Regression and Classification

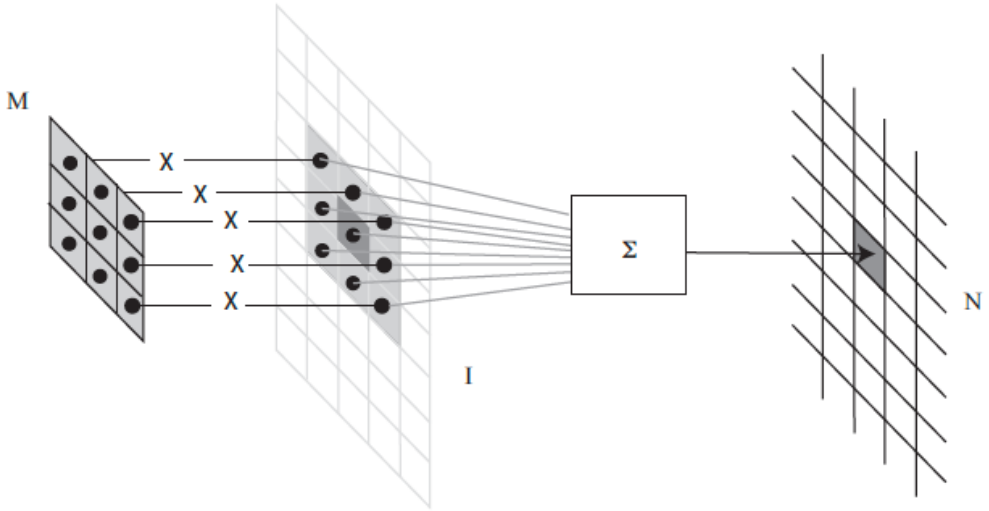
D.A. Forsyth, University of Illinois at Urbana Champaign

SOME APPLICATION SLIDES MISSING

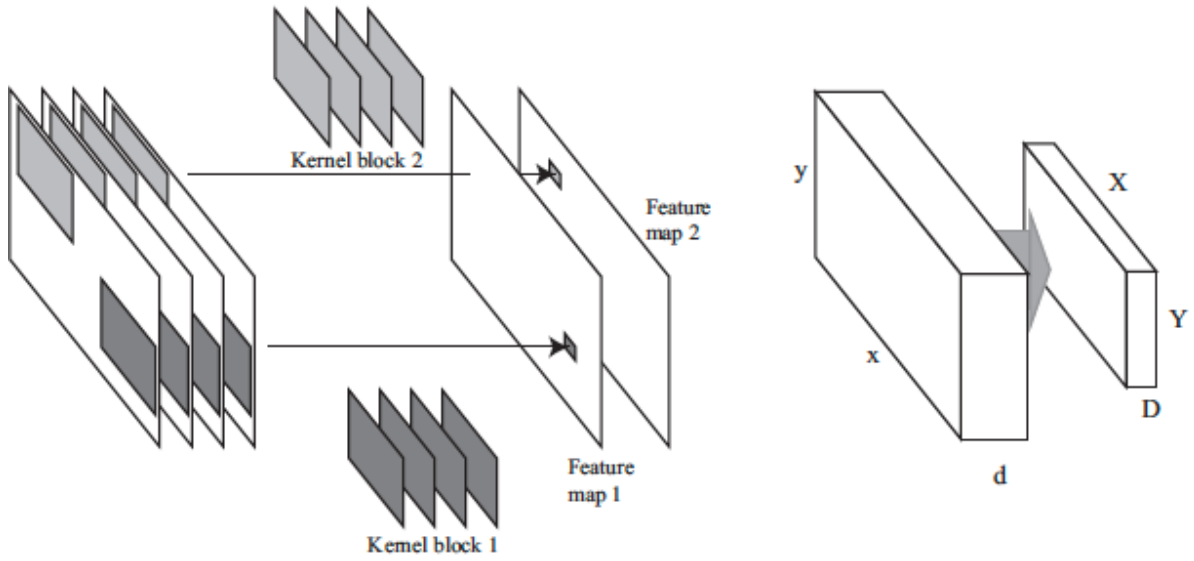
Main points

- You can learn comprehensive image encodings
- Regression:
  - You can learn to decode these into
    - denoised images
    - other good stuff (depth, normal, etc.)
- Classification:
  - You can decode into categories
    - per pixel
    - per image
- Detection
  - is basically lots of classification

Convolution

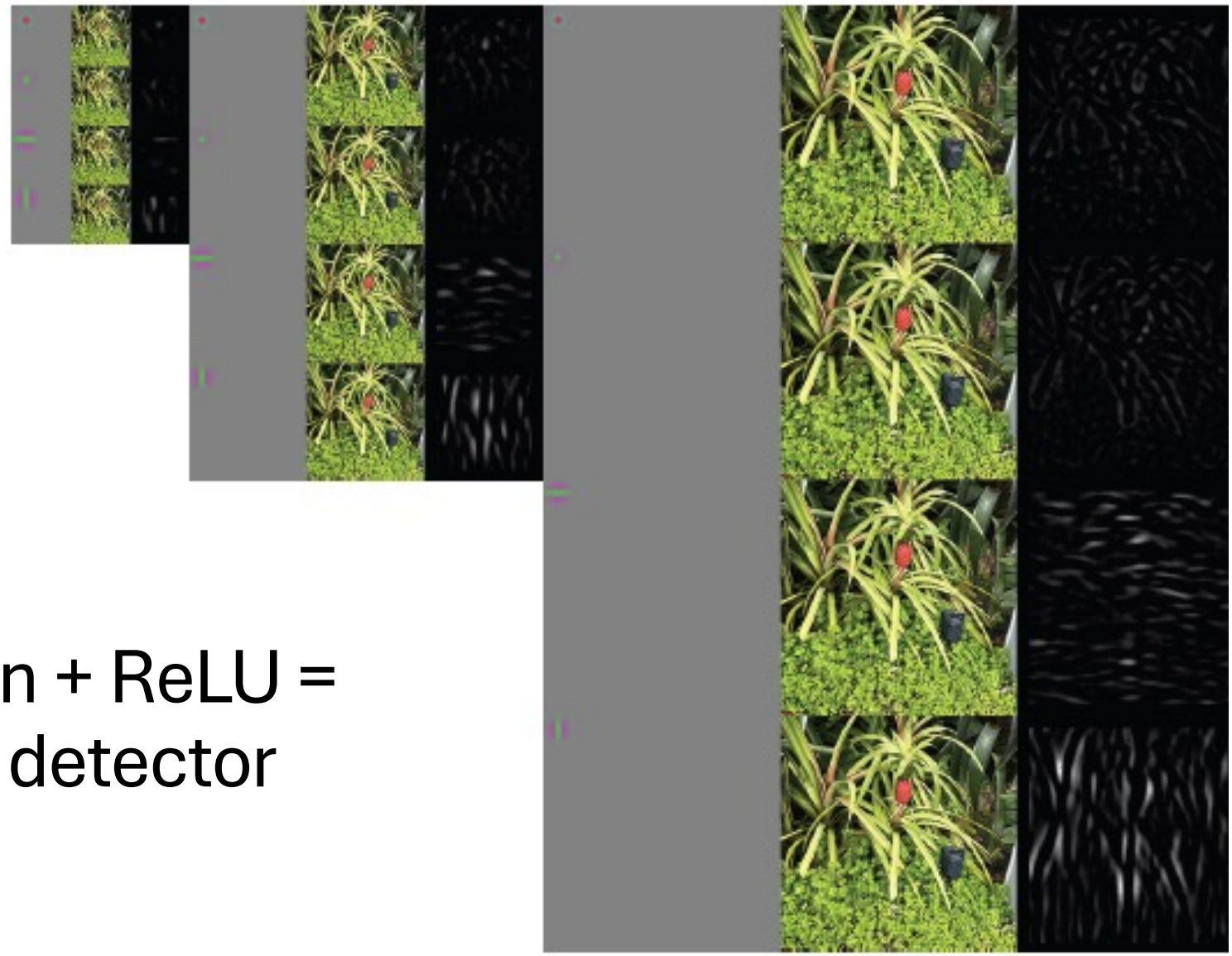


Multi-channel Convolution



ReLU

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



MC Convolution + ReLU =
Simple pattern detector

Image representations - Encoding

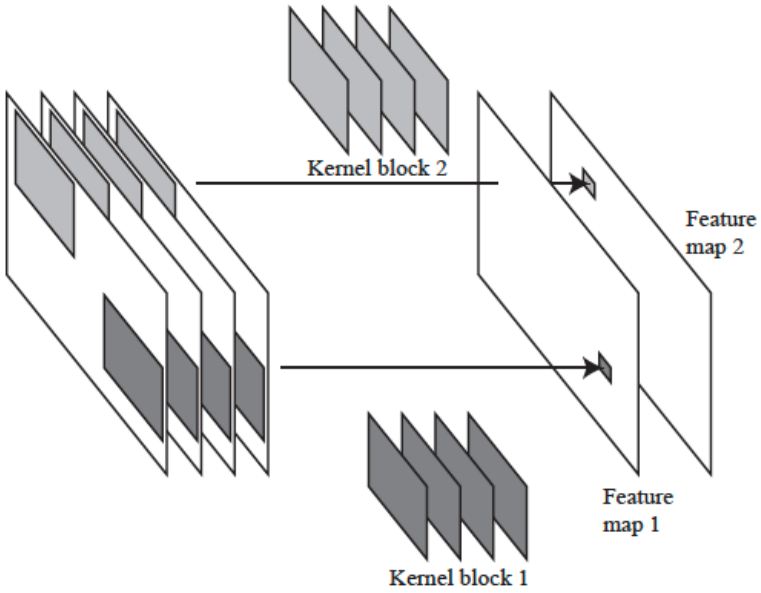
- Idea:
  - Filter banks + ReLU yield scores for many different patterns

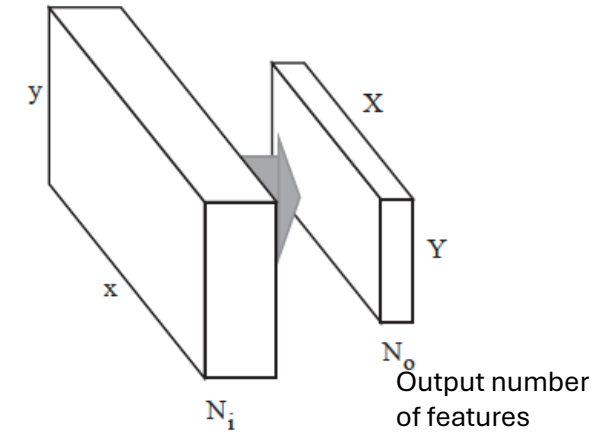
- Idea:
  - You can compose this, so patterns of patterns of …
  - Result: a code that describes the image

- Idea:
  - if filters are well chosen, you could use the representation to:
    - denoise images
    - find edges
    - find interest points
    - classify images…

BUT what filters/patterns should we use?

Convolutional Layers

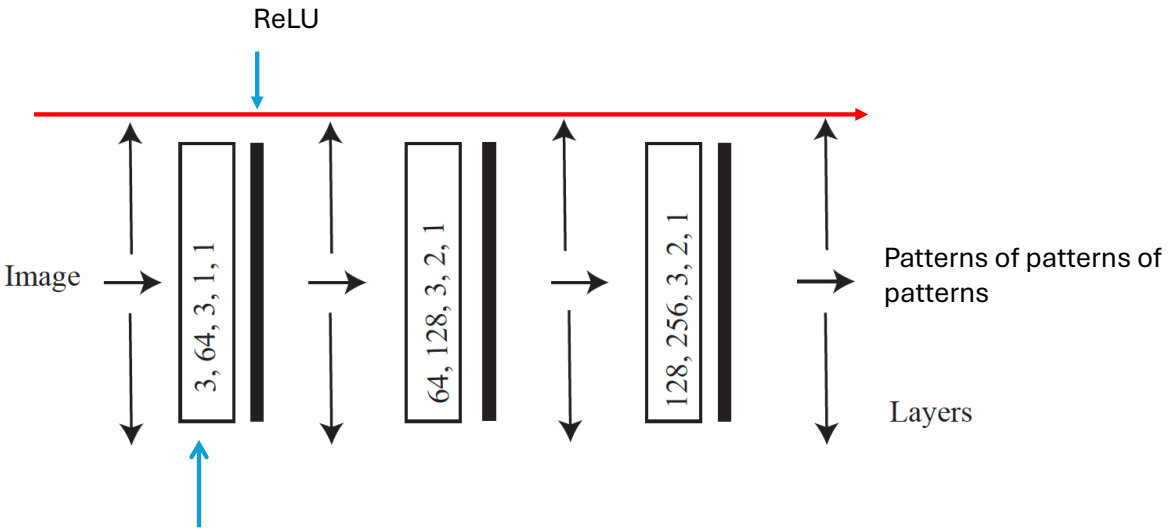




ReLU operates on tensor

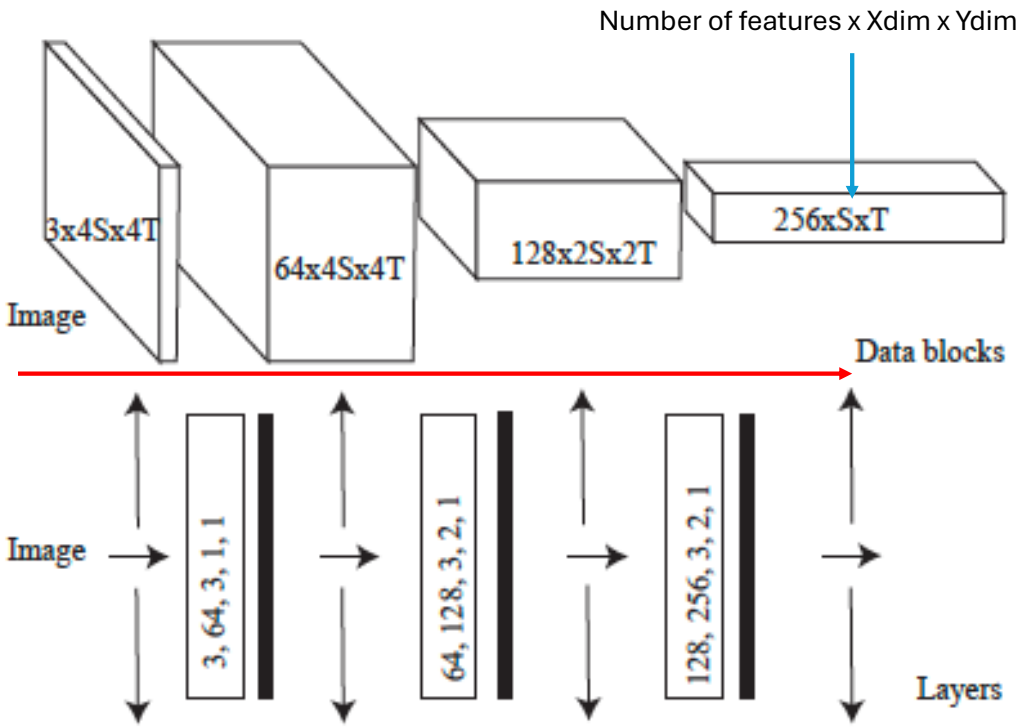
- Trivially – just ReLU at each location

A very simple encoder



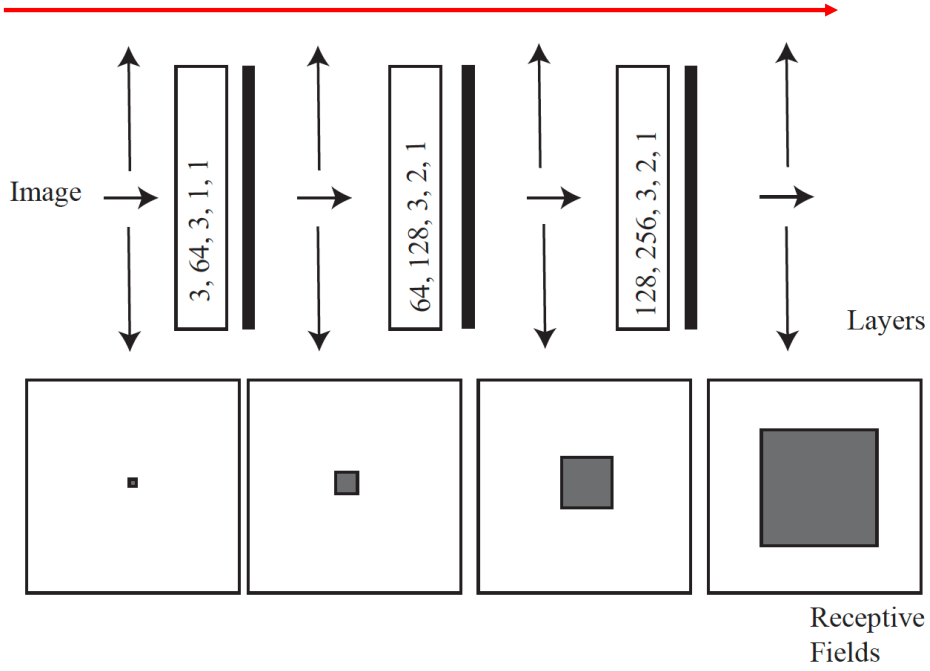
BUT what filters/patterns should we use?

As tensors


Number of features x Xdim x Ydim
3x4Sx4T
64x4Sx4T
128x2Sx2T
256xSxT
Image
Data blocks
Image
3, 64, 3, 1, 1
64, 128, 3, 2, 1
128, 256, 3, 2, 1
Layers

Receptive fields

- Support for a value in the feature map



Images from codes - Decoding

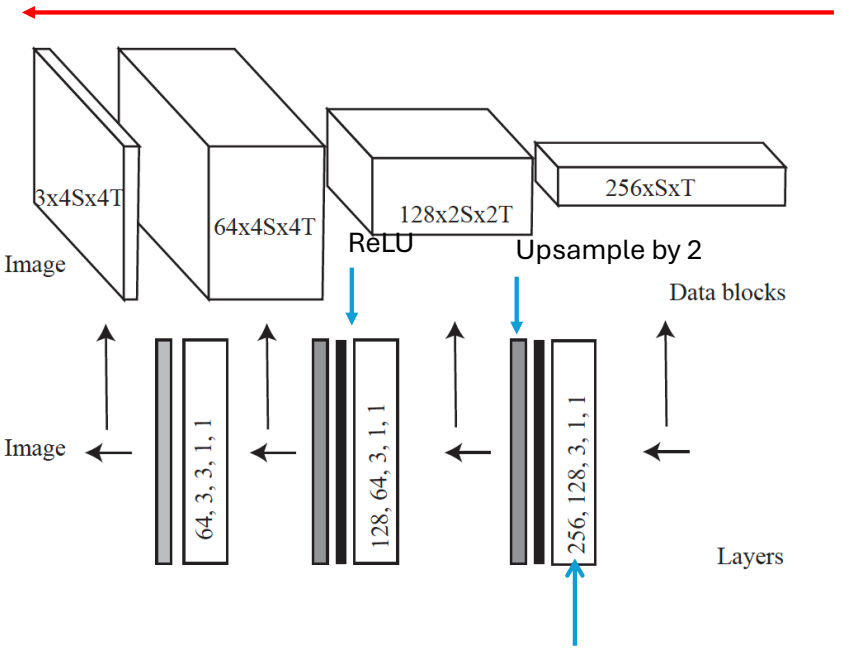
- Idea:
  - Code likely (roughly) invertible
    - Reconstruct filter responses from ReLU outputs
    - Reconstruct image from responses (conv. Theorem)
- Decode by:
  - Place down instances of detected patterns
  - This is filtering
  - Use a ReLU to prevent negative values accumulating

BUT what filters/patterns should we use?

Decoding

- Want:
  - map rep’n (patterns of patterns of patterns…) to image
- Have:
  - If rep’n is filter outputs, convolution is enough
  - Rep’n is spatially smaller than image
- Idea:
  - Filter+ReLU+upsample on occasion might do it

A decoder



Patterns of patterns of patterns

Input number of features, output number of features, kernel size, padding, stride

Big idea

- With the right choice of filters
  - a decoder could reconstruct an image from an encoders rep'n
- The rep'n is overcomplete
  - "sees" the image at many scales
  - so the pair should be able to denoise
- But what is the right choice of filters?

Choose filters/patterns that denoise

Denoising Autoencoder

- Now imagine input image is noisy:
  - The reconstruction might not be
    - (with some care and some luck)

- Idea:
  - Build device that can accept noisy image, produce clean
  - Denoising autoencoder

Helps choose filters/patterns – the ones that denoise

Learning the filters

- Procedure:
  - find many training pairs (noisy image, clean image)
  - adjust filters so that
  - Decode(Encode(noisy image)) is close to clean image
    - on average, over pairs
    - hope that this generalizes to new images
- Result:
  - Denoising autoencoder
  - Encoder has codes that represent images well
  - Opens the door to a lot of procedures

Adjusting the filters:

- Optimization problem, but weird

$$\mathcal{L}_{\mathcal{S}}(\theta) = \frac{1}{N} \sum_{i \in \mathcal{S}} \mathcal{C}(\mathcal{O}(\mathcal{I}_i, \theta), \mathcal{I}_i)$$

- Issues:
  - The cost function is very hard to evaluate (N is big)
  - There are lots of parameters (millions-billions)
    - so no newton's method
  - You don't actually want an optimum
    - you want a set of filters that works well on other images

Stochastic gradient descent

$$\mathcal{L}_{\mathcal{S}}(\theta) = \frac{1}{N} \sum_{i \in \mathcal{S}} \mathcal{C}(\mathcal{O}(\mathcal{I}_i, \theta), \mathcal{I}_i)$$

- Loss is a population mean
  - you can estimate this quite well with a sample mean
  - draw a small batch, average over that

Stochastic gradient descent

$$\theta_{n+1} = \theta_n - \eta_n \hat{\nabla}_\theta \mathcal{L}.$$

- How big a step?
  - Line search
    - you can't – N is too big
  - Fixed length
    - too big (doesn't settle down)
    - too small (no progress)
  - Learning rate schedule
    - start biggish, take steps, make smaller
    - how big is biggish? try

Rich range of improvements, variants, modifications available

Transformers: better encoders

- Architecture:
  - Accepts
    - one or more streams of tokens
      - very often fixed length
  - Produces
    - stream of tokens
    - OR single learned token
  - Origins in NLP
  - Apply to images by
    - tokenizing image
    - getting clever about decoding

Transformers: better encoders

- Inductive bias
  - Convolution, etc has inductive bias
    - mostly, pixels depend on nearby pixels
  - Transformers seem not to
    - or rather, it hasn't made a nuisance of itself
- Features:
  - Admit training in the absence of labels, outputs, etc
  - Every token communicates with every other
- Strong evidence of better encoders
- Issue:
  - vilely inefficient (this is why electricity/RAM/disk are now so expensive)
  - massive, expensive training demands
  - many/most vision transformers are fine-tuned from published weights

The key idea remains

- predict good stuff from images by encoding then decoding
  - but the plumbing is different…
    - different enough to affect the price of electricity/RAM/disk

Depth from single image

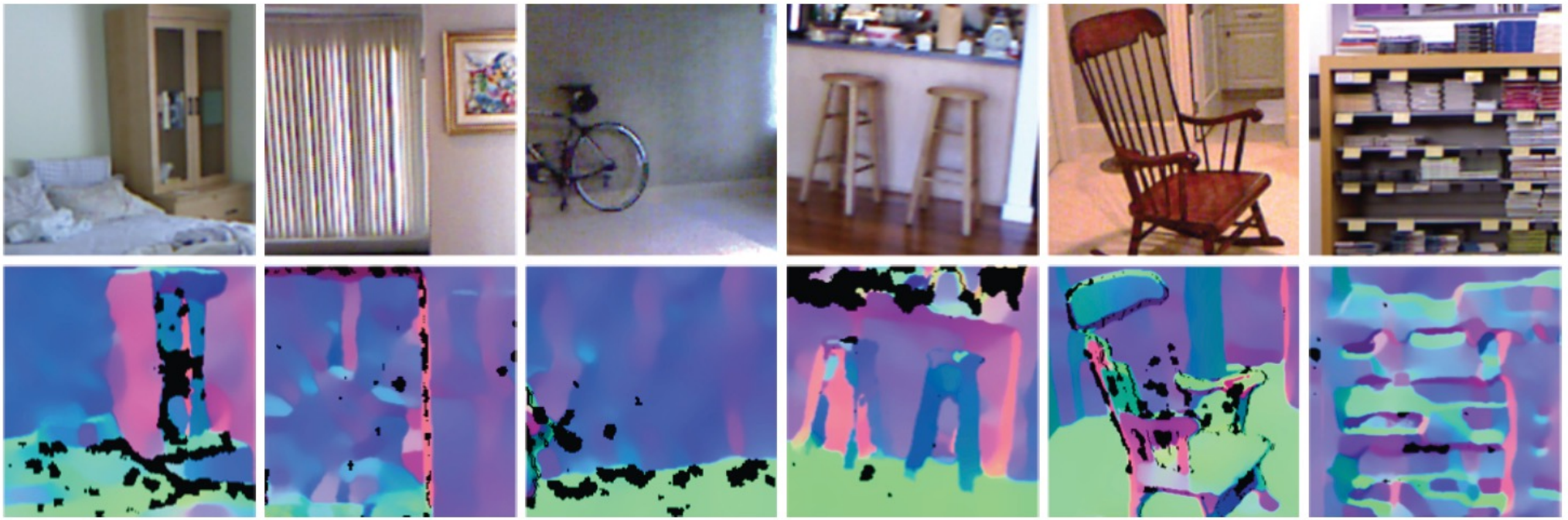
Data example from NYUV2

How and what?

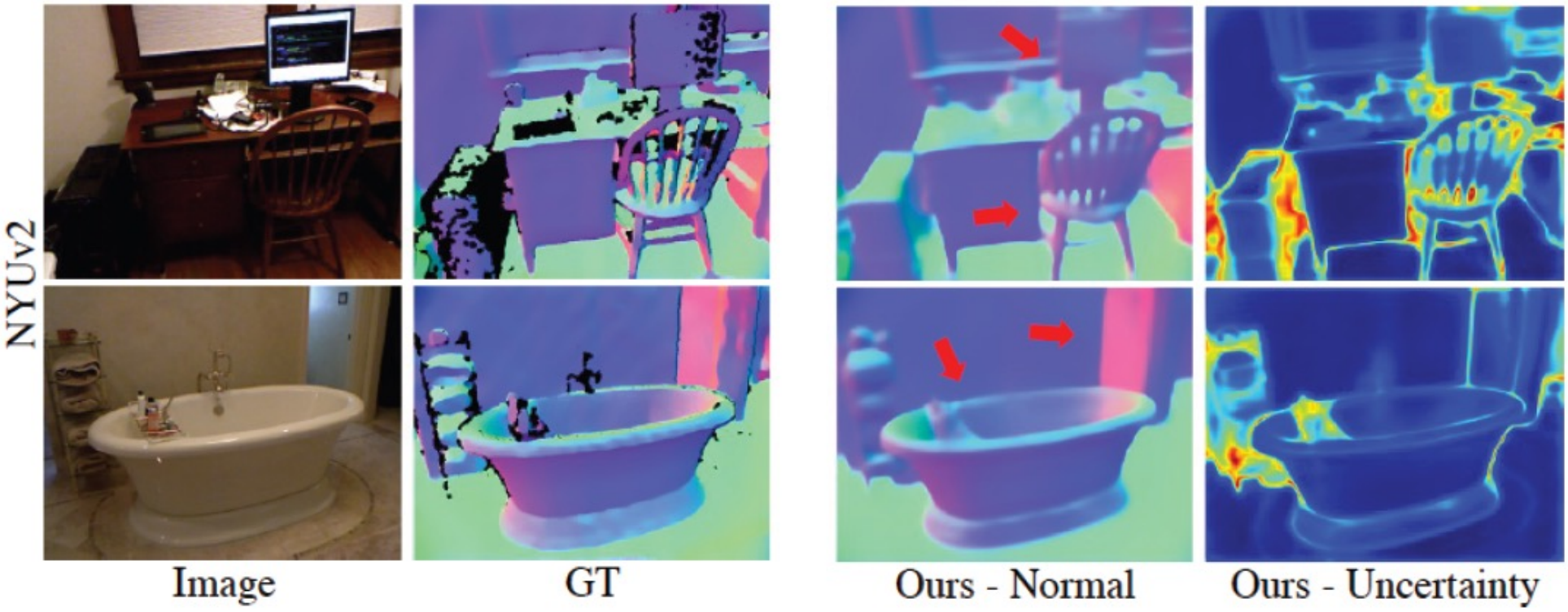
- How: easy
  - Get many pairs (image, depth), make decoder produce depth
- What:
  - Absolute depth:
    - Might be very difficult to get right
      - compare:
        - zoomed picture of a doll-house room
        - picture of real room
  - Relative depth:
    - depth up to per-image scale
    - more likely to get this right, but harder to use

Depth predictors work really well

Normal estimates from depth data are shaky



Normal predictors work really well too

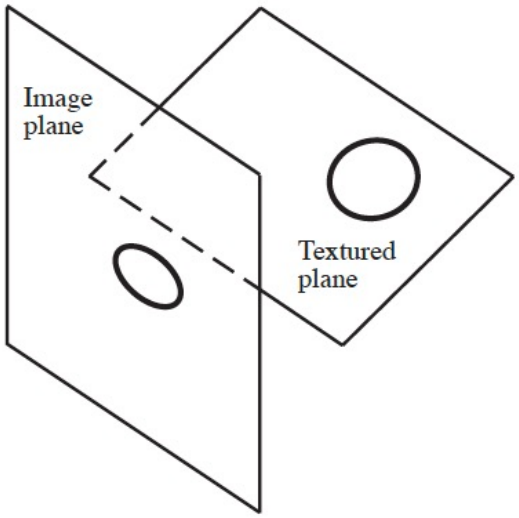


Why does all this work? I

- The depth and normal at a pixel is usually rather similar to the depth and normal at neighboring pixels. Further, large changes in depth and normal are usually signalled by edges.
- Depth maps aren't all that complicated. Scenes tend to have fairly stylized depth maps, so, for example, rooms are boxes with other boxes in them, and so on. Further, many depth regions are roughly either inclined planes, cylinders or spheres.
- Surface textures and shading offer cues to the orientation of a surface and so to the normal. Shading cues are discussed in Section 15.10. Texture cues are

Why does all this work? II

- Texture deformations are a cue to normals
  - (archaic term: shape from texture)



Further image to image mapping

- Generally, it's worth trying anything with
  - image in – image like thing out
- Cases here:
  - Denoising (did this); superresolution; defogging
- Other possibilities:
  - Fix underwater pix; derain; fix color balance; change lighting (?)
- Recipe needs changing for…
  - Edges; semantic segmentation (coming up)

A classifier

- Any procedure that
  - accepts a set of features
  - produces a class
- Hugely useful and general idea
  - Credit card transaction (good/fraud)
  - Should I download this code (secure/insecure)
- Number of classes could be big:
  - Doctor (well, cold, flu, … measles, … etc.)

Summaries of performance

- Error rate:
  - fraction of classification attempts with wrong answer
- Accuracy:
  - fraction of classification attempts with right answer

- Very best error rate
  - Bayes error rate
  - Usually, larger than zero
    - example: alien tries to determine sex from height
  - Usually, very hard to know accurately

Simple, two-class linear classifier

- Assume each data item is (feature vector, label)
  - fv is x

Remember this: *A two-class linear classifier maps a feature vector* $\mathbf{x}$ *to one of two classes, using*

$$sign(\mathbf{a}^T\mathbf{x} + b)$$

where $\mathbf{a}$ *and* b *are parameters of the classifier which are chosen to get strong performance on training data. Linear classifiers are much more powerful than you might expect at first glance. The decision boundary of a linear classifier is a hyperplane in feature space.*

Practical issue

- Take the block of features from the encoder
  - turn them into a vector of fixed size
  - (this will mean classifier accepts images of fixed size)
- Typically:
  - ensure block shrinks spatially as it moves along encoder
    - by stride or pooling
  - when it is small enough, straighten into vector
- Pooling:
  - make a data block smaller by max/average within fixed size windows (usually 2x2)

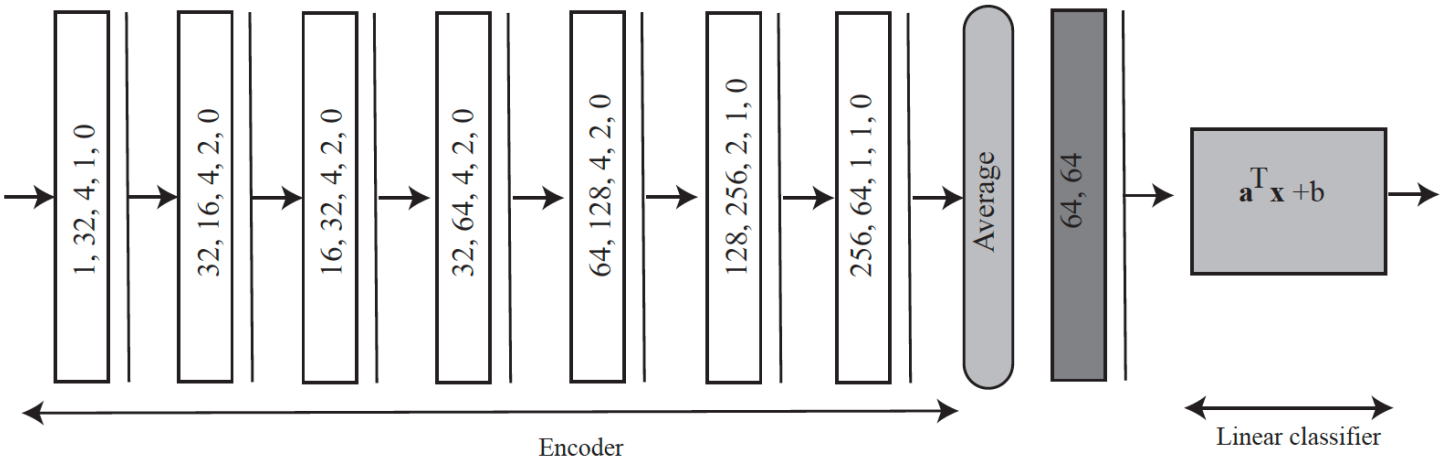
Fully connected layers

- Vector describes the whole image

You could regard the $g \times 1 \times 1$ block as a vector (in some APIs, you need to reshape, but this is housekeeping) and simply pass it to a linear classifier. Alternatively, you could transform this vector with a *fully connected layer*, which maps a vector $\mathbf{u}$ to a vector $\mathcal{C}\mathbf{u} + \mathbf{d}$, where the parameters $\mathcal{C}$ and $\mathbf{d}$ are learned, and $\mathcal{C}$ does not need to be square.

Notice that applying a linear classifier $\mathbf{a}^T\mathbf{x} + b$ to the output of a fully connected layer is not particularly interesting, because the result is $\left(\mathcal{C}^T\mathbf{a}\right)^T\mathbf{u} + \left(\mathbf{a}^T\mathbf{d} + b\right)$, which is just a different linear classifier. Similarly, applying a fully connected layer to another fully connected layer directly is not interesting. Instead, each fully connected layer is followed by a ReLU.

A simple image classifier



Simple Image Classification

- Classify an image into one of two classes by
  - Stack some pooling, FC layers on an encoder
  - Adjust result into a vector
  - Pass to linear classifier
- Learn all parameters with SGD and various losses
    - to get training examples right

Multiclass classification

Imagine you wish to classify something into one of k classes. You have a set of examples where the classes are known for each example. You can encode this information by associating a *one hot vector* with each training example. This vector has k components, one per class. The component corresponding to the class is one, and all others are zero. Write $\mathbf{y}_i$ for that vector for the i'th training example.

Your classifier must now produce a k dimensional vector for an example. This is quite commonly called a *score* Write $\mathbf{x}$ for the example and $\mathbf{s}$ for the vector of scores produced by the classifier. Then a linear classifier predicts

$$\mathbf{s} = \mathcal{A}\mathbf{x} + \mathbf{b}.$$

Stack the entries of $\mathcal{A}$ and $\mathbf{b}$ into a vector θ of all the parameters of the classifier. Notice there is a small disparity between a two class classifier as described in Section 21.2 and this description of a multiclass classifier. This minor nuisance is explored in the exercises **exercises** .

Interpret the score as a probability using

$$P(\text{item is class } u|\mathbf{x}, \theta) = \frac{\exp[s_u]}{\sum_w \exp[s_u]}.$$

The loss could be the negative log-likelihood of the data $\mathbf{y}$ under the model. Write $\mathbf{q}$ for the vector whose u'th component is

$$q_u = \frac{\exp[s_u]}{\sum_w \exp[s_w]}.$$

Then the negative log-likelihood for the example is

$$C(\theta; \mathbf{x}, \mathbf{y}) = -\log\left[\mathbf{y}^T \mathbf{q}\right]$$

and averaging over the whole dataset yields

$$\mathcal{L}_{lr} = \frac{1}{N} \sum_{i \in \text{examples}} C(\theta; \mathbf{x}_i, \mathbf{y}_i).$$

Ideas

- Simple Image Classification
  - Classify an image into one of two classes by
    - Stack some pooling, FC layers on an encoder
    - Adjust result into a vector
    - Pass to linear classifier
  - Learn all parameters with SGD and various losses
    - to get training examples right
- Simple image regression:
  - Build U-Net
    - input is image
    - output is image-like thing (depth,normal, etc)
    - train to produce the right answer
- Merge these to produce class scores at each pixel
  - rather than a depth…

Semantic segmentation

- Label pixels in images with labels taken from a vocabulary

- Notice there are two kinds of label
  - “things” == count nouns == what you can count
    - car, pedestrian, bike, dog, …
  - “stuff” == mass nouns == others
    - grass, sky, water, cloud, …

Semantic segmentation on Kitti (labelled data)

MS-CoCo dataset

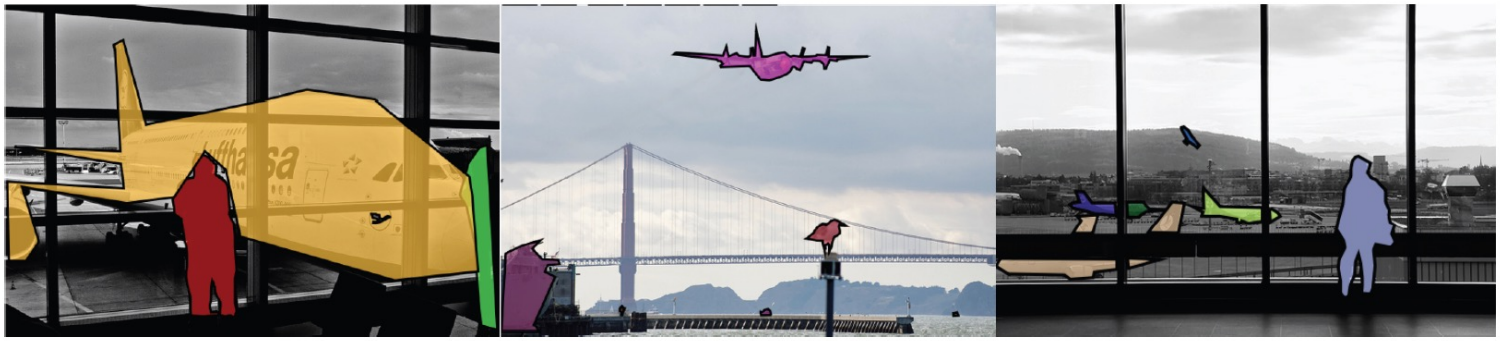


FIGURE 22.8: *The three images in the MS-CoCo dataset that contain an aeroplane, a bird and a person (the bird in the image on the* **left** *is the airline's logo). Note how objects are delineated with polygons and the relatively rich context in which the objects occur.*

Image classification

- Label image by:
  - Insert image into encoder
  - apply multiclass classifier
- Variants:
  - complexity and training process for encoder
  - how encoder is trained
    - using labels from your dataset is one of many options
  - what kinds of label are predicted
- What are the labels?

Recipes for cases

- Cheap and cheerful
  - obtain small encoder
    - likely, small ResNet, possibly pre-trained on ImageNet
  - Train this on your labels
    - possibly fine-tuning encoder
      - small learning rate for encoder, bigger for linear classifier
- Luxury high performance
  - obtain large encoder
    - very likely transformer, heavily pretrained
  - fine-tune on your labels
- Large specialized
  - eg fine-grained recognition
  - architectures and processes quite varied

Types of label

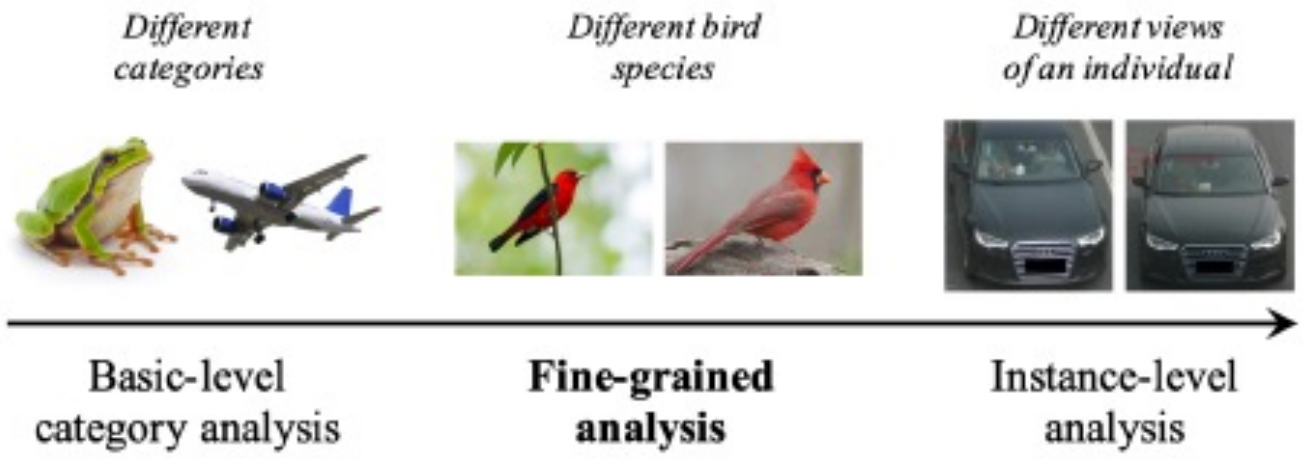
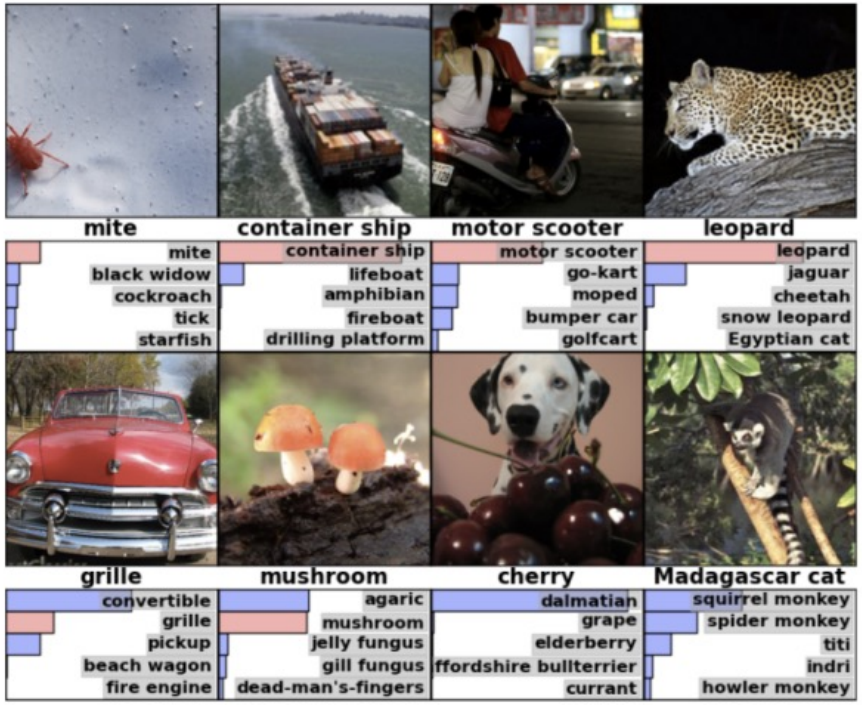


Image credit: Figure 3 of *Fine-Grained Image Analysis with Deep Learning: A Survey*

Types of label: category

- predict the category of the “main object”

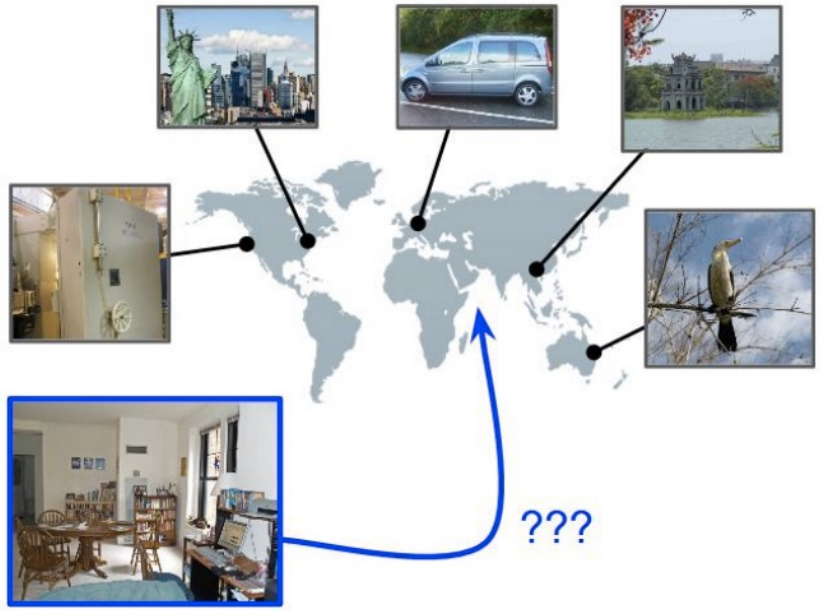


Other types of label…

Date prediction

Vittayakorn et al. (2017)

Location prediction



Vo et al. (2017)

Fine grained category recognition

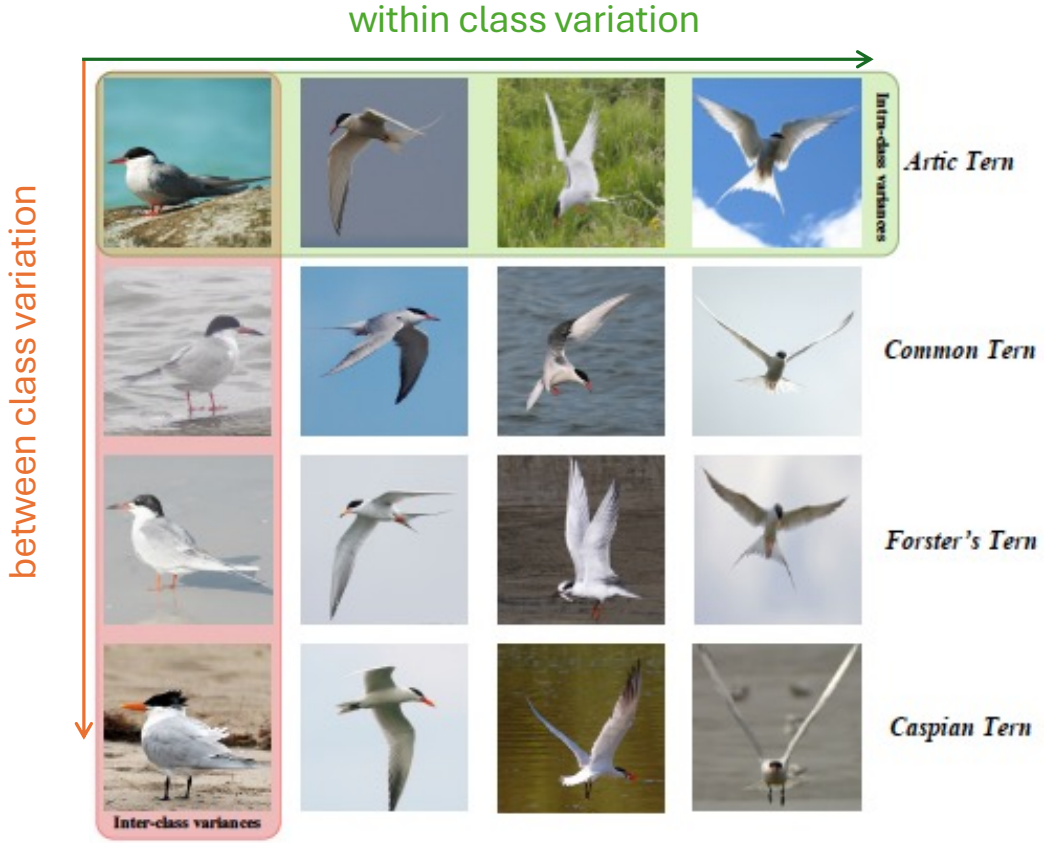


Image credit: Figure 4 of *Fine-Grained Image Analysis with Deep Learning: A Survey*

SOTA - rough summary

- Very high accuracy with 1000's of classes
  - Using
    - very deep residual networks
    - alternative feature construction methods
- Challenges
  - tough with little training data
    - use a pretrained encoder and a classifier
    - link to language (later)
  - sufficient change in dataset presents problems
- FGVC works really well
  - see Merlin BirdID, iNaturalist, etc.

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

Basic architecture

Need..

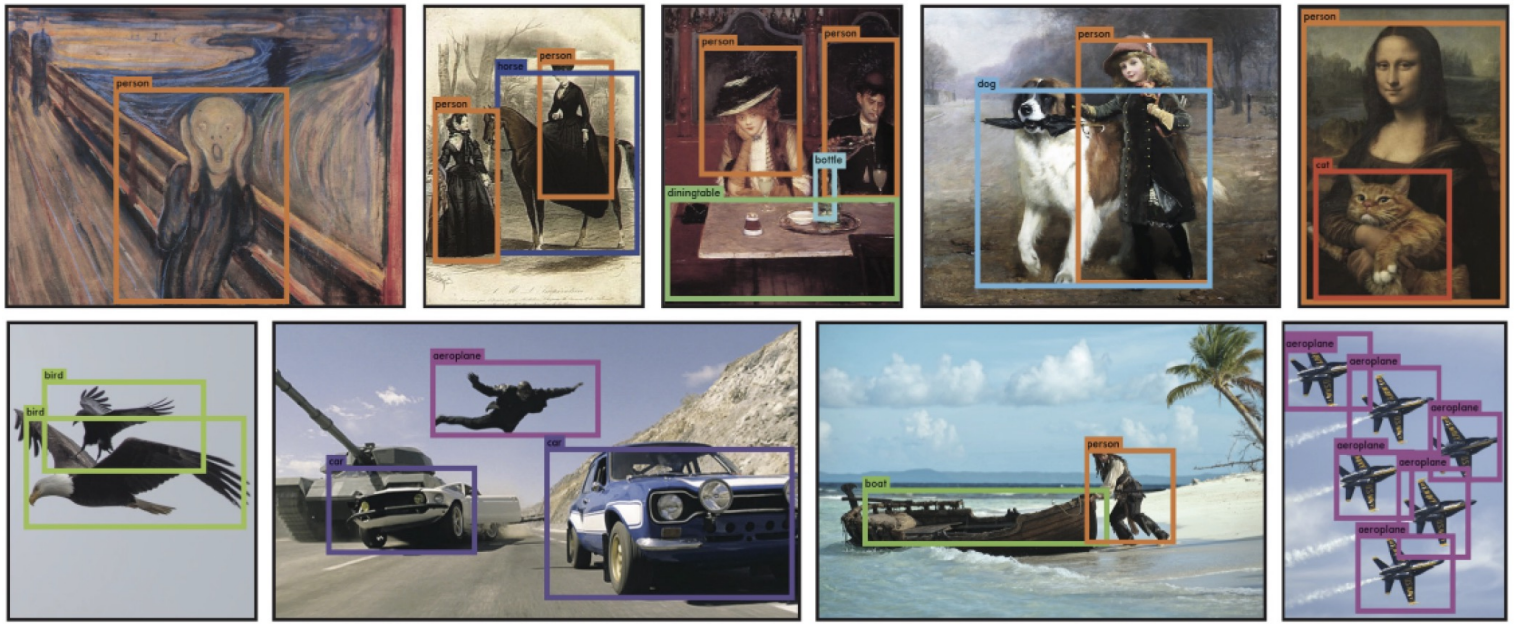
- to control the number of boxes checked
- to manage the classifier
  - may see lots of negatives
  - must be willing to respond when box isn’t precise
- to manage high-scoring boxes
  - non-maximum suppression
- to refine boxes

Three helpful points

- Correlated scores
  - boxes overlap, so box scores are correlated
- Extrapolating box scores
  - manageable, because they are correlated
  - bounding box regression
- Worthwhile windows
  - surprisingly, you can tell whether a box contains an object, even without specifying the object
  - means you can reduce the number of boxes you show to classifier

Two threads

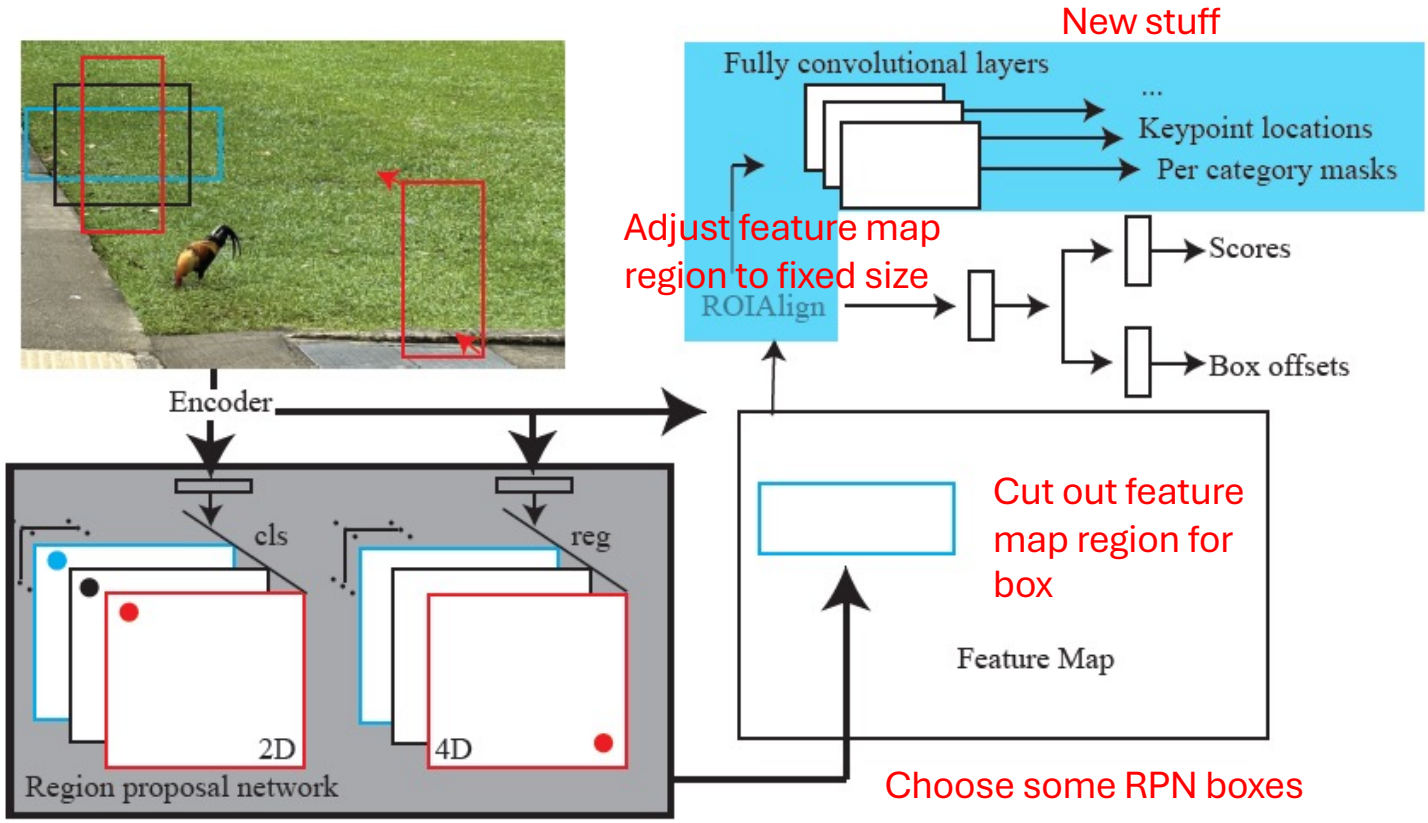
- YOLO: Localize while classifying
  - in parallel, score
    - boxes for “goodness of box”
    - boxes for “what is in it”
  - combine
- Localize then classify
  - find boxes that likely contain objects
  - decide what is in the box



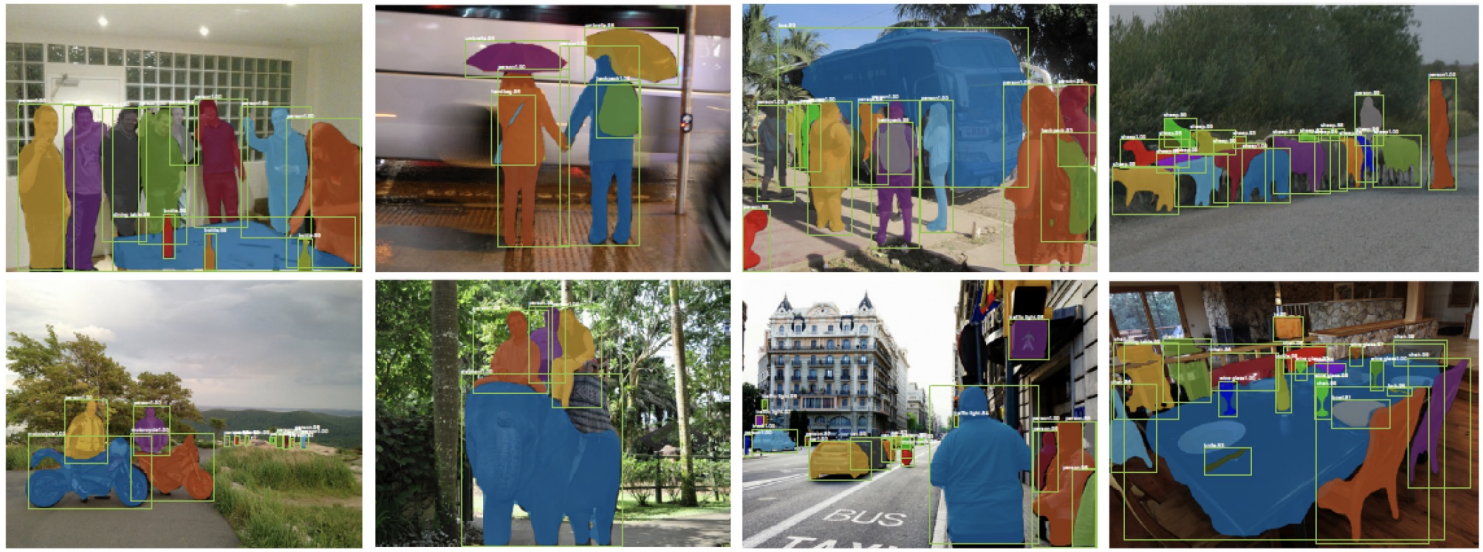
- Original caption: "*YOLO running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane*"

Credit: Figure 5 of You Only Look Once: Unified, Real-Time Object Detection
Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi

MaskRCNN


New stuff
Fully convolutional layers
...
Keypoint locations
Per category masks
Adjust feature map region to fixed size
ROIAlign
Scores
Box offsets
Encoder
cls
reg
2D
4D
Region proposal network
Cut out feature map region for box
Feature Map
Choose some RPN boxes

Detection with masks



- Evaluation:
  - you can compute IoU for masks, too…

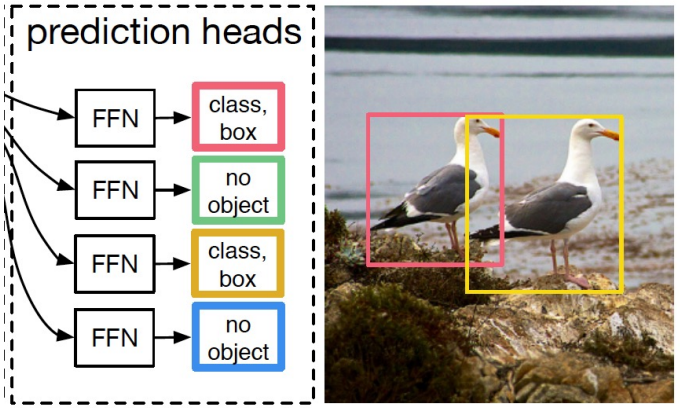
Variants

- Predict more stuff from ROI
  - eg body keypoints

- Bolt onto a semantic segmenter
  - to get panoptic segmentation

DETR – detection by transformer

- prediction head
  - accepts output token
  - produces
    - no object OR
    - class scores, box location



- Training (roughly!)
  - Find large number of images with boxes, classes
  - Loss is min over matchings
    - from predicted tokens to image boxes
      - box loss+class loss
  - Use weighted bipartite graph matching to match
  - Descent on this

End-to-End Object Detection with Transformers Carion et al, 2020

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