



Week 2

DEEP LEARNING FOR COMPUTER VISION

Presented by **Asst. Prof. Dr. Tuchsanai Ploysuwan**



Understanding Images

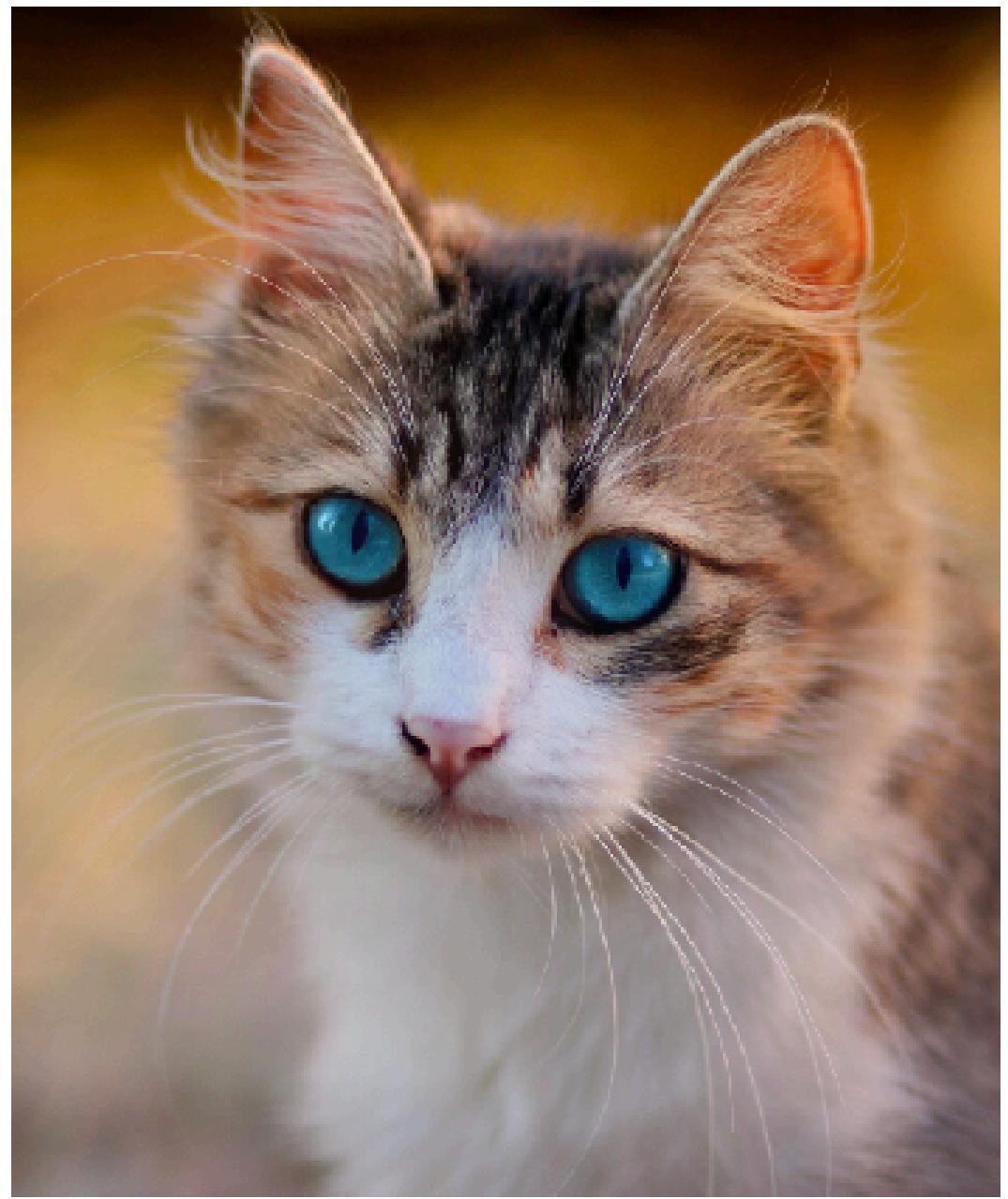
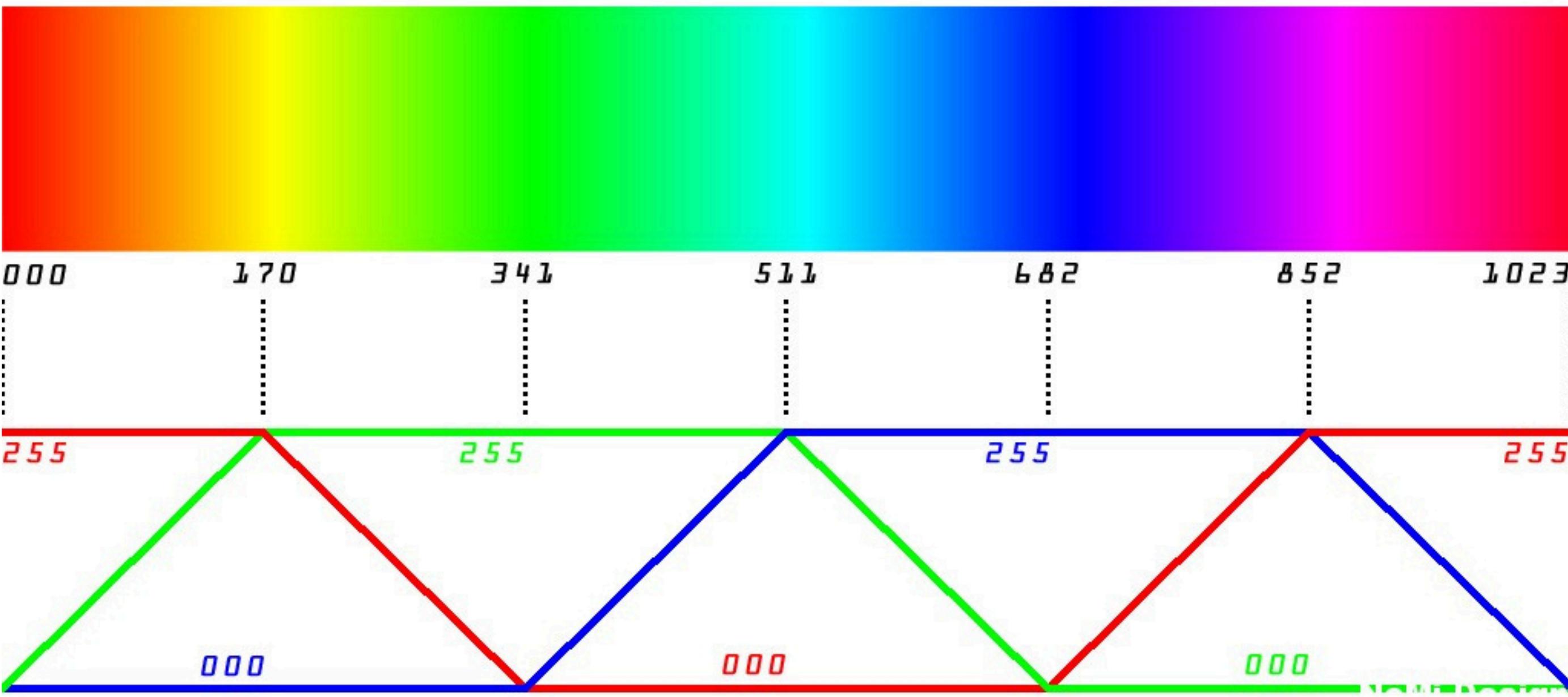


Image Basics



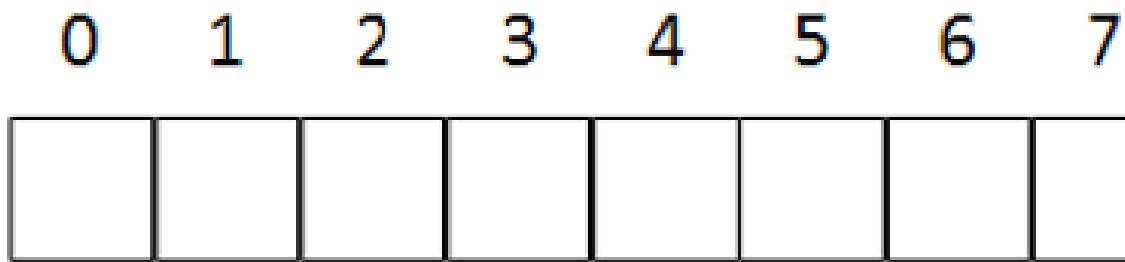
How do Computers ‘see’ Images?



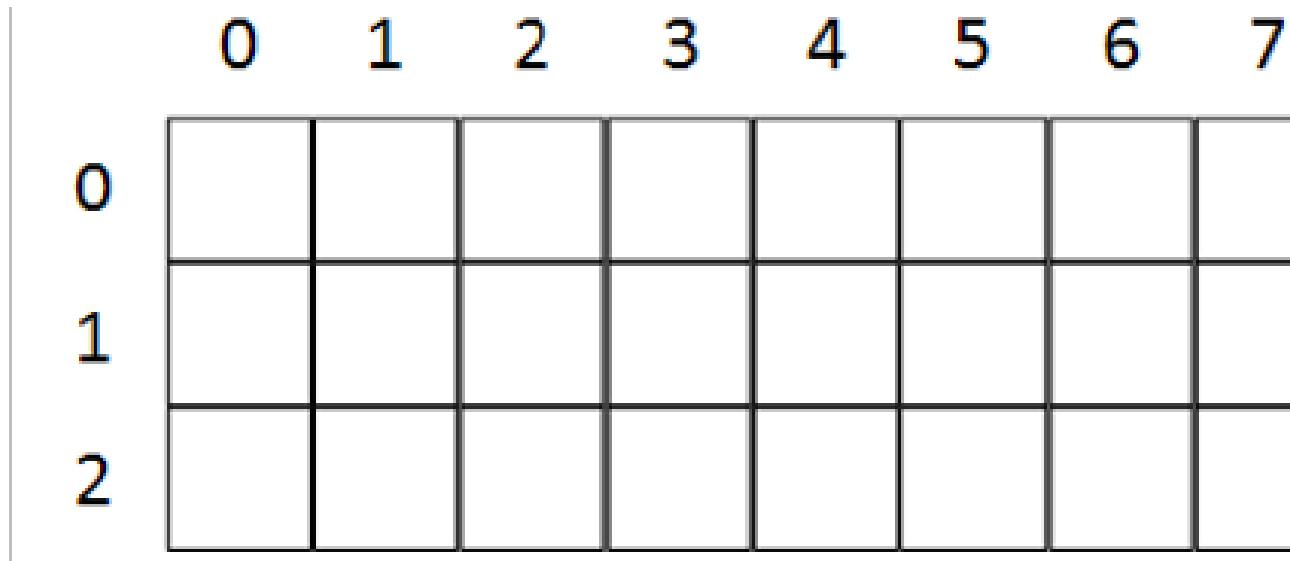
Digital Images Format

Images are stored in Multi-Dimensional Arrays

- A 1-Dimensional array looks like this



- A 2-Dimensional array looks like this



- A 3-Dimensional array looks like this

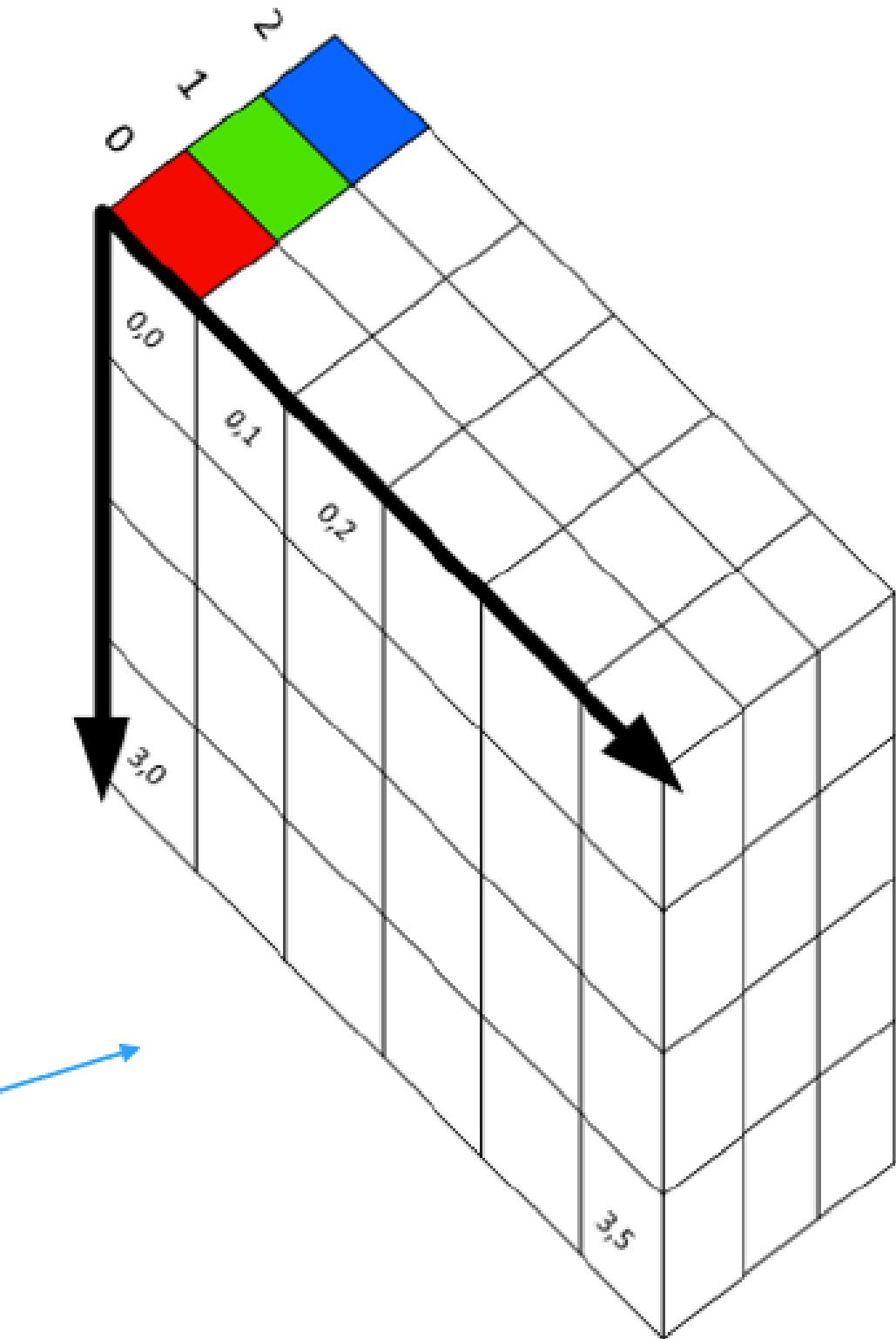
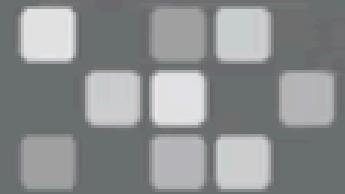


Image File Formats



Raster images

Pixel-based graphics
Resolution dependent
Photos & web graphics

JPG

Web & print
photos and
quick previews

GIF

Animation &
transparency in
limited colors

PNG

Transparency
with millions
of colors

TIFF

High quality
print graphics
and scans

RAW

Unprocessed
data from
digital cameras

PSD

Layered Adobe
Photoshop
design files



Vector images

Curve-based graphics
Resolution independent
Logos, icons, & type

PDF

Print files and
web-based
documents

EPS

Individual
vector design
elements

AI

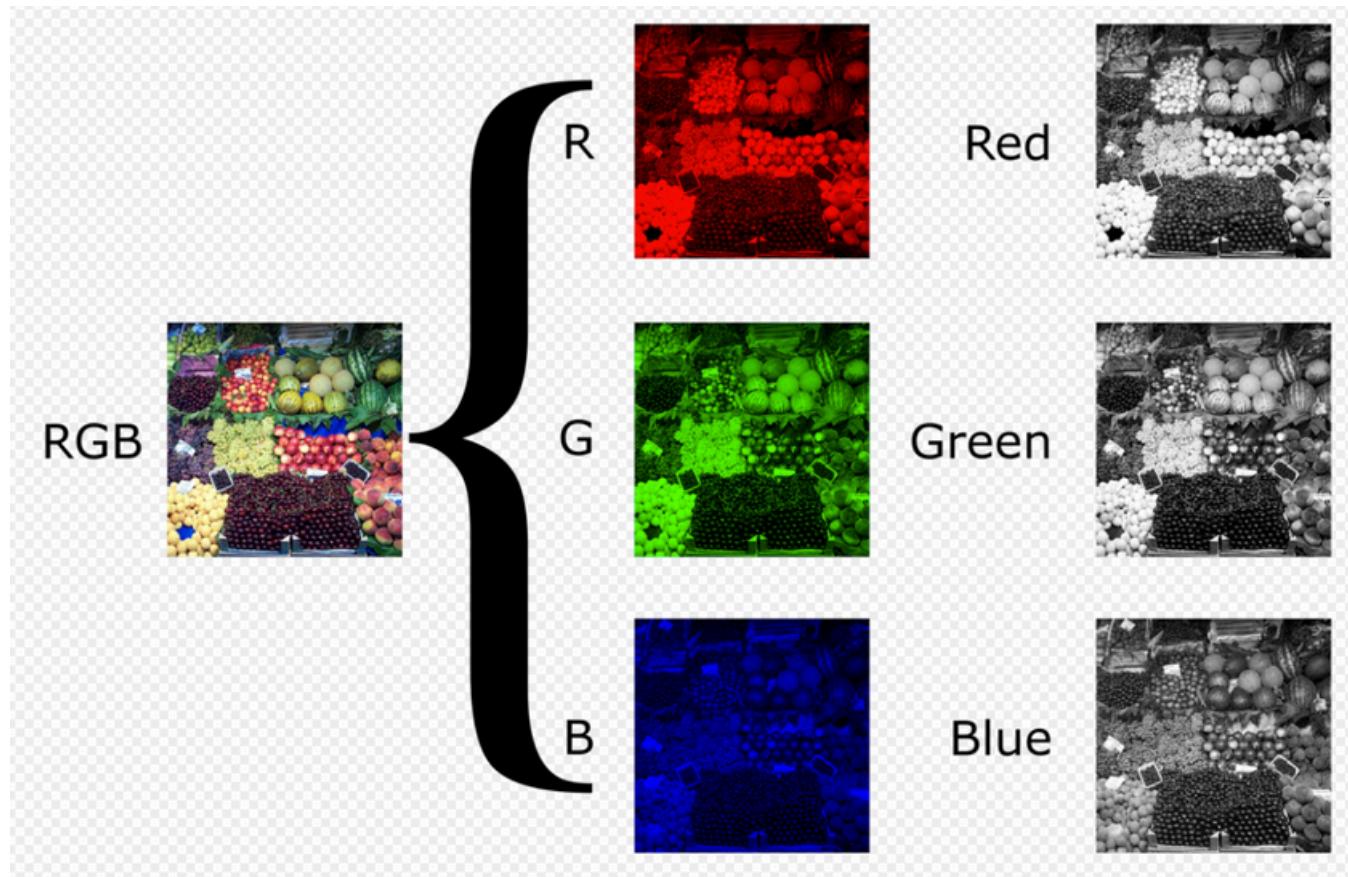
Original Adobe
Illustrator
design files

SVG

Vector files
for web
publishing

Grayscale Image

Sometimes referred to as black and white image



Luma coding in video systems [edit]

Main article: luma (video)

For images in color spaces such as Y'UV and its relatives, which are used in standard color TV and video systems such as PAL, SECAM, and NTSC, a nonlinear luma component (Y') is calculated directly from gamma-compressed primary intensities as a weighted sum, which, although not a perfect representation of the colorimetric luminance, can be calculated more quickly without the gamma expansion and compression used in photometric/colorimetric calculations. In the Y'UV and Y'IQ models used by PAL and NTSC, the rec601 luma (Y') component is computed as

$$Y' \equiv 0.299R' + 0.587G' + 0.114B'$$

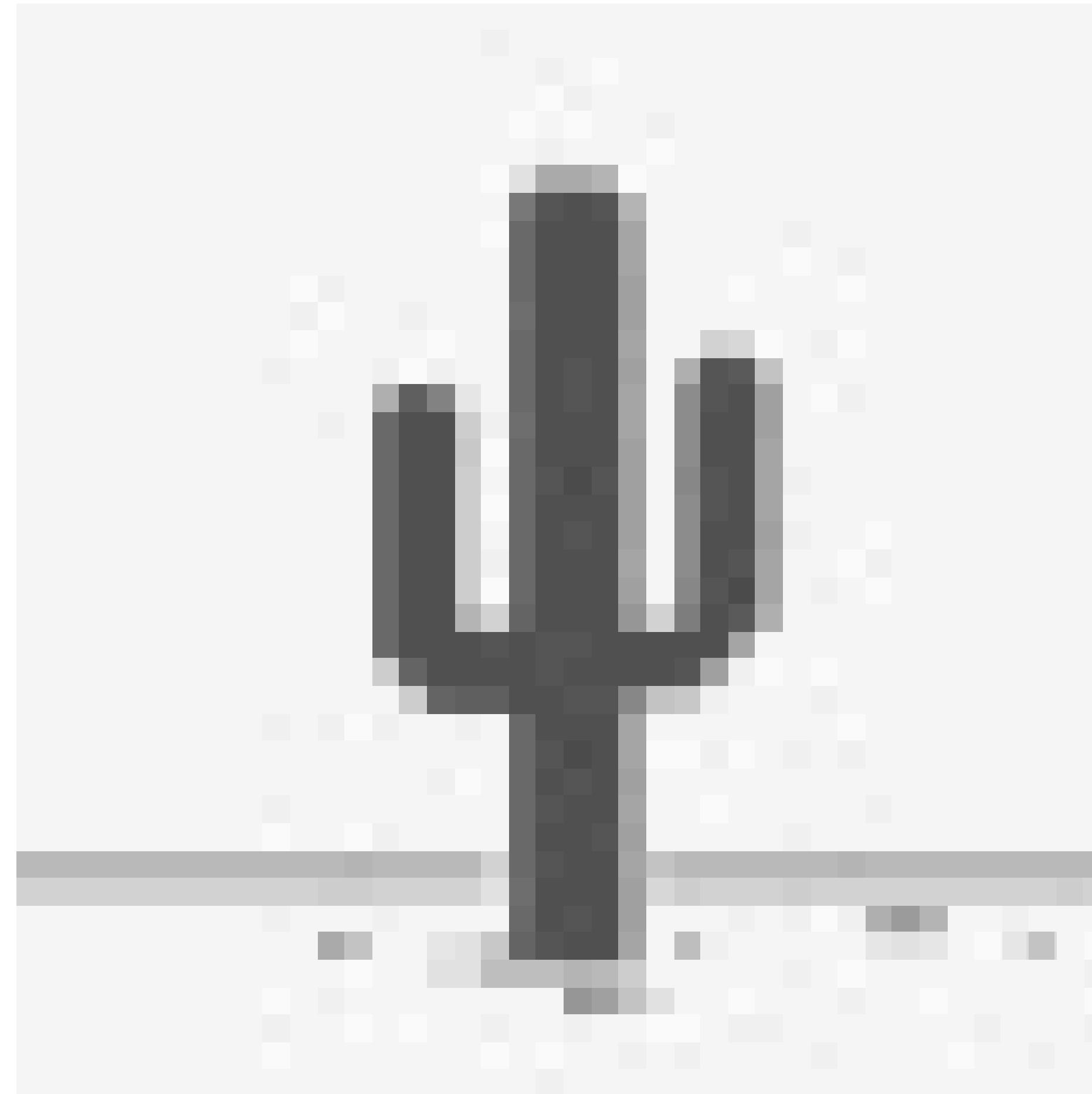
where we use the prime to distinguish these nonlinear values from the sRGB nonlinear values (discussed above) which use a somewhat different gamma compression formula, and from the linear RGB components. The [ITU-R BT.709](#) standard used for [HDTV](#) developed by the [ATSC](#) uses different color coefficients, computing the luma component as

$$Y' \equiv 0.2126R' + 0.7152G' + 0.0722B'.$$

Although these are numerically the same coefficients used in sRGB above, the effect is different because here they are being applied directly to gamma-compressed values rather than to the linearized values. The [ITU-R BT.2100](#) standard for [HDR](#) television uses yet different coefficients, computing the luma component as

$$Y' \equiv 0.2627R' + 0.6780G' + 0.0593B'.$$

Images & Kernel Convolutions



Images & Kernel Convolutions

Images & Kernel Convolutions

Kernel Convolutions

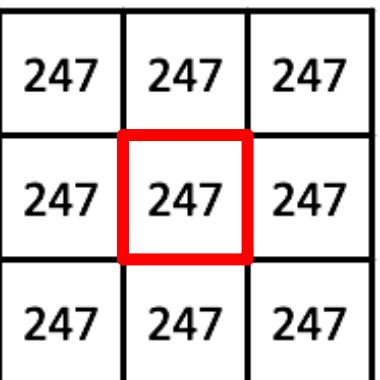
Kernel is an MxN matrix

3x3 Kernel Example

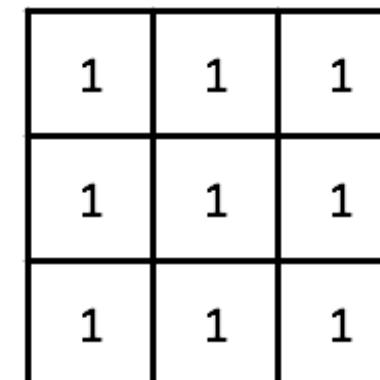
1	1	1
1	1	1
1	1	1

Kernel Convolutions

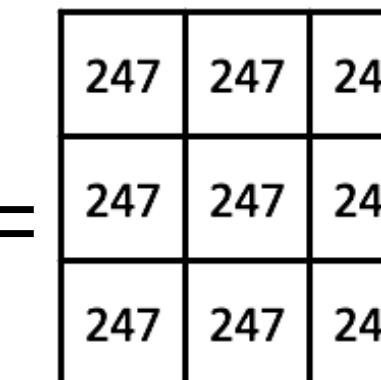
Window



Kernel



Multiplied



Sum = 9

Sum = 2223

New Pixel Value

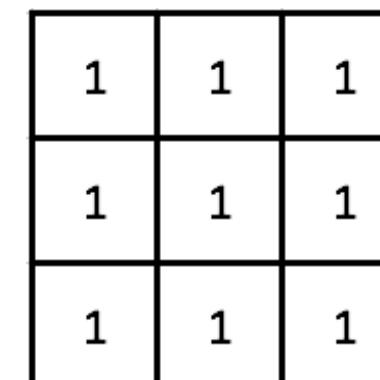
$$2223 / 9 = 247$$

Kernel Convolutions

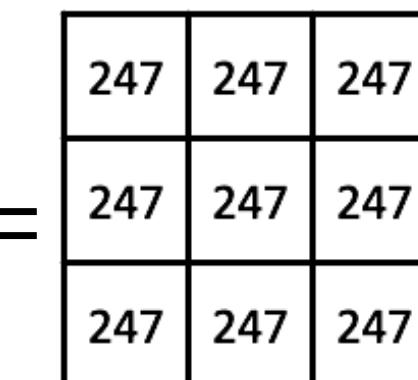
Window



Kernel



Multiplied



$$\text{Sum} = 9$$

Sum = 2223

New Pixel Value

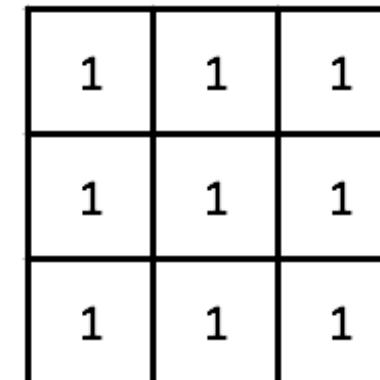
$$2223 / 9 = 247$$

Kernel Convolutions

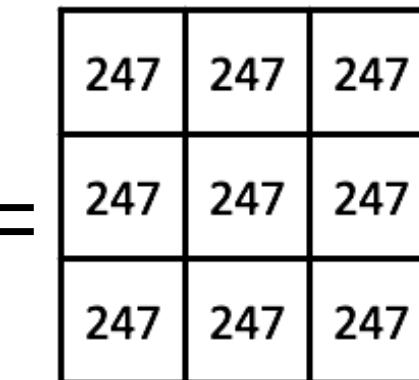
Window



Kernel



Multiplied



Sum = 9

Sum = 2223

New Pixel
Value
 $223 / 9 = 24$

Kernel Convolutions

Window

247	247	120
247	247	83
247	247	83

X

Kernel

1	1	1
1	1	1
1	1	1

Multiplied

	247	247	120
=	247	247	83
	247	247	83

$$\text{Sum} = 9$$

Sum = 1768

New Pixel
Value
 $68 / 9 = 19$
196

Kernel Convolutions

Window

247	120	95
247	83	83
247	83	83

x

Kernel

1	1	1
1	1	1
1	1	1

$$\text{Sum} = 9$$

Multipled

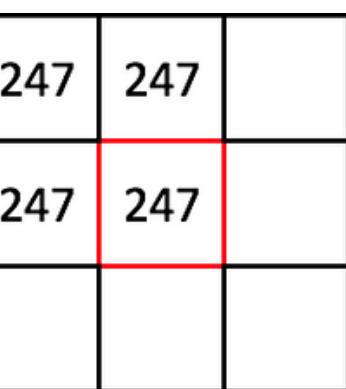
247	120	95
247	83	83
247	83	83

Sum = 1288

New Pixel
Value
 $88 / 9 = 14$
143

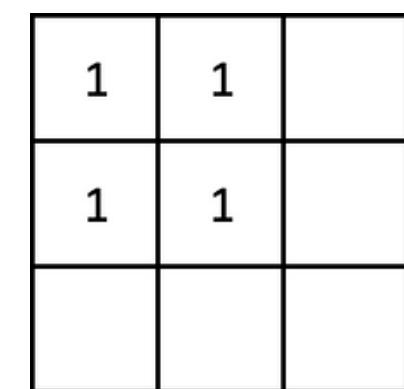
Kernel Convolutions

Window



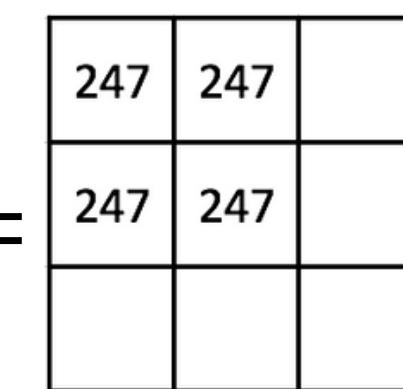
x

Kernel



1

Multiplied



Sum = 4

Sum = 988

New Pixel
Value
 $98 / 4 = 24$
247

Kernel Convolutions

Kernel

1	1	1
1	1	1
1	1	1



247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	233	216	216	230	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	215	180	178	211	246	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	196	143	141	192	245	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	192	138	135	182	231	234	241	247	247	247	247
247	247	247	247	247	242	224	224	174	138	135	164	196	198	224	247	247	247	247	247
247	247	247	247	236	199	199	156	138	135	146	160	163	207	247	247	247	247	247	247
247	247	247	247	229	175	175	138	138	135	135	138	141	195	247	247	247	247	247	247
247	247	247	247	228	173	173	138	138	135	135	138	141	195	247	247	247	247	247	247
247	247	247	247	228	173	161	125	125	123	123	134	149	203	247	247	247	247	247	247
247	247	247	247	234	181	151	109	107	109	118	146	175	221	247	247	247	247	247	247
247	247	247	247	241	206	175	127	107	109	136	182	210	238	247	247	247	247	247	247
247	247	247	247	247	231	212	158	119	121	166	221	238	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	247	245	245	246	247
247	247	247	247	246	246	246	240	188	137	141	191	242	245	247	245	245	243	244	242
247	247	247	247	246	246	246	240	207	171	175	206	242	245	247	245	245	243	244	242
247	247	247	247	246	246	246	240	225	207	212	225	243	245	247	247	247	247	244	242
247	247	247	247	247	247	247	247	247	247	242	239	239	244	247	247	247	247	247	247

Kernel Convolutions

Mean Blur

Kerne

1	1	1
1	1	1
1	1	1



247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	233	216	216	230	247	247	247	247	247	247
247	247	247	247	247	247	247	247	247	215	180	178	211	246	247	247	247	247	247
247	247	247	247	247	247	247	247	247	196	143	141	192	245	247	247	247	247	247
247	247	247	247	247	247	247	247	247	192	138	135	182	231	234	241	247	247	247
247	247	247	247	247	247	247	247	247	242	224	224	174	138	135	164	196	198	224
247	247	247	247	247	247	247	247	247	236	199	199	156	138	135	146	160	163	207
247	247	247	247	247	247	247	247	247	229	175	175	138	138	135	135	138	141	195
247	247	247	247	247	247	247	247	247	228	173	173	138	138	135	135	138	141	195
247	247	247	247	247	247	247	247	247	228	173	161	125	125	123	123	134	149	203
247	247	247	247	247	247	247	247	247	234	181	151	109	107	109	118	146	175	221
247	247	247	247	247	247	247	247	247	241	206	175	127	107	109	136	182	210	238
247	247	247	247	247	247	247	247	247	231	212	158	119	121	166	221	238	247	247
247	247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	247	247	247
247	247	247	247	247	247	247	247	247	192	138	135	190	244	247	247	245	245	246
247	247	247	247	247	246	246	246	240	188	137	141	191	242	245	247	245	245	243
247	247	247	247	247	246	246	246	240	207	171	175	206	242	245	247	245	245	244
247	247	247	247	247	246	246	246	240	225	207	212	225	243	245	247	247	247	242
247	247	247	247	247	247	247	247	247	246	246	246	240	242	247	247	244	244	242

Kernel Convolutions

Blur

Kernel

1	1	1
1	1	1
1	1	1



Kernel Convolutions

Sharpen

Kernel

0	-1	0
-1	5	-1
0	-1	0

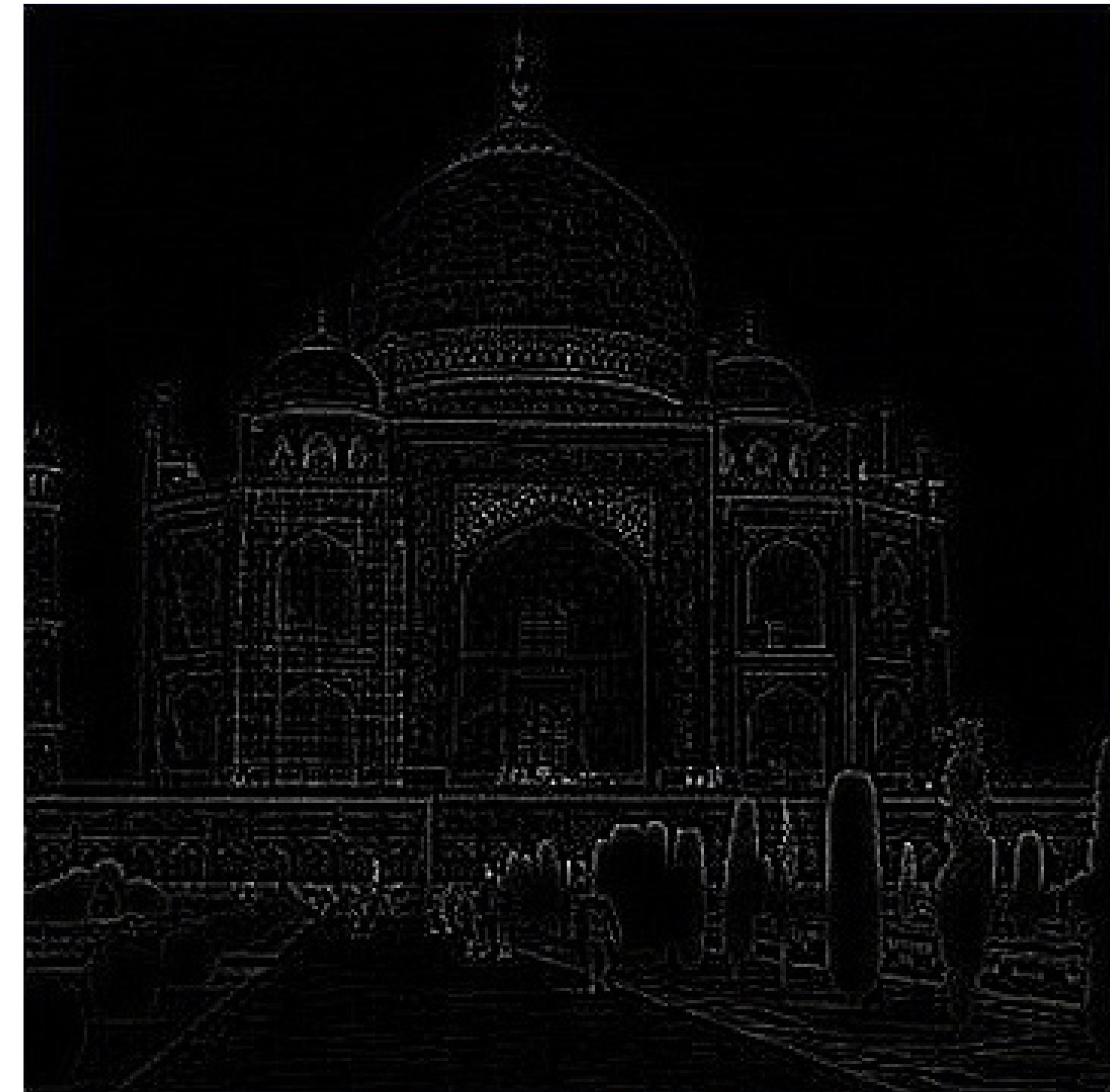


Kernel Convolutions

Edge
Highlight
Kernel



0	1	0
1	-4	1
0	1	0

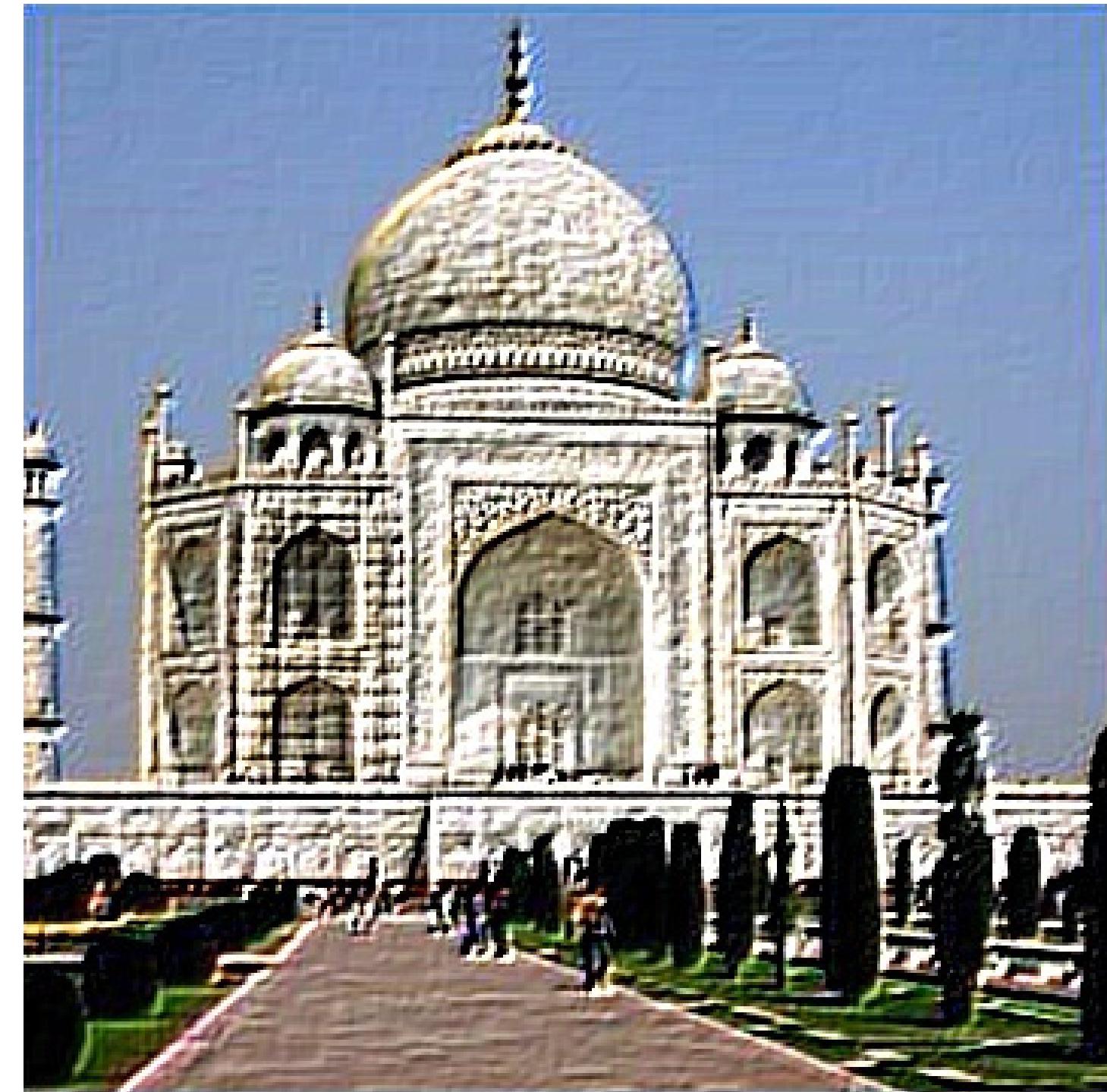


Kernel Convolutions

Emboss

Kernel

-2	-1	0
-1	1	1
0	1	2



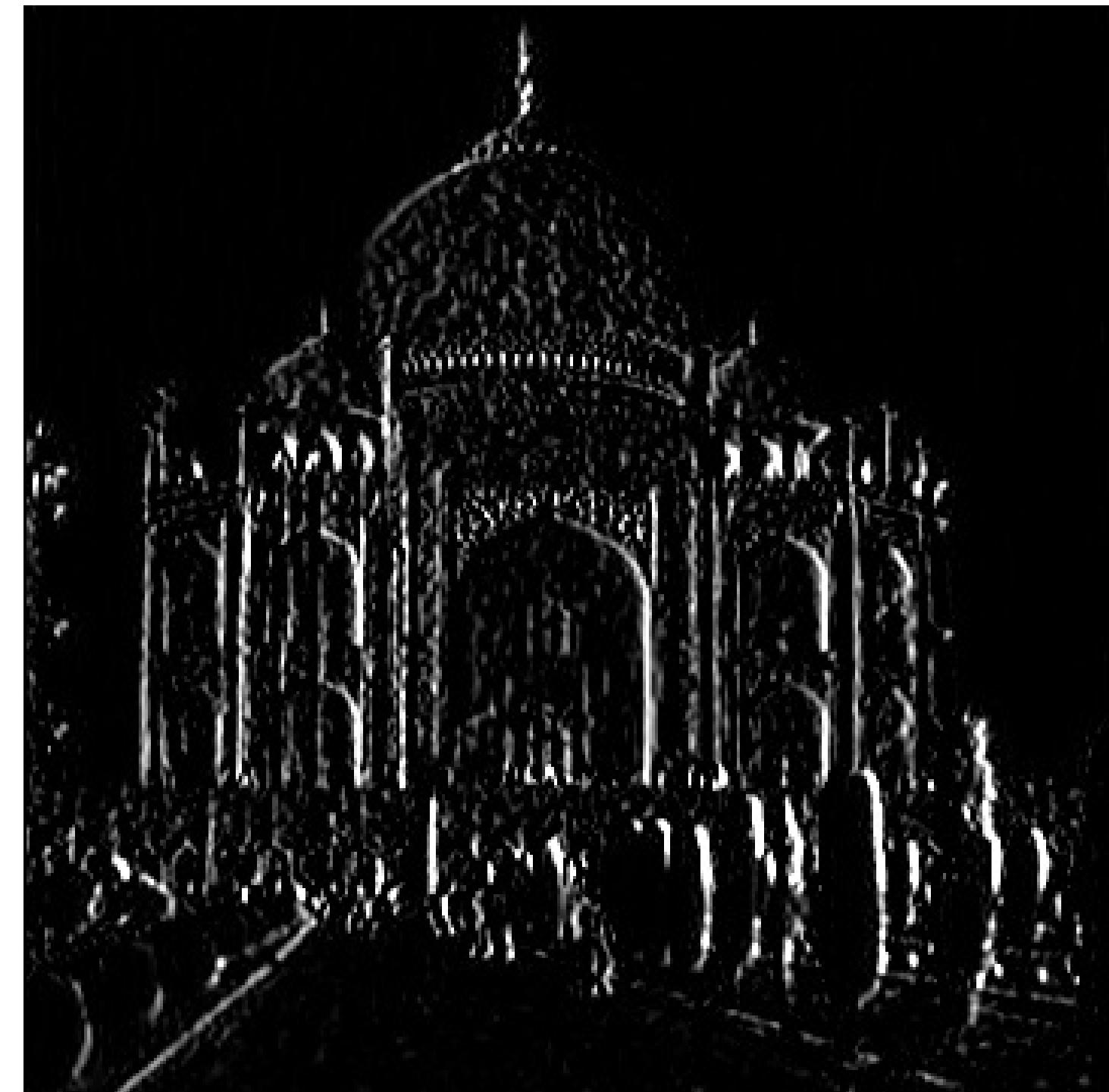
Kernel Convolutions

Horizontal “Derivative”



Kernel

1	0	-1
2	0	-2
1	0	-1



Kernel Convolutions

Vertical “Derivative”



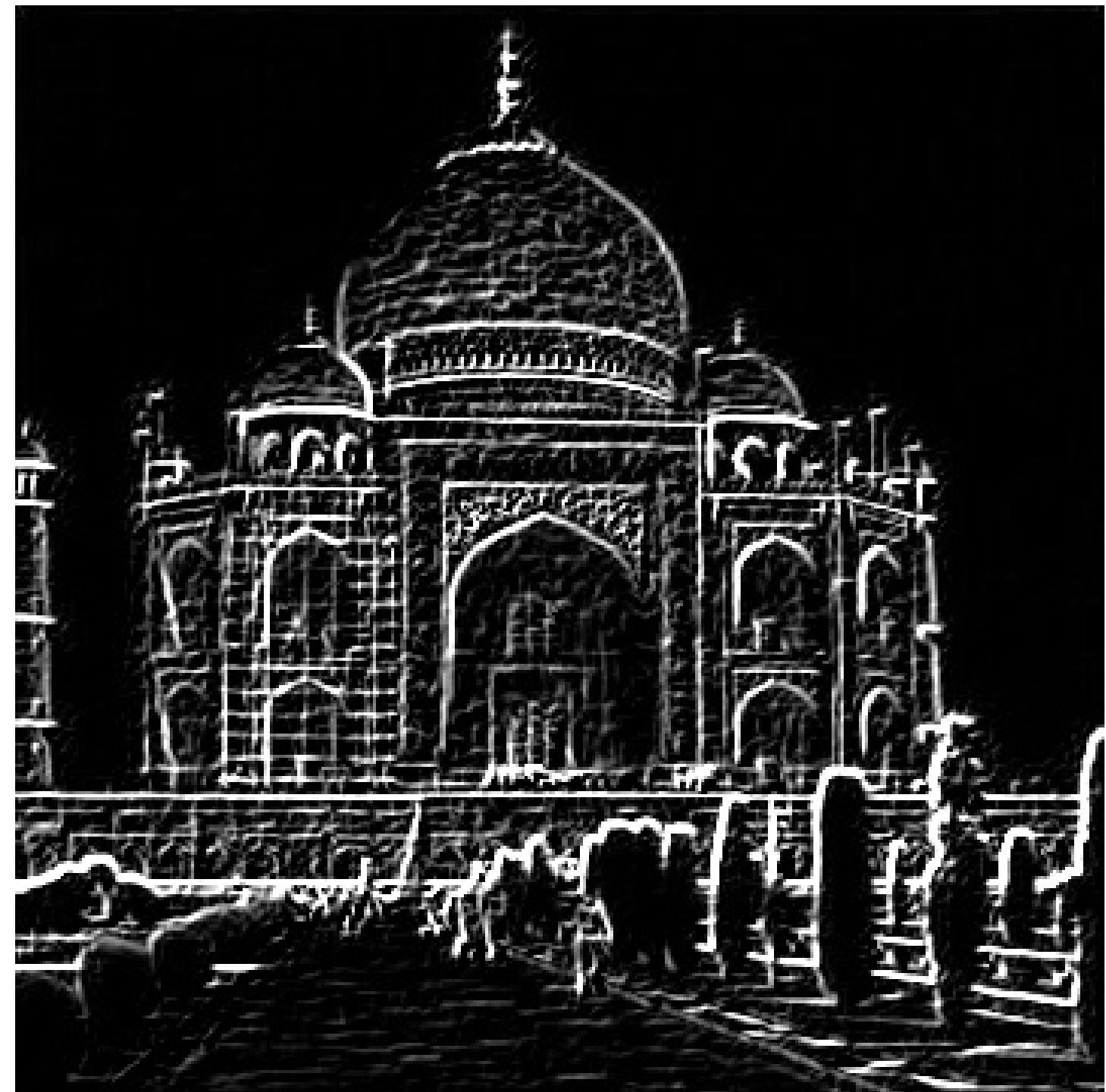
Kernel

1	2	1
0	0	0
-1	-2	-1



Kernel Convolutions

Overlay them to get **Sobel Operator**



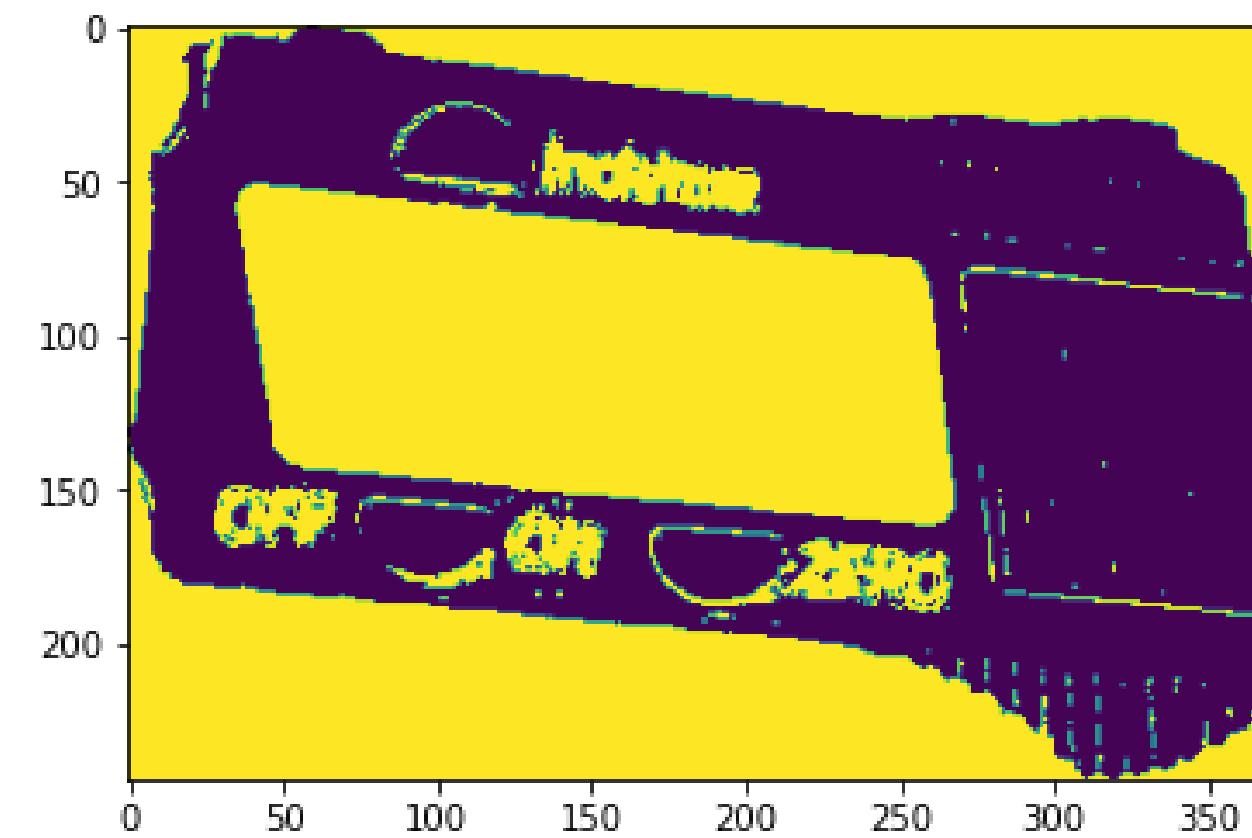
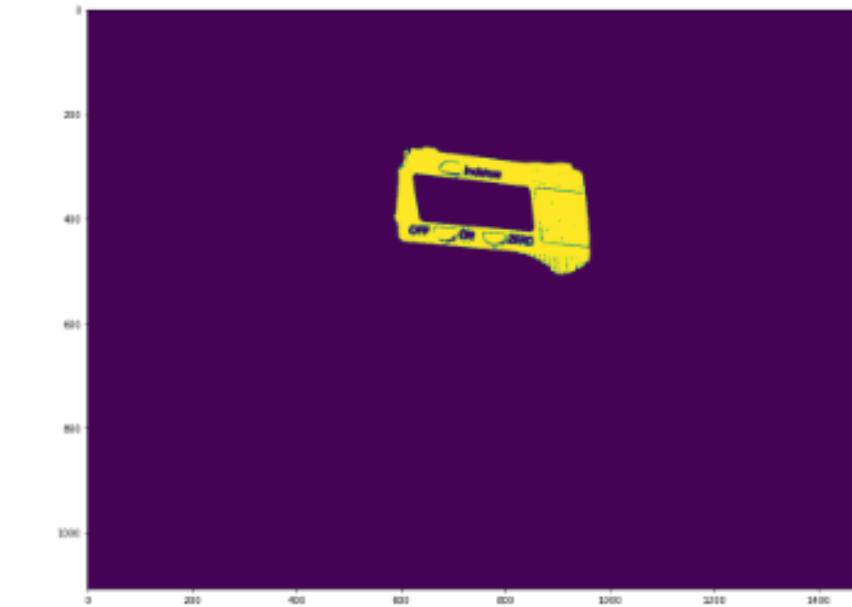
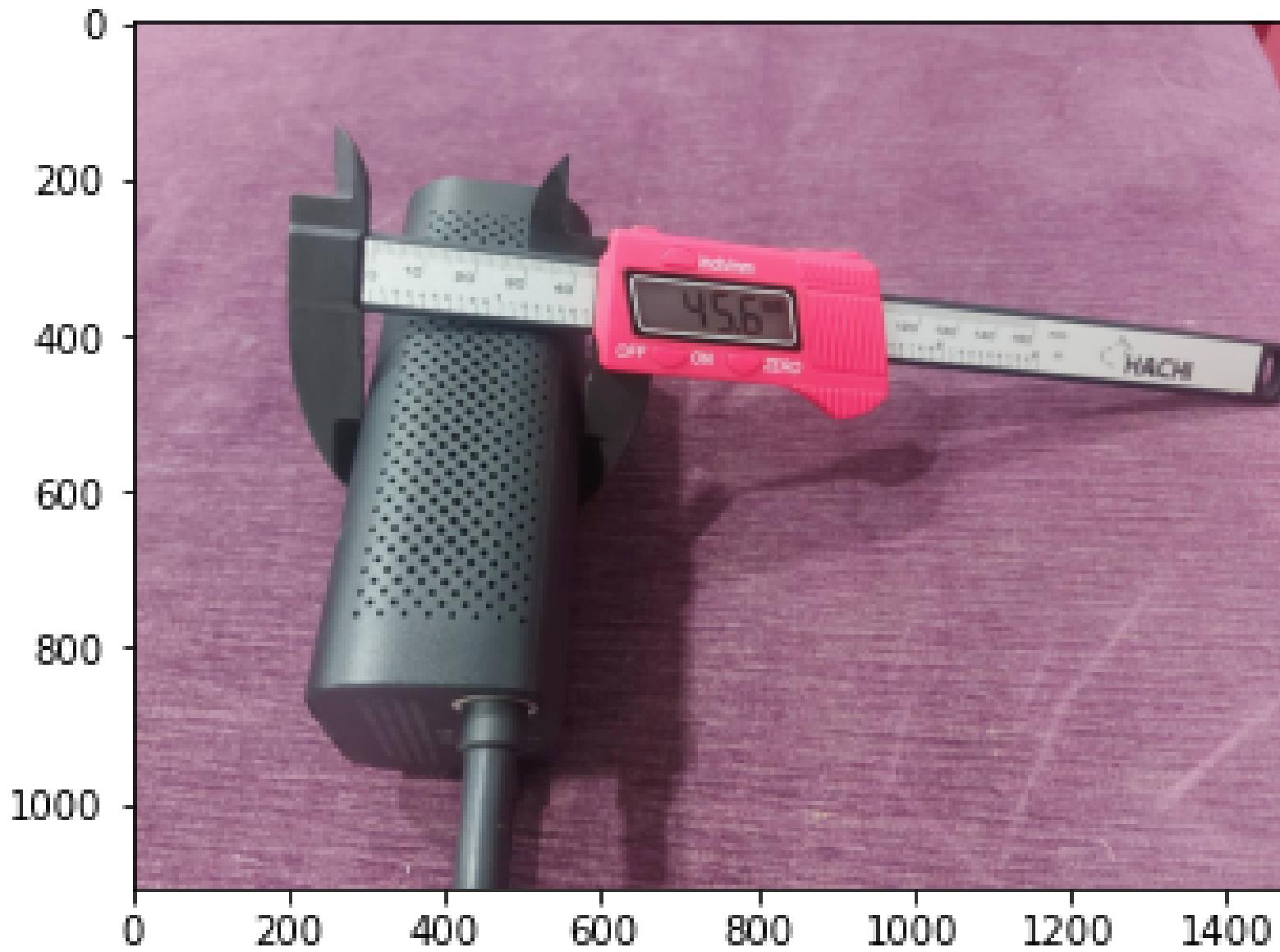
- **Dilation** – Adds pixels to the boundaries of objects in an image
 - **Erosion** – Removes pixels at the boundaries of objects in an image
 - **Opening** - Erosion followed by dilation
 - **Closing** - Dilation followed by erosion

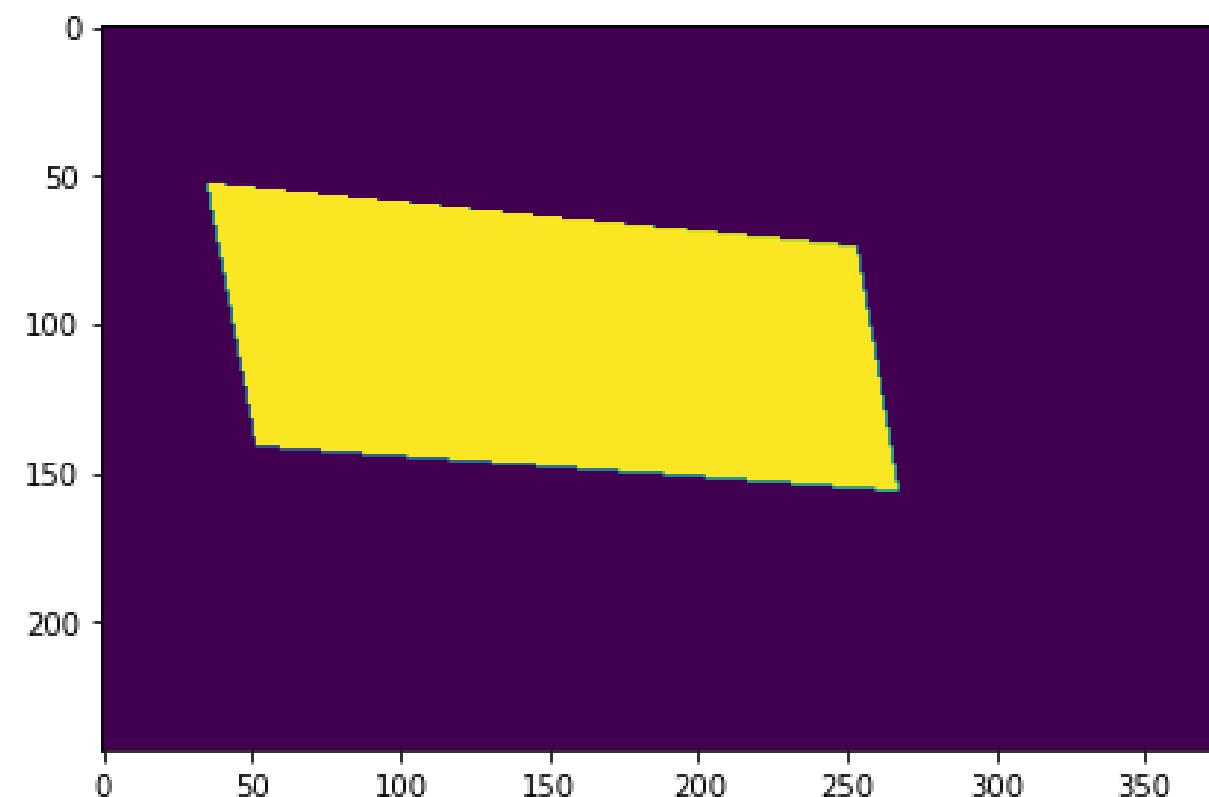
Original

Erosion

Dilation

Home Works





```
: import easyocr
reader = easyocr.Reader(['en'])

:
bounds = reader.readtext(img5)
bounds

[[[[48, 0], [166, 0], [166, 62], [48, 62]], '4561', 0.29313910007476807]]

:
bounds[0][1]

: '4561'
```