```
import cv2
import numpy as np
from imutils.object_detection import non_max_suppression
img = cv2.imread('D:/downloads/OCR_LNP.PNG')

## ------------- Load the pre-trained models ---------------

model = cv2.dnn.readNet('D:/downloads/frozen_east_text_detection.pb')
model1 = cv2.dnn.readNet('D:/downloads/crnn.onnx')
## ------------- Prepare the image ----------------
# use multiple of 32 to set the new img shape
height, width, _ = img.shape
new_height = (height//32)*32
new_width = (width//32)*32
print(new_height, new_width)
# get the ratio change in width and height
h_ratio = height/new_height
w_ratio = width/new_width
print(h_ratio, w_ratio)
blob = cv2.dnn.blobFromImage(img, 1, (new_width, new_height),(123.68, 116.78, 103.94),
True, False)

## ------------- Forward Propagation -------------

model.setInput(blob)
(geometry, scores) = model.forward(model.getUnconnectedOutLayersNames())
## ------------- Post-Processing ----------------
rectangles = []
confidence_score = []
for i in range(geometry.shape[2]):
 for j in range(0, geometry.shape[3]):

 if scores[0][0][i][j] < 0.1:
 continue

 bottom_x = int(j*4 + geometry[0][1][i][j])
 bottom_y = int(i*4 + geometry[0][2][i][j])

 top_x = int(j*4 - geometry[0][3][i][j])
 top_y = int(i*4 - geometry[0][0][i][j])

 rectangles.append((top_x, top_y, bottom_x, bottom_y))
 confidence_score.append(float(scores[0][0][i][j]))
# use Non-max suppression to get the required rectangles
fin_boxes = non_max_suppression(np.array(rectangles), probs=confidence_score,
overlapThresh=0.5)

## ---------------- Load the CRNN decoding functions -------------

def most_likely(scores, char_set):
 text = ""
 for i in range(scores.shape[0]):
 c = np.argmax(scores[i][0])
 text += char_set[c]
 return text
def map_rule(text):
 char_list = []
 for i in range(len(text)):
 if i == 0:
 if text[i] != '-':
 char_list.append(text[i])
 else:
 if text[i] != '-' and (not (text[i] == text[i - 1])):
 char_list.append(text[i])
 return ''.join(char_list)
def best_path(scores, char_set):
 text = most_likely(scores, char_set)
 final_text = map_rule(text)
 return final_text
alphabet_set = "0123456789abcdefghijklmnopqrstuvwxyz"
blank = '-'
char_set = blank + alphabet_set

## ---------------- Recognize the text using CRNN in each segment -----------
```

```
img_copy = img.copy()
for (x1, y1, x2, y2) in fin_boxes:
 x1 = int(x1 * w_ratio)
 y1 = int(y1 * h_ratio)
 x2 = int(x2 * w_ratio)
 y2 = int(y2 * h_ratio)

 segment = img[y1:y2, x1:x2, :]

 segment_gray = cv2.cvtColor(segment, cv2.COLOR_BGR2GRAY)
 blob = cv2.dnn.blobFromImage(segment_gray, scalefactor=1/127.5, size=(100,32), mean=127.5)

 model1.setInput(blob)
 scores = model1.forward()
 text = best_path(scores, char_set)
 cv2.rectangle(img_copy, (x1, y1), (x2, y2), (0, 255, 0), 2)
 cv2.putText(img_copy, text.strip(), (x1,y1-2), cv2.FONT_HERSHEY_COMPLEX, 0.7,
(0,0,255),2)cv2.imshow("Text Detection", img_copy)cv2.waitKey(0)
```