Stability Analysis of a Neural Network

1 Introduction

Object detection in the context of image processing refers to the task of drawing boxes around objects in an image. An example is shown in Figure 1. The different type objects, or classes, can anythin such as $\{\texttt{person}, \texttt{cat}, \texttt{dog}, \ldots\}$. In this paper we only consider object detection for a single class, $\{\texttt{person}\}$

Since the recent success of using deep learing models for tasks such as object detection, companies, media outlets, and academic conferences have been referring to two major phenomena: (1) artificial intelligence (2) autonomous machines. This purpose of this paper is to address these phenomena. First, we argue that image processing is only distantly related to artificial intelligence. Second, we identify important issues limiting the application of computer vision to autonomous machines in commericial settings. The paper addresses these issues primarily by providing a perspective on the goal of object detection for the person class.

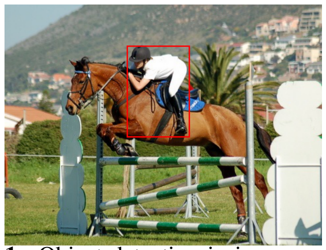


Figure 1. Object detection in image processing refers to the task of drawing boxes around different objects in an image. In this example, the person riding a horse is detected as person. The term *bounding-box* refers to the red box outlining the *object of interest*, which is person here.

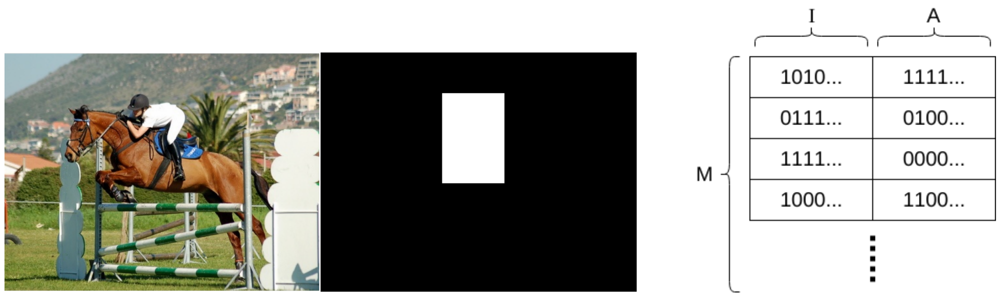
2 The Goal of Object Detection for **person**

The goal of image processing: to enable image data to be used as a reliable sensor, interpreting at least as much information from an image as humans do. There are many different perspectives for this task such as manifold learning classification image generation instance segmentation key-point detection and more This section provides a perspective on the goal of object detection of person.

2.1 Image Processing as Bits

Figure 2 shows two different representations of an image and its annotation for object detection of the person class. Figure 2(a) is the common way to view an image and groundtruth. The left image of Figure 2(a) shows a person riding a horse. The right image of Figure 2(a) is the annotation corresponding annotation, where white indicates the presence of person and black indicates there is not person.

Figure 2(b) shows images (column I) and annotations (column A) as binary strings where each row in the table is a different $(\text{image}, \text{annotation})$ pair. An color image of size 600×400 pixels is a binary number of length $2^{8\times600\times400\times3}$, denoted I. For object detection, the annotation can be represented as another image with size 600×400. The length of the annotation's binary representation is $2^{\times600\times400\times k}$ elements for k classes, denoted A. The number of elements in the table, denoted M, is at *at least* the number of images $2^{8\times600\times400\times3}$, $M \geq I$. In other words, every image is assigned at least one annotation. This is due to the ambiguity of an annotation; one image can have multiple, approximately correct annotations. This concept is illustrated in Figure 3.



(a) (b)

Figure 2. This figure emphasizes the relationship of an image and annotation as a series of binary numbers. Figure (a) illustrates a truth table for mapping an image to an annotation. The binary string on the left, an image, is mapped to a corresponding the binary string on the right, an annotation. The length of a color image size 600×400 in binary numbers is $2^{8\times600\times400\times3}$, denoted I in Figure (a). The length of an annotation for object detection is $2^{600\times400\times k}$ for k classes, denoted A in Figure (a). In this case, there is only one class (`person`), so $k = 1$. The number of elements in the table is denoted M in Figure (a), and M is *at least* I entries long. Figure (b) shows the common representation of an image and annotation.

Before we go further, lets take an aside to better understand how big $2^{8\times600\times400\times3}$ actually is. Let's first convert this to a base 10 number using the change of base formula where $x = 8 \times 600 \times 400 \times 3$: $\log 2^x = \log 10^y$. Then $y = \frac{x}{log2} \approx .3 * x > 600 \times 400$. So we have $2^{8\times600\times400\times3} > 10^{600\times400}$. According to some sources, the estimated total number of atoms in the observable universe is approximately 10^{80}. So the number of image and annotation pairs is *much* greater than the number of atoms. In fact, $\left(10^{600\times400}\right)^{1/2} \approx 10^{490} >> 10^{80}$. There are a lot of image and annotation pairs.

By explicitly viewing the information as binary strings, as in Figure 2(b), this shows how image processing could theoretically be solved with a look-up table. The lookup table also (hopefully) provides a more intuitive understanding of the role machine learning has in image processing. Since the size of the table is too big, machine learning essentially is a "short-cut" for the actual lookup table. The goal of image processing is to create this "short-cut". To be more precise, lets rename the "short-cut" to "function" and label the function that maps each image to each annotation, f.

2.2 Multiple Annotations Per Image

The previous analysis assumes each image is assigned one correct annotation. Clearly, this is not correct. Figure 3 shows how slight changed to the annotation can still be interpreted as an equally the correct annotation. Therefore, each image can be assigned more than one annotation image. The impact on the previous analysis implies the "lookup table" just returns a vector of equally correct annotations. We can consider the original lookup table as a special case where the annotation image is the arithmetic average of all equally correct annotations.

Furthermore, notice how much of the red box covers pixels that clearly do not belong to `person`. This means the annotation method itself, drawing boxes around an object of interest, has some inherent error as well. This error we will refer to as the annotation error bound. As an example, consider three types of annotation classes: (1) squares, (2) rectangles, (3) pixel-wise class assignment. The first class, squares, has a larger annotation error bound than the second class, rectangles. And the second class, rectangles, has a higher annotation error bound than the third class, pixel-wise class assignment. Also note as the annotation error bound decreases, the complexity of the annotation increases.

2.3 Not AI

Artificial Intelligence is defined by webster as "the capability of a machine to imitate intelligent human behavior". Maybe some consider the task of object detection for `person` to be "intelligent human behavior", but I think not. The capability of drawing boxes around people does not seem particularly *human*. Therefore, object detection for `person`, but instead it is to construct the aforementioned "short-cut" function.

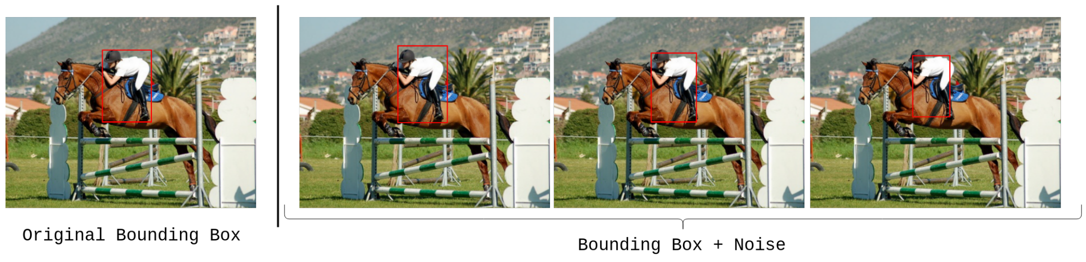


Figure 3. Small changes to the original bounding box create equally accurate bounding box annotations.

3 Constructing The "Short-cut", f

3.1 Verifying f

Now that we understand the goal of object detection is to construction a "short-cut" function, f, in lieu of our lookup table, we have some questions:

- How do we know if f is correct? or even "good"?
  - Even if we could store all the groundth truth to verify if a given f is correct, we don't have enough time for exhaustive comparison. If each comparison took 10^{-6} seconds, we still need more than $10^{600\times 400-6}$ seconds For comparison, the universe is estimated to be about 13.7 billion years old or about 10^{17} seconds.
  - But what about "good"? Here, "good" means f is approximately correct, a purposely vague term. Now "good" implies some error is acceptable. But we still end up in the same situation as above. This is where "PAC" learnability becomes useful.

Clearly, the initial answer seems to be "we can't verify f". This is a **huge** issue. For any autonomous machine, incorrecly interpreting sensor data can be fatal. Verifying that f is correct is (or at least should to be) **required**. So how do we solve this problem?

3.2 Prior Knowledge

Insert the keyword of machine learning: *prior knowledge*. So what if we constrained our f so that we didn't need to explicitly check all the images? What if we defined some properties on our function f (the short-cut for the lookup table in Figure 2(b)) so that if we knew the answer to a few of the $(\text{image}, \text{annotation})$ pairs we knew the answer to all of them? What properties would we want? How would this impact our performance? These assumptions about f is what people in machine learning refer to as *prior knowledge*; it is knowledge about f we use to contrain our function and control f's complexity, which in turn reduces the number of images we need to verify. These concepts are rooted in PAC (probably approximately correct) Learnability. The simpler we construct f, the fewer images we need to verify to have an approximately correct estimate of the actual error. The more complex we construct f, the more images we must verify.

An aside: Note what we wanted is slightly different from what was just stated. The previous paragraph outlines two different topics: *knowing* the remaining output values given some subset of image v.s. our model being *probably approximately correct* (or knowing, approximately, the total error with a specified probability). Below some examples highlight what PAC Learnability provides v.s. what we originally wanted.

One silly example is the constant function, $f(x) = c$. In other words, every image is mapped to the same annotation. Sure, now we only have 1 annotation to check, but we know from our *prior knowldege* that f's correctness is terrible. Another silly example function is $f(x) = g(x_1)$, where g is a function of only the first pixel x_1. Now our function yields 8 different annotations (for `uint8` precision), but again, we know from our *prior knowldege* that f's correctness is terrible.

In contrast to the above examples of controlling for f, we want to *know* other outputs of f given some contraints on the function. A more useful property would be *smoothness* of f. This means that every small change to the input of f yields a proportionately small change to the output of f.

Therefore, if we know that f