CrowdFlower

WHAT WE LEARNED LABELING 1 MILLION IMAGES

A practical guide to image annotation for computer vision

WHAT WE LEARNED LABELING 1 MILLION IMAGES

A practical guide to image annotation for computer vision

INTRODUCTION

On July 7th, 1966, a professor named Seymour Papert proposed a summer project to the Artificial Intelligence Group at MIT. Using a handful of undergrads as research assistants, Papert was confident he and his team could construct "a significant part of a visual system" over the course of a single summer. In fact, Papert titled this whole endeavor "The Summer Vision Project." And while it's a bit difficult to pinpoint what his team actually accomplished, 50 years later, we can safely say that a summer in 1966 probably wasn't quite enough time to solve computer vision for good.

We've come a long way since the summer of ‘66. Facebook's clumsily named DeepFace system achieved human-level accuracy as early as 2014. In that same year, machines were reading radiology images correctly 95% of the time. Pinterest and eBay have both rolled out AI that allows you to shop for products from snapshots. Computer vision algorithms are looking at everything from satellite photographs to microscopic biopsy images and they're getting smarter and more accurate by the day.

Now, before we dig in too far, it’s a good idea to define what we mean when we talk about computer vision. Put simply, the goal of computer vision is to have machines see and understand images and videos. Often, from a business process perspective, computer vision is concerned with automating tasks that humans can do; for example, understanding radiology images so that doctors don’t spend all their time analyzing scans or seeing the road so that our car can drive for us (or, at least, keep us from crashing).

And as far as we’ve come in the field over the past few decades, computer vision accuracy still isn't close to human accuracy. That same AI that can confidently score radiology images? It would be completely at sea looking at images of dogs. That's because AIs learn to "see" through labeled training data. To understand that an image is a dog, AIs need to have seen tens of thousands of images of dogs from all different angles, in all different poses. And even then, they may have problems with oddly colored dogs, drawings of dogs, or occluded pictures where it can only see a pair of eyes or a tail.

This is true for nearly every use case. An AI that can accurately score radiology images can only do so because it's learned from labeled examples that show it what to look for. The same goes for autonomous drones, AIs that can predict deforestation from aerial imagery, penguin-counting algorithms, you name it.

And while this means that a general AI capable of looking at and identifying any object in any image would take a preposterous amount of high-quality training data, it also highlights the fact that, no matter what your algorithm is supposed to "see," the steps to building a well-trained, accurate image classifier are essentially the same.

At CrowdFlower, we've seen and helped with tons of these projects. We've labeled over a million images, helping some of the world's most innovative companies power and validate their computer vision models. In this guide, we'll share a bit of what we've learned along the way. You'll learn how to scope a computer vision project, what kind of source data you need to make it successful, what kind of tools fit your project best, how to label your dataset so your algorithms can learn, and a whole lot more.

SCOPING YOUR PROJECT

If you're doing a computer vision project, the first thing you need to do is decide what your goal actually is. This might seem trivial but it's actually incredibly important to do this right. You want to be as exacting as possible here, thinking both granularly and at a big picture level. Here's an example:

Say your company is working on a self-driving car. If you're working on the AI that will allow the car to drive autonomously, you need to define what that means. Do you expect your car to:

- **Park itself?**
- **Drive itself on the freeway?**
- **How about the city?**
- **What happens in inclement weather?**
- **Drive on the left or the right?**

The answers to all these questions have serious implications for the sort of data you're going to need to train your AI.

As for how much data you'll need? There really isn't a hard-and-fast rule. It's heavily dependent on how complex your problem is, how accurate you need to be, and what you're actually doing with your model. Many projects necessitate an ontology of some kind, so you'll want to plan that out too (as well as be willing to amend and refine it as necessary). Up front, you may create an ontology for a retail vision project that contains only "dress" but then expands to include types and styles of dresses as you analyze your data and your model's performance, for example.

You should also keep in mind what happens when your models are wrong (or just not confident). What's the cost to their inaccuracy? It's probably obvious, but if you're building an algorithm that allows cameras to see inventory on grocery store shelves, your problem is a lot simpler than an autonomous vehicle model. Your surroundings will be more uniform, so you'll likely need much less training data, and, importantly, your penalty for inaccuracy isn't that big of a deal. If your algorithm incorrectly thinks there's no cereal on a shelf, that store ends up with extra Rice Krispies. If a self-driving car makes an error, lives are at stake.

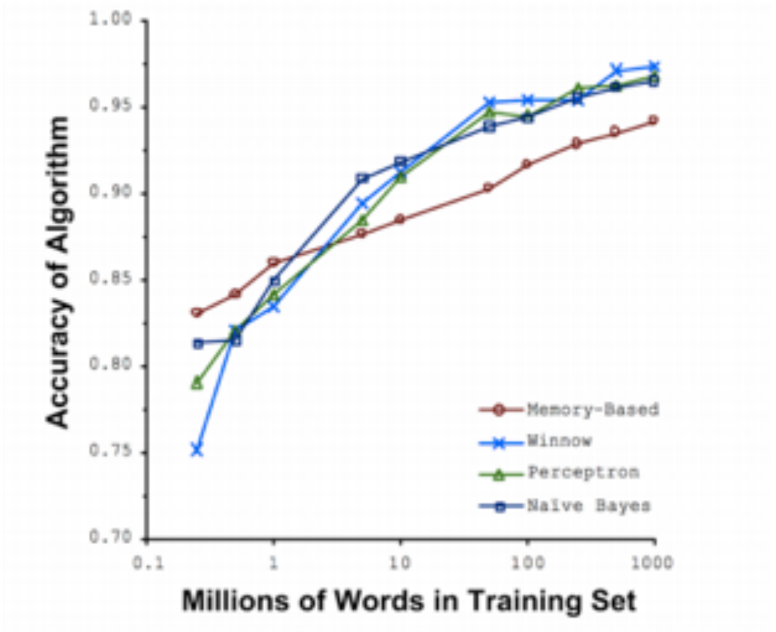
But back to how much data you need: you're not training a good computer vision algorithm on hundreds of images. You're going to need tens or hundreds of thousands of images per category. And even that might be underselling it.

In fact, remember what we said in the intro about Facebook's success with facial recognition? The reason they've enjoyed so much accuracy is because they trained DeepFace on a gigantic, well-labeled image set: user images. Facebook has access to hundreds of billions of photos–labeled by users themselves–with 350 million added daily. That's a lot of training data.

As for where it comes from? Generally, it's best practice to use your own, proprietary data. Open source data is usually not robust enough and, once you've run into edge cases or realize you'll need more of that data, you may find yourself at an impasse. That said, for some applications, something like the CityScape dataset, Microsoft's Coco dataset, ImageNet, or any other robust datasets can be a good place to start.

You'll want to leverage whatever in-house big data you have that fits the bill, but don't be afraid to be scrappy. Taping a phone to a dashboard and taking video for your self-driving car project can provide you with a host of usable stills for annotation. Scraping sites for product images in retail or buying large datasets from satellite providers can be a good first step too.

We've all heard the phrase "garbage in, garbage out." That's especially true for training machine learning algorithms. A security robot trained on good data knows that a broken window means it should alert the police. A security robot trained on bad data wheels itself into a fountain.



the accuracy got way better as they increased the training data set

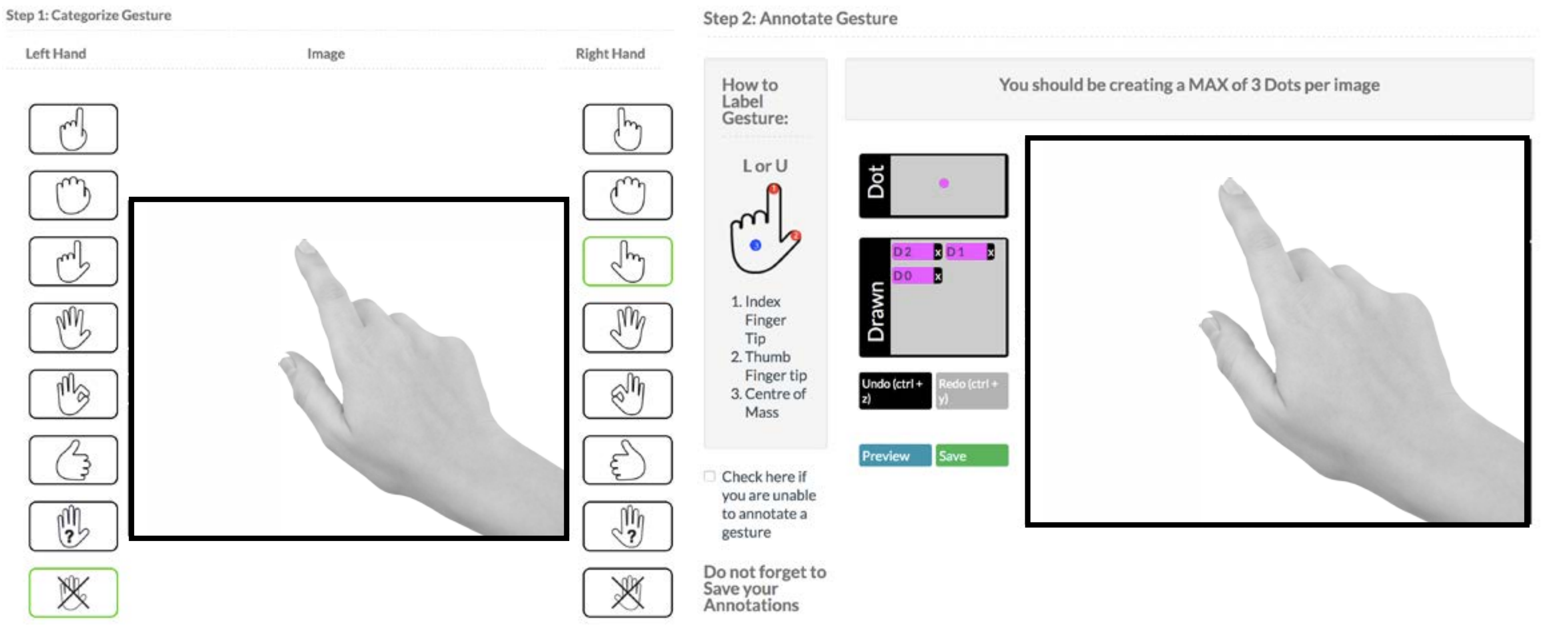
And it's not just quality, it's quantity. Take a look at this graph: What many people see when they look at this chart is that there's a difference between the accuracy in these algorithms. Which, certainly, is one way to look at it. What's at least as important, however, is how they converge as more and more data is added to them. In fact, the difference between state-of-the-art methods and older ones effectively disappears when they've been fed enough information.

In other words, more training data leads to smarter algorithms.

Smaller datasets, like a lot of open source datasets you can get your hands on, are fine for toy use cases or graduate theses or general models that aren't trying for high accuracy. And while there are other reasons computer vision projects fail–unreasonable timelines, fuzzy buy-in from business owners, budgets, changing priorities, and more–most often, it really does boil down to the data you're using. The more you can provide–and the closer it hews to your actual end use case–the better.

But image data–raw files like product images, satellite photos, street snapshots, or anything else–rarely comes labelled. That's where we come in. You choose what you parts of these images you need annotated; our platform leverages a network of human labelers to get the work done.

Put another way: images without labels won't get you anywhere in your computer vision project. You'll just have raw data with pixel values. You'll know where some colors are, essentially, but that's it. Labeled images train algorithms to know the difference between a mother pushing a stroller and a fire hydrant, how to tell if a person's smiling, how to spot a new logging road carving through the rainforest. To label images accurately, you need good technology and good people. We've got both.



You should be creating a MAX of 3 Dots per image

1. Index Finger Tip
2. Thumb Finger tip
3. Centre of Mass

Check here if you are unable to annotate a gesture

Do not forget to Save your Annotations

OKAY, SO TRAINING DATA MATTERS. NOW WHAT?

Let's get back to your project. Once you've taken a first pass at scoping, it's time to align the goals of your project with right kind of image processing tasks. Roughly, these break into three separate categories:

- **Classification**
- **Shape annotation**
- **Pixel labeling**

They all have their pros and cons and each excel for different use cases. Again, what matters most here is which tasks and workflow make sense for the algorithm or classifier you're trying to build. You may need to run the same source data through different annotation tasks to find the most accurate model(s). You may need to use several different kinds of annotation tools for the same overall project or employ different algorithms for specific uses within your project.

Take classification tasks, for example. They can be used to identify which images need annotation and then again to validate your algorithm's performance. In fact, let's start there.

IMAGE CLASSIFICATION

Image classification is an extremely common first step for all sorts of computer vision projects. It helps you parcel out work correctly, teaches you what's actually in your source data, and sets you up for higher accuracy and speed when you get to actually marking up the images themselves.

So what is image classification? It's a fairly simple process where you have human labelers mark whether a picture contains a certain object (or objects) in your ontology. And it's probably easiest to explain with an example.

Say you're trying to build a self-driving car algorithm. That means your source data is almost certainly street view pictures like dashcam stills of highways and crowded intersections. You might be tempted to start annotating those images straight away, but that's actually not the best idea. Let us explain why.

Take those two kinds of images we mentioned above: a city intersection and a stretch of highway. While a fully autonomous car will have to deal with both, the sorts of annotations you'll eventually be doing are wildly different. City images have all sorts of objects and a lot more going on than highway images (for example, you shouldn't expect pedestrians on the freeway).

IMAGE CLASSIFICATION

Image classification is the best way to parcel out that data and keep it clean from the get-go. So for our example, all you'd need to do is present labelers with an image and ask them simple classification questions like:

- **Are there homes in this image?**
- **Are there pools in this image?**
- **Are there roads in this image?**

There's a good reason for doing this. When people start labeling images with boxes (for example), they're much more accurate when they're given discrete tasks. Asking a person to label every car in an image is a lot easier than asking them to mark each part of an image based on a taxonomy. They'll work faster and be far more accurate. Images with pedestrians can be sent to labelers who are working just on annotating pedestrians. They won't see those highway images because, hopefully, there aren't any pedestrians there and your labelers can concentrate on a single task.

Classification can also keep your algorithm from overfitting. We'll get into this more later, but basically, if your image set is mostly cheese, your algorithm is going to assume most objects are cheese. Classifying your images allows you to make sure you're building a model that has a more nuanced view of its world and doesn't assume a mailbox is automatically a block of cheddar.
One last thing: your classifications can be as specific as your project requires. Take the "are there pedestrians in this image?" question. For example, you can ask additional questions like "is this pedestrian pushing an object?" or "is this pedestrian on a bike or skateboard?" A mother pushing a stroller behaves much differently from a bicyclist, after all.

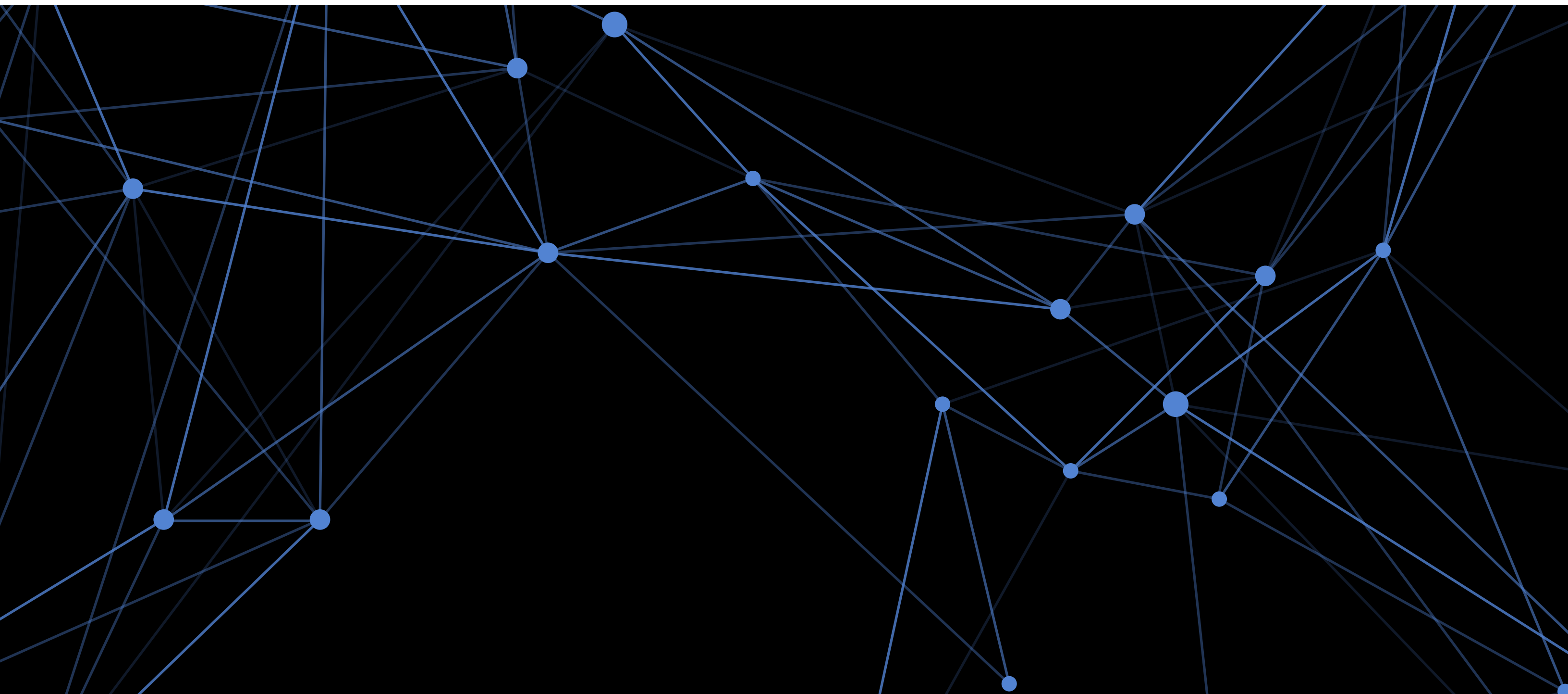
Now, often, we get asked which sort of tool we recommend for a particular project. We've seen a pretty wide variety of annotation jobs and we have a handle on which work well for which use case. But before we dig into that too far, it's important to level-set on what each tool is and how they work. We'll start with shapes, but first, here's a chart to explain some high-level differences:

TOOL COMPARISON CHEAT SHEET

At a glance, here's a quick overview of the tools we'll discuss:

	BOUNDING BOX	DOTS	POLYGON	SEMANTIC SEGMENTATION
COST	Least expensive	More expensive	More expensive	Most expensive
TIME COMMITMENT	Lowest	Medium	Medium	Highest
PRECISION	Good	Great	Great	Excellent
INSTANCE-BASED (OUTPUT CONTAINS DISCRETE OBJECTS)	Yes	Yes	Yes	No
POSSIBLE TO LABEL SEVERAL OBJECTS?	Yes	Yes	Yes	Yes
POSSIBLE TO LABEL SEVERAL CLASSES OF OBJECT IN A SINGLE JOB?	No	No	No	Yes
OUTPUT*	X,Y coordinate, width and length of each box	Series of x,y coordinates	Series of x,y coordinates, with shapes resolving	Coded RGB pixels as an image

* see page 13



SHAPE ANNOTATIONS

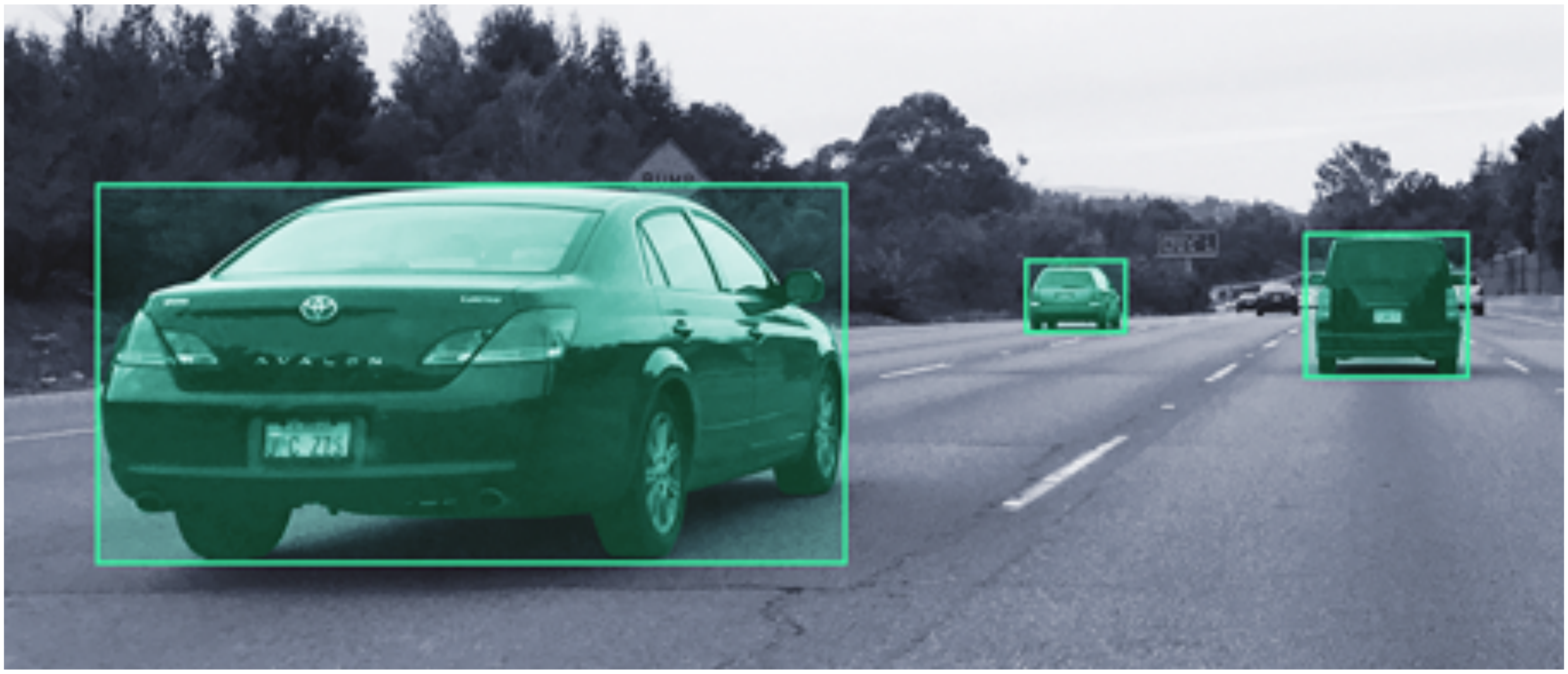
Once you've classified your images and parcelled them out, it's time to get annotating. Again, we recommend creating distinct tasks for labeling. In other words: "Draw a box around every car in this image" or "mark every pedestrian in this image" and so on. You can merge the image data from these separate tasks if an image contains multiple classes your algorithm cares about.

Shape annotation tasks can generally be broken into three main categories:

- **Bounding boxes:**
  Drawing a box around a particular object
- **Dots:**
  Marking an image with a series of dots or points
- **Lines (or polygons):**
  Drawing lines or creating shapes (polygons) with a simple line drawing tool

There are pros and cons to each of these approaches and which one(s) you use, again, depends on what it is you need your model to do. Each approach, however, is attempting to do the same thing at a high level: mark objects in a real image.

Since it's the most common tool, and a great place to start, let's look at bounding boxes.

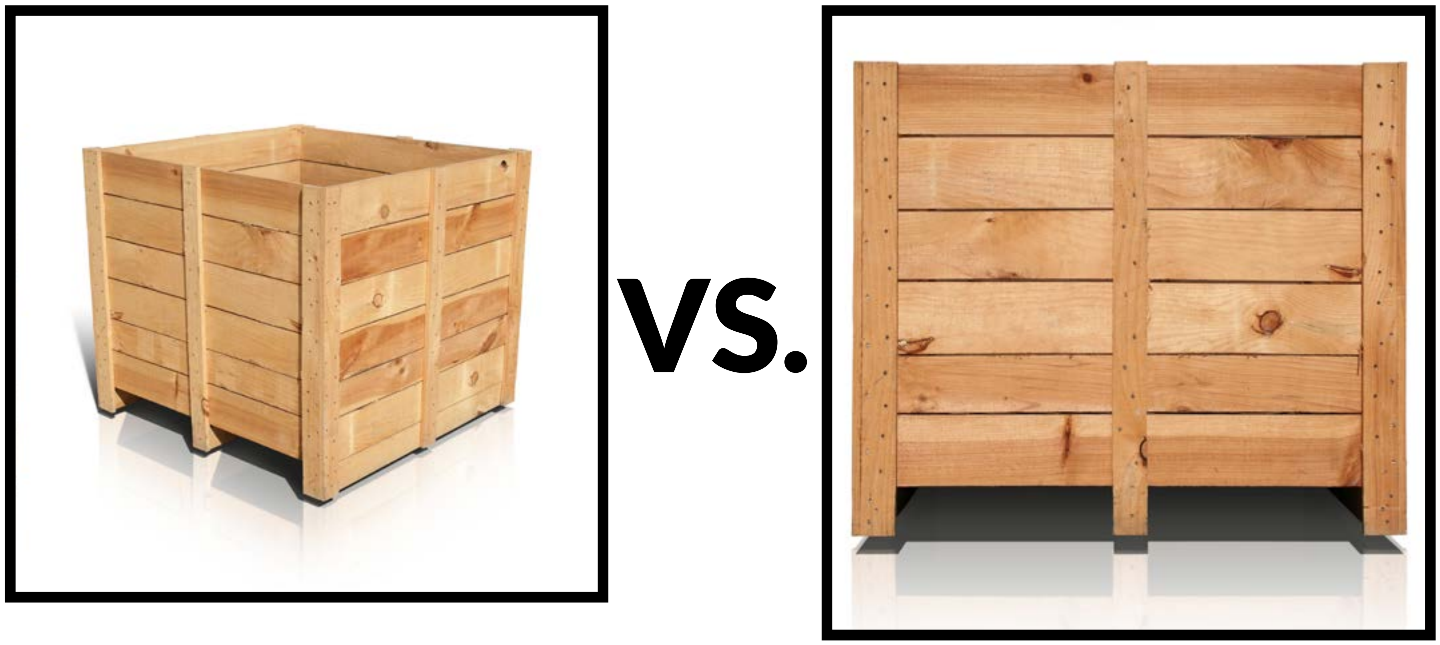


BOUNDING BOXES

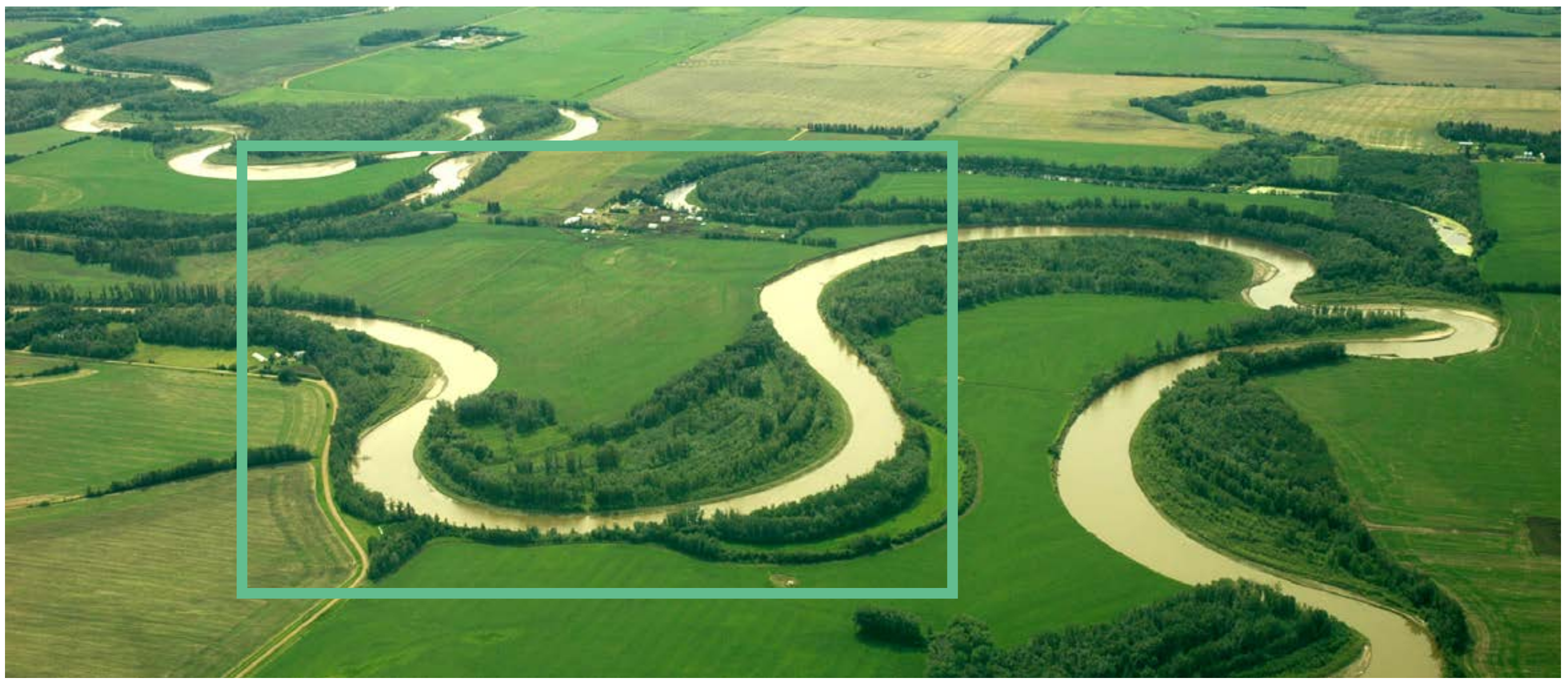
It's probably safe to assume you know what a bounding box on an image looks like, but if not, here's a pretty common example. In the image above, a data scientist has asked a labeler to draw a box around each car.

Bounding boxes are a simple way to capture certain objects in images and are the easiest type of annotation. Depending on your quantity and quality of data, sometimes, a model can learn to identify the objects you need just by training with bounding boxes.

So when should you use bounding boxes? What are some best practices? First off, you'll likely want to use bounding boxes when your objects are, well, boxable. Think of drawing a box around crate, for example. If your image is front-facing, labelers can draw a much tighter box around it than if your image is at, say, a three-quarter view.



By way of example, a product on a shelf can usually be boxed, but, say, a river from an aerial photograph would be a bad candidate.



You'll also want to come up with a strategy for occluded images and this strategy should marry to the goal of your computer vision project. If we return to our self-driving car example, what would you do with a person who's partially obscured behind a parked car or a bus stop bench? Well, you want your model to know that the object is a person, not half a person. Many data scientists would advise boxing the entire individual, even if they're not completely visible. If you were simply building an algorithm that counted people in a particular location, that might not be as necessary.

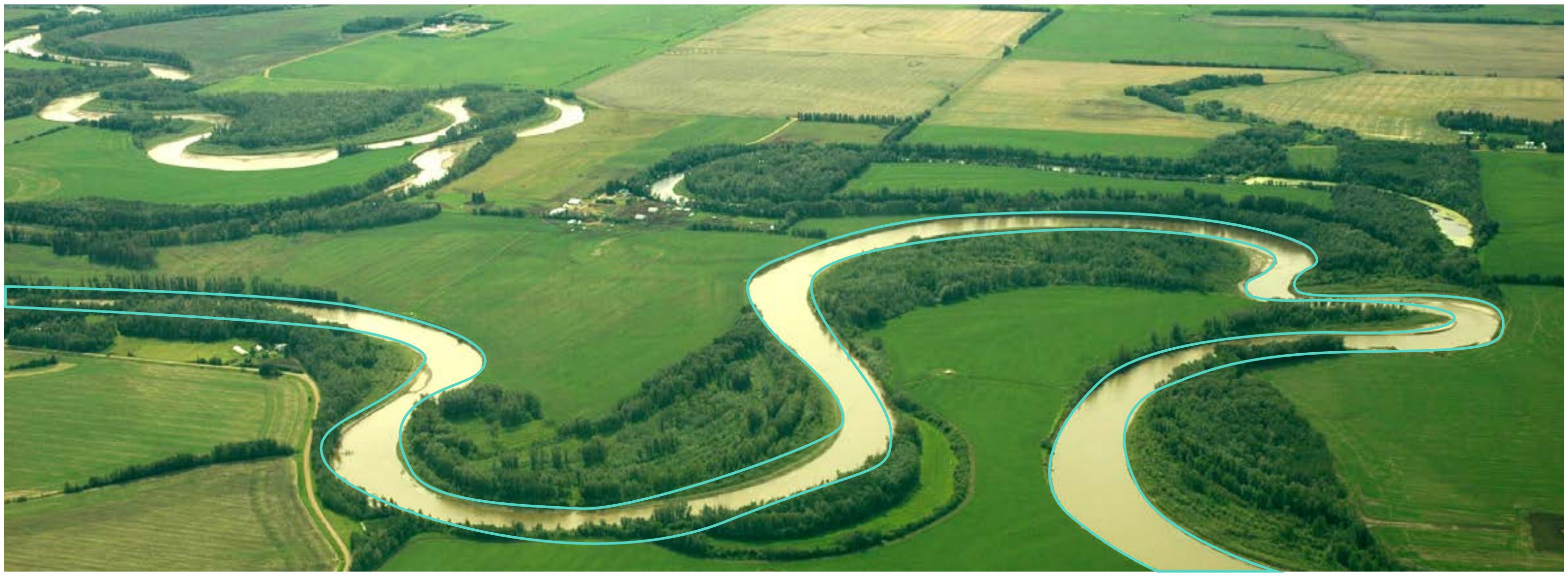
Bounding boxes are the easiest kind of annotation, requiring less attention and less complicated tools than pixel or polygon jobs. By virtue of this, they finish faster and cheaper than other types of image annotation. They're not as precise as the other methods we'll discuss, but they're simple and they work quite well for a lot of applications, including self-driving cars, general object recognition jobs, and multiple retail applications. Lastly, because boxes are less precise than pixels or polygons, you'll likely need a bit more training data to reach the accuracy of those methods. But, again, the labeling goes much quicker and is generally much cheaper, so don't let that fact discourage you.

Next, we're going to look at the bounding box's first cousin: lines and polygons.

LINES AND POLYGONS

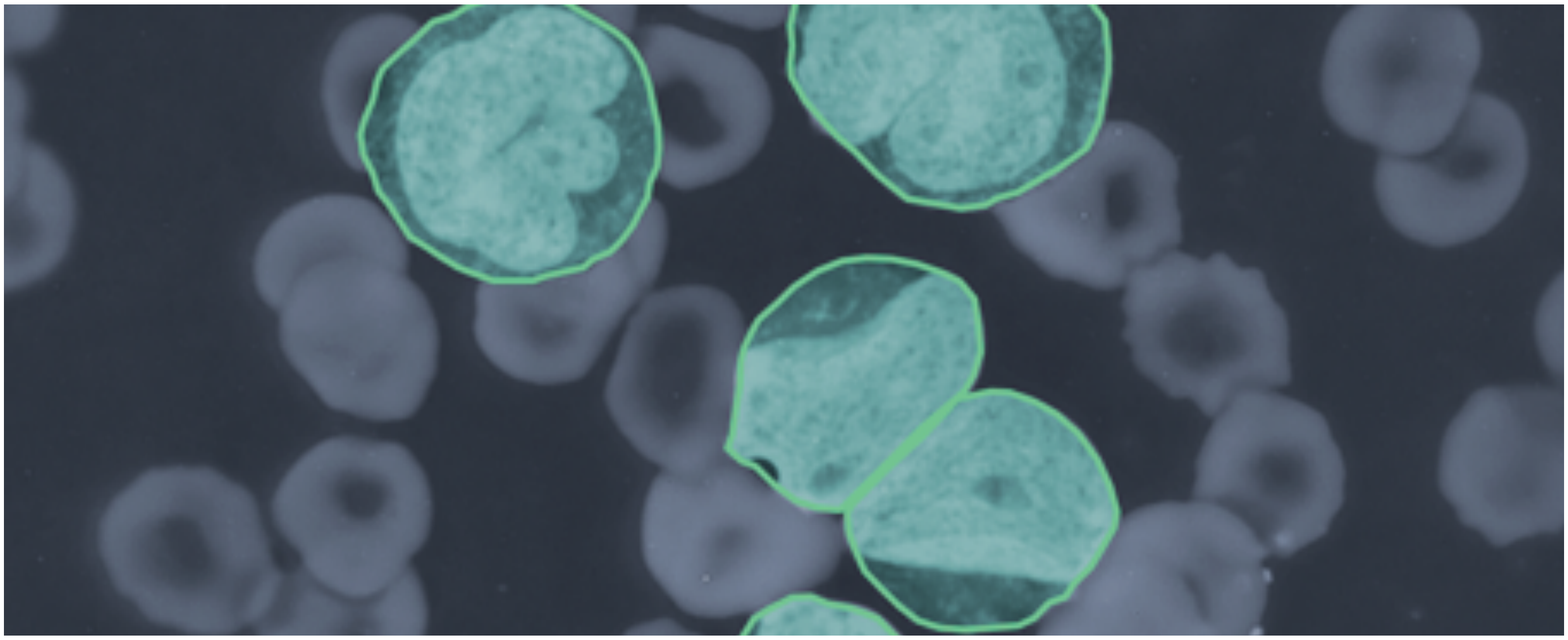
While there are occasional reasons to annotate an image with a single line, a vast majority of so-called "line" jobs involve labelers drawing shapes around objects. In many ways it can be seen as a bridge between boxes and semantic segmentation (a.k.a. pixel labeling).

Line tools (like the one on CrowdFlower) allow labelers to draw tight shapes around the objects you need identified. Unlike boxes which can capture a lot of white space and additional noise, leading to confusion in vision models, polygons are far more precise. Remember the aerial photo from our last section?



This is a perfect candidate for this sort of line/polygon annotation. While a box would capture far more grass than river, a polygon can be far more exacting. Same thing goes for the three-quarters view of a crate we showed above.

Of course, drawing shapes requires more work for labelers, so these types of jobs cost a bit more–both in time and money–than bounding box annotation does. You'll often see them used for everything from aerial imagery to medical research.



Though the output will be a bit different than with bounding boxes (we'll get to that later), the general processes, workflow, and best practices still apply here. Polygons should be as tight as possible and breaking up work with image classification tasks first will absolutely improve both the speed and accuracy of your labeling work.

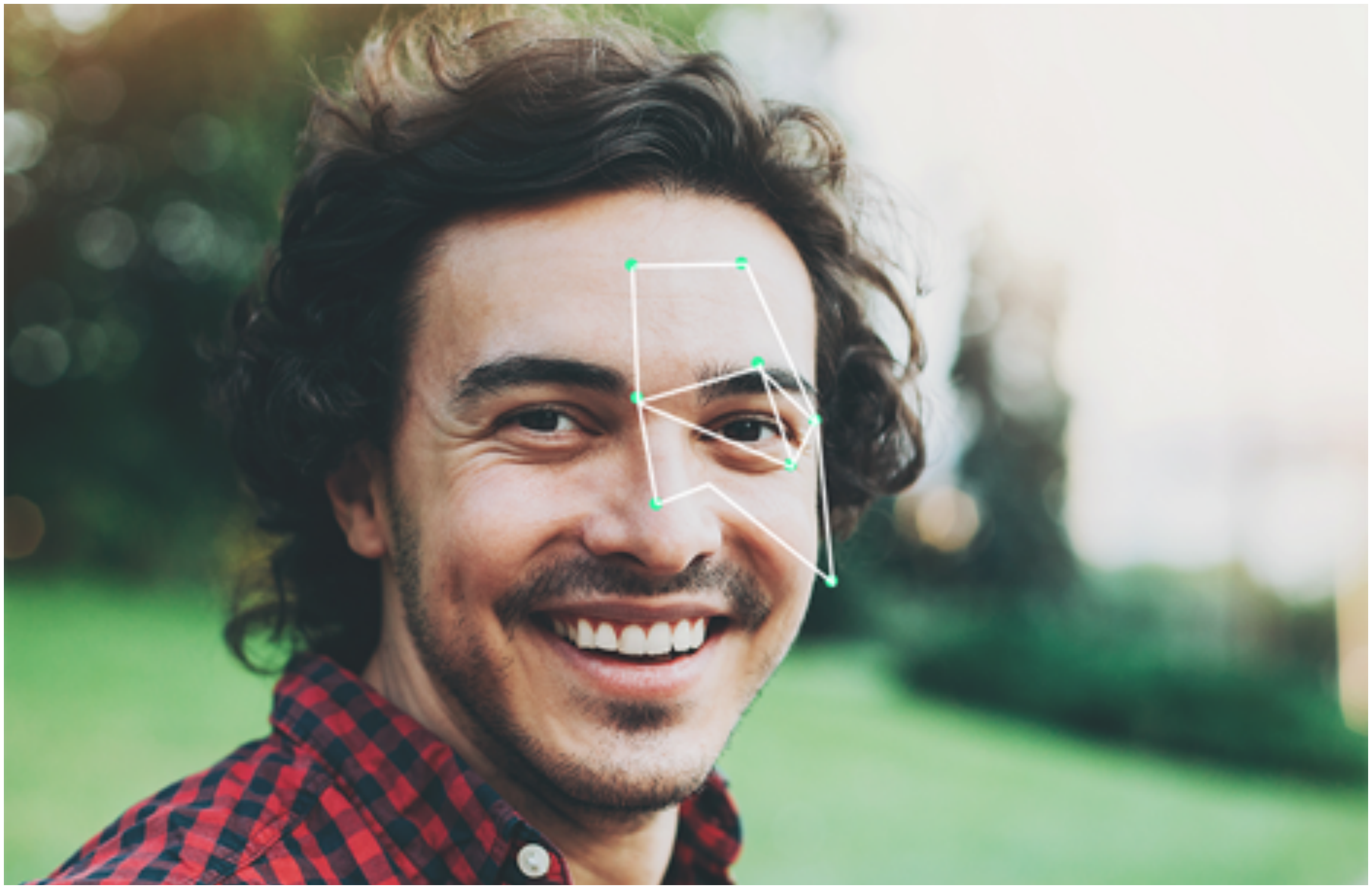
Let's move off shapes into a completely different kind of annotation: dots.

DOT ANNOTATION

Dot annotation is exactly what it sounds like. In these tasks, labelers place dots where you ask them to. Generally, these tools are used most often for counting jobs and gesture or facial recognition tasks.

The counting jobs are pretty self explanatory. Say, for example, you want to count the cars in a parking lot in a large mall as a proxy for shopper density. Here, you'd have labelers annotate aerial imagery on that mall's parking lot, simply putting a dot on each car. More often, however, you'll see dots being used for gesture and facial recognition.

As augmented reality initiatives become more and more common, so do gesture recognition AIs. Dot annotation is usually your best bet for these kinds of projects and the process is fairly simple. Essentially, you'd have labelers mark important points on the human body that your gesture project will care about, for example, every knuckle on a hand and the ends of each finger. With enough high-quality training data, this allows an AI to understand subtle but discrete hand motions, allowing end users to manipulate "objects" in augmented realities.



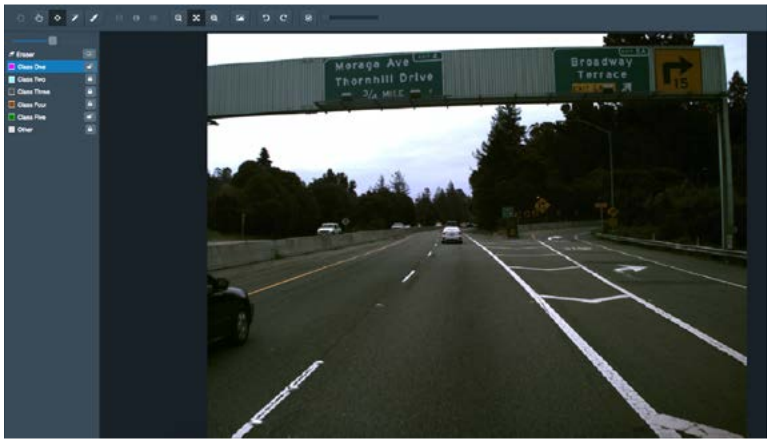
Similarly, facial recognition tasks involve marking certain points on faces: the corners of each eye, the tip of the nose, the corners of the mouth, and so on. These, in turn, allow facial recognition algorithms to recognize individuals by analyzing the unique ratios between these points on a person's face. Additionally, if you combine facial recognition with a categorization task, you could create an algorithm that detected emotion. People are great at understanding, at a glance, if another person is upset, laughing, frustrated, or any other emotion.

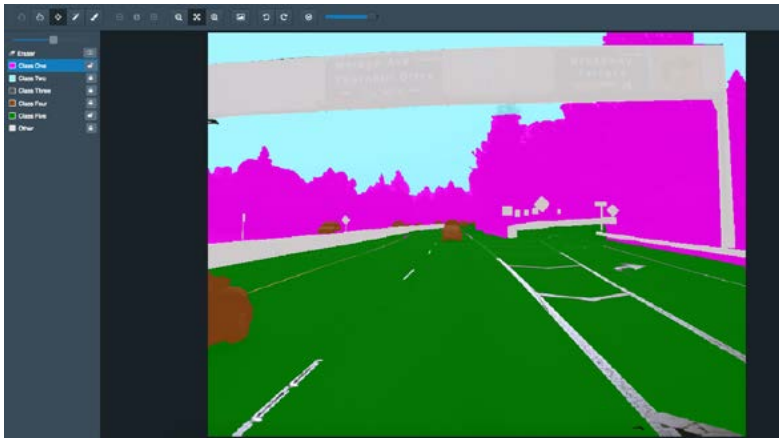
You'll also see a dot tool used for certain consumer packaged goods (or CPG) jobs. Sometimes, this will happen as a simple counting job, i.e. "put a dot on each can of beans in this image." You'll also sometimes see jobs where labelers annotate the corners of, say, cereal or anything else in a box. This actually gets around the issue we were talking about with boxes vs. polygons earlier and works quite well for computer vision algorithms in this space as you can get fairly tight coordinates for every object you're "dotting."

PIXEL LABELING OR SEMANTIC SEGMENTATION

Semantic segmentation is by far the most exacting type of image annotation. It involves labelers labeling every part of an image so that every pixel is accounted for. If this sounds like it takes a while, that's because it does. A fairly average image–in terms of size and complexity–takes anywhere from 45 minutes to an hour to annotate! That said, you'll generally need far fewer of these images to train a computer vision model because they're incredibly accurate.

Generally, it works like this: you provide a labeler with an image and an ontology of objects they need to find. This can be a city street for an automated vehicle project, an open box with parts in it for a manufacturing logistics job, or a whole host of other use cases. A labeler would see something like this:





See those colors on the side? That's the ontology used to label this image. Interestingly, because labelers will be annotating several objects in an entire image, it's often unnecessary to send these through image classification tasks first.

Semantic segmentation has become more and more common in the past few years. There are a few reasons for this. First off, it really is as exacting as you can get and, for a lot of computer vision projects, there's really no such thing as being too accurate. Additionally, semantic segmentation can be really ideal if you don't have copious amounts of source data. Because while more well-labeled data is always a good thing, if you have a limited amount for your project, you can get more actionable information for your models from every single image.

The flipside is that annotating pixel-by-pixel takes a while and, out of all these tasks, the cognitive load on the labeler is the highest for semantic segmentation. Since each image takes serious time and many involve robust ontologies, there's a bit of a higher chance for error here as well. Which is to say, like any annotation task, there are pros and cons. If you're thinking about starting any of these projects, we can help you decide what the right solution is for you and your project.

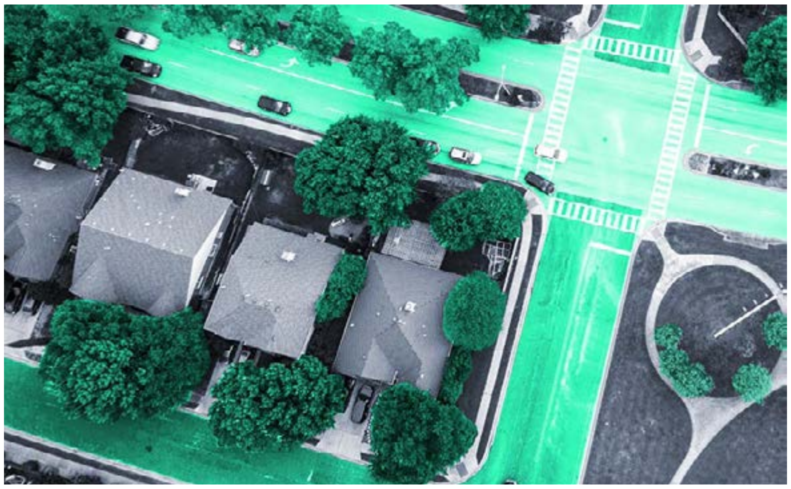
Now that we've gone over scoping and the tools, we thought we'd tell you a bit about how to actually get your images labeled. It's our speciality.

HOW TO GET REAL PEOPLE TO LABEL YOUR DATA

No matter what kind of computer vision project you're undertaking, you need lots of high-quality data to make it work. And while press articles about the latest advances in the field often gloss over this fact, practitioners know that this training data can be more important and more vital than leveraging the newest models.

Some companies have built in image labelers. Take the Facebook example we mentioned earlier: every time you tag your friend or your child on Facebook, somewhere behind the scenes, there's an algorithm understanding a bit more about what that person looks like and slowly learning to label him or her by itself. When a CAPTCHA modal pops up and asks you to click on every picture of a truck, you're actually doing double-duty: you're proving you're not a robot and you're training an algorithm.

Most companies don't have access to these natural "workforces." Instead, they turn to platforms like CrowdFlower to train, test, and tune their computer vision models. On a very high level, it works like this:



First, you upload the data you want annotated in a simple .csv. Then, you design the task you want people to do, using a template or creating your own. You choose the tool you want to use, write the instructions to get exactly what you want, and give a few examples of what you're looking for. Then? You launch your job. At that point, real human labelers–we call them contributors–get to classifying, labeling, and annotating your images. Our contributors work around the clock, following your instructions, and finish labeling your images. You download enriched data (or hook up to our API) and use that in your CV project.

A QUICK WORD ABOUT OUTPUT

The tool you use determines the enriched data you'll receive as an output from your image annotation job. It breaks down as follows:

• **Classification:** Full category breakdown of every image. If you ask "is there a person in this image?" you'll know whether there is for each image. The same is true of every question you ask.

• **Bounding boxes:** The output here is the image coordinate of the top left corner of your bounding box, plus the width and height of the box.

• **Dots:** A list of X/Y coordinates for each dot.

• **Lines or Polygons:** A series of coordinates for each shape. If the shape connects, the first and last coordinate will be the same.

• **Semantic segmentation:** The output here encodes ontology categories into an R,G,B pixel in the R value. If a contributor labels a certain pixel with the first category in your ontology, that pixel's value would be 1,0,0.

FAQs

CAN I USE SEVERAL DIFFERENT TOOLS FOR MY CV PROJECT?

Not only can you, but most projects do. This is a case-by-case determination that you and your team will want to make, but more data–and more diverse data–tend to create smarter, more accurate algorithms.

By the way, this extends beyond just image data. If you're working on autonomous vehicles, you're going to want to look at sensor data, LIDAR data, and a whole host of other sources to improve your performance. Variety's generally not a bad idea.

HELP! MY MODEL'S OVERFITTING!

If your data has a preponderance of a certain category–and if it's missing a lot of another category–it's likely going to overfit. Which is to say, if all it knows are cars, it's going to think everything is a car.

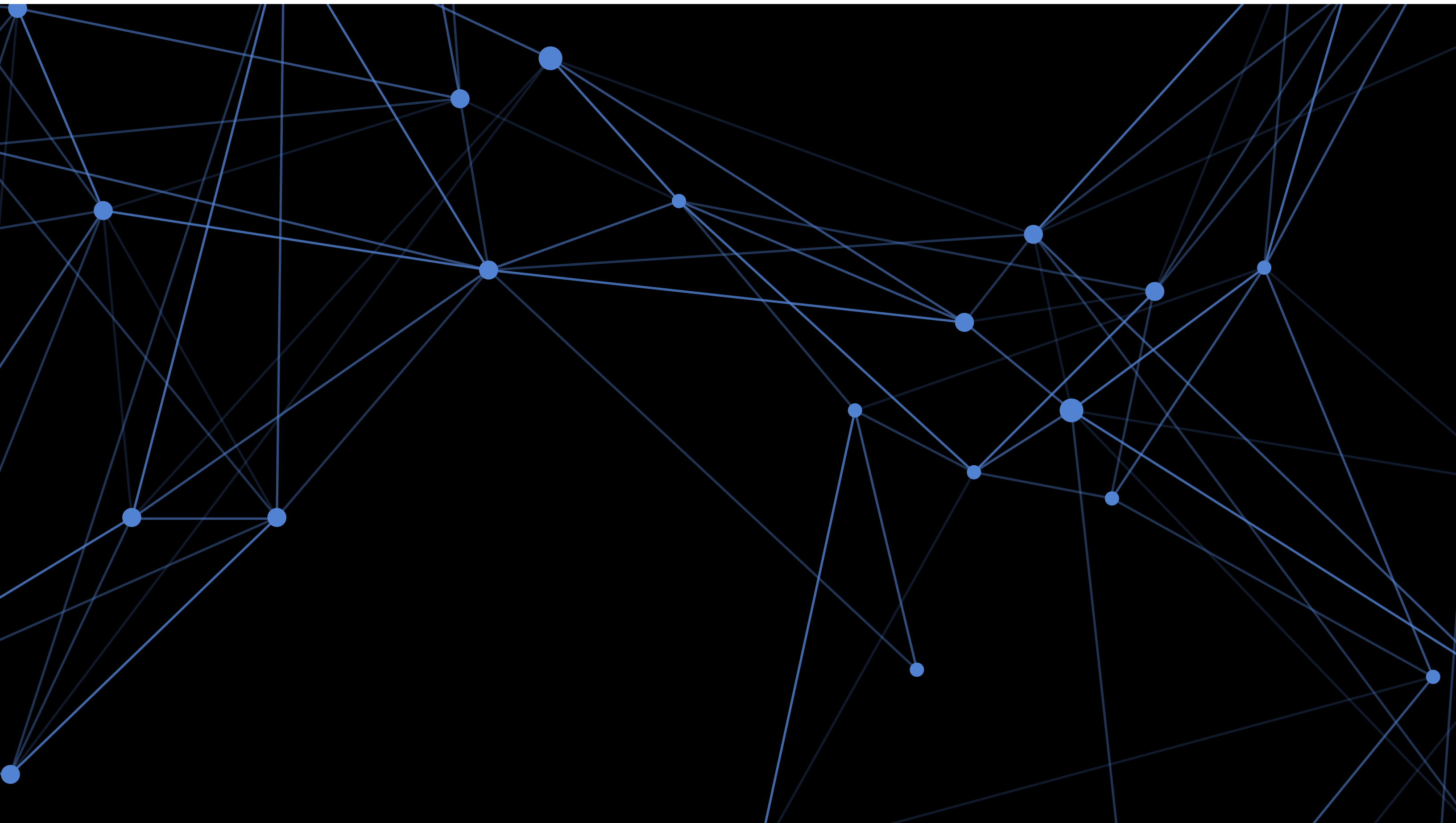
So how do you fix that? Simple! More data from under-represented categories.

WHAT TOOL SHOULD I USE?

Like "how much data should I use?", this is a tough question to answer without knowing the specifics of your project. If you'd like to get in touch, we'd be happy to help answer any questions you have. We're at **sales@crowdflower.com**.

HOW DO I SOURCE IMAGES?

Generally, you're going to want to use your own source data. If you don't have enough for your project, open datasets like Microsoft's Coco or Imagenet can be a good place to start. You can also purchase image sets or scrape public pages in a pinch.



CrowdFlower

www.crowdflower.com