Pooling:  
Technique used in CNN to reduce spatial dimensions of feature map produced by convolution network.

Convolution:  
Mathematical operation that express amount of overlap of one function as it shifted over another function.

CNN:  
Type of deep learning model that is used for image classification task.

Works by learning set of filters that can be applied to input image to extract features.

Filter learned through back propagation, which involves adjusting weight of filter to minimize error between predicted output and actual output.

Consists of convolution layer, pooling and fully connected layers.

Convolution Layer:  
Responsible for learning filters.

Pooling Layer:  
Down samples output of convolution layer to reduce dimensionality of feature map.

Fully Connected Layer:  
Used to classify input image based on learned feature.

Back Propagation:  
Algorithm used to train model by adjusting weight.

Supervised learning algorithm.

Propagated error backwards through network, from output layer to input layer weight based on error.

Forward Pass: input data feed and output are calculated by apply weight to input data and passing through activation function.

Error Calculation: Predicated output – actual output. (Propagated backward)

Backward Pass: Error propagated backward and weight of neurons error.

Repeat: Repeat for each input and training dataset until error is minimized.

Weight are adjusted using optimization algorithm:  
- Stochastic Gradient Descent (SGD)  
-ADAM (Adaptive Moment Estimation)

SDG:  
Algorithm used to train ML model by minimizing lose function.

Works by updating model weight in opposite direction of gradient of loss function with respect to weight.

Learning Rate determines size of step to reach optimal weight.

ADAM:  
Extension to SDG that computes individual adaptive learning rate for difficult parameter.

Computation efficiency.

**Q) What is the difference between yolov8n, yolov8s, yolov8m, yolov8l,yolov8x?**

All of these models belong to the YOLOv8 family, a collection of state-of-the-art real-time object detectors. However, each variant offers different trade-offs between accuracy, speed, and model size:

Number of letters:

n: smallest model, fastest inference but lowest accuracy

s: small model, good balance of speed and accuracy

m: medium model, higher accuracy than small models with moderate inference speed

l: large model, highest accuracy but slowest inference

x: extra-large model, best accuracy for resource-intensive applications

coco\_model = YOLO('yolov8x.pt')

np\_model = YOLO('../model/runs/detect/train/weights/best.pt')

coco\_model: This YOLO model (coco\_model) is initialized using the weights file named 'yolov8n.pt'. The weights file likely contains pre-trained weights on the COCO dataset.

np\_model: This YOLO model (np\_model) is initialized using a different weights file located at '../model/runs/detect/train/weights/best.pt'. The path suggests that these weights may be specific to your use case or have been fine-tuned for a particular task.

videos = glob('./inputs/embossed.mp4')

print(videos)

glob('./inputs/embossed.mp4'): This line uses the glob function to find files matching the pattern './inputs/embossed.mp4'. The glob function returns a list of pathnames that match the specified pattern.

videos: This variable stores the list of video files obtained from the glob function. In this case, it appears you are expecting only one video file to match the pattern.

print(videos): This line prints the list of video files to the console.

It's important to note that the glob function supports wildcard patterns, so if you have multiple video files matching the pattern, they will all be included in the list. If you want to process all video files in a directory, you might use a wildcard like '\*.mp4' instead of specifying a specific file name.

video = cv.VideoCapture(videos[0])

**cv.VideoCapture(videoPath)** creates a **VideoCapture** object for the video file specified by the path. Once the **VideoCapture** object is created, we can use it to read each frame from the video, perform various operations on each frame, and analyze or process the video.

ret = True

In OpenCV, the variable **ret** is a boolean that represents whether a frame was successfully read from the video source (file or camera). When **ret** is *True*, it indicates that the frame was successfully read. Conversely, when **ret** is *False*, it means that there are no more frames to be read, or there was an error in reading the frame.

frame\_number = -1

Setting **frame\_number** to -1 in the context of video processing typically means that you want to read the next frame from the video source. The frame number is often used to navigate to a specific frame in a video. If **frame\_number** is -1, it's a signal to read the next frame in sequence.

vehicles = [2,3,5]

We can directly detect multiple vehicles in a single frame like car, motorbike, truck using the COCO dataset. The COCO dataset has a list of vehicle class IDs. Each vehicle class has a unique ID. For example, car is 2, motorbike is 3, truck is 5. We can search this information in <https://docs.ultralytics.com/datasets/detect/coco/#dataset-yaml>

while ret:

    ret, frame = video.read()

    frame\_number += 1

    if ret:

        results[frame\_number] = {}

This part of the code is a loop that reads frames from the video using video.read() until there are no more frames (ret becomes False). Let's break down this loop:

ret, frame = video.read(): This line reads a frame from the video using the read() method. It returns two values: ret (a boolean indicating whether the frame was successfully read) and frame (the actual frame).

frame\_number += 1: This line increments the frame\_number variable, indicating the current frame number.

if ret:: This condition checks if the frame was successfully read. If ret is True, it means a frame was read, and the code inside the if block will be executed.

results[frame\_number] = {}: This line initializes an empty dictionary at the current frame\_number in the results dictionary. This dictionary will be used to store information about detections in the current frame.

The loop continues until there are no more frames (ret becomes False). Inside the loop, you perform vehicle and license plate detection for each frame and store the results in the results dictionary.

detections = coco\_model.track(frame, *persist*=True)[0]

coco\_model.track(frame, persist=True): This is calling a method or function named track on the coco\_model object. It performs object tracking on the input frame. The frame is passed as an argument to this method.

[0]: The result of this method call is likely a list or iterable containing tracking information. [0] is used to get the first element of this list. The method returns a list, and the tracked objects are in the first element of that list.

So, after this line, detections hold information about the tracked objects in the current frame. It includes information like the bounding boxes, track IDs, scores, class IDs, etc., for the detected objects in the frame.

 for detection in detections.boxes.data.tolist():

            x1, y1, x2, y2, track\_id, score, class\_id = detection

            if int(class\_id) in vehicles and score > 0.5:

                vehicle\_bounding\_boxes = []

                vehicle\_bounding\_boxes.append([x1, y1, x2, y2, track\_id, score])

In this part of the code, you are iterating over the detections obtained from coco\_model.track() and filtering them based on certain criteria.  
for detection in detections.boxes.data.tolist()::  
This loop iterates over each detection in the list of boxes obtained from detections.

x1, y1, x2, y2, track\_id, score, class\_id = detection:  
This line unpacks the values in each detection box into individual variables for easier access. The values typically include coordinates (x1, y1, x2, y2), a track ID, a confidence score, and a class ID.

if int(class\_id) in vehicles and score > 0.5::  
This condition checks whether the class ID is in the list of vehicles and if the confidence score (score) is greater than 0.5. If both conditions are true, it means the detected object is a vehicle with a confidence score above the specified threshold.

vehicle\_bounding\_boxes = []:  
This line initializes an empty list to store the bounding box information of the detected vehicles.

vehicle\_bounding\_boxes.append([x1, y1, x2, y2, track\_id, score]):  
If the conditions are met, the bounding box information for the current detection is appended to the vehicle\_bounding\_boxes list.

for bbox in vehicle\_bounding\_boxes:

                    print(bbox)

                    roi = frame[int(y1):int(y2), int(x1):int(x2)]

for bbox in vehicle\_bounding\_boxes:: This loop iterates over each bounding box in vehicle\_bounding\_boxes.

print(bbox):  
This line prints the bounding box information. The information includes x1, y1, x2, y2, track\_id, and score.

x1, y1, x2, y2, track\_id, score = bbox:  
This line unpacks the values in the bounding box into individual variables for easier access. The variables now hold the coordinates (x1, y1, x2, y2), the track ID, and the confidence score.

roi = frame[int(y1):int(y2), int(x1):int(x2)]:  
This line extracts the region of interest (ROI) from the current frame using the bounding box coordinates. It's creating a sub-image from the frame that corresponds to the region where the detected vehicle is.

license\_plates = np\_model(roi)[0]

np\_model(roi):  
This code is applying the np\_model to the region of interest (roi). The result is likely a list or iterable containing information about detected license plates.

[0]:  
This is used to get the first element of the result. If the np\_model returns a list of predictions, [0] is used to access the first prediction.

So, license\_plates now hold information about the detected license plates in the region of interest.

for license\_plate in license\_plates.boxes.data.tolist():

                        plate\_x1, plate\_y1, plate\_x2, plate\_y2, plate\_score, \_ = license\_plate

for license\_plate in license\_plates.boxes.data.tolist()::

This loop iterates over each detected license plate in license\_plates.

plate\_x1, plate\_y1, plate\_x2, plate\_y2, plate\_score, \_ = license\_plate:

This line unpacks the values in each detected license plate box into individual variables for easier access. The variables now hold the coordinates of the license plate bounding box (plate\_x1, plate\_y1, plate\_x2, plate\_y2), the confidence score (plate\_score), and possibly other information (ignored using \_).

plate = roi[int(plate\_y1):int(plate\_y2), int(plate\_x1):int(plate\_x2)]

plate = roi[int(plate\_y1):int(plate\_y2), int(plate\_x1):int(plate\_x2)]:

This line extracts the license plate region from the ROI using the previously obtained bounding box coordinates (plate\_x1, plate\_y1, plate\_x2, plate\_y2). It uses integer casting to ensure that the coordinates are treated as integers.

cv.imwrite('outputs/plates/roi/'+str(track\_id)+ '.jpg', plate):

This line uses OpenCV's imwrite function to save the extracted license plate as an image file. The file is saved in a directory named 'outputs/plates/roi/' with the filename as str(track\_id) + '.jpg'. This assumes that track\_id is a unique identifier for the current detected vehicle.

 plate\_gray = cv.cvtColor(plate, cv.COLOR\_BGR2GRAY)

 \_, plate\_treshold = cv.threshold(plate\_gray, 64, 255, cv.THRESH\_BINARY\_INV)

plate\_gray = cv.cvtColor(plate, cv.COLOR\_BGR2GRAY):

This line converts the license plate image (plate) from BGR (color) to grayscale using OpenCV's cvtColor function. Grayscale simplifies the image to a single channel, which is often useful for various image processing tasks.

\_, plate\_threshold = cv.threshold(plate\_gray, 64, 255, cv.THRESH\_BINARY\_INV):

This line applies a binary thresholding operation to the grayscale license plate image (plate\_gray). Pixels with intensity values less than 64 are set to 0, and pixels with values greater than or equal to 64 are set to 255. The cv.THRESH\_BINARY\_INV flag inverts the binary threshold, making foreground pixels (the license plate characters) white and background pixels black.

The underscore (\_) is often used as a placeholder for a variable that you don't intend to use. In this case, the threshold value returned by cv.threshold is not used, so it's assigned to \_ to indicate that it's intentionally ignored.

np\_text, np\_score = read\_license\_plate(plate\_treshold)

read\_license\_plate(plate\_threshold): This line calls the read\_license\_plate function, passing the thresholded license plate image (plate\_threshold) as an argument. The function is expected to return information about the recognized text and possibly a confidence score.

np\_text, np\_score = ...: This line unpacks the return values from the read\_license\_plate function into two variables, np\_text and np\_score. This assumes that the function returns a tuple or a pair of values representing the recognized text and the associated confidence score.

def read\_license\_plate(*license\_plate\_crop*):

    detections = reader.readtext(*license\_plate\_crop*)

    for detection in detections:

        bbox, text, score = detection

        text = text.upper().replace(' ', '')

*# verify that text is confirmed to a standard license plate*

        if license\_complies\_format(text):

*# bring text into the default license plate format*

            return format\_license(text), score

    return None, None

reader.readtext(license\_plate\_crop): This line uses the reader.readtext function to perform OCR on the license plate image (license\_plate\_crop). It returns a list of detections, where each detection includes the bounding box (bbox), recognized text (text), and a confidence score (score).

for detection in detections:: This loop iterates over each detection obtained from the OCR.

bbox, text, score = detection: This line unpacks the values from each detection into individual variables for easier access.

text = text.upper().replace(' ', ''): This line converts the recognized text to uppercase and removes any spaces. This is likely done to standardize the format of the license plate text.

if license\_complies\_format(text):: This condition checks whether the processed text complies with a certain format. If it does, the text is further processed.

return format\_license(text), score: If the text complies with the format, the function returns the formatted license plate text and the confidence score.

return None, None: If no valid license plate is found, the function returns None for both the license plate text and the confidence score.

reader = easyocr.Reader(['en'], *gpu*=True)

Initialize the OCR reader. EasyOCR is an open-source library for optical character recognition (OCR) that is used for text recognition. 'en' is the language of the OCR reader.

dict\_char\_to\_int = {'O': '0',

                    'I': '1',

                    'J': '3',

                    'A': '4',

                    'G': '6',

                    'S': '5'}

dict\_int\_to\_char = {'0': 'O',

                    '1': 'I',

                    '3': 'J',

                    '4': 'A',

                    '6': 'G',

                    '5': 'S'}

Here we are mapping dictionaries for character conversion.If we know that the first character in the number plate always is an string eg. `O` then if our OCR reader reads that O as `0` then it would be a mistake. To prevent this we are mapping dictionaries with similar keys and values.

def license\_complies\_format(*text*):

    if len(*text*) != 7:

        return False

    if (*text*[0] in string.ascii\_uppercase or *text*[0] in dict\_int\_to\_char.keys()) and \

       (*text*[1] in string.ascii\_uppercase or *text*[1] in dict\_int\_to\_char.keys()) and \

       (*text*[2] in string.ascii\_uppercase or *text*[2] in dict\_int\_to\_char.keys()) and \

       (*text*[3] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or *text*[3] in dict\_char\_to\_int.keys()) and \

       (*text*[4] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or *text*[4] in dict\_char\_to\_int.keys()) and \

       (*text*[5] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or *text*[5] in dict\_char\_to\_int.keys()) and  \

       (*text*[6] in ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9'] or *text*[6] in dict\_char\_to\_int.keys()):

        return True

    else:

        return False

license\_complies\_format is a function that checks if the license plate complies with the specified format. In this case the format is `[A-Z][A-Z][A-Z][0-9][0-9][0-9]`. We can change this format for specific use cases. For example, now it is configured for Nepali Embossed number plates.

The above character conversion comes handy in this situation where if we are sure that in the second letter of our text we should get a string then if our OCR Reader reads a integer that looks similar to an alphabet maybe 4 then we can neglect the '4 and read 'A' instead.

def format\_license(*text*):

    license\_plate\_ = ''

    mapping = {0: dict\_int\_to\_char, 1: dict\_int\_to\_char, 2: dict\_int\_to\_char, 3: dict\_char\_to\_int, 4: dict\_char\_to\_int,

               5: dict\_char\_to\_int, 6: dict\_char\_to\_int}

    for j in [0, 1, 2, 3, 4, 5, 6]:

        if *text*[j] in mapping[j].keys():

            license\_plate\_ += mapping[j][*text*[j]]

        else:

            license\_plate\_ += *text*[j]

    return license\_plate\_

license\_plate\_ = '': Initializes an empty string to store the formatted license plate.

mapping: A dictionary that maps the position in the license plate text to the corresponding conversion dictionary (dict\_int\_to\_char or dict\_char\_to\_int).

for j in [0, 1, 2, 3, 4, 5, 6]:: Iterates over each position in the license plate text.

if text[j] in mapping[j].keys():: Checks if the character at the current position (j) is present in the corresponding conversion dictionary. If it is, it appends the converted character to license\_plate\_.

else:: If the character is not in the dictionary, it means there's no conversion needed, so the original character is appended to license\_plate\_.

return license\_plate\_: Returns the formatted license plate.

This function essentially applies character conversion based on the specified mappings for each position in the license plate text. If a character at a certain position needs conversion, it uses the appropriate dictionary for that position.

if np\_text is not None:

  results[frame\_number][track\_id] = {

           'car': {

            'bbox': [x1, y1, x2, y2],

            'bbox\_score': score},

           'license\_plate': {

            'bbox': [plate\_x1, plate\_y1, plate\_x2, plate\_y2],

            'bbox\_score': plate\_score,

            'number': np\_text,

            'text\_score': np\_score

                                }

                            }

if np\_text is not None:: This condition checks whether np\_text contains a non-None value, meaning that the OCR process successfully recognized a license plate text.

results[frame\_number][track\_id]: If the condition is True, you are updating the results dictionary. The frame\_number is used as a key to access a dictionary representing a specific frame, and within that frame dictionary, the track\_id is used as a key to store information about a specific tracked object.

'car': This key represents information about the detected car, including the bounding box (bbox) and the confidence score (bbox\_score).

'license\_plate': This key represents information about the license plate, including the bounding box (bbox), confidence score (bbox\_score), recognized number (number), and the text score (text\_score).

So, if a license plate was successfully recognized (np\_text is not None), the relevant information is added to the results dictionary for the specific frame and tracked object.

write\_csv(results, './outputs/results.csv')

video.release()

write\_csv(results, './outputs/results.csv'): This line is calling a function named write\_csv with the results dictionary and a file path as arguments. It is using this function to write the results to a CSV file.

video.release(): This line releases the video capture object (video). It's essential to release the video capture object once you have finished using it to free up system resources.