

Recognition: Past, present, future?



Benozzo Gozzoli, Journey of the Magi, c. 1459

Last time: Overview of recognition

- Brief history of recognition
- Different “dimensions” of recognition
 - What type of content?
 - What type of output?
 - What type of supervision?
- Trends
 - Saturation of supervised learning
 - Transformers
 - Vision-language models
 - “Universal” recognition systems
 - Text-to-image generation
 - From vision to action

Simple machinery

Classification, Detection and Regression

0: Why

I: Classification

- Basic classification - features to logistic regression; facts of life
- variant classification (words from pictures; others)
- lane boundaries
- semantic segmentation; masks?; labelling 3D worlds ala torr

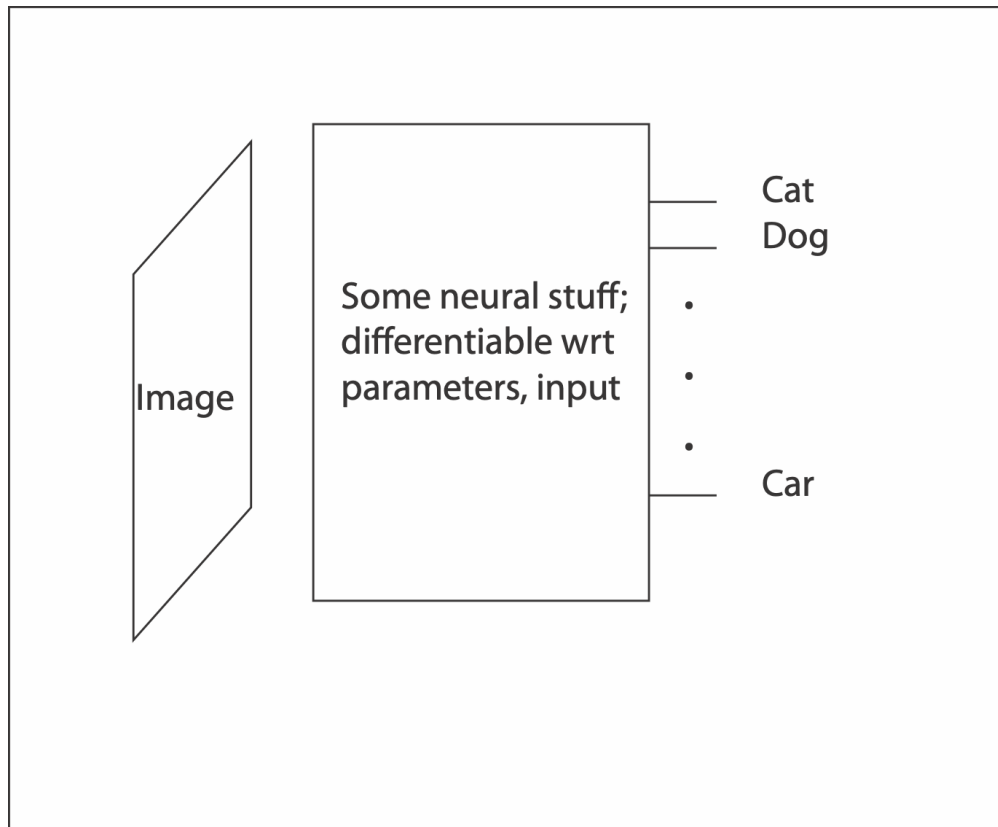
II: Detection

- localization + classification FasterRCNN, YOLO
- MaskRCNN
- 3D detection Det?

III: Regression

- (depth from single images is possible); Boxes and primitives

Image classification



Key ideas

Goal:

- Adjust classifier so that it accurately classifies *UNSEEN* data

Procedure:

- Adjust so that it
 - classifies training data well
 - generalizes
- regularization term, either explicit or implicit

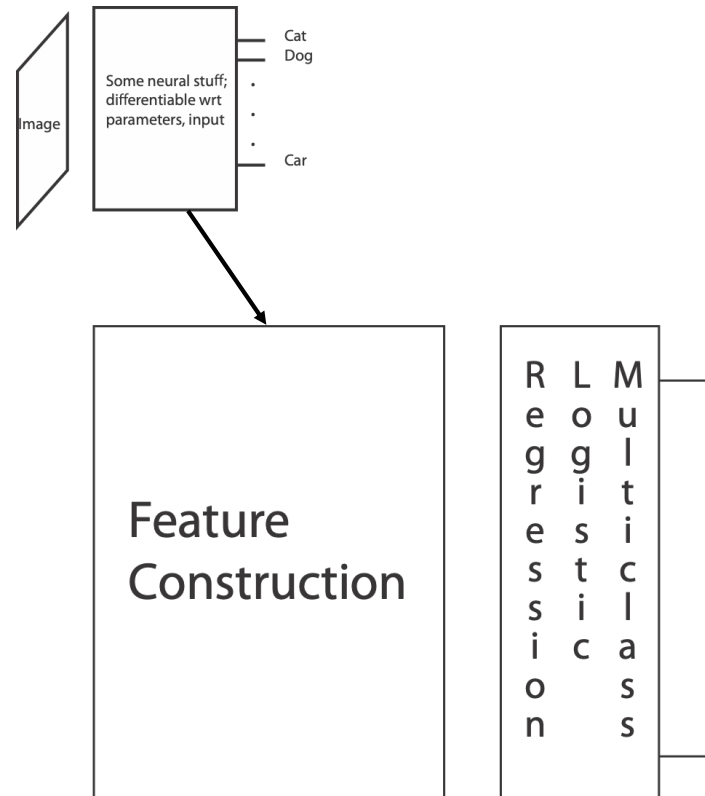
Evaluation:

- Use held out data to check accuracy on *UNSEEN* data

Main Points

Remember this: *A classifier predicts a label from a representation. Classifiers are evaluated by accuracy or error rate, estimated on data not used in training. The standard recipe splits training data into two components (train and test), uses one to train the classifier and the other to evaluate it. Never evaluate on data that was used in training, because your estimate will be wrong.*

Under the hood



Olden days

Learned

Feature
Construction

(by hand,
from insight)

R L M
e o u
g g l
r i t
e s i
s t c
s i a
o c s
n s

Multiclass logistic regression

For classes 1, ..., C

Given a feature vector

Form

$$\mathbf{x}$$
$$\mathbf{w}_i^T \mathbf{x}$$

Interpret by

$$P(\text{example is of class } i) = \frac{e^{\mathbf{w}_i^T \mathbf{x}}}{\sum_k e^{\mathbf{w}_k^T \mathbf{x}}}$$

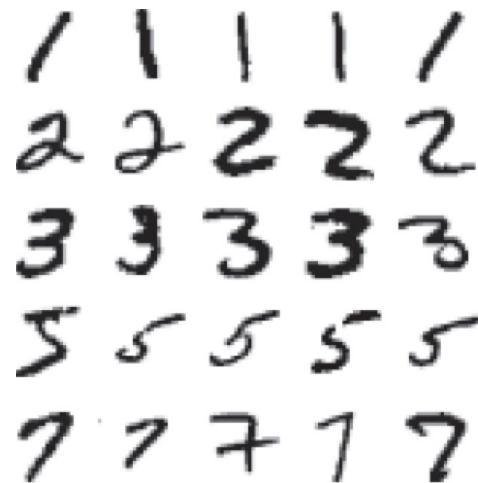
Multiclass logistic regression - II

Adjust w_i to maximize log-likelihood on training data

- possibly regularizing by magnitude

But what is x ?

- Olden days: by hand



Multiclass logistic regression

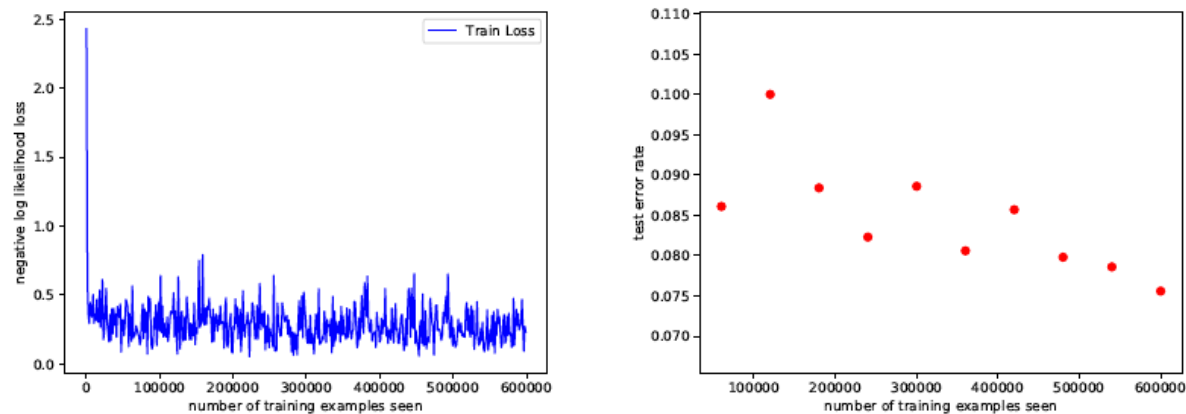


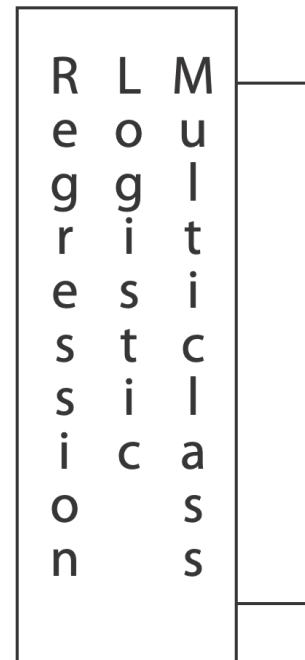
FIGURE 5.1: *On the left, the learning curve for a logistic regression classifier trained on MNIST data. Note the loss falls off quickly, then declines very slowly. The loss plotted here is the loss for a particular batch after a step has been taken using the gradient on that batch. Although the step follows the gradient, it may cause the loss to rise because it goes too far along the gradient direction — there is no search for a step length that guarantees descent and there are no second order terms here. Nonetheless, because the steps are small and approximately in the right direction, the loss declines. On the right, the error rate for the test set plotted at the end of each epoch. Notice how this declines, but not monotonically.*

“Modernity”

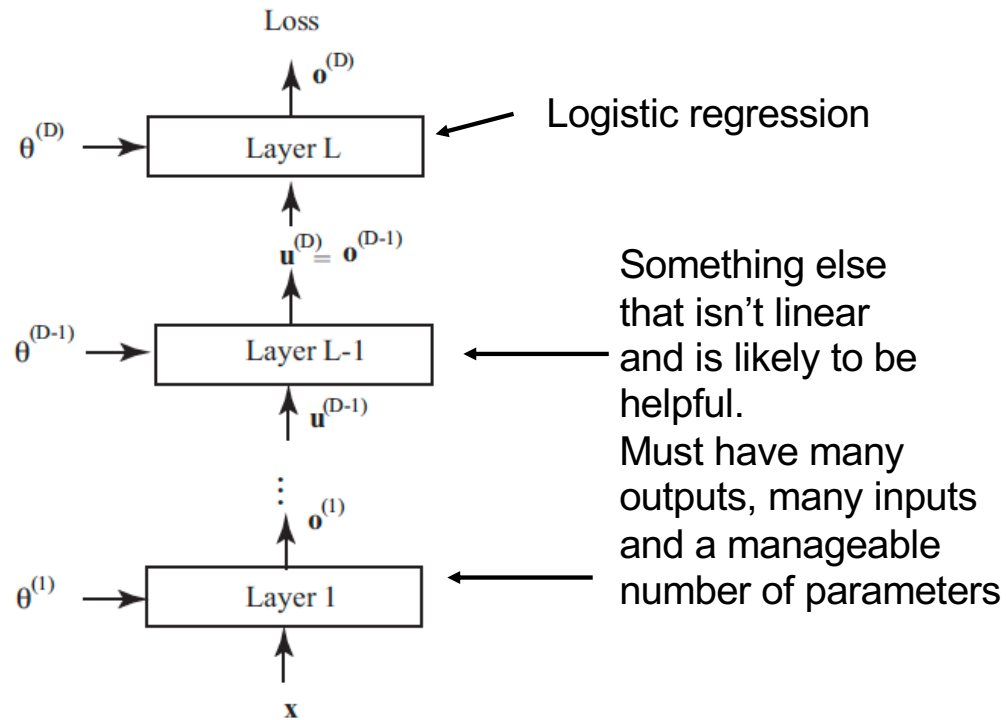
Learned



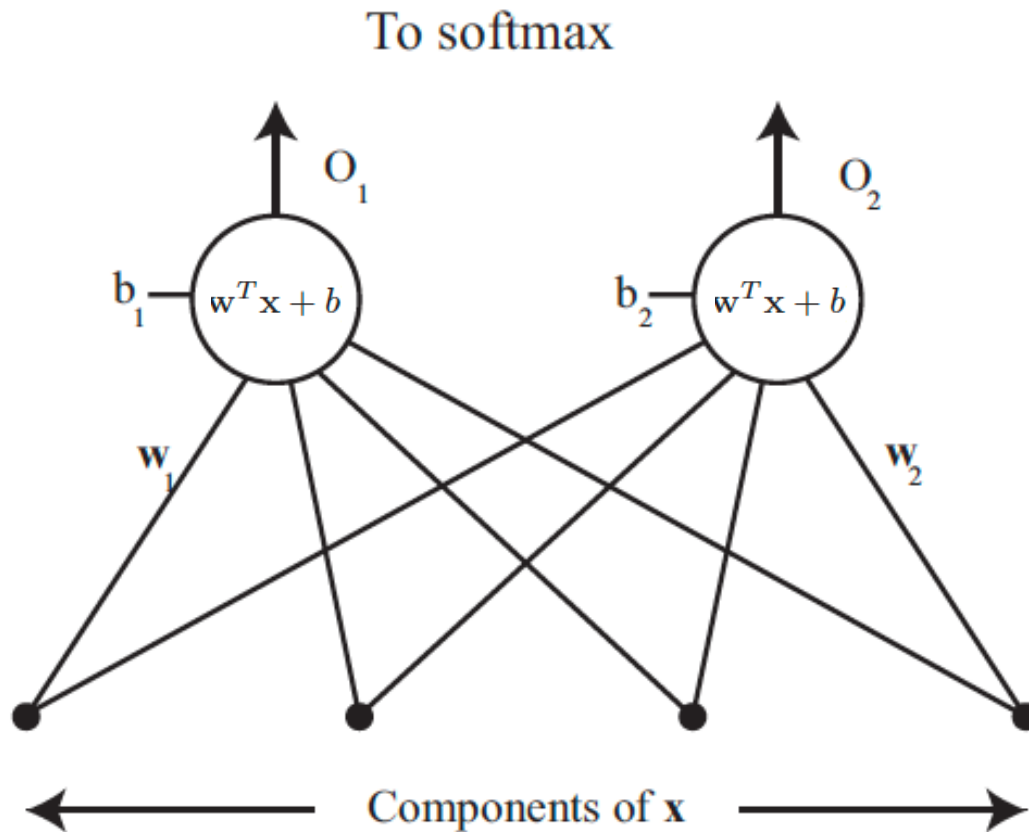
Learned



Making features using layers of functions



Another view of multiclass logistic regression



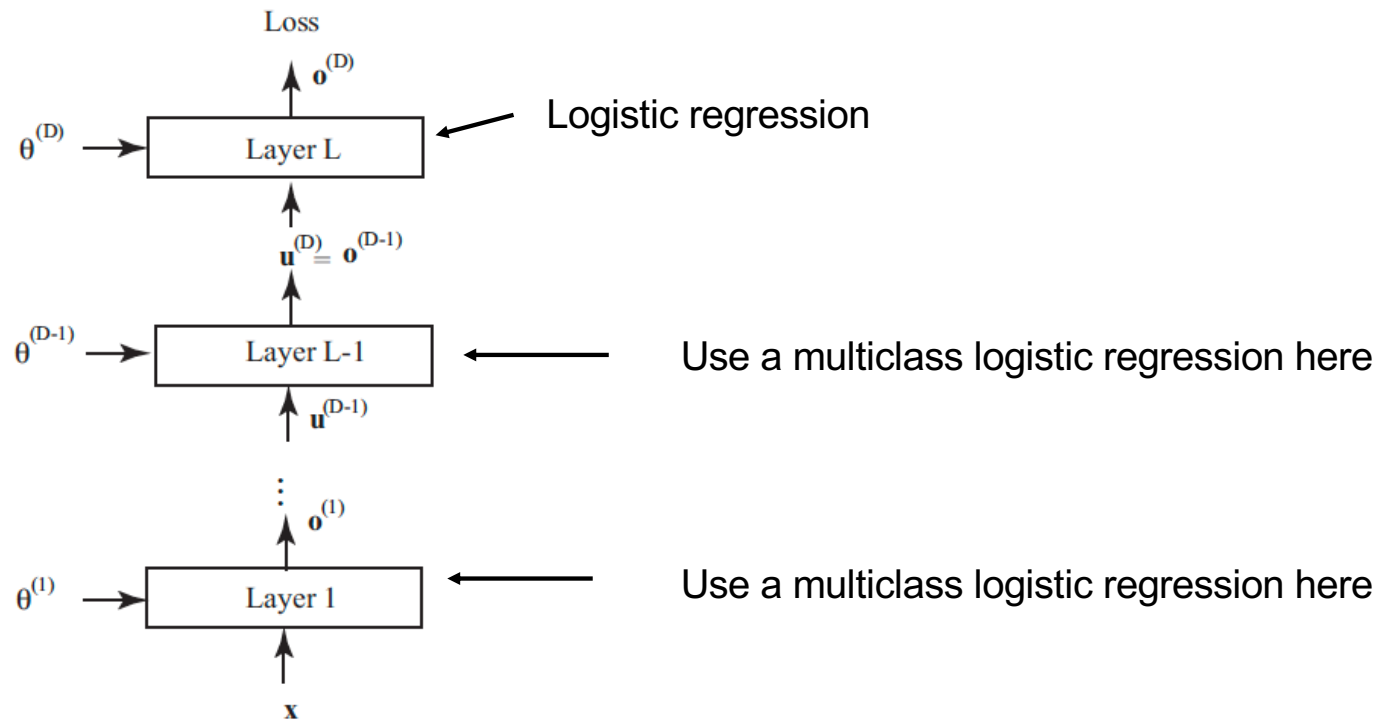
$$\text{softmax}(\mathbf{u}) = s(\mathbf{u}) = \left(\frac{1}{\sum_k e^{u_k}} \right) \begin{bmatrix} e^{u_1} \\ e^{u_2} \\ \dots \\ e^{u_c} \end{bmatrix}$$

Abstract as:

Maps a vector to a vector
Nonlinearity applied to linear map

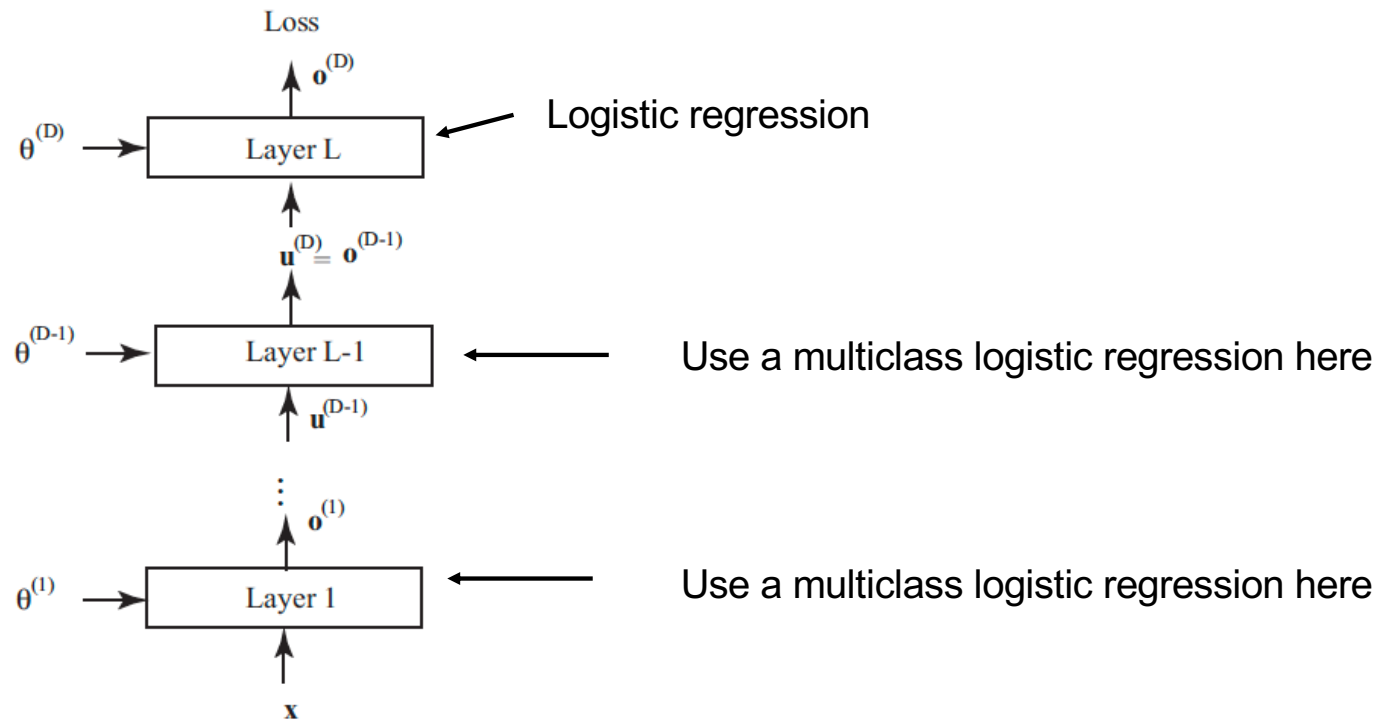
Often referred to as a “fully connected layer”

Making features using layers of functions

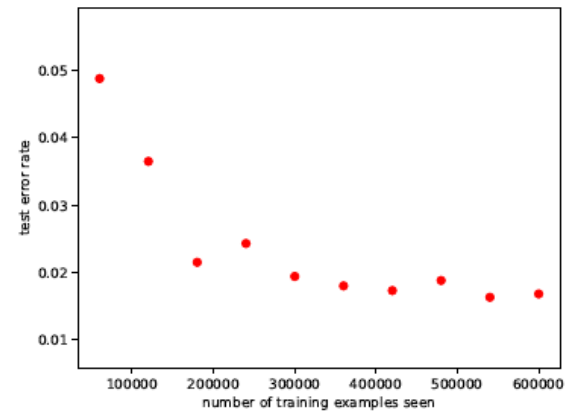
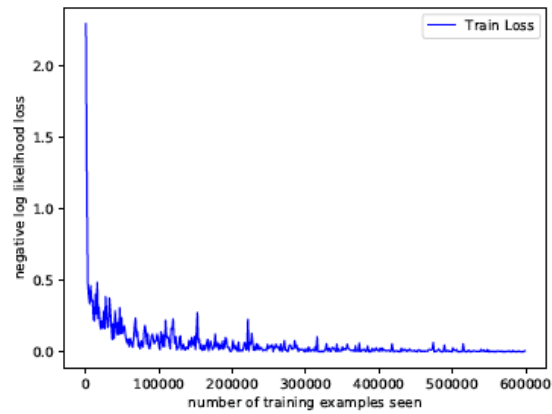


Training: gradient descent

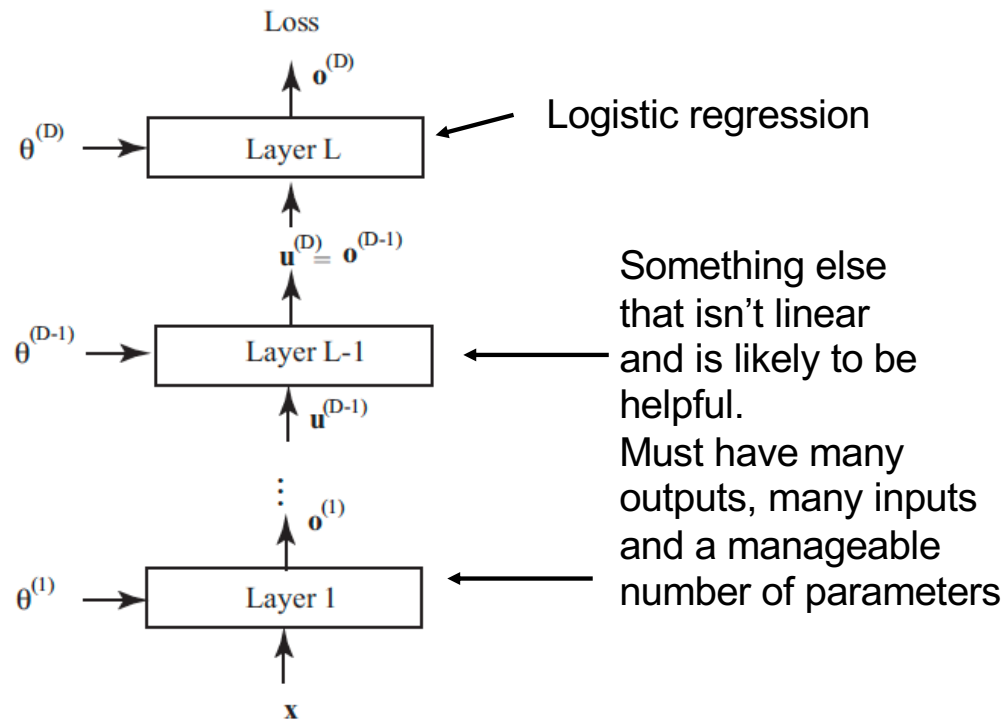
The gradient is obtained by a recursion. This follows from the chain rule. Various autograds will work it out for you.



Multiple feature constructing layers



Making features using layers of functions



Issue: if input is an image then FC layers will have far too many parameters.

Need something smaller.

What do we need to classify?

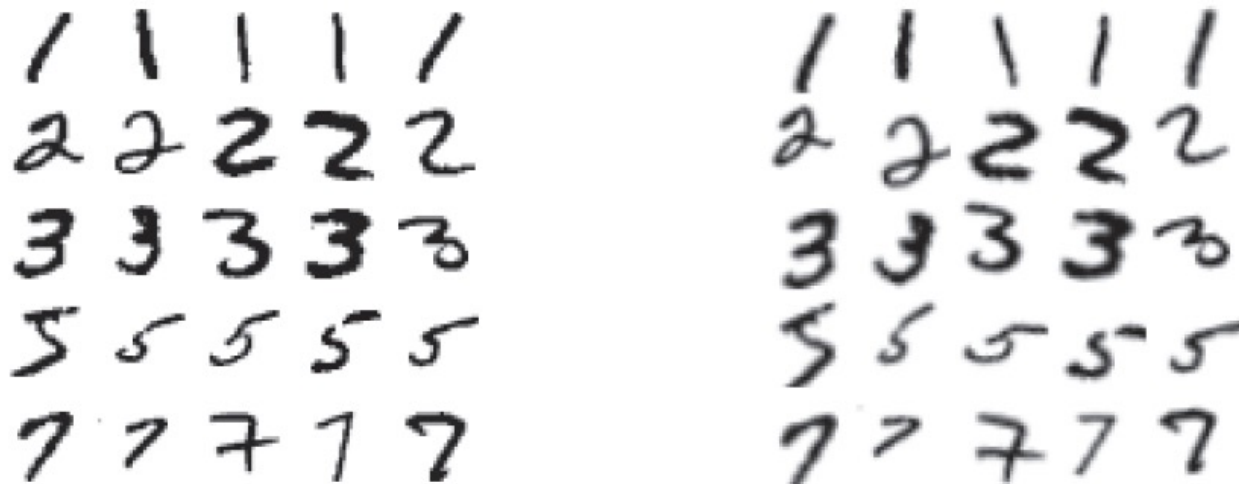


FIGURE 6.12: *On the left, a selection of digits from the MNIST dataset. Notice how images of the same digit can vary, which makes classifying the image demanding. It is quite usual that pictures of “the same thing” look quite different. On the right, digit images from MNIST that have been somewhat rotated and somewhat scaled, then cropped fit the standard size. Small rotations, small scales, and cropping really doesn’t affect the identity of the digit.*

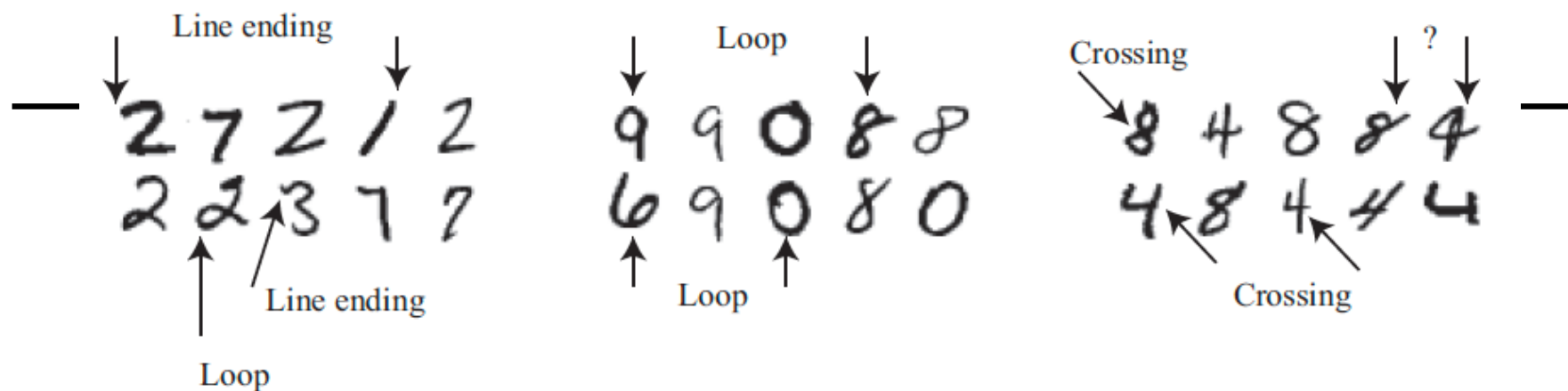


FIGURE 6.13: *Local patterns in images are quite informative. MNIST images, shown here, are simple images, so a small set of patterns is quite helpful. The relative location of patterns is also informative. So, for example, an eight has two loops, one above the other. All this suggests a key strategy: construct features that respond to patterns in small, localized neighborhoods; then other features that look at patterns of those features; then others that look at patterns of those, and so on. Each pattern (here line-endings, crossings and loops) has a range of appearances. For example, a line ending sometimes has a little wiggle as in the three. Loops can be big and open, or quite squashed. The list of patterns isn't comprehensive. The "?" shows patterns that I haven't named, but which appear to be useful. In turn, this suggests learning the patterns (and patterns of patterns; and so on) that are most useful for classification.*

Convolution

$$\mathcal{N} = \text{conv}(\mathcal{I}, \mathcal{W})$$

where

$$\mathcal{N}_{ij} = \sum_{uv} \mathcal{I}_{i-u, j-v} \mathcal{W}_{uv}.$$

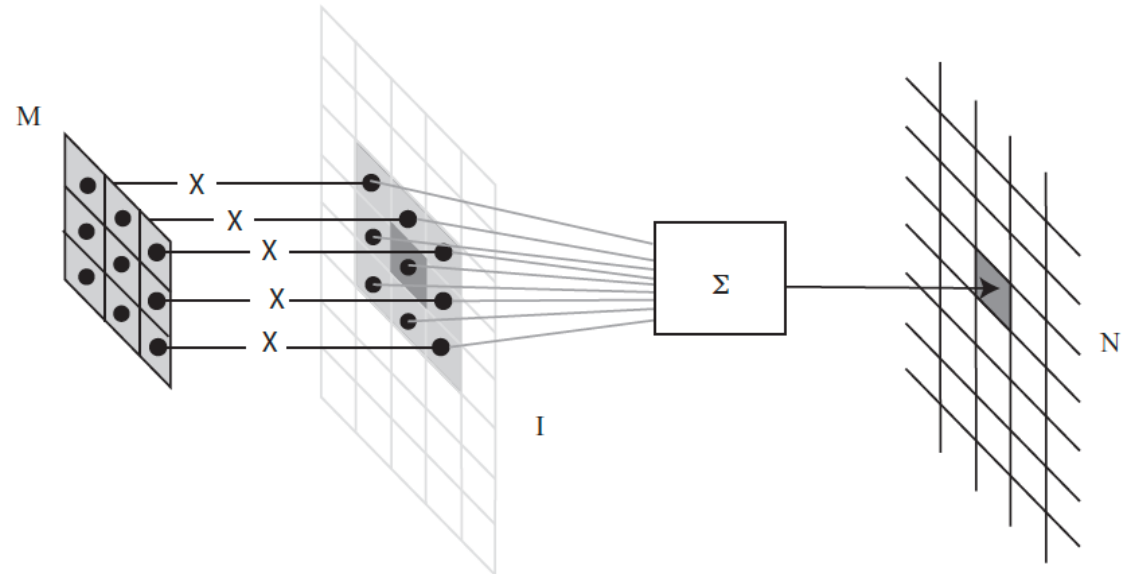


FIGURE 6.1: To compute the value of \mathcal{N} at some location, you shift a copy of \mathcal{M} to lie over that location in \mathcal{I} ; you multiply together the non-zero elements of \mathcal{M} and \mathcal{I} that lie on top of one another; and you sum the results.

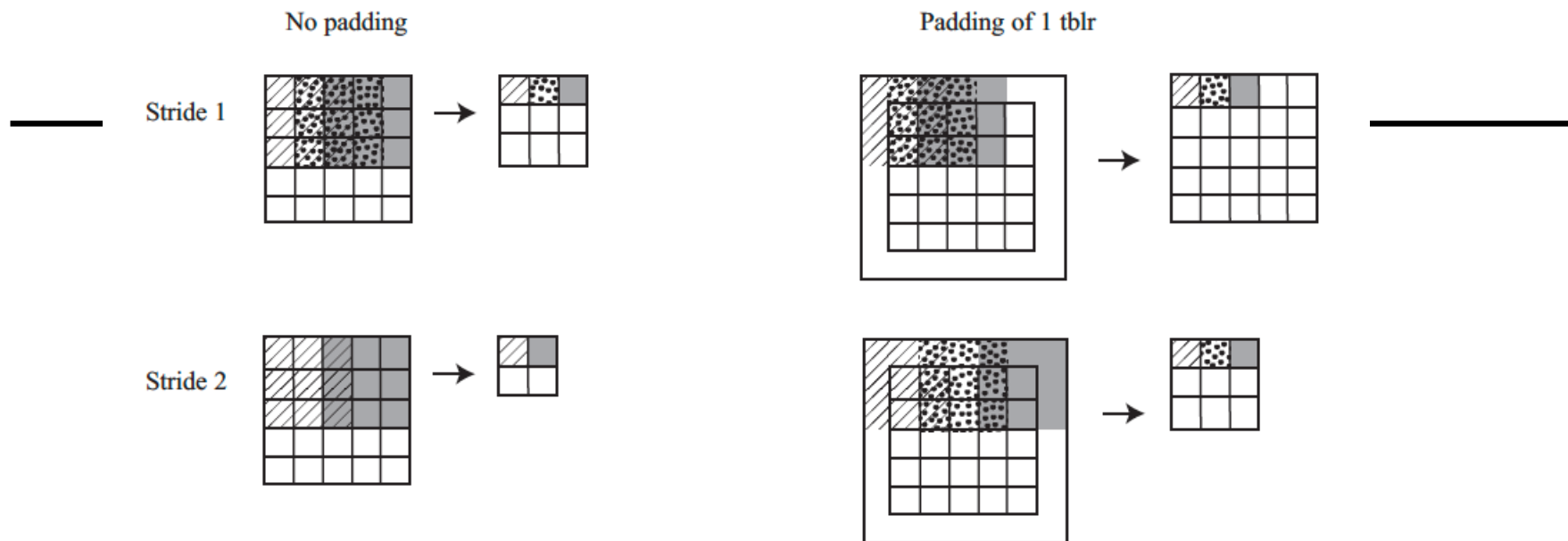


FIGURE 6.14: *The effects of stride and padding on conv. On the left, conv without padding accepts an \mathcal{I} , places a 3×3 \mathcal{M} on grid locations determined by the stride, then reports values for valid windows. When the stride is 1, a 5×5 \mathcal{I} becomes a 3×3 \mathcal{N} . When the stride is 2, a 5×5 \mathcal{I} becomes a 2×2 \mathcal{N} . The hatching and shading show the window used to compute the corresponding value in \mathcal{N} . On the right, conv with padding accepts an \mathcal{I} , pads it (in this case, by one row top and bottom, and one column left and right), places a 3×3 \mathcal{M} on grid locations in the padded result determined by the stride, then reports values for valid windows. When the stride is 1, a 5×5 \mathcal{I} becomes a 5×5 \mathcal{N} . When the stride is 2, a 5×5 \mathcal{I} becomes a 3×3 \mathcal{N} . The hatching and shading show the window used to compute the corresponding value in \mathcal{N} .*

Convolution

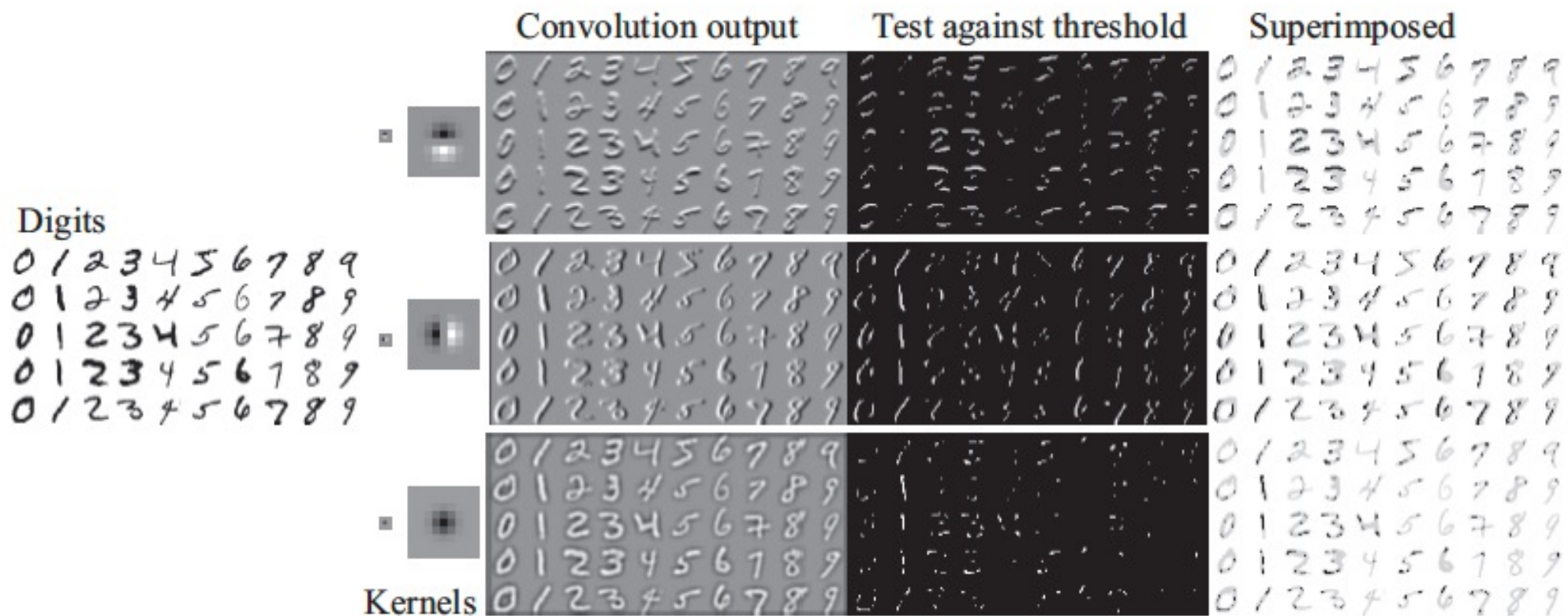
Think of this as a form of dot-product

- between kernel and window

Like dot-products

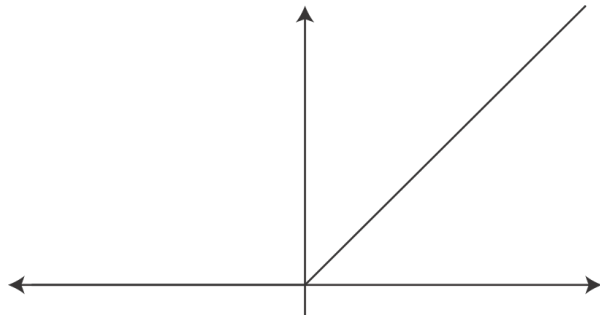
- largest value when kernel matches window
- smallest when kernel matches window with contrast reversal

-> SIMPLE PATTERN DETECTOR!



The ReLU

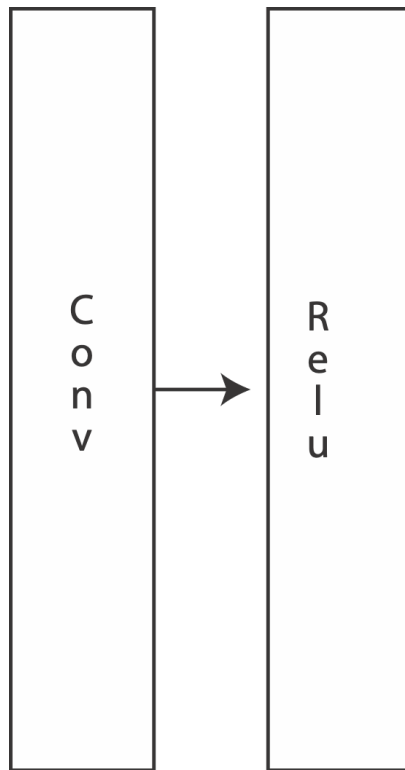
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



Issue: contrast reversal in pattern

If we apply a relu to a conv, then
we have a *signed* pattern detector

Basic pattern detector



Notice - not very many parameters
detects the same pattern at each location

Generalizing convolution

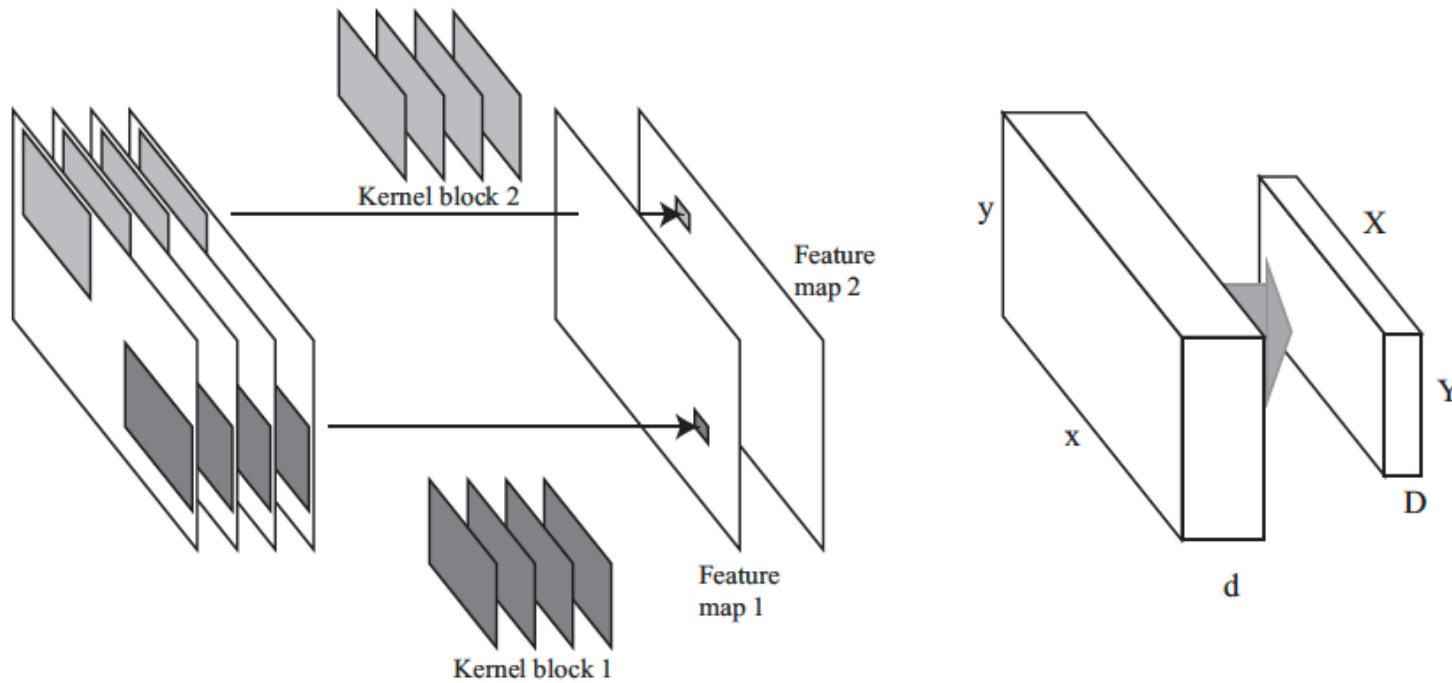
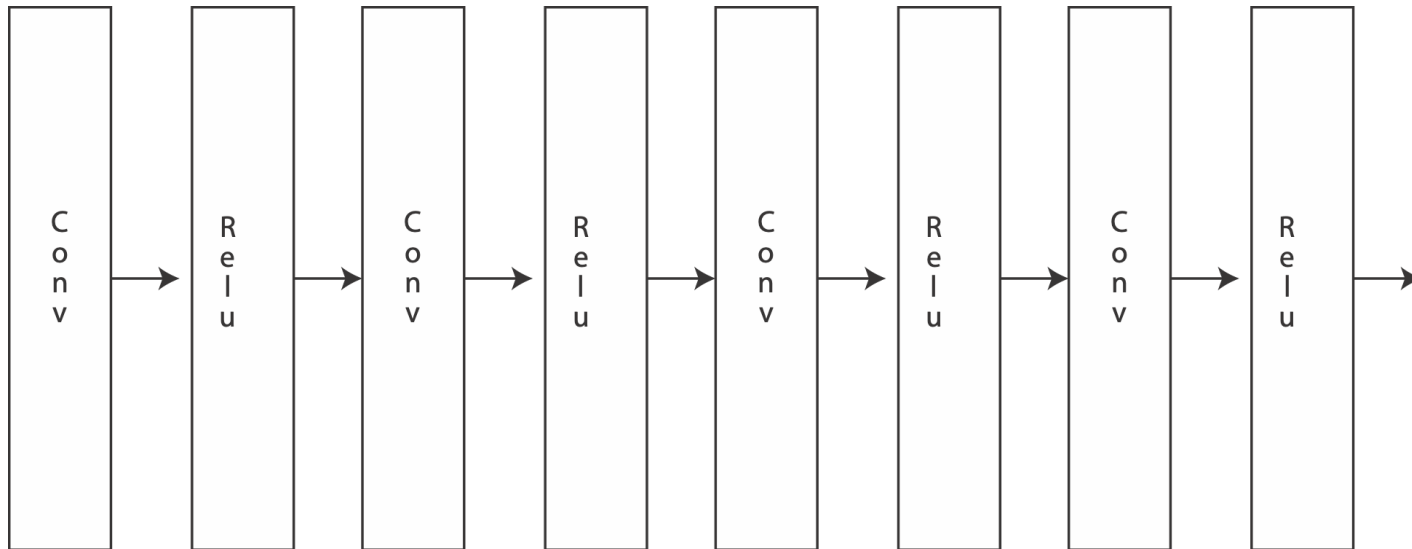


FIGURE 6.15: *On the left, two kernels (now 3D, as in the text) applied to a set of feature maps produce one new feature map per kernel, using the procedure of the text (the bias term isn't shown). Abstract this as a process that takes an $x \times y \times d$ block to an $X \times Y \times D$ block (as on the right).*

Patterns of patterns of patterns....

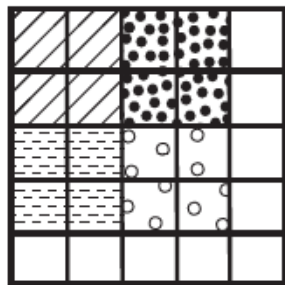


Stride and redundancy

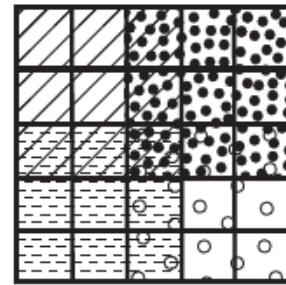
The *receptive field* of a location in a data block (or, equivalently, a unit) is the set of image pixels that affect the value of the location. Usually, all that matters is the size of the receptive field. The receptive field of a location in the first convolutional layer will be given by the kernel of that layer. Determining the receptive field for later layers requires some bookkeeping (among other things, you must account for any stride or pooling effects).

If you have several convolutional layers with stride 1, then each block of data has the same spatial dimensions. This tends to be a problem, because the pixels that feed a unit in the top layer will tend to have a large overlap with the pixels that feed the unit next to it. In turn, the values that the units take will be similar, and so there will be redundant information in the output block. It is usual to try and deal with this by making blocks get smaller. One natural strategy is to occasionally have a layer that has stride 2.

Pooling



Pooling 2x2s2



Pooling 3x3s2

FIGURE 7.1: In a pooling layer, pooling units compute a summary of their inputs, then pass it on. The most common case is 2x2, illustrated here on the left. We tile each feature map with 2x2 windows that do not overlap (so have stride 2). Pooling units compute a summary of the inputs (usually either the max or the average), then pass that on to the corresponding location in the corresponding feature map of the output block. As a result, the spatial dimensions of the output block will be about half those of the input block. On the right, the common alternative of pooling in overlapping 3x3 windows with stride 2.

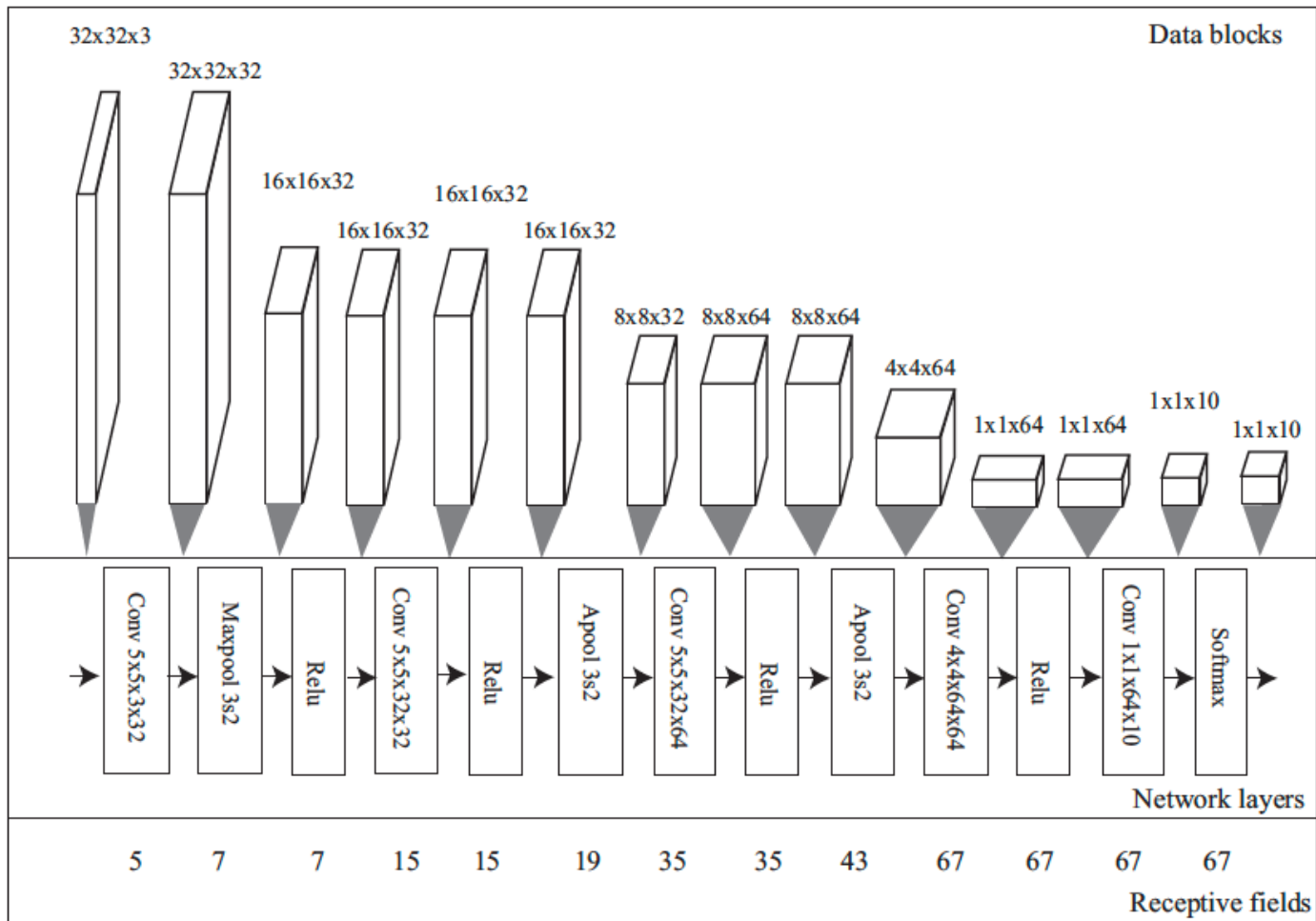


FIGURE 7.3: Three different representations of the simple network used to classify CIFAR-10 images for this example. Details in the text.



FIGURE 7.7: *Visualizing the patterns that the final stage ReLU's respond to for the simple CIFAR example. Each block of images shows the images that get the largest output for each of 10 ReLU's (the ReLU's were chosen at random from the 64 available in the top ReLU layer). Notice that these ReLU outputs don't correspond to class – these outputs go through a fully connected layer before classification – but each ReLU are clearly responds to a pattern, and different ReLU's respond more strongly to different patterns.*

Classification variants

Predict more labels with complex semantics

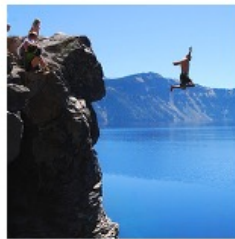
Predict a cost function from the image

- report the minimum

This allows

- Visual question answering
 - function accepts question, offered answers and takes min at best
- Writing sentences
 - choose sentence that minimizes cost

Situations



CLIPPING			
ROLE	VALUE	ROLE	VALUE
AGENT	MAN	AGENT	VET
SOURCE	SHEEP	SOURCE	DOG
TOOL	SHEARS	TOOL	CLIPPER
ITEM	WOOL	ITEM	CLAW
PLACE	FIELD	PLACE	ROOM

JUMPING			
ROLE	VALUE	ROLE	VALUE
AGENT	BOY	AGENT	BEAR
SOURCE	CLIFF	SOURCE	ICEBERG
OBSTACLE	-	OBSTACLE	WATER
DESTINATION	WATER	DESTINATION	ICEBERG
PLACE	LAKE	PLACE	OUTDOOR

SPRAYING			
ROLE	VALUE	ROLE	VALUE
AGENT	MAN	AGENT	FIREMAN
SOURCE	SPRAY CAN	SOURCE	HOSE
SUBSTANCE	PAINT	SUBSTANCE	WATER
DESTINATION	WALL	DESTINATION	FIRE
PLACE	ALLEYWAY	PLACE	OUTSIDE

Yatskar+Zettlemoyer+Farhadi 2016

Visual Question Answering







		
Q. What is the cat wearing? A. Hat	Q. What is the weather like? A. Rainy	Q. What surface is this? A. Clay
		
Q. What is the weather like? A. Sunny	Q. What color is the cat's eyes? A. Green	Q. What toppings are on the pizza? A. Mushrooms

Figure 1.22 Visual question answering systems produce natural language answers to questions about images. It is difficult for a VQA system to hide ignorance in the way that a captioning system can. Here the system is producing quite sensible answers to rather difficult questions about the image (answers are typically chosen from a multiple choice set). Figure courtesy of Devi Parikh, produced by a system described in “Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering” by Goyal, Khot, Summers-Stay, Batra, and Parikh and published in CVPR 2017.

doesn't always work...



Q. How many holes are in the pizza?
A. 8



Q. What color is the front right leg?
A. Brown



Q. What letter is on the racket?
A. w



Q. Why is the sign bent?
A. It's not

Figure 1.23 Because it is difficult for a VQA system to hide ignorance in the way that a captioning system can, the mistakes can be informative and highlight how difficult it is to produce accurate visual representations. For example, the system is guessing about the number of holes in a pizza, because it doesn't understand the conventions about what holes are worth talking about, and it has real difficulty counting. Similarly, the system is describing the cat's leg as brown because it can't localize the leg properly. Figure courtesy of Devi Parikh, produced by a system described in "Making the V in VQA Matter: Elevating the Role of Image Understanding in Visual Question Answering" by Goyal, Khot, Summers-Stay, Batra, and Parikh and published in CVPR 2017.

Sentence generation

Decode features into sentence (with LSTM, etc)

- essentially classification with funky taxonomy



A baby eating a piece of food in his mouth.



A young boy eating a piece of cake

Aneja et al, 2018

doesn't always work...

And scoring system is easily subverted!



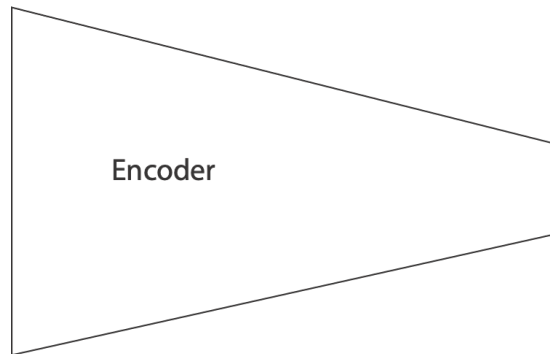
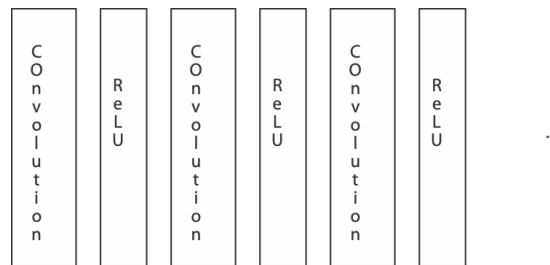
A small bird is perched on a branch



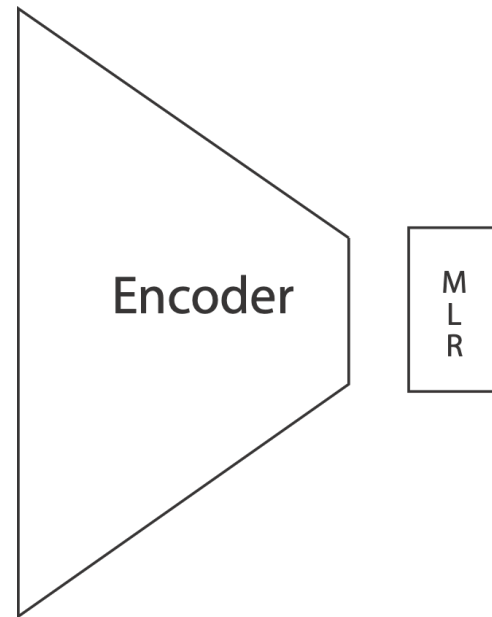
A small brown bear is sitting
in the grass

Aneja et al, 2018

Encoders



A classifier



Can train encoder *without labels*

Encoder yields embedding of the image

Exploit data augmentation

- take image and
 - crop+resize; adjust colormap; etc

Strategy: Contrastive learning

- Adjust embedding so that
 - A and Augment(A) should be close
 - A and B should be far

Then multiclass logistic regression when you have labels

SOA - rough summary

Very high accuracy with 1000's of classes

- Using
 - very deep residual networks
- clever trick to improve training convergence
 - alternative feature construction methods

Classification wrt

- Object present
- Scene type
- Etc

Challenges

- tough with little training data (but encoders are somewhat interchangeable)
- change in dataset presents problems

Open questions

Rules of machine learning

- It all works when test data is “like” training data
 - IID samples from the same distribution
- All bets are off otherwise; very little theoretical support

Practice in computer vision

- It is tough to tell when this condition occurs
- Mostly, it isn't imposed
 - instead, we say that there was a generalization failure when classifier doesn't work

Q: Why don't we get in trouble when we break the rules?

Q: Tell when datasets A, B are “compatible”

- In a crisp, formal way (rather than try and see)

Exploiting registration and classification

Use a classifier to tell:

- how far to the next intersection?
- what is it like?
- is there a bike lane?
- etc.



Pred = 18.5 m

Road layout maps

Potential cues

- streetview
- openmaps

Partially supervised cues

Open Street Maps (OSM)

Map data: OpenStreetMap is an open-source mapping project covering over 21 million miles of road. Unlike proprietary maps, the underlying road coordinates and metadata are freely available for download. Accuracy and overlap with Google Maps is very high, though some inevitable noise is present as information is contributed by individual volunteers or automatically extracted from users' GPS trajectories. For example, roads in smaller cities may lack detailed annotations (e.g., the number of lanes may be unmarked). These inconsistencies result in varying-sized subsets of the data being applicable for different attributes.



Fig. 3. Intersection detection heatmap. Images are cropped from test set GSV panoramas in the direction of travel indicated by the black arrow. The probabilities of “approaching” an intersection output by the trained ConvNet are overlaid on the road. (The images are from the ground level road, not the bridge.)

Seff+Xiao

Partially supervised cues

Google street view

Image collection: Google Street View contains panoramic images of street scenes covering 5 million miles of road across 3,000 cities. Each panorama has a corresponding metadata file storing the panorama's unique "pano_id", geographic location, azimuth orientation, and the pano_ids of adjacent panoramas. Beginning from an initial seed panorama, we collect street view images by running a bread-first search, downloading each image and its associated metadata along the way. Thus far, our dataset contains one million GSV panoramas from the San Francisco Bay Area. GSV panoramas can be downloaded at several different resolutions (marked as "zoom levels"). Finding the higher zoom levels unnecessary for our purposes, we elected to download at a zoom level of 1, where each panorama has a size of 832×416 pixels.

Seff+Xiao

Labelling - I

Match panoramas to roads

- panorama center location, orientation is known
- (essentially) project to plane
- thresholded nearest neighbor to road center polyline
 - thresholding removes panoramas inside buildings, etc.
- some noise
 - under bridges, etc.

Annotations

- Intersections
- Drivable heading
- Heading angle
- Bike lane
- Speed limit, wrong way, etc.



Pred = 0.1 m
True = 1.9 m



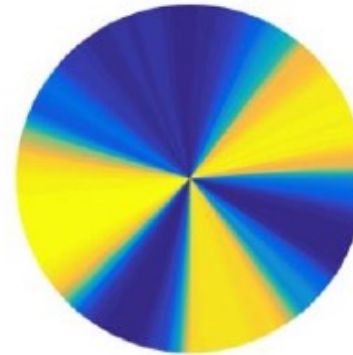
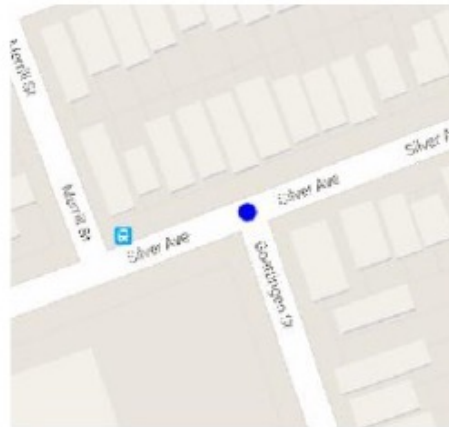
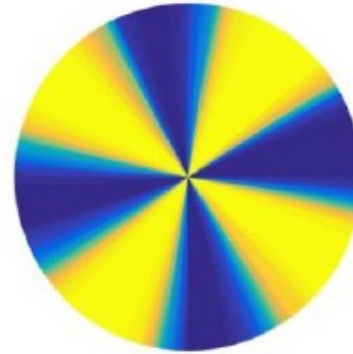
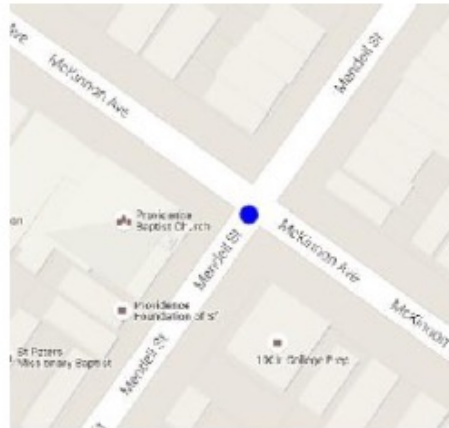
Pred = 18.5 m
True = 19.2 m



Pred = 22.9 m
True = 22.4 m

Fig. 4. Distance to intersection estimation. For images within 30 m of true intersections, our model is trained to estimate the distance from the host car to the center of the intersection across a variety of road types.

Seff+Xiao



Seff+Xiao

Fig. 5. Intersection topology is one of several attributes our model learns to infer from an input GSV panorama. The blue circles on the Google Maps extracts to the left show the locations of the input panoramas. The pie charts display the probabilities output by the trained ConvNet of each heading angle being on a driveable path (see Figure 3 for colormap legend).



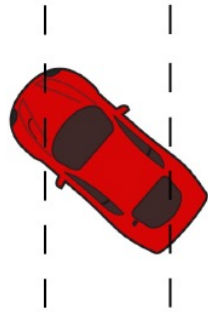
$p(\text{driveable}) = 0.002$

$p(\text{driveable}) = 0.714$

$p(\text{driveable}) = 0.998$

Fig. 6. Driveable headings. A ConvNet is trained to distinguish between non-drivable headings (left) and drivable headings aligned with the road (right). The ConvNet weakly classifies the middle example as drivable because the host car's heading is facing the alleyway between the buildings.

Seff+Xiao



Pred = -52.7°
True = -49.1°



Pred = -18.3°
True = -20.5°



Pred = 31.6°
True = 32.7°

Seff+Xiao

Fig. 7. Heading angle regression. The network learns to predict the relative angle between the street and host vehicle heading given a single image cropped from a GSV panorama. Below each GSV image, the graphic visualizes the ground truth heading angle.



$p(\text{bike lane}) = 0.043$



$p(\text{bike lane}) = 0.604$



$p(\text{bike lane}) = 0.988$

Fig. 8. The ConvNet learns to detect bike lanes adjacent to the vehicle. The GSV images are arranged from left to right in increasing order of probability output by the ConvNet of a bike lane being present (ground truth labels from left to right are negative, negative, positive). The middle example contains a taxi lane, resulting in a weak false positive.

Seff+Xiao



Pred = 26.1 mph
True = 30 mph



Pred = 30.0 mph
True = 50 mph



Pred = 54.3 mph
True = 50 mph

Fig. 9. Speed limit regression. The network learns to predict speed limits given a GSV image of road scene. The model significantly underestimates the speed limit in the middle example as this type of two-way road with a single lane in each direction would generally not have a speed limit as high as 50 mph.

Seff+Xiao



$p(\text{one-way}) = 0.207$



$p(\text{one-way}) = 0.226$



$p(\text{one-way}) = 0.848$

Fig. 10. One-way vs. two-way road classification. The probability output by the ConvNet of each GSV scene being on a one-way road is shown. From left to right the ground truth labels are two-way, two-way, and one-way. The image on the left is correctly classified as two-way despite the absence of the signature double yellow lines.

Seff+Xiao



$p(\text{wrong way}) = 0.555$ $p(\text{wrong way}) = 0.042$ $p(\text{wrong way}) = 0.729$

Fig. 11. Wrong way detection. The probability output by the ConvNet of each GSV image facing the wrong way on the road is displayed. From left to right the ground truth labels are wrong way, right way, and right way. For two-way roads with no lane markings (left), this is an especially difficult problem as it amounts to estimating the horizontal position of the host car. The problem can also be quite ill-defined if there are no context clues as is the case with the rightmost image.

Seff+Xiao



Pred = 2
True = 1



Pred = 2
True = 2



Pred = 3
True = 2

Fig. 12. Number of lanes estimation. The predicted and true number of lanes for three roads are displayed along with the corresponding GSV images. For streets without clearly visible lane markings (left), this is especially challenging. Although the ground truth for the rightmost image is two lanes, there is a third lane that merges just ahead.

Seff+Xiao

At this point

I can tell from an image whether

- I'm pointing in the right direction
- going the right way
- facing an intersection
- available turns, etc.
- what and where street signs are
- ...

Can I build a reliable controller?

BIG GOOD QUESTIONS

Mashup of openmaps and street view

- it could predict drivable directions, steering directions, lanes, signs, etc.

Q: WHY IS THIS NOT DRIVING AROUND NOW?

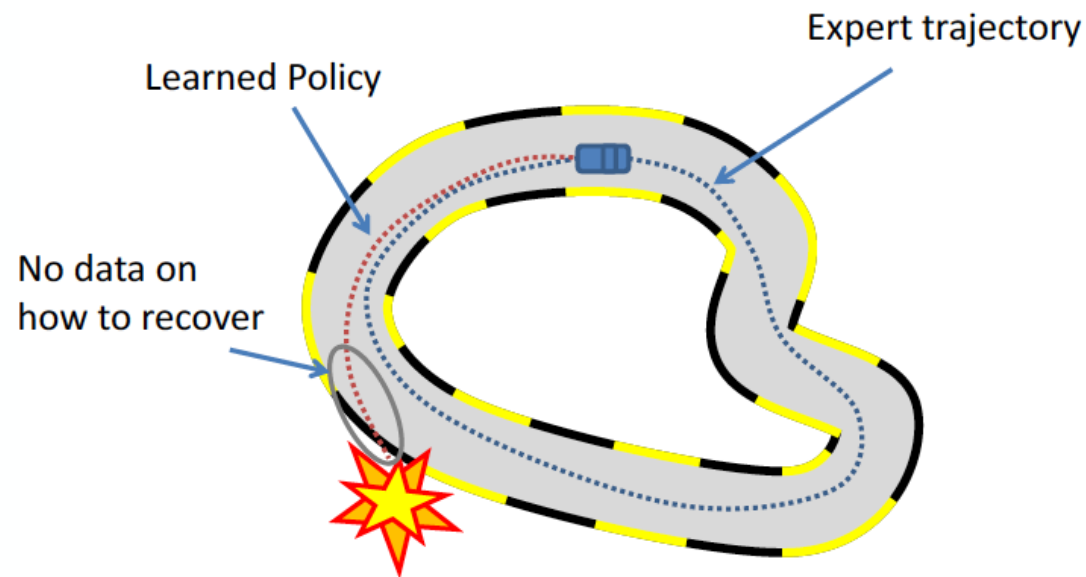
- A: (pretty obviously) because it doesn't work

Q: WHY NOT?

- A: interesting

Data Distribution Mismatch!

$$p_{\pi^*}(o_t) \neq p_{\pi_\theta}(o_t)$$

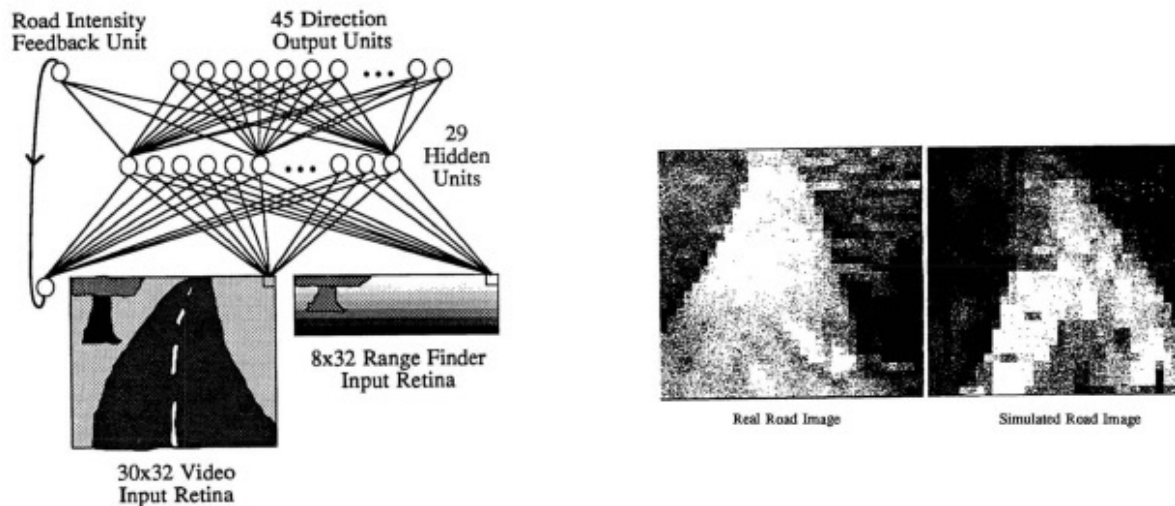


Fragkiadaki, ND

Imitation learning

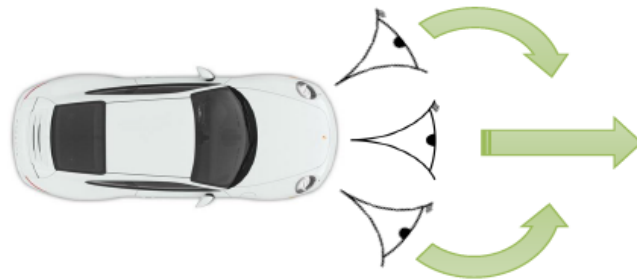
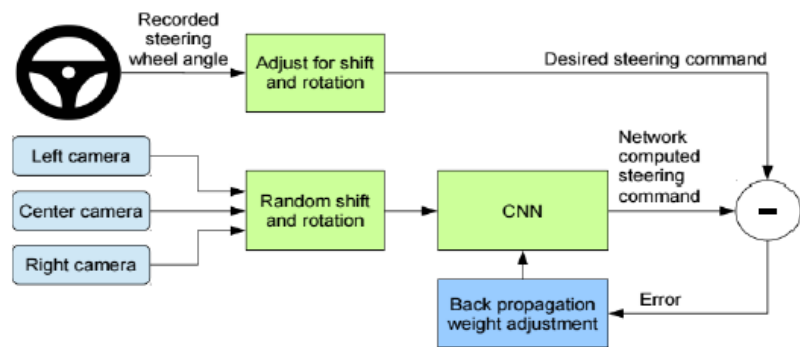
Approaches

- Imitation learning:
 - Train a policy that does “the same thing” as an expert



“In addition, the network must not solely be shown examples of accurate driving, but also how to recover (i.e. return to the road center) once a mistake has been made. Partial initial training on a variety of simulated road images should help eliminate these difficulties and facilitate better performance.” ALVINN: An autonomous Land vehicle in a neural Network, Pomerleau 1989

Demonstration Augmentation: NVIDIA 2016



Additional, left and right cameras with automatic ground-truth labels to recover from mistakes

“DAVE-2 was inspired by the pioneering work of Pomerleau [6] who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system. Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. ...”

End-to-End Learning for Self-Driving Cars, Bojarski et al. 2016
Fragkiadaki, ND

Regression

We must make image-like things from images

Running example:

- depth map from image

A depth map has the depth to closest surface at every pixel

- it is the same size as the image

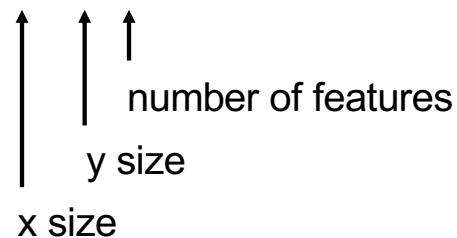
Recall feature construction

Apply “pattern detector” to image

- another to the result
- another to the result
- etc
- occasionally reducing the spatial size of the block of data representing patterns to control redundancy

The resulting block of data is spatially small

- eg in the (very simple) CIFAR network,
- 32x32x3-> 4x4x64



We could now predict an image by..

Take pattern detector results and decode into pattern

- “pattern producer”

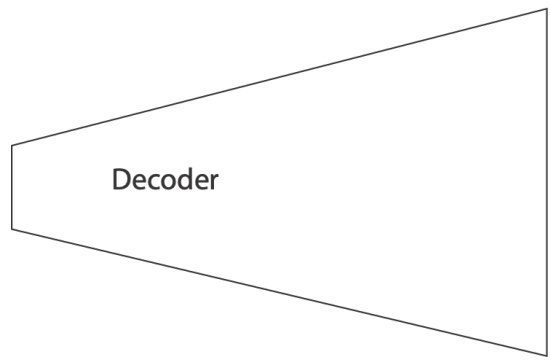
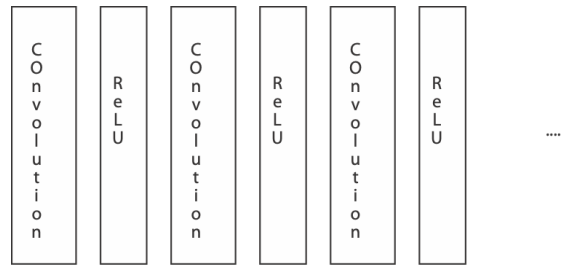
Apply pattern producer to feature block

- another to result
- another to result
- occasionally upsampling as required

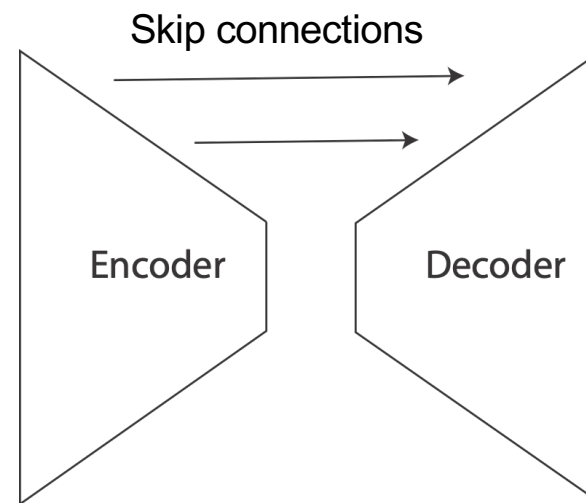
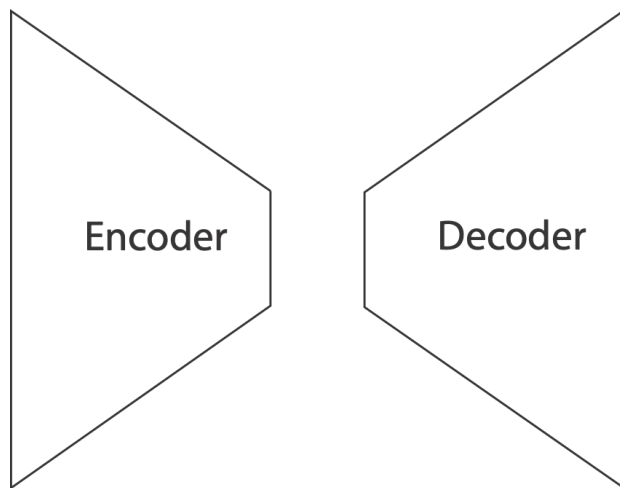
Pattern producer is itself a convolution

- a feature location detects a particular pattern
- scale that pattern by the strength of the response, and place down
- sum at overlap
- => convolution (sometimes called transpose convolution, inverse convolution)

Decoders



Regression



Sometimes known as a U-net

Regression

Train with pairs (image, depth)

- Loss
 - Squared error +abs value of error+other terms as required

Very powerful general recipe

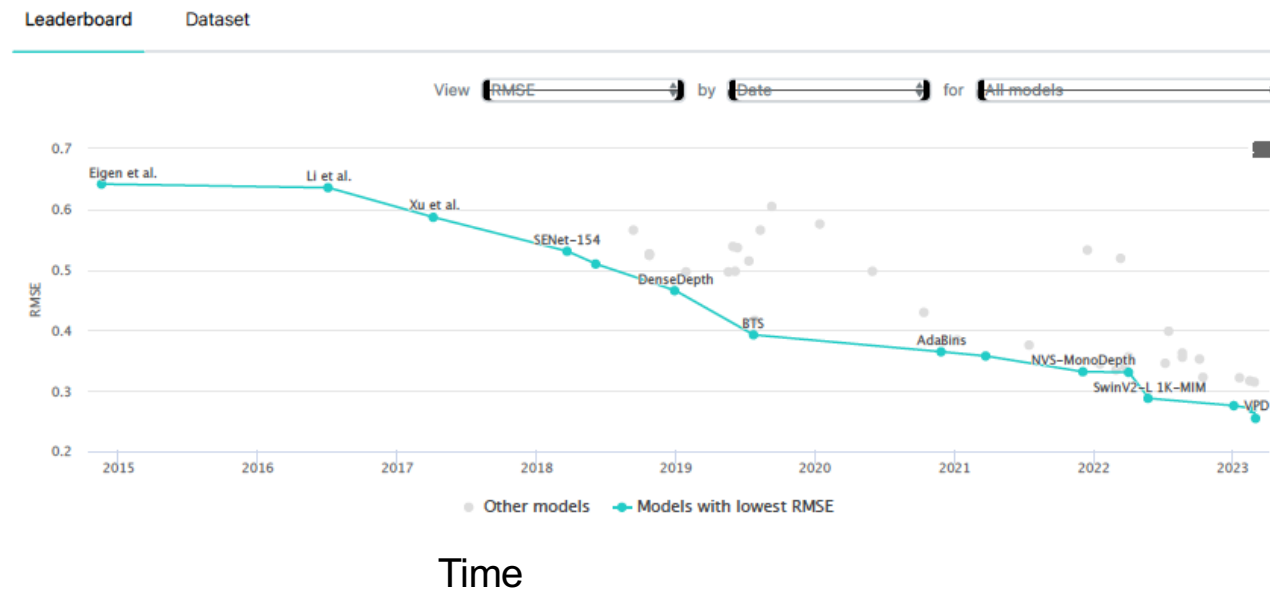
- depth from image
- normal from image
- superresolution
- Semantic segmentation (classification-like, regression-like)

Variants

- more sophisticated encoder

Depth SOTA (early 23)

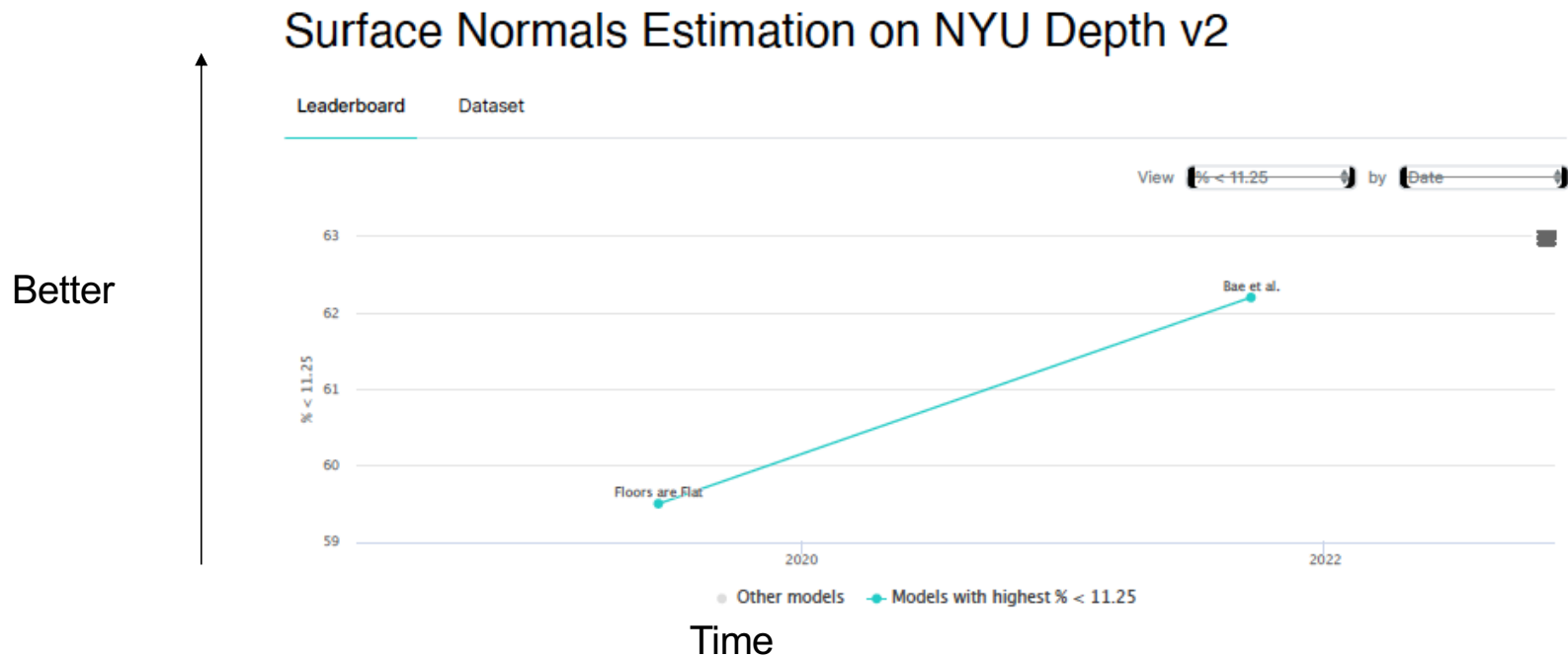
Monocular Depth Estimation on NYU-Depth V2



Better

<https://paperswithcode.com/sota/monocular-depth-estimation-on-nyu-depth-v2>

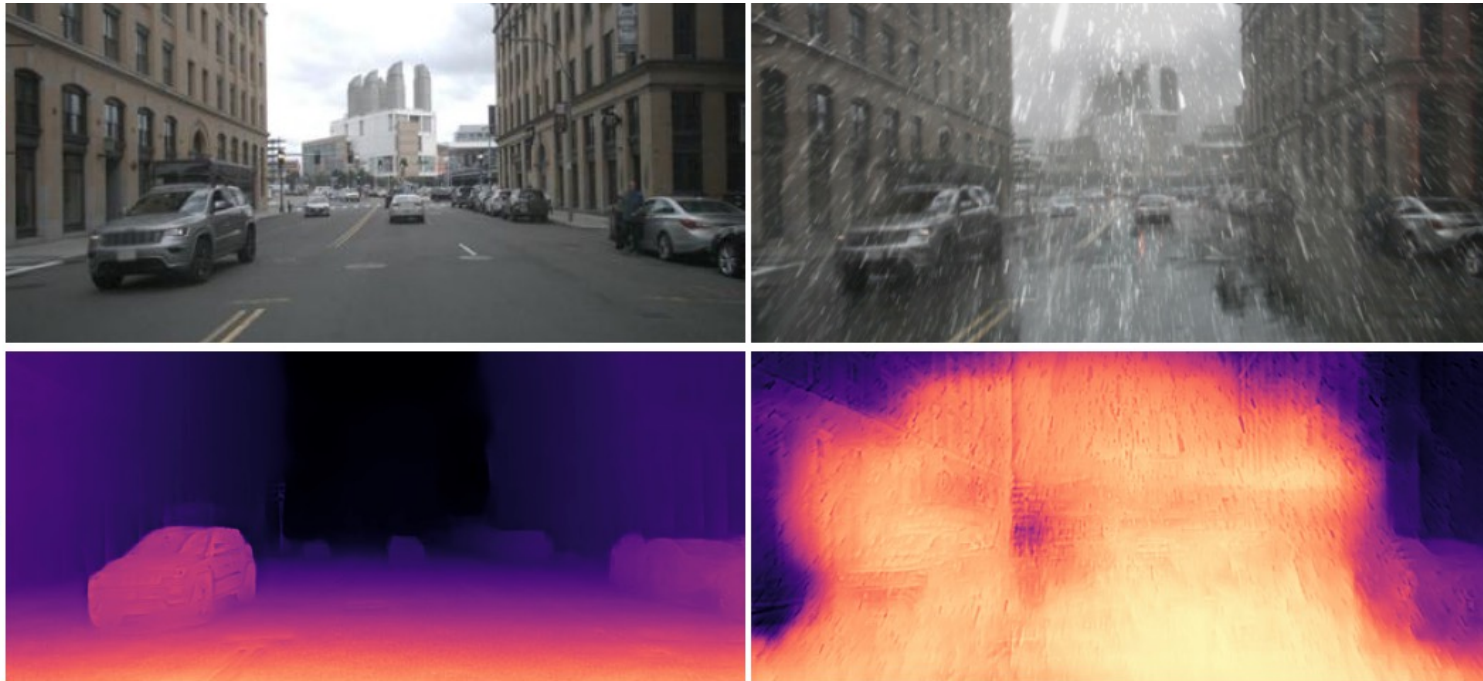
Normal SOTA (early 23)



<https://paperswithcode.com/sota/surface-normals-estimation-on-nyu-depth-v2-1>

Ghost(s) at the party

Depth estimation [25]



Tremblay et al 20

Ghost(s) at the party

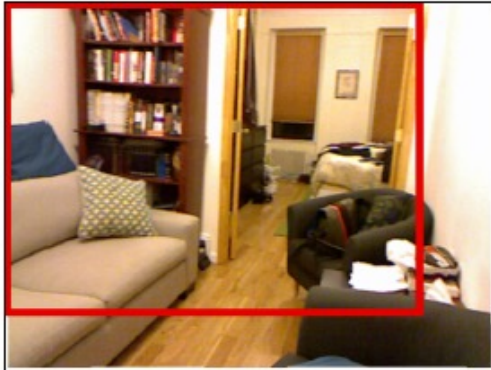
Semantic segmentation [65]



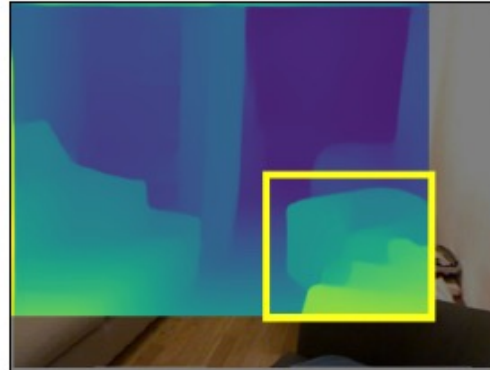
Tremblay et al 20

Further spectral manifestations

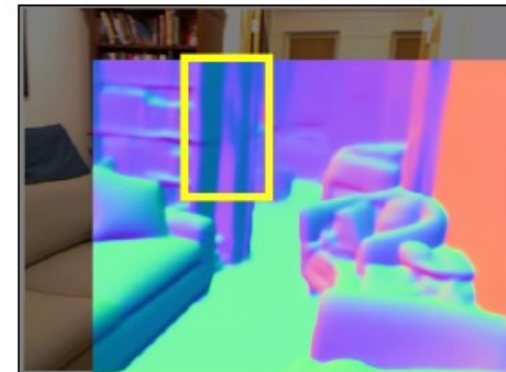
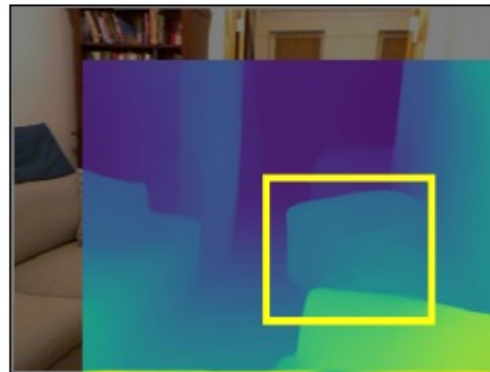
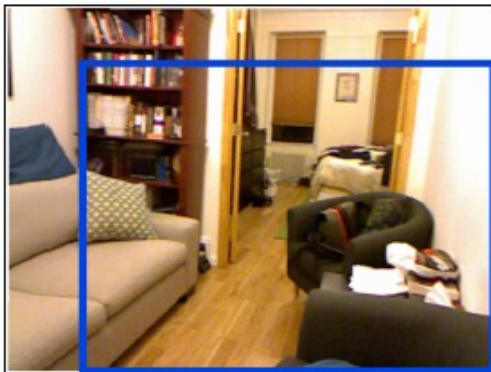
Image Crops



Depth



Surface Normal

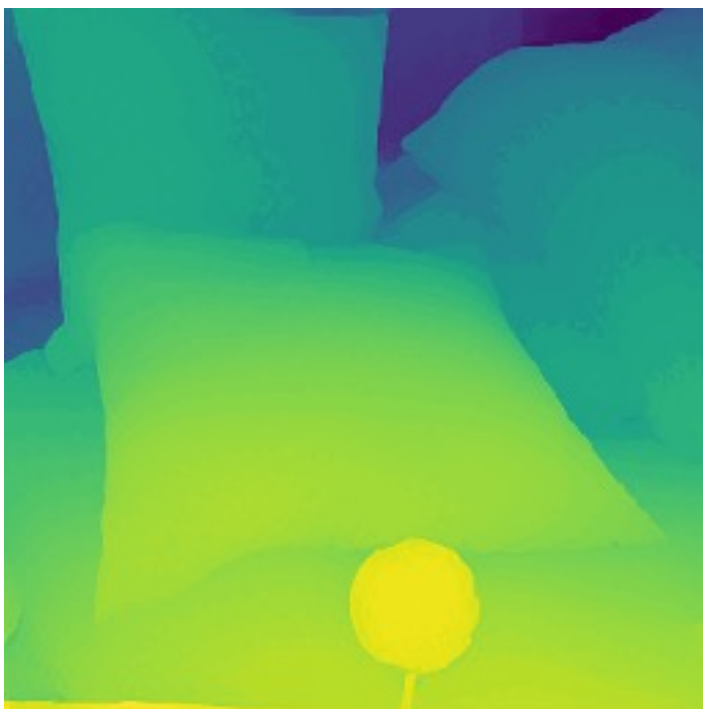


And a scary movie...

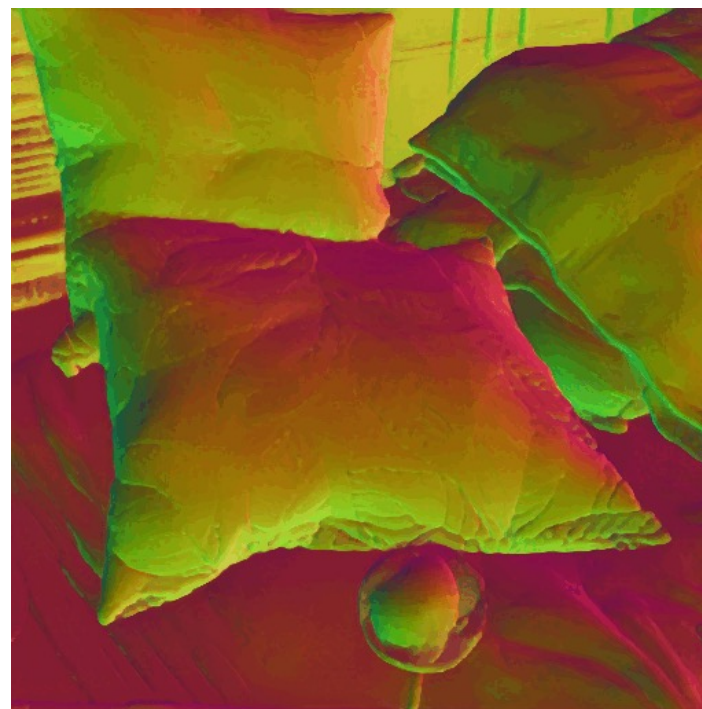
Exa



Depth (omnimap, current best depth est)



Normal (omnimap, current best normal est)



Classification vs detection

Classification:

- there is an X in this image
 - what

Detection:

- there is an X HERE in this image
 - what AND where

Key issues

- how to specify where
- relationship between what and where
 - efficiency, etc
- evaluation
 - surprisingly fiddly

Two threads

Localize then classify

- find boxes that likely contain objects
- decide what is in the box

YOLO: Localize while classifying

- in parallel, score
 - boxes for “goodness of box”
 - boxes for “what is in it”
- combine

Start simple

Where = axis aligned box

- **Decide on a window shape:** this is easy. There are two possibilities: a box, or something else. Boxes are easy to represent, and are used for almost all practical detectors. The alternative — some form of mask that cuts the object out of the image — is hardly ever used, because it is hard to represent.
- **Build a classifier for windows:** this is easy – we've seen multiple constructions for image classifiers.
- **Decide which windows to look at:** this turns out to be an interesting problem. Searching all windows isn't efficient.
- **Choose which windows with high classifier scores to report:** this is interesting, too, because windows will overlap, and we don't want to report the same object multiple times in slightly different windows.
- **Report the precise locations of all faces using these windows:** this is also interesting. It turns out our window is likely not the best available, and we can improve it after deciding it contains a face.

Which window

Astonishing fact

- Easy to tell whether a region is likely to be an object
 - even if you don't know what object (Endres+Hoiem, 10; Uijlings et al 12)
 - if it's an object
- there's contrast with surroundings in texture, etc
 - if not
- often neighbor region is similar

Selective Search

Construct hierarchy of image regions

- using a hierarchical segmenter

Rank regions using a learned score

Make boxes out of high-ranking regions

Selective search pipeline

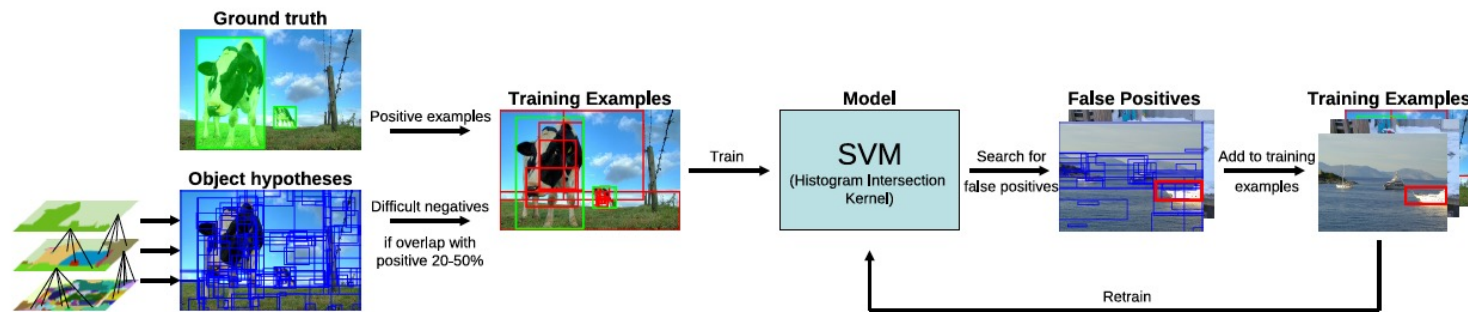


Figure 3: The training procedure of our object recognition pipeline. As positive learning examples we use the ground truth. As negatives we use examples that have a 20-50% overlap with the positive examples. We iteratively add hard negatives using a retraining phase.

You need to search at multiple scales



Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

Simplest detector

Use selective search to propose boxes

Check boxes with classifier

BUT

- boxes likely overlap - non-maximum suppression
- boxes likely in poor location - bounding box regression

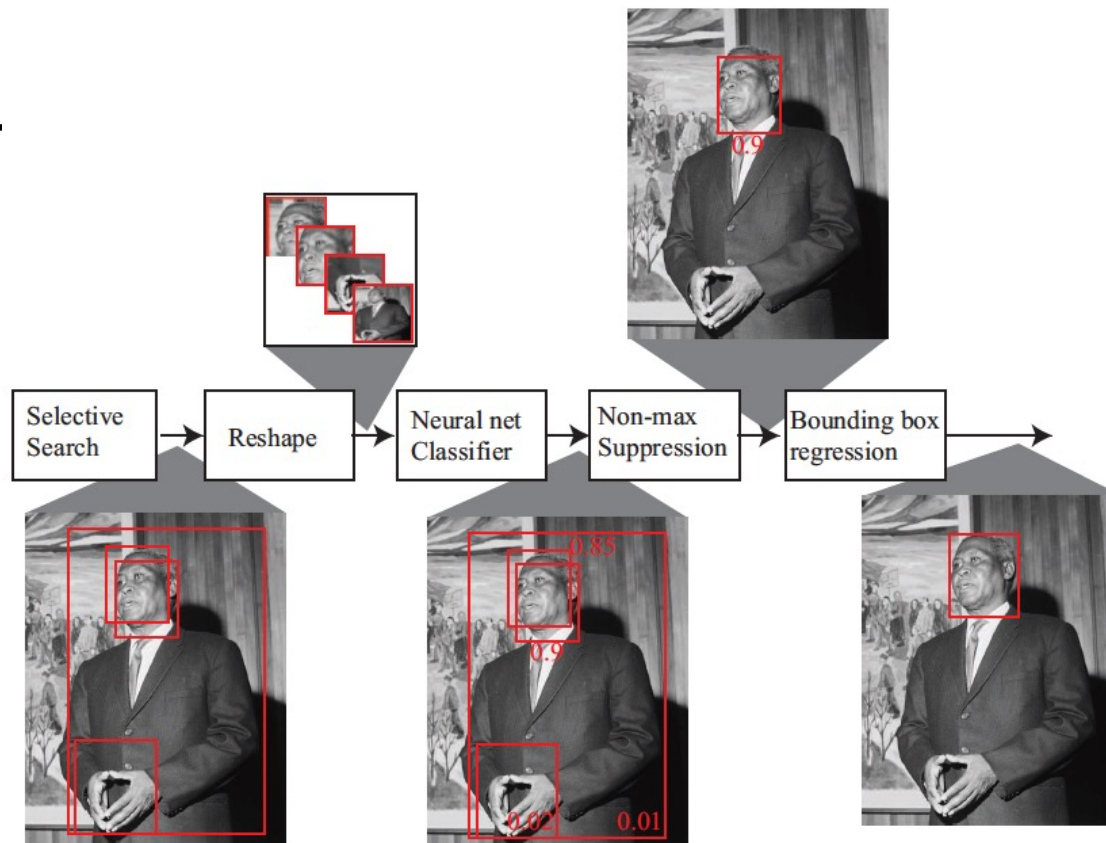


FIGURE 18.6: A schematic picture of how R-CNN works. A picture of Inkosi Albert Luthuli is fed in to selective search, which proposes possible boxes; these are cut out of the image, and reshaped to fixed size; the boxes are classified (scores next to each box); non-maximum suppression finds high scoring boxes and suppresses nearby high scoring boxes (so his face isn't found twice); and finally bounding box regression adjusts the corners of the box to get the best fit using the features inside the box.

Non maximum suppression

Deciding which windows to report presents minor but important problems. Assume you look at 32×32 windows with a stride of 1. Then there will be many windows that overlap the object fairly tightly, and these should have quite similar scores. Just thresholding the value of the score will mean that we report many instances of the same object in about the same place, which is unhelpful. If the stride is large, no window may properly overlap the object and it might be missed. Instead, most methods adopt variants of a greedy algorithm usually called **non-maximum suppression**. First, build a sorted list of all windows whose score is over threshold. Now repeat until the list is empty: choose the window with highest score, and accept it as containing an object; now remove all windows with large enough overlap on the object window.

Bounding box regression

Deciding precisely where the object is also presents minor but important problems. Assume we have a window that has a high score, and has passed through non-maximum suppression. The procedure that generated the window does not do a detailed assessment of all pixels in the window (otherwise we wouldn't have needed the classifier), so this window likely does not represent the best localization of the object. A better estimate can be obtained by predicting a new bounding box using a feature representation for the pixels in the current box. It's natural to use the feature representation computed by the classifier for this bounding box regression step.

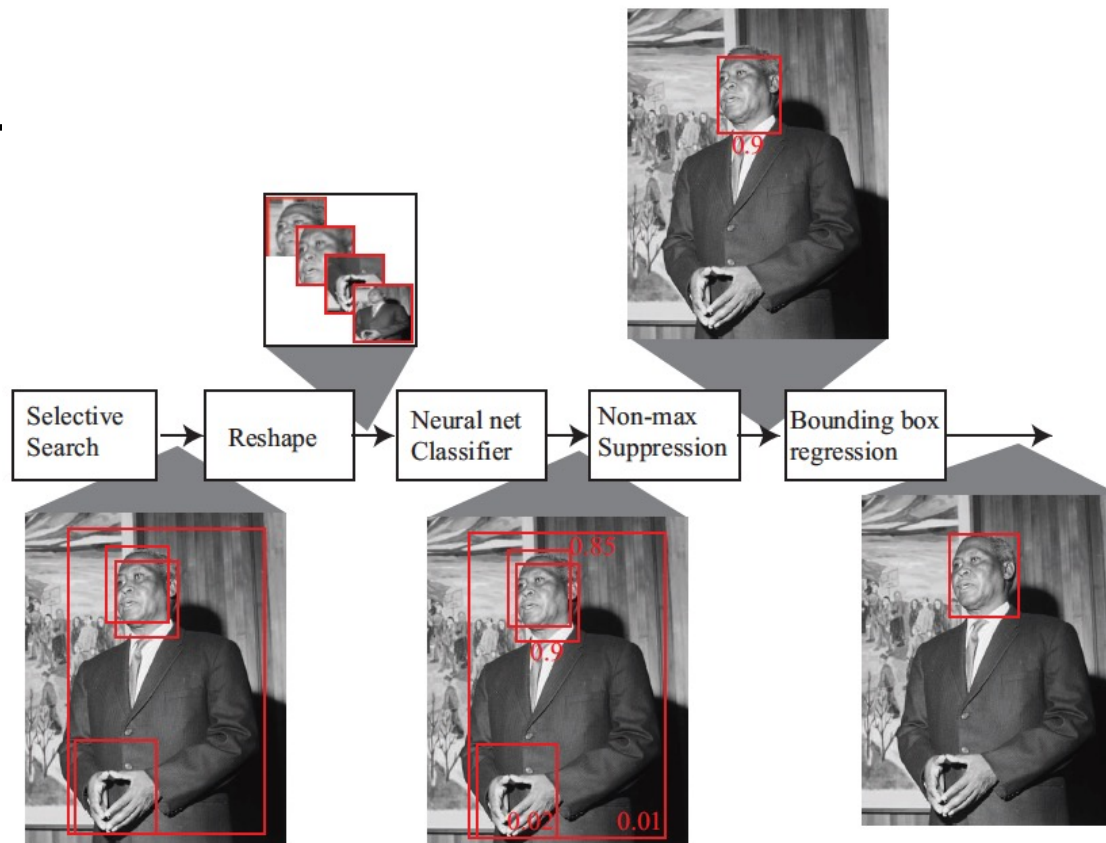


FIGURE 18.6: A schematic picture of how R-CNN works. A picture of Inkosi Albert Luthuli is fed in to selective search, which proposes possible boxes; these are cut out of the image, and reshaped to fixed size; the boxes are classified (scores next to each box); non-maximum suppression finds high scoring boxes and suppresses nearby high scoring boxes (so his face isn't found twice); and finally bounding box regression adjusts the corners of the box to get the best fit using the features inside the box.

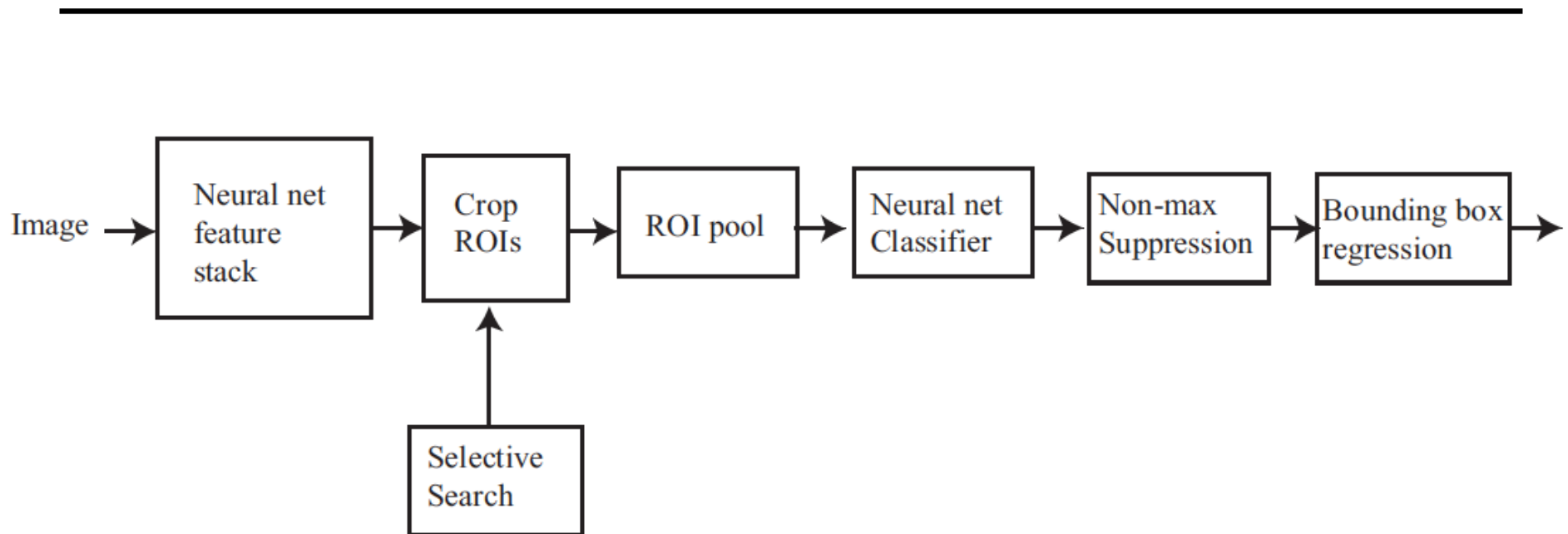


FIGURE 18.7: *Fast R-CNN is much more efficient than R-CNN, because it computes a single feature map from the image, then uses the boxes proposed by selective search to cut regions of interest (ROI's) from it. These are mapped to a standard size by a ROI pooling layer, then presented to a classifier. The rest should be familiar.*

Configuration spaces

You should think of a box as a point in a 4D space

- configuration space of the boxes

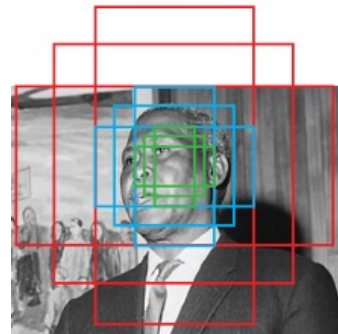
Selective search is weird

- networks don't do lists much

Alternative

- sample the configuration space on some form of grid
 - eg three aspect ratios, three scales, grid of locations
 - important: many possible sampling schemes
- check each sample with rank score

Anchor boxes



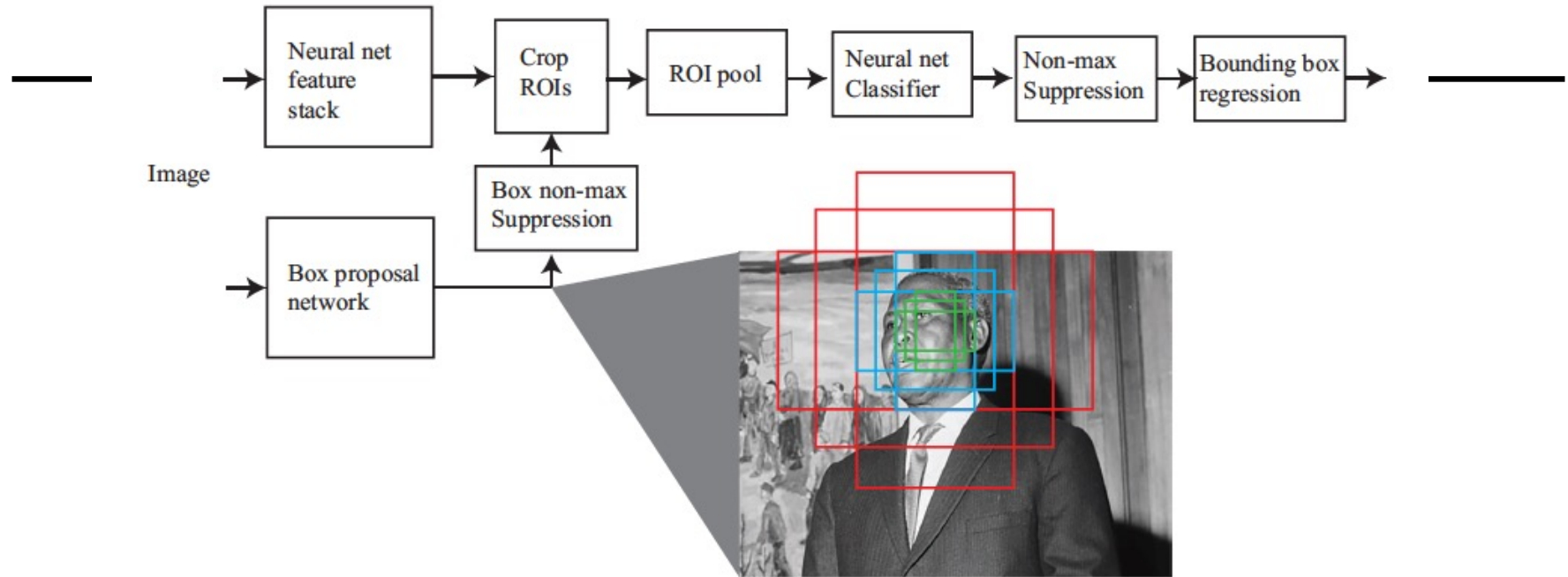


FIGURE 18.8: *Faster RCNN* uses two networks. One uses the image to compute “objectness” scores for a sampling of possible image boxes. The samples (called “anchor boxes”) are each centered at a grid point. At each grid point, there are nine boxes (three scales, three aspect ratios). The second is a feature stack that computes a representation of the image suitable for classification. The boxes with highest objectness score are then cut from the feature map, standardized with ROI pooling, then passed to a classifier. Bounding box regression means that the relatively coarse sampling of locations, scales and aspect ratios does not weaken accuracy.

Evaluating detectors

Compare detected boxes w ground truth boxes

Favor

- right number of boxes with right label in right place

Penalize

- awful lot of boxes
- multiple detections of the same thing

Strategy

Strategy:

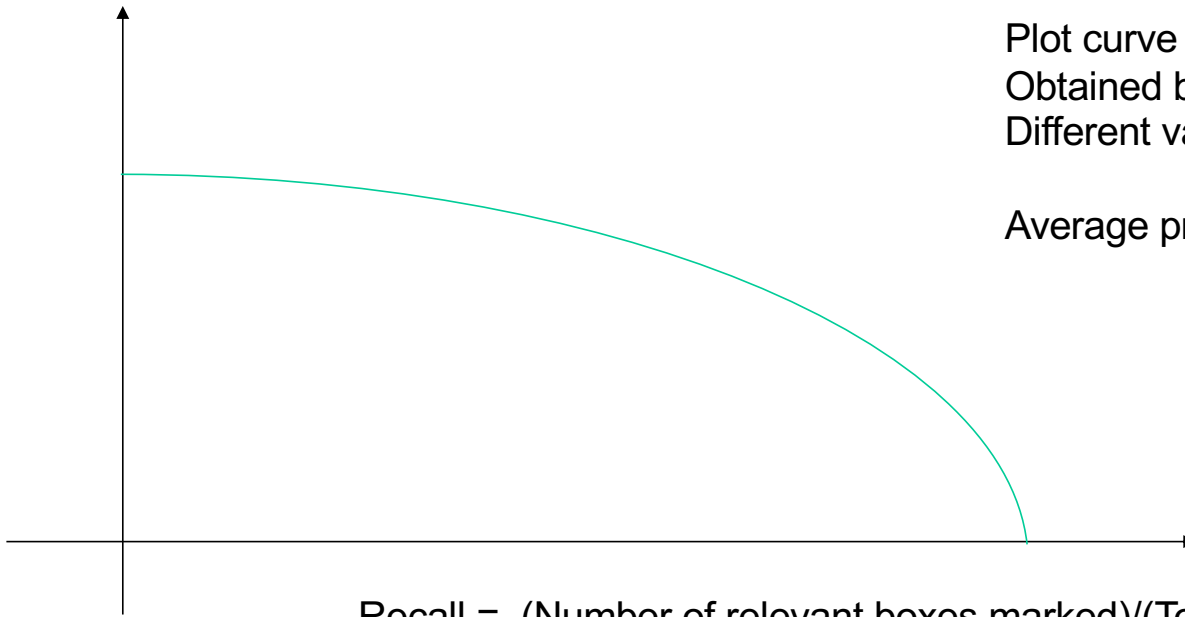
- Detector makes a ranked list of boxes
- GT is a list of boxes
- Mark detector boxes with relevant/irrelevant
- summarize lists

Marking boxes:

- All are irrelevant, then
- For each GT box:
 - Overlap measured as IOU (intersection over union)
 - Find highest ranking box with largest overlap
 - mark relevant if $\text{IOU} > \text{threshold}$

Average precision

Precision = (number of relevant boxes marked)/(total number of boxes marked)



Plot curve by computing recall, precision
Obtained by taking top k boxes in list for
Different values of k

Average precision is area under curve

$$\text{Recall} = (\text{Number of relevant boxes marked})/(\text{Total number of relevant boxes})$$

Strategy

Strategy:

- Detector makes a ranked list of boxes
- GT is a list of boxes
- Mark detector boxes with relevant/irrelevant
- summarize lists

Summarize lists:

- Sort by box ranking
- Compute AP per class
- Compute average of AP

- MAP at IOU 0.5 has been standard for a while
- Higher IOU's are harder.

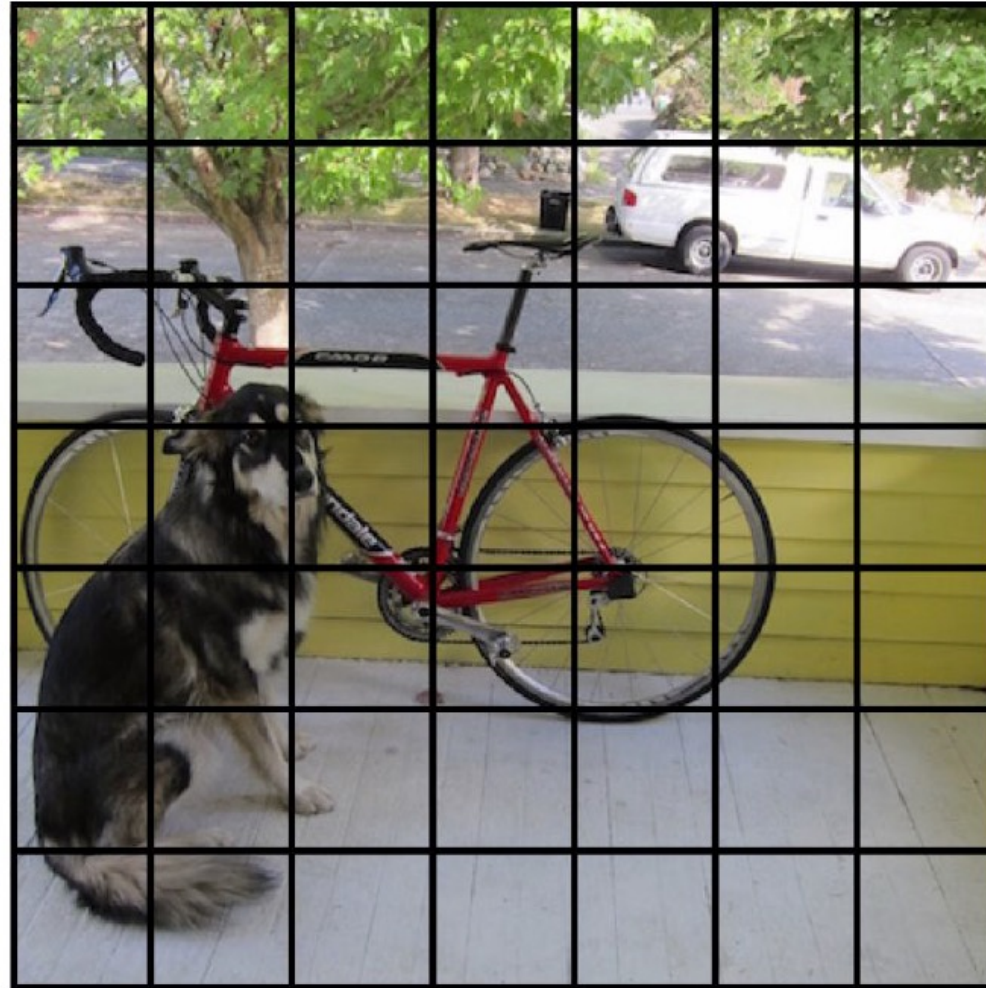
YOLO

YOLO v8 is about as fast and accurate as you can get
[link on webpage](#)

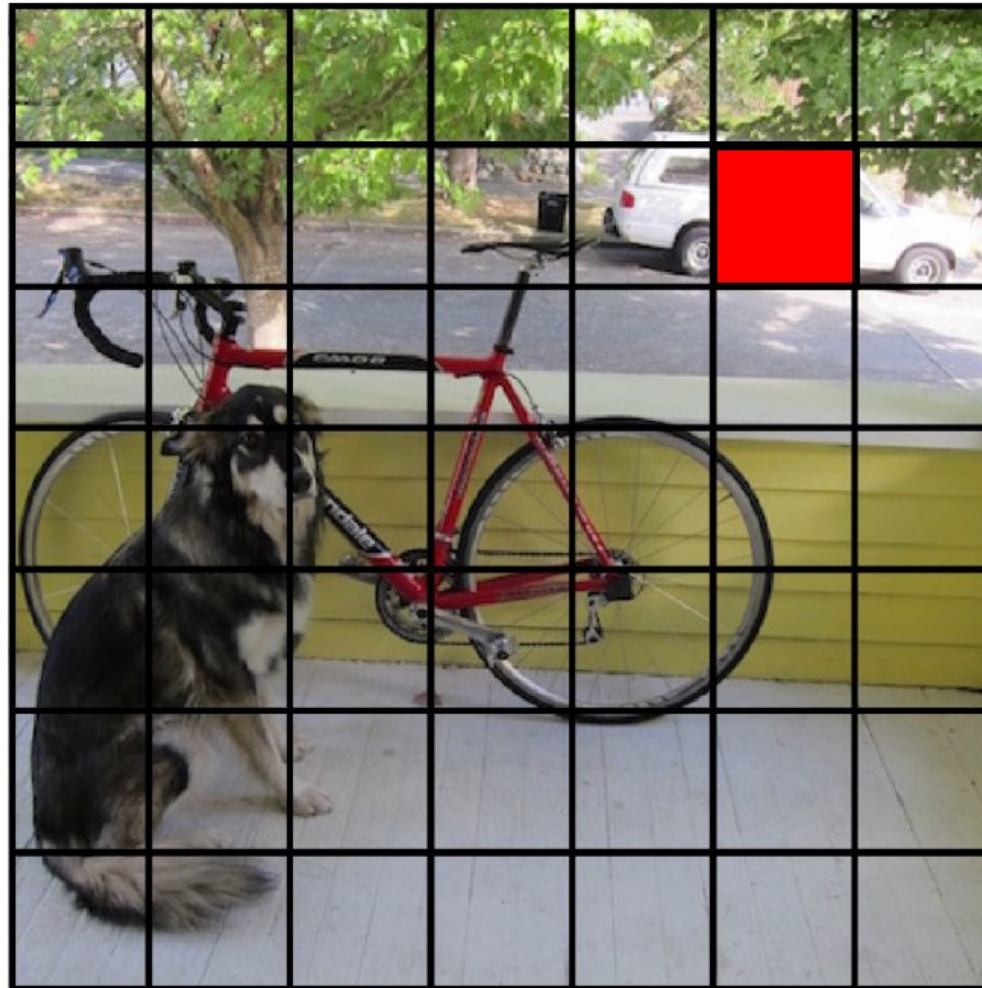
key idea

- look at box scores, label values independently

We split the image into a grid



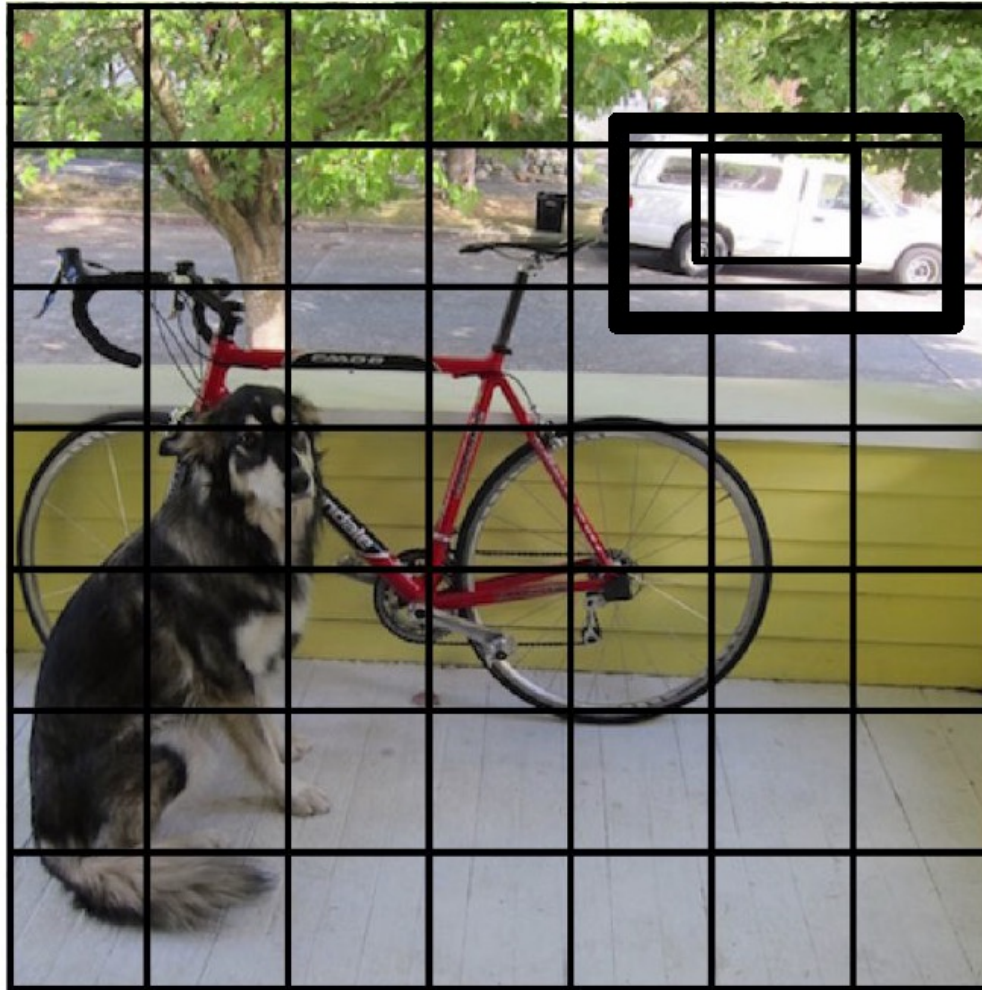
Each cell predicts boxes and confidences: $P(\text{Object})$



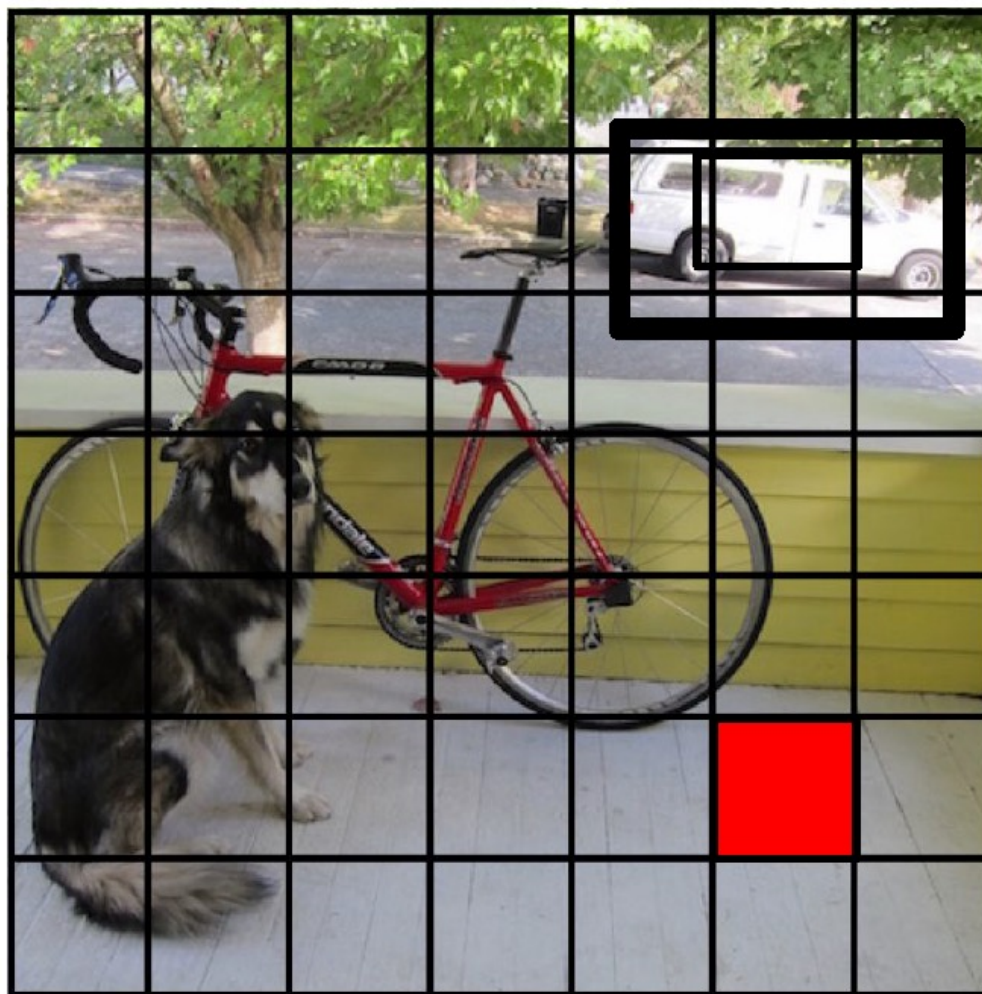
Each cell predicts boxes and confidences: $P(\text{Object})$



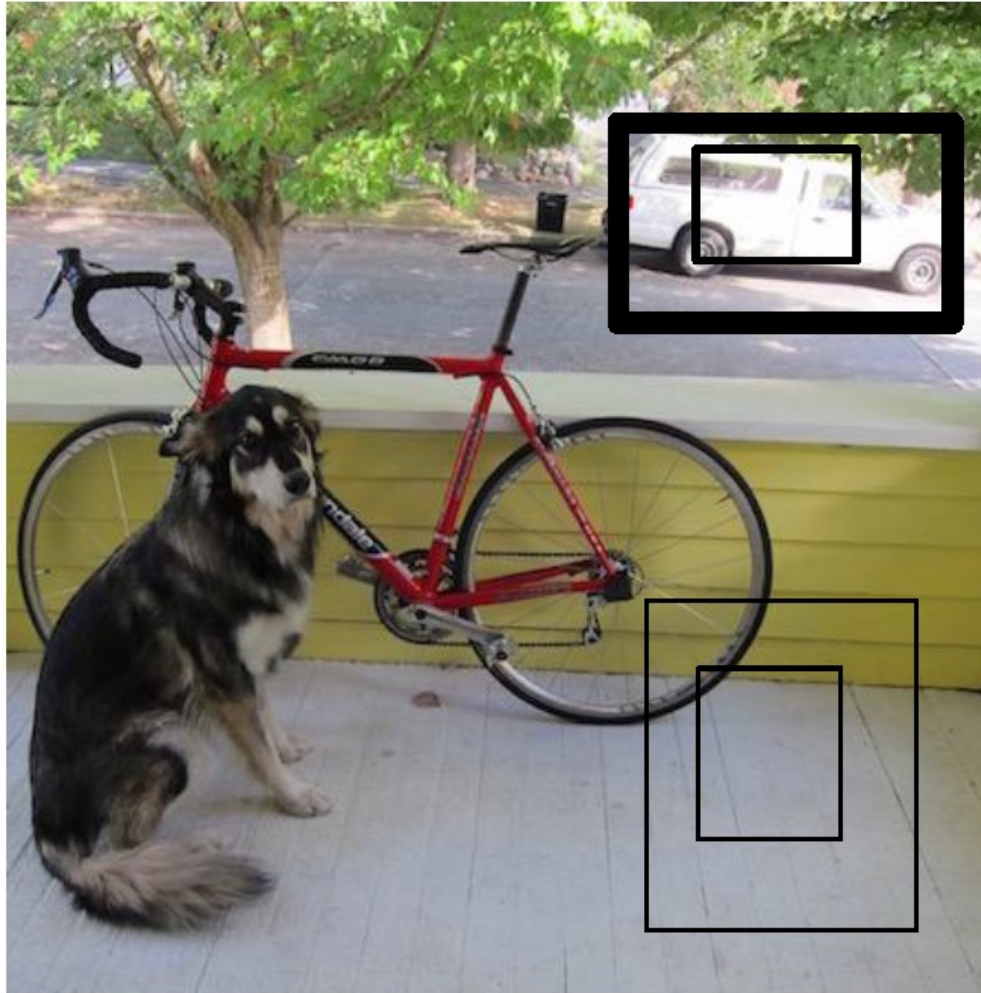
Each cell predicts boxes and confidences: $P(\text{Object})$



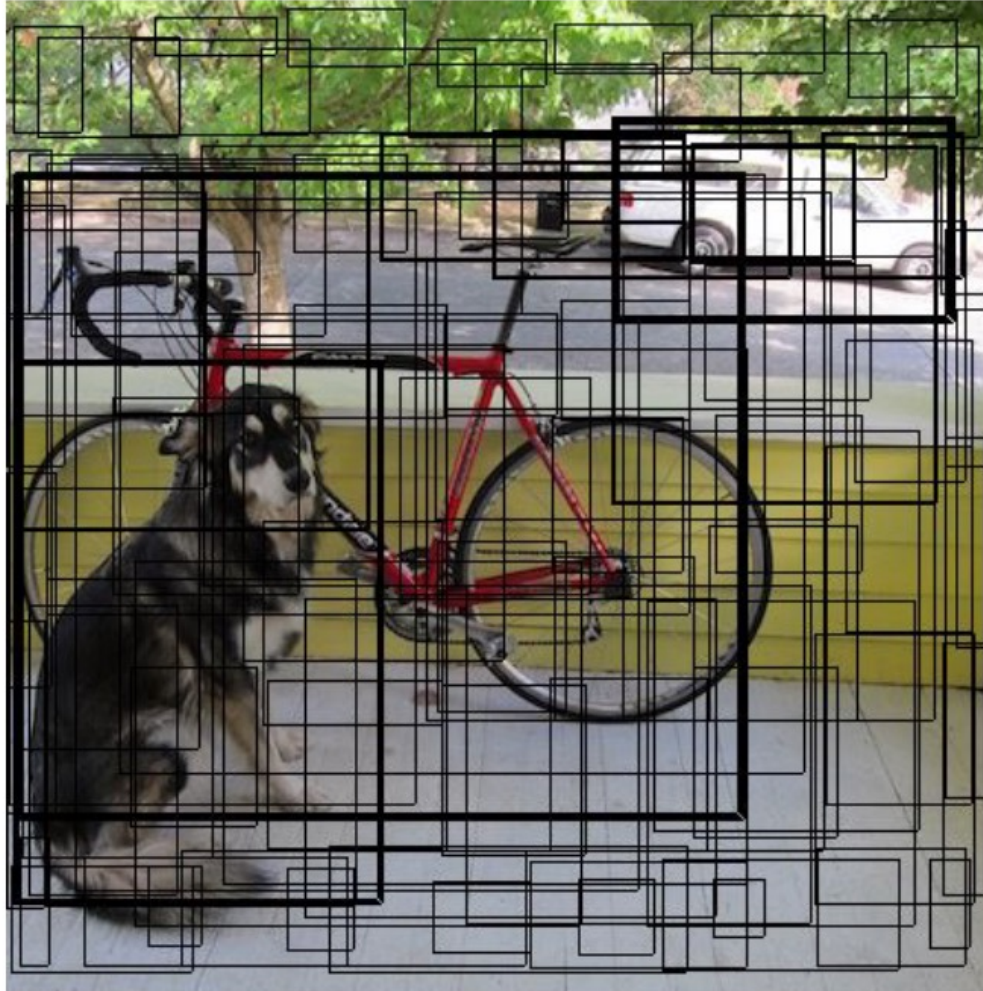
Each cell predicts boxes and confidences: $P(\text{Object})$



Each cell predicts boxes and confidences: $P(\text{Object})$



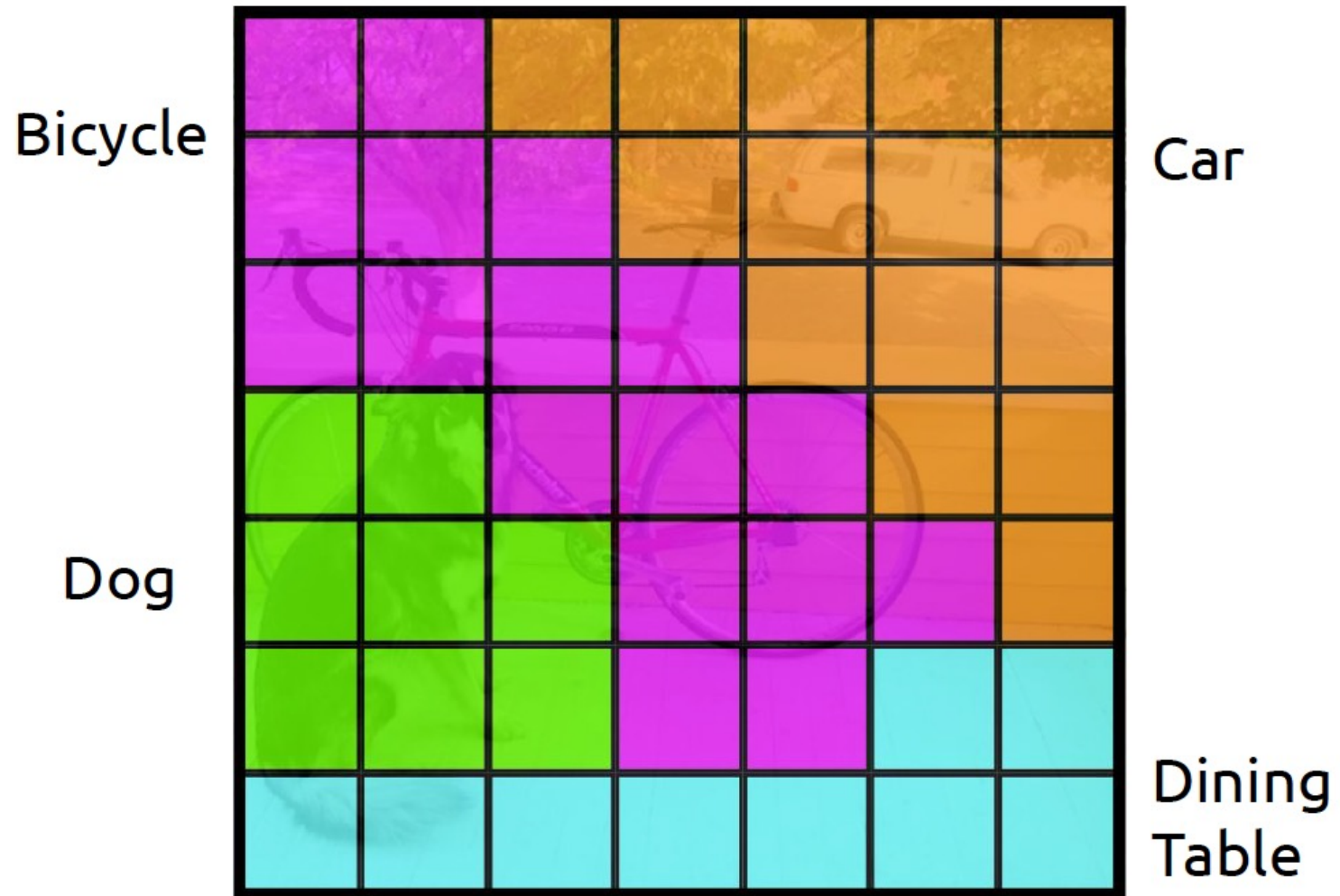
Each cell predicts boxes and confidences: $P(\text{Object})$



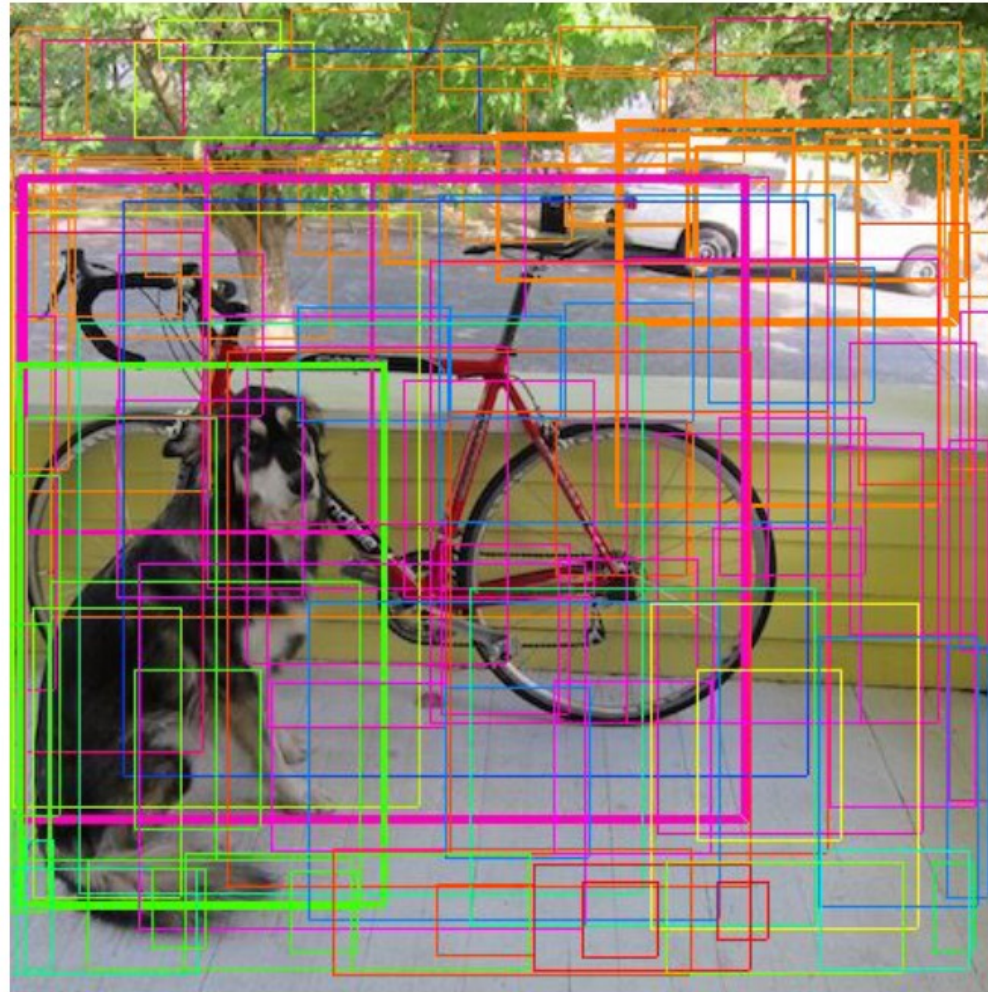
Each cell also predicts a class probability.



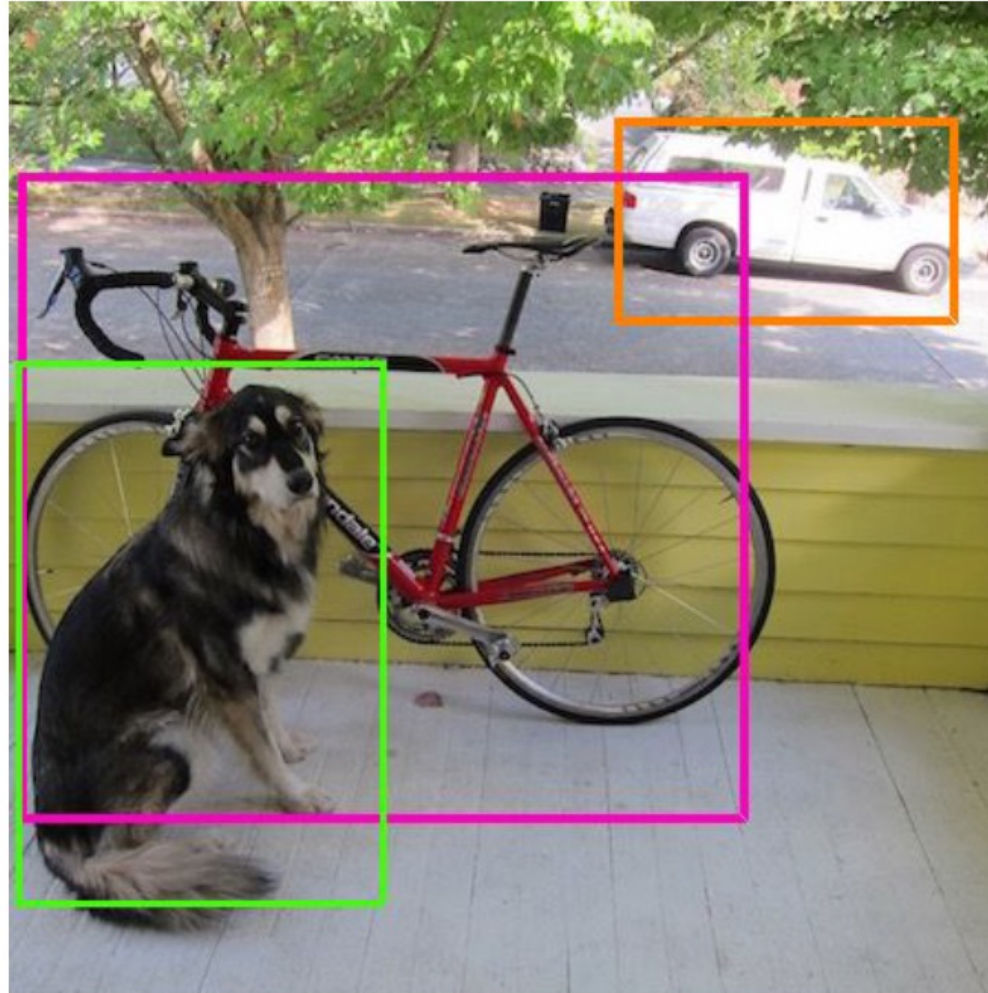
Each cell also predicts a class probability.



Then we combine the box and class predictions.



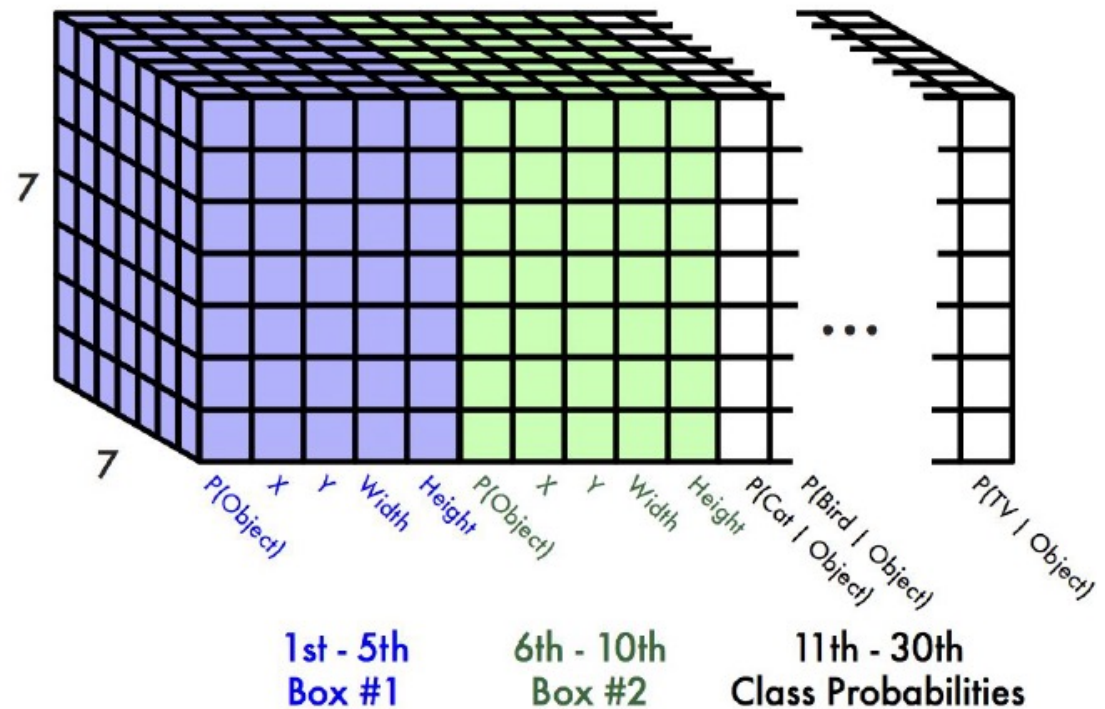
Finally we do NMS and threshold detections



This parameterization fixes the output size

Each cell predicts:

- For each bounding box:
 - 4 coordinates (x, y, w, h)
 - 1 confidence value
- Some number of class probabilities

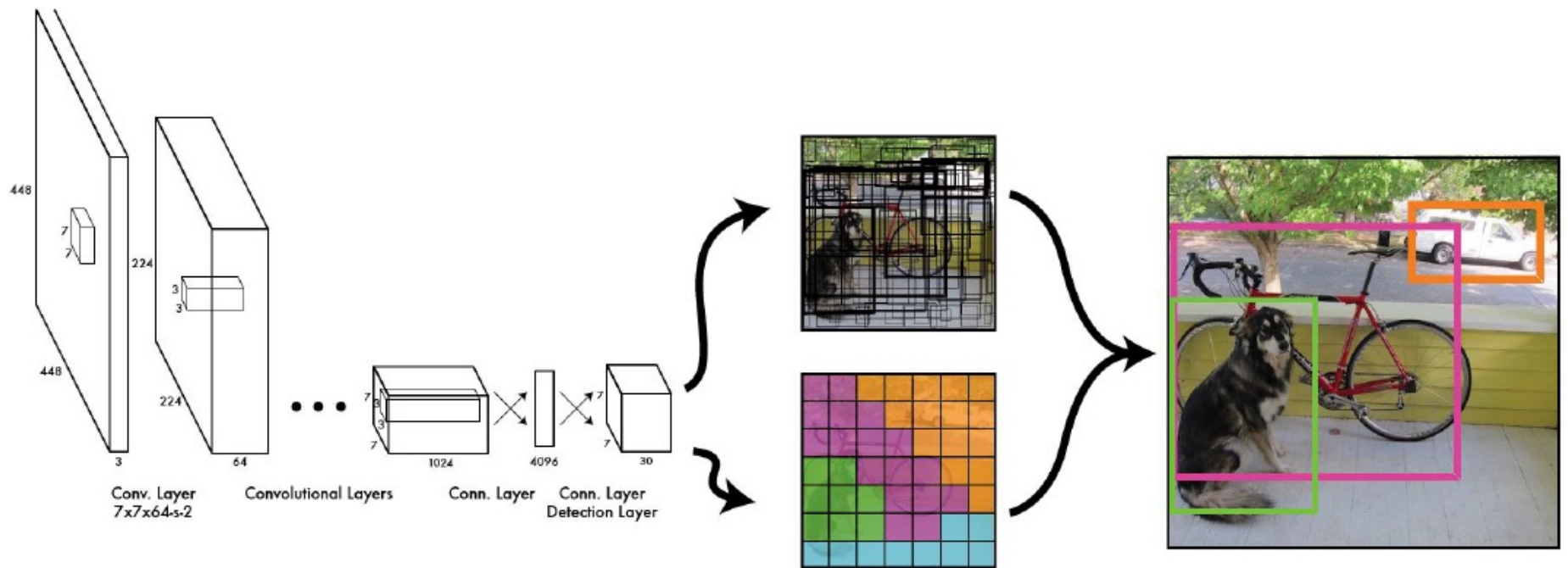


For Pascal VOC:

- 7x7 grid
- 2 bounding boxes / cell
- 20 classes

$7 \times 7 \times (2 \times 5 + 20) = 7 \times 7 \times 30$ tensor = **1470 outputs**

Thus we can train one neural network to be a whole detection pipeline

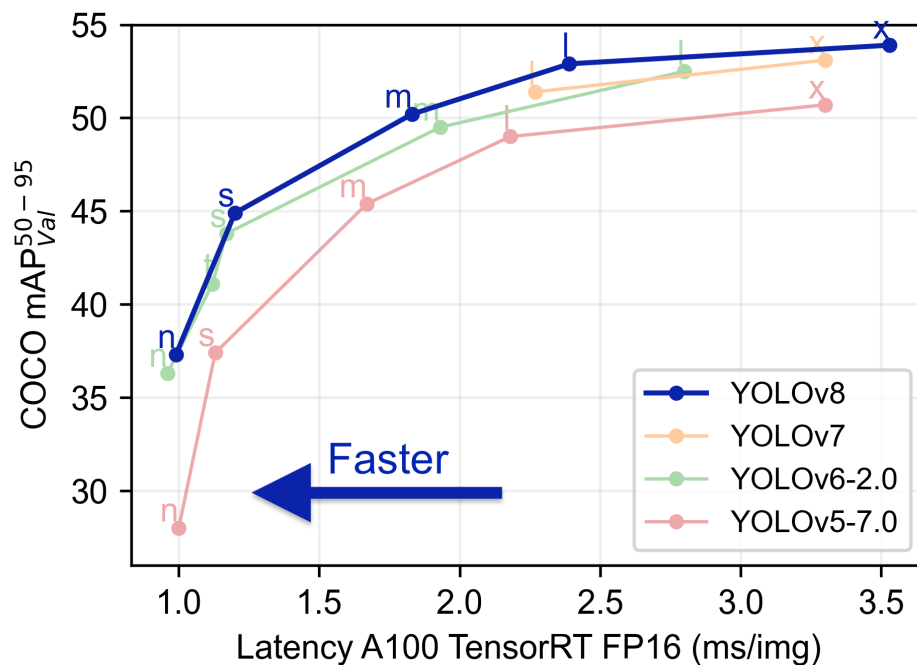
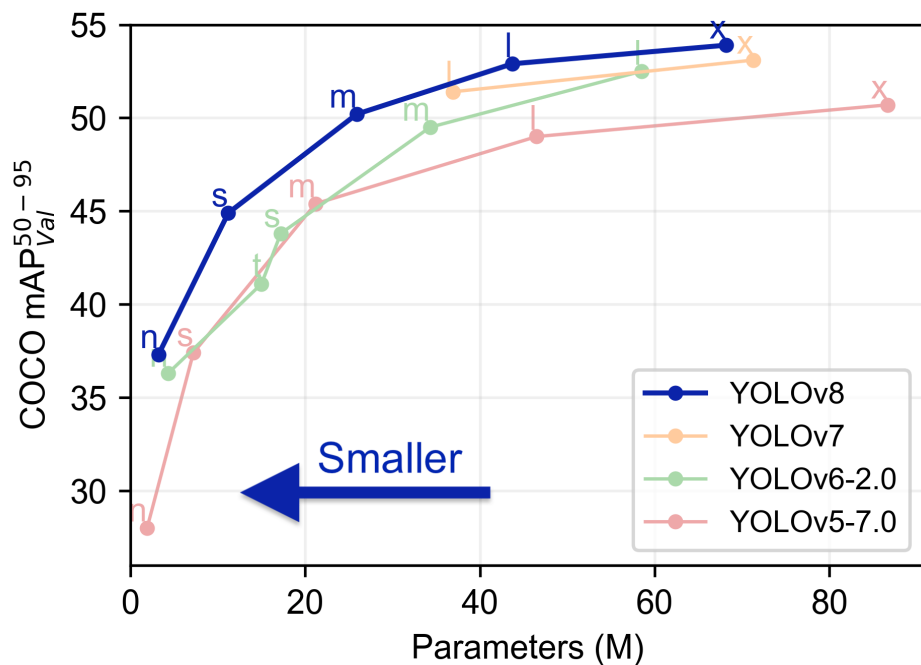


Evaluation, YOLOv8

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

<https://github.com/ultralytics/ultralytics>

YOLOv8 Tuning



SOA and variants: rough summary

Very accurate detection for hundreds of categories

- with enough training data
- important variations in training data available
 - you don't have to put a box on everything

YOLO allows a tradeoff between speed and accuracy

- and can be very fast

Variants

- Localization more accurate than boxes
- Incorporate LIDAR, etc.
- Boxes in 3D rather than 2D
- Variant feature constructions are very important

Ghost(s) at the party

Object detection [73]



Tremblay et al 20