```
# Load the os library
import os
import ssl
ssl._create_default_https_context = ssl._create_unverified_context
```

Create a new folder locally, name it cat_img, store 5 pictures of cat.jpg in it, use plt to read the second picture, and then print out the picture matrix information.

```
files = os.listdir('panda_img')# img.<tab>
import matplotlib.pyplot as plt
import numpy as np

print(os.path.join('panda_img', files[0]))
plt.imread(os.path.join('panda_img', files[0]))

files = [os.path.join('panda_img', file_i)
 for file_i in os.listdir('panda_img')
 if '.jpg' in file_i]

img = plt.imread(files[1])

print(img)
```

```
panda_img\panda1.jpg
[[[116 121  98]
  [112 117  94]
  [107 112  89]
  ...
  [ 34  48  33]
  [ 35  49  34]
  [ 35  49  34]]

 [[116 121  98]
  [112 117  94]
  [107 112  89]
  ...
  [ 34  48  33]
  [ 34  48  33]
  [ 34  48  33]]

 [[117 123  97]
  [112 118  92]
  [107 113  87]
  ...
  [ 33  47  32]
  [ 33  47  32]
```

```
  [ 33   47   32]]

 ...

 [[187 207 122]
  [184 204 119]
  [180 200 115]
  ...
  [ 97 119   54]
  [ 96 118   53]
  [ 96 118   53]]

 [[192 212 127]
  [189 209 124]
  [185 205 120]
  ...
  [100 122   57]
  [ 99 121   56]
  [ 99 121   56]]

 [[196 216 131]
  [193 213 128]
  [189 209 124]
  ...
  [101 123   58]
  [ 99 121   56]
  [ 99 121   56]]]
```

使用 plt.show()函数来打开图片并显示。

```
#plt.imshow(img)
plt.show()
```

****

* *

****

Print the shape information of the image

print(img.shape)

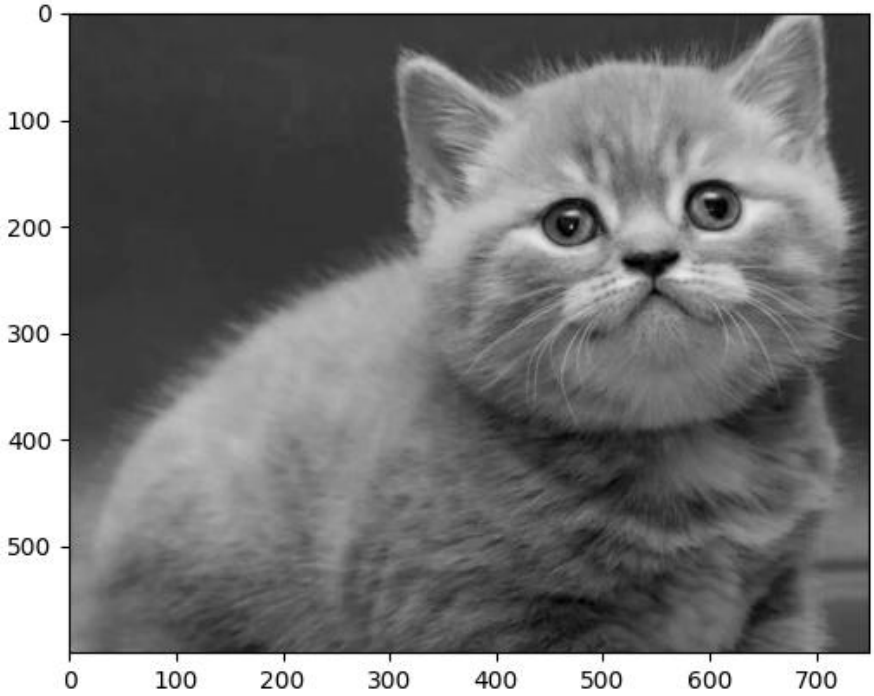
(364, 650, 3)

plt.imshow(img[:, :, 0], cmap='gray') # Red Channel

plt.show()

****

*



*

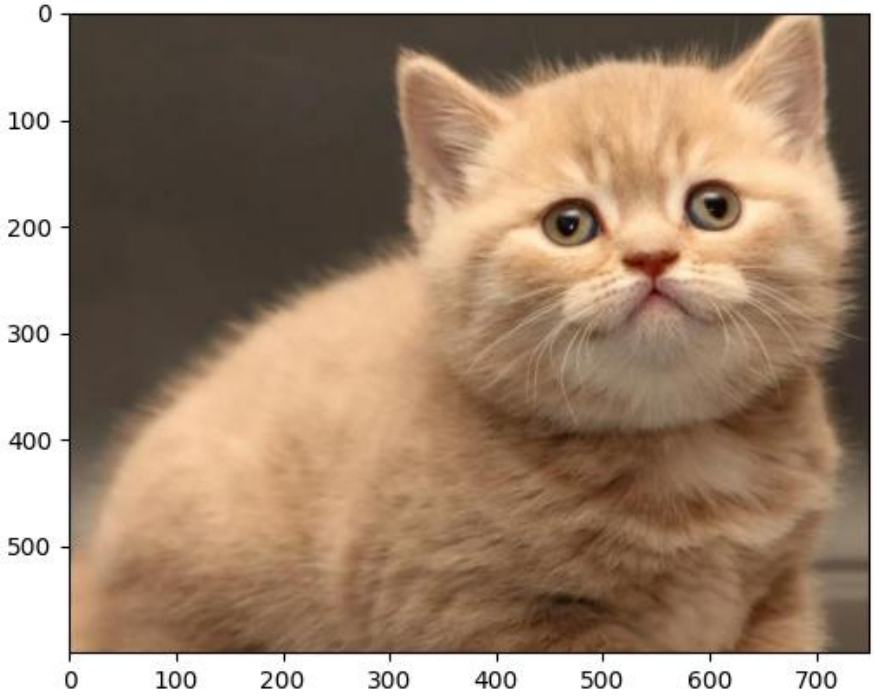
****

---

plt.imshow(img[:, :, 1], cmap='gray') # Green Channel
plt.show()

****

*



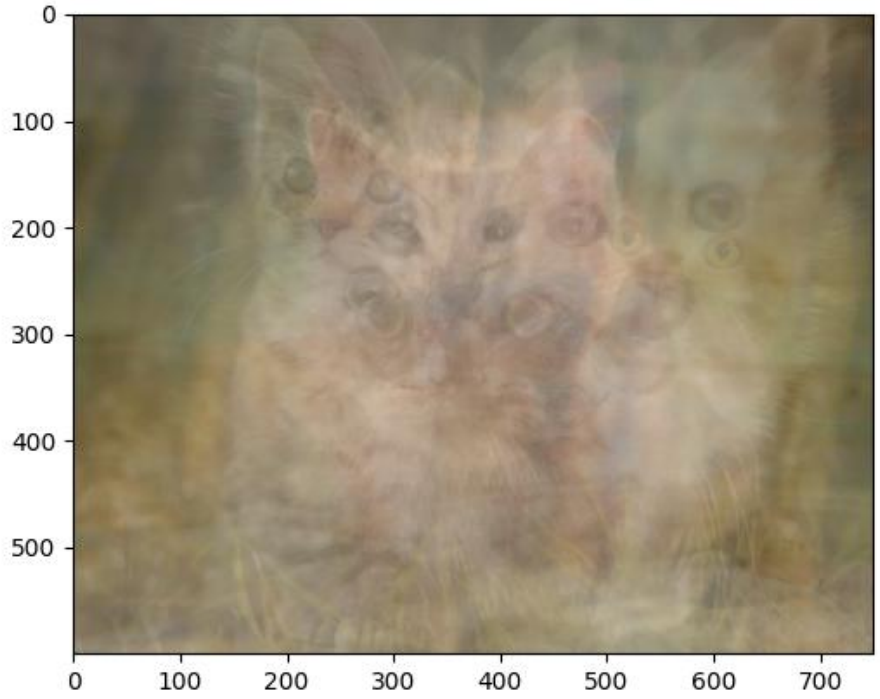
*

****

---

plt.imshow(img[:, :, 2], cmap='gray') # Green Channel
plt.show()

****



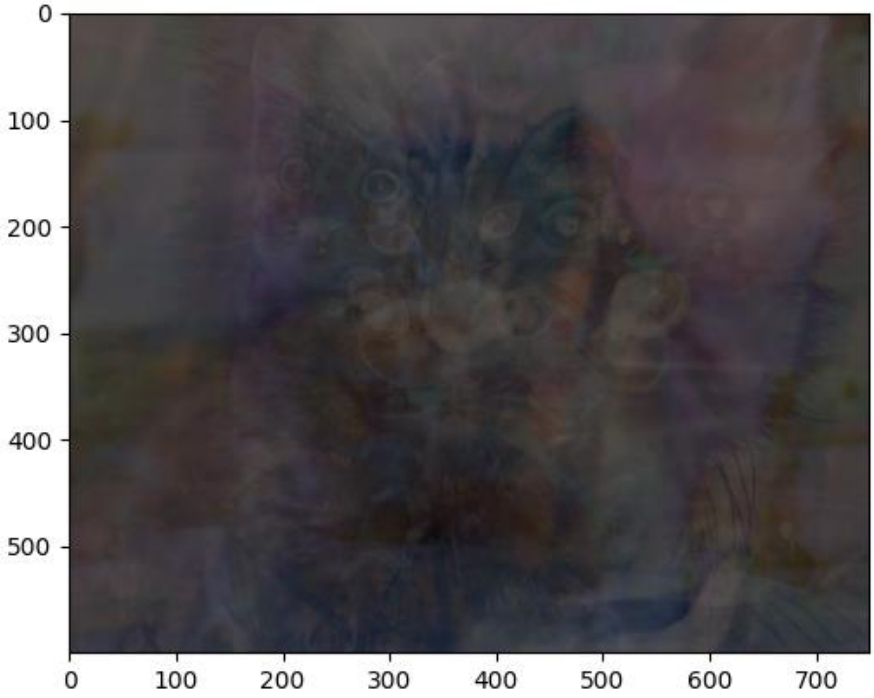
* *

****

```
print(imgs[0].shape)
(749, 522, 3)

data = np.array(imgs[0]) # make 'data' = our numpy array
data.shape
print(data.shape)
print("The shape of our new 'data' object is a 'batch' of 100 images, with a height of 218, width of 178, and 3 colour channels")
print("If your images aren't all the same size to begin with, then this won't work!")

mean_img = np.mean(data, axis=0) # This is the mean of the 'batch' channel
plt.imshow(mean_img.astype(np.uint8))
print("look at this average person")
```

****

*

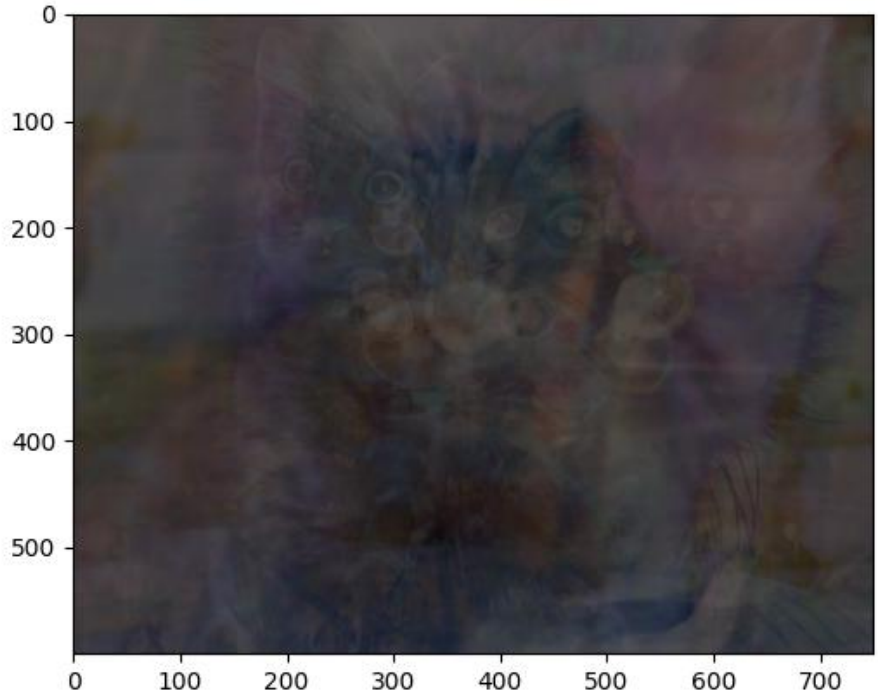


*

****

```
std_img = np.std(data, axis=0)
plt.imshow(std_img.astype(np.uint8))
print("This is the standard deviation - the variance of the mean")
```

****



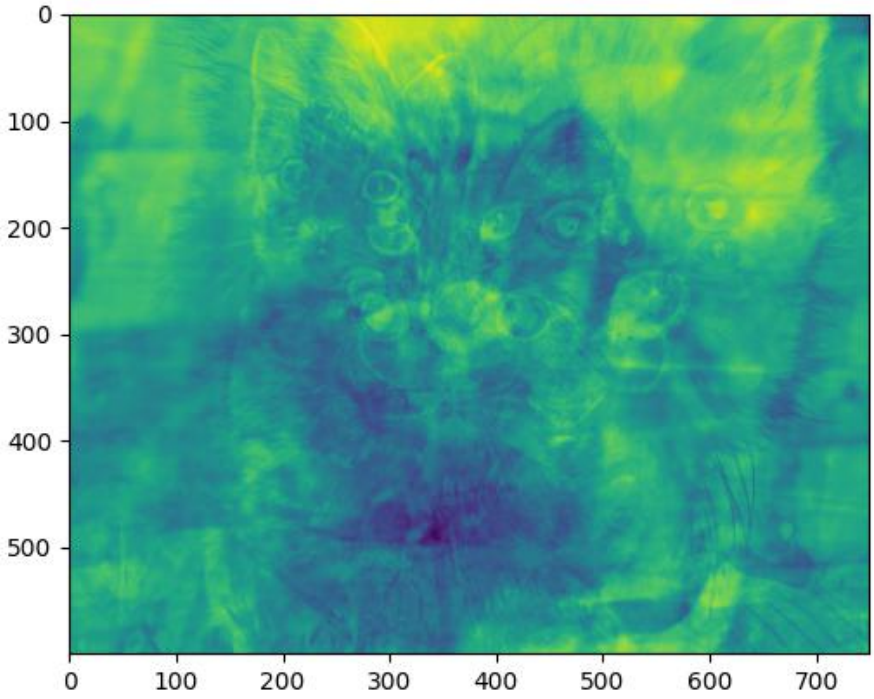
* *

****

```
std_img = np.std(data, axis=0)
plt.imshow(np.mean(std_img, axis=2).astype(np.uint8)) # Mean of all colour channels
print("Mean of all colour channels")
plt.show()
```

****

*



*

****

---

flattened = data.ravel()
# https://docs.scipy.org/doc/numpy/reference/generated/numpy.ravel.html
print(data[:1])
print(flattened[:10])
[[[[51 37 24]
  [51 37 24]
  [54 38 23]
  ...
  [189 171 151]
  [188 170 150]
  [188 170 150]]

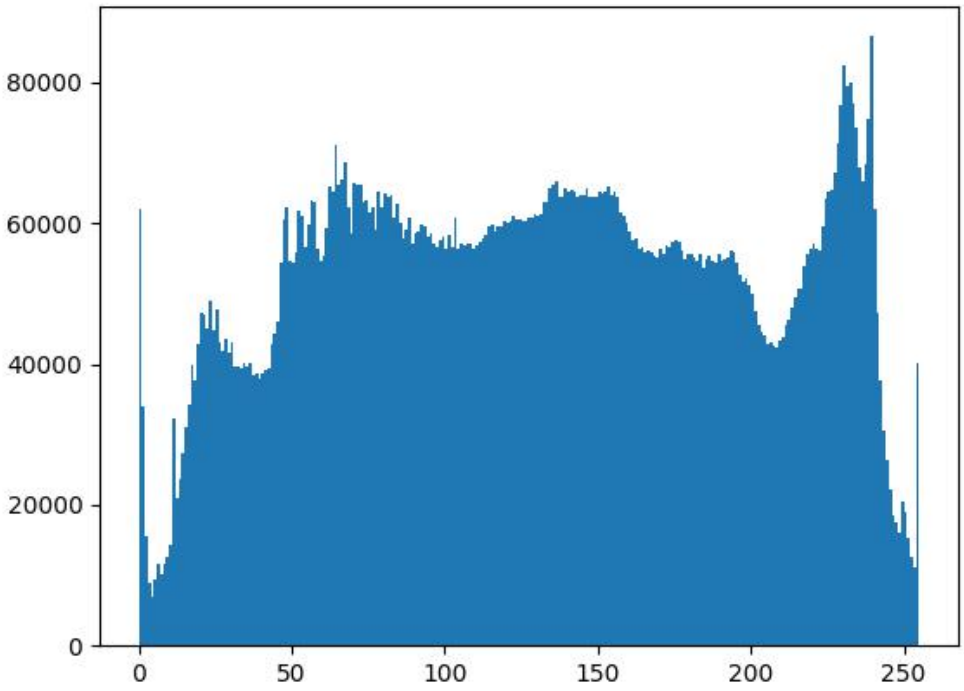
 [[51 37 24]
  [52 38 25]
  [54 38 23]
  ...
  [189 171 151]
  [189 171 151]
  [189 171 151]]

```
   [[ 52   38   25]
    [ 53   39   26]
    [ 55   39   24]
    ...
    [188 170 150]
    [189 171 151]
    [189 171 151]]

   ...

   [[130 119 113]
    [130 119 113]
    [129 118 112]
    ...
    [122 108   95]
    [126 112   99]
    [126 112   99]]

   [[130 119 113]
    [130 119 113]
    [130 119 113]
    ...
    [121 107   94]
    [127 113 100]
    [128 114 101]]

   [[131 120 114]
    [131 120 114]
    [130 119 113]
    ...
    [119 105   92]
    [128 114 101]
    [129 115 102]]]]
[51 37 24 51 37 24 54 38 23 55]

print(plt.hist(flattened.ravel(), 255))
(array([11832., 25883., 29808., 23989., 20970., 19457., 19746., 19186.,
        22176., 19444., 18075., 21675., 18999., 18457., 17572., 17888.,
        17009., 16130., 14996., 13974., 13266., 12781., 12568., 12188.,
        11948., 12063., 12017., 12113., 12409., 12556., 13030., 13334.,
        13677., 13890., 14123., 14133., 14378., 14368., 14667., 14430.,
        14852., 15189., 15327., 15284., 15593., 15534., 16268., 16038.,
        16534., 16562., 16777., 16866., 17222., 17382., 17402., 17454.,
        16800., 16902., 16634., 16394., 16406., 16110., 15873., 16247.,
```

15492., 15227., 15177., 15042., 15114., 16025., 15961., 15667.,
15911., 16535., 17111., 17629., 17766., 18132., 18641., 18735.,
19114., 19038., 19580., 20040., 20276., 20514., 20522., 20817.,
20380., 20414., 20900., 20899., 20860., 20478., 20837., 20596.,
20624., 20413., 20170., 20295., 19595., 19228., 18363., 18607.,
18042., 17920., 17831., 17297., 16717., 16434., 16111., 15248.,
15345., 15308., 15473., 16026., 15608., 15267., 14920., 13908.,
13904., 13433., 13516., 13321., 13283., 13000., 12588., 12400.,
12230., 11961., 11754., 11374., 11230., 10817., 10879., 10604.,
10700., 10429., 10394., 10418., 10504., 10295., 10195., 10278.,
10620., 10592., 10470., 10336., 10324., 9847., 10012., 9751.,
9504., 9736., 9672., 9705., 9747., 10085., 10009., 9815.,
10065., 9982., 9944., 10130., 10078., 10105., 10316., 9836.,
9726., 9931., 9726., 9747., 10103., 10178., 10130., 10328.,
10427., 10617., 10800., 10390., 10877., 11012., 11318., 11154.,
10813., 11087., 11204., 10807., 10798., 10470., 10638., 10818.,
10885., 10908., 11127., 11025., 11187., 11142., 11239., 11146.,
11270., 11013., 10595., 10579., 10076., 10282., 9865., 9840.,
9722., 9522., 9370., 9508., 9585., 9706., 9981., 9693.,
10068., 10290., 10787., 10842., 11531., 11003., 11254., 11495.,
11401., 12155., 12227., 12546., 12992., 13415., 13027., 13042.,
13120., 13377., 14564., 14588., 14628., 13551., 13316., 14011.,
14997., 15198., 14610., 15338., 14662., 14763., 13036., 12397.,
11728., 11714., 11818., 9613., 7908., 5661., 6739.]), array([0., 1.,
2., 3., 4., 5., 6., 7., 8., 9., 10.,
11., 12., 13., 14., 15., 16., 17., 18., 19., 20., 21.,
22., 23., 24., 25., 26., 27., 28., 29., 30., 31., 32.,
33., 34., 35., 36., 37., 38., 39., 40., 41., 42., 43.,
44., 45., 46., 47., 48., 49., 50., 51., 52., 53., 54.,
55., 56., 57., 58., 59., 60., 61., 62., 63., 64., 65.,
66., 67., 68., 69., 70., 71., 72., 73., 74., 75., 76.,
77., 78., 79., 80., 81., 82., 83., 84., 85., 86., 87.,
88., 89., 90., 91., 92., 93., 94., 95., 96., 97., 98.,
99., 100., 101., 102., 103., 104., 105., 106., 107., 108., 109.,
110., 111., 112., 113., 114., 115., 116., 117., 118., 119., 120.,
121., 122., 123., 124., 125., 126., 127., 128., 129., 130., 131.,
132., 133., 134., 135., 136., 137., 138., 139., 140., 141., 142.,
143., 144., 145., 146., 147., 148., 149., 150., 151., 152., 153.,
154., 155., 156., 157., 158., 159., 160., 161., 162., 163., 164.,
165., 166., 167., 168., 169., 170., 171., 172., 173., 174., 175.,
176., 177., 178., 179., 180., 181., 182., 183., 184., 185., 186.,
187., 188., 189., 190., 191., 192., 193., 194., 195., 196., 197.,
198., 199., 200., 201., 202., 203., 204., 205., 206., 207., 208.,
209., 210., 211., 212., 213., 214., 215., 216., 217., 218., 219.,

220., 221., 222., 223., 224., 225., 226., 227., 228., 229., 230.,
231., 232., 233., 234., 235., 236., 237., 238., 239., 240., 241.,
242., 243., 244., 245., 246., 247., 248., 249., 250., 251., 252.,
253., 254., 255.]), <BarContainer object of 255 artists>)

****



* *

****

---

print(plt.hist(mean_img.ravel(), 255))

(array([1.000e+00, 2.000e+00, 5.000e+00, 8.000e+00, 2.000e+01, 3.000e+01,
7.700e+01, 6.900e+01, 8.900e+01, 1.800e+02, 1.950e+02, 2.700e+02,
2.540e+02, 3.660e+02, 2.600e+02, 3.630e+02, 2.440e+02, 2.810e+02,
3.180e+02, 2.480e+02, 3.550e+02, 3.180e+02, 4.090e+02, 3.010e+02,
3.260e+02, 4.240e+02, 3.660e+02, 4.150e+02, 3.780e+02, 4.440e+02,
3.670e+02, 4.560e+02, 4.190e+02, 4.210e+02, 5.140e+02, 4.180e+02,
5.600e+02, 4.650e+02, 6.870e+02, 6.250e+02, 8.880e+02, 7.480e+02,
8.840e+02, 1.135e+03, 9.940e+02, 1.415e+03, 1.147e+03, 1.551e+03,
1.336e+03, 1.443e+03, 1.780e+03, 1.304e+03, 1.766e+03, 1.487e+03,
1.888e+03, 1.696e+03, 2.253e+03, 1.889e+03, 2.009e+03, 2.645e+03,

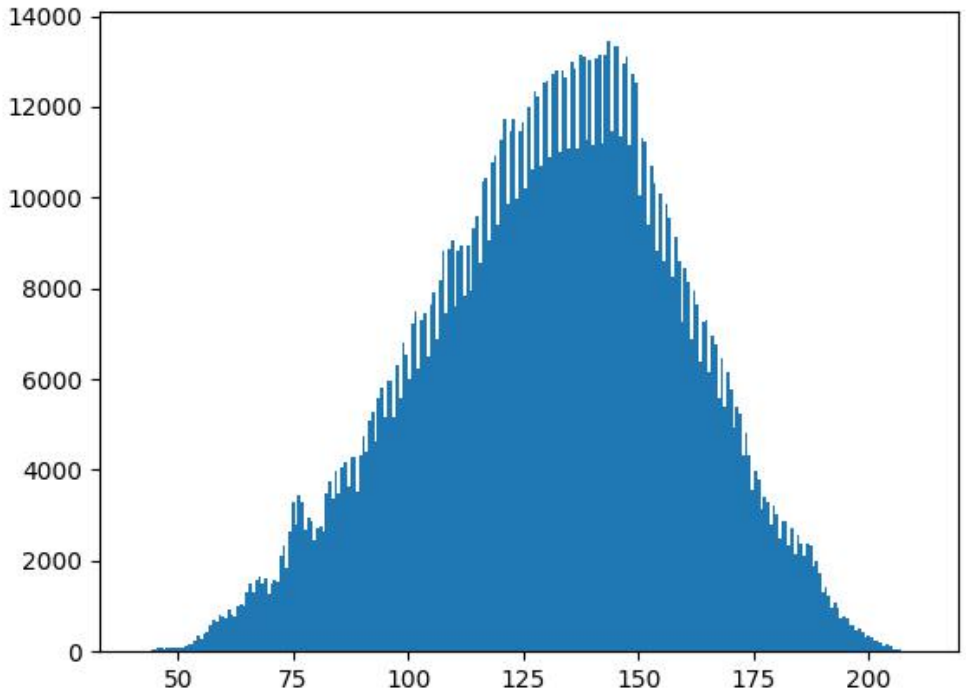
2.082e+03, 2.659e+03, 2.307e+03, 3.065e+03, 2.597e+03, 2.640e+03,
3.500e+03, 2.940e+03, 4.251e+03, 3.771e+03, 5.384e+03, 4.505e+03,
6.128e+03, 4.973e+03, 5.159e+03, 6.441e+03, 5.383e+03, 6.646e+03,
5.580e+03, 6.755e+03, 5.382e+03, 6.663e+03, 5.430e+03, 5.481e+03,
6.813e+03, 5.517e+03, 7.045e+03, 5.574e+03, 7.091e+03, 5.755e+03,
5.533e+03, 7.165e+03, 5.831e+03, 7.762e+03, 6.224e+03, 7.754e+03,
6.197e+03, 7.913e+03, 6.450e+03, 6.595e+03, 7.923e+03, 6.376e+03,
8.074e+03, 6.452e+03, 8.158e+03, 6.552e+03, 6.801e+03, 8.305e+03,
6.835e+03, 8.542e+03, 6.817e+03, 8.658e+03, 6.788e+03, 8.348e+03,
6.483e+03, 6.674e+03, 8.320e+03, 6.583e+03, 8.252e+03, 6.393e+03,
7.649e+03, 6.158e+03, 7.251e+03, 5.755e+03, 5.601e+03, 7.065e+03,
5.608e+03, 6.876e+03, 5.258e+03, 6.164e+03, 4.684e+03, 4.397e+03,
5.538e+03, 4.422e+03, 5.281e+03, 4.211e+03, 5.005e+03, 3.868e+03,
4.859e+03, 3.717e+03, 3.757e+03, 4.795e+03, 3.910e+03, 4.727e+03,
3.717e+03, 4.670e+03, 3.780e+03, 4.763e+03, 3.770e+03, 3.781e+03,
4.921e+03, 3.996e+03, 5.034e+03, 4.010e+03, 4.934e+03, 3.922e+03,
3.809e+03, 4.744e+03, 3.726e+03, 4.409e+03, 3.120e+03, 3.730e+03,
2.887e+03, 3.347e+03, 2.530e+03, 2.274e+03, 2.896e+03, 2.216e+03,
2.583e+03, 2.009e+03, 2.515e+03, 1.966e+03, 1.965e+03, 2.354e+03,
1.873e+03, 2.271e+03, 1.912e+03, 2.263e+03, 1.675e+03, 2.135e+03,
1.617e+03, 1.598e+03, 2.068e+03, 1.629e+03, 1.997e+03, 1.639e+03,
2.001e+03, 1.553e+03, 2.030e+03, 1.647e+03, 1.608e+03, 2.138e+03,
1.728e+03, 2.035e+03, 1.595e+03, 1.992e+03, 1.513e+03, 1.500e+03,
1.806e+03, 1.375e+03, 1.580e+03, 1.217e+03, 1.385e+03, 1.052e+03,
1.239e+03, 8.260e+02, 7.790e+02, 8.590e+02, 6.180e+02, 7.200e+02,
5.610e+02, 6.180e+02, 4.790e+02, 4.820e+02, 5.590e+02, 4.340e+02,
5.040e+02, 4.180e+02, 4.870e+02, 3.670e+02, 4.550e+02, 3.540e+02,
3.590e+02, 4.190e+02, 3.560e+02, 4.280e+02, 3.670e+02, 4.370e+02,
3.690e+02, 4.710e+02, 3.690e+02, 4.000e+02, 4.950e+02, 3.900e+02,
4.950e+02, 4.310e+02, 5.420e+02, 4.220e+02, 4.800e+02, 6.010e+02,
4.220e+02, 5.680e+02, 4.320e+02, 4.720e+02, 3.220e+02, 3.770e+02,
2.260e+02, 1.750e+02, 1.690e+02, 8.200e+01, 7.500e+01, 2.300e+01,
1.700e+01, 1.000e+00, 2.000e+00]), array([9.8 , 10.68784314,
11.57568627, 12.46352941,
13.35137255, 14.23921569, 15.12705882, 16.01490196,
16.9027451 , 17.79058824, 18.67843137, 19.56627451,
20.45411765, 21.34196078, 22.22980392, 23.11764706,
24.0054902 , 24.89333333, 25.78117647, 26.66901961,
27.55686275, 28.44470588, 29.33254902, 30.22039216,
31.10823529, 31.99607843, 32.88392157, 33.77176471,
34.65960784, 35.54745098, 36.43529412, 37.32313725,
38.21098039, 39.09882353, 39.98666667, 40.8745098 ,
41.76235294, 42.65019608, 43.53803922, 44.42588235,
45.31372549, 46.20156863, 47.08941176, 47.9772549 ,

48.86509804, 49.75294118, 50.64078431, 51.52862745,
52.41647059, 53.30431373, 54.19215686, 55.08 ,
55.96784314, 56.85568627, 57.74352941, 58.63137255,
59.51921569, 60.40705882, 61.29490196, 62.1827451 ,
63.07058824, 63.95843137, 64.84627451, 65.73411765,
66.62196078, 67.50980392, 68.39764706, 69.2854902 ,
70.17333333, 71.06117647, 71.94901961, 72.83686275,
73.72470588, 74.61254902, 75.50039216, 76.38823529,
77.27607843, 78.16392157, 79.05176471, 79.93960784,
80.82745098, 81.71529412, 82.60313725, 83.49098039,
84.37882353, 85.26666667, 86.1545098 , 87.04235294,
87.93019608, 88.81803922, 89.70588235, 90.59372549,
91.48156863, 92.36941176, 93.2572549 , 94.14509804,
95.03294118, 95.92078431, 96.80862745, 97.69647059,
98.58431373, 99.47215686, 100.36 , 101.24784314,
102.13568627, 103.02352941, 103.91137255, 104.79921569,
105.68705882, 106.57490196, 107.4627451 , 108.35058824,
109.23843137, 110.12627451, 111.01411765, 111.90196078,
112.78980392, 113.67764706, 114.5654902 , 115.45333333,
116.34117647, 117.22901961, 118.11686275, 119.00470588,
119.89254902, 120.78039216, 121.66823529, 122.55607843,
123.44392157, 124.33176471, 125.21960784, 126.10745098,
126.99529412, 127.88313725, 128.77098039, 129.65882353,
130.54666667, 131.4345098 , 132.32235294, 133.21019608,
134.09803922, 134.98588235, 135.87372549, 136.76156863,
137.64941176, 138.5372549 , 139.42509804, 140.31294118,
141.20078431, 142.08862745, 142.97647059, 143.86431373,
144.75215686, 145.64 , 146.52784314, 147.41568627,
148.30352941, 149.19137255, 150.07921569, 150.96705882,
151.85490196, 152.7427451 , 153.63058824, 154.51843137,
155.40627451, 156.29411765, 157.18196078, 158.06980392,
158.95764706, 159.8454902 , 160.73333333, 161.62117647,
162.50901961, 163.39686275, 164.28470588, 165.17254902,
166.06039216, 166.94823529, 167.83607843, 168.72392157,
169.61176471, 170.49960784, 171.38745098, 172.27529412,
173.16313725, 174.05098039, 174.93882353, 175.82666667,
176.7145098 , 177.60235294, 178.49019608, 179.37803922,
180.26588235, 181.15372549, 182.04156863, 182.92941176,
183.8172549 , 184.70509804, 185.59294118, 186.48078431,
187.36862745, 188.25647059, 189.14431373, 190.03215686,
190.92 , 191.80784314, 192.69568627, 193.58352941,
194.47137255, 195.35921569, 196.24705882, 197.13490196,
198.0227451 , 198.91058824, 199.79843137, 200.68627451,
201.57411765, 202.46196078, 203.34980392, 204.23764706,

205.1254902 , 206.01333333, 206.90117647, 207.78901961,
208.67686275, 209.56470588, 210.45254902, 211.34039216,
212.22823529, 213.11607843, 214.00392157, 214.89176471,
215.77960784, 216.66745098, 217.55529412, 218.44313725,
219.33098039, 220.21882353, 221.10666667, 221.9945098 ,
222.88235294, 223.77019608, 224.65803922, 225.54588235,
226.43372549, 227.32156863, 228.20941176, 229.0972549 ,
229.98509804, 230.87294118, 231.76078431, 232.64862745,
233.53647059, 234.42431373, 235.31215686, 236.2]), <BarContainer object of 255 artists>)

****

*



*

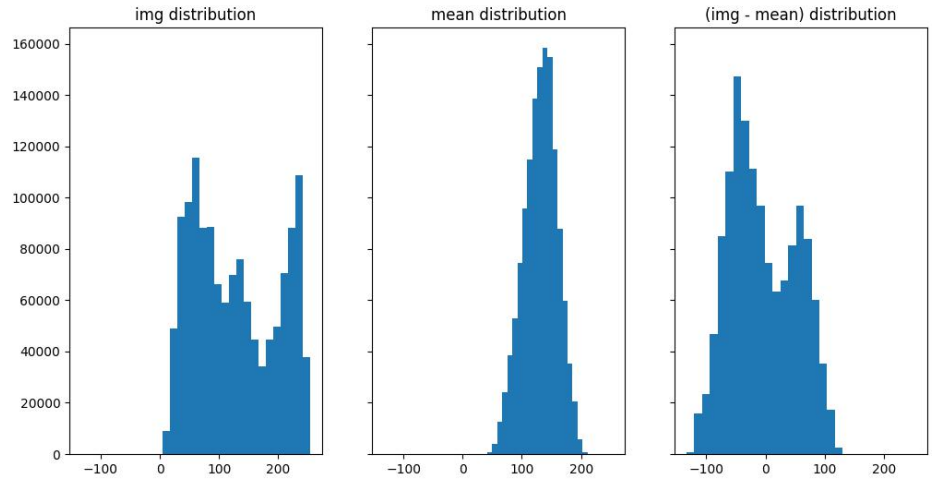
****

```
bins = 20
fig, axs = plt.subplots(1, 3, figsize=(12, 6), sharey=True, sharex=True)
axs[0].hist((data[0]).ravel(), bins)
axs[0].set_title('img distribution')
axs[1].hist((mean_img).ravel(), bins)
```

```
axs[1].set_title('mean distribution')
axs[2].hist((data[0] - mean_img).ravel(), bins)
axs[2].set_title('(img - mean) distribution')
plt.show()
```

****

*

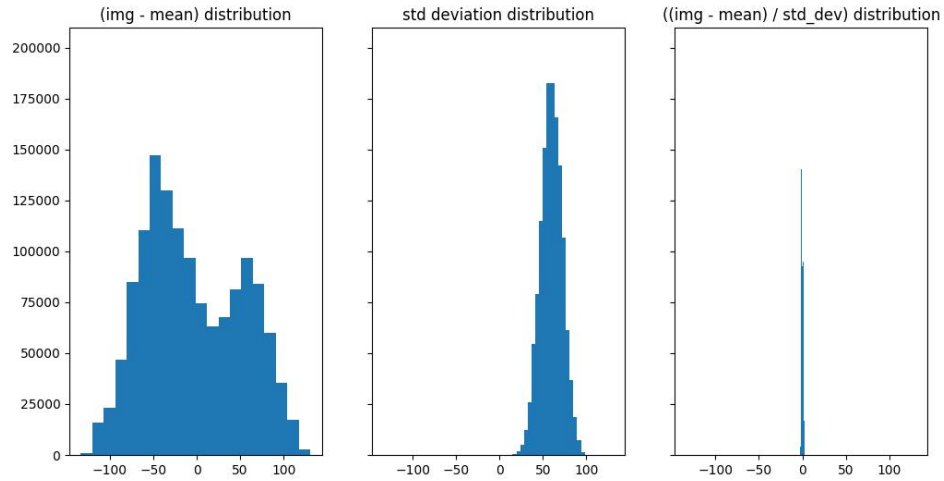


*

****

```
fig, axs = plt.subplots(1, 3, figsize=(12, 6), sharey=True, sharex=True)
axs[0].hist((data[0] - mean_img).ravel(), bins)
axs[0].set_title('(img - mean) distribution')
axs[1].hist((std_img).ravel(), bins)
axs[1].set_title('std deviation distribution')
axs[2].hist(((data[0] - mean_img) / std_img).ravel(), bins)
axs[2].set_title('((img - mean) / std_dev) distribution')
plt.show()
```

****



* *

****

```
import tensorflow._api.v2.compat.v1 as tf
tf.disable_v2_behavior()
x = np.linspace(-3.0, 3.0, 100)
```

```
WARNING:tensorflow:From
D:\Anaconda\envs\tf\lib\site-packages\tensorflow\python\compat\v2_compat.py:101:
disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated
and will be removed in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
```

```
x = np.linspace(-3.0, 3.0, 100)
# Immediately, the result is given to us.   An array of 100 numbers equally spaced from
-3.0 to 3.0.
print(x)
# We know from numpy arrays that they have a `shape`, in this case a 1-dimensional
array of 100 values
print(x.shape)
# and a `dtype`, in this case float64, or 64 bit floating point values.
print(x.dtype)
```

```
[-3.         -2.93939394 -2.87878788 -2.81818182 -2.75757576 -2.6969697
 -2.63636364 -2.57575758 -2.51515152 -2.45454545 -2.39393939 -2.33333333
 -2.27272727 -2.21212121 -2.15151515 -2.09090909 -2.03030303 -1.96969697
 -1.90909091 -1.84848485 -1.78787879 -1.72727273 -1.66666667 -1.60606061
 -1.54545455 -1.48484848 -1.42424242 -1.36363636 -1.3030303  -1.24242424
```

```
 -1.18181818 -1.12121212 -1.06060606 -1.         -0.93939394 -0.87878788
 -0.81818182 -0.75757576 -0.6969697  -0.63636364 -0.57575758 -0.51515152
 -0.45454545 -0.39393939 -0.33333333 -0.27272727 -0.21212121 -0.15151515
 -0.09090909 -0.03030303  0.03030303  0.09090909  0.15151515  0.21212121
  0.27272727  0.33333333  0.39393939  0.45454545  0.51515152  0.57575758
  0.63636364  0.6969697   0.75757576  0.81818182  0.87878788  0.93939394
  1.          1.06060606  1.12121212  1.18181818  1.24242424  1.3030303
  1.36363636  1.42424242  1.48484848  1.54545455  1.60606061  1.66666667
  1.72727273  1.78787879  1.84848485  1.90909091  1.96969697  2.03030303
  2.09090909  2.15151515  2.21212121  2.27272727  2.33333333  2.39393939
  2.45454545  2.51515152  2.57575758  2.63636364  2.6969697   2.75757576
  2.81818182  2.87878788  2.93939394  3.        ]
(100,)
float64

x = tf.linspace(-3.0, 3.0, 100)
print(x)
Tensor("linspace/Slice:0", shape=(100,), dtype=float32)

print([op.name for op in g.get_operations()])
['linspace/start',
 'linspace/stop',
 'linspace/num',
 'linspace/Cast',
 'linspace/Cast_1',
 'linspace/Shape',
 'linspace/Shape_1',
 'linspace/BroadcastArgs',
 'linspace/BroadcastTo',
 'linspace/BroadcastTo_1',
 'linspace/ExpandDims/dim',
 'linspace/ExpandDims',
 'linspace/ExpandDims_1/dim',
 'linspace/ExpandDims_1',
 'linspace/Shape_2',
 'linspace/Shape_3',
 'linspace/strided_slice/stack',
 'linspace/strided_slice/stack_1',
 'linspace/strided_slice/stack_2',
 'linspace/strided_slice',
 'linspace/add/y',
 'linspace/add',
 'linspace/SelectV2/condition',
 'linspace/SelectV2/t',
```

```
 'linspace/SelectV2',
 'linspace/sub/y',
 'linspace/sub',
 'linspace/Maximum/y',
 'linspace/Maximum',
 'linspace/sub_1/y',
 'linspace/sub_1',
 'linspace/Maximum_1/y',
 'linspace/Maximum_1',
 'linspace/sub_2',
 'linspace/Cast_2',
 'linspace/truediv',
 'linspace/GreaterEqual/y',
 'linspace/GreaterEqual',
 'linspace/SelectV2_1/e',
 'linspace/SelectV2_1',
 'linspace/range/start',
 'linspace/range/delta',
 'linspace/range/Cast',
 'linspace/range',
 'linspace/Cast_3',
 'linspace/range_1/start',
 'linspace/range_1/delta',
 'linspace/range_1',
 'linspace/Equal',
 'linspace/Equal',
 'linspace/SelectV2_2/e',
 'linspace/SelectV2_2',
 'linspace/Reshape',
 'linspace/mul',
 'linspace/add_1',
 'linspace/concat',
 'linspace/zeros_like',
 'linspace/SelectV2_3',
 'linspace/Slice']

 # We're first going to create a session:
sess = tf.Session()

# Now we tell our session to compute anything we've created in the tensorflow graph.
computed_x = sess.run(x)
print(computed_x)

# Alternatively, we could tell the previous Tensor to evaluate itself using this session:
```

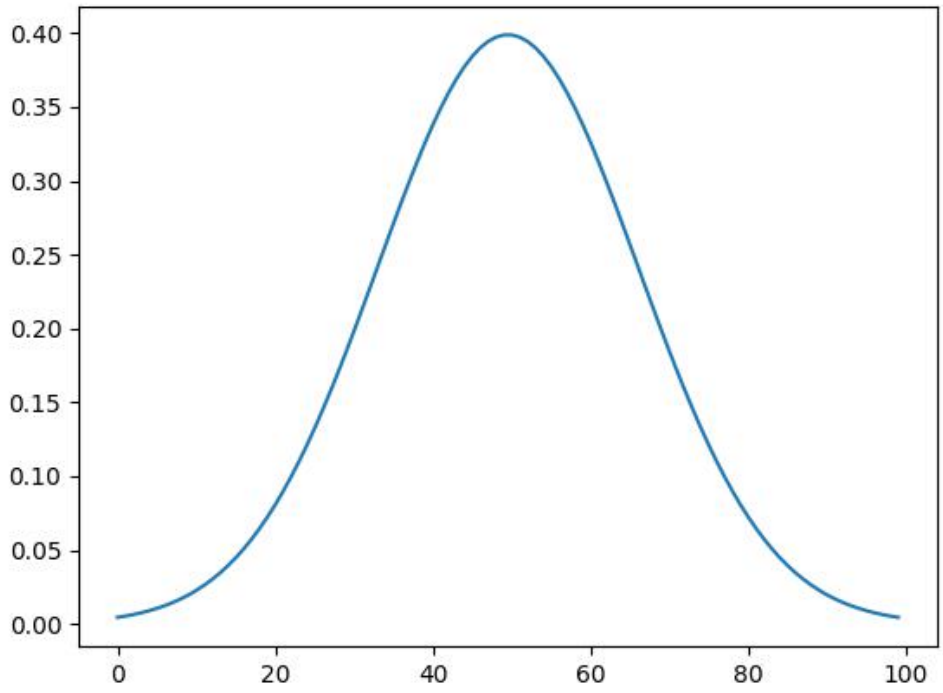
```
computed_x = x.eval(session=sess)
print(computed_x)

# We can close the session after we're done like so:
sess.close()
```

```
[-3.         -2.939394   -2.878788   -2.8181818  -2.7575758  -2.6969697
 -2.6363635  -2.5757575  -2.5151515  -2.4545455  -2.3939395  -2.3333333
 -2.2727273  -2.2121212  -2.151515   -2.090909   -2.030303   -1.969697
 -1.9090909  -1.8484848  -1.7878788  -1.7272727  -1.6666666  -1.6060605
 -1.5454545  -1.4848485  -1.4242424  -1.3636363  -1.3030303  -1.2424242
 -1.1818181  -1.121212   -1.060606   -1.         -0.939394   -0.87878776
 -0.81818175 -0.75757575 -0.6969695  -0.6363635  -0.5757575  -0.5151515
 -0.4545455  -0.39393926 -0.33333325 -0.27272725 -0.21212101 -0.151515
 -0.090909   -0.030303    0.030303    0.09090924  0.15151525  0.21212125
  0.2727275   0.3333335   0.3939395   0.4545455   0.5151515   0.57575774
  0.63636374  0.69696975  0.757576    0.818182    0.878788    0.939394
  1.          1.060606    1.121212    1.1818185   1.2424245   1.3030305
  1.3636365   1.4242425   1.4848485   1.5454545   1.606061    1.666667
  1.727273    1.787879    1.848485    1.909091    1.969697    2.030303
  2.090909    2.1515155   2.2121215   2.2727275   2.3333335   2.3939395
  2.4545455   2.5151515   2.575758    2.636364    2.69697     2.757576
  2.818182    2.878788    2.939394    3.         ]
[-3.         -2.939394   -2.878788   -2.8181818  -2.7575758  -2.6969697
 -2.6363635  -2.5757575  -2.5151515  -2.4545455  -2.3939395  -2.3333333
 -2.2727273  -2.2121212  -2.151515   -2.090909   -2.030303   -1.969697
 -1.9090909  -1.8484848  -1.7878788  -1.7272727  -1.6666666  -1.6060605
 -1.5454545  -1.4848485  -1.4242424  -1.3636363  -1.3030303  -1.2424242
 -1.1818181  -1.121212   -1.060606   -1.         -0.939394   -0.87878776
 -0.81818175 -0.75757575 -0.6969695  -0.6363635  -0.5757575  -0.5151515
 -0.4545455  -0.39393926 -0.33333325 -0.27272725 -0.21212101 -0.151515
 -0.090909   -0.030303    0.030303    0.09090924  0.15151525  0.21212125
  0.2727275   0.3333335   0.3939395   0.4545455   0.5151515   0.57575774
  0.63636374  0.69696975  0.757576    0.818182    0.878788    0.939394
  1.          1.060606    1.121212    1.1818185   1.2424245   1.3030305
  1.3636365   1.4242425   1.4848485   1.5454545   1.606061    1.666667
  1.727273    1.787879    1.848485    1.909091    1.969697    2.030303
  2.090909    2.1515155   2.2121215   2.2727275   2.3333335   2.3939395
  2.4545455   2.5151515   2.575758    2.636364    2.69697     2.757576
  2.818182    2.878788    2.939394    3.         ]
```

```
 sess = tf.Session(graph=g)
sess.close()
```

```
g2 = tf.Graph()
sess = tf.InteractiveSession()
x.eval()
[-3.         -2.939394   -2.878788   -2.8181818  -2.7575758  -2.6969697
 -2.6363635  -2.5757575  -2.5151515  -2.4545455  -2.3939395  -2.3333333
 -2.2727273  -2.2121212  -2.151515   -2.090909   -2.030303   -1.969697
 -1.9090909  -1.8484848  -1.7878788  -1.7272727  -1.6666666  -1.6060605
 -1.5454545  -1.4848485  -1.4242424  -1.3636363  -1.3030303  -1.2424242
 -1.1818181  -1.121212   -1.060606   -1.         -0.939394   -0.87878776
 -0.81818175 -0.75757575 -0.6969695  -0.6363635  -0.5757575  -0.5151515
 -0.4545455  -0.39393926 -0.33333325 -0.27272725 -0.21212101 -0.151515
 -0.090909   -0.030303    0.030303    0.09090924  0.15151525  0.21212125
  0.2727275   0.3333335   0.3939395   0.4545455   0.5151515   0.57575774
  0.63636374  0.69696975  0.757576    0.818182    0.878788    0.939394
  1.          1.060606    1.121212    1.1818185   1.2424245   1.3030305
  1.3636365   1.4242425   1.4848485   1.5454545   1.606061    1.666667
  1.727273    1.787879    1.848485    1.909091    1.969697    2.030303
  2.090909    2.1515155   2.2121215   2.2727275   2.3333335   2.3939395
  2.4545455   2.5151515   2.575758    2.636364    2.69697     2.757576
  2.818182    2.878788    2.939394    3.          ]

  # We can find out the shape of a tensor like so:
print(x.get_shape())

# %% Or in a more friendly format
print(x.get_shape().as_list())
(100,)
[100]

mean = 0.0
sigma = 1.0

# Don't worry about trying to learn or remember this formula.  I always have to refer to
textbooks or check online for the exact formula.
z = (tf.exp(tf.negative(tf.pow(x - mean, 2.0) /
                        (2.0 * tf.pow(sigma, 2.0)))) *
     (1.0 / (sigma * tf.sqrt(2.0 * 3.1415))))
res = z.eval()
plt.plot(res)
plt.show()
-------------------------------------

            ***********
```

*

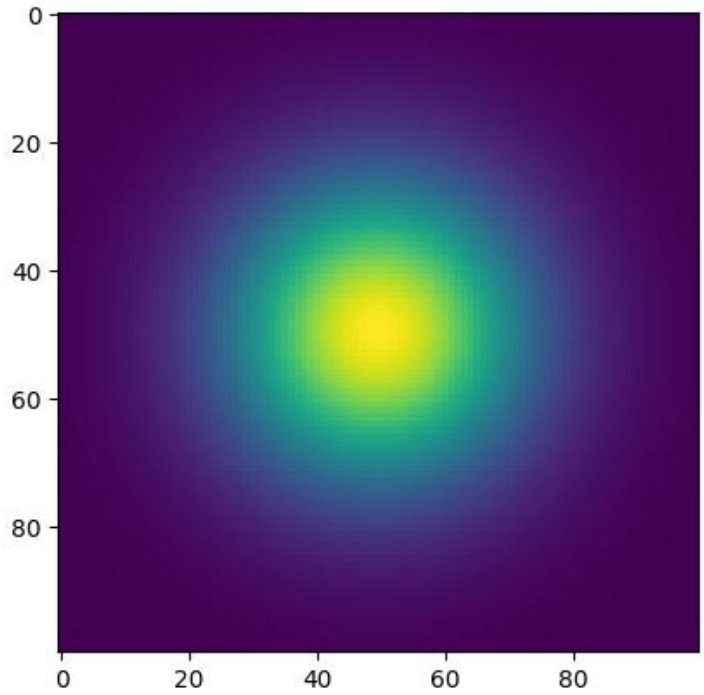


*

****

```
ksize = z.get_shape().as_list()[0]

# Let's multiply the two to get a 2d gaussian
z_2d = tf.matmul(tf.reshape(z, [ksize, 1]), tf.reshape(z, [1, ksize]))

# Execute the graph
plt.imshow(z_2d.eval())
plt.show()
```

****


0
20
40
60
80
0
20
40
60
80

* *

****
