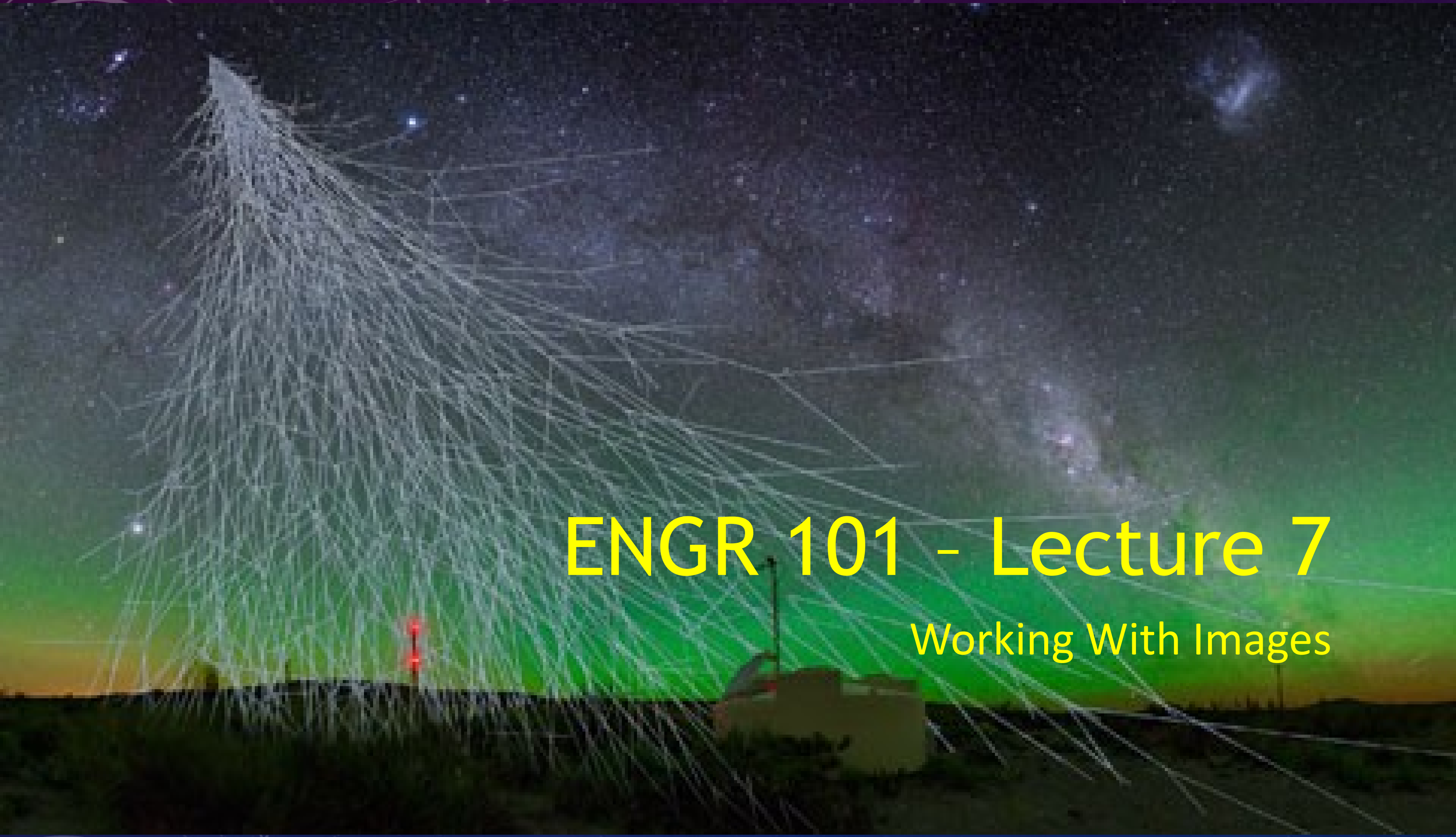
While you are waiting

ANAGRAM

Re-arrange the letters to reveal the related word...

silent



ENGR 101 – Lecture 7

Working With Images

Laura Alford, James Juett, Rick Niciejewski 9/27/17

Lecture Goals

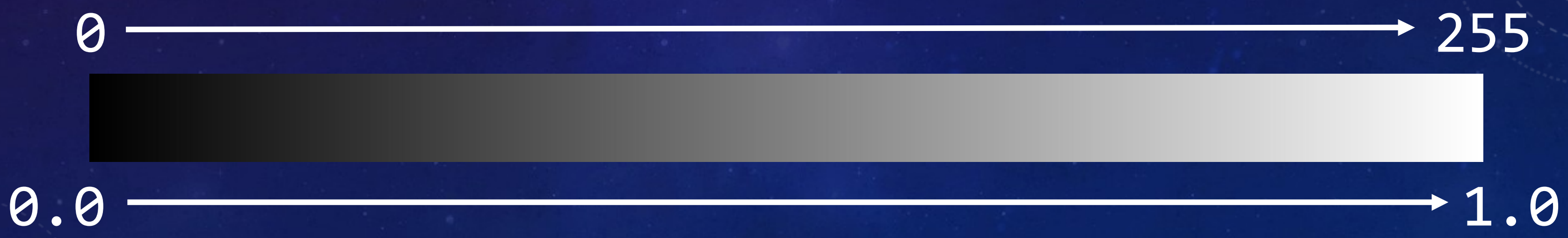
- Today's lecture: Working with Images
  - Greyscale image representation
  - RGB image representation
  - HSV image representation
  - Suggested readings, Attaway, Chap 13.2
  - Project 2 Overview
- Download today's lecture, project2 overview, and 'cat' support files from
  - 00_Todays_Lecture

Grayscale Image Representation

- We'll start with grayscale images (i.e. no colour).
- Each pixel is simply a single **intensity value**.
  - The higher the value, the closer to white.

- There are two ways to represent intensity:
  - An **unsigned integer** between 0 and 255, inclusive.
  - A real number (a **double**) between 0.0 and 1.0, inclusive.

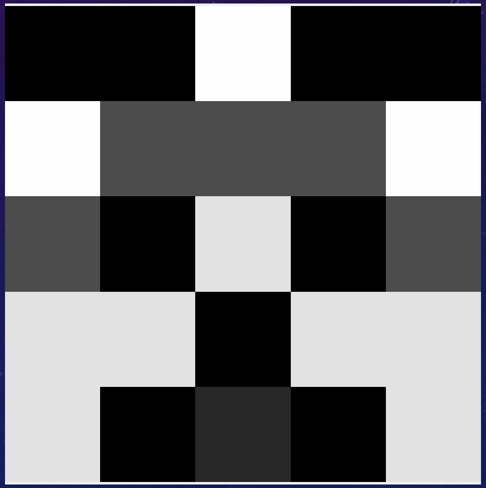
Internally, MATLAB considers integers and doubles to be different "types" of data.



Grayscale Image Representation

- A grayscale image is just a grid of intensity values.
- In MATLAB, this is just a matrix of numbers!

0	0	255	0	0
255	76	76	76	255
76	0	226	0	76
226	226	0	226	226
226	0	40	0	226

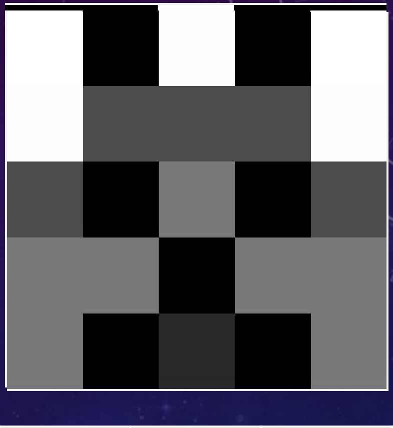


This is supposed to look like a dog. See it?

- That's it, really!

Images are Just Numbers in a Matrix

- Today we'll see how to perform a variety of image processing operations just by manipulating matrices.



- For example:

  -
    ```
    grayImg(1, 1) = 255;
    grayImg(1, 5) = 255;
    ```

  -
    ```
    grayImg(grayImg == 226) = 120;
    ```

255	0	255	0	255
255	76	76	76	255
76	0	120	0	76
120	120	0	120	120
120	0	40	0	120

grayImg

File Input/Output for Images

- To load an image from a file, use the `imread` function.

```
img = imread('filename.jpg')
```

- To save an image to a file, use the `imwrite` function.

```
imwrite(img, 'filename.jpg')
```

- MATLAB can handle most common image file formats:
  - `.jpg`, `.png`, `.gif`, `.bmp`, `.ppm`, etc.

The imshow Function

- First, load the file using imread:
  - `cat_gray = imread('cat_gray.jpg');`

If you forget this semicolon, MATLAB will try to print out a giant image matrix. ☺

If it does, hit ctrl-c to tell it to stop.

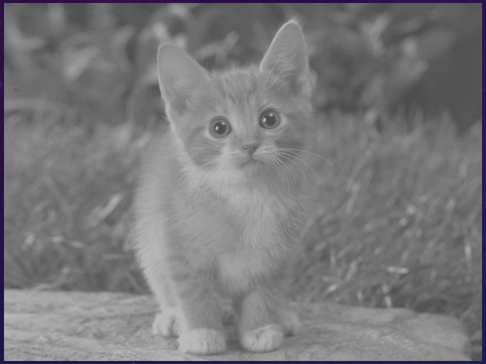
- Now, you can use the imshow function to display the image.
  - `imshow(cat_gray);`

MATLAB will open another window to display the image.

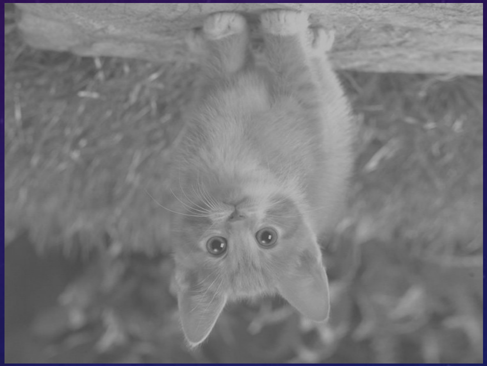
Your turn: Basic Image Operations

4
min

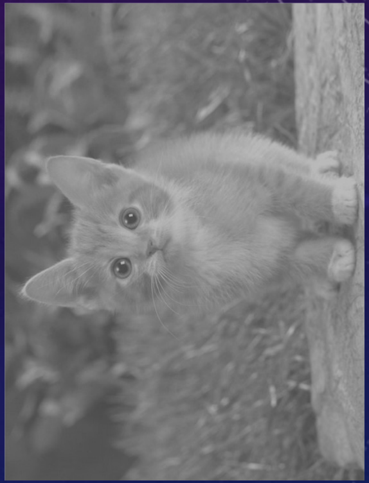
- Write MATLAB expressions to create flipped and rotated versions of the `cat_gray` image.[1]



Horizontal Flip



Vertical Flip



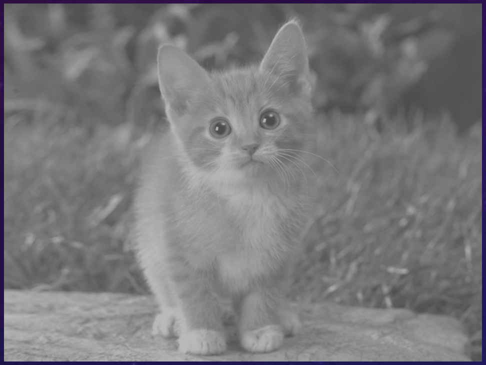
Rotate 90 deg, CCW

- *Hint: You may find the transpose operator* `'`, *the range expression* *`[end:-1:1]`*, *and row/column indexing very useful for this exercise.*

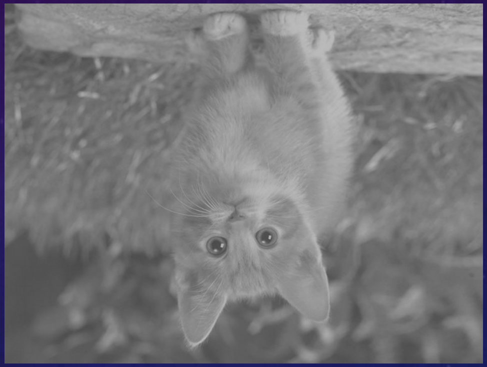
[1] There are built-in MATLAB functions for each of these, but try to do these exercises without using them.

Solution: Basic Image Operations

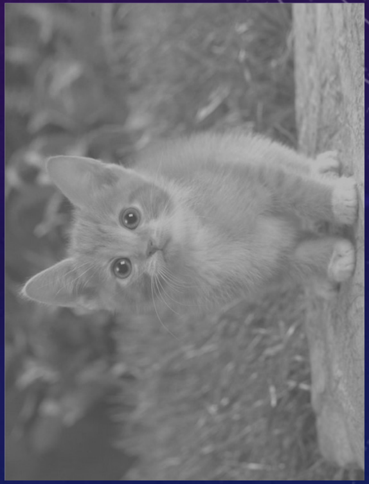
- Write MATLAB expressions to create flipped and rotated versions of the `cat_gray` image.



Horizontal Flip



Vertical Flip



Rotated 90 degrees

```
vf = cat_gray([end:-1:1], :);
```

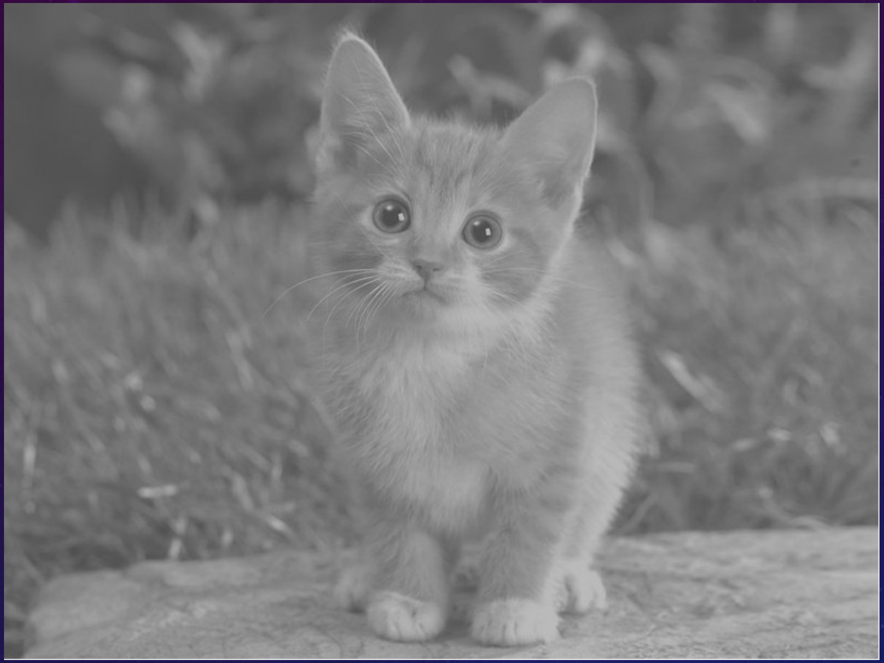
```
hf = cat_gray(:, [end:-1:1]);
```

```
t = cat_gray';
r = t([end:-1:1], :);
```

or

```
imshow(cat_gray(:,[end:-1:1])');
```

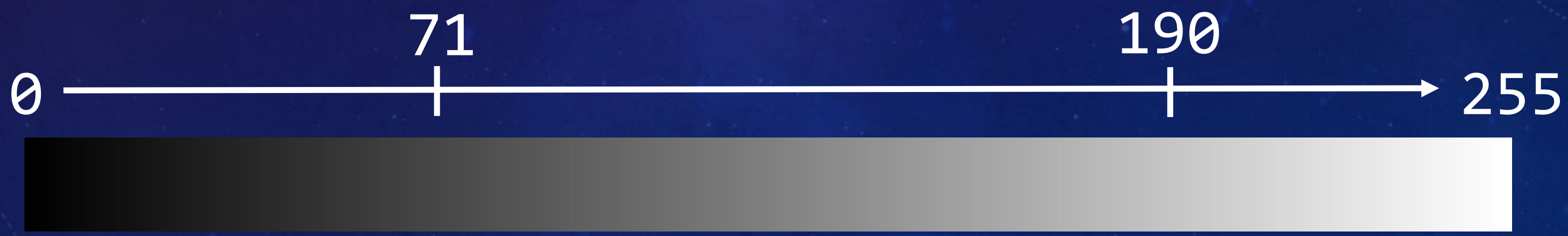
What's wrong with this image?



cat_gray.jpg

It has very poor contrast. The cat kind of fades into the background.

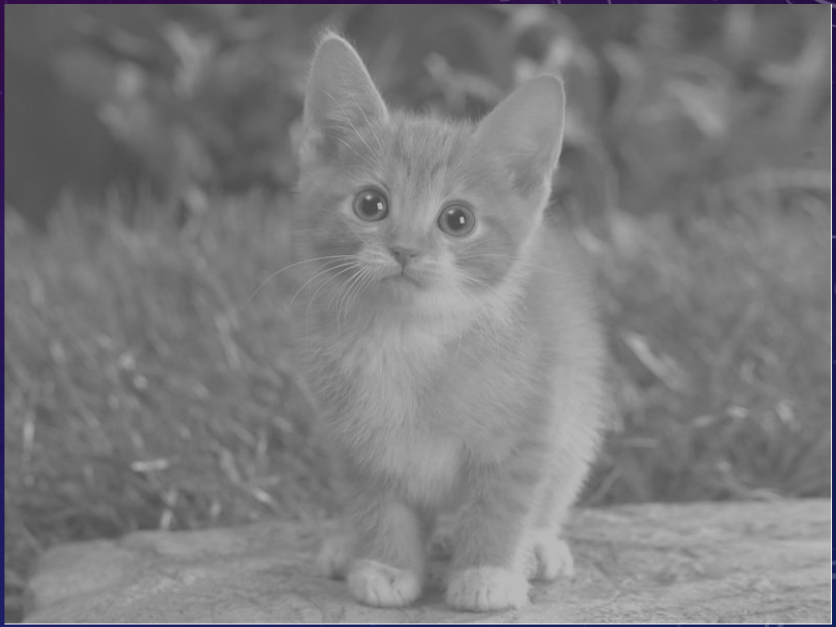
Question: What are the max/min intensities used in this image?



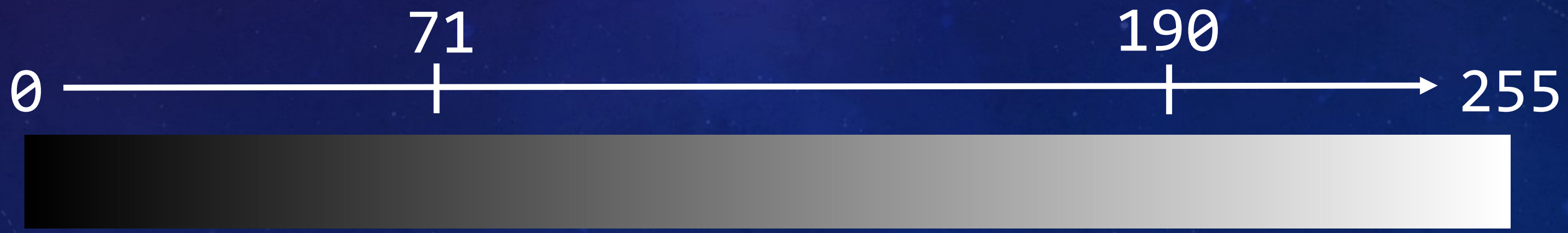
Your turn: Contrast Stretching

4
min

- We can improve the image by using more of the possible intensity values.
- This is a *linear interpolation* problem: stretch the range [71,190] to be [0,255].
- To do this:
  - Subtract 71 from each pixel
  - Then multiply each pixel by 2.14

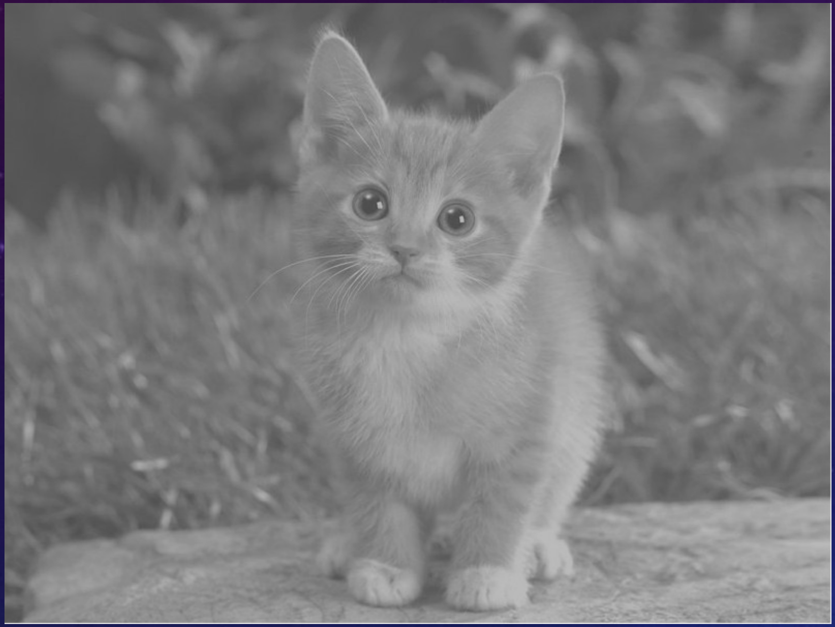


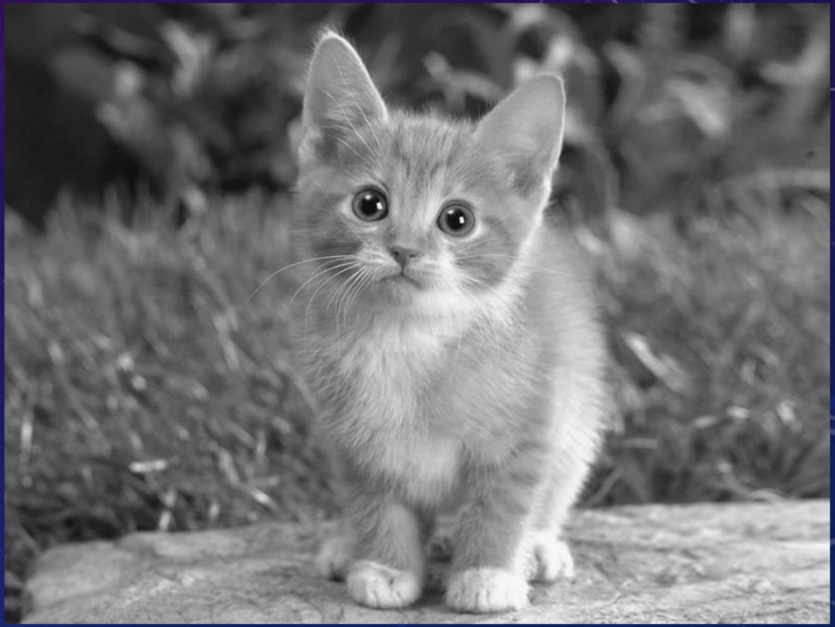
cat_gray.jpg



Solution: Contrast Stretching

```
cat_gray = (cat_gray - 71) .* 2.14
```





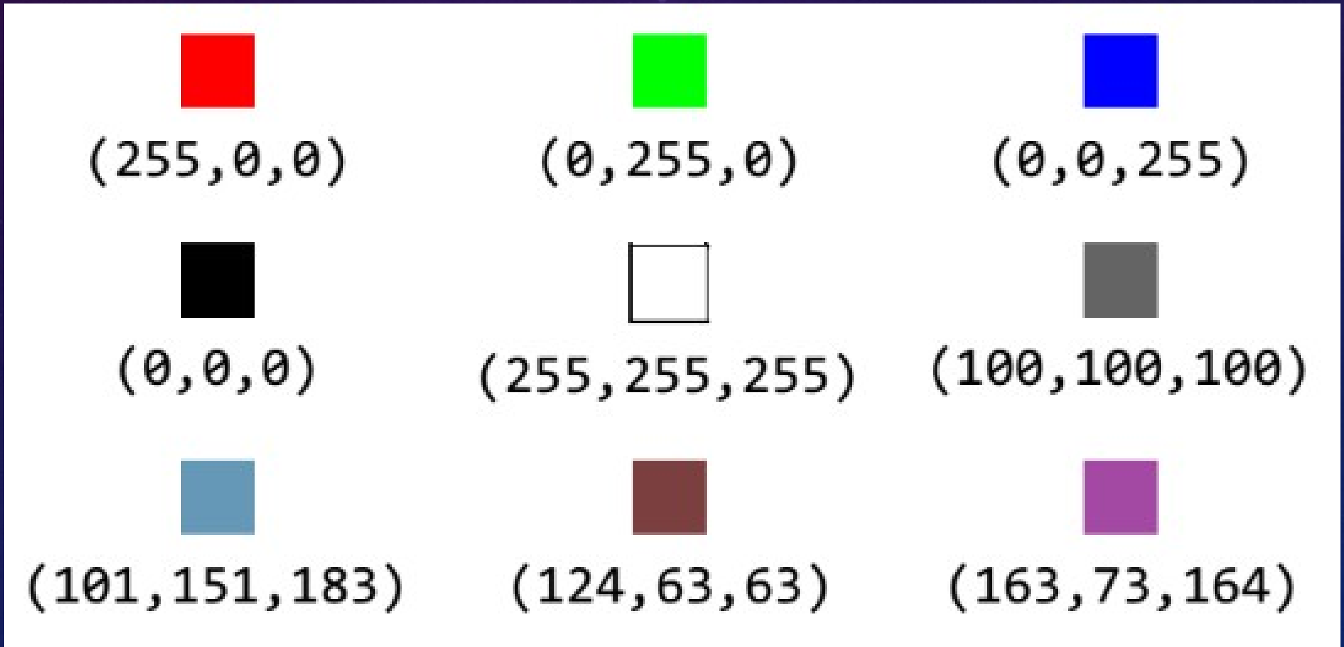
or

```
imshow(255./(max(max(cat_gray))-min(min(cat_gray))).*(cat_gray-71));
```

RGB Color Image Representation

- To represent a color, we need three different values for the amounts of the primary colors red, green, and blue.[1]

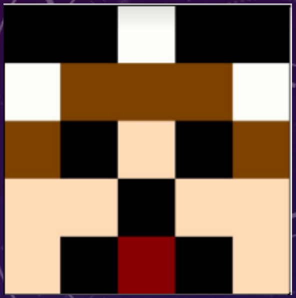
(R, G, B)



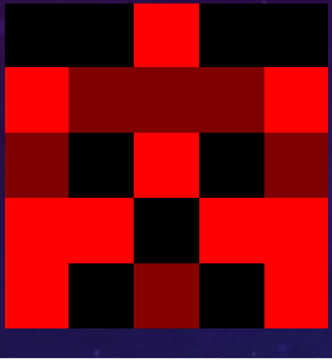
(255,0,0)	(0,255,0)	(0,0,255)
(0,0,0)	(255,255,255)	(100,100,100)
(101,151,183)	(124,63,63)	(163,73,164)

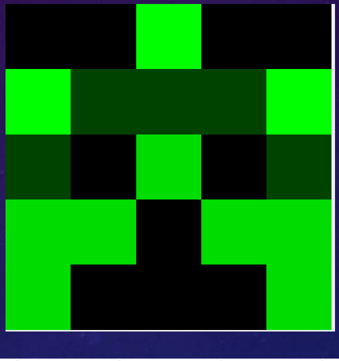
[1] These are the primary colors of light. You may also be familiar with the primary colors of pigment, which are magenta, yellow, and cyan.

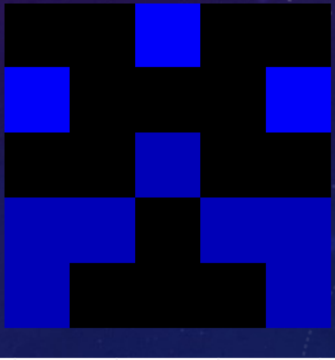
RGB Color Image Representation



- In MATLAB, a color image is represented as three different color **channels**.







0	0	255	0	0
255	126	126	126	255
126	0	255	0	126
255	255	0	255	255
255	0	134	0	255

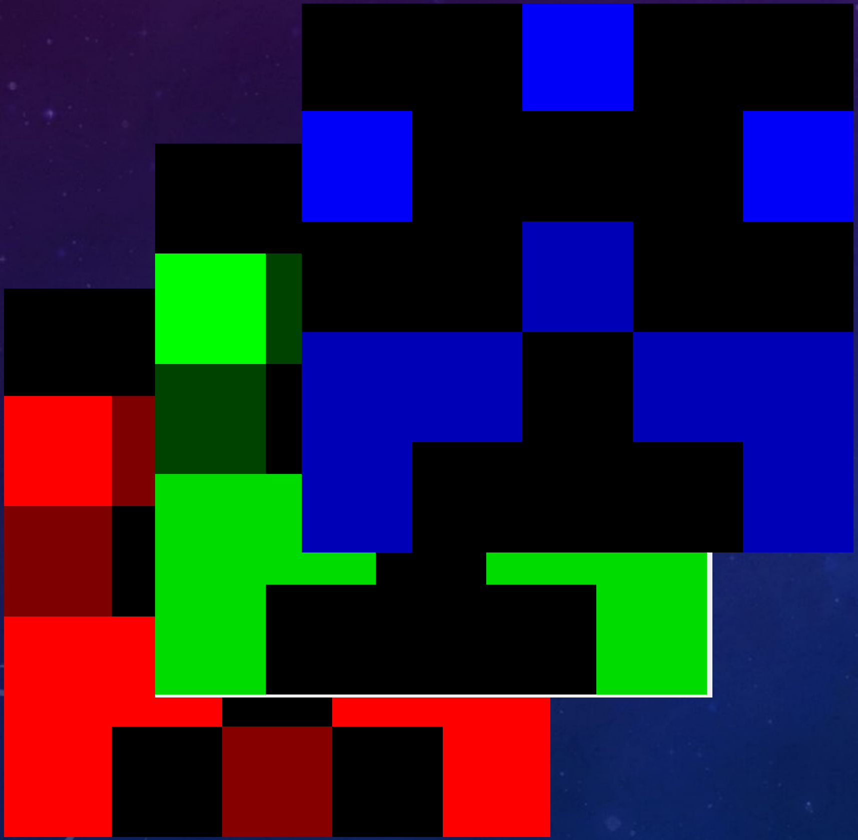
0	0	255	0	0
255	66	66	66	255
66	0	219	0	66
219	219	0	219	219
219	0	0	0	219

0	0	250	0	0
250	0	0	0	250
0	0	183	0	0
183	183	0	183	183
183	0	0	0	183

[1] These are the primary colors of light. You may also be familiar with the primary colors of pigment, which are magenta, yellow, and cyan.

RGB Color Image Representation

- We could store these color channels as three individual matrices, but then we have more things to keep track of…
- Instead, we'll layer them on top of each other in a 3D array!



0	0	250	0	0
250	0	0	0	250
0	0	183	0	0
183	183	0	183	183
183	0	0	0	183

0	0	255	0	0
255	66	66	66	255
66	0	219	0	66
219	219	0	219	219
219	0	0	0	219

0	0	255	0	0
255	126	126	126	255
126	0	255	0	126
255	255	0	255	255
255	0	134	0	255

Accessing Parts of a 3D Array

- Layers in a 3D array are controlled by a 3rd dimension.
- Row/column indexing becomes row/column/layer indexing.

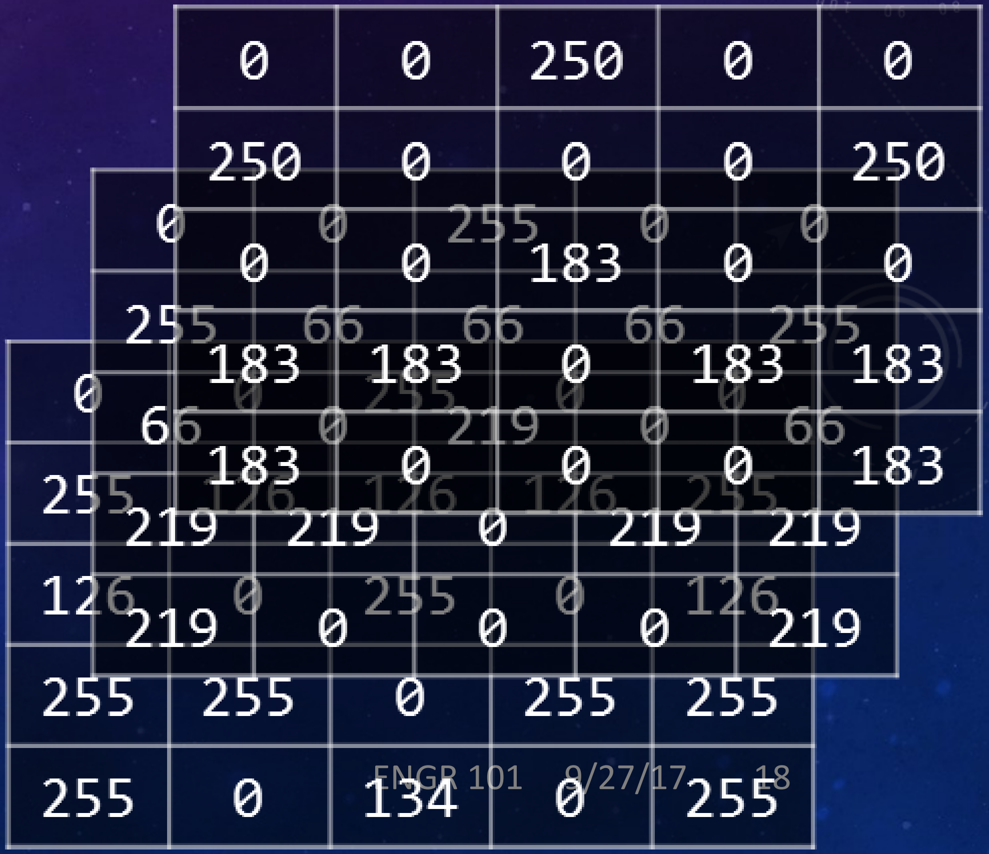
```
mat3d(:,:,1)

mat3d(:,:,3)

mat3d(4,:,2)

mat3d(:,[2,5],3)

mat3d(3,end,1)
```



0	0	250	0	0
250	0	0	0	250
0	0	183	0	0
183	183	0	183	183
183	0	0	0	183

0	0	255	0	0
255	66	66	66	255
66	0	219	0	66
219	219	0	219	219
219	0	0	0	219

0	0	255	0	0
255	126	126	126	255
126	0	255	0	126
255	255	0	255	255
255	0	134	0	255

Accessing Parts of a 3D Array

- Layers in a 3D array are controlled by a 3rd dimension.
- Row/column indexing becomes row/column/layer indexing.

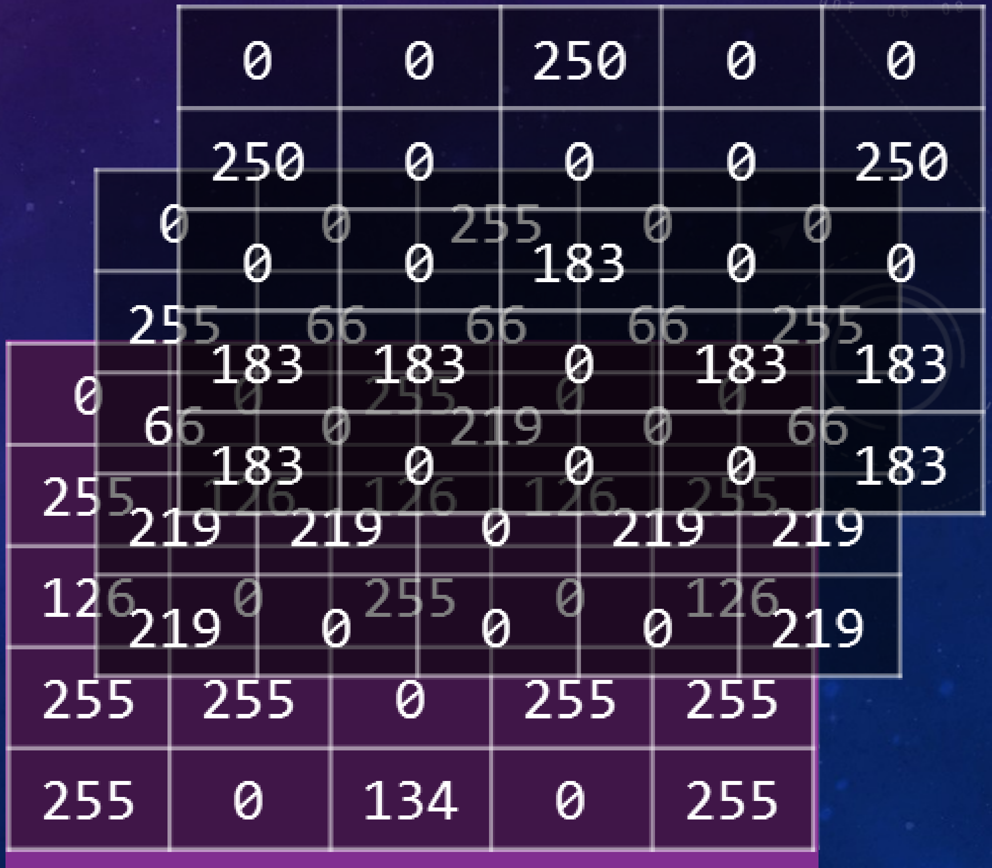
```
mat3d(:,:,1)

mat3d(:,:,3)

mat3d(4,:,2)

mat3d(:,[2,5],3)

mat3d(3,end,1)
```



0	0	250	0	0
250	0	0	0	250
0	0	183	0	0
183	183	0	183	183
183	0	0	0	183

0	0	255	0	0
255	66	66	66	255
66	0	219	0	66
219	219	0	219	219
219	0	0	0	219

0	0	255	0	0
255	126	126	126	255
126	0	255	0	126
255	255	0	255	255
255	0	134	0	255

Accessing Parts of a 3D Array

- Layers in a 3D array are controlled by a 3rd dimension.
- Row/column indexing becomes row/column/layer indexing.

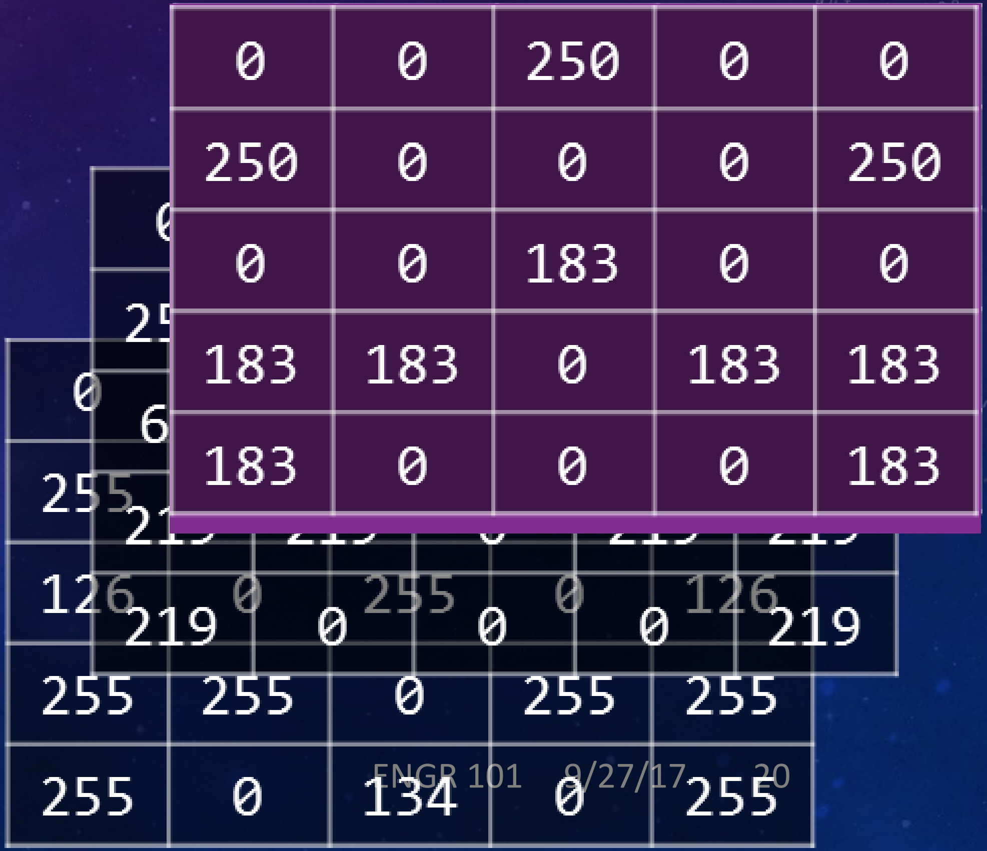
```
mat3d(:,:,1)

mat3d(:,:,3)

mat3d(4,:,2)

mat3d(:,[2,5],3)

mat3d(3,end,1)
```



0	0	250	0	0
250	0	0	0	250
0	0	183	0	0
183	183	0	183	183
183	0	0	0	183

219 0 0 0 219
255 255 0 255 255
255 0 134 0 255

Accessing Parts of a 3D Array

- Layers in a 3D array are controlled by a 3rd dimension.
- Row/column indexing becomes row/column/layer indexing.

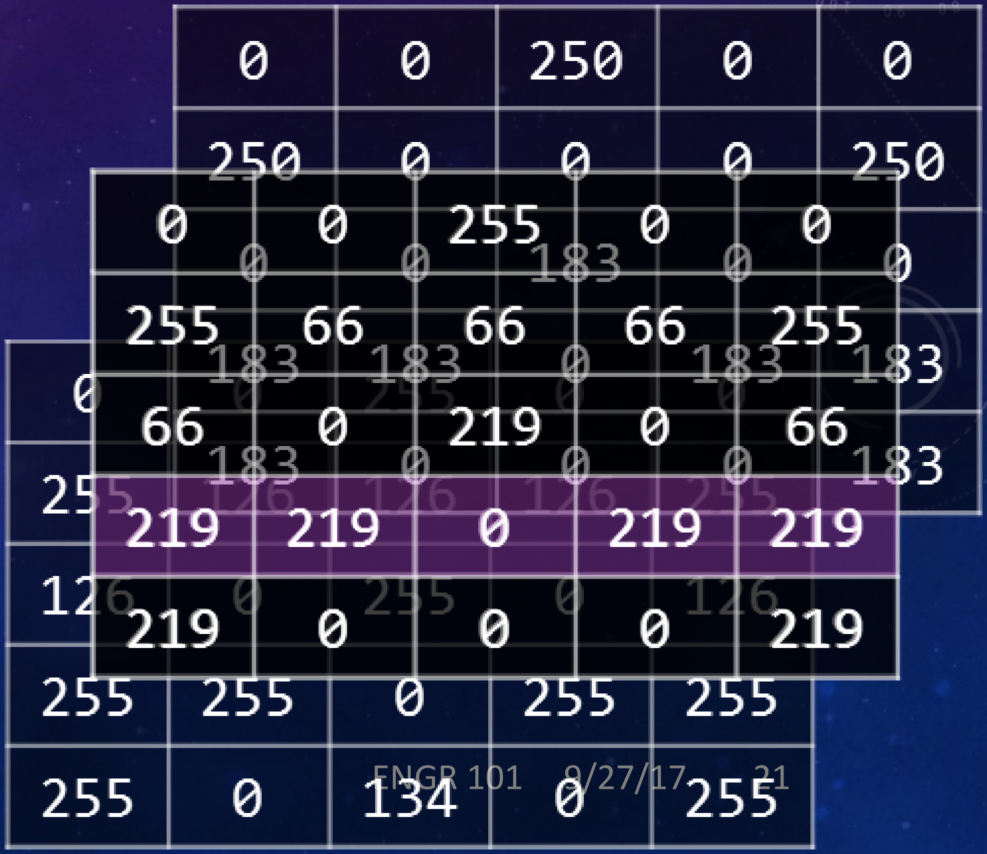
```
mat3d(:,:,1)

mat3d(:,:,3)

mat3d(4,:,2)

mat3d(:,[2,5],3)

mat3d(3,end,1)
```



Accessing Parts of a 3D Array

- Layers in a 3D array are controlled by a 3rd dimension.
- Row/column indexing becomes row/column/layer indexing.

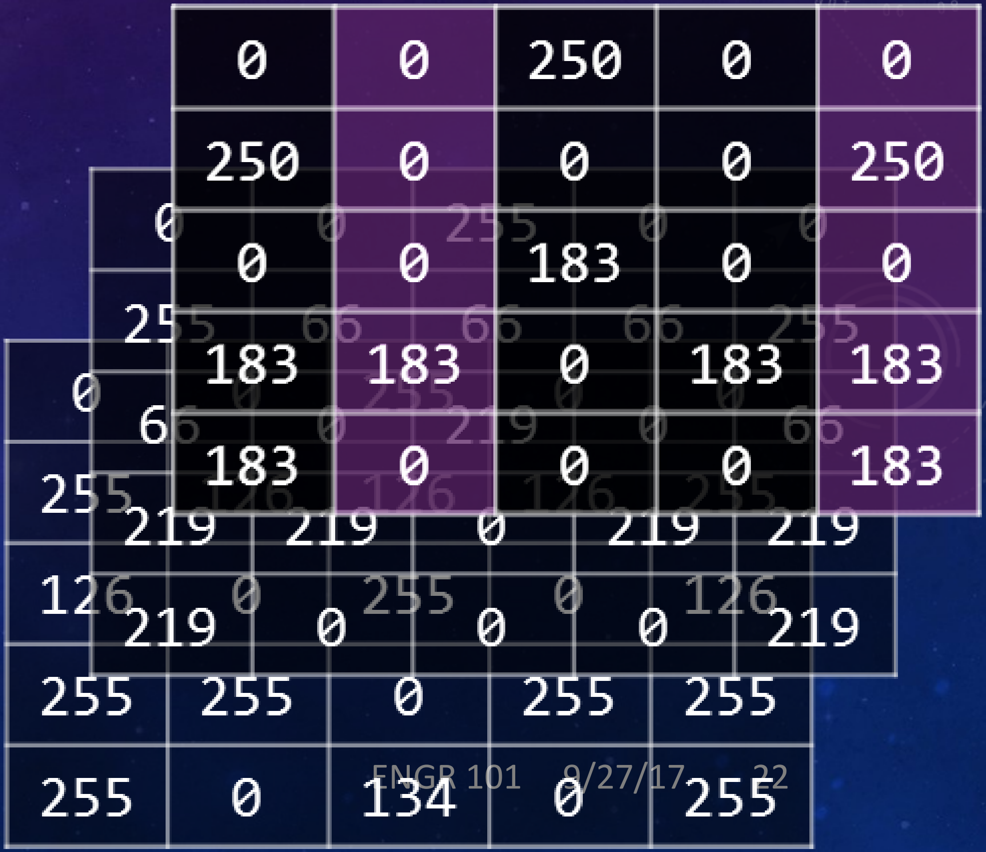
```
mat3d(:,:,1)

mat3d(:,:,3)

mat3d(4,:,2)

mat3d(:,[2,5],3)

mat3d(3,end,1)
```

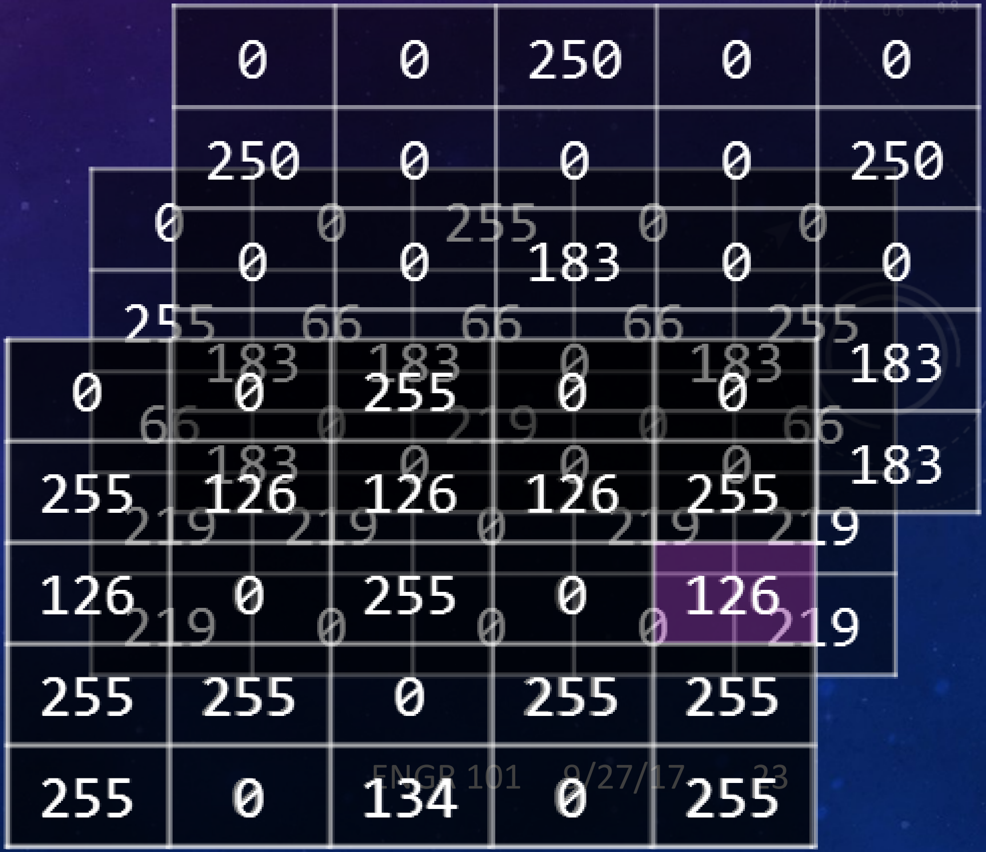


0	0	250	0	0
250	0	0	0	250
0	0	183	0	0
183	183	0	183	183
183	0	0	0	183

Accessing Parts of a 3D Array

- Layers in a 3D array are controlled by a 3rd dimension.
- Row/column indexing becomes row/column/layer indexing.

```
mat3d(:,:,1)
mat3d(:,:,3)
mat3d(4,:,2)
mat3d(:,[2,5],3)
mat3d(3,end,1)
```



0	0	250	0	0
250	0	0	0	250
0	0	183	0	0
183	183	0	183	183
183	0	0	0	183

0	0	255	0	0
255	66	66	66	255
66	0	219	0	66
219	219	0	219	219
219	0	0	0	219

0	0	255	0	0
255	126	126	126	255
126	0	255	0	126
255	255	0	255	255
255	0	134	0	255

Working With Images as 3D Arrays

- There are two main modes of operation…

"I want the whole image."

```
img(___, ___, :)
```

Use the : to select all channels.

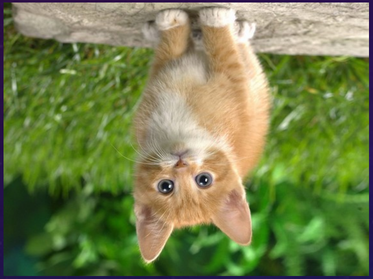
"I want a single channel."

```
img(___, ___, 2)
```

Select only the channel you want.

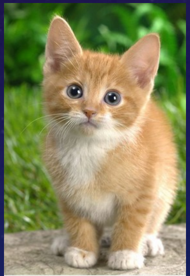
Example: Working With the Whole Image

```
cat = imread('cat_color.jpg');
imshow(cat);
```



We want to flip all the channels.

```
% Vertical Flip
vf = cat([end:-1:1], :, :);
imshow(vf);
```



Again, select all channels to be cropped.

```
% Crop the image
cropped = cat(:, [200:600], :);
imshow(cropped);
```

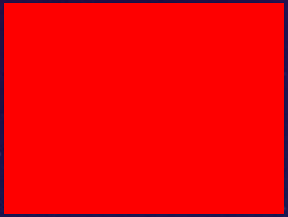
Select ONLY columns 200 through 600.

Example: Working With a Single Channel

- A pattern for working with single channels:

red

```
% Pull out the red channel
% to work with it individually
red = cat(:,:,1);
```



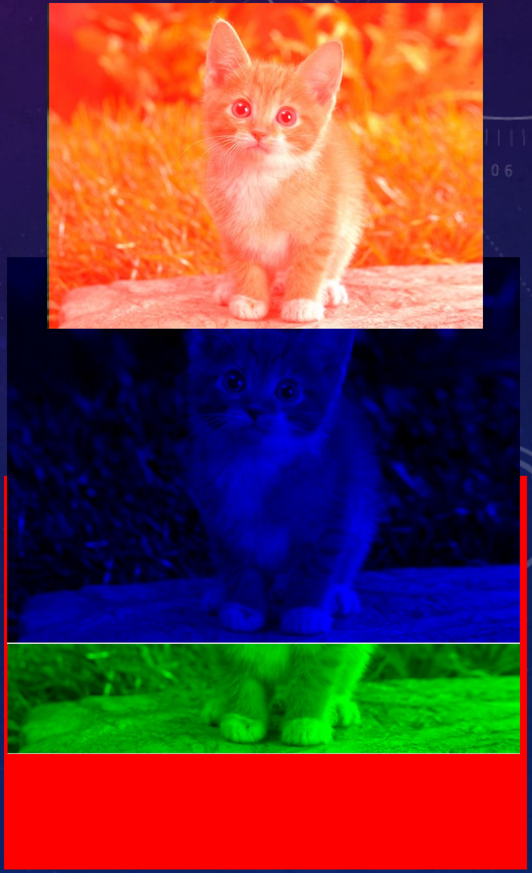
red

```
% Make changes to the red channel
red(:) = 255;
```

```
% Put the red channel back in
cat(:,:,1) = red;
```

IMPORTANT
red is a **copy** of the red channel. You need this assignment to copy the changes back in.

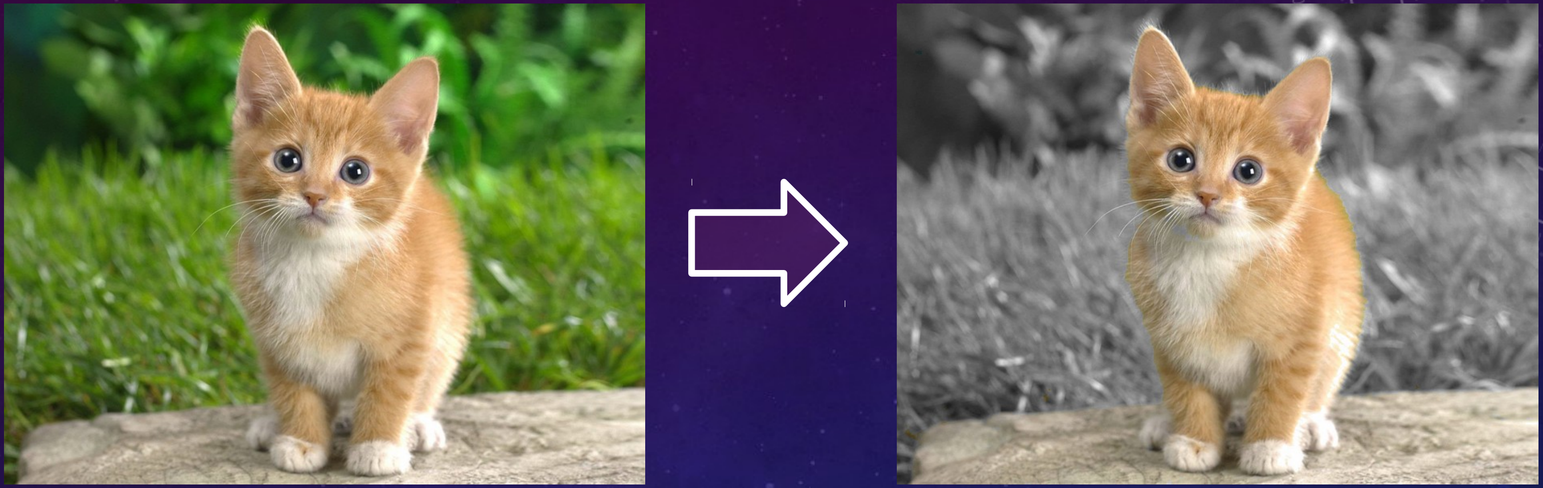
cat



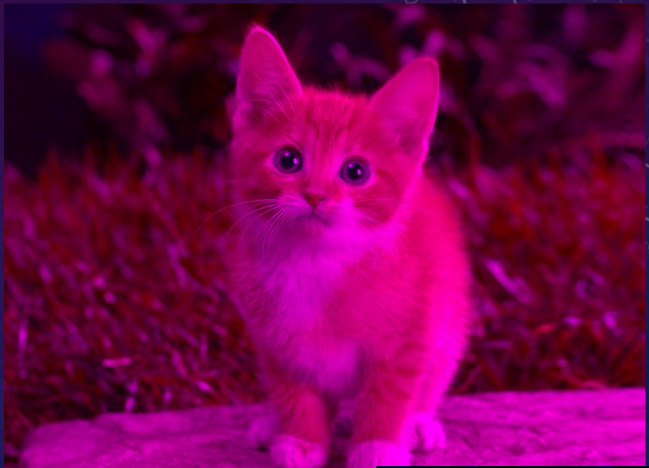
Break Time

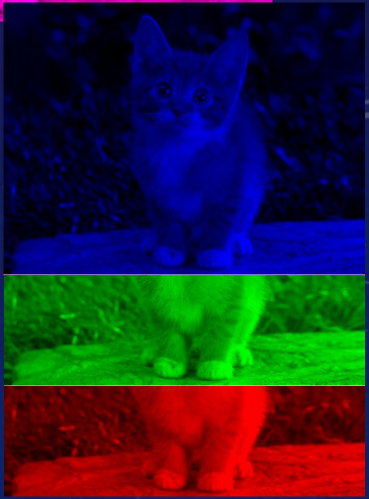
We'll start again in 5 minutes.

This looks artsy. Let's try it. Any ideas?



This looks artsy. Let's try it. Any ideas?

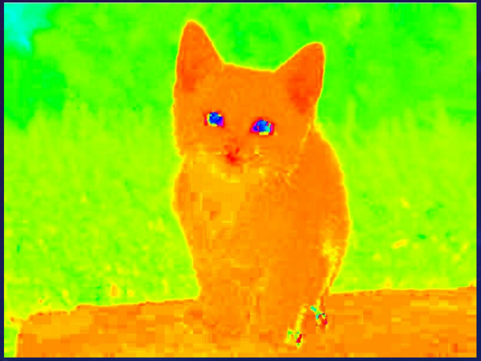




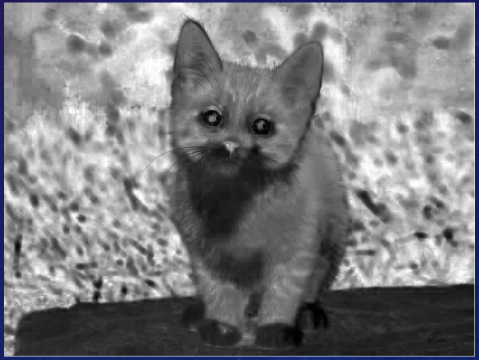
```
green = cat(:,:,2); % copy green channel
green(:) = 0;       % set to zero
cat(:,:,2) = green; % copy back into image
% It doesn't work O.o
```

HSV Color Image Representation

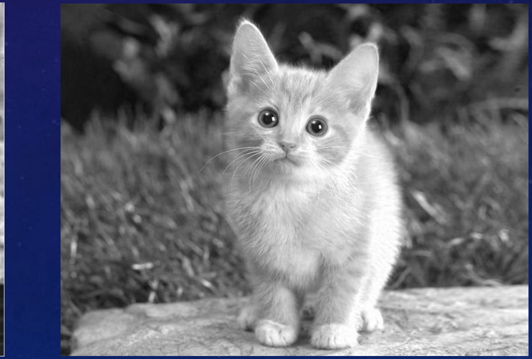
- RGB is only one of several image representations.
- HSV is an alternate that works well for certain applications.
  - **Hue**: "which color?"
  - **Saturation**: "how strong is the color?"
  - **Value**: "how bright?"



Hue



Saturation



Value

HSV Color Image Representation

- HSV images are also stored as a 3D array.
- However, in MATLAB HSV channel values range from 0.0 to 1.0
  - (In MATLAB, RGB values range between 0 and 255)

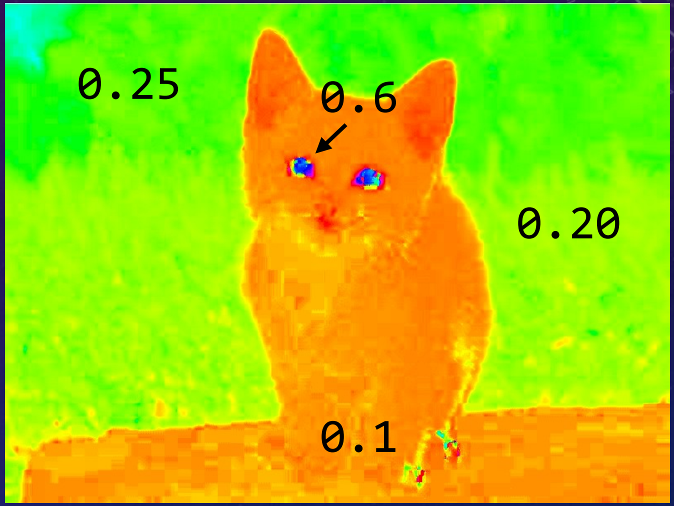
- To convert between RGB and HSV, use built-in functions:

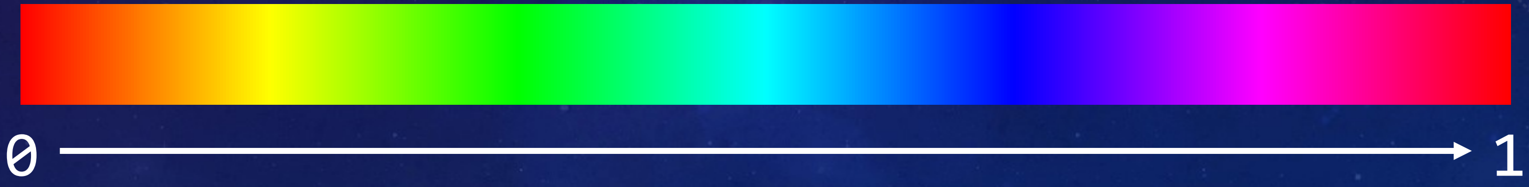
  -
    ```
    % convert img from RGB to HSV
    hsvImg = rgb2hsv(img);
    ```

  -
    ```
    % convert hsvImg from HSV back to RGB
    img = hsv2rgb(hsvImg);
    ```

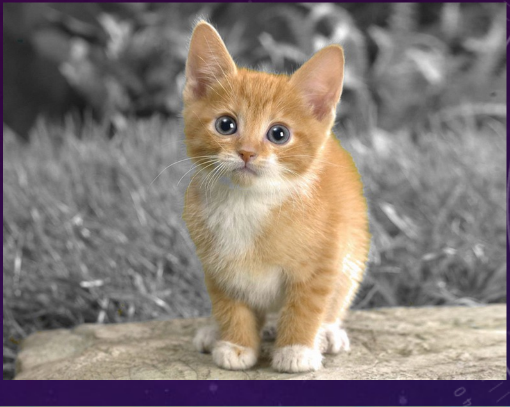
Hue

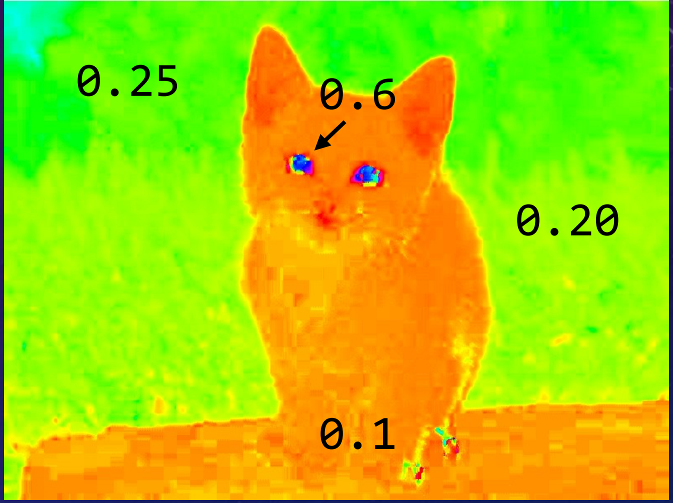
- The hue channel encodes a color as a number between 0 and 1

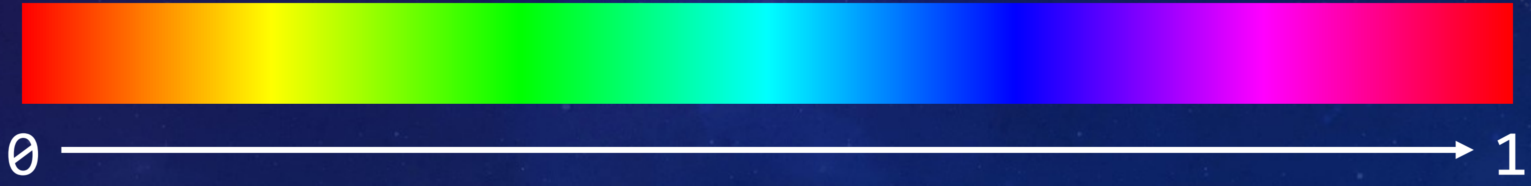




Can we use HSV to do this?

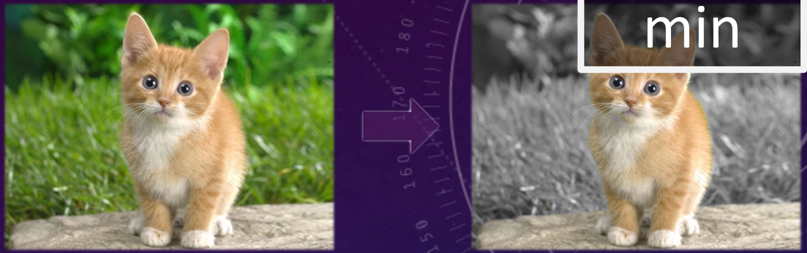


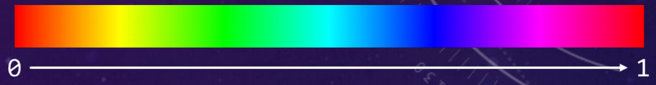




5
min

Your turn: Removing Color





- First, convert to HSV:

```
cat = imread('cat_color.jpg');
hsv = rgb2hsv(cat);
```

- Next, make copies of the hue and saturation channels to work with.

```
hue = hsv(:,:,1);
sat = hsv(:,:,2);
```

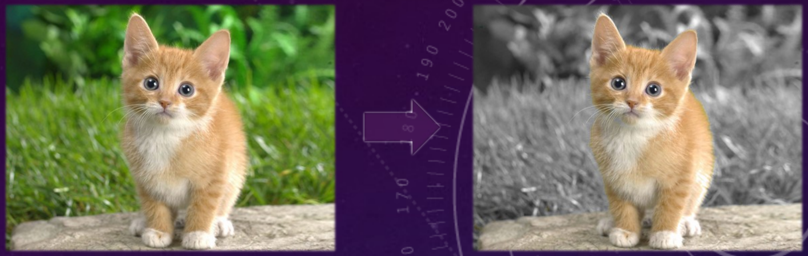
- Now, the tricky part. Find all location with a hue between 0.14 and 0.5, and set the saturation to 0 (i.e. meaning color "strength" of 0). *Hint: Use logical indexing.*

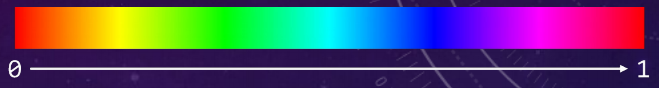
  You do this part...

- Finally, copy the saturation channel back in (it's the one we changed), convert back to rgb format, and display using `imshow`.

```
hsv(:,:,2) = sat;
imshow(hsv2rgb(hsv));
```

Your turn: Removing Color





- First, convert to HSV:

```
cat = imread('cat_color.jpg');
hsv = rgb2hsv(cat);
```

- Next, make copies of the hue and saturation channels to work with.

```
hue = hsv(:,:,1);
sat = hsv(:,:,2);
```

- Now, the tricky part. Find all locations with a hue between 0.14 and 0.5, and set the saturation to 0 (i.e. meaning color "strength" of 0). *Hint: Use logical indexing.*

```
sat(0.14 < hue & hue < 0.5) = 0;
```

- Finally, copy the saturation channel back in (it's the one we changed), convert back to rgb format, and display using `imshow`.

```
hsv(:,:,2) = sat;
imshow(hsv2rgb(hsv));
```