0. Setup Paths

In [3]:
```
import os
```

In [4]:
```
CUSTOM_MODEL_NAME = 'my_ssd_mobnet'
PRETRAINED_MODEL_NAME = 'ssd_mobilenet_v2_fpnlite_320x320_coco17_tpu-8'
PRETRAINED_MODEL_URL = 'http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_fp
TF_RECORD_SCRIPT_NAME = 'generate_tfrecord.py'
LABEL_MAP_NAME = 'label_map.pbtxt'
```

In [5]:
```
paths = {
    'WORKSPACE_PATH': os.path.join('Tensorflow', 'workspace'),
    'SCRIPTS_PATH': os.path.join('Tensorflow','scripts'),
    'APIMODEL_PATH': os.path.join('Tensorflow','models'),
    'ANNOTATION_PATH': os.path.join('Tensorflow', 'workspace','annotations'),
    'IMAGE_PATH': os.path.join('Tensorflow', 'workspace','images'),
    'MODEL_PATH': os.path.join('Tensorflow', 'workspace','models'),
    'PRETRAINED_MODEL_PATH': os.path.join('Tensorflow', 'workspace','pre-trained-models'),
    'CHECKPOINT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME),
    'OUTPUT_PATH': os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'export'),
    'TFJS_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfjsexport'),
    'TFLITE_PATH':os.path.join('Tensorflow', 'workspace','models',CUSTOM_MODEL_NAME, 'tfliteexport'),
    'PROTOC_PATH':os.path.join('Tensorflow','protoc')
 }
```

In [6]:
```
files = {
    'PIPELINE_CONFIG':os.path.join('Tensorflow', '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)
}
```

In [7]:
```
for path in paths.values():
    if not os.path.exists(path):
        if os.name == 'posix':
            !mkdir -p {path}
        if os.name == 'nt':
            !mkdir {path}
```

1. Download TF Models Pretrained Models from Tensorflow Model Zoo and Install TFOD

In [ ]:
```
# https://www.tensorflow.org/install/source_windows
```

In [ ]:
```
if os.name=='nt':
    !pip install wget
    import wget
```

In [ ]:
```
if not os.path.exists(os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection')):
    !git clone https://github.com/tensorflow/models {paths['APIMODEL_PATH']}
```

In [ ]:
```
# Install Tensorflow Object Detection
if os.name=='posix':
    !apt-get install protobuf-compiler
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && cp object_detect

if os.name=='nt':
    url="https://github.com/protocolbuffers/protobuf/releases/download/v3.15.6/protoc-3.15.6-win64.zip"
    wget.download(url)
    !move protoc-3.15.6-win64.zip {paths['PROTOC_PATH']}
    !cd {paths['PROTOC_PATH']} && tar -xf protoc-3.15.6-win64.zip
    os.environ['PATH'] += os.pathsep + os.path.abspath(os.path.join(paths['PROTOC_PATH'], 'bin'))
    !cd Tensorflow/models/research && protoc object_detection/protos/*.proto --python_out=. && copy object_dete
    !cd Tensorflow/models/research/slim && pip install -e .
```

In [ ]:
```
VERIFICATION_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'builders', 'model_b
# Verify Installation
!python {VERIFICATION_SCRIPT}
```

In [ ]:
```
!pip install tensorflow==2.9.0 --upgrade
```

In [ ]:
```
!pip3 install tensorflow-rocm
```

In [ ]:
```
!pip install pyyaml
```

In [ ]:
```
!pip uninstall protobuf matplotlib -y
!pip install protobuf matplotlib
```

In [ ]:

```
import object_detection
```

In [ ]:

```
!pip list
```

In [ ]:

```
if os.name =='posix':
    !wget {PRETRAINED_MODEL_URL}
    !mv {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
    !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}
if os.name == 'nt':
    wget.download(PRETRAINED_MODEL_URL)
    !move {PRETRAINED_MODEL_NAME+'.tar.gz'} {paths['PRETRAINED_MODEL_PATH']}
    !cd {paths['PRETRAINED_MODEL_PATH']} && tar -zxvf {PRETRAINED_MODEL_NAME+'.tar.gz'}
```

2. Create Label Map

In [8]:

```
labels = [{'name':'licence', 'id':1}]

with open(files['LABELMAP'], 'w') as f:
    for label in labels:
        f.write('item { \n')
        f.write('\tname:\'{}\'\n'.format(label['name']))
        f.write('\tid:{}\n'.format(label['id']))
        f.write('}\n')
```

3. Create TF records

In [ ]:

```
# OPTIONAL IF RUNNING ON COLAB
ARCHIVE_FILES = os.path.join(paths['IMAGE_PATH'], 'archive.tar.gz')
if os.path.exists(ARCHIVE_FILES):
  !tar -zxvf {ARCHIVE_FILES}
```

In [ ]:

```
if not os.path.exists(files['TF_RECORD_SCRIPT']):
    !git clone https://github.com/nicknochnack/GenerateTFRecord {paths['SCRIPTS_PATH']}
```

In [ ]:

```
!pip install pytz
```

In [ ]:

```
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'train')} -l {files['LABELMAP']} -o {
!python {files['TF_RECORD_SCRIPT']} -x {os.path.join(paths['IMAGE_PATH'], 'test')} -l {files['LABELMAP']} -o {o
```

4. Copy Model Config to Training Folder

In [ ]:

```
if os.name =='posix':
    !cp {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.join(
if os.name == 'nt':
    !copy {os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MODEL_NAME, 'pipeline.config')} {os.path.joi
```

5. Update Config For Transfer Learning

In [10]:

```
import tensorflow as tf
from object_detection.utils import config_util
from object_detection.protos import pipeline_pb2
from google.protobuf import text_format
```

In [11]:

```
config = config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])
```

In [ ]:

```
config
```

In [ ]:

```
pipeline_config = pipeline_pb2.TrainEvalPipelineConfig()
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "r") as f:
    proto_str = f.read()
    text_format.Merge(proto_str, pipeline_config)
```

In [ ]:

```
pipeline_config.model.ssd.num_classes = len(labels)
pipeline_config.train_config.batch_size = 4
pipeline_config.train_config.fine_tune_checkpoint = os.path.join(paths['PRETRAINED_MODEL_PATH'], PRETRAINED_MOD
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path= files['LABELMAP']
pipeline_config.train_input_reader.tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PATH'
pipeline_config.eval_input_reader[0].label_map_path = files['LABELMAP']
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[:] = [os.path.join(paths['ANNOTATION_PAT
```

In [ ]:

```
config_text = text_format.MessageToString(pipeline_config)
with tf.io.gfile.GFile(files['PIPELINE_CONFIG'], "wb") as f:
    f.write(config_text)
```

6. Train the model

In [ ]:
```
TRAINING_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'model_main_tf2.py')
```

In [ ]:
```
command = "python {} --model_dir={} --pipeline_config_path={} --num_train_steps=5000".format(TRAINING_SCRIPT, p
```

In [ ]:
```
print(command)
```

In [ ]:
```
#!pip uninstall pycocotools -y
!pip install pycocotools
```

In [ ]:
```
!pip list
```

In [ ]:
```
!{command}
```

7. Evaluate the Model

In [ ]:
```
command = "python {} --model_dir={} --pipeline_config_path={} --checkpoint_dir={}".format(TRAINING_SCRIPT, path
```

In [ ]:
```
print(command)
```

In [ ]:
```
!{command}
```

8. Load Train Model From Checkpoint

In [12]:
```
import os
import tensorflow as tf
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils
from object_detection.utils import config_util
```

In [ ]:
```
import sys
sys.path.append('path/to/Tensorflow/models/research')
```

In [13]:
```
from object_detection.builders import model_builder
```

In [14]:
```
# 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-8')).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
```

9. Detect from an Image

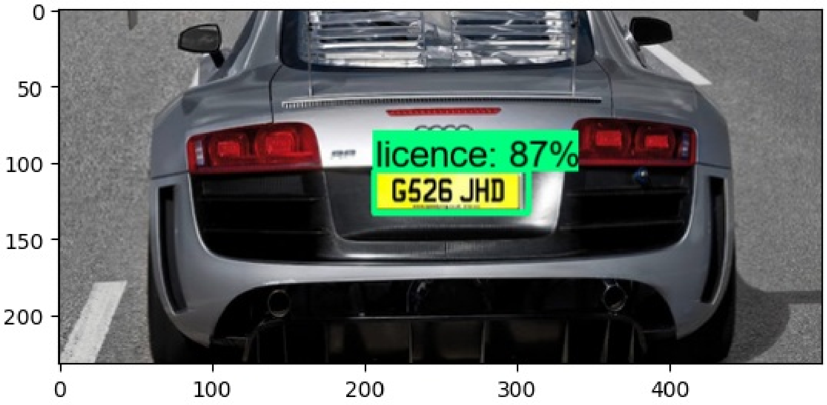
In [15]:
```
import cv2
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
```

In [16]:
```
category_index = label_map_util.create_category_index_from_labelmap(files['LABELMAP'])
```

In [17]:
```
IMAGE_PATH = os.path.join(paths['IMAGE_PATH'], 'test', 'Cars425.png')
```

In [23]:
```
img = cv2.imread(IMAGE_PATH)
image_np = np.array(img)

input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
detections = detect_fn(input_tensor)

num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
```

```
              for key, value in detections.items()}
detections['num_detections'] = num_detections

# detection_classes should be ints.
detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

label_id_offset = 1
image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
            image_np_with_detections,
            detections['detection_boxes'],
            detections['detection_classes']+label_id_offset,
            detections['detection_scores'],
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=5,
            min_score_thresh=.8,
            agnostic_mode=False)

plt.imshow(cv2.cvtColor(image_np_with_detections, cv2.COLOR_BGR2RGB))
plt.show()
```



Apply OCR to Detection

In [ ]:

```
!pip install easyocr
```

In [19]:

```
import easyocr
```

In [25]:

```
detection_threshold=0.7
```

In [26]:

```
image = image_np_with_detections #grabbing image
scores = list(filter(lambda x: x> detection_threshold, detections['detection_scores']))
boxes = detections['detection_boxes'][:len(scores)]
classes = detections['detection_classes'][:len(scores)]
```

In [27]:

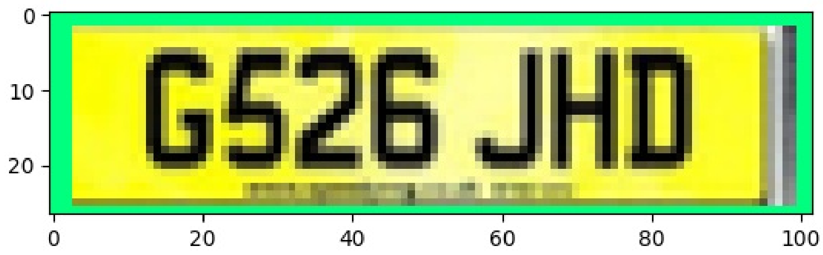
```
width = image.shape[1]
height = image.shape[0]
```

In [28]:

```
# Apply ROI filtering and OCR
for idx, box in enumerate(boxes):
    print(box)
    roi = box*[height, width, height, width]
    print(roi)
    region = image[int(roi[0]):int(roi[2]),int(roi[1]):int(roi[3])]
    reader = easyocr.Reader(['en'])
    ocr_result = reader.readtext(region)
    print(ocr_result)
    plt.imshow(cv2.cvtColor(region, cv2.COLOR_BGR2RGB))
```

```
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.
Downloading detection model, please wait. This may take several minutes depending upon your network connection.
[0.45900598 0.4139297  0.57607603 0.6165743 ]
[106.48938775 206.96485043 133.64963913 308.28714371]
Progress: |██████████████████████████████████████████████████| 100.0% Complete
Downloading recognition model, please wait. This may take several minutes depending upon your network connectio
n.
Progress: |██████████████████████████████████████████████████| 100.0% Complete
```

```
C:\Users\ADMIN\ANPR\anprsys\lib\site-packages\torchvision\models\_utils.py:252: UserWarning: Accessing the mode
l URLs via the internal dictionary of the module is deprecated since 0.13 and will be removed in 0.15. Please a
ccess them via the appropriate Weights Enum instead.
  warnings.warn(
C:\Users\ADMIN\ANPR\anprsys\lib\site-packages\torchvision\models\_utils.py:208: UserWarning: The parameter 'pre
trained' is deprecated since 0.13 and will be removed in 0.15, please use 'weights' instead.
  warnings.warn(

C:\Users\ADMIN\ANPR\anprsys\lib\site-packages\torchvision\models\_utils.py:223: UserWarning: Arguments other th
an a weight enum or `None` for 'weights' are deprecated since 0.13 and will be removed in 0.15. The current beh
avior is equivalent to passing `weights=None`.
  warnings.warn(msg)
[([[9, 3], [89, 3], [89, 23], [9, 23]], '6526 JHD', 0.7098373991313709)]
```



In [30]:
```python
for result in ocr_result:
    print(np.sum(np.subtract(result[0][2],result[0][1])))
    print(result[1])
```

```
20
6526 JHD
```

OCR Filtering

In [37]:
```python
region_threshold = 0.05
```

In [31]:
```python
def filter_text(region, ocr_result, region_threshold):
    rectangle_size = region.shape[0]*region.shape[1]

    plate = []
    for result in ocr_result:
        length = np.sum(np.subtract(result[0][1], result[0][0]))
        height = np.sum(np.subtract(result[0][2], result[0][1]))

        if length*height / rectangle_size > region_threshold:
            plate.append(result[1])
    return plate
```

In [32]:
```python
filter_text(region, ocr_result, region_threshold)
```

Out[32]:
```
['6526 JHD']
```

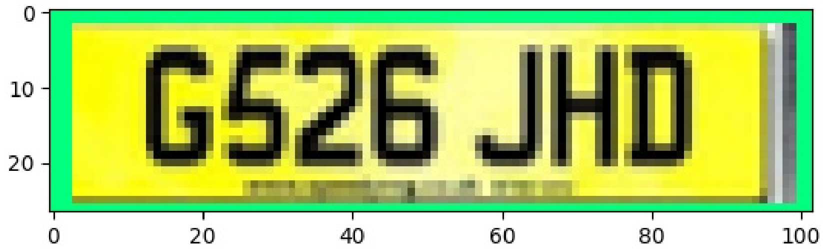
Bring it Together

In [33]:
```python
#region_threshold = 0.6
```

In [38]:
```python
def ocr_it(image, detections, detection_threshold, region_threshold):

    # Scores, boxes and classes above threhold
    scores = list(filter(lambda x: x> detection_threshold, detections['detection_scores']))
    boxes = detections['detection_boxes'][:len(scores)]
    classes = detections['detection_classes'][:len(scores)]

    # Full image dimensions
    width = image.shape[1]
    height = image.shape[0]

    # Apply ROI filtering and OCR
    for idx, box in enumerate(boxes):
        roi = box*[height, width, height, width]
        region = image[int(roi[0]):int(roi[2]),int(roi[1]):int(roi[3])]
        reader = easyocr.Reader(['en'])
        ocr_result = reader.readtext(region)

        text = filter_text(region, ocr_result, region_threshold)

        plt.imshow(cv2.cvtColor(region, cv2.COLOR_BGR2RGB))
        plt.show()
        print(text)
        return text, region
```

In [39]:
```python
text, region = ocr_it(image_np_with_detections, detections, detection_threshold, region_threshold)
```

```
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.
```



```
['6526 JHD']
```

In [40]:

```
text
```

Out[40]:

```
['6526 JHD']
```

10. Real Time Detections from your Webcam

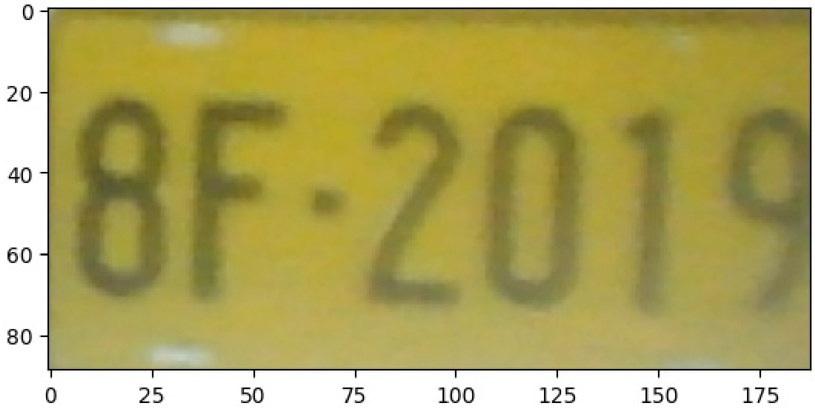
In [ ]:

```
!pip uninstall opencv-python-headless -y
```

In [41]:

```
cap = cv2.VideoCapture(0)
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

while cap.isOpened():
    ret, frame = cap.read()
    image_np = np.array(frame)

    input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
    detections = detect_fn(input_tensor)

    num_detections = int(detections.pop('num_detections'))
    detections = {key: value[0, :num_detections].numpy()
                  for key, value in detections.items()}
    detections['num_detections'] = num_detections

    # detection_classes should be ints.
    detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

    label_id_offset = 1
    image_np_with_detections = image_np.copy()

    viz_utils.visualize_boxes_and_labels_on_image_array(
                image_np_with_detections,
                detections['detection_boxes'],
                detections['detection_classes']+label_id_offset,
                detections['detection_scores'],
                category_index,
                use_normalized_coordinates=True,
                max_boxes_to_draw=5,
                min_score_thresh=.8,
                agnostic_mode=False)

    try:
        text, region = ocr_it(image_np_with_detections, detections, detection_threshold, region_threshold)
        save_results(text, region, 'realtimeresults.csv', 'Detection_Images')
    except:
        pass

    cv2.imshow('object detection',  cv2.resize(image_np_with_detections, (800, 600)))

    if cv2.waitKey(10) & 0xFF == ord('q'):
        cap.release()
        cv2.destroyAllWindows()
        break
```

```
CUDA not available - defaulting to CPU. Note: This module is much faster with a GPU.
```



```
['8F:201']
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In [41], line 10
      7 image_np = np.array(frame)
      9 input_tensor = tf.convert_to_tensor(np.expand_dims(image_np, 0), dtype=tf.float32)
---> 10 detections = detect_fn(input_tensor)
     12 num_detections = int(detections.pop('num_detections'))
     13 detections = {key: value[0, :num_detections].numpy()
     14               for key, value in detections.items()}

File ~\ANPR\anprsys\lib\site-packages\tensorflow\python\util\traceback_utils.py:150, in filter_traceback.<local
s>.error_handler(*args, **kwargs)
    148 filtered_tb = None
    149 try:
--> 150   return fn(*args, **kwargs)
    151 except Exception as e:
    152   filtered_tb = _process_traceback_frames(e.__traceback__)

File ~\ANPR\anprsys\lib\site-packages\tensorflow\python\eager\def_function.py:915, in Function.__call__(self, *
args, **kwds)
    912 compiler = "xla" if self._jit_compile else "nonXla"
    914 with OptionalXlaContext(self._jit_compile):
--> 915   result = self._call(*args, **kwds)
    917 new_tracing_count = self.experimental_get_tracing_count()
    918 without_tracing = (tracing_count == new_tracing_count)

File ~\ANPR\anprsys\lib\site-packages\tensorflow\python\eager\def_function.py:947, in Function._call(self, *arg
s, **kwds)
    944   self._lock.release()
    945   # In this case we have created variables on the first call, so we run the
    946   # defunned version which is guaranteed to never create variables.
--> 947   return self._stateless_fn(*args, **kwds)  # pylint: disable=not-callable
    948 elif self._stateful_fn is not None:
    949   # Release the lock early so that multiple threads can perform the call
    950   # in parallel.
    951   self._lock.release()

File ~\ANPR\anprsys\lib\site-packages\tensorflow\python\eager\function.py:2496, in Function.__call__(self, *arg
s, **kwargs)
   2493 with self._lock:
   2494   (graph_function,
   2495    filtered_flat_args) = self._maybe_define_function(args, kwargs)
-> 2496 return graph_function._call_flat(
   2497     filtered_flat_args, captured_inputs=graph_function.captured_inputs)

File ~\ANPR\anprsys\lib\site-packages\tensorflow\python\eager\function.py:1862, in ConcreteFunction._call_flat(
self, args, captured_inputs, cancellation_manager)
   1858 possible_gradient_type = gradients_util.PossibleTapeGradientTypes(args)
   1859 if (possible_gradient_type == gradients_util.POSSIBLE_GRADIENT_TYPES_NONE
   1860     and executing_eagerly):
   1861   # No tape is watching; skip to running the function.
-> 1862   return self._build_call_outputs(self._inference_function.call(
   1863       ctx, args, cancellation_manager=cancellation_manager))
   1864 forward_backward = self._select_forward_and_backward_functions(
   1865     args,
   1866     possible_gradient_type,
   1867     executing_eagerly)
   1868 forward_function, args_with_tangents = forward_backward.forward()

File ~\ANPR\anprsys\lib\site-packages\tensorflow\python\eager\function.py:499, in _EagerDefinedFunction.call(se
lf, ctx, args, cancellation_manager)
    497 with _InterpolateFunctionError(self):
    498   if cancellation_manager is None:
--> 499     outputs = execute.execute(
    500         str(self.signature.name),
    501         num_outputs=self._num_outputs,
    502         inputs=args,
    503         attrs=attrs,
    504         ctx=ctx)
    505   else:
    506     outputs = execute.execute_with_cancellation(
    507         str(self.signature.name),
    508         num_outputs=self._num_outputs,
   (...)
    511         ctx=ctx,
    512         cancellation_manager=cancellation_manager)

File ~\ANPR\anprsys\lib\site-packages\tensorflow\python\eager\execute.py:54, in quick_execute(op_name, num_outp
uts, inputs, attrs, ctx, name)
     52 try:
     53   ctx.ensure_initialized()
---> 54   tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
     55                                       inputs, attrs, num_outputs)
     56 except core._NotOkStatusException as e:
     57   if name is not None:

KeyboardInterrupt: 
```

Save Results

In [42]:
```
import csv
import uuid
```

In [45]:
```
'{}.jpg'.format(uuid.uuid1())
```

Out[45]:
```
'7c641c68-32b2-11ed-b85e-ec8eb51092a7.jpg'
```

In [46]:
```
def save_results(text, region, csv_filename, folder_path):
    img_name = '{}.jpg'.format(uuid.uuid1())

    cv2.imwrite(os.path.join(folder_path, img_name), region)

    with open(csv_filename, mode='a', newline='') as f:
        csv_writer = csv.writer(f, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)
        csv_writer.writerow([img_name, text])
```

In [47]:
```
text
```

Out[47]:
```
['8F:201']
```

In [48]:
```
save_results(text, region, 'detection_results.csv', 'Detection_Images')
```

10. Freezing the Graph

In [ ]:
```
FREEZE_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'exporter_main_v2.py ')
```

In [ ]:
```
command = "python {} --input_type=image_tensor --pipeline_config_path={} --trained_checkpoint_dir={} --output_d
```

In [ ]:
```
print(command)
```

In [ ]:
```
!{command}
```

11. Conversion to TFJS

In [ ]:
```
!pip install tensorflowjs
```

In [ ]:
```
command = "tensorflowjs_converter --input_format=tf_saved_model --output_node_names='detection_boxes,detection_
```

In [ ]:
```
print(command)
```

In [ ]:
```
!{command}
```

In [ ]:
```
# Test Code: https://github.com/nicknochnack/RealTimeSignLanguageDetectionwithTFJS
```

12. Conversion to TFLite

In [ ]:
```
TFLITE_SCRIPT = os.path.join(paths['APIMODEL_PATH'], 'research', 'object_detection', 'export_tflite_graph_tf2.p
```

In [ ]:
```
command = "python {} --pipeline_config_path={} --trained_checkpoint_dir={} --output_directory={}".format(TFLITE_
```

In [ ]:
```
print(command)
```

In [ ]:
```
!{command}
```

In [ ]:
```
FROZEN_TFLITE_PATH = os.path.join(paths['TFLITE_PATH'], 'saved_model')
TFLITE_MODEL = os.path.join(paths['TFLITE_PATH'], 'saved_model', 'detect.tflite')
```

In [ ]:
```
command = "tflite_convert \
--saved_model_dir={} \
--output_file={} \
--input_shapes=1,300,300,3 \
--input_arrays=normalized_input_image_tensor \
--output_arrays='TFLite_Detection_PostProcess','TFLite_Detection_PostProcess:1','TFLite_Detection_PostProcess:2
--inference_type=FLOAT \
--allow_custom_ops".format(FROZEN_TFLITE_PATH, TFLITE_MODEL, )
```

In [ ]:
```
print(command)
```

In [ ]:
```
!{command}
```

13. Zip and Export Models

In [ ]:

```
!tar -czf models.tar.gz {paths['CHECKPOINT_PATH']}
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js