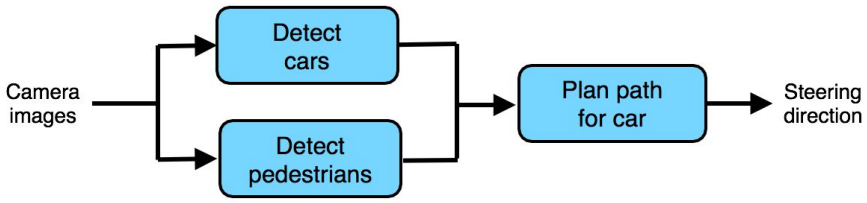
50 Choosing pipeline components: Data availability

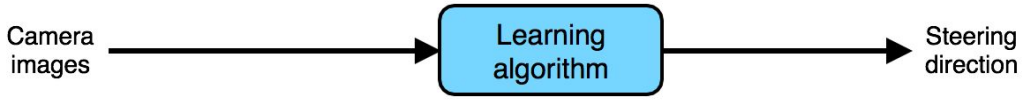
When building a non-end-to-end pipeline system, what are good candidates for the components of the pipeline? How you design the pipeline will greatly impact the overall system's performance. One important factor is whether you can easily collect data to train each of the components.

For example, consider this autonomous driving architecture:



You can use machine learning to detect cars and pedestrians. Further, it is not hard to obtain data for these: There are numerous computer vision datasets with large numbers of labeled cars and pedestrians. You can also use crowdsourcing (such as Amazon Mechanical Turk) to obtain even larger datasets. It is thus relatively easy to obtain training data to build a car detector and a pedestrian detector.

In contrast, consider a pure end-to-end approach:



To train this system, we would need a large dataset of (Image, Steering Direction) pairs. It is very time-consuming and expensive to have people drive cars around and record their steering direction to collect such data. You need a fleet of specially-instrumented cars, and a huge amount of driving to cover a wide range of possible scenarios. This makes an end-to-end system difficult to train. It is much easier to obtain a large dataset of labeled car or pedestrian images.

More generally, if there is a lot of data available for training "intermediate modules" of a pipeline (such as a car detector or a pedestrian detector), then you might consider using a

pipeline with multiple stages. This structure could be superior because you could use all that available data to train the intermediate modules.

Until more end-to-end data becomes available, I believe the non-end-to-end approach is significantly more promising for autonomous driving: Its architecture better matches the availability of data.

51 Choosing pipeline components: Task simplicity

Other than data availability, you should also consider a second factor when picking components of a pipeline: How simple are the tasks solved by the individual components? You should try to choose pipeline components that are individually easy to build or learn. But what does it mean for a component to be "easy" to learn?

Consider these machine learning tasks, listed in order of increasing difficulty:

1. Classifying whether an image is overexposed (like the example above)
2. Classifying whether an image was taken indoor or outdoor
3. Classifying whether an image contains a cat
4. Classifying whether an image contains a cat with both black and white fur
5. Classifying whether an image contains a Siamese cat (a particular breed of cat)

Each of these is a binary image classification task: You have to input an image, and output either 0 or 1. But the tasks earlier in the list seem much "easier" for a neural network to learn. You will be able to learn the easier tasks with fewer training examples.

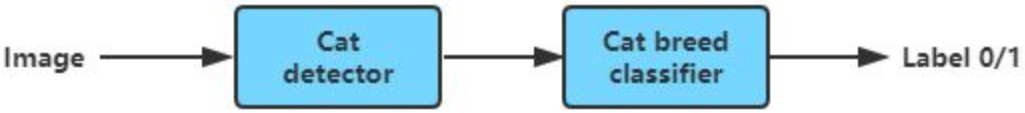
Machine learning does not yet have a good formal definition of what makes a task easy or hard.[16] With the rise of deep learning and multi-layered neural networks, we sometimes say a task is "easy" if it can be carried out with fewer computation steps (corresponding to a shallow neural network), and "hard" if it requires more computation steps (requiring a deeper neural network). But these are informal definitions.

[16]Information theory has the concept of "Kolmogorov Complexity", which says that the complexity of a learned function is the length of the shortest computer program that can produce that function. However, this theoretical concept has found few practical applications in AI. See also: https://en.wikipedia.org/wiki/Kolmogorov_complexity

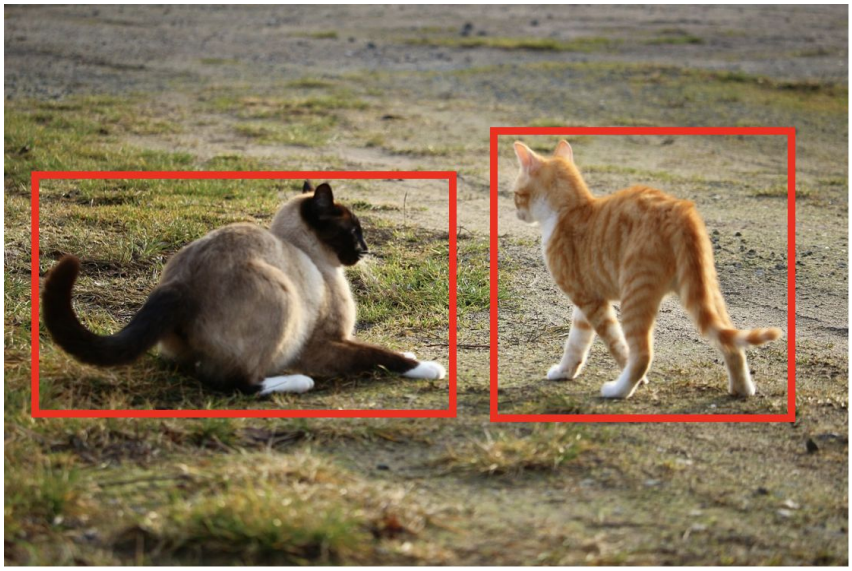
If you are able to take a complex task, and break it down into simpler sub-tasks, then by coding in the steps of the sub-tasks explicitly, you are giving the algorithm prior knowledge that can help it learn a task more efficiently.

Suppose you are building a Siamese cat detector. This is the pure end-to-end architecture:

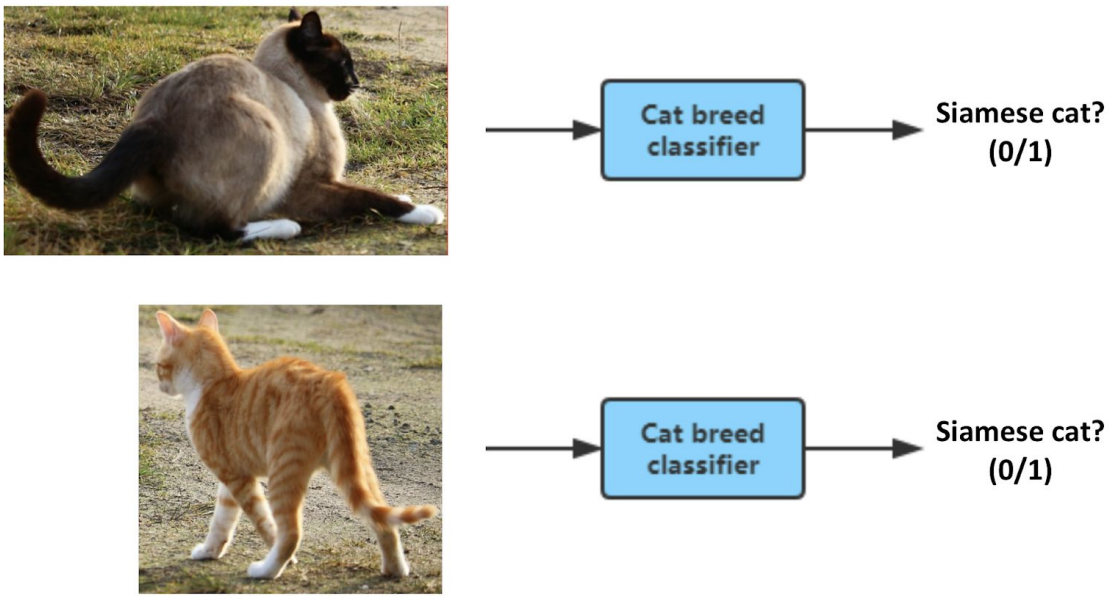
In contrast, you can alternatively use a pipeline with two steps:



The first step (cat detector) detects all the cats in the image.



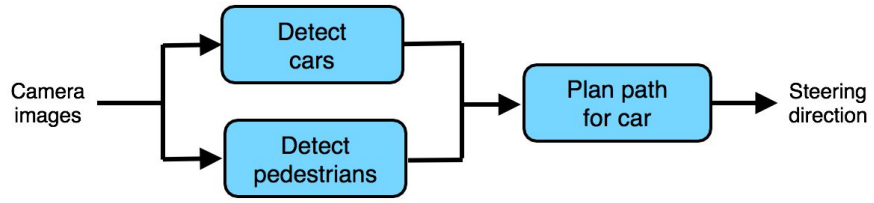
The second step then passes cropped images of each of the detected cats (one at a time) to a cat species classifier, and finally outputs 1 if any of the cats detected is a Siamese cat.



Compared to training a purely end-to-end classifier using just labels 0/1, each of the two components in the pipeline--the cat detector and the cat breed classifier--seem much easier to learn and will require significantly less data.[17]

[17] If you are familiar with practical object detection algorithms, you will recognize that they do not learn just with 0/1 image labels, but are instead trained with bounding boxes provided as part of the training data. A discussion of them is beyond the scope of this chapter. See the Deep Learning specialization on Coursera (http://deeplearning.ai) if you would like to learn more about such algorithms.

As one final example, let's revisit the autonomous driving pipeline.



By using this pipeline, you are telling the algorithm that there are 3 key steps to driving: (1) Detect other cars, (2) Detect pedestrians, and (3) Plan a path for your car. Further, each of these is a relatively simpler function--and can thus be learned with less data--than the purely end-to-end approach.

In summary, when deciding what should be the components of a pipeline, try to build a pipeline where each component is a relatively "simple" function that can therefore be learned from only a modest amount of data.

52 Directly learning rich outputs

An image classification algorithm will input an image x, and output an integer indicating the object category. Can an algorithm instead output an entire sentence describing the image?

For example:

x = 

y = "A yellow bus driving down a road with green trees and green grass in the background."

Traditional applications of supervised learning learned a function $h:X \rightarrow Y$, where the output y was usually an integer or a real number. For example:

Problem	X	Y
Spam classification	Email	Spam/Not spam (0/1)
Image recognition	Image	Integer label
Housing price prediction	Features of house	Price in dollars
Product recommendation	Product & user features	Chance of purchase

One of the most exciting developments in end-to-end deep learning is that it is letting us directly learn y that are much more complex than a number. In the image-captioning example above, you can have a neural network input an image (x) and directly output a caption (y).

Here are more examples:

Problem	X	Y	Example Citation
Image captioning	Image	Text	Mao et al., 2014
Machine translation	English text	French text	Suskever et al., 2014
Question answering	(Text,Question) pair	Answer text	Bordes et al., 2015
Speech recognition	Audio	Transcription	Hannun et al., 2015
TTS	Text features	Audio	van der Oord et al., 2016

This is an accelerating trend in deep learning: When you have the right (input,output) labeled pairs, you can sometimes learn end-to-end even when the output is a sentence, an image, audio, or other outputs that are richer than a single number.