APPLIED COMPUTER VISION IN UBICOMP 2

CSE 590 Ubiquitous Computing | Lecture 7 | May 10

Jon Froehlich • Liang He (TA)

DESIGN USE BUILD

MAKEABILITY LAB

PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

UNIVERSITY of WASHINGTON

SCHEDULE TODAY: 6:30-9:20

06:30-06:50: Discussion of required reading led by Tim Sawyer

06:50-06:55: Optional reading discussion led by Pankaj Parag

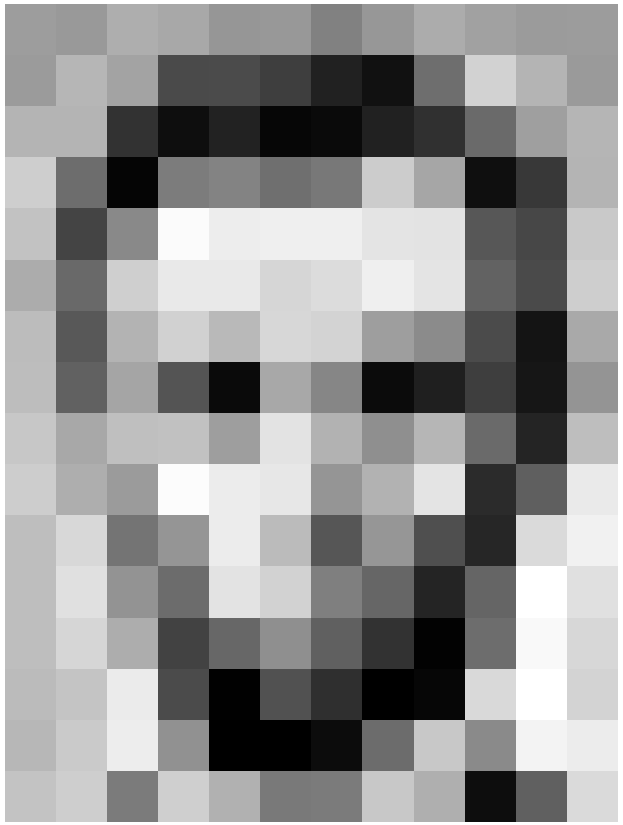
06:55-08:00: Basic Computer Vision

08:00-08:05: Break

08:05-08:40: Continue Computer Vision Exercises

08:40-09:20: Work on A4 and show proper wiring with external battery

DIGITAL IMAGES ARE MATRICES OF PIXEL VALUES



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Each cell is **0-255** where 0 is black and 255 is white. Cells could also be normalized to **0.0-1.0** again corresponding to intensity (0.0=black, 1.0=white)

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Source: http://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

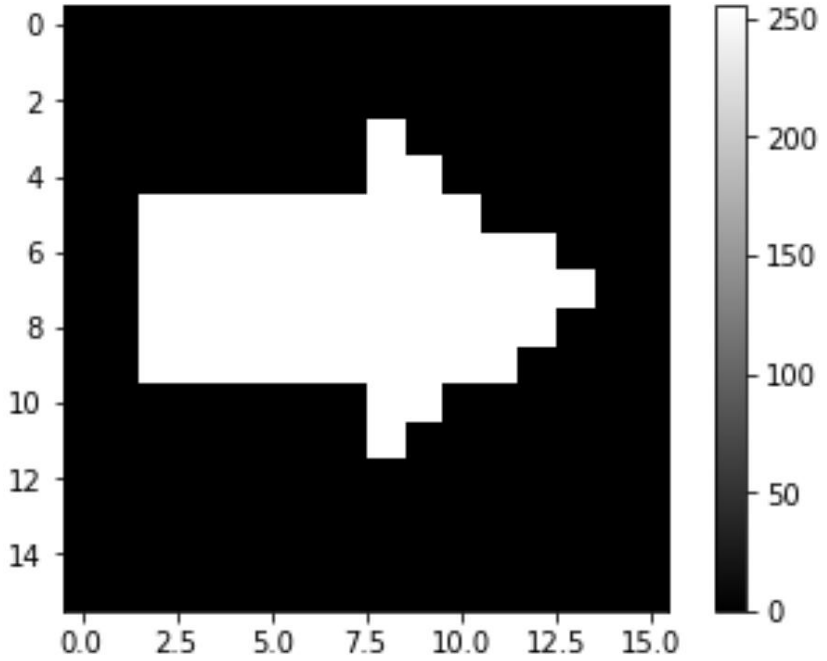
WHAT IS THIS IMAGE?

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

WHAT IS THIS IMAGE?

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

Black & white image of an arrow



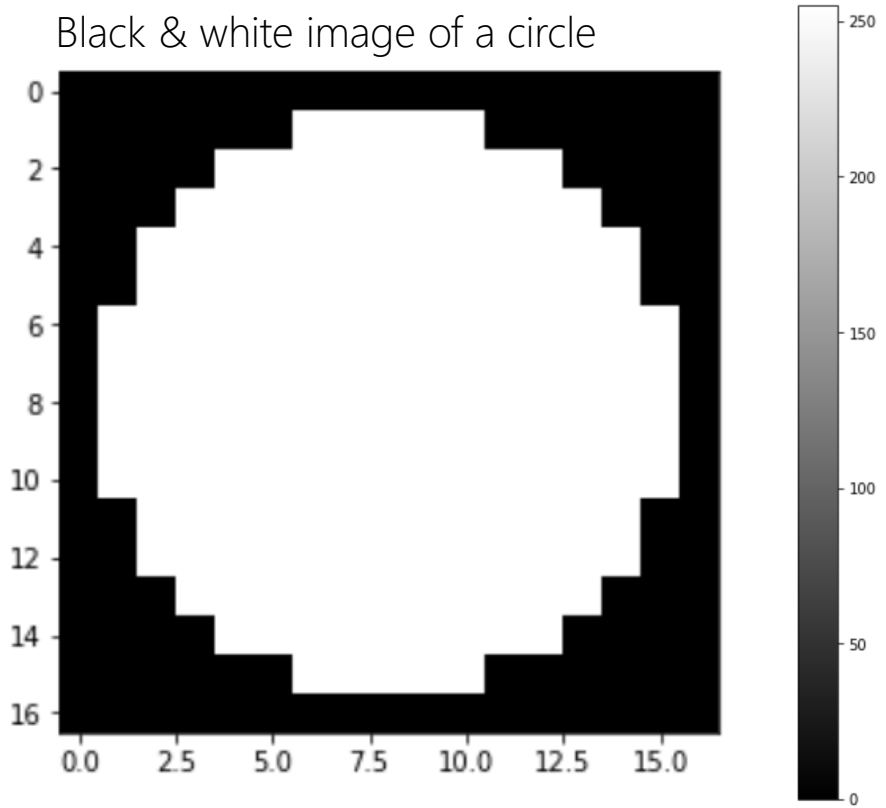
WHAT IS THIS IMAGE?

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

WHAT IS THIS IMAGE?

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

Black & white image of a circle



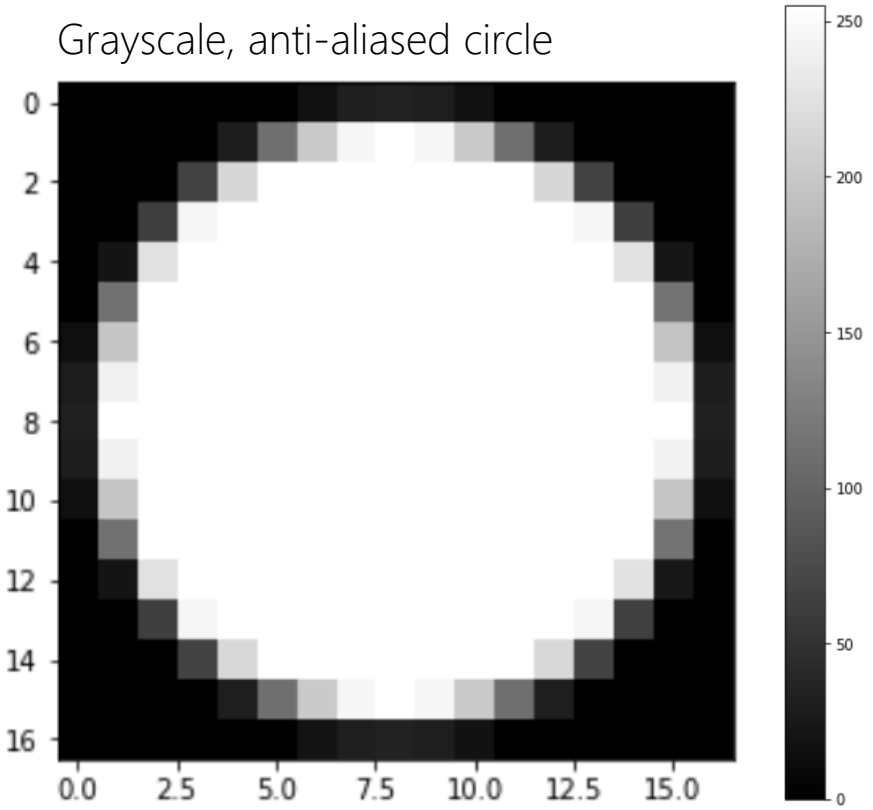
WHAT IS THIS IMAGE?

```
[  0   0   0   0   0   0  18  30  34  30  18   0   0   0   0   0   0]
[  0   0   0   0  29 111 201 246 255 246 201 111  29   0   0   0   0]
[  0   0   0  67 215 255 255 255 255 255 255 255 215  67   0   0   0]
[  0   0  62 246 255 255 255 255 255 255 255 255 255 246  63   0   0]
[  0  20 225 255 255 255 255 255 255 255 255 255 255 255 226  22   0]
[  0 113 255 255 255 255 255 255 255 255 255 255 255 255 255 115   0]
[ 15 197 255 255 255 255 255 255 255 255 255 255 255 255 255 196  15]
[ 28 240 255 255 255 255 255 255 255 255 255 255 255 255 255 241  29]
[ 33 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255  33]
[ 29 241 255 255 255 255 255 255 255 255 255 255 255 255 255 242  29]
[ 15 197 255 255 255 255 255 255 255 255 255 255 255 255 255 197  15]
[  0 113 255 255 255 255 255 255 255 255 255 255 255 255 255 115   0]
[  0  20 225 255 255 255 255 255 255 255 255 255 255 255 226  23   0]
[  0   0  63 246 255 255 255 255 255 255 255 255 255 246  64   0   0]
[  0   0   0  67 216 255 255 255 255 255 255 255 216  67   0   0   0]
[  0   0   0   0  30 111 202 247 255 246 201 111  30   0   0   0   0]
[  0   0   0   0   0   0  19  30  34  30  18   0   0   0   0   0   0]
```

WHAT IS THIS IMAGE?

```
[  0   0   0   0   0   0  18  30  34  30  18   0   0   0   0   0   0]
[  0   0   0   0  29 111 201 246 255 246 201 111  29   0   0   0   0]
[  0   0   0  67 215 255 255 255 255 255 255 255 215  67   0   0   0]
[  0   0  62 246 255 255 255 255 255 255 255 255 255 246  63   0   0]
[  0  20 225 255 255 255 255 255 255 255 255 255 255 255 226  22   0]
[  0 113 255 255 255 255 255 255 255 255 255 255 255 255 255 115   0]
[ 15 197 255 255 255 255 255 255 255 255 255 255 255 255 255 196  15]
[ 28 240 255 255 255 255 255 255 255 255 255 255 255 255 255 241  29]
[ 33 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255  33]
[ 29 241 255 255 255 255 255 255 255 255 255 255 255 255 255 242  29]
[ 15 197 255 255 255 255 255 255 255 255 255 255 255 255 255 197  15]
[  0 113 255 255 255 255 255 255 255 255 255 255 255 255 255 115   0]
[  0  20 225 255 255 255 255 255 255 255 255 255 255 255 226  23   0]
[  0   0  63 246 255 255 255 255 255 255 255 255 255 246  64   0   0]
[  0   0   0  67 216 255 255 255 255 255 255 255 216  67   0   0   0]
[  0   0   0   0  30 111 202 247 255 246 201 111  30   0   0   0   0]
[  0   0   0   0   0   0  19  30  34  30  18   0   0   0   0   0   0]
```

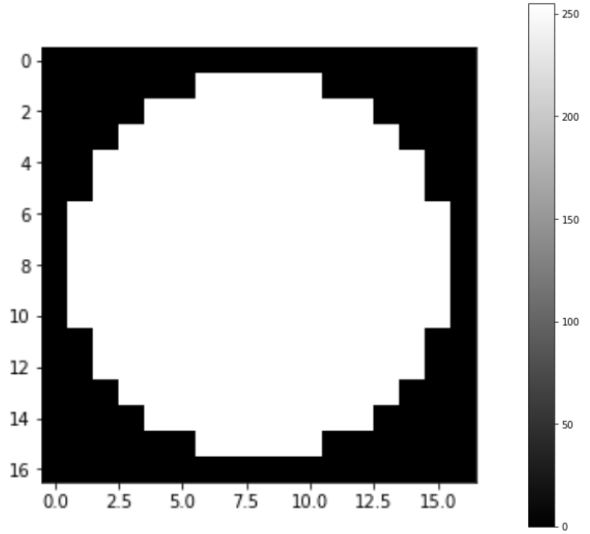
Grayscale, anti-aliased circle



WHAT IS THIS IMAGE?

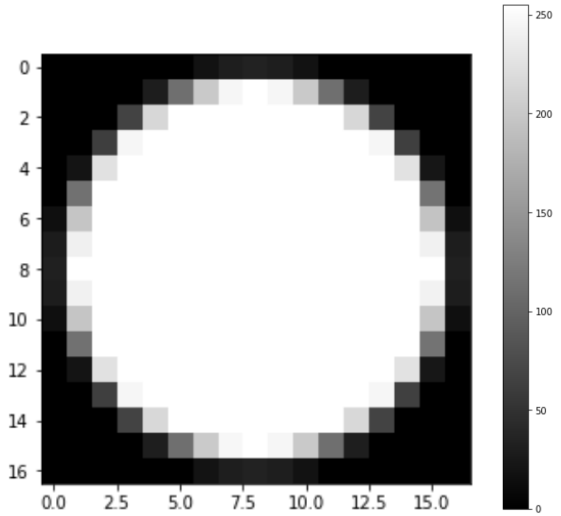
Black & white circle

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



Anti-aliased circle

```
[  0   0   0   0   0   0  18  30  34  30  18   0   0   0   0   0   0]
[  0   0   0   0  29 111 201 246 255 246 201 111  29   0   0   0   0]
[  0   0   0  67 215 255 255 255 255 255 255 255 215  67   0   0   0]
[  0   0  62 246 255 255 255 255 255 255 255 255 255 246  63   0   0]
[  0  20 225 255 255 255 255 255 255 255 255 255 255 255 226  22   0]
[  0 113 255 255 255 255 255 255 255 255 255 255 255 255 255 115   0]
[ 15 197 255 255 255 255 255 255 255 255 255 255 255 255 255 196  15]
[ 28 240 255 255 255 255 255 255 255 255 255 255 255 255 255 241  29]
[ 33 255 255 255 255 255 255 255 255 255 255 255 255 255 255 255  33]
[ 29 241 255 255 255 255 255 255 255 255 255 255 255 255 255 242  29]
[ 15 197 255 255 255 255 255 255 255 255 255 255 255 255 255 197  15]
[  0 113 255 255 255 255 255 255 255 255 255 255 255 255 255 115   0]
[  0  20 225 255 255 255 255 255 255 255 255 255 255 255 226  23   0]
[  0   0  63 246 255 255 255 255 255 255 255 255 246  64   0   0   0]
[  0   0   0  67 216 255 255 255 255 255 255 255 216  67   0   0   0]
[  0   0   0   0  30 111 202 247 255 246 201 111  30   0   0   0   0]
[  0   0   0   0   0   0  19  30  34  30  18   0   0   0   0   0   0]
```



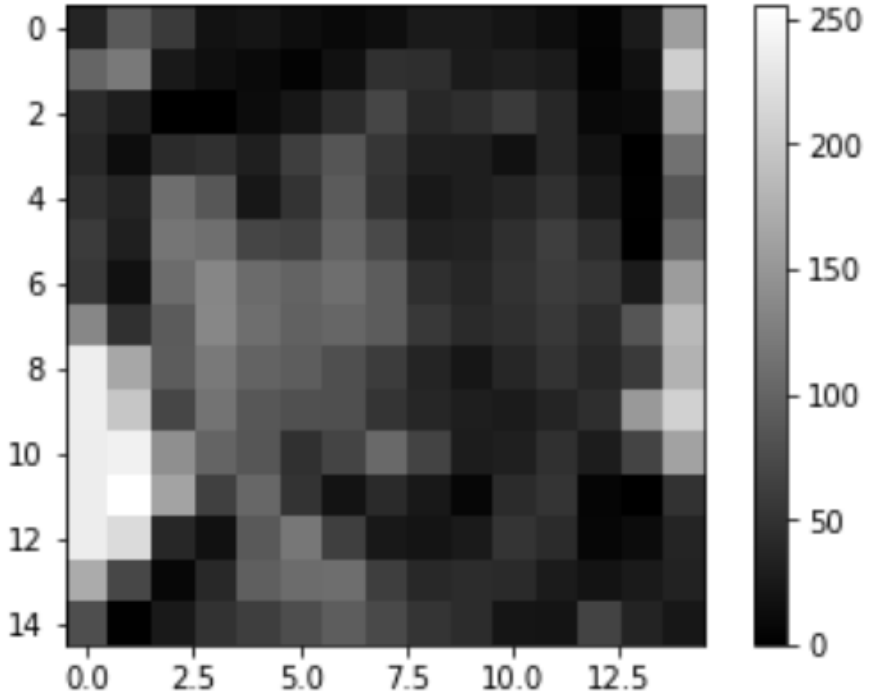
LAST ONE: WHAT IS THIS IMAGE?

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

WHAT IS THIS IMAGE?

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

15x15 grayscale

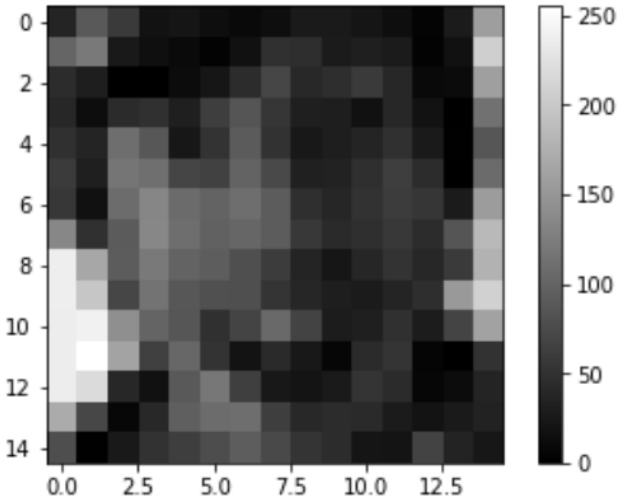


WHAT ARE DIGITAL IMAGES?

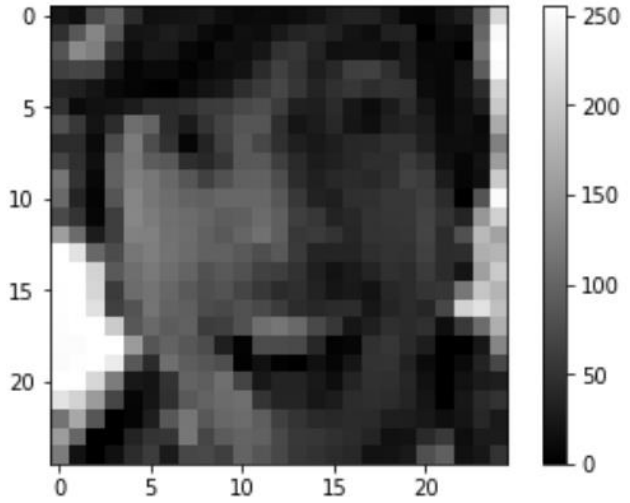
CAN ANYONE TELL YET?

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

15x15 grayscale



25x25 grayscale

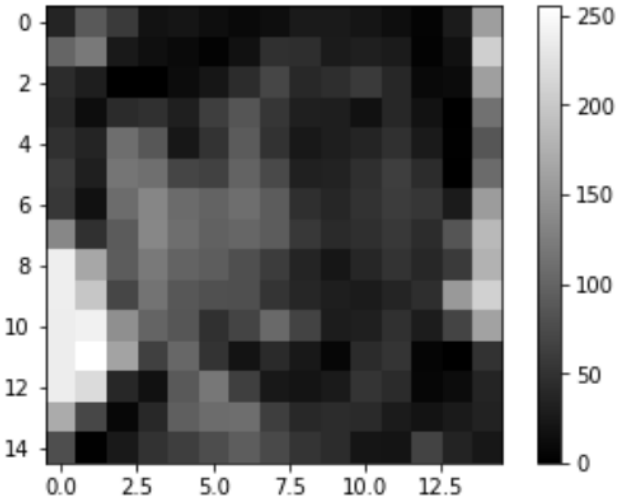


WHAT ARE DIGITAL IMAGES?

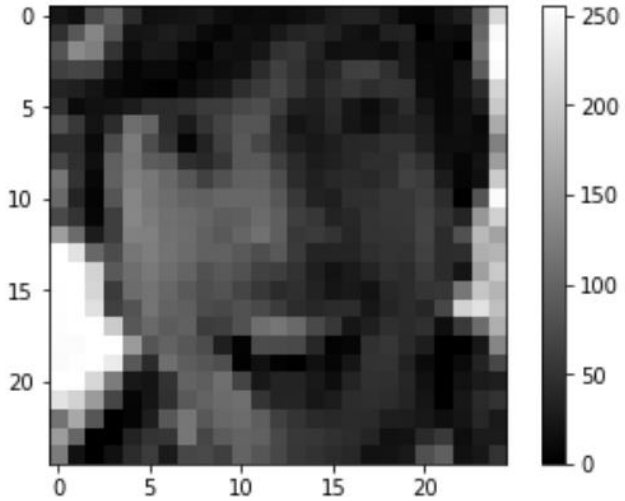
CAN ANYONE TELL YET?

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

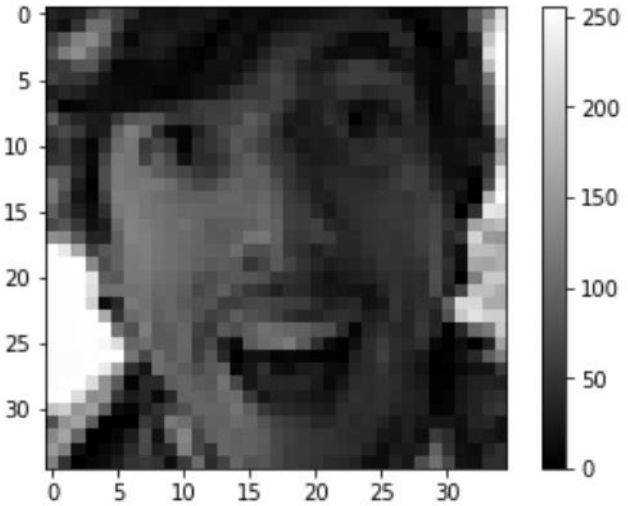
15x15 grayscale



25x25 grayscale



35x35 grayscale

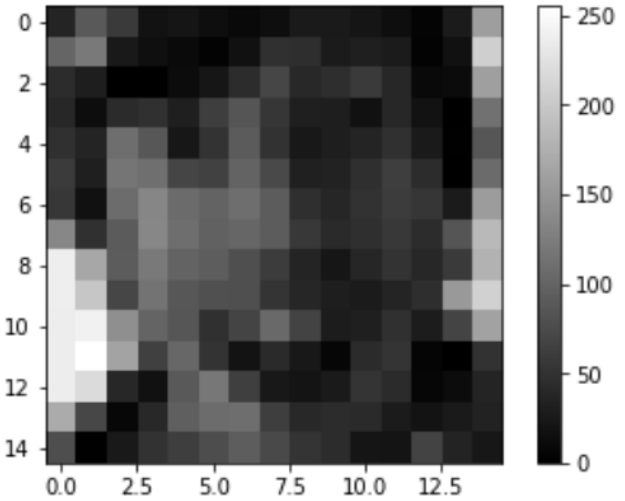


WHAT ARE DIGITAL IMAGES?

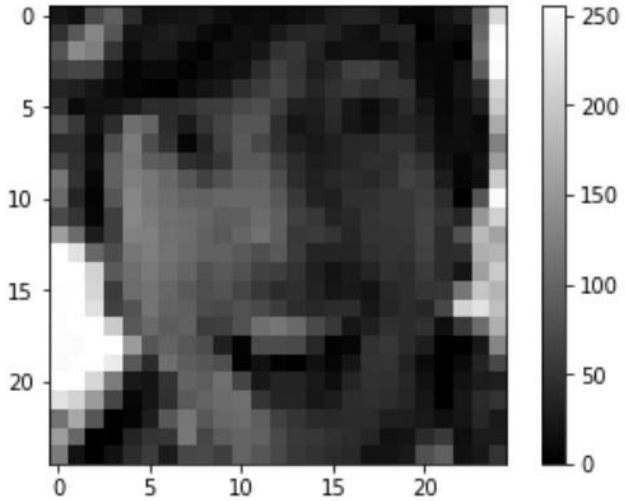
CAN ANYONE TELL YET?

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

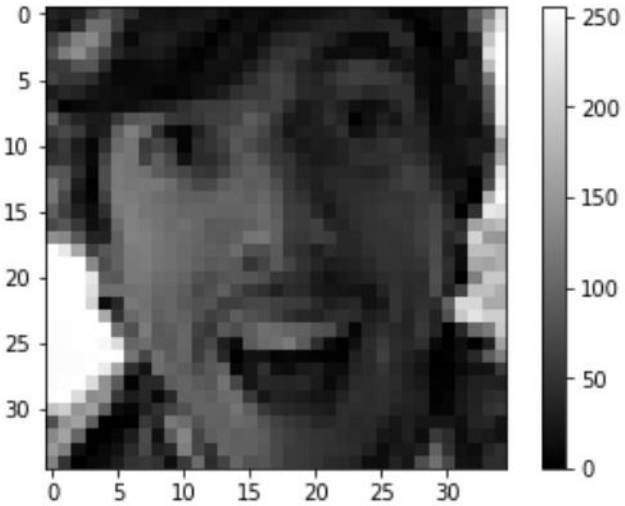
15x15 grayscale



25x25 grayscale

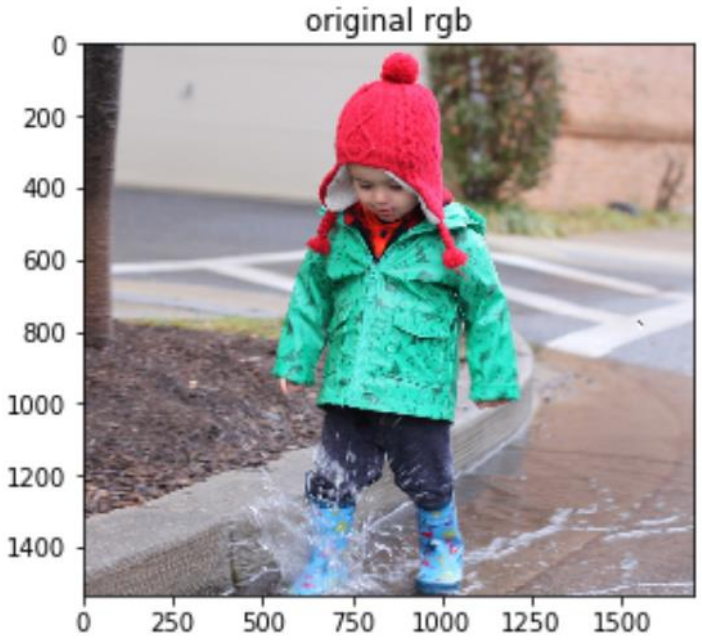


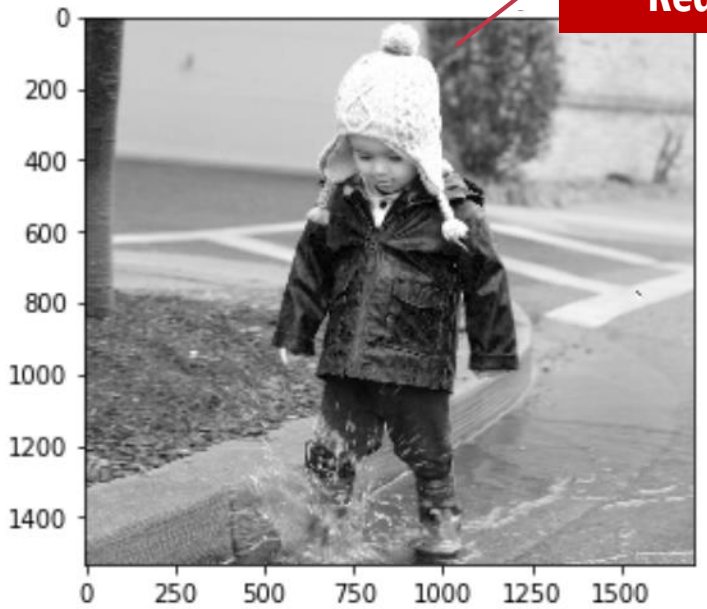
35x35 grayscale



RGB IMAGES

RGB images have three channels (red, green, blue). Each channel has intensities from 0-255.

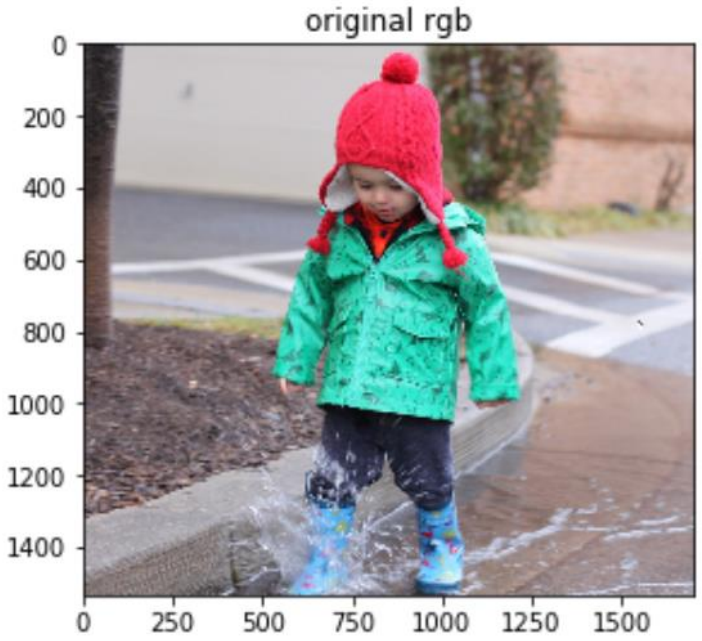


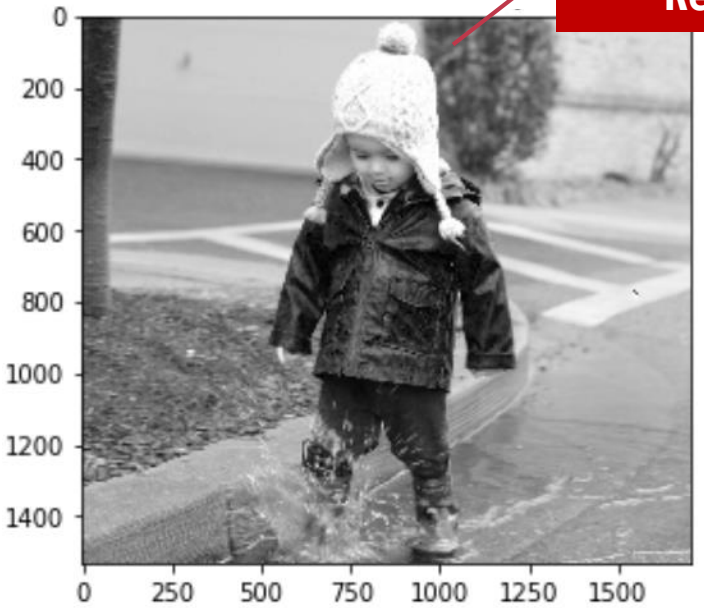


What do you think I'm graphing here?
Red intensities, **green**, or **blue**?

RGB IMAGES

RGB images have three channels (red, green, blue). Each channel has intensities from 0-255.





What do you think I'm graphing here?
Red channel, **green**, or **blue**?

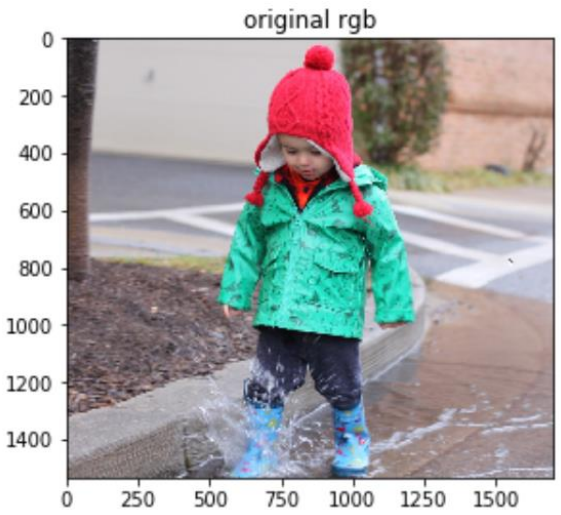
Answer:

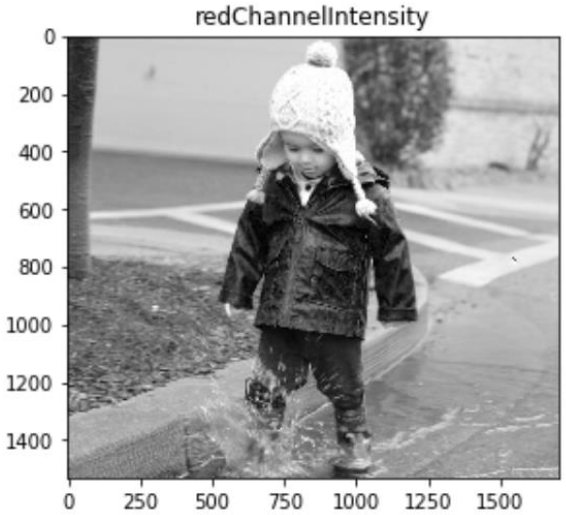
This must be the red channel because the highest intensity areas (*i.e.*, values closest to 255) map to the red areas of the original image

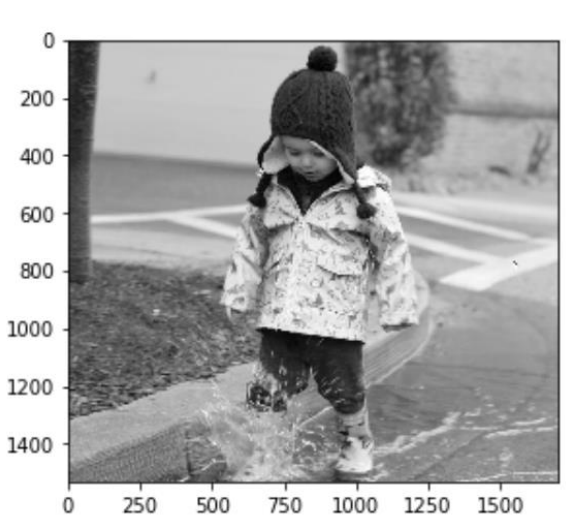
WHAT ARE DIGITAL IMAGES

RGB IMAGES

RGB images have three channels (red, green, blue). Each channel has intensities from 0-255.



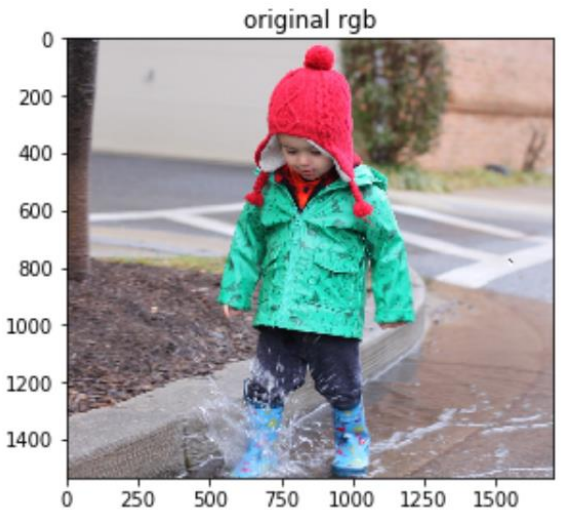


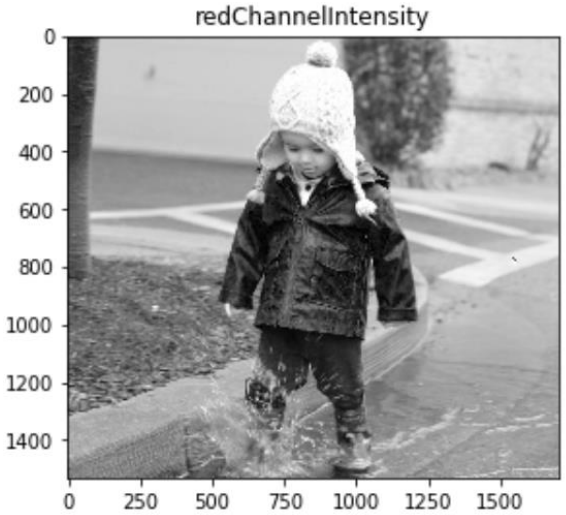


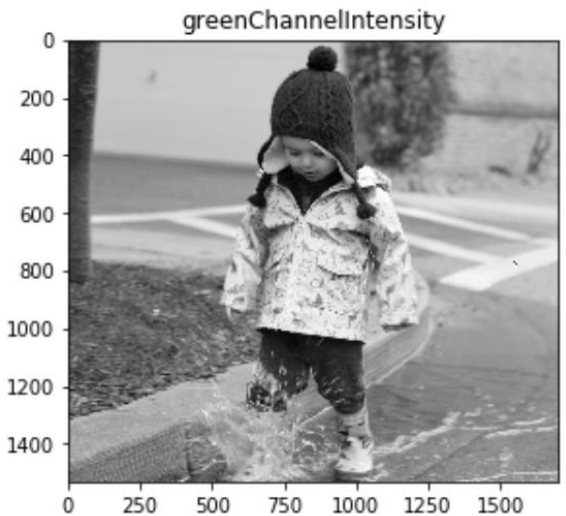
WHAT ARE DIGITAL IMAGES

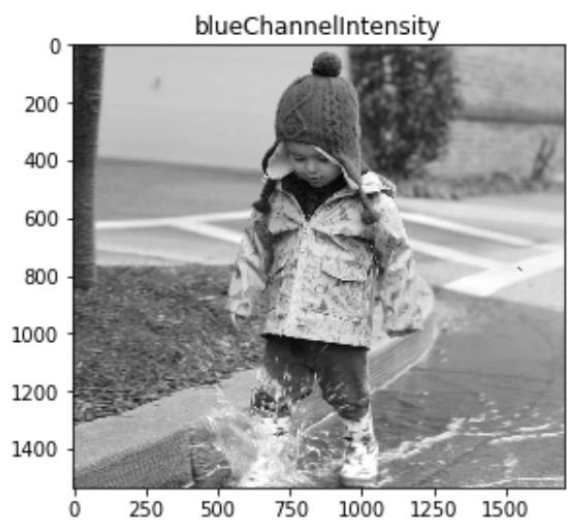
RGB IMAGES

RGB images have three channels (red, green, blue). Each channel has intensities from 0-255.





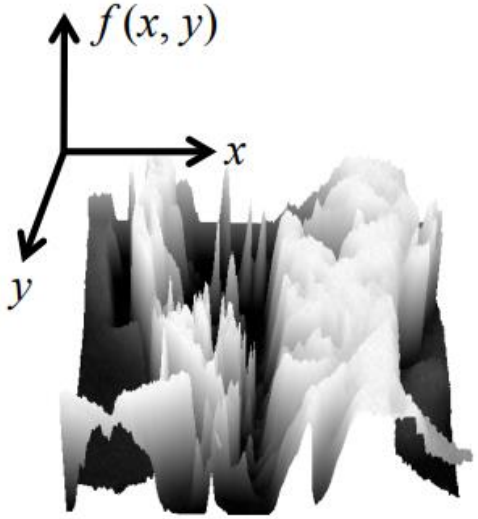




DIGITAL IMAGES AS FUNCTIONS

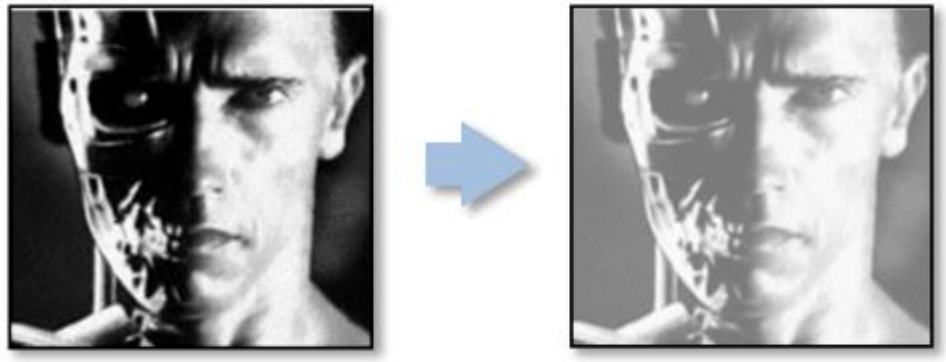
In addition to thinking about images as matrices, we can think of them as a function $f = \mathbb{R}^2 \rightarrow \mathbb{R}$ that maps real numbers in two-dimensional space into intensity values.

A digital image is a discrete (sampled, quantized) version of this function



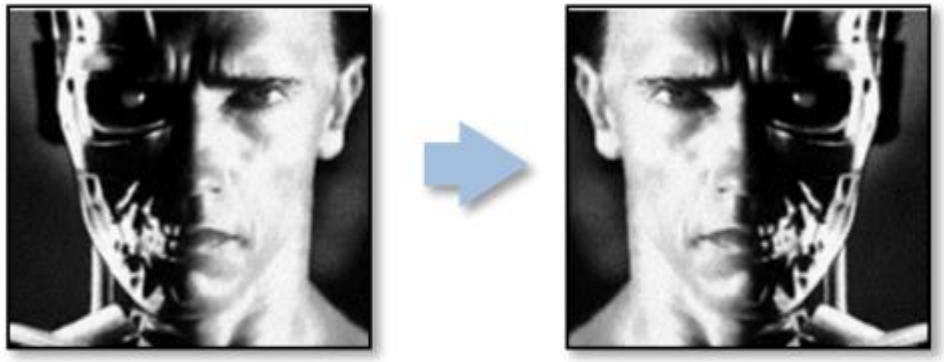
AS WITH ANY FUNCTION, WE CAN APPLY OPERATORS

Increase the intensity of each pixel by 20



$$g(x,y) = f(x,y) + 20$$

Flip the image along the horizontal



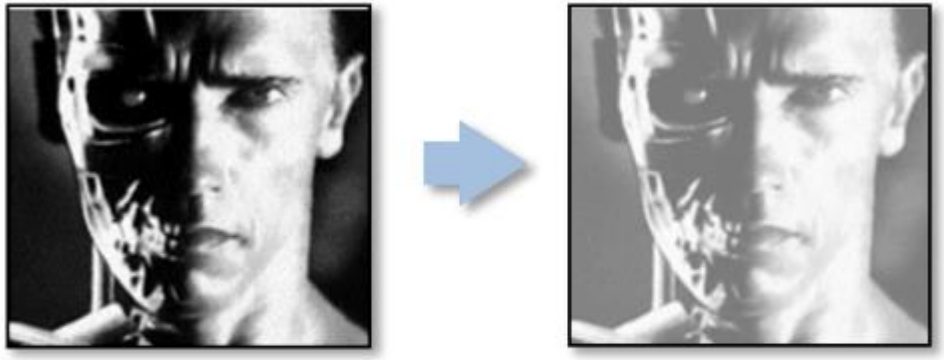
$$g(x,y) = f(-x,y)$$

g(x,y) defines a new image ***g*** in terms of an existing image ***f(x,y)***

Based on slides by Noah Snavely at Cornell: http://www.cs.cornell.edu/courses/cs5670/2018sp/lectures/lectures.html & Raquel Urtasun at TTI: https://www.cs.toronto.edu/~urtasun/courses/CV/lecture02.pdf

FORMALIZING THIS A BIT MORE…

Increase the intensity of each pixel by 20



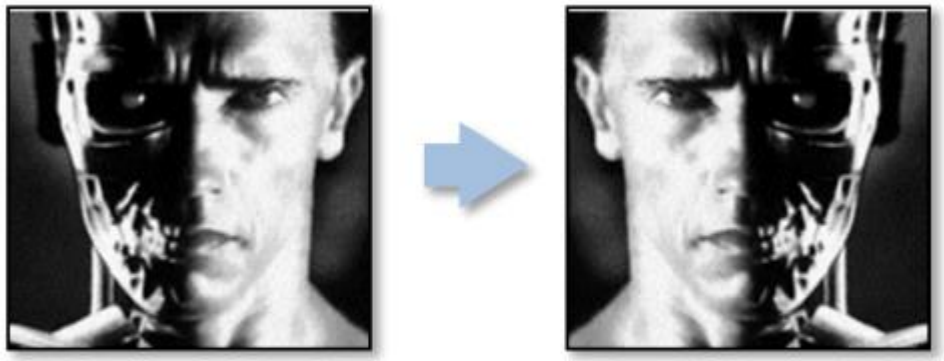
$$g(x,y) = f(x,y) + 20$$

Range Transformations:

Transform the output of f(x,y)

$$g(x,y) = t(f(x,y))$$

Flip the image along the horizontal



$$g(x,y) = f(-x,y)$$

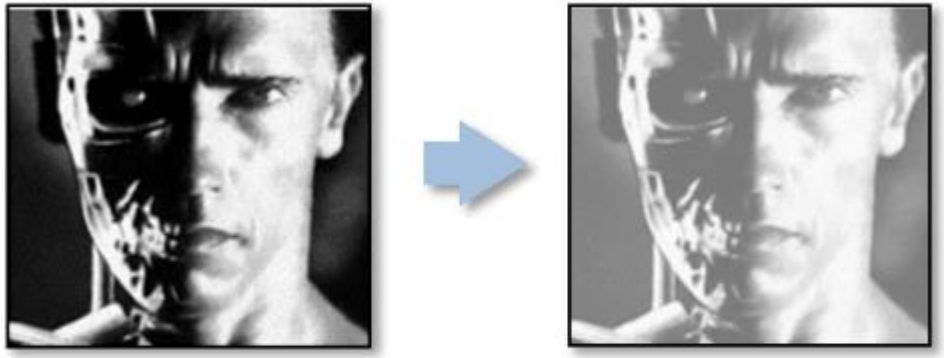
Domain Transformations:

Transform the input to f(x,y)

$$g(x,y) = f(t_x(x,y), t_y(x,y))$$

FORMALIZING THIS A BIT MORE…

Increase the intensity of each pixel by 20



$$g(x,y) = f(x,y) + 20$$

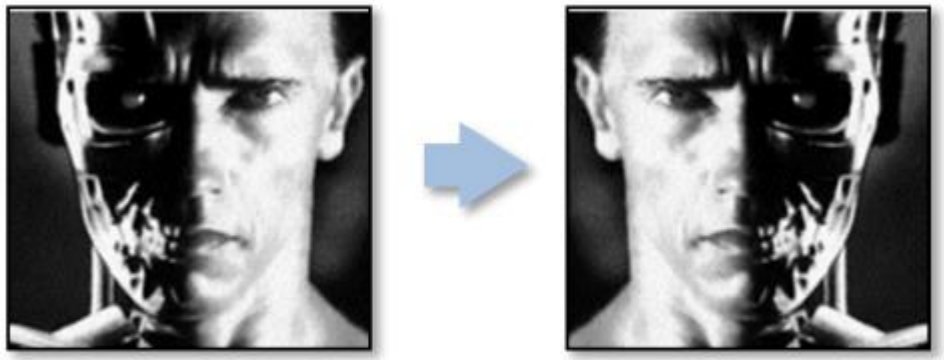
Range Transformations:

Transform the output of f(x,y)

$$g(x,y) = t(f(x,y))$$

e.g., inversion, grayscale, brightness, contrast

Flip the image along the horizontal



$$g(x,y) = f(-x,y)$$

Domain Transformations:

Transform the input to f(x,y)

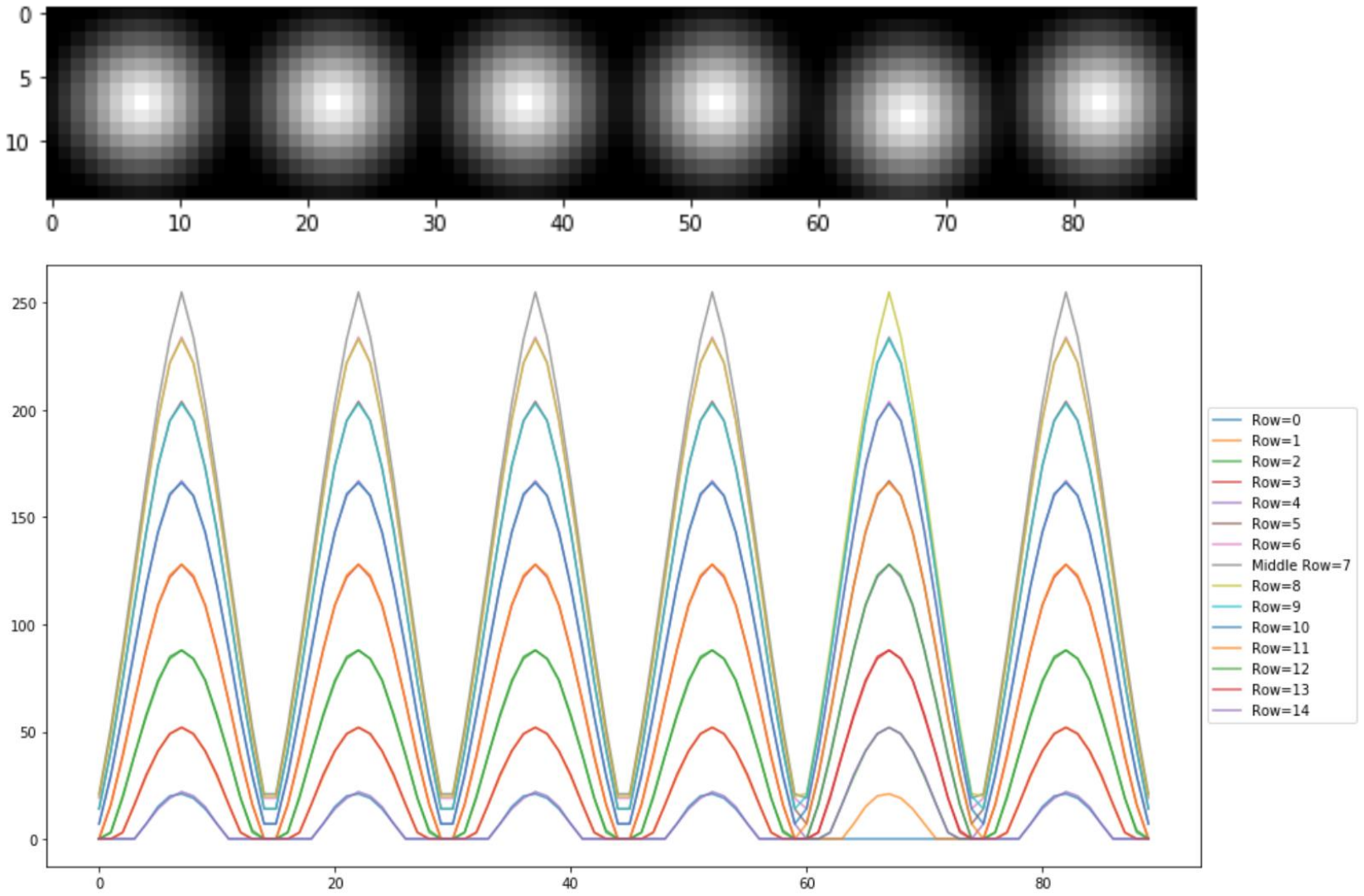
$$g(x,y) = f(t_x(x,y), t_y(x,y))$$

e.g., zoom, translate, rotate, shear,

WE CAN GRAPH & APPLY SIGNAL PROCESSING TECHNIQUES

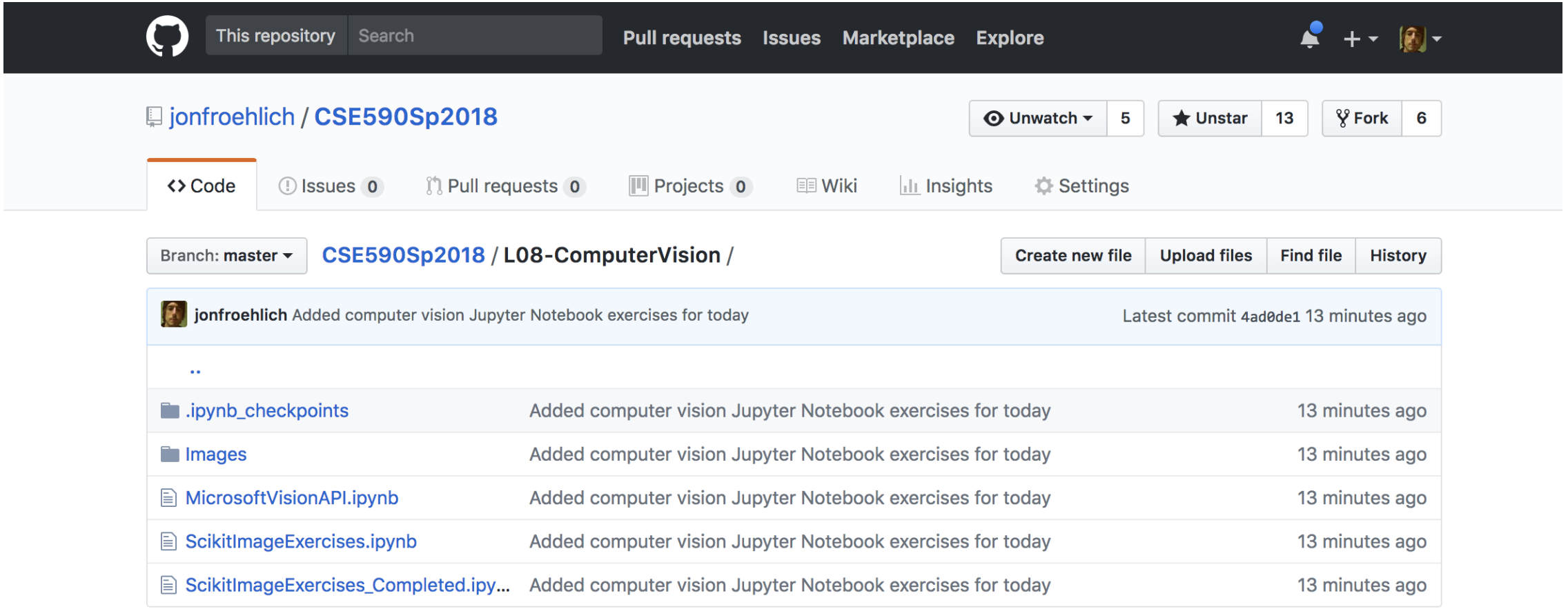
Just as we did in previous parts of the course, we can plot, analyze, and **perform signal processing on images** (peak finding, smoothing, thresholding, and far more complicated operations)

Here, I'm graphing the intensity values row-by-row.



LET'S DO SOME BASIC IMAGE PROCESSING EXERCISES!

In Jupyter Notebook, I want you to perform the following image processing tasks using basic pixel-point arithmetic operations. **Do work** with a partner (if you want). **Try not** to Google for answers (I want you to *think* about this!). ☺

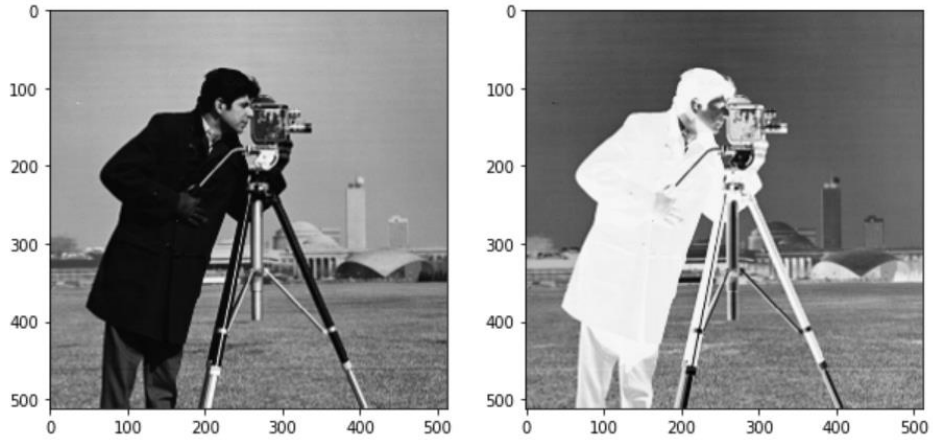


https://github.com/jonfroehlich/CSE590Sp2018/tree/master/L08-ComputerVision

LET'S DO SOME BASIC IMAGE PROCESSING EXERCISES!

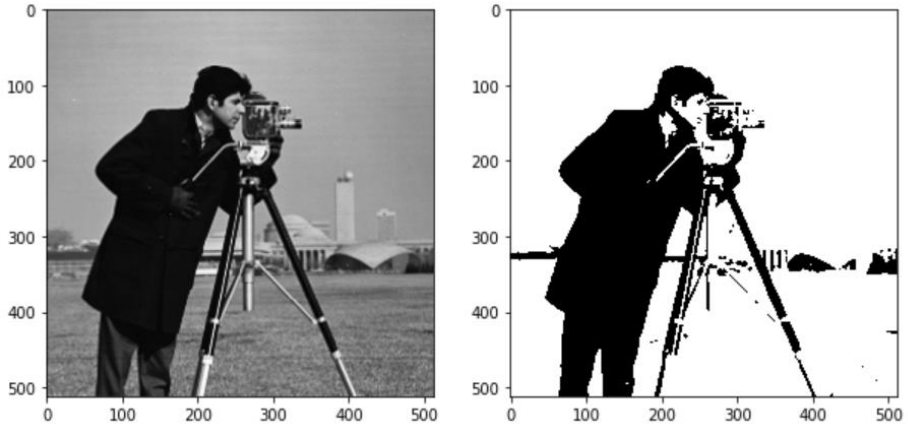
In Jupyter Notebook, I want you to perform the following image processing tasks using basic pixel-point arithmetic operations. **Do work** with a partner (if you want). **Try not** to Google for answers (I want you to *think* about this!). ☺

Invert an image



Binarize an image

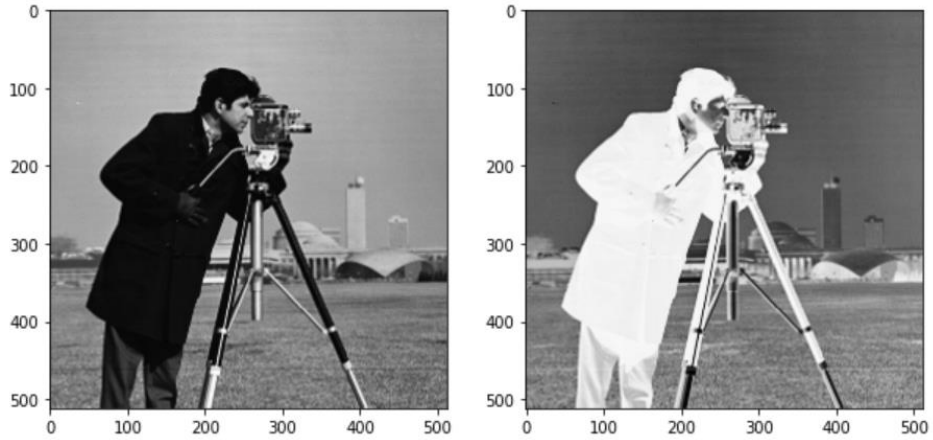
(*i.e.,* convert all pixels to either 255 or 0 depending on some threshold)



LET'S DO SOME BASIC IMAGE PROCESSING EXERCISES!

In Jupyter Notebook, I want you to perform the following image processing tasks using basic pixel-point arithmetic operations. **Do work** with a partner (if you want). **Try not** to Google for answers (I want you to *think* about this!). ☺

Invert an image

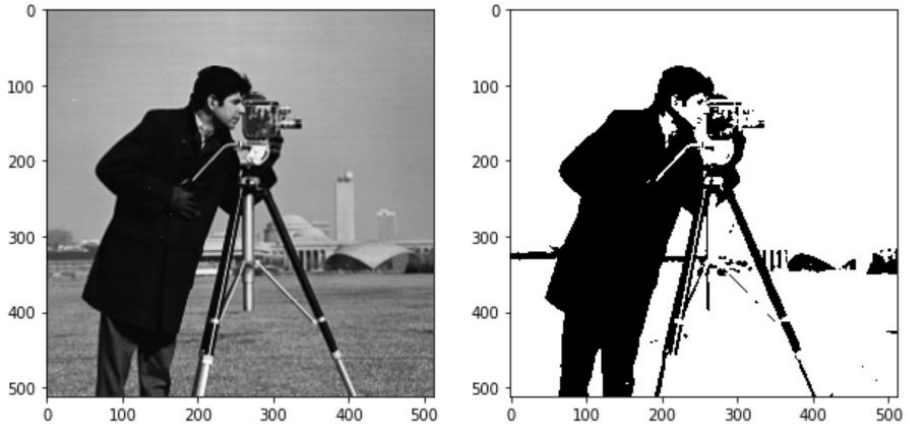


Answer:

```
img = data.camera()
invertedImg = 255 - img
```

Binarize an image

(*i.e.,* convert all pixels to either 255 or 0 depending on some threshold)



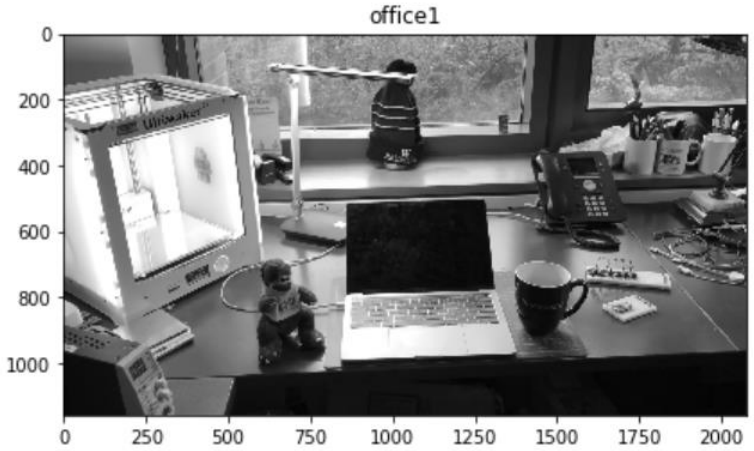
Answer:

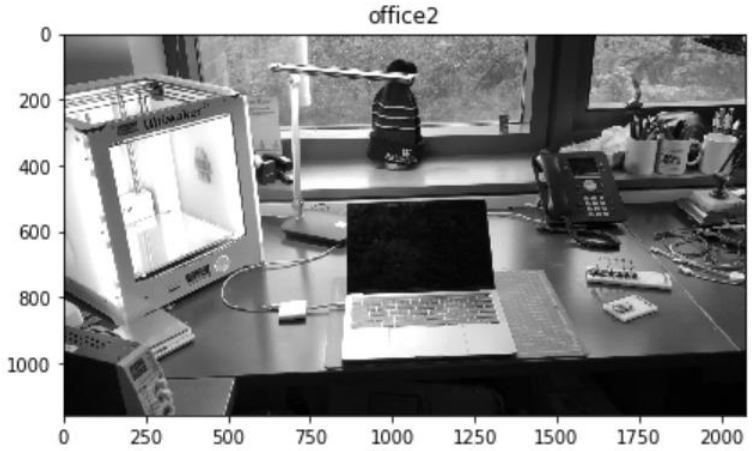
```
img = data.camera()
threshold = 77
binarizedImage = img > threshold
```

LET'S DO SOME BASIC IMAGE PROCESSING EXERCISES!

In Jupyter Notebook, I want you to perform the following image processing tasks using basic pixel-point arithmetic operations. **Do work** with a partner (if you want). **Try not** to Google for answers (I want you to *think* about this!). ☺

Can anyone tell me what the difference is between these two images?

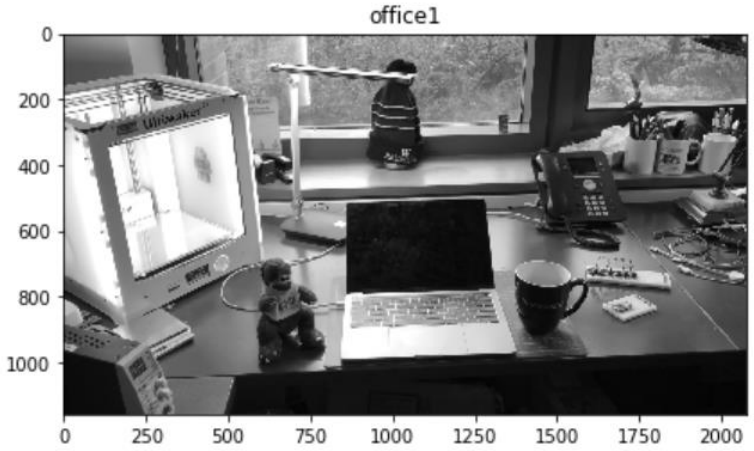


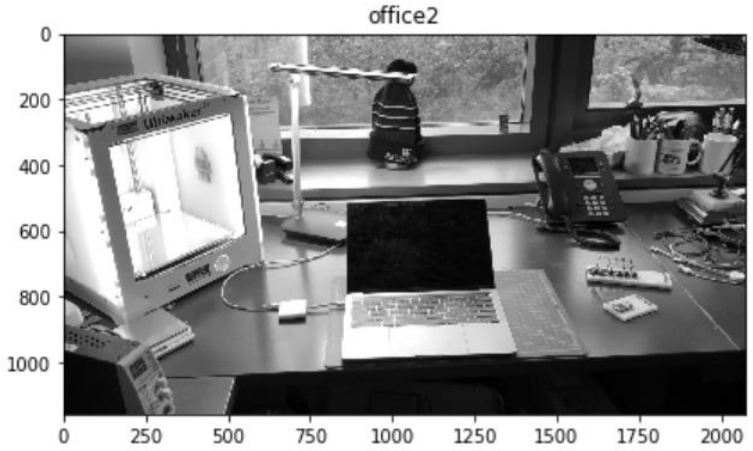


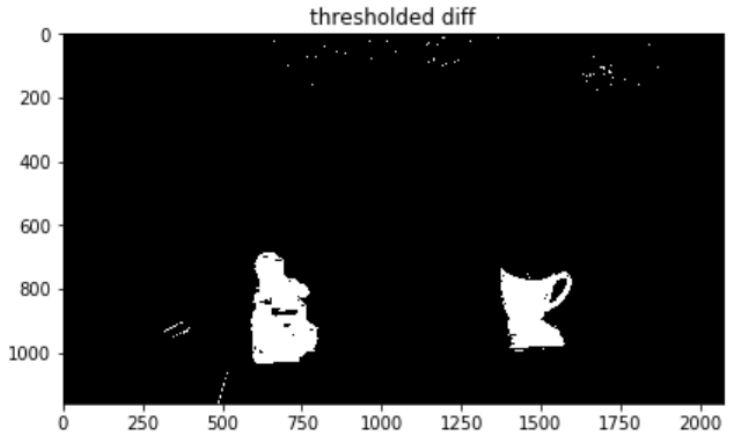
LET'S DO SOME BASIC IMAGE PROCESSING EXERCISES!

In Jupyter Notebook, I want you to perform the following image processing tasks using basic pixel-point arithmetic operations. **Do work** with a partner (if you want). **Try not** to Google for answers (I want you to *think* about this!). ☺

Diff an image



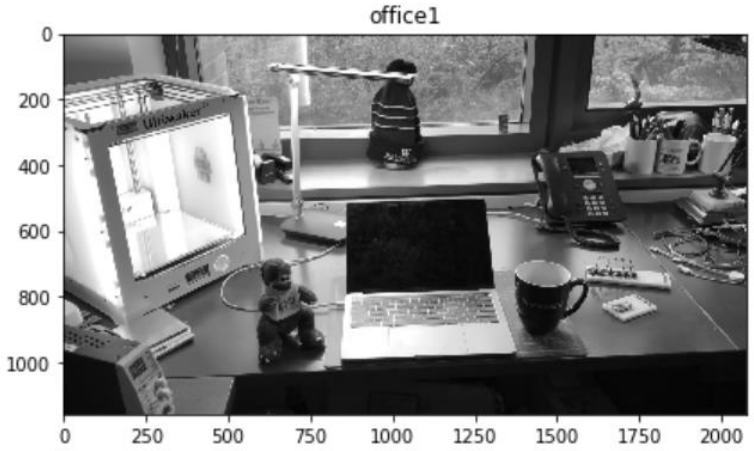


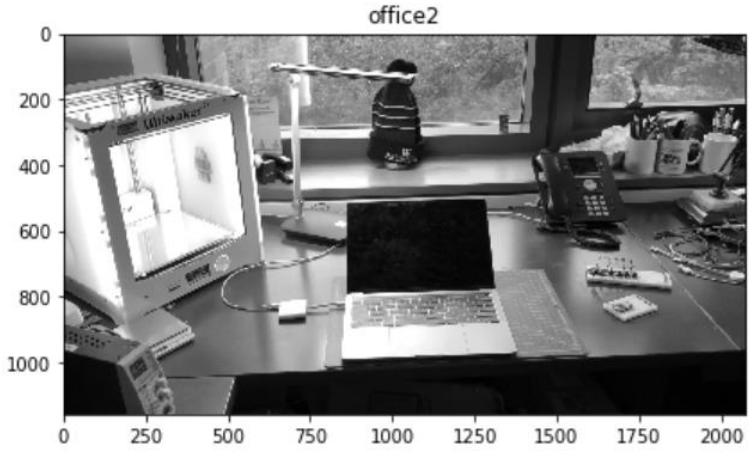


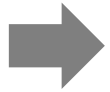
LET'S DO SOME BASIC IMAGE PROCESSING EXERCISES!

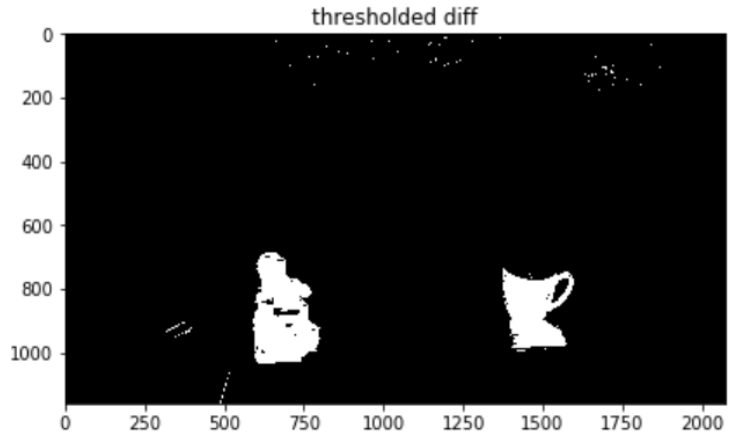
In Jupyter Notebook, I want you to perform the following image processing tasks using basic pixel-point arithmetic operations. **Do work** with a partner (if you want). **Try not** to Google for answers (I want you to *think* about this!). ☺

Diff an image





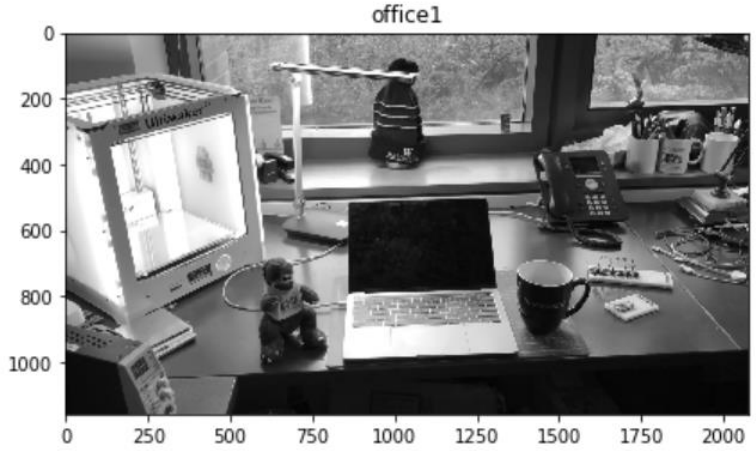


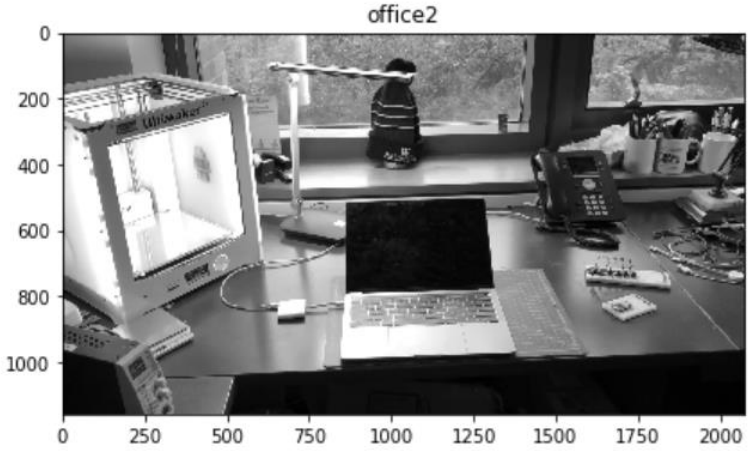


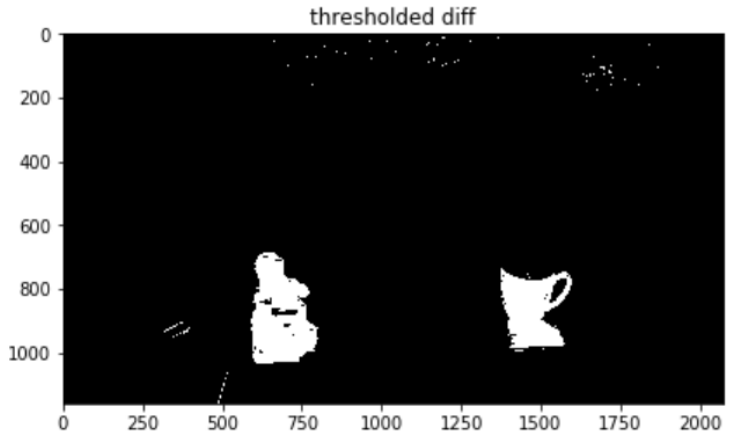
One Possible Answer:

```
officeDiff = office2 - office1 # use basic subtraction to find diffs

# Some values will be less than zero, so normalize the array
officeDiffNormalized = (officeDiff - officeDiff.min())  / (officeDiff.max() - officeDiff.min())

# Binarize image to highlight significant differences
binarizedImage = officeDiffNormalized > threshold
```

WE CAN DO ALL SORTS OF COOL THINGS WITH THIS…







Once we have the detected changes, we can do all sorts of interesting things:

- Move the dinosaur and coffee cup into a different scene
- Begin object tracking (use the underlying pixels as a template)
- Redraw scene with differences highlighted (this is the basis for those sports vis)

OK, so you are **building up some familiarity** with scikit-image (all images are numpy arrays, yay!) and a **basic understanding of image manipulation/processing**.

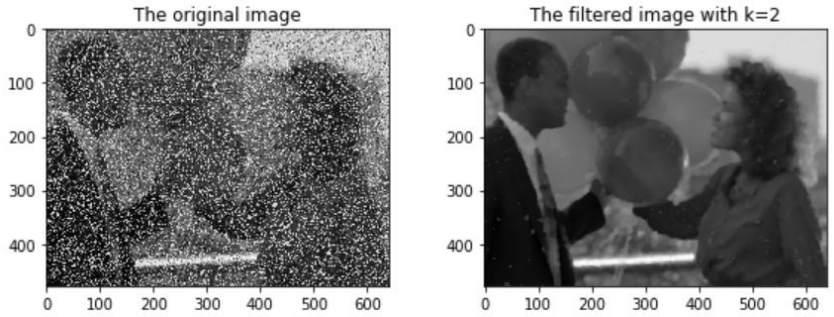
OK, so you are **building up some familiarity** with scikit-image (all images are numpy arrays, yay!) and a **basic understanding of image manipulation/processing**. Let's build on this and move on to one of the key concepts in image processing and computer vision: **cross-correlation and convolution.**

IMAGE FILTERING

There are three major applications of filtering: enhancement, information/feature extraction, pattern detection

Image Enhancement

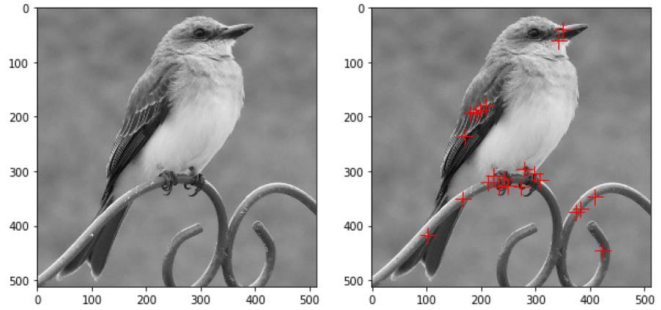
e.g., denoise, color correction



Example using skimage.filters.median:
http://scikit-image.org/docs/dev/api/skimage.filters.html#skimage.filters.median

Feature Detection/Extraction

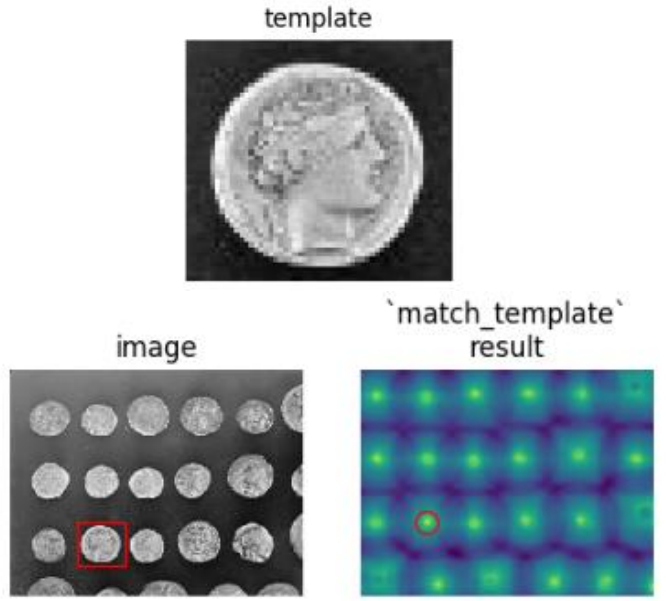
e.g., textures, corners, edges



Example using skimage.feature.corner_harris
http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_corner.html

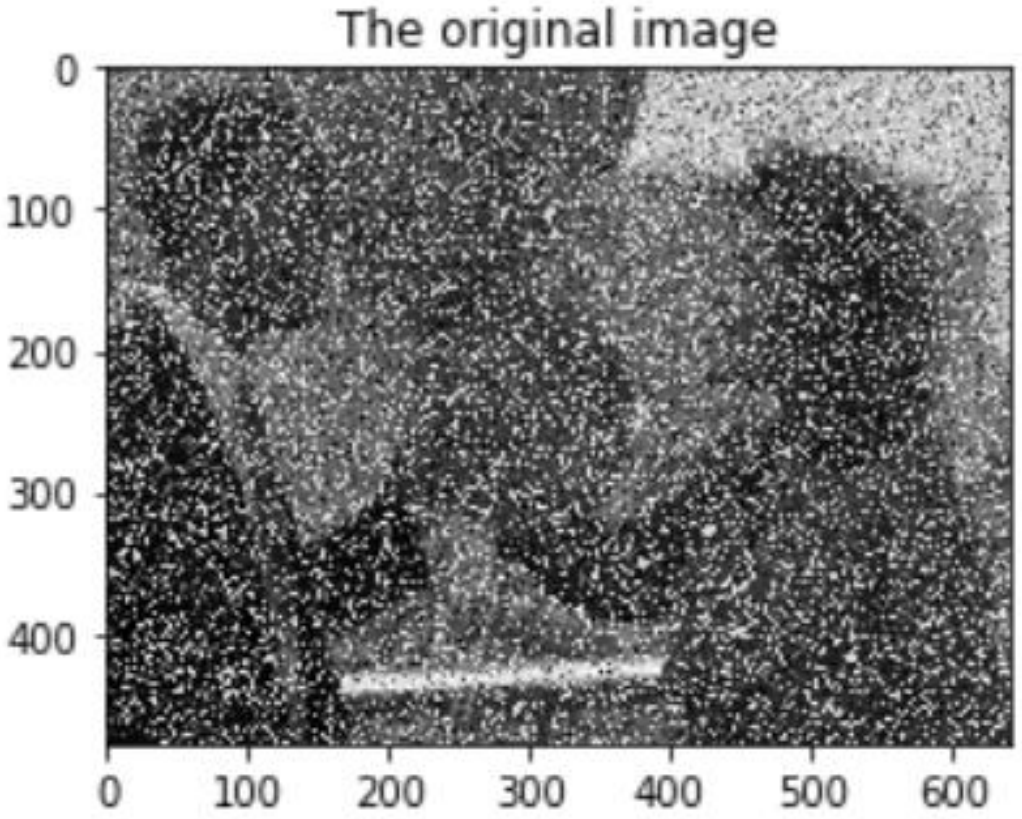
Pattern Detection

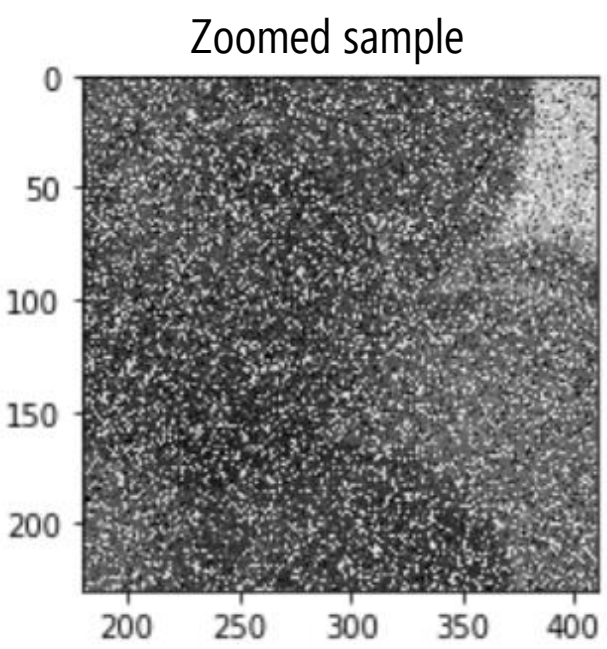
e.g., template matching



Example from:
http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_template.html

HOW CAN WE REDUCE NOISE IN AN IMAGE?





HOW CAN WE REDUCE NOISE IN AN IMAGE?

Idea 1

Treat each row as an independent signal

Use any of your prior smoothing approaches

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

HOW CAN WE REDUCE NOISE IN AN IMAGE?

Idea 1

Treat each row as an independent signal

Use any of your prior smoothing approaches

For example, let's try our favorite—a **1D moving window average filter**

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

IMAGE FILTERING

1D MOVING WINDOW AVERAGE FILTER

Let's try a 1D moving average filter with k=2 thus the window size = 5

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input Image: f(x,y)

```
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
```

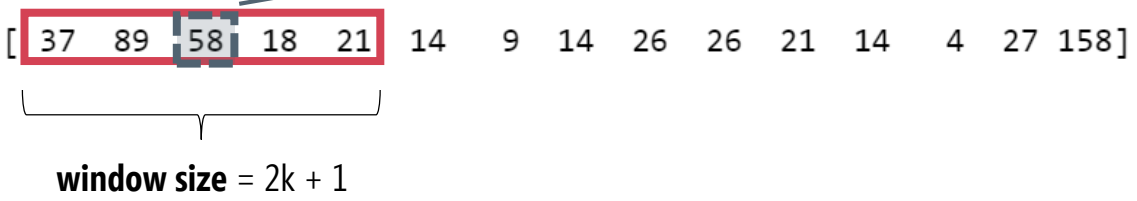
Output Image: g(x,y)

IMAGE FILTERING

1D MOVING WINDOW AVERAGE FILTER

k is the number of neighbors we explore in each direction of the current pixel

Current pixel with index (i, j)



window size = 2k + 1

New pixel value with index (i,j) based on filter operation. In this case, an average:

$$g(i,j) = \frac{1}{(2k+1)} \sum_{-k}^{+k} f(i,j+k)$$

Input Image: F(i,j)

Output Image: G(i,j)

IMAGE FILTERING

1D MOVING WINDOW AVERAGE FILTER

k is the number of neighbors we explore in each direction of the current pixel

Current pixel with index (i, j)

[37 89 58 18 21 14 9 14 26 26 21 14 4 27 158]

window size = 2k + 1

New pixel value with index (i,j) based on filter operation. In this case, an average:

$$g(i,j) = \frac{1}{(2k+1)} \sum_{-k}^{+k} f(i,j+k)$$

[45]

Input Image: F(i,j)

Output Image: G(i,j)

IMAGE FILTERING

1D MOVING WINDOW AVERAGE FILTER

Let's try a 1D moving average filter with k=2 thus the window size = 5

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input Image: F(i,j)

```
[          45  40                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
```

Output Image: G(i,j)

1D MOVING WINDOW AVERAGE FILTER

Let's try a 1D moving average filter with k=2 thus the window size = 5

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input Image: F(i,j)

```
[                45  40  24                                  ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
[                                                            ]
```

Output Image: G(i,j)

IMAGE FILTERING

1D MOVING WINDOW AVERAGE FILTER

Let's try a 1D moving average filter with k=2 thus the window size = 5

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input Image: F(i,j)

```
[                 45  40  24  15                 ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
[                                                ]
```

Output Image: G(i,j)

1D MOVING WINDOW AVERAGE FILTER

Let's try a 1D moving average filter with k=2 thus the window size = 5

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input Image: F(i,j)

```
[  0   0  44  40  24  15  16  17  19  20  18  18  44   0   0]
[  0   0  54  34  14  18  24  28  33  35  26  21  57   0   0]
[  0   0  17  12  15  29  37  44  51  50  38  32  55   0   0]
[  0   0  34  39  53  56  52  52  43  34  27  20  37   0   0]
[  0   0  61  61  72  60  47  49  46  37  33  28  39   0   0]
[  0   0  77  78  92  83  67  60  57  49  43  37  52   0   0]
[  0   0  84  93 111 108  91  77  67  57  49  45  69   0   0]
[  0   0 104  96 107 107  91  77  67  58  49  55  83   0   0]
[  0   0 143 114  96  90  73  58  47  41  37  41  72   0   0]
[  0   0 141 110  86  82  67  55  45  36  35  58  94   0   0]
[  0   0 161 123  89  81  74  63  59  56  40  40  67   0   0]
[  0   0 164 127  80  55  47  28  26  33  26  21  30   0   0]
[  0   0 120  96  65  62  63  50  37  33  29  27  30   0   0]
[  0   0  76  64  72  83  83  72  59  43  34  31  29   0   0]
[  0   0  43  43  61  71  71  68  57  42  41  37  33   0   0]
```

Output Image: G(i,j)

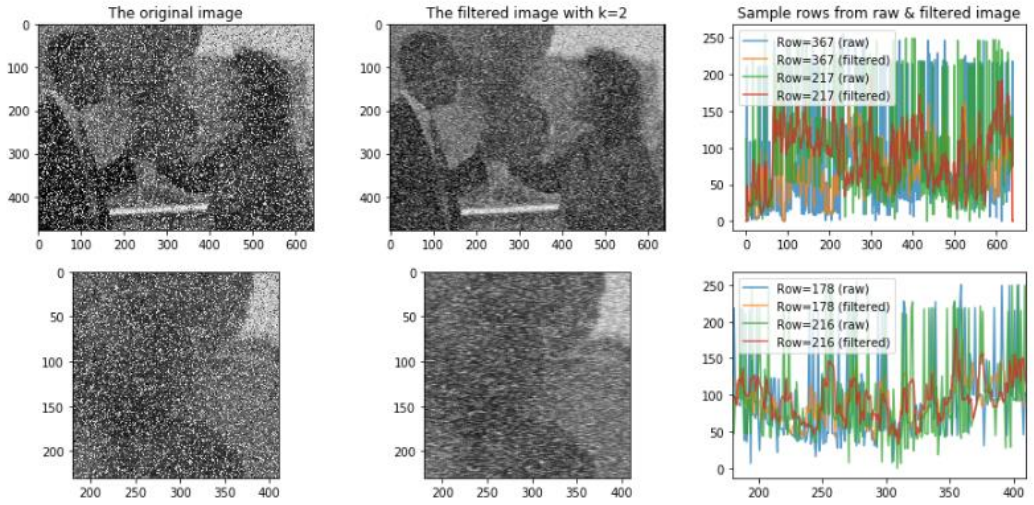
IMPLEMENT A 1D MOVING AVERAGE FILTER!

IMAGE FILTERING

IMPLEMENT A 1D MOVING AVERAGE FILTER!

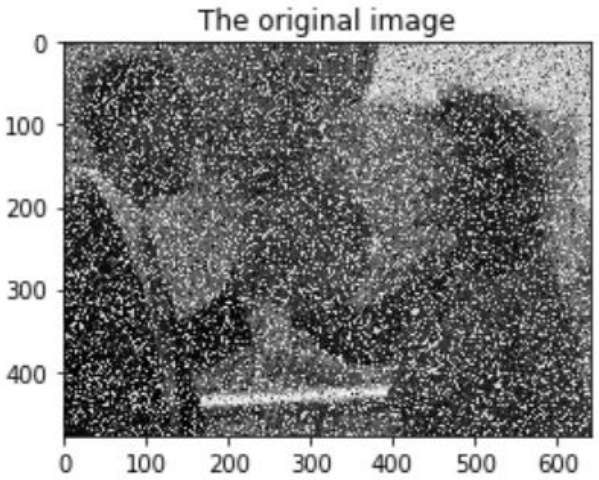
```
# ONE DIMENSIONAL IMAGE SMOOTHING ALGORITHM
# Let's build our own window-based mean filter for an image. This is very similar
# to the smoothing approaches we've previously done in the course.
# Here, we are going to manually walk through each pixel in our numpy array
# and take an average of the pixels around it (using a 1-dimensional sliding window)
# Note that manually stepping through numpy arrays is not typically done as the
# numpy library allows you to perform full matrix operations without setting up your own
# for loops. But we'll do it for now for educational purposes!
# Also note that this simple example is only smoothing in the x-direction

# # img = io.imread("./Images/bears_0.05_noisy.jpg", as_grey=True)
img = io.imread("./Images/balloons_noisy_bw.png")

k = 2 # number of neighbors to look at, the larger k, the "blurrier" the image will get
windowSize1D = (2 * k) + 1 # the window size of our average filter
imageHeight, imageWidth = img.shape

# initialize our smoothed image to all zeros. We must specify our dtype
# here because scikit-image converted grayscale images are 64-bit floats
# vs. raw grayscale images are 8-bit ints
smoothedImage = np.full((imageHeight, imageWidth), 0, dtype=img.dtype)

# numpy stores data in row major order
# Loop through every pixel and calculate an average for a window around that pixel
# Note that we are only smoothing here in the x-direction
print("Smoothing with a windowSize1D=", windowSize1D)
for j in range(0, imageHeight): # for each row
    for i in range(k, imageWidth - k): # and each column
        pixelsInWindow = img[j, i - k : i + k + 1] # first arg retu
        avg = np.mean(pixelsInWindow)
        smoothedImage[j,i] = avg

plotFilteredImage(img, smoothedImage, k, (180,410), (230,0))
```

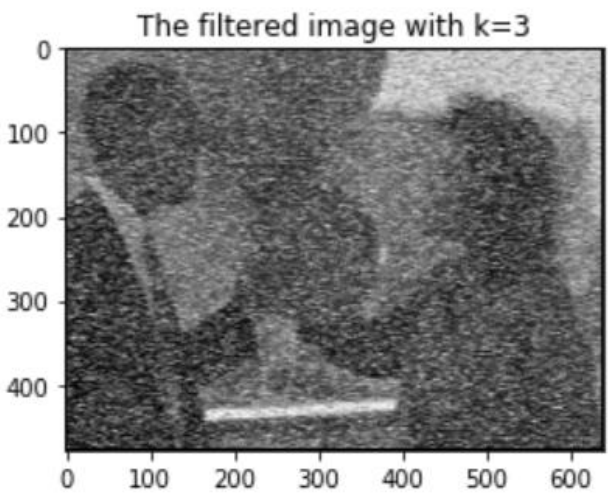
```
Smoothing with a windowSize1D= 5
```

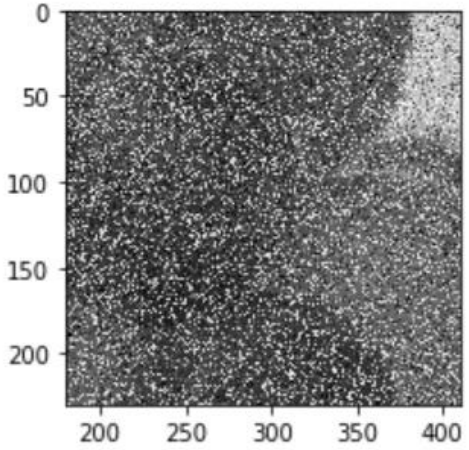


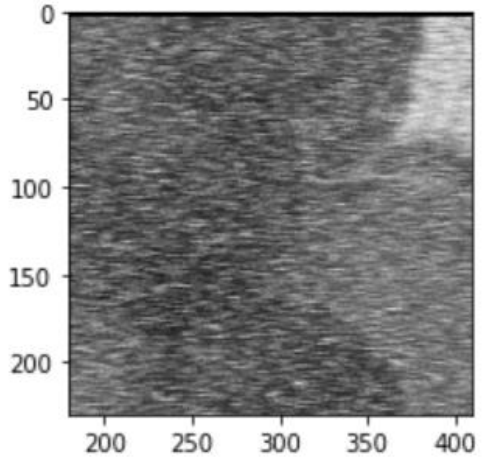
1D MOVING WINDOW AVERAGE FILTER RESULTS

Window Size=7







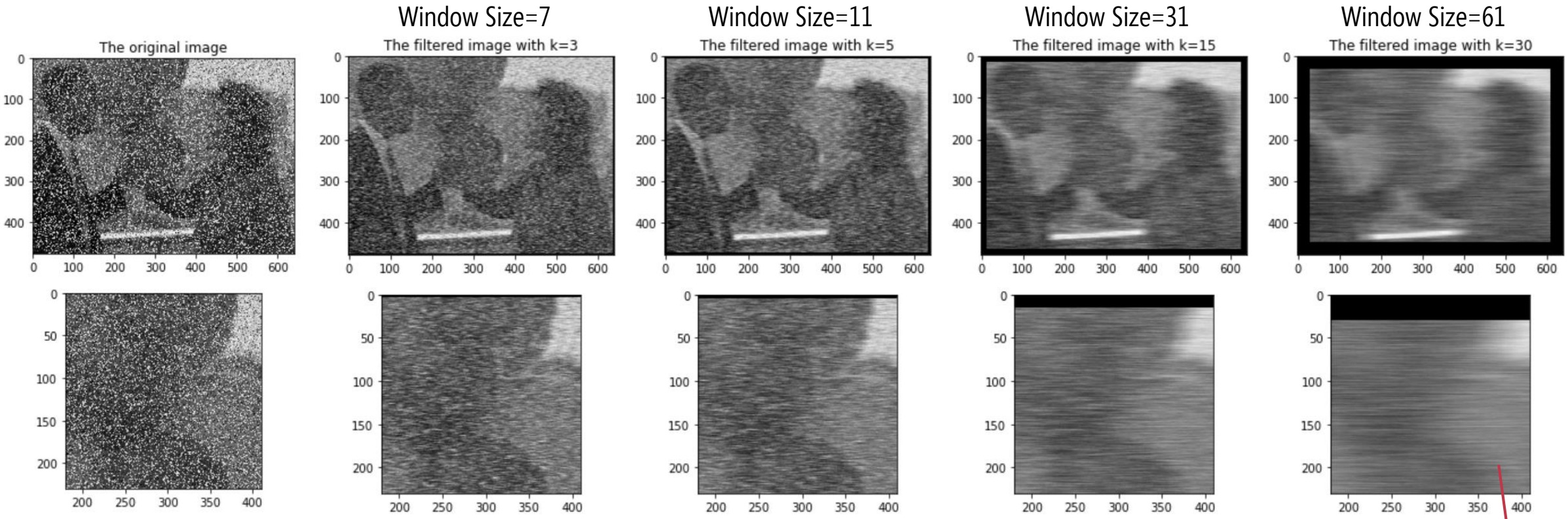


Definitely better than original.

But…

A 1-DIMENSIONAL WINDOW FILTER IS INSUFFICIENT

A 1-dimensional filter (like we've used previously) is insufficient for a 2D signal. With a 1D filter, we are only using part of the information we have to filter the signal. We can do better.



As the filter window size gets larger, artifacts from our 1D approach become clear

Notice the **horizontal streaks**, which is an artifact of smoothing only in the x-direction

WE NEED A 2D FILTER! BUT HOW?

Rather than using a 1D sliding window, we will use a 2D sliding window (often called a box filter). We'll use the same concept as before. Slide the window across the image and perform some calculation over the pixels within the window.

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input image F(i,j) with a 1D sliding window

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input image F(i,j) with a 2D sliding window

WE NEED A 2D FILTER! BUT HOW?

All of our metrics are the same (k, window size, etc.) but we add a new one: num_cells_in_window

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input image F(i,j) with a 1D sliding window

k is still the number of neighbors we explore in each direction of the current pixel

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
```

window size = $2k + 1$

total window size (aka num_cells_in_window) = $(2k + 1)^2$

Input image F(i,j) with a 2D sliding window

IMAGE FILTERING

2D MOVING AVERAGE FILTER

To make this easier on us, let's set k=1, so window size=3, and num_cells_in_wind

New pixel value with index (i,j) based on filter operation. In this case, an average:

$$g(i,j) = \frac{1}{(2k+1)^2} \sum_{u=-k}^{+k} \sum_{v=-k}^{+k} f(i+u, j+v)$$

37	89	58	18	21	14	9	14	26	26	21	14	4	27	158
101	121	25	15	10	3	17	48	46	27	31	27	3	17	207
45	30	0	0	12	22	44	70	41	46	58	39	9	11	159
39	13	43	48	31	62	85	54	31	30	17	39	18	0	113
49	36	110	87	23	51	91	50	24	30	36	49	26	1	86
59	31	117	111	69	65	99	74	32	34	47	62	44	0	107
55	17	108	134	107	99	110	92	47	38	50	60	54	27	156
135	49	91	135	110	97	102	92	56	42	47	57	45	85	185
238	167	92	121	99	93	79	61	36	22	38	51	39	58	178
238	198	70	115	87	80	79	52	38	30	27	36	46	153	209
237	241	143	100	86	48	68	105	67	28	31	49	28	68	162
237	255	163	64	103	51	19	42	24	6	43	53	5	0	50
237	219	39	17	89	119	63	24	20	26	52	43	6	12	37
171	71	7	40	95	108	110	62	40	44	42	27	19	26	34
78	1	25	50	62	77	93	74	52	45	21	20	67	35	23

Input image F(i,j)

56

$$g(i,j) = \frac{1}{9} * (37 + 89 + 58 + 101 + 121 + 25 + 45 + 30 + 0)$$

Output Image: G(i,j)

IMAGE FILTERING

2D MOVING AVERAGE FILTER

To make this easier on us, let's set k=1, so window size=3, and num_cells_in_window=9

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input image F(i,j)

```
[                                                           ]
[     56  40                                                ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
[                                                           ]
```

$$g(i,j) = \frac{1}{9} * (89 + 58 + 18 + 121 + 25 + 15 + 30 + 0 + 0)$$

Output Image: G(i,j)

IMAGE FILTERING

2D MOVING AVERAGE FILTER

To make this easier on us, let's set k=1, so window size=3, and num_cells_in_window=9

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input image F(i,j)

```
[                                                          ]
[     56   40  18                                          ]
[                                                          ]
[                                                          ]
[          g(i,j) = 1/9 * (58 + 18 + 21 + 25 + 15 + 10 + 0 + 0 + 12)  ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
[                                                          ]
```

Output Image: G(i,j)

2D MOVING AVERAGE FILTER RESULTS

To make this easier on us, let's set k=1, so window size=3, and num_cells_in_window=9

```
[ 37  89  58  18  21  14   9  14  26  26  21  14   4  27 158]
[101 121  25  15  10   3  17  48  46  27  31  27   3  17 207]
[ 45  30   0   0  12  22  44  70  41  46  58  39   9  11 159]
[ 39  13  43  48  31  62  85  54  31  30  17  39  18   0 113]
[ 49  36 110  87  23  51  91  50  24  30  36  49  26   1  86]
[ 59  31 117 111  69  65  99  74  32  34  47  62  44   0 107]
[ 55  17 108 134 107  99 110  92  47  38  50  60  54  27 156]
[135  49  91 135 110  97 102  92  56  42  47  57  45  85 185]
[238 167  92 121  99  93  79  61  36  22  38  51  39  58 178]
[238 198  70 115  87  80  79  52  38  30  27  36  46 153 209]
[237 241 143 100  86  48  68 105  67  28  31  49  28  68 162]
[237 255 163  64 103  51  19  42  24   6  43  53   5   0  50]
[237 219  39  17  89 119  63  24  20  26  52  43   6  12  37]
[171  71   7  40  95 108 110  62  40  44  42  27  19  26  34]
[ 78   1  25  50  62  77  93  74  52  45  21  20  67  35  23]
```

Input image F(i,j)

```
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
[  0  56  39  17  12  16  26  35  38  35  32  22  16  66   0]
[  0  46  32  20  22  31  45  48  43  36  34  26  18  59   0]
[  0  40  40  39  37  46  58  54  41  34  38  32  21  47   0]
[  0  55  66  71  60  64  70  60  39  31  38  37  26  43   0]
[  0  64  83  96  82  79  81  68  46  37  45  47  35  55   0]
[  0  73  88 109 103  95  92  78  56  43  48  51  48  78   0]
[  0 105 101 110 110  99  91  75  54  41  45  49  52  91   0]
[  0 142 115 102 104  91  81  66  47  37  38  42  63 110   0]
[  0 180 138 101  92  79  73  65  48  35  34  38  58 104   0]
[  0 198 149 103  81  69  60  54  43  32  33  35  48  80   0]
[  0 196 137  89  75  71  59  48  38  33  36  34  29  40   0]
[  0 155  97  68  76  84  66  44  32  33  37  32  21  21   0]
[  0  94  52  47  73  90  81  59  43  38  35  33  28  28   0]
[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
```

Output Image: G(i,j)

IMAGE FILTERING

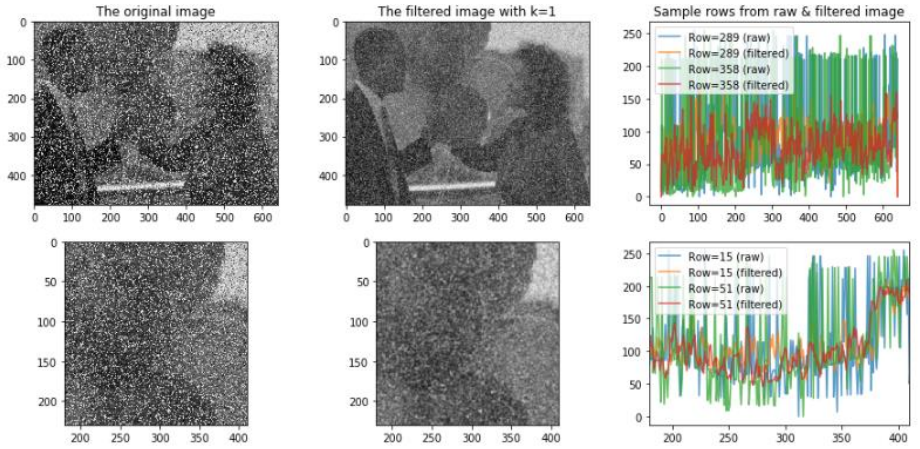
IMPLEMENT A 2D MOVING AVERAGE FILTER!

IMAGE FILTERING

IMPLEMENT A 2D MOVING AVERAGE FILTER!

```
# TWO DIMENSIONAL IMAGE SMOOTHING ALGORITHM
# Let's extend our one-dimensional image smoothing approach to *two* dimensions
# So now our sliding window is really a sliding box! And we can use numpy's
# fancy slicing syntax to make this really easy :)
# See: https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.indexing.html

# img = io.imread("./Images/bears_0.05_noisy.jpg", as_grey=True)
img = io.imread("./Images/balloons_noisy_bw.png")

k = 1 # number of neighbors to look at, the larger k, the "blurrier" the image will get
windowSize2D = (2 * k) + 1 # the window size of our average filter
imageHeight, imageWidth = img.shape

# initialize our smoothed image to all zeros. We must specify our dtype
# here because scikit-image converted grayscale images are 64-bit floats
# vs. raw grayscale images are 8-bit ints
smoothedImage = np.full((imageHeight, imageWidth), 0, dtype=img.dtype)

# numpy stores data in row major order
# Loop through every pixel and calculate an average for a window around that pixel
# Note that we are only smoothing here in the x-direction
print("Smoothing with a windowSize2D=", windowSize2D)
for j in range(k, imageHeight - k): # for each row
    for i in range(k, imageWidth - k): # and each column
        # get the matrix within our sliding window
        pixelsInWindow = img[j - k : j + k + 1, i - k : i + k + 1]
        avg = np.mean(pixelsInWindow)
        smoothedImage[j,i] = avg

plotFilteredImage(img, smoothedImage, k, (180,410), (230,0))
```

```
Smoothing with a windowSize2D= 3
```

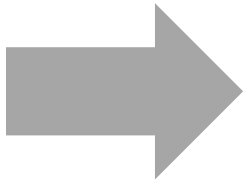


2D MOVING AVERAGE FILTER WEIGHTED

Original Equation for Average Filter

$$g(i,j) = \frac{1}{(2k+1)^2} \sum_{u=-k}^{+k} \sum_{v=-k}^{+k} f(i+u, j+v)$$

But imagine that we want to **weight pixels** closer to selected pixel more than those further away. For this, we have to introduce another "lookup" matrix that holds these weights.



v	a	b	c
	d	e	f
	g	h	i
		u	

Let's call this matrix: H(u,v)

The example above is for k=1.

H is often referred to as a kernel or mask.

New Equation for Weighted Average Filter

$$g(i,j) = \sum_{u=-k}^{+k} \sum_{v=-k}^{+k} h(u,v) f(i+u, j+v)$$

Grab weights from our h(u,v) matrix

This is called the cross-correlation operation

$$G = H \otimes F$$

IMAGE FILTERING

FROM MOVING AVERAGE TO CROSS-CORRELATION

A Mathematical Representation for Smoothing

$$G[3,3] = \frac{1}{9}(A + B + C + D + E + F + G + H + I)$$

$$G[i,j] = \frac{1}{(2k+1)^2} \sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]$$

a	b	c
d	e	f
g	h	i

20	20	10	20	10	20	10	10	13
30	0	0	0	0	0	0	0	30
20	0	A	B	C	90	90	0	20
20	0	D	E	F	90	90	0	20
10	0	G	H	I	90	90	0	10
10	0	90	90	90	90	90	0	10
10	0	90	90	90	90	90	0	10
20	0	0	0	0	0	0	0	20
20	20	10	20	10	20	10	10	13

More Generally,

$$G[3,3] = a*A + b*B + c*C + d*D + e*E + f*F + h*H + i*I$$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] F[i+u, j+v]$$

Attribute non-uniform weights

Source: Irfan Essa for his Computational Photography course: https://classroom.udacity.com/courses/ud955/lessons/3173838562/concepts/31611185510923

WHAT DOES H[U,V] LOOK LIKE FOR MOVING AVERAGE?

For k=1. So, window size = 3 and num cells = 9

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

IMAGE FILTERING

CROSS-CORRELATION & CONVOLUTION

Cross Correlation

$$G[i,j] = \sum_{u=-k}^{+k} \sum_{v=-k}^{+k} H[u,v]F[i+u,j+v]$$

$$G = H \otimes F$$

Convolution

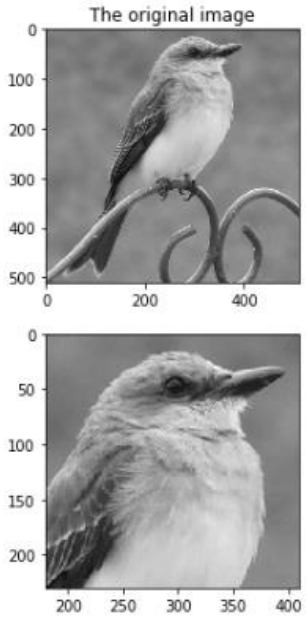
$$G[i,j] = \sum_{u=-k}^{+k} \sum_{v=-k}^{+k} H[u,v]F[i-u,j-v]$$

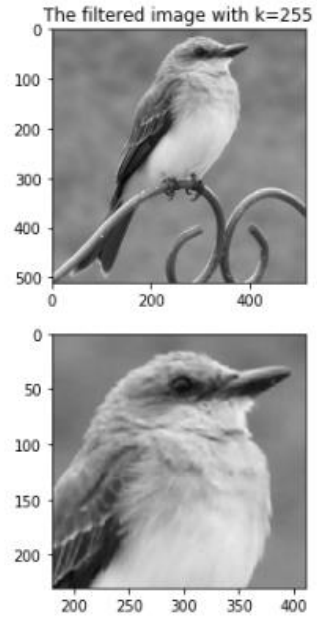
$$G = H \star F$$

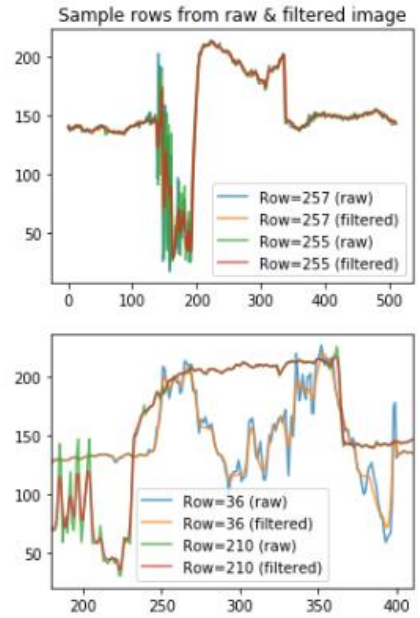
Source: Steve Seitz

CONVOLUTION-BASED IMAGE FILTERING

```
# Now, let's do this the *right* way--using a kernel and convolution! :)
# Some example kernels: https://en.wikipedia.org/wiki/Kernel_(image_processing)
img = birdImageGrayscale # feel free to change this

# 3x3 mean "box" filter
meanBoxFilter3x3 = np.array([[1/9, 1/9, 1/9],
                             [1/9, 1/9, 1/9],
                             [1/9, 1/9, 1/9],])

k = int((meanBoxFilter3x3Result.shape[0] - 1)/2.0) # calculate k

# We use scipy's signal library (just like we did in earlier parts of the course) but
# this time it's a 2D convolution rather than a 1D
meanBoxFilter3x3Result = signal.convolve2d(img, meanBoxFilter3x3, boundary='symm', mode='same')
plotFilteredImage(img, meanBoxFilter3x3Result, k, (180,410), (230,0))
```



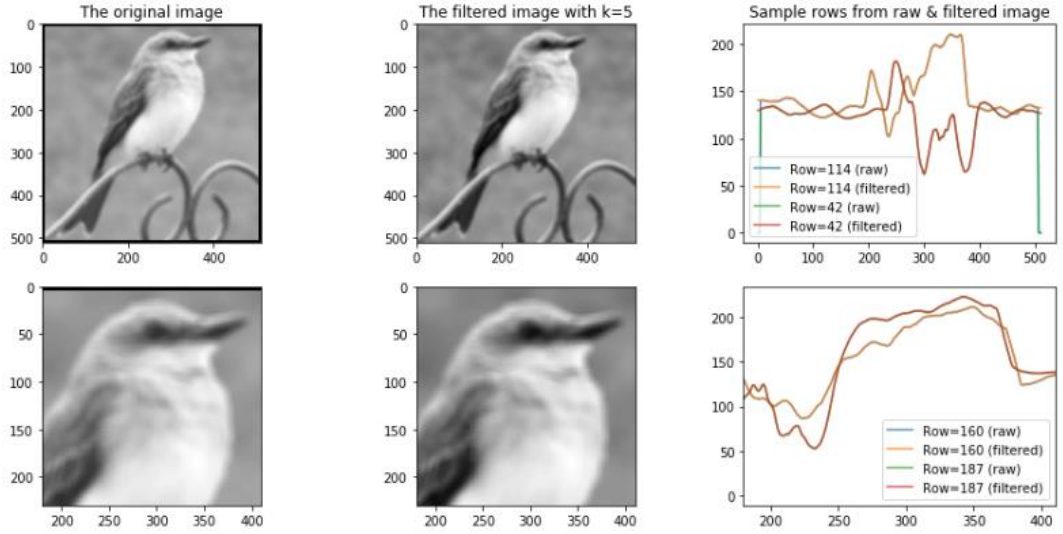




VALIDATING OUR MANUAL 2D ALG VS. CONVOLUTION

```
# Simple check between our own algorithm and convolution :)
k = 5
img = birdImageGrayscale
imageHeight, imageWidth = img.shape
manualSmoothedImage = np.full((imageHeight, imageWidth), 0, dtype=img.dtype)
slidingWindowDimension = (2 * k + 1)
numCellsInSlidingWindow = slidingWindowDimension ** 2 # numCells = (2k+1)^2
meanBoxFilter = np.full((slidingWindowDimension, slidingWindowDimension), 1 / numCellsInSlidingWindow)

# First manually smooth the image using a 2D sliding window
print("Smoothing with k={}, boxFilter.width={}, boxFilter.numCells={}".format(
    k, meanBoxFilter.shape[1], numCellsInSlidingWindow))
for j in range(k, imageHeight - k): # for each row
    for i in range(k, imageWidth - k): # and each column
        # get the matrix within our sliding window
        pixelsInWindow = img[j - k : j + k + 1, i - k : i + k + 1]
        avg = np.mean(pixelsInWindow)
        manualSmoothedImage[j,i] = avg

convolutionSmoothedImage = signal.convolve2d(img, meanBoxFilter, boundary='symm', mode='same')
print(manualSmoothedImage.shape)
print(convolutionSmoothedImage.shape)
# diffImage = manualSmoothedImage - convolutionSmoothedImage

# The titles of the plotted images are wrong (because we are not displaying raw and filtered
# but instead two filtered). Still, this let's us compare and verify that our manual alg works :)
# Because imshow auto-normalizes the images when it displays them, we have to turn this off
# the convolution image is going to look a bit darker (because the manually smoothed image
# has a border of intensity 0)
plotFilteredImage(manualSmoothedImage, convolutionSmoothedImage, k, (180,410), (230,0))
```

```
Smoothing with k=5, boxFilter.width=11, boxFilter.numCells=121
(512, 512)
(512, 512)
```



Note: these titles are wrong since both images are filtered but good enough for us to get a sense that things are working!

IMAGE FILTERING

CONVOLUTION EXERCISES IN JUPYTER NOTEBOOK

For smoothing

For edge detection