```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg # image related ops
import numpy as np
import cv2 # opencv lib

img_path="/content/car.jpeg"

car1 = mpimg.imread(img_path)

car1.shape

(168, 300, 3)

car1[167].shape

(300, 3)

plt.imshow(car1)

<matplotlib.image.AxesImage at 0x7f8e7fff1df0>
```


```
car1[0][0]

array([237, 237, 237], dtype=uint8)

car1_cv2 = cv2.imread(img_path)

car1_cv2

array([[[237, 237, 237],
        [237, 237, 237],
        [237, 237, 237],
        ...,
        [237, 237, 237],
        [237, 237, 237],
```

```
        [237, 237, 237]],

       [[237, 237, 237],
        [237, 237, 237],
        [237, 237, 237],
        ...,
        [237, 237, 237],
        [237, 237, 237],
        [237, 237, 237]],

       [[237, 237, 237],
        [237, 237, 237],
        [237, 237, 237],
        ...,
        [237, 237, 237],
        [237, 237, 237],
        [237, 237, 237]],

       ...,

       [[216, 216, 216],
        [216, 216, 216],
        [216, 216, 216],
        ...,
        [211, 211, 211],
        [211, 211, 211],
        [211, 211, 211]],

       [[217, 217, 217],
        [217, 217, 217],
        [217, 217, 217],
        ...,
        [213, 213, 213],
        [213, 213, 213],
        [213, 213, 213]],

       [[217, 217, 217],
        [217, 217, 217],
        [217, 217, 217],
        ...,
        [214, 214, 214],
        [214, 214, 214],
        [214, 214, 214]]], dtype=uint8)
```

```
from google.colab.patches import cv2_imshow
cv2_imshow(car1_cv2)
```

```
plt.imshow(car1_cv2) # cv2 reads images as BGR and in matplotlib reads as RGB
```

```
<matplotlib.image.AxesImage at 0x7f8e7ff6a7f0>
```


```
car1_cv2_BGR_RGB = cv2.cvtColor(car1_cv2, cv2.COLOR_BGR2RGB)
plt.imshow(car1_cv2_BGR_RGB)
```

```
<matplotlib.image.AxesImage at 0x7f8e7ff04850>
```

car1_cv2_BGR_RGB

```
array([[[237, 237, 237],
        [237, 237, 237],
        [237, 237, 237],
        ...,
        [237, 237, 237],
        [237, 237, 237],
        [237, 237, 237]],

       [[237, 237, 237],
        [237, 237, 237],
        [237, 237, 237],
        ...,
        [237, 237, 237],
        [237, 237, 237],
        [237, 237, 237]],

       [[237, 237, 237],
        [237, 237, 237],
        [237, 237, 237],
        ...,
        [237, 237, 237],
        [237, 237, 237],
        [237, 237, 237]],

       ...,

       [[216, 216, 216],
        [216, 216, 216],
        [216, 216, 216],
        ...,
```

```
        [211, 211, 211],
        [211, 211, 211],
        [211, 211, 211]],

       [[217, 217, 217],
        [217, 217, 217],
        [217, 217, 217],
        ...,
        [213, 213, 213],
        [213, 213, 213],
        [213, 213, 213]],

       [[217, 217, 217],
        [217, 217, 217],
        [217, 217, 217],
        ...,
        [214, 214, 214],
        [214, 214, 214],
        [214, 214, 214]]], dtype=uint8)
```

```
car1_cv2_BGR_RGB.shape
```

```
(168, 300, 3)
```

```
car1_cv2_BGR_GRAY = cv2.cvtColor(car1_cv2, cv2.COLOR_BGR2GRAY)
plt.imshow(car1_cv2_BGR_GRAY, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8ee06e6640>
```


```
car1_cv2_BGR_GRAY
```

```
array([[237, 237, 237, ..., 237, 237, 237],
       [237, 237, 237, ..., 237, 237, 237],
```

```
       [237, 237, 237, ..., 237, 237, 237],
       ...,
       [216, 216, 216, ..., 211, 211, 211],
       [217, 217, 217, ..., 213, 213, 213],
       [217, 217, 217, ..., 214, 214, 214]], dtype=uint8)
```

```
car1_cv2_BGR_GRAY.shape
```

```
(168, 300)
```

```
car1_cv2_BGR_GRAY.min(), car1_cv2_BGR_GRAY.max()
```

```
(0, 255)
```

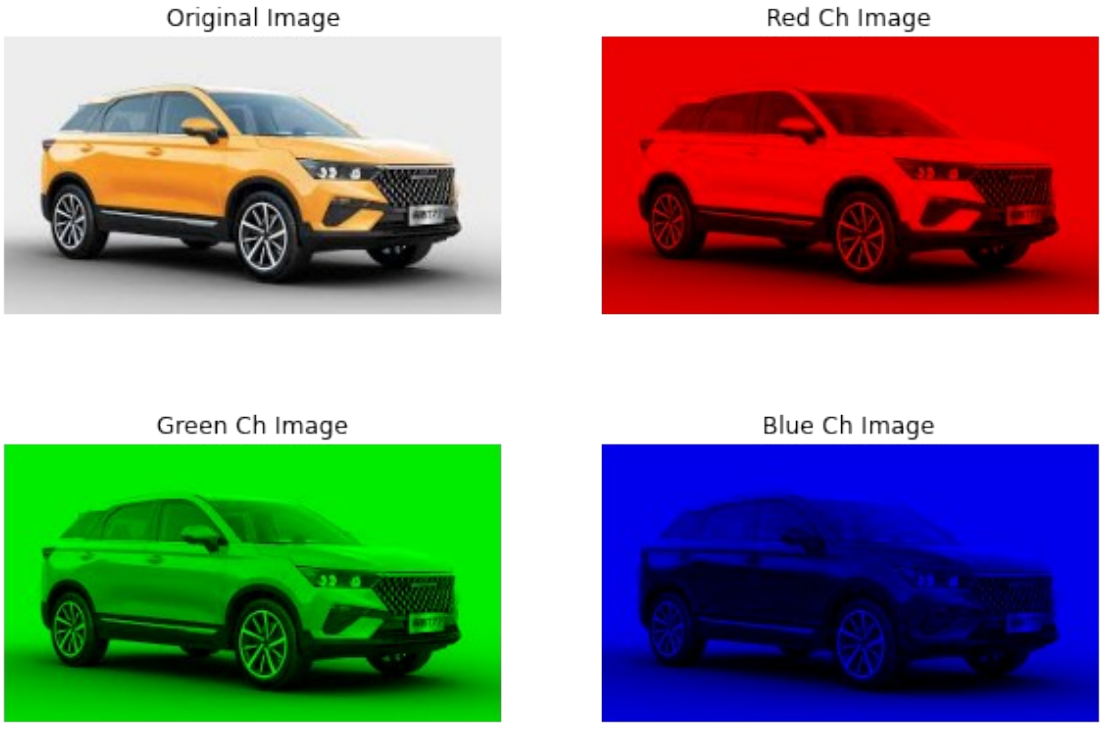
```
cv2.split(car1_cv2)
```

```
(array([[237, 237, 237, ..., 237, 237, 237],
        [237, 237, 237, ..., 237, 237, 237],
        [237, 237, 237, ..., 237, 237, 237],
        ...,
        [216, 216, 216, ..., 211, 211, 211],
        [217, 217, 217, ..., 213, 213, 213],
        [217, 217, 217, ..., 214, 214, 214]], dtype=uint8),
 array([[237, 237, 237, ..., 237, 237, 237],
        [237, 237, 237, ..., 237, 237, 237],
        [237, 237, 237, ..., 237, 237, 237],
        ...,
        [216, 216, 216, ..., 211, 211, 211],
        [217, 217, 217, ..., 213, 213, 213],
        [217, 217, 217, ..., 214, 214, 214]], dtype=uint8),
 array([[237, 237, 237, ..., 237, 237, 237],
        [237, 237, 237, ..., 237, 237, 237],
        [237, 237, 237, ..., 237, 237, 237],
        ...,
        [216, 216, 216, ..., 211, 211, 211],
        [217, 217, 217, ..., 213, 213, 213],
        [217, 217, 217, ..., 214, 214, 214]], dtype=uint8))
```

understanding composition of colored images

```
def viusalize_RGB_channel(imgArray=None, figsize=(10,7)):
  # splitting the RGB components
  B, G, R = cv2.split(imgArray)

  # create zero matrix of shape of image
  Z = np.zeros(B.shape, dtype=B.dtype) # can use any channel

  # init subplots
  fig, ax = plt.subplots(2,2, figsize=figsize)
```

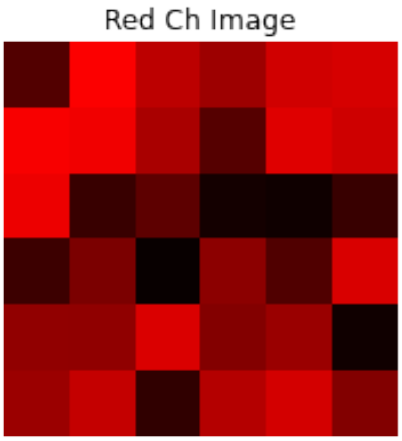
```
  # plotting the actual image and RGB images
  [axi.set_axis_off() for axi in ax.ravel()]

  ax[0,0].set_title("Original Image")
  # ax[0,0].set_axis_off()
  ax[0,0].imshow(cv2.merge((R,G,B)))

  ax[0,1].set_title("Red Ch Image")
  ax[0,1].imshow(cv2.merge((R,Z,Z)))

  ax[1,0].set_title("Green Ch Image")
  ax[1,0].imshow(cv2.merge((Z,G,Z)))

  ax[1,1].set_title("Blue Ch Image")
  ax[1,1].imshow(cv2.merge((Z,Z,B)))

viusalize_RGB_channel(imgArray=car1_cv2)
```

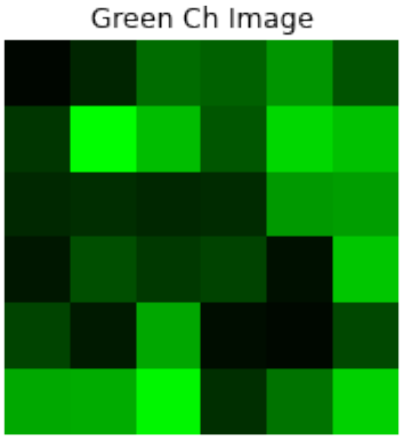


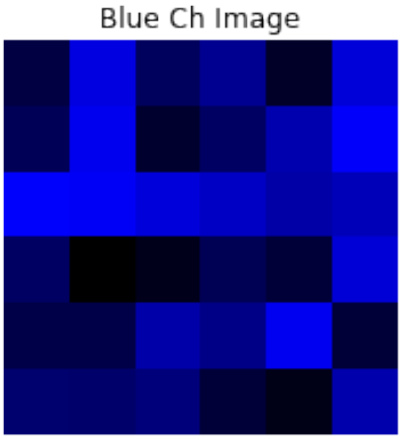

```
random_colored_img = np.random.randint(0, 255, (6,6,3))

random_colored_img
```

```
array([[[ 66,   6,  82],
        [224,  37, 252],
        [ 93, 108, 188],
        [144,  97, 157],
        [ 40, 149, 208],
```

```
         [216,  83, 214]],

        [[ 86,  54, 248],
         [239, 254, 241],
         [ 46, 189, 169],
         [ 98,  86,  85],
         [175, 215, 223],
         [251, 193, 206]],

        [[253,  40, 237],
         [247,  46,  56],
         [218,  39,  93],
         [196,  43,  20],
         [166, 153,  15],
         [184, 158,  56]],

        [[ 98,  23,  60],
         [  0,  79, 123],
         [ 26,  57,   8],
         [ 84,  67, 140],
         [ 57,  16,  80],
         [214, 198, 217]],

        [[ 72,  68, 145],
         [ 73,  26, 144],
         [169, 167, 218],
         [134,  13, 131],
         [240,   8, 154],
         [ 56,  72,  16]],

        [[110, 168, 155],
         [107, 172, 196],
         [123, 246,  48],
         [ 56,  47, 180],
         [ 23, 115, 211],
         [172, 209, 130]]])

viusalize_RGB_channel(imgArray=random_colored_img)
```







understanding filters

```
sobel = np.array([[ 1, 0,-1],
                  [ 2, 0,-2],
                  [ 1, 0,-1]])

print("highlighting Vertical edges:\n", sobel)
```

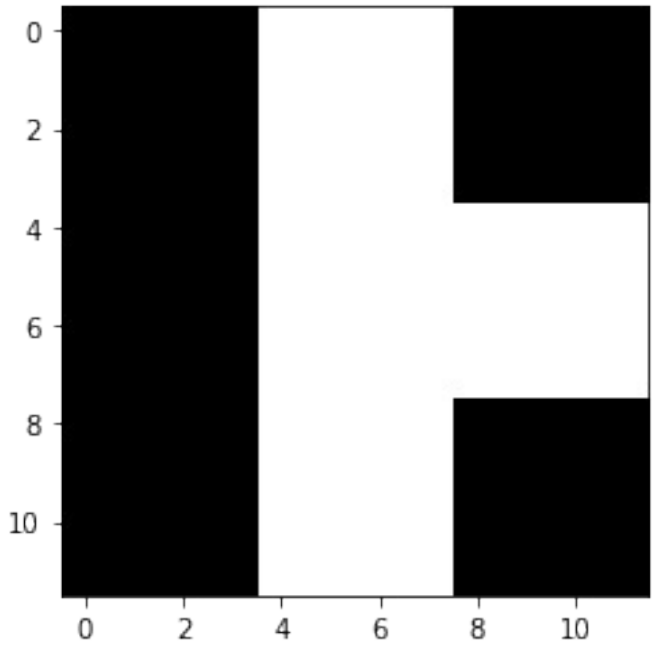
```
highlighting Vertical edges:
 [[ 1  0 -1]
 [ 2  0 -2]
 [ 1  0 -1]]
```

```
print("highlighting Horizontal edges:\n", sobel.T)
```

```
highlighting Horizontal edges:
 [[ 1  2  1]
 [ 0  0  0]
 [-1 -2 -1]]
```

```
example1 = [
    [0,0,0,0,255,255,255,255,0,0,0,0],
```

```
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,255,255,255,255],
    [0,0,0,0,255,255,255,255,255,255,255,255],
    [0,0,0,0,255,255,255,255,255,255,255,255],
    [0,0,0,0,255,255,255,255,255,255,255,255],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
            ]

example1 = np.array(example1)

example1

array([[  0,   0,   0,   0, 255, 255, 255, 255,   0,   0,   0,   0],
       [  0,   0,   0,   0, 255, 255, 255, 255,   0,   0,   0,   0],
       [  0,   0,   0,   0, 255, 255, 255, 255,   0,   0,   0,   0],
       [  0,   0,   0,   0, 255, 255, 255, 255,   0,   0,   0,   0],
       [  0,   0,   0,   0, 255, 255, 255, 255, 255, 255, 255, 255],
       [  0,   0,   0,   0, 255, 255, 255, 255, 255, 255, 255, 255],
       [  0,   0,   0,   0, 255, 255, 255, 255, 255, 255, 255, 255],
       [  0,   0,   0,   0, 255, 255, 255, 255, 255, 255, 255, 255],
       [  0,   0,   0,   0, 255, 255, 255, 255,   0,   0,   0,   0],
       [  0,   0,   0,   0, 255, 255, 255, 255,   0,   0,   0,   0],
       [  0,   0,   0,   0, 255, 255, 255, 255,   0,   0,   0,   0],
       [  0,   0,   0,   0, 255, 255, 255, 255,   0,   0,   0,   0]])

plt.imshow(example1, cmap="gray")

<matplotlib.image.AxesImage at 0x7f8e7fc8fc10>
```




```
def simple_conv(imgFilter=None, picture=None):
  # extract the shape of the image
  p_row, p_col = picture.shape

  k = imgFilter.shape[0] # k =3

  temp = list()

  stride = 1

  # resulant image size
  final_cols = (p_col - k)//stride + 1
  final_rows = (p_row - k)//stride + 1

  # take vertically down stride across row by row
  for v_stride in range(final_rows):
    # take horizontal right stride across col by col
    for h_stride in range(final_cols):
      target_area_of_pic = picture[v_stride: v_stride + k, h_stride:
h_stride + k]
      z = sum(sum(imgFilter * target_area_of_pic))
      temp.append(z)

  resulant_image = np.array(temp).reshape(final_rows, final_cols)
  return resulant_image

k = 3
v_stride = 0
h_stride = 0 + 1 + 1
```

```
target_area = example1[v_stride: v_stride + k, h_stride: h_stride + k]
target_area
```

```
array([[  0,   0, 255],
       [  0,   0, 255],
       [  0,   0, 255]])
```

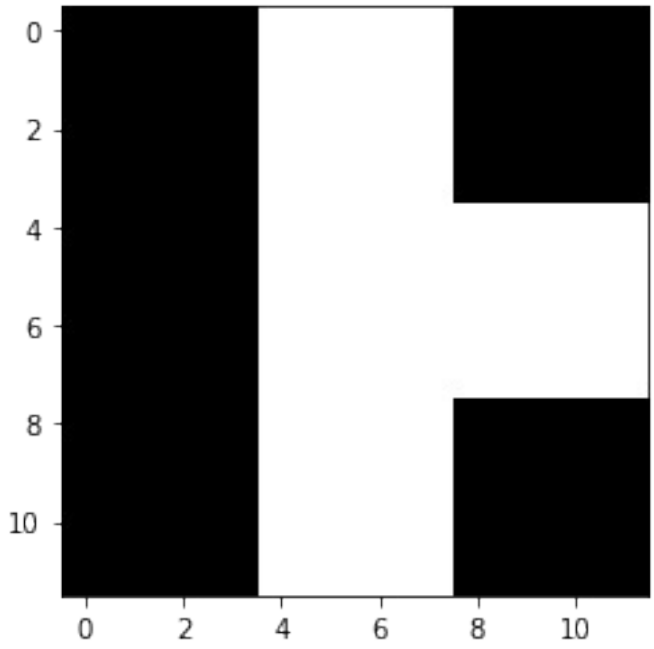
```
result = simple_conv(imgFilter=sobel, picture=example1)
```

```
result
```

```
array([[    0,     0, -1020, -1020,     0,     0,  1020,  1020,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,  1020,  1020,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,   765,   765,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,   255,   255,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,     0,     0,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,     0,     0,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,   255,   255,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,   765,   765,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,  1020,  1020,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,  1020,  1020,     0,
            0]])
```

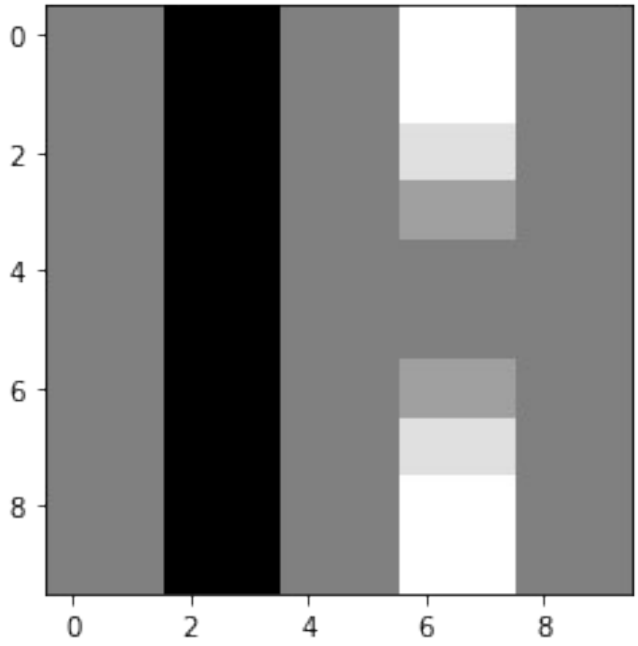
```
plt.imshow(example1, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7f8e9b50>
```



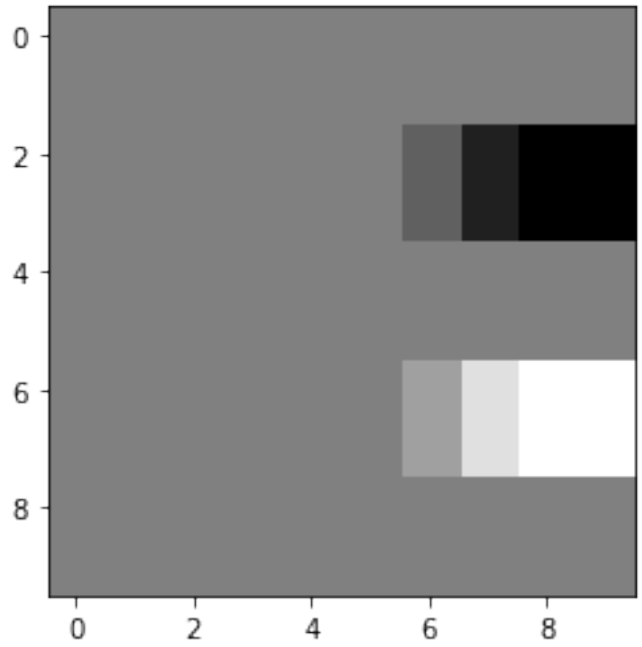

```
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7f054c40>
```



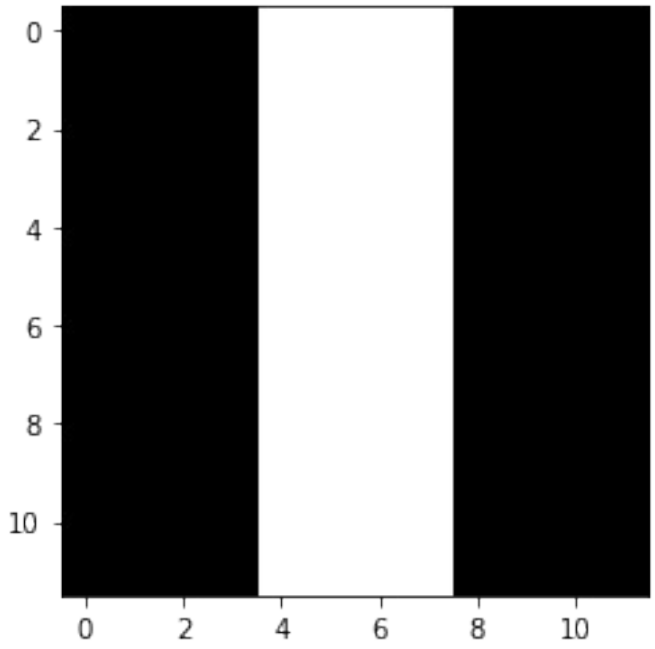

```
result = simple_conv(imgFilter=sobel.T, picture=example1)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7f023bb0>
```



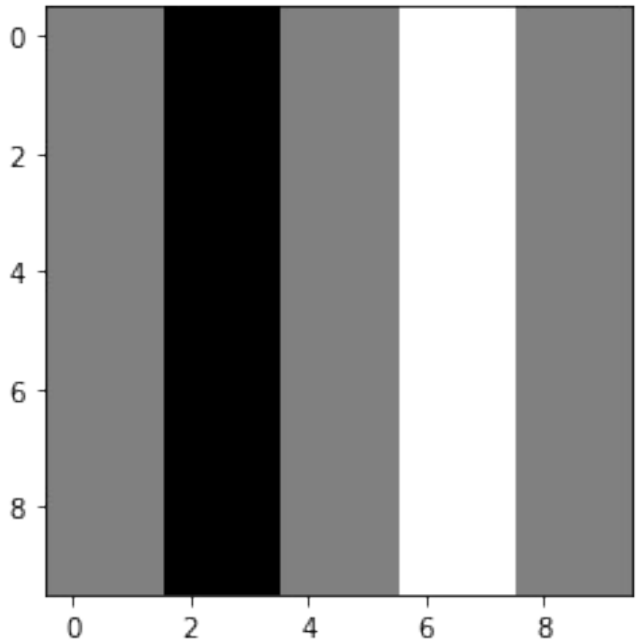

```
example2 = [
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
            ]

example2 = np.array(example2)
plt.imshow(example2, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7ef8ad30>
```



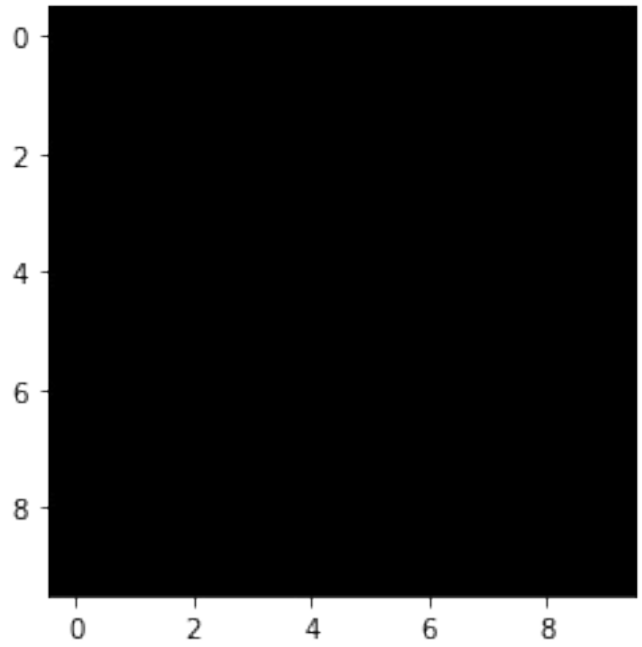

```
result = simple_conv(imgFilter=sobel, picture=example2)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7ef4cd30>
```



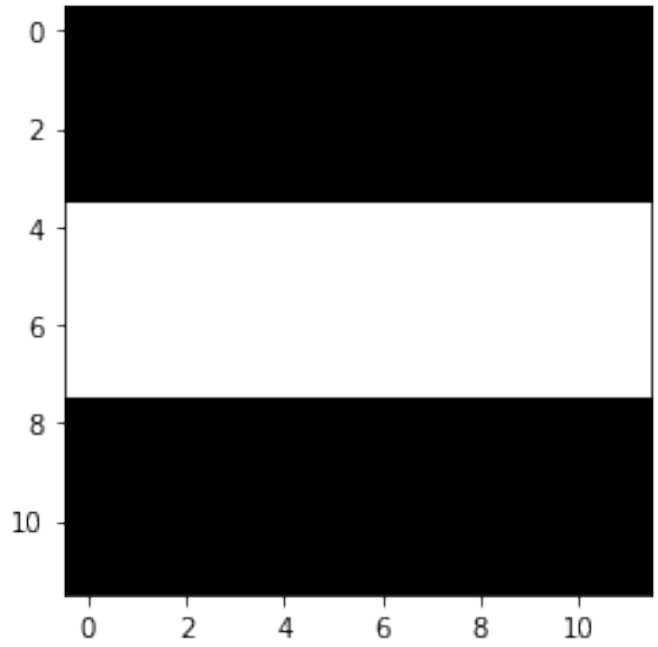

```
result = simple_conv(imgFilter=sobel.T, picture=example2)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7fa795e0>
```



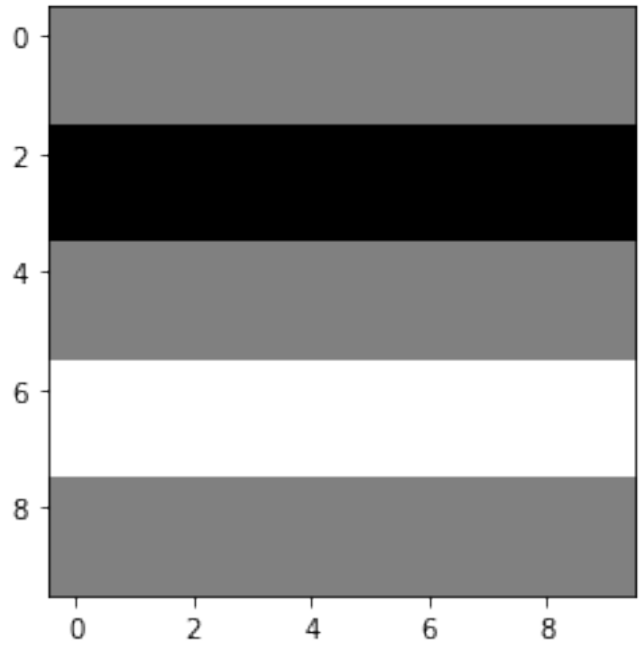

```
example2_T = np.array(example2.T)
plt.imshow(example2_T, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7ef1e0d0>
```



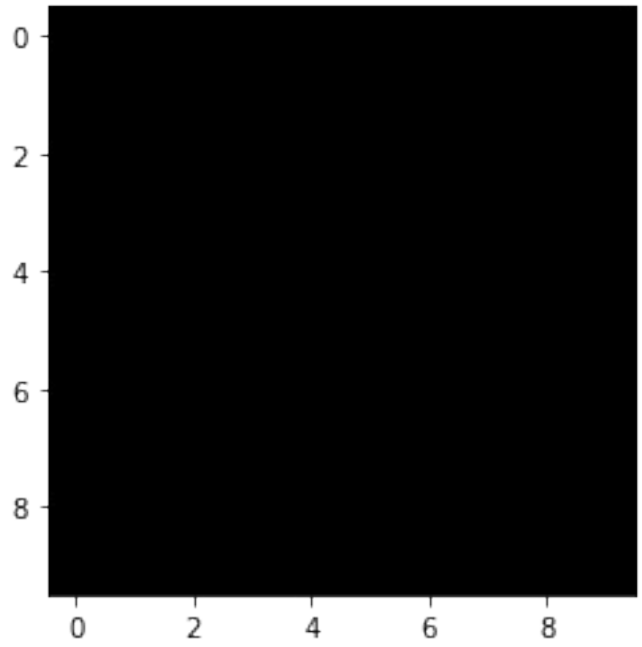

```
result = simple_conv(imgFilter=sobel.T, picture=example2_T)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7eee1eb0>
```



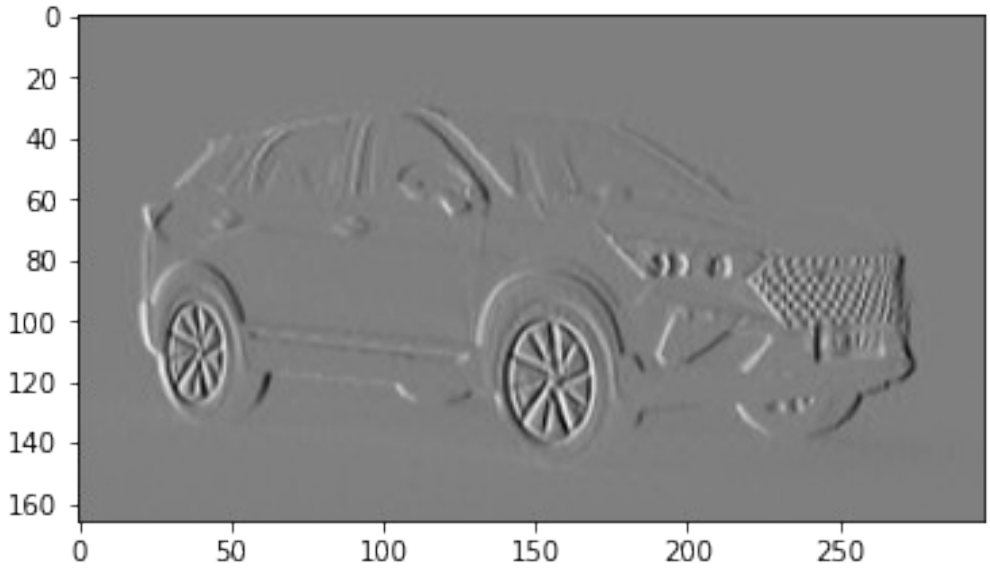

```
result = simple_conv(imgFilter=sobel, picture=example2_T)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8ee06d5a60>
```



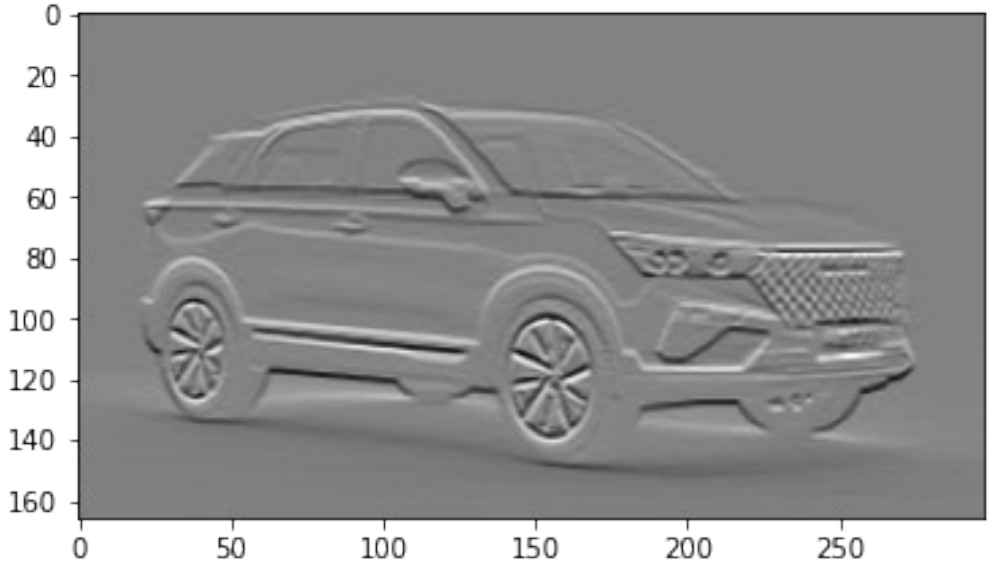

```
result = simple_conv(imgFilter=sobel, picture=car1_cv2_BGR_GRAY)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7ecf82e0>
```



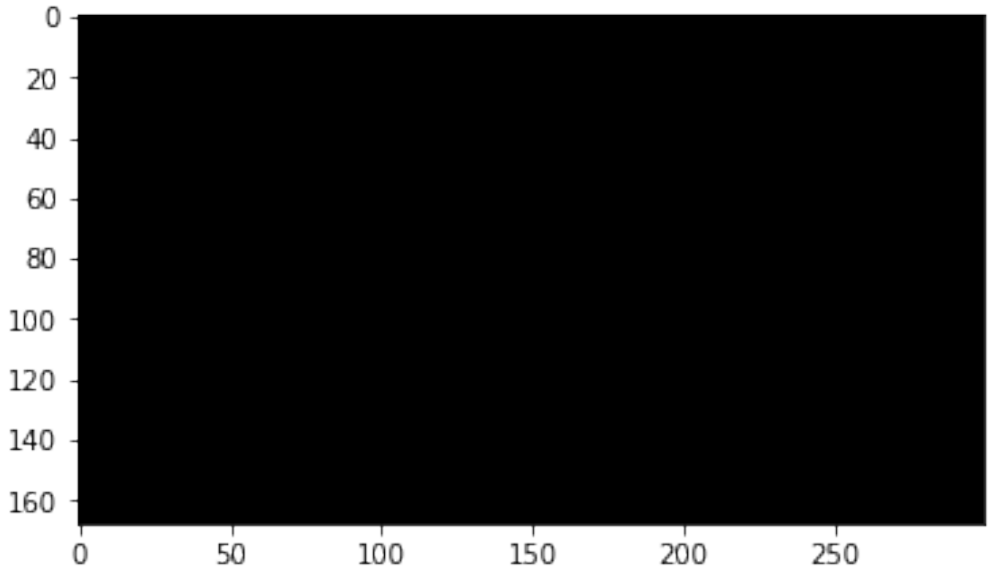

```
result = simple_conv(imgFilter=sobel.T, picture=car1_cv2_BGR_GRAY)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7ee4aeb0>
```



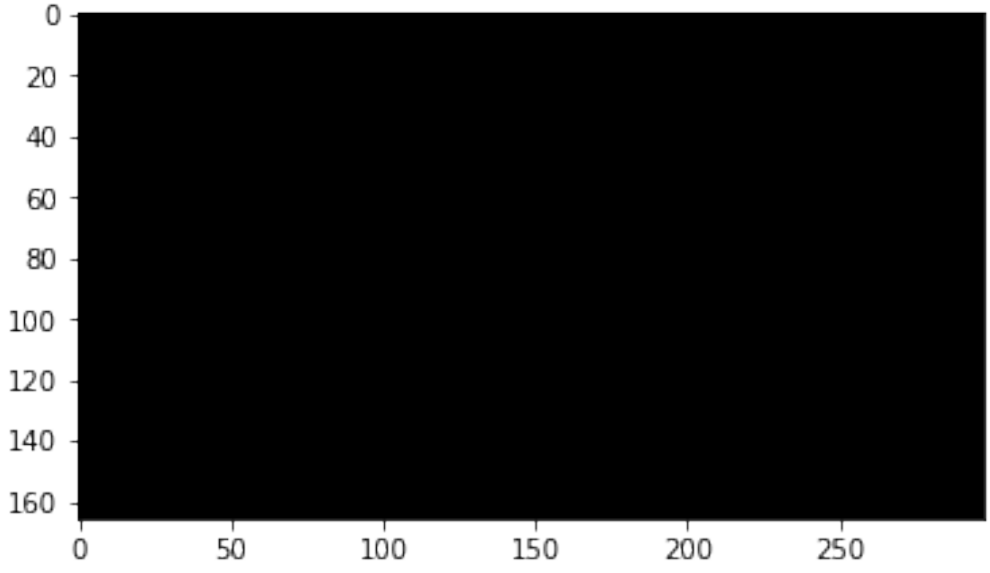

```
nothing = np.zeros(car1_cv2_BGR_GRAY.shape)
plt.imshow(nothing, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8ee0690cd0>
```



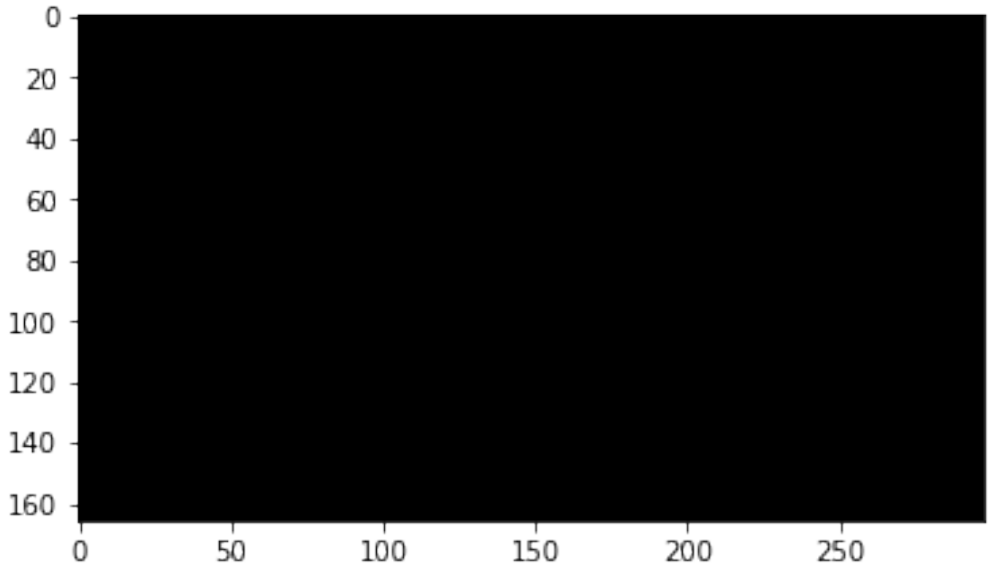

```
result = simple_conv(imgFilter=sobel.T, picture=nothing)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8ee0579cd0>
```



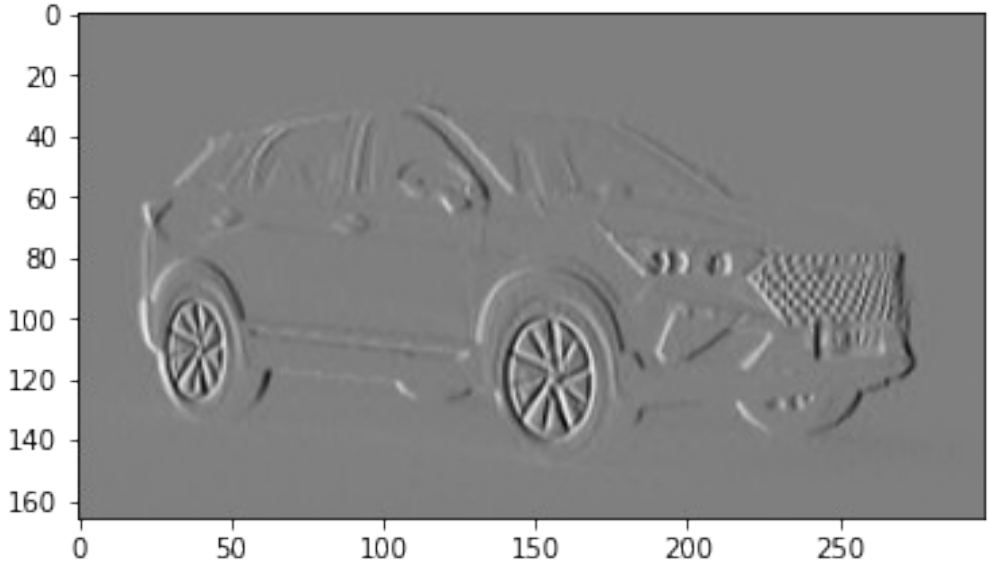

```
result = simple_conv(imgFilter=sobel, picture=nothing)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8ee023cbb0>
```




```
result = simple_conv(imgFilter=sobel, picture=car1_cv2_BGR_GRAY)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8ee0117af0>
```

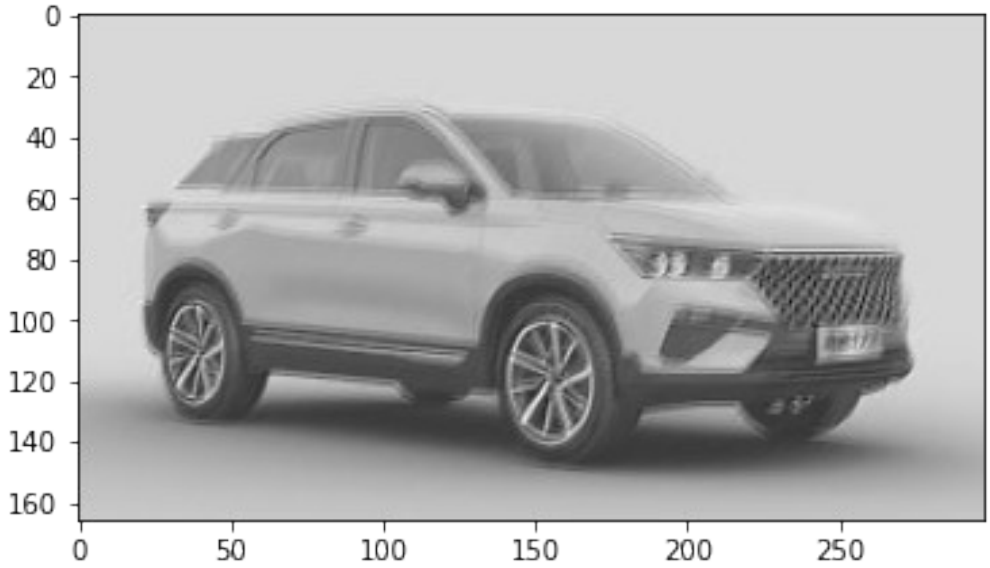



```
random_f = np.random.randn(3,3)
random_f
```

```
array([[-0.54436561,  1.91978522,  1.88748458],
       [ 0.50419995,  0.64010863, -1.8611337 ],
       [ 1.00152367,  1.19797409, -0.02787324]])
```

```
result = simple_conv(imgFilter=random_f, picture=car1_cv2_BGR_GRAY)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7ed66bb0>
```




```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import cv2

def read_img(path, grayscale=True):
  img = cv2.imread(path)
  if grayscale:
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    plt.imshow(img,cmap="gray")
    return img
  from google.colab.patches import cv2_imshow
  cv2_imshow(img)
  return img

#greyscale image
car = read_img(img_path)
```


```
#color image
color_car = read_img(img_path, grayscale=False)
```

```
# (1,r,c,1)
row,col = car.shape
car = car.reshape(1,row,col,1) # grayscale
# color_car = color_car.reshape(1,row,col,3) # colored
car.shape
```

```
(1, 168, 300, 1)
```

```
CONV_LAYER = [tf.keras.layers.Conv2D(filters=1,
                                     kernel_size=(3,3),
                                     strides=(1,1),
                                     input_shape=car.shape[1:])]
```

```
conv_model = tf.keras.Sequential(CONV_LAYER)
conv_model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 166, 298, 1)       10

=================================================================
Total params: 10
Trainable params: 10
Non-trainable params: 0
_________________________________________________________________
```

```
out = conv_model.predict(car)
```

```
1/1 [==============================] - 0s 54ms/step
```

```
out.shape
```

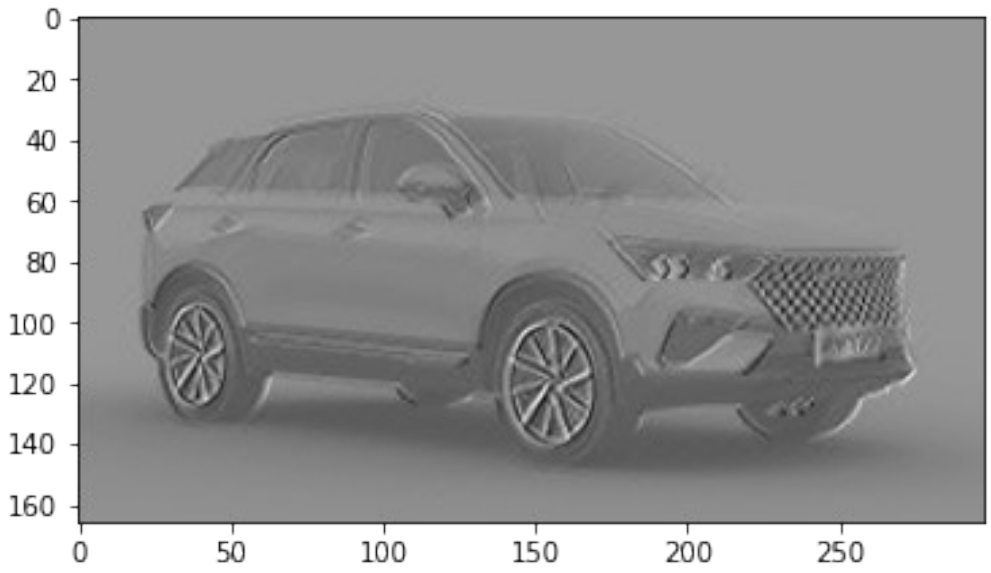
```
(1, 166, 298, 1)
```

```
row, col = out.shape[1:-1]
```

```
reshape_out = out.reshape(row, col)
```

```
plt.imshow(reshape_out, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f8e7ecd4250>
```




```
# (1,r,c,1)
row,col,depth = color_car.shape
# car = car.reshape(1,row,col,1) # grayscale
```

```
color_car = color_car.reshape(1,row,col,depth) # colored
color_car.shape
```

```
(1, 168, 300, 3)
```

```
CONV_LAYER = [tf.keras.layers.Conv2D(filters=1,
                                     kernel_size=(3,3),
                                     strides=(1,1),
                                     input_shape=color_car.shape[1:])]

conv_model = tf.keras.Sequential(CONV_LAYER)
conv_model.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_5 (Conv2D)           (None, 166, 298, 1)       28

=================================================================
Total params: 28
Trainable params: 28
Non-trainable params: 0
_________________________________________________________________
```

```
out = conv_model.predict(color_car)
```

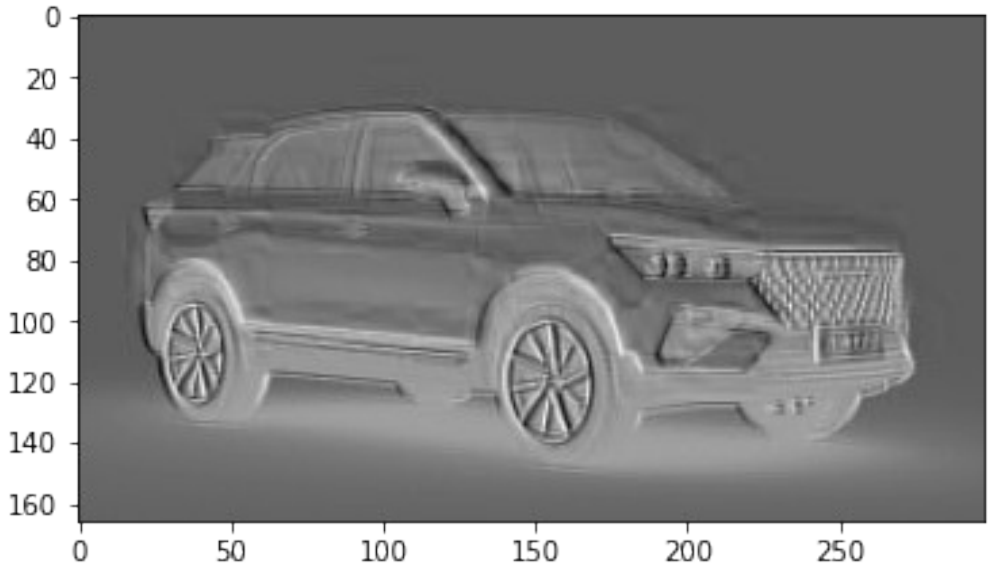
```
WARNING:tensorflow:5 out of the last 5 calls to <function
Model.make_predict_function.<locals>.predict_function at
0x7f8e7eded700> triggered tf.function retracing. Tracing is expensive
and the excessive number of tracings could be due to (1) creating
@tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function
has reduce_retracing=True option that can avoid unnecessary retracing.
For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.

1/1 [==============================] - 0s 69ms/step
```

```
out.shape
```

```
(1, 166, 298, 1)
```

```
row, col = out.shape[1:-1]
reshape_out = out.reshape(row, col)

plt.imshow(reshape_out, cmap="gray")
```

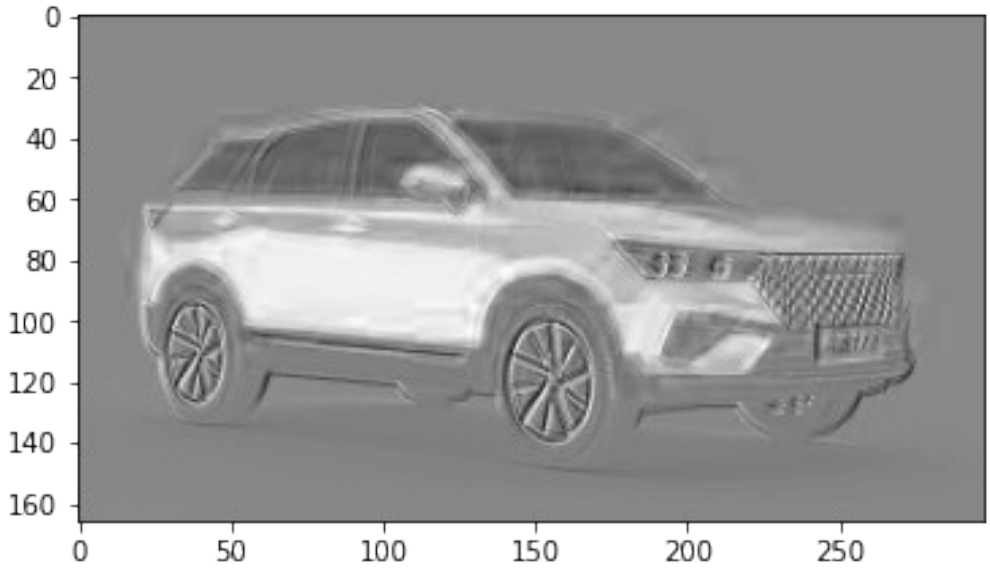
```
<matplotlib.image.AxesImage at 0x7f8e7ec43640>
```




```
def reshaping_in(img, grayscale=True):
  if grayscale:
    row,col = img.shape
    img = img.reshape(1,row,col,1) # grayscale
    return img
  row,col,depth = img.shape
  color_img = img.reshape(1,row,col,depth) # colored
  return color_img

def get_conv_model(filters=1, filter_size=(3,3), strides=(1,1),
input_shape=None, padding="valid"):
  CONV_LAYER = [tf.keras.layers.Conv2D(filters=filters,
                                       kernel_size=filter_size,
                                       strides=(1,1),
                                       input_shape=input_shape,
                                       padding=padding)]

  conv_model = tf.keras.Sequential(CONV_LAYER)
  conv_model.summary()
  return conv_model

def apply_conv_model_and_visualize(img, conv_model):
  try:
    out = conv_model.predict(img)
    print(out.shape)
    row, col, depth = out.shape[1:]
    reshape_out = out.reshape(row, col, depth)

    for d in range(depth):
      plt.imshow(reshape_out[:,:,d], cmap="gray")
      plt.show()
```

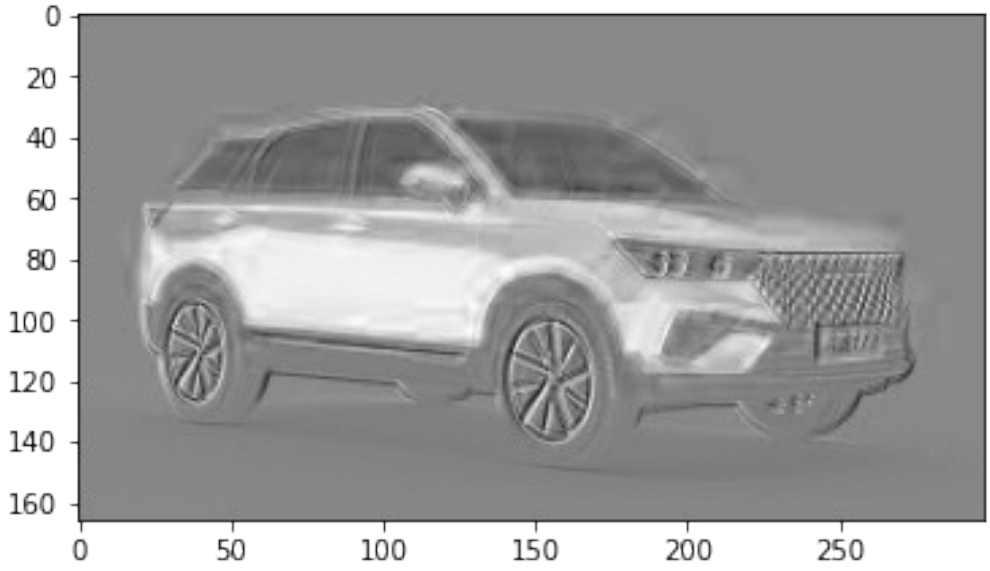
```
  except Exception as e:
    raise e

img = read_img(img_path, grayscale=False)
input_img = reshaping_in(img, grayscale=False)
model = get_conv_model(filters=1, filter_size=(3,3), strides=(1,1),
input_shape=input_img.shape[1:])
apply_conv_model_and_visualize(input_img, model)
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_6 (Conv2D)           (None, 166, 298, 1)       28

=================================================================
Total params: 28
Trainable params: 28
Non-trainable params: 0
_________________________________________________________________

WARNING:tensorflow:6 out of the last 6 calls to <function
Model.make_predict_function.<locals>.predict_function at
0x7f8e7ec5de50> triggered tf.function retracing. Tracing is expensive
and the excessive number of tracings could be due to (1) creating
@tf.function repeatedly in a loop, (2) passing tensors with different
shapes, (3) passing Python objects instead of tensors. For (1), please
define your @tf.function outside of the loop. For (2), @tf.function
has reduce_retracing=True option that can avoid unnecessary retracing.
For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.

1/1 [==============================] - 0s 50ms/step
(1, 166, 298, 1)
```



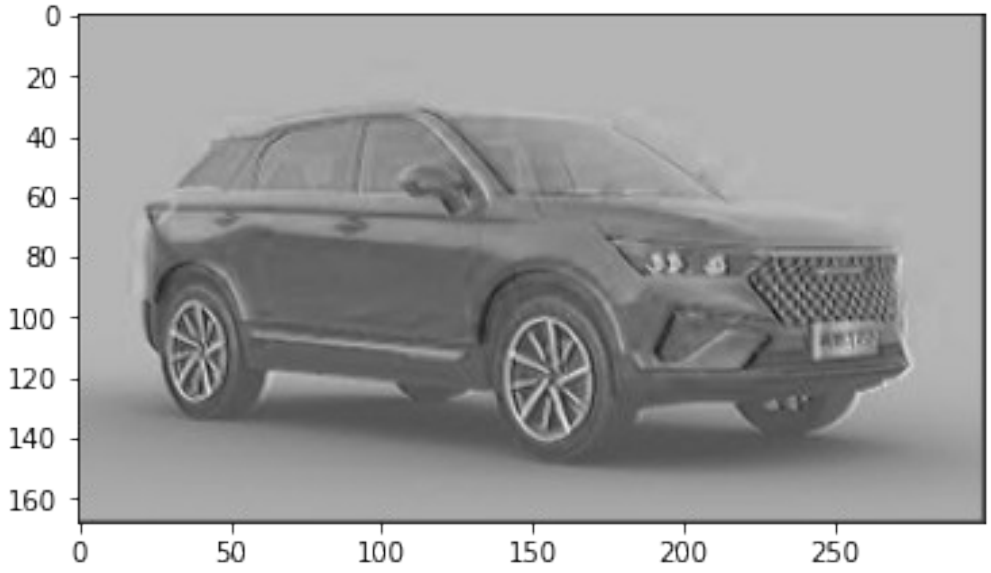

```
img = read_img(img_path, grayscale=False)
input_img = reshaping_in(img, grayscale=False)
model = get_conv_model(filters=10, filter_size=(3,3), strides=(1,1),
input_shape=input_img.shape[1:])
apply_conv_model_and_visualize(input_img, model)
```

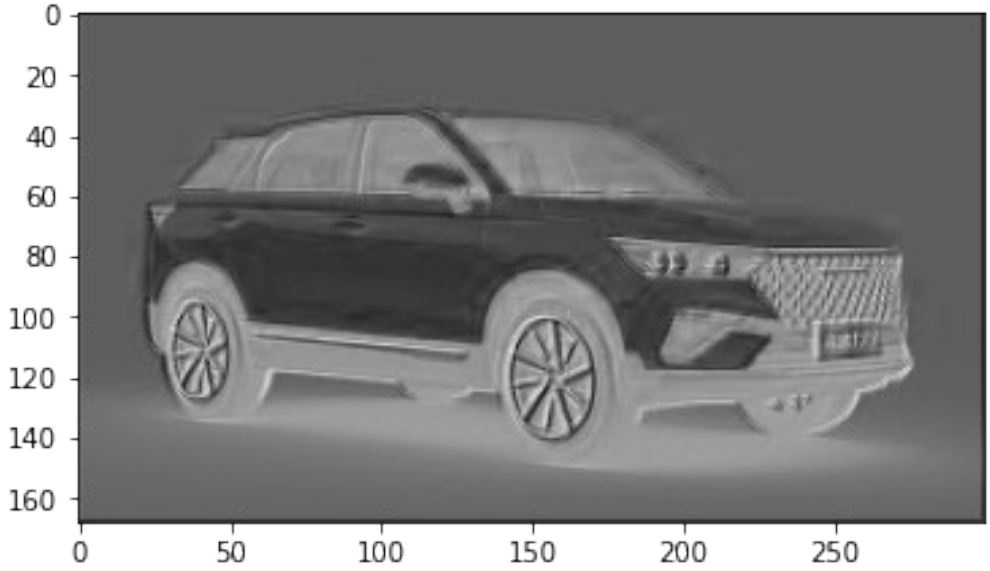
```
1/1 [==============================] - 0s 27ms/step
(1, 166, 298, 1)
```

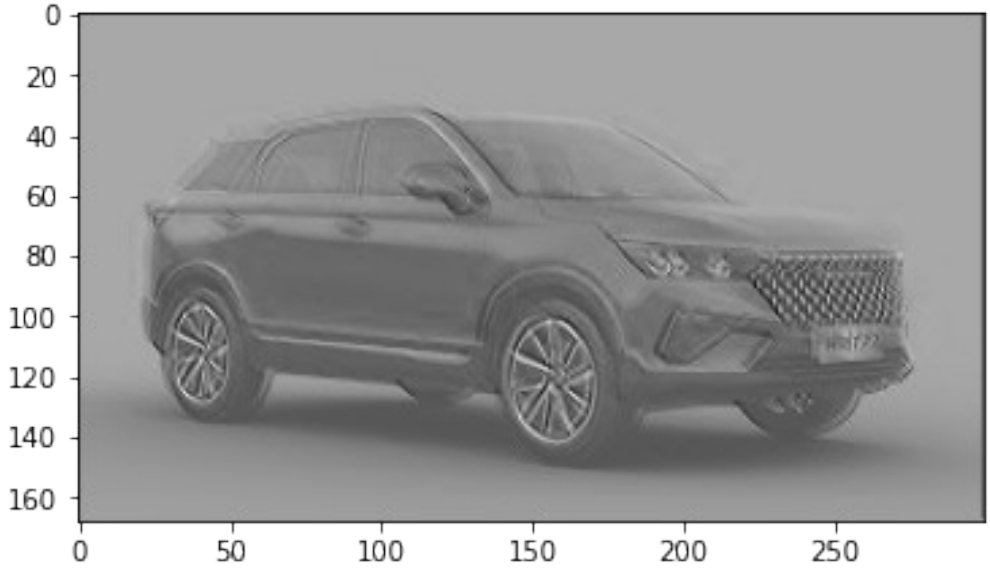


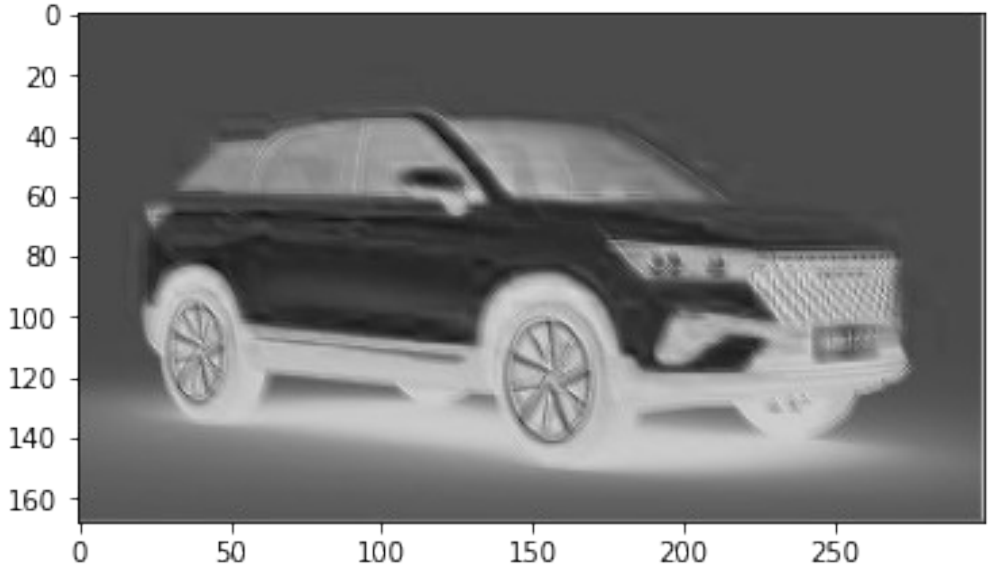

```
img = read_img(img_path, grayscale=False)
input_img = reshaping_in(img, grayscale=False)
model = get_conv_model(filters=10, filter_size=(3,3), strides=(1,1),
input_shape=input_img.shape[1:], padding="same")
apply_conv_model_and_visualize(input_img, model)
```

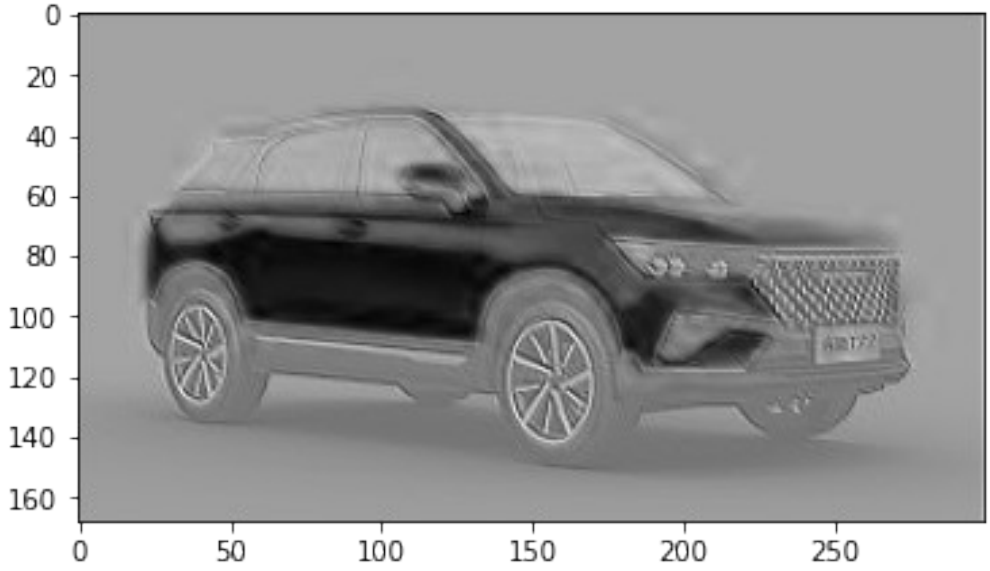
```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_7 (Conv2D)           (None, 168, 300, 10)      280

=================================================================
Total params: 280
Trainable params: 280
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 0s 71ms/step
(1, 168, 300, 10)
```

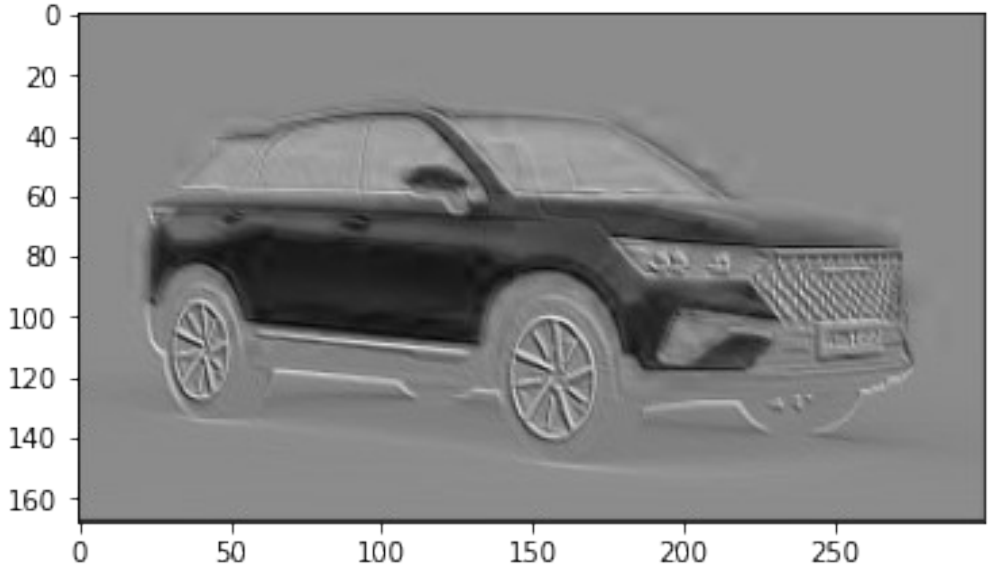


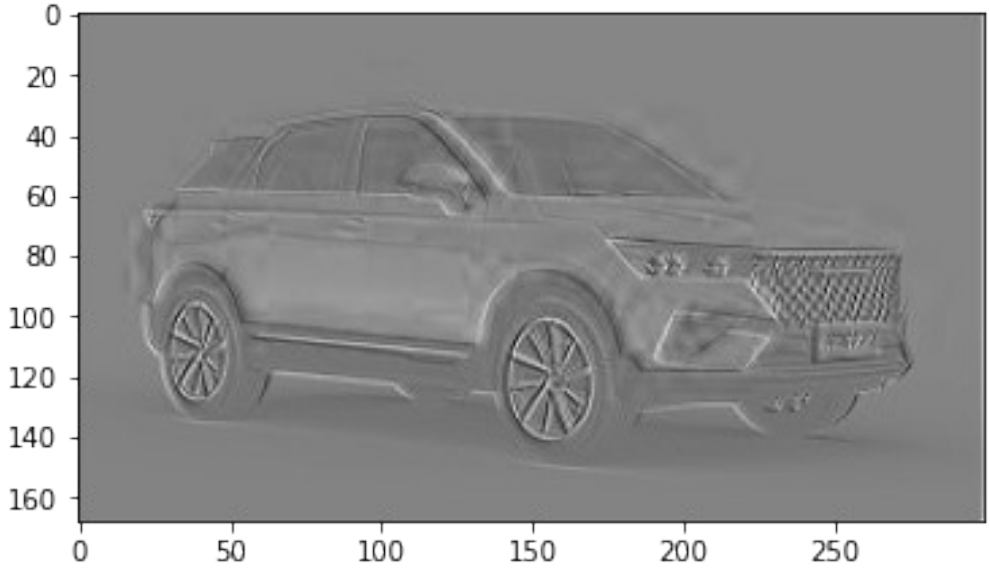

0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

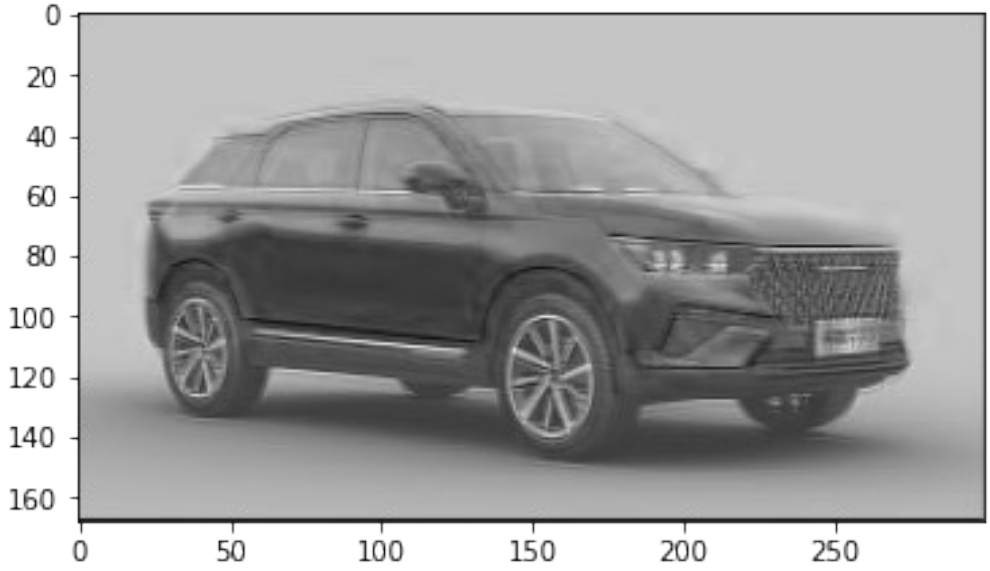

0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

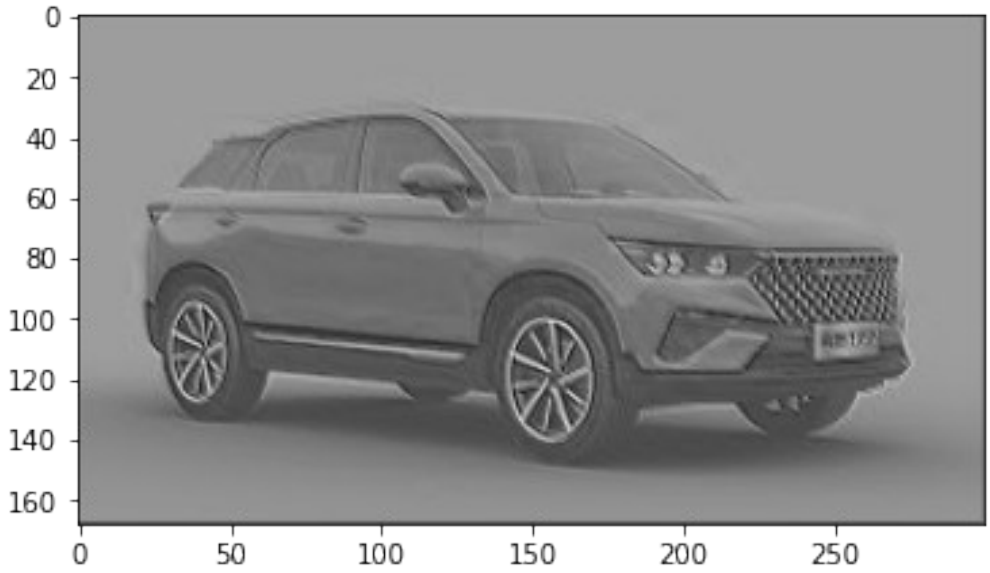

0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250


0
20
40
60
80
100
120
140
160
0
50
100
150
200
250

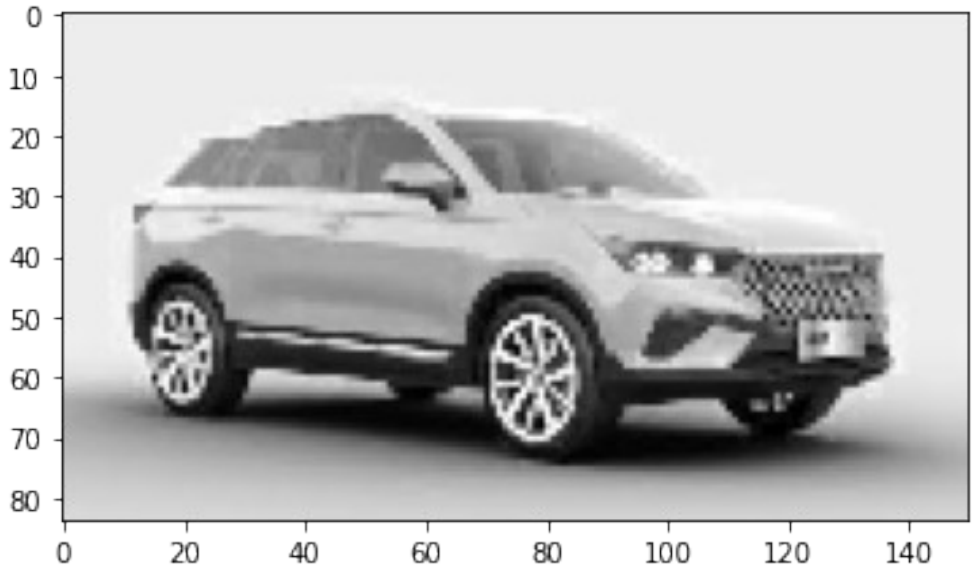

0
20
40
60
80
100
120
140
160
0
50
100
150
200
250




```
def max_pooling(img, pool_size=(2,2), strides=(2,2)):
  reshaped_img = reshaping_in(img)
  pooling_layer = tf.keras.layers.MaxPool2D(pool_size=pool_size,
strides=strides)
  result = pooling_layer(reshaped_img)
  return result

img = read_img(img_path)
print(img.shape)
result = max_pooling(img)
print(result.shape)
```

```
(168, 300)
(1, 84, 150, 1)
```

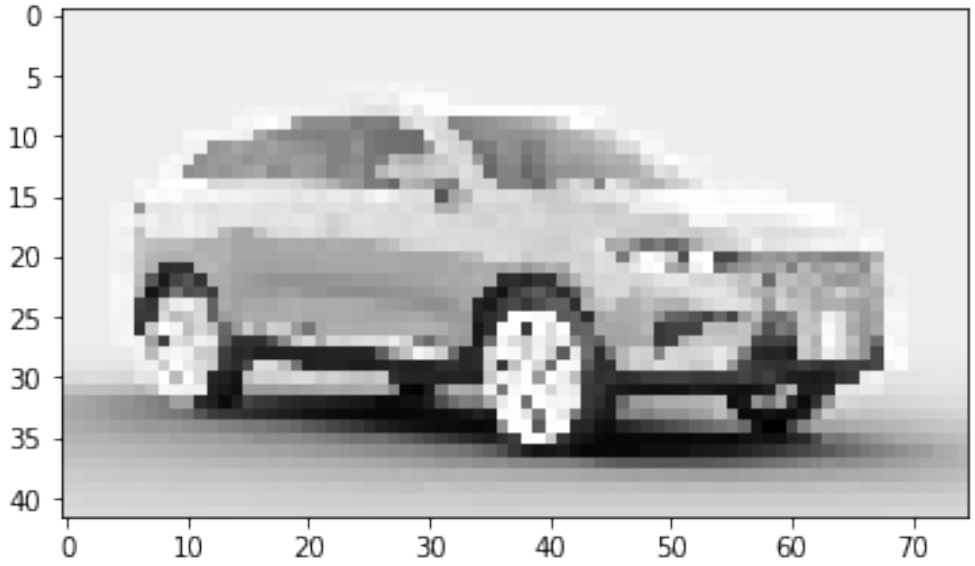

```
def plot_pooling(result):
  _, row, col, _ = result.shape
  reshape = tf.reshape(result, (row, col))
  plt.imshow(reshape, cmap="gray")

plot_pooling(result)
```




```
_, row, col, _ = result.shape
reshape = tf.reshape(result, (row, col))

result = max_pooling(reshape.numpy())
print(result.shape)
```

```
(1, 42, 75, 1)
```

```
plot_pooling(result)
```

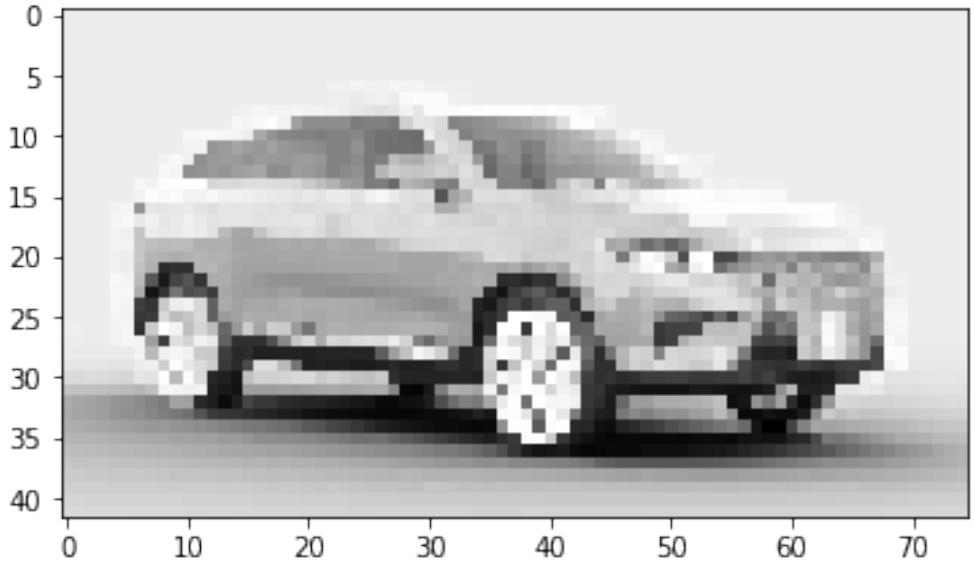


GlobalAvgPool2D

```
def global_avg_pooling(img, grayscale):

  reshaped_img = reshaping_in(img, grayscale)
  pooling_layer = tf.keras.layers

  result = pooling_layer(reshaped_img)
  return result

img = read_img(img_path, grayscale=False)
print(img.shape)
result = global_avg_pooling(img, grayscale=False)
print(result.shape)
print(result)
```

```
(168, 300, 3)

---------------------------------------------------------------------------
-----
TypeError                                 Traceback (most recent call
last)
<ipython-input-88-55cc8fd5883a> in <module>
      1 img = read_img(img_path, grayscale=False)
      2 print(img.shape)
----> 3 result = global_avg_pooling(img, grayscale=False)
      4 print(result.shape)
      5 print(result)

<ipython-input-86-a05212f2582b> in global_avg_pooling(img, grayscale)
      4   pooling_layer = tf.keras.layers
      5
----> 6   result = pooling_layer(reshaped_img)
      7   return result

TypeError: 'module' object is not callable
```

```
def plot_pooling(result):
  _, row, col, _ = result.shape
  reshape = tf.reshape(result, (row, col))
  plt.imshow(reshape, cmap="gray")

plot_pooling(result)
```




```
CONV_LAYER = [tf.keras.layers.Conv2D(filters=1,
                                     kernel_size=(3,3),
                                     strides=(1,1),
                                     input_shape=color_car.shape[1:],
                                     activation="relu" ),
```

```
              tf.keras.layers.MaxPool2D(pool_size=(2,2),
strides=(2,2)),
              tf.keras.layers.GlobalAveragePooling2D(),
              tf.keras.layers.Dense(10,activation="relu"),
              tf.keras.layers.Dense(2,activation="softmax")]

conv_model = tf.keras.Sequential(CONV_LAYER)
conv_model.summary()

model.save("model.h5")

a = np.zeros((2, 3, 4))

a

array([[[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]],

       [[0., 0., 0., 0.],
        [0., 0., 0., 0.],
        [0., 0., 0., 0.]]])
```