Numpy

- NumPy is the fundamental package for scientific computing in Python.
- It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

Installation

```
pip install numpy
```

For plotting we will use matplotlib

```
pip install matplotlib
```

```
pip install numpy matplotlib python-opencv
```

Matrix Operations

source)

1 - Add

$$\begin{bmatrix} 1 & 3 & 1 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 & 1+5 \\ 1+7 & 0+5 & 0+0 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 6 \\ 8 & 5 & 0 \end{bmatrix}$$

In [4]:

```
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
```

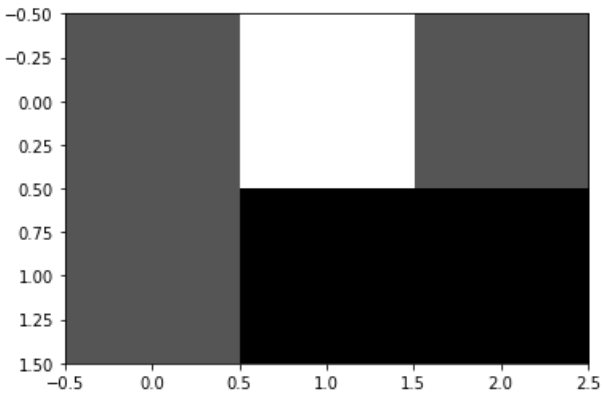
In [23]:

```
def plot(arr):
    print(arr)
    plt.imshow(arr, cmap='gray')
    plt.show()
```

In [24]:

```
a = np.array(
[ [1, 3, 1], [1, 0, 0] ]
)
plot(a)
```

```
[[1 3 1]
 [1 0 0]]
```



In [25]:

```
b = np.array(
[ [0, 0, 5], [7, 5, 0] ]
)
b
```

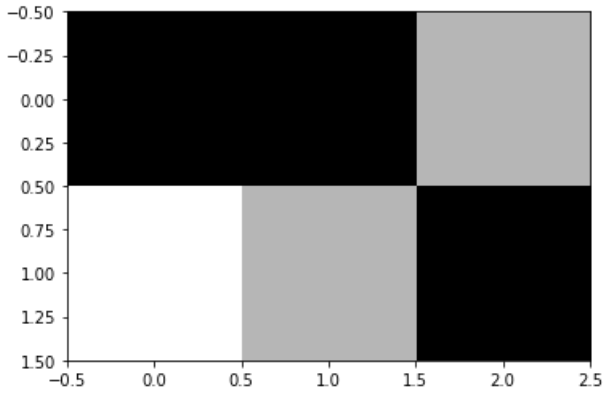
Out[25]:

```
array([[0, 0, 5],
       [7, 5, 0]])
```

In [26]:

```
plot(b)
```

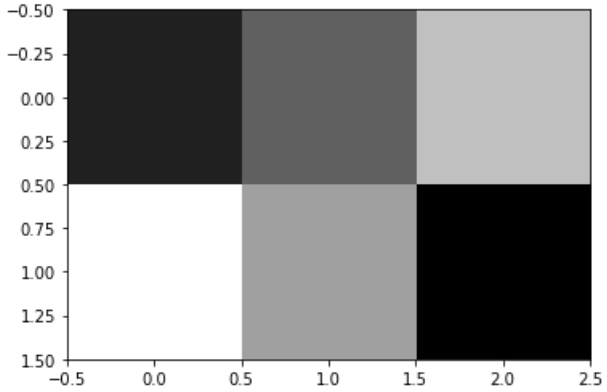
```
[[0 0 5]
 [7 5 0]]
```



In [27]:

```
r = a + b
plot(r)
```

```
[[1 3 6]
 [8 5 0]]
```

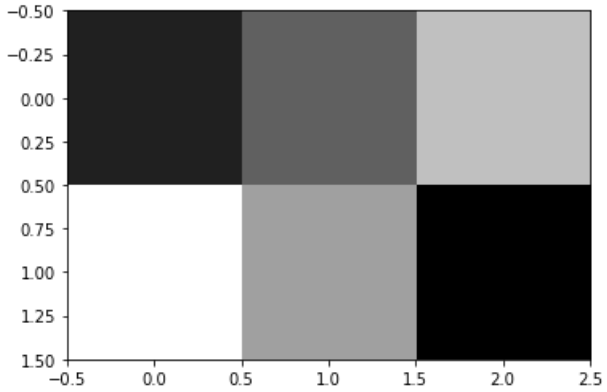


Using numpy to add 2 numbers source

In [28]:

```
r = np.add(a,b)
plot(r)
```

```
[[1 3 6]
 [8 5 0]]
```



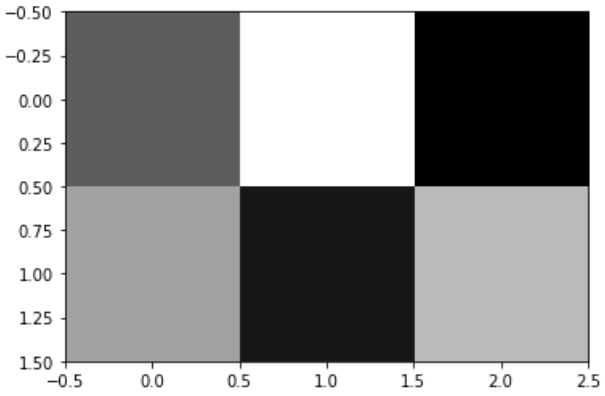
2 - Scalar Dot Matrix

$$2 \cdot \begin{bmatrix} 1 & 8 & -3 \\ 4 & -2 & 5 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 & 2 \cdot 8 & 2 \cdot -3 \\ 2 \cdot 4 & 2 \cdot -2 & 2 \cdot 5 \end{bmatrix} = \begin{bmatrix} 2 & 16 & -6 \\ 8 & -4 & 10 \end{bmatrix}$$

In [29]:

```
a = np.array(
[ [1, 8, -3], [4, -2, 5] ]
)
plot(a)
```

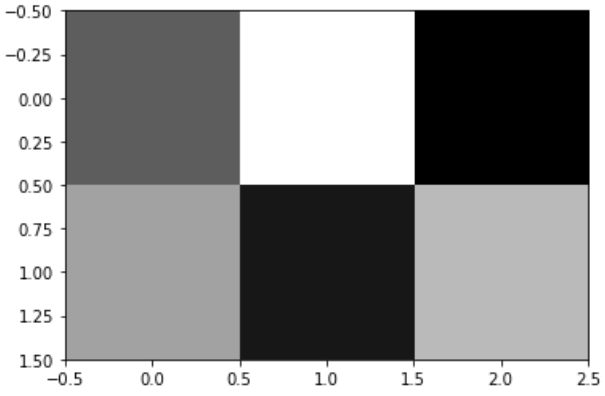
```
[[ 1  8 -3]
 [ 4 -2  5]]
```



In [30]:

```
r = 2 * a
plot(r)
```

```
[[ 2 16 -6]
 [ 8 -4 10]]
```

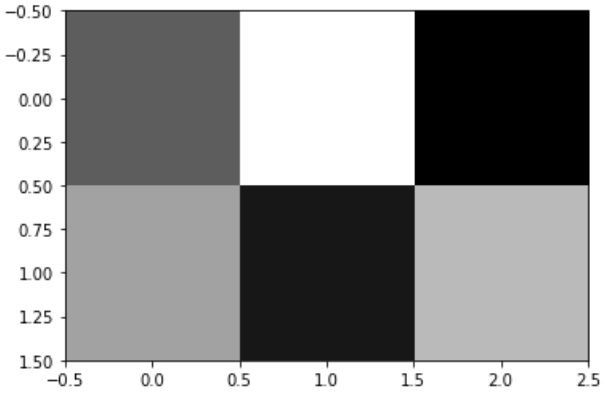


Using numpy dot function source

In [31]:

```
r = np.dot(2, a)
plot(r)
```

```
[[ 2 16 -6]
 [ 8 -4 10]]
```



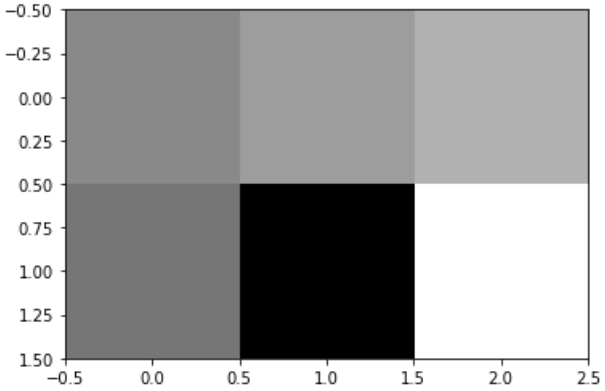
3 - Transpose Matrix

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & -6 & 7 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 0 \\ 2 & -6 \\ 3 & 7 \end{bmatrix}$$

In [32]:

```
a = np.array(
[
    [1,2,3],
    [0,-6,7]
]
)
plot(a)
```

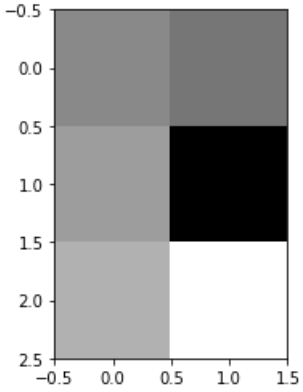
```
[[ 1  2  3]
 [ 0 -6  7]]
```



In [33]:

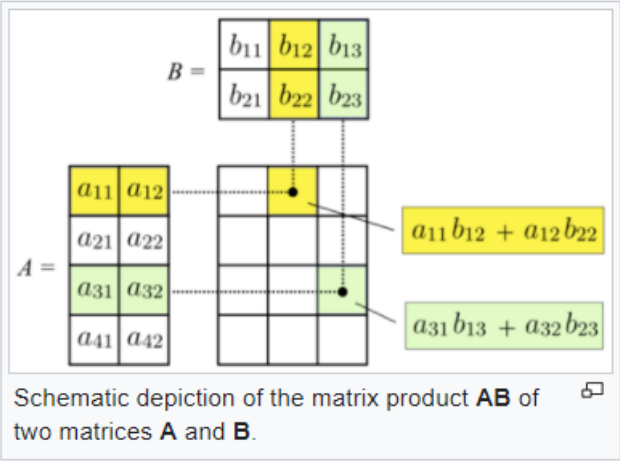
```
r = np.transpose(a)
plot(r)
```

```
[[ 1  0]
 [ 2 -6]
 [ 3  7]]
```



4 - Matrix Multiplication

$$[\mathbf{AB}]_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \cdots + a_{i,n}b_{n,j} = \sum_{r=1}^{n} a_{i,r}b_{r,j},$$



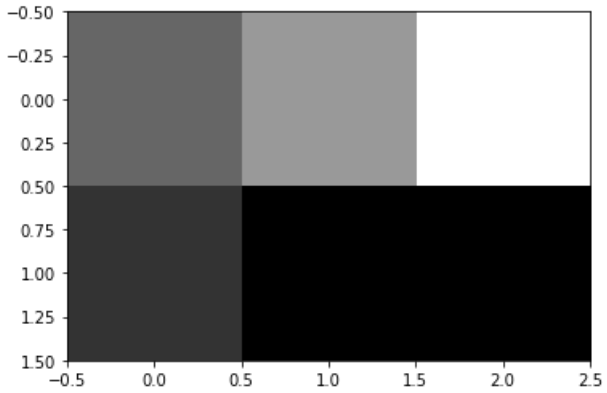
Schematic depiction of the matrix product **AB** of two matrices **A** and **B**.

$$\begin{bmatrix} \underline{2} & \underline{3} & \underline{4} \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & \underline{1000} \\ 1 & \underline{100} \\ 0 & \underline{10} \end{bmatrix} = \begin{bmatrix} 3 & \underline{2340} \\ 0 & 1000 \end{bmatrix}.$$

In [34]:

```
a = np.array([
    [2,3,5],
    [1,0,0]
])
plot(a)
```

```
[[2 3 5]
 [1 0 0]]
```



In [35]:

```
a.shape
```

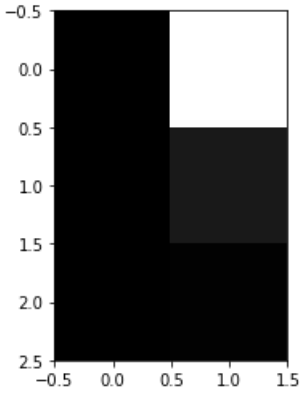
Out[35]:

```
(2, 3)
```

In [36]:

```
b = np.array([
    [0,1000],
    [1,100],
    [0,10]
])
plot(b)
```

```
[[   0 1000]
 [   1  100]
 [   0   10]]
```



In [37]:

```
b.shape
```

Out[37]:

```
(3, 2)
```

In [38]:

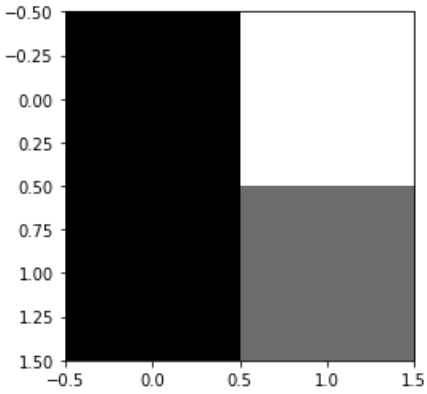
```
a * b
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-9bc1a869709f> in <module>
----> 1 a * b

ValueError: operands could not be broadcast together with shapes (2,3) (3,2)
```

In [39]:

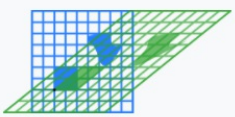
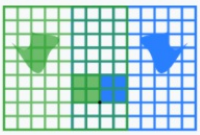
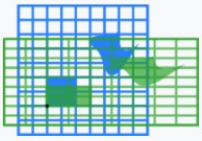
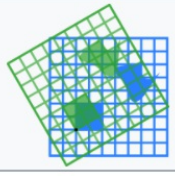
```
r = np.dot(a, b)
plot(r)
```

```
[[   3 2350]
 [   0 1000]]
```



4 - Linear equations

$$\mathbf{A} = \begin{bmatrix} a & c \\ b & d \end{bmatrix}$$

Horizontal shear with m = 1.25.	Reflection through the vertical axis	Squeeze mapping with r = 3/2	Scaling by a factor of 3/2	Rotation by $\pi/6$ = 30°
$\begin{bmatrix} 1 & 1.25 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \frac{3}{2} & 0 \\ 0 & \frac{2}{3} \end{bmatrix}$	$\begin{bmatrix} \frac{3}{2} & 0 \\ 0 & \frac{3}{2} \end{bmatrix}$	$\begin{bmatrix} \cos\left(\frac{\pi}{6}\right) & -\sin\left(\frac{\pi}{6}\right) \\ \sin\left(\frac{\pi}{6}\right) & \cos\left(\frac{\pi}{6}\right) \end{bmatrix}$
				

Linear Operations

Disclaimer source

In [40]:

```
aux = np.ones((100, 100), dtype=int)
aux
```

Out[40]:

```
array([[1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       ...,
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 1, 1, 1]])
```

In [41]:

```
aux.shape
```

Out[41]:

```
(100, 100)
```

In [42]:

```
src = np.vstack(
    [
        np.c_[aux, 2*aux],
        np.c_[3*aux, 4*aux]
    ]
)
src
```

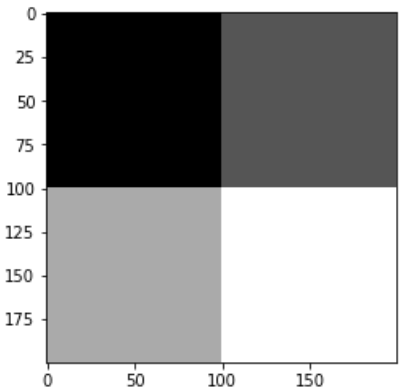
Out[42]:

```
array([[1, 1, 1, ..., 2, 2, 2],
       [1, 1, 1, ..., 2, 2, 2],
       [1, 1, 1, ..., 2, 2, 2],
       ...,
       [3, 3, 3, ..., 4, 4, 4],
       [3, 3, 3, ..., 4, 4, 4],
       [3, 3, 3, ..., 4, 4, 4]])
```

In [43]:

```
plot(src)
```

```
[[1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 ...
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]]
```



Linear Transformation

Scaling the plane in the x-axis by a factor of 1.5

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1.5 & 0 \\ 0 & 0 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix},$$

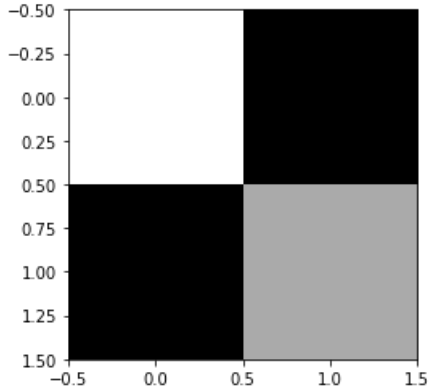
In [44]:

```
def linear_transformation(src, a):
    M, N = src.shape
    points = np.mgrid[0:N, 0:M].reshape((2, M*N))
    new_points = np.linalg.inv(a).dot(points).round().astype(int)
    x, y = new_points.reshape((2, M, N), order='F')
    indices = x + N*y
    return np.take(src, indices, mode='wrap')
```

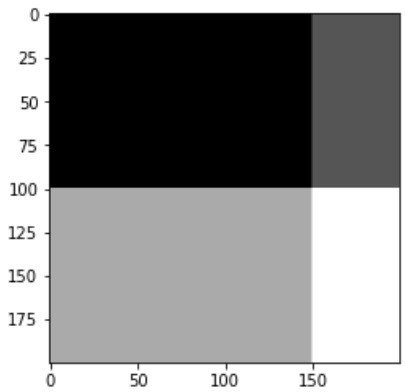
In [45]:

```
a = np.array([
    [1.5, 0],
    [0, 1]
])
plot(a)
dst = linear_transformation(src, a)
plot(dst)
```

```
[[1.5 0. ]
 [0.  1. ]]
```




```
[[1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 ...
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]]
```



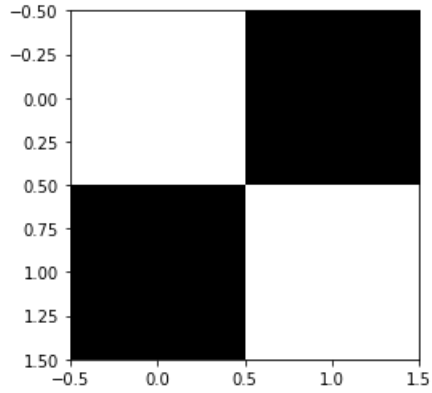
Dilating the plane by a factor of 1.8

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1.8 & 0 \\ 0 & 1.8 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix},$$

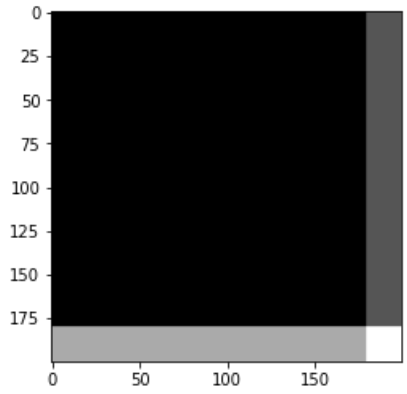
In [46]:

```
a = 1.8*np.eye(2)
plot(a)
dst = linear_transformation(src, a)
plot(dst)
```

```
[[1.8 0. ]
 [0.  1.8]]
```




```
[[1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 ...
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]]
```



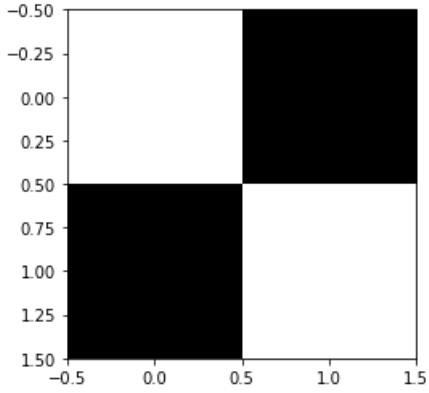
Dilating the plane by a factor of 0.5

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix},$$

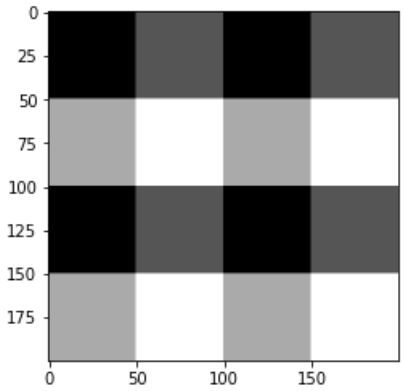
In [47]:

```
a = .5*np.eye(2)
plot(a)
dst = linear_transformation(src, a)
plot(dst)
```

```
[[0.5 0. ]
 [0.  0.5]]
```




```
[[1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 ...
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]]
```



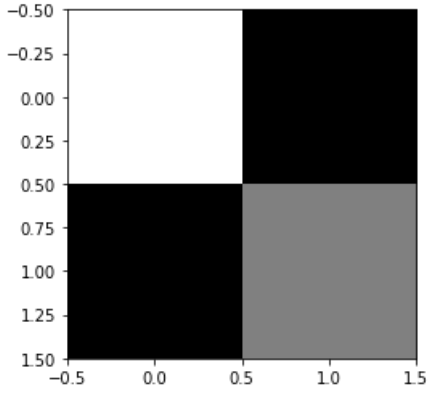
Scaling the plane in the y-axis by a factor of 0.5

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0.5 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix},$$

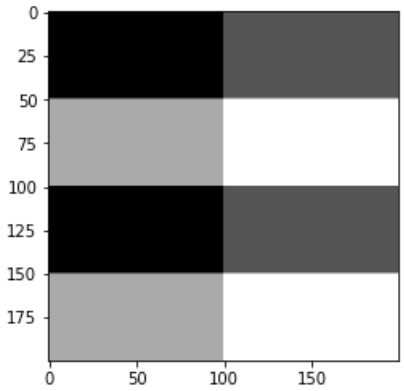
In [48]:

```
a = np.array([[1, 0],
              [0, .5]])
plot(a)
dst = linear_transformation(src, a)
plot(dst)
```

```
[[1.  0. ]
 [0.  0.5]]
```




```
[[1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 ...
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]]
```



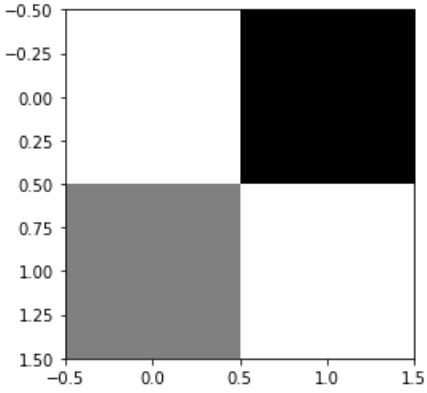
Shearing about the y-axis with a vertical displacement of +x/2

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix},$$

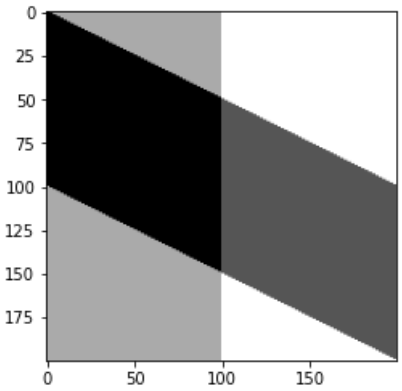
In [49]:

```
a = np.array([[1, 0],
              [.5, 1]])
plot(a)
dst = linear_transformation(src, a)
plot(dst)
```

```
[[1.  0. ]
 [0.5 1. ]]
```




```
[[1 1 3 ... 4 4 4]
 [1 1 1 ... 4 4 4]
 [1 1 1 ... 4 4 4]
 ...
 [3 3 3 ... 2 2 2]
 [3 3 3 ... 4 2 2]
 [3 3 3 ... 4 4 4]]
```



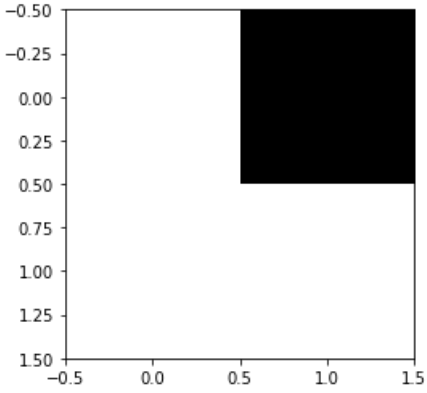
Rotation through 45◦ about the origin

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix},$$

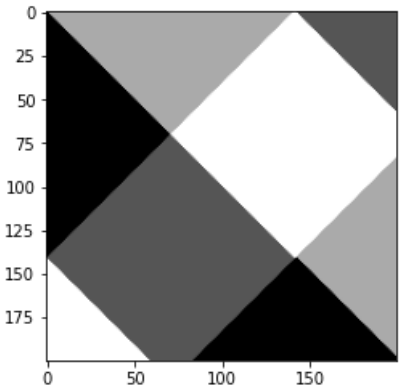
In [50]:

```
alpha = np.pi/4
a = np.array([[np.cos(alpha), -np.sin(alpha)],
              [np.sin(alpha), np.cos(alpha)]])
plot(a)
dst = linear_transformation(src, a)
plot(dst)
```

```
[[ 0.70710678 -0.70710678]
 [ 0.70710678  0.70710678]]
```




```
[[1 3 3 ... 2 2 2]
 [1 1 3 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 ...
 [4 4 4 ... 1 1 1]
 [4 4 4 ... 1 1 1]
 [4 4 4 ... 1 1 1]]
```



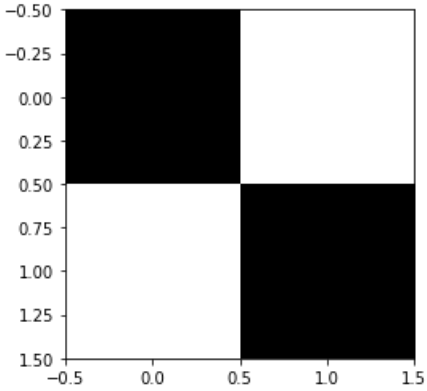
Reflexion in a line with inclination of 45◦ through the origin

$$f\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \cos\frac{\pi}{2} & \sin\frac{\pi}{2} \\ \sin\frac{\pi}{2} & -\cos\frac{\pi}{2} \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix},$$

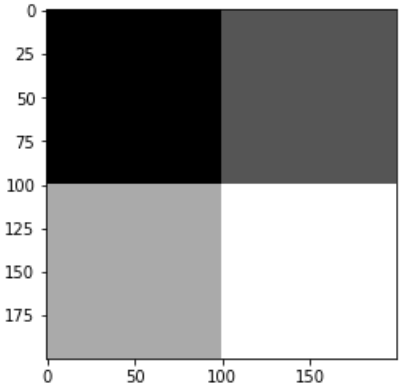
In [51]:

```
alpha = np.pi/4
a = np.array([[np.cos(2*alpha), np.sin(2*alpha)],
              [np.sin(2*alpha), -np.cos(2*alpha)]])
plot(a)
dst = linear_transformation(src, a)
plot(src)
plot(dst)
```

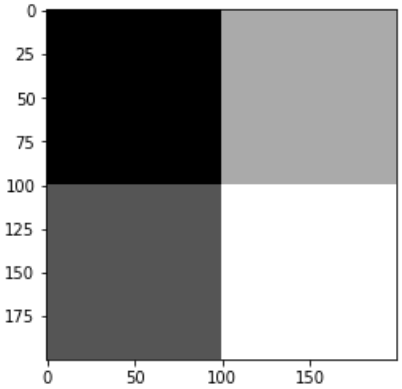
```
[[ 6.123234e-17  1.000000e+00]
 [ 1.000000e+00 -6.123234e-17]]
```




```
[[1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 [1 1 1 ... 2 2 2]
 ...
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]
 [3 3 3 ... 4 4 4]]
```




```
[[1 1 1 ... 3 3 3]
 [1 1 1 ... 3 3 3]
 [1 1 1 ... 3 3 3]
 ...
 [2 2 2 ... 4 4 4]
 [2 2 2 ... 4 4 4]
 [2 2 2 ... 4 4 4]]
```

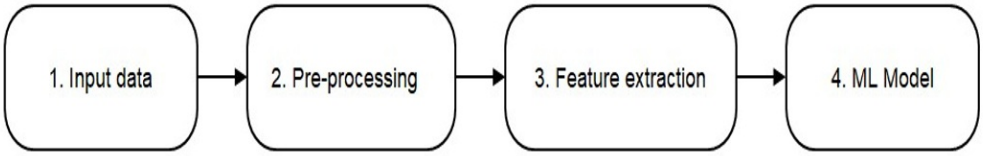


If you remember the slides from the last life

- OpenCV: offers all kinds of algorithms from **basic image processing** to **advanced computer vision.**

What is computer vision pipeline?

It is a sequence of distinct steps to **process** and **analyze image data**. Here is a conventional visual pattern recognition pipeline:



The first library to computer vision in OpenCv

OpenCV was started at Intel in 1999 by Gary Bradsky, and the first release came out in 2000. Vadim Pisarevsky joined Gary Bradsky to manage Intel's Russian software OpenCV team. In 2005, OpenCV was used on Stanley, the vehicle that won the 2005 DARPA Grand Challenge. Later, its active development continued under the support of Willow Garage with Gary Bradsky and Vadim Pisarevsky leading the project. OpenCV now supports a multitude of algorithms related to Computer Vision and Machine Learning and is expanding day by day.

OpenCV-Python OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

In [12]:

```
import numpy as np
import cv2 as cv
```

basic operations using openCV

Pyimage search is once of the best resources for learning computer vision stuff source

The following code is from Pyimage search source

In [13]:

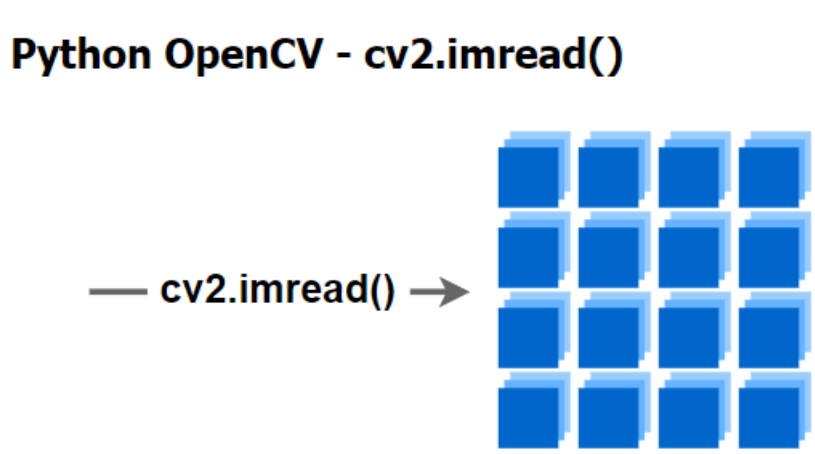
```
# When reading image with opencv it is loaded with BGR (Blue, Green, Red) => (255, 255, 255)
image = cv.imread("alhasif.png")

(h, w, d) = image.shape

print("width={}, height={}, depth={}".format(w, h, d))
```

```
width=2528, height=1615, depth=3
```

In [14]:

```
plot(image)
```

```
[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 ...

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]]
```



Syntax

```
cv2.imread("/path/to/image", flag)
```

- The flag:
  - cv2.IMREAD_COLOR: reads the image with RGB-BGR colors but no transparency channel, default selection.
  - cv2.IMREAD_GRAYSCALE: reads the image as grey image.
  - cv2.IMREAD_UNCHANGED: reads the image as is from the source. If the source image is an RGB, it loads the image into array with Red, Green and Blue channels. If the source image is ARGB, it loads the image with three color components along with the alpha or transparency channel.

In [56]:

```
# When reading image with opencv it is loaded with RGB (Red, Green, Blue) => (255, 255, 255)
image = cv.imread("alhasif.png", cv2.IMREAD_COLOR)

(h, w, d) = image.shape

print("width={}, height={}, channels={}".format(w, h, d))

plot(image)
```

```
width=2528, height=1615, channels=3
[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 ...

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]]
```

(height, width, number_of_channels)

- (h, w, d) = image.shape
- (100, 100, 3)
- the image is 400 pixels, width is 640 and 3 are three color channels (RGB) in the image
- (100, 100, 4)
- the image is 400 pixels, width is 640 and 4 are three color channels (RGBA) in the image A(Transperancey)

- JPEG, PNG, TIFF

source

In [21]:

```
image = cv.imread("alhasif.png", cv2.IMREAD_GRAYSCALE)

print(image.shape)

(h, w) = image.shape

print("width={}, height={}".format(w, h))

plot(image)
```

```
(1615, 2528)
width=2528, height=1615
[[255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 ...
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]]
```

In [59]:

```
image = cv.imread("alhasif.png", cv2.IMREAD_UNCHANGED)

print(image.shape)

(h, w, d) = image.shape

print("width={}, height={}, channels={}".format(w, h, d))

# plot(image)

plt.imshow(image)
plt.show()
```

```
(1615, 2528, 3)
width=2528, height=1615, channels=3
```

Color Space

Various color spaces such as RGB, BGR, HSV can be mutually converted using OpenCV

In [60]:

```
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
print(image.shape)

(h, w, d) = image.shape

print("width={}, height={}, channels={}".format(w, h, d))

# plot(image)

plot(image)
```

```
(1615, 2528, 3)
width=2528, height=1615, channels=3
[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 ...

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]]
```

Resize Image

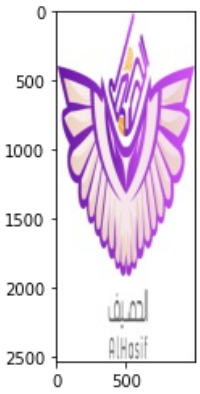
```
cv2.resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])
```

- src: is the source, original or input image in the form of numpy array
- dsize: is the desired size of the output image, given as tuple
- fx: is the scaling factor along X-axis or Horizontal axis
- fy: is the scaling factor along Y-axis or Vertical axis
- interpolation: Based on the interpolation technique selected, respective algorithm is used. You can think interpolation as a method that decides which pixel gets which value based on its neighboring pixels and the scale at which the image is being resized.
  - INTER_NEAREST
  - INTER_LINEAR
  - INTER_AREA
  - INTER_CUBIC
  - INTER_LANCZOS4

In [63]:

```
output = cv2.resize(image, (1000, 2528))

(h, w, d) = output.shape

print("width={}, height={}, channels={}".format(w, h, d))

plot(output)
```

```
width=1000, height=2528, channels=3
[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 ...

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]]
```



In [64]:

```
output = cv2.resize(image, (1615, 1000))

(h, w, d) = output.shape

print("width={}, height={}, channels={}".format(w, h, d))

plot(output)
```

```
width=1615, height=1000, channels=3
[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 ...

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]]
```

In [69]:

```
# be careful => dsize = (width, height)
output = cv2.resize(image, (image.shape[1]//2, image.shape[0]//2))

(h, w, d) = output.shape

print("width={}, height={}, channels={}".format(w, h, d))

plot(output)
```

```
width=1264, height=807, channels=3
[[[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 ...

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]

 [[255 255 255]
  [255 255 255]
  [255 255 255]
  ...
  [255 255 255]
  [255 255 255]
  [255 255 255]]]
```

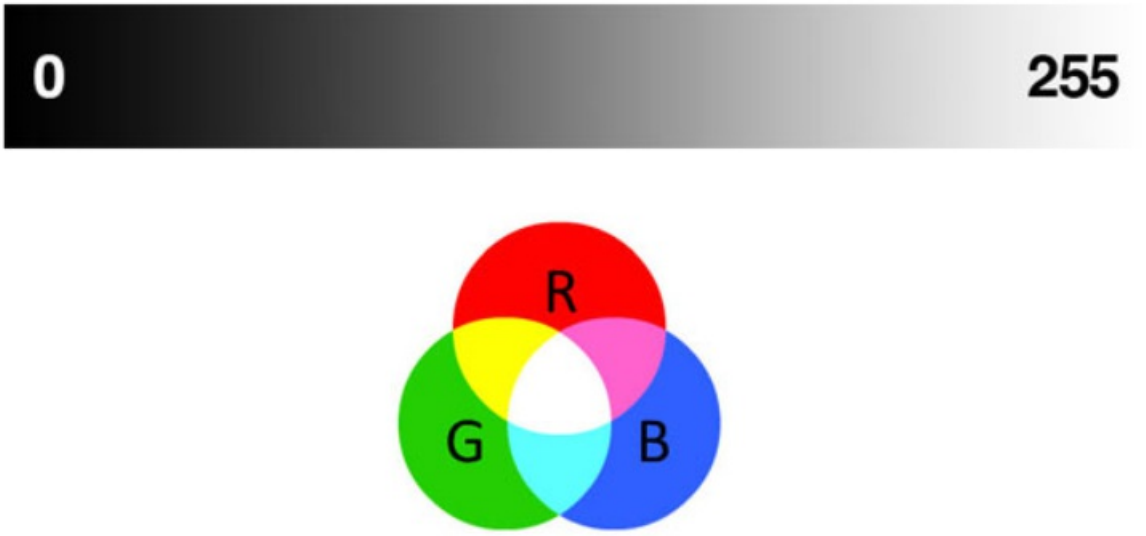


Figure 2: *Top*: grayscale gradient where brighter pixels are closer to 255 and darker pixels are closer to 0. *Bottom*: RGB venn diagram where brighter pixels are closer to the center.

What is a pixel

All images consist of pixels which are the raw building blocks of images. Images are made of pixels in a grid. A 640 x 480 image has 640 columns (the width) and 480 rows (the height). There are 640 * 480 = 307200 pixels in an image with those dimensions.

In [74]:

```
image = cv.imread("alhasif.png", cv2.IMREAD_UNCHANGED)
plt.imshow(image)
plt.show()
```

In [75]:

```
(B, G, R) = image[100, 50]
print(B, G, R)
```

```
255 255 255
```

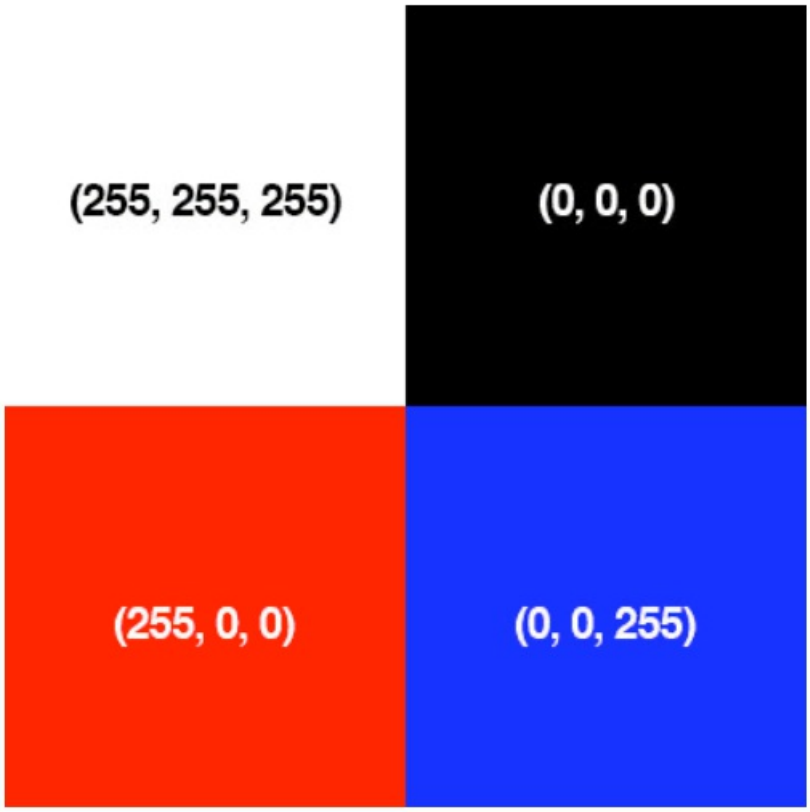
In [76]:

```
for i in range(1, 500):
    for j in range(1,500):
        image[i, j] = (0,0,0)
plt.imshow(image)
plt.show()
```

Pixel Can Be

- Black: (0, 0, 0)
- White: (255, 255, 255)
- Red: (255, 0, 0)
- Green: (0, 255, 0)
- Blue: (0, 0, 255)
- Aqua: (0, 255, 255)
- Fuchsia: (255, 0, 255)
- Maroon: (128, 0, 0)
- Navy: (0, 0, 128)
- Olive: (128, 128, 0)
- Purple: (128, 0, 128)
- Teal: (0, 128, 128)
- Yellow: (255, 255, 0)

source



In [79]:

```
for i in range(1, 500):
    for j in range(1,500):
        image[i, j] = (255, 255, 0)
plt.imshow(image)
plt.show()
```

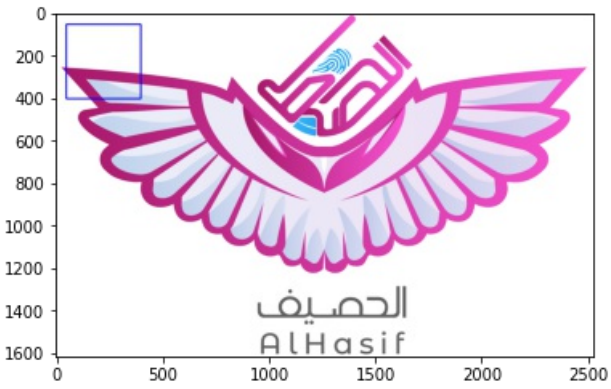
Drawing

In [148]:

```
image = cv.imread("alhasif.png", cv2.IMREAD_UNCHANGED)
plt.imshow(image)
plt.show()
```

In [149]:

```
output = image.copy()
cv2.rectangle(output, (50, 50), (400, 400), (0, 0, 255), 5)
plt.imshow(output)
plt.show()
```




```
rectangle(image, top_left_pixel_location, bottom_right_pixel_location, rectangle_color, thickness)
```

In [190]:

```
output = image.copy()
# if border thickness is -1 then it is called filling
cv2.rectangle(output, (50, 50), (400, 400), (0, 0, 255), -1)
plt.imshow(output)
plt.show()
```

In [150]:

```
output = image.copy()
cv2.rectangle(output, (900, 10), (1700, 600), (255, 0, 0), 10)
plt.imshow(output)
plt.show()
```

In [151]:

```
output = image.copy()
cv2.circle(output, (1300, 200), 100, (0, 0, 255), -1)
plt.imshow(output)
plt.show()
```


```
cv2.circle(image, center_of_circle, circle_radius, circle_color, thickness)
```

In [153]:

```
output = image.copy()
cv2.line(output, (60, 20), (500, 600), (0, 0, 255), 5)
plt.imshow(output)
plt.show()
```


```
cv2.line(image, top_left_pixel_location, bottom_right_pixel_location, line_color, thickness)
```

In [174]:

```
output = image.copy()
cv2.putText(output, "GDSC UofK", (1150, 1300), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 0, 0), 7)
plt.imshow(output)
plt.show()
```


```
cv2.putText(img, text, starting_point, font_type, font_scale, font_color, thickness)
```

In [188]:

```
output = image.copy()
cv2.putText(output, "GDSC UofK", (50, 1100), cv2.FONT_HERSHEY_SIMPLEX, 5, (0, 0, 255), 7)
cv2.putText(output, "Ibrahim Algadi", (50, 1300), cv2.FONT_HERSHEY_SIMPLEX, 5, (255, 0, 0), 7)
cv2.putText(output, "Sudan", (50, 1500), cv2.FONT_HERSHEY_SIMPLEX, 5, (0, 255, 0), 7)
plt.imshow(output)
plt.show()
```

I want to carify each function input so it is clear for people what the input does ...

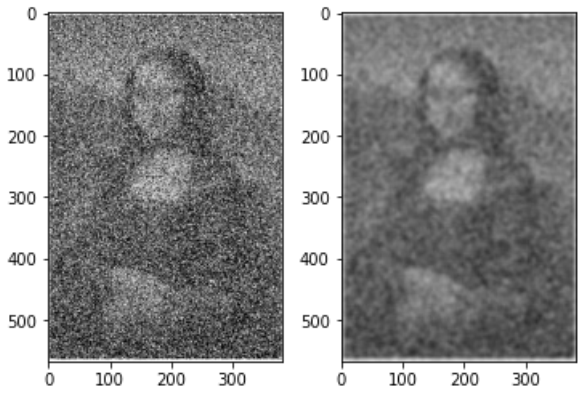
Transformation

Blur

In [140]:

```
image = cv.imread("noise.png", cv2.IMREAD_UNCHANGED)
blurred = cv2.GaussianBlur(image, (11, 11),cv2.BORDER_DEFAULT)
```

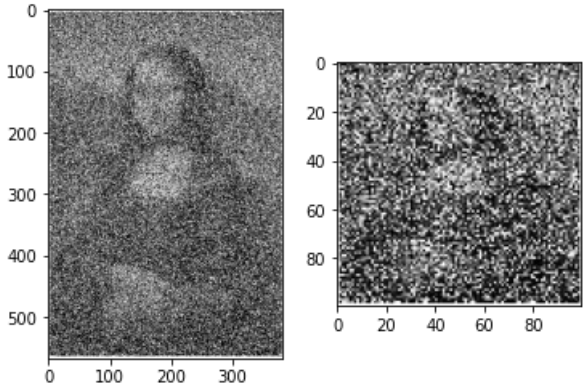
```
# using Subplot matlab style
# https://matplotlib.org/stable/tutorials/introductory/pyplot.html

plt.subplot(121)
plt.imshow(image)
plt.subplot(122)
plt.imshow(blurred)
plt.show()
```



Resizing

In [139]:

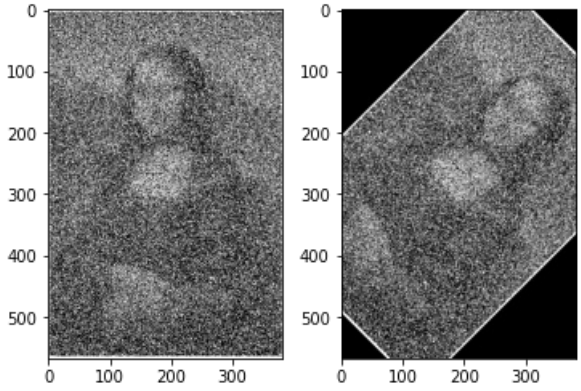
```
image = cv.imread("noise.png", cv2.IMREAD_UNCHANGED)
result = cv2.resize(image, (100, 100))
# using Subplot matlab style
plt.subplot(121)
plt.imshow(image)
plt.subplot(122)
plt.imshow(result)
plt.show()
```



Rotation

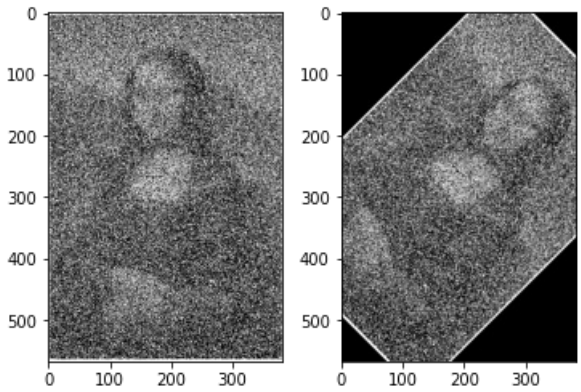
In [141]:

```
image = cv.imread("noise.png", cv2.IMREAD_UNCHANGED)

(h, w, d) = image.shape

# get the image center
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, -45, 1.0)

result = cv2.warpAffine(image, M, (w, h))
# using Subplot matlab style
plt.subplot(121)
plt.imshow(image)
plt.subplot(122)
plt.imshow(result)
plt.show()
```



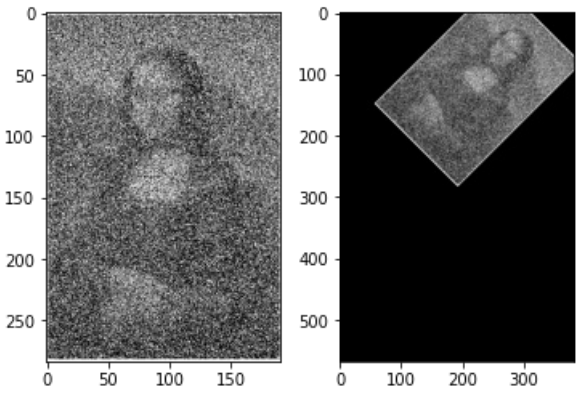
In [144]:

```
image = cv.imread("noise.png", cv2.IMREAD_UNCHANGED)

(h, w, d) = image.shape

# get the image center
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, -45, 1.0)
# Affine Transformation
result = cv2.warpAffine(image, M, (w, h))

# using Subplot matlab style
plt.subplot(121)
plt.imshow(image)
plt.subplot(122)
plt.imshow(result)
plt.show()
```



In [146]:

```
image = cv.imread("noise.png", cv2.IMREAD_UNCHANGED)

(h, w, d) = image.shape

resized_image = cv2.resize(image, (w // 2, h // 2))

# get the image center
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, -45, 1.0)

# Affine Transformation
result = cv2.warpAffine(resized_image, M, (w, h))

# using Subplot matlab style
plt.subplot(121)
plt.imshow(resized_image)
plt.subplot(122)
plt.imshow(result)
plt.show()
```



Ready Made detection models

http://www.worldlicenseplates.com/world/AF_SUDA.html

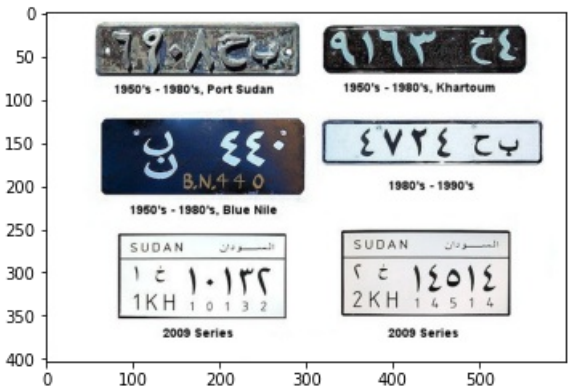
In [286]:

```
image = cv.imread("AF_SUDA_GI.jpg", cv2.IMREAD_UNCHANGED)
print(image.shape)
```

```
(403, 600, 3)
```

In [287]:

```
plt.imshow(image)
plt.show()
```



Pytesseract OCR

```
pip install pytesseract
```

In [288]:

```
import pytesseract
```

In [289]:

```
print(pytesseract.image_to_string(image))
```

```
1950's - 1980's, Khartoum

1950's - 1980's, Blue Nile



SUDAN SUDAN
hoe Ge 6
aT] [bees



2009 Series 2009 Series
```

In [290]:

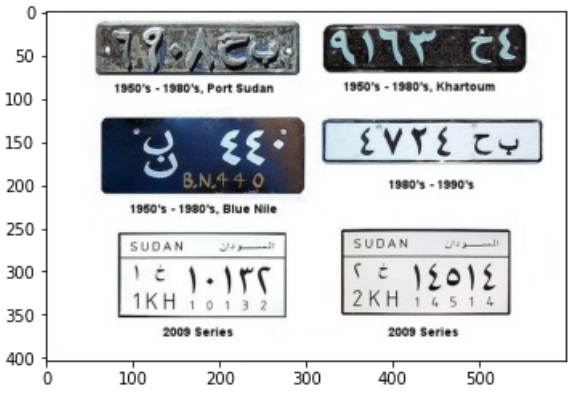
```
print(pytesseract.image_to_string(image, lang='Script/Arabic'))
```

```
1950's - 1980's, Khartoum

1950's - 1980's, Blue Nile


SUDAN SUDAN
ê € ۵
E lk


2009 Series 2009 Series
```

In [291]:

```
(h, w, _) = image.shape
cropped = image[:h, :w]
print(cropped.shape)
plt.imshow(cropped)
plt.show()
```

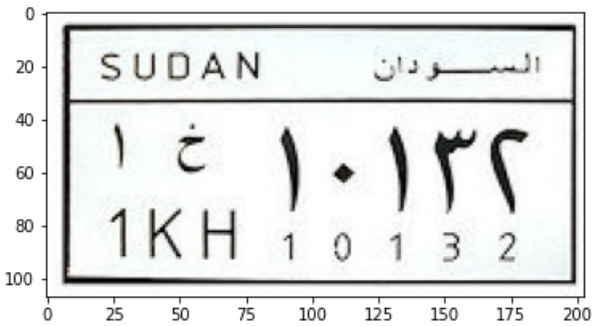
```
(403, 600, 3)
```



In [292]:

```
(h, w, _) = image.shape
cropped = image[h-152:h-45, 77:w-320]
print(cropped.shape)
plt.imshow(cropped)
plt.show()
```

```
(107, 203, 3)
```



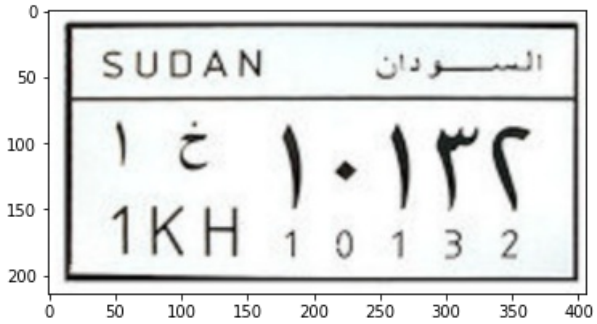
In [293]:

```
print(pytesseract.image_to_string(cropped))
```

In [294]:

```
output = cv2.resize(cropped, None, (403, 600), fx=2, fy=2, interpolation=cv2.INTER_LINEAR)
```

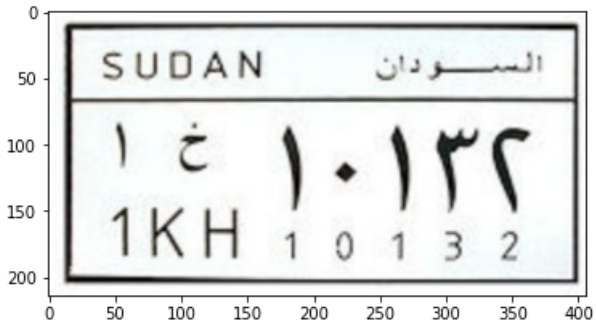
```
plt.imshow(output)
plt.show()

print(pytesseract.image_to_string(output, lang='Script/Arabic'))
```




```
ABN

1KH 1 o
```

In [295]:

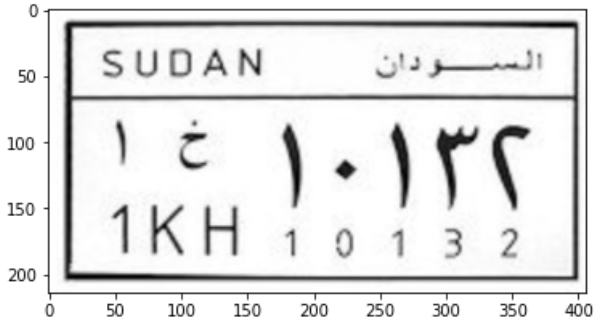
```
output = cv2.resize(cropped, None, (403, 600), fx=2, fy=2, interpolation=cv2.INTER_LINEAR)

blurred = cv2.GaussianBlur(output, (1, 1),cv2.BORDER_DEFAULT)

plt.imshow(blurred)
plt.show()

print(pytesseract.image_to_string(blurred, lang='Script/Arabic'))
```




```
ABN

1KH 1 o
```

In [296]:

```
output = cv2.resize(cropped, None, (403, 600), fx=2, fy=2, interpolation=cv2.INTER_LINEAR)

# blurred = cv2.GaussianBlur(output, (1, 1),cv2.BORDER_DEFAULT)

grey_scale = cv2.cvtColor(output, cv2.COLOR_BGR2GRAY)

plt.imshow(grey_scale, cmap='gray')
plt.show()

print(pytesseract.image_to_string(grey_scale, config="--psm 3", lang='script/Arabic'))
```

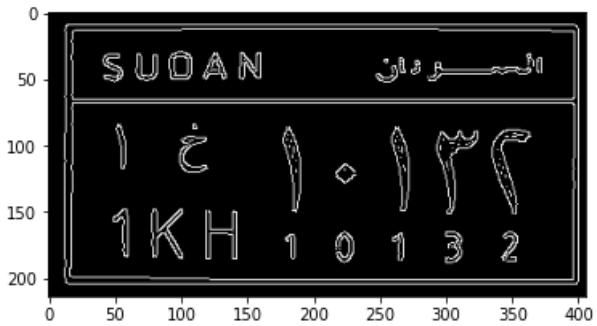


AB

1KH 1 o

In [297]:

```
# edge

edge_bin = cv2.Canny(grey_scale, 100, 100)

plt.imshow(edge_bin, cmap='gray')
plt.show()

print(pytesseract.image_to_string(edge_bin, config="--psm 3", lang='script/Arabic'))
```



اسر

٢

| SUOAN

aR

0

In [298]:

```
# Thresholding Image

#thresholding the image to a binary image
thresh,img_bin = cv2.threshold(edge_bin,128,255,cv2.THRESH_BINARY |cv2.THRESH_OTSU)

#inverting the image
img_bin = 255-img_bin
```

```
print(pytesseract.image_to_string(img_bin, config="--psm 3", lang='script/Arabic'))

#Plotting the image to see the output
plotting = plt.imshow(img_bin,cmap='gray')
plt.show()
```

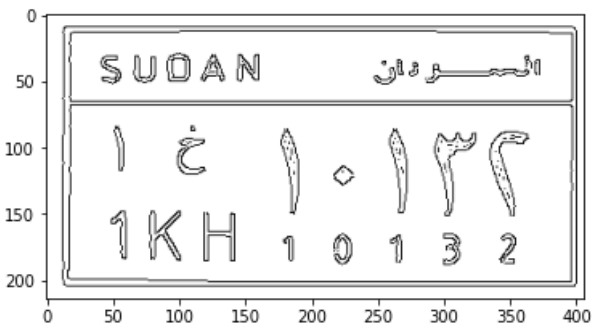
اسر

٢

| SUOAN

aR

0

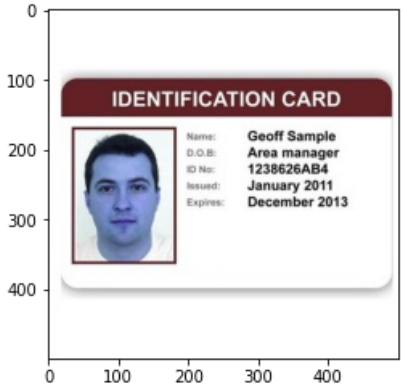


ID card Image

https://www.google.com/search?q=Id+card&tbm=isch&ved=2ahUKEwj7tLqprdPzAhUOwIUKHZlgB4YQ2-cCegQIABAA&oq=Id+card&gs_lcp=CgNpbWcQAzIHCAAQsQMQQzIECAAQQzIICAAQgAQQsQMyBQgAEIAEMgUIABCABDIFCAAQgAQyBQgAAFoAHAAeACAAYgCiAHwDZIBBTAuMy41mAEAoAEBqgELZ3dzLXdpei1pbWfAAQE&sclient=img&ei=EBdtYfvsGo6AlwSZwZ2wCA&bih=722&biv

In [299]:

```
image = cv.imread("id-card.jpg", cv2.IMREAD_UNCHANGED)
plt.imshow(image)
plt.show()
print(image.shape)
card_text = pytesseract.image_to_string(image, lang='Script/Arabic')
print(card_text)
```




```
(500, 500, 3)
IDENTIFICATION CARD

Name: Geoff Sample
D.O.B: Area manager
ID No: 1238626AB4
Issued: January 2011
Expires: December 2013
```

In [300]:

```
card_text.split('\n')
```

Out[300]:

```
['IDENTIFICATION CARD',
 '',
 'Name: Geoff Sample',
 'D.O.B: Area manager',
 'ID No: 1238626AB4',
 'Issued: January 2011',
 'Expires: December 2013',
 '\x0c']
```

In [301]:

```
for line in card_text.split('\n'):
    print(line)
```

```
IDENTIFICATION CARD

Name: Geoff Sample
D.O.B: Area manager
ID No: 1238626AB4
Issued: January 2011
Expires: December 2013
```

In [302]:

```
for line in card_text.split('\n'):
    if 'ID' in line:
        print(line)
```

```
IDENTIFICATION CARD
ID No: 1238626AB4
```

In [303]:

```
for line in card_text.split('\n'):
    if 'ID No' in line:
        print(line)
```

```
ID No: 1238626AB4
```

In [304]:

```
for line in card_text.split('\n'):
    if 'ID No' in line:
        print(line.replace('ID No: ', ''))
```

```
1238626AB4
```

Face Detection

using pretrained models

https://www.analyticsvidhya.com/blog/2018/07/top-10-pretrained-models-get-started-deep-learning-part-1-computer-vision/

https://towardsdatascience.com/darkeras-execute-yolov3-yolov4-object-detection-on-keras-with-darknet-pre-trained-weights-5e8428b959e2

https://learnopencv.com/pytorch-for-beginners-image-classification-using-pre-trained-models/

Haarcascade => https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08

So what is Haar Cascade?

It is an Object Detection Algorithm used to identify faces in an image or a real time video. The algorithm uses edge or line detection features proposed by Viola and Jones in their research paper "Rapid Object Detection using a Boosted Cascade of Simple Features" published in 2001. The algorithm is given a lot of positive images consisting of faces, and a lot of negative images not consisting of any face to train on them.

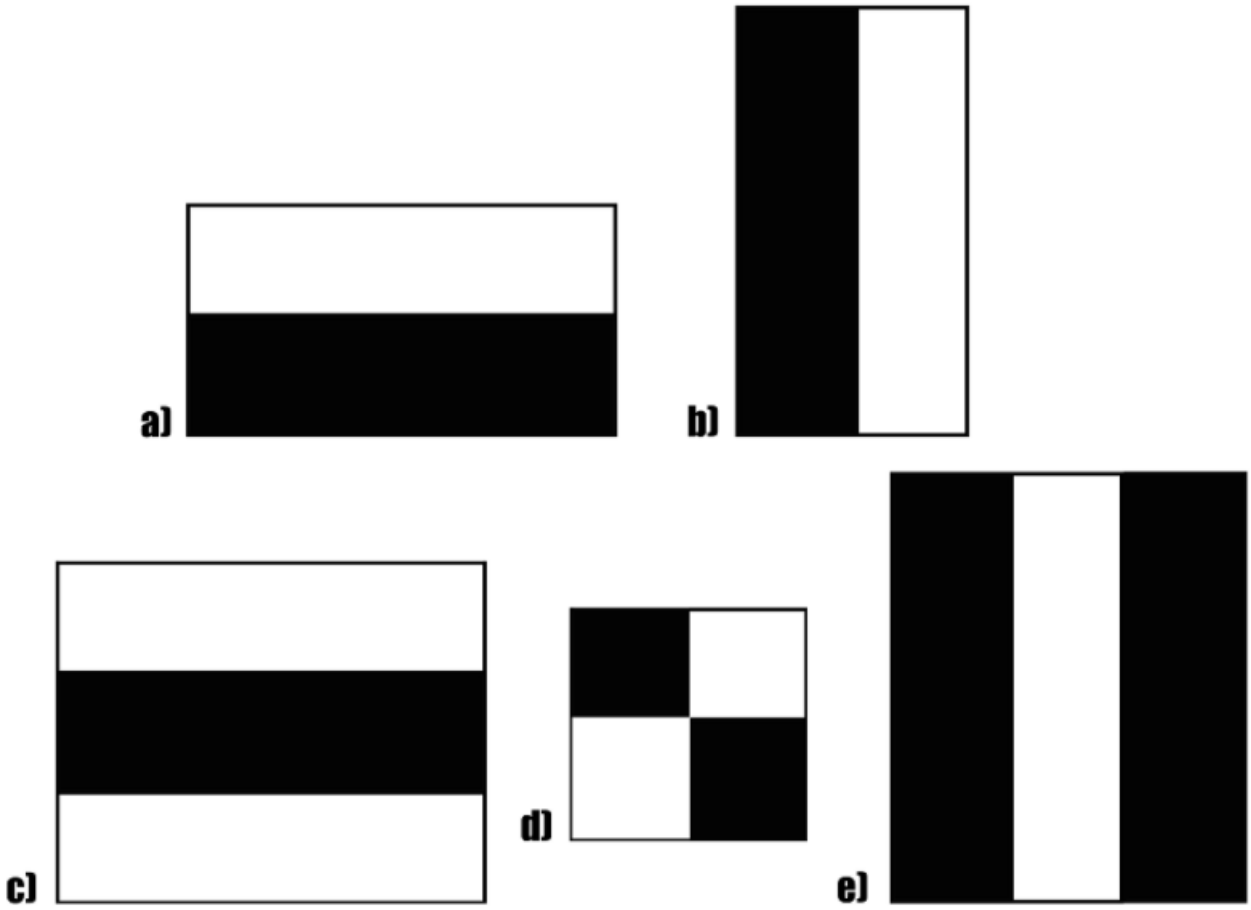


Fig. A sample of Haar features used in the Original Research Paper published by Viola and Jones.

https://github.com/opencv/opencv/tree/master/data/haarcascades

In [329]:

```
import cv2
```

In [341]:

```
# Read the input image
img = cv2.imread('sudan_famus/2.jpg')

# Convert into grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

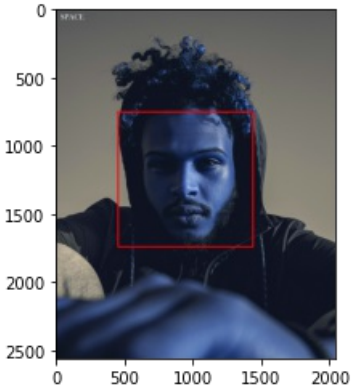
In [342]:

```
# Load the cascade
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
# Detect faces
faces = face_cascade.detectMultiScale(gray, 1.1, 4)

print(faces)
```

```
[[457 753 988 988]]
```

In [343]:

```
# Draw rectangle around the faces
tmep_image = img.copy()


for (x, y, w, h) in faces:
    cv2.rectangle(tmep_image, (x, y), (x+w, y+h), (255, 0, 0), 10)

    # Display the output
plt.imshow(tmep_image)
plt.show()
```



In [344]:

```
# Read the input image
img = cv2.imread('sudan_famus/1.jpg')

# Convert into grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Load the cascade
right_image_cascade = cv2.CascadeClassifier('haarcascade_righteye_2splits.xml')
# Detect faces
eyes = right_image_cascade.detectMultiScale(gray, 1.1, 4)

print(eyes)

# Draw rectangle
tmep_image = img.copy()

for (x, y, w, h) in eyes:
    cv2.rectangle(tmep_image, (x, y), (x+w, y+h), (255, 0, 0), 10)

    # Display the output
plt.imshow(tmep_image)
plt.show()
```

```
[[578 205  52  52]]
```

Face Detection

https://google.github.io/mediapipe/

mediapipe

```
pip install mediapipe
```

In [43]:

```
import cv2
import mediapipe as mp
import matplotlib as mpl
import matplotlib.pyplot as plt
```

In [44]:

```
mp_face_detection = mp.solutions.face_detection
```

In [45]:

```
mp_drawing = mp.solutions.drawing_utils
```

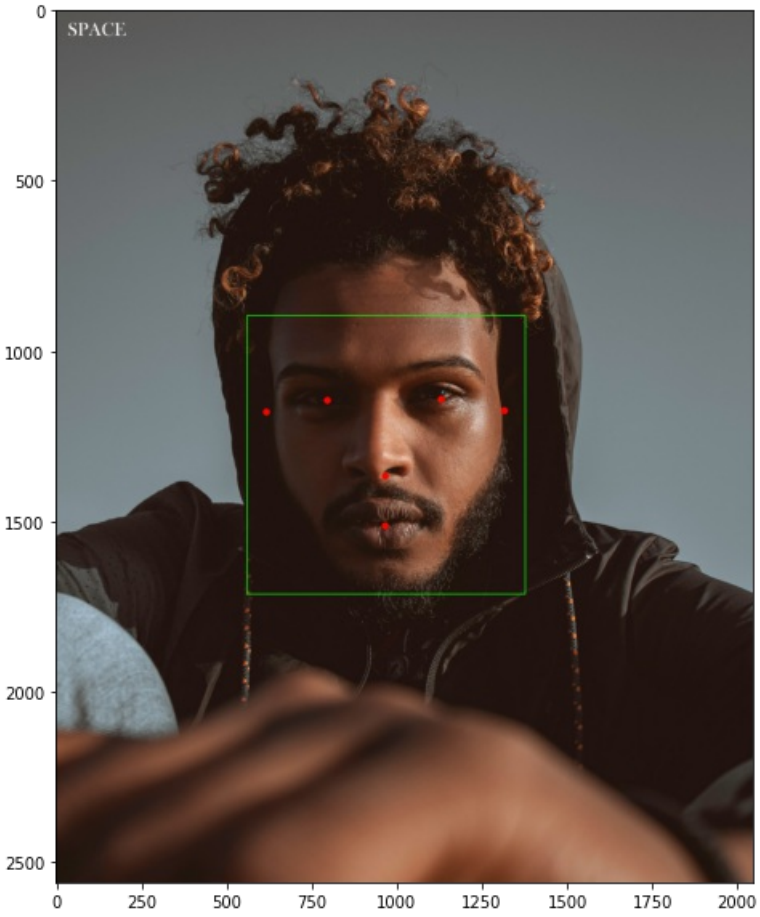
In [53]:

```
# Read the input image
img = cv2.imread('sudan_famus/2.jpg')

# Convert into grayscale
rgb_image = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

In [60]:

```
with mp_face_detection.FaceDetection() as face_detection:
    # print(face_detection)
    results = face_detection.process(rgb_image)
    # print(results.detections)
    for detection in results.detections:
        print('Nose tip:')
        print(
            mp_face_detection.get_key_point(
                detection, mp_face_detection.FaceKeyPoint.NOSE_TIP
            )
        )

        # copy image
        image_copy = rgb_image.copy()

        mp_drawing.draw_detection(
            image_copy,
            detection,
            keypoint_drawing_spec=mp.solutions.drawing_utils.DrawingSpec(
                color=(255,0,0),
                thickness=10,
                circle_radius=5
            )
        )


        # show image
        plt.figure(figsize=(10,10))
        plt.imshow(image_copy)
        plt.show()
```

```
Nose tip:
x: 0.4726942777633667
y: 0.5342410206794739
```



In [41]:

```
dir(mp_face_detection.FaceKeyPoint)
```

Out[41]:

```
['LEFT_EAR_TRAGION',
 'LEFT_EYE',
 'MOUTH_CENTER',
 'NOSE_TIP',
 'RIGHT_EAR_TRAGION',
 'RIGHT_EYE',
 '__class__',
 '__doc__',
 '__members__',
 '__module__']
```

In [62]:

```
mp_drawing = mp.solutions.drawing_utils
mp_face_mesh = mp.solutions.face_mesh
```

In [72]:

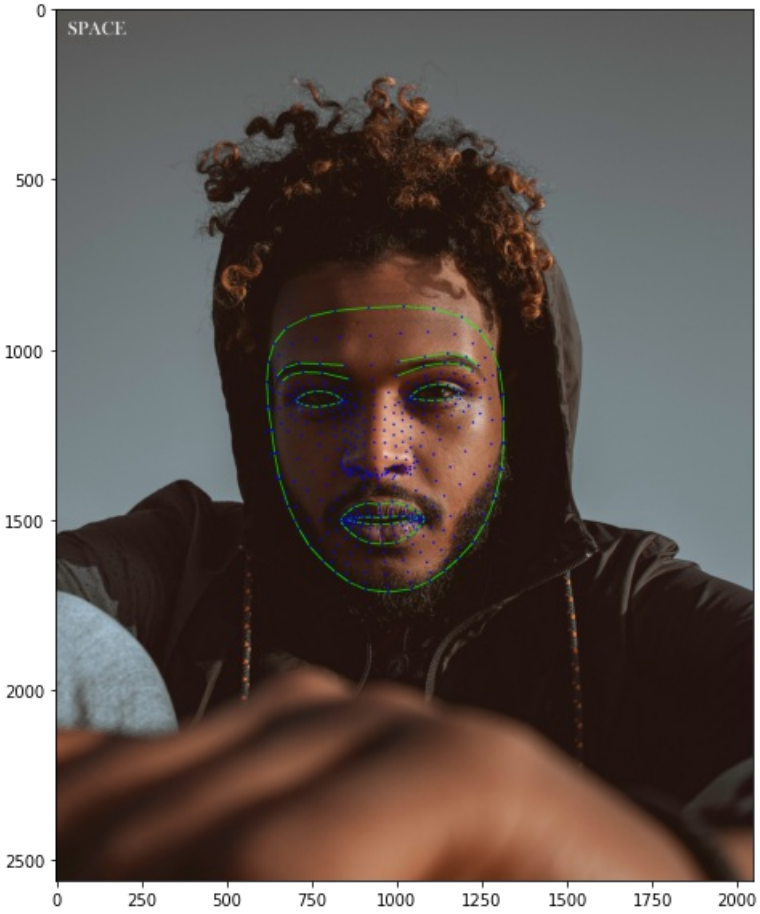
```
dir(mp_face_mesh)
```

Out[72]:

```
['BINARYPB_FILE_PATH',
 'FACE_CONNECTIONS',
 'FaceMesh',
 'NamedTuple',
 'SolutionBase',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'association_calculator_pb2',
 'constant_side_packet_calculator_pb2',
 'detections_to_rects_calculator_pb2',
 'gate_calculator_pb2',
 'image_to_tensor_calculator_pb2',
 'inference_calculator_pb2',
 'logic_calculator_pb2',
 'non_max_suppression_calculator_pb2',
 'np',
 'rect_transformation_calculator_pb2',
 'split_vector_calculator_pb2',
 'ssd_anchors_calculator_pb2',
 'tensors_to_classification_calculator_pb2',
 'tensors_to_detections_calculator_pb2',
 'tensors_to_landmarks_calculator_pb2',
 'thresholding_calculator_pb2']
```

In [73]:

```
with mp_face_mesh.FaceMesh() as face_mesh:

    results = face_mesh.process(rgb_image)

    # Print and draw face mesh landmarks on the image.

    # print(results.multi_face_landmarks)

    # copy image
    image_copy = rgb_image.copy()

    for face_landmarks in results.multi_face_landmarks:
        # print('face_landmarks:', face_landmarks)
        mp_drawing.draw_landmarks(
            image=image_copy,
            landmark_list=face_landmarks,
            connections=mp_face_mesh.FACE_CONNECTIONS
        )

    # show image
    plt.figure(figsize=(10,10))
    plt.imshow(image_copy)
    plt.show()
```



Object Detection

PyTorch

```
pip install torch torchvision
pip install opencv-contrib-python
```

https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

https://www.pyimagesearch.com/2021/08/02/pytorch-object-detection-with-pre-trained-networks/

In [26]:

```
# import the necessary packages
from torchvision import models
import numpy as np
import torch
import cv2
from PIL import Image
```

In [27]:

```
img = Image.open('dog.jpg')
img
```

Out[27]:

In [28]:

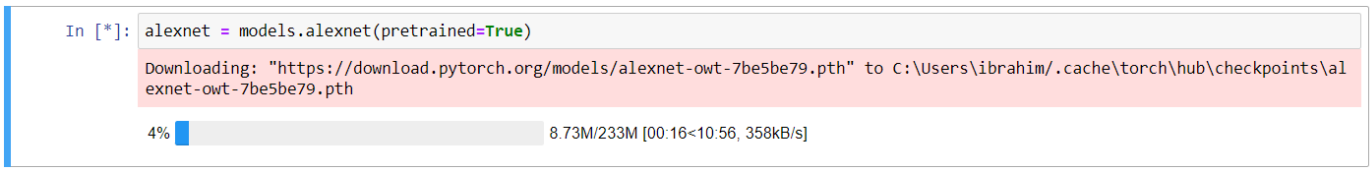
```
dir(models)
```

Out[28]:

```
['AlexNet',
 'DenseNet',
 'GoogLeNet',
 'GoogLeNetOutputs',
 'Inception3',
 'InceptionOutputs',
 'MNASNet',
 'MobileNetV2',
 'MobileNetV3',
 'ResNet',
 'ShuffleNetV2',
 'SqueezeNet',
 'VGG',
 '_GoogLeNetOutputs',
 '_InceptionOutputs',
 '__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__path__',
 '__spec__',
 '_utils',
 'alexnet',
 'densenet',
 'densenet121',
 'densenet161',
 'densenet169',
 'densenet201',
 'detection',
 'googlenet',
 'inception',
 'inception_v3',
 'mnasnet',
 'mnasnet0_5',
```

```
'mnasnet0_75',
'mnasnet1_0',
'mnasnet1_3',
'mobilenet',
'mobilenet_v2',
'mobilenet_v3_large',
'mobilenet_v3_small',
'mobilenetv2',
'mobilenetv3',
'quantization',
'resnet',
'resnet101',
'resnet152',
'resnet18',
'resnet34',
'resnet50',
'resnext101_32x8d',
'resnext50_32x4d',
'segmentation',
'shufflenet_v2_x0_5',
'shufflenet_v2_x1_0',
'shufflenet_v2_x1_5',
'shufflenet_v2_x2_0',
'shufflenetv2',
'squeezenet',
'squeezenet1_0',
'squeezenet1_1',
'utils',
'vgg',
'vgg11',
'vgg11_bn',
'vgg13',
'vgg13_bn',
'vgg16',
'vgg16_bn',
'vgg19',
'vgg19_bn',
'video',
'wide_resnet101_2',
'wide_resnet50_2']
```

AlexNet. It is one of the early breakthrough networks in Image Recognition.



In [4]:

```
alexnet = models.alexnet(pretrained=True)
```

```
Downloading: "https://download.pytorch.org/models/alexnet-owt-7be5be79.pth" to C:\Users\ibrahim/.cache\torch\hub\checkpoints\alexnet-owt-7be5be79.pth
```

In [29]:

```
print(alexnet)
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

In [30]:

```
from torchvision import transforms

transform = transforms.Compose([            #[1]
    transforms.Resize(256),                    #[2]
    transforms.CenterCrop(224),                #[3]
    transforms.ToTensor(),                     #[4]
    transforms.Normalize(                      #[5]
        mean=[0.485, 0.456, 0.406],                #[6]
        std=[0.229, 0.224, 0.225]                  #[7]
    )
])
```

Next, pre-process the image and prepare a batch to be passed through the network.

In [31]:

```
img_t = transform(img)
batch_t = torch.unsqueeze(img_t, 0)

# Finally, it's time to use the pre-trained model to see what the model thinks the image is.
# First, we need to put our model in eval mode.

alexnet.eval()

out = alexnet(batch_t)
print(out.shape)
```

```
torch.Size([1, 1000])
```

output vector out with 1000 elements? We still haven't got the class (or label) of the image. For this, we will first read and store the labels from a text file having a list of all the 1000 labels

https://gist.github.com/ageitgey/4e1342c10a71981d0b491e1b8227328b

In [32]:

```
with open('imagenet_classes.txt') as f:
    classes = [line.strip() for line in f.readlines()]

classes
```

Out[32]:

```
['0, tench',
 '1, goldfish',
 '2, great_white_shark',
 '3, tiger_shark',
 '4, hammerhead',
 '5, electric_ray',
 '6, stingray',
 '7, cock'
```

'7, cock',
'8, hen',
'9, ostrich',
'10, brambling',
'11, goldfinch',
'12, house_finch',
'13, junco',
'14, indigo_bunting',
'15, robin',
'16, bulbul',
'17, jay',
'18, magpie',
'19, chickadee',
'20, water_ouzel',
'21, kite',
'22, bald_eagle',
'23, vulture',
'24, great_grey_owl',
'25, European_fire_salamander',
'26, common_newt',
'27, eft',
'28, spotted_salamander',
'29, axolotl',
'30, bullfrog',
'31, tree_frog',
'32, tailed_frog',
'33, loggerhead',
'34, leatherback_turtle',
'35, mud_turtle',
'36, terrapin',
'37, box_turtle',
'38, banded_gecko',
'39, common_iguana',
'40, American_chameleon',
'41, whiptail',
'42, agama',
'43, frilled_lizard',
'44, alligator_lizard',
'45, Gila_monster',
'46, green_lizard',
'47, African_chameleon',
'48, Komodo_dragon',
'49, African_crocodile',
'50, American_alligator',
'51, triceratops',
'52, thunder_snake',
'53, ringneck_snake',
'54, hognose_snake',
'55, green_snake',
'56, king_snake',
'57, garter_snake',
'58, water_snake',
'59, vine_snake',
'60, night_snake',
'61, boa_constrictor',
'62, rock_python',
'63, Indian_cobra',
'64, green_mamba',
'65, sea_snake',
'66, horned_viper',
'67, diamondback',
'68, sidewinder',
'69, trilobite',
'70, harvestman',
'71, scorpion',
'72, black_and_gold_garden_spider',
'73, barn_spider',
'74, garden_spider',
'75, black_widow',
'76, tarantula',
'77, wolf_spider',
'78, tick',
'79, centipede',
'80, black_grouse',
'81, ptarmigan',
'82, ruffed_grouse',
'83, prairie_chicken',
'84, peacock',

'84, peacock',
'85, quail',
'86, partridge',
'87, African_grey',
'88, macaw',
'89, sulphur-crested_cockatoo',
'90, lorikeet',
'91, coucal',
'92, bee_eater',
'93, hornbill',
'94, hummingbird',
'95, jacamar',
'96, toucan',
'97, drake',
'98, red-breasted_merganser',
'99, goose',
'100, black_swan',
'101, tusker',
'102, echidna',
'103, platypus',
'104, wallaby',
'105, koala',
'106, wombat',
'107, jellyfish',
'108, sea_anemone',
'109, brain_coral',
'110, flatworm',
'111, nematode',
'112, conch',
'113, snail',
'114, slug',
'115, sea_slug',
'116, chiton',
'117, chambered_nautilus',
'118, Dungeness_crab',
'119, rock_crab',
'120, fiddler_crab',
'121, king_crab',
'122, American_lobster',
'123, spiny_lobster',
'124, crayfish',
'125, hermit_crab',
'126, isopod',
'127, white_stork',
'128, black_stork',
'129, spoonbill',
'130, flamingo',
'131, little_blue_heron',
'132, American_egret',
'133, bittern',
'134, crane',
'135, limpkin',
'136, European_gallinule',
'137, American_coot',
'138, bustard',
'139, ruddy_turnstone',
'140, red-backed_sandpiper',
'141, redshank',
'142, dowitcher',
'143, oystercatcher',
'144, pelican',
'145, king_penguin',
'146, albatross',
'147, grey_whale',
'148, killer_whale',
'149, dugong',
'150, sea_lion',
'151, Chihuahua',
'152, Japanese_spaniel',
'153, Maltese_dog',
'154, Pekinese',
'155, Shih-Tzu',
'156, Blenheim_spaniel',
'157, papillon',
'158, toy_terrier',
'159, Rhodesian_ridgeback',
'160, Afghan_hound',

'161, basset',
'162, beagle',
'163, bloodhound',
'164, bluetick',
'165, black-and-tan_coonhound',
'166, Walker_hound',
'167, English_foxhound',
'168, redbone',
'169, borzoi',
'170, Irish_wolfhound',
'171, Italian_greyhound',
'172, whippet',
'173, Ibizan_hound',
'174, Norwegian_elkhound',
'175, otterhound',
'176, Saluki',
'177, Scottish_deerhound',
'178, Weimaraner',
'179, Staffordshire_bullterrier',
'180, American_Staffordshire_terrier',
'181, Bedlington_terrier',
'182, Border_terrier',
'183, Kerry_blue_terrier',
'184, Irish_terrier',
'185, Norfolk_terrier',
'186, Norwich_terrier',
'187, Yorkshire_terrier',
'188, wire-haired_fox_terrier',
'189, Lakeland_terrier',
'190, Sealyham_terrier',
'191, Airedale',
'192, cairn',
'193, Australian_terrier',
'194, Dandie_Dinmont',
'195, Boston_bull',
'196, miniature_schnauzer',
'197, giant_schnauzer',
'198, standard_schnauzer',
'199, Scotch_terrier',
'200, Tibetan_terrier',
'201, silky_terrier',
'202, soft-coated_wheaten_terrier',
'203, West_Highland_white_terrier',
'204, Lhasa',
'205, flat-coated_retriever',
'206, curly-coated_retriever',
'207, golden_retriever',
'208, Labrador_retriever',
'209, Chesapeake_Bay_retriever',
'210, German_short-haired_pointer',
'211, vizsla',
'212, English_setter',
'213, Irish_setter',
'214, Gordon_setter',
'215, Brittany_spaniel',
'216, clumber',
'217, English_springer',
'218, Welsh_springer_spaniel',
'219, cocker_spaniel',
'220, Sussex_spaniel',
'221, Irish_water_spaniel',
'222, kuvasz',
'223, schipperke',
'224, groenendael',
'225, malinois',
'226, briard',
'227, kelpie',
'228, komondor',
'229, Old_English_sheepdog',
'230, Shetland_sheepdog',
'231, collie',
'232, Border_collie',
'233, Bouvier_des_Flandres',
'234, Rottweiler',
'235, German_shepherd',
'236, Doberman',
'237, miniature_pinscher',

'238, Greater_Swiss_Mountain_dog',
'239, Bernese_mountain_dog',
'240, Appenzeller',
'241, EntleBucher',
'242, boxer',
'243, bull_mastiff',
'244, Tibetan_mastiff',
'245, French_bulldog',
'246, Great_Dane',
'247, Saint_Bernard',
'248, Eskimo_dog',
'249, malamute',
'250, Siberian_husky',
'251, dalmatian',
'252, affenpinscher',
'253, basenji',
'254, pug',
'255, Leonberg',
'256, Newfoundland',
'257, Great_Pyrenees',
'258, Samoyed',
'259, Pomeranian',
'260, chow',
'261, keeshond',
'262, Brabancon_griffon',
'263, Pembroke',
'264, Cardigan',
'265, toy_poodle',
'266, miniature_poodle',
'267, standard_poodle',
'268, Mexican_hairless',
'269, timber_wolf',
'270, white_wolf',
'271, red_wolf',
'272, coyote',
'273, dingo',
'274, dhole',
'275, African_hunting_dog',
'276, hyena',
'277, red_fox',
'278, kit_fox',
'279, Arctic_fox',
'280, grey_fox',
'281, tabby',
'282, tiger_cat',
'283, Persian_cat',
'284, Siamese_cat',
'285, Egyptian_cat',
'286, cougar',
'287, lynx',
'288, leopard',
'289, snow_leopard',
'290, jaguar',
'291, lion',
'292, tiger',
'293, cheetah',
'294, brown_bear',
'295, American_black_bear',
'296, ice_bear',
'297, sloth_bear',
'298, mongoose',
'299, meerkat',
'300, tiger_beetle',
'301, ladybug',
'302, ground_beetle',
'303, long-horned_beetle',
'304, leaf_beetle',
'305, dung_beetle',
'306, rhinoceros_beetle',
'307, weevil',
'308, fly',
'309, bee',
'310, ant',
'311, grasshopper',
'312, cricket',
'313, walking_stick',
'314, cockroach',

'315, mantis',
'316, cicada',
'317, leafhopper',
'318, lacewing',
'319, dragonfly',
'320, damselfly',
'321, admiral',
'322, ringlet',
'323, monarch',
'324, cabbage_butterfly',
'325, sulphur_butterfly',
'326, lycaenid',
'327, starfish',
'328, sea_urchin',
'329, sea_cucumber',
'330, wood_rabbit',
'331, hare',
'332, Angora',
'333, hamster',
'334, porcupine',
'335, fox_squirrel',
'336, marmot',
'337, beaver',
'338, guinea_pig',
'339, sorrel',
'340, zebra',
'341, hog',
'342, wild_boar',
'343, warthog',
'344, hippopotamus',
'345, ox',
'346, water_buffalo',
'347, bison',
'348, ram',
'349, bighorn',
'350, ibex',
'351, hartebeest',
'352, impala',
'353, gazelle',
'354, Arabian_camel',
'355, llama',
'356, weasel',
'357, mink',
'358, polecat',
'359, black-footed_ferret',
'360, otter',
'361, skunk',
'362, badger',
'363, armadillo',
'364, three-toed_sloth',
'365, orangutan',
'366, gorilla',
'367, chimpanzee',
'368, gibbon',
'369, siamang',
'370, guenon',
'371, patas',
'372, baboon',
'373, macaque',
'374, langur',
'375, colobus',
'376, proboscis_monkey',
'377, marmoset',
'378, capuchin',
'379, howler_monkey',
'380, titi',
'381, spider_monkey',
'382, squirrel_monkey',
'383, Madagascar_cat',
'384, indri',
'385, Indian_elephant',
'386, African_elephant',
'387, lesser_panda',
'388, giant_panda',
'389, barracouta',
'390, eel',
'391, coho',

'392, rock_beauty',
'393, anemone_fish',
'394, sturgeon',
'395, gar',
'396, lionfish',
'397, puffer',
'398, abacus',
'399, abaya',
'400, academic_gown',
'401, accordion',
'402, acoustic_guitar',
'403, aircraft_carrier',
'404, airliner',
'405, airship',
'406, altar',
'407, ambulance',
'408, amphibian',
'409, analog_clock',
'410, apiary',
'411, apron',
'412, ashcan',
'413, assault_rifle',
'414, backpack',
'415, bakery',
'416, balance_beam',
'417, balloon',
'418, ballpoint',
'419, Band_Aid',
'420, banjo',
'421, bannister',
'422, barbell',
'423, barber_chair',
'424, barbershop',
'425, barn',
'426, barometer',
'427, barrel',
'428, barrow',
'429, baseball',
'430, basketball',
'431, bassinet',
'432, bassoon',
'433, bathing_cap',
'434, bath_towel',
'435, bathtub',
'436, beach_wagon',
'437, beacon',
'438, beaker',
'439, bearskin',
'440, beer_bottle',
'441, beer_glass',
'442, bell_cote',
'443, bib',
'444, bicycle-built-for-two',
'445, bikini',
'446, binder',
'447, binoculars',
'448, birdhouse',
'449, boathouse',
'450, bobsled',
'451, bolo_tie',
'452, bonnet',
'453, bookcase',
'454, bookshop',
'455, bottlecap',
'456, bow',
'457, bow_tie',
'458, brass',
'459, brassiere',
'460, breakwater',
'461, breastplate',
'462, broom',
'463, bucket',
'464, buckle',
'465, bulletproof_vest',
'466, bullet_train',
'467, butcher_shop',
'468, cab',

'469, caldron',
'470, candle',
'471, cannon',
'472, canoe',
'473, can_opener',
'474, cardigan',
'475, car_mirror',
'476, carousel',
"477, carpenter's_kit",
'478, carton',
'479, car_wheel',
'480, cash_machine',
'481, cassette',
'482, cassette_player',
'483, castle',
'484, catamaran',
'485, CD_player',
'486, cello',
'487, cellular_telephone',
'488, chain',
'489, chainlink_fence',
'490, chain_mail',
'491, chain_saw',
'492, chest',
'493, chiffonier',
'494, chime',
'495, china_cabinet',
'496, Christmas_stocking',
'497, church',
'498, cinema',
'499, cleaver',
'500, cliff_dwelling',
'501, cloak',
'502, clog',
'503, cocktail_shaker',
'504, coffee_mug',
'505, coffeepot',
'506, coil',
'507, combination_lock',
'508, computer_keyboard',
'509, confectionery',
'510, container_ship',
'511, convertible',
'512, corkscrew',
'513, cornet',
'514, cowboy_boot',
'515, cowboy_hat',
'516, cradle',
'517, crane',
'518, crash_helmet',
'519, crate',
'520, crib',
'521, Crock_Pot',
'522, croquet_ball',
'523, crutch',
'524, cuirass',
'525, dam',
'526, desk',
'527, desktop_computer',
'528, dial_telephone',
'529, diaper',
'530, digital_clock',
'531, digital_watch',
'532, dining_table',
'533, dishrag',
'534, dishwasher',
'535, disk_brake',
'536, dock',
'537, dogsled',
'538, dome',
'539, doormat',
'540, drilling_platform',
'541, drum',
'542, drumstick',
'543, dumbbell',
'544, Dutch_oven',
'545, electric_fan',

'546, electric_guitar',
'547, electric_locomotive',
'548, entertainment_center',
'549, envelope',
'550, espresso_maker',
'551, face_powder',
'552, feather_boa',
'553, file',
'554, fireboat',
'555, fire_engine',
'556, fire_screen',
'557, flagpole',
'558, flute',
'559, folding_chair',
'560, football_helmet',
'561, forklift',
'562, fountain',
'563, fountain_pen',
'564, four-poster',
'565, freight_car',
'566, French_horn',
'567, frying_pan',
'568, fur_coat',
'569, garbage_truck',
'570, gasmask',
'571, gas_pump',
'572, goblet',
'573, go-kart',
'574, golf_ball',
'575, golfcart',
'576, gondola',
'577, gong',
'578, gown',
'579, grand_piano',
'580, greenhouse',
'581, grille',
'582, grocery_store',
'583, guillotine',
'584, hair_slide',
'585, hair_spray',
'586, half_track',
'587, hammer',
'588, hamper',
'589, hand_blower',
'590, hand-held_computer',
'591, handkerchief',
'592, hard_disc',
'593, harmonica',
'594, harp',
'595, harvester',
'596, hatchet',
'597, holster',
'598, home_theater',
'599, honeycomb',
'600, hook',
'601, hoopskirt',
'602, horizontal_bar',
'603, horse_cart',
'604, hourglass',
'605, iPod',
'606, iron',
"607, jack-o'-lantern",
'608, jean',
'609, jeep',
'610, jersey',
'611, jigsaw_puzzle',
'612, jinrikisha',
'613, joystick',
'614, kimono',
'615, knee_pad',
'616, knot',
'617, lab_coat',
'618, ladle',
'619, lampshade',
'620, laptop',
'621, lawn_mower',
'622, lens_cap',

'623, letter_opener',
'624, library',
'625, lifeboat',
'626, lighter',
'627, limousine',
'628, liner',
'629, lipstick',
'630, Loafer',
'631, lotion',
'632, loudspeaker',
'633, loupe',
'634, lumbermill',
'635, magnetic_compass',
'636, mailbag',
'637, mailbox',
'638, maillot',
'639, maillot',
'640, manhole_cover',
'641, maraca',
'642, marimba',
'643, mask',
'644, matchstick',
'645, maypole',
'646, maze',
'647, measuring_cup',
'648, medicine_chest',
'649, megalith',
'650, microphone',
'651, microwave',
'652, military_uniform',
'653, milk_can',
'654, minibus',
'655, miniskirt',
'656, minivan',
'657, missile',
'658, mitten',
'659, mixing_bowl',
'660, mobile_home',
'661, Model_T',
'662, modem',
'663, monastery',
'664, monitor',
'665, moped',
'666, mortar',
'667, mortarboard',
'668, mosque',
'669, mosquito_net',
'670, motor_scooter',
'671, mountain_bike',
'672, mountain_tent',
'673, mouse',
'674, mousetrap',
'675, moving_van',
'676, muzzle',
'677, nail',
'678, neck_brace',
'679, necklace',
'680, nipple',
'681, notebook',
'682, obelisk',
'683, oboe',
'684, ocarina',
'685, odometer',
'686, oil_filter',
'687, organ',
'688, oscilloscope',
'689, overskirt',
'690, oxcart',
'691, oxygen_mask',
'692, packet',
'693, paddle',
'694, paddlewheel',
'695, padlock',
'696, paintbrush',
'697, pajama',
'698, palace',
'699, panpipe',

'700, paper_towel',
'701, parachute',
'702, parallel_bars',
'703, park_bench',
'704, parking_meter',
'705, passenger_car',
'706, patio',
'707, pay-phone',
'708, pedestal',
'709, pencil_box',
'710, pencil_sharpener',
'711, perfume',
'712, Petri_dish',
'713, photocopier',
'714, pick',
'715, pickelhaube',
'716, picket_fence',
'717, pickup',
'718, pier',
'719, piggy_bank',
'720, pill_bottle',
'721, pillow',
'722, ping-pong_ball',
'723, pinwheel',
'724, pirate',
'725, pitcher',
'726, plane',
'727, planetarium',
'728, plastic_bag',
'729, plate_rack',
'730, plow',
'731, plunger',
'732, Polaroid_camera',
'733, pole',
'734, police_van',
'735, poncho',
'736, pool_table',
'737, pop_bottle',
'738, pot',
"739, potter's_wheel",
'740, power_drill',
'741, prayer_rug',
'742, printer',
'743, prison',
'744, projectile',
'745, projector',
'746, puck',
'747, punching_bag',
'748, purse',
'749, quill',
'750, quilt',
'751, racer',
'752, racket',
'753, radiator',
'754, radio',
'755, radio_telescope',
'756, rain_barrel',
'757, recreational_vehicle',
'758, reel',
'759, reflex_camera',
'760, refrigerator',
'761, remote_control',
'762, restaurant',
'763, revolver',
'764, rifle',
'765, rocking_chair',
'766, rotisserie',
'767, rubber_eraser',
'768, rugby_ball',
'769, rule',
'770, running_shoe',
'771, safe',
'772, safety_pin',
'773, saltshaker',
'774, sandal',
'775, sarong',
'776, sax',

'777, scabbard',
'778, scale',
'779, school_bus',
'780, schooner',
'781, scoreboard',
'782, screen',
'783, screw',
'784, screwdriver',
'785, seat_belt',
'786, sewing_machine',
'787, shield',
'788, shoe_shop',
'789, shoji',
'790, shopping_basket',
'791, shopping_cart',
'792, shovel',
'793, shower_cap',
'794, shower_curtain',
'795, ski',
'796, ski_mask',
'797, sleeping_bag',
'798, slide_rule',
'799, sliding_door',
'800, slot',
'801, snorkel',
'802, snowmobile',
'803, snowplow',
'804, soap_dispenser',
'805, soccer_ball',
'806, sock',
'807, solar_dish',
'808, sombrero',
'809, soup_bowl',
'810, space_bar',
'811, space_heater',
'812, space_shuttle',
'813, spatula',
'814, speedboat',
'815, spider_web',
'816, spindle',
'817, sports_car',
'818, spotlight',
'819, stage',
'820, steam_locomotive',
'821, steel_arch_bridge',
'822, steel_drum',
'823, stethoscope',
'824, stole',
'825, stone_wall',
'826, stopwatch',
'827, stove',
'828, strainer',
'829, streetcar',
'830, stretcher',
'831, studio_couch',
'832, stupa',
'833, submarine',
'834, suit',
'835, sundial',
'836, sunglass',
'837, sunglasses',
'838, sunscreen',
'839, suspension_bridge',
'840, swab',
'841, sweatshirt',
'842, swimming_trunks',
'843, swing',
'844, switch',
'845, syringe',
'846, table_lamp',
'847, tank',
'848, tape_player',
'849, teapot',
'850, teddy',
'851, television',
'852, tennis_ball',
'853, thatch',

'854, theater_curtain',
'855, thimble',
'856, thresher',
'857, throne',
'858, tile_roof',
'859, toaster',
'860, tobacco_shop',
'861, toilet_seat',
'862, torch',
'863, totem_pole',
'864, tow_truck',
'865, toyshop',
'866, tractor',
'867, trailer_truck',
'868, tray',
'869, trench_coat',
'870, tricycle',
'871, trimaran',
'872, tripod',
'873, triumphal_arch',
'874, trolleybus',
'875, trombone',
'876, tub',
'877, turnstile',
'878, typewriter_keyboard',
'879, umbrella',
'880, unicycle',
'881, upright',
'882, vacuum',
'883, vase',
'884, vault',
'885, velvet',
'886, vending_machine',
'887, vestment',
'888, viaduct',
'889, violin',
'890, volleyball',
'891, waffle_iron',
'892, wall_clock',
'893, wallet',
'894, wardrobe',
'895, warplane',
'896, washbasin',
'897, washer',
'898, water_bottle',
'899, water_jug',
'900, water_tower',
'901, whiskey_jug',
'902, whistle',
'903, wig',
'904, window_screen',
'905, window_shade',
'906, Windsor_tie',
'907, wine_bottle',
'908, wing',
'909, wok',
'910, wooden_spoon',
'911, wool',
'912, worm_fence',
'913, wreck',
'914, yawl',
'915, yurt',
'916, web_site',
'917, comic_book',
'918, crossword_puzzle',
'919, street_sign',
'920, traffic_light',
'921, book_jacket',
'922, menu',
'923, plate',
'924, guacamole',
'925, consomme',
'926, hot_pot',
'927, trifle',
'928, ice_cream',
'929, ice_lolly',
'930, French_loaf',

```
'931, bagel',
'932, pretzel',
'933, cheeseburger',
'934, hotdog',
'935, mashed_potato',
'936, head_cabbage',
'937, broccoli',
'938, cauliflower',
'939, zucchini',
'940, spaghetti_squash',
'941, acorn_squash',
'942, butternut_squash',
'943, cucumber',
'944, artichoke',
'945, bell_pepper',
'946, cardoon',
'947, mushroom',
'948, Granny_Smith',
'949, strawberry',
'950, orange',
'951, lemon',
'952, fig',
'953, pineapple',
'954, banana',
'955, jackfruit',
'956, custard_apple',
'957, pomegranate',
'958, hay',
'959, carbonara',
'960, chocolate_sauce',
'961, dough',
'962, meat_loaf',
'963, pizza',
'964, potpie',
'965, burrito',
'966, red_wine',
'967, espresso',
'968, cup',
'969, eggnog',
'970, alp',
'971, bubble',
'972, cliff',
'973, coral_reef',
'974, geyser',
'975, lakeside',
'976, promontory',
'977, sandbar',
'978, seashore',
'979, valley',
'980, volcano',
'981, ballplayer',
'982, groom',
'983, scuba_diver',
'984, rapeseed',
'985, daisy',
"986, yellow_lady's_slipper",
'987, corn',
'988, acorn',
'989, hip',
'990, buckeye',
'991, coral_fungus',
'992, agaric',
'993, gyromitra',
'994, stinkhorn',
'995, earthstar',
'996, hen-of-the-woods',
'997, bolete',
'998, ear',
'999, toilet_tissue']
```

In [33]:

```
# we need to find out the index where the maximum score in output vector out occurs
_, index = torch.max(out, 1)

# print(index)
```

```
# We will use this index to find out the prediction
percentage = torch.nn.functional.softmax(out, dim=1)[0] * 100

# print(percentage)

print(classes[index[0]], ' => ' ,percentage[index[0]].item())
```

```
208, Labrador_retriever  =>  42.46735763549805
```

In [34]:

```
_, indices = torch.sort(out, descending=True)

[
    (classes[idx], percentage[idx].item()) for idx in indices[0][:5]
]
```

Out[34]:

```
[('208, Labrador_retriever', 42.46735763549805),
 ('207, golden_retriever', 16.6086483001709),
 ('176, Saluki', 15.473832130432129),
 ('172, whippet', 2.7881932258605957),
 ('173, Ibizan_hound', 2.3617053031921387)]
```

Where to go from here

start searching for pretrained models and use them according to your purpose, the pretrained models have ton of examples online where you can copy portion of code and then, it depends on your undersanding of python and how to use python in your examples and work.

Next level is to start building your network and improve the created models and thier problems, if you didn't use the model you will never be able to improve and add your own work

In [ ]: