1. Path Setups

In [1]:
```
import os
import glob
import cv2
import time
import numpy as np
import matplotlib.pyplot as plt

#from tensorflow.keras.preprocessing.image import ImageDataGenerator
#from tensorflow.keras.preprocessing.image import load_img, img_to_array

#from tensorflow.keras.utils import get_file
from PIL import Image, ImageDraw, ImageFont, ImageFilter

import easyocr
from exif import Image as exf

from IPython.display import clear_output
import warnings

%matplotlib inline
warnings.filterwarnings('ignore')
```

In [2]:
```
CUSTOM_MODEL_NAME = 'alpr_ssd_mobnet'
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'
```

In [3]:
```
paths = {
    'WORKSPACE_PATH': os.path.join('codes', 'workspace'),
    'SCRIPTS_PATH': os.path.join('codes','scripts'),
    'APIMODEL_PATH': os.path.join('codes','models'),
    'ANNOTATION_PATH': os.path.join('codes', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('codes', 'workspace','images'),
    'MODEL_PATH': os.path.join('codes', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('codes', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('codes', 'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('codes', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('codes', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('codes', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('codes','protoc')
 }
```

In [4]:
```
files = {
    'PIPELINE_CONFIG':os.path.join('codes', 'workspace','models', CUSTOM_MODEL_NAME, 'pipeline.config'),
    'TF_RECORD_SCRIPT': os.path.join(paths['SCRIPTS_PATH'], TF_RECORD_SCRIPT_NAME),
    'LABELMAP': os.path.join(paths['ANNOTATION_PATH'], LABEL_MAP_NAME)
}
```

2. Load Model

In [5]:
```
import tensorflow as tf
import object_detection
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util
```

In [6]:
```
# Load pipeline config and build a detection model
configs = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
detection_model = model_builder.build(model_config=configs['model'], is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-11')).expect_partial()

@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections
```

3. Image Preprocessing and Number Plate Recognition

In [7]:
```
category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
```

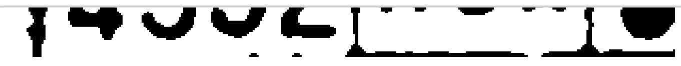
In [8]:
```
#ORIGINAL_IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', '*.jpg')
ORIGINAL_IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test')
origina_image_paths = list(glob.glob(ORIGINAL_IMAGE_PATH))
```

In [9]:
```
IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'plates', '*.jpg')
images_path= list(glob.glob(IMAGE_PATH))
```

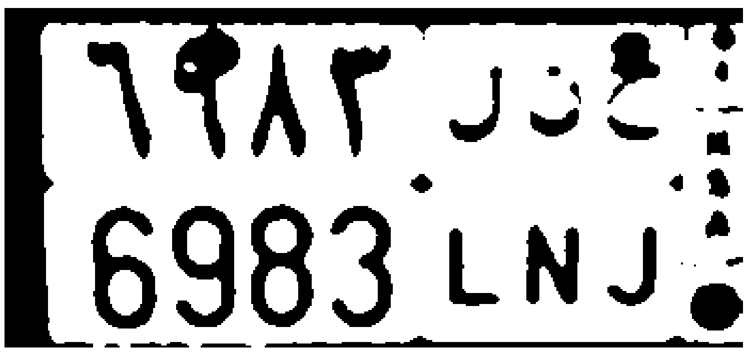
In [10]:
```
print(f'There are {len(images_path):,} images in the dataset')
```

```
There are 56 images in the dataset
```

In [11]:
```
reader = easyocr.Reader(['en'])
for i, image in enumerate(images_path):
    img = cv2.imread(image)
    img_scaled = cv2.resize(img, None, fx=4, fy=4, interpolation = cv2.INTER_LANCZOS4)
    gray_image = cv2.cvtColor(img_scaled, cv2.COLOR_BGR2GRAY)


    blur = cv2.GaussianBlur(gray_image, (7,7), 0)
    _, th3 = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

    image_np = np.array(th3)

    display(Image.fromarray(image_np))
    #reader = easyocr.Reader(['en'])
    ocr_result = reader.readtext(image_np)
    ocr_scores = [result[2] for result in ocr_result]

    with open(image, 'rb') as img_file:
        img = exf(img_file)

    if(img.has_exif):
        plate_text  = img.get('make')
        print(plate_text)

    if(len(ocr_scores) >= 1):
        plate_text = ocr_result[ocr_scores.index(max(ocr_scores))][1]
```



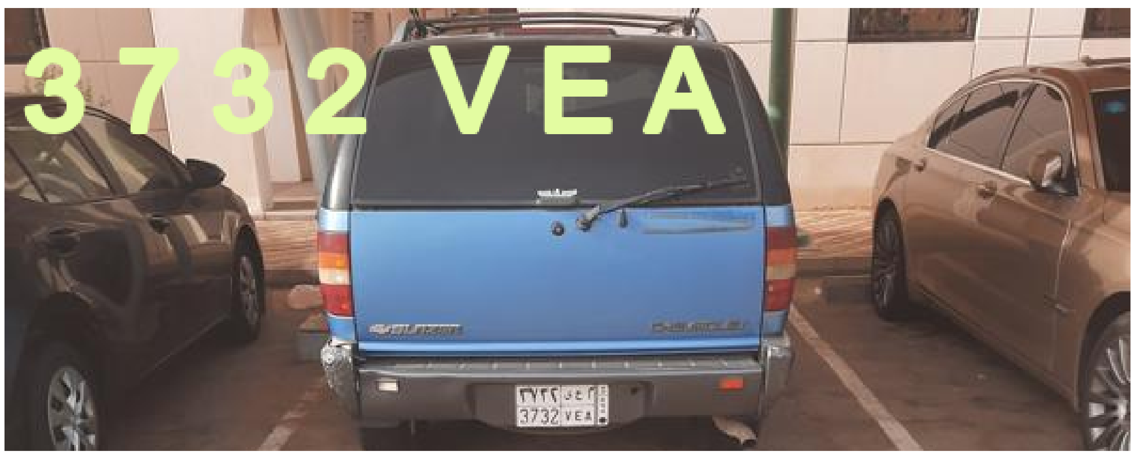
```
4 5 5 2  H J A
```



```
6 9 8 3  L N J
```

4. Number Plates on Original Image

In [12]:
```
for i, image in enumerate(images_path):
    file_name = os.path.basename(image)[1:]
    img =  Image.open(ORIGINAL_IMAGE_PATH+'/'+file_name)

    with open(image, 'rb') as img_file:
        imgp = exf(img_file)

    if(imgp.has_exif):
        plate_text  = imgp.get('make')

    draw = ImageDraw.Draw(img)
    font = ImageFont.truetype("arial.ttf", size = 60)
    draw.text((10,10),plate_text.upper(), font = font, stroke_width=2, fill="#E3FCA1")

    draw = ImageDraw.Draw(img)
    display(img)
```



5. Number Plates in Arabic

In [21]:
```
#%run -i arabic.py
```

In [20]:
```
for i, image in enumerate(images_path):
    file_name = os.path.basename(image)[1:]
    img =  Image.open(ORIGINAL_IMAGE_PATH+'/'+file_name)

    with open(image, 'rb') as img_file:
        imgp = exf(img_file)

    if(imgp.has_exif):
        plate_text  = imgp.get('make')

    arabic_text=""
    for x in plate_text:
        #print(arabic_dict[x])
        arabic_text = arabic_text + (arabic_dict[x])

    draw = ImageDraw.Draw(img)
    font = ImageFont.truetype("arial.ttf", size = 60)
    draw.text((10,10),plate_text.upper(), font = font, stroke_width=2, fill="#FF0B00")
    draw.text((10,80),arabic_text, font = font, stroke_width=2, fill="#FF0B00")

    draw = ImageDraw.Draw(img)
    display(img)
```