# What is OCR

Optical Character Recognition (OCR) is a technology used to convert different types of documents, such as scanned paper documents, PDF files, or images captured by a digital camera, into editable and searchable data. The primary function of OCR is to recognize text within these documents and convert it into a machine-readable form.

**`Uses of OCR:`**

`Document Digitization:` OCR is widely used to digitize printed documents, converting them into editable formats. This is particularly useful for archiving and retrieving information without manually inputting data.

`Business Process Automation:` In business environments, OCR streamlines processes by automating the data extraction from physical documents such as invoices, receipts, and forms, thus reducing the need for manual entry and improving accuracy.

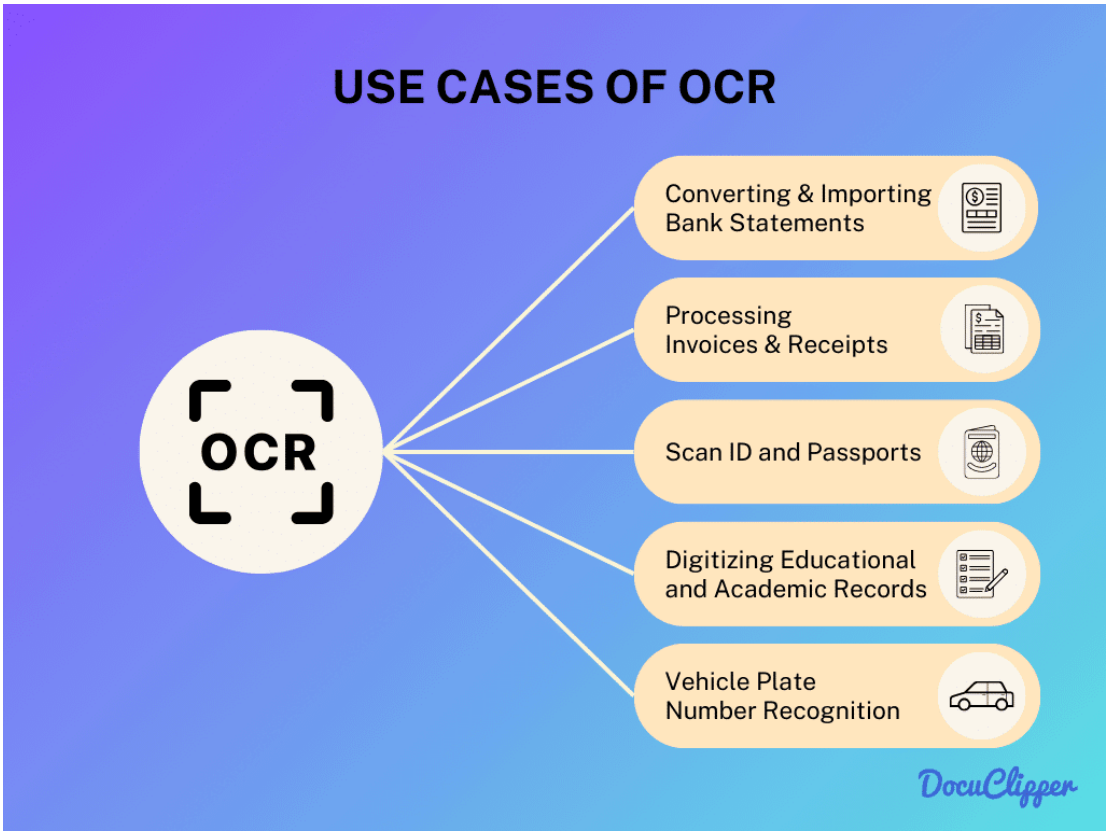
`Accessibility:` OCR technology helps visually impaired individuals by converting written content into spoken words using text-to-speech technologies, thus making information more accessible.

`Translation and Language Recognition:` OCR can be used to translate documents by first converting scanned text to digital text, which can then be processed by translation software.

`Search and Retrieval:` OCR makes it possible to search for specific texts within a large volume of scanned documents, which is useful in legal, medical, and academic fields.

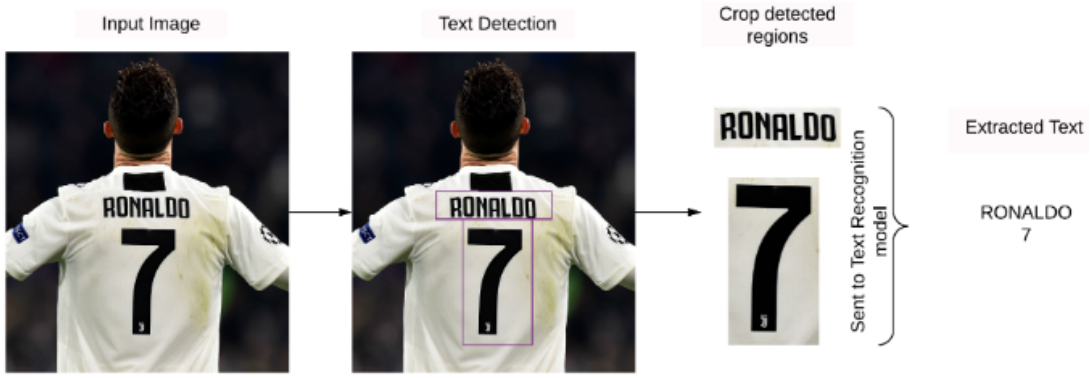
`Traffic and License Plate Recognition:` In the public safety sector, OCR technology is employed to automatically read vehicle license plates for traffic management and security purposes.

By converting static images of text into editable formats, OCR plays a critical role in numerous applications, enhancing efficiency and accessibility across various industries.




## Exploring OCR Engine

Text extraction can be achieved in two steps, i.e., `text detection` and `text recognition` or by training a single model to achieve both text detection and recognition. I'll me explaining the 2 steps process.



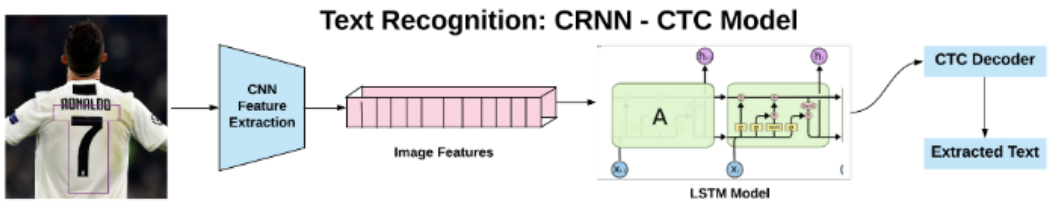

*`Text detection`* helps identify the region in the image where the text is present. It takes in an image as an input, and the outputs bounding boxes.

*`Text recognition`* extracts the text from the input image using the bounding boxes obtained from the text detection model. It takes in an image and some bounding boxes as inputs and outputs some raw text.

Text detection is very similar to the object detection task where the object which needs to be detected is nothing but the text. We can use algorithms like RCNN, Faster-RCNN.

SSD, YOLO. Here, we will mainly focus on explaining the CRNN-CTC network for text recognition.

## Text Recognition Pipeline

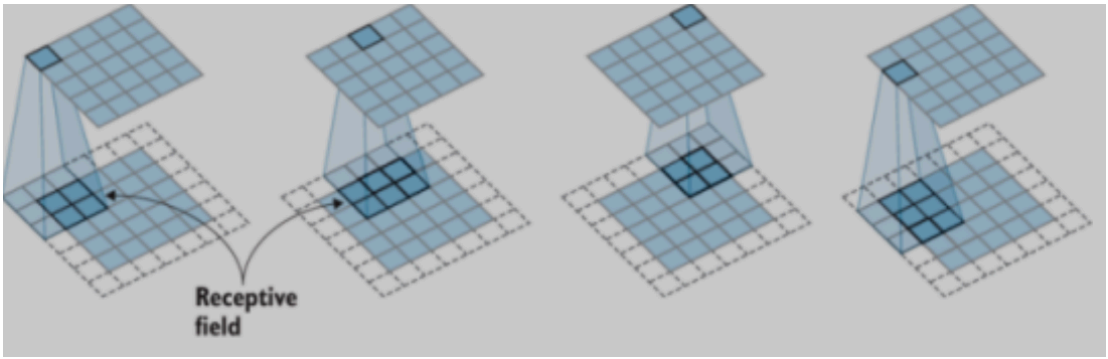



After the text detection step, regions, where the text is present, are cropped and sent through convolutional layers to get the features from the image. Later these features are fed to many-to-many LSTM architecture, which outputs softmax probabilities over the vocabulary. These outputs from different time steps are fed to the CTC decoder to finally get the raw text from images.

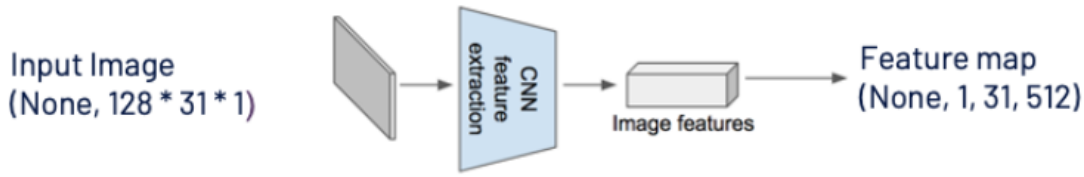
### `CNN Features and Receptive Field`

In CNN the kernel slides over the original image pixel by pixel and does some math calculations to get the values of the new "convolved" image on the next layer. The area of the image that the filter convolves is called the `receptive field`.

In the initial layers, the receptive field is small because each output feature only sees a small patch of the input image directly through the filter applied. As you progress deeper into the network, each layer aggregates the information from previous layers, expanding the area of the input image that influences the output features. Thus, deeper layers have a larger receptive field and capture more global information rather than local details




A grayscale image with width 128 and height 32 is sent through a series of convolutional & max-pooling layers. Layers are designed in such a manner that we obtain feature maps of the shape (None, 1, 31, 512) . "None" here is nothing but the batch size which could take any value.



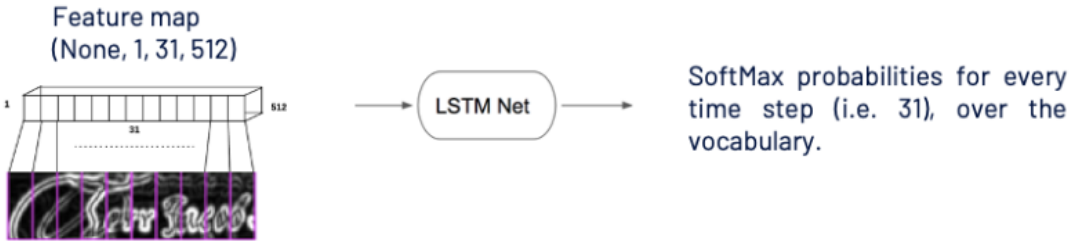

In convolutional networks, you look at an image through a smaller window and move that window to the right and down. That way you can find features in that window, for example a horizontal line or a vertical line or a curve etc... What exactly a convolutional neural network considers an important feature is defined while learning.

Wherever you find those features, you report that in the feature maps. A certain combination of features in a certain area can signal a larger, more complex feature exists there.

For example, your first feature map could for example looks for curves. The next feature map could look at a combination of curves that build circles. The next feature map could detect a bicycle from lines and circle features.

### `CNN Features to LSTM Model`

(None, 1, 31, 512) can be easily reshaped to (None, 31, 512) , and 31 corresponds to the number of time steps, and 512 is nothing but the number of features at every time step. One can relate this to training any LSTM model with word embeddings like word2vec, Glove, fastText, and the input shape is usually like (batch_size, no_time_steps, word_embedding_dimension).




Feature maps coming from CNN layers are fed to the LSTM model, as shown above. You might be thinking now that LSTM models are known to work with sequential data and how feature maps are sequential!! Receptive fields play a significant role here. As one can see from the above image first value(first row, first column) in the feature map has visibility on the left part of the input image and last value(first row, last column) has visibility on the end part of the image, and yes this is sequential !! From the LSTM model for every time step i.e., 31, we get a softmax probability over vocabulary. Now let us move on to the exciting part, calculating the loss value for this architecture setup.

### `CTC(Connectionist Temporal Classification) Loss`

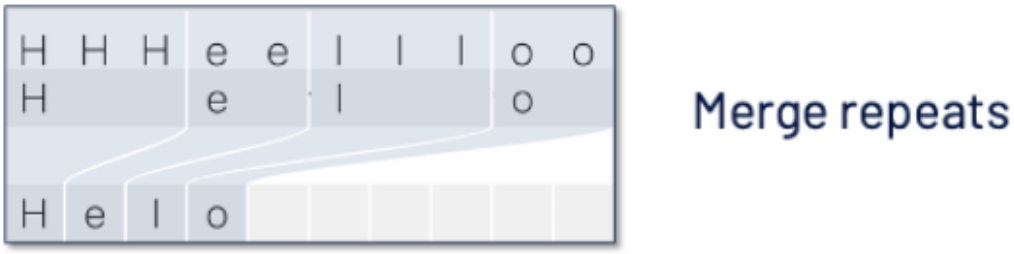
Connectionist Temporal Classification (CTC) loss is a popular method used in machine learning for training sequence-to-sequence models, especially when the alignment between the inputs and the outputs is unknown. It's particularly useful in tasks like

speech recognition, handwriting recognition, and other forms of sequence prediction where the timing of the output labels relative to the input sequence is not predefined.


Output from LSTM model for 31 timesteps ..

| Time step | t1 | t2 | t3 | t4 | t5 | ... | ... . | t27 | t28 | t29 | t30 | t31 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Prediction | H | H | H | e | e | ... | ... | l | o | o | o | o |

The length of ground truth is 5, which is not equal to the length of prediction i.e., 31.



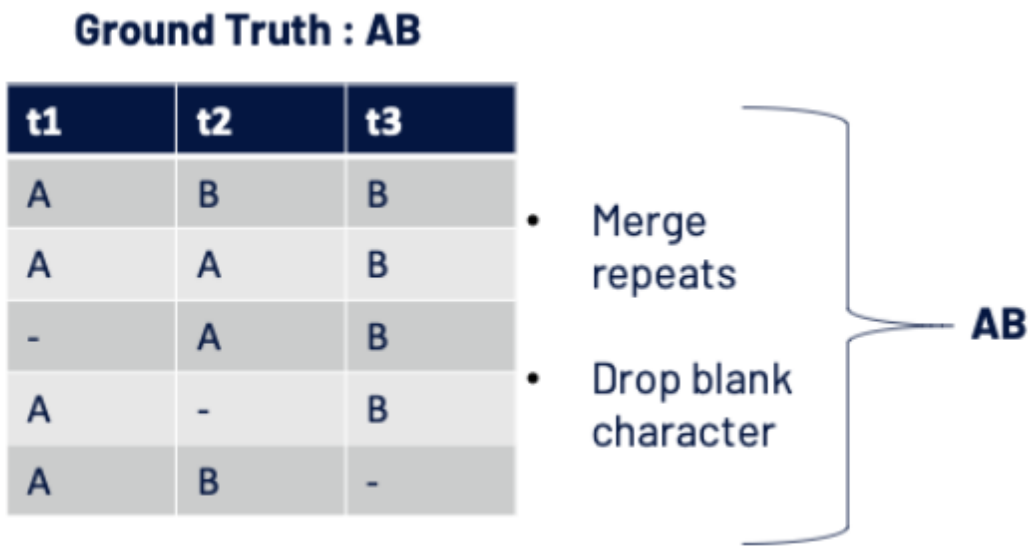

If we merge the repeats, we lose the repetitions, as shown in the above image. With just merging, we end up with a single letter "l," which was supposed to be "ll." So a special character called "blank character" is introduced to avoid this.


Now the decode operation consists of 2 steps:

1. Merge repeats
2. Remove blank characters. Now you can see "ll," which is retained.

For simplicity lets say, the vocabulary is { A, B, - }

we have predictions for 3-time steps from LSTM network (SoftMax probabilities over vocabulary at t1, t2, t3 )




Let us say the Softmax probabilities for 3-time steps are as below:

SoftMax probabilities

| | t1 | t2 | t3 |
|---|---|---|---|
| A | 0.8 | 0.7 | 0.1 |
| B | 0.1 | 0.1 | 0.8 |
| - | 0.1 | 0.2 | 0.1 |

Loss is calculated as - log(Probability of getting ground truth) Probability of getting GT AB: = P(ABB) + P(AAB) + P(-AB) + P(A-B) + P(AB-) Score for one path: AAB = (0.8 x 0.7 x 0.8) and similarly for other paths.

Creating an OCR engine is a challenging research task and requires great knowledge in image processing, feature extraction and machine learning. However, there are several open source projects that provide OCR framework and are widely used in the creation of OCR-related applications. In order not to reinvent the wheel and also to save time for development

## OCR using pytesseract

In [ ]:
```
import cv2
import matplotlib.pyplot as plt
```

In [ ]:
```
import pytesseract
```

In [ ]:
```
# Load the image
# img = cv2.imread("data\\01_raw_data\\bibek_pan.jpg")
img = cv2.imread("data\\01_raw_data\\sample_image1.jpg")
```

```
# displaying image using matplotlib
rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
plt.imshow(rgb_img)
plt.axis('off')
plt.show()
```


In [ ]:
```
gray_img =  cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blurred =  cv2.GaussianBlur(gray_img, (3, 3), 0)
adp_thresh = cv2.adaptiveThreshold(blurred, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2

contours, _ = cv2.findContours(adp_thresh, mode = cv2.RETR_EXTERNAL, method = cv

# Select the largest contour (assuming the ID card is the largest object)
largest_contour = None
largest_area = 0
for cnt in contours:
    area = cv2.contourArea(cnt)
    if area > largest_area:
        largest_contour = cnt
        largest_area = area

x, y, w, h = cv2.boundingRect(largest_contour)

print("contours", (x, y, w, h))
print("Area", largest_area)
contour_id = rgb_img[y:y+h, x:x+w]

plt.imshow(contour_id)
plt.axis('off')
plt.show()
```

```
contours (133, 55, 743, 465)
Area 335355.0
```


In [ ]:
```
cv2.imwrite("data\\02_intermediate_data\contour_id.jpg", cv2.cvtColor(contour_id
```

Out[ ]:
```
True
```

In [ ]:
```
gray_contour_id =  cv2.cvtColor(contour_id, cv2.COLOR_RGB2GRAY)
```

In [ ]:
```
# Check if the image was loaded successfully
# if img is None:
#     print(f"Error: Unable to load image at")
# else:
#     # Convert the image to grayscale for better OCR results
#     gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#     # Apply thresholding to preprocess the image
#     # cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU can be replaced with other metho
#     # _, threshold_img = cv2.threshold(gray_img, 0, 255, cv2.THRESH_BINARY_INV
#     # plt.imshow(threshold_img, cmap = 'gray')
#     # plt.show()

#     # Extract the text from the preprocessed image
#     # psm 6 stands for assuming a single uniform block of text.
#     custom_config = r'--oem 3 --psm 3 -l eng+osd --tessdata-dir "C:\\Program F
#     text = pytesseract.image_to_string(gray_img, config=custom_config)

#     # Print the extracted text to the console
#     print(text)
```

`Tesseract Configuration (custom_config)`

--oem 3: Sets the OCR Engine Mode to 3, which means Tesseract will use both its LSTM and legacy engines to process the image. This is typically used to get the most accurate results.

--psm 3: Sets the Page Segmentation Mode to 3, indicating that Tesseract should treat the image as a single uniform block of text. This mode assumes a single column of text of variable sizes.

-l eng+osd: Specifies the language models to be used. Here, eng is for English, and osd is for orientation and script detection, which helps in determining text direction.

--tessdata-dir "C:\Program Files\Tesseract-OCR\tessdata": Directs Tesseract to use a specific directory for its training data files. This is useful when you have custom training data or multiple versions of tessdata.

-c tessedit_char_whitelist=abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ012 This restricts the characters Tesseract will recognize to alphanumeric characters (both upper and lower case). This is useful for filtering out any symbols or punctuation marks that are not needed.

In [ ]:
```
custom_config = r'--oem 3 --psm 3 -l eng+osd --tessdata-dir "C:\\Program Files\\
text = pytesseract.image_to_string(gray_contour_id, config=custom_config)

print(text)
```

```
FNAAaT

arfrtayferargt
AbhishekTiwari
SteatfefDOB07052002

lon Ge4MALE

AZ

a

817329973675
```

In [ ]:
```
# Use pytesseract to detect text and its bounding boxes
data = pytesseract.image_to_data(gray_contour_id, config=custom_config, output_t

# Iterate through each detected text item
for i in range(len(data['text'])):
    if int(data['conf'][i]) > 60:  # Only consider boxes with a confidence > 60
        x, y, w, h = data['left'][i], data['top'][i], data['width'][i], data['he
        cv2.rectangle(contour_id, (x, y), (x + w, y + h), (0, 255, 0), 2)  # Dra

# Display the image with bounding boxes
plt.imshow(contour_id)
plt.axis('off')
plt.show()
```


In [ ]:
```
# data['text']
```

## OCR using PaddleOCR

Due to its high accuracy and good speed, CRNN is an optimal choice for OCR. Latest libraries like Easy-ocr, Keras-ocr and PaddleOCR are based on CRNN and provide easy-to-use pretrained models.

PaddleOCR supports more than 80 languages (depending upon the OCR algorithm used). But the flagship PP-OCR provides support for both Chinese and English languages. The flagship OCR algorithm PP-OCR is one of the best OCR tools available. So far, It has three versions as of now PP-OCR, PP-OCRv2 and PP-OCRv3. All of these models are built on CRNN as seen in the previous section and are ultra-lightweight. Let's take a look and apply it to some of the various types of scenarios.

`PaddleOCR Initialization Parameters`

use_angle_cls=True: This parameter tells PaddleOCR to use angle classification, which is useful when the text in the images could be oriented in different directions. With use_angle_cls set to True, the model will attempt to correct the orientation of the text before recognizing it, which generally improves the accuracy of text recognition.

use_gpu=False: This parameter specifies whether to use a GPU for processing. Setting use_gpu=False forces the tool to use the CPU instead. Using the GPU can significantly speed up OCR processing, especially for large volumes of images or high-resolution images, but it requires a compatible GPU and proper configuration. Setting it to False is useful if you are running the software on a system without a GPU or you want to ensure compatibility without requiring GPU resources.

In [ ]:
```
from paddleocr import PaddleOCR, draw_ocr
ocr = PaddleOCR(use_angle_cls=True, use_gpu=False)
def show_ocr(ocr, img_path, font):
```

```
    result = ocr.ocr(img_path)
    # save_path = os.path.join(out_path, img_path.split('/')[-1] + 'output')

    image = cv2.imread(img_path)

    boxes = [line[0] for line in result[0]]
    txts = [line[1][0] for line in result[0]]
    scores = [line[1][1] for line in result[0]]

    im_show = draw_ocr(image, boxes, txts, scores, font_path=font)

    # cv2.imwrite(save_path, im_show)

    img = cv2.cvtColor(im_show, cv2.COLOR_BGR2RGB)
    plt.imshow(img)
    plt.axis('off')
    # return result
```

In [ ]:
```
img_path = "data\\02_intermediate_data\\contour_id.jpg"
font = "data\\01_raw_data\\simfang.ttf"
show_ocr(ocr, img_path, font)
```

```
[2024/04/28 12:18:08] ppocr DEBUG: dt_boxes num : 15, elapsed : 0.107684612274169
92
[2024/04/28 12:18:09] ppocr DEBUG: cls num  : 15, elapsed : 0.0500185489654541
[2024/04/28 12:18:09] ppocr DEBUG: rec_res num  : 15, elapsed : 0.500306129455566
4
```


## Keras OCR

In [ ]:
```
import keras_ocr
import matplotlib.pyplot as plt
```

In [ ]:
```
pipeline = keras_ocr.pipeline.Pipeline()
# Read images from folder path to image object
images = [
    keras_ocr.tools.read(img) for img in ['data\\01_raw_data\\pan_2.jpg',
                                          'data\\01_raw_data\\pan_3.webp',]
]
```

```
Looking for C:\Users\rauth\.keras-ocr\craft_mlt_25k.h5
Looking for C:\Users\rauth\.keras-ocr\crnn_kurapan.h5
```

In [ ]:
```
# generate text predictions from the images
prediction_groups = pipeline.recognize(images)

# plot the text predictions
fig, axs = plt.subplots(nrows=len(images), figsize=(10, 20))
```

```
for ax, image, predictions in zip(axs, images, prediction_groups):
    keras_ocr.tools.drawAnnotations(image=image,
                                    predictions=predictions,
                                    ax=ax)
```

```
1/1 [==============================] - 7s 7s/step
2/2 [==============================] - 5s 788ms/step
```



## Using easyOCR

In [ ]:
```
import easyocr
```

In [ ]:
```
img_path = 'data\\02_intermediate_data\\contour_id.jpg'
reader = easyocr.Reader(['en'])
```

```
result = reader.readtext(img_path)
confidence_threshold =  .80
filtered_text = ""  # Initialize an empty string to store filtered text
for text in result:
    bounding_box, recognized_text, confidence = text
    if confidence > confidence_threshold:
        filtered_text += recognized_text + "|"  # Append filtered text with newl
```

```
Neither CUDA nor MPS are available - defaulting to CPU. Note: This module is much
faster with a GPU.
```

In [ ]:
```
filtered_text
```

Out[ ]:
```
'Abhishek Tiwari|DOB|07/05/2002|MALE|8|'
```