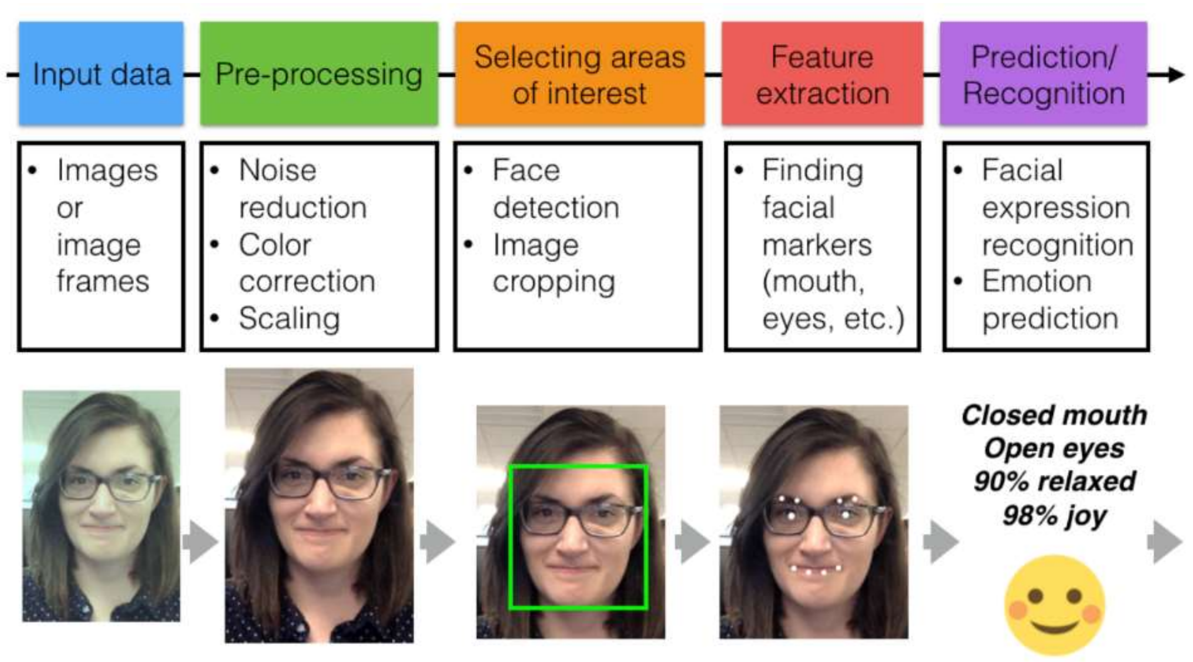
Emotional intelligence :- is the ability to understand and influence human emotion. For example, observing that someone looks sad based on their facial expression, body language, and what you know about them - then acting to comfort them or asking them if they want to talk, etc. For humans, this kind 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 pipline :-

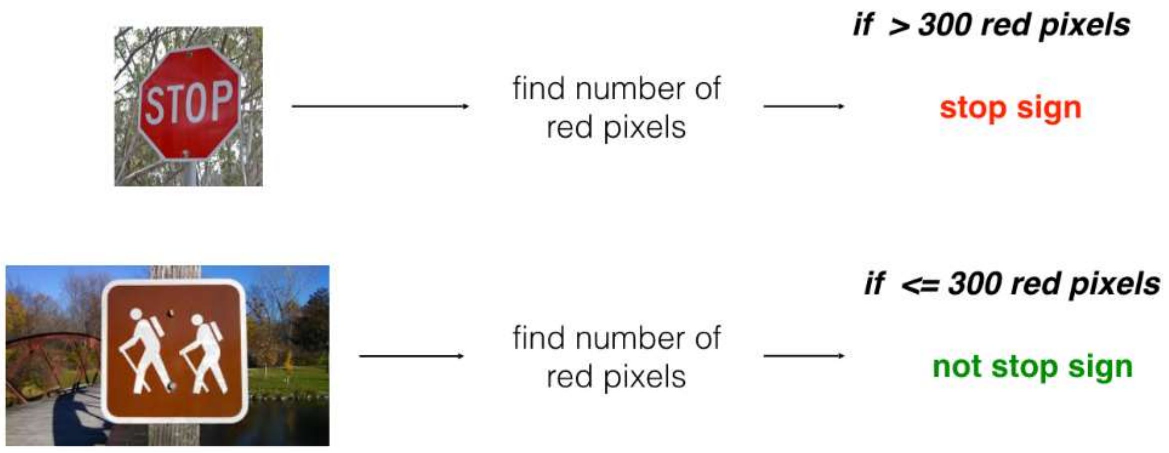
A computer vision pipeline is a series of steps that most computer vision applications will go through. Many vision applications start off by acquiring images and data, then processing that data, performing some analysis and recognition steps, then finally performing an action. The general pipeline and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



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.

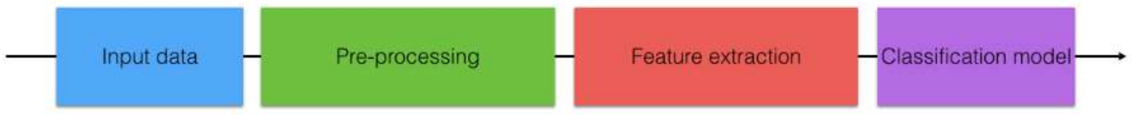
For example, imagine that you've created a simple algorithm to distinguish between stop signs and other traffic lights.

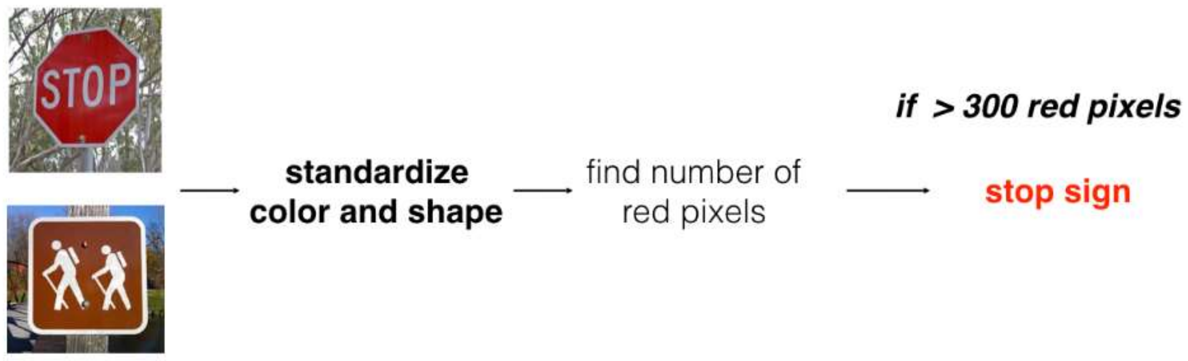


Images of traffic signs; a stop sign is on top and a hiking sign is on the bottom.

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.



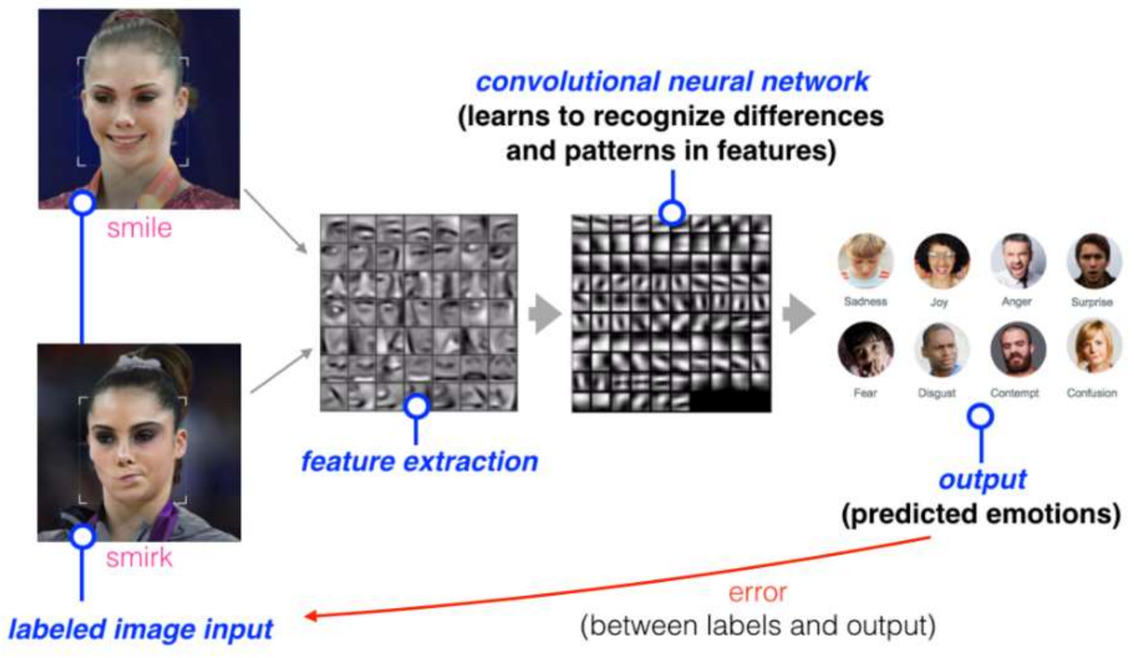


Training a Neural Network

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!



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.

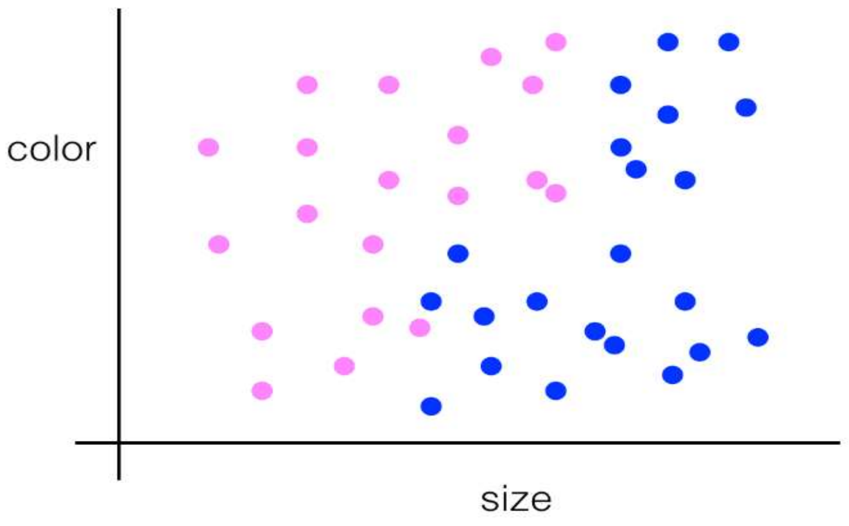
Machine Learning and Neural Networks

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

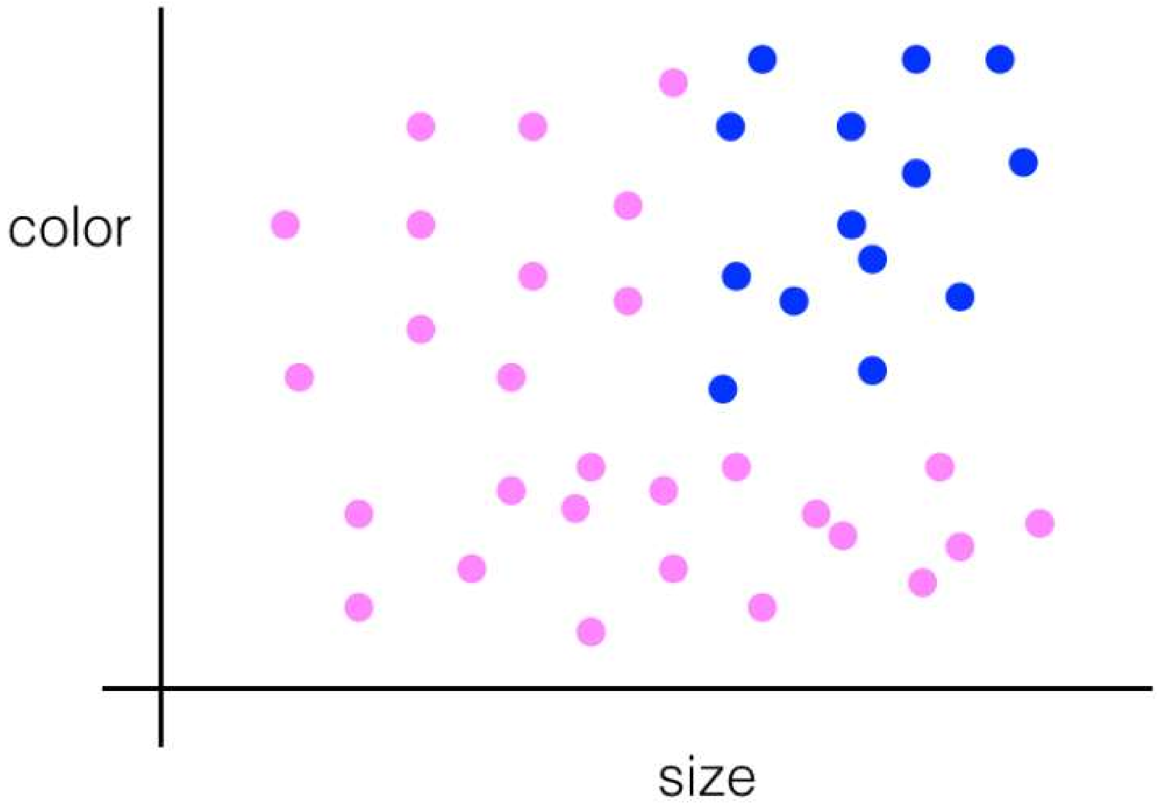
Say you want to separate two types of image data: images of bikes and of cars. You look at the color of each image and the apparent size of the vehicle in it and plot the data on a graph. Given the following points (pink dots are bikes and blue are cars), how would you choose to separate this data?



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.

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.

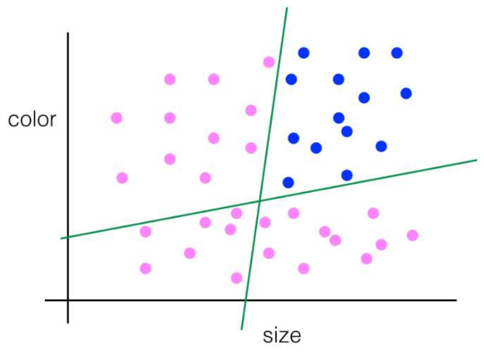
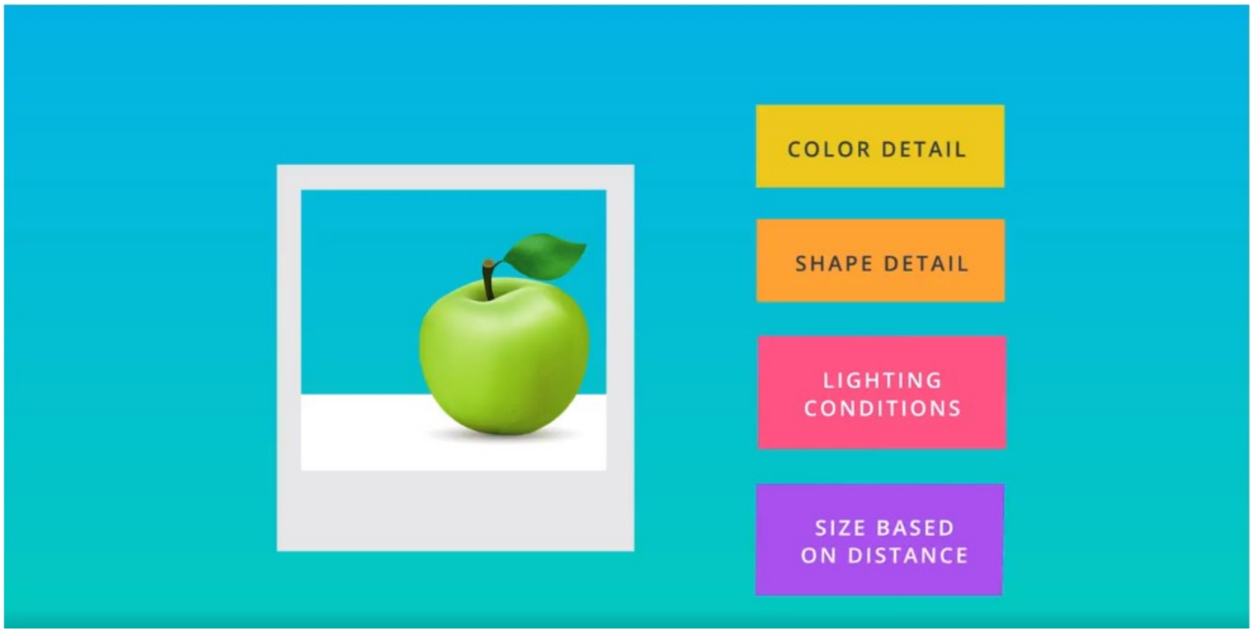


Image Formation



COLOR DETAIL :-

The computer measure and detect the color of the image ‘ object and backgroud ’

SHAPE DETAIL :-

The computer measure the shape of the image and the shape of object

LIGHTING CONDITION :-

The computer measure the light rays

SIZE BASED ON DISTANCE :-

The computer measure distance between object and edge or another object

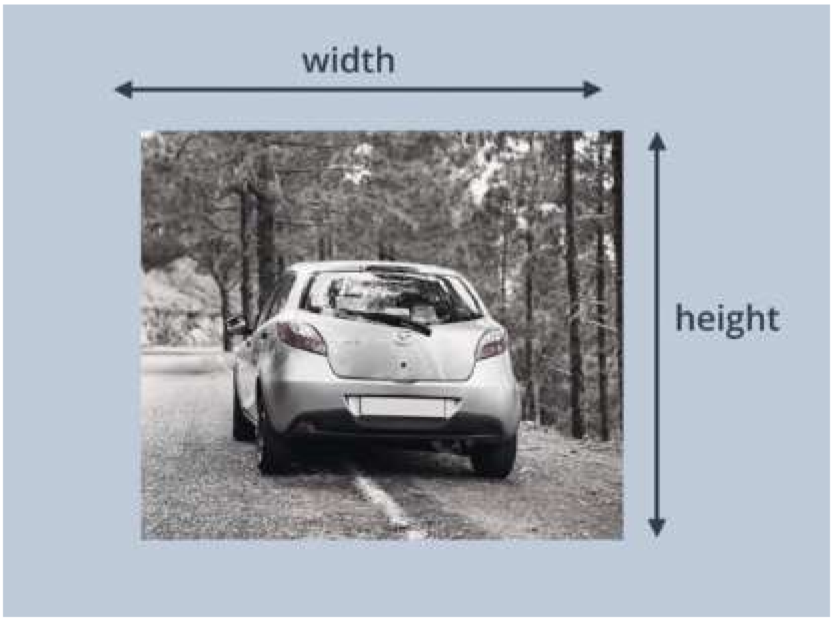
Images as Numerical Data

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.

Gray scale is 2d (Width and Height)

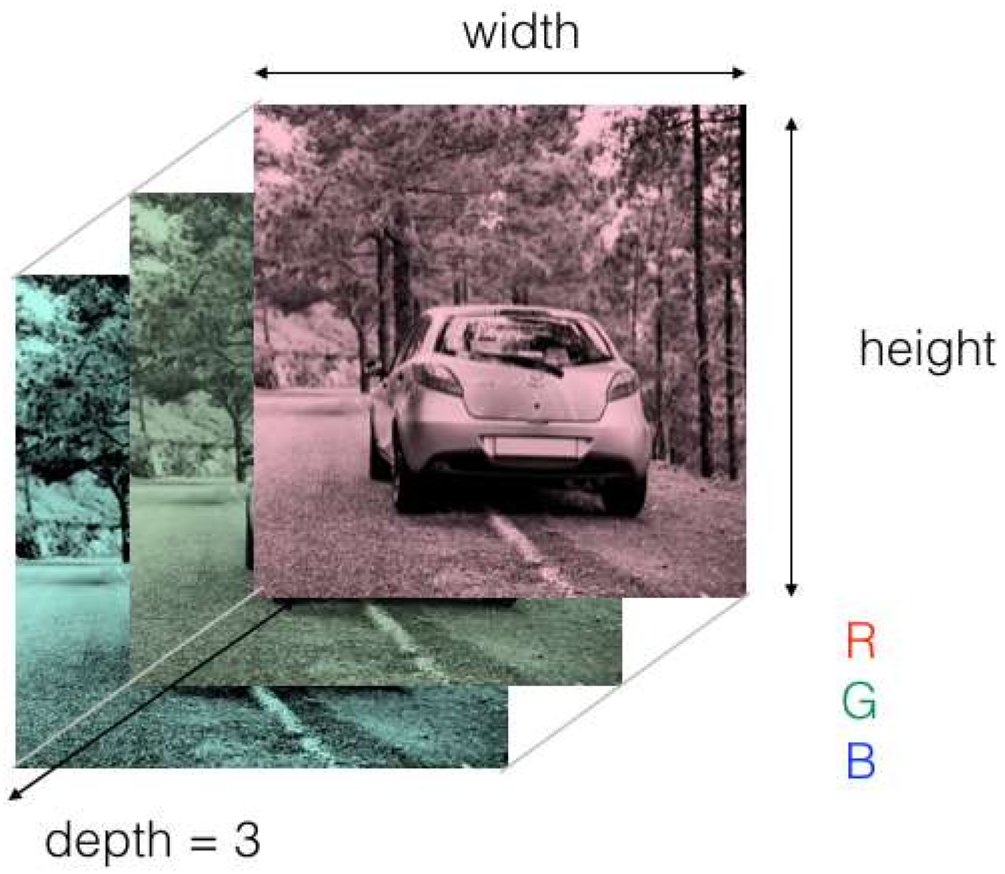


Color Images

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!

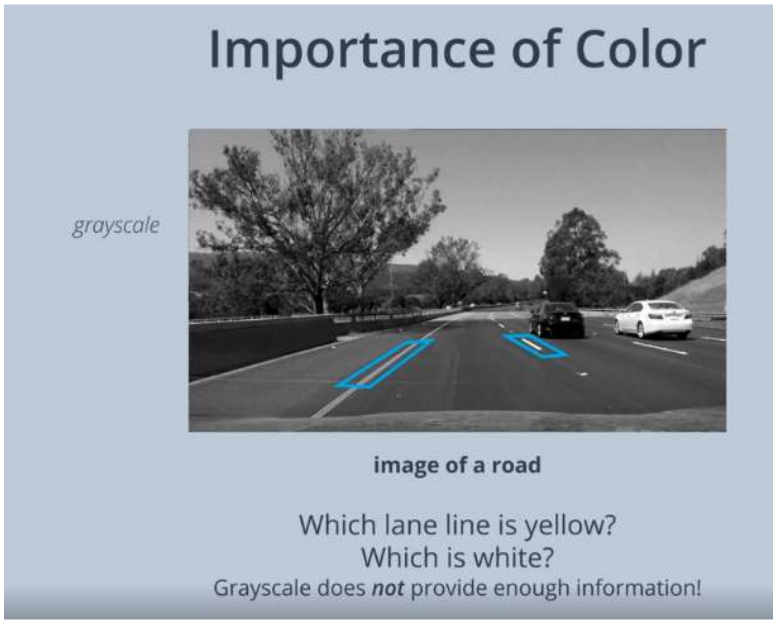

Importance of Color

image of a road

Which lane line is yellow?
Which is white?
Grayscale does *not* provide enough information!

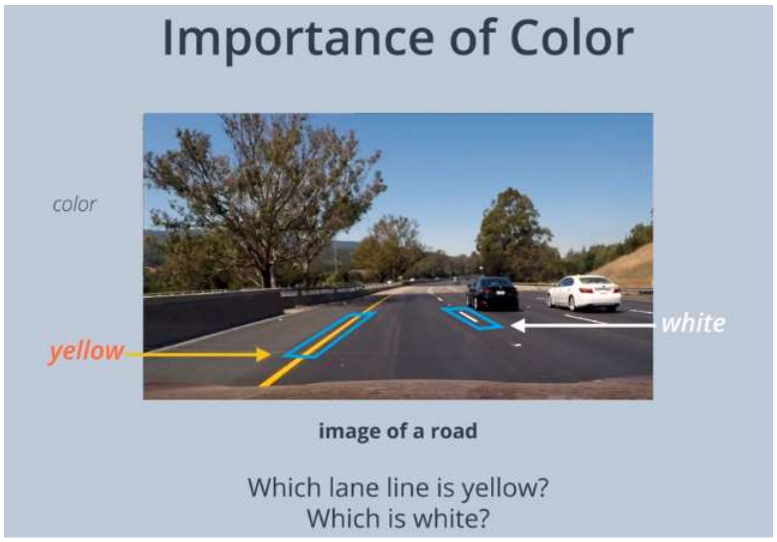

Importance of Color

image of a road

Which lane line is yellow?
Which is white?

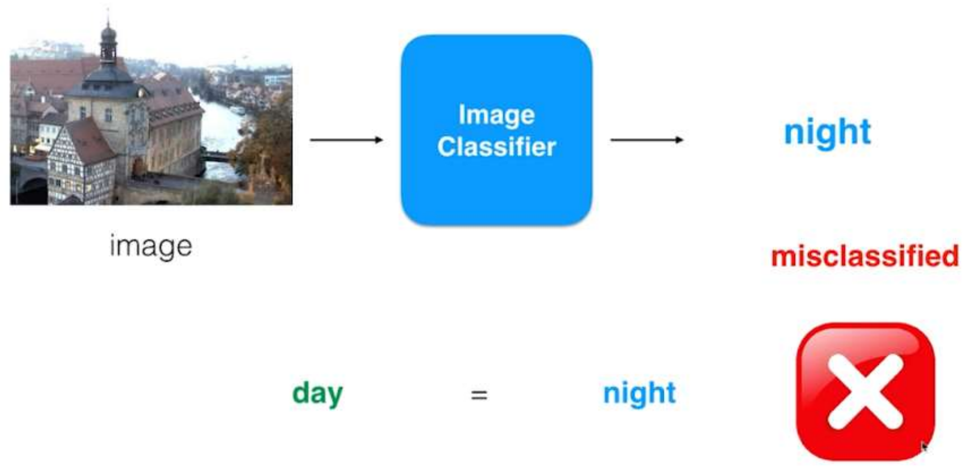
Why do we need labels?

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.

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 repent 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; 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.