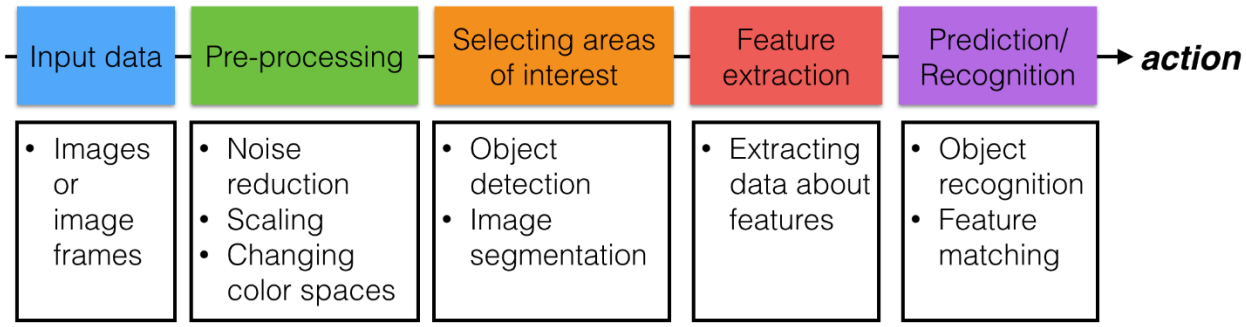
Cognitive and Emotional Intelligence

Cognitive intelligence is the ability to reason and understand the world based on observations and facts. It's often what is measured on academic tests and what's measured to calculate a person's IQ.

Emotional intelligence **is the ability to understand and influence human emotion. For example, observing that someone looks sad based on their facial expression, body language, and what you know about them - then acting to comfort them or asking them if they want to talk, etc. For humans, this kind of intelligence allows us to form meaningful connections and build a trustworthy network of friends and family. It's also often thought of as only a human quality and is not yet a part of traditional AI systems.**

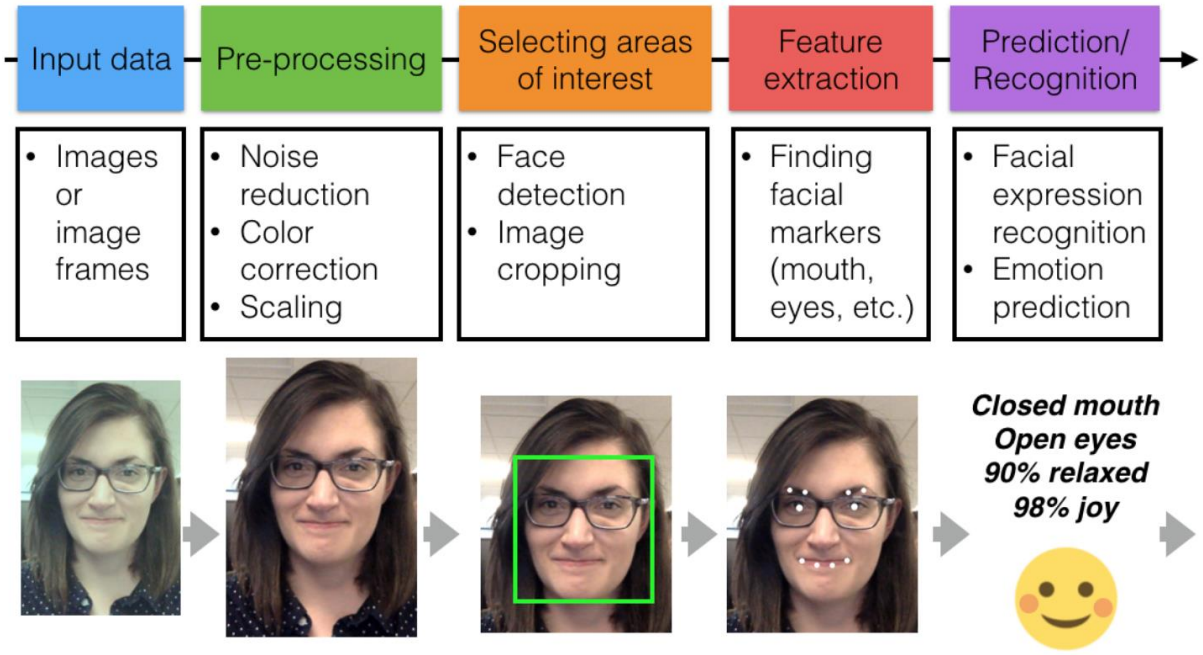
Computer Vision Pipeline

A computer vision pipeline is a series of steps that most computer vision applications will go through. Many vision applications start off by acquiring images and data, then processing that data, performing some analysis and recognition steps, then finally performing an action. The general pipeline is pictured below!



General computer vision processing pipeline

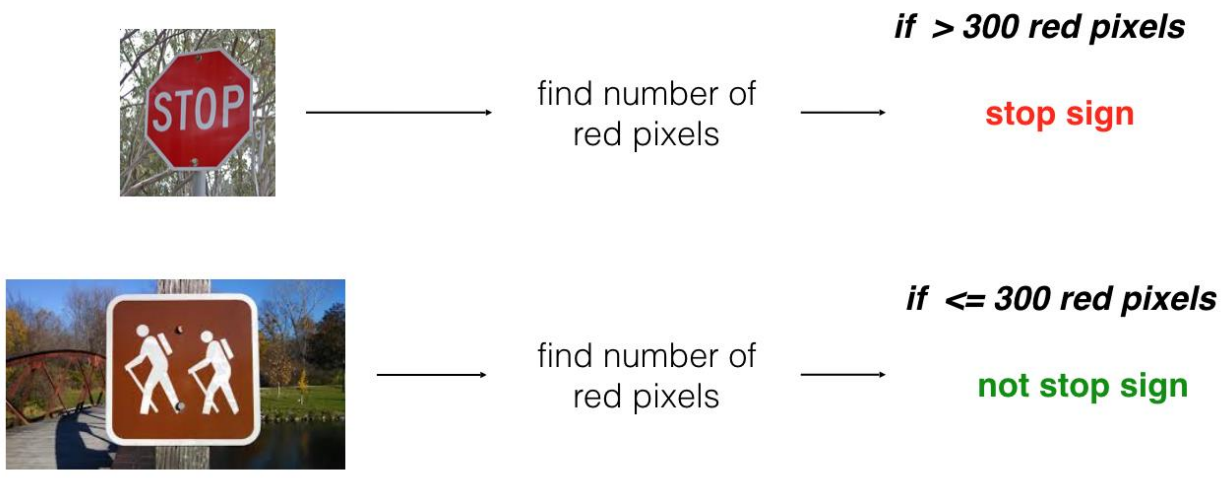
Now, let's take a look at a specific example of a pipeline applied to facial expression recognition.



Facial recognition pipeline.

Standardizing Data

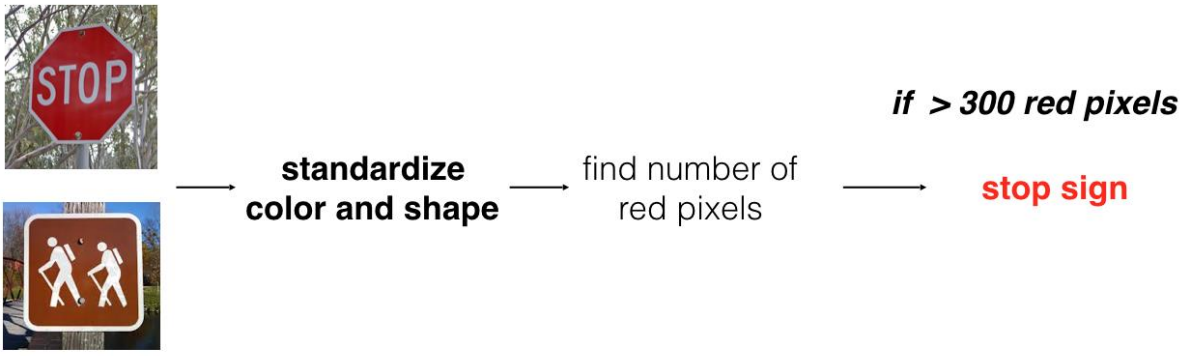
Pre-processing images is all about **standardizing** input images so that you can move further along the pipeline and analyze images in the same way. In machine learning tasks, the pre-processing step is often one of the most important.
For example, imagine that you've created a simple algorithm to distinguish between stop signs and other traffic lights.



Images of traffic signs; a stop sign is on top and a hiking sign is on the bottom.

If the images are different sizes, or even cropped differently, then this counting tactic will likely fail! So, it's important to pre-process these images so that they are standardized before they move along the pipeline. In the example below, you can see that the images are pre-processed into a standard square size.

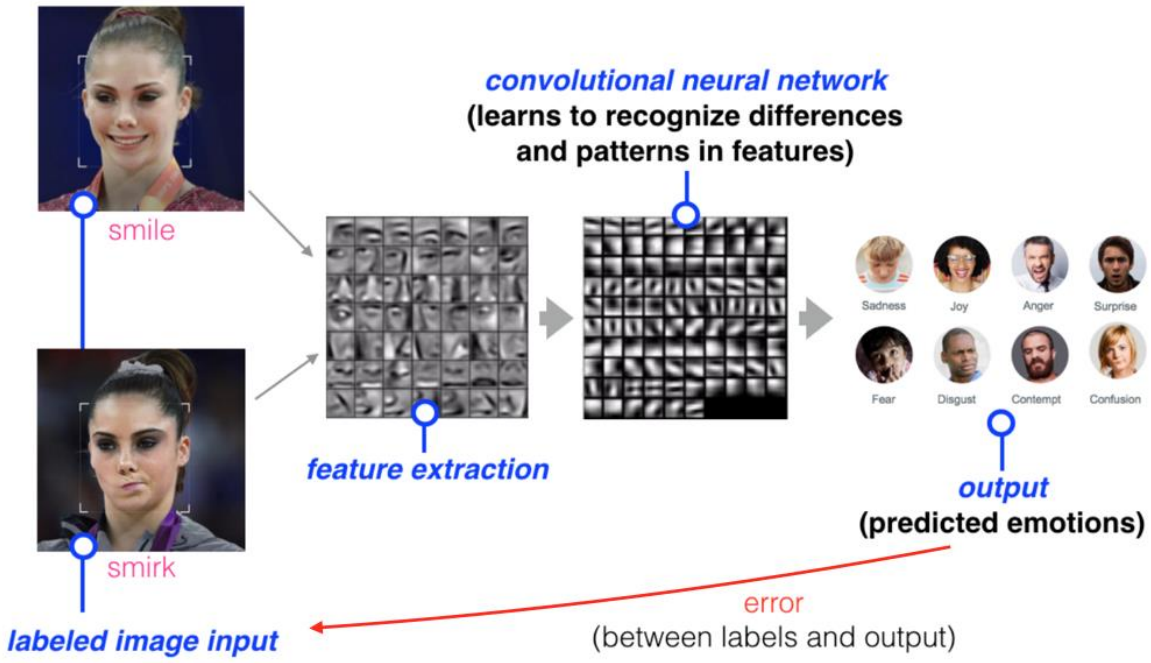
The algorithm counts up the number of red pixels in a given image and if there are enough of them, it classifies an image as a stop sign. In this example, we are just extracting a color feature and skipping over selecting an area of interest (we are looking at the *whole* image). In practice, you'll often see a classification pipeline that looks like this.



Training a Neural Network

To train a computer vision neural network, we typically provide sets of **labelled images**, which we can compare to the **predicted output** label or recognition measurements. The neural network then monitors any errors it makes (by comparing the correct label to the output label) and corrects for them by modifying how it finds and prioritizes patterns and differences among the image data. Eventually, given enough labelled data, the model should be able to characterize any new, unlabeled, image data it sees!
A training flow is pictured below. This is a convolutional neural network that *learns* to recognize and distinguish between images of a smile and a smirk.
This is a very high-level view of training a neural network, and we'll be diving more into how this works later on in this course. For now, we are explaining this so that you'll be able to jump into coding a computer vision application soon!

IMAGE REPRESETATION



Example of a convolutional neural network being trained to distinguish between images of a smile and a smirk.

Gradient descent *is a a mathematical way to minimize error in a neural network. More information on this minimization method can be found here.*
Convolutional neural networks are a specific type of neural network that are commonly used in computer vision applications. They learn to recognize patterns among a given set of images. If you want to learn more, refer to this resource, and we'll be learning more about these types of networks, and how they work step-by-step, at a different point in this course!

Sure. The process is similar to the pipeline you just described

We have 45 facial muscles that drive thousands that drive thousands of different expressions on our face.

For example.

1- When your smile, you teeth might show, but that`s not the case with a smirk.

Machine Learning and Neural Networks

When we talk about **machine learning** and **neural networks** used in image classification and pattern recognition, we are really talking about a set of algorithms that can *learn* to recognize patterns in data and sort that data into groups.
The example we gave earlier was sorting images of facial expressions into two categories: smile or smirk. A neural network might be able to learn to separate these expressions based on their different traits; a neural network can effectively learn how to draw a line that **separates** two kinds of data based on their unique shapes (the different shapes of the eyes and mouth, in the case of a smile and smirk). *Deep* neural networks are similar, only they can draw multiple and more complex separation lines in the sand. Deep neural networks layer separation layers on top of one another to separate complex data into groups.

Separating Data

Say you want to separate two types of image data: images of bikes and of cars. You look at the color of each image and the apparent size of the vehicle in it and plot the data on a graph. Given the following points (pink dots are bikes and blue are cars), how would you choose to separate this data?

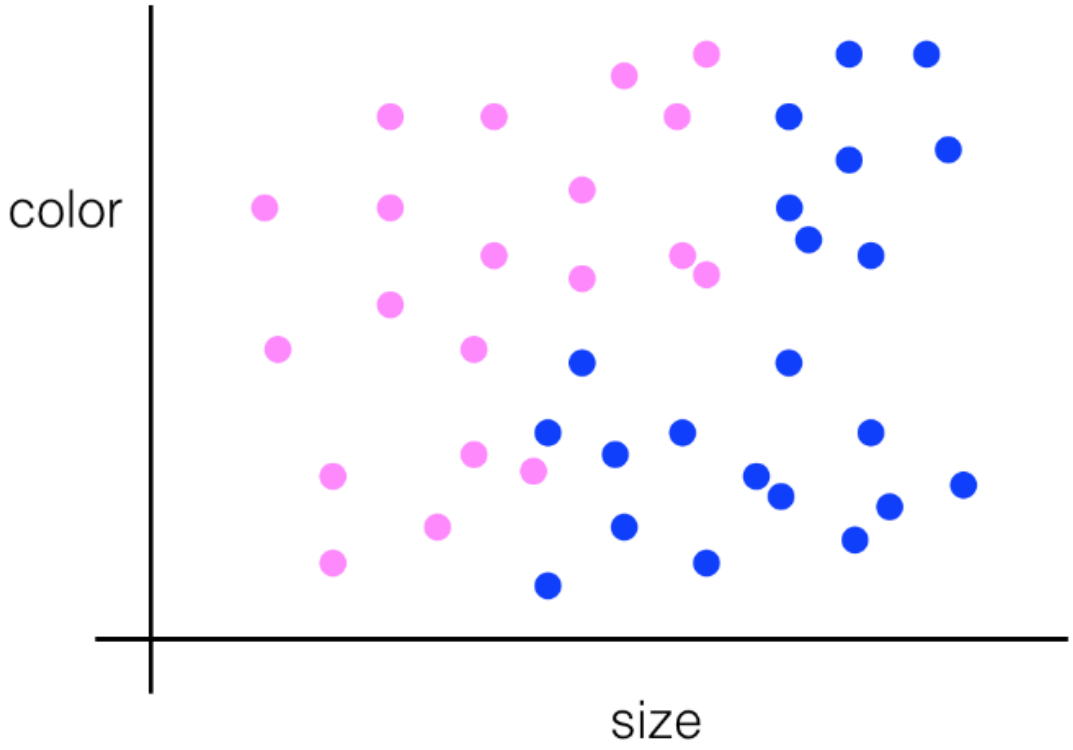
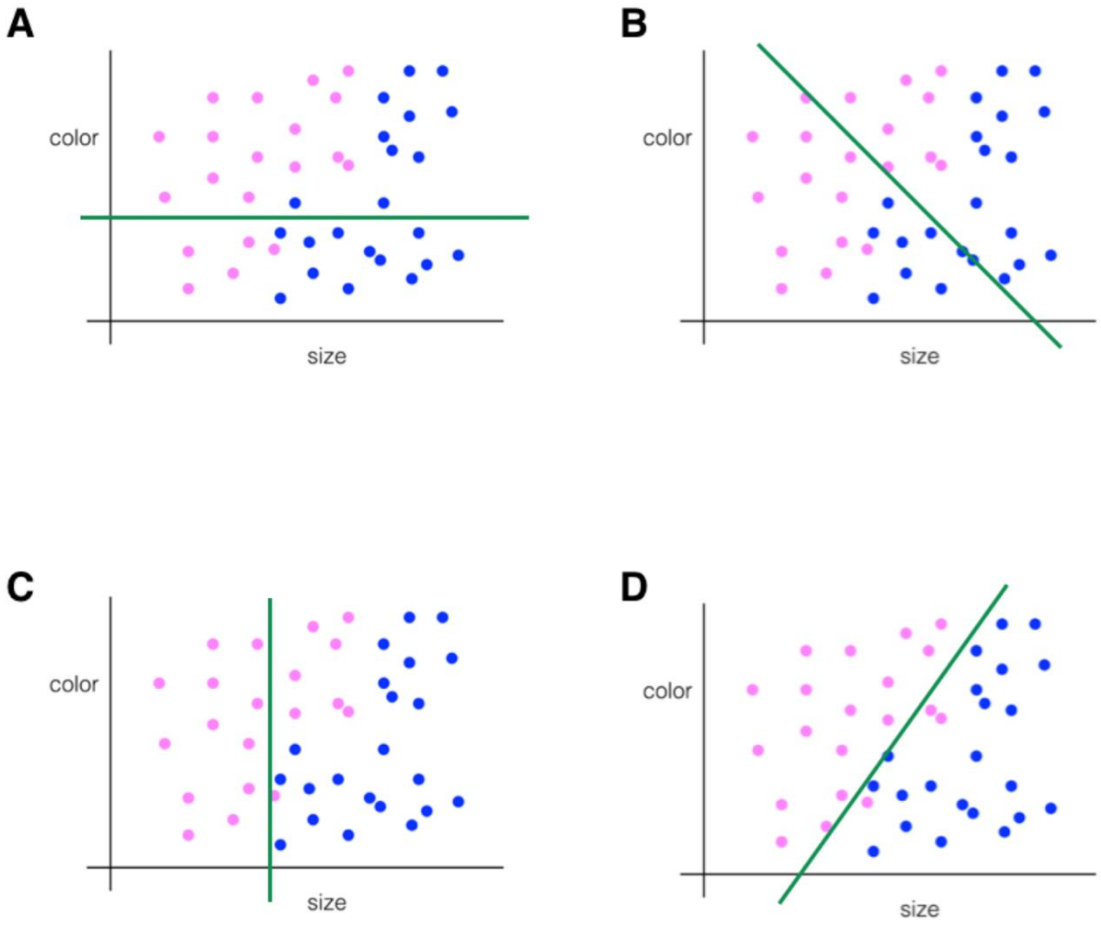


IMAGE REPRESETATION

Pink and blue dots representing the size and color of bikes (pink) and cars (blue). The size is on the x-axis and the color on the left axis. Cars tend to be larger than bikes, but both come in a variety of colors.



QUIZ QUESTION

Given the above choices, which line would you choose to best separate this data?

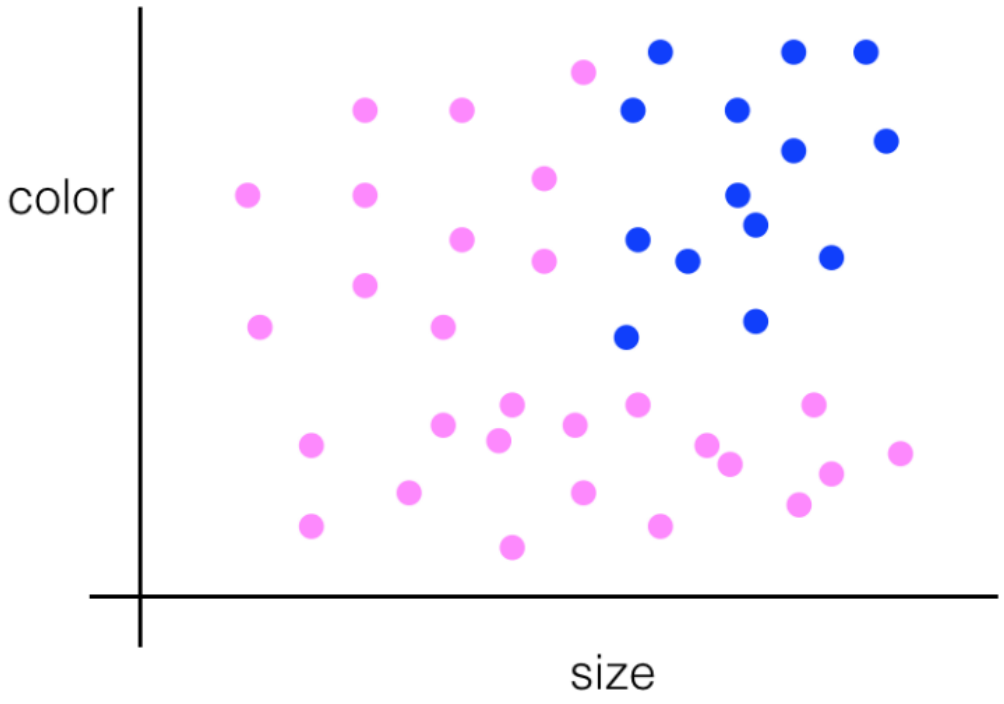
- A (horizontal line)
- B (diagonal line from top-left to bottom-right)

- C (vertical line)
- D (diagonal line from top-right to bottom-left)

SUBMIT

Layers of Separation

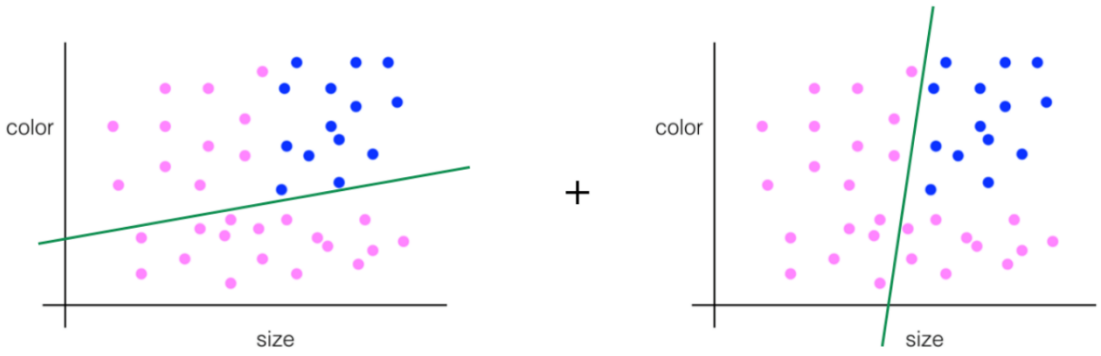
What if the data looked like this?



Pink (bike) and blue (car) dots on a similar size-color graph. This time, the blue dots are collected in the top right quadrant of the graph, indicating that cars come in a more limited color palette.

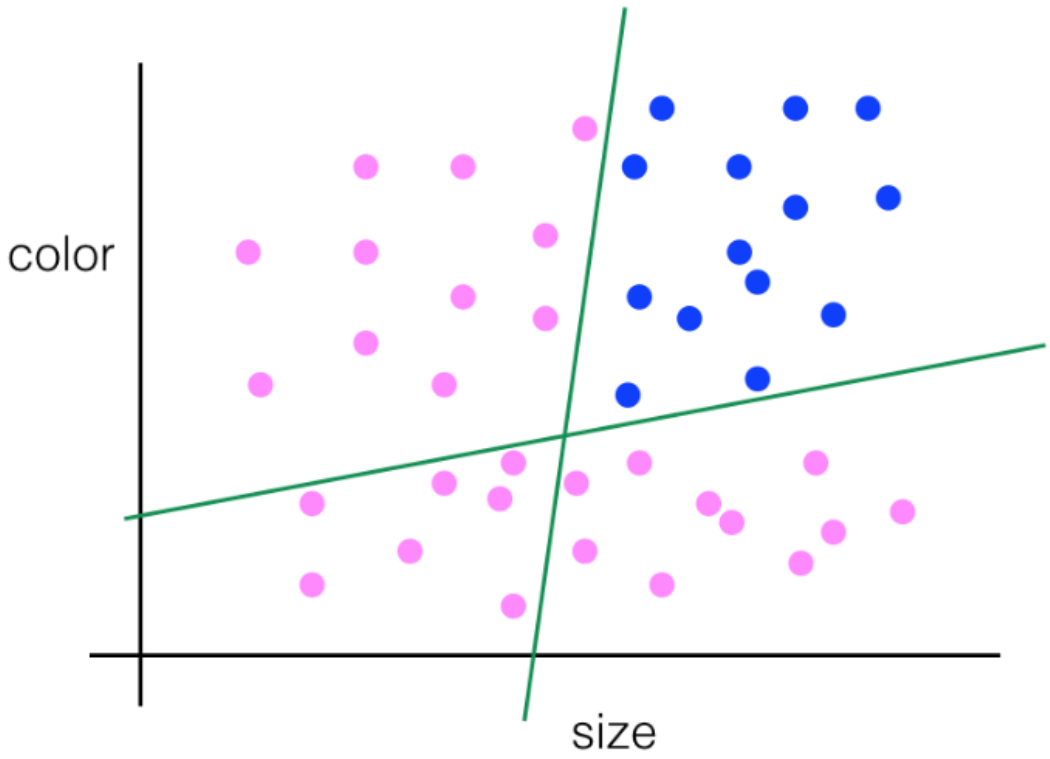
You could combine two different lines of separation! You could even plot a curved line to separate the blue dots from the pink, and this is what machine learning *learns* to do — to choose the best algorithm to separate any given data.

IMAGE REPRESETATION



Two, slightly-angled lines, each of which divides the data into two groups.

For



Both lines, combined, clearly separate the car and bike data!

IMAGE REPRESETATION

What is the point in the bounding box ? it`s mapping it into a probability score for each emotion. So *let`s give this is a try.*

We can see that the probability score of the joy classifier goes up.

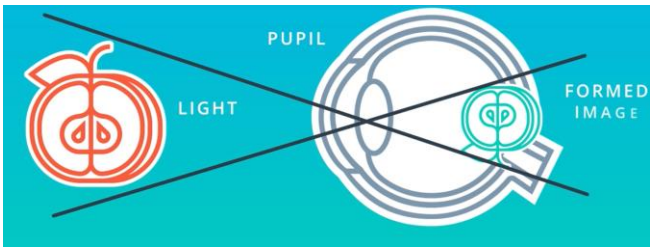
It`s also mapping your most dominant emotion into an emoji. So you can experiment with you know.

It also detects the presence of glasses and gender.

The image contain details about

1- *The color*
2- *The shape*
3- *It also has shading that varies based on lighting condition's,*
4- *And Apparent size that varies based on how close or far away the picture has been taken.*

For example, the apple will appear bigger the closer the camera is to it. When a camera forms an image like this,



It`s looking at the world similar to how our eyes do, by focusing the light that`s reflected off of objects in the world

Let`s see the example

- *Here is a simple model of a camera called a pinhole camera model.*

In this case through a small pinhole, the camera focus the light that`s reflected off an apple and forms a 2d image a the back of the camera where a sensor or some film would be placed.

***In fact**, the image it forms here will be upside down and reversed because rays of light that enter from the top of an object will continue on the angled path through pinhole and end up at the bottom of the formed image.*

1- *Light that reflects off the right side of an object will travel to the left of the formed image.*

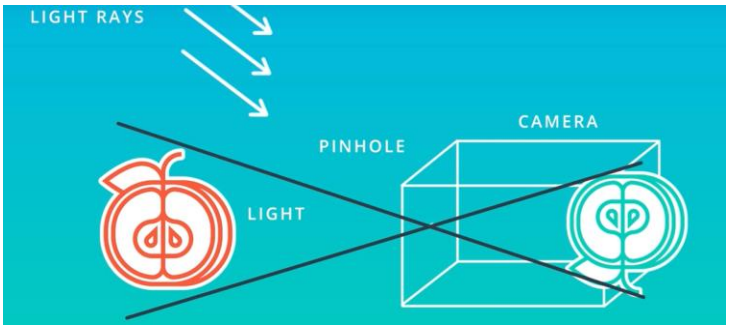


IMAGE REPRESETATION

A digital camera will record this image and flip it to give us a familiar 2d image of an apple or any other object.

images as Numerical Data

Every pixel in an image is just a numerical value and, we can also change these pixel values. We can multiply every single one by a scalar to change how bright the image is, we can shift each pixel value to the right, and many more operations!

Treating images as grids of numbers is the basis for many image processing techniques.
Most color and shape transformations are done just by mathematically operating on an image and changing it pixel-by-pixel.

Now image of car is 427 pixels in height and 640 in width. And the pixel locations are on a gird that starts at index zero.

1- *From zero to 369 columns, and from zero to 426 rows.*

An example. At the location x equals 190 and y equals 375, we have an pixel o this wheel at the bottom left of the image.

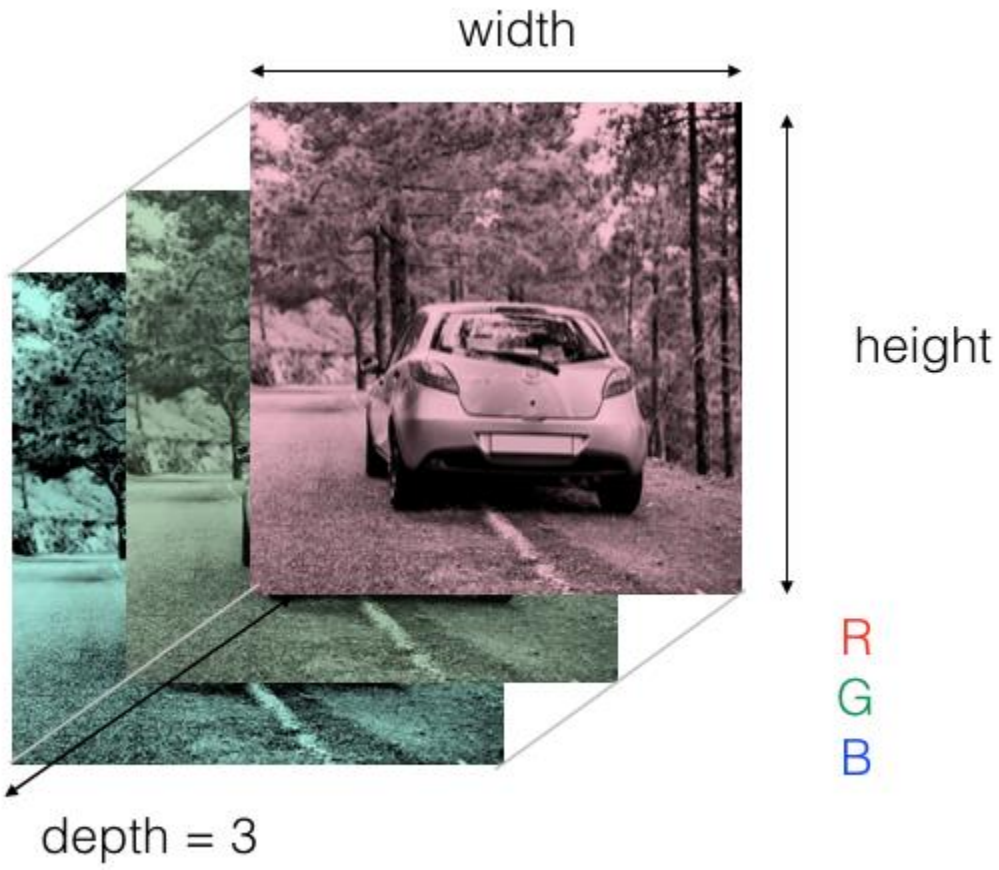
The pixel value is 28 a dark dark grey

Color Images

Color images are interpreted as 3D cubes of values with width, height, and depth!

The depth is the number of colors. Most color images can be represented by combinations of only 3 colors: red, green, and blue values; these are known as RGB images. And for RGB images, the depth is 3!

It's helpful to think of the depth as three stacked, 2D color layers. One layer is Red, one Green, and one Blue. Together they create a complete color image.



RGB layers of a car image.

Importance of Color

In general, when you think of a classification challenge, like identifying lane lines or cars or people, you can decide whether color information and color images are useful by thinking about your own vision.

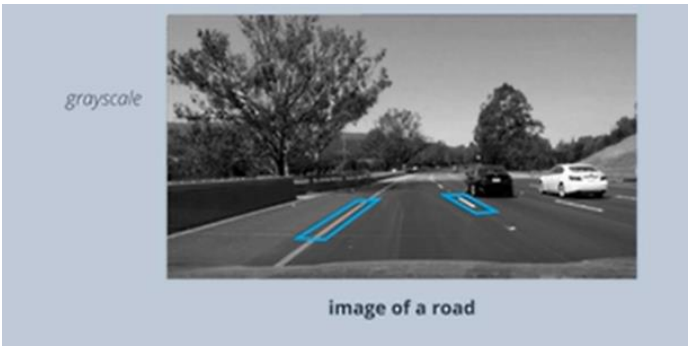
If the identification problem is easier in color for us humans, it's likely easier for an algorithm to see color images too!

IMAGE REPRESETATION

For example, say you want to classify lane lines in this image of a road.

One of these lines is yellow and one is white but which is which?

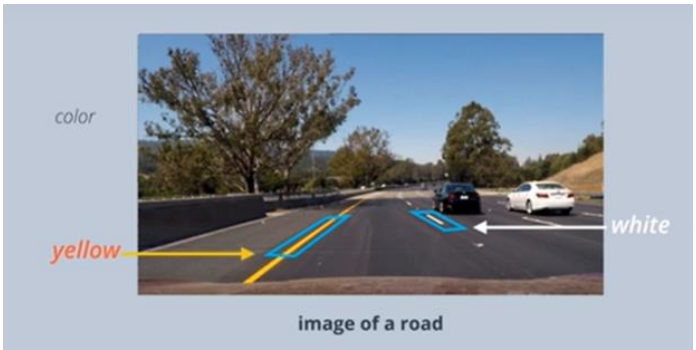
You might see slight difference in the grayscale intensity of lane lines but the difference is so small and it varies under different lighting conditions.



So, this grayscale image does not provide enough information to distinguish between the yellow and white lane lines.

Let`s see the color image for comparison.

Here we can see the difference between white and yellow lane lines.



And so we can tell the machine to recognize the difference too. So because this identifications task is dependent on color, it`s important that we work with color images.

When is color important with color images?

*"**In general, if objects or traits are easier to identify in color for us humans it`s better to provide color images algorithms**."*

When you think the computer vision applications like identifying lane lines or cars or people, you can decide whether color information and color images are useful, by thinking about your own vision.

1- *If the identifications problem is easier in color for us humans, it`s likely easier for an algo him to see color images too*

QUIZ QUESTION

For each recognition task listed, check the box if **color is necessary** or would be extremely helpful in completing the task. Leave a box *un-checked* if grayscale images would be sufficient for the task. (multiple boxes may be checked)

- *Identifying different types of traffic lights (red, yellow, and green).*
- *Recognizing a red stop sign.*

OpenCV

OpenCV is a popular computer vision library that has meany built in tools for image analysis and understanding!
Note: In the example above and in later examples, I'm using my own Jupyter notebook and sets of images stored on my personal computer. You're encouraged to set up a similar environment and use images of your own to practice! You'll also be given some code quizzes (coming up next), with images provided, to practice these techniques.

Why BGR instead of RGB?

OpenCV **reads in images in BGR format** (instead of RGB) because when OpenCV was first being developed, BGR color format was popular among camera manufacturers and image software providers. The red channel was considered one of the least important color channels, so was listed last, and many bitmaps use BGR format for image storage. However, now the standard has changed and most image software and cameras use RGB format, which is why, in these examples, it's good practice to initially convert BGR images to RGB before analyzing or manipulating them.

Changing Color Spaces

To change color spaces, we used OpenCV's `cvtColor` function, whose documentation is here.

Image mask : Mask is very common way to isolate a selected area of interest and do something with that area.

1- We can create a mask over the blue area using OpenCv in inRange Function.

‘’’

This function

1- take this image
2- Our lower
3- And upper color bounds

And define the mask and asking of the color value of each image pixel falls in the range of the lower and upper color thresholds.
If does fall in this range, the mask will be allowed it to be displayed and if not, it will block it out and turn the pixel back.

In fact, we can visualize the mask by plotting it as would an image

1- *This white area is where the image will be allowed to show through and the black will be blocked out.*

In numerical values, we can look at this mask as a 2d grid with the same dimension as out image;

1- *514 in height*
2- *816 in width*

And each coordinate in this mask has a value of either 255 for white or 0 for black, sort of like a grayscale image.

And the first thing we want to do is let the pizza show through and block the blue screen background.

- *First step we called mask image of our color changed image copy,*
  - *One way to select the blue screen area is by asking for the part of that image that overlaps with the part of the mask that is **white** or not **black**.*
  - *We will select the part of image where the area of mask is not equal to zero, using the exclamation point equals Boolean operator where the exclamation point mean not.*
- *And to block this background area out we than set these pixel to black.*
- *(in RGB black is just zeros fall all three color values) when we display image the pizza area is the only tah show through the blue scree back ground is gone.*

IMAGE REPRESETATION

Now we just have one last step which is to apply a background to this image

1- *First, I will read in image of outer space and convert it to RGB COLOR.*
2- *I will also crop it so that it`s same size as our pizza image;*
3- *I`m calling the mage*
4- *I apply the mask this time using opposite mask, meaning I want the background to show through and not the pizza area.*
5- *If you look back at the mask in this case I'm blocking the part of the background image where the mask is equal zero and for this we say mask == 0*
6- *I will plot the resulting image and I get the background with the pizza cut out*
7- *Finally I just need to add two these two images together. Since the black area is equivalence to zero in pixel color values, a simple addition to want*

Color threshold depend on

1- *Even lighting and consistent blue color.*

If you have shadow ?

1- ~~*Even lighting and consistent blue color.*~~

But color fails!

Color space

1- *RGB*
2- *HSV*
3- *HLS*

- *Color space – Isolate the value (V) component which varies the most under different* ***lighting conditions****.*
- *The H channel says* ***fairly consistent*** *in* ***shadow*** *or* ***excessive brightness****.*

And if we rely mainly on this channel and discard the information in the v channel, we should be able to detect colored objects more reliably than IN RGB color space.

To see the relative value of these color or isolate each of these color values, which I will call color channels , and display them.

1- *To isolate the red channel image array, image copy and take all the x and the y values in the first two array column.*
2- *Then the zero index of the third column which is the red value of each pixel.*
3- *Then I plot each of these channels in grayscale to see their relative intensities,*
   a- *The brighter pixels indicates higher values of red, green or blue, respectively.*

We can see that the pink balloons have high values for red and medium high for blue.

But there`s a lot of variations especially when the balloon is in shadow.

Now repeat this same process for HSV color space.

First, only to convert this image to HSV color space using CVT color as before

The same process

Here are three channels the hue, saturation, and value. Compare with the original image look?

Here we can see that the hue channel is pretty high for pink balloons. And even of shadow the hue level is pretty consistent.

The saturation and value channels vary a lot more especially under shadow and the edges of balloons.

The next step creates a color threshold in both these colors spaces to compare.

1- *Here I `v determined the pink color range using a color selector, and defined the lower and upper boundaries in our RGB values to reflect this.*
2- *These thresholds allow some high values of red and medium values of blue.*

Next, I will do same thing in HSV color space.

(Remember that use only goes from 0 - 180 as measurement of degrees.)

And I'm allowing any value for saturation and value channels 0 - 255

Create a mask image to selects the pink balloons.

1- *First I create the RGB mask.*
2- *Apply the mask , setting the pixel equal to zero where the mask is equal to zero or black.*
3- *Show result*

In same process in HSV image creating mask and passing lower and upper and image

Show result

Color Selection

To select the most accurate color boundaries, it's often useful to use a **color picker** and choose the color boundaries that define the region you want to select!

Why do we need labels?

You can tell if an image is night or day, but a computer cannot unless we tell it explicitly with a label!
This becomes especially important when we are testing the accuracy of a classification model.

A classifier takes in an image as input and should output a `predicted_label` that tells us the predicted class of that image. Now, when we load in data, like you've seen, we load in what are called the `true_labels` which are the *correct* labels for the image.
To check the accuracy of a classification model, we compare the predicted and true labels. If the true and predicted labels match, then we've classified the image correctly! Sometimes the labels do not match, which means we've misclassified an image.

A misclassified image example. The true_label is "day" and the predicted_label is "night".

Accuracy

After looking at many images, the accuracy of a classifier is defined as the number of correctly classified images (for which the predicted_label matches the true label) divided by the total number of images. So, say we tried to classify 100 images total, and we correctly classified 81 of them. We'd have 0.81 or 81% accuracy!

We can tell a computer to check the accuracy of a classifier only when we have these predicted and true labels to compare. We can also learn from any mistakes the classifier makes, as we'll see later in this lesson.

Numerical labels

It's good practice to use numerical labels instead of strings or categorical labels. They're easier to track and compare. So, for our day and night, binary class example, instead of "day" and "night" labels we'll use the numerical labels: 0 for night and 1 for day.

Okay, now you're familiar with the day and night image data AND you know what a label is and why we use them; you're ready for the next steps. We'll be building a classification pipeline from start to end!

Let's first brainstorm what steps we'll take to classify these images.

Say you have 500 day and night test images, and you send all of them through a classifier. What is the accuracy of this classifier if it *misclassifies* 80 images?
500 – 80 = 420

Accuracy = 420 / 500 = 0.84

After visualizing the day and night images, what traits do you think distinguish the two classes?
There are many traits that distinguish a night from a day image. You may have thought about the sky: a day image has a bright and sometimes blue sky, and is generally brighter. Night images also often contain artificial lights, and so they have some small, very bright areas and a mostly dark background. All of these traits and more can help you classify these images!

Distinguishing and Measurable Traits

When you approach a classification challenge, you may ask yourself: how can I tell these images apart? What traits do these images have that differentiate them, and how can I write code to represent their differences? Adding on to that, how can I ignore irrelevant or overly similar parts of these images?

You may have thought about a number of distinguishing features: day images are much brighter, generally, than night images. Night images also have these really bright small spots, so the brightness over the whole image varies a lot more than the day images. There is a lot more of a gray/blue color palette in the day images.

There are lots of measurable traits that distinguish these images, and these measurable traits are referred to as **features**.
A feature a measurable component of an image or object that is, ideally, unique and recognizable under varying conditions - like under varying light or camera angle. And we'll learn more about features soon.

Standardizing and Pre-processing

But we're getting ahead of ourselves! To extract features from any image, we have to pre-process and standardize them!

Next we'll take a look at the standardization steps we should take before we can consistently extract features.

Numerical vs. Categorical

Let's learn a little more about labels. After visualizing the image data, you'll have seen that each image has an attached label: "day" or "night," and these are known as **categorical values**.
Categorical values are typically text values that represent various traits about an image. A couple examples are:

- An "animal" variable with the values: "cat," "tiger," "hippopotamus," and "dog."
- A "color" variable with the values: "red," "green," and "blue."

Each value represents a different category, and most collected data is labeled in this way!

These labels are descriptive for us, but may be inefficient for a classification task. Many machine learning algorithms do not use categorical data; they require that all output be numerical. Numbers are easily compared and stored in memory, and for this reason, we often have to convert categorical values into **numerical labels**. There are two main approaches that you'll come across:

1. Integer encoding
2. One hot-encoding

Integer Encoding

Integer encoding means to assign each category value an integer value. So, day = 1 and night = 0. This is a nice way to separate binary data, and it's what we'll do for our day and night images.

One-hot Encoding

One-hot encoding is often used when there are more than 2 values to separate. A one-hot label is a 1D list that's the length of the number of classes. Say we are looking at the animal variable with the values: "cat," "tiger," "hippopotamus," and "dog." There are 4 classes in this category and so our one-hot labels will be a list of length four. The list will be all 0's and one 1; the 1 indicates which class a certain image is.

For example, since we have four classes (cat, tiger, hippopotamus, and dog), we can make a list in that order: [cat value, tiger value, hippopotamus value, dog value]. In general, order does not matter.

If we have an image and it's one-hot label is `[0, 1, 0, 0]`, what does that indicate? In order of [cat value, tiger value, hippopotamus value, dog value], that label indicates that it's an image of a tiger! Let's do one more example, what about the label `[0, 0, 0, 1]`?

Average Brightness

Here were the steps we took to extract the average brightness of an image.

1. Convert the image to HSV color space (the Value channel is an approximation for brightness)
2. Sum up all the values of the pixels in the Value channel
3. Divide that brightness sum by the area of the image, which is just the width times the height. This gave us one value: the average brightness or the average Value of that image.
   In the next notebook, make sure to look at a variety of day and night images and see if you can think of an average brightness value that will separate the images into their respective classes!

The next step will be to feed this data into a classifier. A classifier might be as simple as a conditional statement that checks if the average brightness is above some threshold, then this image is labeled as 1 (day) and if not, it's labeled as 0 (night).

On your own, you can choose to create more features that help distinguish these images from one another, and we'll soon learn about testing the accuracy of a model like this.

This video cuts off a bit early, but all the better for you to test your intuition and jump into coding a threshold of your own creation in the next notebook!

Classification Task

Let's now complete our day and night classifier. After we extracted the average brightness value, we want to turn this feature into a `predicted_label` that classifies the image.
Remember, we want to generate a numerical label, and again, since we have a binary dataset, I'll create a label that is a 1 if an image is predicted to be day and a 0 for images predicted to be night.
I can create a complete classifier by writing a function that takes in an image, extracts the brightness feature, and then checks if the average brightness is above some threshold X.

If it is, this classifier returns a 1 (day), and if it's not, this classifier returns a 0 (night)!
Next, you'll take a look at this notebook and get a chance to tweak the threshold parameter. Then, when you're able to generate predicted labels, you can compare them to the true labels, and check the accuracy of our model!

Accuracy

The accuracy of a classification model is found by comparing predicted and true labels. For any given image, if the `predicted_label` matches the `true_label`, then this is a correctly classified image, if not, it is misclassified.
The accuracy is given by the number of correctly classified images divided by the total number of images. We'll test this classification model on new images, this is called a test set of data.

Test Data

Test data is previously unseen image data. The data you *have* seen, and that you used to help build a classifier is called training data, which we've been referring to. The idea in creating these two sets is to have one set that you can analyze and learn from (training), and one that you can get a sense of how your classifier might work in a real-world, general scenario. You could imagine going through each image in the training set and creating a classifier that can classify all of these training images correctly, but, you actually want to build a classifier that **recognizes general patterns in data**, so that when it is faced with a real-world scenario, it will still work!
So, we use a new, test set of data to see how a classification model might work in the real-world and to determine the accuracy of the model.

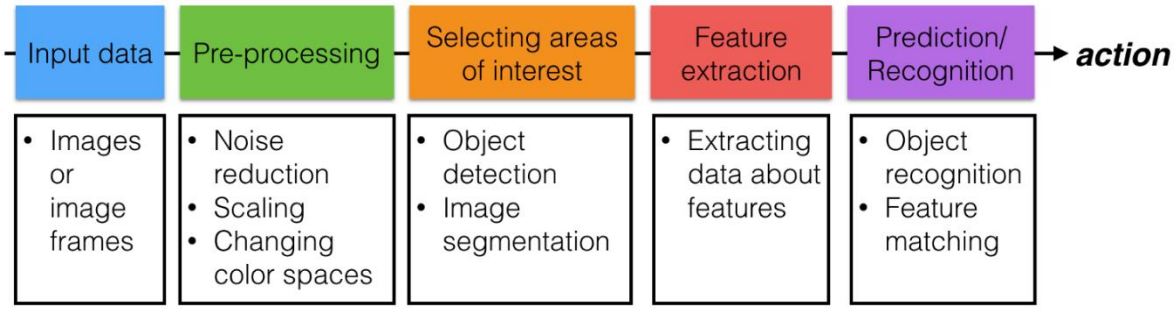
Misclassified Images

In this and most classification examples, there are a few misclassified images in the test set. To see how to improve, it's useful to take a look at these misclassified images; look at what they were mistakenly labeled as and where your model fails. It will be up to you to look at these images and think about how to improve the classification model!

Review and the Computer Vision Pipeline

In this lesson, you've really made it through a lot of material, from learning how images are represented to programming an image classifier!

You approached the classification challenge by completing each step of the **Computer Vision Pipeline** ste-by-step. First by looking at the classification problem, visualizing the image data you were working with, and planning out a complete approach to a solution.
The steps include **pre-processing** images so that they could be further analyzed in the same way, this included changing color spaces. Then we moved on to **feature extraction**, in which you decided on distinguishing traits in each class of image, and tried to isolate those features! You may note that skipped the pipeline step of "Selecting Areas of Interest," and this is because we focused on classifying an image as a whole and did not need break it up into different segments, but we'll see where this step can be useful later in this course.
Finally, you created a complete **classifier** that output a label or a class for a given image, and analyzed your classification model to see its accuracy!



Computer Vision Pipeline.

Now, you're ready to build a more complex classifier, and learn more about feature extraction and deep learning architectures! Good luck and great work!!