605",
            "Wednesday": "LIBRARY",
            "Thursday": "BREAK",
            "Friday": "BREAK",
            "Saturday": ""
        },
        {
            "Time/Day": "2:00-3:00",
            "Monday": "OS [MRE] CR. 401",
            "Tuesday": "SE [PMI] CR. 407",
            "Wednesday": "AI [RBV] CL 801 D2",
            "Thursday": "IVP [AVB] CC 702 D1",
            "Friday": "SE [PMI] CR. 402",
            "Saturday": "IVP [AVB] CC 702 D2"
        },
        {
            "Time/Day": "3:00-4:00",
            "Monday": "BREAK",
            "Tuesday": "BREAK",
            "Wednesday": "",
            "Thursday": "",
            "Friday": "AI [RBV] CL 2 D1",
            "Saturday": ""
        },
        {
            "Time/Day": "4:00-5:00",
            "Monday": "OE-1",
            "Tuesday": "OE-1",
            "Wednesday": "OE-1",
            "Thursday": "",
            "Friday": "",
            "Saturday": "OE-1"
        },
        {
            "Time/Day": "5:00-6:00",
            "Monday": "OE-1",
            "Tuesday": "OE-1",
            "Wednesday": "OE-1",
            "Thursday": "",
            "Friday": "",
            "Saturday": "OE-1"
        }
    ]
}
```

Fig. 6.2. Figure shows the remaining part of the output generated to store the text in JSON format.

V. Discussion

This study was performed after trying several possible methods, given the review that the authors conducted. Various image processing techniques were tried in order to enhance the quality of the image and improve the accuracy of the OCR being used. The image showed the best results when it was grey scaled, and morphological operations were applied to it.

When it came to cell detection, contours were used and small noise was filtered out. However, this code could only detect the external boundary of the table, not the individual cells (as seen in fig. 7).


Detected Cells

SVKM's NMIMS
Mukesh Patel School of Technology Management & Engineering
(Mumbai Campus) A.Y 2025-26

Program: BTech Computer Engineering Integrated   Sem: IX Div: D   W.e.f.: 1st September 2025

Time/Day	Monday	Tuesday	Wednesday	Thursday		Friday		Saturday
8:00-9:00	OE-II	MAD [VFK] CR 404		MAD [VFK] CL 402 D1		OR [SAN] CL 802 D2		
9:00-10:00	OE-II							
10:00-11:00	MAD [VFK] CC 803 D2	BREAK	OR [SAN] CL 804 D1	SE [PMI] CL 801 D2	OS [MRE] CL 802 D1	OS [MRE] CL 404 D2		
11:00-12:00		AI [RBV] CR. 606						OR [VJP] CR 501
12:00-1:00	BREAK	BREAK	BREAK	IVP [AVB] TR. 501		OS [MRE] CR. 504		OR [VJP] CR 501
1:00-2:00	AI [RBV] CR. 503	IVP [AVB] CR. 605	LIBRARY	BREAK		BREAK		
2:00-3:00	OS [MRE] CR. 401	SE [PMI] CR. 407	AI [RBV] CL 801 D2 / IVP [AVB] CC 702 D1	SE [PMI] CR. 402		IVP [AVB] CC 702 D2	SE [PMI] CL AI LAB D1	
3:00-4:00	BREAK	BREAK		AI [RBV] CL 2 D1				
4:00-5:00	OE-I	OE-I	OE-I					OE-I
5:00-6:00	OE-I	OE-I	OE-I			OE-II		OE-I

Initial	Faculty name	Lab details		Subject details		
PMI	Prof.Payal Mishra	CL-401,402	Computer lab 4th Floor	SE	Software Engineering	
RBV	Prof Rama Bhatti Vardney	CC 702	Computer Center 7th Floor	AI	Artificial Intelligence	
AVB	Prof. Avani Bhave	Cr. 501,502	Class Room 5th Floor	IVP	Image and Video Processing	
MRE	Prof. Mohini Reddy	Tr. 401	Tutorial Room 4th Floor	OS	Operating Systems	
VFK	Prof. Fahim Khan	CC 803	Computer Center 8th Floor	MAD	Mobile Application Development	
VJP	Prof.Jyothi Pulgilla	CL-704,402	Computer lab 7th Floor	OR	Operations Research	
SAN	Prof Shahram Ansari	CL-803,804	Computer lab 8th Floor	OE-1	Open Elective I	
		AI LAB	AI lab 6th Floor	OE-2	Open Elective II	
		CL-2 LAB	Computer lab 1st Floor			

Date: 28/08/2025   [Dr. Dhirendra Mishra]   Dean [Dr. Koteswararao [illegible]]

Fig. 7.Figure shows the inaccuracy of initial cell detection where it could only find the external boundary of the table.

To ensure accurate detection of cells, the image was inverted, and then horizontal and vertical lines were detected before going for contours. Once the contours were drawn; cell coordinates were stored successfully.

Initially, we tried using the following OCR systems for text detection:

1. EasyOCR – It could not detect the text in the table at all; it was only able to detect the heading at the very top of the image.
2. PyTesseract – It was able to detect the table; however, it could not identify the text accurately at all.
3. TrOCR – It gave almost the same result as PyTesseract.

VI. Conclusion

In this study, the authors have successfully implemented a pipeline for digitizing scanned images of Timetables and applying various processing techniques to them, in order to extract text and store it in JSON format. The process starts off with uploading the scanned PDF and converting it to image format. Then, image processing techniques like grayscale conversion, noise reduction, adaptive thresholding, and various morphological operations are applied to the image. This enhances its quality for further text detection. It also helps in detecting cells using contours. The image is cropped to focus only on the relevant cells. After this, the Mistral AI's OCR functionality is used to detect the individual contents of the cell, and then they are stored in a structure JSON format.

VII. Future Work

Further, the digitized data can be used to send reminder notifications to students and faculty, through a university app, regarding lecture details such as subject name, timing, location, and more. This has to be done by plugging in this pipeline into an Android mobile app.

References

[1] Yasuhiro Kobayashi, Shunya Mimuro, Shin-nosuke Suzuki, Yousuke Iijima, Akira Okada, Basic research on a handwritten note image recognition system that combines two OCRs, Procedia Computer Science, Volume 192, 2021, Pages 2596-2605, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2021.09.029.

[2] Aydın, Ömer. (2021). Classification of Documents Extracted from Images with Optical Character Recognition Methods. Computer Science. 6. 48-55.

[3] Sinha, Rakhi, & B S., Rekha (2025). Digitization of document and information extraction using OCR (arXiv:2506.11156). arXiv. https://doi.org/10.48550/arXiv.2506.11156

[4] Gurung, Aakash. (2025). Advanced OCR Post-Processing: Integrating Vision-Language Models for Enhanced Performance.

[5] Nam Quan Nguyen, Xuan Phong Pham, & Tuan-Anh Tran. (2024). RTSR: A real-time table structure recognition approach. In Ulle Endriss et al. (Eds.), Frontiers in Artificial Intelligence and Applications: Vol. 392. ECAI 2024 (pp. 681–687). IOS Press. https://doi.org/10.3233/FAIA240549

[6] Surti, Rishabh & Chavan, Divyesh & Saturwar, Dr. (2020). Time-Table to Mobile Reminders. International Journal of Recent Technology and Engineering (IJRTE). 9. 960-962. 10.35940/ijrte.A1535.059120.