Deep Learning for Computer Vision

Fall 2022

https://cool.ntu.edu.tw/courses/189345 (NTU COOL)

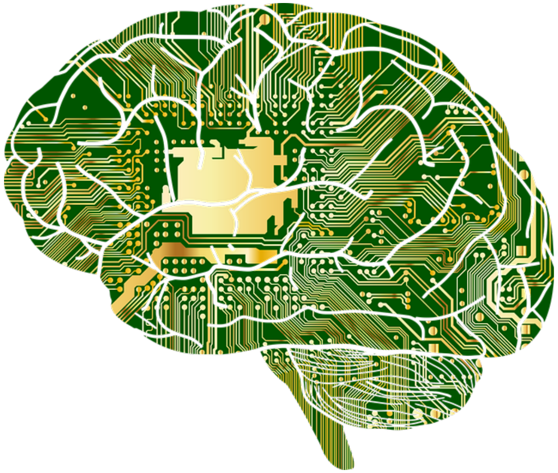
http://vllab.ee.ntu.edu.tw/dlcv.html (Public website)

Yu-Chiang Frank Wang 王鈺強, Professor

Dept. Electrical Engineering, National Taiwan University

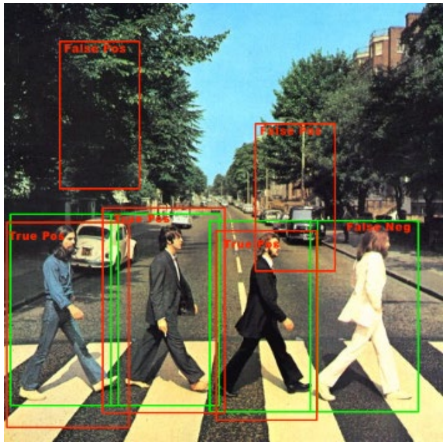
2022/09/27

What’s to Be Covered Today…



- Segmentation
- Object Detection
- Generative Model

- HW #1 is out & due Oct. 10^{th} Mon 23:59



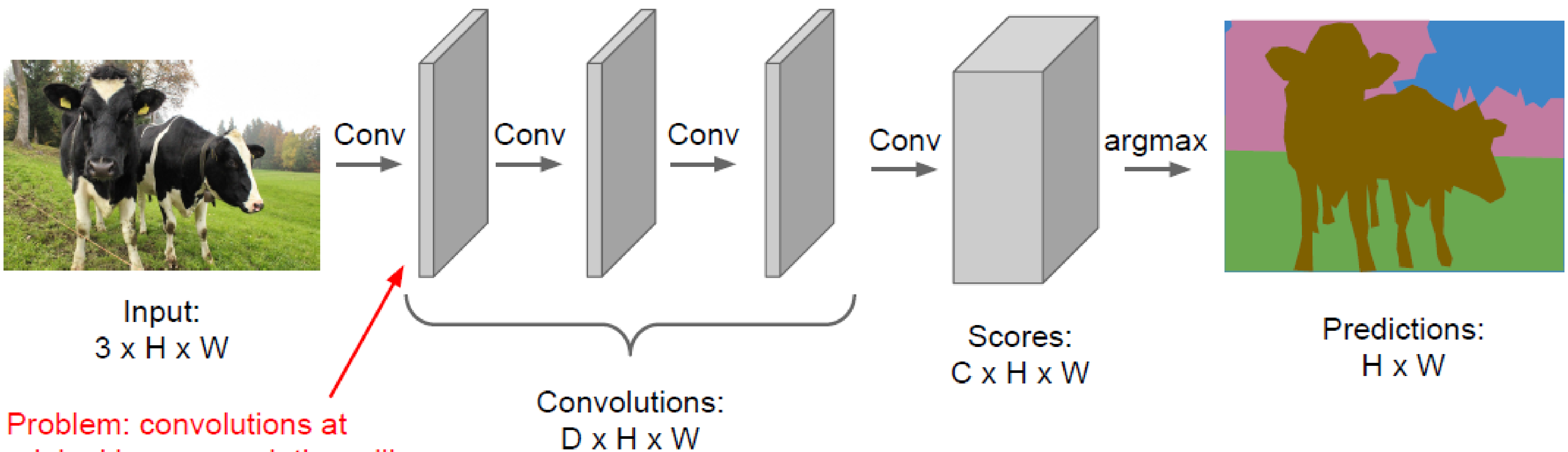
A Practical Segmentation Task

- Semantic Segmentation
  - Supervised learning
  - Assign a class label to each pixel in the input image (i.e., pixel-level classification)
  - Not like instance segmentation, do not differentiate instances;
    only care about pixel labels

Semantic Segmentation

- Fully Convolutional Nets

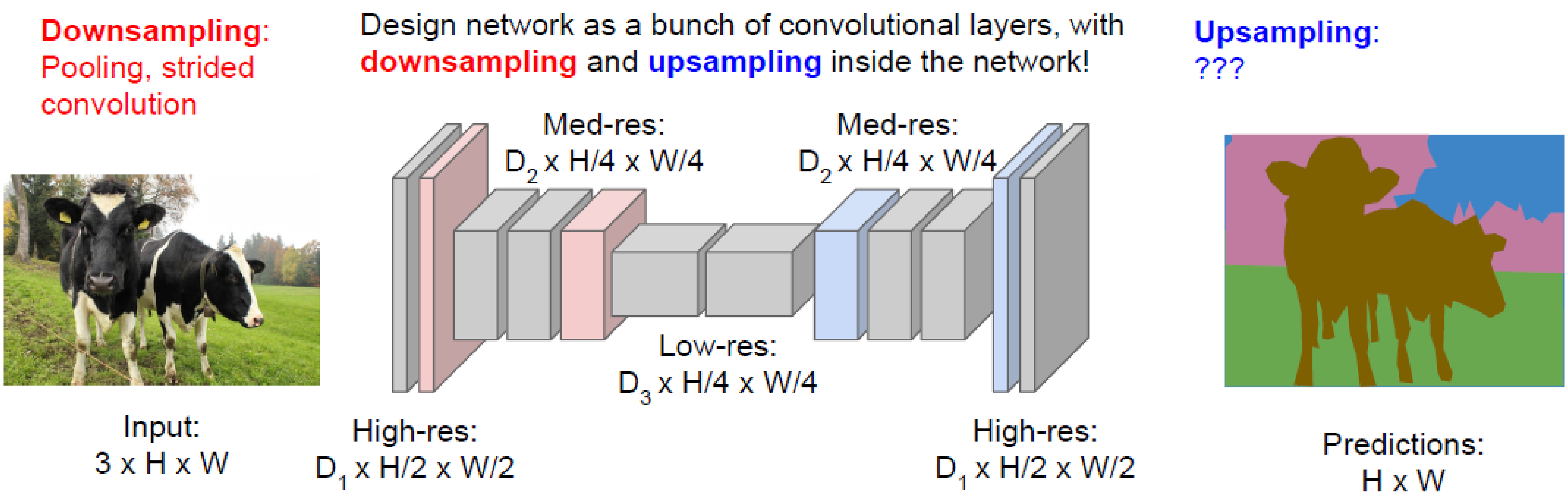
Design a network as a bunch of convolutional layers
to make predictions for pixels all at once!



Problem: convolutions at original image resolution will be very expensive ...

Semantic Segmentation

- Fully Convolutional Nets (cont'd)

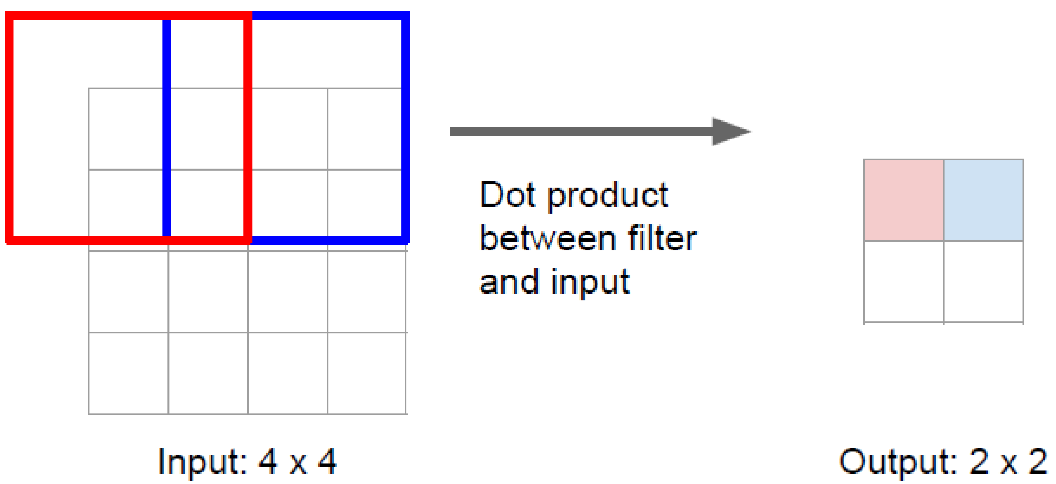


Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

In-Network Downsampling

- Revisit: Learnable Downsampling: Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1



Filter moves 2 pixels in the input for every one pixel in the output

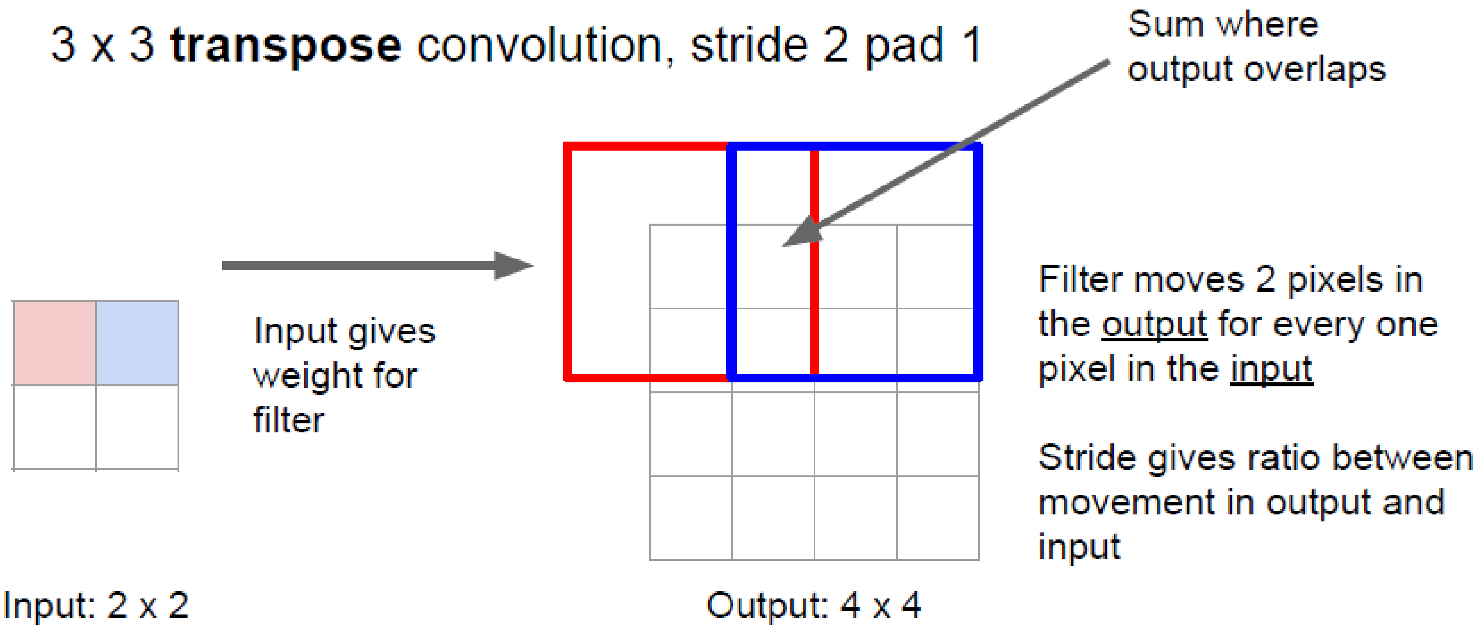
Stride gives ratio between movement in input and output

In-Network Upsampling

- Transpose Convolution

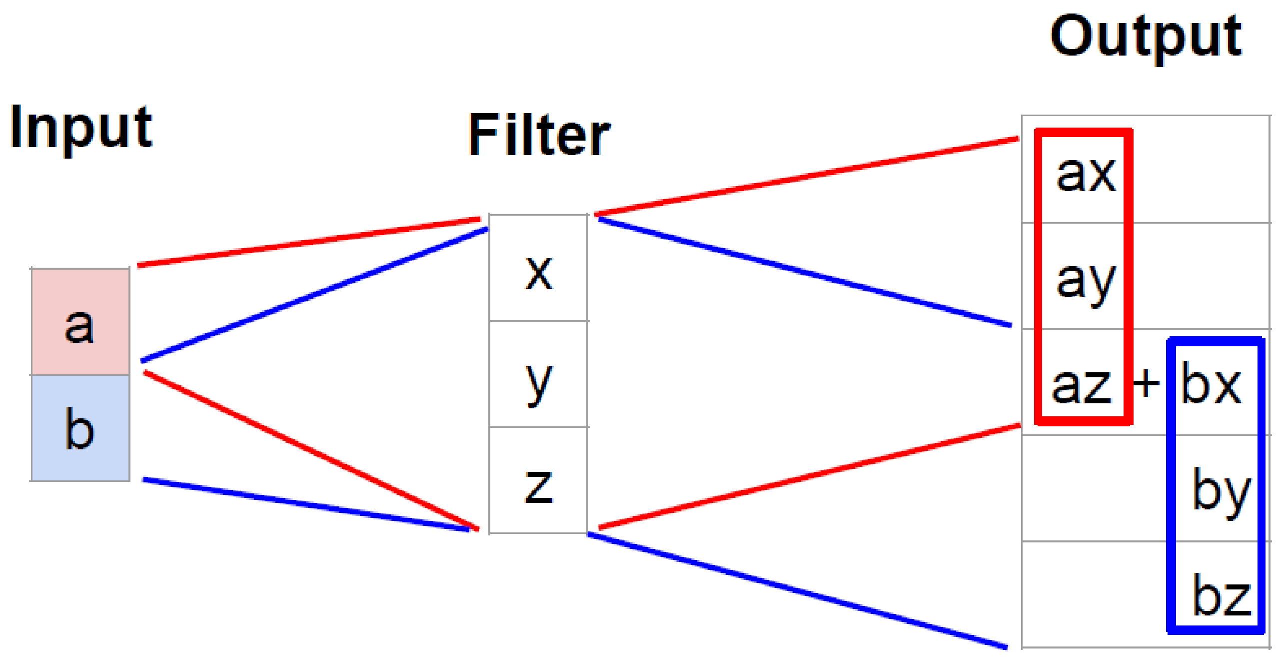
Other names:
-Deconvolution (bad)
-Upconvolution
-Fractionally strided convolution
-Backward strided convolution

3 x 3 **transpose** convolution, stride 2 pad 1



In-Network Upsampling

- Transpose Convolution
  - 1D example



Output contains copies of the filter weighted by the input, summing at where at overlaps in the output

Need to crop one pixel from output to make output exactly 2x input

In-Network Upsampling

- Transpose Convolution
  - Example as matrix multiplication

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & x & 0 & 0 & 0 \\ 0 & x & y & x & 0 & 0 \\ 0 & 0 & x & y & x & 0 \\ 0 & 0 & 0 & x & y & x \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ ax + by + cz \\ bx + cy + dz \\ cx + dy \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=1, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

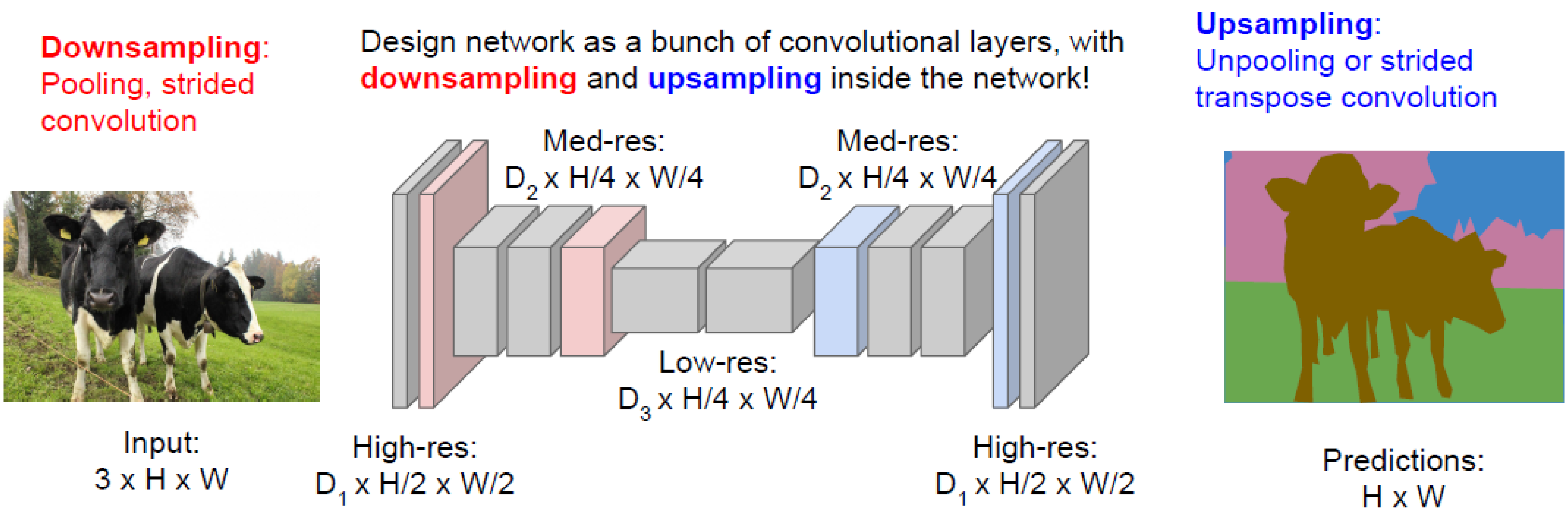
$$\vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & 0 & 0 & 0 \\ y & x & 0 & 0 \\ z & y & x & 0 \\ 0 & z & y & x \\ 0 & 0 & z & y \\ 0 & 0 & 0 & z \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} ax \\ ay + bx \\ az + by + cx \\ bz + cy + dx \\ cz + dy \\ dz \end{bmatrix}$$

When stride=1, convolution transpose is just a regular convolution (with different padding rules)

Fully Convolutional Networks (FCN)

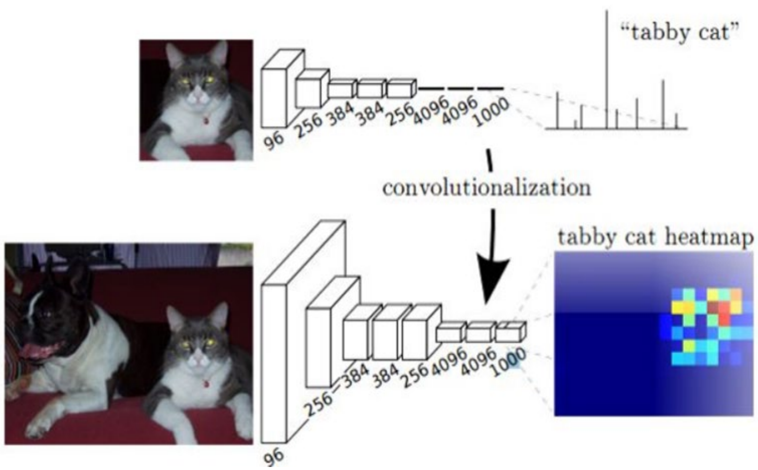
- Remarks
  - All layers are convolutional
  - End-to-end training

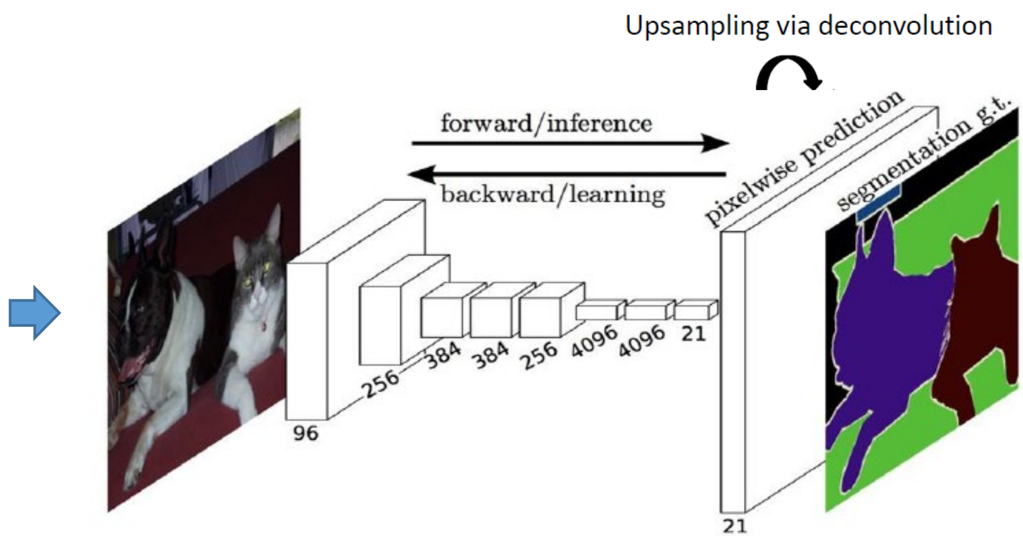


Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

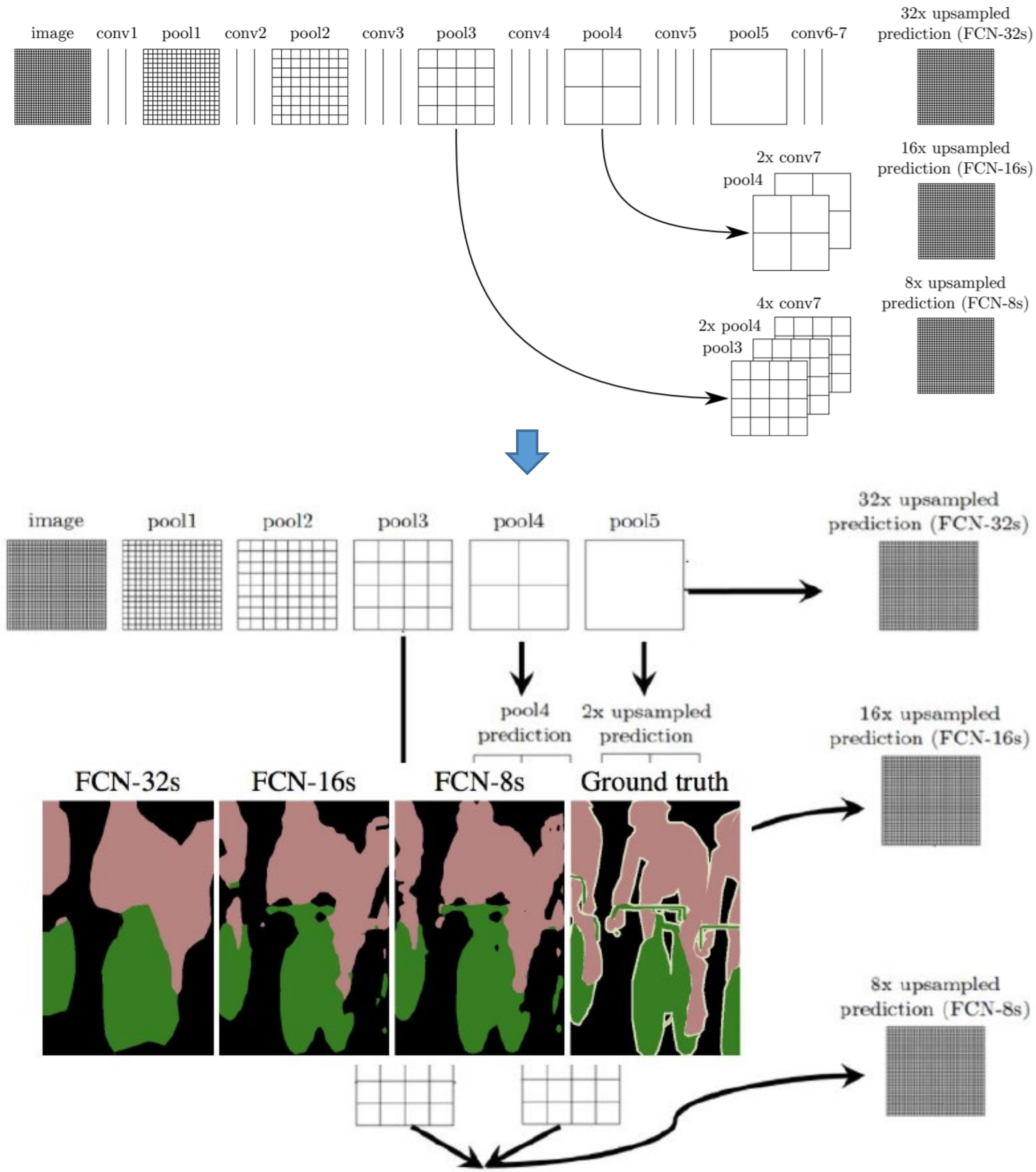
Fully Convolutional Networks (FCN)

- More details
  - Use transpose convolution to upsample pixel-wise classification results
  - Adapt existing classification network to fully convolutional forms
  - Remove flatten layer and replace fully connected layers with conv layers
  - Append 1 x 1 conv layer with channel dims to predict scores for each class





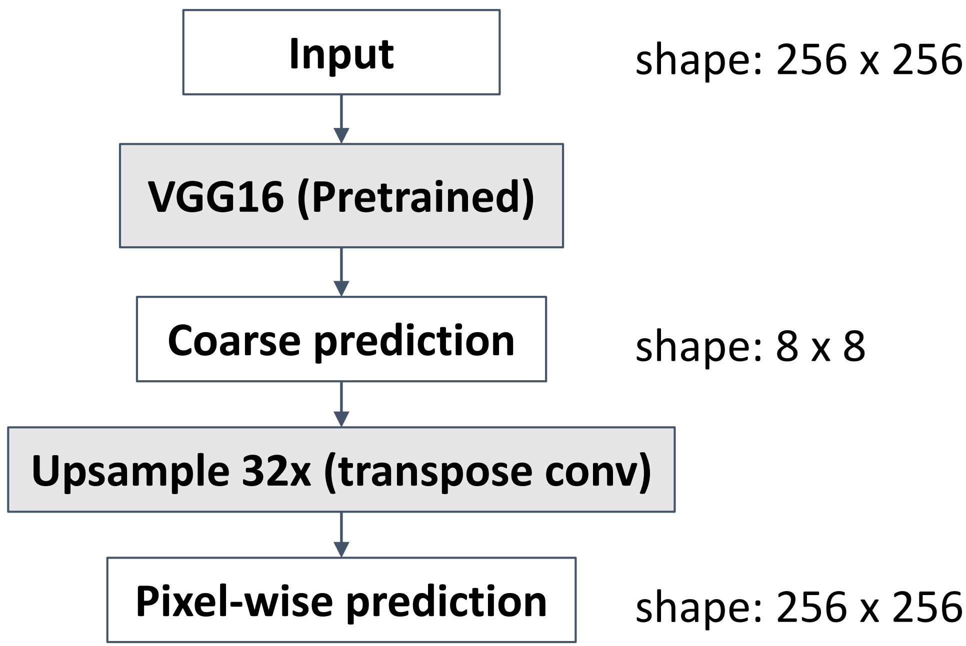
Fully Convolutional Networks (FCN)



Fully Convolutional Networks (FCN)

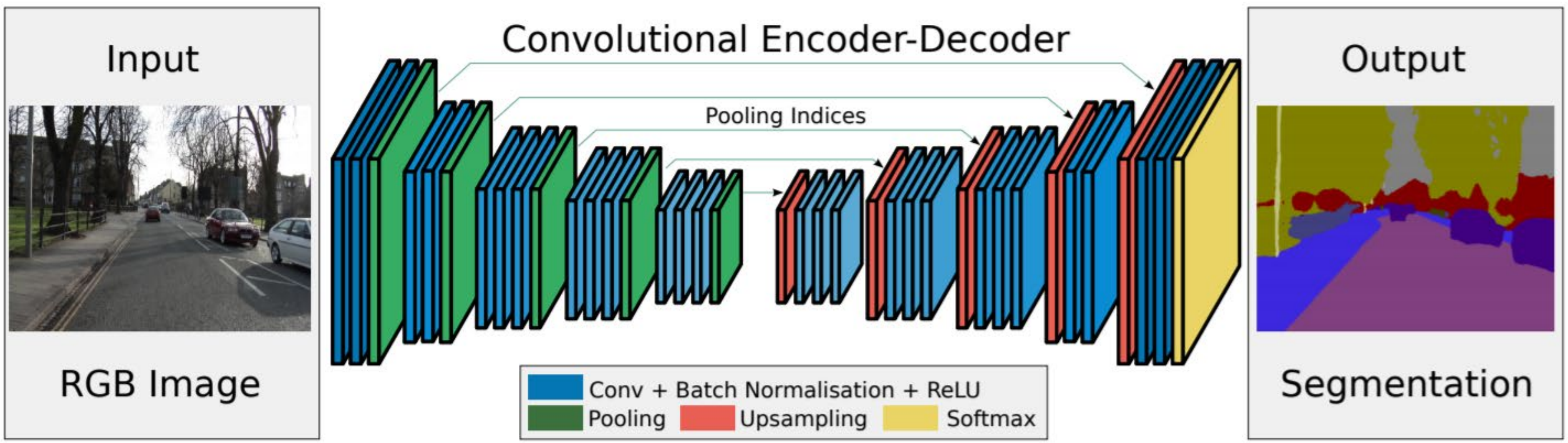
- Example
  - **VGG16-FCN32s**
  - Loss: pixel-wise cross-entropy

  i.e., compute cross-entropy between each pixel and its label, and average over all of them



SegNet

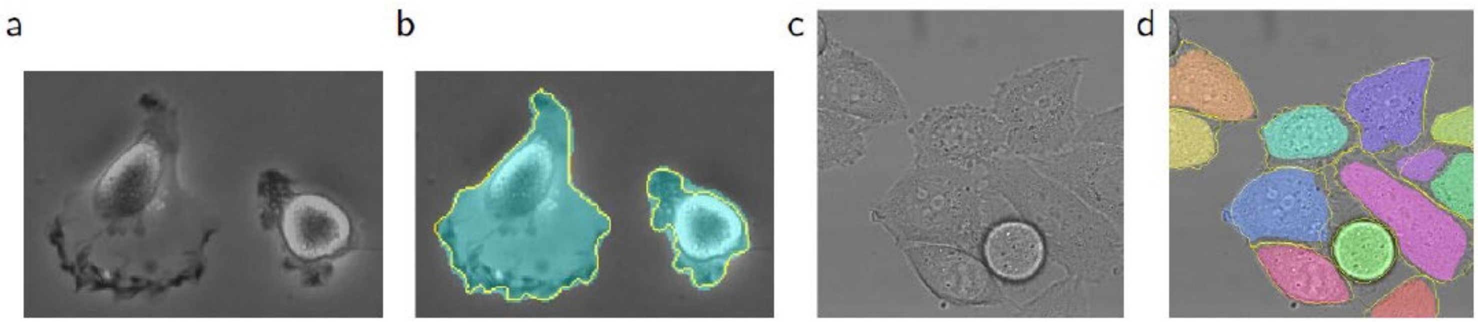
- Efficient architecture (memory + computation time)
- Upsampling reusing max-unpooling indices
- Reasonable results without performance boosting addition
- Comparable to FCN



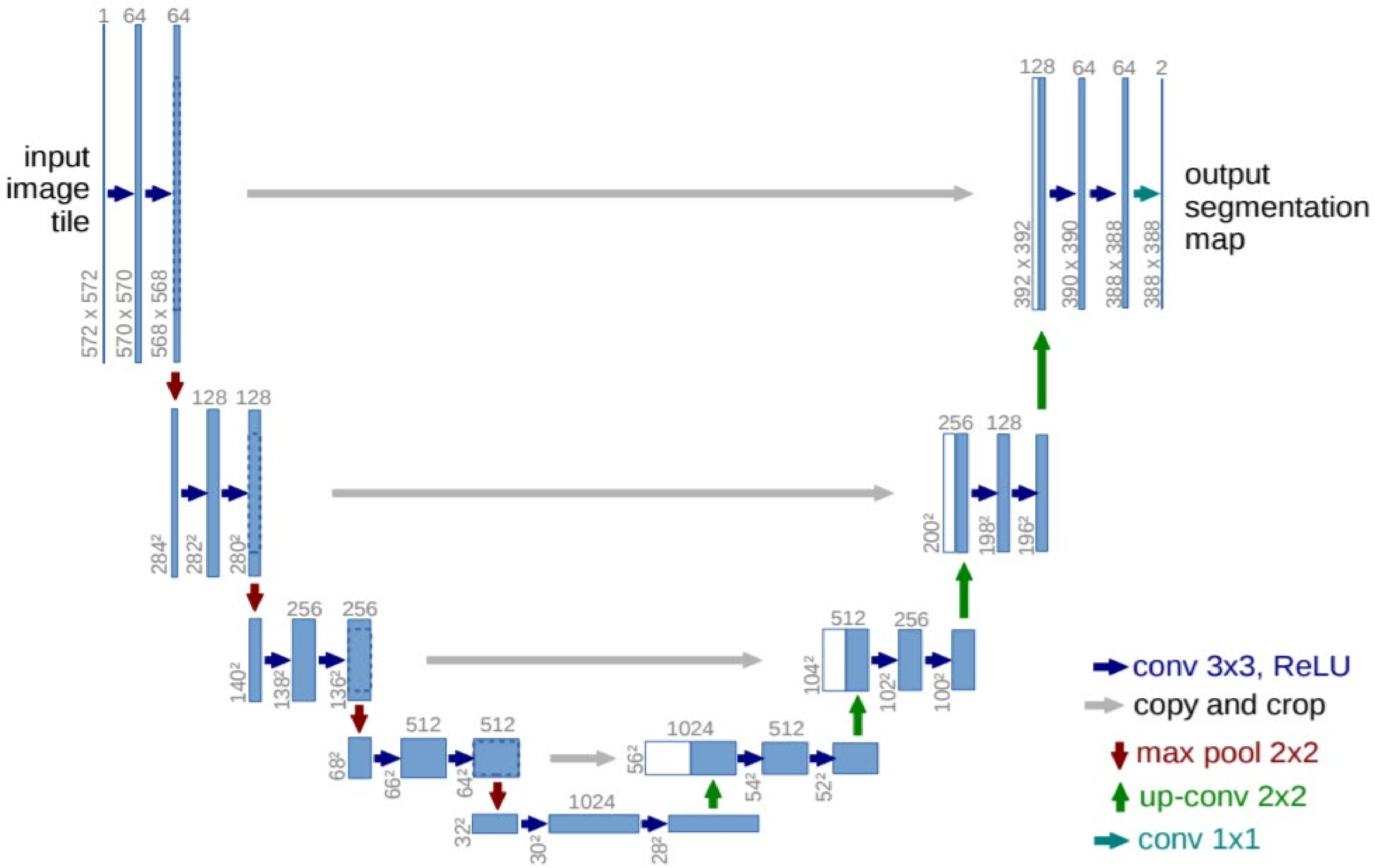
“SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation” [link]

U-Net (Ronneberger et al., MICCAI’15)

- Remarks
  - In biomedical image segmentation, localization is critical; in other words, precise semantic segmentation is desirable
  - Plus, # of training images might not be sufficient.



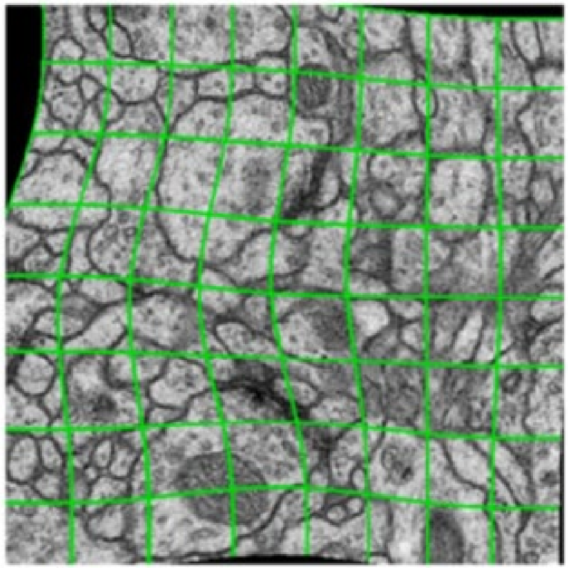
U-Net (cont'd)



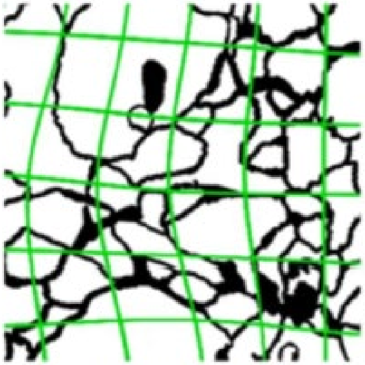
U-Net: Convolutional Networks for Biomedical Image Segmentation [link]

Additional Remarks:
Elastic Deformation for Pre-processing

- Data augmentation is crucial for U-Net (and more DL models)
- Elastic deformation allows manipulation of medical images & GT seg maps



resulting deformed image
(for visualization: no rotation, no shift, no extrapolation)



correspondingly deformed
manual labels

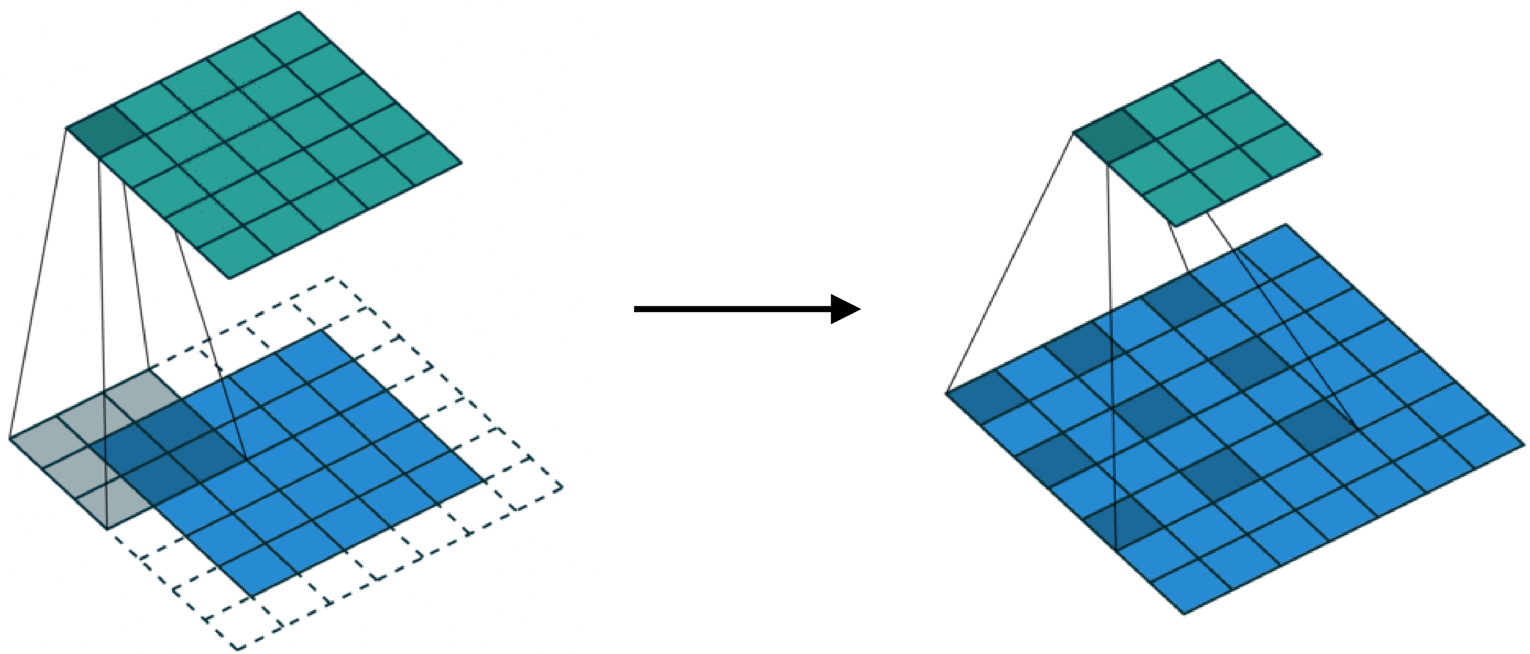
Code: [elastic_distortion.py]

Additional Remarks:
Enhanced Spatial Information

- For semantic segmentation,
  spatial information is of great importance
- It is desirable for the model to observe
  - Both the target pixel and its **neighboring areas**
    - Recall: **Atrous (or dilated) convolution**
  - Features across **different scales** should be considered
    - **Spatial Pyramid Pooling**

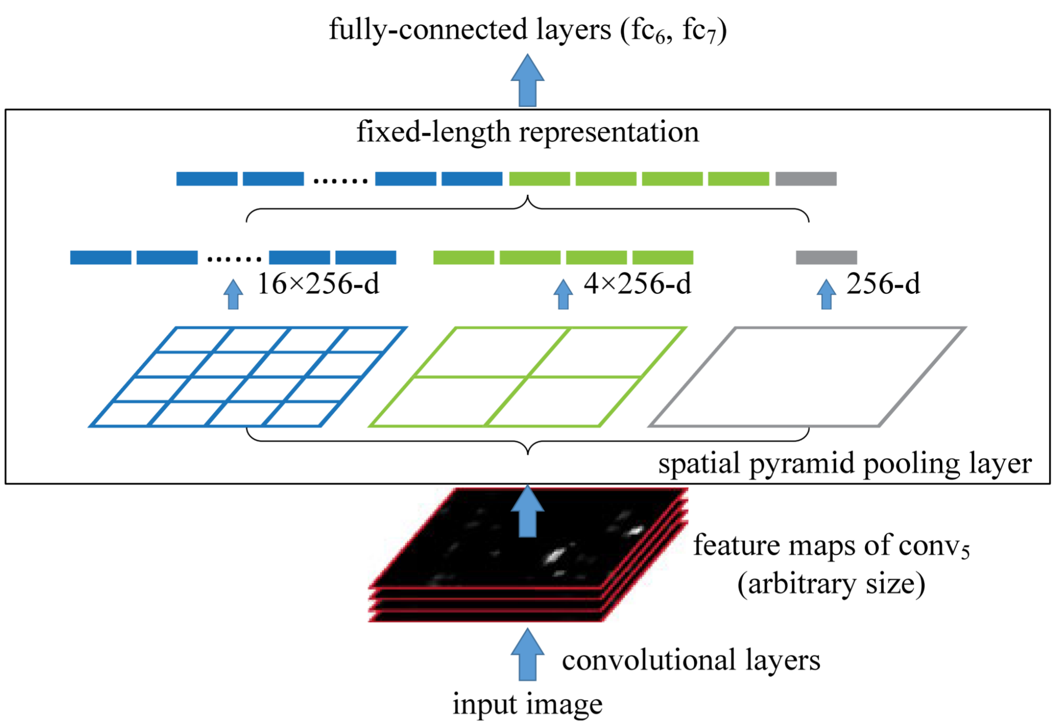
Revisit of Dilated Convolution

- Atrous (Dilated) Convolution
  - Larger receptive field with the same kernel size
    (e.g., a 3x3 kernel depicted below with different receptive field)



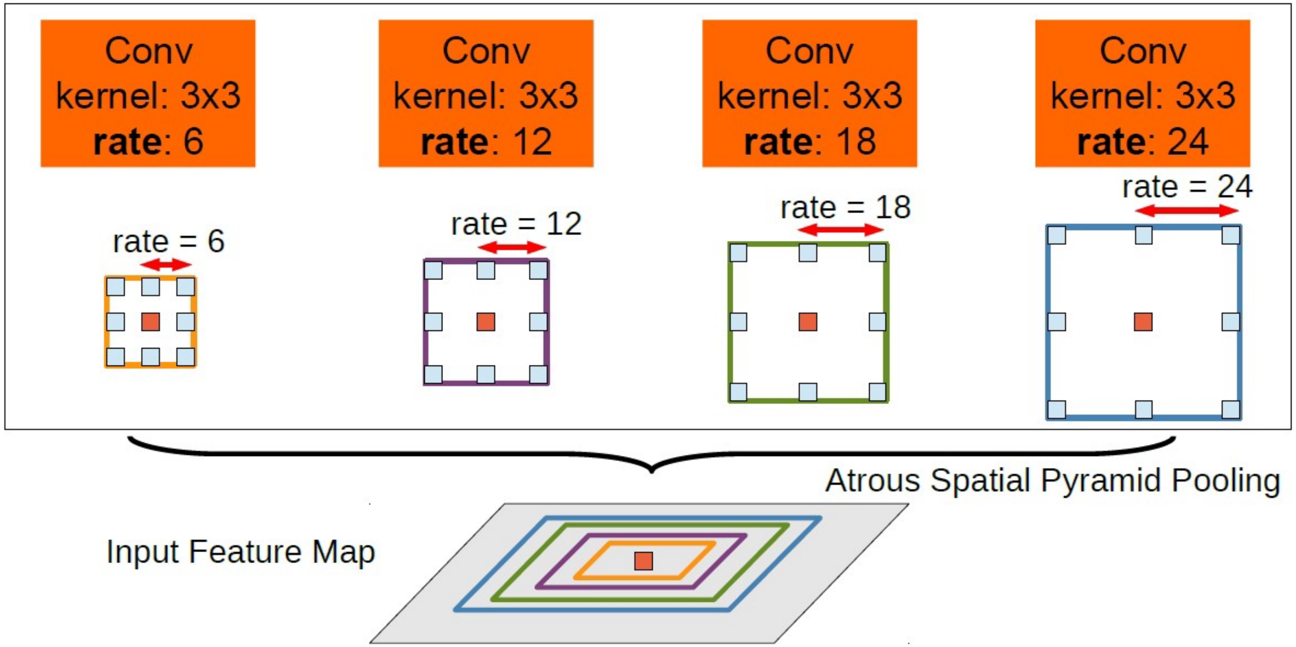
Spatial Pyramid Pooling (SPM)

- Goal:
  - Integrating information viewed under different scales

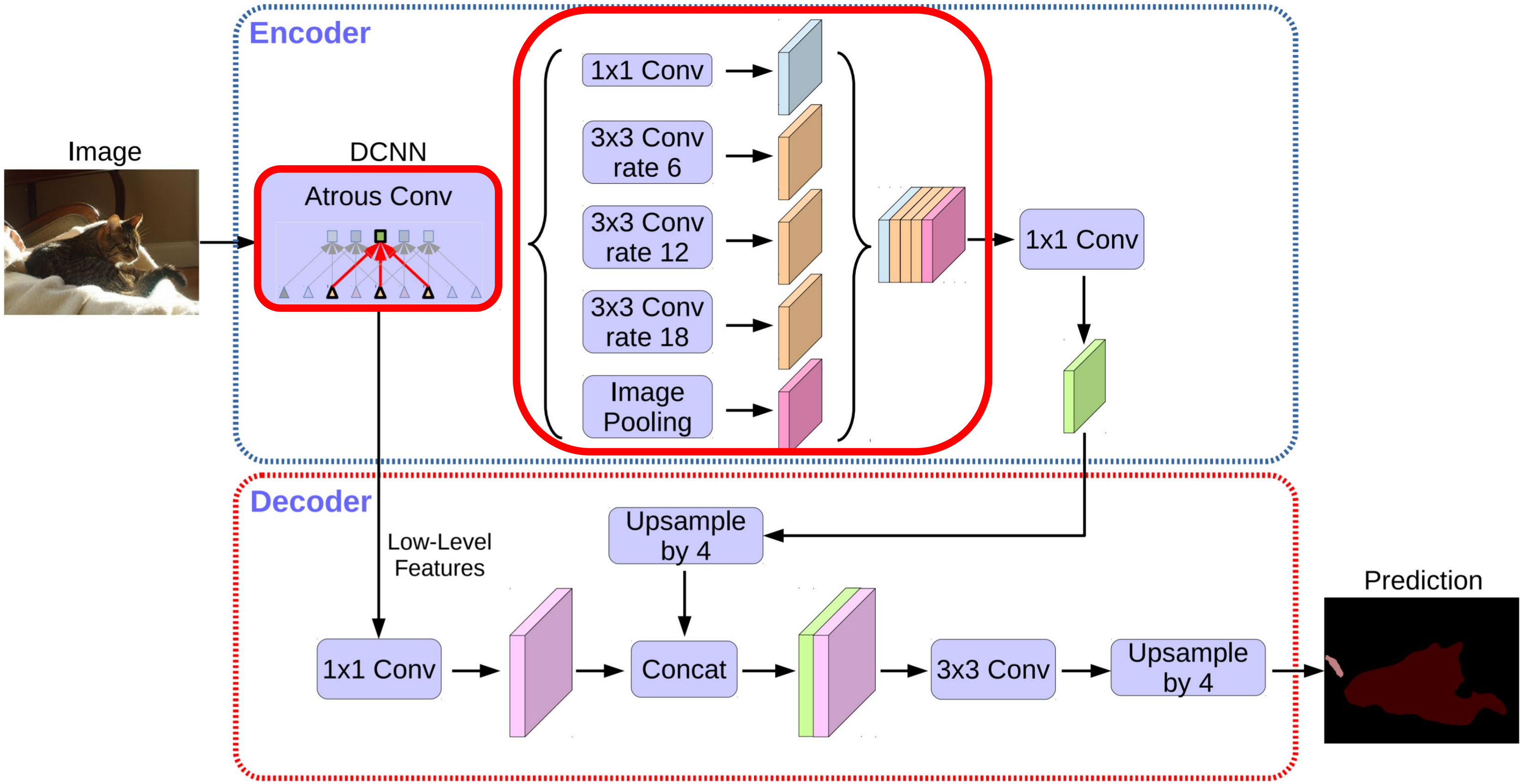


Thus, we have…

- Atrous Spatial Pyramid Pooling
  - Combines both techniques for producing enhanced spatial info



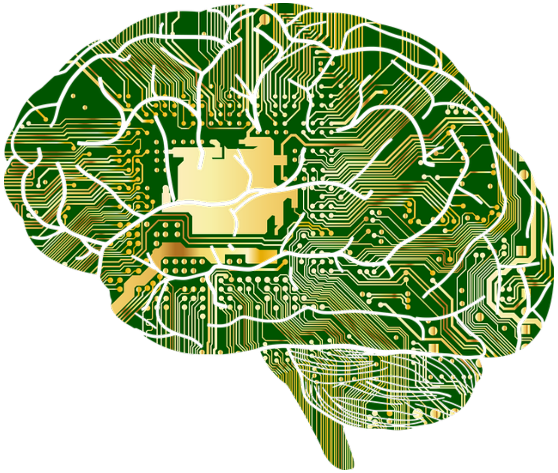
DeepLabv3+

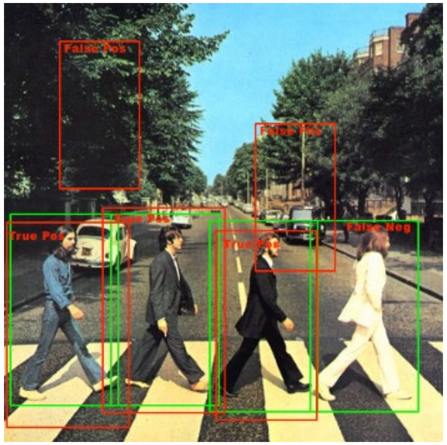


Chen et al. "Encoder-decoder with atrous separable convolution for semantic image segmentation," *ECCV* 2018

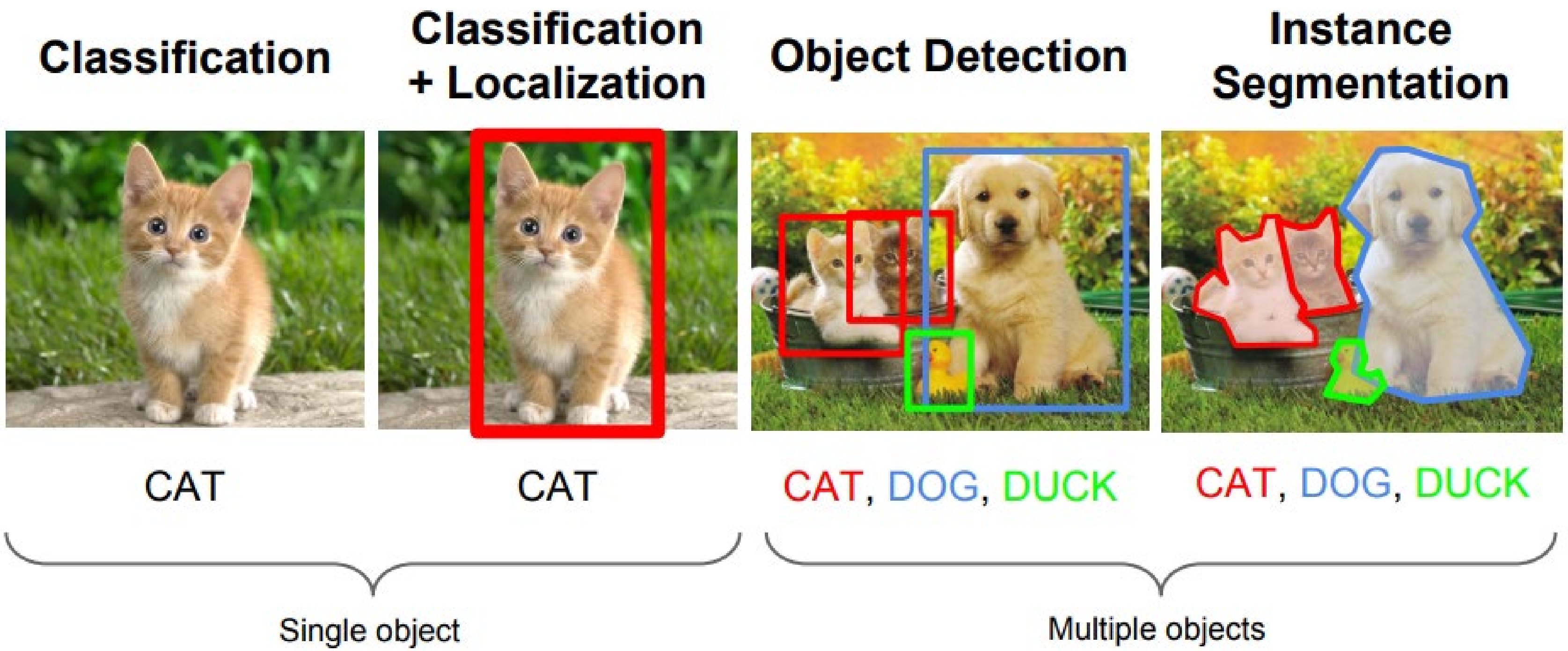
What’s to Be Covered Today…

- Segmentation
- Object Detection
- Generative Model





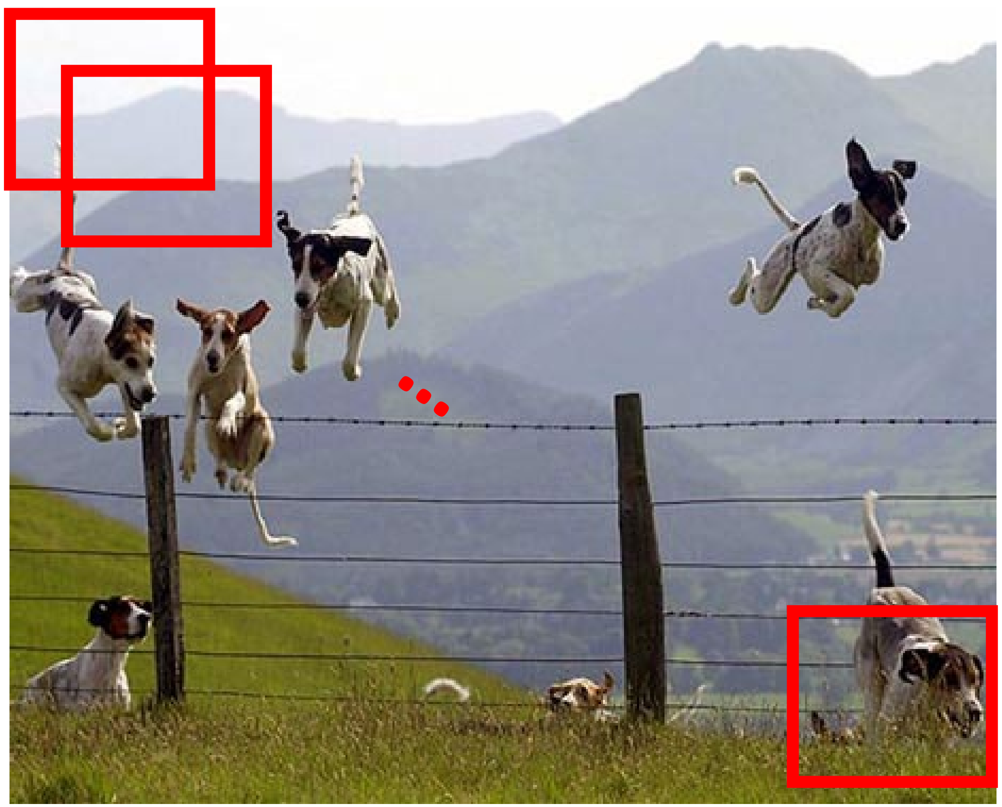
Roadmap



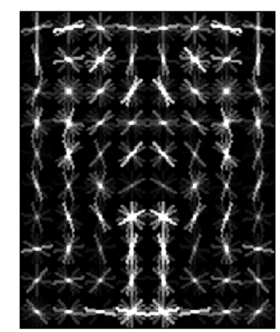
Slide from A. Karpathy

Object Category Detection

- Focus on object search: “Where is it?”
- Build templates that quickly differentiate object patch from background patch



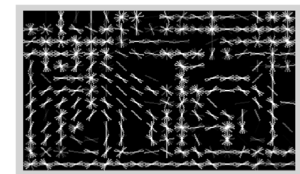
Dog Model



Object or

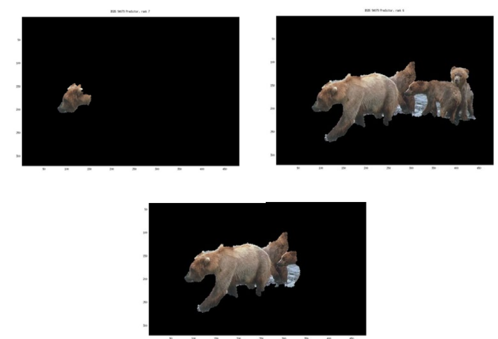
Non-Object?

General Process of Object Recognition



Specify Object Model

Generate Hypotheses



Score Hypotheses

Gradient/region based or CNN features, usually based on summary representation with classification/voting results

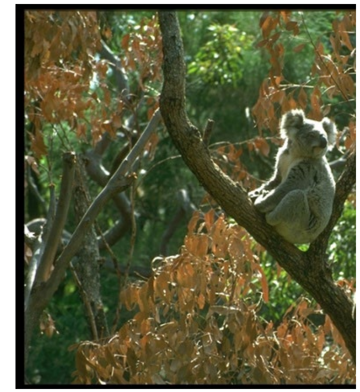
Resolve Detections

Rescore each proposed object based on the entire candidate set

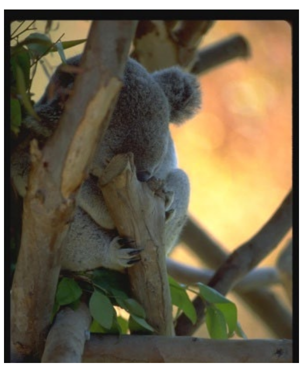
Challenges in Modeling the Object Classes

Illumination

Object pose



Clutter



Occlusion

Intra-class appearance

Viewpoint

Slide from K. Grauman, B. Leibe

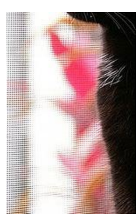
Challenges in Modeling the Non-object Classes

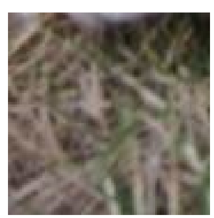
True Detection

Bad Localization

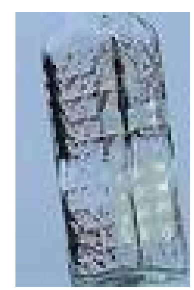
Confused with Similar Object

Misc. Background

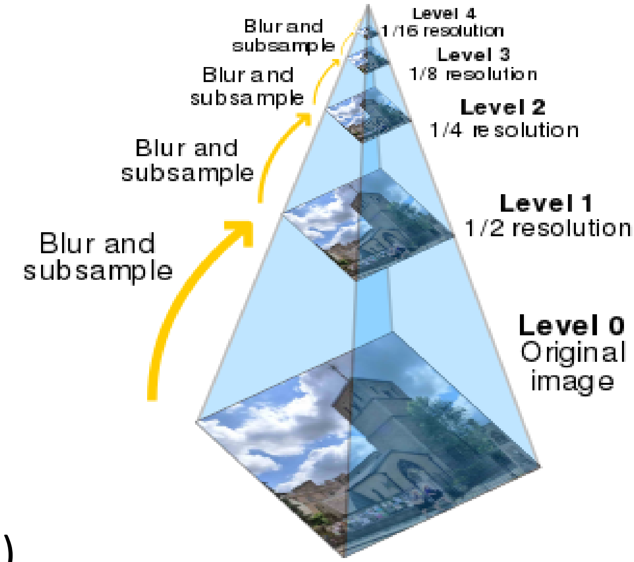




Confused with Dissimilar Objects



Type of Approaches

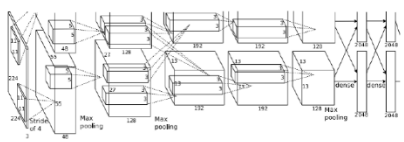


- Sliding Windows
  - "Slide" a box around the input image
  - Classify each cropped image region inside the box and determine if it's an object of interest or not
  - E.g., HOG (person) detector by Dalal and Triggs (2005)
    Deformable part-based model by Felzenswalb et al. (2010)
    Real-time (face) detector by Viola and Jones (2001)
- Region (Object) Proposals
  - Generate region (object) proposals
  - Classify each image region and determine it's an object or not

Type of Approaches (cont'd)

- CNN-based Methods

This image is CC0 public domain



Treat localization as a regression problem!

Vector: 4096

"What"

Fully Connected: 4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Correct label: Cat

Softmax Loss

"Where"

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

Correct box: (x', y', w', h')

L2 Loss

Weighted Sum

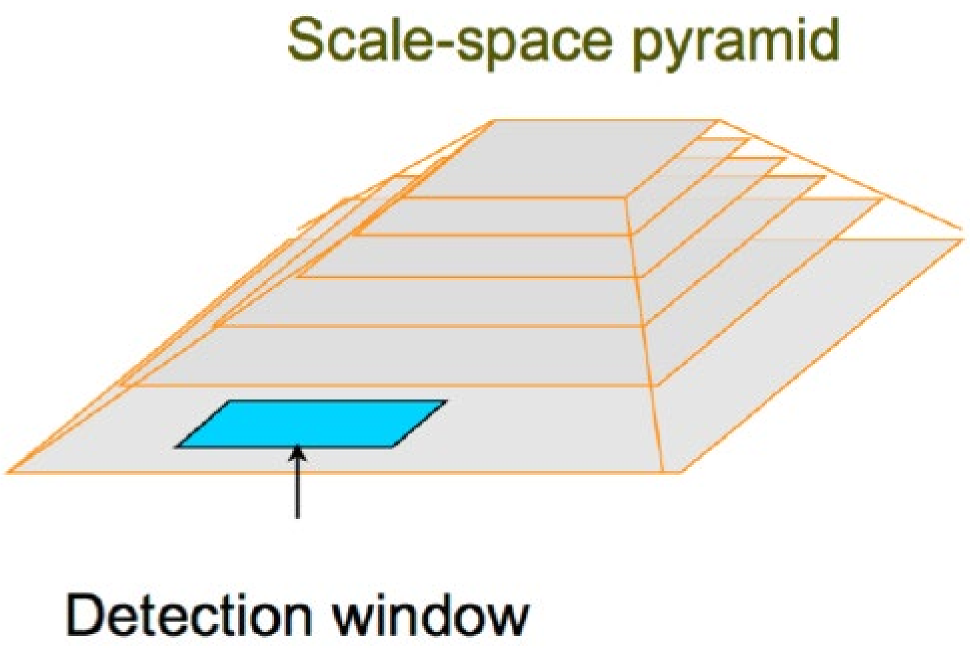
Loss

Slide credit: UMich EECS 498-007

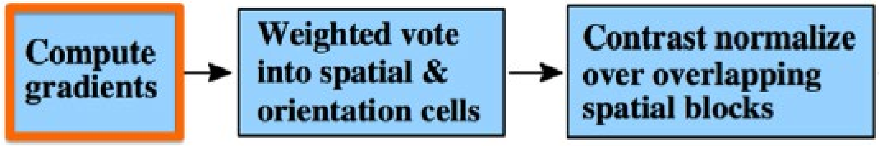
Before the Rise/Resurgence of CNN: The HOG Detector

- Histogram of Oriented Gradients
- Sliding window detector find objects in 4 steps:
  - Inspect every window
  - Extract features in window
  - Classify & accept window if score > threshold
  - Clean-up (post-processing) stage

- Step 1: Inspect every window
    - Objects can vary in sizes, what to do?
    - Sliding window + image pyramid!



- Step 2: Extract Features in Window
    - Histogram of Oriented Gradients (HOG) features
    - Similar to SIFT in some ways…
        - ever heard of SIFT?



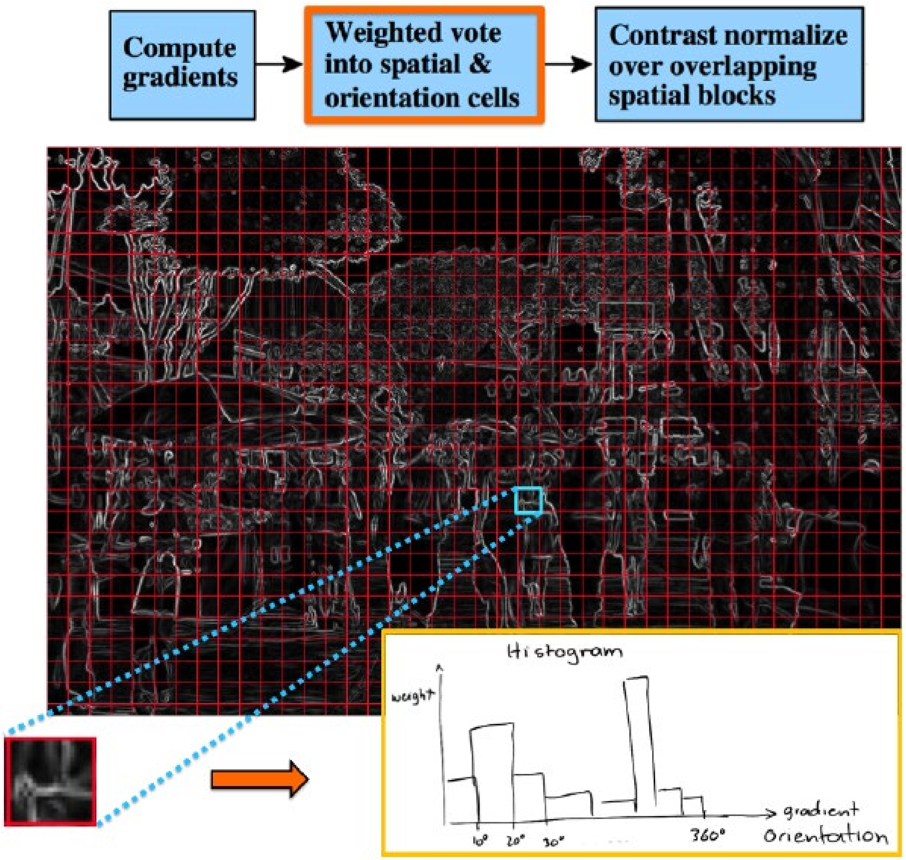
- Step 2: Extract Features in Window
  - Histogram of Gradients (HOG) features
  - Ways to compute image gradients…

Mask Type	1D centered	1D uncentered	1D cubic-corrected	2x2 diagonal	3x3 Sobel
Operator	[-1, 0, 1]	[-1, 1]	[1, -8, 0, 8, -1]	$\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Miss rate at 10^{-4} FPPW	11%	12.5%	12%	12.5%	14%

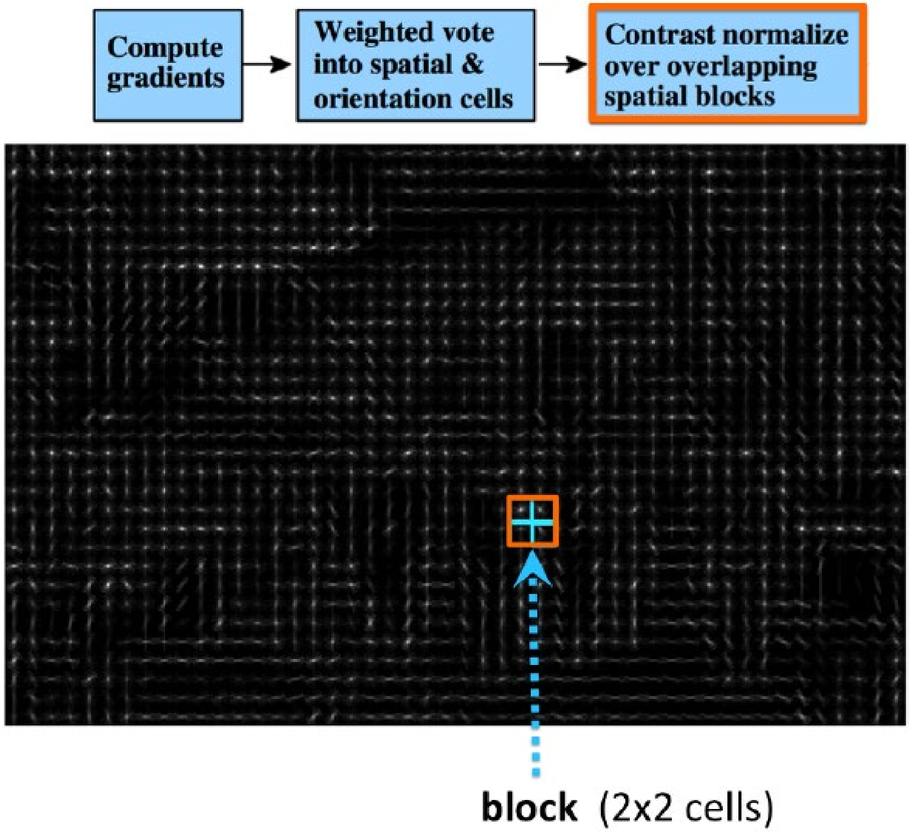
(Miss rate: smaller is better)

This gradient filter gives the best performance

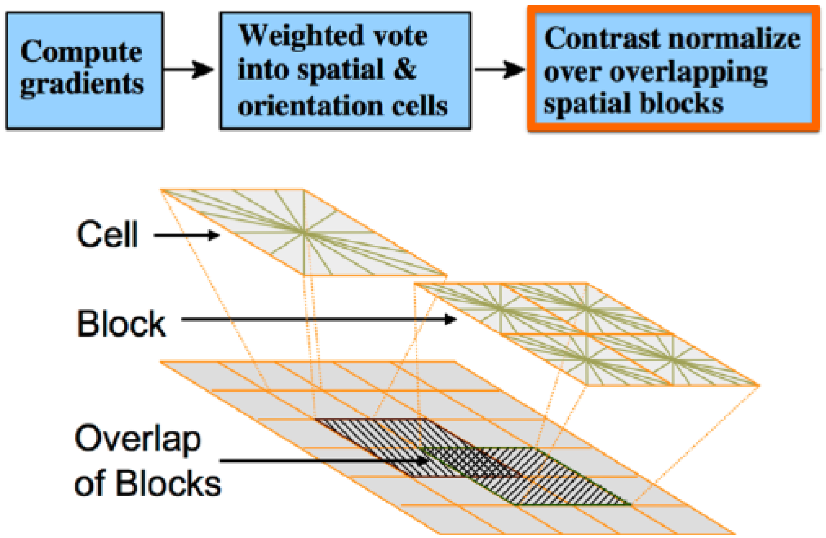
- Step 2: Extract Features in Window
  - Histogram of Gradients (HOG) features
  - Divide the image into non-overlapping cells (grids) of 8 x 8 pixels
  - Compute a histogram of orientations in each cell, resulting in a 9-dimensional feature vector.



- Step 2: Extract Features in Window
  - Histogram of Gradients (HOG) features
  - Divide the image into non-overlapping cells (grids) of 8 x 8 pixels
  - Compute a histogram of orientations in each cell (similar to SIFT), resulting in a 9-dimensional feature vector.
  - We now take blocks, where each has 2 x 2 cells, for HOG normalization.



- Step 2: Extract Features in Window
  - Compute a histogram of orientations in each cell (similar to SIFT), resulting in a 9-dimensional feature vector.
  - We now take blocks, where each has 2 x 2 cells, for HOG normalization
  - Normalize each feature vector, such that each block has unit norm. This does not change the dim of the feature, just the magnitude.

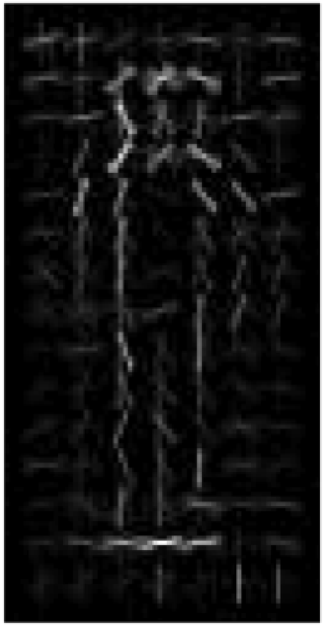


Feature vector *f* = [..., ..., ...]

L2 normalization in each block:

$$\mathbf{f} = \frac{\mathbf{f}}{\sqrt{||\mathbf{f}||_2^2 + \epsilon^2}}$$

- Step 2: Extract Features in Window
  - Normalize each feature vector, such that each block has unit norm. This does not change the dim of the feature, just the magnitude.
  - Each cell is in 4 blocks thus has 4 different normalizations; we make each as a feature representation.
  - For each class of *person*, window is 15 x 7 HOG cells.
  - We vectorize each the feature matrix in each window.



Final descriptor for window
(person class in this case)

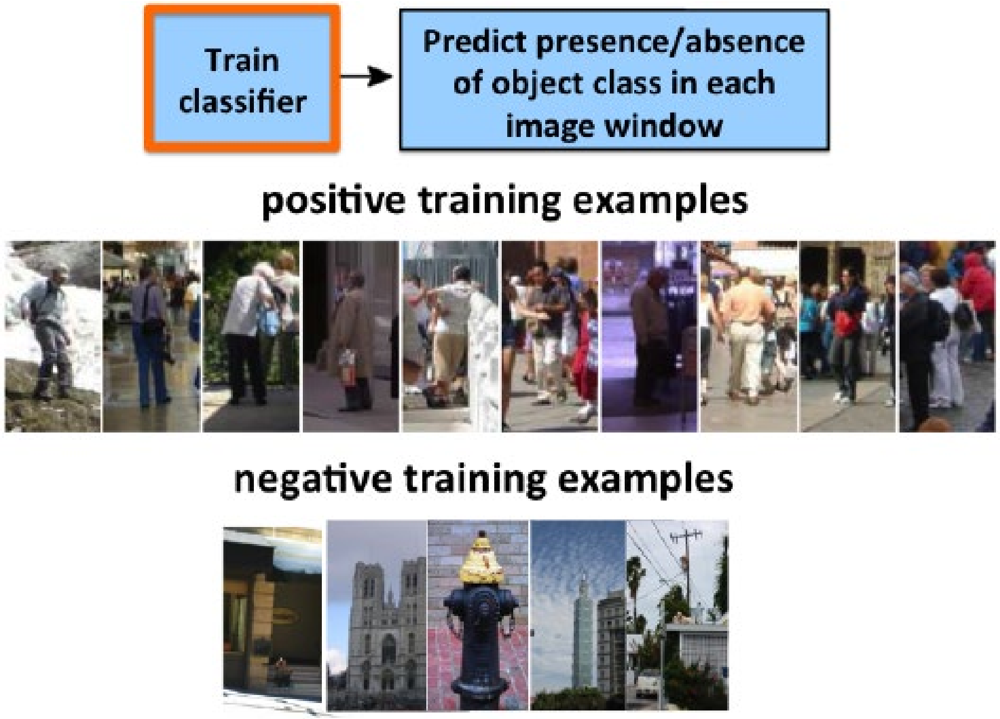
orientations

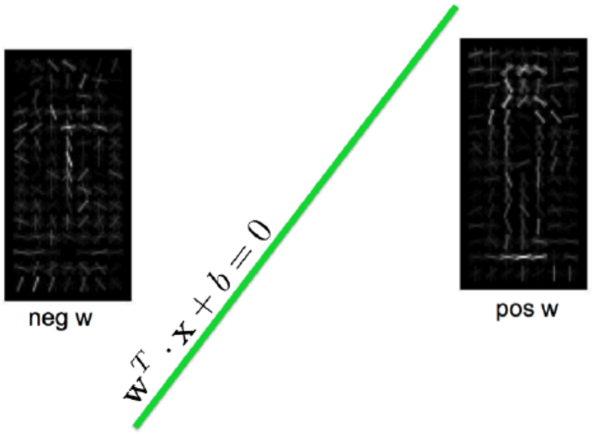
features = 15 x 7 x 9 x 4 = 3780

cells

normalizations by neighboring cells

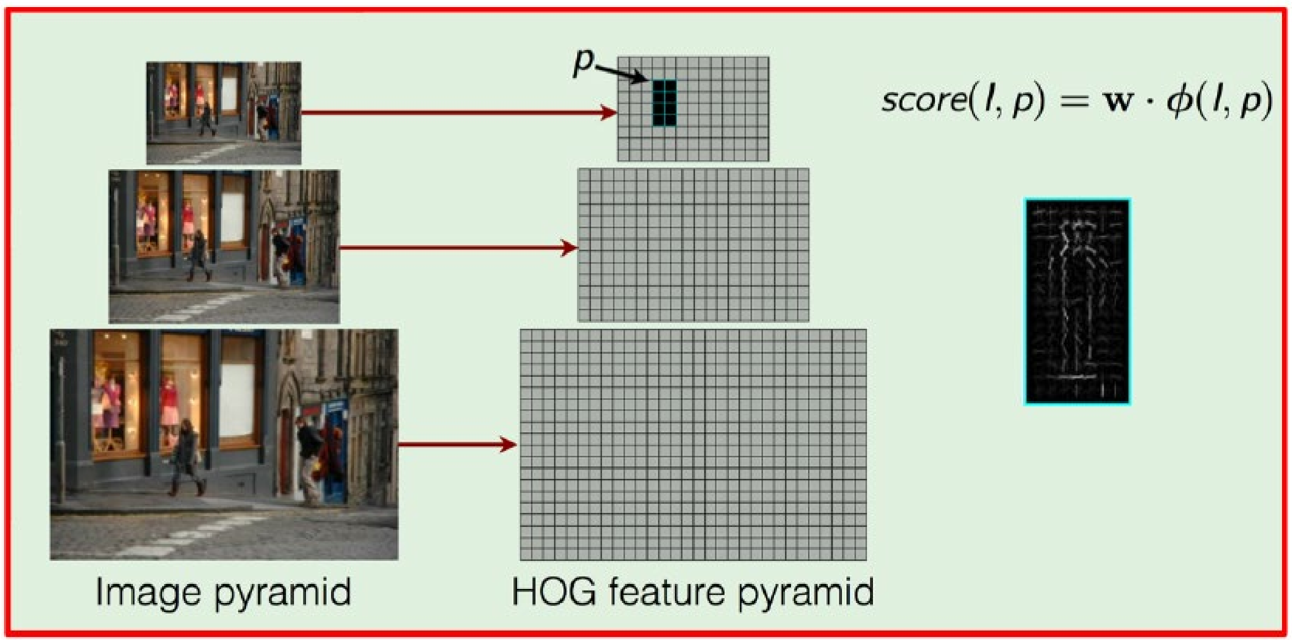
- Step 3: Detection (classify & accept window if score > threshold)
  - Train a window classifier (e.g., linear or non-linear classifiers)
  - Use the trained classifier to predict presence of object class in each window





Train classifier. SVM (Support Vector Machines) is typically used.

- Step 3: Detection (Classify & accept window if score > threshold)
  - Train a window classifier
  - Use the trained classifier to predict presence of object class in each window
  - During testing, compute the score $\mathbf{w}^T\mathbf{x}+\mathbf{b}$ in each location, which can be viewed as performing cross-correlation (or convolution) with template **w** (and add bias **b**).



- Step 4: **Cleaning-Up**
  - Perform a greedy algorithm of **non-maxima suppression (NMS)** to pick the bounding box with highest score

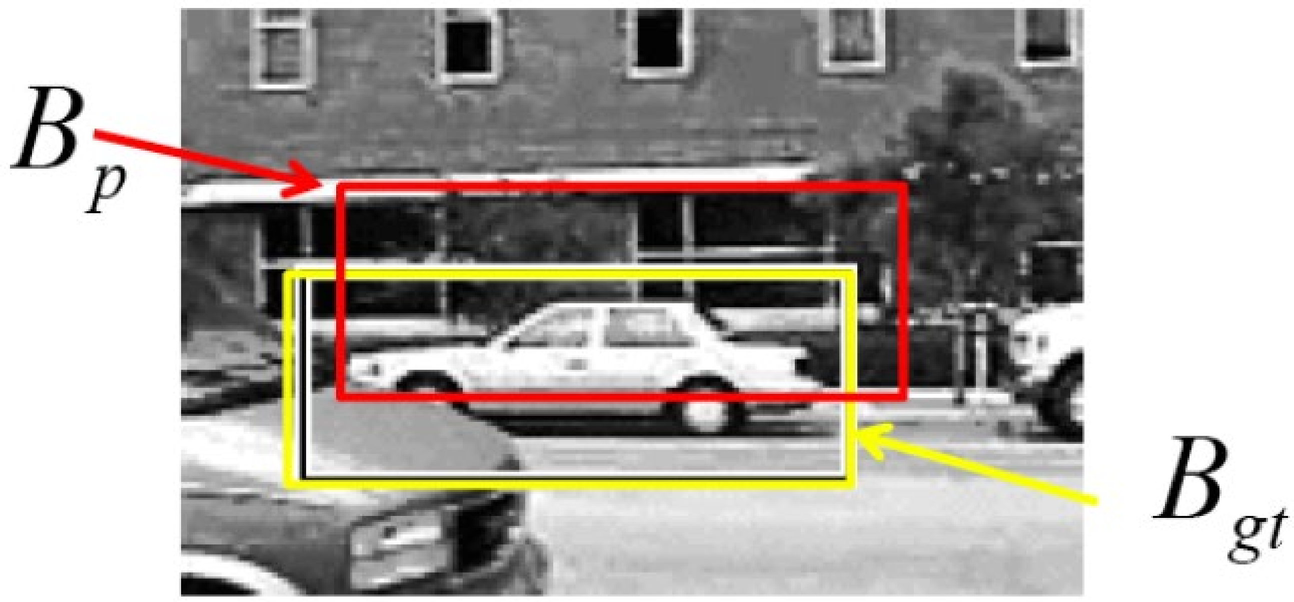
Non-maxima suppression (NMS)

$$\text{overlap} = \frac{\text{area}(box_1 \cap box_2)}{\text{area}(box_1 \cup box_2)} > 0.5 \Rightarrow \boxed{\text{remove } box_2}$$

- Remove all boxes that overlap more than XX (typically 50%) with the chosen box

- Evaluation
  - IoU (intersection over union)
    - E.g, detection is correct if IoU between bounding box and ground truth > 50%

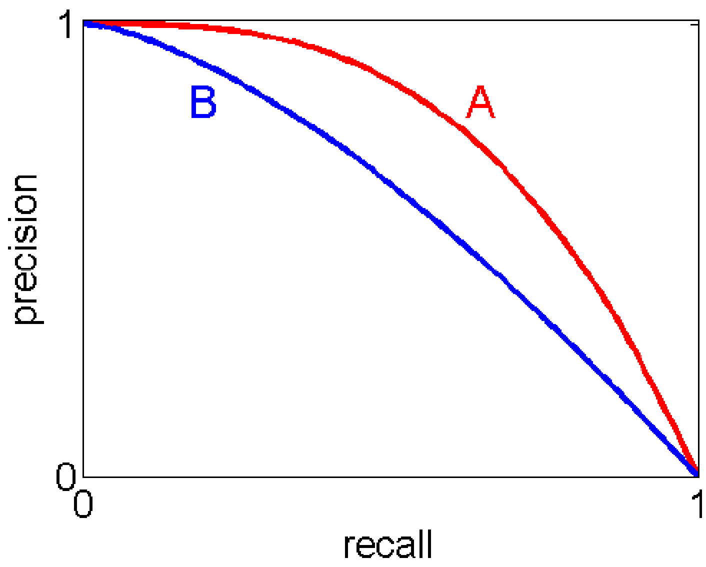
$$a_0 = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})}$$



- Evaluation
  - IOU (intersection over union)
    - Mean IOU (mIOU): average IOU across classes
  - Precision and Recall
    - Sort all the predicted boxes according to scores, in a descending order
    - For each location in the sorted list,
      we compute precision and recall obtained when using top k boxes in the list.

$$\text{recall} = \frac{\#\text{correct boxes}}{\#\text{ground-truth boxes}}$$

$$\text{precision} = \frac{\#\text{correct boxes}}{\#\text{all predicted boxes}}$$

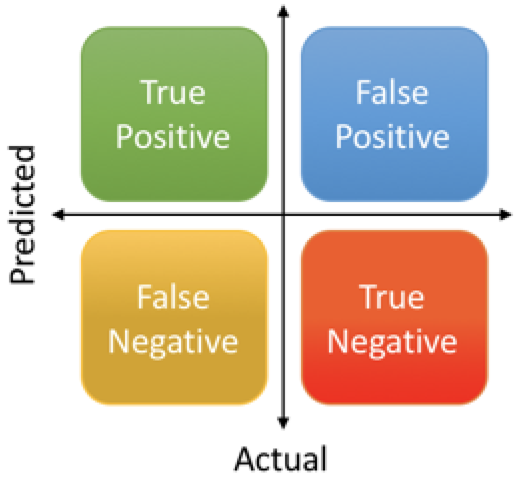


- Evaluation
    - IOU (intersection over union)
    - Precision and Recall

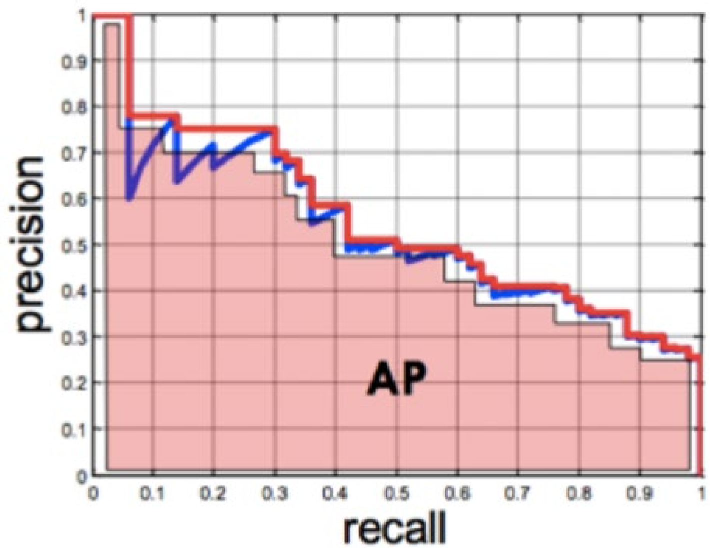
$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

$$\textbf{Accuracy} = \frac{\text{True Positive + True Negative}}{\text{Total}}$$



- Evaluation
  - IoU (intersection over union)
  - Precision and Recall
  - Average Precision (AP):
    - Compute the area under P-R curve
    - mean Average Precision (mAP): average of AP across classes



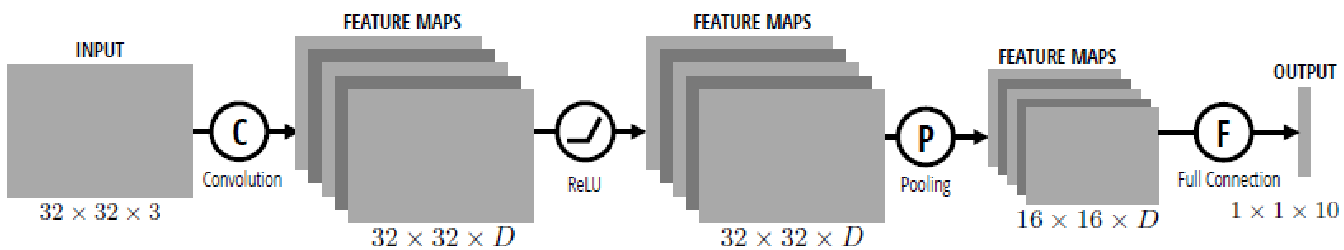
Something to Think About…

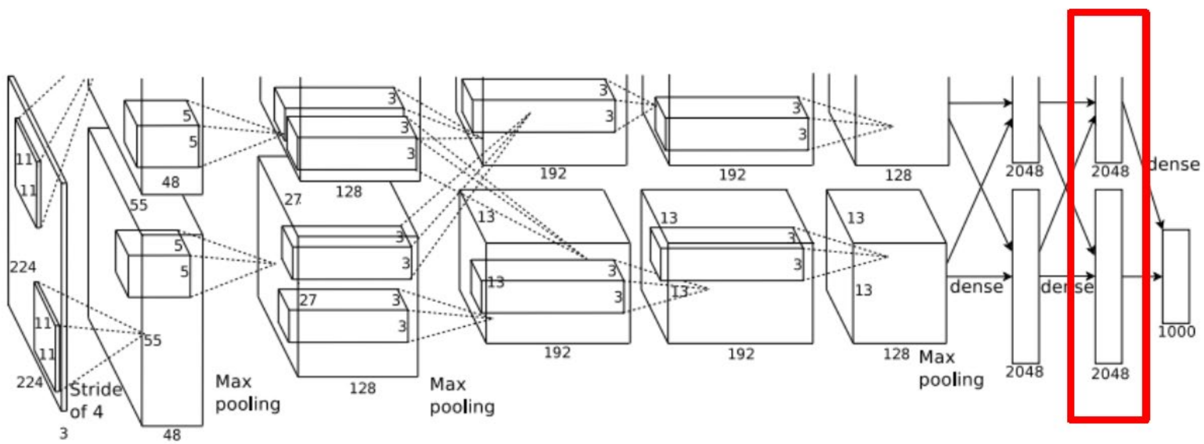
- Sliding window detectors work
  - *very well* for faces
  - *fairly well* for cars and pedestrians
  - *badly* for cats and dogs

- Why are some classes easier than others?

Recall that

- Visual Features derived by Convolutional Neural Networks



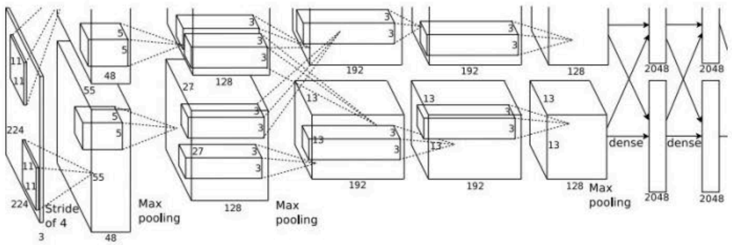


CNN as Feature Extractor

Image credit: Justin Johnson

CNN as Feature Extractor

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

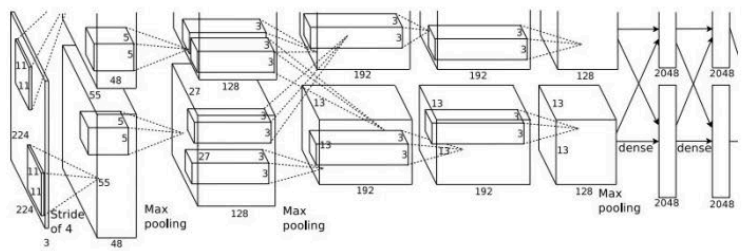


Dog? YES
Cat? NO
Background? NO

Slides by Justin Johnson

CNN as Feature Extractor

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

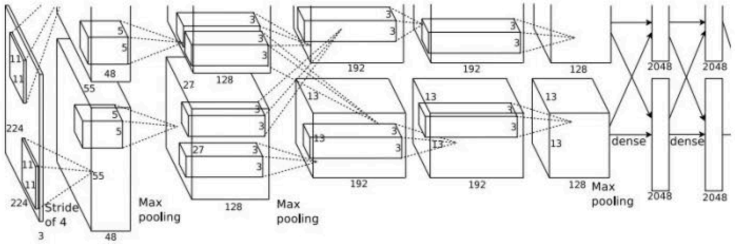


Dog? YES
Cat? NO
Background? NO

Slides by Justin Johnson

CNN as Feature Extractor

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? YES
Background? NO

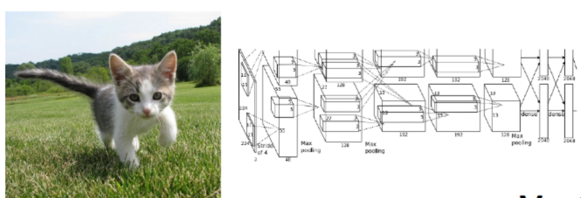
Slides by Justin Johnson

CNN as Feature Extractor

- What could be the problems?
  - Suppose we have an image of 600 x 600 pixels.
    If sliding window size is 20 x 20,
    then have (600-20+1) x (600-20+1) = ~330,000 windows to compute.
  - What if more accurate results are needed,
    need to perform multi-scale detection by
    - Resize image
    - Multi-scale/shape sliding windows
  - For each image, we need to forward pass image regions through CNN
    for at least ~330,000 times. -> **Slow!!!**

Recap: CNN for Object Detection

- Need to deal with more than one object
  - How?



This image is CC0 public domain

Treat localization as a regression problem!

Vector: 4096

Fully Connected: 4096 to 1000

"What"

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Correct label: Cat

Softmax Loss

Weighted Sum → **Loss**

Fully Connected: 4096 to 4

"Where"

Box Coordinates (x, y, w, h)

L2 Loss

Correct box: (x', y', w', h')

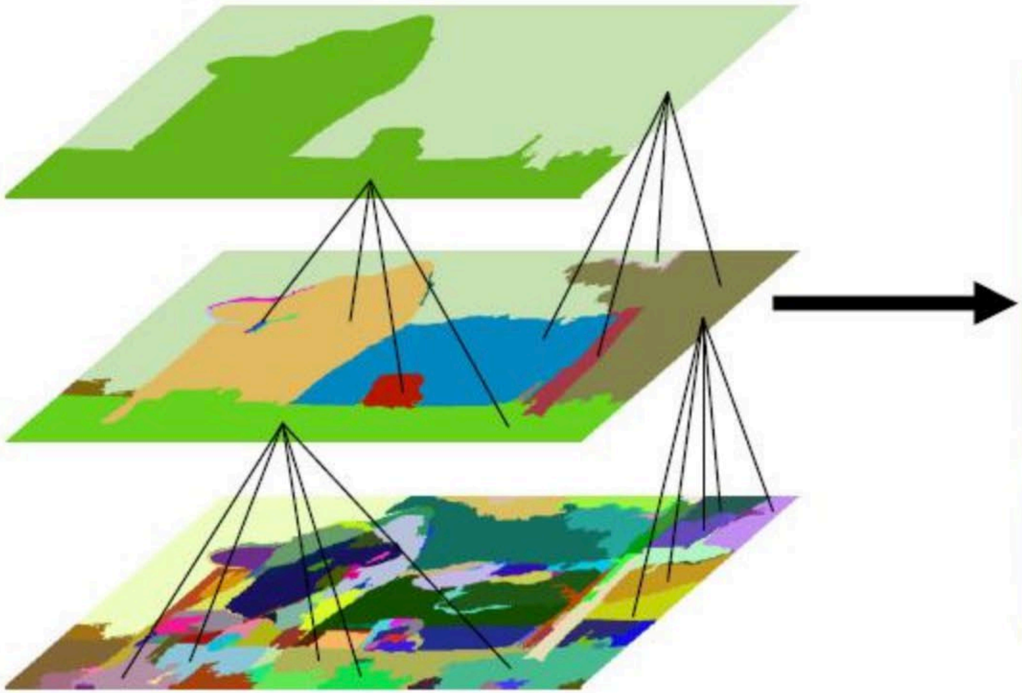
Two-Stage vs. One-Stage Object Detection

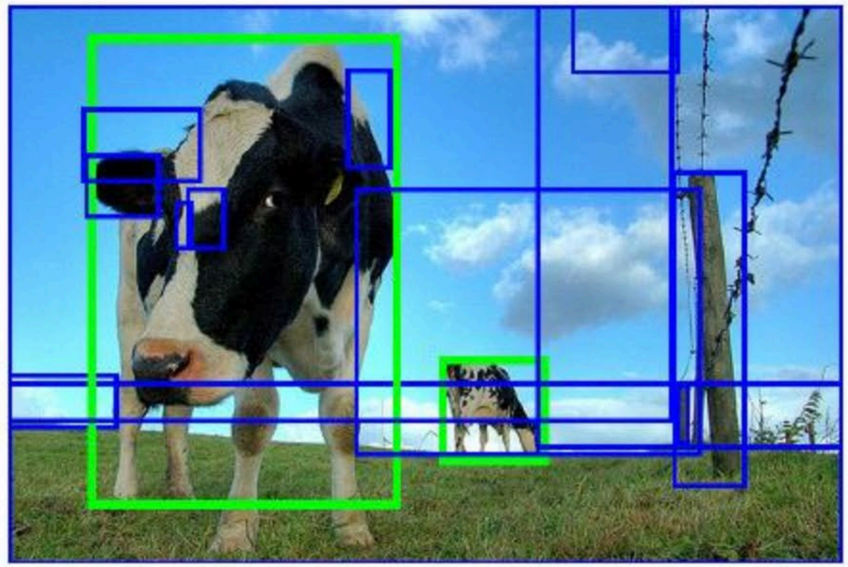
Methods

- Sliding Windows
- **Two–stage Frameworks**
  - R–CNN
  - Fast R–CNN
  - Mask R–CNN
  - ⋮
- **One–stage Frameworks**
  - YOLO
  - YOLOv2
  - YOLOv3
  - ⋮

Region Proposal

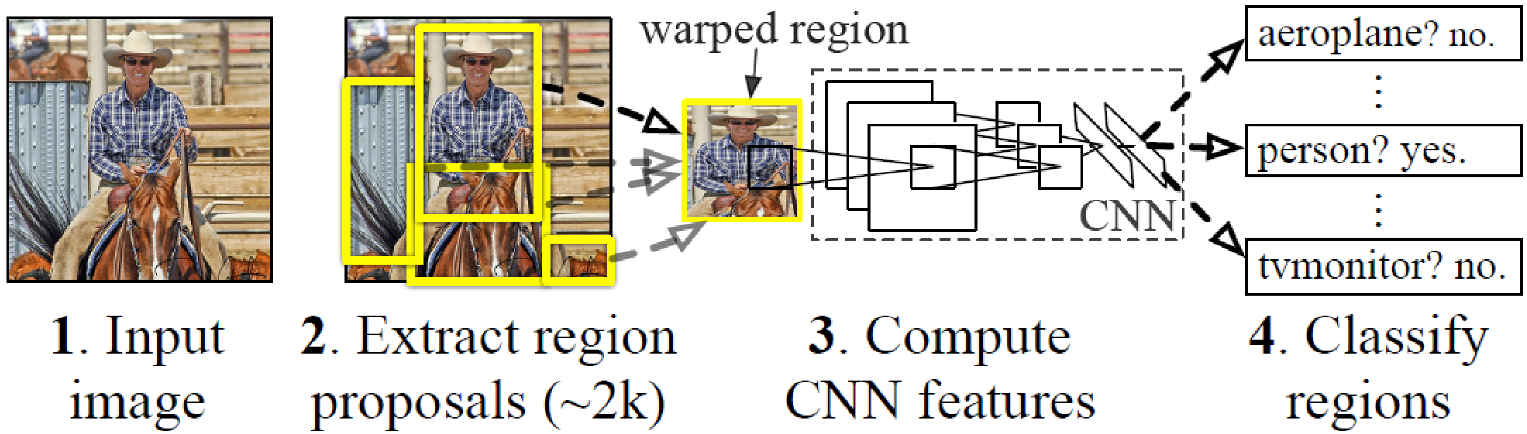
- Solution
  - Use pre-processing algorithms to filter out some regions first, and feed the regions of interest (i.e., region proposals) into CNN
  - E.g., selective search





Uijilings et al. IJCV 2013

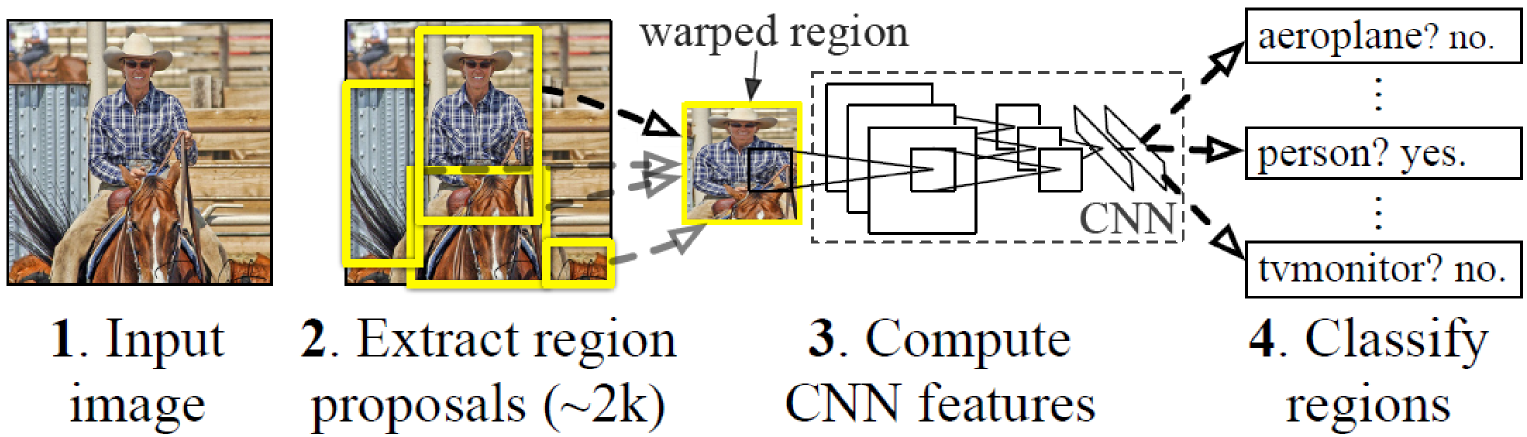
R-CNN (Girshick et al. CVPR 2014)



- Replace sliding windows with “selective search” region proposals (Uijilings et al. IJCV 2013)
- Extract rectangles around regions and resize to 227x227 pixels
- Extract features with fine-tuned CNN (e.g., initialized with network pre-trained on ImageNet)
- Classify last layer of network features with linear classifiers (e.g., SVM/MLP), and refine bounding box localization (bbox regression) simultaneously

http://arxiv.org/pdf/1311.2524.pdf

R-CNN (Girshick et al. CVPR 2014)

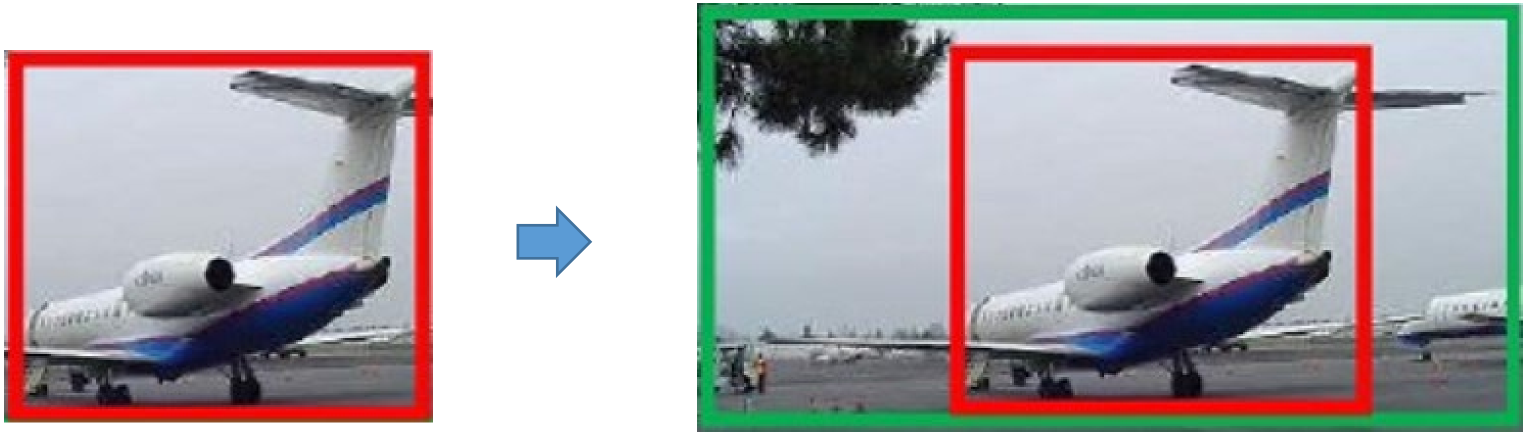


- Ad hoc training objectives:
  - Object class: Fine-tune network with softmax classifier (log loss)
  - Object class: Train post-hoc linear SVMs for each class (hinge loss)
  - Bbox location: Train post-hoc bounding-box regressors (least squares loss)
- Training is extremely slow with lots of disk space.
- Implementation/testing cannot be done in real time.

http://arxiv.org/pdf/1311.2524.pdf

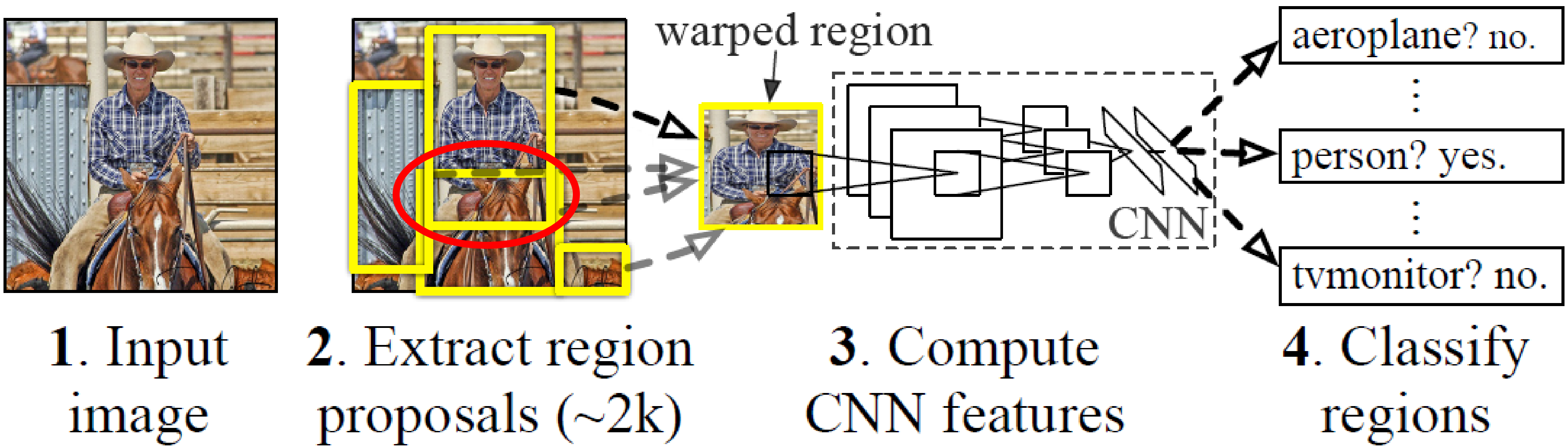
Bounding Box Regression

- Intuition
  - If you observe parts of an object, according to the seen examples, you should be able to predict/refine the localization.
    - E.g., given the red bounding box below, since you've seen many airplanes, you know this is not a good localization, you will adjust it to the green one.



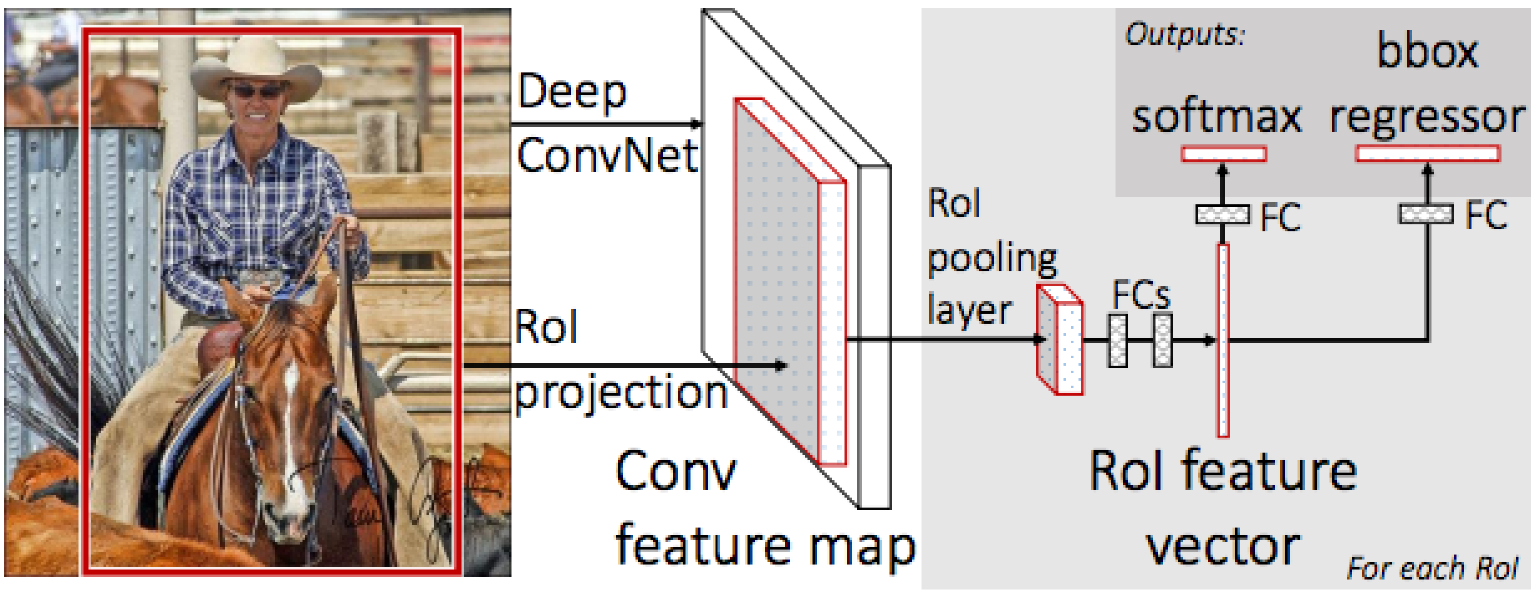
R-CNN (Girshick et al. CVPR 2014)

- What could be the problems?
  - Repetitive computation!
    For overlapping regions, we feed it multiple times into CNN



Fast R-CNN (Girshick ICCV 2015)

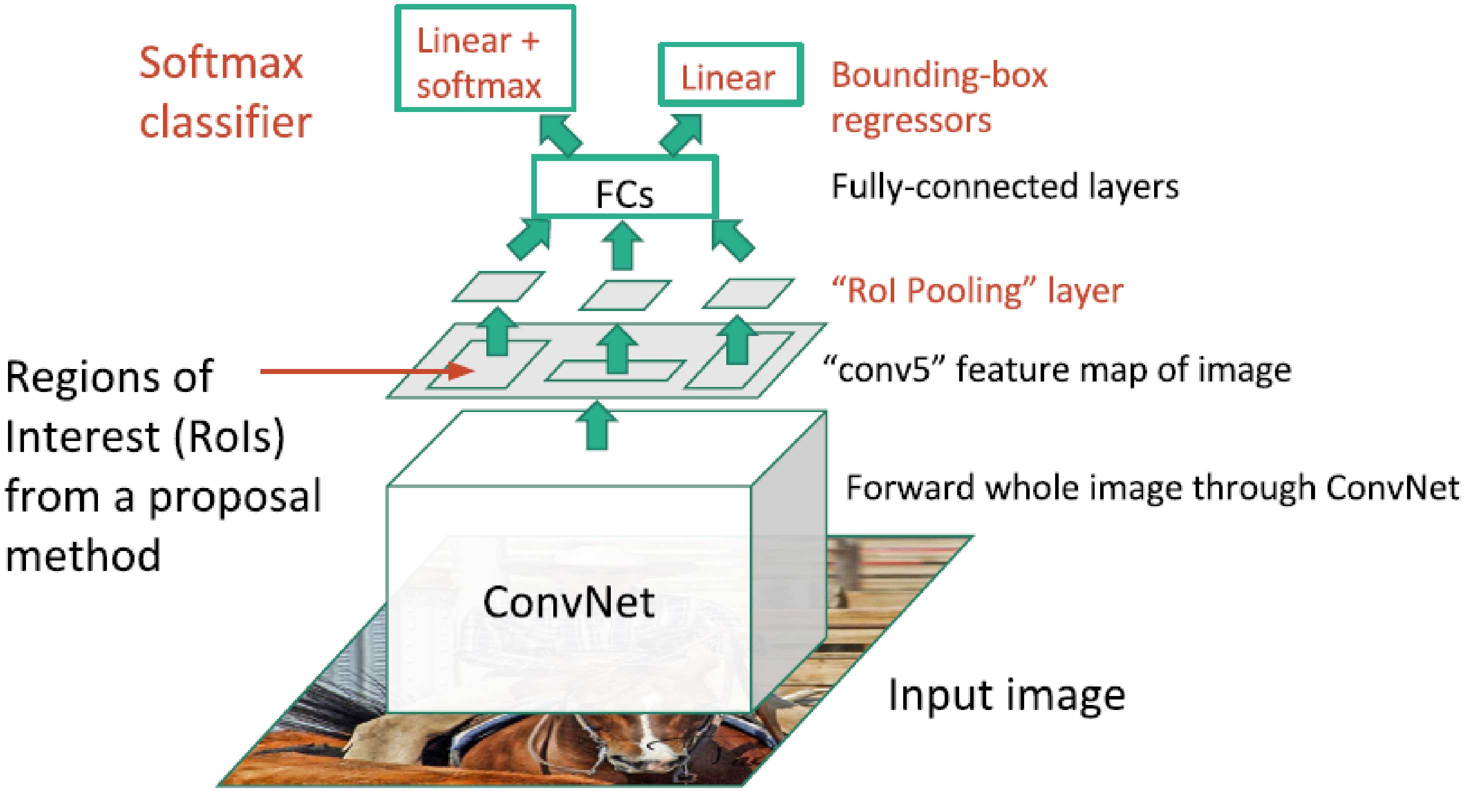
- Solution
  - Why not feed the whole image into CNN only once?
  - Then, crop the feature map instead of the image itself



https://arxiv.org/pdf/1504.08083.pdf

Fast R-CNN (Girshick ICCV 2015)

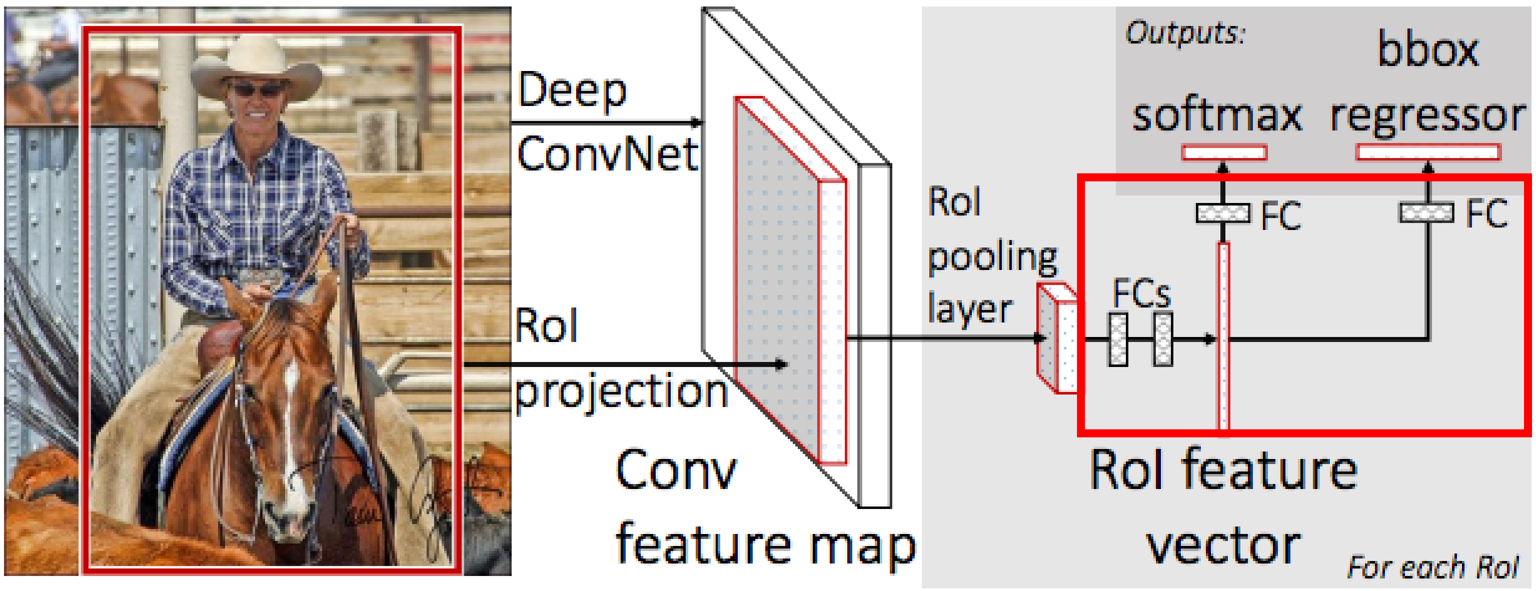
- Solution



https://arxiv.org/pdf/1504.08083.pdf

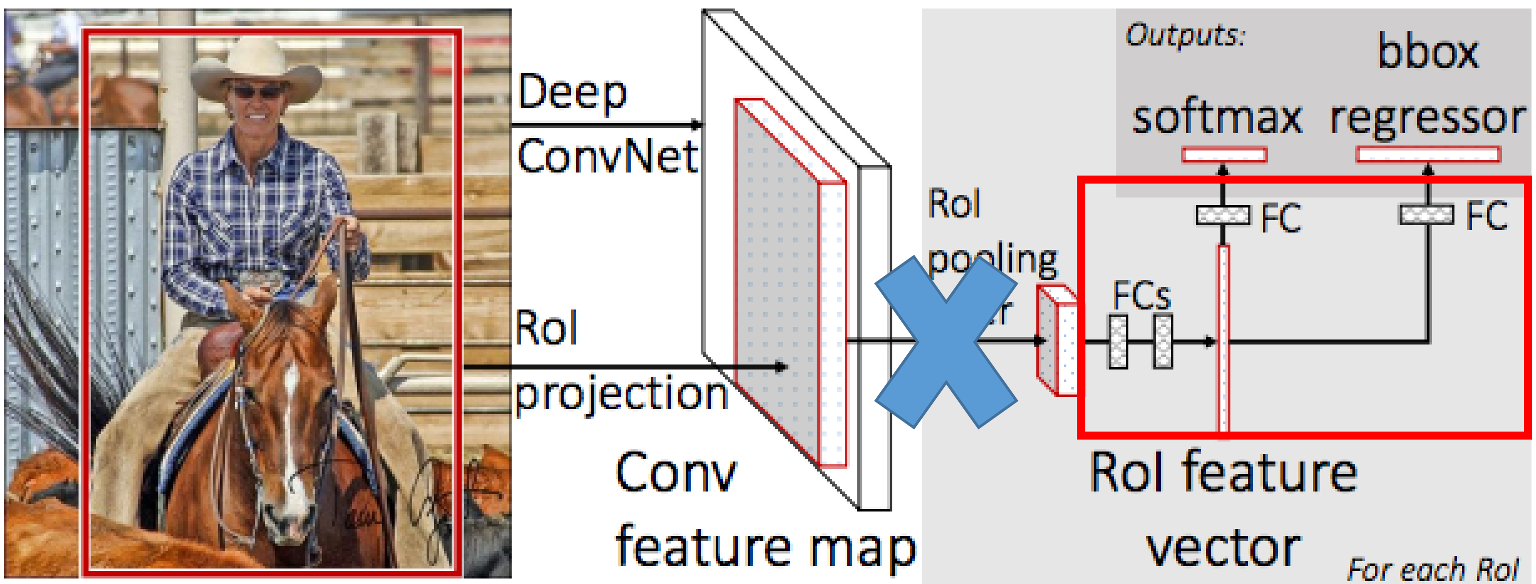
Fast R-CNN (Girshick ICCV 2015)

- How to crop features?
  - Since we have fully-connected layers, the size of feature map for each bounding box should be a fixed number



Fast R-CNN (Girshick ICCV 2015)

- How to crop features?
  - Since we have fully-connected layers, the size of feature map for each bounding box should be a fixed number
  - Resize/Interpolate the feature map as fixed size?
    - Not optimal. This operation is hard to backprop.
      -> we cannot train the conv layers in CNNs…

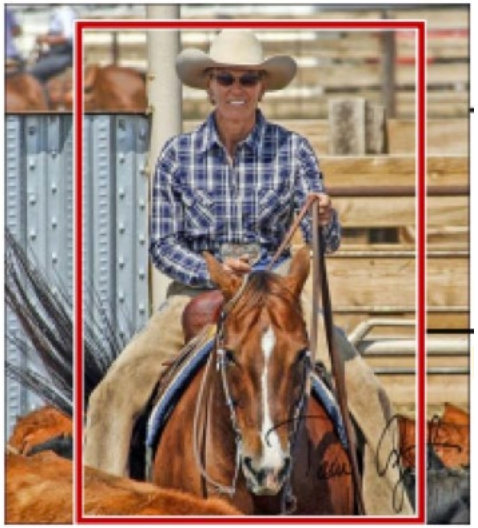


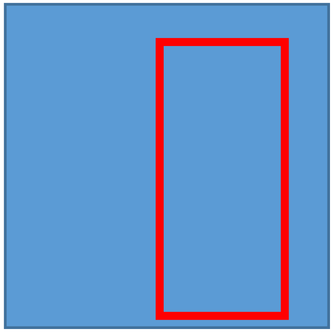
Fast R-CNN (Girshick ICCV 2015)

- How to crop features?
  - Since we have fully-connected layers, the size of feature map for each bounding box should be a fixed number
  - Resize/Interpolate the feature map as fixed size?
    - Not optimal. This operation is hard to backprop.
      -> we cannot train the conv layers for this problem…
  - RoI (Region of Interest) Pooling
    - How?

RoI Pooling

- Step 1:
  Get bounding box for feature map from bounding box for image
  - Due to the (down)convolution/pooling operations,
    feature map would have a smaller size than the original image.





Feature map

RoI Pooling

- Step 2:
  Divide cropped feature map into fixed number of sub-regions
    - The last column and last row might be smaller

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Feature map
4 x 4 x 1

Make it as
2x2 grids

0	1	2
4	5	6
8	9	10

1	2	3
5	6	7
9	10	11
13	14	15

RoI Pooling

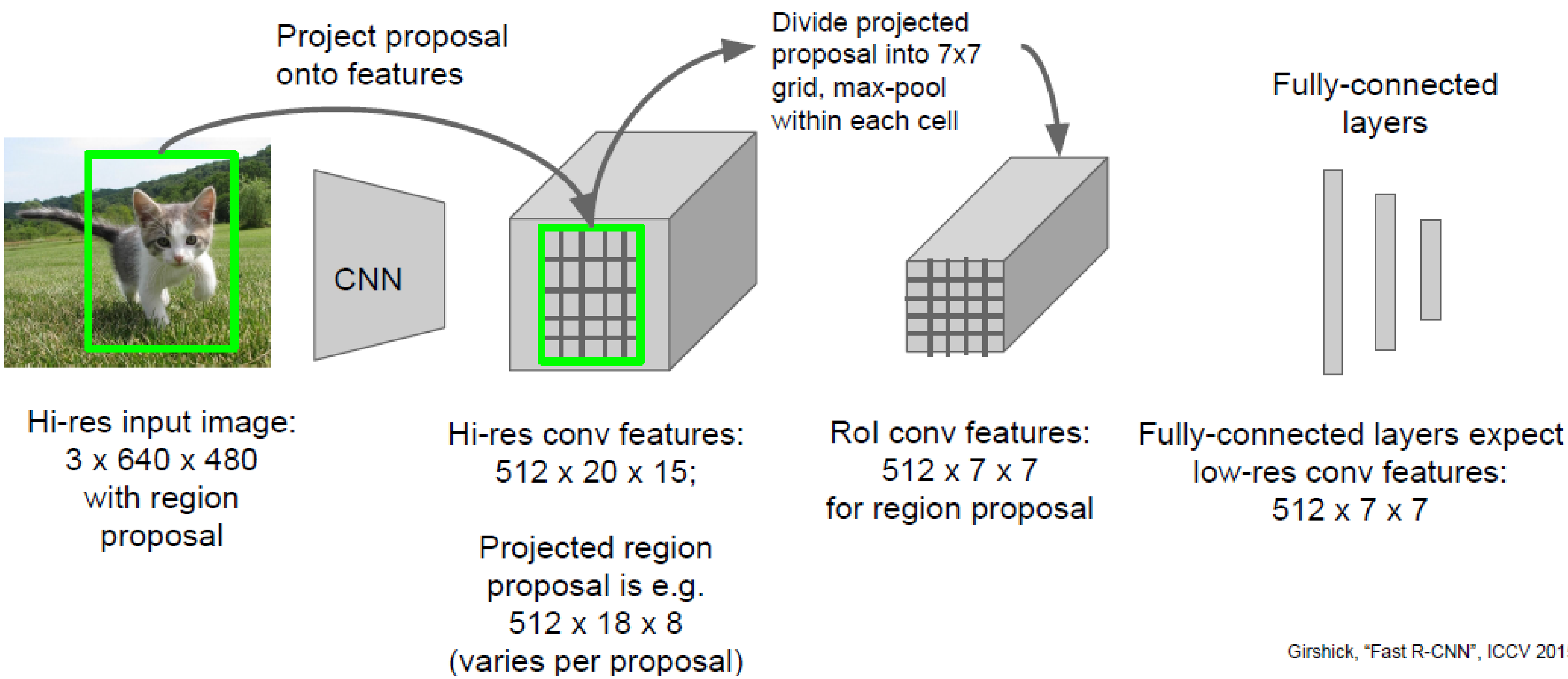
- Step 3:
  For each sub-region, perform max pooling (pick the max one)

0	1	2
4	5	6
8	9	10

Max pooling

5	6
9	10

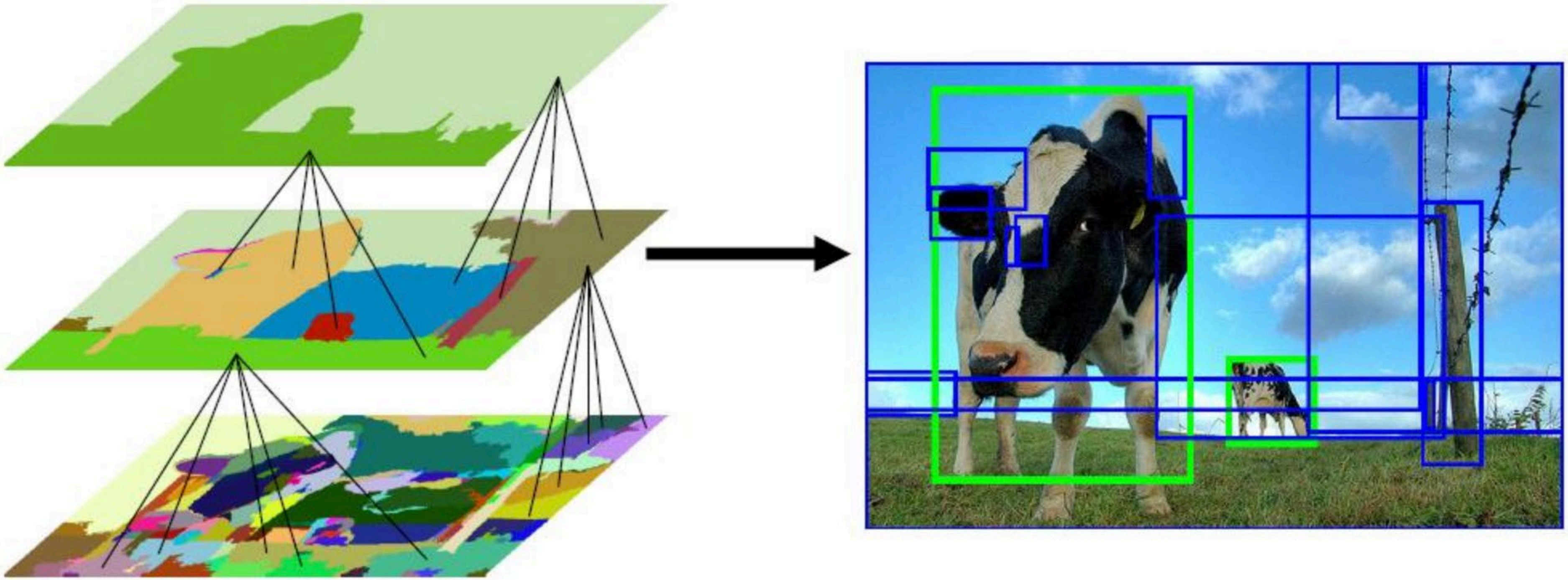
RoI Pooling



Girshick, "Fast R-CNN", ICCV 2015.

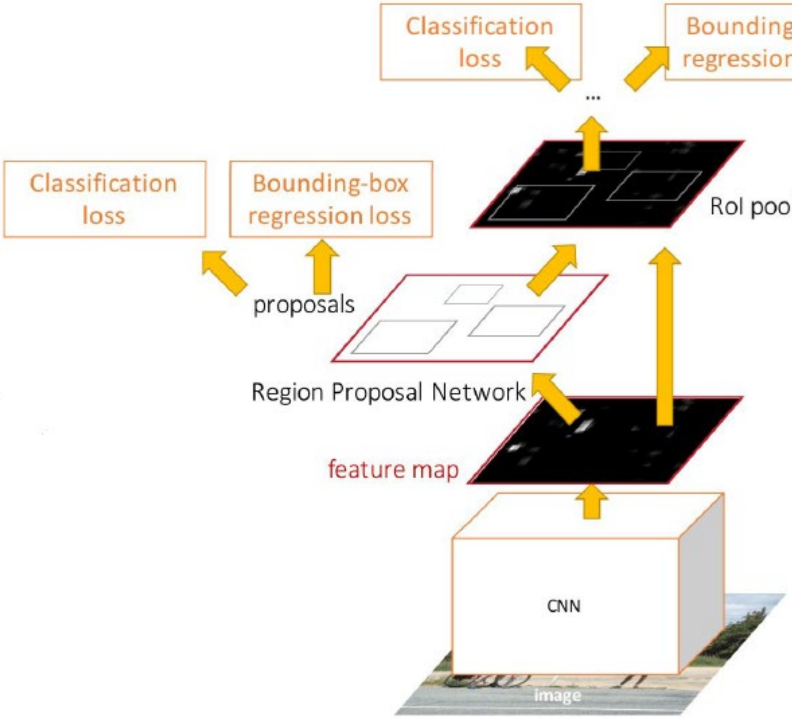
Fast R-CNN (Girshick ICCV 2015)

- What could be the problems?
  - We still need to collect the region proposals from a pre-processing step, which does not allow end-to-end learning.



Faster R-CNN (Ren et al. NIPS 2015)

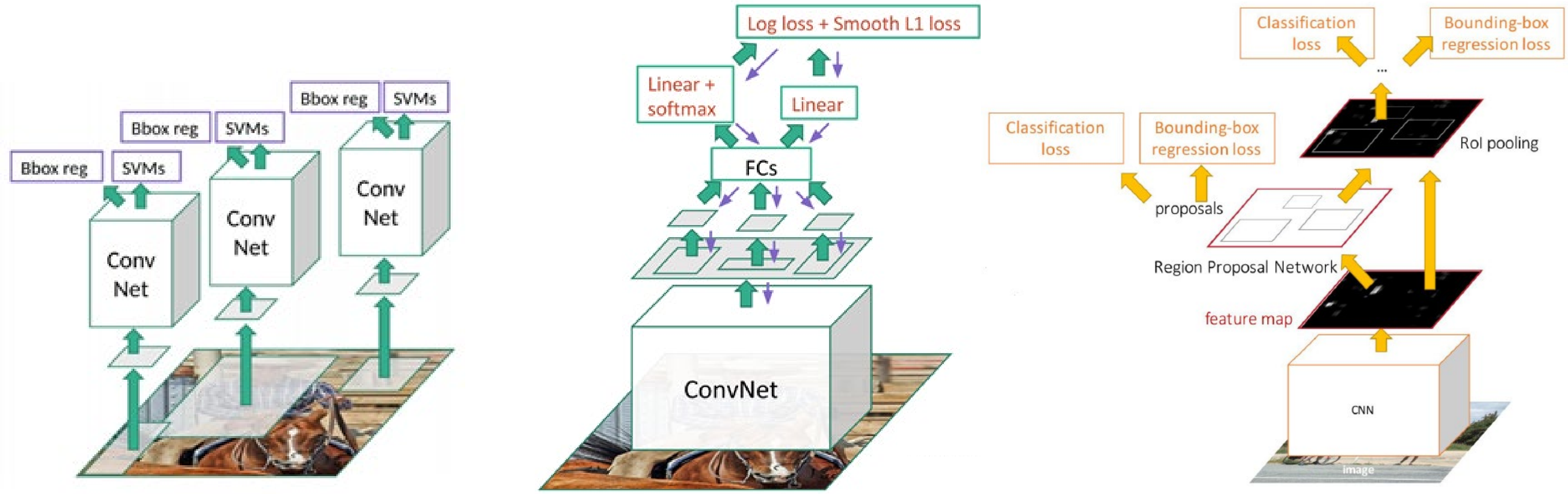
- Solution
  - Why not generate region proposals using CNN?
    -> Insert Region Proposal Network (RPN) to predict proposals from features
  - Jointly train with 4 losses:
    - RPN classification loss
    - RPN regress box coordinates
    - Final classification loss
    - Final box coordinates



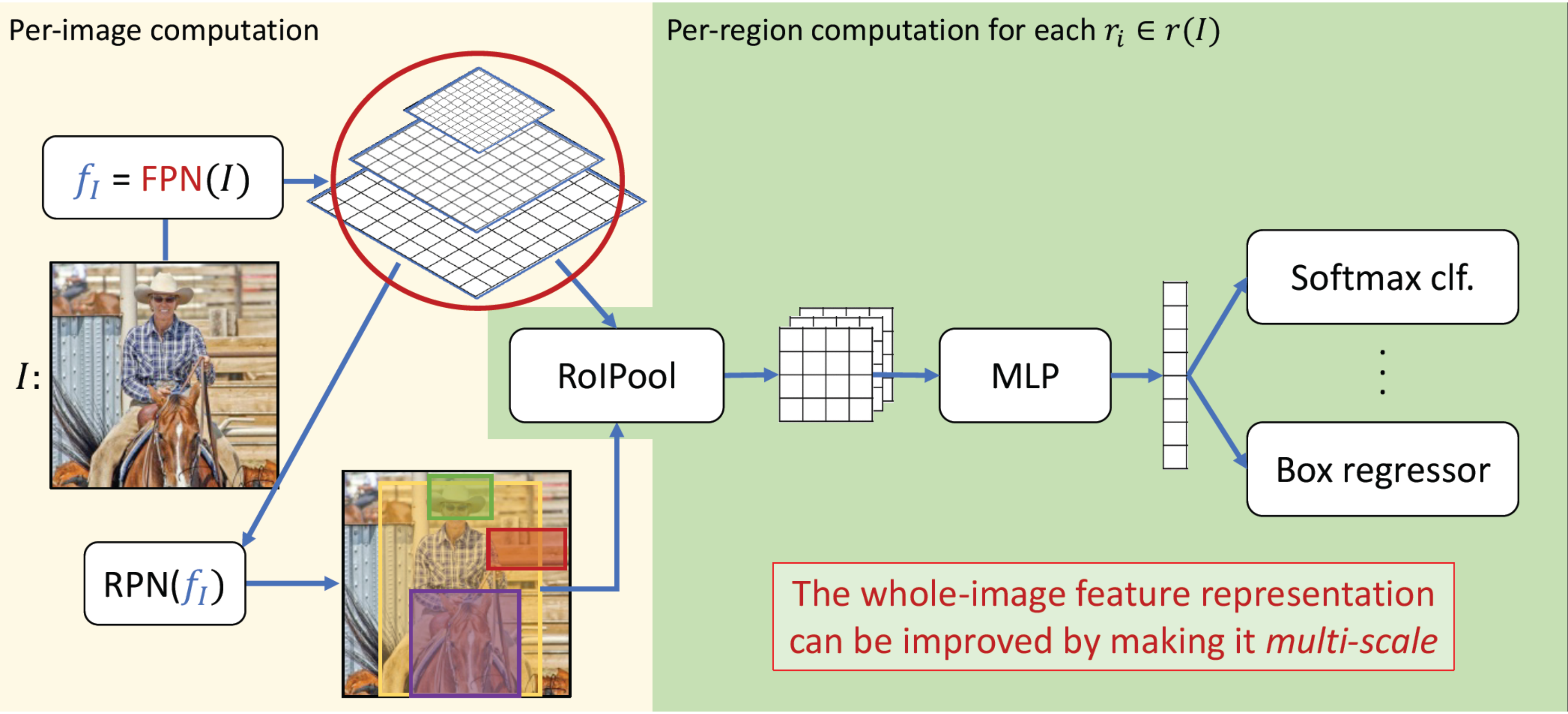
https://arxiv.org/pdf/1506.01497.pdf

Image credit: http://zh.gluon.ai/chapter_computer-vision/object-detection.html

R-CNN, Fast R-CNN, & Faster R-CNN



Faster R-CNN with Feature Pyramid Network

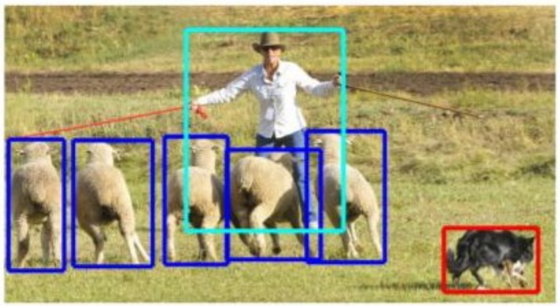


Slide credit: Ross Girshick

Faster R-CNN (Ren et al. NIPS 2015)

- What could be the problems
  - Two-stage detection pipeline is still too slow for real-time detection in videos…
  - What about instance-wise information?

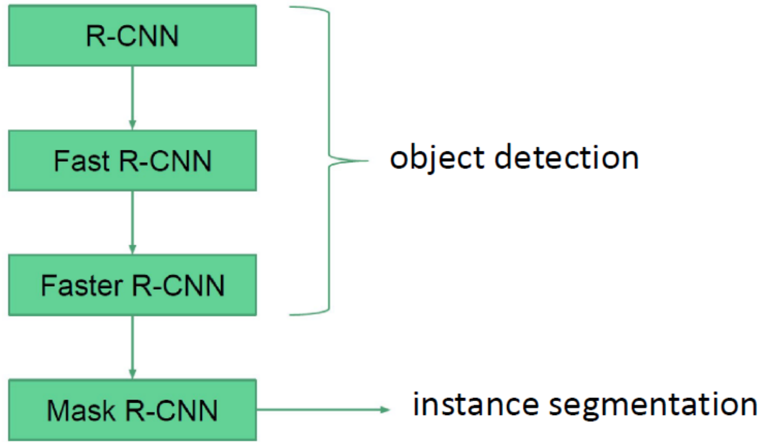
(a) Image classification


(b) Object localization

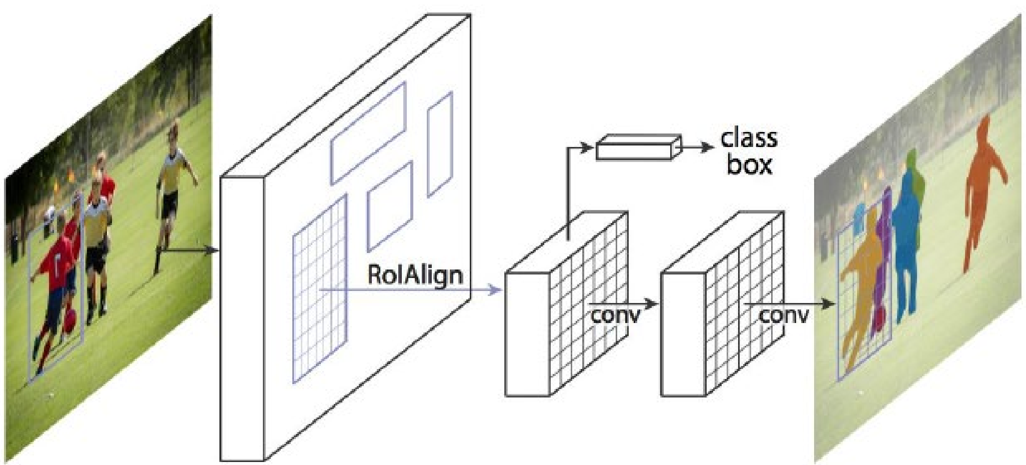

(c) Semantic segmentation


(d) Instance segmentation

Mask R-CNN (ICCV2017)

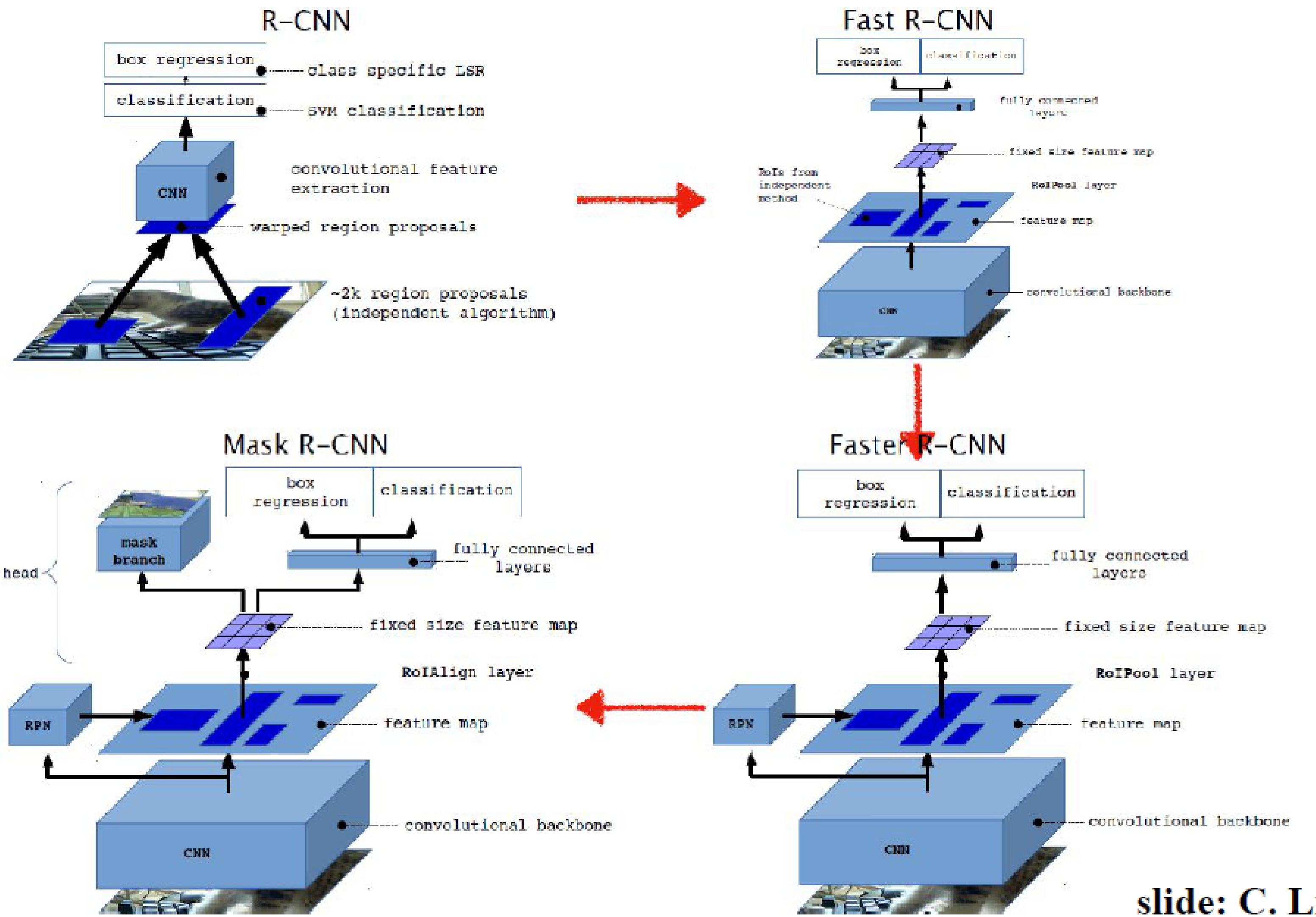


- Goals:
  - Refined detection + precise segmentation
  - Faster R-CNN + FCN
- Overall design:
  - Use of ResNet or feature pyramid net for feature extraction
  - Use RPN to produce proposals (w/ ROI align)
  - Use one detection branch for box classification + regression
  - Use one segmentation branch for box segmentation



He et al, "Mask R-CNN", arXiv 2017

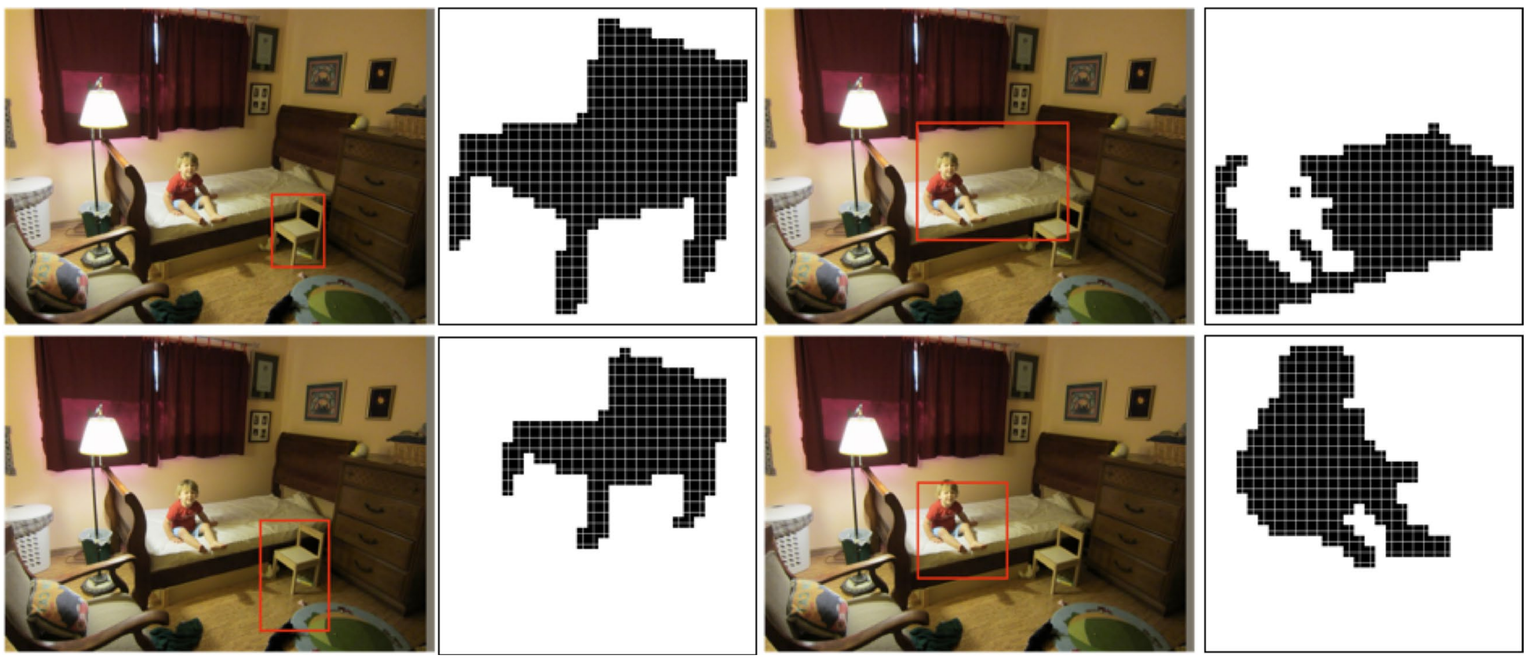
R-CNN Family



slide: C. Lim

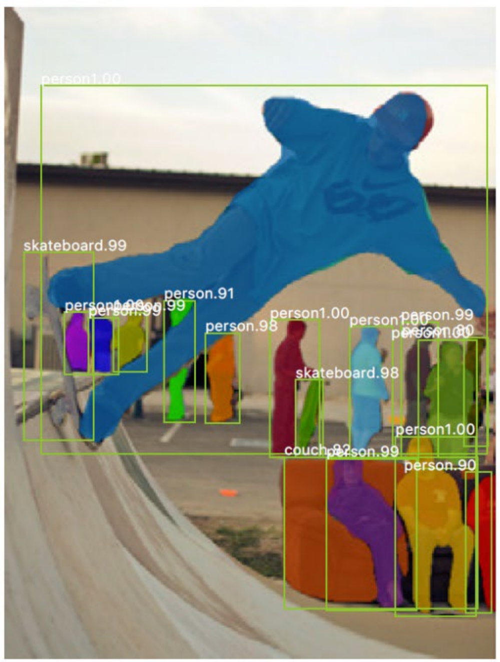
Mask R-CNN (cont'd)

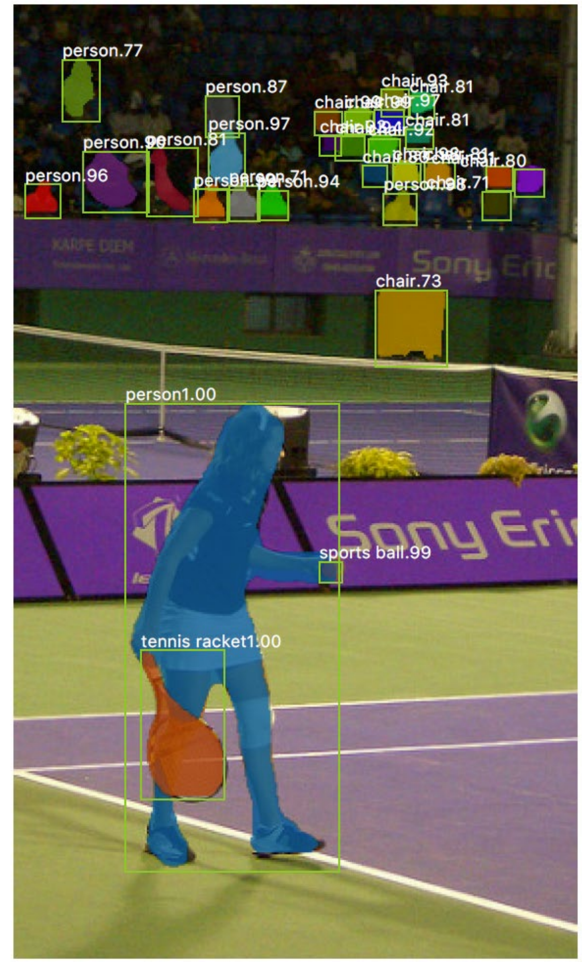
- Example Training Data (requires pixel-level labels)



Mask R-CNN

- Very good results!
    - running at 5fps though





Recap

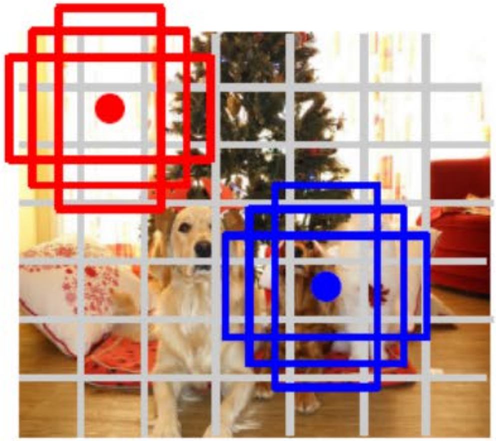
- So far, the introduced methods follow a **two-stage** framework.
    1. Region Proposal
    2. Per-Region Classification/Regression
- Can we make it faster by integrating the above two steps into one single network?

Object Detection Methods
- Sliding Windows
- **Two−stage Frameworks**
    - R−CNN
    - Fast R−CNN
    - Mask R−CNN
    - ⋮
- **One−stage Frameworks**
    - YOLO
    - YOLOv2
    - YOLOv3
    - ⋮

One-Stage Object Detection: Detection without Proposals

Go from input image to tensor of scores with one big convolutional network!

Input image
3 x H x W



Divide image into grid
7 x 7

Image a set of **base boxes** centered at each grid cell
Here B = 3

Within each grid cell:
- Regress from each of the B base boxes to a final box with 5 numbers:
  (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)

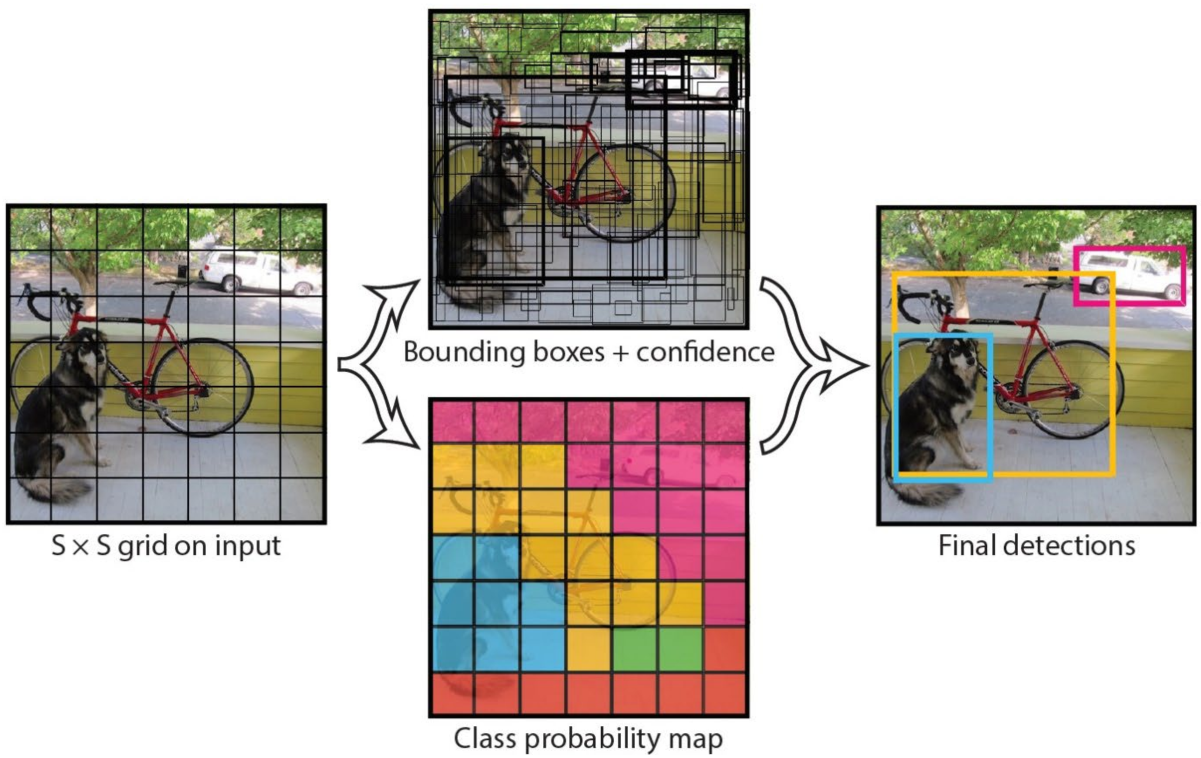
Output:
7 x 7 x (5 * B + C)

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016

You Only Look Once (YOLO)

Divide the image into an $S \times S$ grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities.

These predictions are encoded as an $S \times S \times (B * 5 + C)$ tensor.



You Only Look Once (YOLO)

$\boxed{\textbf{class confidence score} = \text{box confidence score} \times \text{conditional class probability}}$

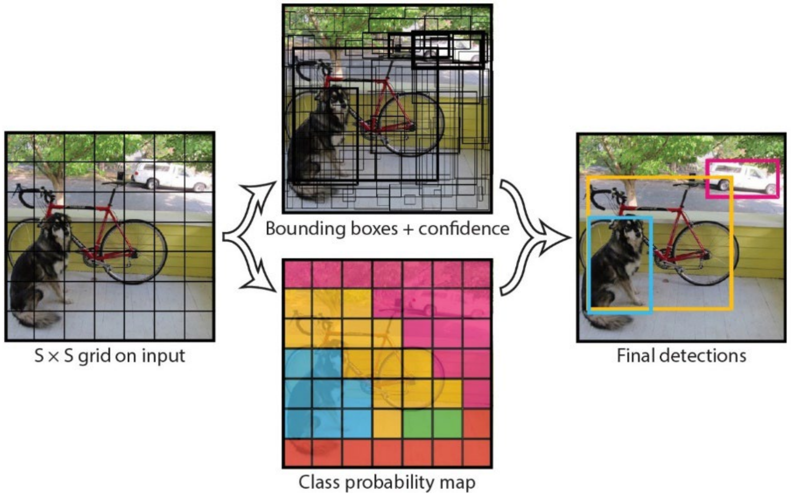
$$\begin{aligned} \text{box confidence score} &\equiv P_r(object) \cdot IoU \\ \text{conditional class probability} &\equiv P_r(class_i | object) \\ \text{class confidence score} &\equiv P_r(class_i) \cdot IoU \\ &= \text{box confidence score} \times \text{conditional class probability} \end{aligned}$$

where

$P_r(object)$ is the probability the box contains an object.
IoU is the IoU (intersection over union) between the predicted box and the ground truth.
$P_r(class_i | object)$ is the probability the object belongs to $class_i$ given an object is presence.
$P_r(class_i)$ is the probability the object belongs to $class_i$

You Only Look Once (YOLO)

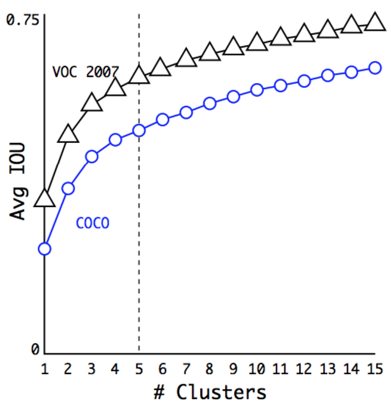
- **Fast**.
  Good for real-time processing.
- **End-to-end learnable**.
  Predictions (object locations and classes) are made from one **single network**.
- **Access to the entire image**.
  Region proposal methods limit the classifier to the specific region.
  YOLO **accesses to the whole image** in predicting boundaries.
  With additional context, result in fewer false positives in background areas.
- **Spatial diversity**.
  Detect one object per grid cell. It enforces spatial diversity in making predictions.

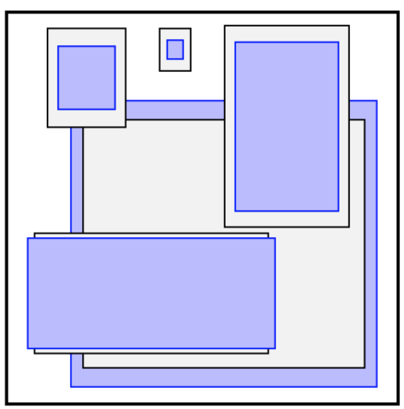


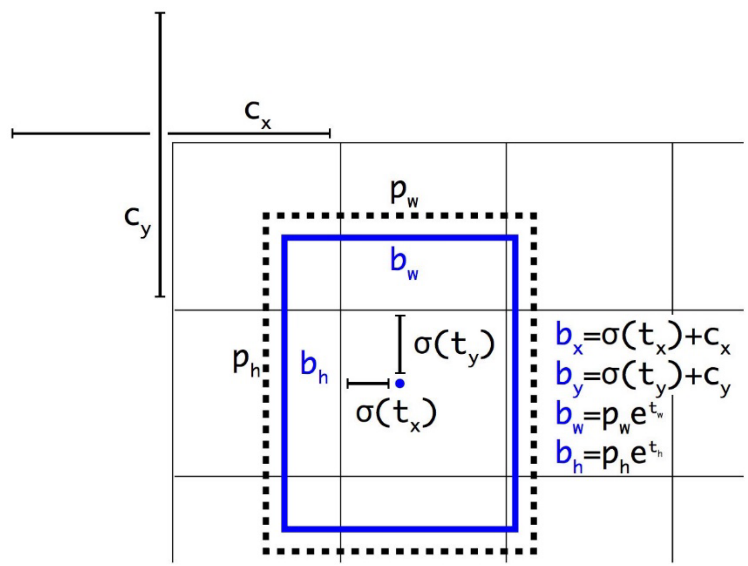
YOLOv2

- **Predetermined bounding box shape** (*anchor boxes*)
  Guesses that are common for real-life objects using *k*-means clustering
  $\Longrightarrow$ Predicts **offsets** rather than bounding boxes themselves
- **Move the class prediction from the cell level to the boundary box level**
  Each *bounding box* (instead of each *cell*) produces a class prediction

$$S \times S \times (B * 5 + C) \Longrightarrow S \times S \times \color{red}{(B * (5 + C))}$$



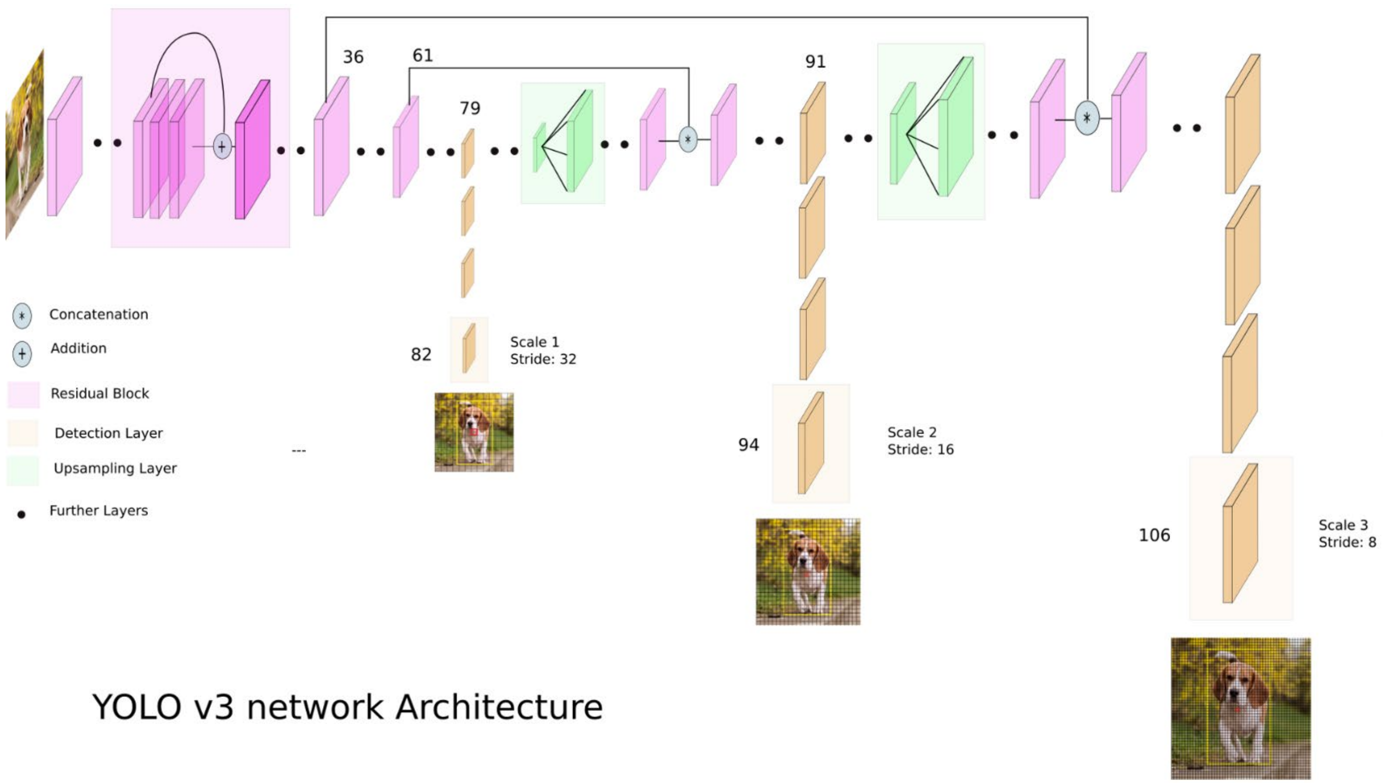




YOLOv3

- **Feature Pyramid Networks** (FPN) **like Feature Pyramid**
  YOLOv3 makes predictions at 3 different scales (similar to the FPN).

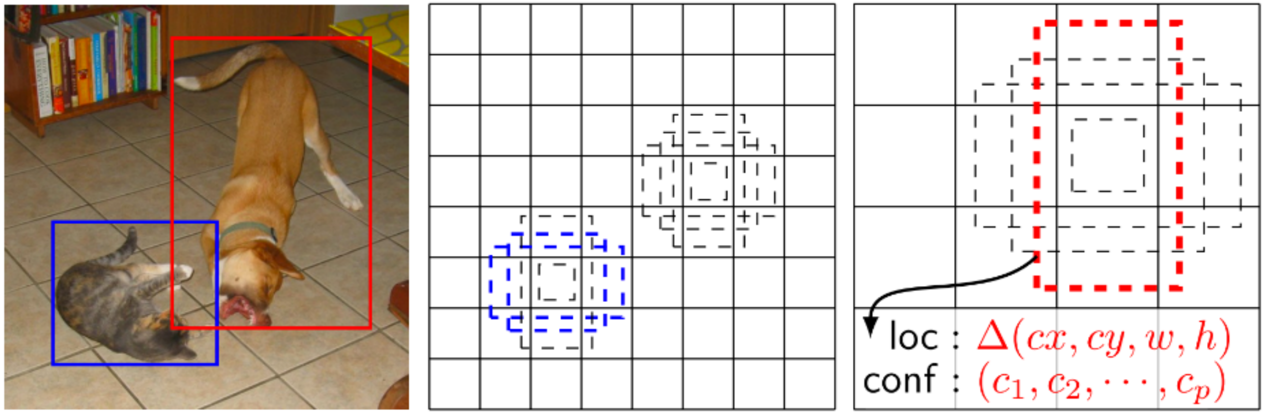
$$S \times S \times (\textcolor{red}{3} * (5 + C))$$



YOLO v3 network Architecture

Single Shot MultiBox Detector (SSD)

Propose multiple default boxes per grid at different scales



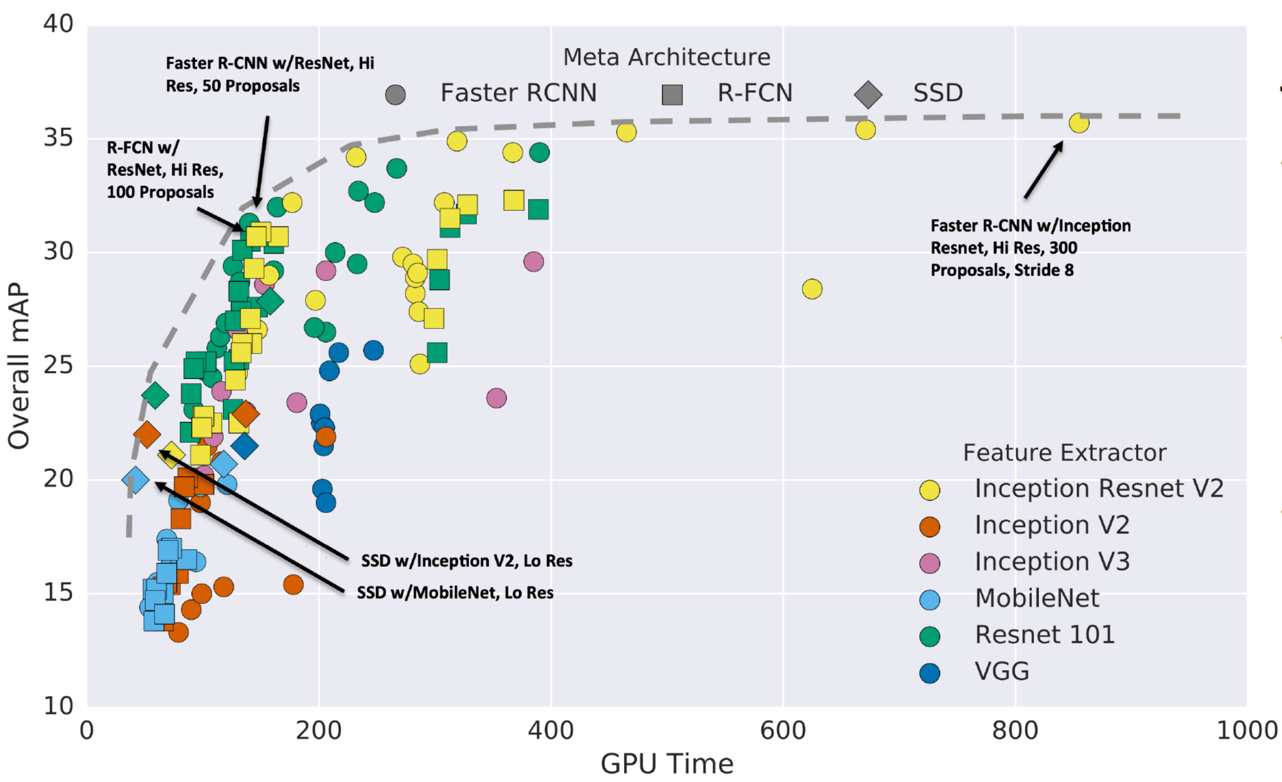
(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Recap

Object Detection Methods

- Sliding Windows
- **Two–stage Frameworks**
  (High Accuracy, Slow)
  - R–CNN
  - Fast R–CNN
  - Mask R–CNN
  - ⋮
- **One–stage Frameworks**
  (Good Accuracy, Very Fast)
  - YOLO
  - YOLOv2
  - YOLOv3
  - ⋮

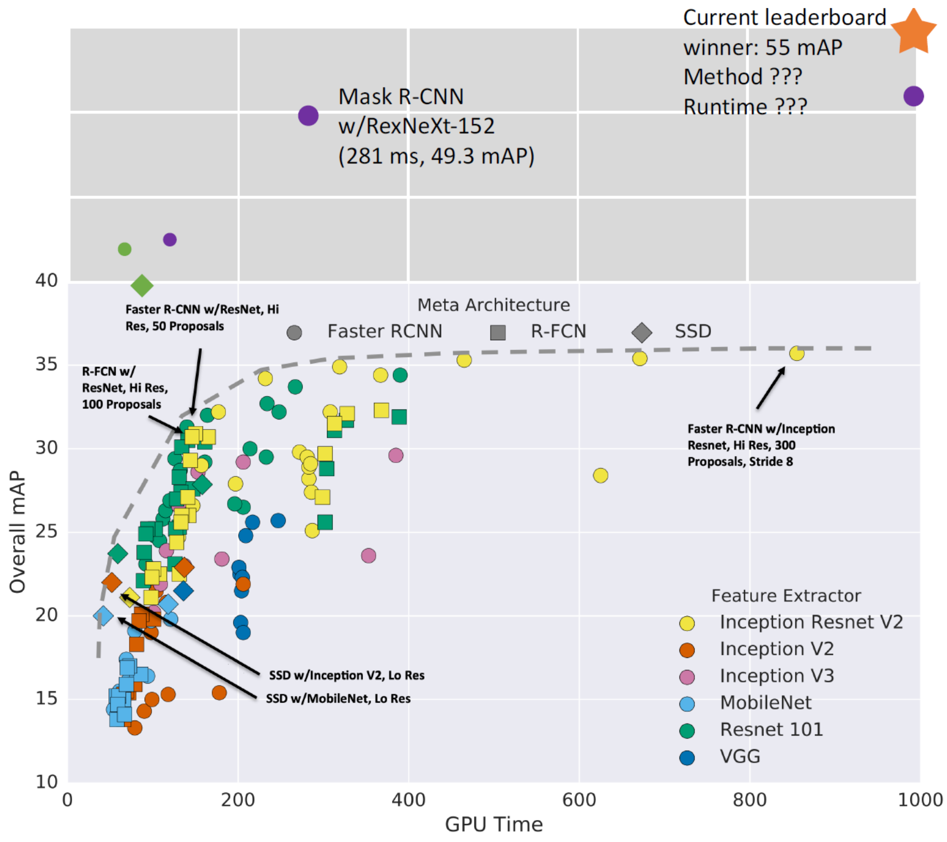
Remarks



Takeaways:

- Two stage method (Faster R-CNN) get the best accuracy, but are slower
- Single-stage methods (SSD) are much faster, but don't perform as well
- Bigger backbones improve performance, but are slower

Remarks (cont'd)

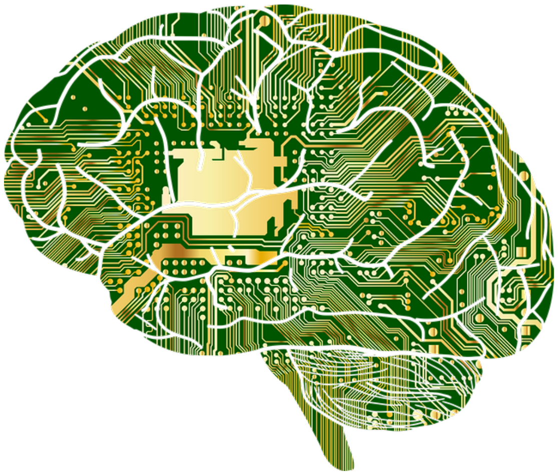


These results are a few years old … since then GPUs have gotten faster, and we've improved performance with many tricks:

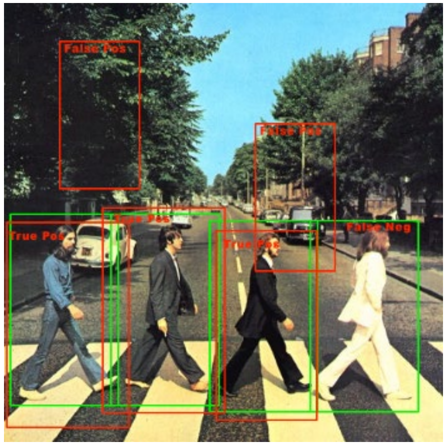
- Train longer!
- Multiscale backbone: Feature Pyramid Networks
- Better backbone: ResNeXt
- Single-Stage methods have improved
- Very big models work better
- Test-time augmentation pushes numbers up
- Big ensembles, more data, etc

Slide credit: UMich EECS 498-007

What's to Be Covered Today…

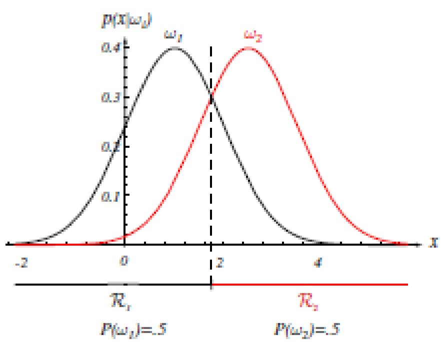


- Segmentation
- Object Detection
- Generative Model
  - Autoencoder (AE)
  - Variational Autoencoder (VAE) (next week)

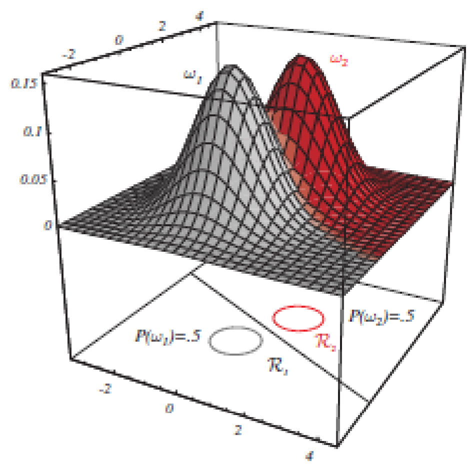


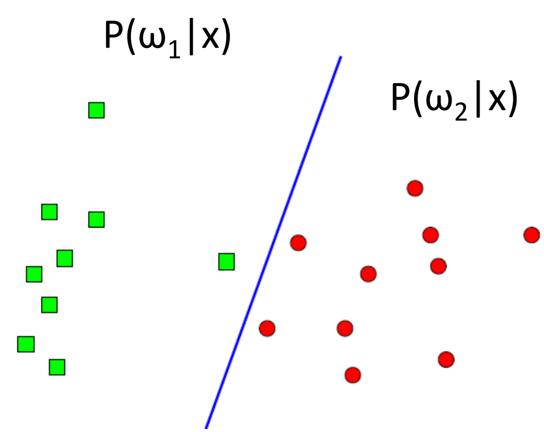
Discriminative vs. Generative Models

- Discriminative Models
  - Model posteriors P(ω|x) from likelihoods P(x|ω)
    where x is the input data, and ω indicates the class of interest
  - Example (posterior)



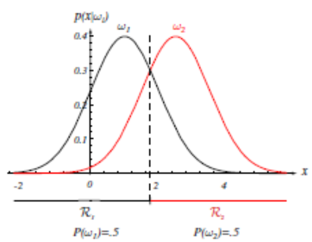
or

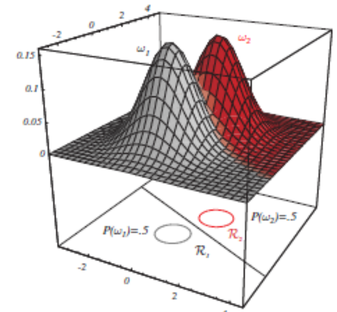




Discriminative vs. Generative Models (cont'd)

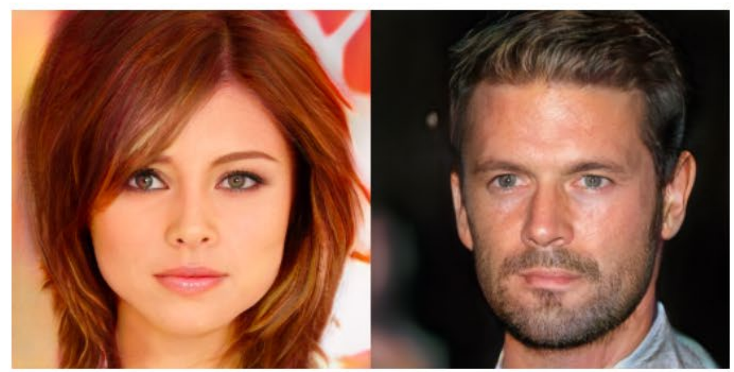
- Generative Models
  - Model likelihoods P(x|ω) with priors P(ω) (i.e., modeling P(x|ω) P(ω)) where x is the input data, and ω indicates the class of interest





  - Example

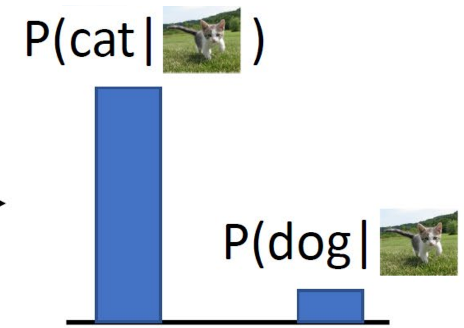
Training Data
(CelebA)



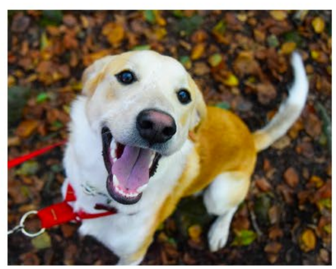
Sample Generator
(Karras et al, 2017)

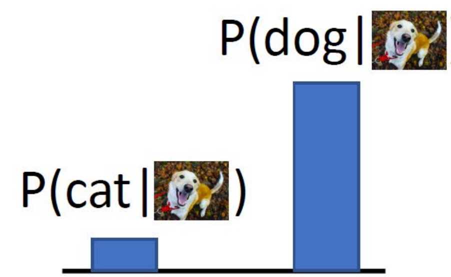
Discriminative vs. Generative Models (cont'd)

Discriminative Model: Learn a probability distribution p(y|x)



Generative Model: Learn a probability distribution p(x)





Conditional Generative Model: Learn p(x|y)

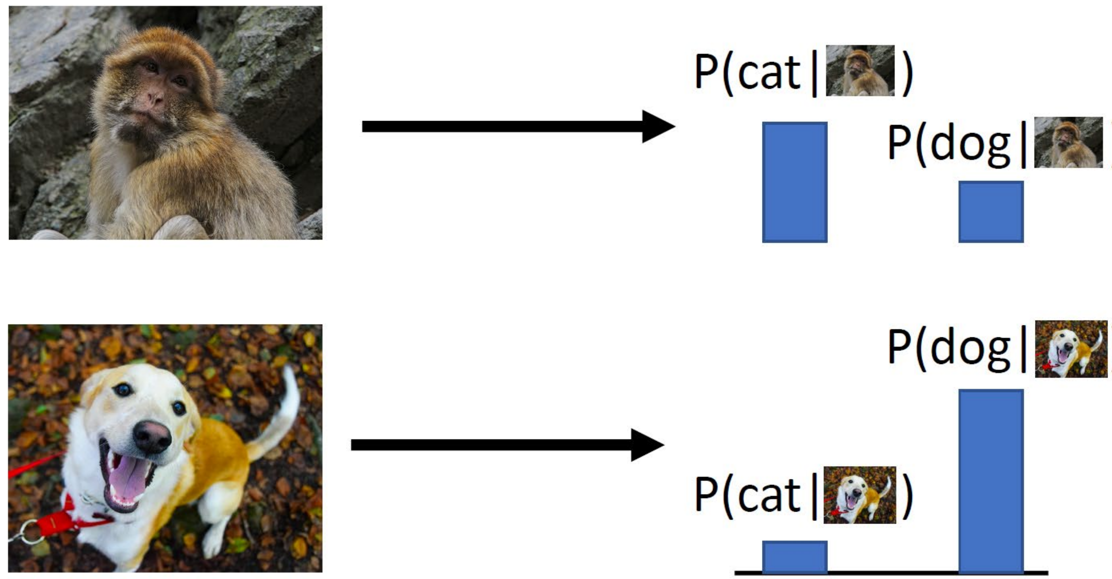
Discriminative model: the possible labels for each input "compete" for probability mass. But no competition between **images**

Discriminative vs. Generative Models (cont'd)

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



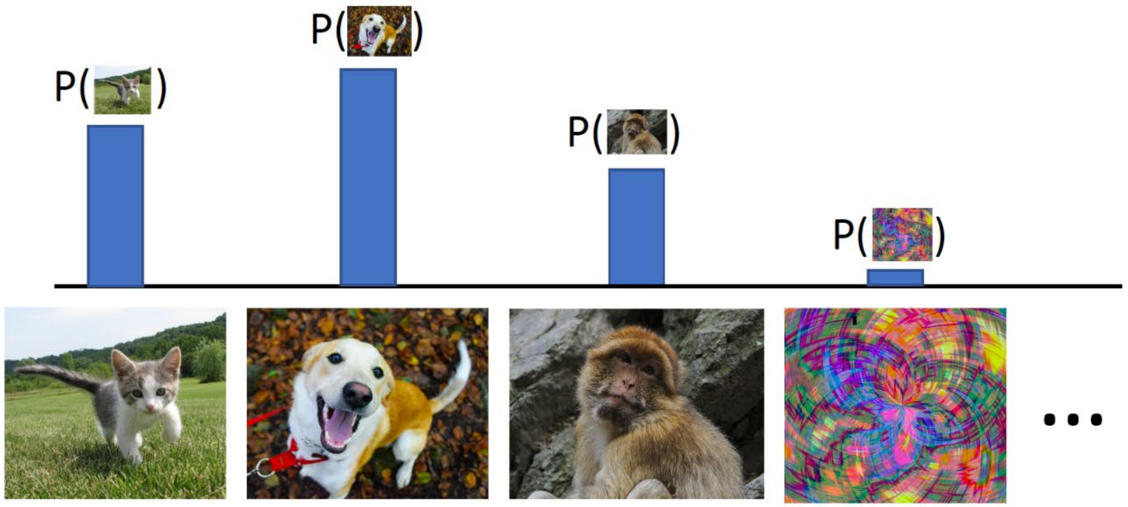
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Discriminative vs. Generative Models (cont'd)

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Generative model: All possible images compete with each other for probability mass

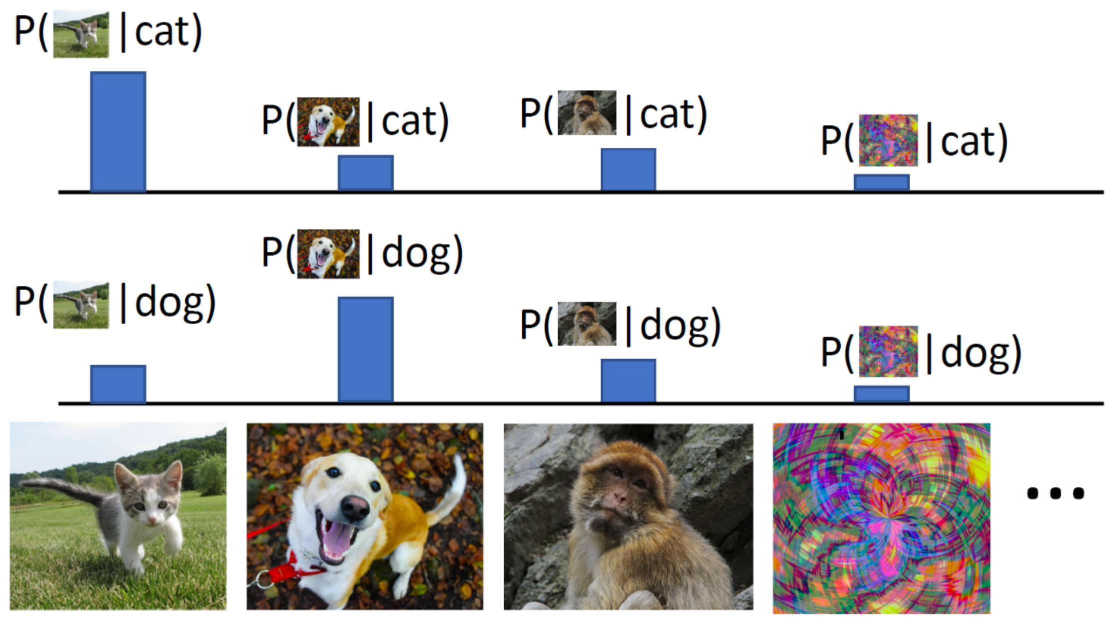
Model can "reject" unreasonable inputs by assigning them small values

Discriminative vs. Generative Models (cont'd)

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Conditional Generative Model: Each possible label induces a competition among all images

Discriminative vs. Generative Models (cont'd)

Discriminative Model:
Learn a probability distribution p(y|x)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**

$$\underbrace{\boxed{P(x \mid y)}}_{\text{Conditional Generative Model}} = \frac{\overbrace{\boxed{P(y \mid x)}}^{\text{Discriminative Model}}}{\underbrace{\boxed{P(y)}}_{\text{Prior over labels}}} \overbrace{\boxed{P(x)}}^{\text{(Unconditional) Generative Model}}$$

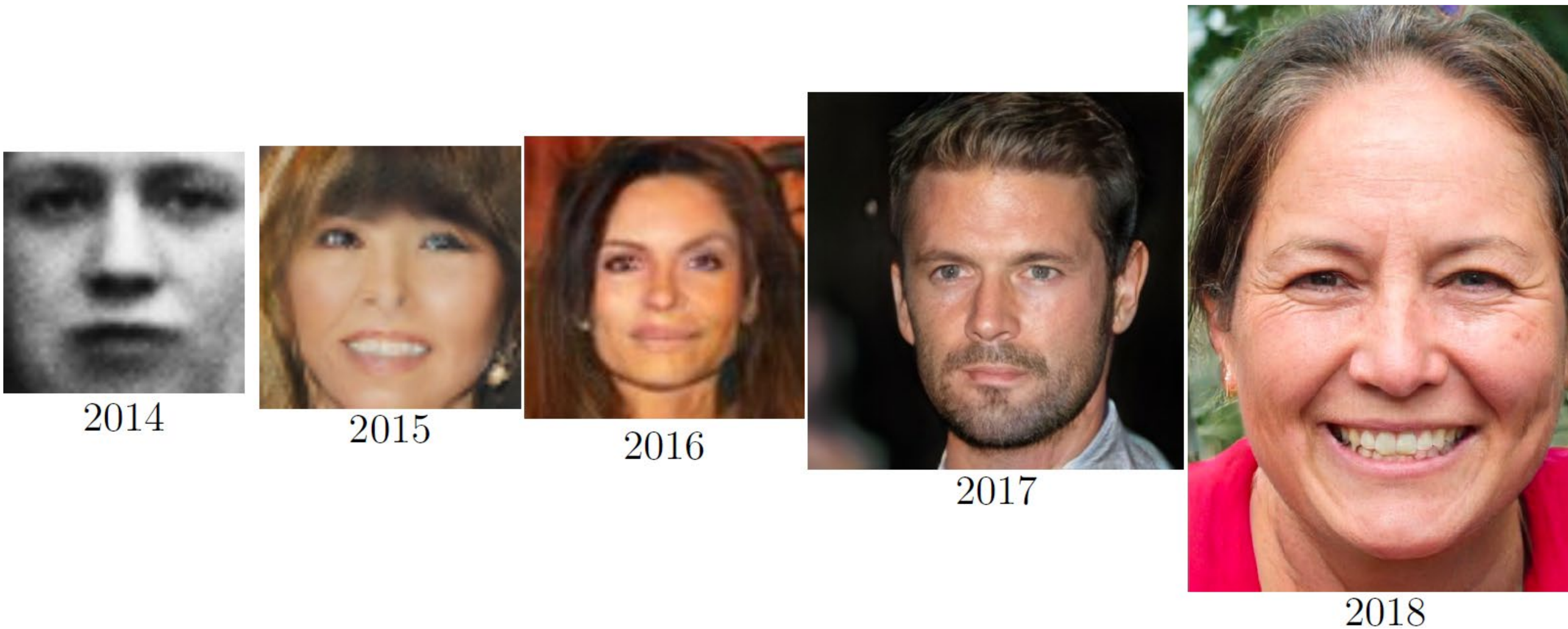
We can build a conditional generative model from other components!

Additional Remarks

- Discriminative Models
  - Learn a (posterior)probability distribution $p(y|x)$
  - Assign labels to each instance x
  - Supervised learning
- Generative Models
  - Learn a probability distribution $p(x)$
  - Data representation, detect outliers, etc.
  - Unsupervised learning

What Have Been Done
Using Deep Generative Models?

- 5+ years of progress on synthesizing face images



Slide credit: I. Goodfellow

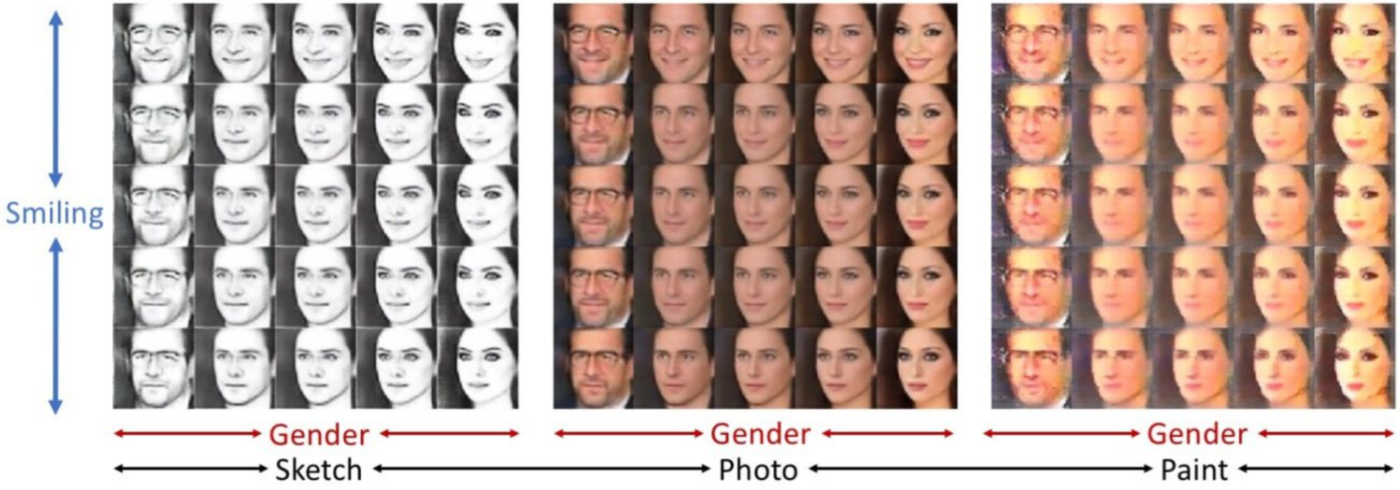
What Have Been Done Using Deep Generative Models?

- 2 years of progress on synthesizing images (ImageNet)

Slide credit: I. Goodfellow

Why We Need Generative Models?

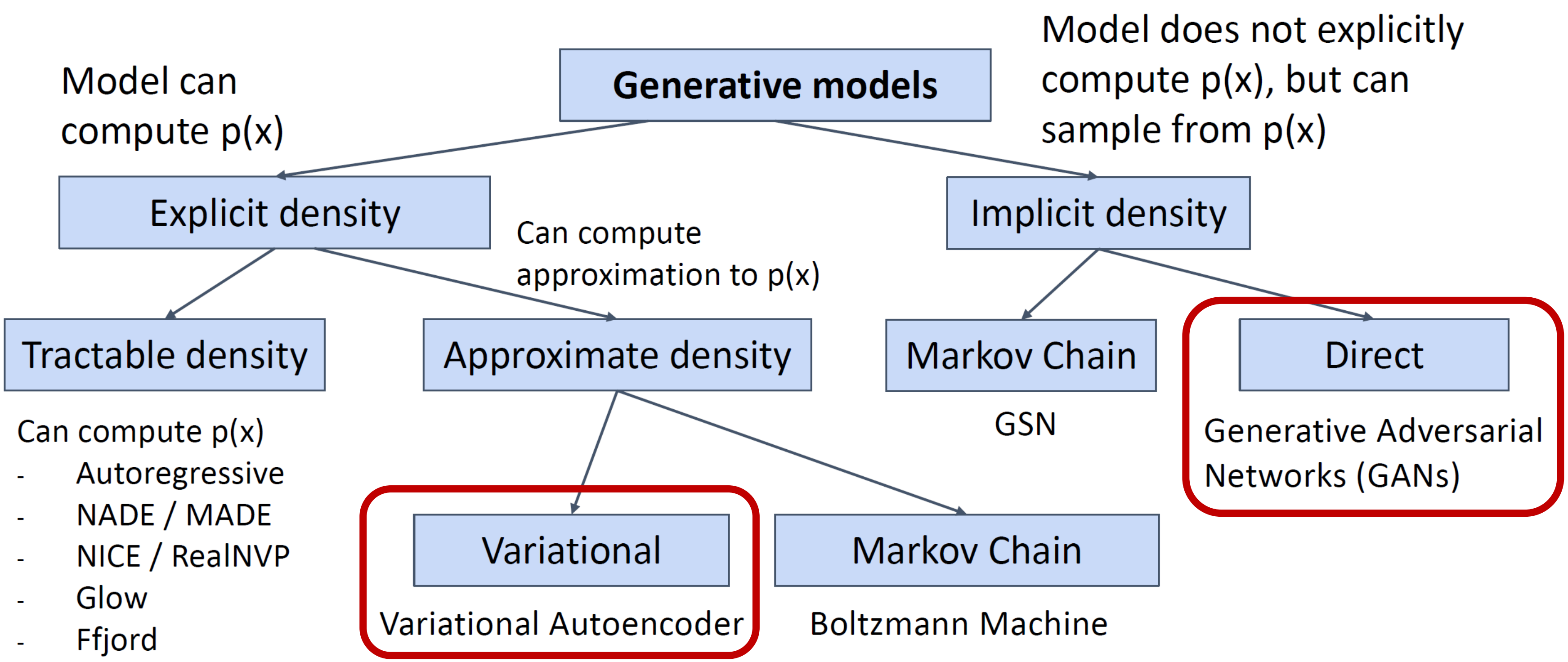
- Remarks
  - Able to process data information (e.g., priors like attribute, category, etc.) for synthesis, prediction, or recognition purposes
    - For example, with latent feature z derived from x, one may have P(z) may describe image variants.
    - Or, z in P(z) may annotate object categorical or attribute information.



  - We will talk about a variety of visual applications based on generative models

Liu et al., NeurIPS 2018

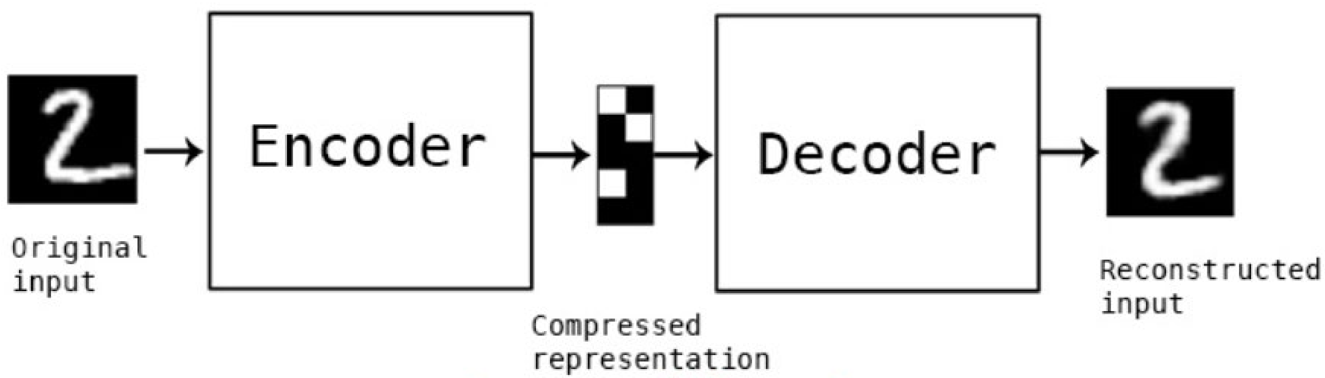
Taxonomy of Generative Models



Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017

Take a Deep Look to Discover Latent Variables/Representations

- Autoencoder
  - Autoencoding = encoding itself with recovery purposes
  - In other words, encode/decode data with reconstruction guarantees
  - Latent variables/features as deep representations
  - Example objective/loss function at output:
    - L2 norm between input and output, i.e.,

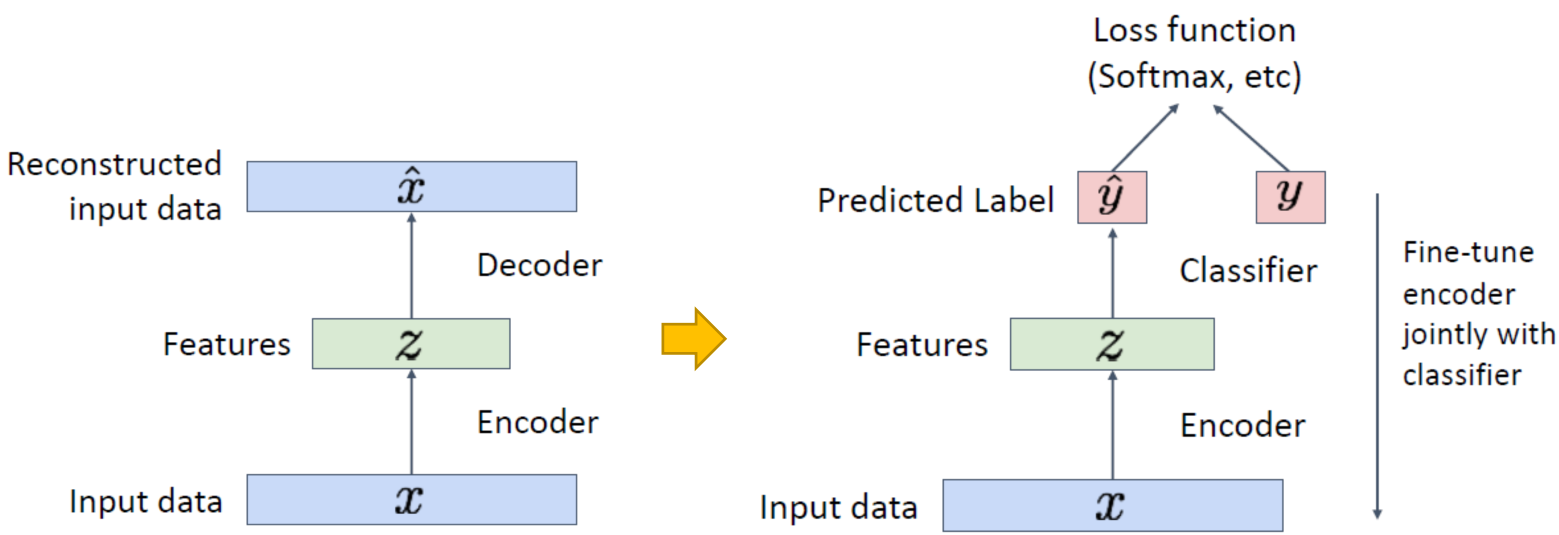


most important information of input image ☞ latent space

Slide credit: W. Chiu

Take a Deep Look to Discover Latent Variables/Representations (cont'd)

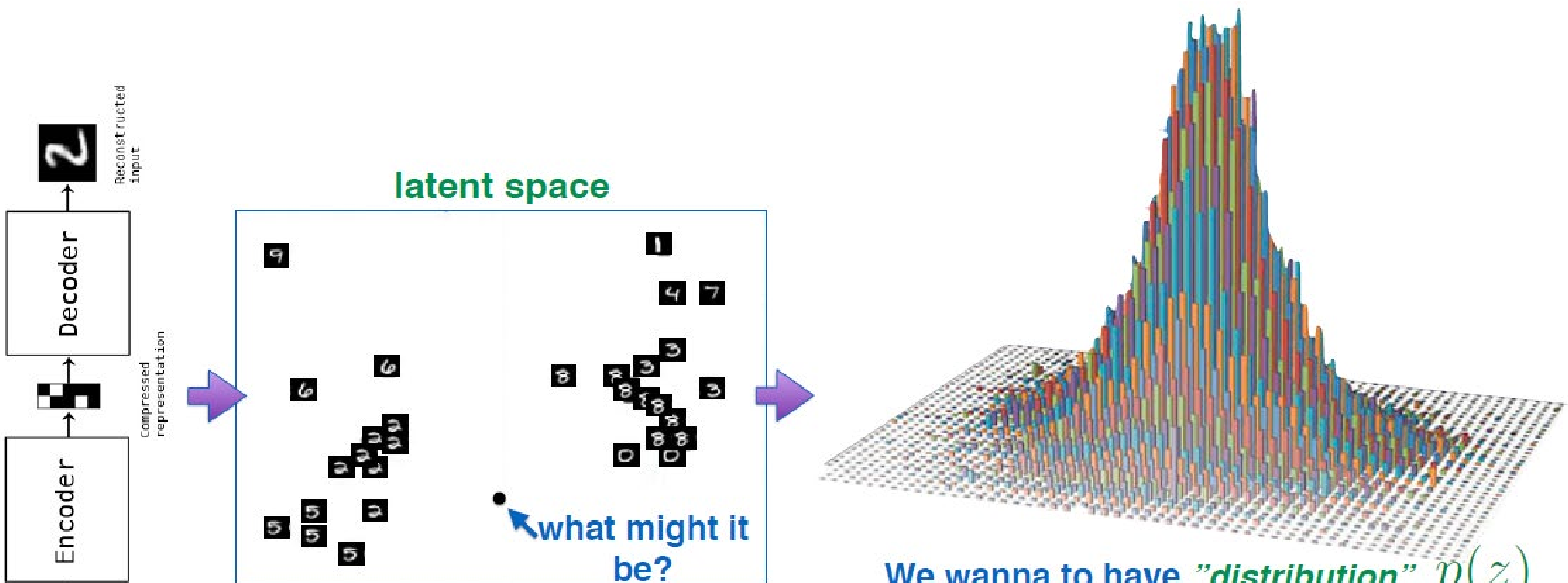
- Autoencoder (AE) for downstream tasks
  - Train AE with reconstruction guarantees
  - Keep encoder (and the derived features) for downstream tasks (e.g., classification)
  - Thus, a trained encoder can be applied to initialize a supervised model



Slide credit: UMich EECS 498-007

Take a Deep Look to Discover Latent Variables/Representations (cont'd)

- What's the Limitation of Autoencoder?

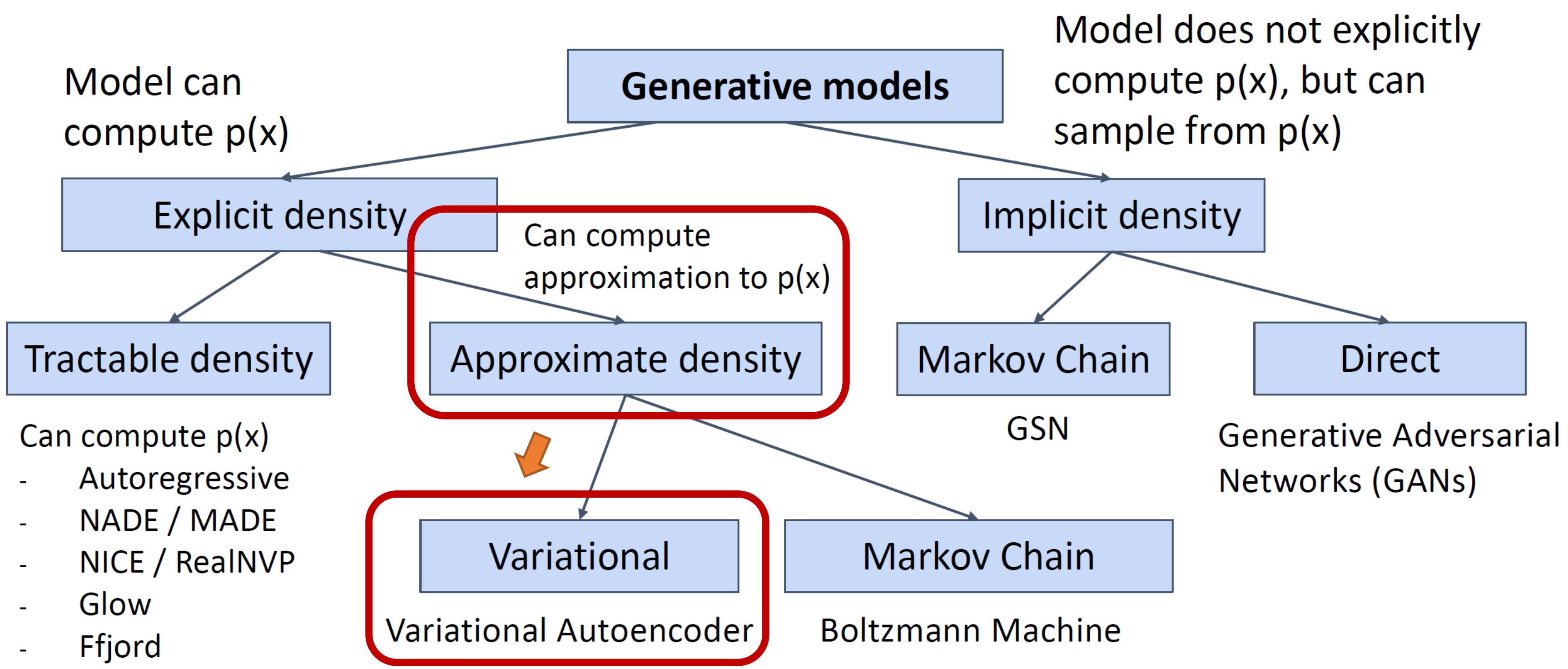


Only samples, how about other regions not covered?

We wanna to have *"distribution"* $p(z)$ where we can sample from any location

Slide credit: W. Chiu

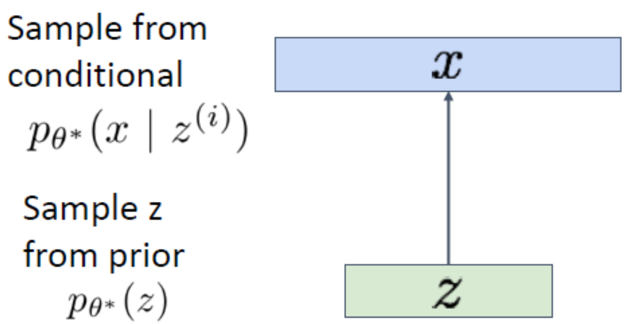
Taxonomy of Generative Models



Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017

Variational Autoencoder

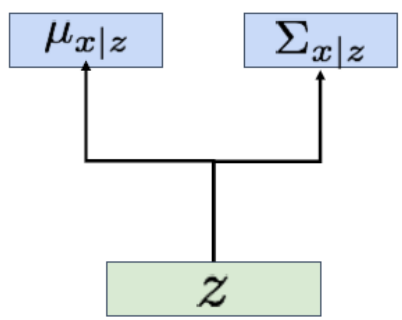
- Probabilistic Spin on AE
  - Learn latent feature z from raw data x
  - Sample from the latent space (via model) to generate data



Assume simple prior p(z), e.g. Gaussian

Represent p(x|z) with a neural network (Similar to **decoder** from autencoder)

  - p(x|z) is implemented via a (probabilistic) decoder



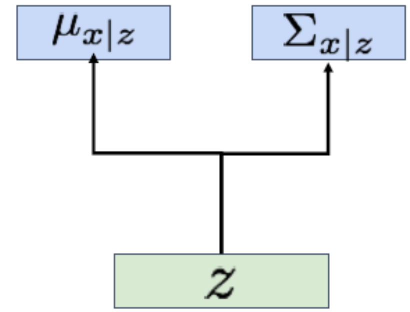
Decoder inputs z, outputs mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Sample x from Gaussian with mean $\mu_{x|z}$ and (diagonal) covariance $\Sigma_{x|z}$

Variational Autoencoder (cont'd)

- Remarks
  - Train VAE via maximum likelihood of data
  - Note that we don't observe z & need to marginalize it:

$$p_\theta(x) = \int p_\theta(x, z)dz = \int p_\theta(x|z)p_\theta(z)dz$$



  - We can compute with the decoder module, and we assume Gaussian prior for z, i.e.,
  - However, can't integrate over all possible z!
  - Recall that we have Bayes' rule:

$$p_\theta(x) = \frac{p_\theta(x \mid z)p_\theta(z)}{p_\theta(z \mid x)}$$

We can't compute $p_\theta(z \mid x)$, but we can train the encoder module to learn

$$q_\phi(z \mid x) \approx p_\theta(z \mid x)$$

Variational Autoencoder (cont'd)

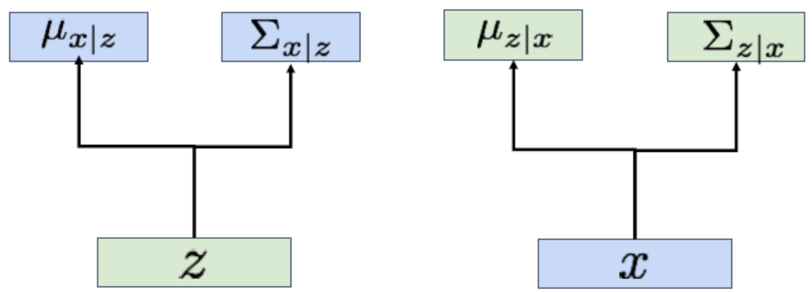
- Now we have…

Decoder network inputs latent code z, gives distribution over data x

$$p_\theta(x \mid z) = N(\mu_{x|z}, \Sigma_{x|z})$$

Encoder network inputs data x, gives distribution over latent codes z

$$q_\phi(z \mid x) = N(\mu_{z|x}, \Sigma_{z|x})$$

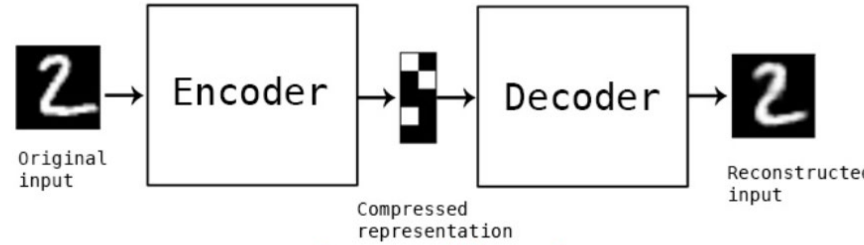


- If we ensure $q_\phi(z \mid x) \approx p_\theta(z \mid x)$ then we have

$$p_\theta(x) \approx \frac{p_\theta(x \mid z)p(z)}{q_\phi(z \mid x)}$$

Slide credit: UMich EECS 498-007

Training VAE



$$\log p_\theta(x) = \log \frac{p_\theta(x \mid z)p(z)}{p_\theta(z \mid x)} = \log \frac{\boxed{p_\theta(x|z)}\boxed{p(z)}\boxed{q_\phi(z|x)}}{\boxed{p_\theta(z|x)}\boxed{q_\phi(z|x)}}$$

$$= E_z[\log \boxed{p_\theta(x|z)}] - E_z\left[\log \frac{\boxed{q_\phi(z|x)}}{\boxed{p(z)}}\right] + E_z\left[\log \frac{\boxed{q_\phi(z|x)}}{\boxed{p_\theta(z|x)}}\right]$$

$$= E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right) + D_{KL}(q_\phi(z|x), p_\theta(z|x))$$

Data reconstruction

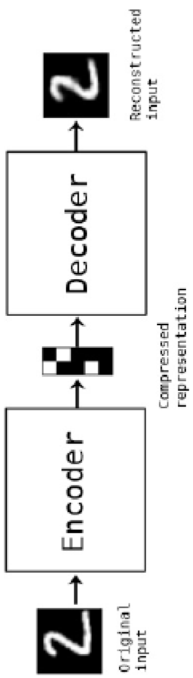
KL divergence between sample distribution from the encoder and the prior

KL divergence between sample distribution from the encoder and the posterior of data

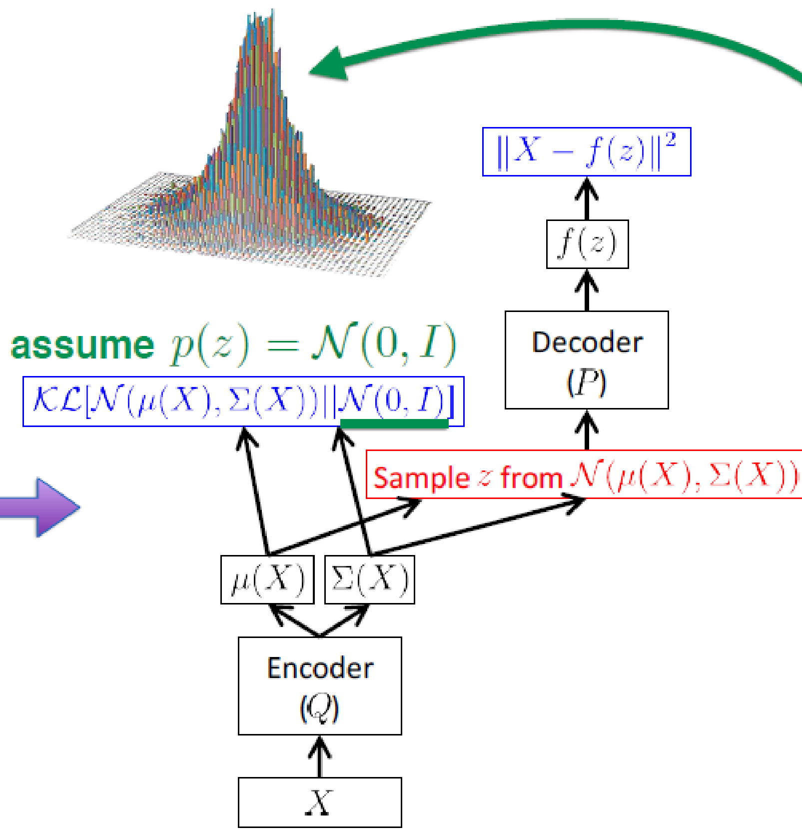
$$\log p_\theta(x) \geq E_{z \sim q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}\left(q_\phi(z|x), p(z)\right)$$

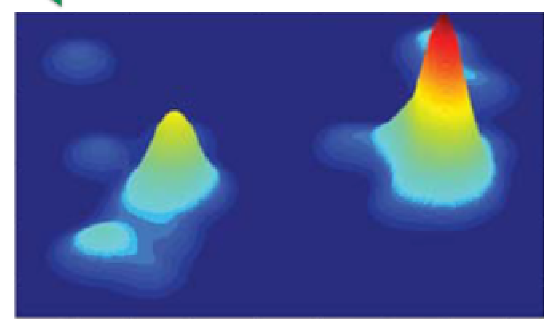
i.e., variational lower bound on the data likelihood $p_\Theta(x)$

Summary:
From Autoencoder to Variational Autoencoder



Now is a "*distribution*", we can assume it to be
a distribution easy to sample from, e.g. Gaussian



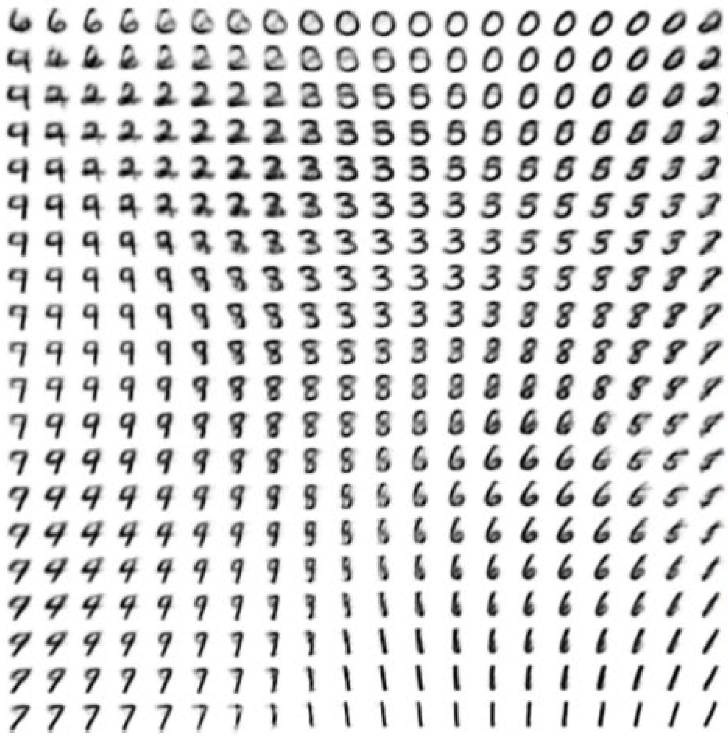


Slide credit: W. Chiu

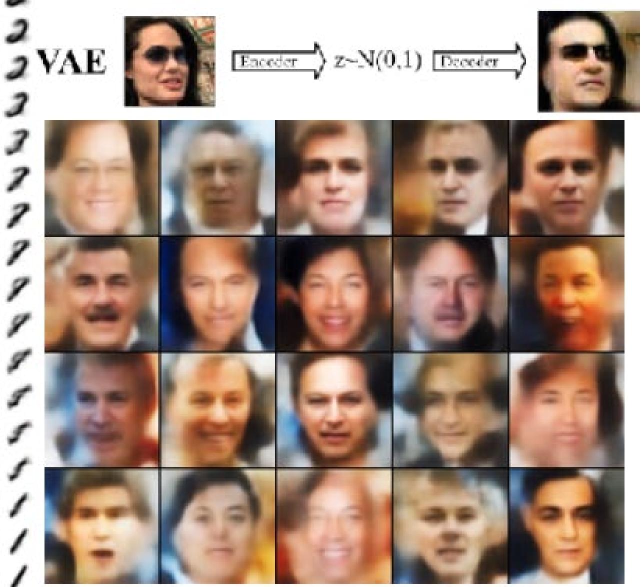
From Autoencoder to Variational Autoencoder (cont'd)

- Example Results

(a) Learned Frey Face manifold



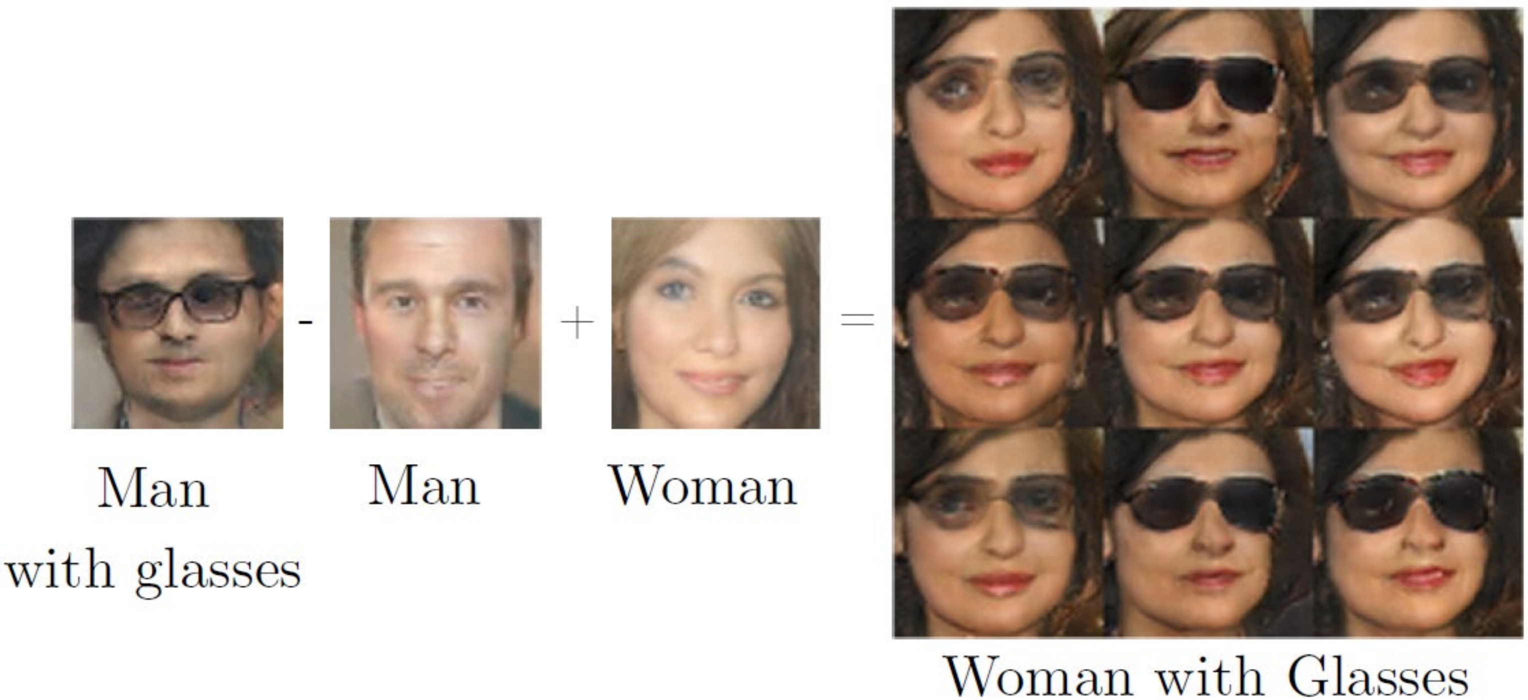
(b) Learned MNIST manifold



Kingma et al., 2013

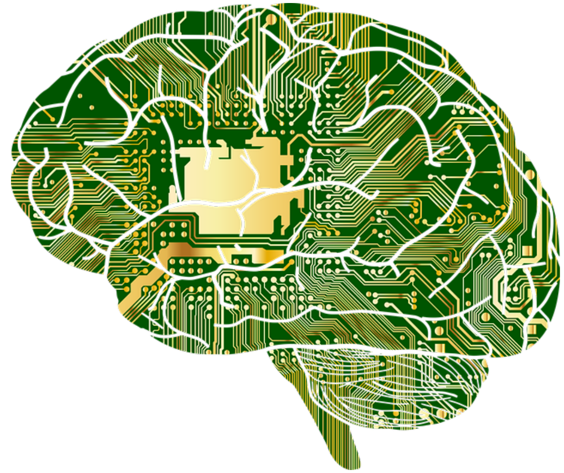
From Autoencoder to Variational Autoencoder (cont'd)

- Example Results
  - A' – A + B = B'



Radford et al., 2015

What We’ve Covered Today…



- Segmentation
- Object Detection
- Generative Model
- Next time: GAN & Diffusion Models

- HW #1 is out & due Oct. 10^{th} Mon 23:59