Designing, Visualizing and Understanding Deep Neural Networks

Lecture 8: Object Detection and Segmentation

CS 182/282A Spring 2020
John Canny

Slides originated from Chen, Chou, Li, Karpathy, Johnson, and Yang

Last Time: Batch Normalization

[Ioffe and Szegedy, 2015]

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

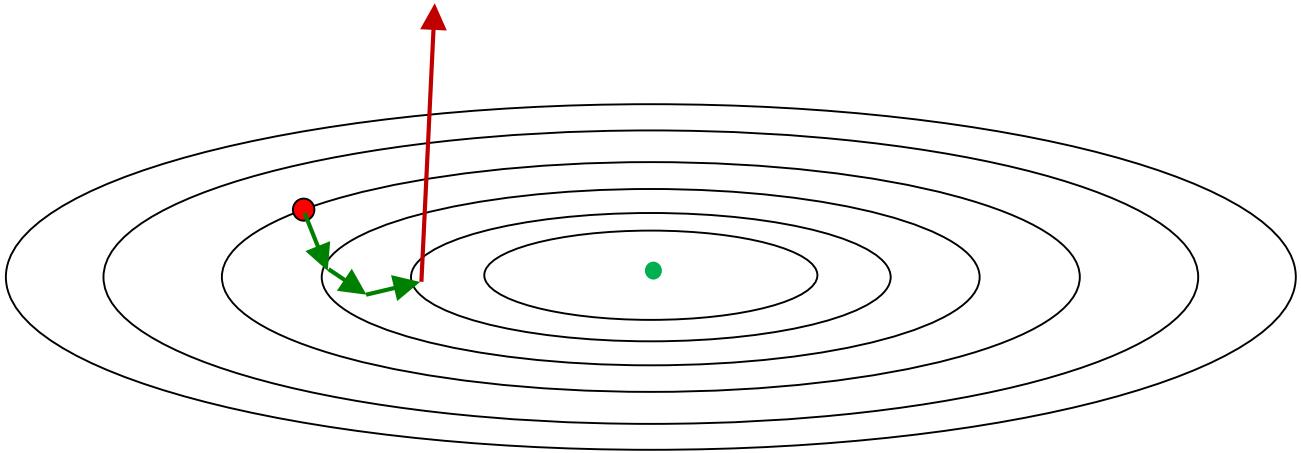
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Reduces need for dropout

Un-normalization!! Re-compute and apply the optimal scaling and bias for each neuron!
Learn γ and β (same dims as μ and σ^2).
It can (should?) learn the identity mapping!

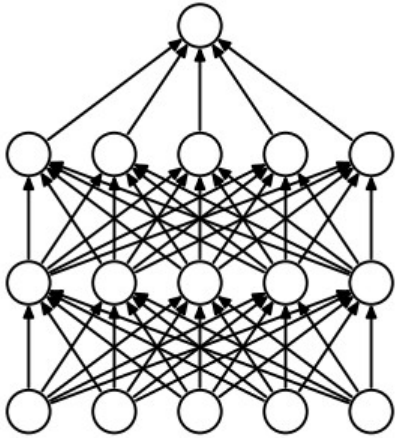
Last Time: Gradient Clipping by Value or Norm



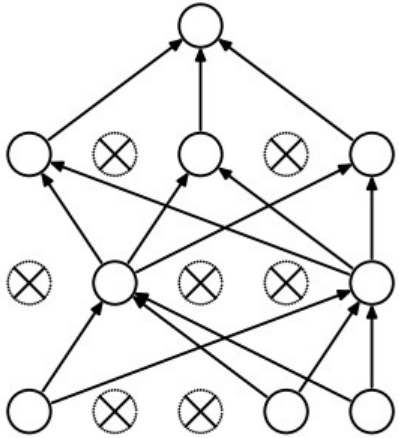
Last Time: Dropout

"randomly set some neurons to zero in the forward pass"

i.e. multiply by random bernoulli variables with parameter p.



(a) Standard Neural Net



(b) After applying dropout.

Note, p is the probability of keeping a neuron

[Srivastava et al., 2014]

Last Time: Ensembles (VGGNet and CIFAR 10)

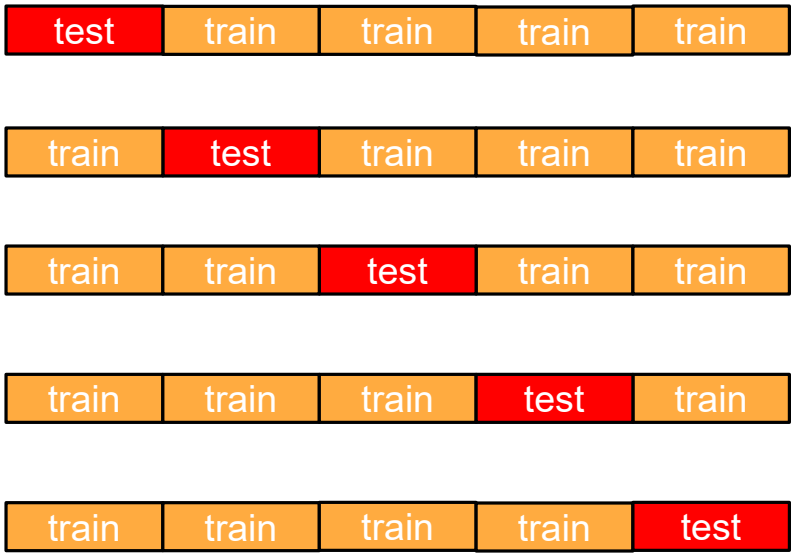
Model	Prediction method	Test Accuracy
Baseline (10 epochs)	Single model	0.837
True ensemble of 10 models	Average predictions	0.855
True ensemble of 10 models	Voting	0.851
Snapshots (25) over 10 epochs	Average predictions	0.865
Snapshots (25) over 10 epochs	Voting	0.861
Snapshots (25) over 10 epochs	Parameter averaging	0.864

Course Updates/Logistics

- CS282A Project Proposals are due Tuesday
- Assignment 1 is due Wednesday…

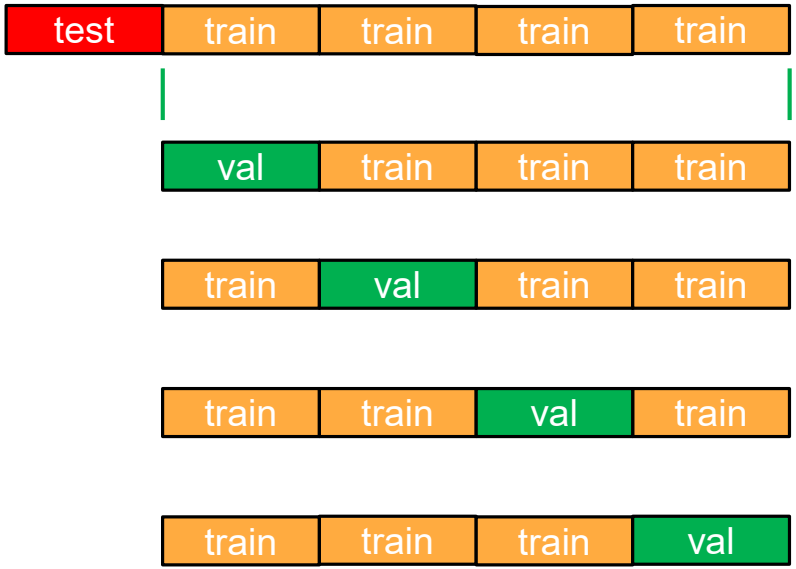
This Time: Hyperparameter Optimization

Normal Cross-Validation: Use a blocked design for testing



Hyperparameter Optimization

Cross-Validation: You can use a validation set(s) for hyperparameter evaluation/tuning ***within each training block***.



Hyper-Parameter Exploration strategy

coarse -> fine cross-validation in stages

First stage: only a few epochs, wide range of parameter values to get rough idea of what params work
Second stage: longer running time, finer search
… (repeat as necessary)

Tip for detecting explosions in the solver:
If the cost is ever > 3 * original cost, break out early

For example: run coarse search for 5 epochs

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)

      trainer = ClassifierTrainer()
      model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
      trainer = ClassifierTrainer()
      best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                              model, two_layer_net,
                                              num_epochs=5, reg=reg,
                                              update='momentum', learning_rate_decay=0.9,
                                              sample_batches = True, batch_size = 100,
                                              learning_rate=lr, verbose=False)
```

note it's best to optimize in log space!

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

nice

Now run finer search...

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

adjust range

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good for a 2-layer neural net with 50 hidden neurons.

Now run finer search...

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

adjust range

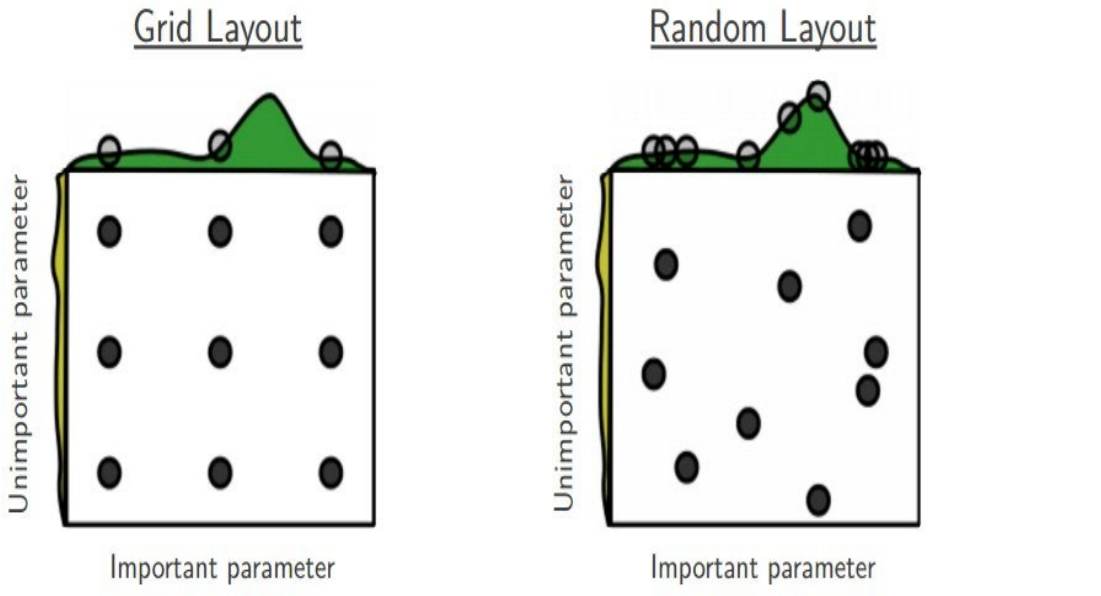
```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good for a 2-layer neural net with 50 hidden neurons.

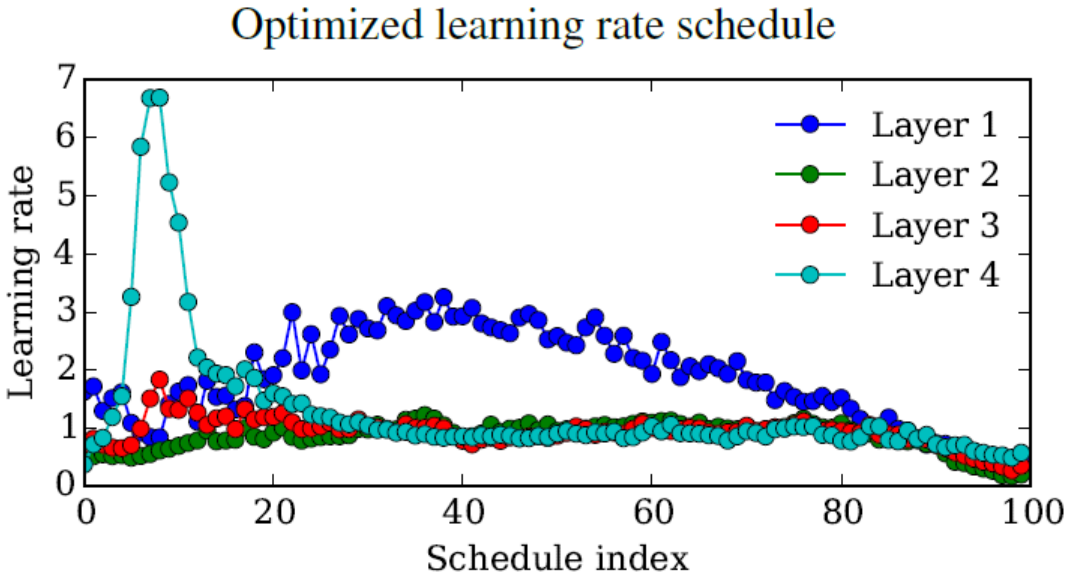
But this best cross-validation result is worrying. Why?

Random Search vs. Grid Search



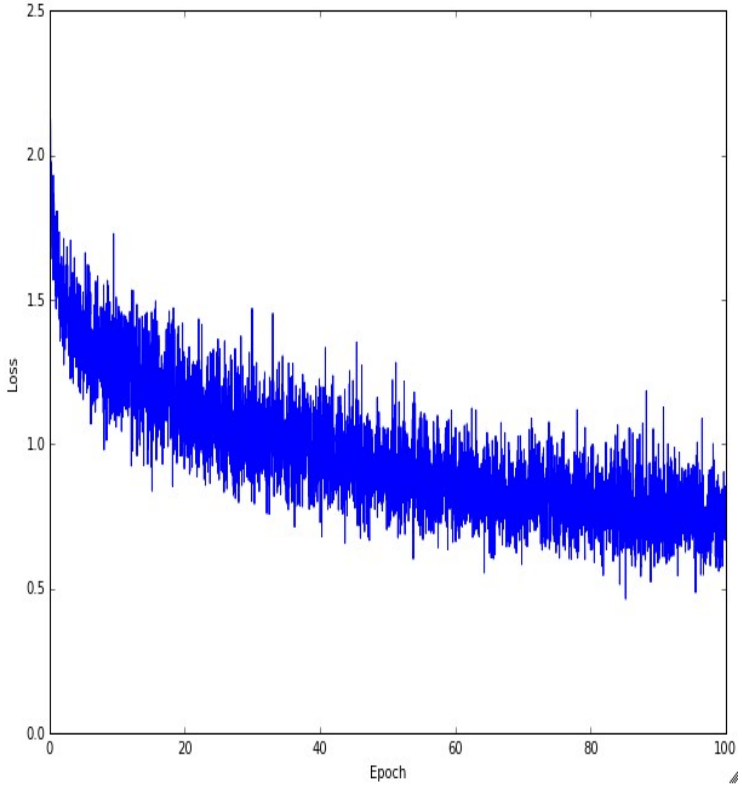
Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012

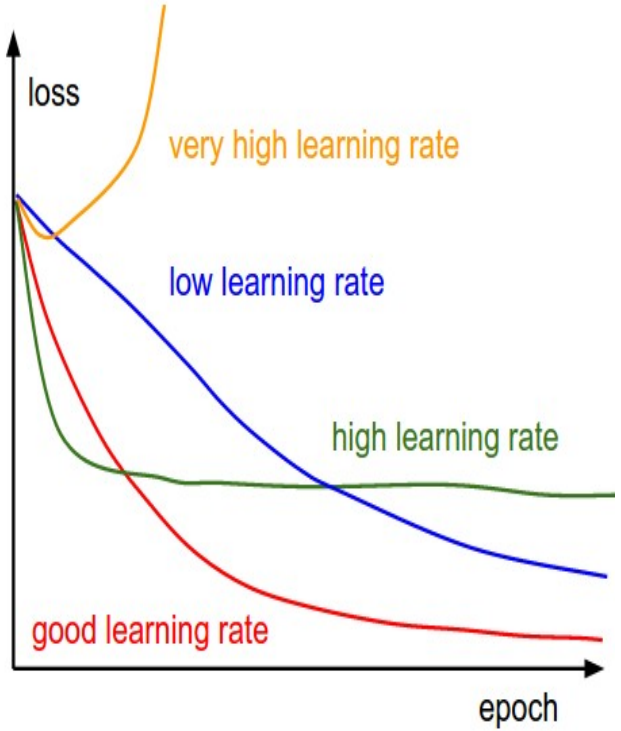
Optimal Hyper-parameters



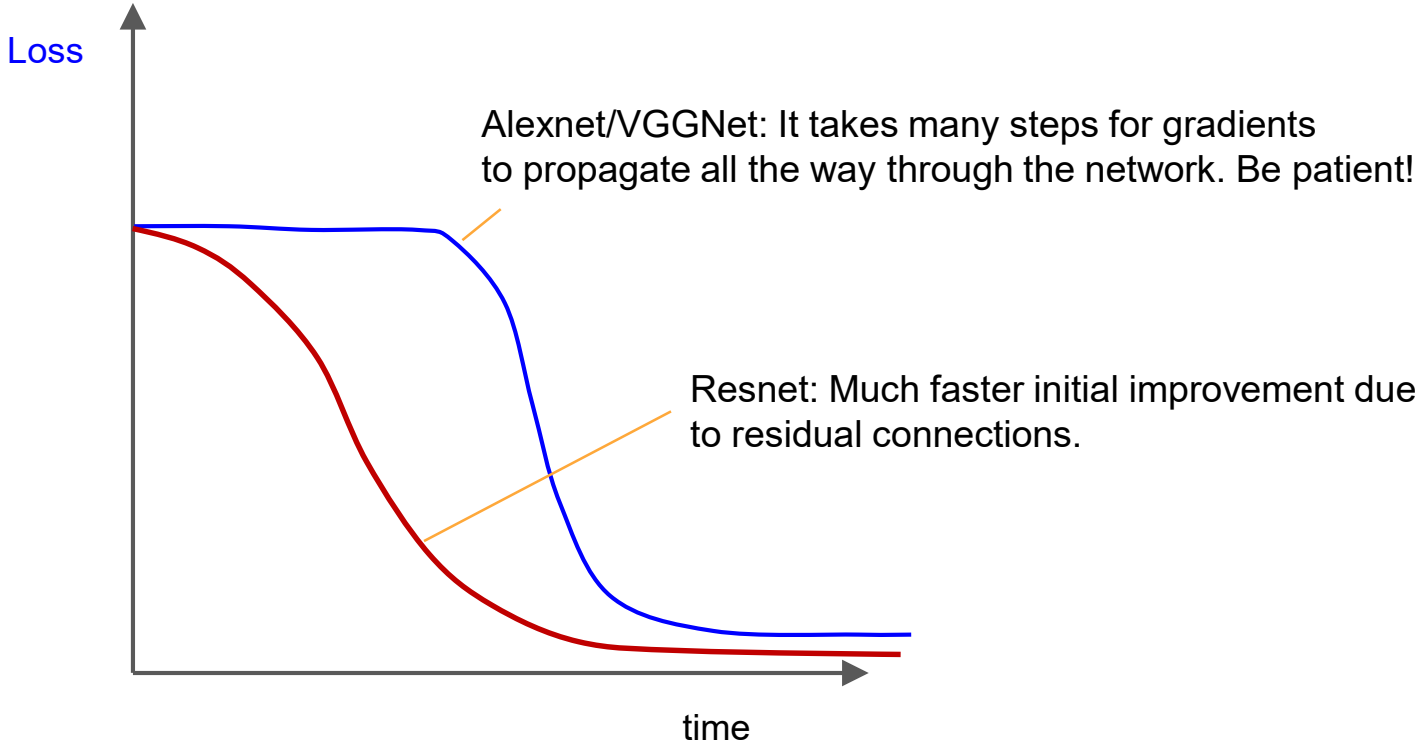
From [“Gradient-based Hyperparameter Optimization through Reversible Learning”] Dougal et al., 2015

Monitor and visualize the loss curve





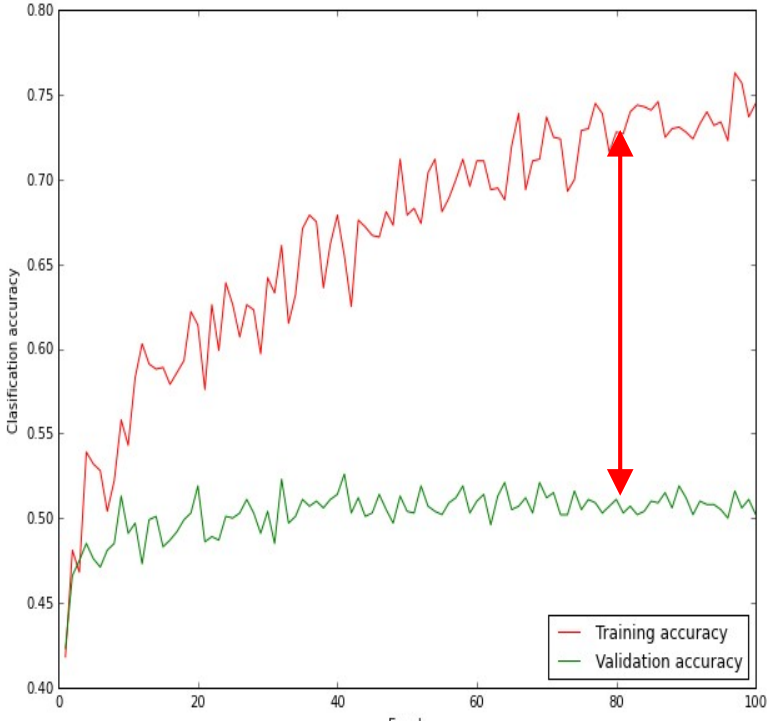
Visualize the loss curve: Check your Expectations



Other Features to capture and plot

- Per-layer activations:
  - Magnitude, center (mean or median), breadth (sdev or quartiles)
  - Spatial/feature-rank variations
- Gradients
  - Magnitude, center (mean or median), breadth (sdev or quartiles)
  - Spatial/feature-rank variations
- Learning trajectories
  - Plot parameter values in a low-dimensional space

Monitor and visualize the accuracy:



Big gap = overfitting

=> increase regularization strength

Very small gap

⇒ increase model capacity

⇒ Try a larger learning rate

Track the ratio of weight updates / weight magnitudes:

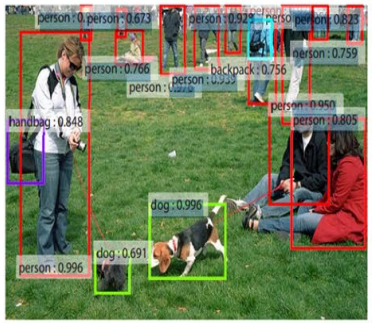
```
# assume parameter vector W and its gradient vector dW
param_scale = np.linalg.norm(W.ravel())
update = -learning_rate*dW # simple SGD update
update_scale = np.linalg.norm(update.ravel())
W += update # the actual update
print update_scale / param_scale # want ~1e-3
```

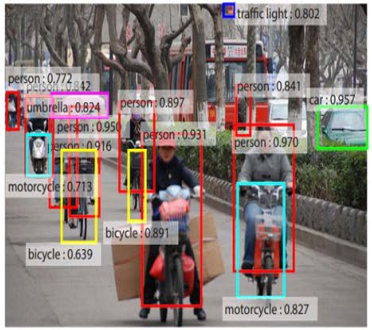
ratio between the values and updates: ~ 0.0002 / 0.02 = 0.01

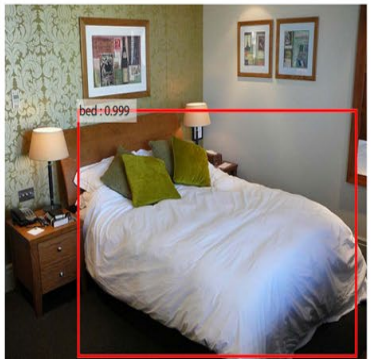
This is learning rate dependent and will typically decrease over time.

Shouldn't be too large initially. 0.01 is probably too high…

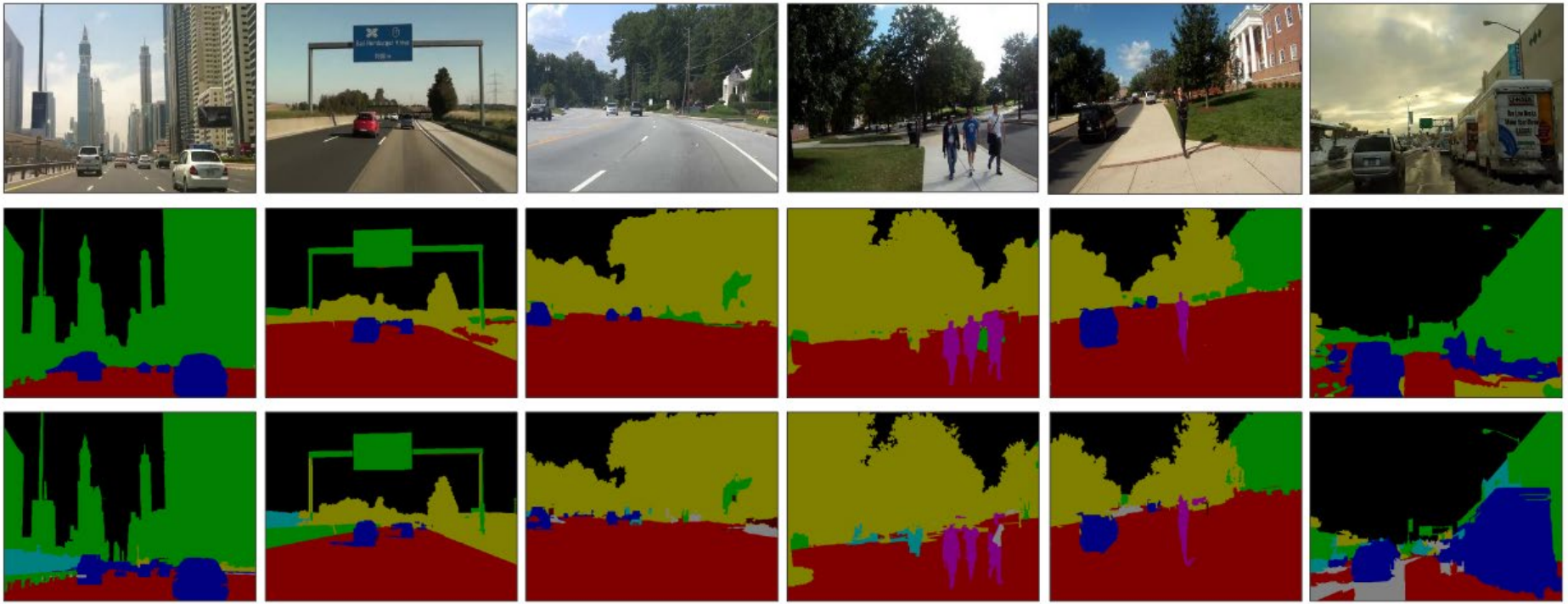
This Time: Localization and Detection



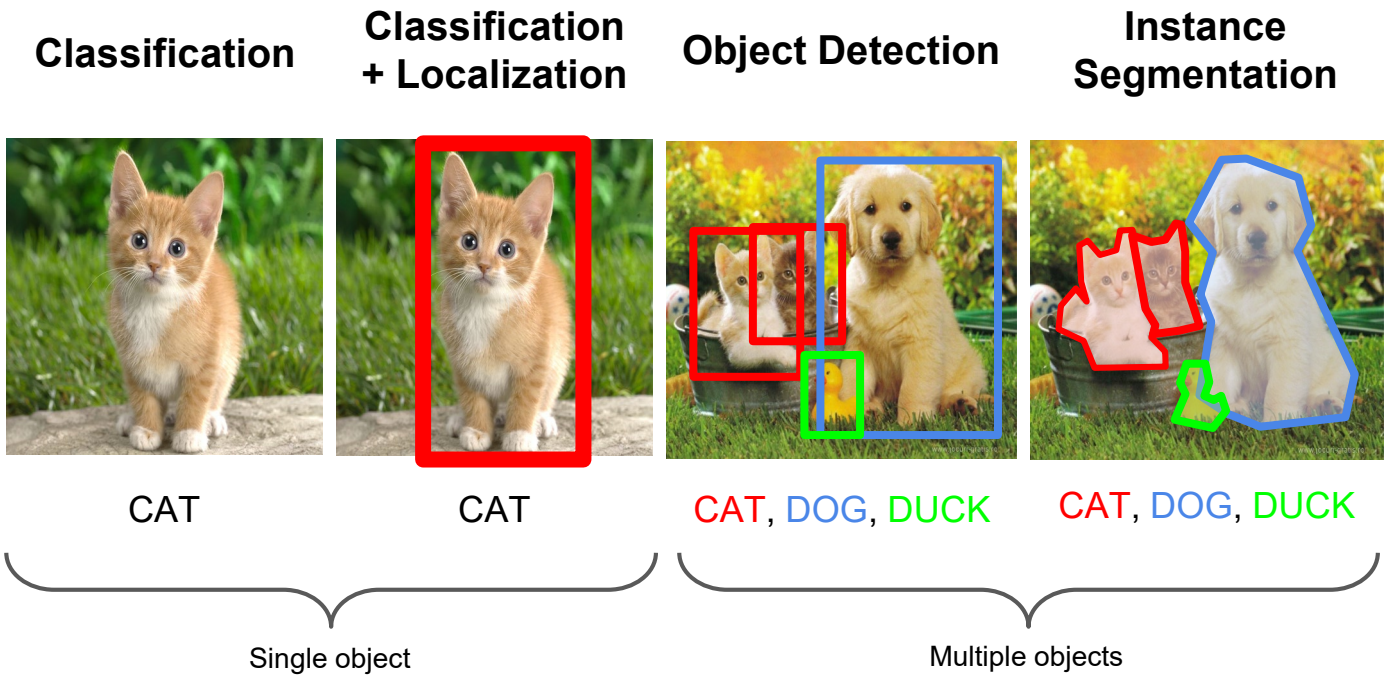




This Time: Localization and Detection



Computer Vision Tasks



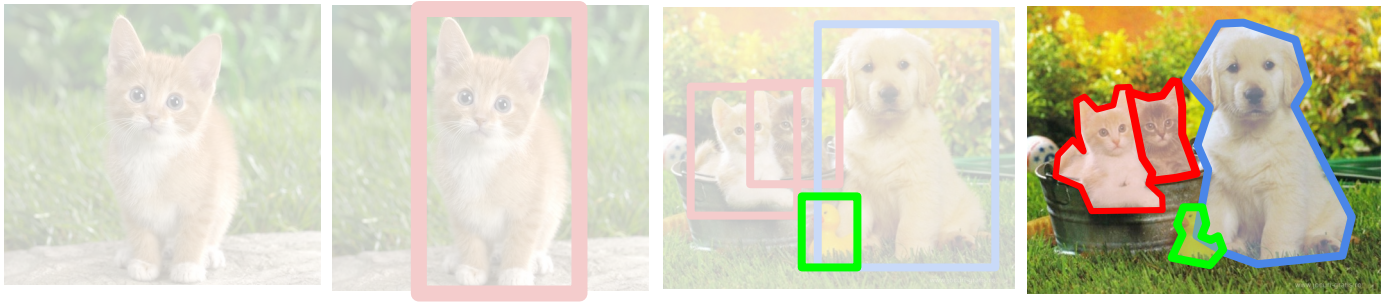
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation



Semantic Segmentation

Label every pixel!

Don't differentiate instances, only worry about pixels

Classic computer vision problem

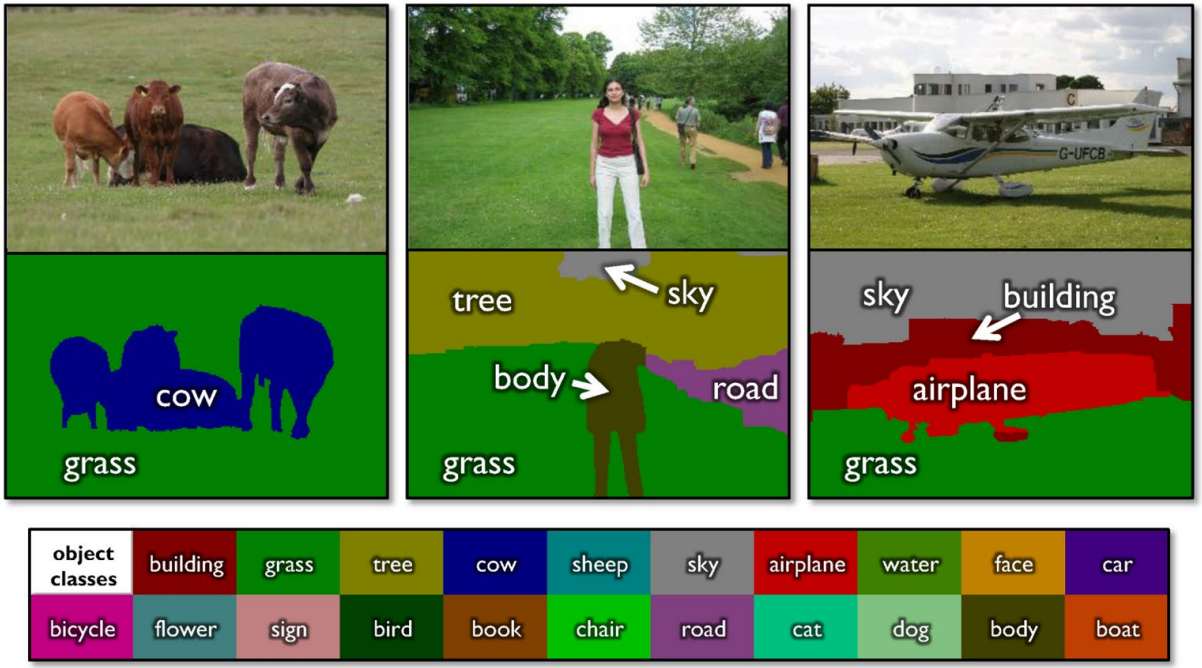
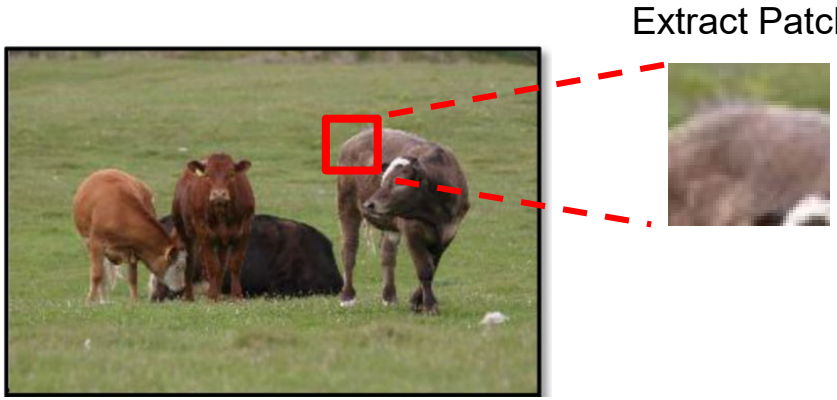


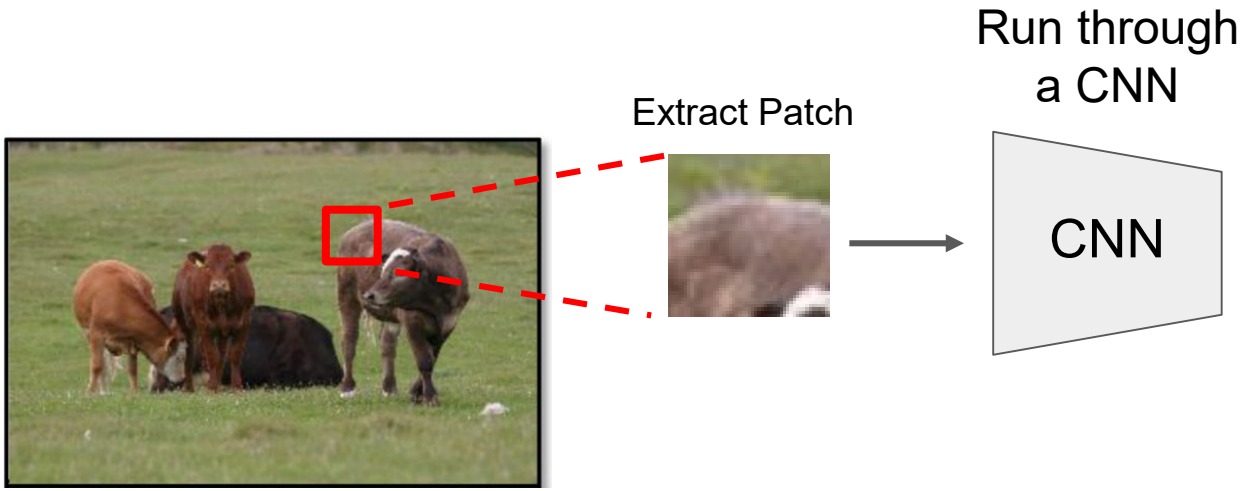
Figure credit: Shotton et al, "TextonBoost for Image Understanding: Multi-Class Object Recognition and Segmentation by Jointly Modeling Texture, Layout, and Context", IJCV 2007

Idea #1 – Classify Every Pixel

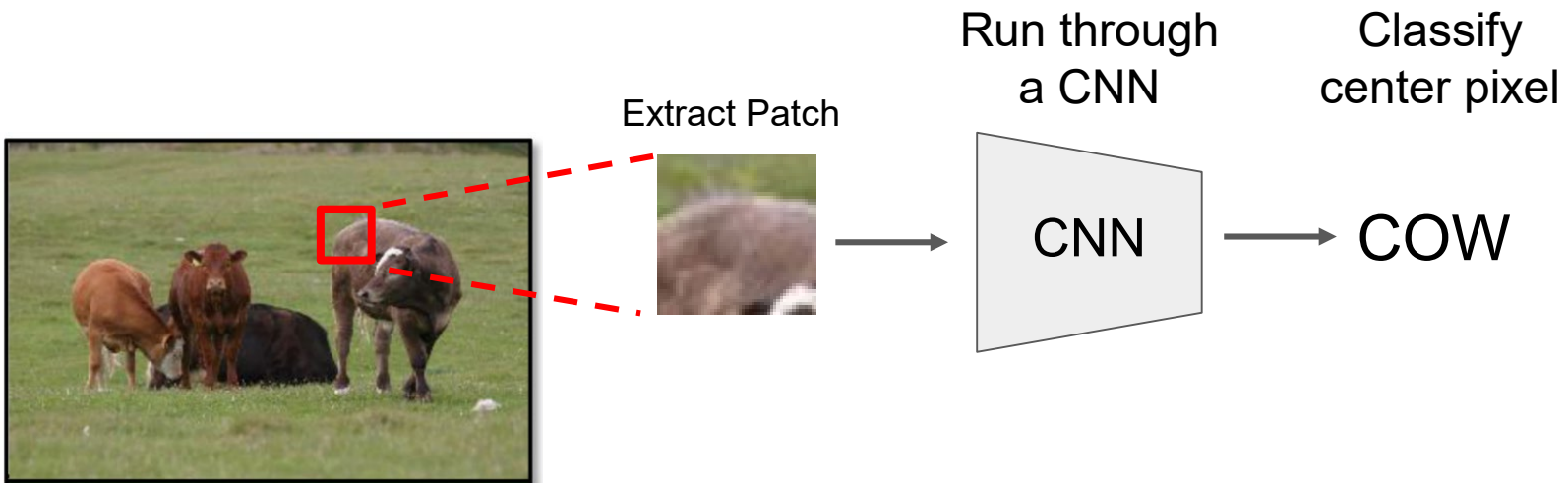
Idea #1 – Classify Every Pixel



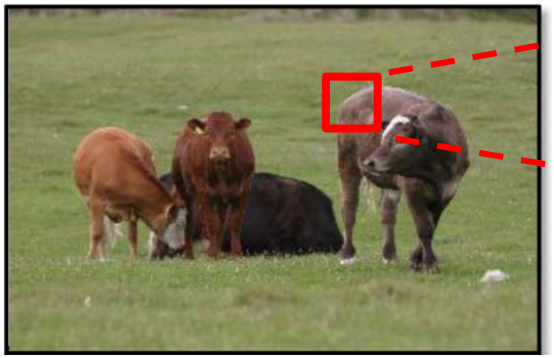
Idea #1 – Classify Every Pixel



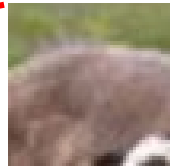
Idea #1 – Classify Every Pixel



Idea #1 – Classify Every Pixel



Extract Patch



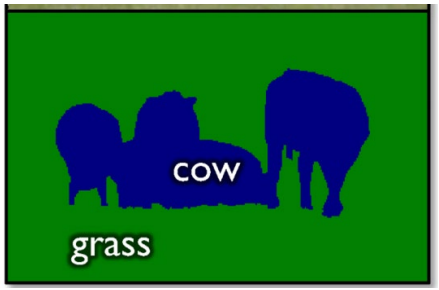
Run through a CNN

CNN

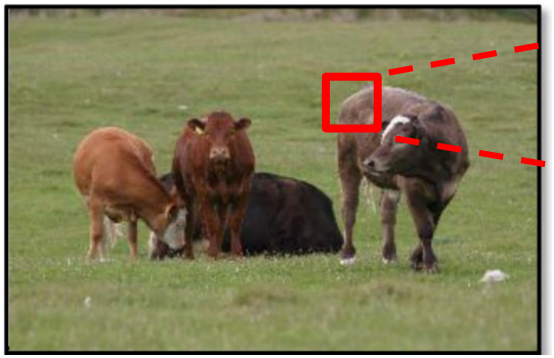
Classify center pixel

COW

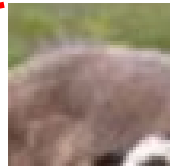
Repeat for every pixel



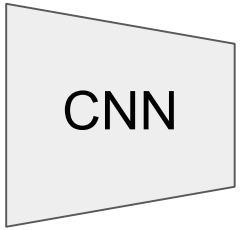
Idea #1 – Classify Every Pixel



Extract Patch

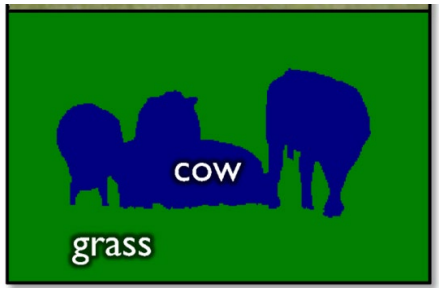


Run through a CNN



Classify center pixel

COW



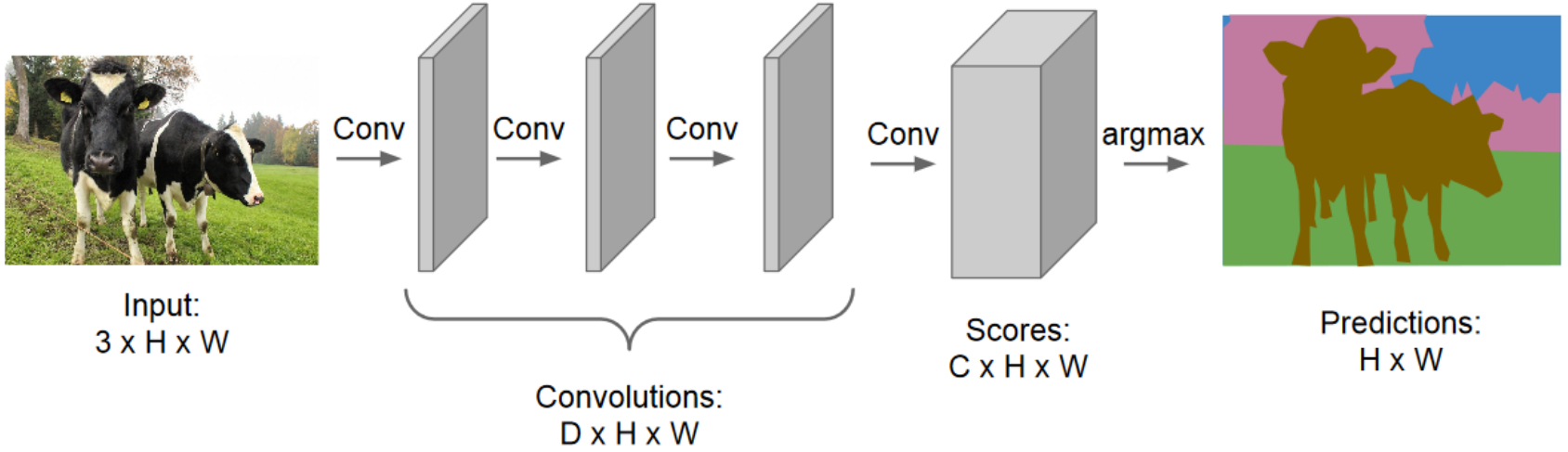
Repeat for every pixel

Issues:

- **Repeats the same computation multiple times**
- **Doesn't make use of global information**

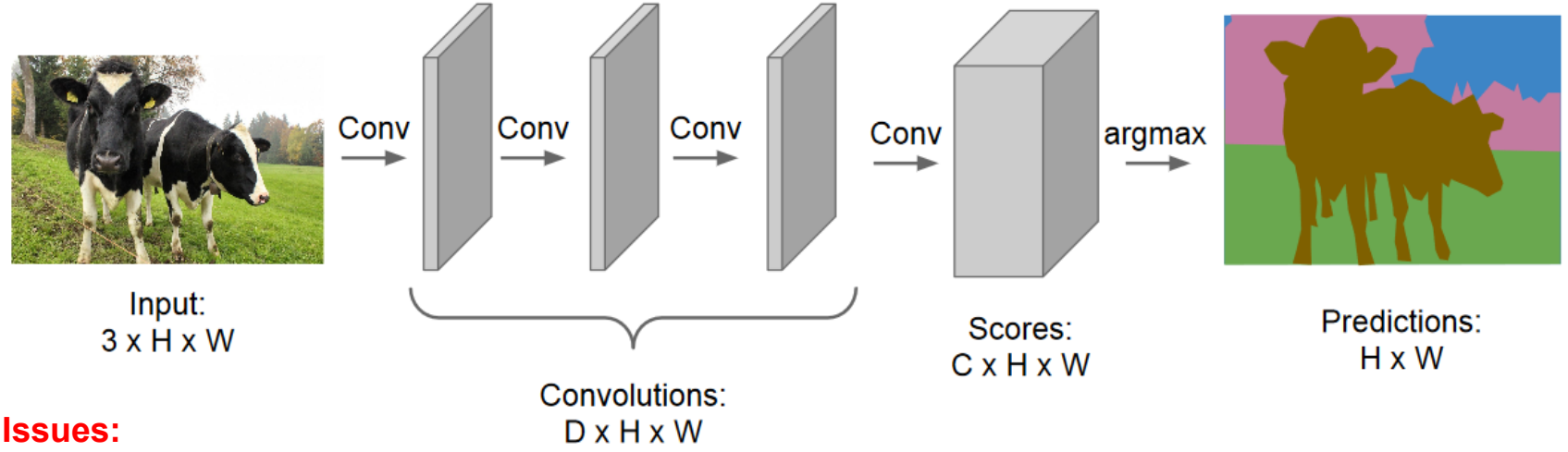
Idea #2 – Convolutional Network

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



Idea #2 – Convolutional Network

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



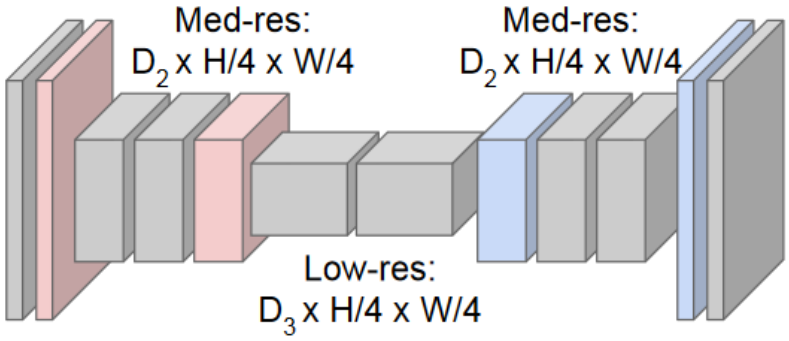
Issues:

- **Convolutions at full resolution can be expensive**

Idea #3 – Fully Connected CNN

Design a network as a bunch of convolutional layers with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W



High-res:
D1 x H/2 x W/2

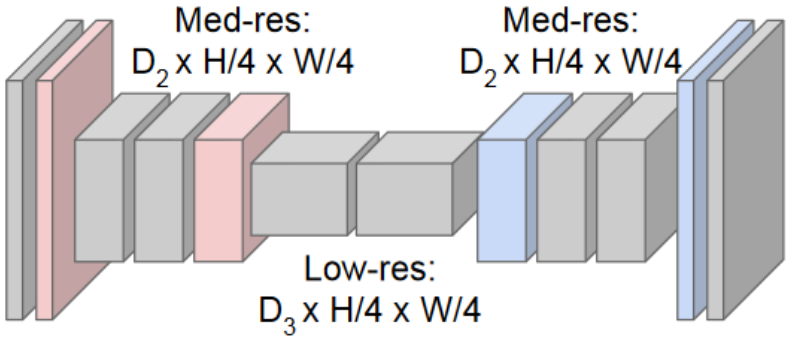
High-res:
D1 x H/2 x W/2

Predictions:
H x W

Idea #3 – Fully Connected CNN

Design a network as a bunch of convolutional layers with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W



High-res:
D1 x H/2 x W/2

High-res:
D1 x H/2 x W/2

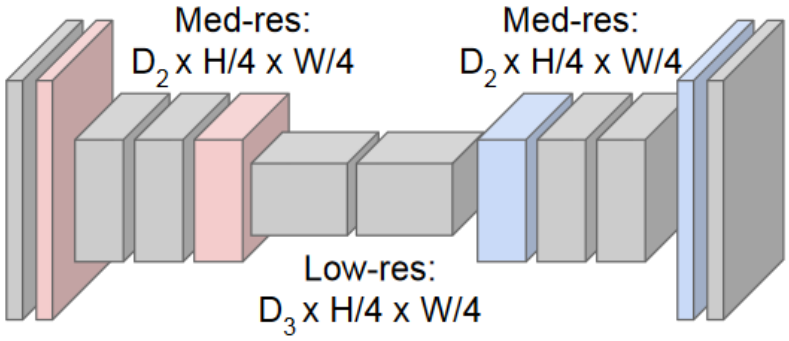
Predictions:
H x W

Downsampling:
Pooling, strided convolution

Idea #3 – Fully Connected CNN

Design a network as a bunch of convolutional layers with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W

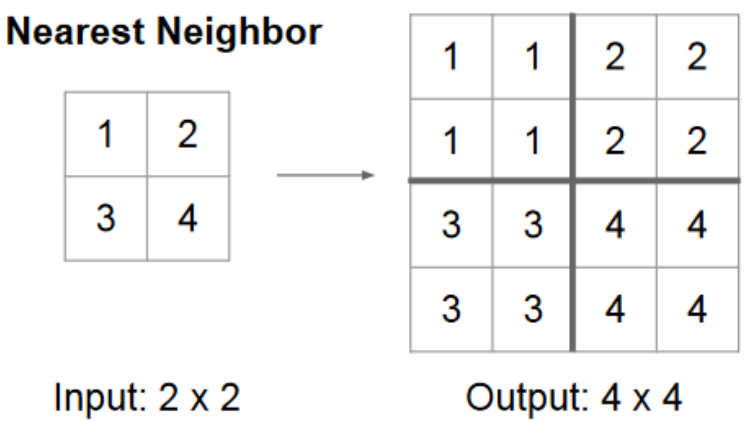


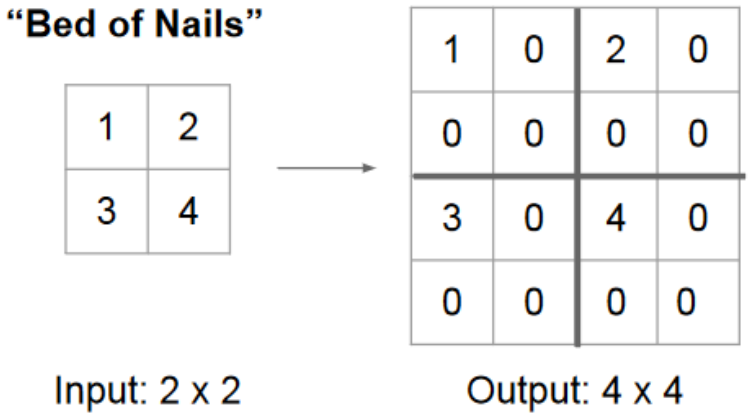
Predictions:
H x W

Downsampling:
Pooling, strided convolution

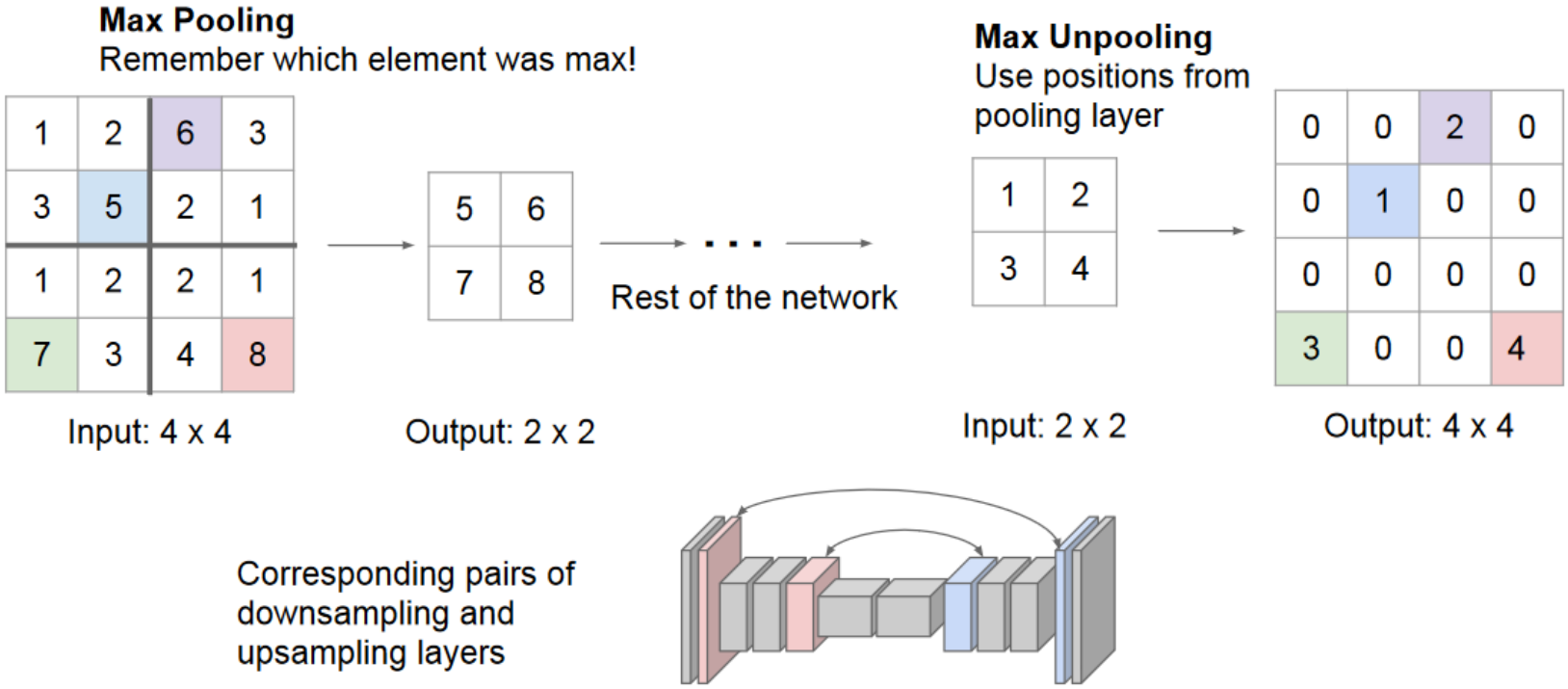
Upsampling:
???

In-Network Upsampling: Striding



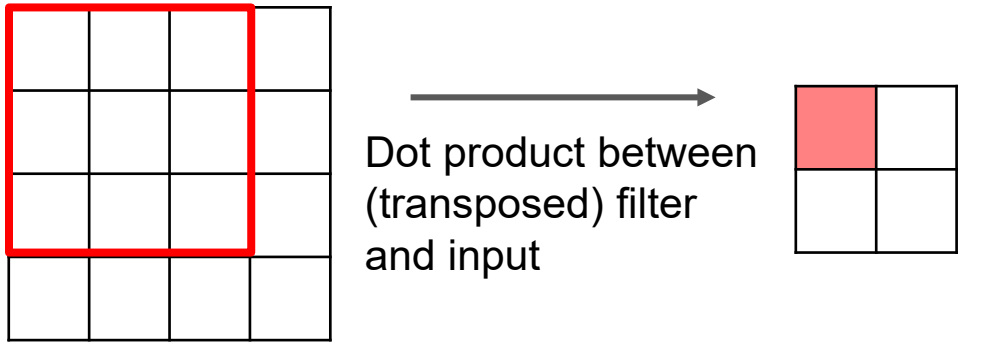


In-Network Upsampling: Max Unpooling



Learnable Upsampling: Transpose Convolution

Recall: Typical 3 x 3 convolution, stride 1

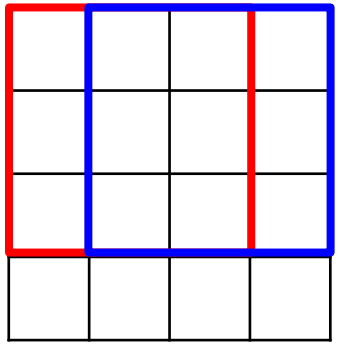


Input: 4 x 4

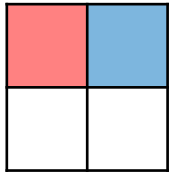
Output: 2 x 2

Learnable Upsampling: Transpose Convolution

Recall: Typical 3 x 3 convolution, stride 1



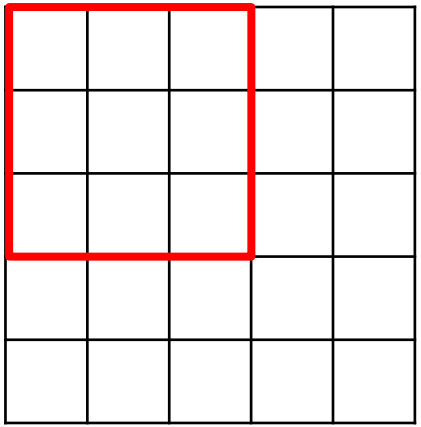
Input: 4 x 4



Output: 2 x 2

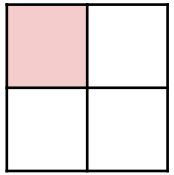
Learnable Upsampling: Transpose Convolution

Recall: Typical 3 x 3 convolution, **stride 2**



Input: 5 x 5

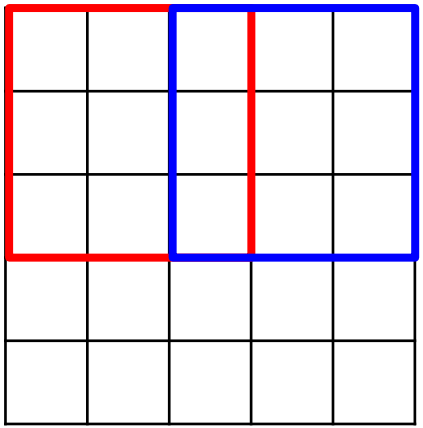
Dot product between (transposed) filter and input



Output: 2 x 2

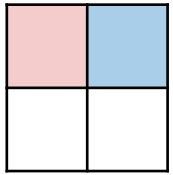
Learnable Upsampling: Transpose Convolution

Recall: Typical 3 x 3 convolution, **stride 2**



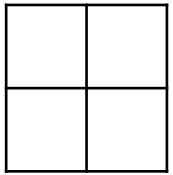
Input: 5 x 5

Dot product between transposed filter and input

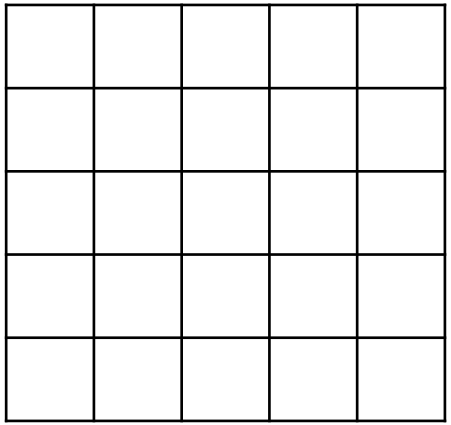


Output: 2 x 2

Learnable Upsampling: Transpose Convolution

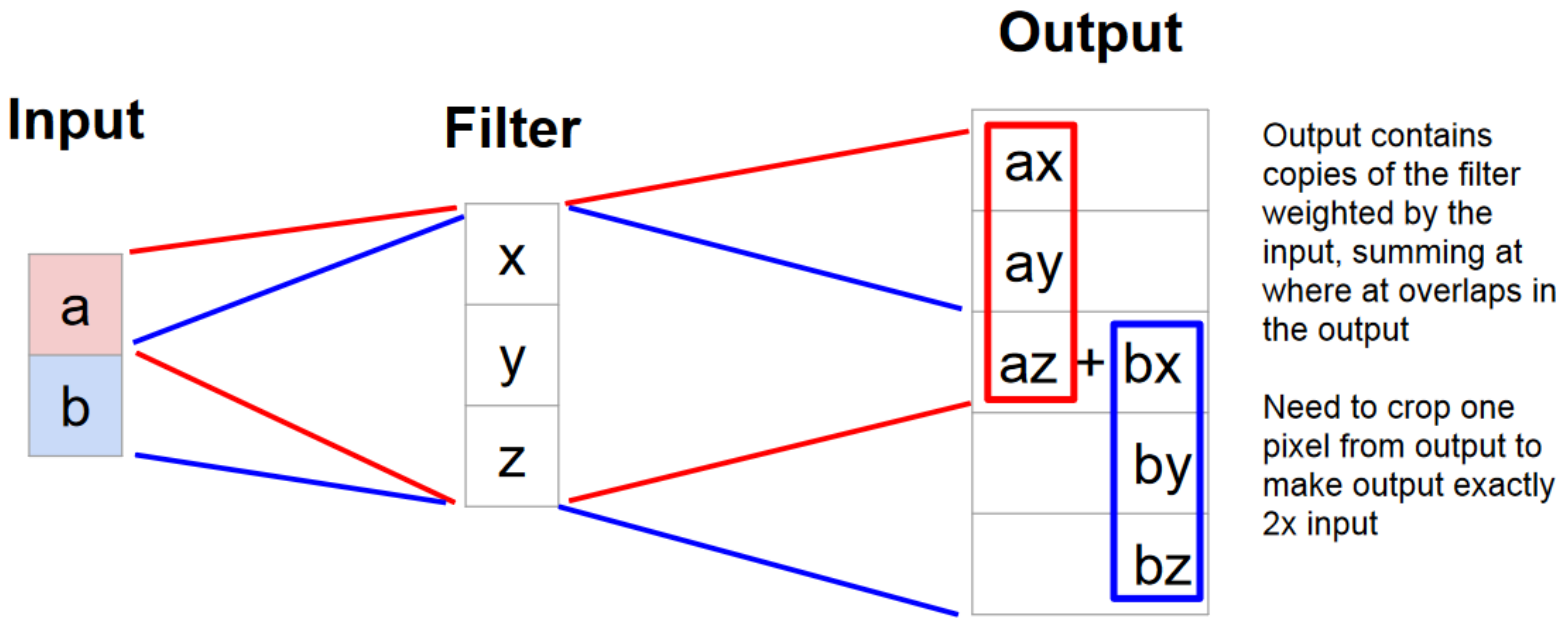


Input: 2 x 2



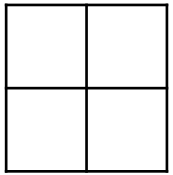
Output: 5 x 5

Learnable Upsampling: Transpose Convolution

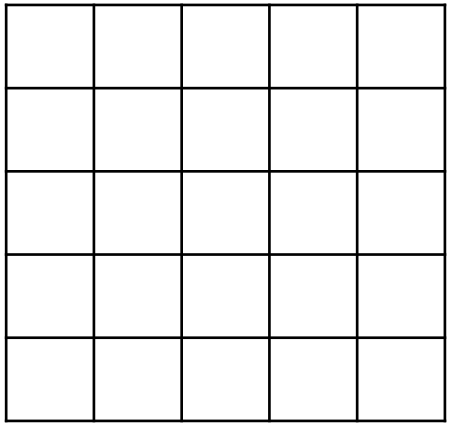


Learnable Upsampling: Transpose Convolution

Transpose Convolution: Typical 3 x 3 convolution, stride 2



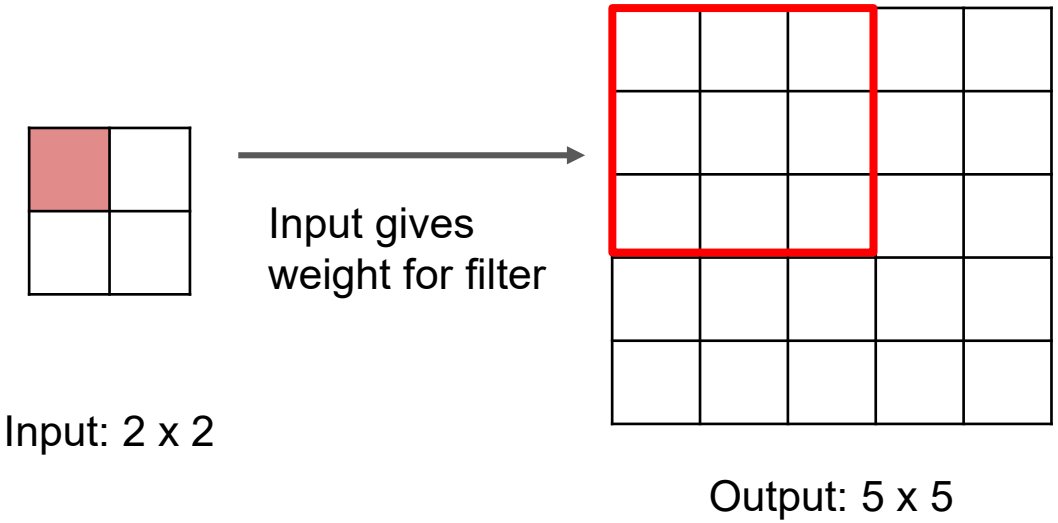
Input: 2 x 2



Output: 5 x 5

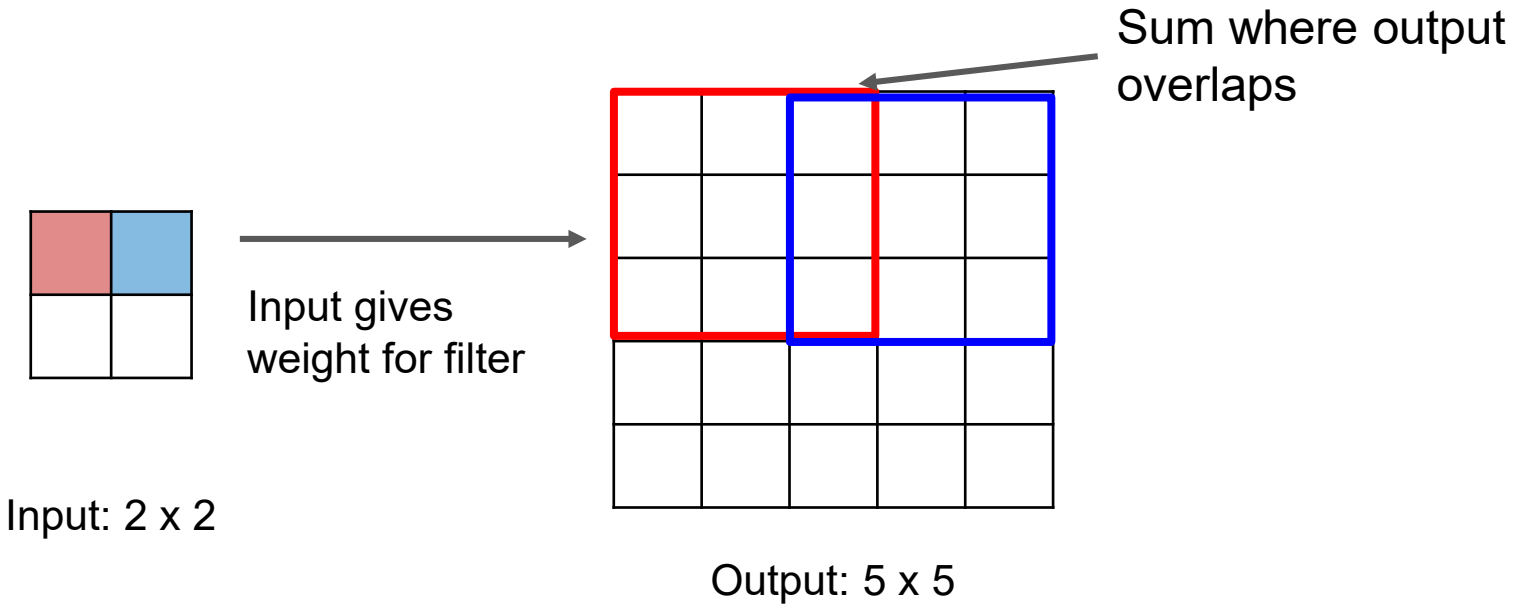
Learnable Upsampling: Transpose Convolution

Transpose Convolution: Typical 3 x 3 convolution, stride 2



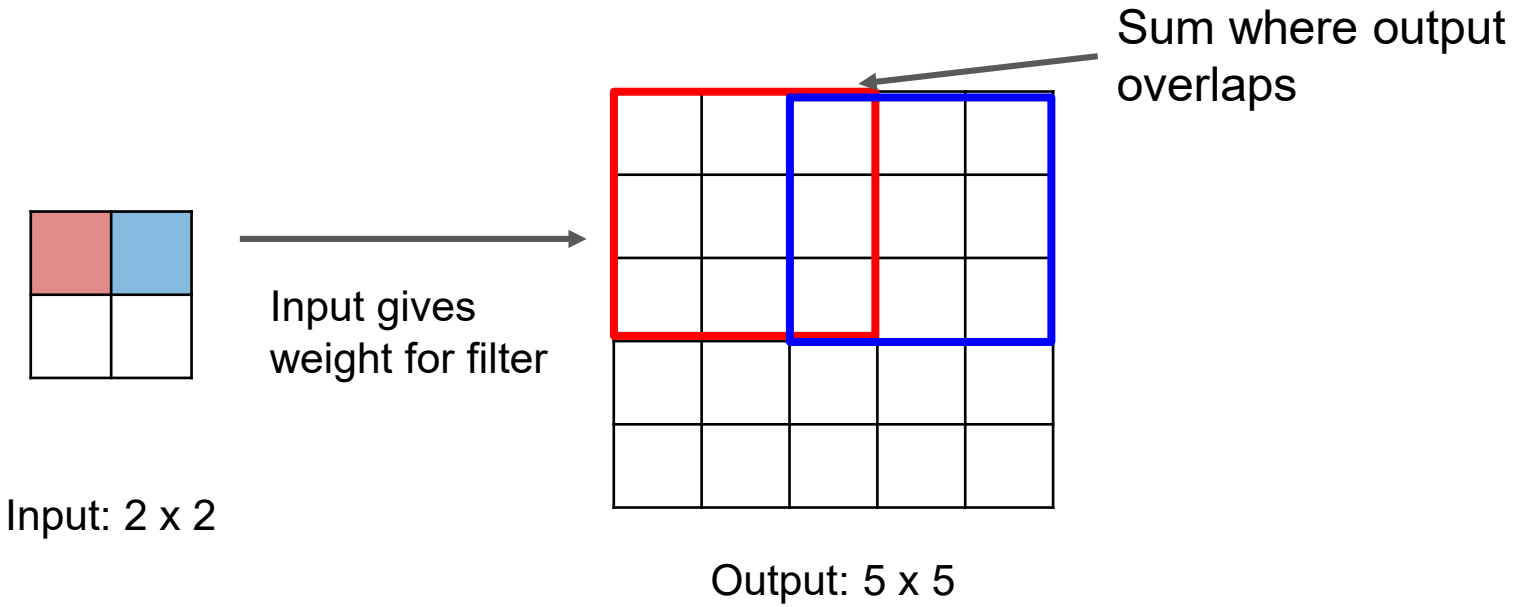
Learnable Upsampling: Transpose Convolution

Transpose Convolution: Typical 3 x 3 convolution, stride 2



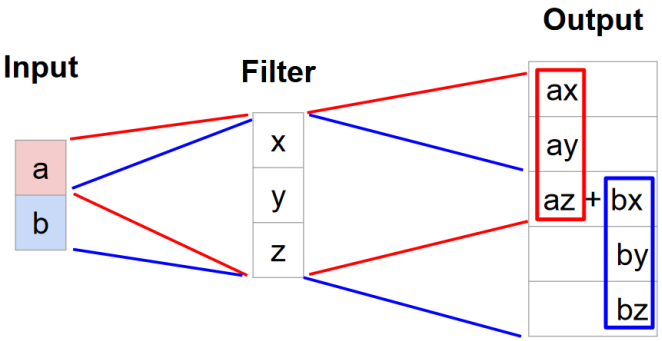
Learnable Upsampling: Transpose Convolution

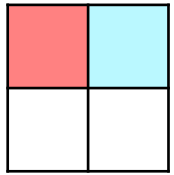
Transpose Convolution: Typical 3 x 3 convolution, stride 2



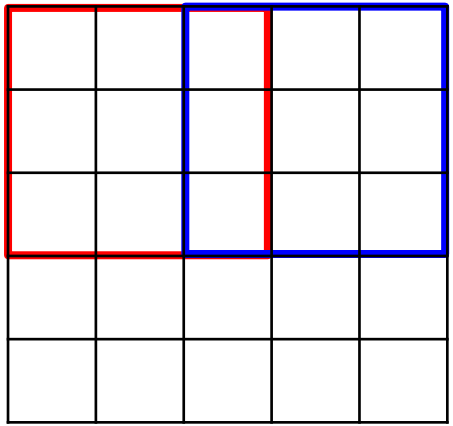
Same as backward pass for normal convolution!

Learnable Upsampling: Transpose Convolution





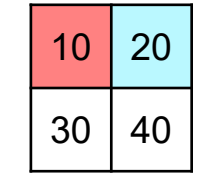
Input gives weight for filter



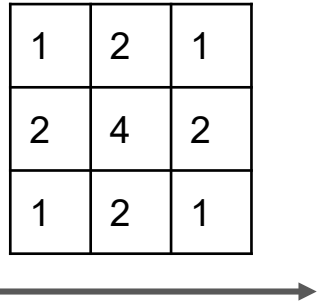
Input: 2 x 2

Output: 5 x 5

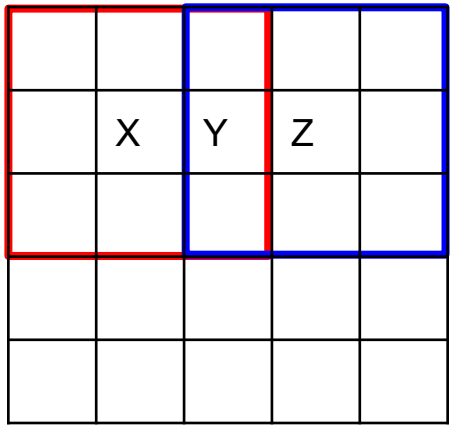
Learnable Upsampling: Transpose Convolution



Input: 2 x 2



Filter: 3 x 3



Output: 5 x 5

X, Y, Z, are respectively:

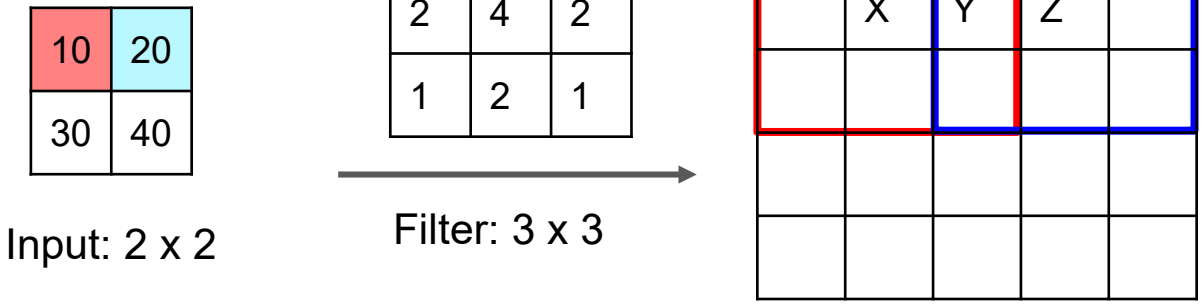
A. 20, 20, 20

B. 20, 30, 40

C. 40, 60, 80

D. 80, 60, 40

Oops!



Input: 2 x 2

Filter: 3 x 3

Output: 5 x 5

X, Y, Z, are respectively:

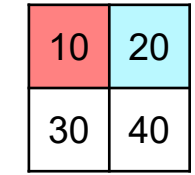
A. 20, 20, 20

No, the input is not constant horizontally

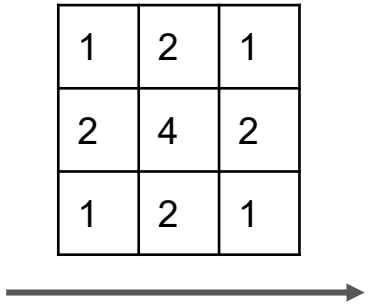
Try Again

Continue

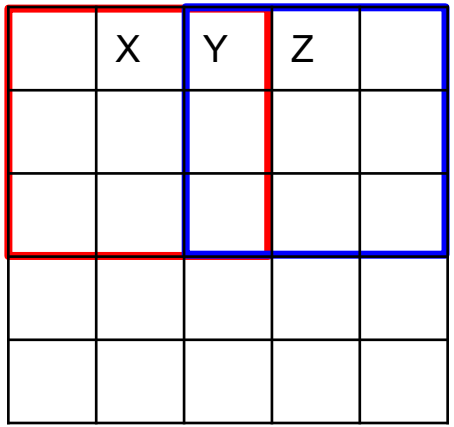
Oops!



Input: 2 x 2



Filter: 3 x 3



Output: 5 x 5

X, Y, Z, are respectively:

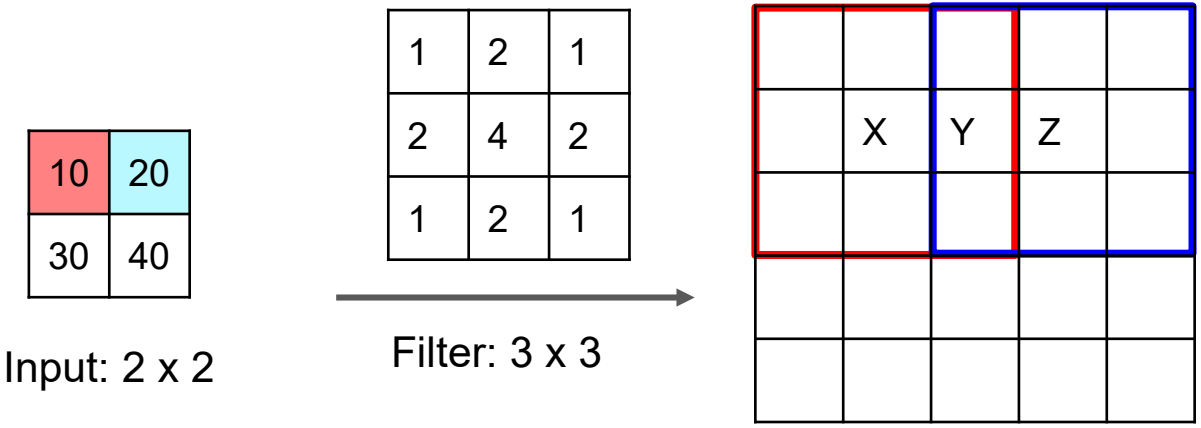
B. 20, 30, 40

No, but correct for the row above

Try Again

Continue

Correct!



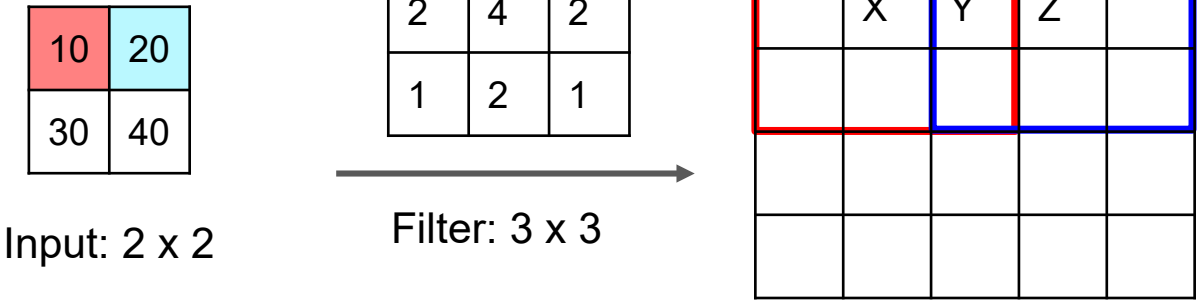
X, Y, Z, are respectively:

C. 40, 60, 80

Try Again

Continue

Oops!



Input: 2 x 2

Filter: 3 x 3

Output: 5 x 5

X, Y, Z, are respectively:

D. 80, 60, 40

No, wrong horizontal order

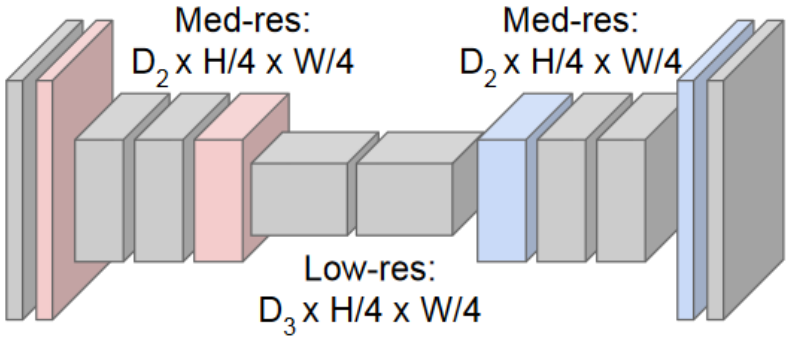
Try Again

Continue

Idea #3 – Fully Connected CNN

Design a network as a bunch of convolutional layers with **downsampling** and **upsampling** inside the network!

Input:
3 x H x W



High-res:
D1 x H/2 x W/2

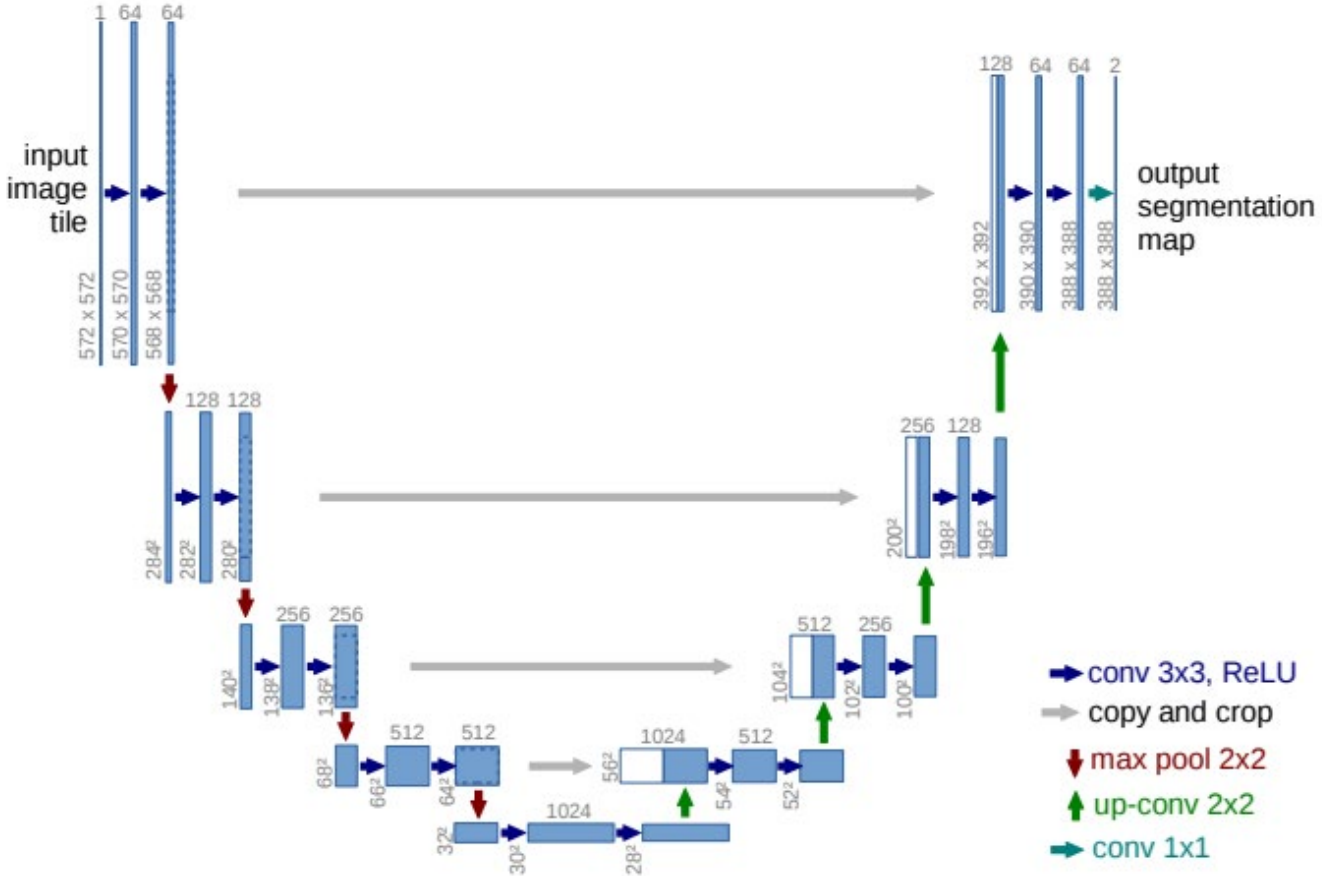
High-res:
D1 x H/2 x W/2

Predictions:
H x W

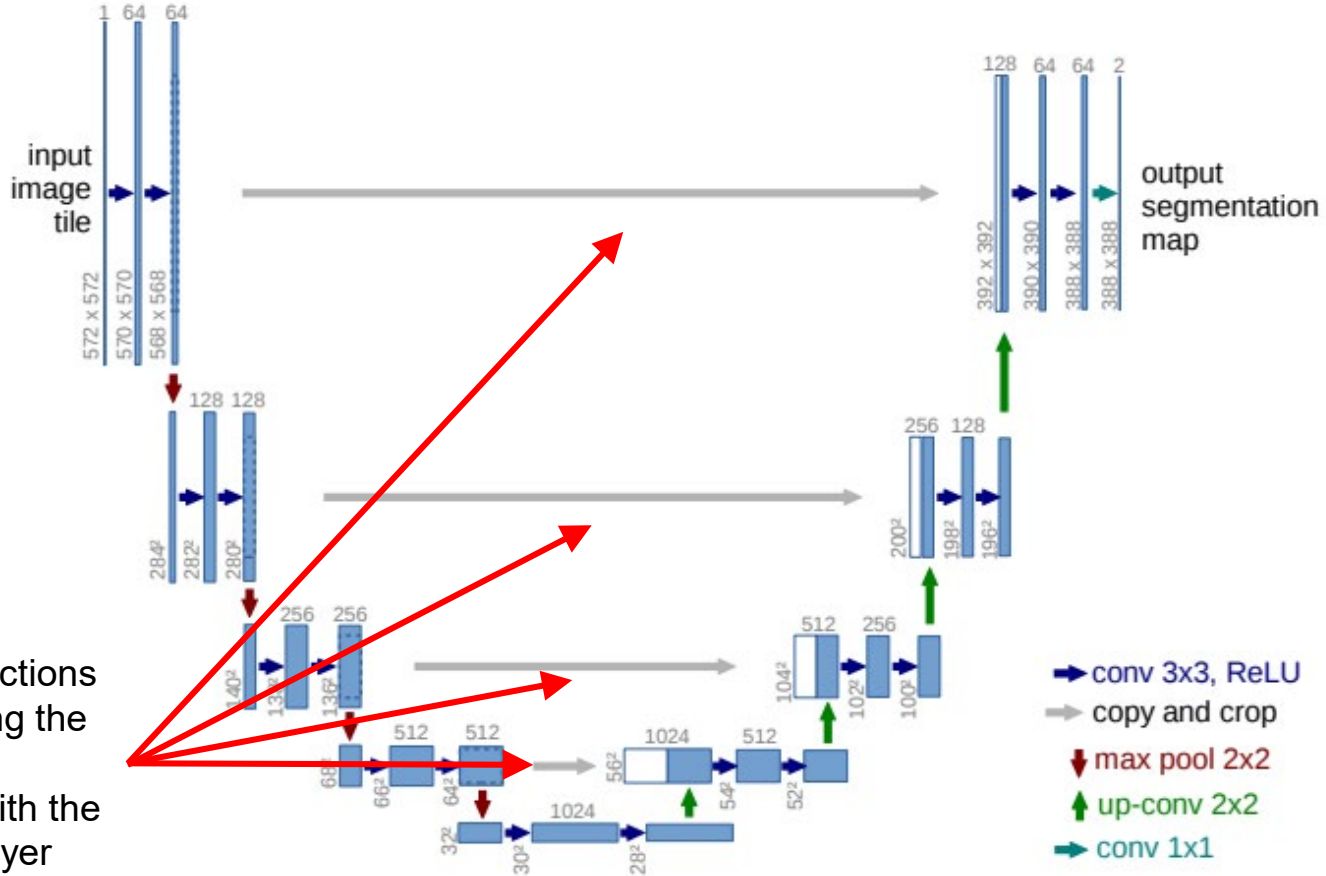
Downsampling:
Pooling, strided convolution

Upsampling:
Strided Transpose convolution

Idea #4 – UNet (FCNN + Residuals)



Idea #4 – UNet (FCNN + Residuals)



Residual Connections Made by copying the input, and concatenating with the upsampled layer

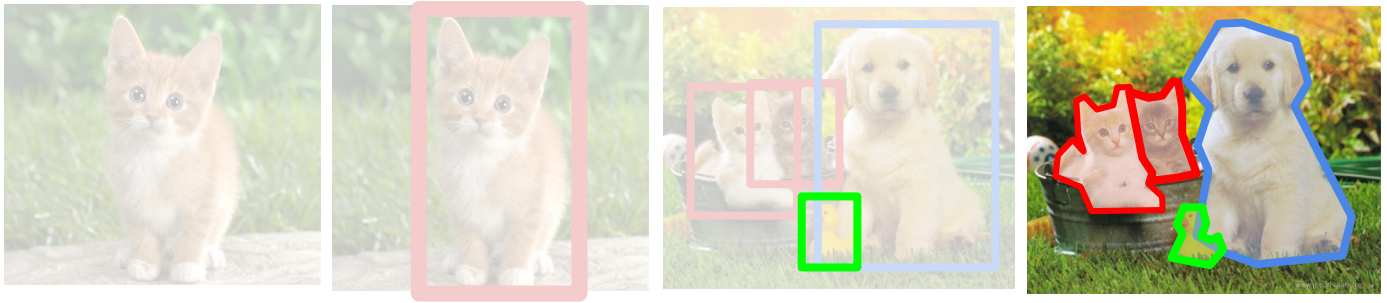
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation



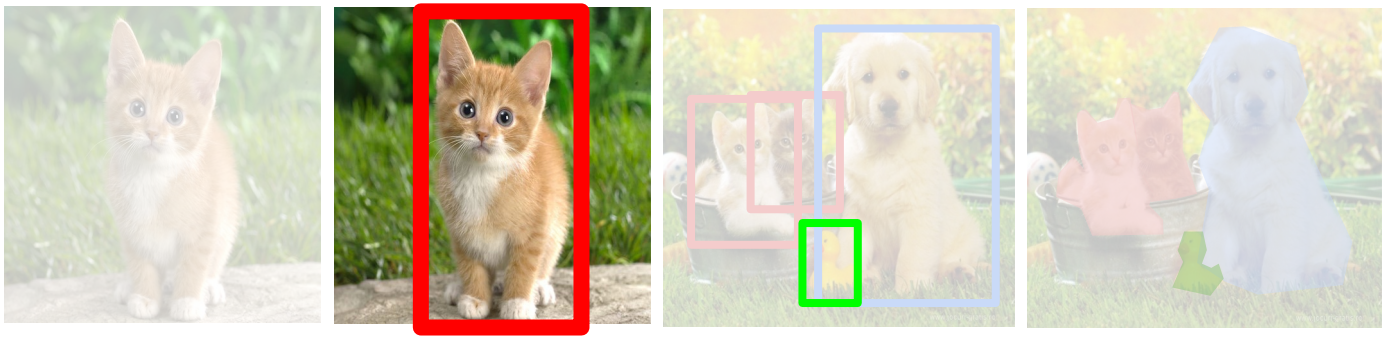
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation



Classification + Localization

Classification

Input: Image
Output: Class Label
Evaluation Metric: Class Accuracy

Cat

Classification + Localization

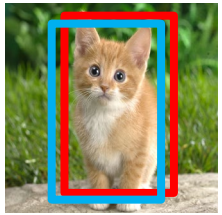
Classification

Input: Image
Output: Class Label
Evaluation Metric: Class Accuracy

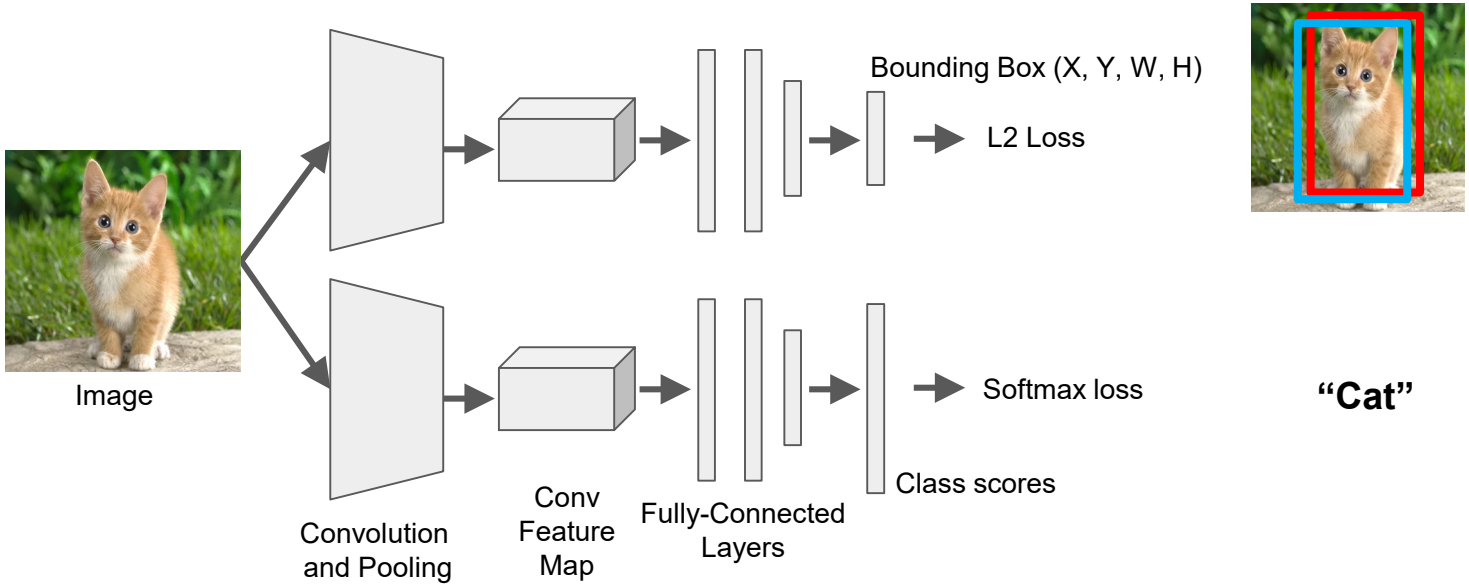
Cat

Localization

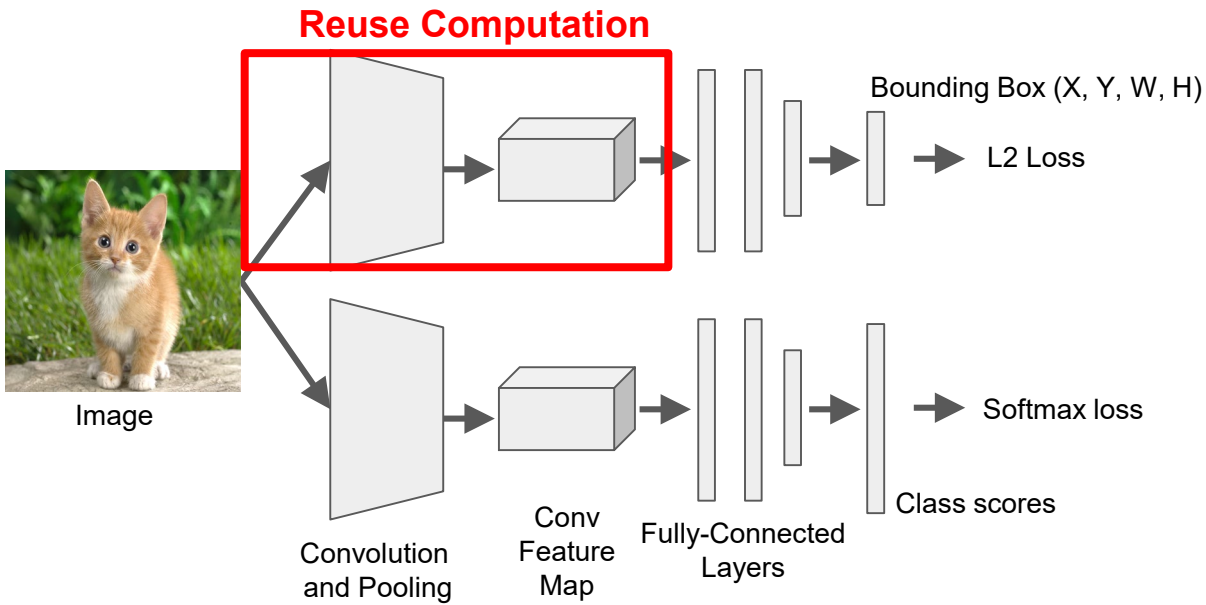
Input: Image
Output: Bounding Box
Evaluation Metric: Intersection Over Union (IoU)



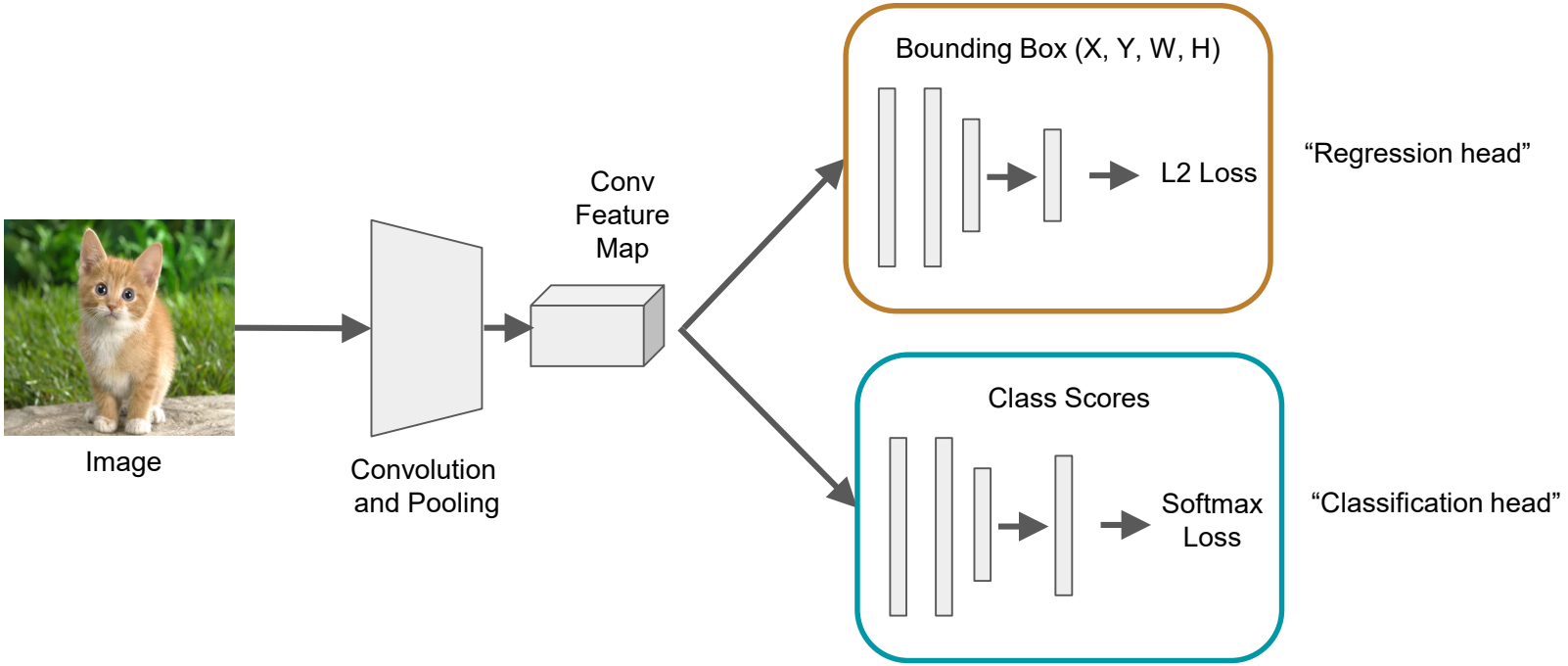
Simple Classification + Localization



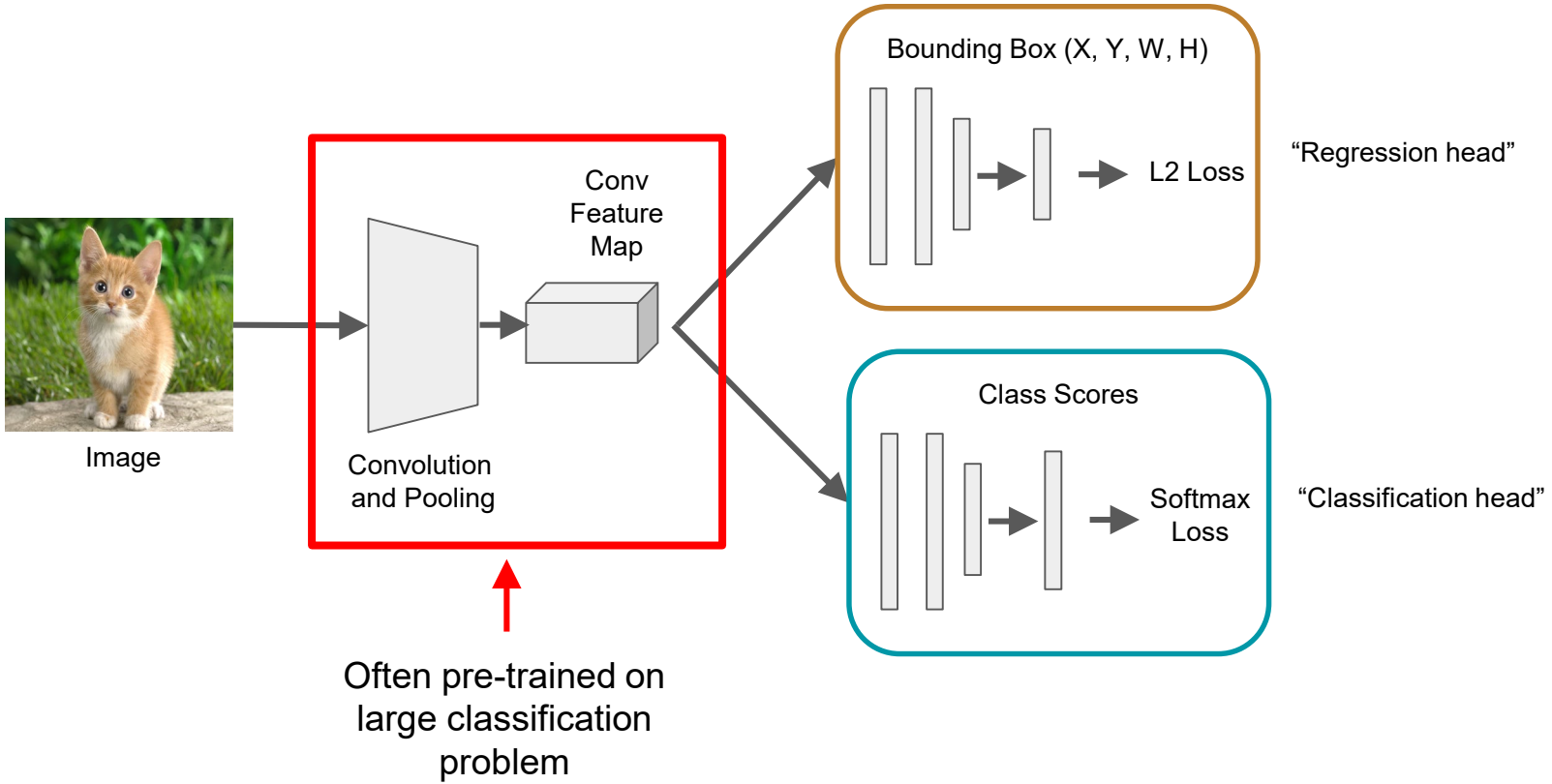
Simple Classification + Localization



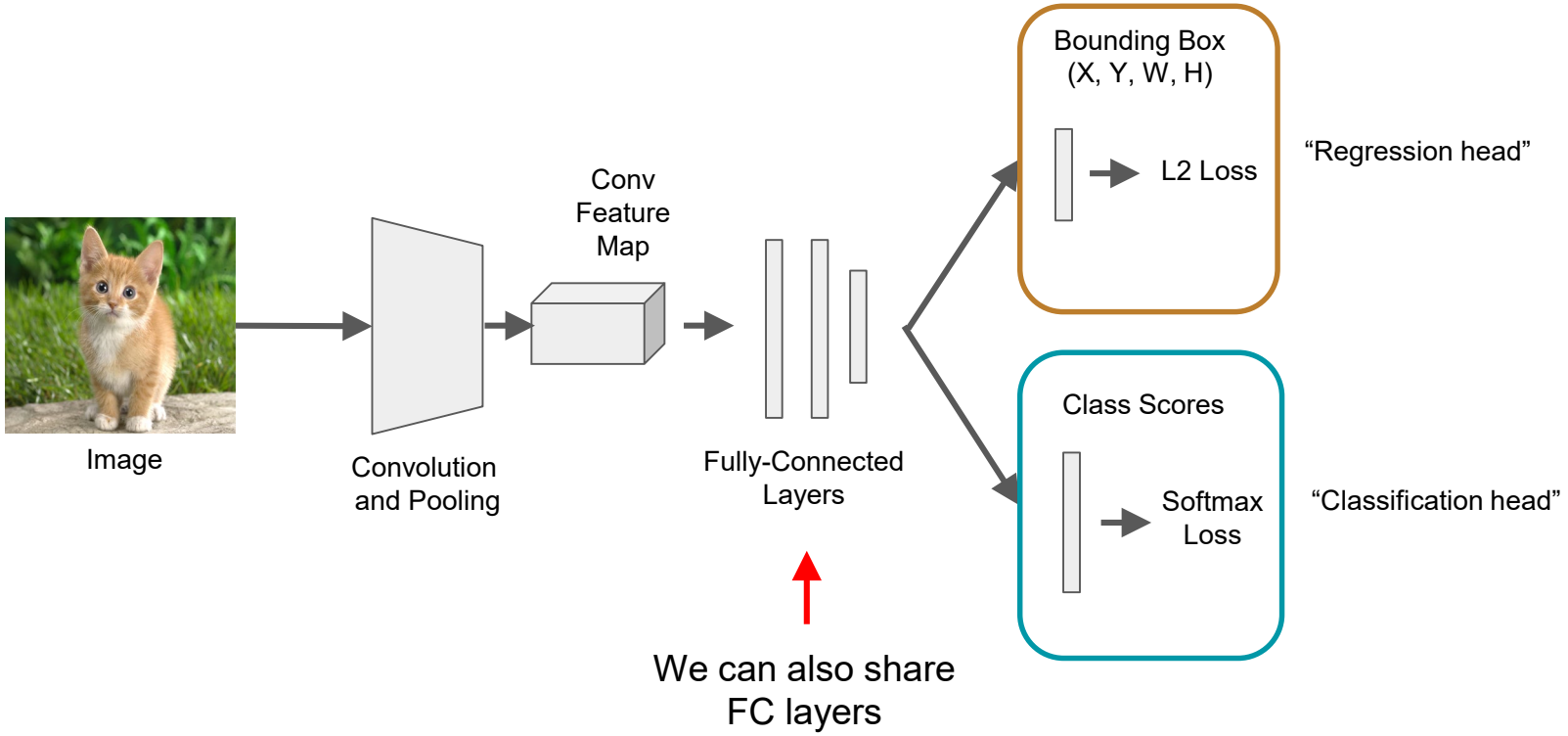
Simple Classification + Localization



Simple Classification + Localization

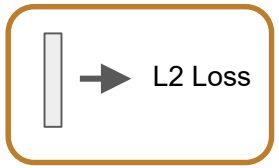


Simple Classification + Localization

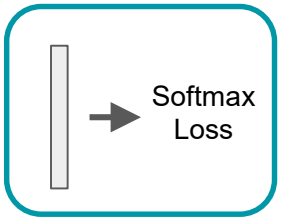


Per-Class vs. Class Agnostic Regression

Class Agnostic

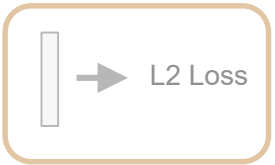


Bounding Box
(X, Y, W, H)
4 Numbers



Class Scores
1 Class selected
(# of Classes output)

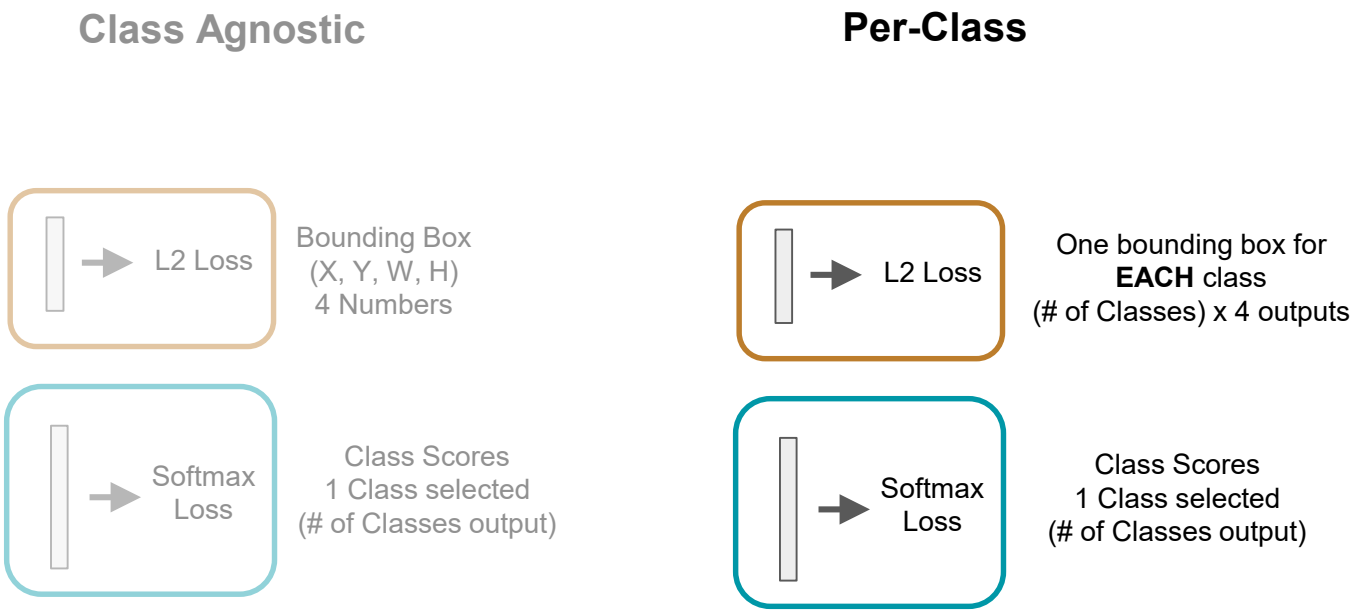
Per-Class



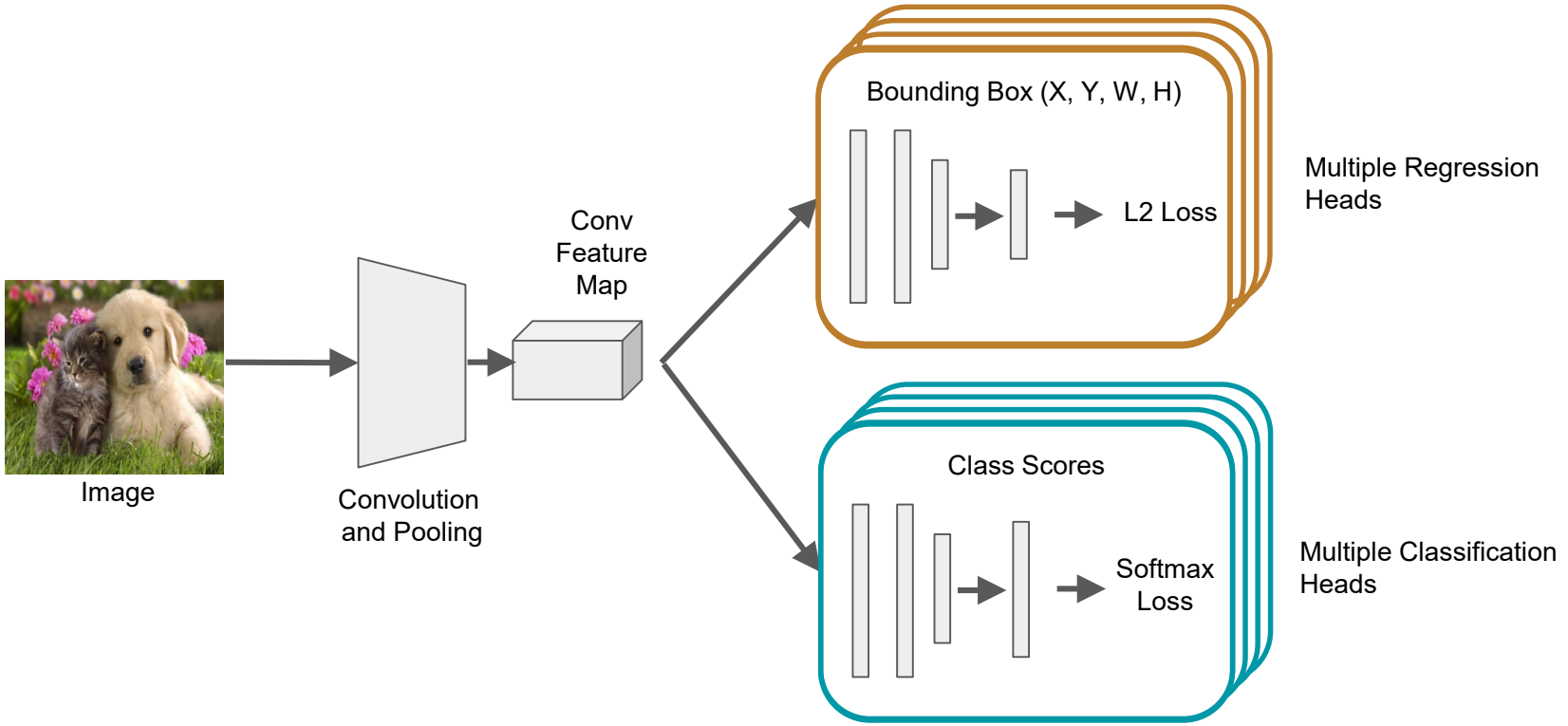
One bounding box for
EACH class
(# of Classes) x 4 outputs

Class Scores
1 Class selected
(# of Classes output)

Per-Class vs. Class Agnostic Regression



Aside: Localizing Multiple Objects



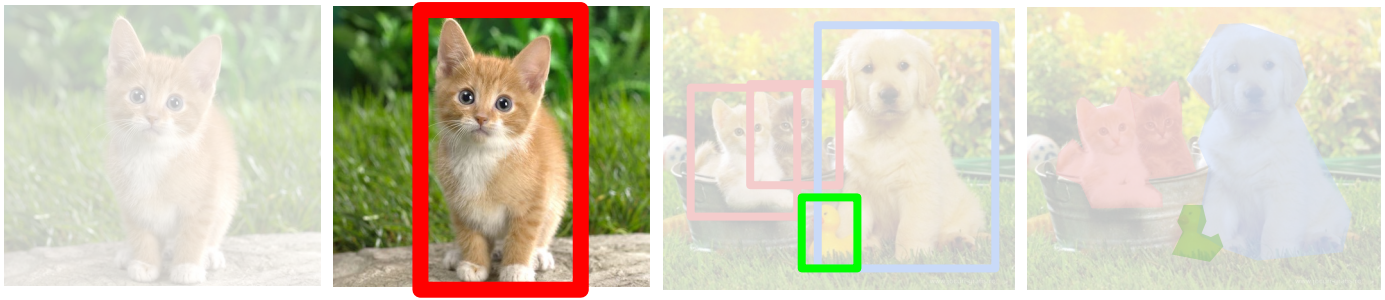
Computer Vision Tasks

Classification

Classification + Localization

Object Detection

Instance Segmentation



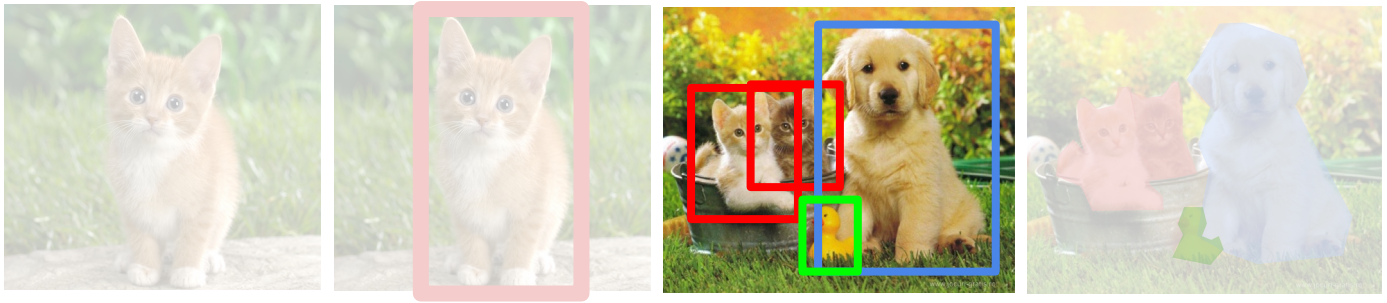
Computer Vision Tasks

Classification

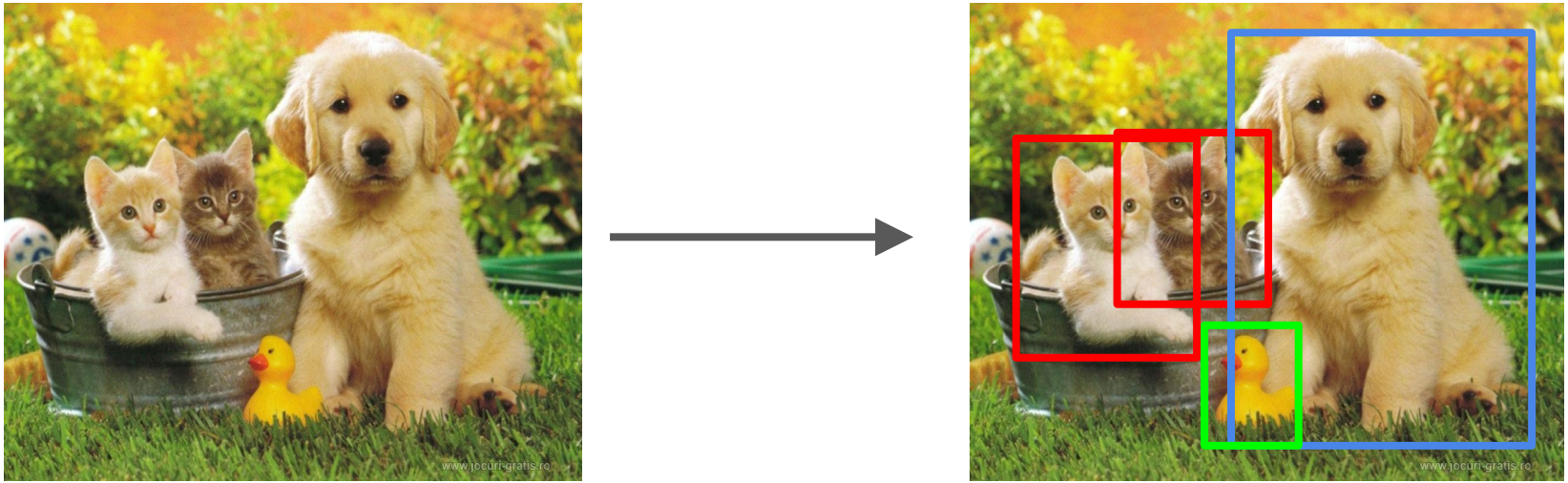
Classification + Localization

Object Detection

Instance Segmentation

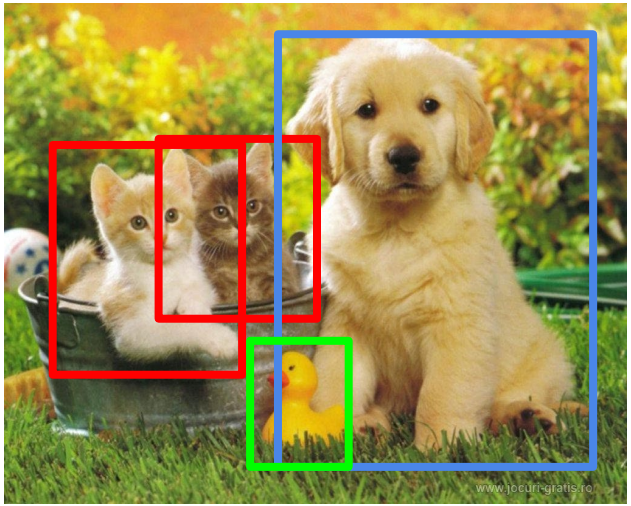


Object Detection



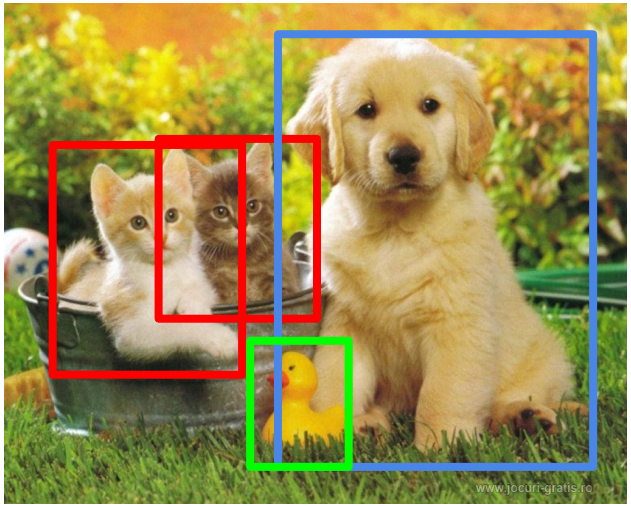
Object Detection

- We use a metric called “mean average precision” (mAP)



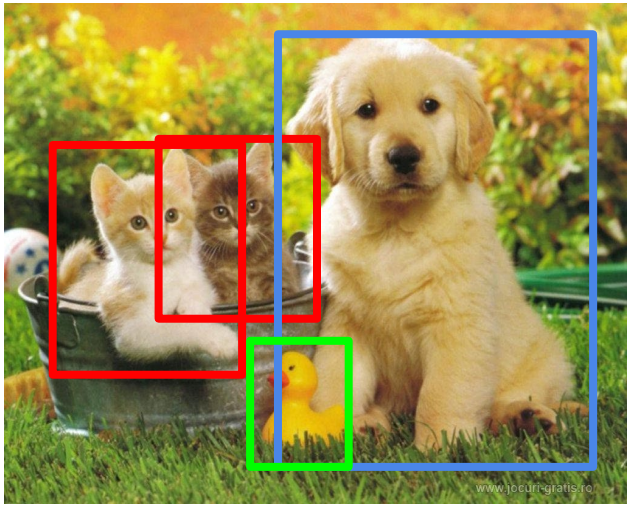
Object Detection

- We use a metric called “mean average precision” (mAP)
- Compute average precision (AP) separately for each class, then average over classes



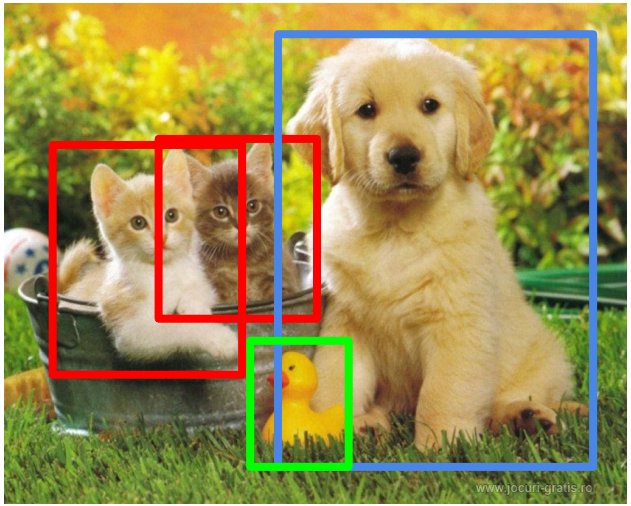
Object Detection

- We use a metric called “mean average precision” (mAP)
- Compute average precision (AP) separately for each class, then average over classes
- A detection is a true positive if it has IoU (Intersection over Union) with a ground-truth box greater than some threshold (usually 0.5) (mAP@0.5)



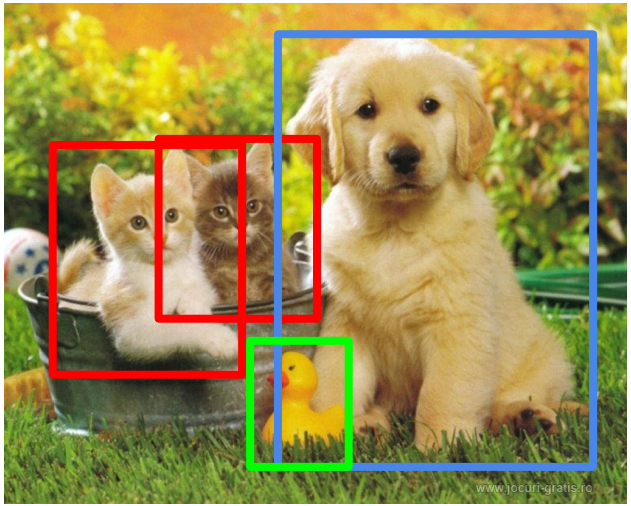
Object Detection

- We use a metric called “mean average precision” (mAP)
- Compute average precision (AP) separately for each class, then average over classes
- A detection is a true positive if it has IoU (Intersection over Union) with a ground-truth box greater than some threshold (usually 0.5) (mAP@0.5)
- Combine all detections from all test images to draw a precision / recall curve for each class; AP is area under the curve



Object Detection

- We use a metric called “mean average precision” (mAP)
- Compute average precision (AP) separately for each class, then average over classes
- A detection is a true positive if it has IoU (Intersection over Union) with a ground-truth box greater than some threshold (usually 0.5) (mAP@0.5)
- Combine all detections from all test images to draw a precision / recall curve for each class; AP is area under the curve
- TL;DR mAP is a number from 0 to 100; high is good



Detection using Regression?

DOG (x, y, w, h)
CAT (x, y, w, h)
CAT (x, y, w, h)
DUCK (x, y, w, h)

= 16 numbers

Detection using Regression?

DOG (x, y, w, h)
CAT (x, y, w, h)
CAT (x, y, w, h)
DUCK (x, y, w, h)

= 16 numbers

DOG (x, y, w, h)
CAT (x, y, w, h)

= 8 numbers

Detection using Regression?

DOG (x, y, w, h)
CAT (x, y, w, h)
CAT (x, y, w, h)
DUCK (x, y, w, h)

= 16 numbers

DOG (x, y, w, h)
CAT (x, y, w, h)

= 8 numbers

CAT, (x, y, w, h)
CAT, (x, y, w, h)
…

= Many Numbers

Detection using Regression?

DOG (x, y, w, h)
CAT (x, y, w, h)
CAT (x, y, w, h)
DUCK (x, y, w, h)

= 16 numbers

DOG (x, y, w, h)
CAT (x, y, w, h)

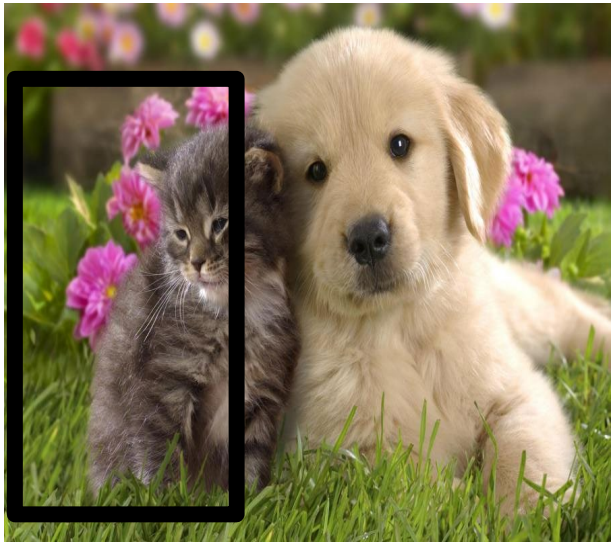
= 8 numbers

CAT, (x, y, w, h)
CAT, (x, y, w, h)
…

= Many Numbers

Need variable sized outputs

Detection using Classification?



CAT? NO

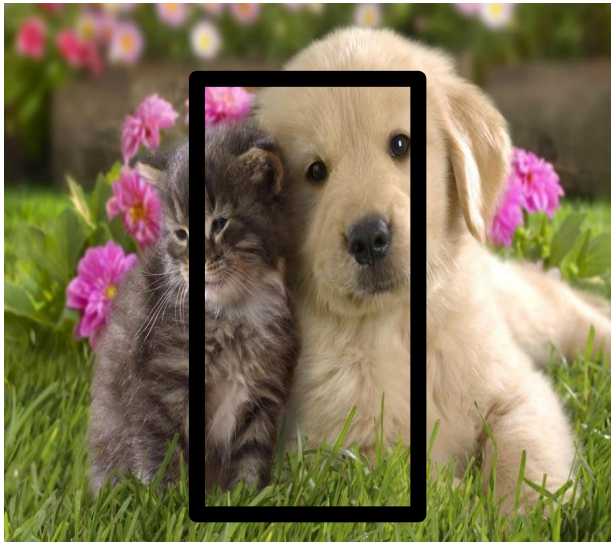
DOG? NO

Detection using Classification?

CAT? YES!

DOG? NO

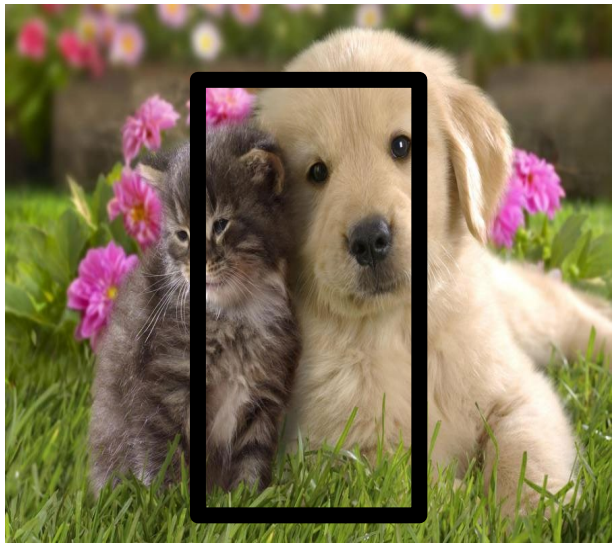
Detection using Classification?



CAT? NO

DOG? NO

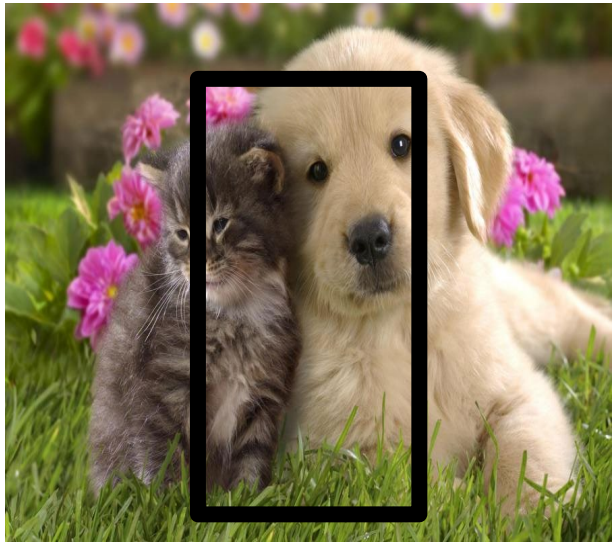
Detection using Classification?



Problem: Need to test many positions and scales

Solution: If your classifier is fast enough, just do it

Detection using Classification?



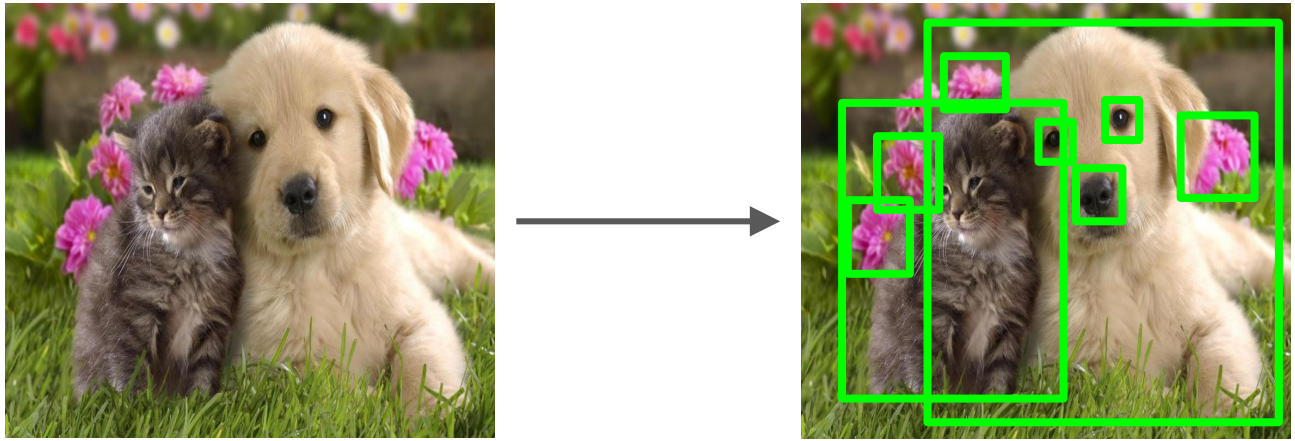
Problem: Need to test many positions and scales

Solution: If your classifier is fast enough, just do it

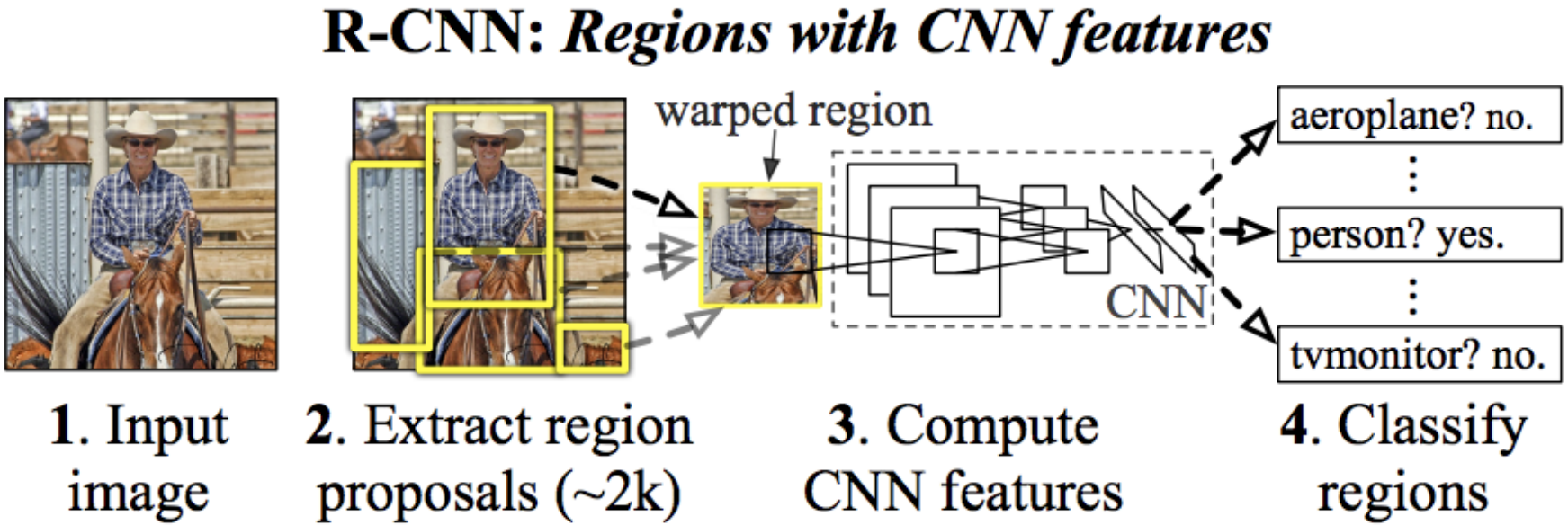
Solution: Only look at promising regions of the image

Region Proposals

- Find “blobby” image regions that are likely to contain objects
- “Class-agnostic” object detector
- Look for “blob-like” regions



Classification + Region Proposals: R-CNN

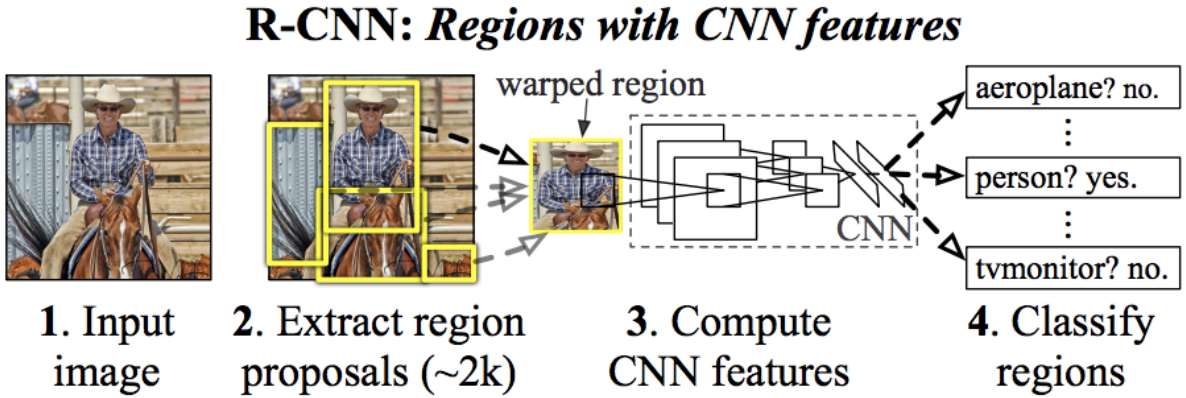


Girshick et. al. "Rich feature hierarchies for accurate object detection and semantic segmentation". CVPR 2014

Classification + Region Proposals: R-CNN

Issues

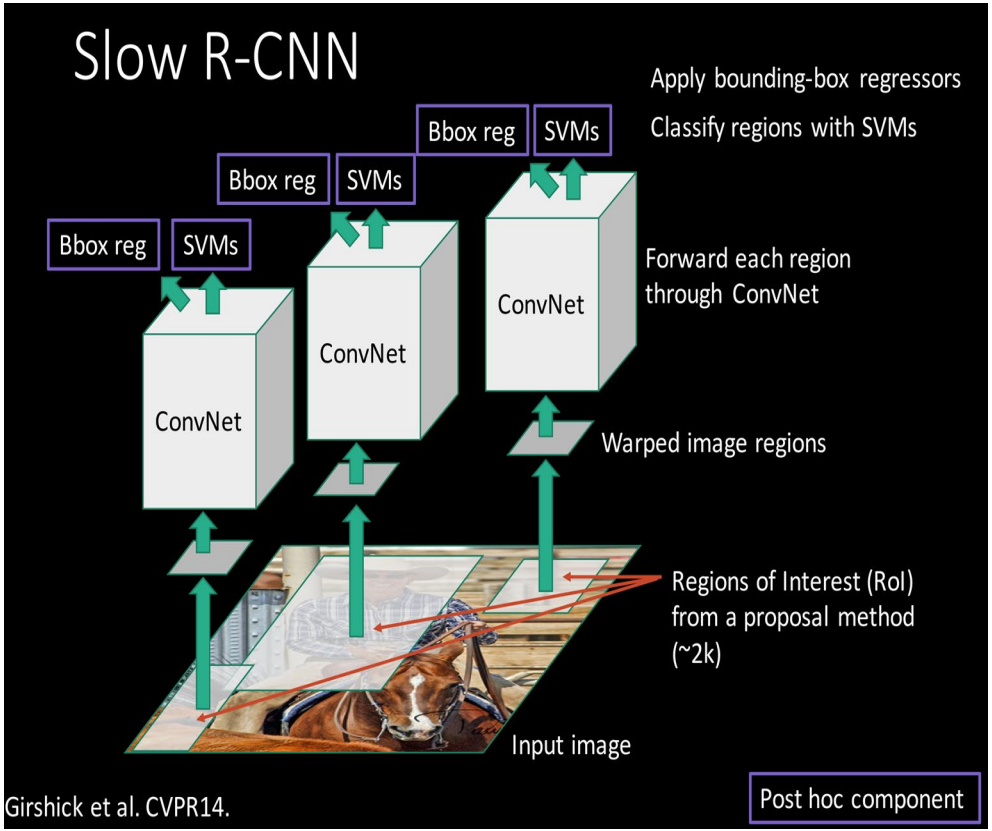
- Finding region proposals can be hard/time consuming
- Classifying each part of the image is time/space consuming



Girshick et. al. "Rich feature hierarchies for accurate object detection and semantic segmentation". CVPR 2014

Fast R-CNN

R-CNN Problem #1:
Slow at test-time due to independent forward passes of the CNN

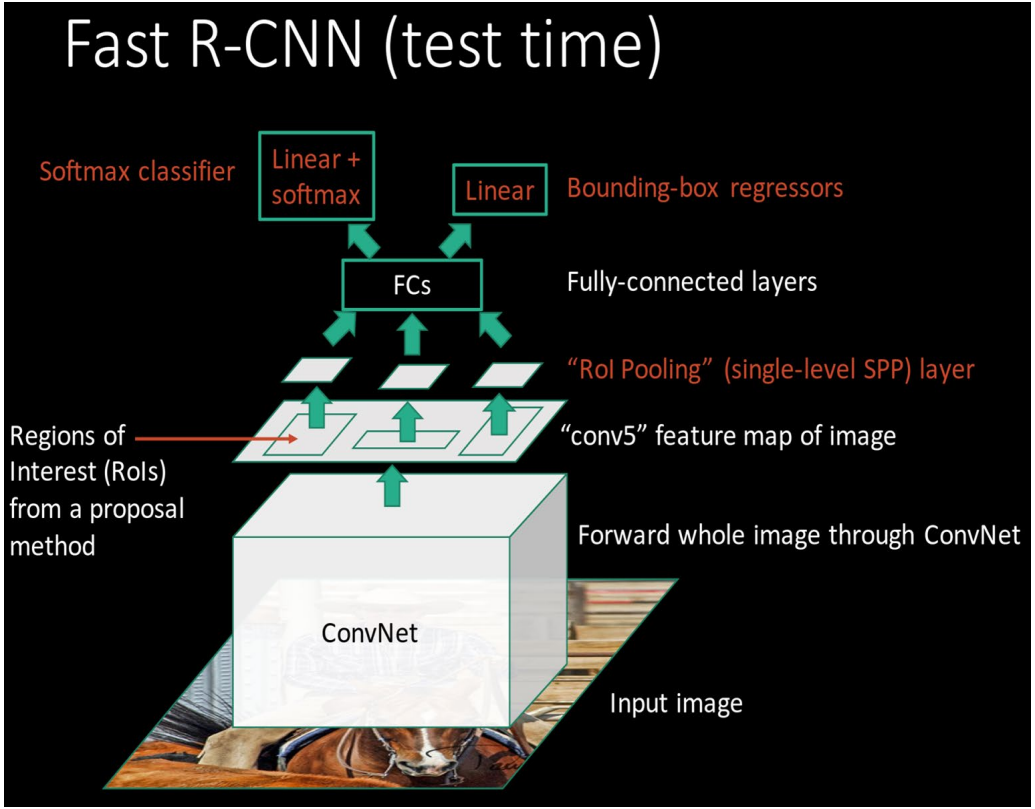


Girschick, "Fast R-CNN", ICCV 2015. Slide credit: Ross Girschick

Fast R-CNN

R-CNN Problem #1:
Slow at test-time due to independent forward passes of the CNN

Solution:
Share computation of convolutional layers between proposals for an image



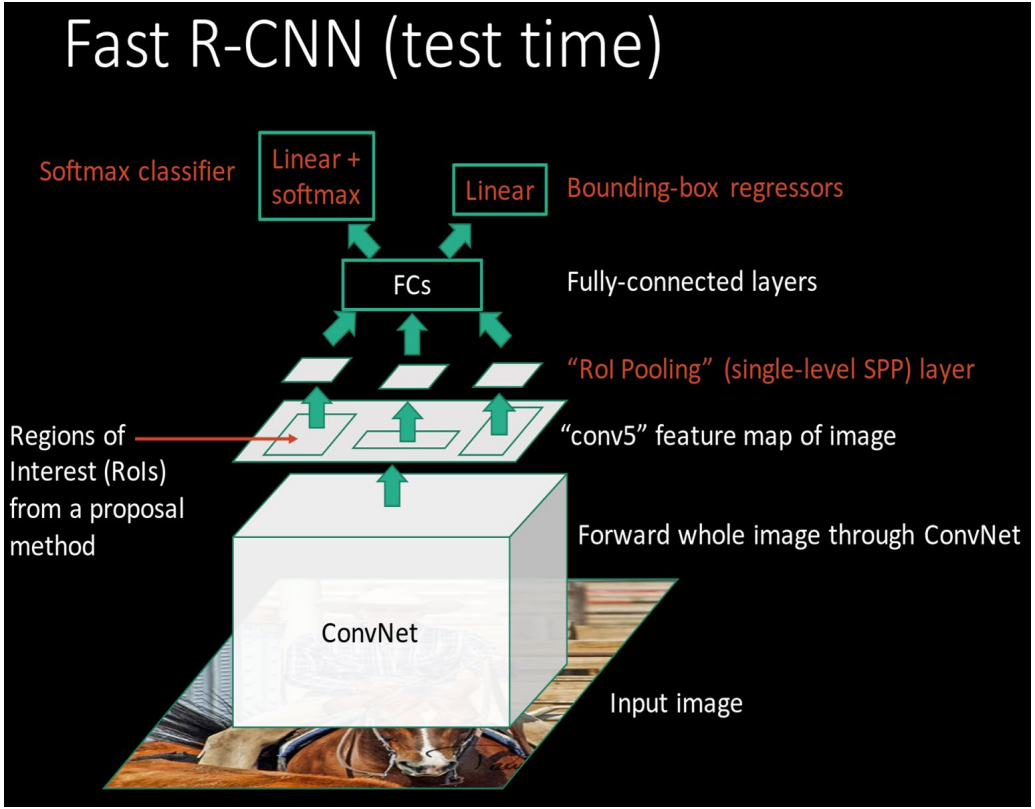
Girschick, "Fast R-CNN", ICCV 2015. Slide credit: Ross Girschick

Fast R-CNN

R-CNN Problem #2:
Post-hoc training: CNN not updated in response to final classifiers and regressors

R-CNN Problem #3:
Complex training pipeline

Solution:
Just train the whole system end-to-end all at once!

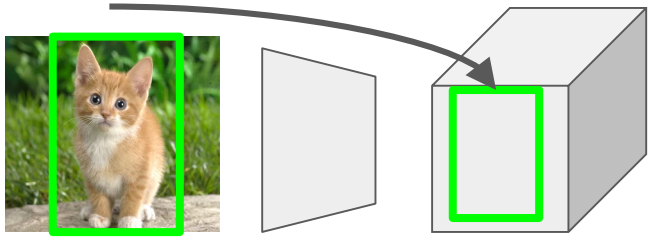


Girschick, "Fast R-CNN", ICCV 2015. Slide credit: Ross Girschick

Fast R-CNN

Convolution and Pooling

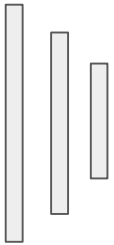
Project region proposal onto conv feature map



Hi-res input image:
3 x 800 x 600
with region proposal

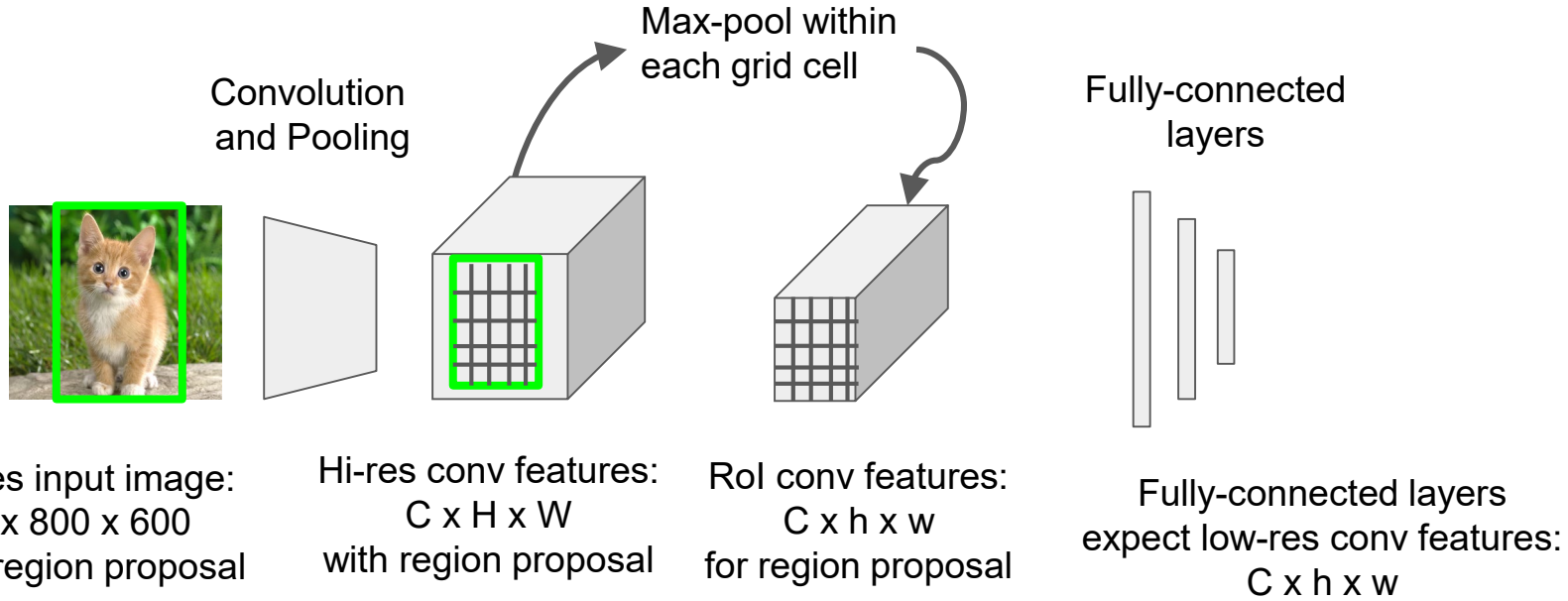
Hi-res conv features:
C x H x W
with region proposal

Fully-connected layers



Problem: Fully-connected layers expect low-res conv features: C x h x w

Fast R-CNN


Convolution
and Pooling
Max-pool within
each grid cell
Fully-connected
layers
Hi-res input image:
3 x 800 x 600
with region proposal
Hi-res conv features:
C x H x W
with region proposal
RoI conv features:
C x h x w
for region proposal
Fully-connected layers
expect low-res conv features:
C x h x w

Fast R-CNN

	R-CNN	Fast R-CNN
Training Time:	84 hours	**9.5 hours**
(Speedup)	1x	**8.8x**
Test time per image	47 seconds	**0.32 seconds**
(Speedup)	1x	**146x**
mAP (VOC 2007)	66.0	**66.9**

Region proposals are still slow!

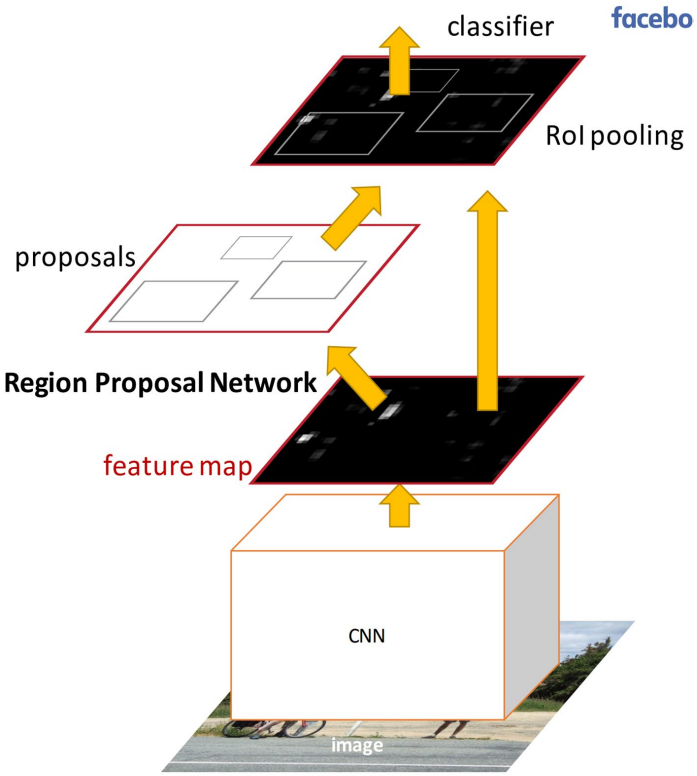
Using VGG-16 CNN on Pascal VOC 2007 dataset

Faster R-CNN

Insert a **Region Proposal Network (RPN)** after the last convolutional layer

RPN trained to produce region proposals directly; no need for external region proposals!

After RPN, use RoI Pooling and an upstream classifier and bbox regressor just like Fast R-CNN



Slide credit: Ross Girschick

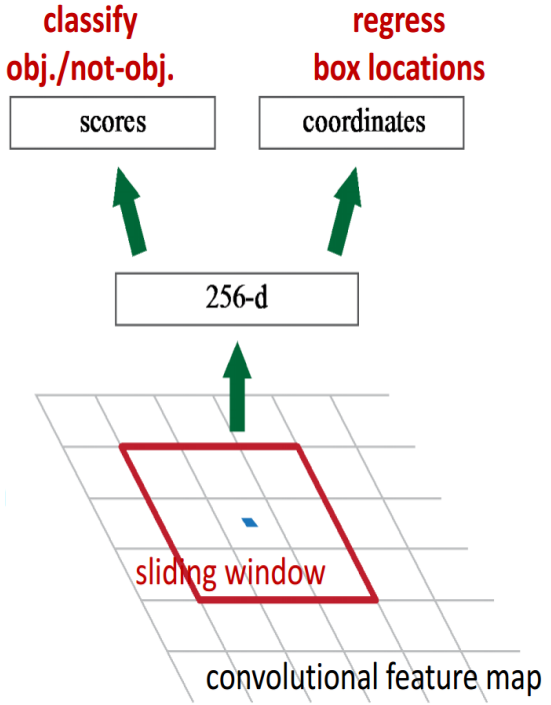
Faster R-CNN

Slide a small window on the feature map

Build a small network for:
- classifying object or not-object, and
- regressing bbox locations

Position of the sliding window provides localization information with reference to the image

Box regression provides finer localization information with reference to this sliding window



Slide credit: Kaiming He

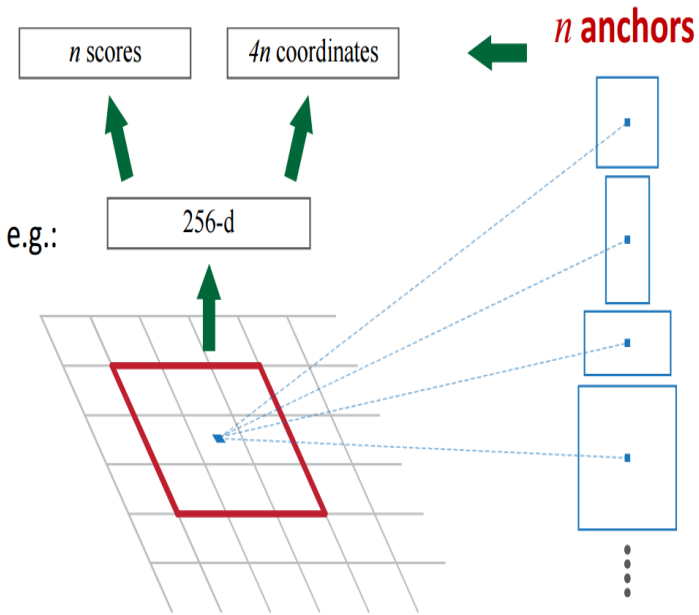
Faster R-CNN

Use **N anchor boxes** at each location

Anchors are **translation invariant**: use the same ones at every location

Regression gives offsets from anchor boxes

Classification gives the probability that each (regressed) anchor shows an object



Slide credit: Kaiming He

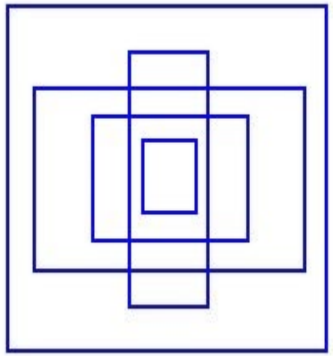
Faster R-CNN

Use **N anchor boxes** at each location

Anchors are **translation invariant**: use the same ones at every location

Regression gives offsets from anchor boxes

Classification gives the probability that each (regressed) anchor shows an object



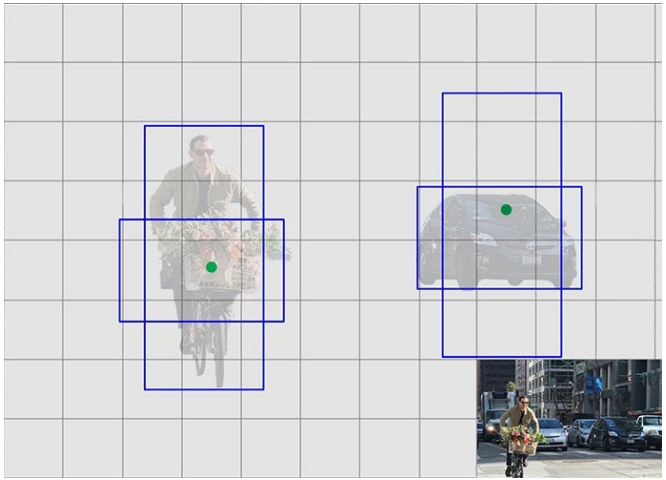


Image Credit: https://medium.com/@andersasac/anchor-boxes-the-key-to-quality-object-detection-ddf9d612d4f9

Faster R-CNN

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**66.9**

Slide credit: Kaiming He

Recap

Semantic Segmentation

- Classify at a pixel level
- Ignores “objectness” focuses on semantics
- Mask-RCNN/UNet for pixel-level semantics

Recap

Semantic Segmentation

- Classify at a pixel level
- Ignores “objectness” focuses on semantics
- Mask-RCNN/UNet for pixel-level semantics

Localization:

- Find a fixed number of objects (one or many)
- L2 regression from CNN features to box coordinates
- Much simpler than detection; consider it for your projects!
- Overfeat: Regression + efficient sliding window with FC -> conv conversion
- Deeper networks do better

Recap

Semantic Segmentation
- Classify at a pixel level
- Ignores “objectness” focuses on semantics
- Mask-RCNN/UNet for pixel-level semantics

Localization:
- Find a fixed number of objects (one or many)
- L2 regression from CNN features to box coordinates
- Much simpler than detection; consider it for your projects!
- Overfeat: Regression + efficient sliding window with FC -> conv conversion
- Deeper networks do better

Object Detection:
- Find a variable number of objects by classifying image regions
- Before CNNs: dense multiscale sliding window (HoG, DPM)
- Avoid dense sliding window with region proposals
- R-CNN: Selective Search + CNN classification / regression
- Fast R-CNN: Swap order of convolutions and region extraction
- Faster R-CNN: Compute region proposals within the network
- Deeper networks do better