



Computer Vision: Crash Course Computer Science #35

Crash Course: Computer Science

<https://youtube.com/watch?v=-4E2-0sxVUM>

<https://nerdfighteria.info/v/-4E2-0sxVUM>

Hi, I'm Carrie Anne and welcome to Crash Course Computer Science.

Today, let's start by thinking about how important vision can be. Most people rely on it to prepare food, walk around obstacles, read street signs, watch videos like this, and do hundreds of other tasks.

Vision is the highest bandwidth sense and it provides a fire hose of information about the state of the world and how to act on it. For this reason, computer sciences have been trying to give computers vision for half a century, birthing the sub-field of computer vision.

It's goal is to give computers the ability to extract high level understanding from digital images and videos. And as everyone with a digital camera or smartphone knows, computers are already really good at capturing photos with incredible fidelity and detail. Much better than humans, in fact.

But as computer vision professor Fei-Fei Li recently said, "Just like to hear is not to listen, to take pictures is not the same as to see."

===== Intro (0:52-1:00) =====

As a refresher, images on computers are most often sorted as big grids of pixels. Each pixel is defined as a color, stored as a combination of three additive primary colors: red, green, and blue. By combining different intensities of these three colors, we can represent any color, which are called RGB values.

Perhaps the simplest computer vision algorithm and a good place to start is to track a colored object like a bright pink ball. The first thing we need to do is record the ball's color. For that, we'll take the RGB value of the center-most pixel.

With that value saved, we can give a computer program an image and ask it to find the pixel with the closest color match. An algorithm like this might start in the upper right corner and check each pixel one at a time, calculating the difference from our target color. Now, having looked at every pixel, the best match is very likely a pixel from our ball.

And we're not limited to running this algorithm on a single photo. We can run it on every frame in a video, allowing us to track the ball over time.

Of course, due to variations in lighting, shadows, and other effects, the ball on the field is almost certainly not going to be the exact same RGB value as our target color, but merely the closest match.

In more extreme cases like at a game at night, the tracking might be poor. And if one of the team's jerseys use the same color as the ball, our algorithm might get totally confused. For these reasons, color marker tracking and similar algorithms are rarely used unless the environment can be tightly controlled.

This color tracking example was able to search pixel by pixel because colors are stored inside of single pixels. But this approach doesn't work for features larger than a single pixel like edges of objects, which are inherently made up of many pixels.

To identify these types of features in images, computer vision algorithms have to consider small regions of pixels called patches. As an example, let's talk about an algorithm that finds vertical edges in a scene. Let's say to help a drone navigate safely through a field of obstacles.

To keep things simple, we're going to convert our image into grayscale, although most algorithms can handle color. Now let's

zoom into one of these poles to see what an edge looks like up close.

We can easily see where the left edge of the pole starts because there's a change in color that persists across many pixels vertically. We can define this behavior more formally by creating a rule that says the likelihood of a pixel being a vertical edge is the magnitude of the difference in color between some pixels to its left and some pixels to its right.

The bigger the color difference between these two sets of pixels, the more likely the pixel is on an edge. If the color difference is small, it's probably not an edge at all. The mathematical equation for this operation looks like this.

It's called a kernel or filter. It contains the values for a pixel-wide multiplication, the sum of which is saved into the center pixel.

Let's see how this works for our example pixel. I've gone ahead and labeled all of the pixels with their grayscale values. Now we take our kernel and center it over our pixel of interest. This specifies what each pixel value underneath should be multiplied by.

Then we just add up all those numbers. In this example, that gives us 147. That becomes our new pixel value. This operation of applying a kernel to a patch of pixels is called a convolution.

Now let's apply our kernel to another pixel. In this case, the result is 1. Just 1. In other words, it's a very small color difference and not an edge.

If we apply our kernel to every pixel in the photo, the result looks like this, with the highest pixel values are where there are strong vertical edges. Note that strong horizontal edges like those platforms in the background are almost invisible.

If we wanted to highlight those features, we'd have to use a different kernel, one that's sensitive to horizontal edges.

Both of these edge-enhancing kernels are called Prewitt Operators, named after their inventor. These are just two examples of a huge variety of kernels able to perform many different image transformations.

For example, here's a kernel that sharpens images. And here's a kernel that blurs them. Kernels can also be used like little image cookie-cutters that match only certain shapes. So our edge kernels looked for image patches with strong differences from right to left or up and down.

But we could also make kernels that are good at finding lines with edge on both sides. And even islands of pixels surrounded by contrasting colors. These types of kernels can begin to characterize simple shapes.

For example, on faces the bridge of the nose tends to be brighter than the sides of the nose, resulting in higher values for line sensitive kernels.

Eyes are also distinctive, a dark circle surrounded by lighter pixels. A pattern other kernels are sensitive to.

When a computer scans through an image, most often by sliding around a search window, it can look for combinations of features indicative of a human face. Although each kernel is a weak face detector by itself, combined they can be quite accurate.

It's unlikely that a bunch of face-like features will cluster together if they're not a face. This was the basis of an early and influential



Computer Vision: Crash Course Computer Science #35

Crash Course: Computer Science

<https://youtube.com/watch?v=-4E2-0sxVUM>

<https://nerdfighteria.info/v/-4E2-0sxVUM>

algorithm called Viola-Jones Face Detection.

Today, the hot new algorithms on the block are Convolutional Neural Networks. We talked about neural nets last episode if you need a primer.

In short, an artificial neuron, which is the building block of a neural network, takes a series of inputs and multiplies each by a specified weight and then sums those values all together.

This should sound vaguely familiar, because it's a lot like a convolution. In fact, if we pass a neuron 2D pixel data, rather than a one-dimensional list of inputs, it's exactly like a convolution.

The input weights are equivalent to kernel values, but unlike a predefined kernel, neural networks can learn their own useful kernels that are able to recognize interesting features in images.

Convolutional neural networks use banks of those neurons to process image data, each outputting a new image, essentially digested by different learned kernels.

These outputs are then processed by subsequent layers of neurons, allowing for convolutions on convolutions. The very first convolutional layer might find things like edges. And that's what a single convolution can recognize, as we've already discussed.

The next layer might have neurons that convolve on those edge features to recognize simple shapes comprised of edges like corners. A layer beyond that might convolve on those corner features and contain neurons that can recognize simple objects like mouths and eyebrows.

And this keeps going, building up in complexity until there's a layer that does a convolution that puts it together; eyes, ears, mouth, nose, the whole nine yards and says, "Aha! It's a face!"

Convolutional neural networks aren't required to be many layers deep, ut they usually are, in order to recognize complex objects and scenes. That's why the technique is considered deep learning.

Both Viola-Jones and convolutional neural networks can be applied to many image recognition problems. Like recognizing handwritten text, spotting tumors in CT scans, and monitoring traffic flow on roads.

But we're going to stick with faces. Regardless of what algorithm was used, once we've isolated a face in a photo, we can apply more specialized computer vision algorithms to pinpoint facial landmarks like the tip of the nose and corners of the mouth.

This data can be used for determining things like if the eyes are open, which is pretty easy once you have the landmarks. It's the distance between points.

We can also track the position of the eyebrows. Their relative position to teh eyes can be an indicator of surprise or delight. Smiles are also pretty straightforward to detect based on the shape of mouth landmarks.

All of this information can be interpreted by motion recognition algorithms, giving computers the ability to infer when you're happy, sad, frustrated, confused, and so on.

In turn, that could allow computers to intelligently adapt their behavior. Maybe offer tips when you're confused and not ask to restore updates when you're frustrated. This is just one example of how vision can give your computers the ability to be context

sensitive.

That is, aware of their surroundings. And not just your physical surroundings like if you're at work or on a train, but also your social surroundings like if you're in a formal business meeting versus a friend's birthday party.

You behave differently in those surroundings and so should computing devices if they're smart.

Facial landmarks also capture the geometry of your face like the distance between your eyes and the height of your forehead. This is one form of biometric data and it allows computers with cameras to recognize you.

Whether it's your smartphone automatically unlocking itself when it sees you or governments tracking people using CCTV cameras, the applications of facial recognition seem limitless.

There have also been recent breakthroughs in landmark tracking for hands and whole bodies, giving computers the ability to interpret a user's body language. And what hand gestures they're frantically waving at their internet connected microwave.

As we've talked about many times in this series, abstraction is the key to building complex systems. And the same is true in computer vision. At the hardware level, you have engineers building better and better cameras, giving computers improved sight with each passing year.

Which I can't say for myself.

Using that camera data, you have computer vision algorithms crunching pixels to find things like faces and hands. And then using output from those algorithms, you have even more specialized algorithms for interpreting things like user-faical expression and hand gestures.

On top of that, there are people building novel interactive experiences like smart TVs and intelligent tutoring systems that respond to hand gestures and emotion.

Each of these levels are active areas of research with breakthroughs happening every year. And that's just the tip of the iceberg!

Today, computer vision is everywhere. Whether it's barcodes being scanned at stores, self-driving cars waiting at red lights, or Snapchat filters superimposing mustaches.

And the most exciting thing is that computer sciences are just getting started, enabled by recent advances in computing like super fast GPUs. Computers with human-like ability to see are going to totally change how we interact with them.

Of course, it would also be nice if they could hear and speak, which we'll discuss next week. I'll see you then!

===== Credits (10:16) =====

Hey guys! Before I go, I wanted to tell you about a great new show from PBS Digital Studios: the Origin of Everything. In this series, host Danielle Bainbridge investigates the history of well, everything.

From the hashtag and US healthcare to Godzill and even killer clowns. It's a show about how the complex intersections with our



Computer Vision: Crash Course Computer Science #35

Crash Course: Computer Science

<https://youtube.com/watch?v=-4E2-0sxVUM>

<https://nerdfighteria.info/v/-4E2-0sxVUM>

past and informed the world we know today and I think you'll love it.

Crash Course Computer Science is produced in association with PBS Digital Studios. At their channel, you can check out a playlist of shows like PhysicsGirl, Deep Look, and PBS Spacetime.

This episode was filmed at the Chad and Stacey Emigholtz Studio in Indianapolis. And it was made with the help of all these nice people and our wonderful graphics team, Thought Cafe.

Thanks for watching and try turning it off and then back on again!