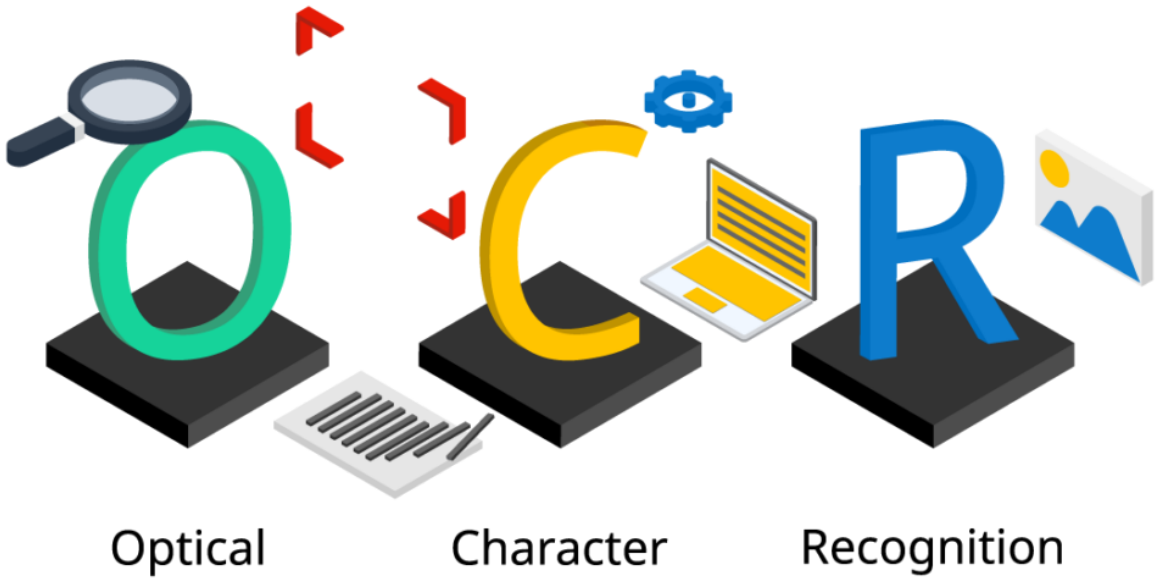
Digitalizing Handwritten text

This project implants the OCR model using python and TrOCR model with the help of Transformer and Tokenizers. This model takes handwritten text and converts it to digital text. In the project we take an image containing hand written text as input and give final output as PDF having digital text.



Initially we tried different models to compare and select which model to choose.

1) easyOCR
2) pytesseract OCR
3) TrOCR

These are the models which we tested and selected TrOCR as final model.

Why TrOCR over other models?

Tesseract is performing well for high-resolution images. Certain morphological operations such as dilation, erosion, OTSU binarization can

help increase pytesseract performance. It can not give accuracy for images having low resolution.

EasyOCR is a lightweight model which is giving a good performance for receipt or PDF conversion. It is giving more accurate results with organized texts like pdf files, receipts, bills. For unorganized text it dont give that good results.

As TrOCR is good for images with low resolution as well fr unorganized text TrOCR is better model than other tried models so we finalized TrOCR as model for project.

Why TrOCR is better than other CNN based models?

Attention mechanism: The transformer model introduced the attention mechanism, which allows it to focus on relevant parts of the input during processing. This mechanism enables the model to pay attention to specific characters or regions within an image, making it more robust to variations

in text size, font, and orientation. This can be particularly useful in handling challenging OCR scenarios.

End-to-end approach: Transformers can be trained in an end-to-end manner, directly taking the raw image as input and producing the recognized text as output. This eliminates the need for explicit feature extraction steps typically performed in CNN-based models. The end-to-end approach simplifies the OCR pipeline and potentially improves overall performance by jointly optimizing the feature extraction and recognition steps.

TrOCR Model Architecture

TrOCR is built up with the Transformer architecture, including an image Transformer for extracting the visual features

and a text Transformer for language modeling. We adopt the

vanilla Transformer encoder-decoder structure in TrOCR.

The encoder is designed to obtain the representation of the

image patches and the decoder is to generate the wordpiece

sequence with the guidance of the visual features and previous

A-star Algorithm for separating the lines

A* is a popular algorithm used for finding the shortest path between two points in a graph or grid.The astar function takes an array representing the grid, a start point, and a goal point as input. Neighbors list shows possible direction of movement(up,right,left,down).

The point with the lowest f-score is extracted from the open heap.

If the current point is the goal point, the function reconstructs and returns the path by following the came_from dictionary.

Otherwise, the current point is added to the close_set.

For each neighbor of the current point, the code checks if it's a valid point within the grid and not blocked (denoted by a value of 1 in the array).

If the neighbor is in the close_set and the tentative g-score (cost from start) is greater than or equal to the existing g-score for that neighbor, the loop continues to the next neighbor.

If the tentative g-score is lower than the existing g-score for the neighbor or the neighbor is not in the open heap, the came_from, gscore, and fscore dictionaries are updated with the new values, and the neighbor is added to the open heap.

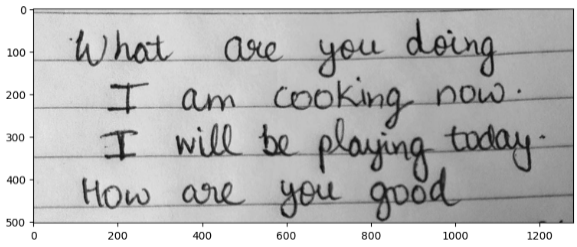
If the loop completes without finding the goal point, an empty list is returned.

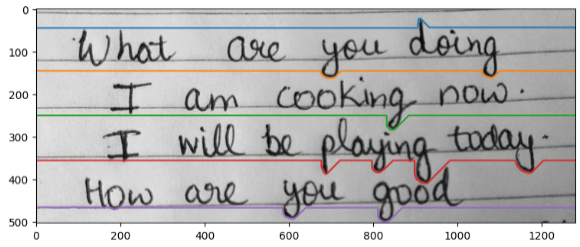
First image is converted to binary image using get_binary().

The code iterates over peak_groups, which seems to be a list of indices representing groups of peaks or regions in the image. For each group, it performs line segmentation.

It extracts the corresponding region from the binary_image using the indices from sub_image_index.

Now A* is applied to these regions and we get segmented lines in the image.

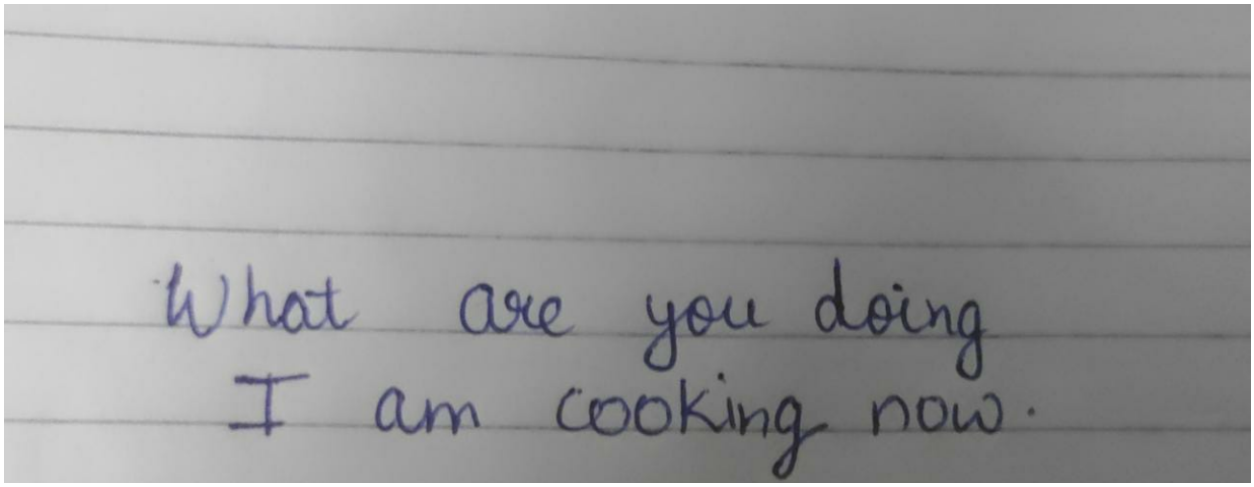




USING PRE-TRAINED MODEL

1) Input- Here we take images as input. Show_image function is used to output the which we are using as input.

```
def show_image(pathStr):
    img = Image.open(pathStr).convert("RGB")
    display(img)
    return img
```



2) By finding peak regions based on a threshold, the algorithm segments an image. The threshold is calculated as the average value of the input signal's or image's maximum and minimum values. The segmented image is then cleaned up by removing the indicated peak regions while leaving the remainder of the image in tact. For path planning, the code additionally clusters the recognised peak regions together. It excludes small regions with a length of under 10 and divides the peak indices into groups based on successive differences. The outcome is printed along with the number of peak groups discovered.
3) Now using these peak group we apply A* algorithm to particular region and find segmented lines in segment_seperarting_lines.
4) Use sepereted_images[] to get all the separated images.

5) Now pass these images to our transformer model and get digitized text for all images.
6) Output- We finally output the pdf containing these text in proper order.