Data 188: Introduction to Deep Learning

Computer Vision

Speaker: Eric Kim
Lecture 12 (Week 06)
2026-02-26, Spring 2026. UC Berkeley.

Acknowledgement: slides are based on CMU's Deep Learning Systems course (10-414/714, Fall 2025). Thanks to: Prof. Zico Kolter, Prof. Tianqi Chen, Prof. Tim Dettmers.

Rev02: 2026-03-07

Announcements

- HW2 is out!
- HW1 Course Survey in Gradescope
- Midterm in 2 weeks!
  - (Action Needed) Midterm scheduling form (on Edstem)
    - Please fill this form out!
    - Due: Friday February 27th, 2026, 11:59 PM PST
  - **DSP students with exam accommodations**: we emailed you a separate Google form, please fill this out ASAP!
  - "Midterm Study Guide" released (see Edstem post)

Outline

Elements of practical convolutions

Convolutional networks (Resnets)

Interpreting feature maps

Differentiating convolutions

Computer Vision

Outline

Elements of practical convolutions

Convolutional networks (Resnets)

Interpreting feature maps

Differentiating convolutions

Computer Vision

Conv2d layer

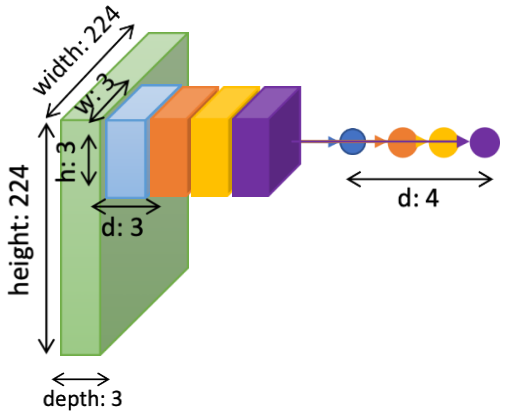
A Conv2d layer applies a **series of learned 2D filters** to a multichannel 2D spatial feature map, producing a multichannel 2D spatial feature map.

Input: $[\text{batchsize}, C_{\text{in}}, \text{h}_{\text{in}}, \text{w}_{\text{in}}]$

Output: $[\text{batchsize}, C_{\text{out}}, h_{\text{out}}, w_{\text{out}}]$

Layer trainable parameters:

Filter weights ("filter bank"): $[\text{C}_{\text{out}}, \text{C}_{\text{in}}, \text{k}, \text{k}]$
Bias parameters: $[\text{C}_{\text{out}}]$



Tip: number of filters is C_{out}

k is kernel size. Determines spatial extent of filter.

Bias is a learned offset applied to the filter outputs, eg broadcasted add.

Forward pass calculates: $conv2d(Z_{in}, G) = Z_{out} + bias$

Where: $Z_{out}[i,:,:] = Z_{in} \star G[i,:,:,:]$

Z_{in} has shape=$[B, C_{in}, h_{in}, w_{in}]$
G is filter weights, shape= $[\text{C}_{\text{out}}, \text{C}_{\text{in}}, \text{k}, \text{k}]$
Z_{out} has shape=$[B, C_{out}, h_{out}, w_{out}]$
bias has shape=$[C_{out}]$

Note: here, I'm defining a "3d conv" to be "sum the elemwise prod of Z_{in} and $G[i,:,:]$, for each spatial location in Z_{in}"

Sweep (convolve) each filter $G[i,:,:]$ to our input spatial feature map Z_{in}, and store the filter response to $Z_{out}[i,:,:]$

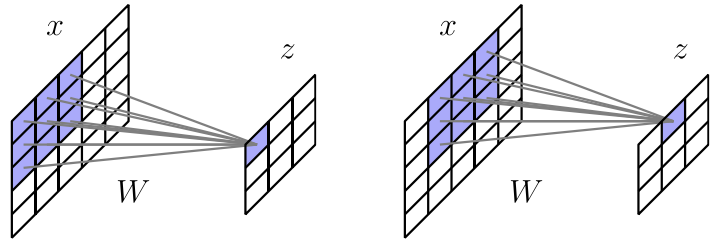
bias broadcast: each output channel in C_{out} has a separate scalar offset applied to the entire spatial window.

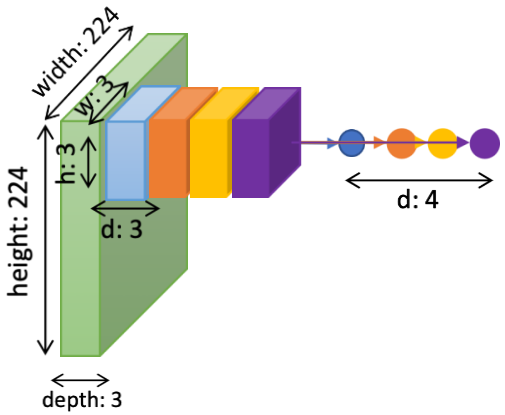
Conv2d: additional parameters

torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')

So far, we've only considered stride=1, padding=0 (and ignored "dilation", "groups").

We have additional control over how the convolution is done, via these parameters: **stride, padding, dilation, groups**.





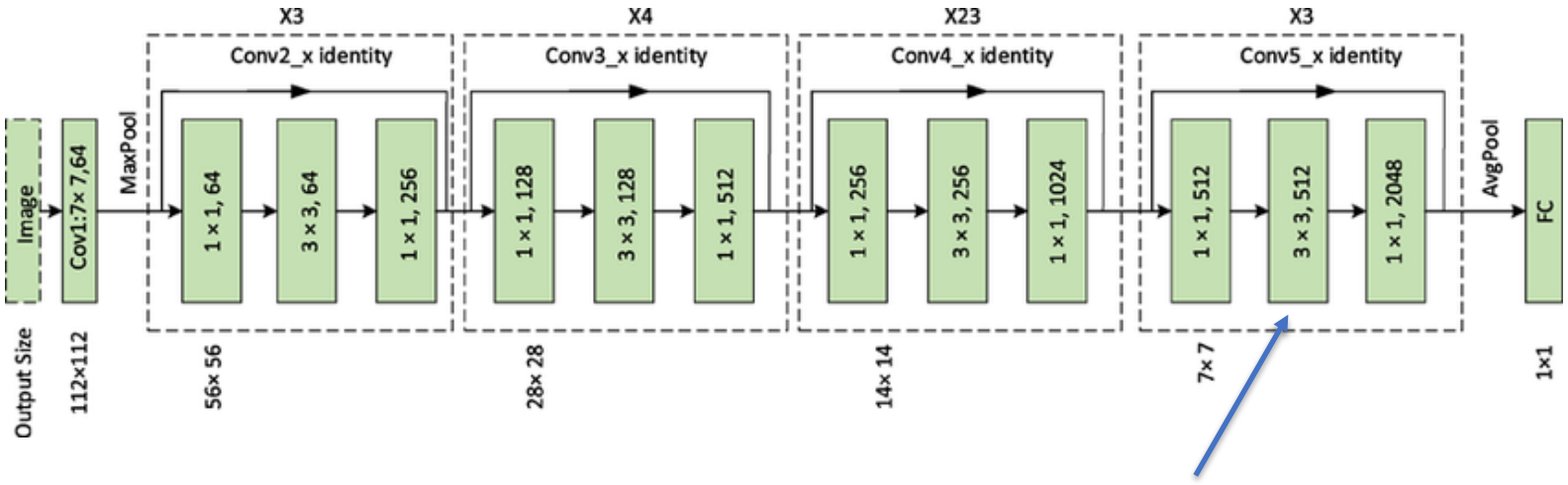
Pictured: input has shape=$[1, 3, 224, 224]$.
Filter weights ("bank") shape=$[C_{out} = 4, C_{in} = 3, 3, 3]$
Stride=1, padding=0.

https://docs.pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

Grouped Convolutions

Challenge: for large numbers of input/output channels, filters can have a large number of parameters, leading to (possible) overfitting + slow computation

Ex: in ResNet101, the final conv layers ("conv5") have 512 to 2048 channels.



Z_{in} shape: [B, 512, 7, 7]
Filter bank shape*: [512, 512, 3, 3]

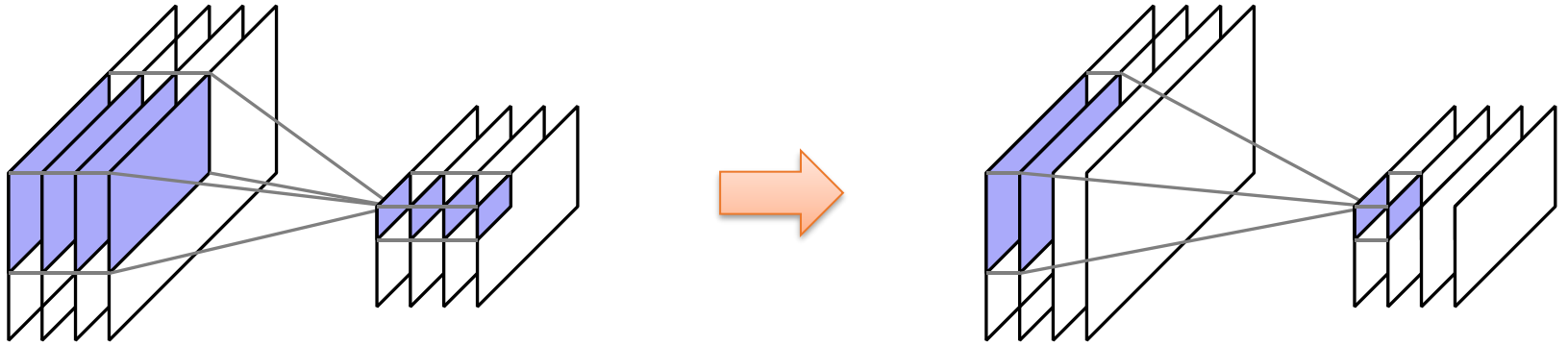
https://www.researchgate.net/figure/Typical-architecture-of-the-Resnet101_fig2_372853599

*If done naively. We'll see later that ResNet reduces C_{in} (via 1x1 convs) to avoid costly computation

Grouped Convolutions

Challenge: for large numbers of input/output channels, filters can have a large number of parameters, leading to (possible) overfitting + slow computation

Solution: Group together channels, so that groups of channels in output only depend on corresponding groups of channels in input (equivalently, enforce filter weight matrices to be block-diagonal)



Groups=1 (default)

Filter bank shape=[C_out, C_in, k, k]

Groups=2

Filter bank shape=[C_out, C_in / 2, k, k]

Note: we group both input and output channels. Thus, C_{in} and C_{out} must both be divisible by `groups`

Ex: if you have $C_{in} = 512$, $C_{out} = 256$, groups=2, then you have:
2 groups of C_{in} (each with 512/2=256 channels)
2 groups of C_{out} (each with 256/2=128 channels)

Grouped Conv: Example

Suppose Z_{in} has shape=[B, 64, 7, 7]

Conv2d(C_{in}=64, C_{out}=32, **groups=4**, k=3, padding=1)

Filter bank G has shape=[32, 16, 3, 3]

Z_{out} has shape=[B, 32, 7, 7], calculated as:

Input channel group size: $\frac{C_{in}}{groups} = \frac{64}{4} = 16$

Output channel group size: $\frac{C_{out}}{groups} = \frac{32}{4} = 8$

Output group 1 (Input group 1):

$$\begin{aligned} Z_{out}[:,0,:,:] &= Z_{in}[:,0{:}16,:,:] \star G[0,:,:,:] \\ Z_{out}[:,1,:,:] &= Z_{in}[:,0{:}16,:,:] \star G[1,:,:,:] \\ &\dots \\ Z_{out}[:,7,:,:] &= Z_{in}[:,0{:}16,:,:] \star G[7,:,:,:] \end{aligned}$$

Output group 2 (Input group 2):

$$\begin{aligned} Z_{out}[:,8,:,:] &= Z_{in}[:,16{:}32,:,:] \star G[8,:,:,:] \\ Z_{out}[:,9:,:] &= Z_{in}[:,16{:}32,:,:] \star G[9,:,:,:] \\ &\dots \\ Z_{out}[:,15,:,:] &= Z_{in}[:,16{:}32,:,:] \star G[15,:,:,:] \end{aligned}$$

Output group 3 (Input group 3):

$$\begin{aligned} Z_{out}[:,16,:,:] &= Z_{in}[:,32{:}48,:,:] \star G[16,:,:,:] \\ Z_{out}[:,17,:,:] &= Z_{in}[:,32{:}48,:,:] \star G[17,:,:,:] \\ &\dots \\ Z_{out}[:,23:,:] &= Z_{in}[:,32{:}48,:,:] \star G[23,:,:,:] \end{aligned}$$

Output group 4 (Input group 4):

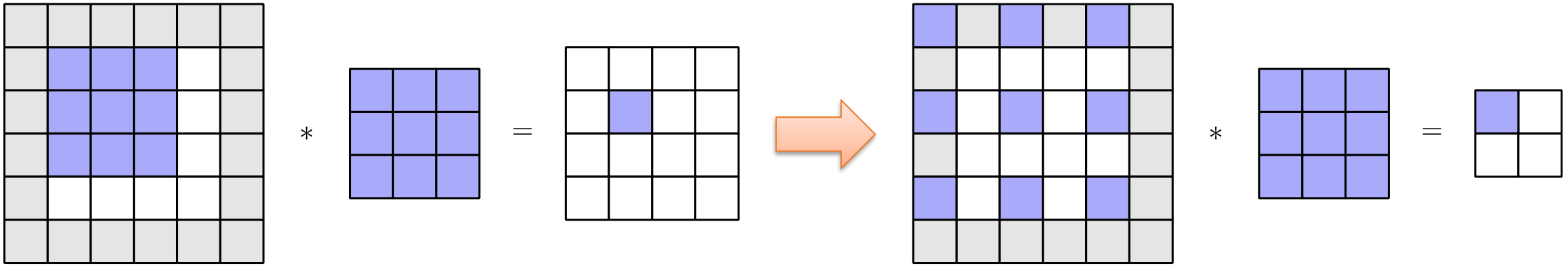
$$\begin{aligned} Z_{out}[:,24,:,:] &= Z_{in}[:,48{:}64,:,:] \star G[24,:,:,:] \\ Z_{out}[:,25,:,:] &= Z_{in}[:,48{:}64,:,:] \star G[25,:,:,:] \\ &\dots \\ Z_{out}[:,31,:,:] &= Z_{in}[:,48{:}64,:,:] \star G[31,:,:,:] \end{aligned}$$

Dilations

aka "spatial extent"

Challenge: Convolutions each have a relatively small receptive field size

Solution: *Dilate* (spread out) convolution filter, so that it covers more of the image. Note that getting an image of the same size again requires adding more padding



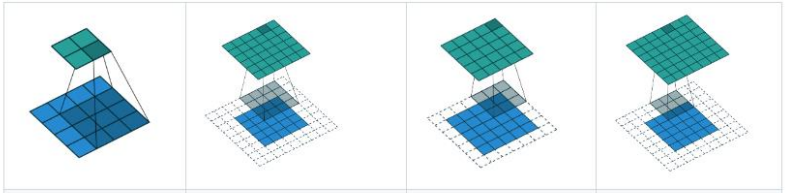
dilation=1 (eg no dilation)

dilation=2

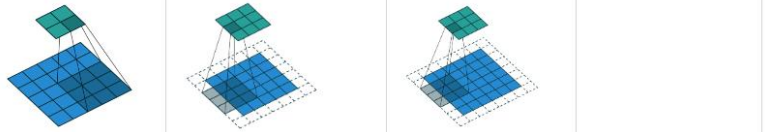
Conv2d visualizations (more)

Convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.



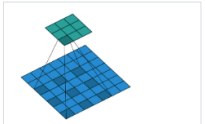
No padding, no strides | Arbitrary padding, no strides | Half padding, no strides | Full padding, no strides



No padding, strides | Padding, strides | Padding, strides (odd)

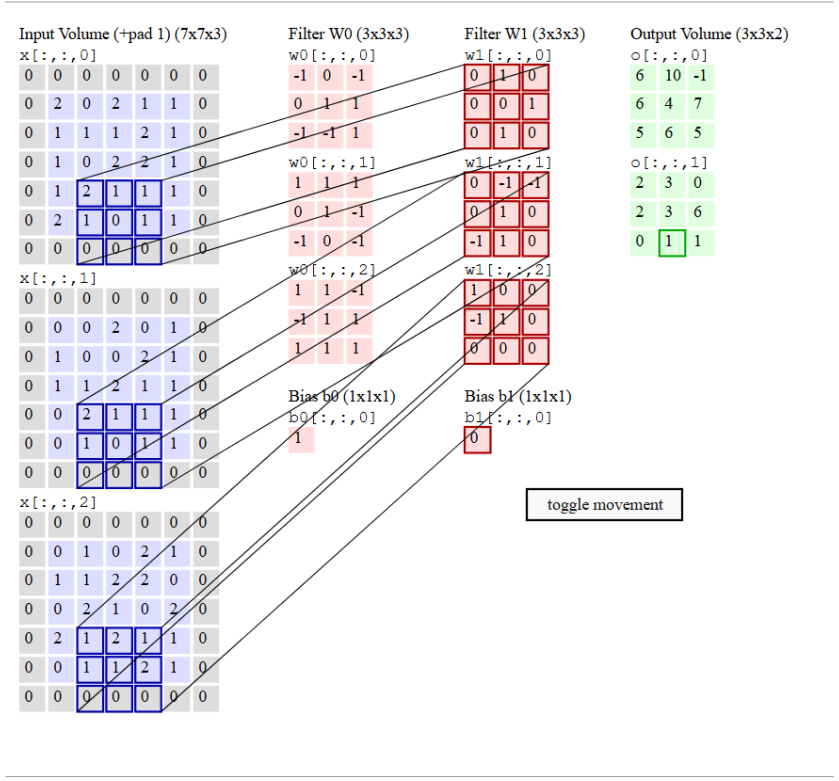
Dilated convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.



No padding, no stride, dilation

https://github.com/vdumoulin/conv_arithmetic/blob/master/README.md



https://cs231n.github.io/convolutional-networks/

Outline

Elements of practical convolutions

Convolutional networks (Resnets)

Interpreting feature maps

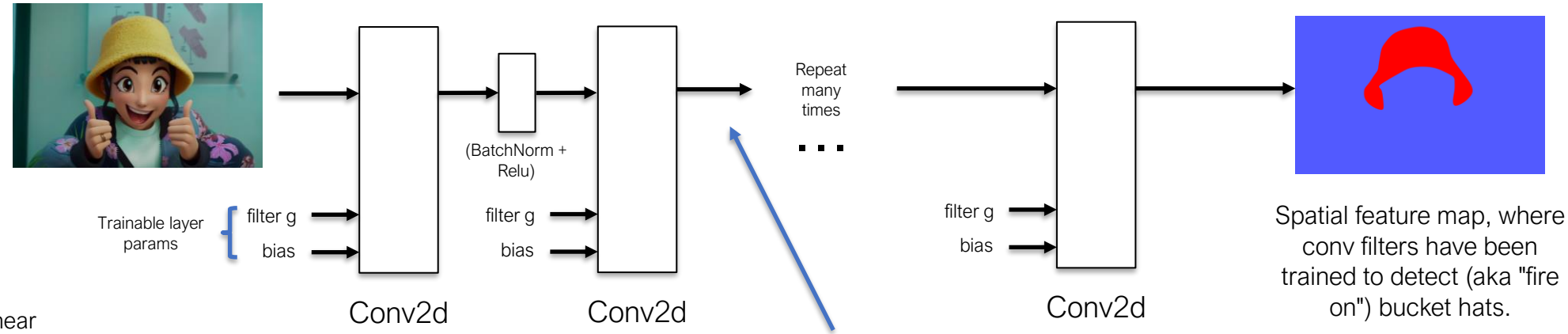
Differentiating convolutions

Computer Vision

Convolutional networks ("Convnets")

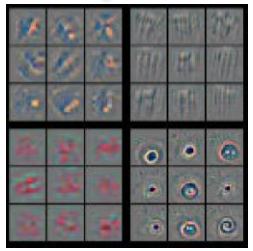
Good news: conv2d is a differentiable operation with respect to the input and the filter/bias parameters. Notably, conv2d.gradient() can be implemented efficiently.

Means we can train models with conv2d operations!



Similar to MLP's Linear layers: each Conv2d has its own learnable filter/bias parameters.

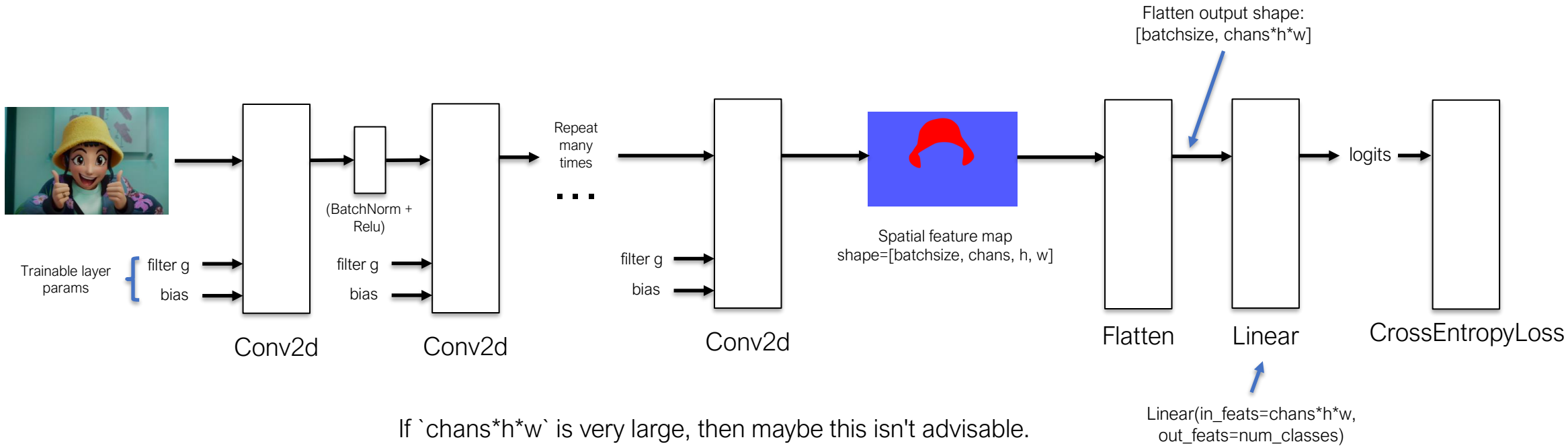
Each Conv2d layer outputs a **spatial** feature map, with shape=[batchsize, chans, h, w]



https://arxiv.org/abs/1311.2901

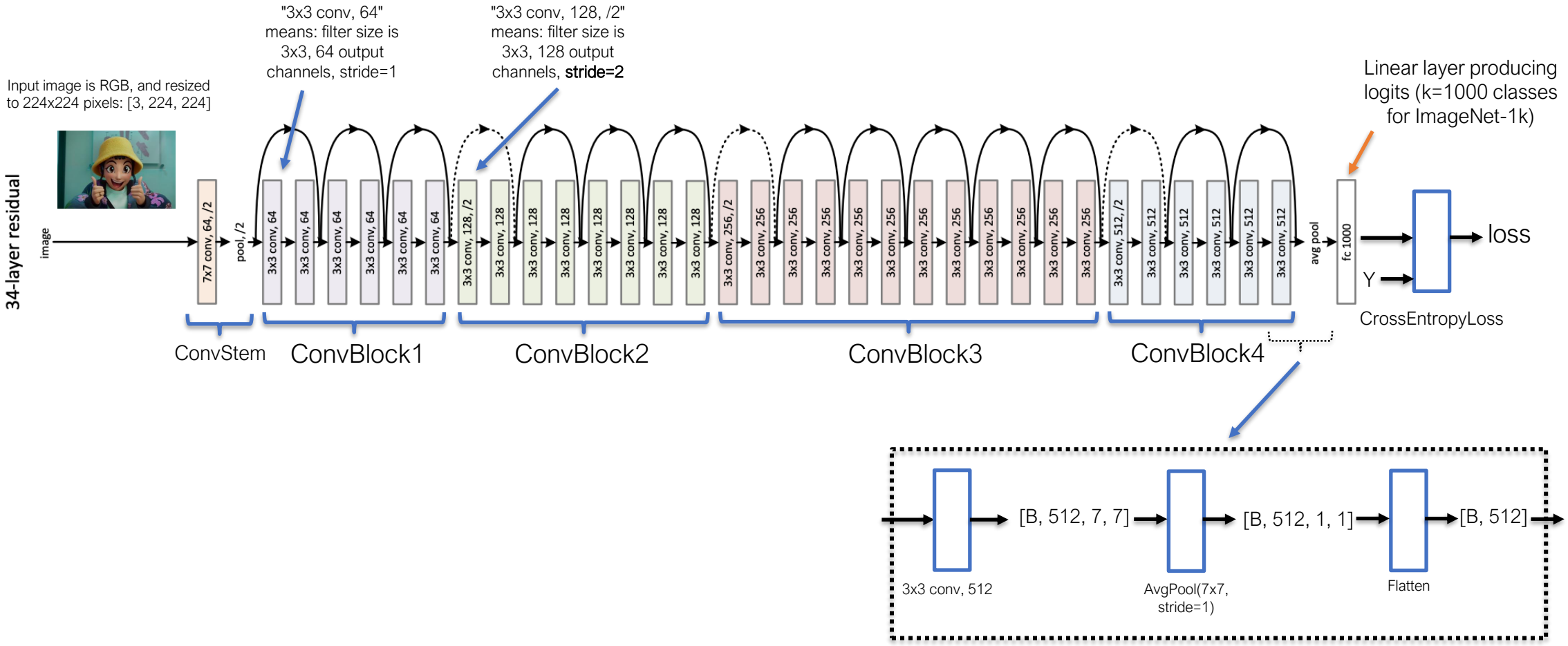
Convolution layer: classification

A simple image classification convnet: flatten the final spatial feature map, and attach a final Linear layer to produce the predicted logits.



If `chans*h*w` is very large, then maybe this isn't advisable.
Ex: chans=512, h=w=7, then this is 512*7*7=25088, which is rather large when it comes to linear layers.
(next slide we'll see a more compact way)

ResNet(s): a (very) popular convnet

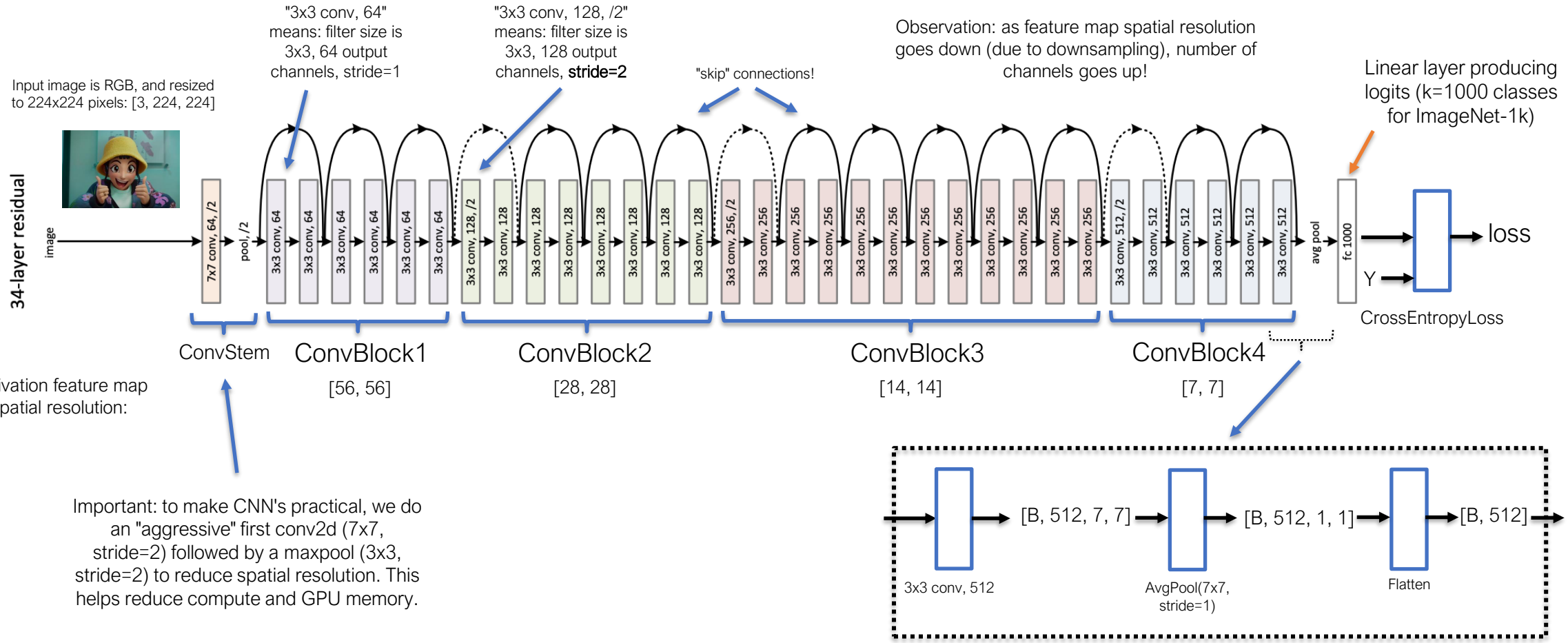


Intuition: final AvgPool summarizes the information in each spatial feature map prior to the linear classifier.
Idea: perhaps each of the 512 channels contains the "visual concepts" present in the image?

Resnet paper, "Deep Residual Learning for Image Recognition".

(optional) if you're curious, here is the resnet model code in pytorch:
https://github.com/huggingface/transformers/blob/main/src/transformers/models/resnet/modeling_resnet.py
Huggingface page: https://huggingface.co/microsoft/resnet-50
Config for resnet50: https://huggingface.co/microsoft/resnet-50/blob/main/config.json

ResNets: details



Intuition: final AvgPool summarizes the information in each spatial feature map prior to the linear classifier.
Idea: perhaps each of the 512 channels contains the "visual concepts" present in the image?

Resnet paper, "Deep Residual Learning for Image Recognition".

(optional) if you're curious, here is the resnet model code in pytorch:
https://github.com/huggingface/transformers/blob/main/src/transformers/models/resnet/modeling_resnet.py
Huggingface page: https://huggingface.co/microsoft/resnet-50
Config for resnet50: https://huggingface.co/microsoft/resnet-50/blob/main/config.json

Residual/Skip connections

To train very deep convnets (with 34, 50, 101, or even 151 conv layers), ResNets added "residual connections" ("skip" connections)

This helped mitigate the "vanishing gradient" problem. (defined next slide)

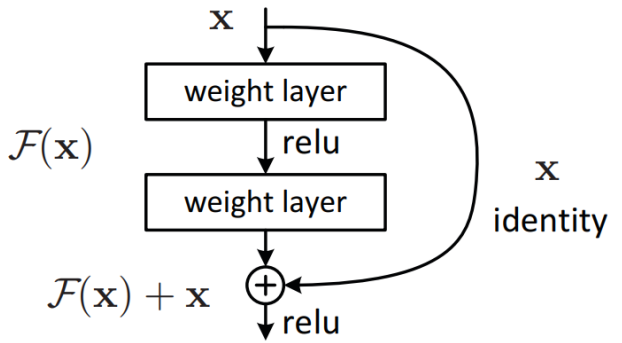
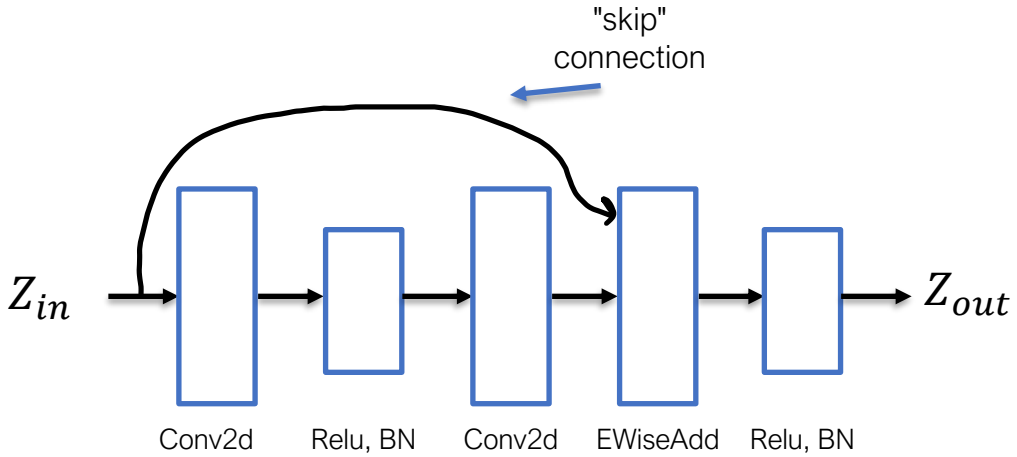


Figure 2. Residual learning: a building block.

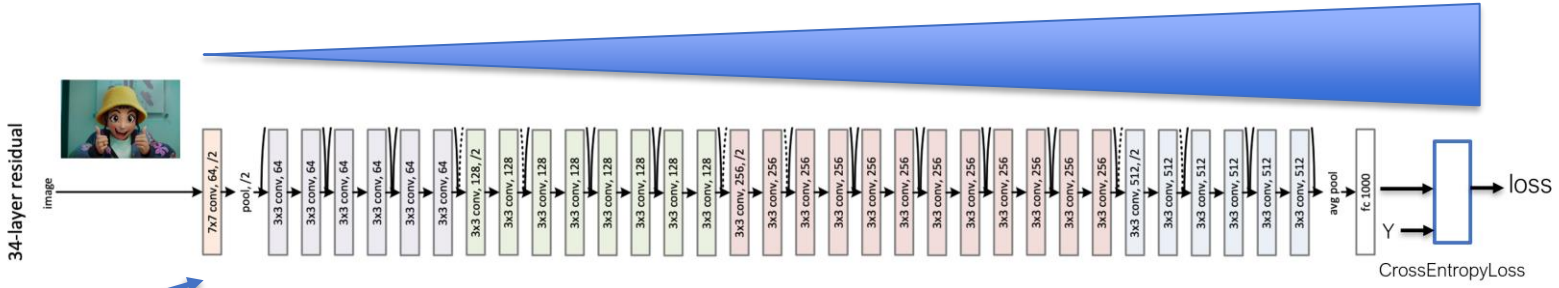


Vanishing gradient

Often, when training deep models, the gradient signal (originating from your loss) will progressively get "weaker" as it passes through each layer.

Result: early layers can receive weak (or even 0 if underflow!) gradients, significantly slowing down training.

Gradient magnitude at layer i: $\|\nabla_{\theta_i} f(\theta)\|$



$$\overline{v_1} = \frac{\partial loss}{\partial v_1} = \frac{\partial loss}{\partial v_{34}} \frac{\partial v_{34}}{\partial v_{33}} \frac{\partial v_{33}}{\partial v_{32}} \frac{\partial v_{32}}{\partial v_{31}} \frac{\partial v_{31}}{\partial v_{30}} \cdots \frac{\partial v_3}{\partial v_2} \frac{\partial v_2}{\partial v_1}$$

Ex: multiplying 0.9*0.9*...*0.9 = 0.9^34 = 0.0278
0.5^34 = 5.82e-11

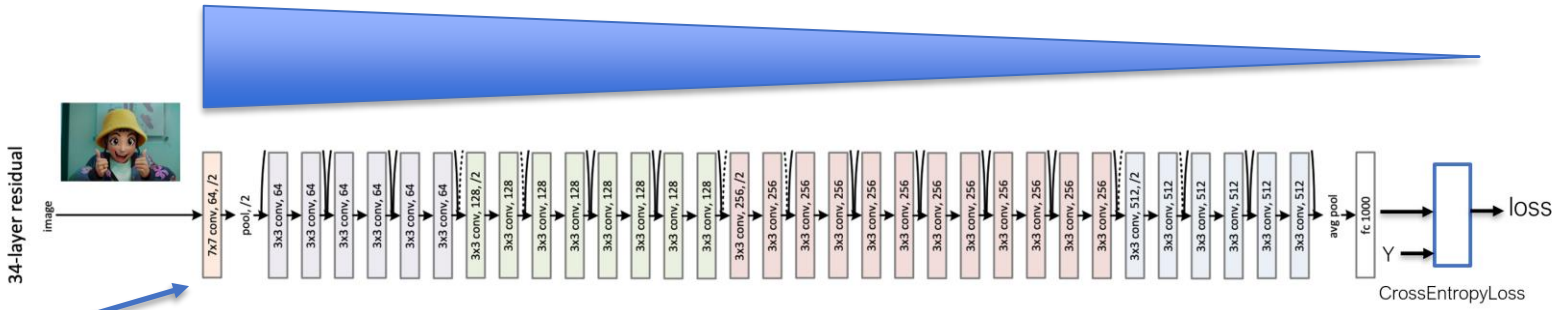
Intuition: when calculating adjoints $\overline{v_i}$ for early layers, we multiply terms proportional to model depth.
Recall: multiplying small (<1.0) values together yields a smaller number: 0.1*0.1 = 0.01

Related: exploding gradients

Related problem: gradients growing out of control! Results in training instability, numerical overflow, NaN/Inf losses.

Partially mitigated by: careful parameter initialization, activation normalization, Adam (sort of), decreasing learning rate, and a few other tricks (ex: gradient clipping)

Gradient magnitude at layer i: $\|\nabla_{\theta_i} f(\theta)\|$



$$\overline{v_1} = \frac{\partial loss}{\partial v_1} = \frac{\partial loss}{\partial v_{34}} \frac{\partial v_{34}}{\partial v_{33}} \frac{\partial v_{33}}{\partial v_{32}} \frac{\partial v_{32}}{\partial v_{31}} \frac{\partial v_{31}}{\partial v_{30}} \cdots \frac{\partial v_3}{\partial v_2} \frac{\partial v_2}{\partial v_1}$$

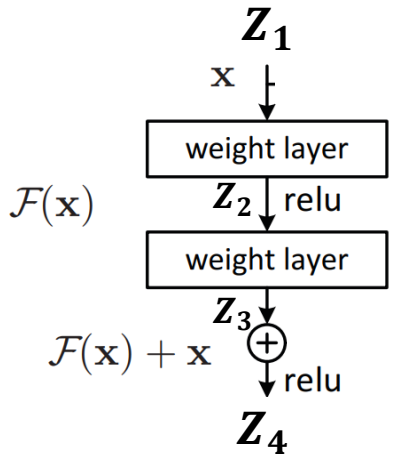
Ex: multiplying 1.9*1.9*...*1.9 = 1.9^34 = 3e9

Intuition: when calculating adjoints $\overline{v_i}$ for early layers, we multiply terms proportional to model depth. Recall: multiplying large (>1.0) values together yields a larger number: 2*2 = 4

Vanishing gradient and skip connection (1/2)

Skip connections mitigate this by adding an identity path for "distant" gradients to flow through unmodified.

Assume Z_2, Z_3 is output of "weight layer" (eg conv2d)

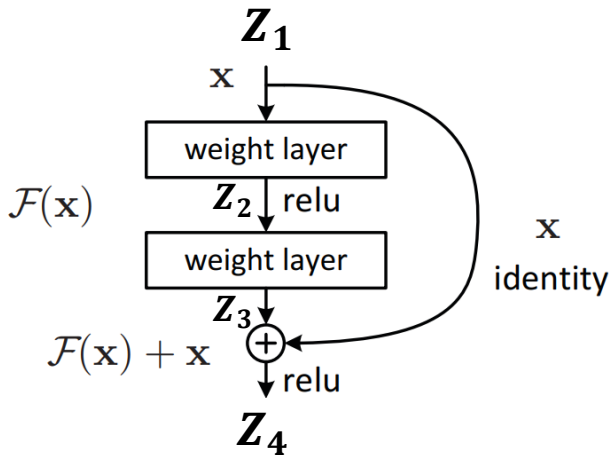


$$\overline{Z_1} = \frac{\partial loss}{\partial Z_1} = \overline{Z_2}\frac{\partial Z_2}{\partial Z_1}$$

This term contains terms proportional to model depth

$$\overline{Z_2} = \frac{\partial loss}{\partial Z_2} = \frac{\partial loss}{\partial Z_{34}}\frac{\partial Z_{34}}{\partial Z_{33}}\frac{\partial Z_{33}}{\partial Z_{32}}\cdots\frac{\partial Z_3}{\partial Z_2}$$

Multiplying many numbers together can lead to vanishing/exploding gradients, based on gradient magnitude



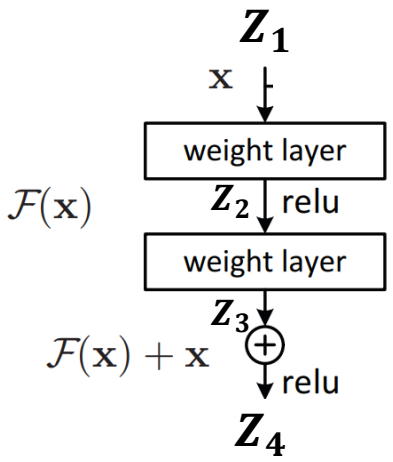
$$\overline{Z_1} = \overline{Z_2}\frac{\partial Z_2}{\partial Z_1} + \overline{Z_4}\frac{\partial Z_4}{\partial Z_1}$$

Gradient $\overline{Z_4}$ now can contribute directly to $\overline{Z_1}$, without having to go through the $\overline{Z_2}$ path.

Vanishing gradient and skip connection (2/2)

Let's take it one step further for dramatic effect...

Assume Z_2, Z_3 is output of "weight layer" (eg conv2d)

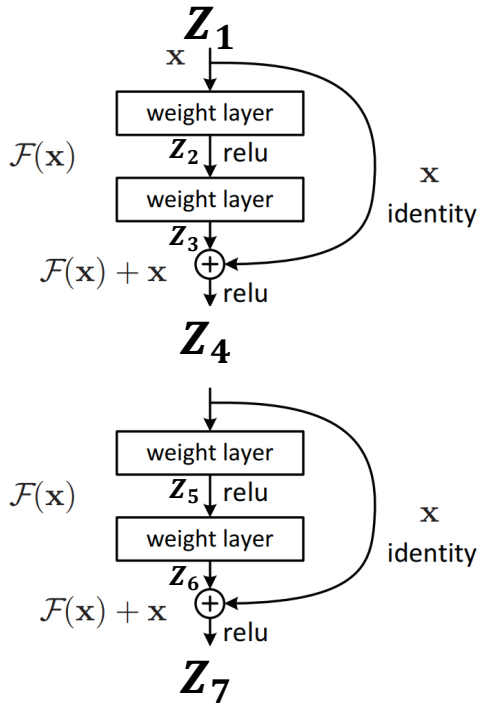


$$\overline{Z_1} = \frac{\partial loss}{\partial Z_1} = \overline{Z_2}\frac{\partial Z_2}{\partial Z_1}$$

This term contains terms proportional to model depth

$$\overline{Z_2} = \frac{\partial loss}{\partial Z_2} = \frac{\partial loss}{\partial Z_{34}}\frac{\partial Z_{34}}{\partial Z_{33}}\frac{\partial Z_{33}}{\partial Z_{32}}\cdots\frac{\partial Z_3}{\partial Z_2}$$

Multiplying many numbers together can lead to vanishing/exploding gradients, based on gradient magnitude



$$\overline{Z_1} = \overline{Z_2}\frac{\partial Z_2}{\partial Z_1} + \overline{Z_4}\frac{\partial Z_4}{\partial Z_1}$$

Recall: $\overline{Z_4} = \overline{Z_5}\frac{\partial Z_5}{\partial Z_4} + \overline{Z_7}\frac{\partial Z_7}{\partial Z_4}$

$$\overline{Z_1} = \overline{Z_2}\frac{\partial Z_2}{\partial Z_1} + \overline{Z_5}\frac{\partial Z_5}{\partial Z_4}\frac{\partial Z_4}{\partial Z_1} + \overline{Z_7}\frac{\partial Z_7}{\partial Z_4}\frac{\partial Z_4}{\partial Z_1}$$

Here, $\overline{Z_7}$ has a much closer contribution to $\overline{Z_1}$ than via the $\overline{Z_2}$ path. "Distant" healthy gradients (eg from CrossEntropyLoss) can now reach our early layers much easier, thanks to these skip connections!

Skip connections: make learning easier

Another interpretation of skip connections: they allow layers to learn "adjustments" to an input feature, which makes learning easier.

"one shot" transform F:

$Z_{out} = F(Z_{in})$

Learn an adjustment F(x)

$Z_{out} = F(Z_{in}) + Z_{in}$

Ex: if there were information in Z_{in} that's useful to maintain in Z_{out}, it's very simple for F to do this ("zero" out weights).

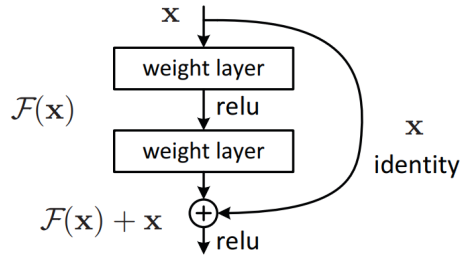
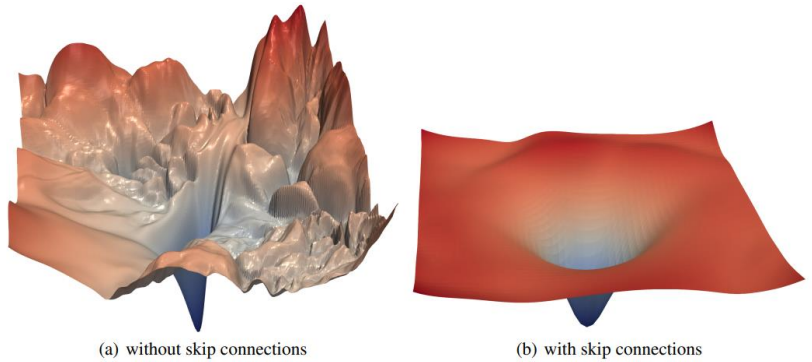


Figure 2. Residual learning: a building block.



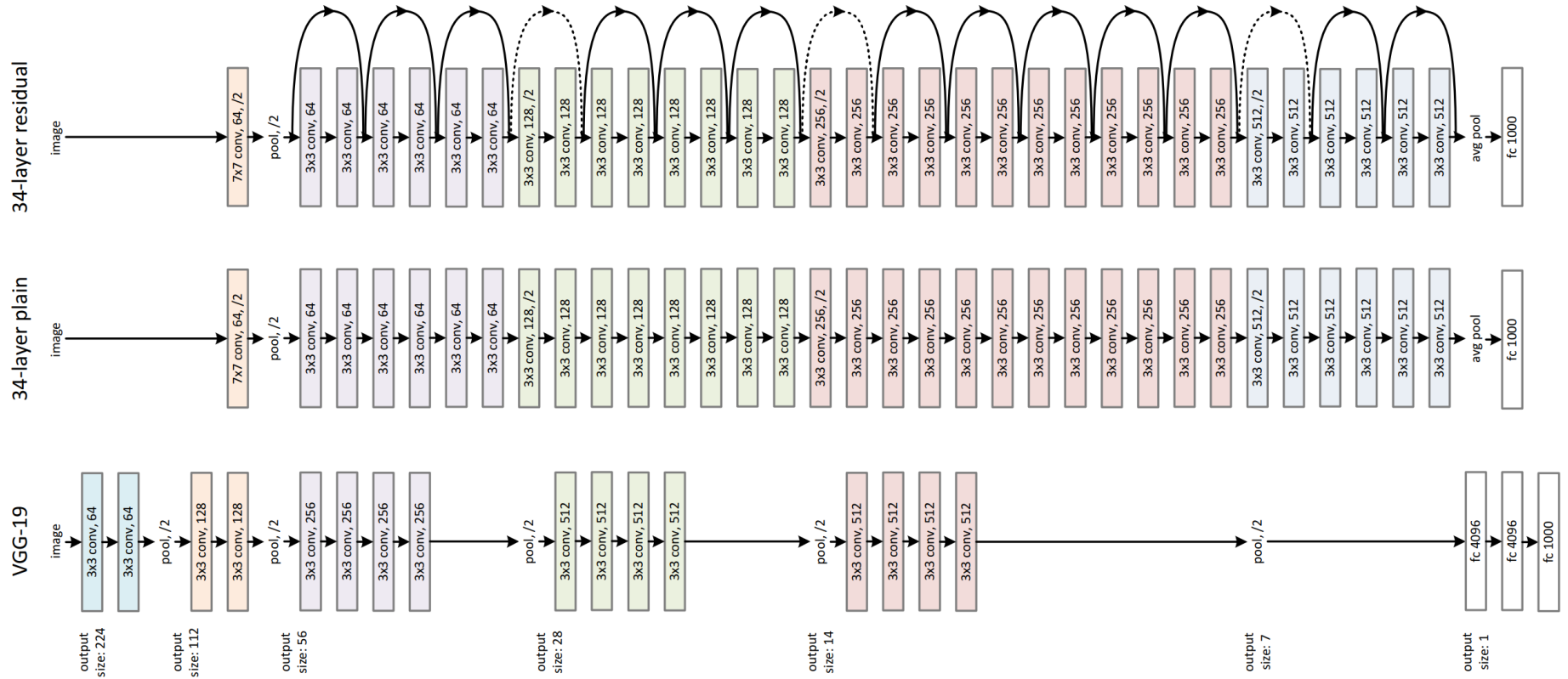
(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

"Visualizing the Loss Landscape of Neural Nets", https://arxiv.org/abs/1712.09913

ResNets: enabling deeper convnets

Thanks to skip connections, we can train significantly deeper convnets than before!

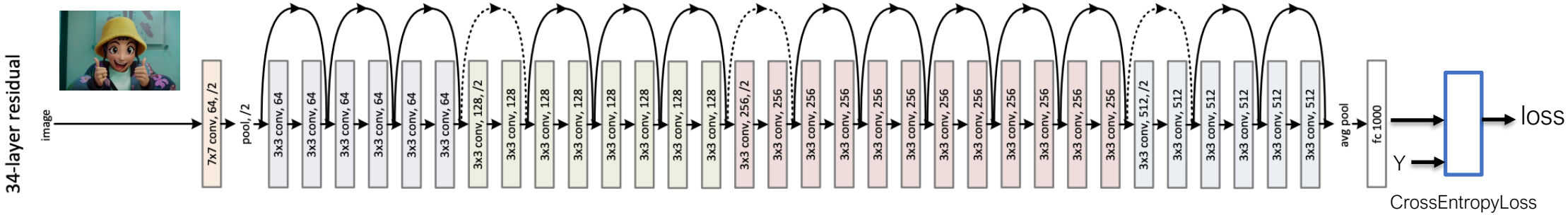


ResNet architecture configs

ResNet "blocks". Aka repeated Conv2d->BatchNorm->Relu with skip connections.

Each block shrinks the feature map spatial resolution by 2x via conv2d(stride=2).
Tip: to account for reduction in feature map spatial resolution ("representation power"), they double the number of channels each

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9



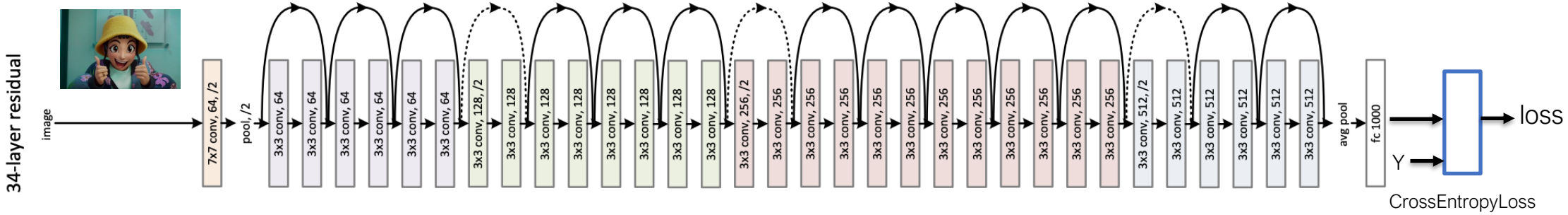
ResNet: "bottleneck" layer

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Trick: to reduce computation for very-deep resnets (50, 101, 152 layers), they employ 1x1 conv2d to reduce the number of channels prior to each 3x3 conv2d.

Z_{in} -> conv2d(C_{in}=256, C_{out}=64, k=1) cheap 1x1, reduce chans
-> conv2d(C_{in} =64, C_{out} =64, k=3) "expensive 3x3 conv"
-> conv2d(C_{in} =64, C_{out} =256, k=1) cheap 1x1, expand chans

For more info, see: "How 1x1 Convolutions Reduce CNN Cost - Without Losing the Plot"
Also: see Resnet source code: `ResNetBottleNeckLayer`



Outline

Elements of practical convolutions

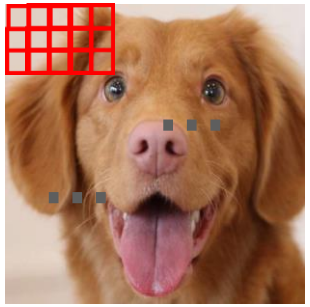
Convolutional networks (Resnets)

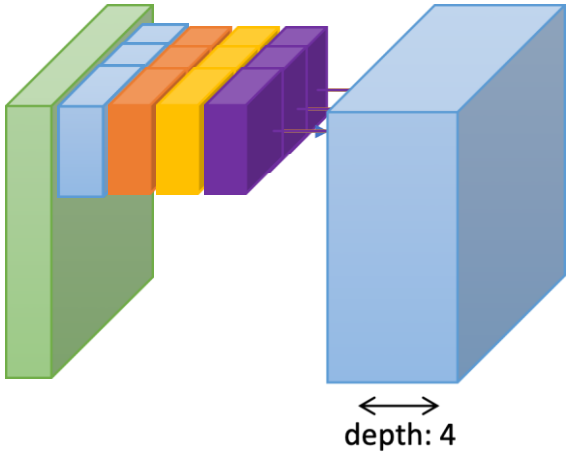
Interpreting feature maps

Differentiating convolutions

Computer Vision

Feature map interpretation





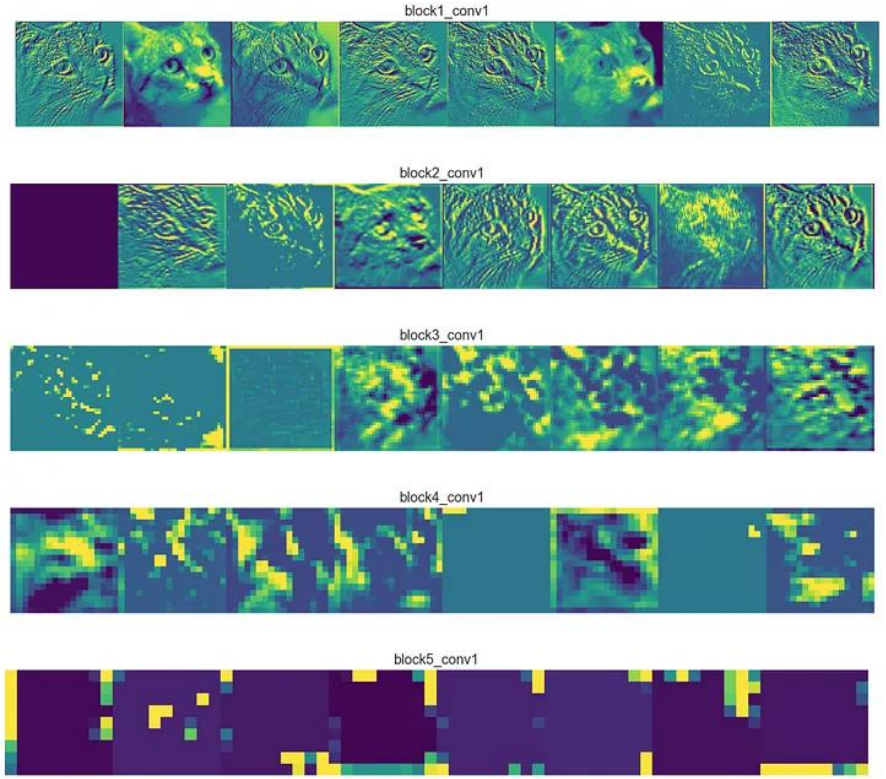
What will our output look like?

"Sliding" the filter along the image

It actually looks quite like an "image" itself… interesting…

"Spatial feature map"

Low level: edge detectors



High level: object detectors. Intuition/hope: high activations ideally mean "there is a semantic object (cat, dog) in this image!"

Interpretation: by stacking many conv layers, CNNs learn **hierarchical features**.
Lower level layers: Low-level image features (edges).
Middle layers: Mid-level image features (shapes)
Final layers: "semantic" features (eg part detectors, face detectors)

https://medium.com/@saba99/feature-map-35ba7e6c689e

Visualizing convnet learned filters

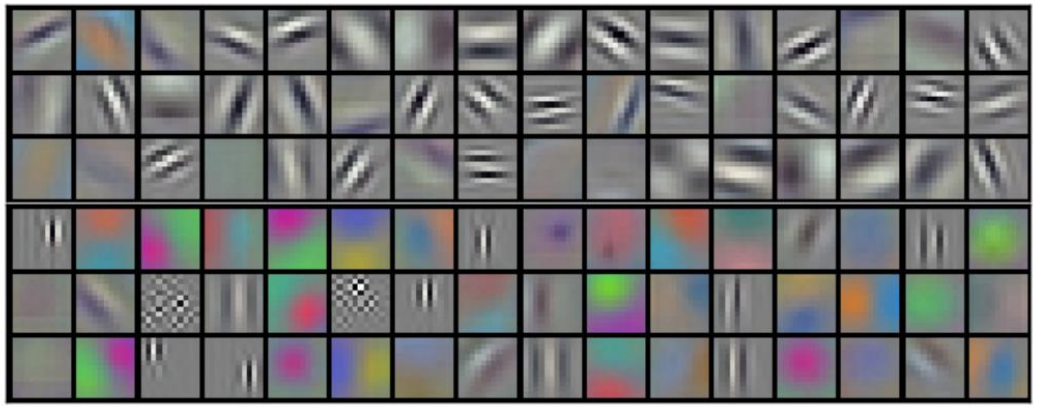


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

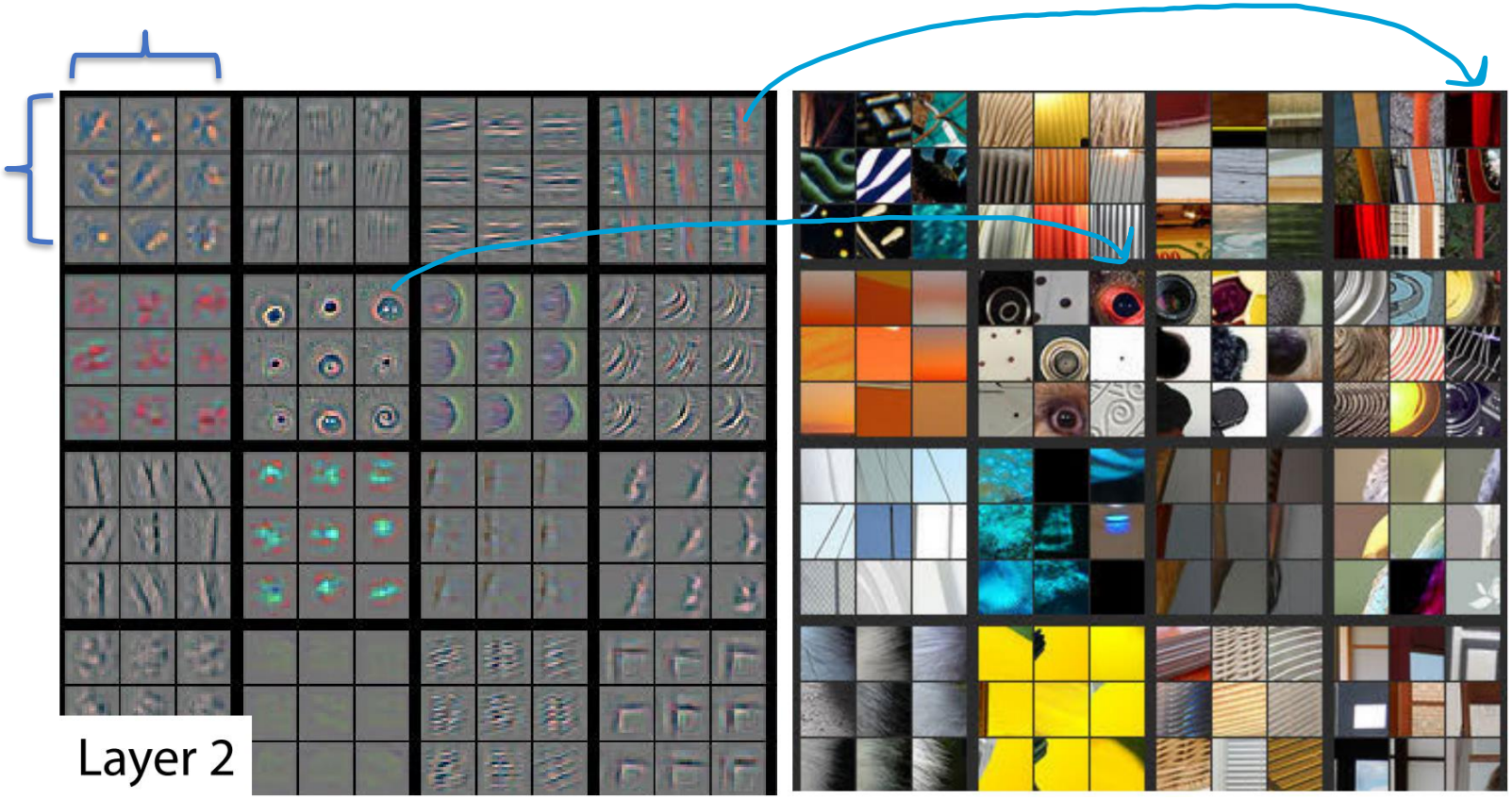
Looks like low-level oriented edge detectors!

https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Visualizing convnet spatial feature maps (1/4)

Interpretation: each 3x3 subgrid contains the top-9 activations for a specific spatial feature map at a given Layer (eg the output of some Conv2d layer).

Within a 3x3 subgrid, each cell represents a different image that resulted in that spatial feature map activation.



Validation images

https://arxiv.org/abs/1311.2901

Visualizing convnet spatial feature maps (2/4)

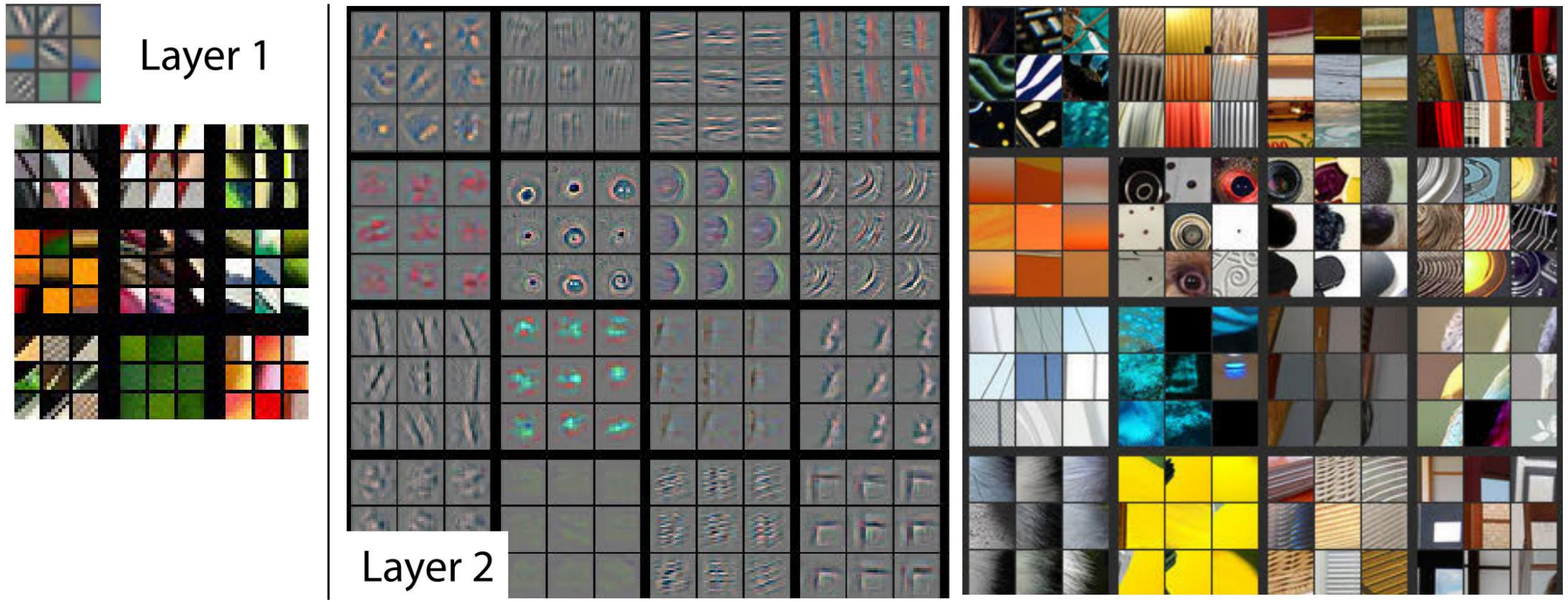


Figure 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are *not* samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form.

https://arxiv.org/abs/1311.2901

Visualizing convnet spatial feature maps (3/4)

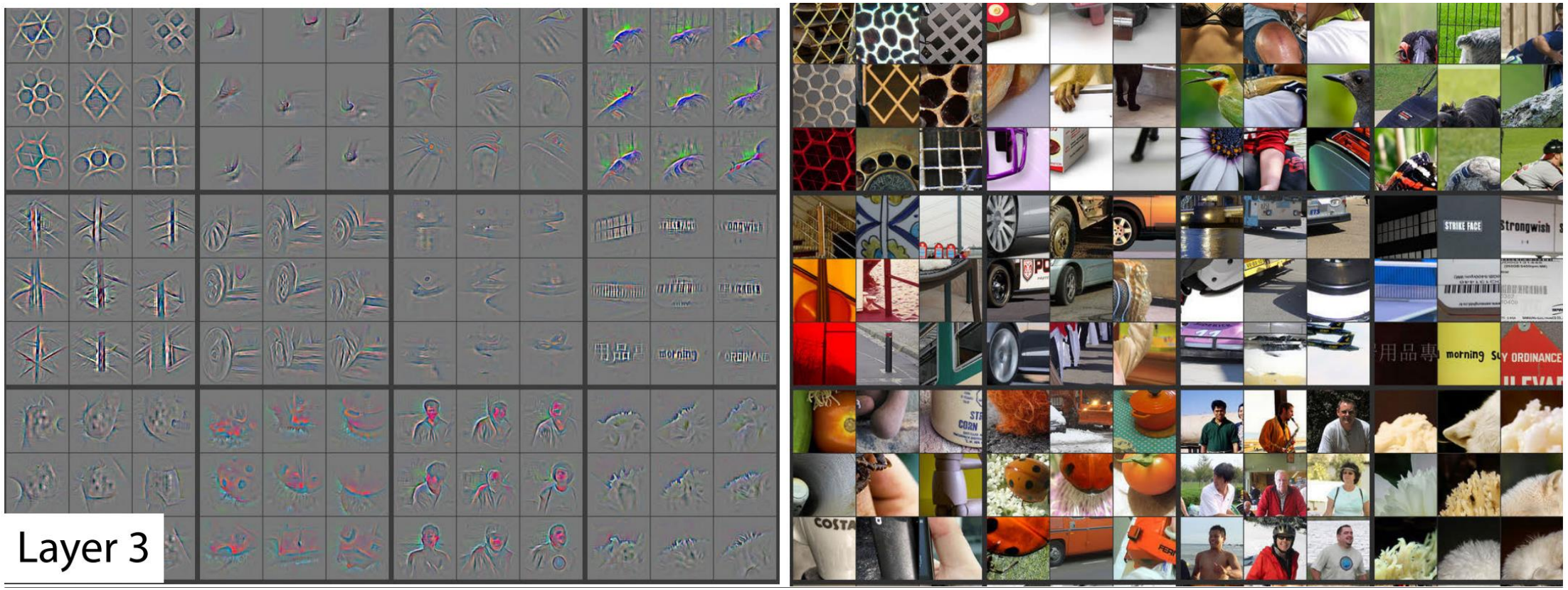
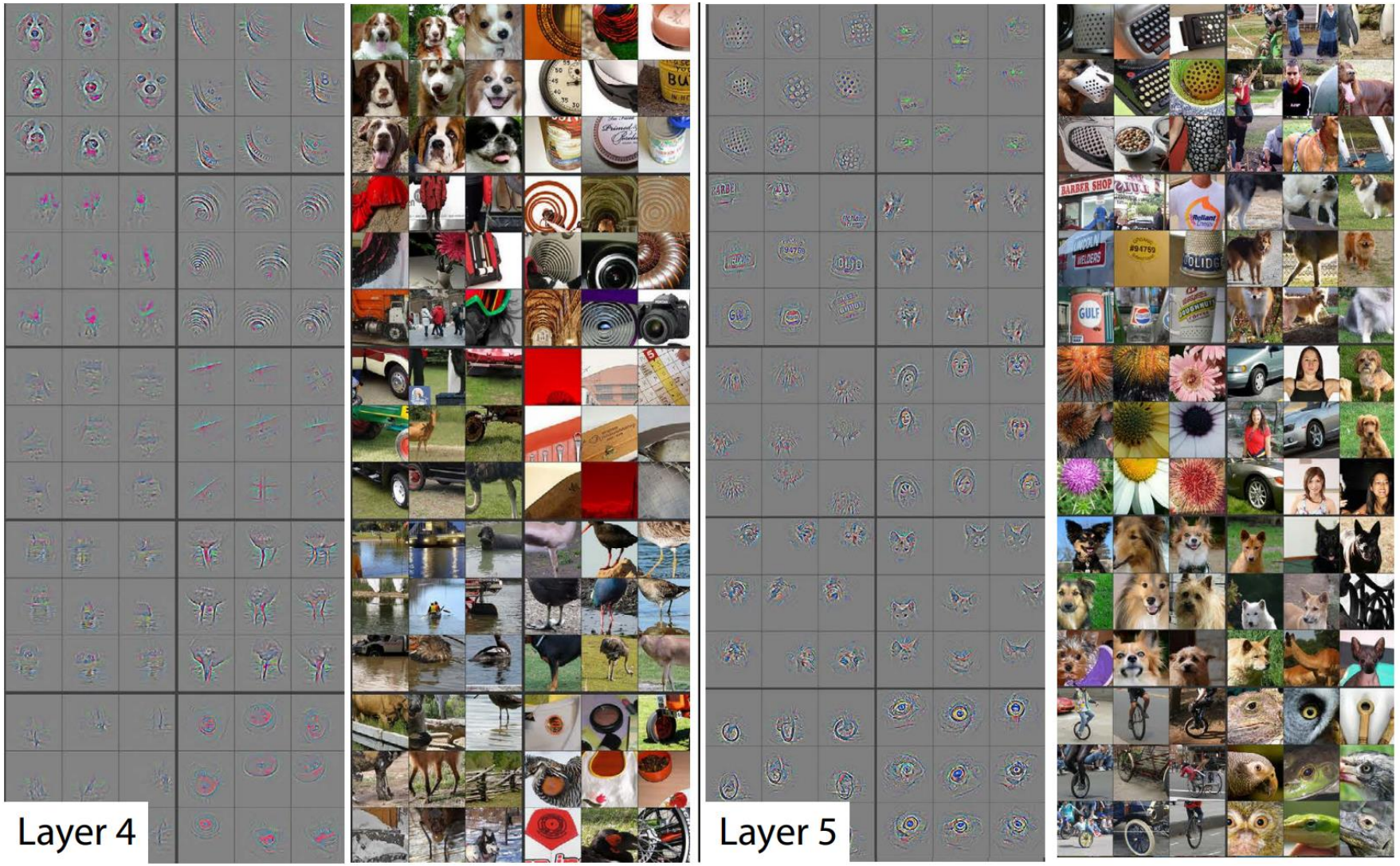


Figure 2. Visualization of features in a fully trained model. For layers 2-5 we show the top 9 activations in a random subset of feature maps across the validation data, projected down to pixel space using our deconvolutional network approach. Our reconstructions are *not* samples from the model: they are reconstructed patterns from the validation set that cause high activations in a given feature map. For each feature map we also show the corresponding image patches. Note: (i) the the strong grouping within each feature map, (ii) greater invariance at higher layers and (iii) exaggeration of discriminative parts of the image, e.g. eyes and noses of dogs (layer 4, row 1, cols 1). Best viewed in electronic form.

https://arxiv.org/abs/1311.2901

Visualizing convnet spatial feature maps (4/4)



https://arxiv.org/abs/1311.2901

Outline

Elements of practical convolutions

Convolutional networks (Resnets)

Interpreting feature maps

Differentiating convolutions

Computer Vision

What is needed to differentiate convolution?

Recall that in order to integrate any operation into a deep network, we need to be able to multiply by its partial derivatives (adjoint operation)

So if we define our operation

$$z = \mathrm{conv}(x, W)$$

how do we multiply by the adjoints

$$\bar{v}\frac{\partial \mathrm{conv}(x, W)}{\partial W}, \qquad \bar{v}\frac{\partial \mathrm{conv}(x, W)}{\partial x}$$

Refresher on differentiating matrix multiplication

Let's consider the simpler case of a matrix-vector product operation

$$z = Wx$$

Then $\frac{\partial z}{\partial x} = W$, so we need to compute the adjoint product

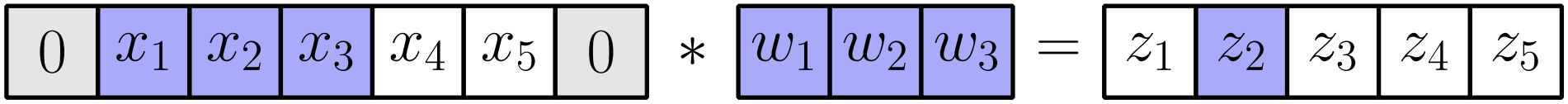
$$\bar{v}^T W \Leftrightarrow W^T \bar{v}$$

In other words, for a matrix vector multiply operation Wx, computing the backwards pass requires multiplying by the *transpose* W^T

So what is the "transpose" of a convolution?

Convolutions as matrix multiplication: Version 1

To answer this question, consider a 1D convolution to keep things a bit simpler:



We can write a 1D convolution $x * w$ (e.g., with zero padding) as a matrix multiplication $\hat{W}x$ for some $\hat{W}$ properly defined in terms of the filter w

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = x * w = \underbrace{\begin{bmatrix} w_2 & w_3 & 0 & 0 & 0 \\ w_1 & w_2 & w_3 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 \\ 0 & 0 & 0 & w_1 & w_2 \end{bmatrix}}_{\hat{W}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix}$$

The adjoint of a convolution

So how can we multiply by the transpose $\widehat{W}^T$?

Written out as in the previous slide, it's quite easy:

$$\widehat{W}^T = \begin{bmatrix} w_2 & w_1 & 0 & 0 & 0 \\ w_3 & w_2 & w_1 & 0 & 0 \\ 0 & w_3 & w_2 & w_1 & 0 \\ 0 & 0 & w_3 & w_2 & w_1 \\ 0 & 0 & 0 & w_3 & w_2 \end{bmatrix}$$

But notice that the operation $\widehat{W}^T v$ it itself just a convolution with the "flipped" filter: $[w_3 \quad w_2 \quad w_1]$; $\Longrightarrow$ adjoint operator $\bar{v}\frac{\partial \mathbf{conv}(x,W)}{\partial x}$ just requires convolving $\bar{v}$ with a the flipped W!

Convolutions as matrix multiplication: Version 2

What about the other adjoint, $\bar{v}\frac{\partial \mathbf{conv}(x,W)}{\partial W}$?

For this term, observe that we can *also* write the convolution as a matrix-vector product treating the *filter* as the vector

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ z_5 \end{bmatrix} = x * w = \begin{bmatrix} 0 & x_1 & x_2 \\ x_1 & x_2 & x_3 \\ x_2 & x_3 & x_4 \\ x_3 & x_4 & x_5 \\ x_4 & x_5 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

So adjoint requires multiplying by the transpose of this x-based matrix!

(for fun) Creating this x-based matrix is known as `im2col`. See this post that (concisely) explains why this `im2col()` operation is used to implement efficient conv2d: "Why is im2col more efficient than naive 2d convolution?"

Outline

Elements of practical convolutions

Convolutional networks (Resnets)

Interpreting feature maps

Differentiating convolutions

Computer Vision

Standard computer vision problems

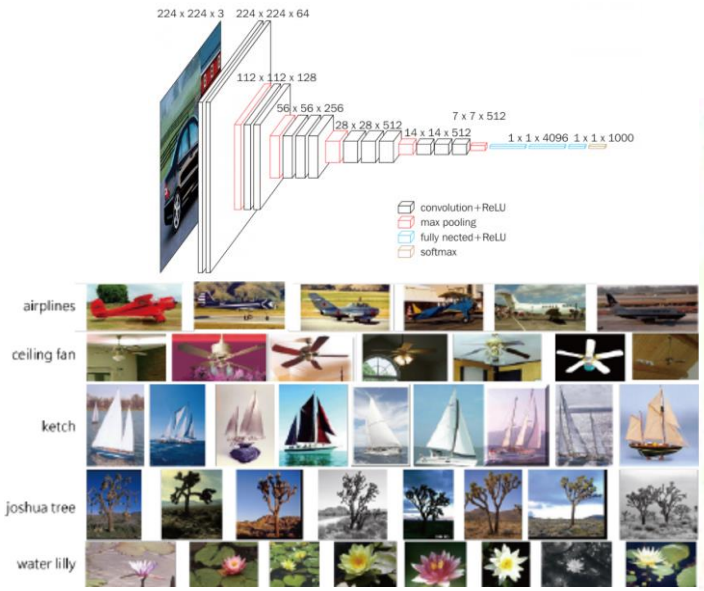
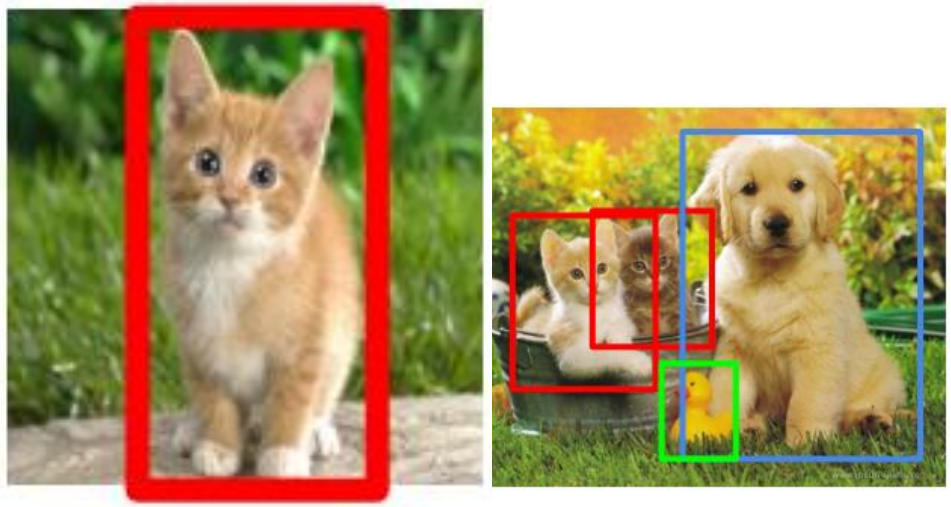
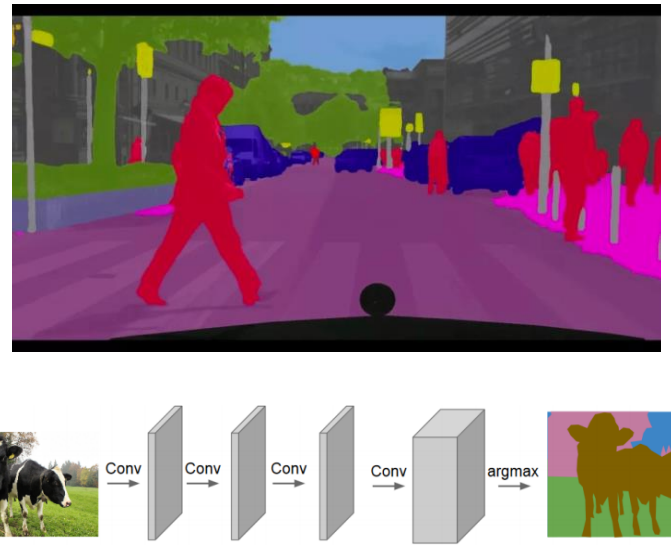


image classification

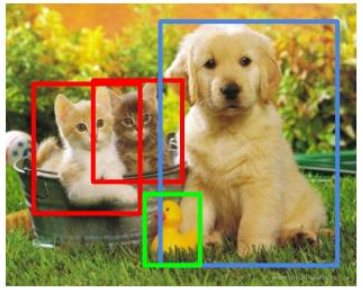


object detection



semantic segmentation
a.k.a. scene understanding

Object detection setup



Note: an image may have multiple ground truth objects!

Before: $\mathcal{D} = \{(x_i, y_i)\}$

image class label (categorical)

Now: $\mathcal{D} = \{(x_i, y_i)\}$

image $y_i = (\ell_i, x_i, y_i, w_i, h_i)$

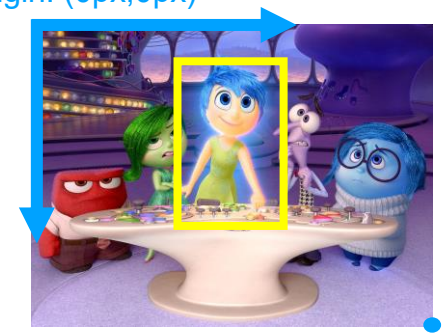
Example: ("cat", 0.2, 0, 0.6, 1.0)
Tip: rather than use pixel (absolute) coordinates, we often use preprocess the data to use "normalized" coordinates.

Representing bbox coordinates

- Image sizes/coords are often represented in terms of pixels units
  - "**absolute**" coordinates
- Better: represent coords as "**normalized**" coordinates
  - $x_{norm} = \frac{x_{px}}{w_{img}}$
  - $y_{norm} = \frac{y_{px}}{h_{img}}$
- **Question**: would it be better if our bbox training set is in absolute coordinates, or normalized coordinates? Why?
- **Answer**: Generally normalized coordinates are preferred since it generalizes well to multiple image shapes, and makes the learning problem a little easier.
  - If you know in advance that your images are always a fixed size (eg 224x224), then absolute is maybe OK...

Example: this image has dimensions width=1697px, height=1281px.

Origin: (0px,0px)



(1696px, 1280px)

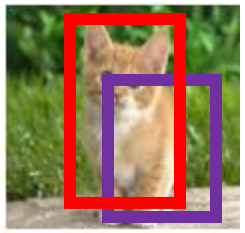
$box_{absolute}$ = (x=509px, y=128px, w=424px, h=896px)

$box_{normalized}$ = (x=0.3, y=0.1, w=0.25, h=0.7)

https://www.fxguide.com/fxfeatured/inside-out-rendering/

Measuring localization accuracy

learned model

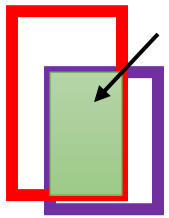


(x, y, w, h) ← predicted bounding box

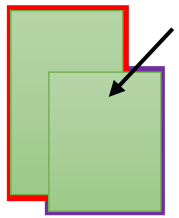
"cat": 0.64 ← prediction score (e.g., probability)

Did we get it right?

Intersection over Union (IoU)



intersection area (**I**)



union area (**U**)

IoU = I / U

Different datasets have different protocols, but one reasonable one is: **correct if IoU > 0.5**

If also outputting class label (usually the case): **correct if IoU > 0.5 and class is correct**

IoU ranges from [0.0, 1.0], where higher is better.

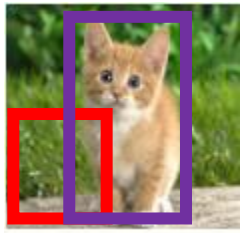
For perfect predicted box: IoU = 1.0

This is **not** a loss function! Just an evaluation standard

Object localization

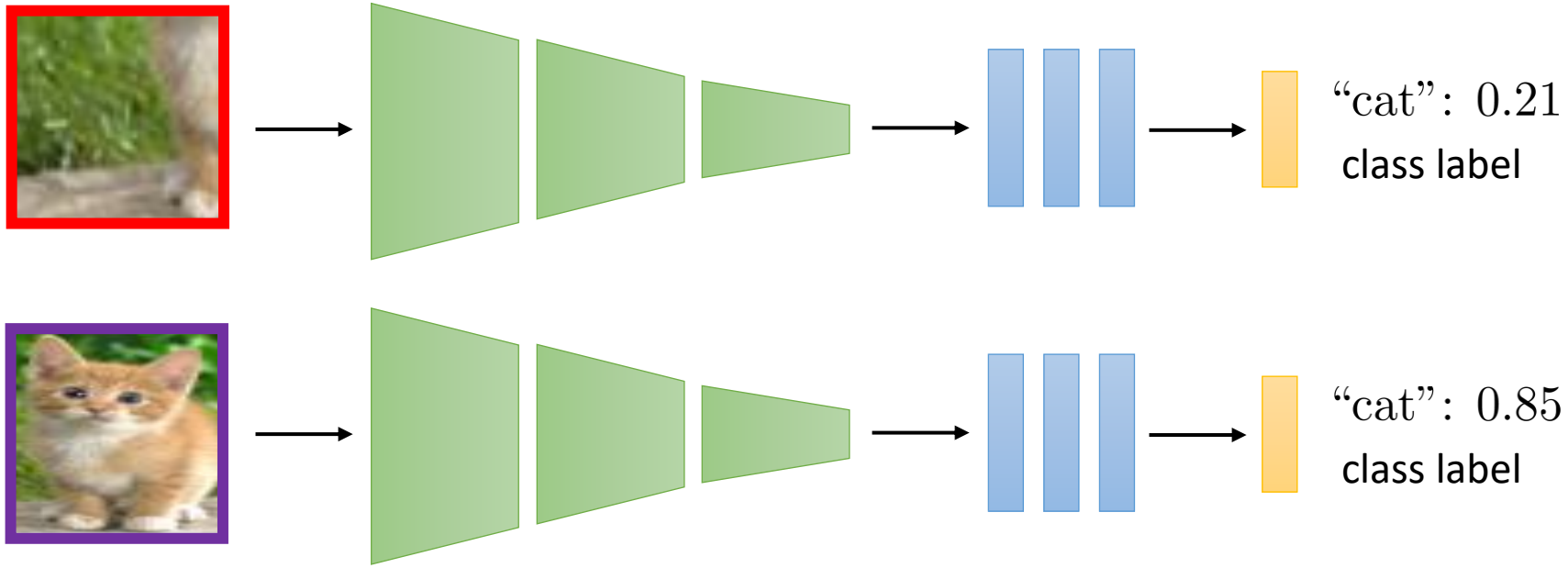
Naive attempt: Sliding windows

$\mathcal{D} = \{(x_i, y_i)\}$ $\quad y_i = (\ell_i, x_i, y_i, w_i, h_i)$



Suppose we had a trained image classifier, and we want to use it as a "subroutine" to implement an object detector.
Idea: What if we classify **every** patch in the image?

Problem: In theory, it could work...but it'd be so, so slow. There are many candidate patches in an image. Not practical!



For N image patches, would require N separate forward passes. Ouch!

Object localization as regression

$\mathcal{D} = \{(x_i, y_i)\}$ $y_i = (\ell_i, x_i, y_i, w_i, h_i)$

Classification task ("head")

cross-entropy loss

"cat": 0.64

class label

Bbox regression task ("head")

(x, y, w, h)

Bbox regression loss

- Most object detection systems look something like this
- One "head" for doing classification
- Another "head" for predicting bounding boxes
- Different approaches may change the picture, but the spirit is still there

Reasonable bbox loss: sum squared error (L2) of each term:

$$\|box_{pred} - box_{gt}\|_2^2 = (x_{pred} - x_{gt})^2 + (y_{pred} - y_{gt})^2 + (w_{pred} - w_{gt})^2 + (h_{pred} - h_{gt})^2$$

"gt": ground truth