```
import matplotlib.pyplot as plt # Plotting images
import matplotlib.image as mpimg # image related ops
import urllib.request # FOr url conversion
import numpy as np
import cv2 # opencv lib
import tensorflow as tf
```

```
# Image Url of own preference
img_url = "https://upload.wikimedia.org/wikipedia/commons/thumb/5/5c/Lamborghini_Urus_19.09.20_JM_%282%29_%28cropped%29.jpg/1200px-Lambor
```

```
# Downloading and saving image to local content of Google colab
urllib.request.urlretrieve(img_url, '/content/car.jpg')
```

```
('/content/car.jpg', <http.client.HTTPMessage at 0x7f7390771b10>)
```

```
img_path = '/content/car.jpg'
```

```
# Reading image from local file using mpimg ( Matplotlib.image)
car1 = mpimg.imread('/content/car.jpg')
```

```
# Shape of Image
car1.shape
```

```
(636, 1200, 3)
```

```
# Shape of particular row out of 636 rows
car1[167].shape
```

```
(1200, 3)
```

```
# Plotting the image of our imported url
plt.imshow(car1)
```

```
<matplotlib.image.AxesImage at 0x7f7420ec3e50>
```


```
# first [0] identifies 0 th row and again [0] refers to 0 th column of 0 th row means top left corner of the image.
car1[0][0]
```

```
array([43, 57, 40], dtype=uint8)
```

```
# Reading image using cv2 with path given by img_path cv2.imread is used to read in color mode and will have three color channel (R,G,B)
car1_cv2 = cv2.imread(img_path)
```

```
# Array value of car1
car1_cv2
```

```
array([[[ 40,  57,  43],
        [ 36,  53,  39],
        [ 36,  53,  39],
        ...,
        [ 16,  19,  17],
        [ 16,  19,  17],
        [ 18,  21,  19]],

       [[ 47,  64,  50],
        [ 44,  61,  47],
        [ 45,  62,  48],
        ...,
        [ 16,  19,  17],
```

```
        [ 16,  19,  17],
        [ 18,  21,  19]],

       [[ 50,  67,  53],
        [ 48,  65,  51],
        [ 49,  66,  52],
        ...,
        [ 17,  19,  19],
        [ 17,  20,  18],
        [ 18,  21,  19]],

       ...,

       [[125, 115, 115],
        [131, 121, 121],
        [137, 127, 127],
        ...,
        [ 78,  70,  70],
        [ 70,  62,  62],
        [ 74,  66,  66]],

       [[128, 118, 118],
        [133, 123, 123],
        [136, 126, 126],
        ...,
        [ 81,  73,  73],
        [ 69,  61,  61],
        [ 67,  59,  59]],

       [[131, 121, 121],
        [136, 126, 126],
        [134, 124, 124],
        ...,
        [ 91,  83,  83],
        [ 76,  68,  68],
        [ 70,  62,  62]]], dtype=uint8)
```

```
# Importing cv2_imshow to display the numpy array converted image , cv2_imshow is function provided by colab to display image using OpenC
from google.colab.patches import cv2_imshow
cv2_imshow(car1_cv2)
```

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

```
<matplotlib.image.AxesImage at 0x7f738e4b8a00>
```

```
# Converting car1_cv2 form BRG to RGB format as cv2 read in BRG format using cv2.cvtColor function of OpenCv2
car1_cv2_BGR_RGB = cv2.cvtColor(car1_cv2, cv2.COLOR_BGR2RGB)
plt.imshow(car1_cv2_BGR_RGB)
```

```
<matplotlib.image.AxesImage at 0x7f738e52a740>
```

```
# Array value od car1 image after conversion from BRG to RGB
car1_cv2_BGR_RGB
```

```
array([[[ 43,  57,  40],
        [ 39,  53,  36],
        [ 39,  53,  36],
        ...,
        [ 17,  19,  16],
        [ 17,  19,  16],
        [ 19,  21,  18]],

       [[ 50,  64,  47],
        [ 47,  61,  44],
        [ 48,  62,  45],
        ...,
        [ 17,  19,  16],
        [ 17,  19,  16],
        [ 19,  21,  18]],

       [[ 53,  67,  50],
        [ 51,  65,  48],
        [ 52,  66,  49],
        ...,
        [ 19,  19,  17],
        [ 18,  20,  17],
        [ 19,  21,  18]],

       ...,

       [[115, 115, 125],
        [121, 121, 131],
        [127, 127, 137],
        ...,
        [ 70,  70,  78],
        [ 62,  62,  70],
        [ 66,  66,  74]],

       [[118, 118, 128],
        [123, 123, 133],
        [126, 126, 136],
        ...,
        [ 73,  73,  81],
        [ 61,  61,  69],
        [ 59,  59,  67]],

       [[121, 121, 131],
        [126, 126, 136],
```

```
        [124, 124, 134],
        ...,
        [ 83,  83,  91],
        [ 68,  68,  76],
        [ 62,  62,  70]]], dtype=uint8)
```

```
car1_cv2_BGR_RGB.shape
```

```
(636, 1200, 3)
```

```
# converting car1_cv2 BGR fromat to Gray format using cv2.cvtColor function
car1_cv2_BGR_GRAY = cv2.cvtColor(car1_cv2, cv2.COLOR_BGR2GRAY)
plt.imshow(car1_cv2_BGR_GRAY, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f738e59dc60>
```


```
# Gray format will just have one dimension rather than 3 like in RGB or BGR
car1_cv2_BGR_GRAY.shape
```

```
(636, 1200)
```

```
car1_cv2_BGR_GRAY
```

```
array([[ 51,  47,  47, ...,  18,  18,  20],
       [ 58,  55,  56, ...,  18,  18,  20],
       [ 61,  59,  60, ...,  19,  19,  20],
       ...,
       [116, 122, 128, ...,  71,  63,  67],
       [119, 124, 127, ...,  74,  62,  60],
       [122, 127, 125, ...,  84,  69,  63]], dtype=uint8)
```

```
# Max and Min out of GRAY converted image.
car1_cv2_BGR_GRAY.min(), car1_cv2_BGR_GRAY.max()
```

```
(0, 255)
```

```
# cv2.split will split the provided image in there corresponding R,G,B channels.
cv2.split(car1_cv2)
```

```
(array([[ 40,  36,  36, ...,  16,  16,  18],
        [ 47,  44,  45, ...,  16,  16,  18],
        [ 50,  48,  49, ...,  17,  17,  18],
        ...,
        [125, 131, 137, ...,  78,  70,  74],
        [128, 133, 136, ...,  81,  69,  67],
        [131, 136, 134, ...,  91,  76,  70]], dtype=uint8),
 array([[ 57,  53,  53, ...,  19,  19,  21],
        [ 64,  61,  62, ...,  19,  19,  21],
        [ 67,  65,  66, ...,  19,  20,  21],
        ...,
        [115, 121, 127, ...,  70,  62,  66],
        [118, 123, 126, ...,  73,  61,  59],
        [121, 126, 124, ...,  83,  68,  62]], dtype=uint8),
 array([[ 43,  39,  39, ...,  17,  17,  19],
        [ 50,  47,  48, ...,  17,  17,  19],
        [ 53,  51,  52, ...,  19,  18,  19],
        ...,
        [115, 121, 127, ...,  70,  62,  66],
        [118, 123, 126, ...,  73,  61,  59],
        [121, 126, 124, ...,  83,  68,  62]], dtype=uint8))
```

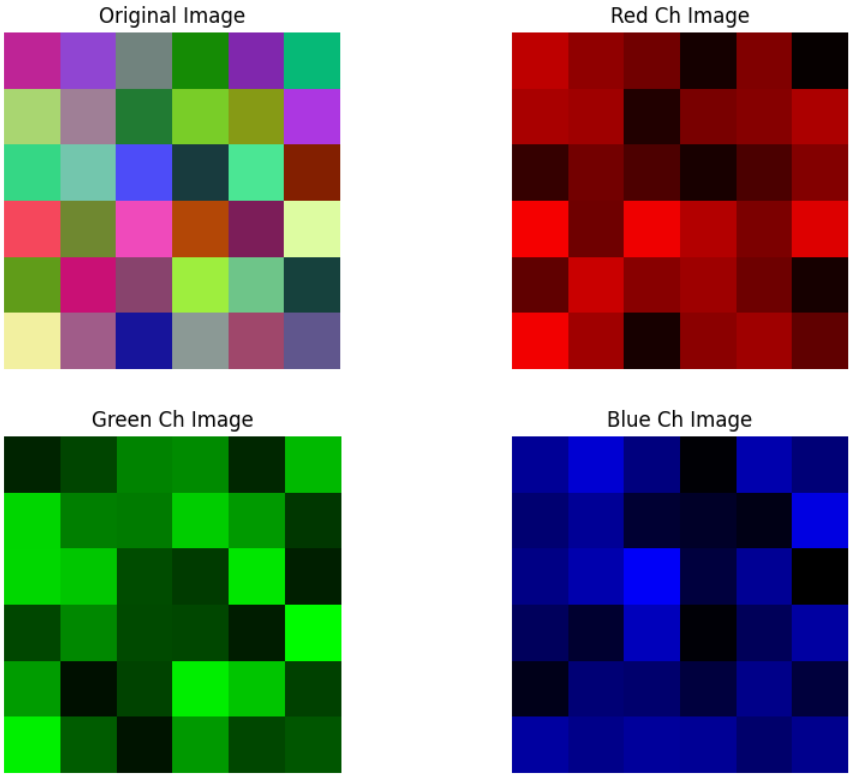
understanding composition of colored images

```
def viusalize_RGB_channel(imgArray=None, figsize=(10,8)):
  # cv2.split() splits input image into three color channels(Bule , Green and Red)
  B, G, R = cv2.split(imgArray)

  # create zero matrix of size B (Blue) channel and keeping data type pf B(Blue).
  Z = np.zeros(B.shape, dtype=B.dtype) # can use any channel

  # Creating 2x2 grid of subplots with fig represent whole window (2x2) with  \n ax as array of subplot
  # used to set properties to individual subplot , ex -ax[0,1] represent 1st row and 2nd column.
  fig, ax = plt.subplots(2,2, figsize=figsize)

  # ax.ravel() return 1D array for all subplot in ax array for loop will then iterate
  # over each subplot and set_axis_off() to turn of corrosponding axiz of subplot.
  # We turn off axis to hide x , y axis labels from view.
  [axi.set_axis_off() for axi in ax.ravel()]

  # ax[0,0] top left subplot location 0 th index & 1st row and 0 th index & 1st column and set title.
  ax[0,0].set_title("Original Image")
  # ax[0,0] set axis off and show image in R,G,B format
  ax[0,0].imshow(cv2.merge((R,G,B)))

  # ax[0,0] top left subplot location 1 th row and 2 th column and set title.
  ax[0,1].set_title("Red Ch Image")
  # ax[0,1] set off axis and show image in R channel and z means zero matrices for other
  ax[0,1].imshow(cv2.merge((R,Z,Z)))

  # ax[0,0] top left subplot location 2 th row and 1 th column and set title.
  ax[1,0].set_title("Green Ch Image")
  # ax[0,1] set off axis and show image in G channel and z means zero matrices for other
  ax[1,0].imshow(cv2.merge((Z,G,Z)))

  # ax[0,0] top left subplot location 2 th row and 2 th column and set title.
  ax[1,1].set_title("Blue Ch Image")
  # ax[0,1] set off axis and show image in B channel and z means zero matrices for other
  ax[1,1].imshow(cv2.merge((Z,Z,B)))


viusalize_RGB_channel(imgArray=car1_cv2)
```


```
# creating 3 dim numpy array of size 6x6x3 with random integer value within (0-255) range
random_colored_img = np.random.randint(0, 255, (6,6,3))


# Printing array value
random_colored_img

     array([[[150,  36, 190],
             [210,  69, 144],
             [126, 131, 113],
             [  5, 139,  21],
```

```
       [173,  39, 128],
       [119, 185,   6]],

      [[113, 214, 169],
       [150, 127, 159],
       [ 51, 123,  33],
       [ 40, 205, 121],
       [ 21, 154, 134],
       [225,  55, 172]],

      [[133, 215,  53],
       [173, 198, 115],
       [248,  76,  77],
       [ 62,  59,  24],
       [148, 230,  75],
       [  0,  31, 131]],

      [[ 92,  71, 245],
       [ 48, 136, 111],
       [187,  74, 239],
       [  6,  71, 179],
       [ 89,  29, 124],
       [161, 252, 221]],

      [[ 25, 156,  96],
       [117,  16, 201],
       [109,  67, 136],
       [ 62, 238, 158],
       [137, 197, 110],
       [ 61,  65,  22]],

      [[160, 240, 242],
       [137,  92, 160],
       [155,  20,  23],
       [149, 153, 139],
       [107,  71, 159],
       [141,  86,  96]]])
```

```
# Passing same to function to see R,G,B appearance.
viusalize_RGB_channel(imgArray=random_colored_img)
```



understanding filters

```
# sobel is filter and is used in image processing for edge detection
sobel = np.array([[ 1, 0,-1],
```

```
                  [ 2, 0,-2],
                  [ 1, 0,-1]])


# In idle form it's verical edges
print("highlighting Vertical edges:\n", sobel)
```

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

```
# To print in horizontal form we need to transpose it.
print("highlighting Horizontal edges:\n", sobel.T)
```

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

```
# Consider below example to check custom image formation using 0 and 255.
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],
           ]
```

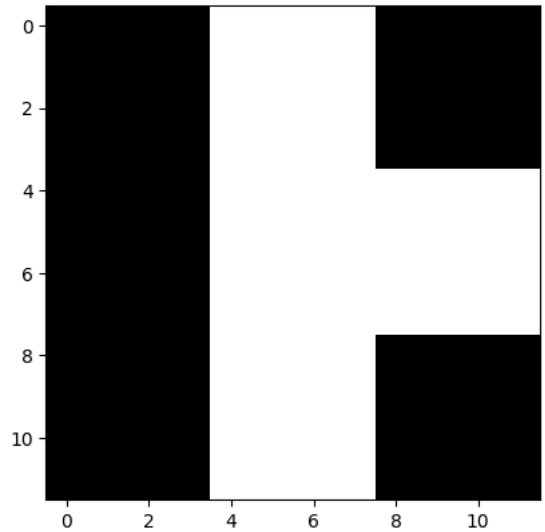
```
# Converting it to array so that we can plot the image.
example1 = np.array(example1)
```

```
# Printing 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]])
```

```
# plotting example1 and cmap = gray using imshow function specift colormap to displaying image.
plt.imshow(example1, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f738c0d0970>
```



```
def simple_conv(imgFilter=None, picture=None):

  p_row, p_col = picture.shape # extract the shape of the image and assigning it to p_row and p_column.

  k = imgFilter.shape[0] # Extracting size of first dimension of Imagefilter.

  temp = list() # Creating blank list to store list of convolution operation.

  stride = 1 # Defining movement of the filter window in each direction with stride of 1.

  # resulant image size dimensions calculated based on image input size, filter and strides.
  final_cols = (p_col - k)//stride + 1
  final_rows = (p_row - k)//stride + 1

  # Vertically down stride across row by row
  for v_stride in range(final_rows):
    # 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) # appending z to blank list

  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]])
```

```
# Applying filter over image example1
result = simple_conv(imgFilter=sobel, picture=example1)


result
```

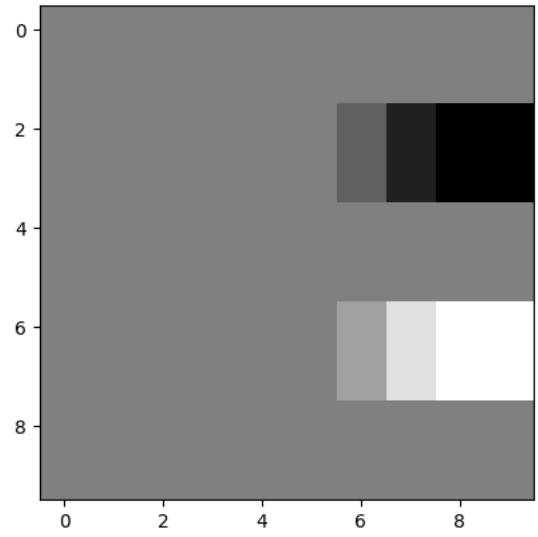
```
array([[    0,     0, -1020, -1020,     0,     0,  1020,  1020,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,  1020,  1020,     0,
            0],
       [    0,     0, -1020, -1020,     0,     0,   765,   765,     0,
            0],
       [    0,     0, -1020, -1020,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

```
# Plotting image of result which is produced applying filter over exapmle1
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f738c149d50>
```

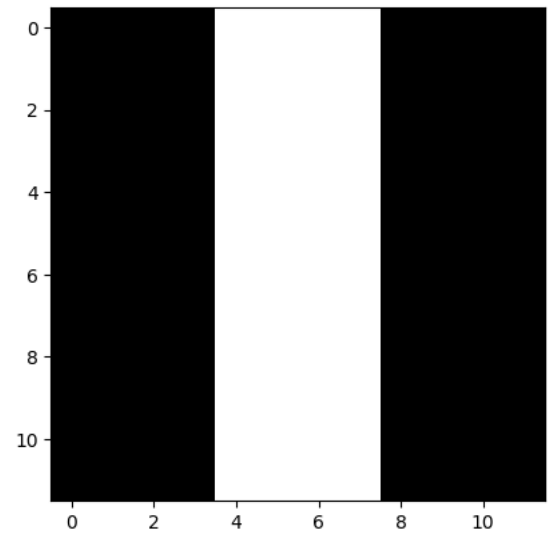
```
# Applying transposed filter to example1 and further plotting it using imshow
result = simple_conv(imgFilter=sobel.T, picture=example1)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f7387f01b40>
```



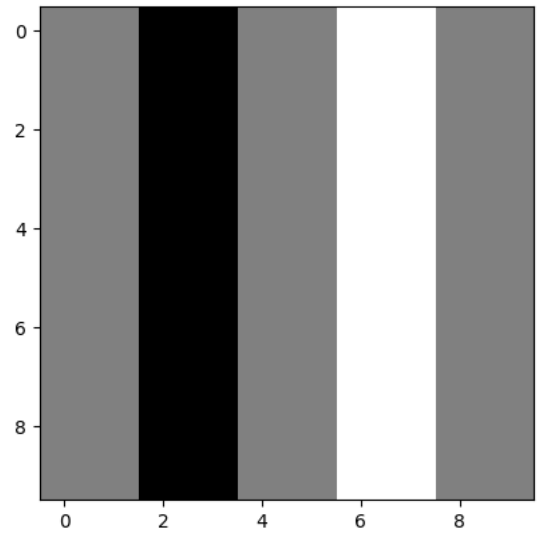
```
# Considering different example and converting same to array and further plotting the image
example2 = [
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [0,0,0,0,255,255,255,255,0,0,0,0],
    [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 0x7f7387f74a30>
```



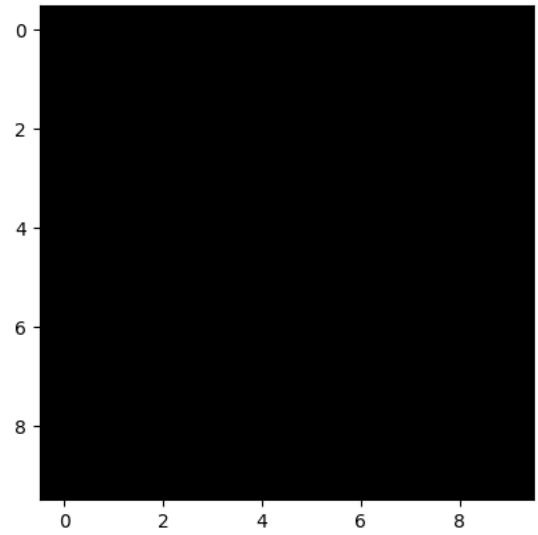
```
# Plotting image after applying sobel filter to example2
result = simple_conv(imgFilter=sobel, picture=example2)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f7387fee530>
```



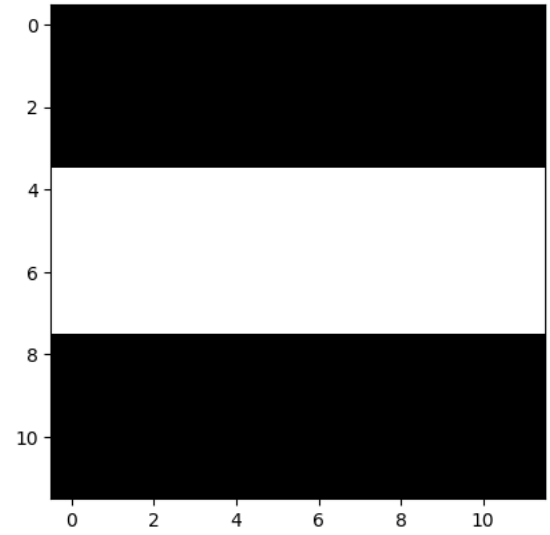
```
# Plotting image by applying trnpose of filter to example2
result = simple_conv(imgFilter=sobel.T, picture=example2)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f7387e5dcf0>
```



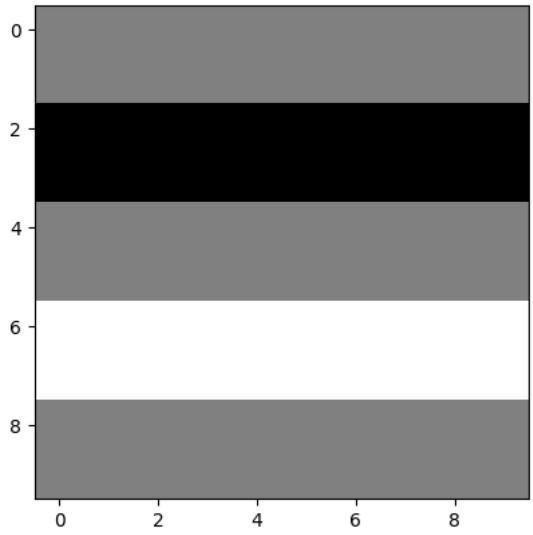
```
# Plotting transpose image of example2 without applying filter to it.
example2_T = np.array(example2.T)
plt.imshow(example2_T, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f7387ec99c0>
```



```
# Plotting image after applying tranpose filter to tranpose of example2
result = simple_conv(imgFilter=sobel.T, picture=example2_T)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f7387d367d0>
```




```
# Applying simple filter to transpose of example2
result = simple_conv(imgFilter=sobel, picture=example2_T)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f7387da0e50>
```

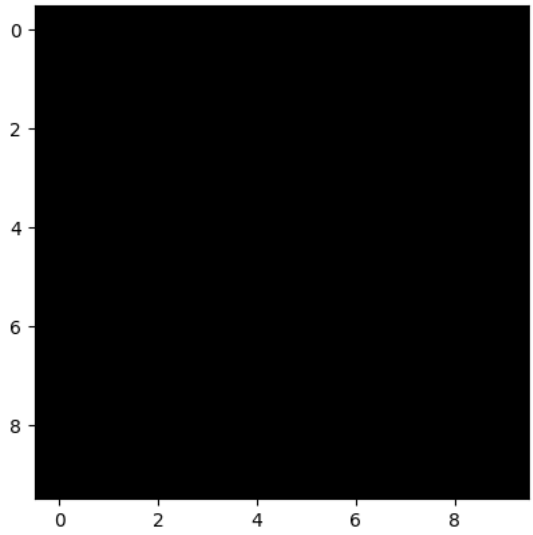
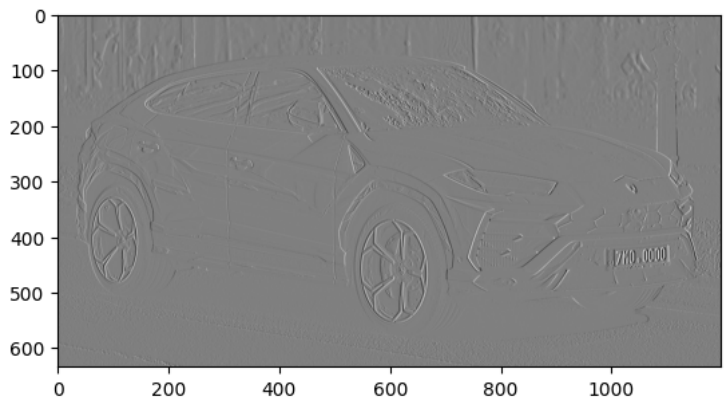


Image of Transposed filter to normal image example2 is same as normal filter applyied to transposed image example2 (Full dark with pixel 0).

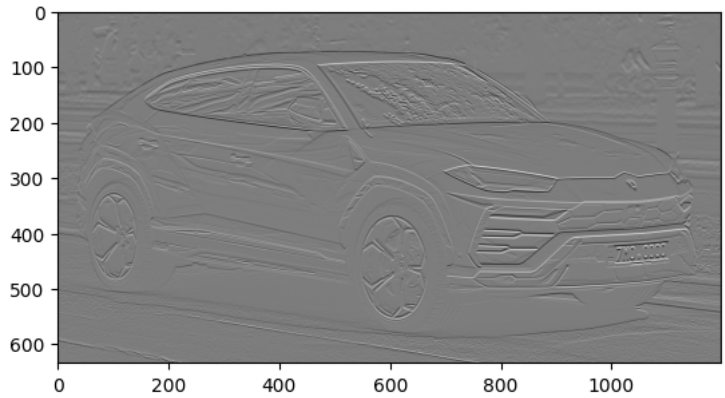
```
# Plotting image after applying sobel filter to image of car converted from BGR to GRAY format.
result = simple_conv(imgFilter=sobel, picture=car1_cv2_BGR_GRAY)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f7387c06b00>
```



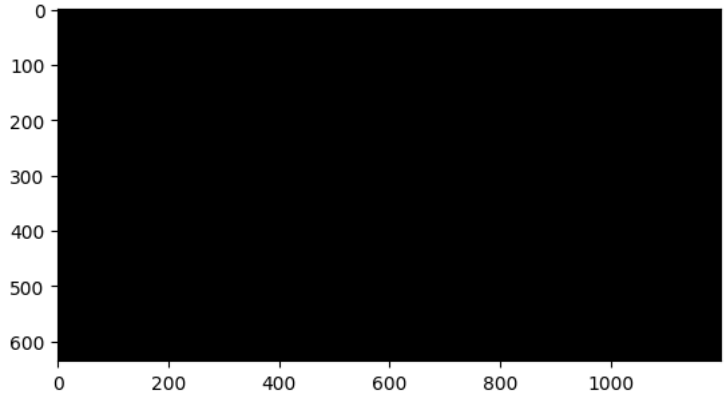
```
# Plotting image after applying transposed sobel filter to image of car converted from BGR to GRAY format.
result = simple_conv(imgFilter=sobel.T, picture=car1_cv2_BGR_GRAY)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f7387c86da0>
```



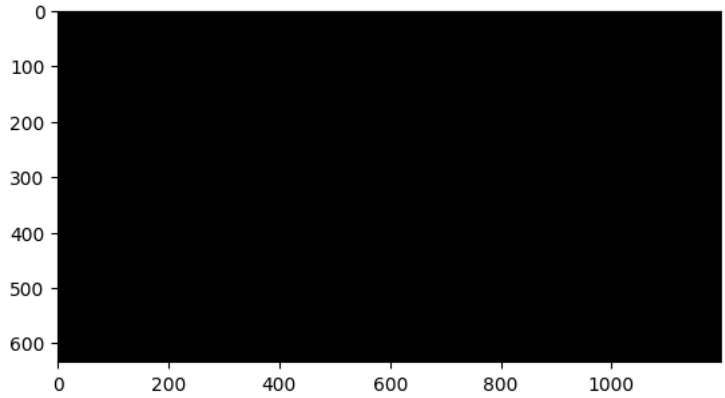

```
nothing = np.zeros(car1_cv2_BGR_GRAY.shape) # Converting the pixels and replacing it with 0 in the car1_cv2 numpy array and storing in nc
plt.imshow(nothing, cmap="gray") # Plotting nothing having all zero pixels
```

```
<matplotlib.image.AxesImage at 0x7f73867e59c0>
```




```
# PLotting image with pixel 0 applying filter to it in transpose form as it's will also be blank.
result = simple_conv(imgFilter=sobel.T, picture=nothing)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f73868759f0>
```




```
# Creating custom random filter of 3x3
random_f = np.random.randn(3,3)
random_f
```

```
array([[ 2.15236481,  1.62763388, -1.54010984],
       [ 0.45772849, -0.37678069, -0.45294726],
       [ 0.88143569,  1.81538313,  1.68537137]])
```

```
# Plotting image after applying custom created filter witn image car1_cv2
result = simple_conv(imgFilter=random_f, picture=car1_cv2_BGR_GRAY)
plt.imshow(result, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f73869e8fa0>
```


```
def read_img(path, grayscale=True):
  img = cv2.imread(path) # Reading path of the image
  if grayscale: # If Grayscale is true will initiate the loop
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) # Convert the BGR 3 channel image to Gray 1 channel
    plt.imshow(img,cmap="gray") # Plotting Gray channel 1 image.
    return img
  from google.colab.patches import cv2_imshow # If Grayscale is false will plot original image using cv2_imshow
  cv2_imshow(img)
  return img


# Plotting image using greyscale= True
car = read_img(img_path)
```


```
# Plotting image with Grayscale = False
color_car = read_img(img_path, grayscale=False)
```

```
# (1,r,c,1)
row,col = car.shape
car = car.reshape(1,row,col,1) # converting 3 channel image to 1 and reshaping it.
# color_car = color_car.reshape(1,row,col,3)
car.shape # Printing new shape of car
```

```
(1, 636, 1200, 1)
```



```
# CONV_LAYER is list containing single Conv2D layer with 1 filter , filter size 3x3 with single stride in both direction.
CONV_LAYER = [tf.keras.layers.Conv2D(filters=1,
                                     kernel_size=(3,3),
                                     strides=(1,1),
                                     input_shape=car.shape[1:])] # Input shape is augmentented to car.shape[1:]

conv_model = tf.keras.Sequential(CONV_LAYER) # Creating single layered Conv model Sequential model with CONV_LAYER as argument
conv_model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 634, 1198, 1)      10

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

```
# Passing car through each conv model and making prediction
out = conv_model.predict(car)
```

```
1/1 [==============================] - 0s 102ms/step
```

```
# Printing shape of the prediction
out.shape
```

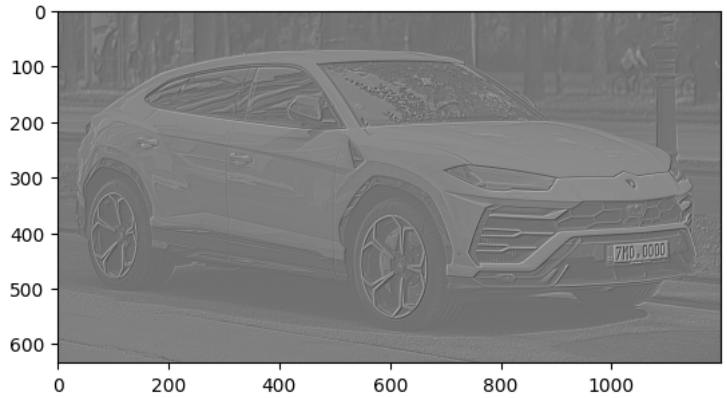
```
(1, 634, 1198, 1)
```

```
# Seprating row and column out of predicted output
row, col = out.shape[1:-1]
```

```
# Reshaping output prediction
reshape_out = out.reshape(row, col)
```

```
# Plotting reshaped image
plt.imshow(reshape_out, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f738729ada0>
```




```
# reshaping image of color car ( Original Image )
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, 636, 1200, 3)
```

```
# Applying conv layer as same as earlier to 3 channel image
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_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 634, 1198, 1)      28

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

```
# Making prediction by passing color_car through conv model
output = conv_model.predict(color_car)
```

```
1/1 [==============================] - 0s 138ms/step
```

```
# Shape of output prediction
output.shape
```

```
(1, 634, 1198, 1)
```

```
# Seprating row and column out of it.
row, col = output.shape[1:-1]
reshape_out = output.reshape(row, col)
```

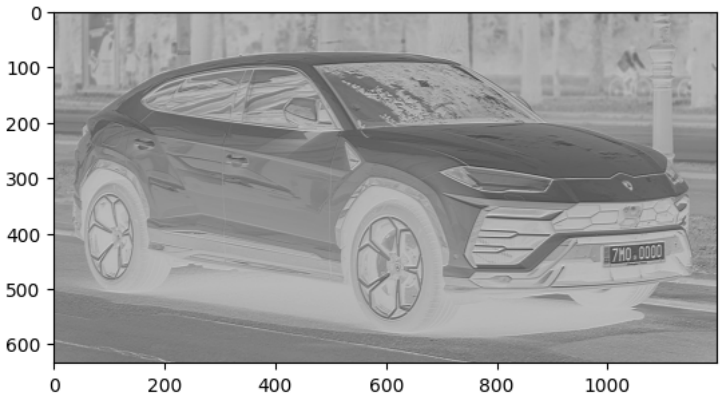
```
# Plotting image with cmap ='gray' of above output
plt.imshow(reshape_out, cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7f7387e41bd0>
```


```
def reshaping_in(img, grayscale=True): # Passing image and grayscale parameters
  if grayscale: # if grayscale is true will continue
    row,col = img.shape # collecting row and column info
    img = img.reshape(1,row,col,1) # reshaping the image
    return img # return image
  row,col,depth = img.shape # When Grayscale is false collect info or row , column and depth.
  color_img = img.reshape(1,row,col,depth) # colored image reshaping
  return color_img # return coloured image

def get_conv_model(filters=1, filter_size=(3,3), strides=(1,1), input_shape=None, padding="valid"):
  # Creating list with single Con2D layer and rest parameters of strides , filter_size , imput shape, no of filters and 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) # Creating single layered Sequential layered model with parameter CONV_LAYER
  conv_model.summary() # Printing summary of model
  return conv_model

def apply_conv_model_and_visualize(img, conv_model): # Passing image and model for visualization
  try:
    out = conv_model.predict(img) # Prediction of output
```

```
      print(out.shape) # Printing shape of output
      row, col, depth = out.shape[1:] # collecting row,column and depth info
      reshape_out = out.reshape(row, col, depth) # reshaping

      for d in range(depth):
        plt.imshow(reshape_out[:,:,d], cmap="gray") # Plotting the image with reshaped dimension in range of depth
        plt.show()

    except Exception as e:
      raise e


img = read_img(img_path, grayscale=False) # Grayscale is false so will plot 3 channel image
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:]) # Passing parameters to train model
apply_conv_model_and_visualize(input_img, model) # passing input image and model for visualization
```

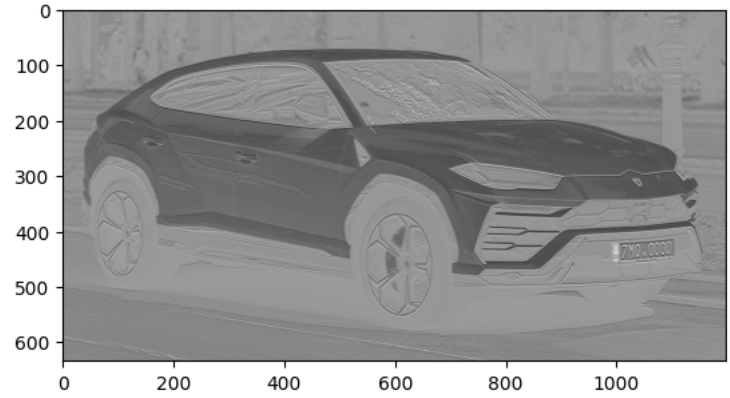
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_2 (Conv2D)           (None, 634, 1198, 1)      28

=================================================================
Total params: 28
Trainable params: 28
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 0s 261ms/step
(1, 634, 1198, 1)
```

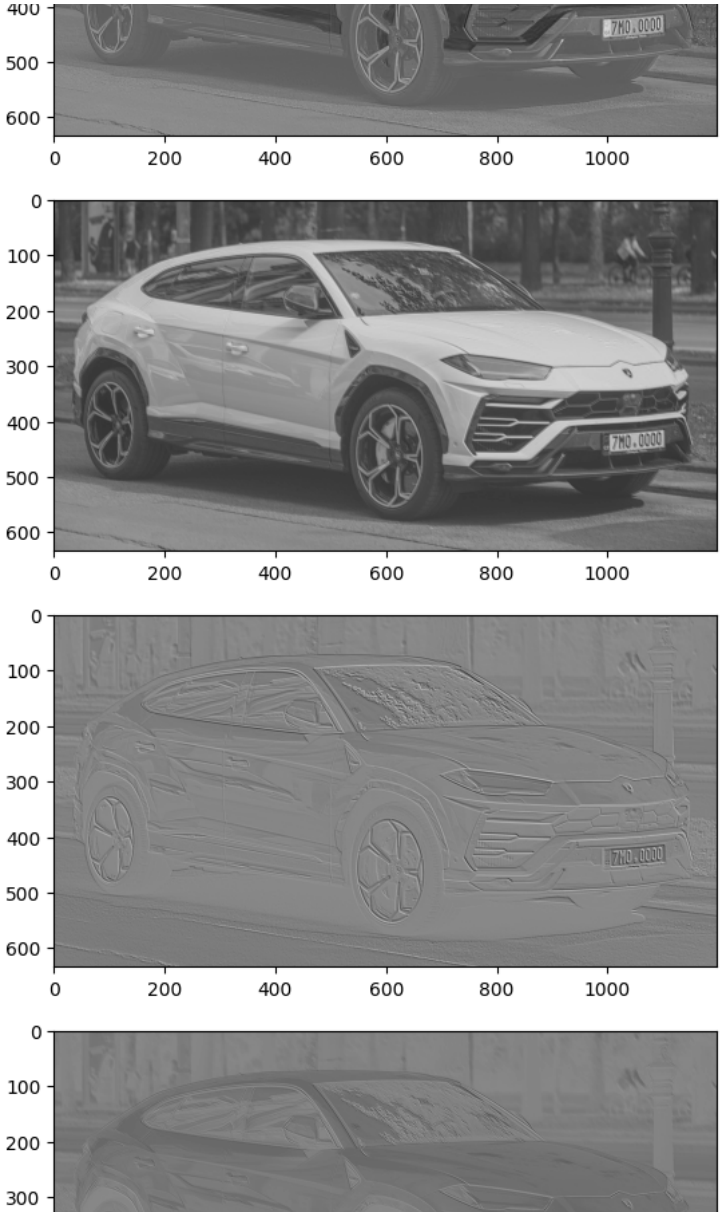



```
img = read_img(img_path, grayscale=False)
input_img = reshaping_in(img, grayscale=False)
model = get_conv_model(filters=10, filter_size=(3,3), strides=(1,1), input_shape=input_img.shape[1:]) # Passing 10 filters with filter s
```

```
model = get_conv_model(filters=10, filter_size=(3,3), strides=(1,1), input_shape=input_img.shape[1:]) # Passing 10 filters  with filter s
apply_conv_model_and_visualize(input_img, model)
```

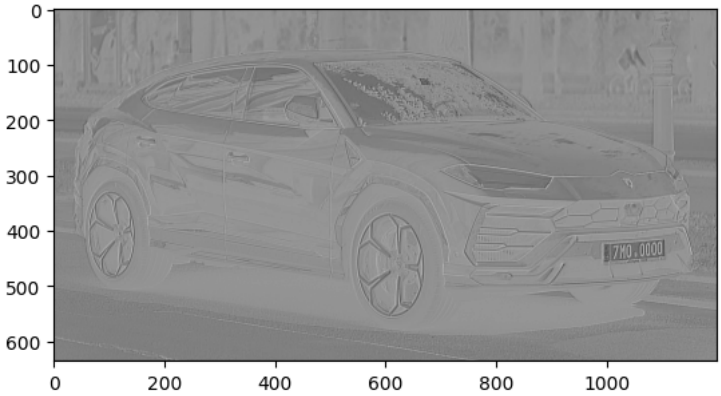

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_3 (Conv2D)           (None, 634, 1198, 10)     280

=================================================================
Total params: 280
Trainable params: 280
Non-trainable params: 0
_________________________________________________________________
1/1 [==============================] - 0s 384ms/step
(1, 634, 1198, 10)
```

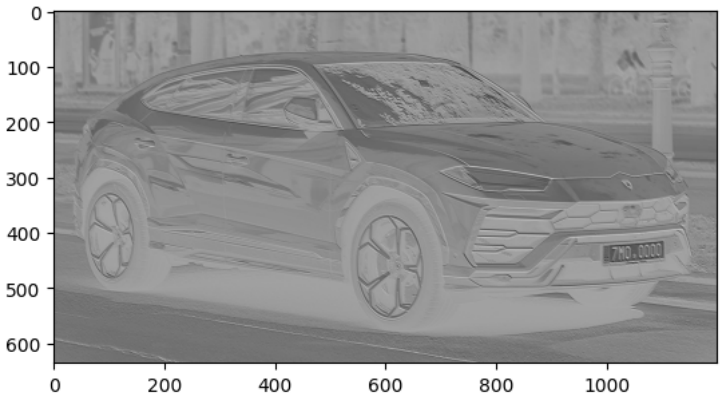


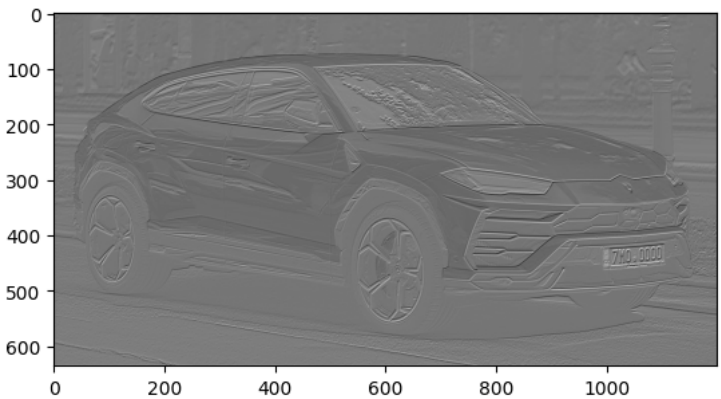


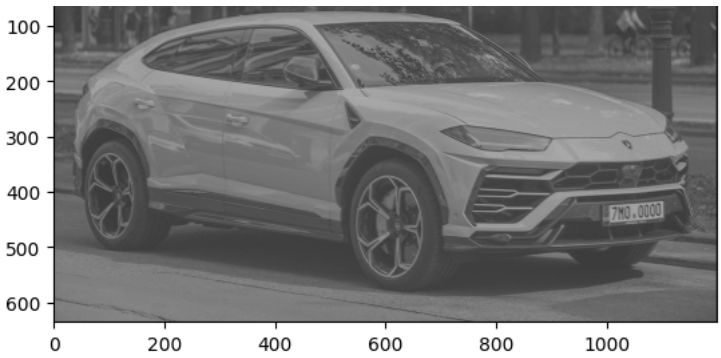
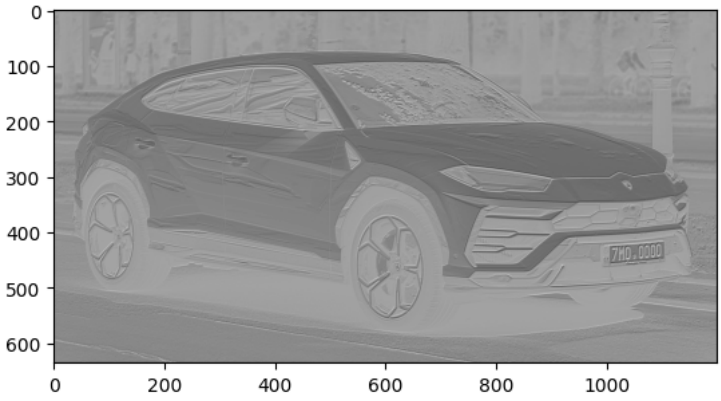
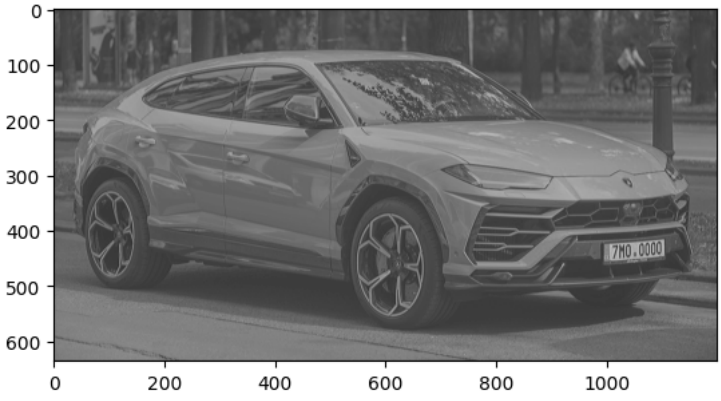
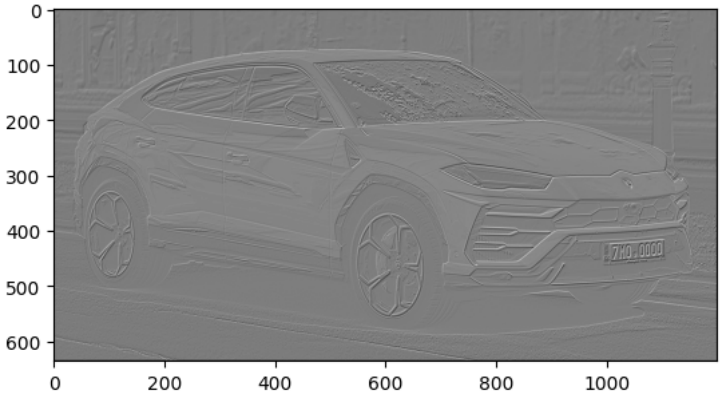
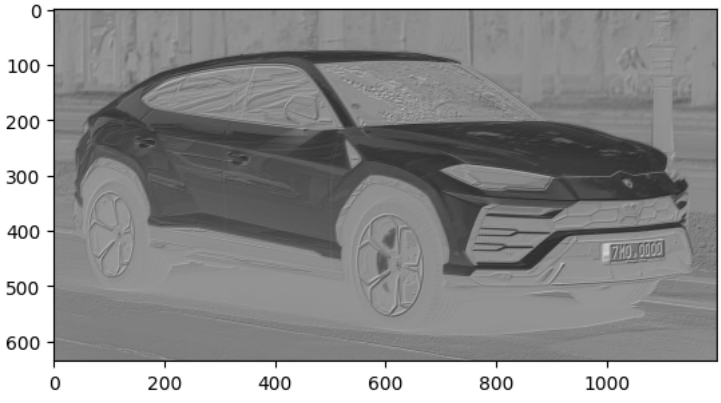
```
img = read_img(img_path, grayscale=False)
input_img = reshaping_in(img, grayscale=False)
model = get_conv_model(filters=10, filter_size=(3,3), strides=(1,1), input_shape=input_img.shape[1:], padding="same") # Passing same para
apply_conv_model_and_visualize(input_img, model)
```

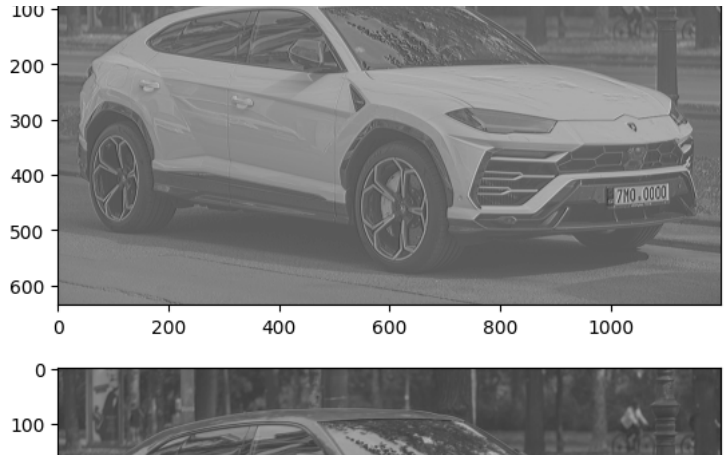
```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 636, 1200, 10)     280

=================================================================
Total params: 280
Trainable params: 280
Non-trainable params: 0
_________________________________________________________________
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_functior
1/1 [==============================] - 0s 299ms/step
(1, 636, 1200, 10)
```







```
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)
```

```
(636, 1200)
(1, 318, 600, 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)
```

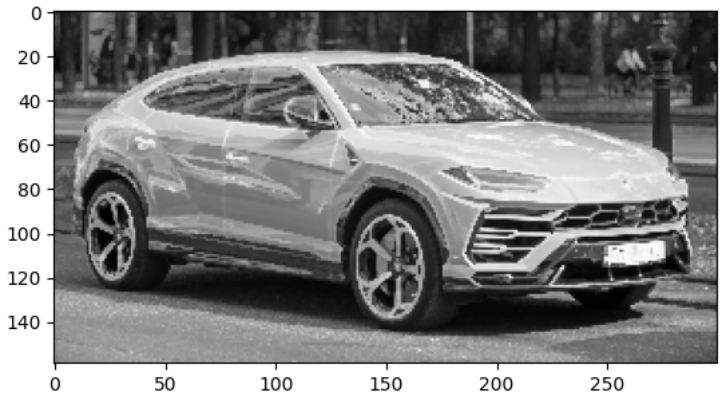
```
(1, 159, 300, 1)
```

```
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


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: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
 conv2d_5 (Conv2D)           (None, 634, 1198, 1)      28

 max_pooling2d_2 (MaxPooling  (None, 317, 599, 1)      0
 2D)

 global_average_pooling2d (G  (None, 1)                0
 lobalAveragePooling2D)

 dense (Dense)               (None, 10)                20

 dense_1 (Dense)             (None, 2)                 22

=================================================================
Total params: 70
Trainable params: 70
Non-trainable params: 0
_________________________________________________________________
```

```
model.save("model.h5")
```

```
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty
```

✓ 0s completed at 7:12 PM