Chapter - 2

**Computer Vision**

**SSD: Object detection**

How SSD is different, The Multi-Box Concept,

Predicting Object Positions, The Scale Problem.

**2.1 Introduction**

In this chapter we will learn about Single Shot Multi-Box Detection algorithm.

* What we will learn in this chapter:
* How Single Shot Detection (SSD) is different and what are the innovations that it has become one of the top algorithms in the world.
* The Multi-Box concept
* Predicting object positions & how SSD handles that.
* The scale problem

**2.2 SSD: why it is different?**

* Consider following image: we can see some sheep walking around on a field. The question is, how would normally a computer or object detection algorithms detect these sheep?
* We need to detect exactly where the sheep are and put boxes around them.



* Normally these algorithms would try to save time or the computational cost by through certain tricks and hacks.
* Object proposal: A huge portion of the algorithms would engage in something called an Object Proposal Methodology.
* Basically you've got an image and of course is a huge image.
* Normally these images are resized to smaller images and the algorithms work with smaller images and then maybe they resized back in and so on.

* But we still have the same problem, for instance convert this whole image with rectangle and then try to guess which one contains a sheep or not.
* We can see that most of the rectangles don't actually have the sheep and you would be spending a lot of time or the computational cost by finding these sheep. And the algorithm would be slow and that's a big problem.

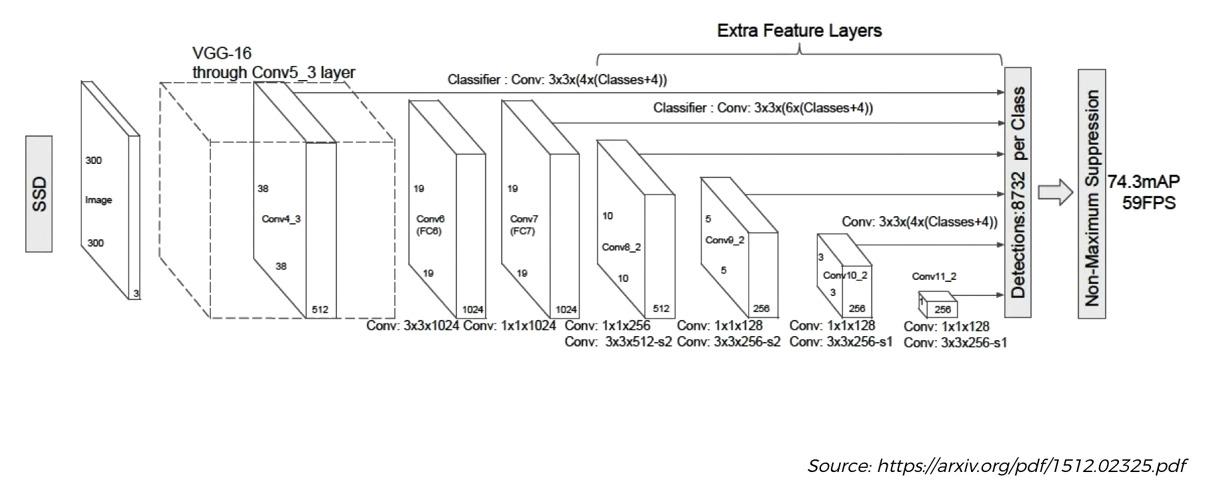


* The object proposal break down the image into parts to suggest where the objects could present. For instance by looking at the gradients in color (in gradient mode, empty field area has nearly same gradient). Therefore you could make a proposal that there is no object in here.
* It depends: Sometimes a shadow would be detected (because of gradient change).
* That’s how we are predicting any types of objects that were in that area. For example: we could be looking for mice and we could be looking for cars and we could look looking for giraffes, anything.
* So it's very dangerous to discard anything any object because then you might miss out some real object.



* Sacrifice On Accuracy: So there is a sacrifice on accuracy, because there's are lots of other details too. In short to minimize computational cost we have to sacrifice accuracy.
* But in some implication in other applications we need High Accuracy especially in self-driving cars and things like that you can't afford to sacrifice accuracy but at the same time you *also need for computational efficiency*.

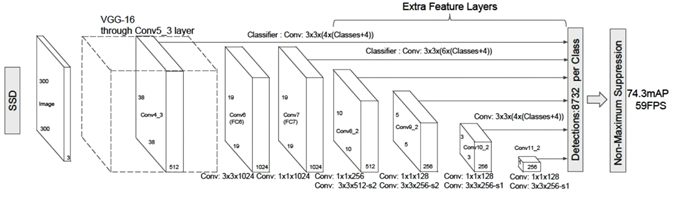
**2.3 The idea of SSD**



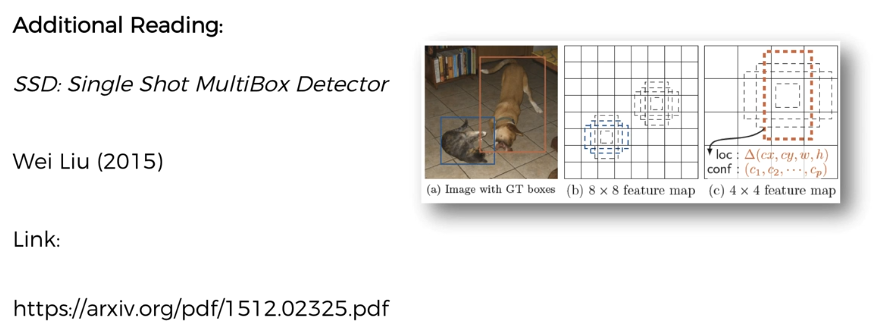
* The idea is: everything is done in One Shot. That's why it's called the Single Shot Multi-Box detection algorithm.
* It just looks at the image once,
* It doesn't have to go back to it.
* It doesn't have to do this Object Proposal
* It doesn't have to then run many CNN (that's the part where the computational cost comes in).
* Manny CNN computational costly: is For example, if you break the image down in to several parts and then you run a of neural network for every single one of those rectangles to detect a then that's can be really computational costly and that's where you need to do these object proposals to ignore some image parts. To run the minimum CNN.



* But in SSD algorithm they only look at it once. It does only one CNN.
* Note (YOLO): There's only one other algorithm in the world that does the same thing. It's called the YOLO algorithm. YOLO (You Only Look Once) is an open-source object detection system. It can recognize objects on a single image or a video stream rapidly.
* Whereas SSD (Single-Shot Multi-box Detection) detects objects with high precision in a single forward pass computing feature map.
* Both YOLO and SSD are top object detection tools, one quicker and the other more precise.
* YOLO and SSD both have tons of applications. Some of the applications include *traffic monitoring*, *media forensic*, and *security*.
* *YOLO* is better where we can *ignore* a *slight inaccuracy*. Some examples are live traffic monitoring, life form detection in inaccessible areas, and fruit-vegetable monitoring.
* *SSD* is beneficial for *more precise object detection*. It is more suitable for video forensics, legal detections, and landmark detections.
* Apart from YOLO and SSD there is no other algorithm that actually only looks at the image once and does only one CNN.
* All of the following Boxes-Of layers are executed at the same time.
* And moreover not just to go through the network at the same time, these networks also remembers which boxes dealing which features they are training with.



* Also there's many convolutions to reduce the image size, you can see it stars with 300px, 38px, 19px, 10px, 5px, 3px, 1px.
* When image size reduces, it helps to deal with different sizes of objects on the original image.
* In simple words, what SSD do is: So there are lots of hacks in this one algorithm and probably one of the most important ones is of course that they do everything in a single shot. All these it all happens in one huge convolution.
* It does everything in a single shot.
* It can deal with training of identifying objects and also identifying the borders
* It can deal with scale at the same time.
* Additional Readings: The original paper by Wei Liu (2015) and others is called Single Shot Multi Box Detector you can find our archive there's a link.



**2.3 Multi-Box concept**

Here we'll discuss about the MULTI BOX concept behind the SSD algorithm.

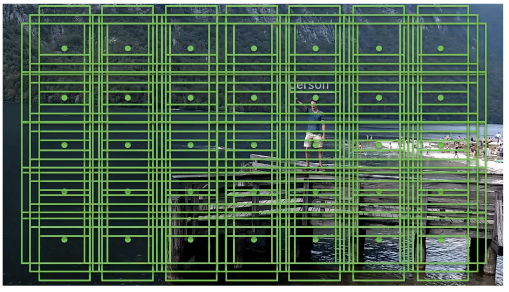
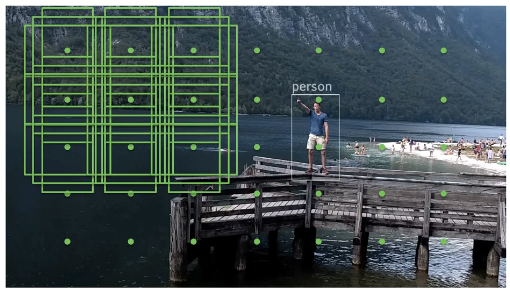


* We're going to take this Box (around the person) as the GROUND TRUTH and then we're going to see how SSD works with its own boxes to ***understand & identify*** that person and build this ***same exact box***.
* Ground Truth: A ground truth is a concept that is used in many places (not just computer science) and it separates the observed evidence or empirical evidence from inferred evidence. (Ground truth is some kind of labeled data.)
* For instance, when you *actually look at something* and *physically understand* that this is a true state of things, then that's your ground truth.
* Inferred Boxes: When you apply an algorithm and it goes processes all of the given visual data through its CNN and comes back with boxes, those are going to be the INFERRED BOXES (inferred evidence).
* Those are going to be suggestion brought up by the algorithm, even if they're correct, they're still suggestions and in order to make sure that they're correct we need to have the GROUND TRUTH.
* So in the process of training SSD we need the GROUND TRUTH. So during the training process, we need these pre-drawn boxes around the person to be identified.
* How The Algorithm Will Work: Let's see how the algorithm will work and build its own boxes and how the training will happen.
* It will break down the *image* into *segments* and then for every segment it will construct *several boxes*. However, these exact set up can vary. We are doing it for *simplicity* and for *learning* *purpose*.

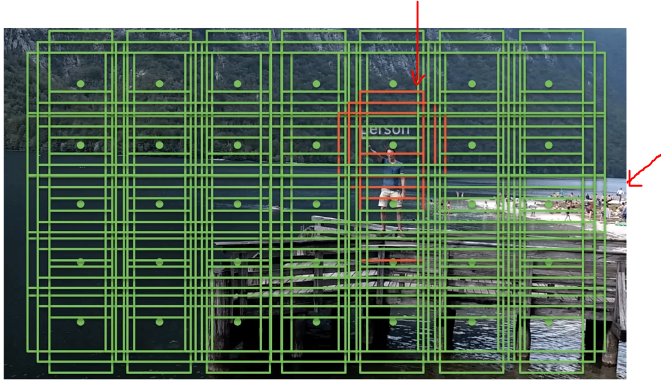




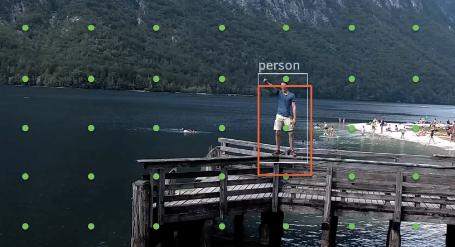
* Basically the whole image will get covered with these boxes. As you can see it's covered entirely.



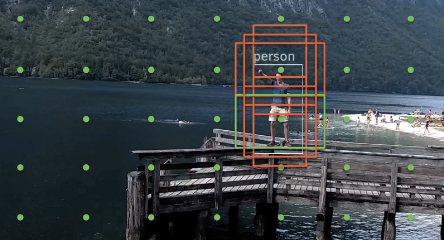
* Then for every single one of these boxes, it's going to ask the question ***"Is there's an object or not?"***, not just an object, it's going for every single class of objects that it trained for (during test) or training for (in the training process).
* So in our case it will identify that none of these boxes except for the ones highlighted in RED that actually have an object of interest, a person.



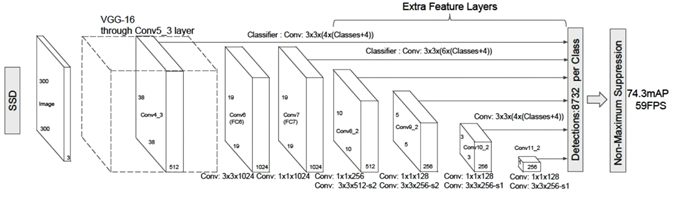
* For simplicity's sake in this case we're only looking for people. There are also people in the background, but they're very small. And the algorithm will not be able to pick up their features and therefore will not detect those people.
* Lets remove all boxes and just look at one box, we can see that there is a person (or a Car or a Boat or a Cat or a Dog) within this box. It doesn't have to be a full person. As long as it can find a sufficient number of features it will produce a probability for the existence of a person inside that box.



* And then all of following boxes, we have identified almost the same features.



* Notice there is also a green box it only sees from the elbow to downwards. In this case it's not seeing enough features (we say that for learning purpose, in reality that would be enough features for the SSD algorithm to detect a human being).



* Training process: And every time all of these box-of-layers predict something, we'll have an output. The neural network searches for features that there's a human inside that box.
* On the other hand, if it predicted that some of other boxes that don't have a human. Then at the end it would generate an error by comparing the Ground Truth (the white box).
* The *red* boxes predict there *is a human*, and other green boxes predict there are *no human*.
* If these boxes predict wrongly then we get an error. At the end of the algorithm that error can be calculated and can be expressed mathematically.
* This error will be back propagated through the NN and the weights of the network will be updated so that it can prevent the error in next iteration.

That's how the training process works.

* Note: The important thing is, in a simple CNN we do label images (EG: to predict Cat or Dog), but in this case it's not sufficient.
* To detect a person (or an object) we need this white-box (the ground truth) already on the image for the training process.
* If we take each one of these boxes (red/green) as a separate image where the CNN is applied, then it is similar to the simple CNN. And each one is trying to understand if there a human inside or not (doesn't care about the rest of the image).
* Then the algorithm puts everything together at the and it calculates the error and propagates through the NN and trains everything up.

**2.4 Predicting Object POSITIONS**

Here we'll discuss about how *SSD* predicts *object positions*.

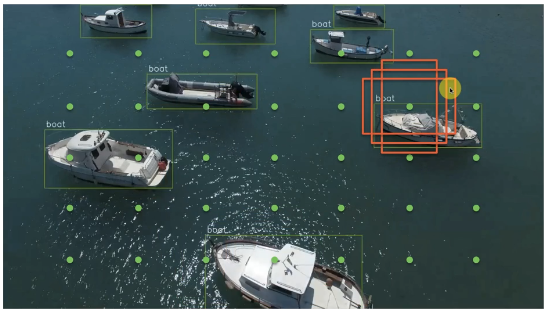


* We're going to overlay a grid over this image
* For every single circle/points on this grid, we're going to introduce a couple of rectangles and we're going to stick with our example with three rectangles for each grid.
* We're going to take each one of these *rectangles* as a *separate image*.
* Within each rectangle the CNN will ask the question: *" is there a boat or not "*.

|  |  |
| --- | --- |
| * However, some boat can be recognized as other objects, for example an airplane or a boat and an airplane etc. So it's looking for all classes of objects at the same time. Sometimes those errors happen, it can confuse an object. * Basically each one of these squares/rectangles, is going to ask * "Is There Any Object Inside Here?". * "Can I detect any features of an object?". * Yes or no. * In our case, after the training in the ideal scenario of these following rectangles will pick up features of Boats. |  |

|  |  |
| --- | --- |
|  |  |

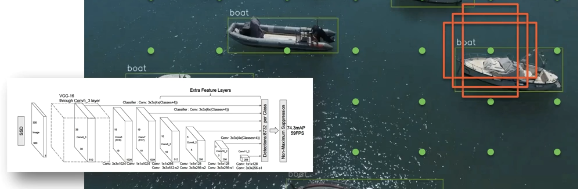
* You can see that these rectangles do overlap with our ground truth and therefore it's fair to say that SSD will pick up some sort of certain features of our Boats.
* Note that, during training process, SSD will do some mistakes: for example, detecting wrong object, detecting object at empty area etc.
* That’s the part of learning, during training it will learn from its mistakes. They will learn what the features of boats are and how to pick them up and so on.
* And after the training it can detect the object with better accuracy.
* So after the training, the trained model (after the network has trained) will detect the boats, as the right-side image (with red squares).
* So let's have a look at one specific set of these three boxes for instance following one.



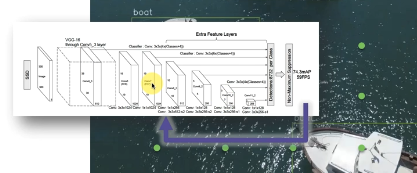
* The interesting thing is that these boxes have picked up features of a boat. Maybe the *front of the boat*; the *anchor line in the water*; *controller area* where the driver stands. Some of the box can see more, some of them can see less. Not all of them can see the full image of the boat.
* None of the box is seeing the full Ground Truth of the boat.
* So, how can a box (the temporary/inferred boxes) predict where exactly the Full Boat will be?
* That is another special part in the SSD algorithm.
* The underlying principle is that we have the ground truth during the training. During the training the SSD algorithm we have these Ground Truths, those actual boxes (white boxes).
* Then based on these individual boxes, SSD will make a prediction about the Full Boat (the full rectangle), about its size, its shape. It will *predict* about the *Box around the Ground Truth*.

|  |  |
| --- | --- |
| * However we do have overlapping, * That makes it easier to predict, that's of course is a high probability of a boat. * But nevertheless it still makes errors, for example, that the boat is higher, longer or wider or something like that. |  |

* It is also resolved during the training.



* So at the end, once it's made its predictions there will be a certain *Box around the Ground Truth* (not the real box but a prediction).
* Then the predicted box will be compared with the *Ground Truth* (real box) to calculate the ERROR. (In terms of coordinates it can be calculated in terms of distance. There's lots of mathematics going on in the background.)
* That ERROR then propagated back through the network in order to update the weights to minimize the ERROR.



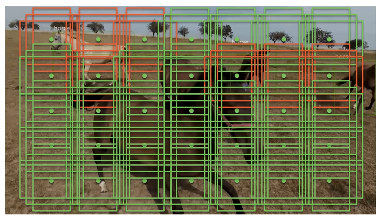
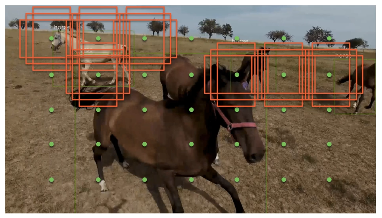
* **The main point is, through training two things happen:**
* First thing is that, each box will learn to better classify objects or better identify if it has objects inside there. Each box will use the Ground Truth to assess that.
* The second thing is they will get better at identifying the exact final output rectangle. More accurately predict the Box around the Ground Truth.
* SSD will predict the rectangle which matches the *real Box around ground truth*.
* So gradually, we're building up the different components of this SSD.
* How to learn to classify objects through the ground truth
* But also to identify the correct widths and heights of those boxes.

**2.5 Scale Problem**

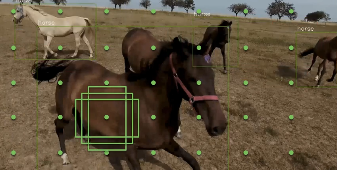
Here we'll see how SSD solves the scale problem.

* Scale problem: Well let's have a look at following image. We have some Horses. How SSD will detect those Horses.

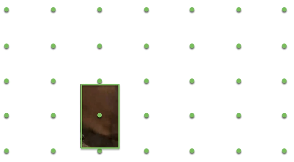


* We can see *some* of the *boxes* identified some *horse features* and some are not.
* After removing the green boxes we can see that somehow this Horse Right In Front, the Main Horse not being detected. The algorithm did not pick it up.
* The reason is that this horse is too close. It's too big for the algorithm to pick up features.

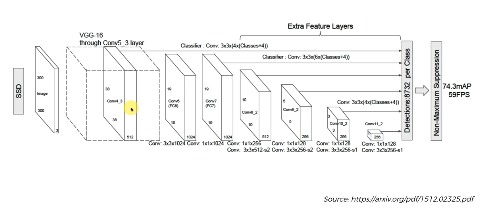


* The horse doesn't fit inside the rectangles, but also remember that, each of these rectangles act like a *separate image*. For this reason the Biggest Horse in front is not detected.

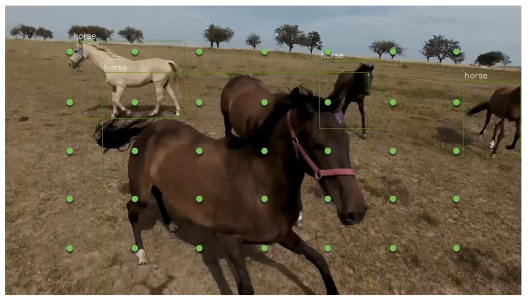


|  |  |
| --- | --- |
| * What about other features: But there's other features on the horse that you could identify from this image. For example the hooves, the tail or the eyes. Well let's have a look at that as well. Let's have a look at another rectangle. * We can kind of identify that there is an eye. The SSD algorithm should be also leveraging that picture. |  |

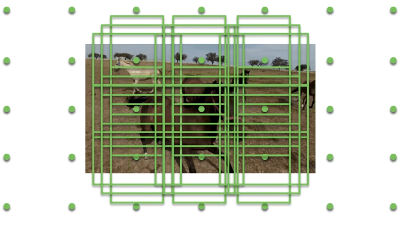
* But it's also hard to predict the Full Box: But remember that SSD not only identify the horse but it also need to make a prediction for the actual box that contains this horse.
* From the given piece of rectangle image we cannot tell where the horse is actually located.
* So it's very hard for the algorithm to predict what the horse looks like what the full image of the horses are like. And most rectangles cannot even see that horse. So it is very hard to predict about the full box that contains the Horse. As a result the Horse is not identified.
* Scaling: So this is where the SSD's next component that comes to the rescue.



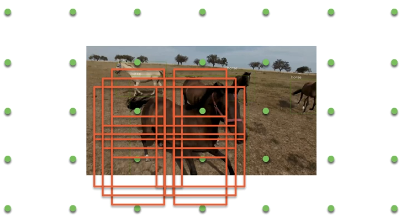
* Here in the architecture we can see there's many layers. Through these layers, the image region (rectangles) is constantly reduced in smaller-size. It applies a *Convolution* operation to *reduce* the *image size*.
* So each steps we are seeing in the above ar5chitecture, the image is reduced.
* All the SSD techniques we've talked about are applied on every single step on all of those reduced images.
* As a result we can get 8732 detections for all the class, since there are so many iterations.
* Every time we convolve the image further we are applying previous steps classifiers, to detect objects (Horses). Thus on every single step, we are detecting Horses on the reduced (convolved) images.
* Everything is being repeated again and again, and simultaneously Horses being detected (on different size images).
* Let's have a look at how that happens, following is our image.
* During the convolution it won't just resized but it also looks different (for simplicity & visual purposes we considered resizing).



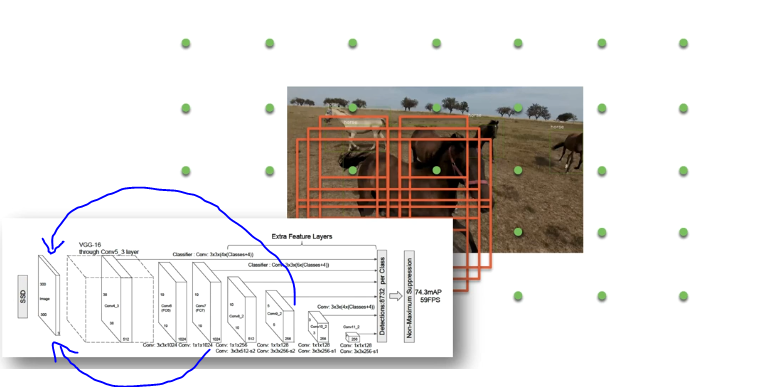
* In reality it's not just resized, when the image goes through convolution, it will be completely random looking thing for us (visually) but it will preserve the features.



* So for now we stick with only resizing. So the image is resized but the rectangles are staying the same size. Then those rectangles are applied to this new resized (and convolved) image.
* Since the image is now smaller, that means that horse is smaller. And these rectangles detects the features of horses for the biggest Horse (that previously was not detected).
* But the other horses in the background wont picked up because they are too small now.



* Now the question is, since we detected the biggest horse, how can we draw the box for this horse in the original image?
* We know how the algorithm predicts about the box in this version of image (basically the algorithms learn to predict the box from lots of Ground Truth, given examples of horses or other objects).
* But remember that we got this image not just resizing but through a convolution operation. So the question is: How do we take this box and move it back?



* Well, the answer is pretty straightforward. Let's look at our architecture again.
* The algorithm is constructed in such a way that it preserves information about how to return/scale-back to the original image.
* Let's the ***Big Horse*** is detected a layer of ***30x30px layer***, the SSD algorithm remembers that. And it will also *know* how to *get back* from *this layer* to *original scale*. Thus the SSD knows how to *de-convolve* from any of its *layer* to the *original image scale* and where it should place the horse.

In this way, SSD handles the objects of different sizes.

* One Neural Network: The algorithm simply doesn't just resize the image and then proceeds do the conversion operation again.
* The idea here is that all this happens at the same time all of these boxes that we're detecting all of the positions of the boxes the classes inside the boxes and even the resizing of the different scales of the images.
* It all happens in One Neural Network.
* The main beauty of SSD is: Instead of just having copy image in different size, and running neural networks for each one, we're running all of this in One Neural Network. And why is that good.
* Why it's important? Because we can utilize and share the features of a specific layer, across the different layers so that we can train the network to learn how to detect horses in different size.
* So it’s the way that way All Of The Layers Are Learning Together.
* And that gives it that *additional power "the power in numbers"* that they're seeing more different horses at the same time.
* That’s how it has more examples to work off and the network can better adjust its weights to properly detect those horses.

That's how the SSD algorithm deals with the SCALE PROBLEM.