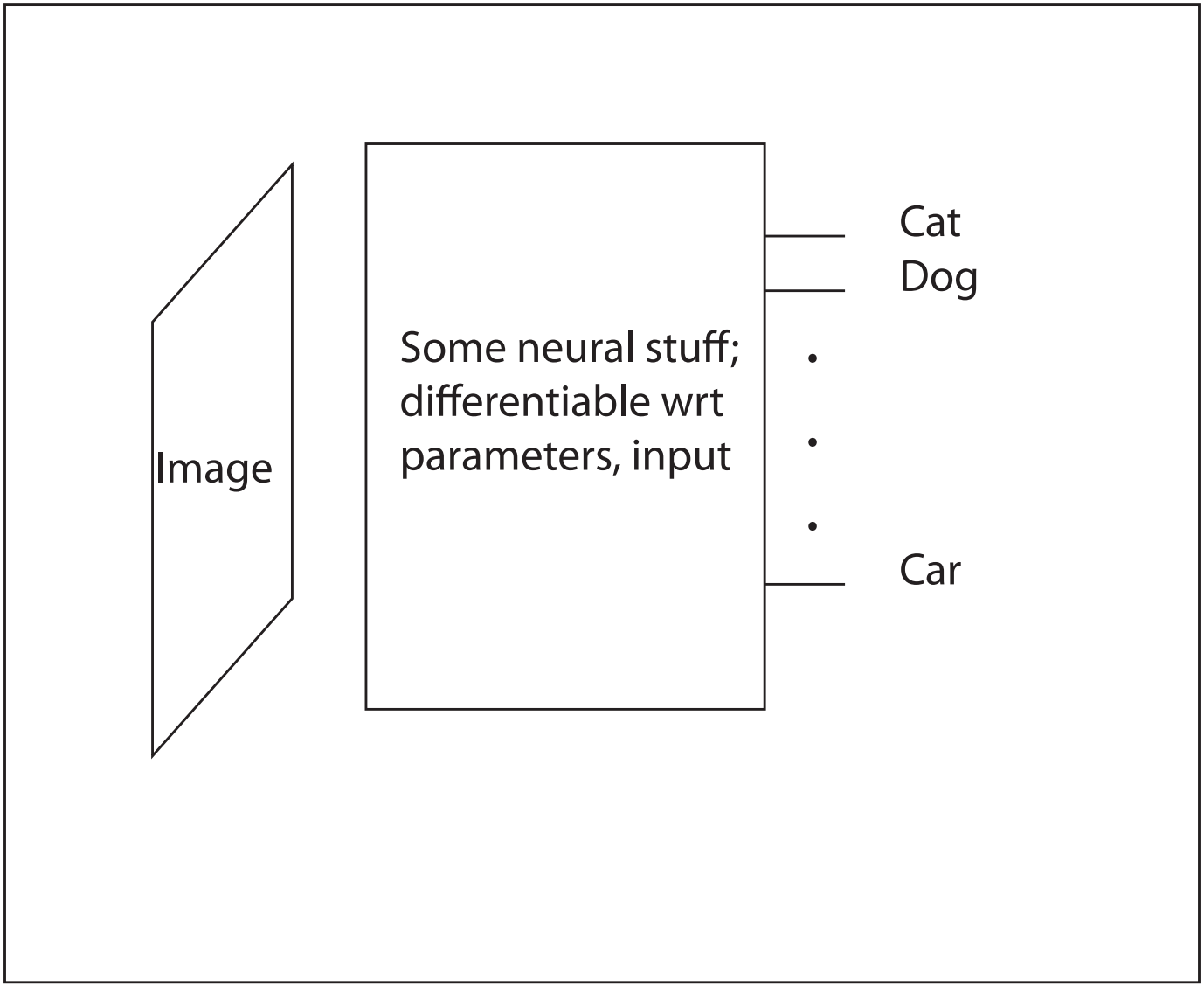
Classification, Detection and Regression

D.A. Forsyth, UIUC

Classification, Detection and Reconstruction

- 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



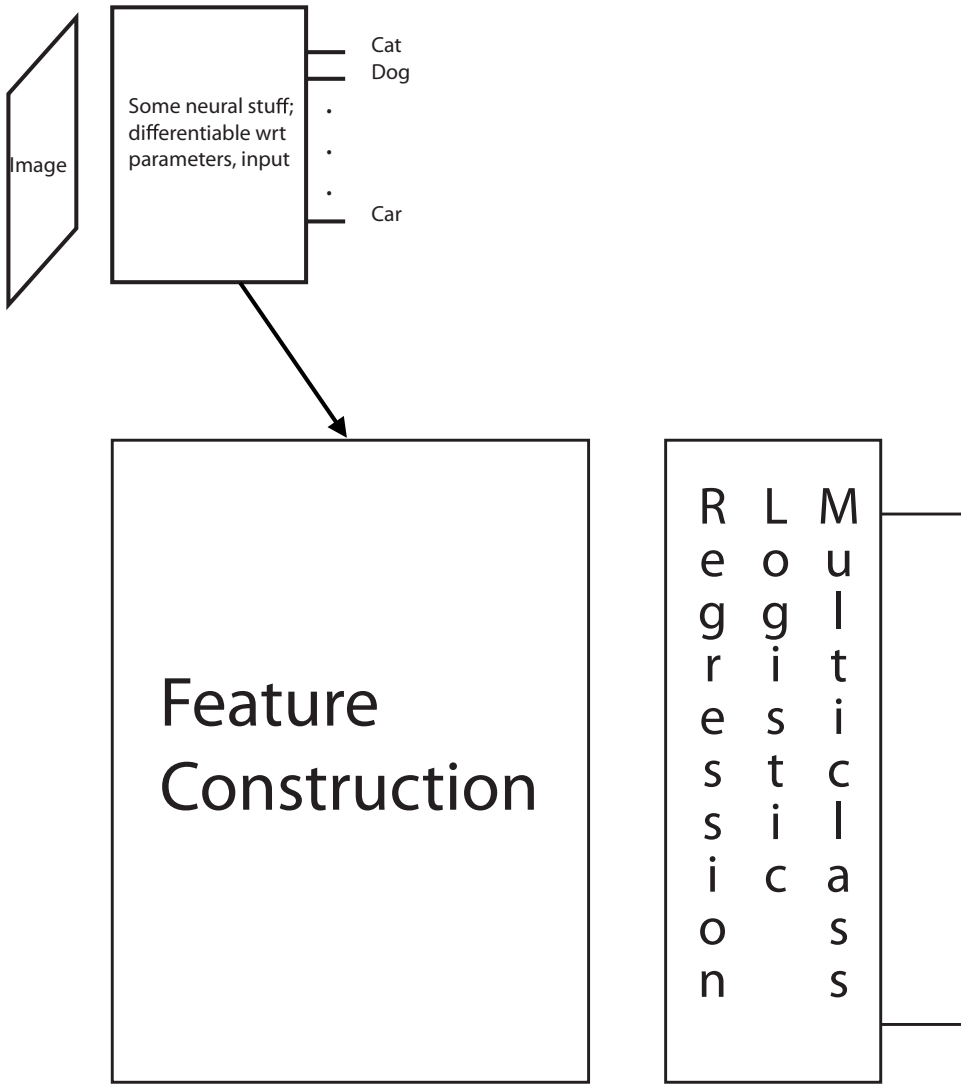
Key ideas

- 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

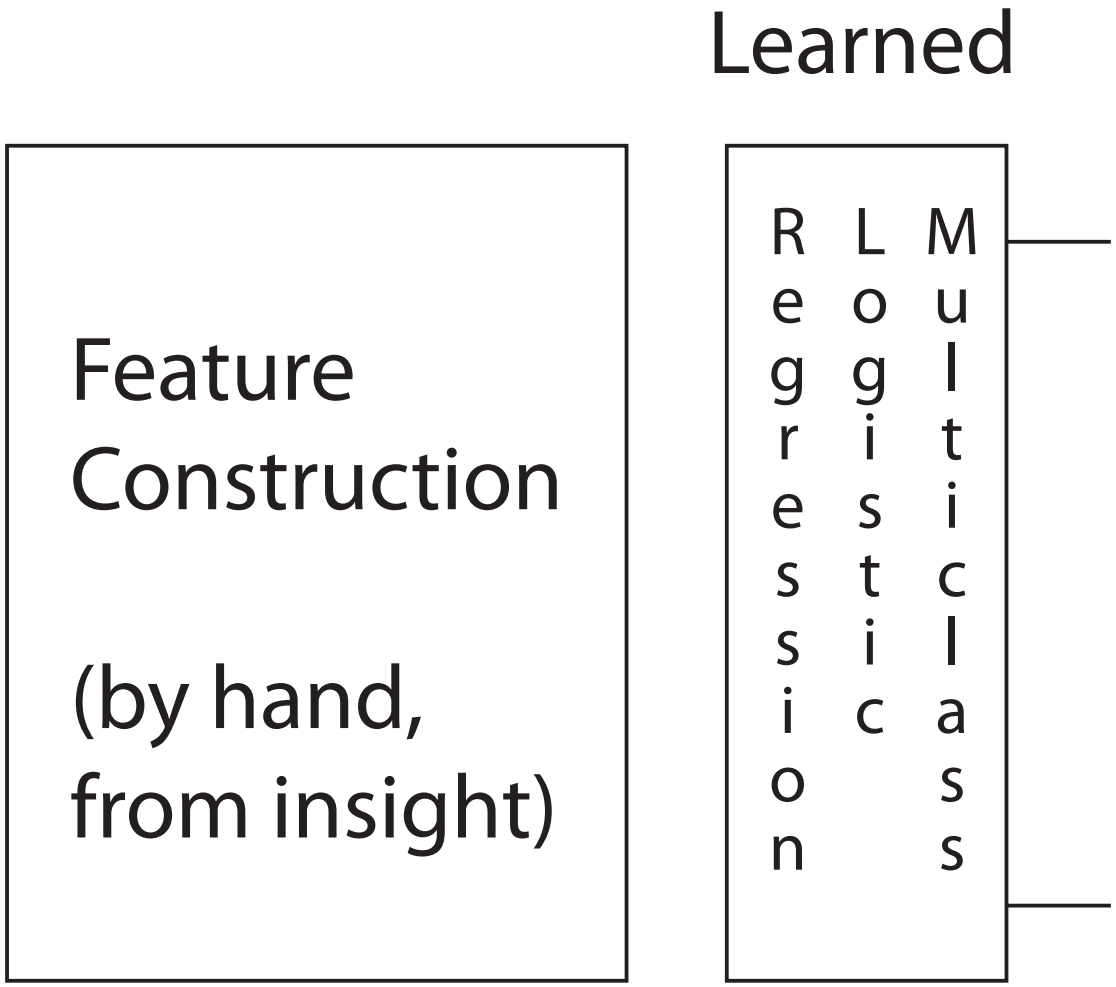
Main Points

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)
Regression
Logistic
Multiclass

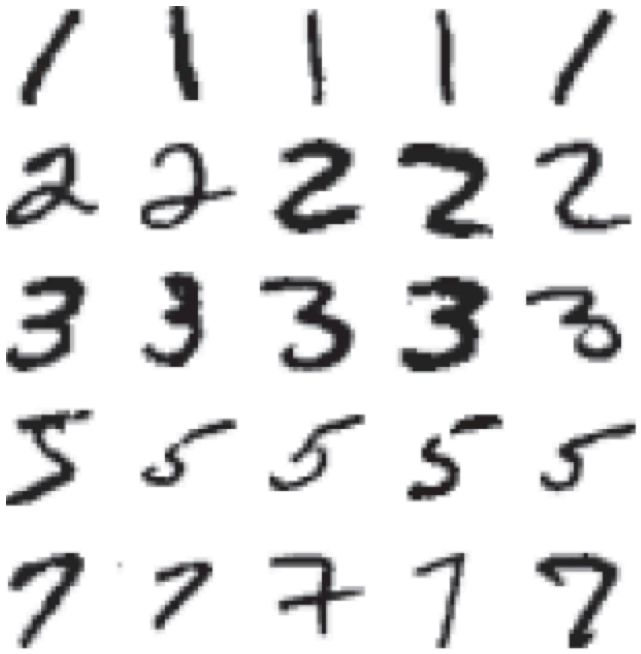
Multiclass logistic regression

- For classes 1, .., C
- Given a feature vector $\mathbf{x}$
- Form $\mathbf{w}_i^T \mathbf{x}$
- 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

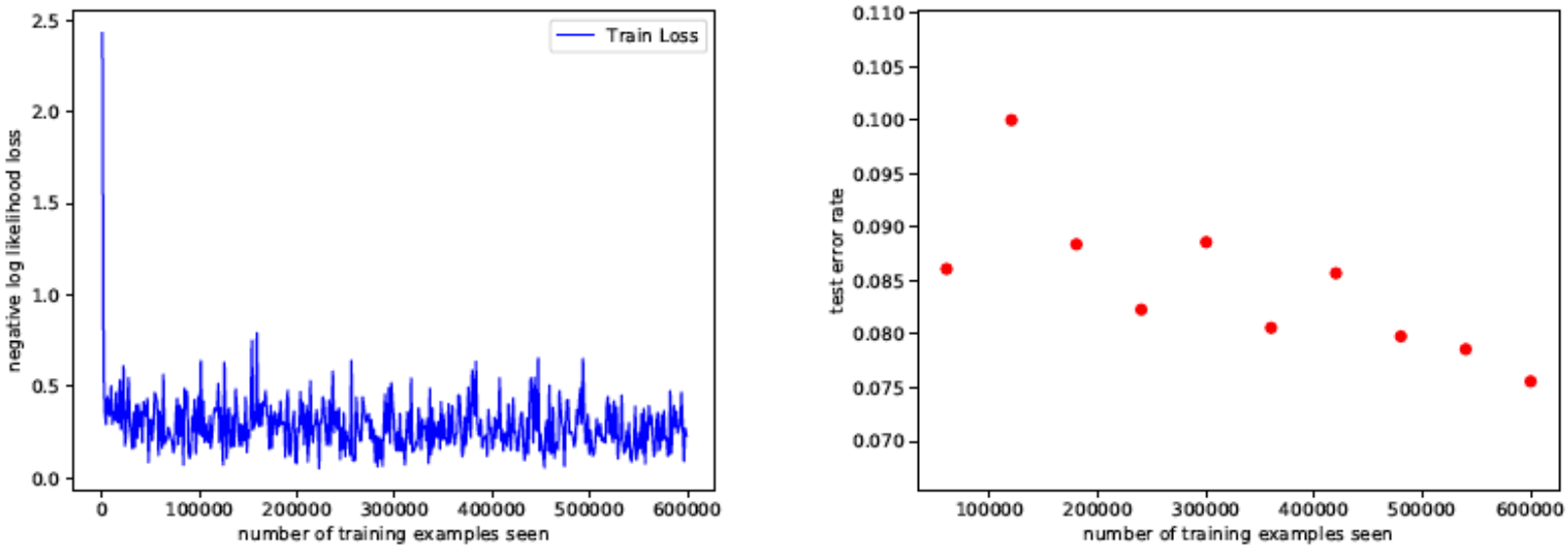
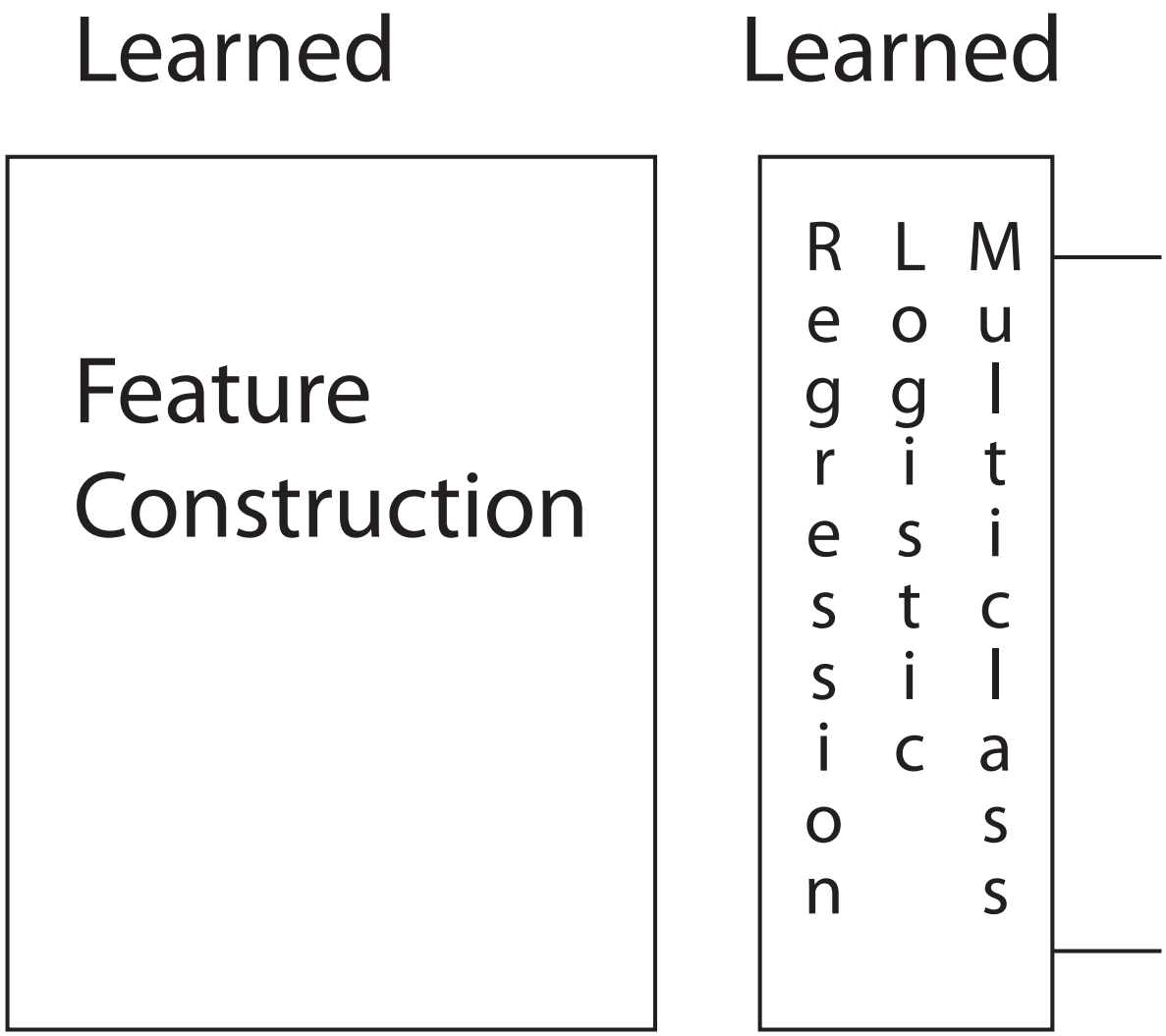
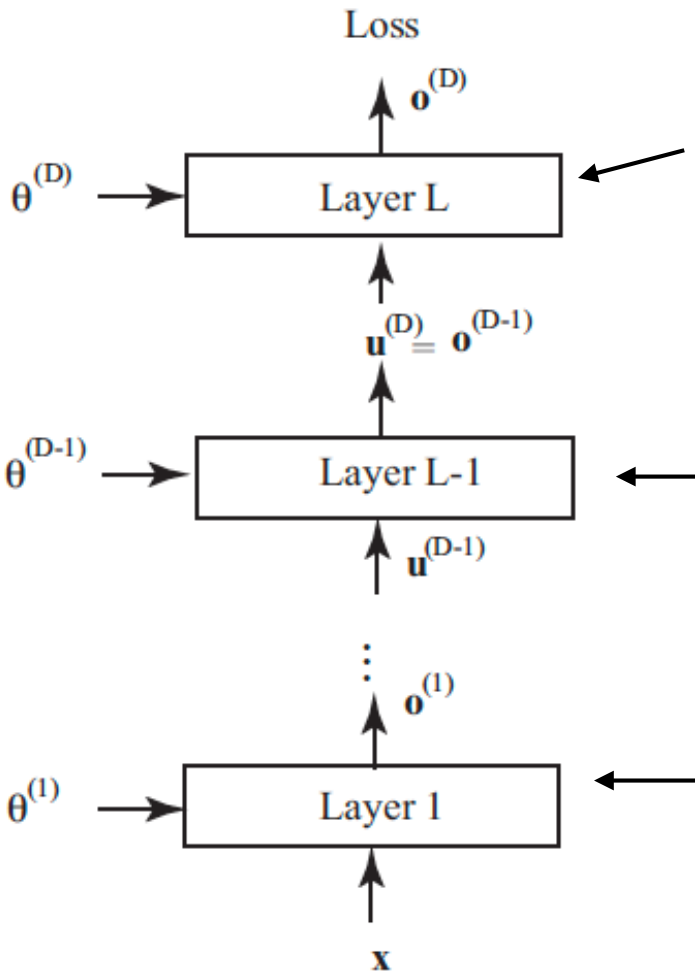


FIGURE 5.1: *On the* **left**, *the learning curve for a logistic regression classifier trained on MNIST data. Note the loss falls off quickly, then declines very slowly. The loss plotted here is the loss for a particular batch after a step has been taken using the gradient on that batch. Although the 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
Feature Construction
Regression
Logistic
Multiclass

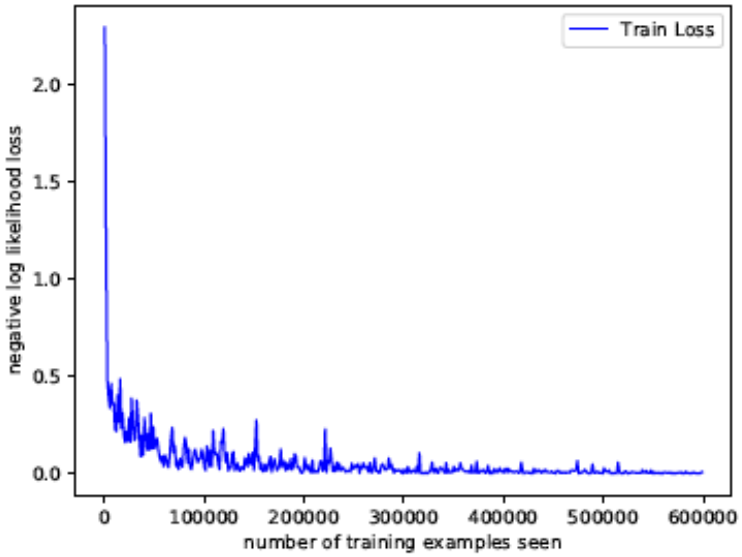
Making features using layers of functions

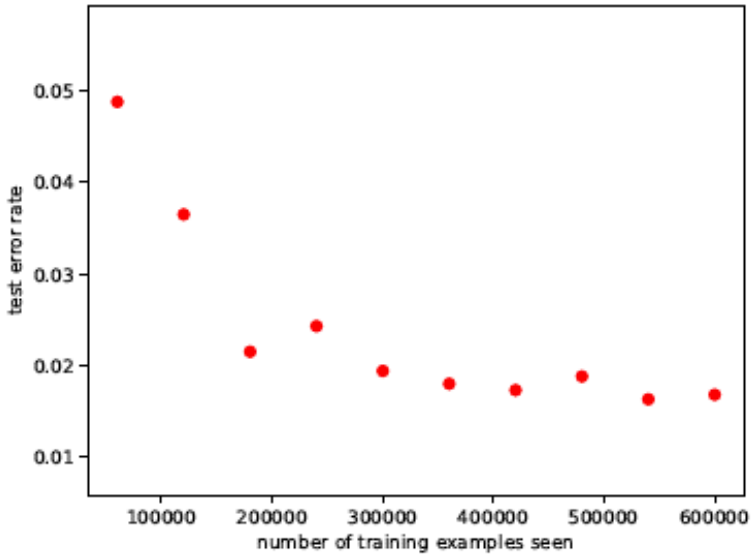


Logistic regression

Something else that isn't linear and is likely to be helpful.
Must have many outputs, many inputs and a manageable number of parameters

Multiple feature constructing layers





What do we need to classify?

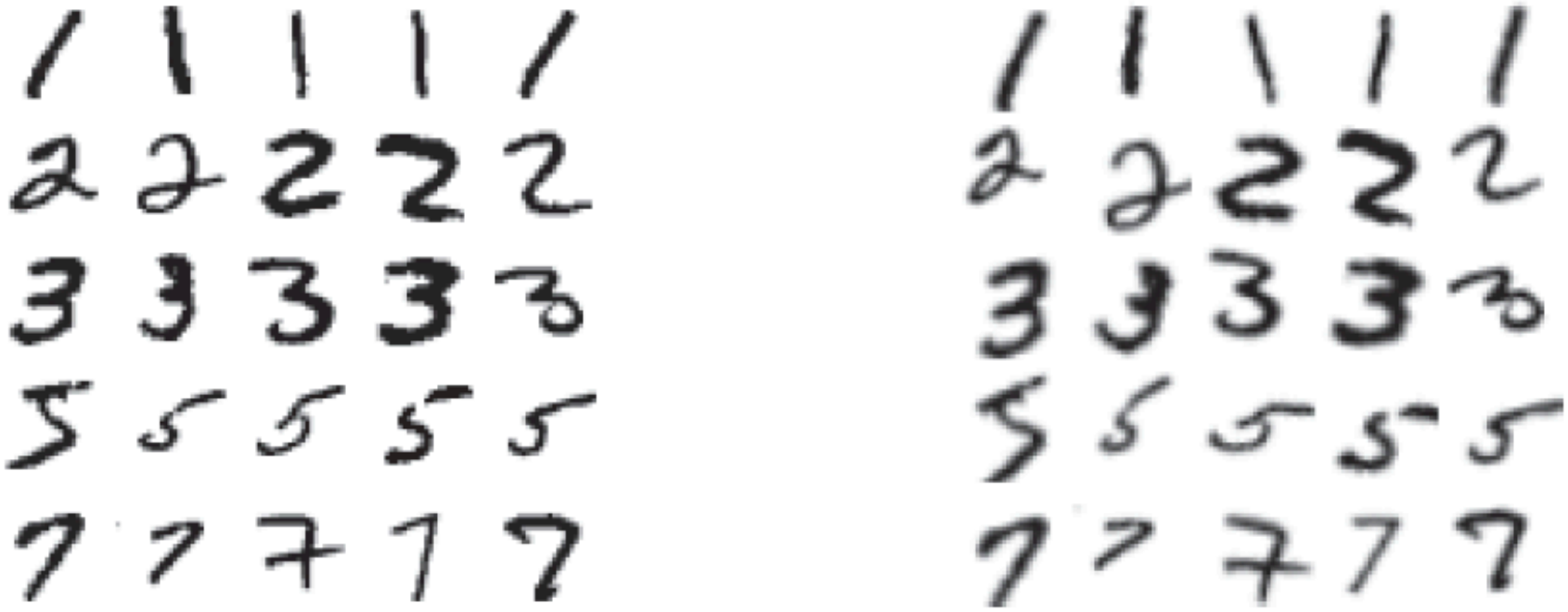


FIGURE 6.12: *On the* **left**, *a selection of digits from the MNIST dataset. Notice how images of the same digit can vary, which makes classifying the image demanding. It is quite usual that pictures of "the same thing" look quite different. On the* **right**, *digit images from MNIST that have been 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.*

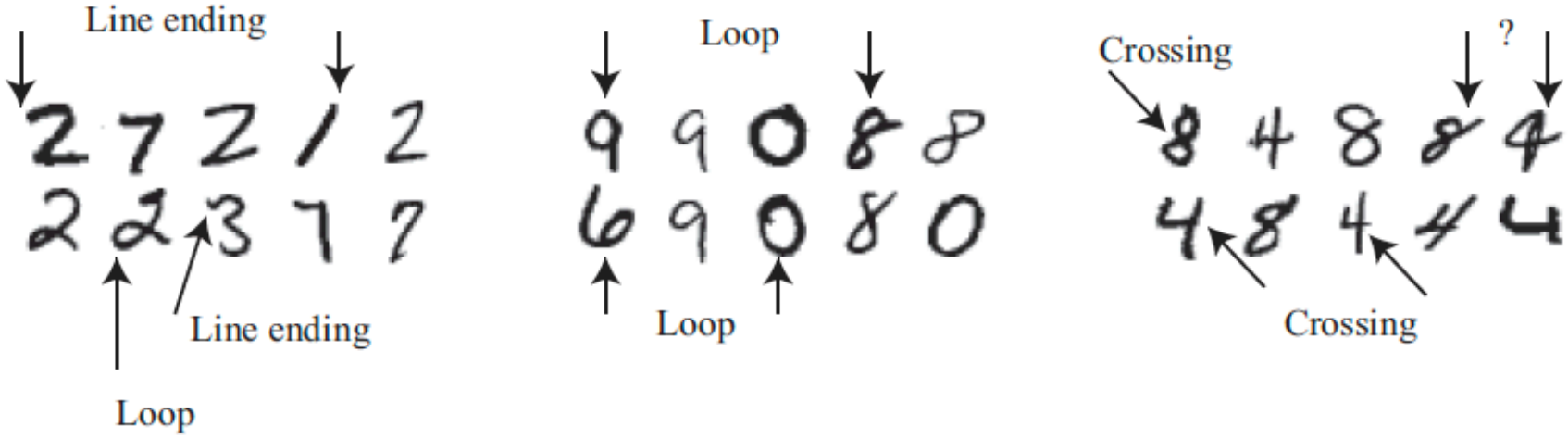


FIGURE 6.13: *Local patterns in images are quite informative. MNIST images, shown here, are simple images, so a small set of patterns is quite helpful. The relative location of patterns is also informative. So, for example, an eight has two loops, one above the other. All this suggests a key strategy: 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

$$\begin{aligned} \mathcal{N} &= \mathbf{conv}(\mathcal{I}, \mathcal{W}) \\ &\text{where} \\ \mathcal{N}_{ij} &= \sum_{uv} \mathcal{I}_{i-u,j-v} \mathcal{W}_{uv}. \end{aligned}$$

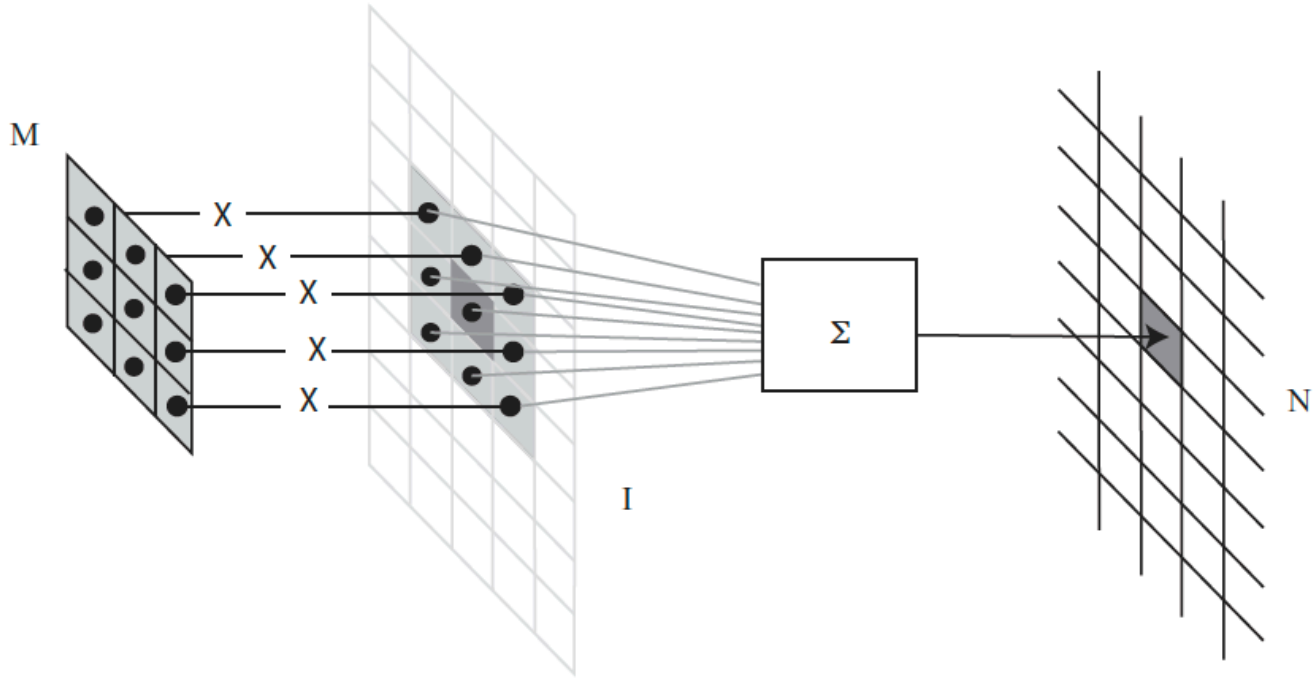


FIGURE 6.1: *To compute the value of $\mathcal{N}$ at some location, you shift a copy of $\mathcal{M}$ to lie over that location in $\mathcal{I}$; you multiply together the non-zero elements of $\mathcal{M}$ and $\mathcal{I}$ that lie on top of one another; and you sum the results.*

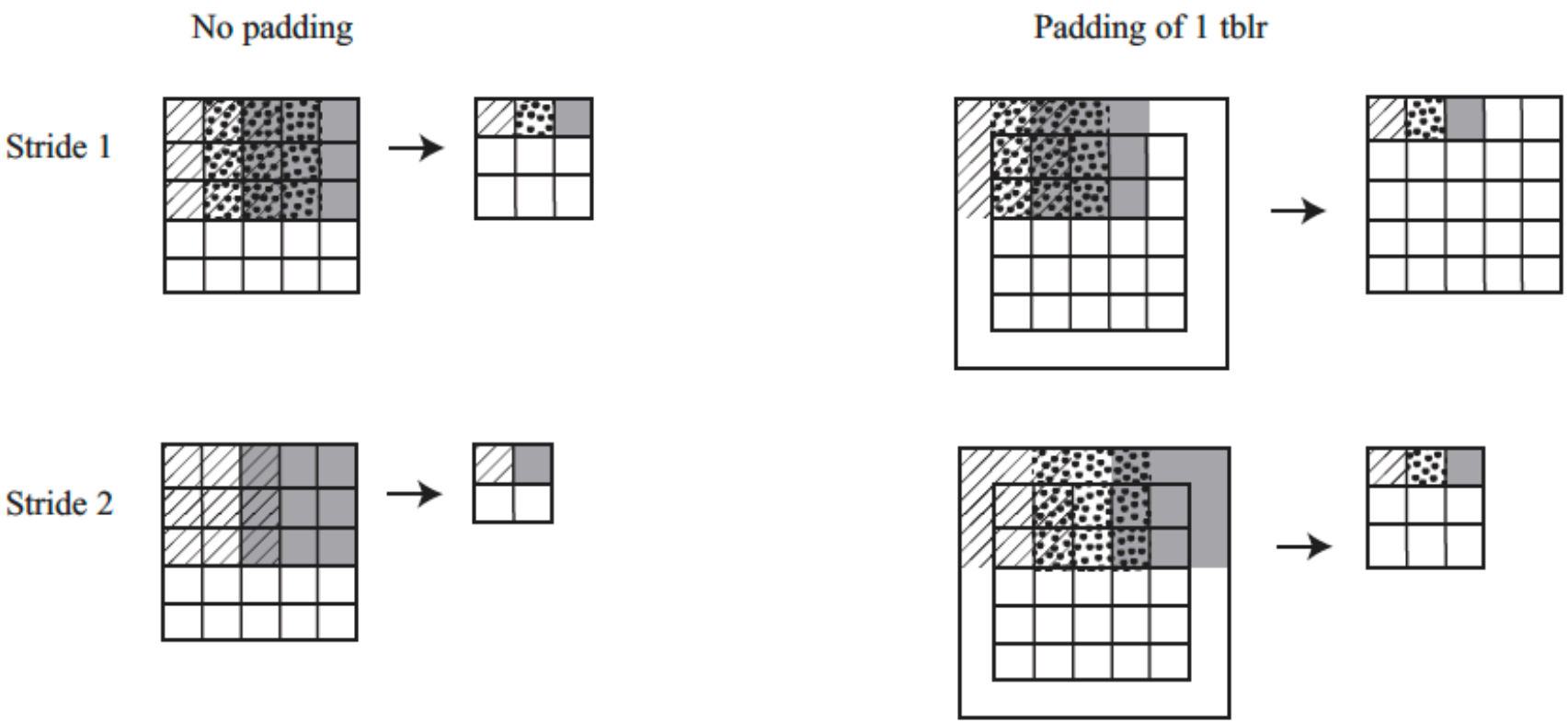
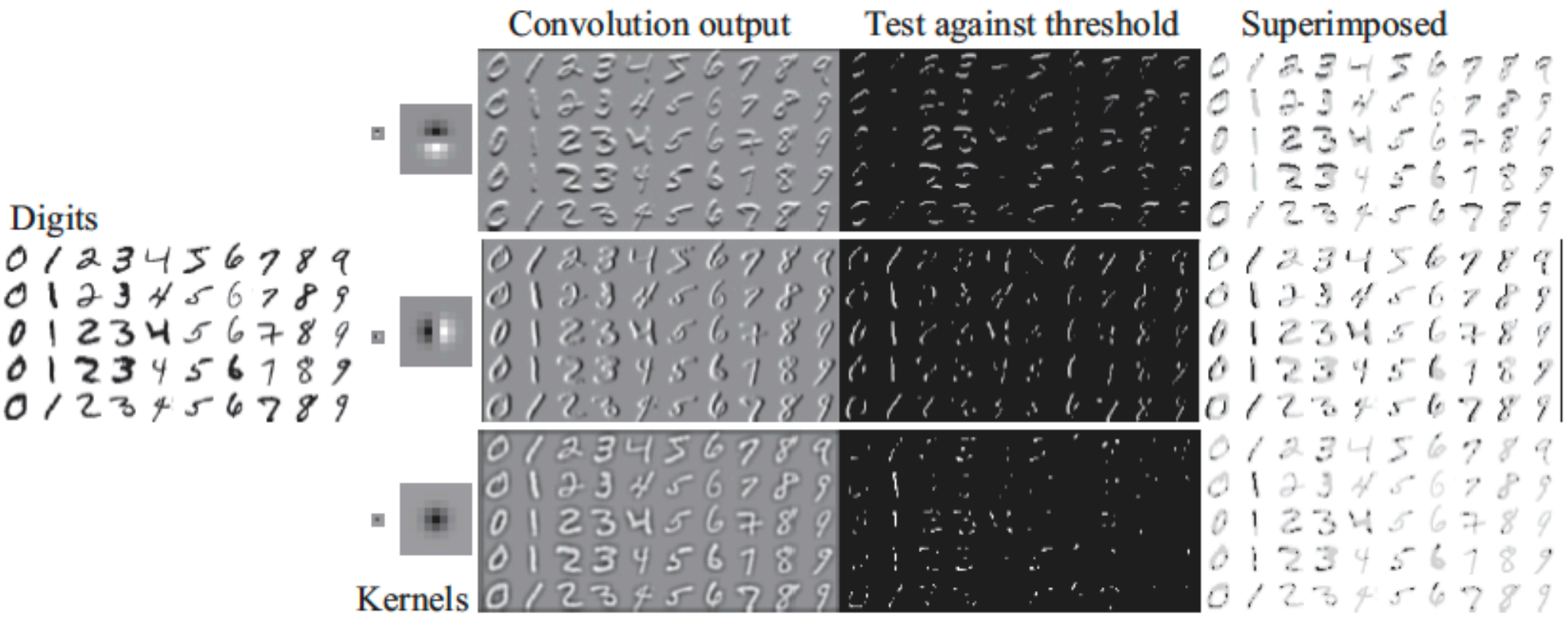


FIGURE 6.14: *The effects of stride and padding on* conv. *On the* **left**, conv *without padding accepts an* $\mathcal{I}$, *places a* 3×3 $\mathcal{M}$ *on grid locations determined by the stride, then reports values for valid windows. When the stride is* 1, *a* 5×5 $\mathcal{I}$ *becomes a* 3×3 $\mathcal{N}$. *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* $\mathcal{N}$.

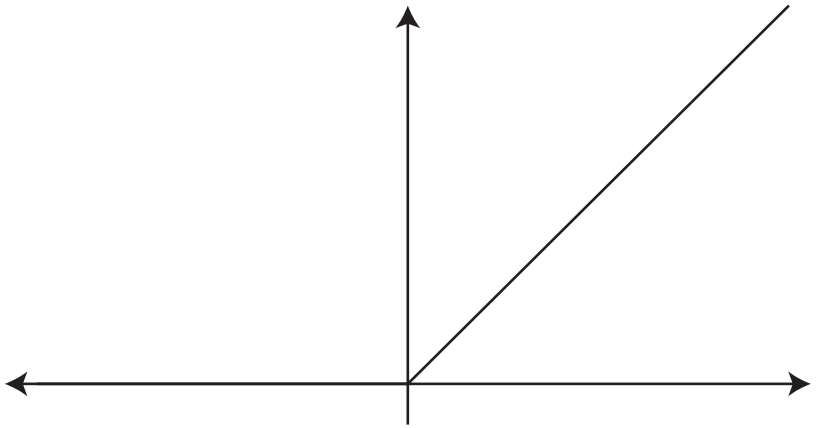
Convolution

- 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!


Convolution output
Test against threshold
Superimposed
Digits
Kernels

The ReLU

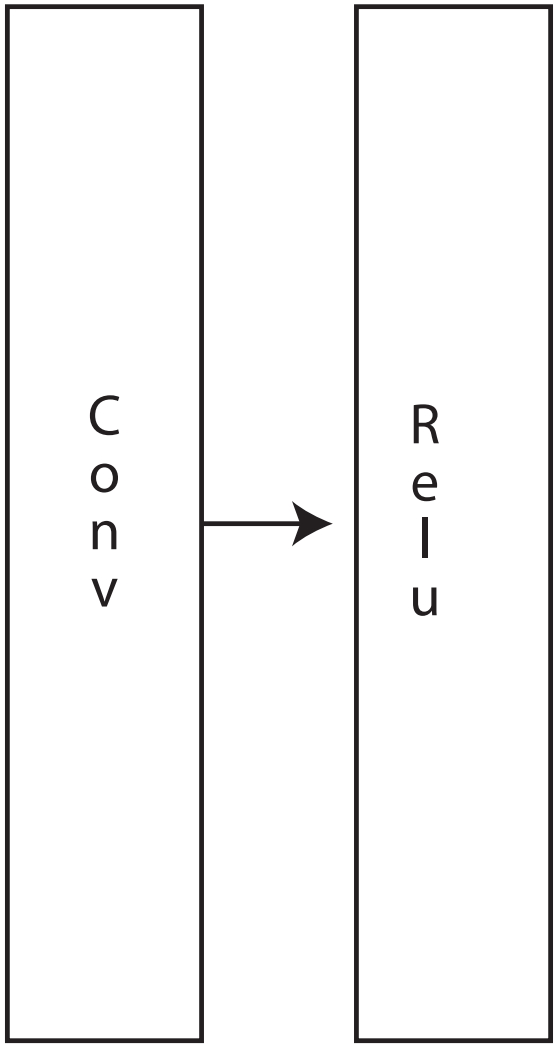
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$



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

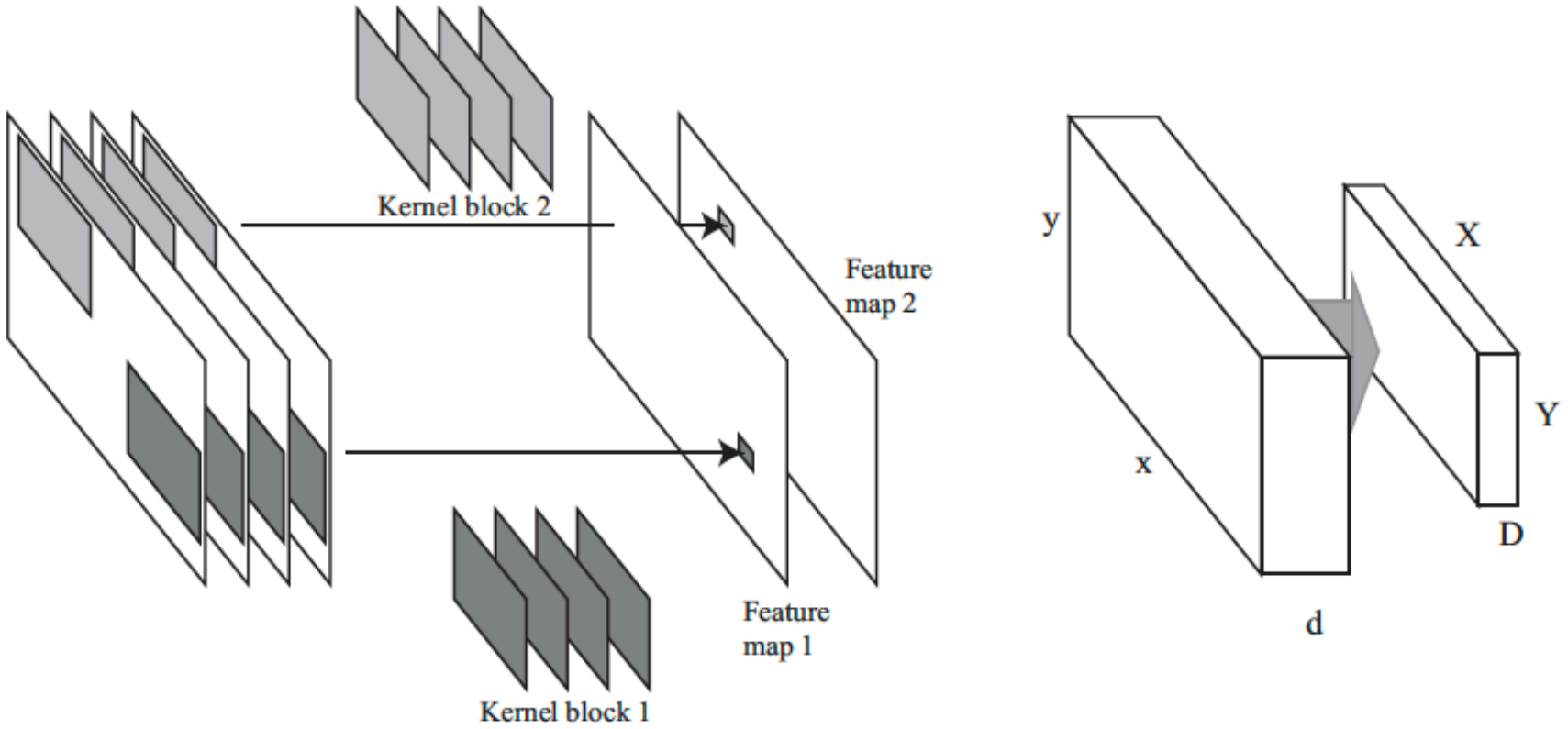
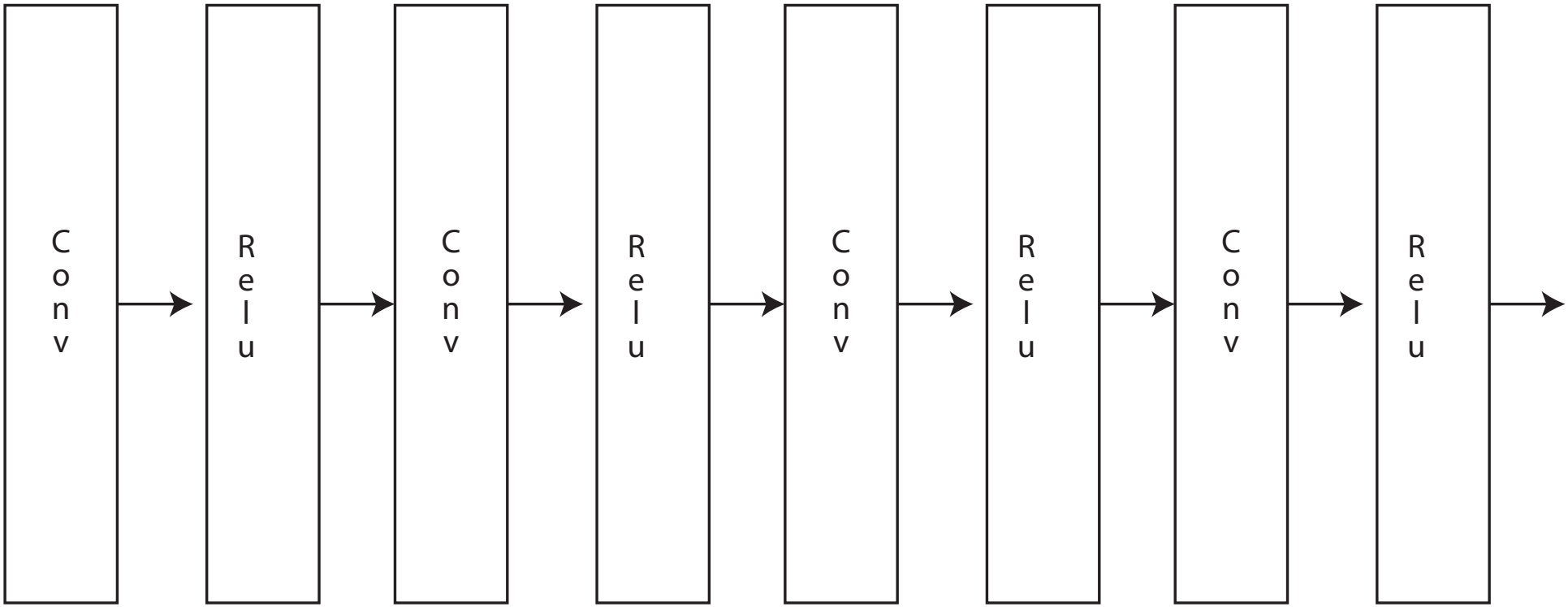


FIGURE 6.15: *On the* **left**, *two kernels (now 3D, as in the text) applied to a set of feature maps produce one new feature map per kernel, using the procedure of the text (the bias term isn't shown). Abstract this as a process that takes an* $x \times y \times d$ *block to an* $X \times Y \times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

Patterns of patterns of patterns….

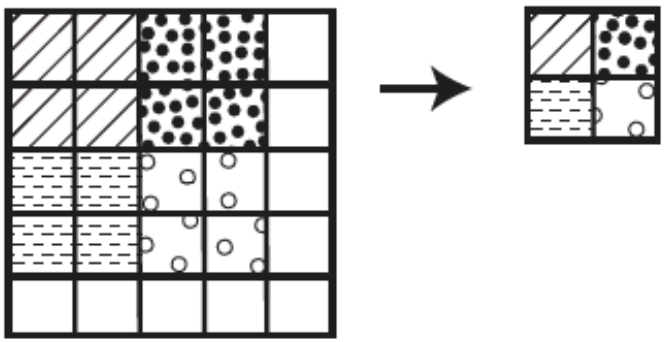


Stride and redundancy

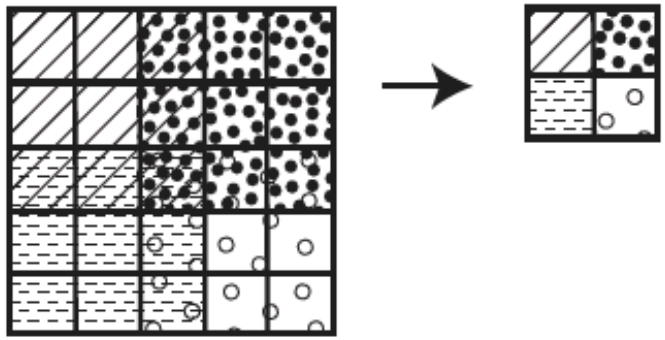
The *receptive field* of a location in a data block (or, equivalently, a unit) is the set of image pixels that affect the value of the location. Usually, all that matters is the size of the receptive field. The receptive field of a location in the first convolutional layer will be given by the kernel 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.*

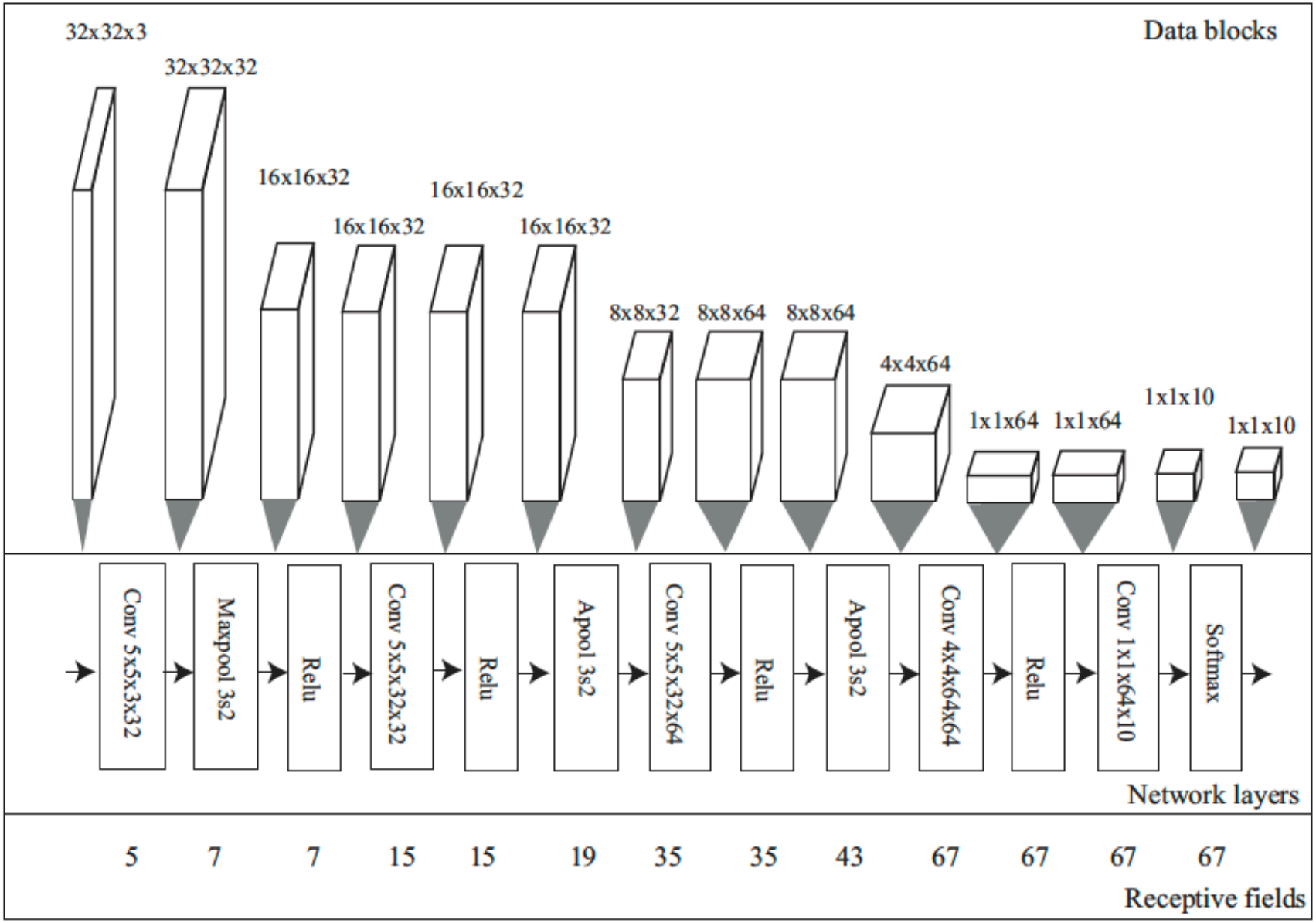


FIGURE 7.3: *Three different representations of the simple network used to classify CIFAR-10 images for this example. Details in the text.*

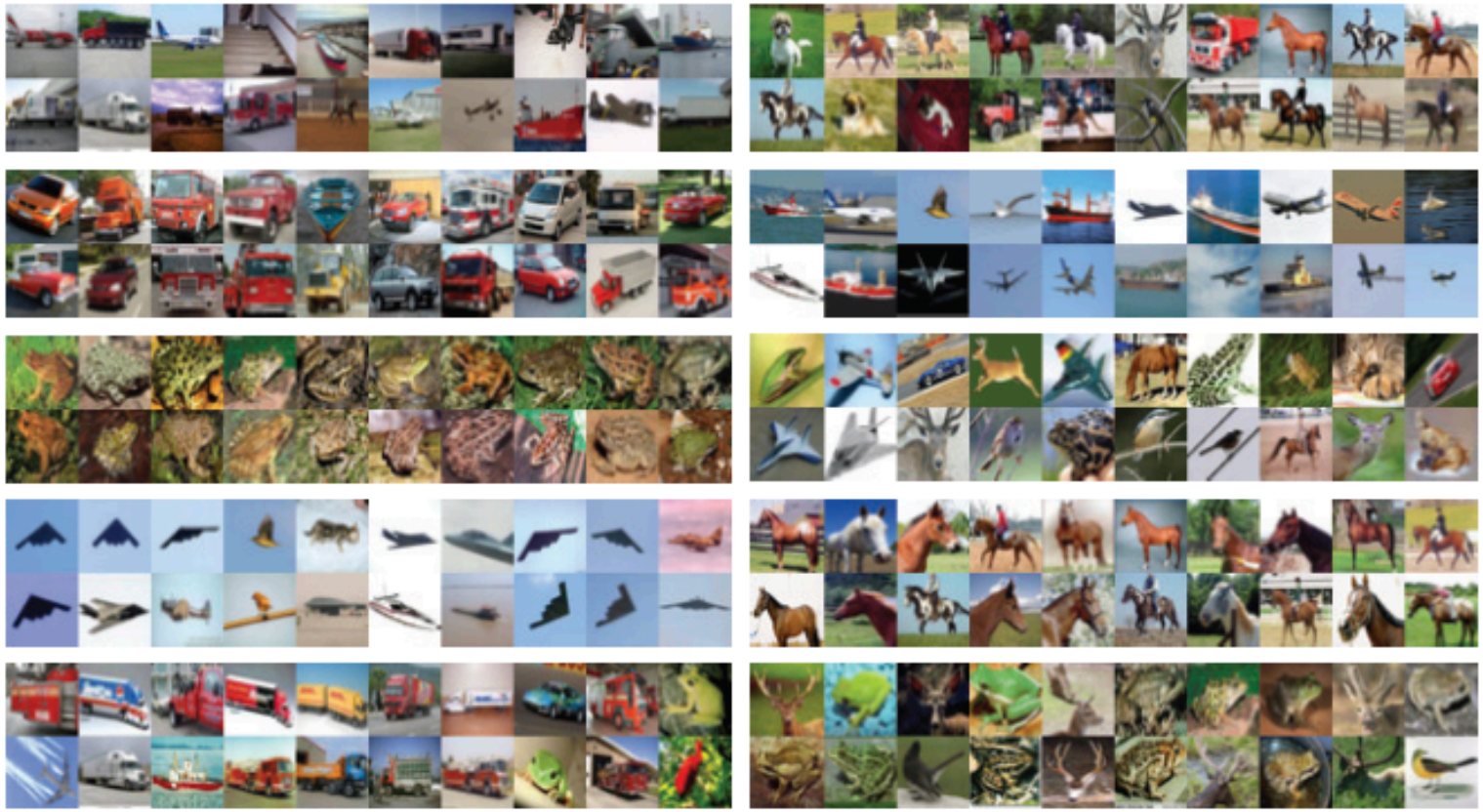


FIGURE 7.7: *Visualizing the patterns that the final stage ReLU's respond to for the simple CIFAR example. Each block of images shows the images that get the largest output for each of 10 ReLU's (the ReLU's were chosen at random from the 64 available in the top ReLU layer). Notice that these ReLU outputs 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.*

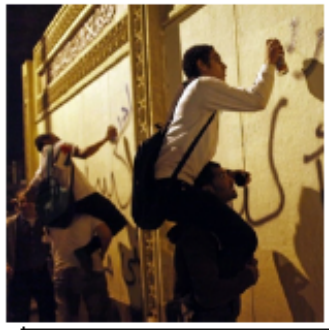
Classification variants

- 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

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

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…

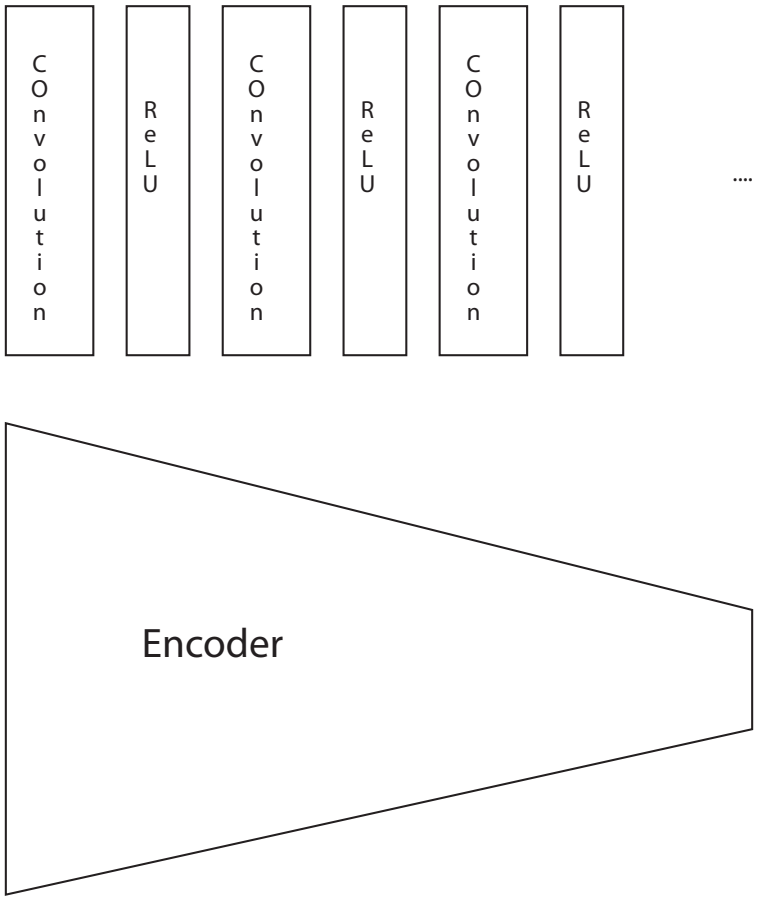
- And scoring system is easily subverted!

A small bird is perched on a branch

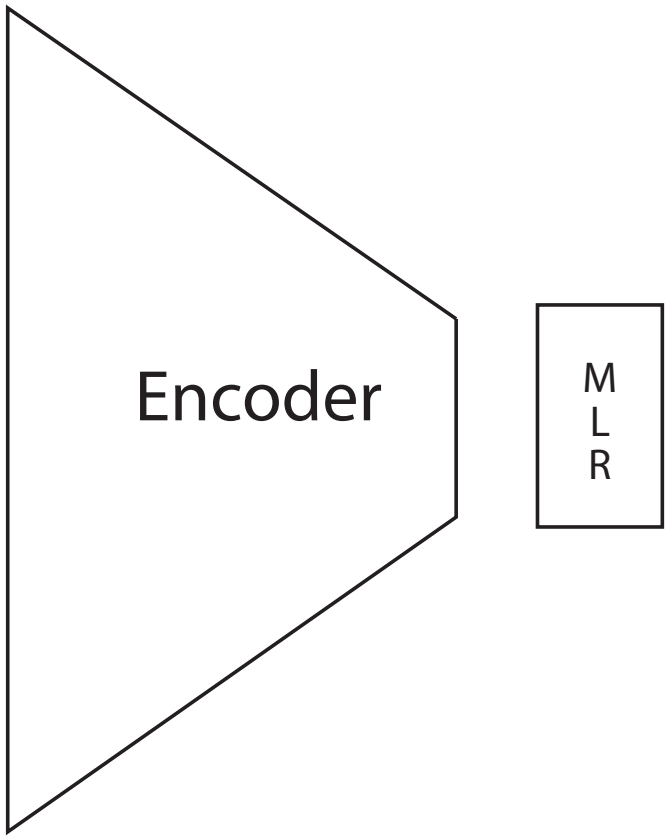
A small brown bear is sitting in the grass

Aneja et al, 2018

Encoders


Convolution
ReLU
Convolution
ReLU
Convolution
ReLU
....
Encoder

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)

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

General strategy

- Construct hierarchy of image regions
  - using a hierarchical segmenter
- Rank regions using a learned score
- Make boxes out of high-ranking regions

- Selective search

Selective search pipeline

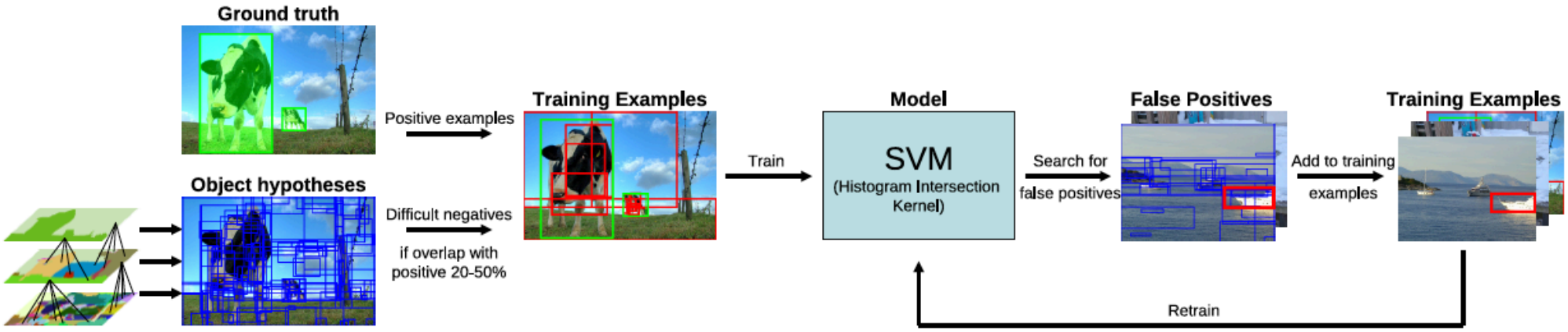


Figure 3: The training procedure of our object recognition pipeline. As positive learning examples we use the ground truth. As negatives we use examples that have a 20-50% overlap with the positive examples. We iteratively add hard negatives using a retraining phase.

Uijlings et al, 12

You need to search at multiple scales

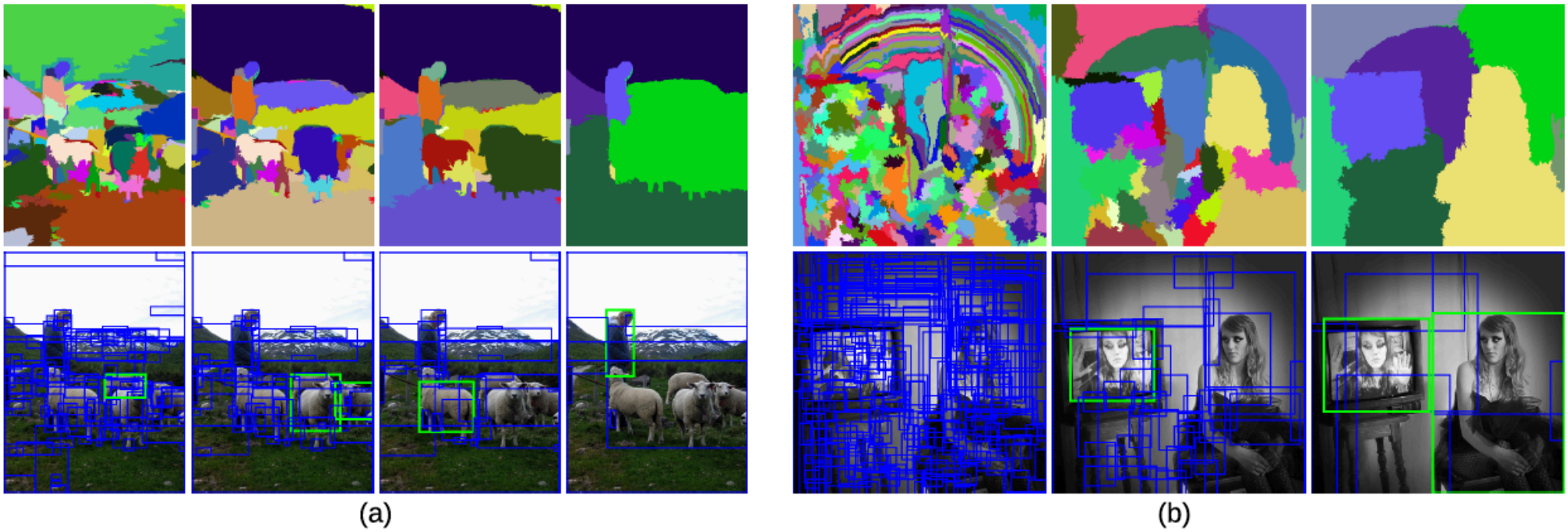


Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

Simplest detector

- Use selective search to propose boxes
- Check with classifier

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

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.

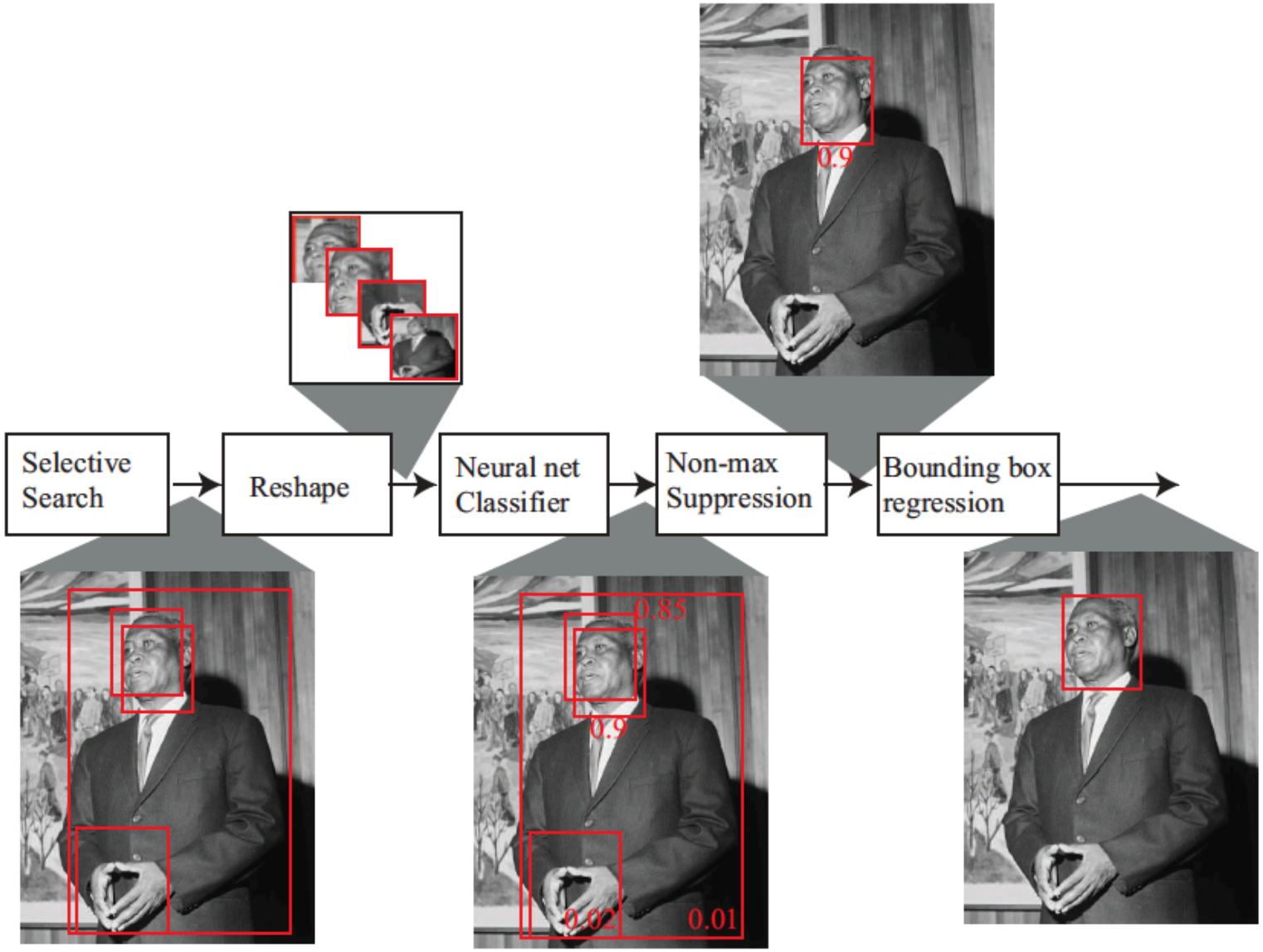


FIGURE 18.6: *A schematic picture of how R-CNN works. A picture of Inkosi Albert Luthuli is fed in to selective search, which proposes possible boxes; these are cut out of the image, and reshaped to fixed size; the boxes are classified (scores next to each box); non-maximum suppression finds high scoring 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.*

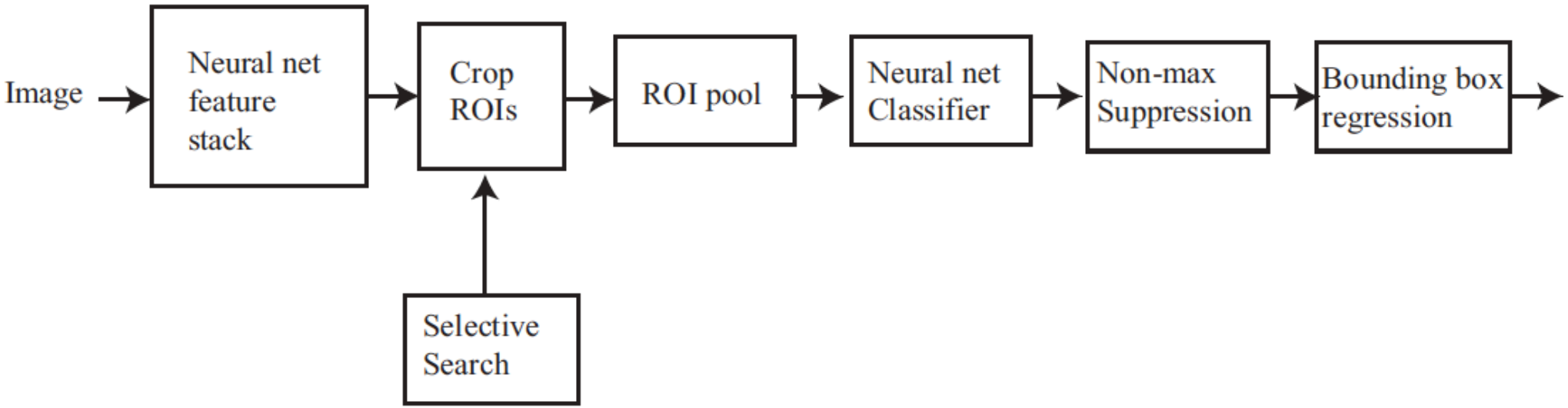
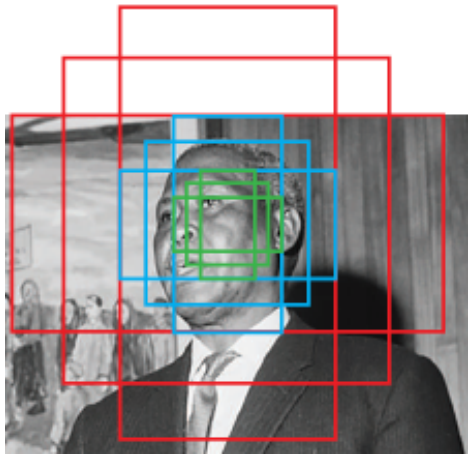


FIGURE 18.7: *Fast R-CNN is much more efficient than R-CNN, because it computes a single feature map from the image, then uses the boxes proposed by selective search to cut regions of interest (ROI's) from it. These are mapped to a standard size by a ROI pooling layer, then presented to a classifier.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



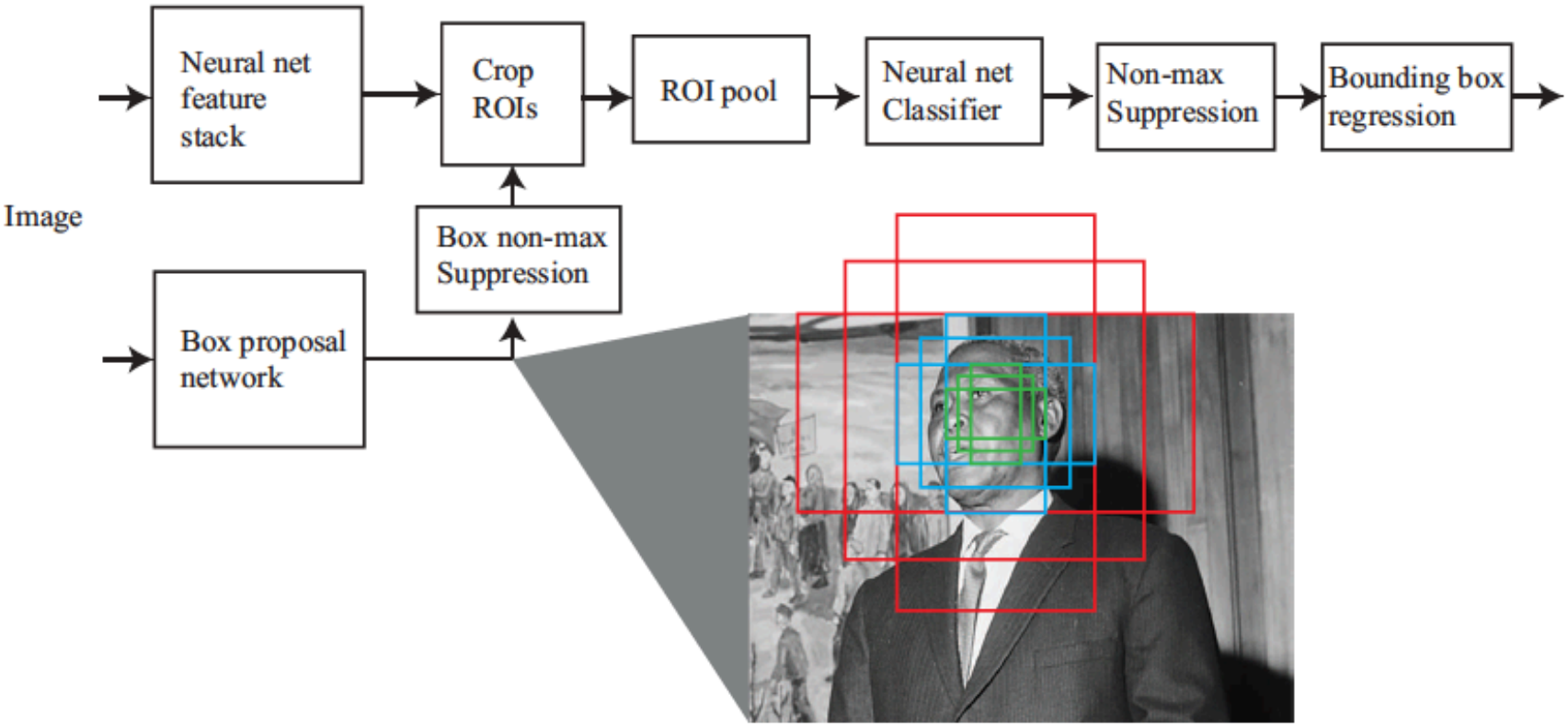
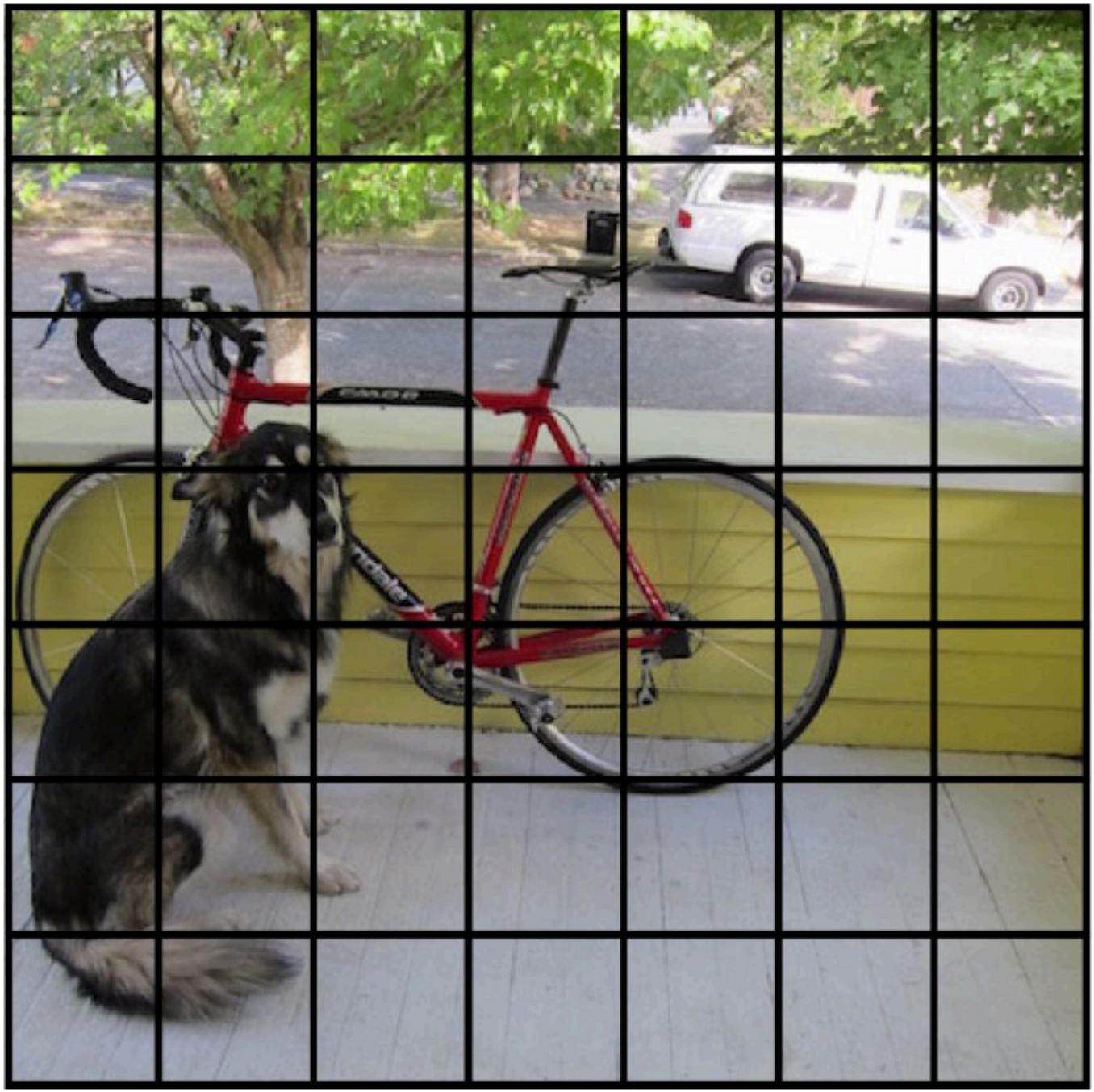


FIGURE 18.8: *Faster RCNN uses two networks. One uses the image to compute "objectness" scores for a sampling of possible image boxes. The samples (called "anchor boxes") are each centered at a grid point. At each grid point, there are nine boxes (three scales, three aspect ratios). The second is a feature 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.*

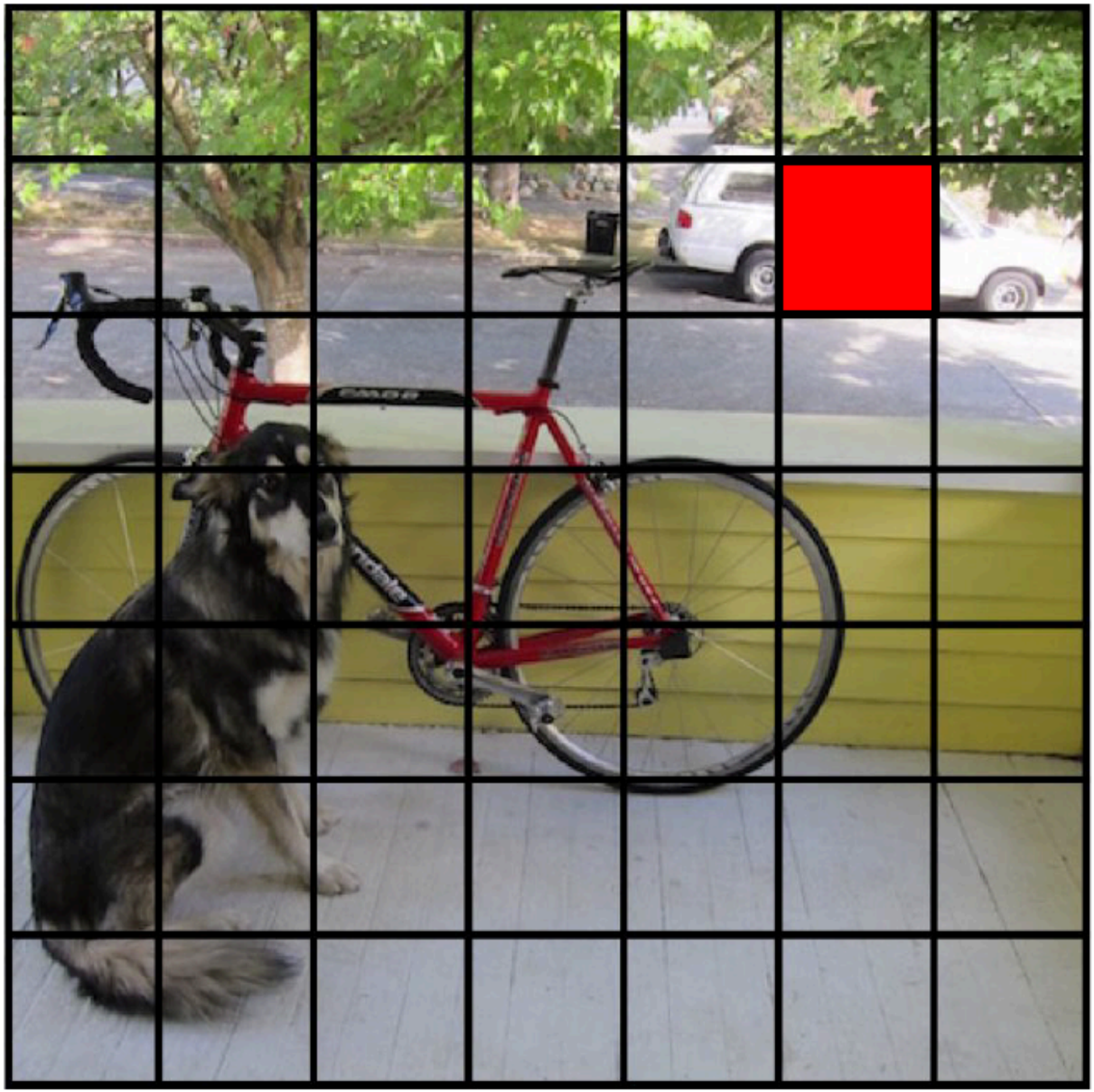
YOLO

- YOLO v3 is about as fast and accurate as you can get
- link on webpage
- key idea
  - look at box scores, label values independently

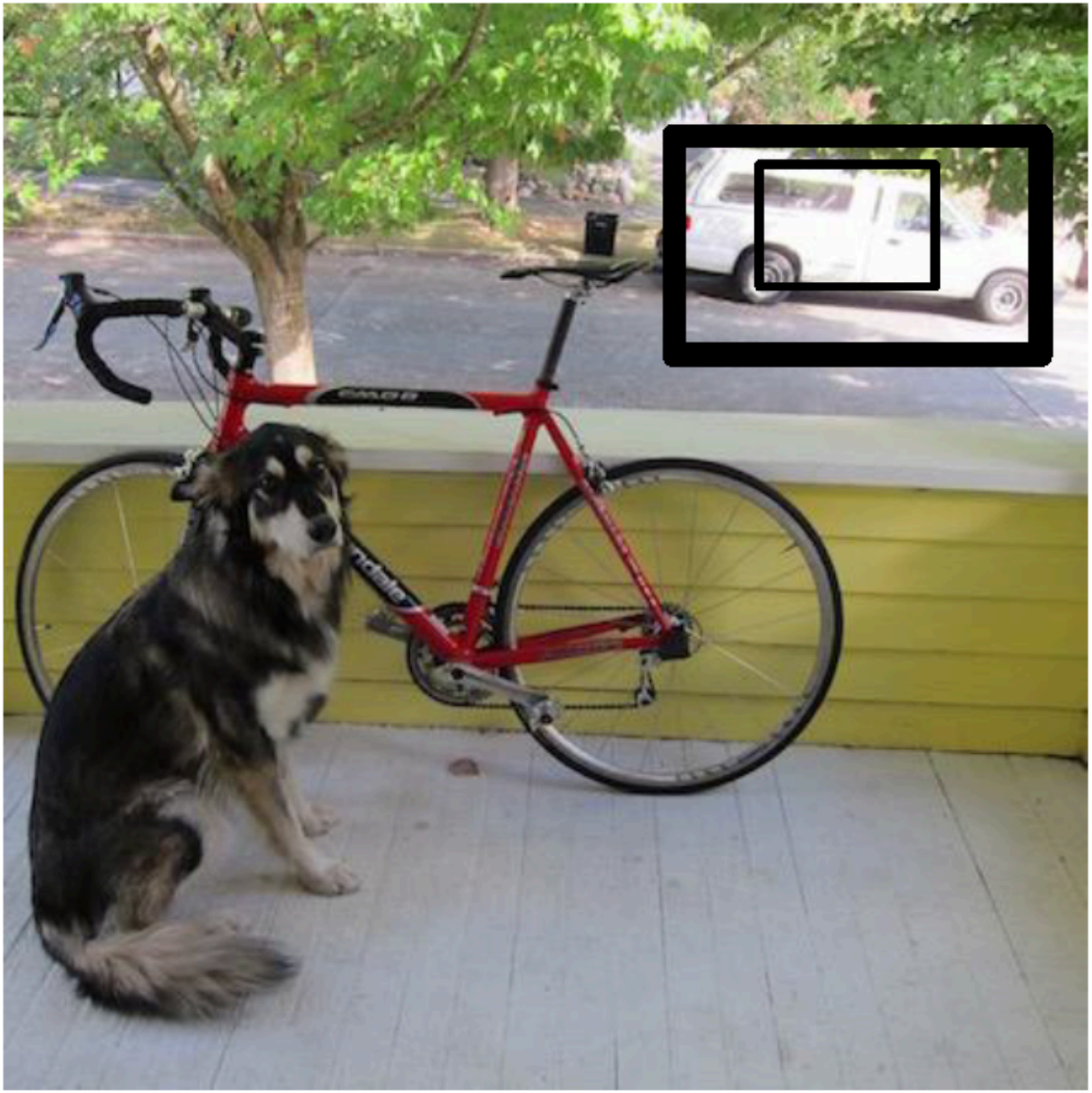
We split the image into a grid



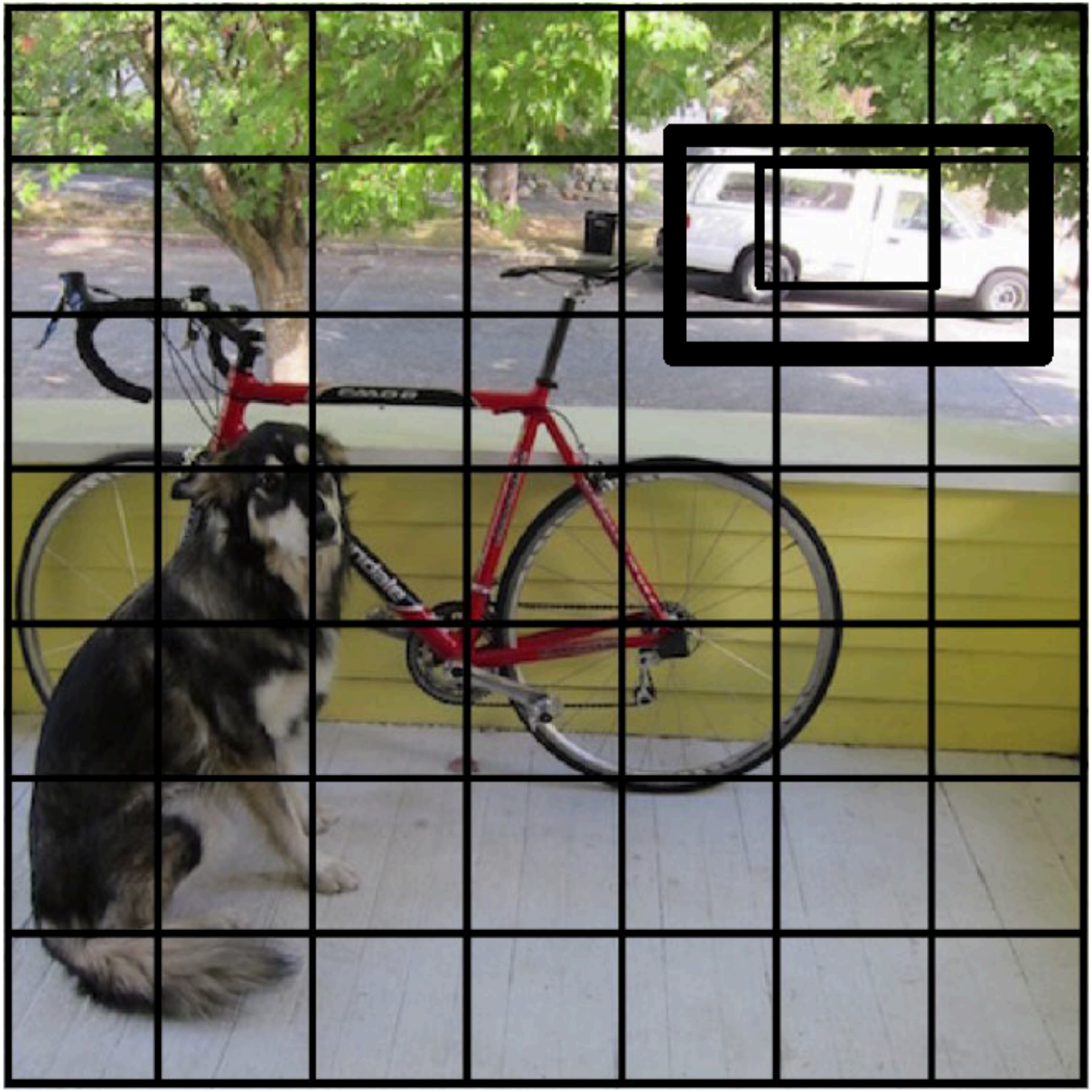
Each cell predicts boxes and confidences: P(Object)



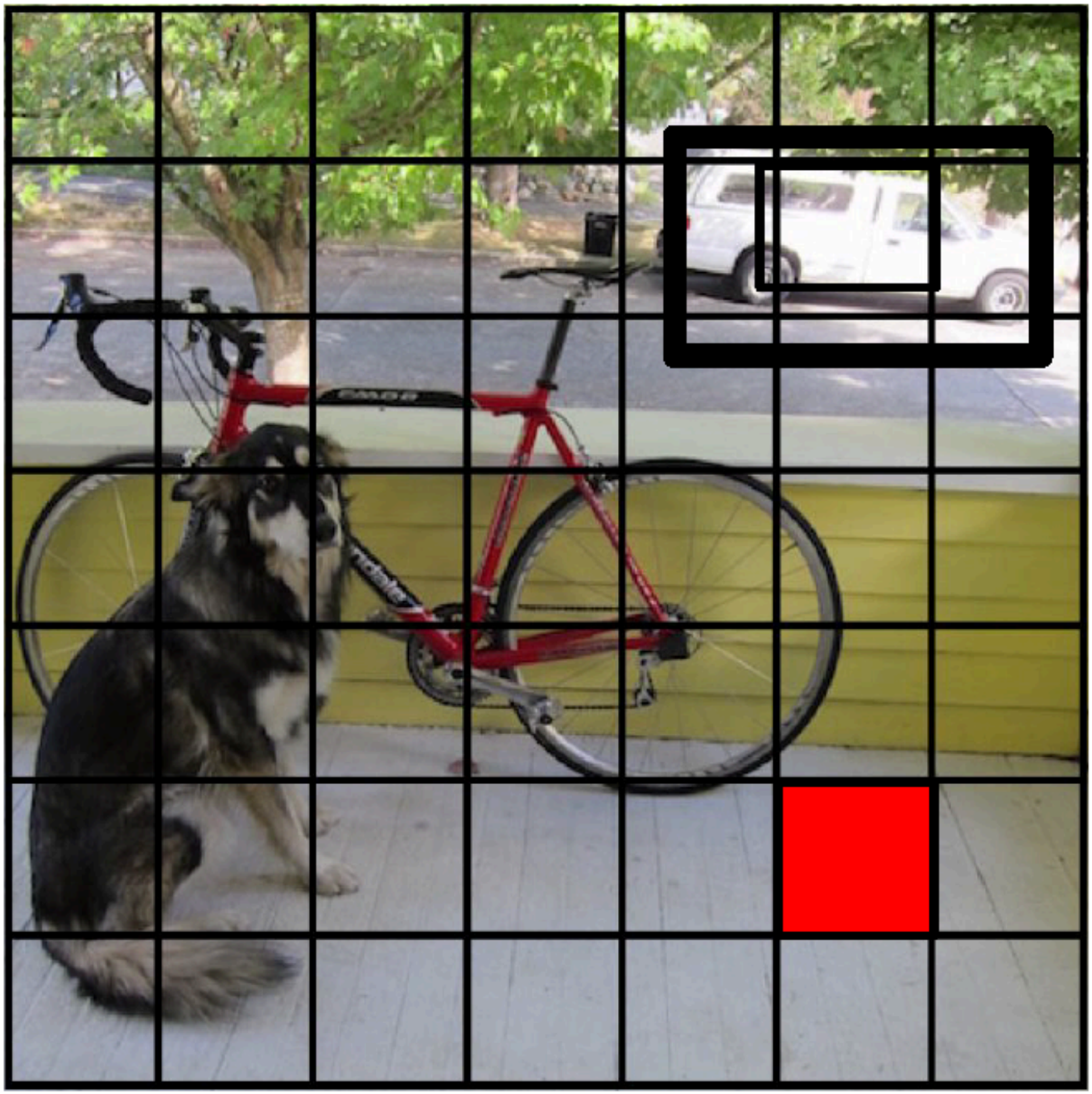
Each cell predicts boxes and confidences: P(Object)



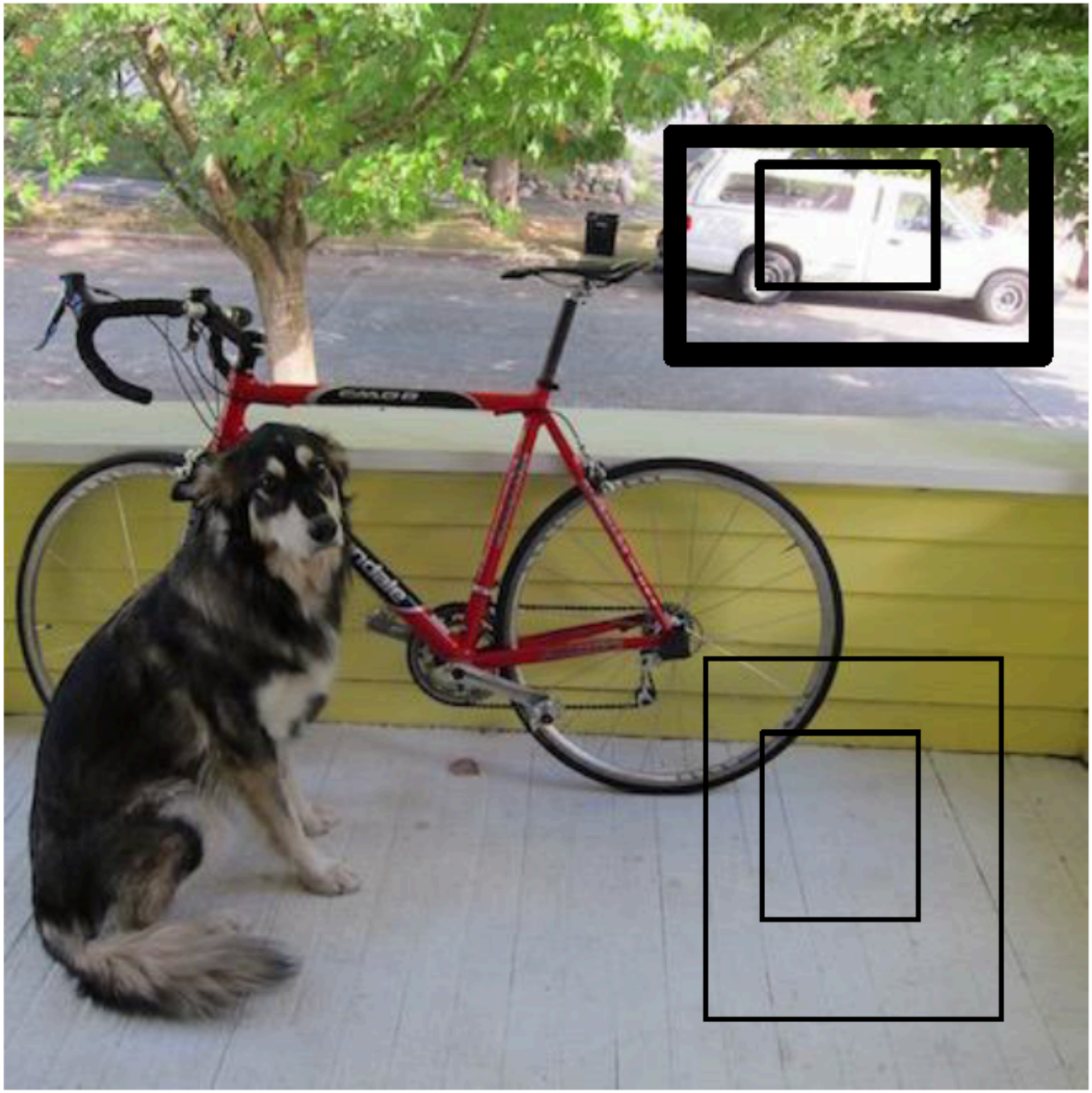
Each cell predicts boxes and confidences: P(Object)



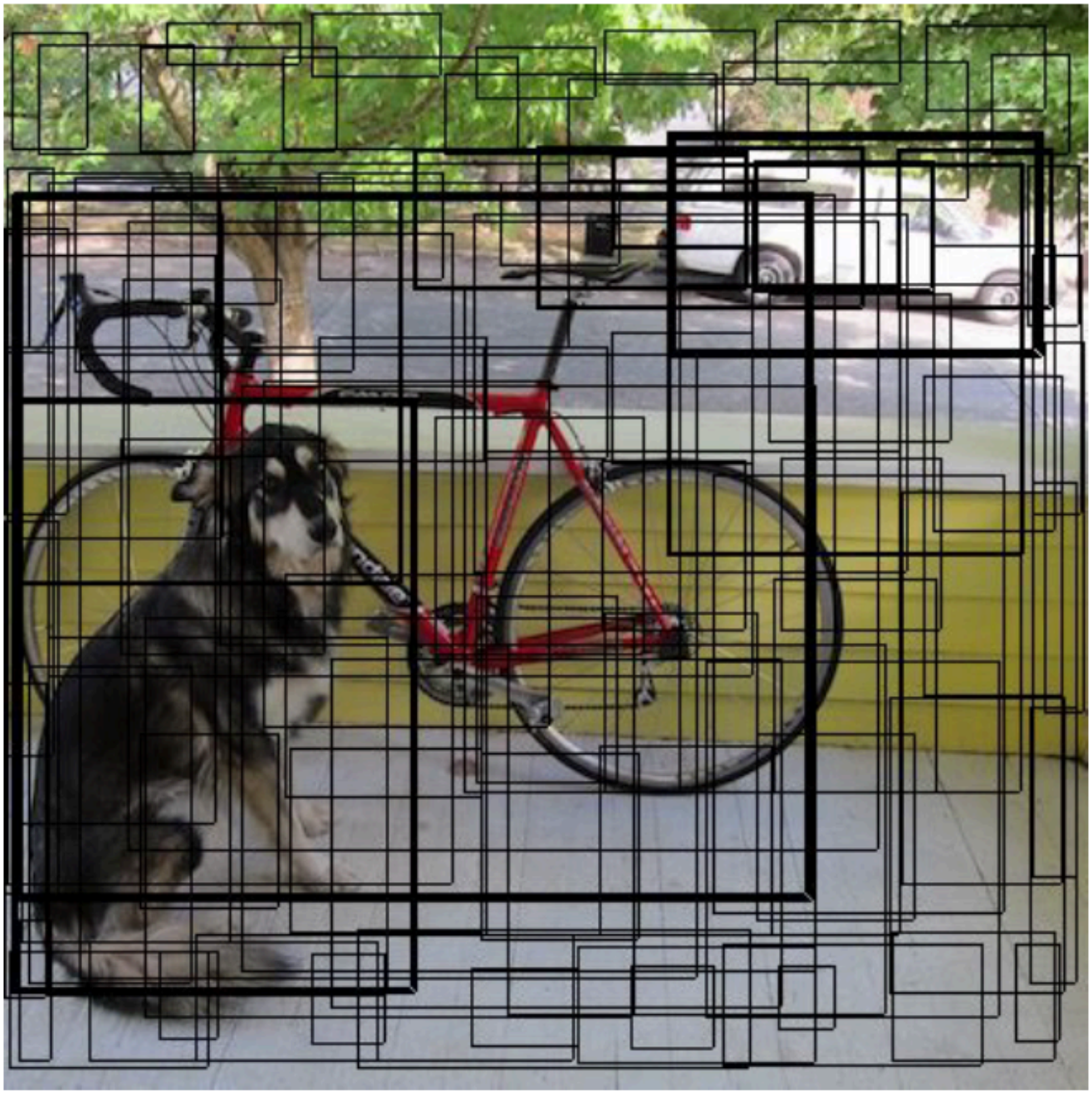
Each cell predicts boxes and confidences: P(Object)



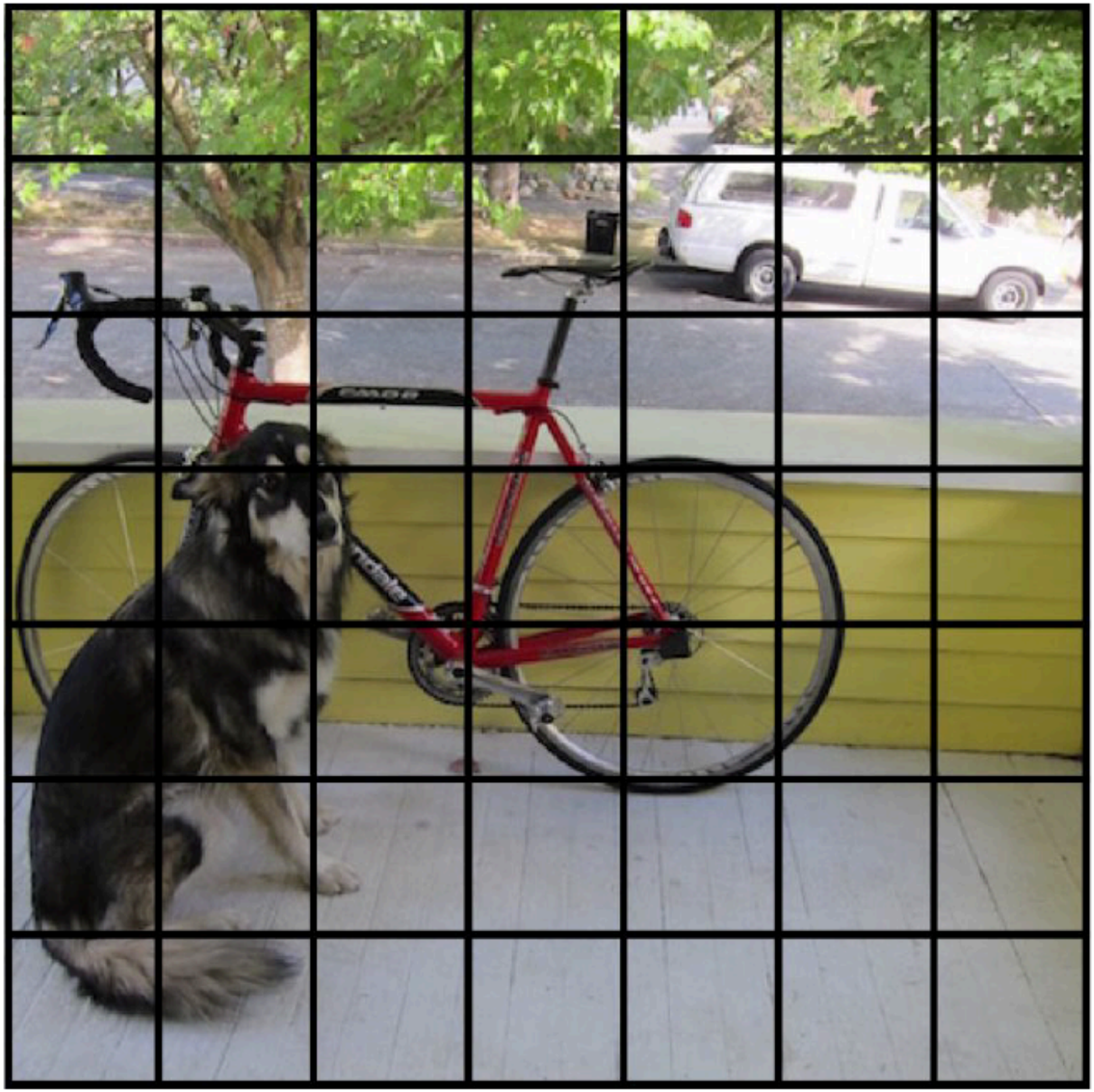
Each cell predicts boxes and confidences: P(Object)



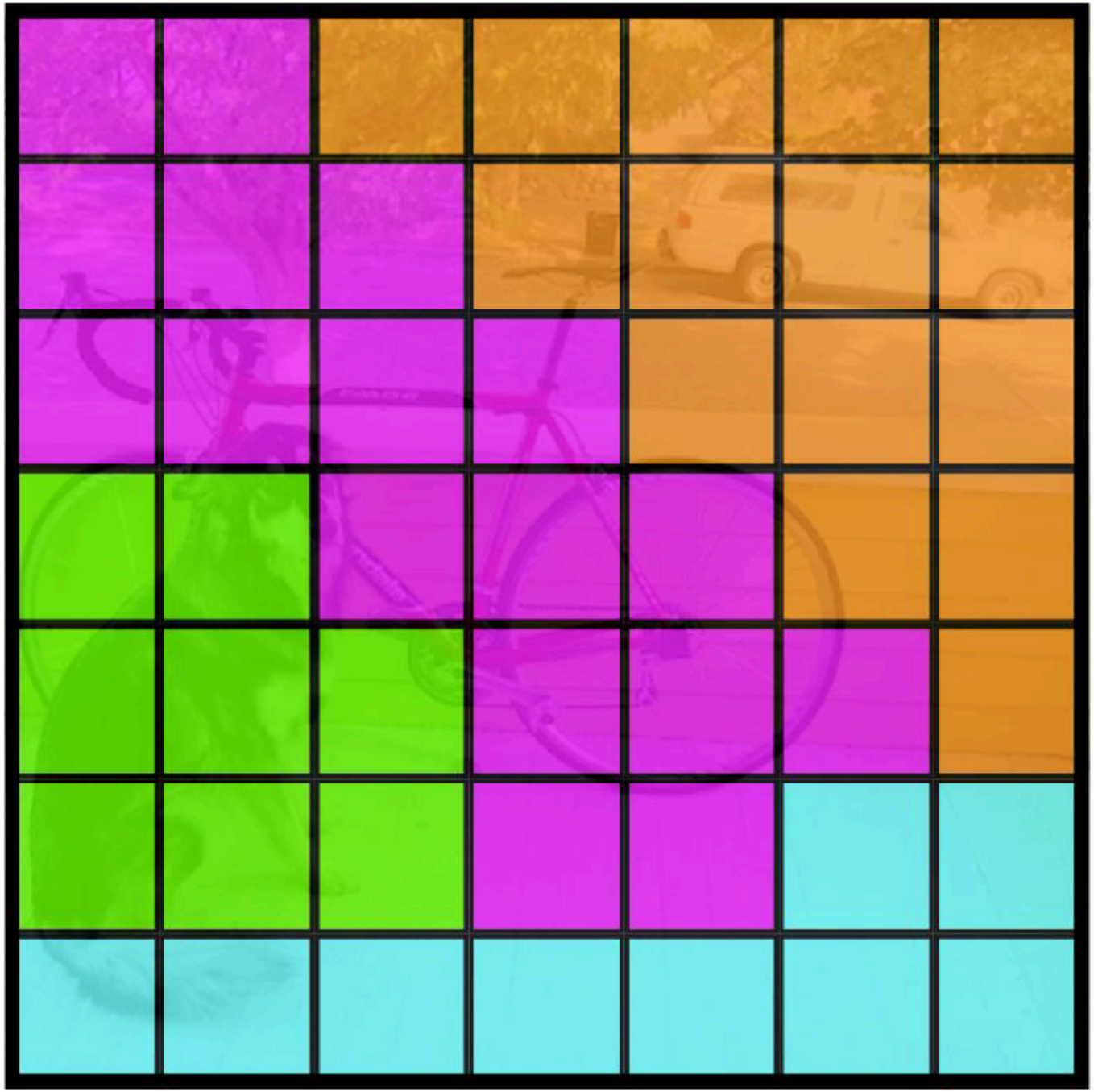
Each cell predicts boxes and confidences: P(Object)



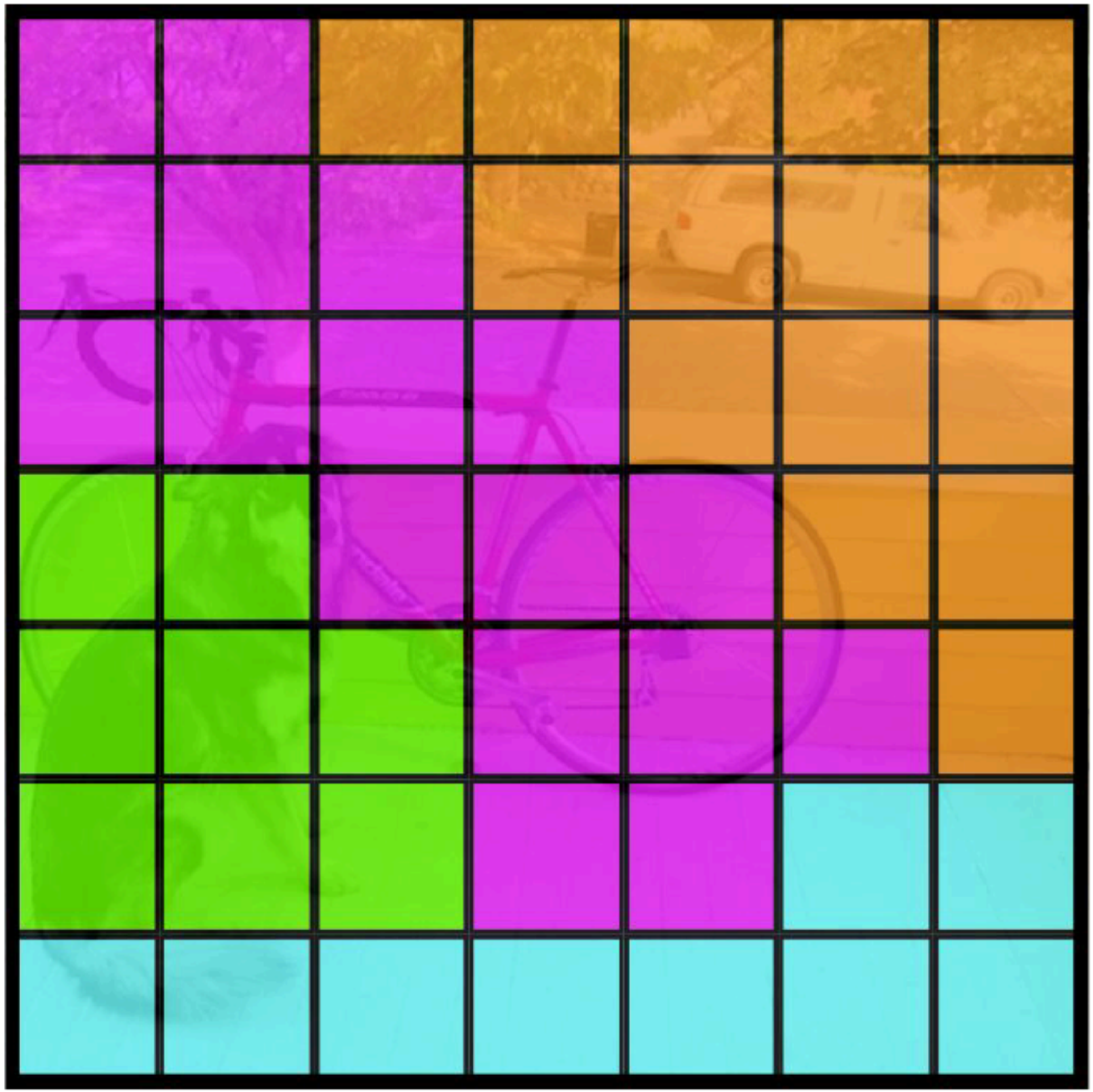
Each cell also predicts a class probability.



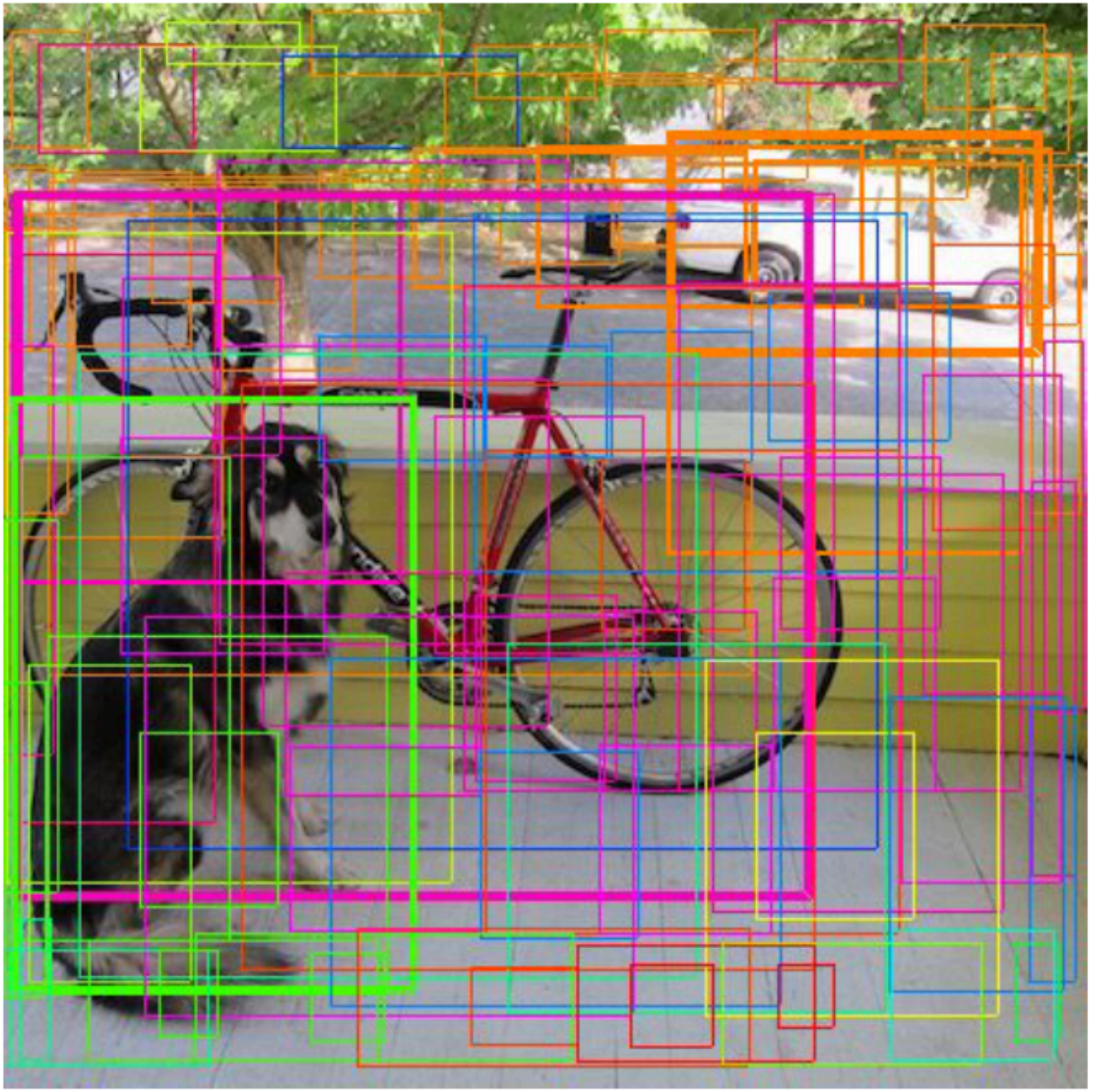
Each cell also predicts a class probability.



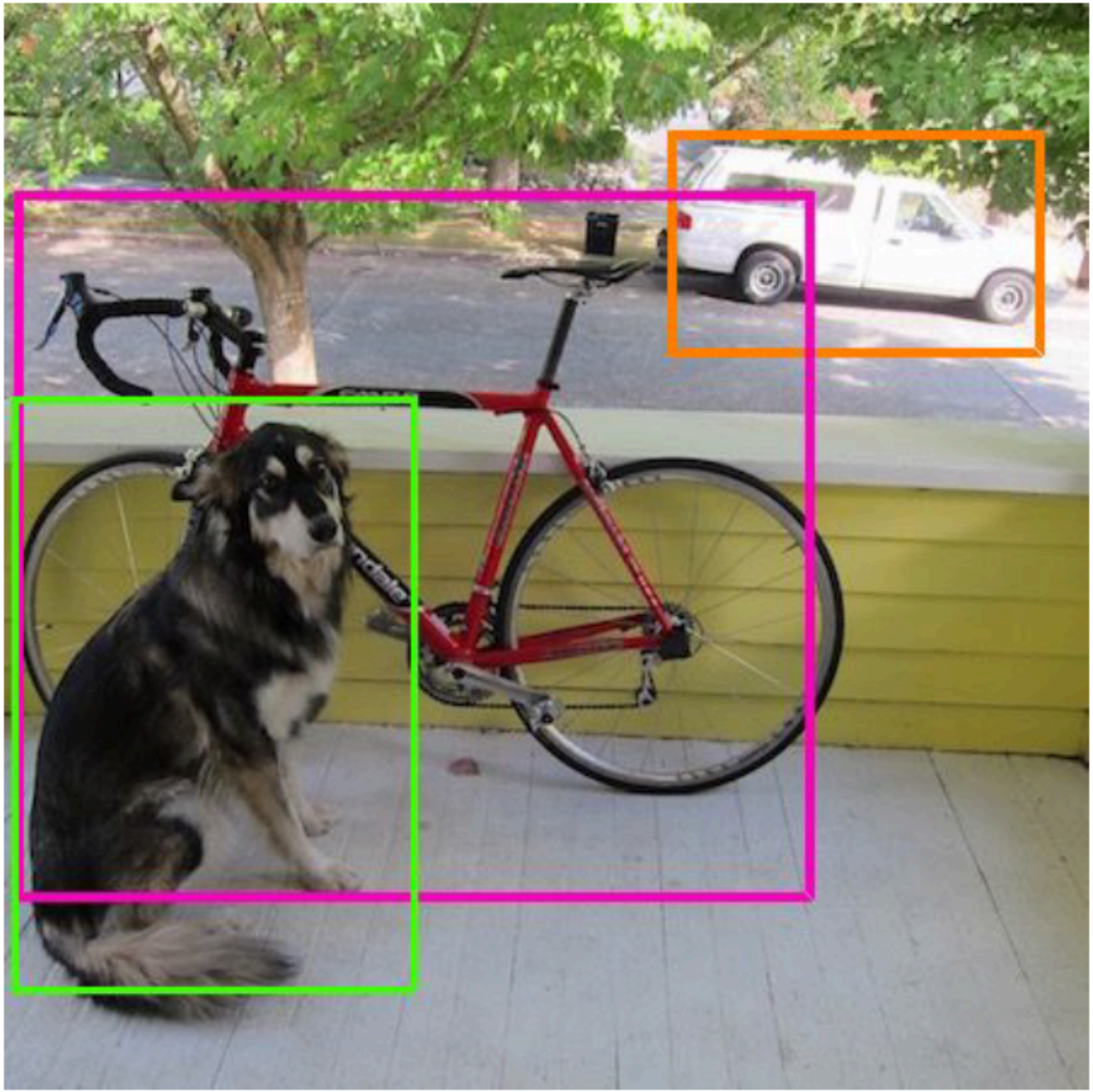
Conditioned on object: P(Car | Object)



Then we combine the box and class predictions.



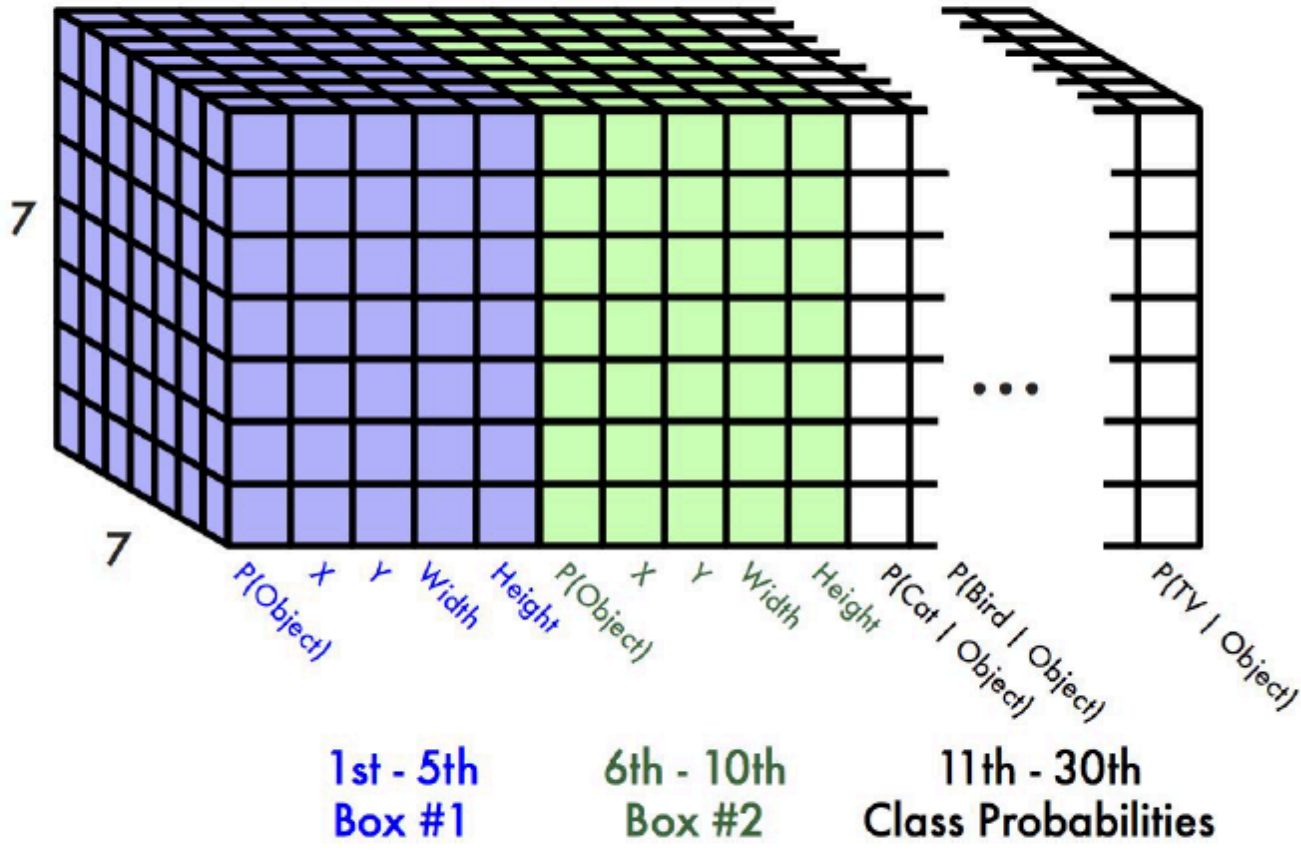
Finally we do NMS and threshold detections



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

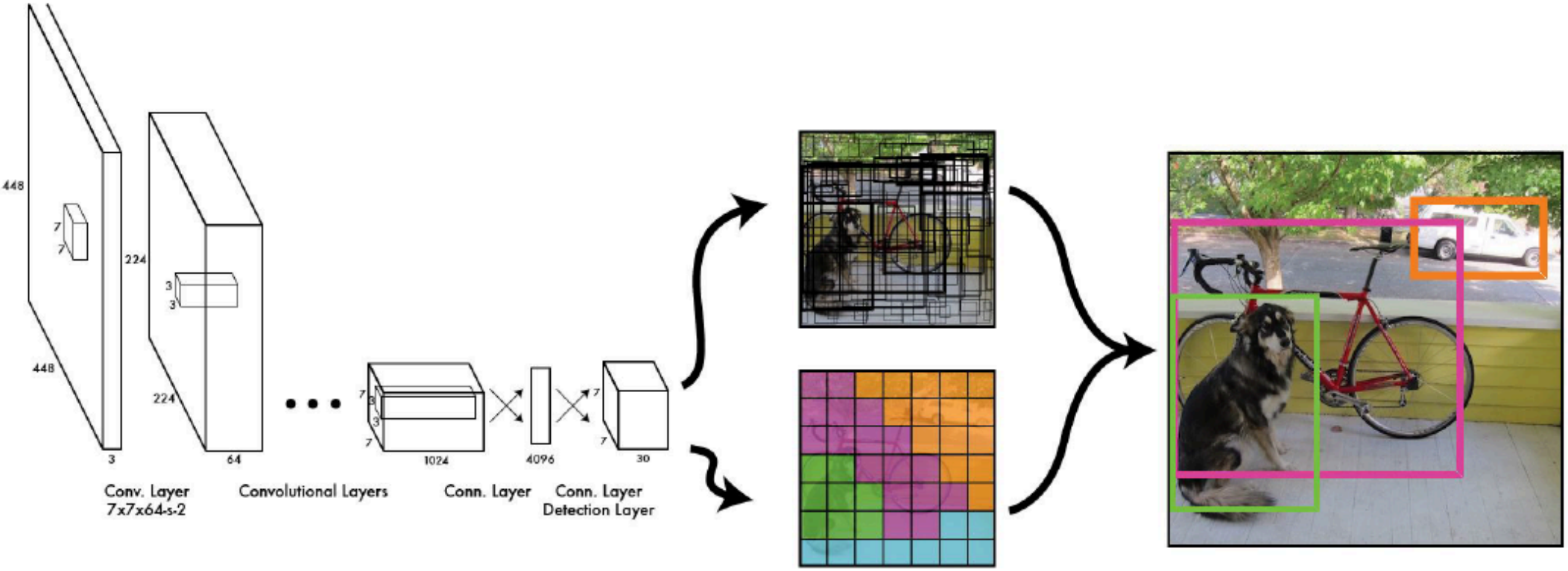


For Pascal VOC:

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

7 x 7 x (2 x 5 + 20) = 7 x 7 x 30 tensor = **1470 outputs**

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



	Pascal 2007 mAP	Speed	
DPM v5	33.7	.07 FPS	14 s/img
R-CNN	66.0	.05 FPS	20 s/img
Fast R-CNN	70.0	.5 FPS	2 s/img
Faster R-CNN	73.2	7 FPS	140 ms/img
YOLO	~~63.4~~ 69.0	45 FPS	22 ms/img

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

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

Application: Lane boundary detection

- HUH?!?!?
- Not even in “Computer Vision for Autonomous Vehicles”
  - (recent review by Janai et al - very good)
- Lane boundaries are very important
  - lots of money in good lane boundary detection
  - easy cases are firmly solved; hard cases remain hard
- Interplay of detection, geometry
  - variance and bias
- Firmly scene understanding

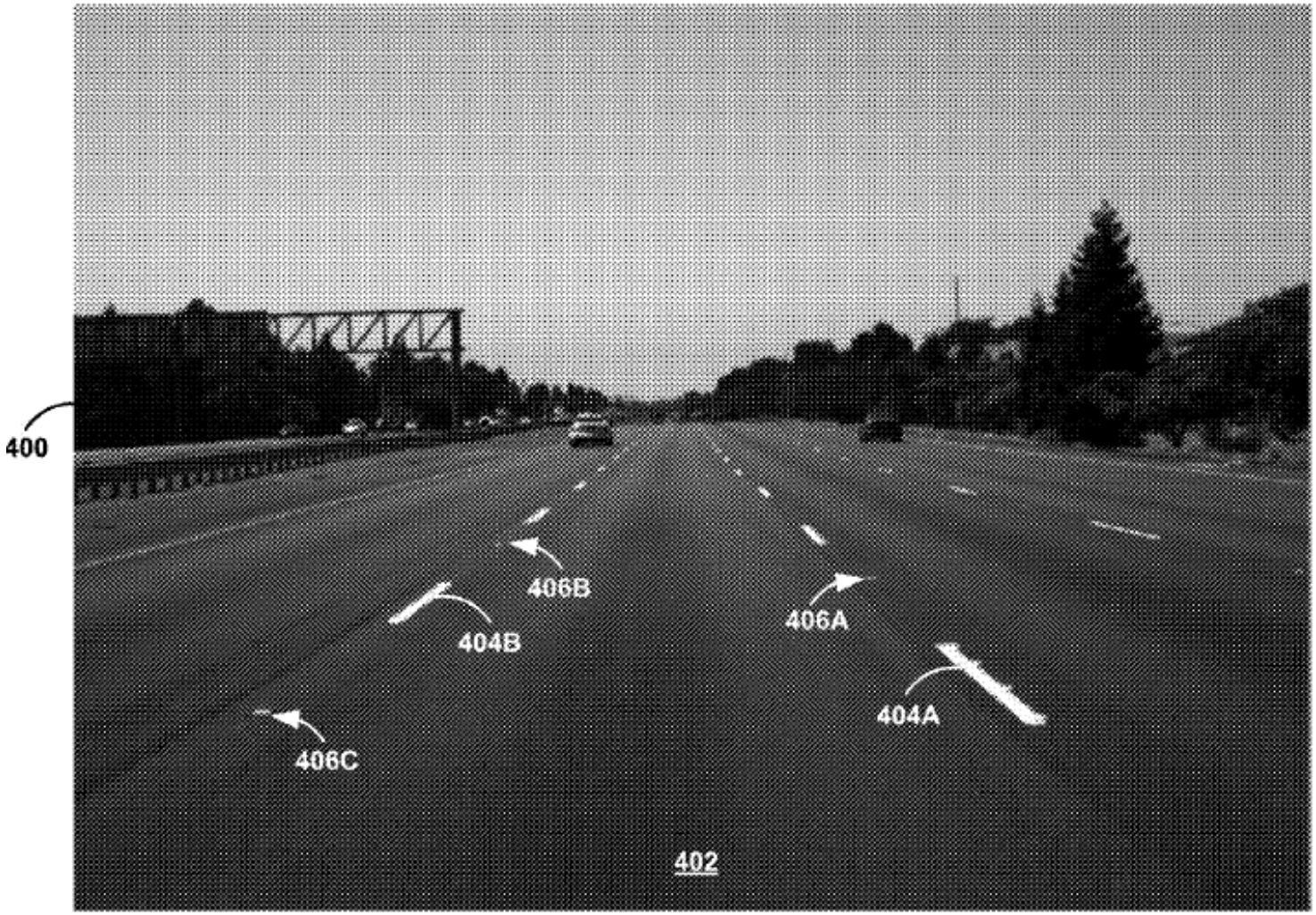


FIGURE 4A

Strategy: detect markers (reflective paint), join up exercise in robust fitting of curves

US 9081385

Issues

- You have to do it fast
- You have to do it right
- Paint detection problems
- Geometric model problems

Labelled data methods

- Generally, rack up a labelled dataset and regress
  - Datasets
    - Oxford lane boundaries
      - https://oxford-robotics-institute.github.io/road-boundaries-dataset/
    - CULane
      - https://xingangpan.github.io/projects/CULane.html
    - CalTech
      - http://www.mohamedaly.info/datasets/caltech-lanes
    - TUSimple
      - https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection

Simple marker method

- Place markers on lane boundaries
  - organized into lanes (colors)
- Notice
  - datasets contain lanes, not marker locations

Simple marker method

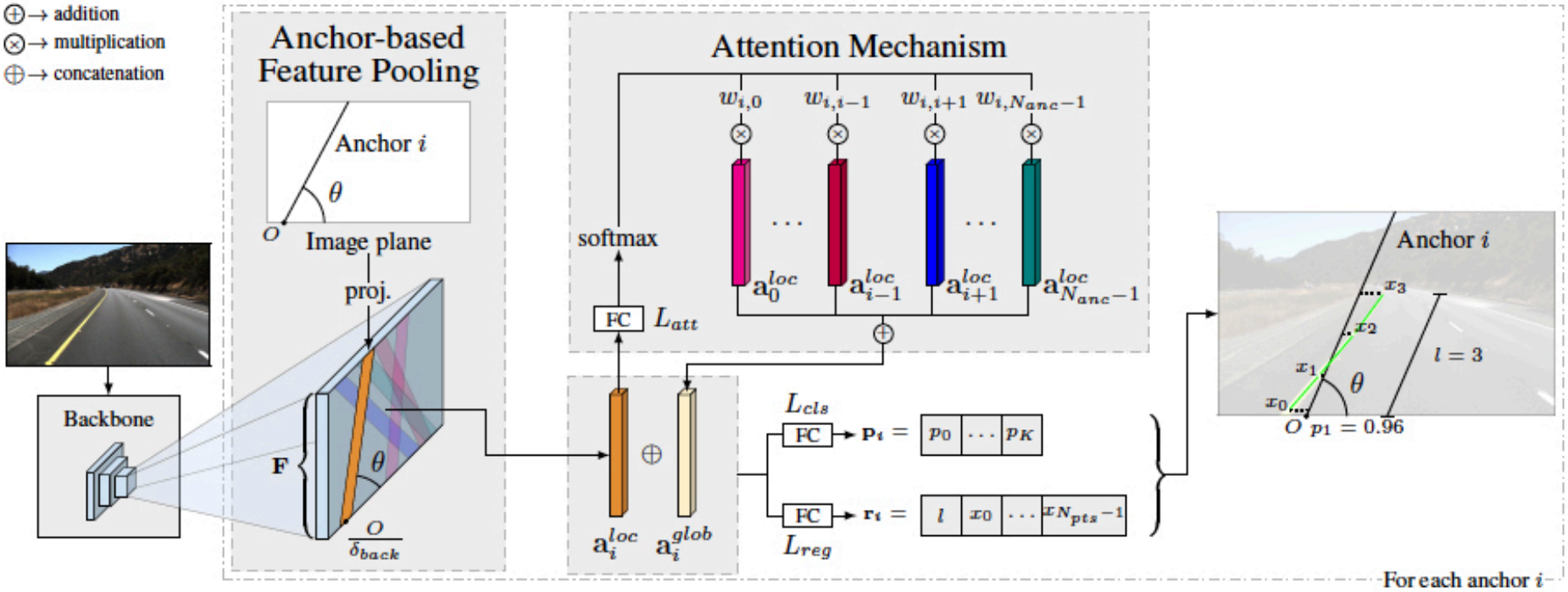


Figure 1. Overview of the proposed method. A backbone generates feature maps from an input image. Subsequently, each anchor is projected onto the feature maps. This projection is used to pool features that are concatenated with another set of features created in the attention module. Finally, using this resulting feature set, two layers, one for classification and another for regression, make the final predictions.

Tabelini et al 21

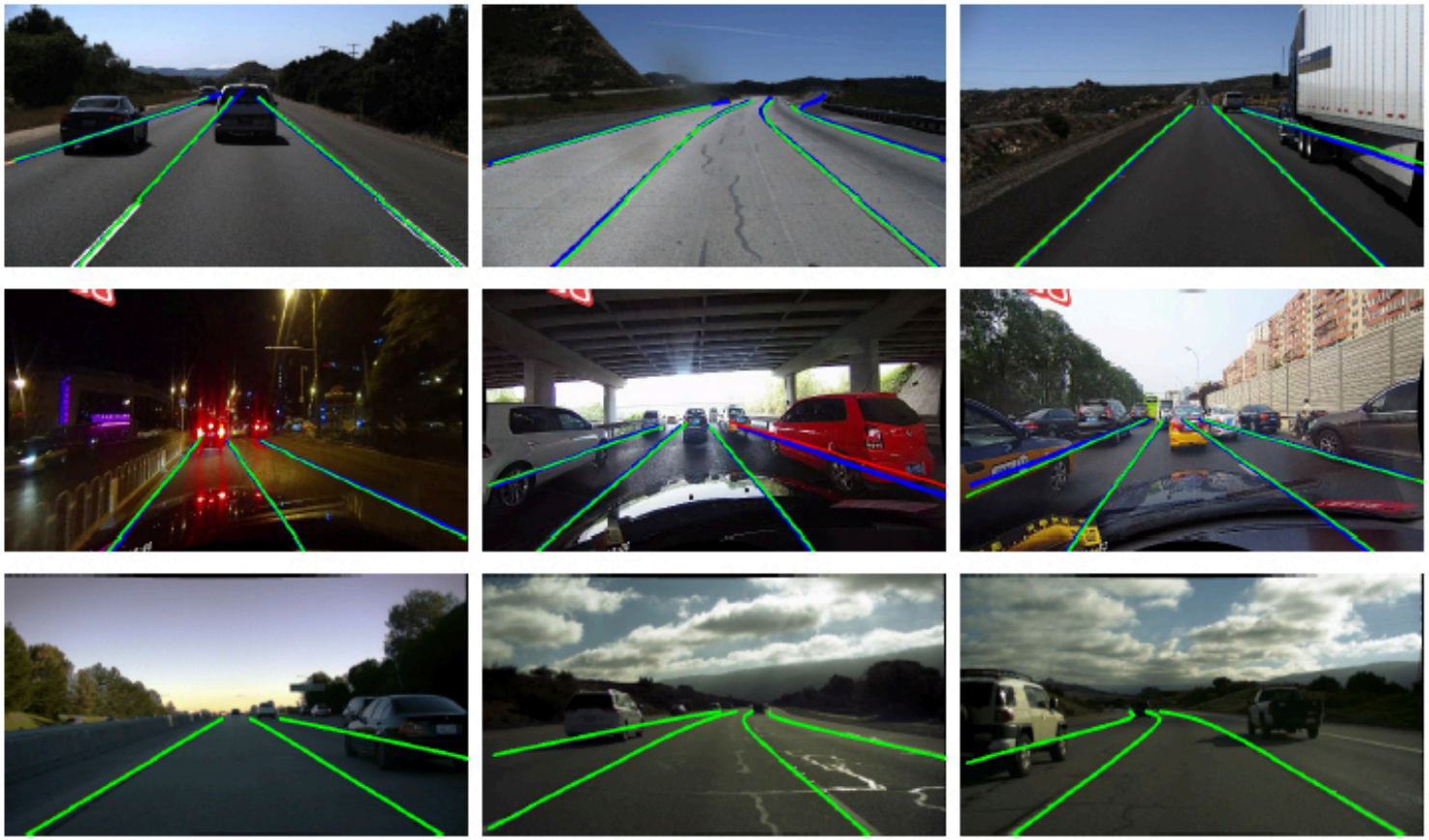


Figure 2. LaneATT qualitative results on TuSimple (top row), CU-Lane (middle row), and LLAMAS (bottom row). Blue lines are ground-truth, while green and red lines are true-positives and false-positives, respectively. See more samples in the videos[1].

Tabelini et al 21

It’s fast

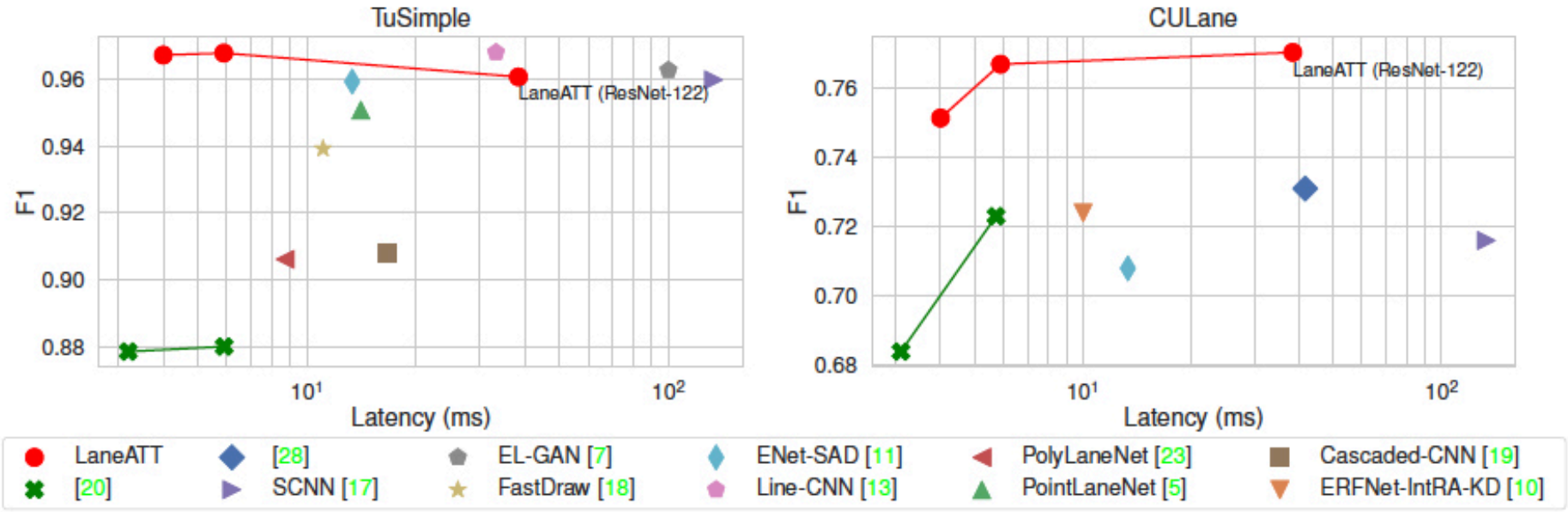


Figure 3. Model latency vs. F1 of state-of-the-art methods on CULane and TuSimple.

Tabelini et al 21

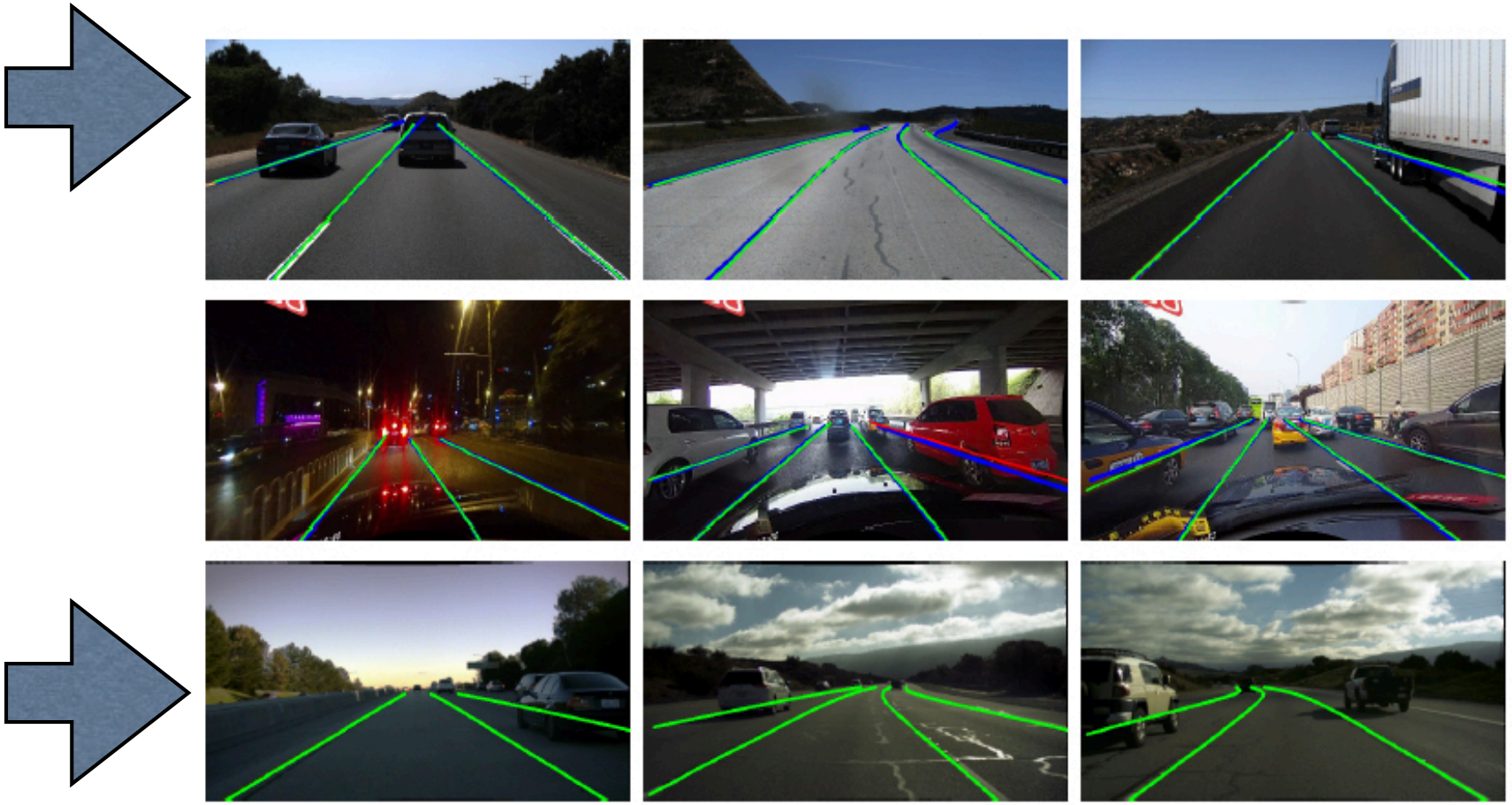


Figure 2. LaneATT qualitative results on TuSimple (top row), CU-Lane (middle row), and LLAMAS (bottom row). Blue lines are ground-truth, while green and red lines are true-positives and false-positives, respectively. See more samples in the videos[1].

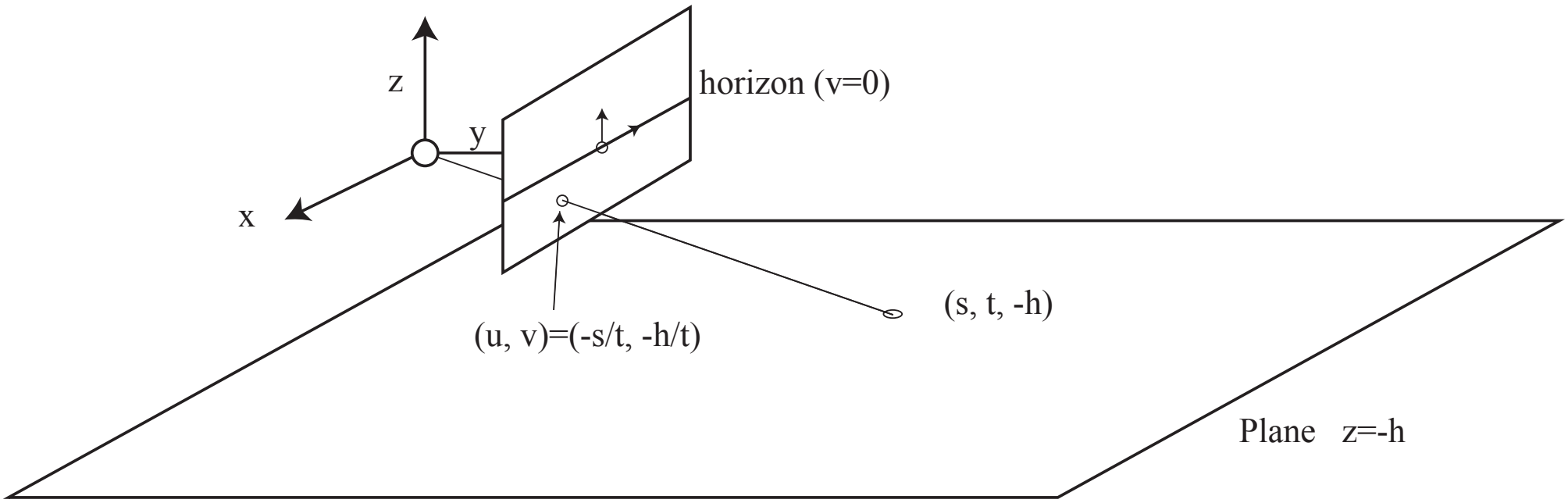
Tabelini et al 21

Khan et al 20

- Strategy
  - detect keypoints in image
  - rectify
    - using estimated horizon from vanishing points
  - impose structural model on keypoints in rectified image

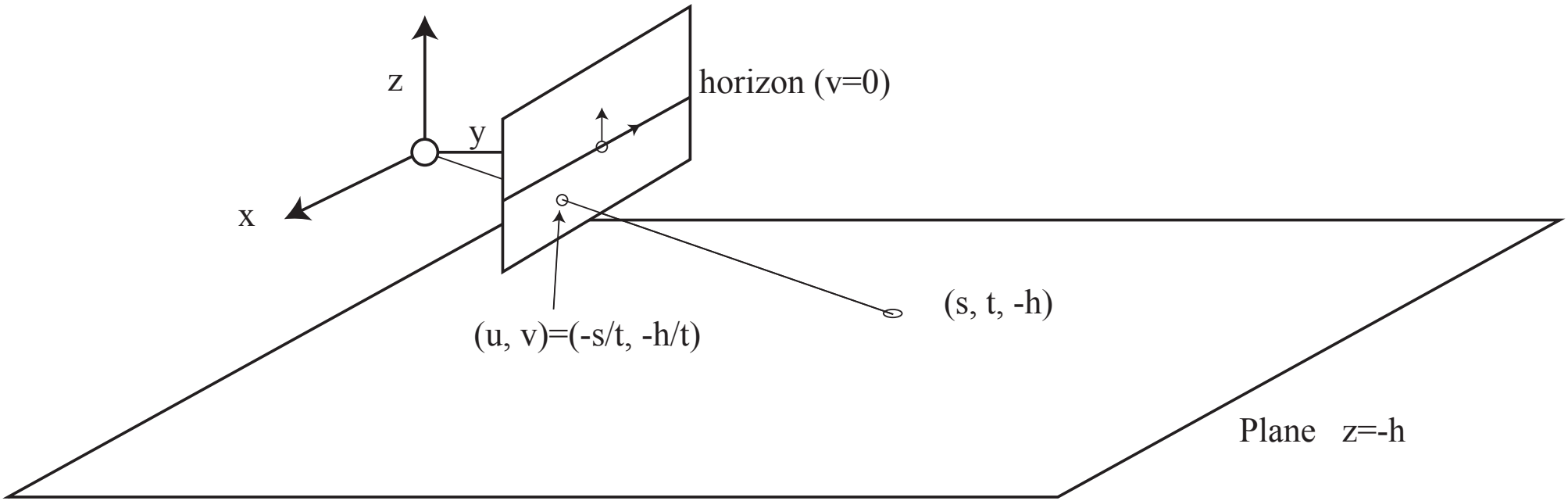
Rectification

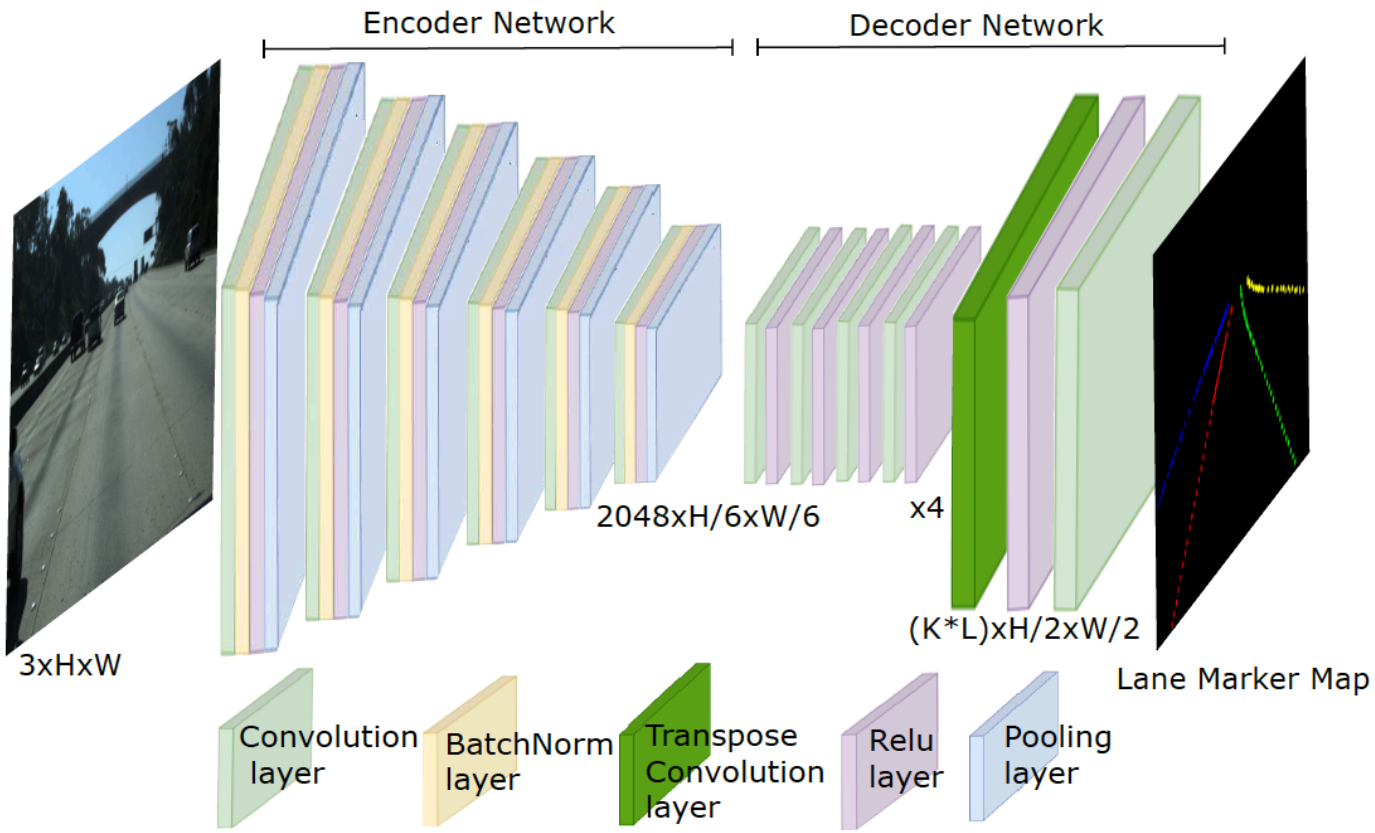
- Imagine a camera at a fixed height
  - moving rigidly over a textured ground plane
  - bottom half of image is distorted ground plane texture
  - If we know the camera, we can map image plane texture to ground plane



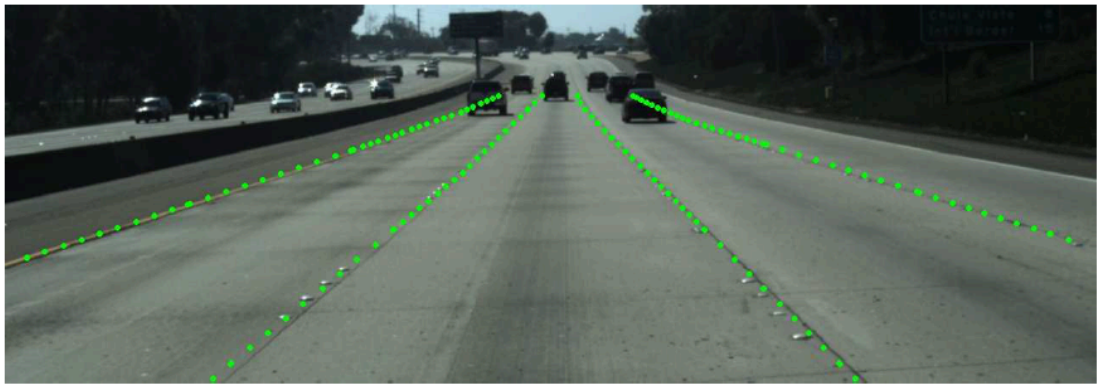
Estimating the camera

- Height
  - from car (calibrated and known)
- Roll and pitch
  - from horizon
    - roll is why horizon isn't parallel to image plane
    - pitch is why it isn't centerline





(a)



(b)

Figure 2. (a) The figure shows the architecture of the lane boundary marker network. (b) The sampled keypoints from the ground truth lane line are shown here.

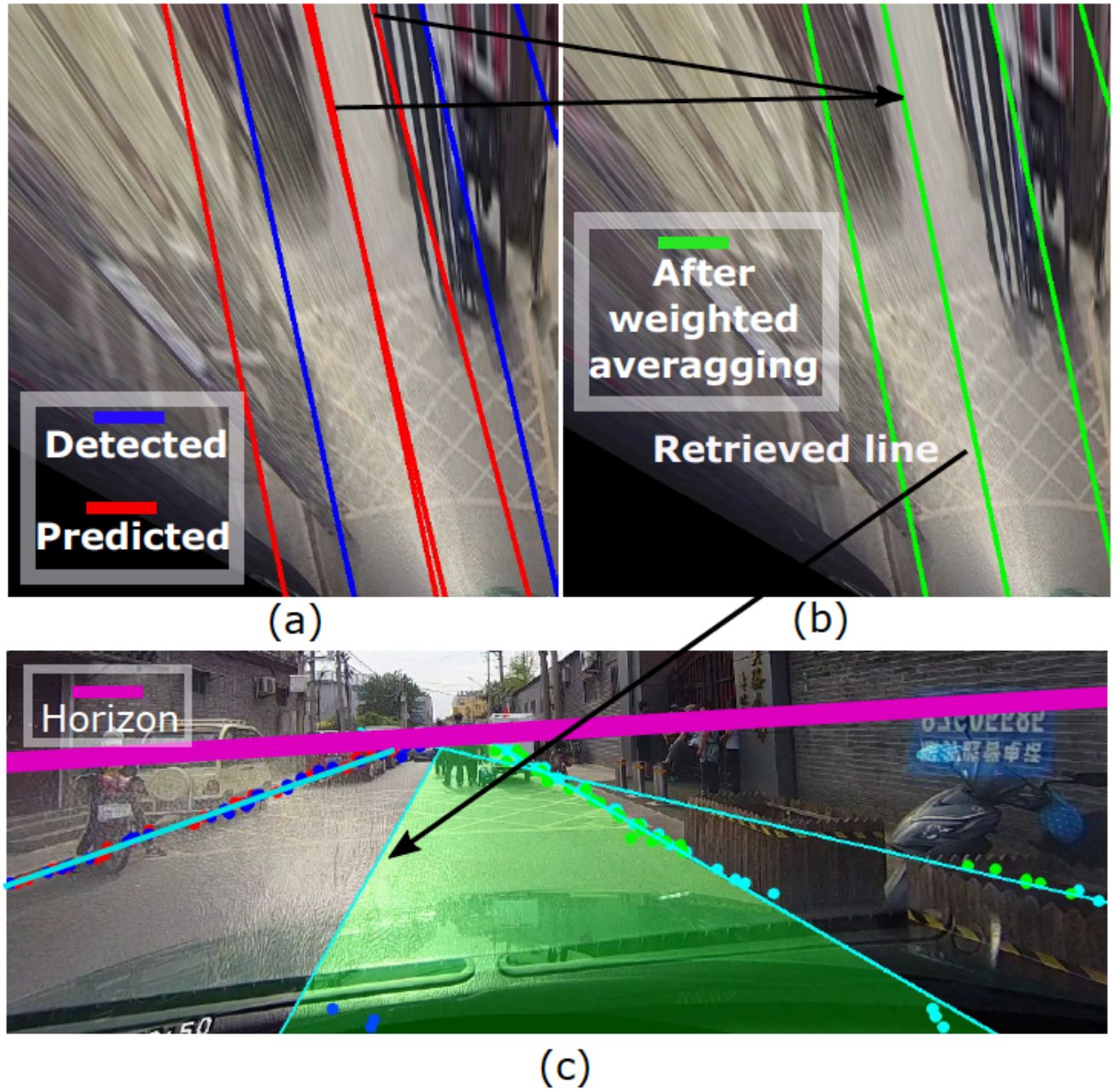


Figure 5. Detecting missing lane boundries. (a) Rectified view. Lane boundaries are predicted in red using c_o from section 3.3.2. (b) Filtered lane boundaries after weighted averaging (c) Recovered perspective view with all the four lane boundaries.

Key point: it's easier to impose a geometric model on keypoints in rectified frame.

Geometric model

- Lane markers lie on
  - a quadratic curve OR a straight line (in rectified frame)
    - fitted using a version of RANSAC
  - lane boundaries are parallel
    - lines - easy
    - curves - look at tangents
    - Q: why not use a (latent) center curve?
- Search:
  - There are four boundaries (three lanes)
    - or three
    - or two

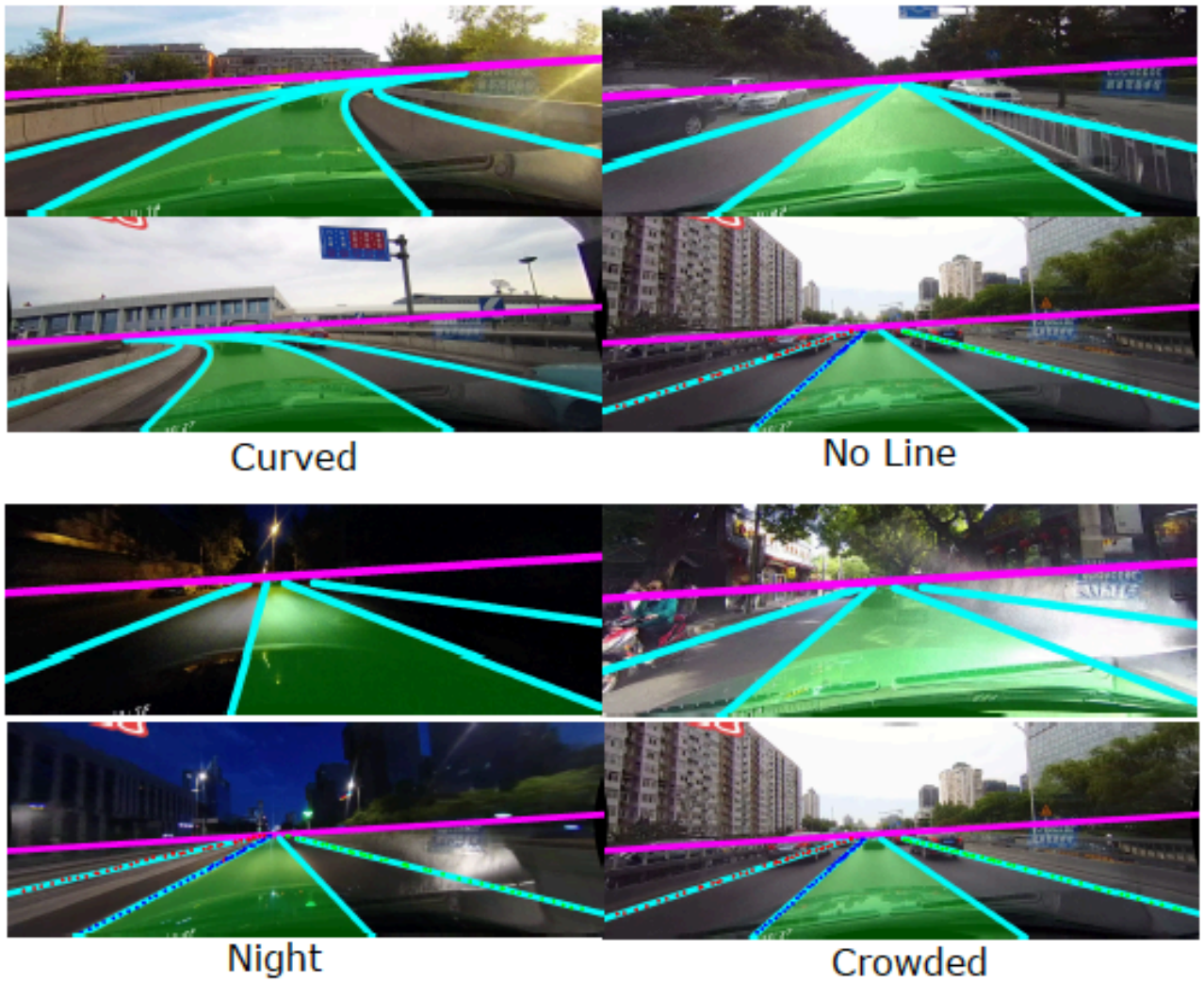


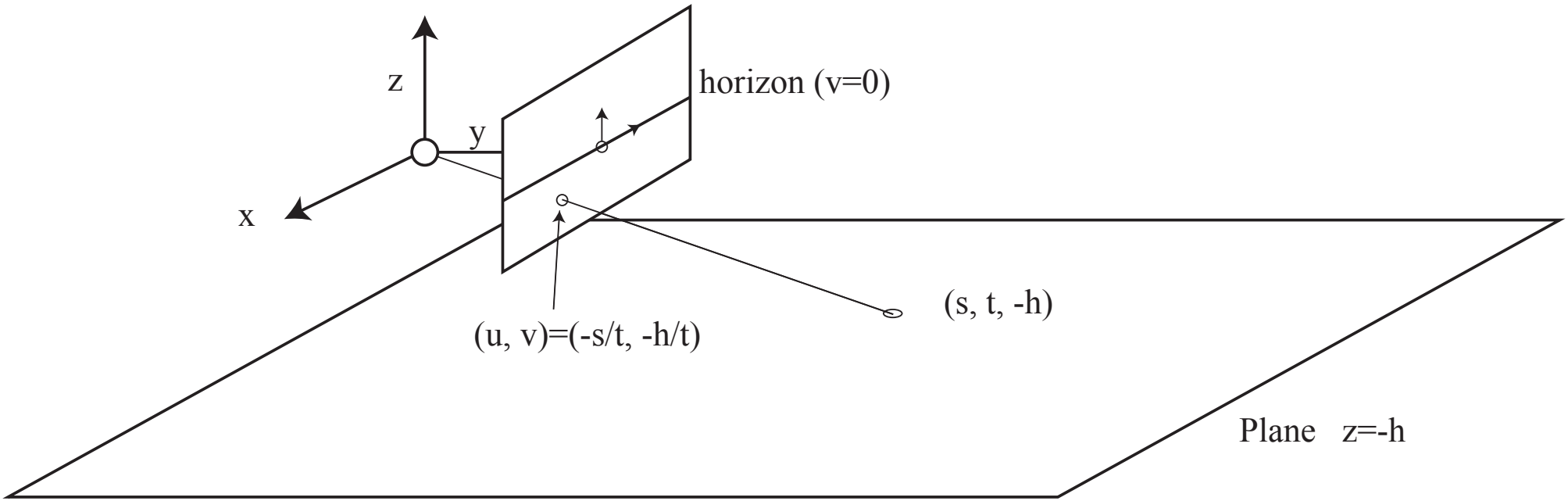
Figure 1. Sample results of our algorithm on examples from four different classes of CULane dataset [33] are shown here. Cyan lines are the detected lane boundaries, green region represents the ego lane and magenta line displays the estimated horizon. In the *No Line* class, there is actually no line markings on the road but the ground truth carries the lines shown.

Khan 20: Notice

- Current SOA on many datasets
  - for list of datasets, see Khan 20 - v. good on this
- Q:
  - what about less structured drivable regions?
  - can this be learned with less data? or none?
    - need data to learn keypoint finder
  - can rectification estimate be improved
    - better horizon finders out there - see Jacobs papers on website
- Q:
  - could Tabelini be improved by a horizon estimate?

Estimating the camera

- Height
  - from car (calibrated and known)
- Roll and pitch
  - from horizon
    - roll is why horizon isn't parallel to image plane
    - pitch is why it isn't centerline



Issues

- You have to do it fast
- You have to do it right
- Paint detection problems
- Geometric model problems

Missing paint, dirt

Curvy road

No paint


OFF
MEDIAN

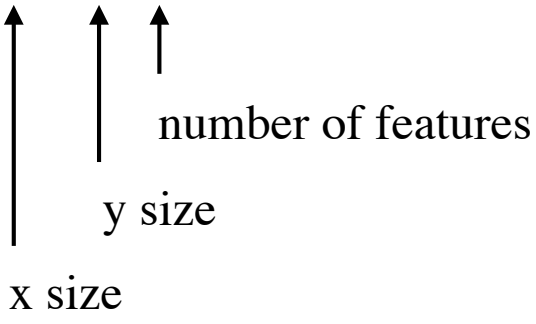

Robert Berdan ©

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

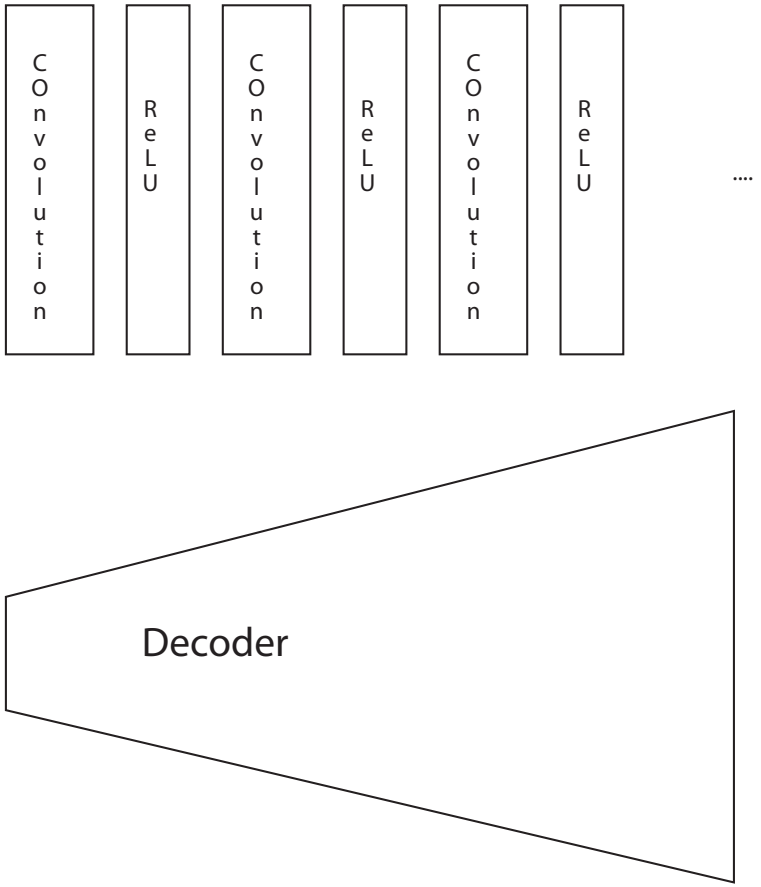
- Apply “pattern detector” to image
  - another to the result
  - another to the result
  - etc
  - occasionally reducing the spatial size of the block of data representing patterns to control redundancy
- The resulting block of data is spatially small
  - eg in the (very simple) CIFAR network,
  - 32x32x3-> 4x4x64



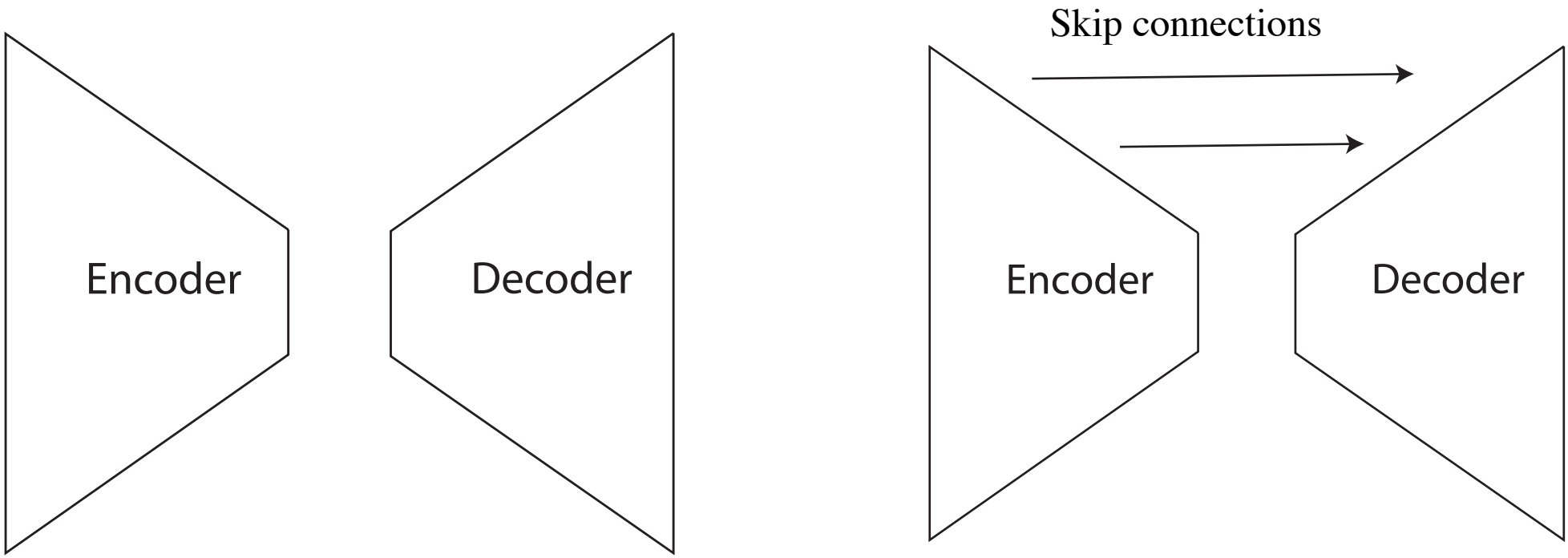
We could now predict an image by..

- 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


Convolution
ReLU
Convolution
ReLU
Convolution
ReLU
....
Decoder

Regression



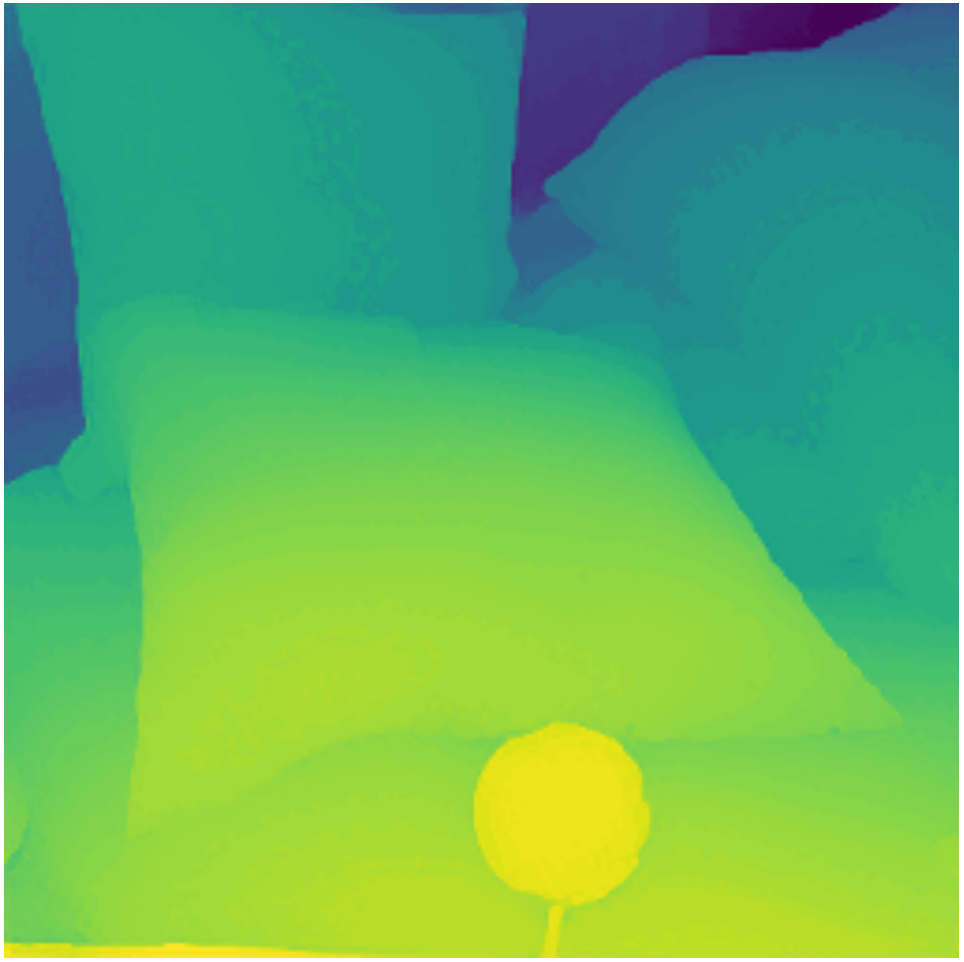
Sometimes known as a U-net

Regression

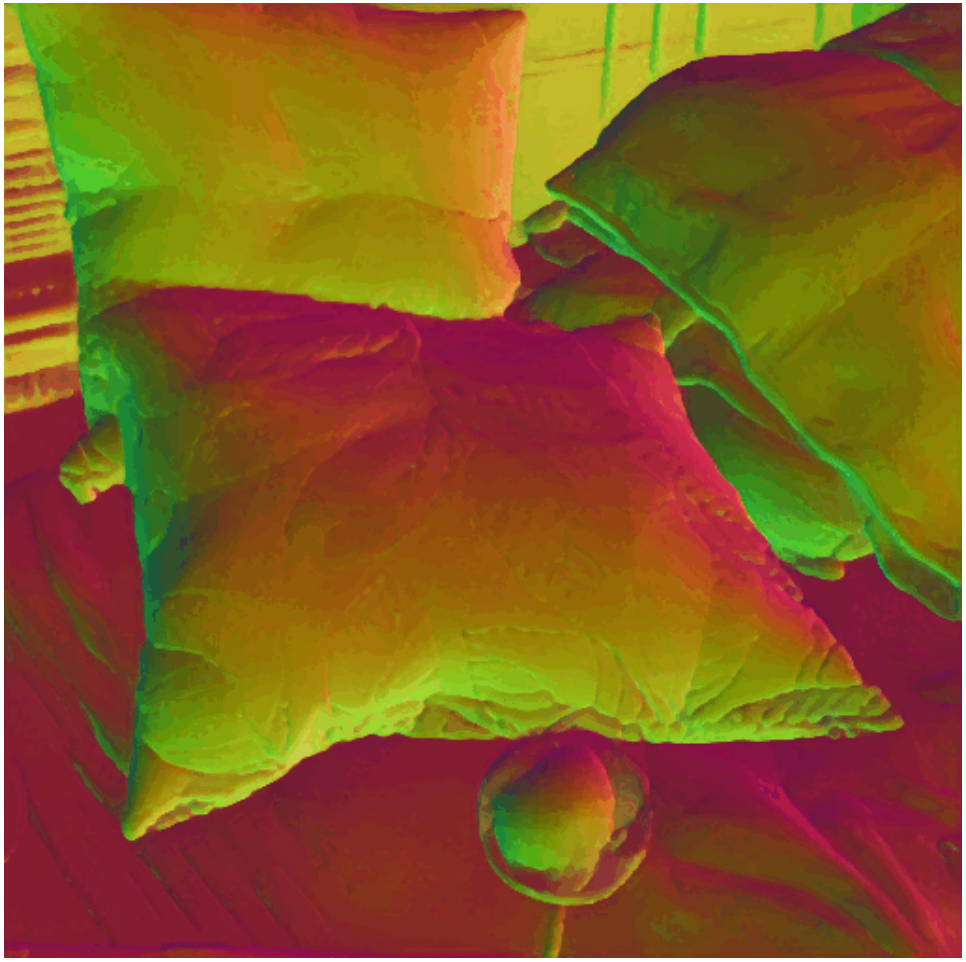
- Train with pairs (image, depth)
  - Loss
    - Squared error +abs value of error+other terms as required
- Very powerful general recipe
  - depth from image
  - normal from image
  - superresolution
  - etc.
- Variants
  - more sophisticated encoder

Examples

Depth (omnimap, current best depth est)



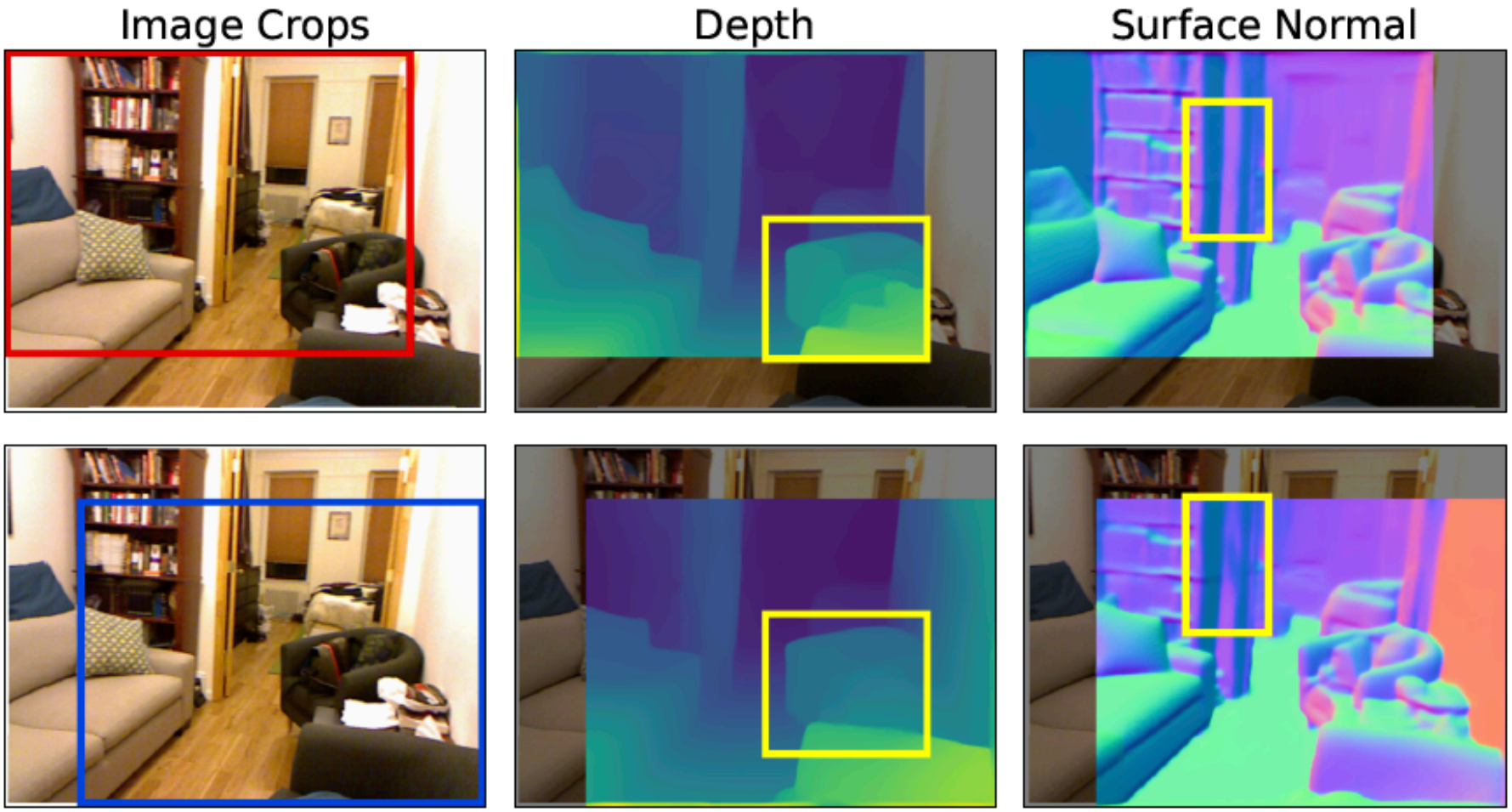
Normal (omnimap, current best normal est)



Open problems

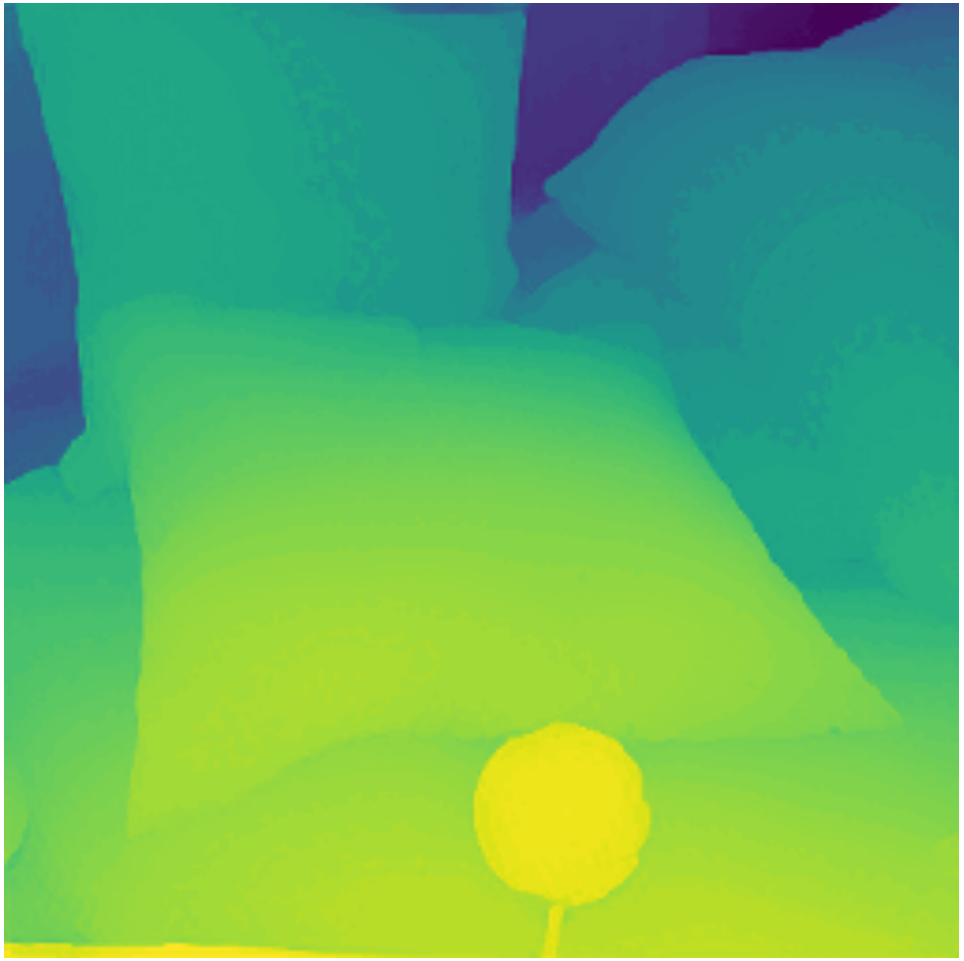
- Equivariance
  - two different crops of the same image should yield compatible results
- Lighting
  - change scene lighting; depth, normals shouldn't change
- Data
  - what if it is extremely hard to get data?
    - eg albedo

Equivariance problems



Lighting problems

Depth (omnimap, current best depth est)



Normal (omnimap, current best normal est)