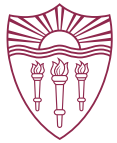
USC Viterbi
School of Engineering



CONVOLUTIONAL NEURAL NETWORKS

EE 541 – UNIT 8B

DR. BRANDON FRANZKE

Fall 2026

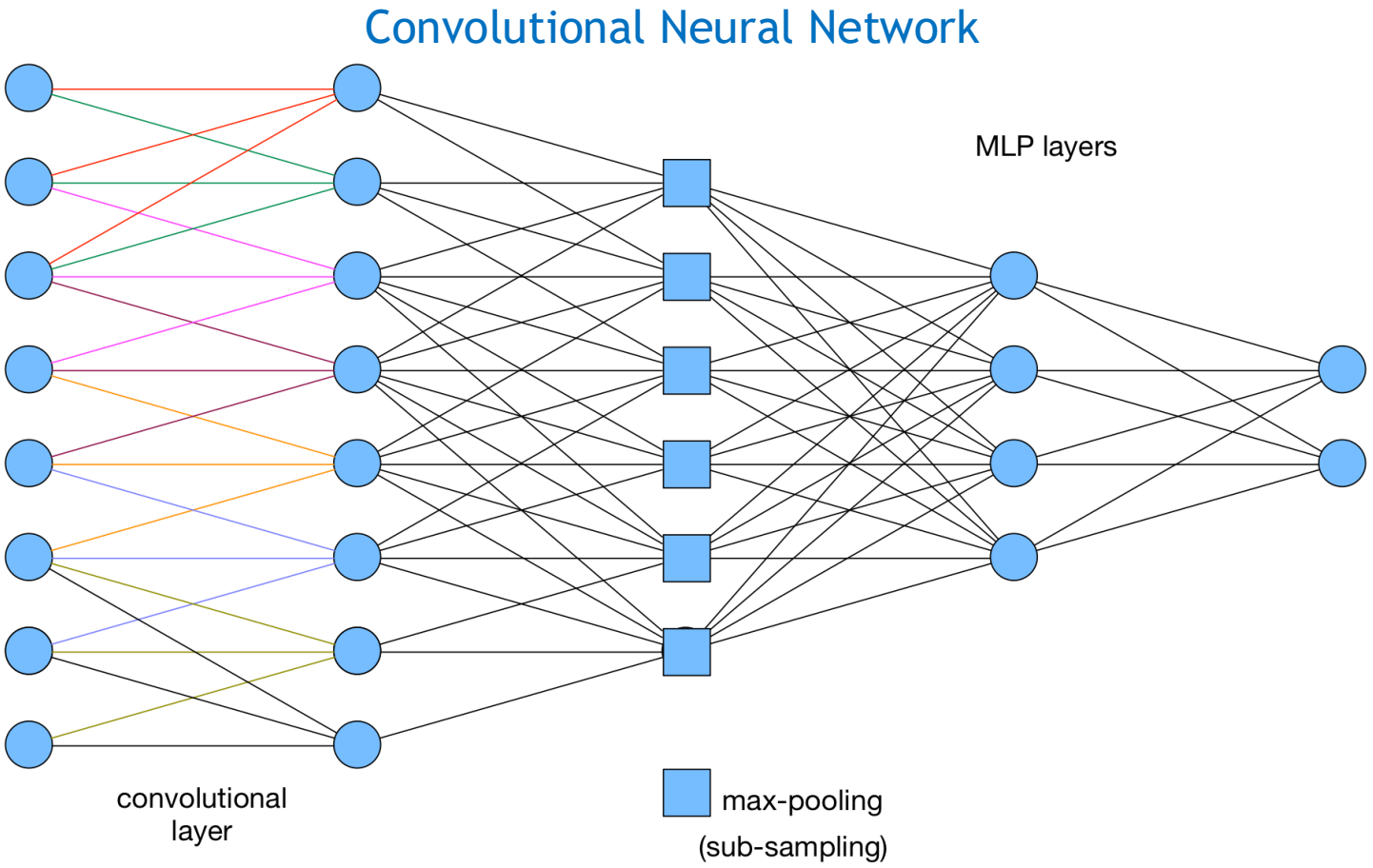
OUTLINE FOR SLIDES

- Motivation, applications
- Basic 2D convolution operations
  - PyTorch 2Dconv layer
- Pooling and stride
- Fashion MNIST example
- Visualization methods
- Some common CNN structures
- Reduced complexity CNN architectures
- Outline of Back-propagation for CNNs



CONVNETS

(TYPES OF NEURAL NETWORKS)



Can view convolutions as feature extractors for MLP classifier
(this feature extraction is learned)



VISUALIZATION

DOGS VS. CATS 😃















DOGS VS. CATS 😃

Dataset available here

https://www.kaggle.com/c/dogs-vs-cats

let's explore a simple CNN and see if we can get some insight into what the filters are looking for and how they respond to a given input image

DOGS-V-CATS: CATS AND DOGS – CNN.IPYNB

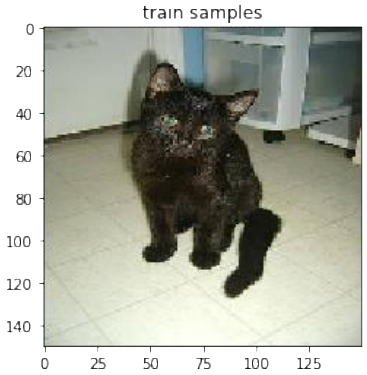
Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 32, 150, 150]	896
Conv2d-2	[-1, 64, 150, 150]	18,496
MaxPool2d-3	[-1, 64, 75, 75]	0
Conv2d-4	[-1, 128, 75, 75]	73,856
Conv2d-5	[-1, 128, 75, 75]	147,584
MaxPool2d-6	[-1, 128, 37, 37]	0
Conv2d-7	[-1, 256, 37, 37]	295,168
Conv2d-8	[-1, 512, 37, 37]	1,180,160
MaxPool2d-9	[-1, 512, 18, 18]	0
Conv2d-10	[-1, 512, 18, 18]	2,359,808
Conv2d-11	[-1, 512, 18, 18]	2,359,808
MaxPool2d-12	[-1, 512, 8, 8]	0
Dropout2d-13	[-1, 32768]	0
Linear-14	[-1, 512]	16,777,728
Linear-15	[-1, 1]	513

Total params: 23,214,017

Trainable params: 23,214,017

Non-trainable params: 0

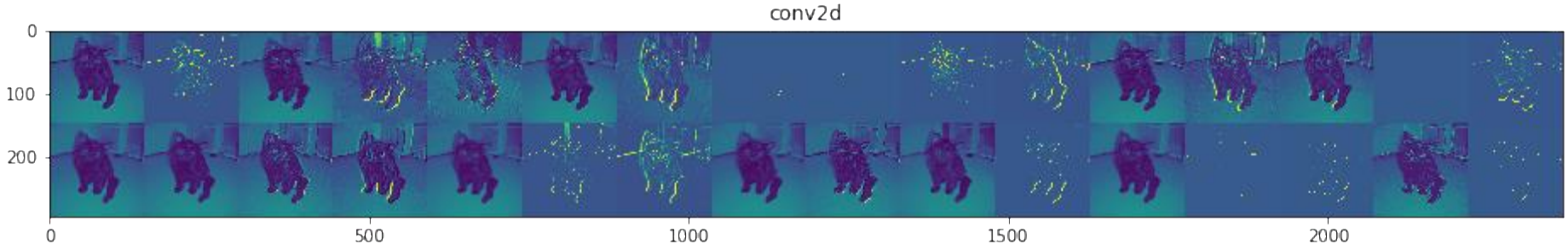
DOGS-V-CATS: VISUALIZING CNN FEATURE MAPS



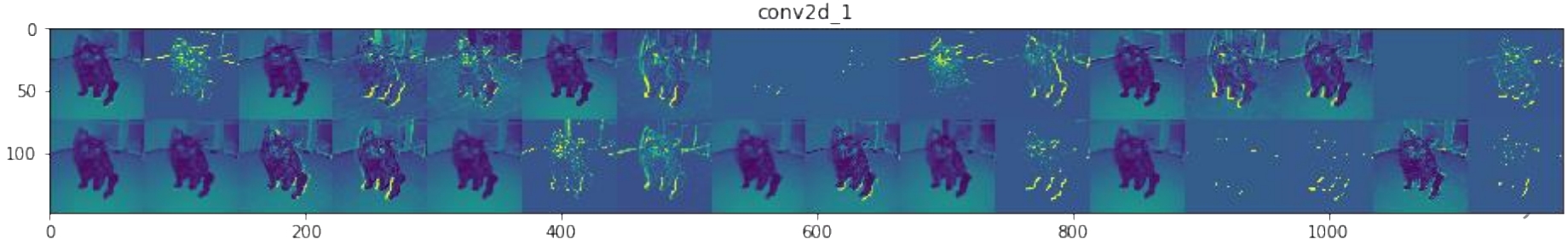
input image

Cats and Dogs - viz.ipynb

1st conv2D



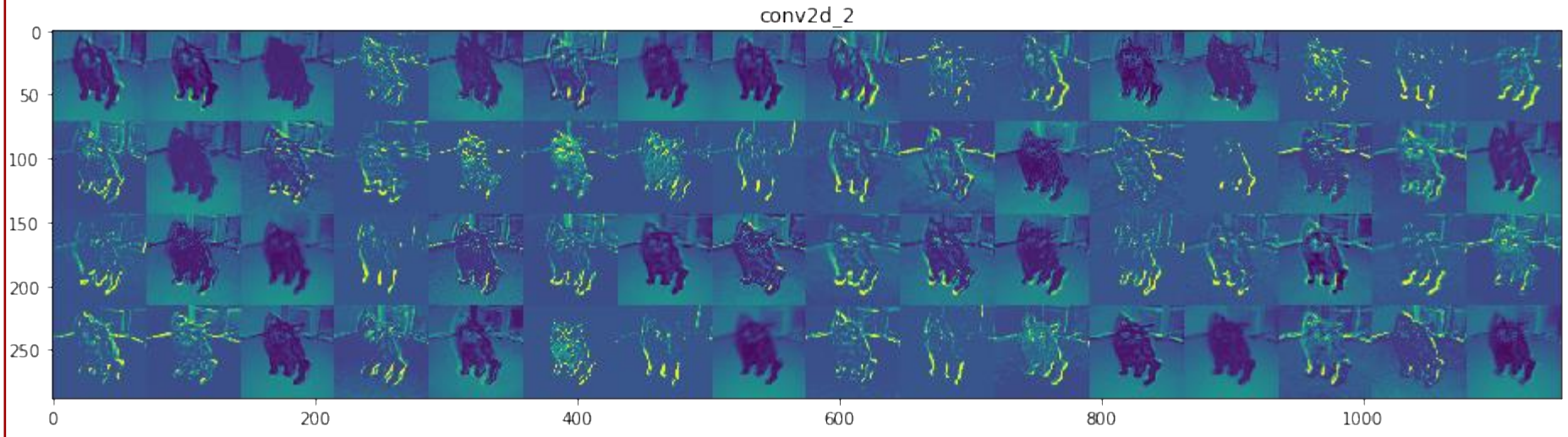
2nd conv2D



DOGS-V-CATS: VISUALIZING CNN FEATURE MAPS

Cats and Dogs – viz.ipynb

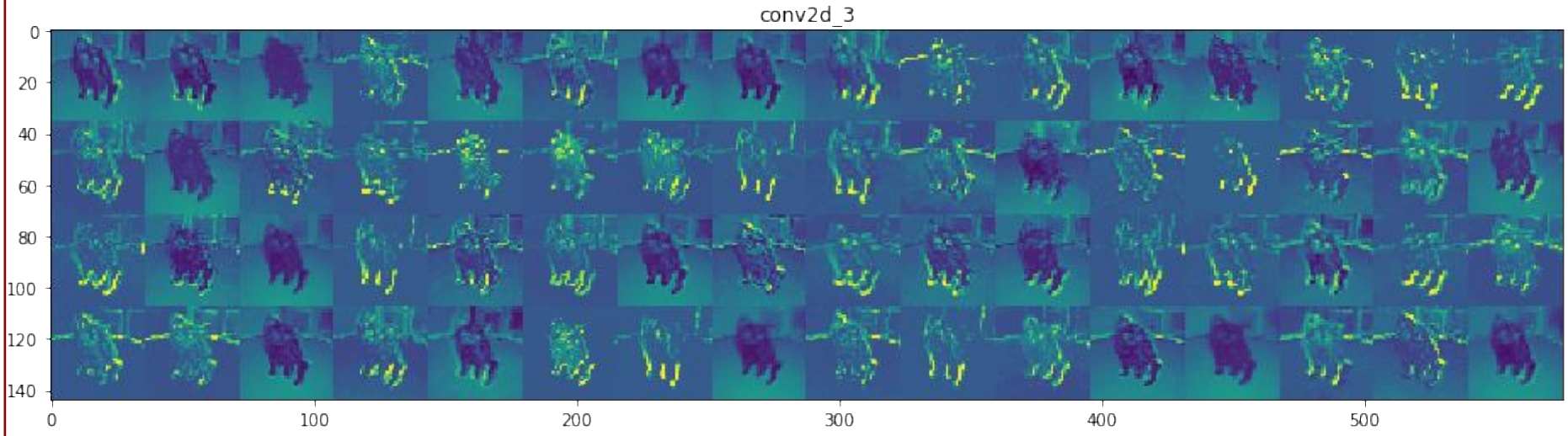
3rd conv2D



DOGS-V-CATS: VISUALIZING CNN FEATURE MAPS

Cats and Dogs – viz.ipynb

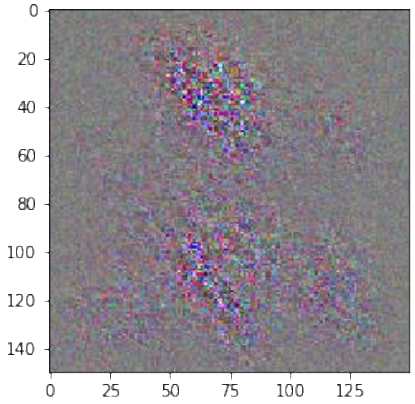
4th conv2D



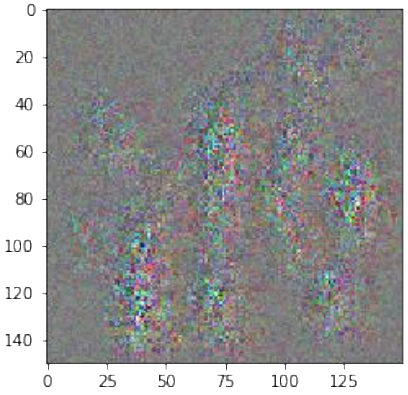
DOGS-V-CATS: MAX FILTER RESPONSE

train an input image so that it maximizes the output energy in a particular filter

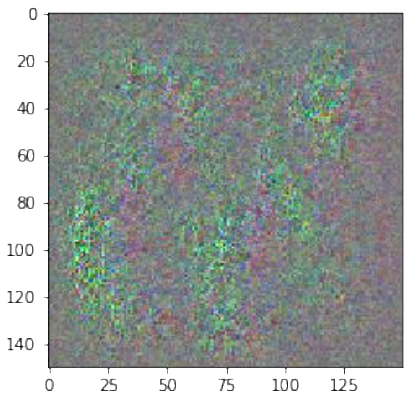
Cats and Dogs – viz.ipynb



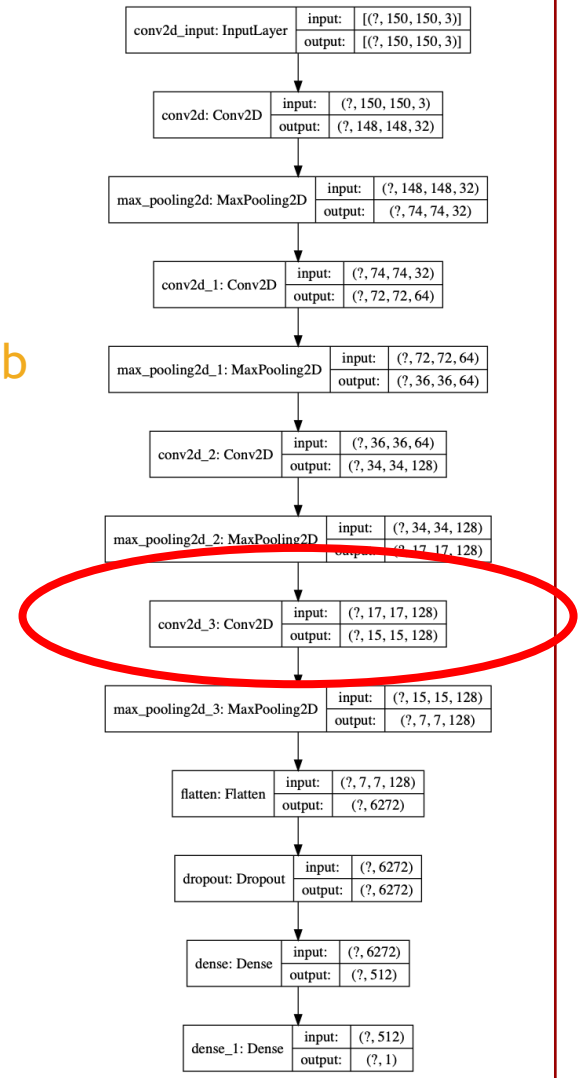
channel 16



channel 71

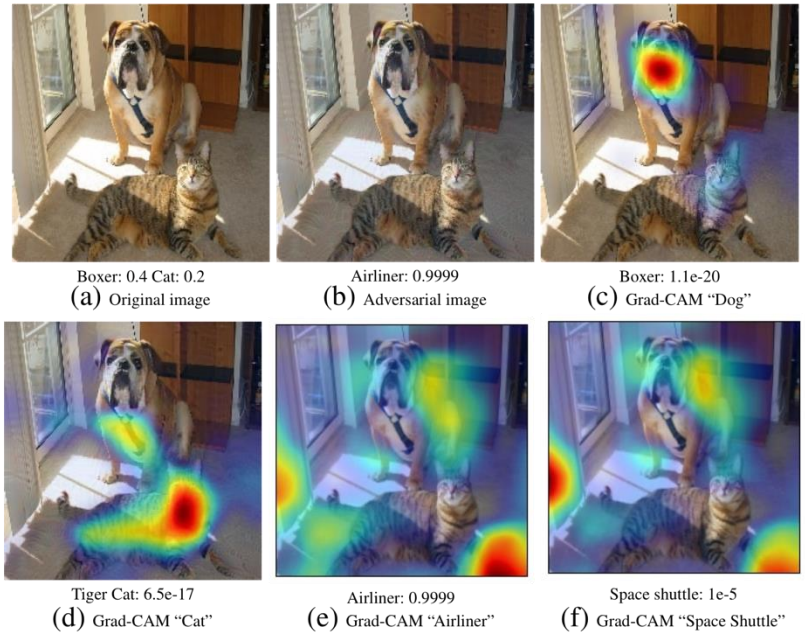


channel 121



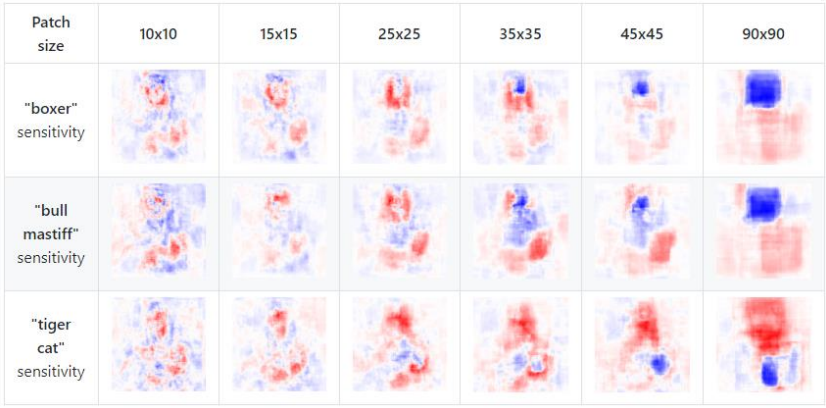
CNN VISUALIZATION: GRAD-CAM

Gradient Weighted Class Activation Mapping



pyimagesearch tutorial (keras)

demo
https://github.com/kazuto1011/grad-cam-pytorch



see where a layer is "looking" for a given class

Selvaraju, Ramprasaath R., et al. "Grad-cam: Visual explanations from deep networks via gradient-based localization." Proceedings of the IEEE international conference on computer vision. 2017.

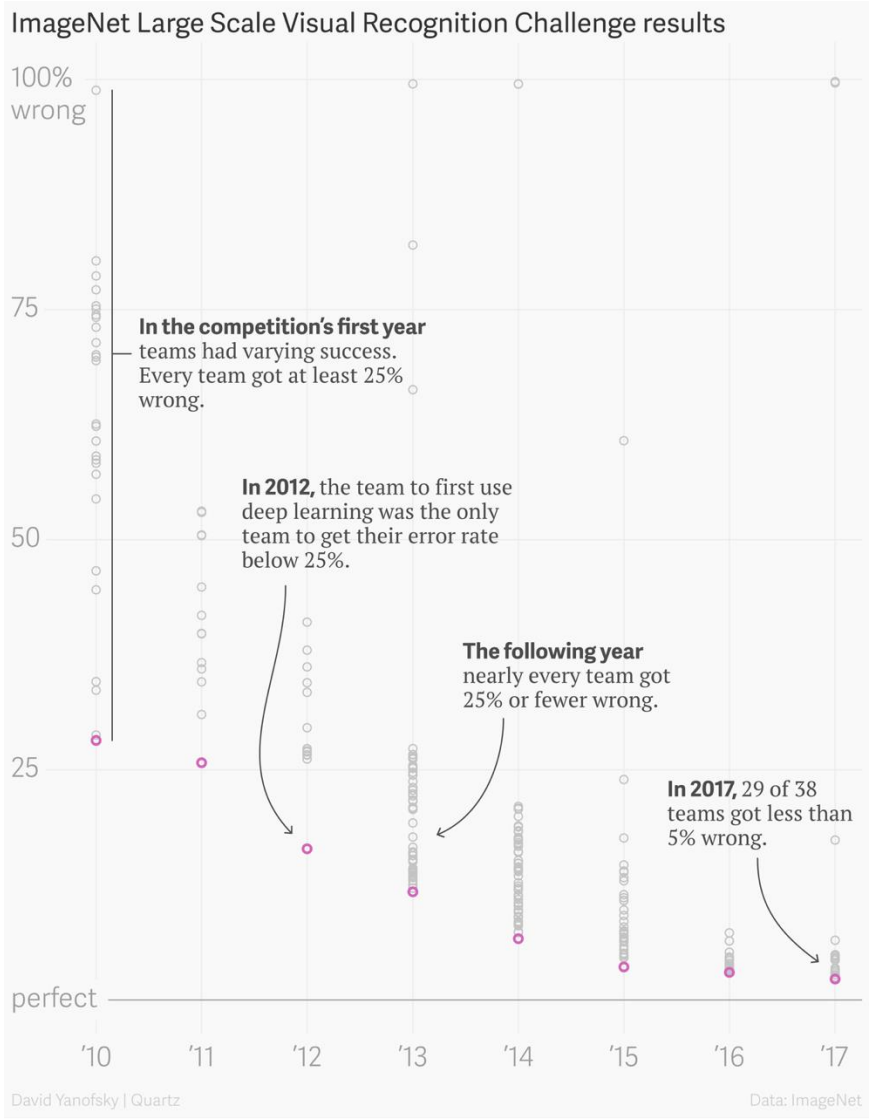
OUTLINE FOR SLIDES

- Motivation, applications
- Basic 2D convolution operations
  - PyTorch 2Dconv layer
- Pooling and stride
- Fashion MNIST example
- Visualization methods
- Some common CNN structures
- Reduced complexity CNN architectures
- Outline of Back-propagation for CNNs



BLOCK STRUCTURES

CNNS: USE WHEN FEATURE INFORMATION IS LOCALIZED



2012: AlexNet

- ~60M parameters
- 16.4% top-5 error

2014: VGG

- ~140M parameters
- 7.3% top-5 error

2014: Inception (aka GoogLeNet)

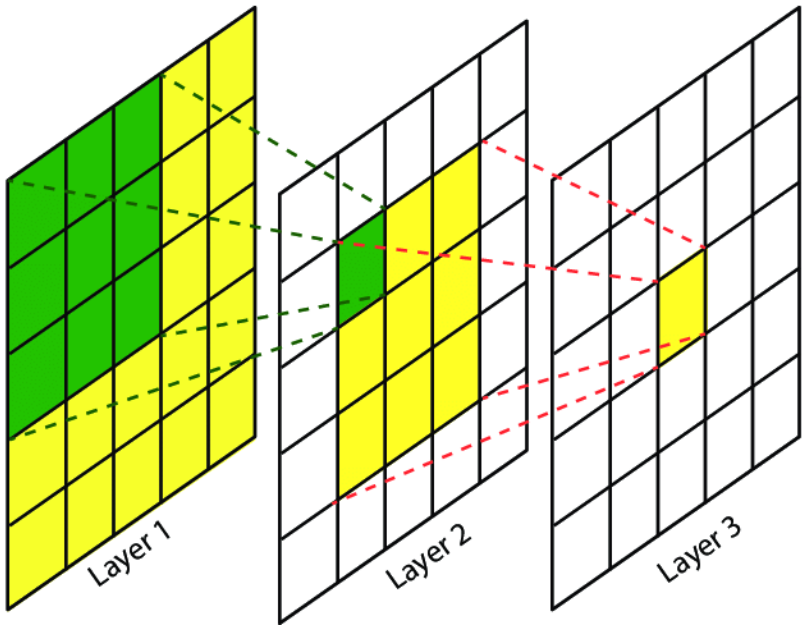
- ~7M parameters
- ~7% top-5 error

2015 ResNet

- ~60M parameters
- 3.6% top-5 error

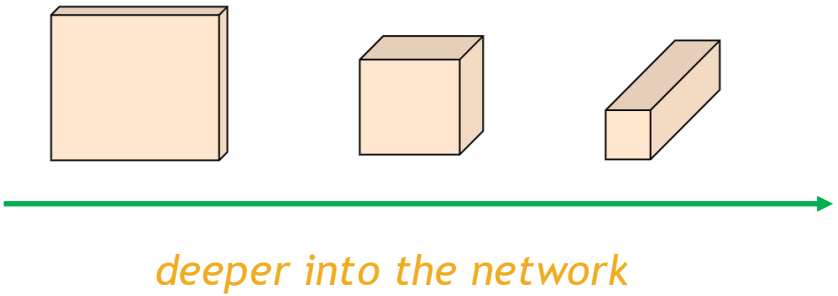
The leap that transformed AI research—and possibly the world

RECEPTIVE FIELD AS WE GO DEEPER



deeper in the network, each pixel in the feature map can "see" more of the input image

reason why height and width of the feature map can be reduced as we go deeper



Lin, Haoning, Zhenwei Shi, and Zhengxia Zou. "Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network." Remote sensing 9.5 (2017): 480.

RECEPTIVE FIELD AS WE GO DEEPER

simple script to find input pixels that can affect output pixels for a specific CNN architecture (pytorch-receptive-field)

```
receptive_field_dict = receptive_field(model, (3, 256, 256))
receptive_field_for_unit(receptive_field_dict, '2', (2,2))
```

```
class Net(nn.Module):
 def __init__(self):
  super(Net, self).__init__()
  self.conv = nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3,      bias=False)
  self.bn = nn.BatchNorm2d(64)
  self.relu = nn.ReLU(inplace=True)
  self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

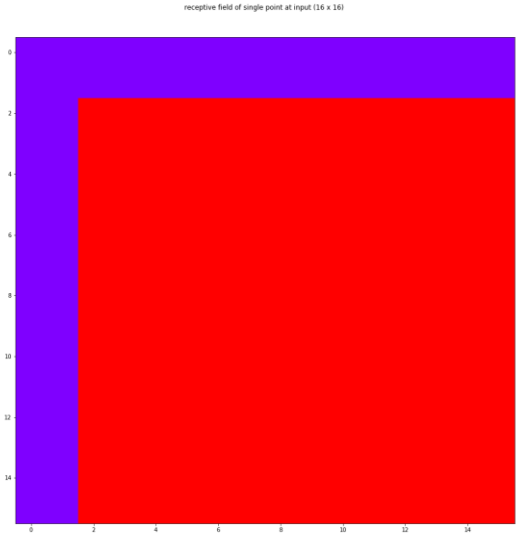
 def forward(self, x):
  y = self.conv(x)
  y = self.bn(y)
  y = self.relu(y)
  y = self.maxpool(y)
  return y
```

```
----------------------------------------------------------------
     Layer (type)   map size     start      jump receptive_field
================================================================
        0        [256, 256]      0.5       1.0         1.0
        1        [128, 128]      0.5       2.0         7.0
        2        [128, 128]      0.5       2.0         7.0
        3        [128, 128]      0.5       2.0         7.0
        4         [64, 64]       0.5       4.0        11.0
================================================================
Receptive field size for layer 2, unit_position (1, 1),  is
 [(0, 6.0), (0, 6.0)]
```

Lin, Haoning, Zhenwei Shi, and Zhengxia Zou. "Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network." Remote sensing 9.5 (2017): 480.

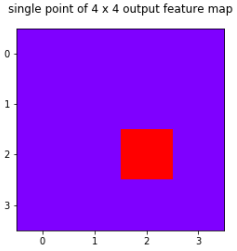
RECEPTIVE FIELD AS WE GO DEEPER

simple script to find input pixels that can affect output pixels for a specific CNN architecture



inverse image

receptive field



this could also be computed by hand by book-keeping the inverse image of each conv2D and pool layer

pytorch-receptive-field

POPULAR CNN ARCHITECTURES/PATTERNS

There are pretrained ImageNet models in PyTorch

("model-zoo")

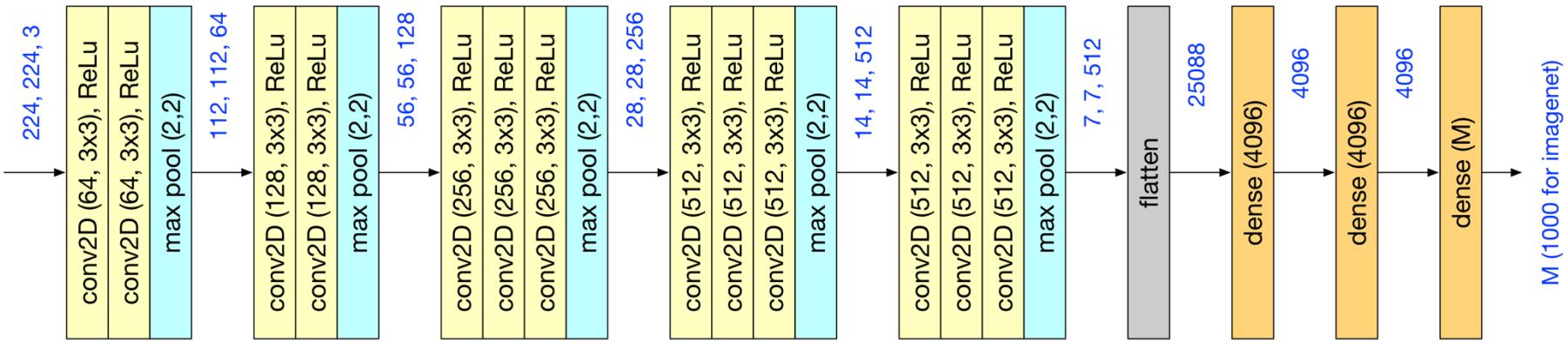
```
import torchvision.models as models
resnet18 = models.resnet18(pretrained=True)
alexnet = models.alexnet(pretrained=True)
squeezenet = models.squeezenet1_0(pretrained=True)
vgg16 = models.vgg16(pretrained=True)
densenet = models.densenet161(pretrained=True)
inception = models.inception_v3(pretrained=True)
googlenet = models.googlenet(pretrained=True)
shufflenet = models.shufflenet_v2_x1_0(pretrained=True)
mobilenet = models.mobilenet_v2(pretrained=True)
resnext50_32x4d = models.resnext50_32x4d(pretrained=True)
wide_resnet50_2 = models.wide_resnet50_2(pretrained=True)
mnasnet = models.mnasnet1_0(pretrained=True)
```

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

https://pytorch.org/docs/stable/torchvision/models.html

COMMON CNN ARCHITECTURE PATTERNS - VGG16



Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

COMMON CNN ARCHITECTURE PATTERNS – RESNET(S)

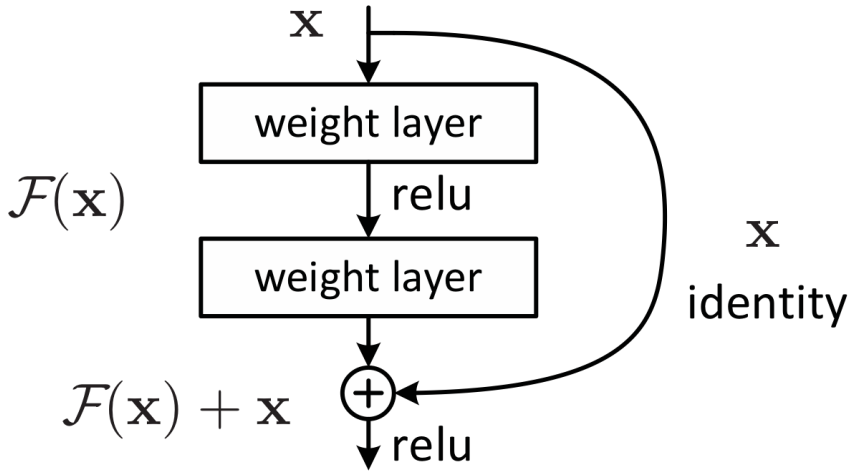


Figure 2. Residual learning: a building block.

residual connections:

make very deep networks trainable (**deeper ones no longer fit worse**)

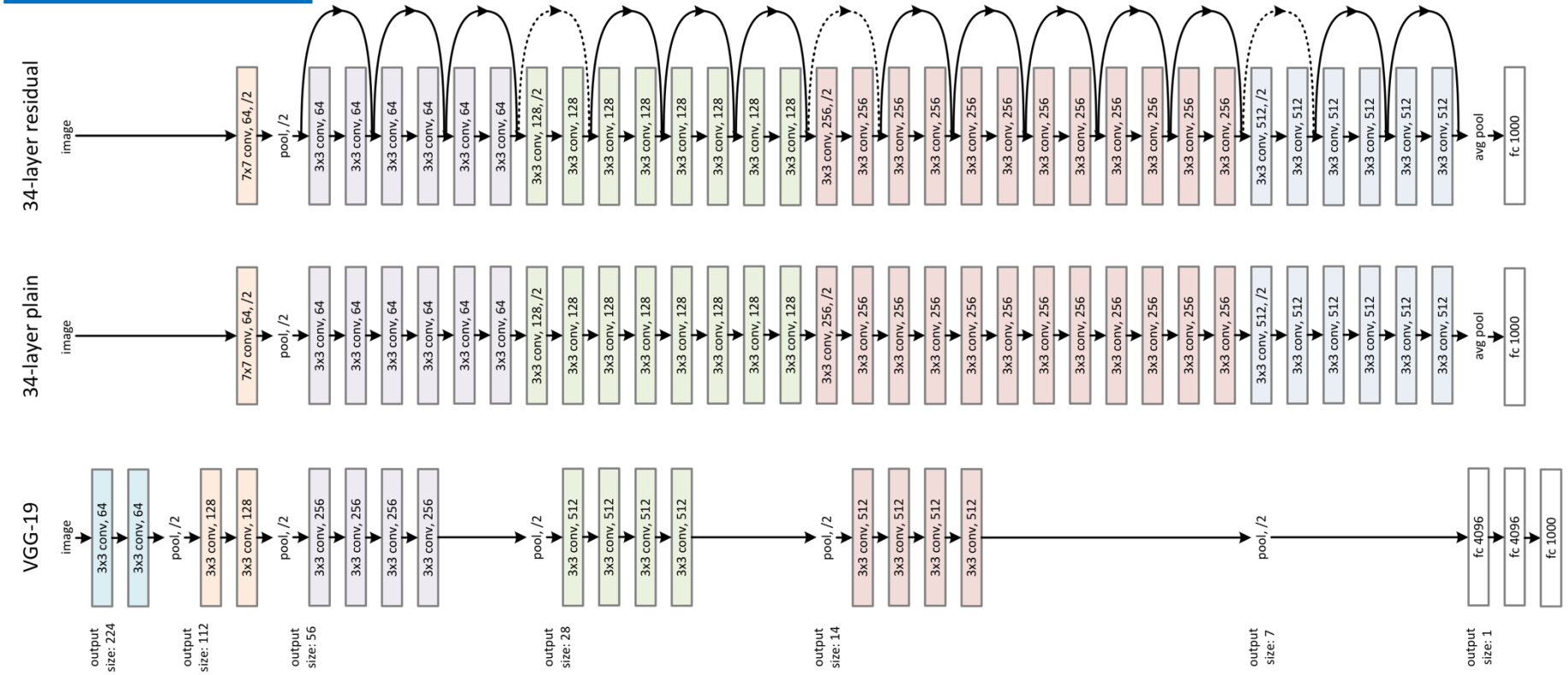
allow learning of "alternative" networks

— *e.g.*, can learn to bypass the two "weight layers" in this figure

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

COMMON CNN ARCHITECTURE PATTERNS – RESNET(S)

ResNet34



He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

COMMON CNN ARCHITECTURE PATTERNS – RESNET(S)

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

Note:

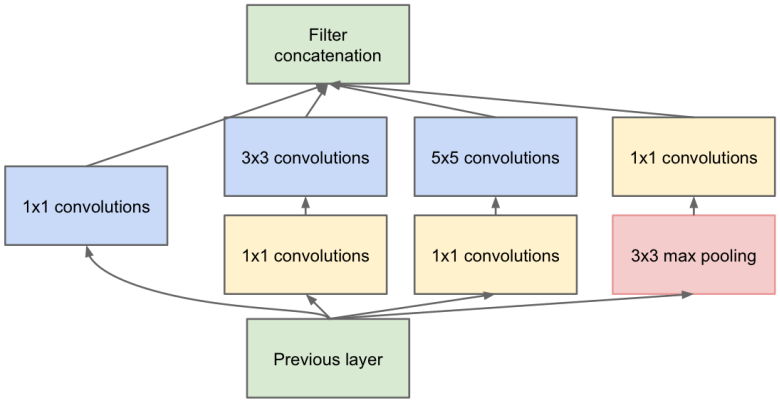
there are *v2* versions of these

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

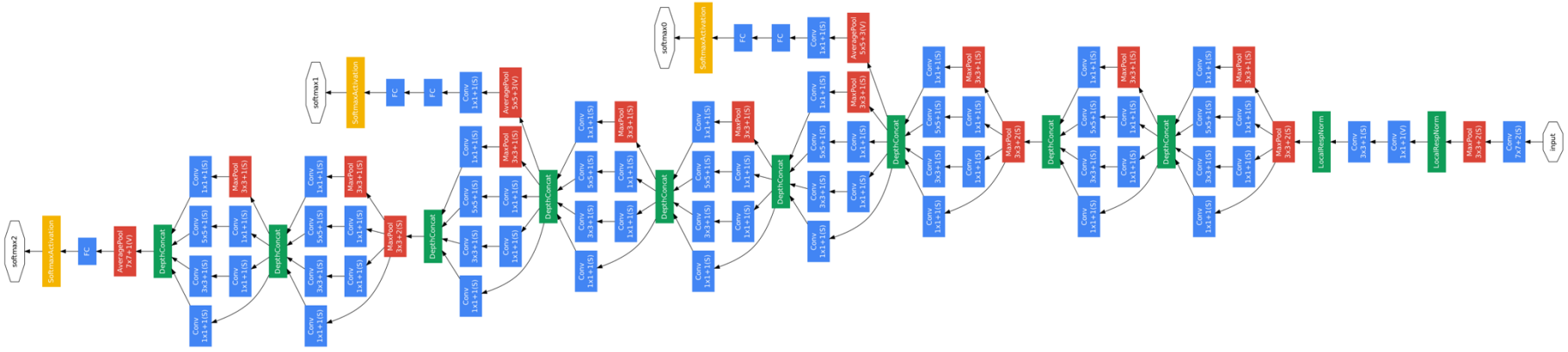
COMMON CNN ARCHITECTURE PATTERNS - INCEPTION

aka GoogLeNet



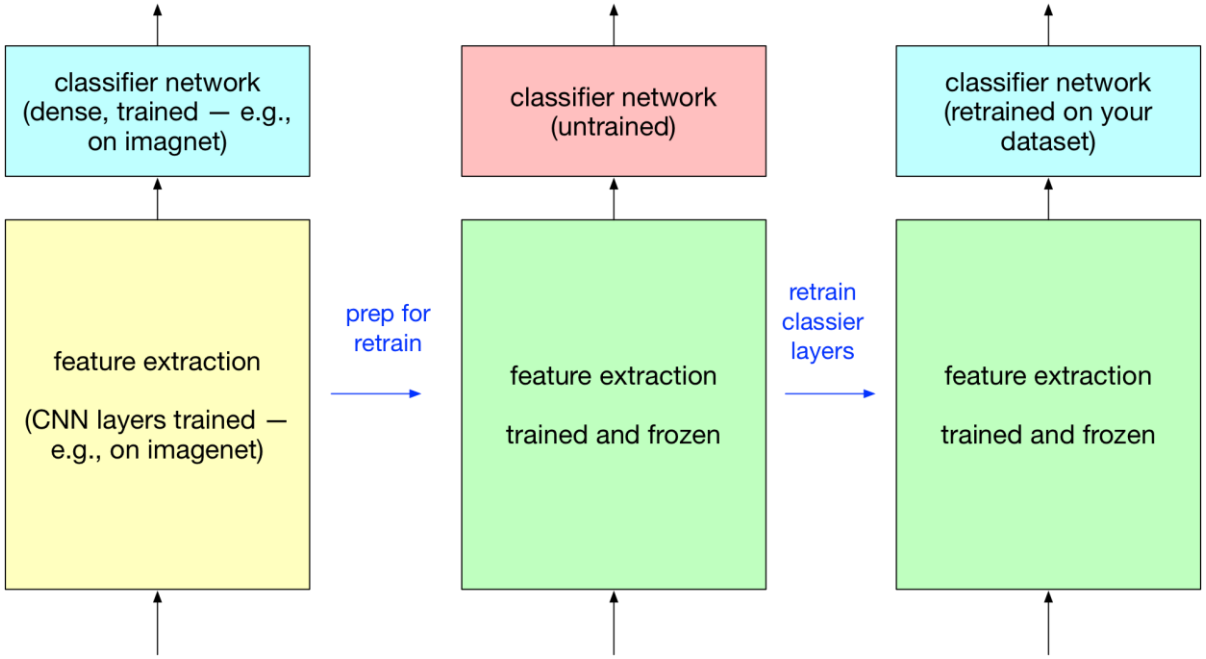


(b) Inception module with dimensionality reduction



Szegedy, Christian, et al. "Going deeper with convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.

USING FIXED CNN LAYERS FOR A DIFFERENT CV TASK

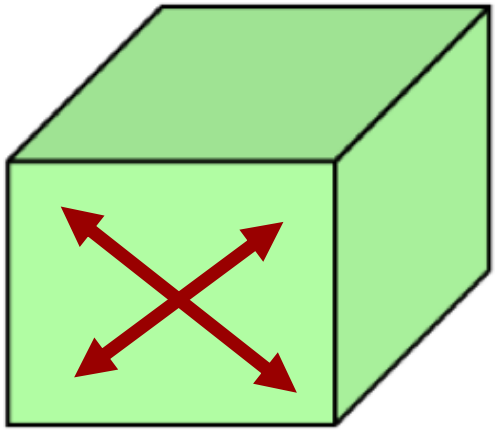


features needed for many CV tasks are similar to Imagenet classification features

you can reuse all or part of the feature extraction network

```
import torchvision.models as models
model = models.resnet50(pretrained=True)
```

ONE LAST LAYER TYPE: GLOBAL POOLING



pool over the pixels in a channel

```
torch.nn.MaxPool2d(kernel_size=image_size)
```

```
torch.nn.AvgPool2d(kernel_size=image_size)
```

follow with: `x.squeeze()`

Input: 4D tensor with shape (batch_size, channels, rows, cols)

Output: 2D tensor with shape (batch_size, channels)

this is used after the last conv2D/pool layer before the "flatten" in many recent models

reduces the complexity of the dense classification network without sacrificing performance



OUTLINE FOR SLIDES

- Motivation, applications
- Basic 2D convolution operations
  - PyTorch 2Dconv layer
- Pooling and stride
- Fashion MNIST example
- Visualization methods
- Some common CNN structures
- Reduced complexity CNN architectures
- Outline of Back-propagation for CNNs



REDUCING COMPLEXITY



REDUCED PARAMETER/COMPUTATION APPROACHES

For larger CNNs, the number of parameters is so large, that **storage complexity** becomes a significant issue

this is an issue for running these models in inference mode on mobile devices

computational complexity (during inference and training) is also an issue

there has been a lot of work on reducing the storage and computational complexity of CNNs – most have focused on inference of trained models

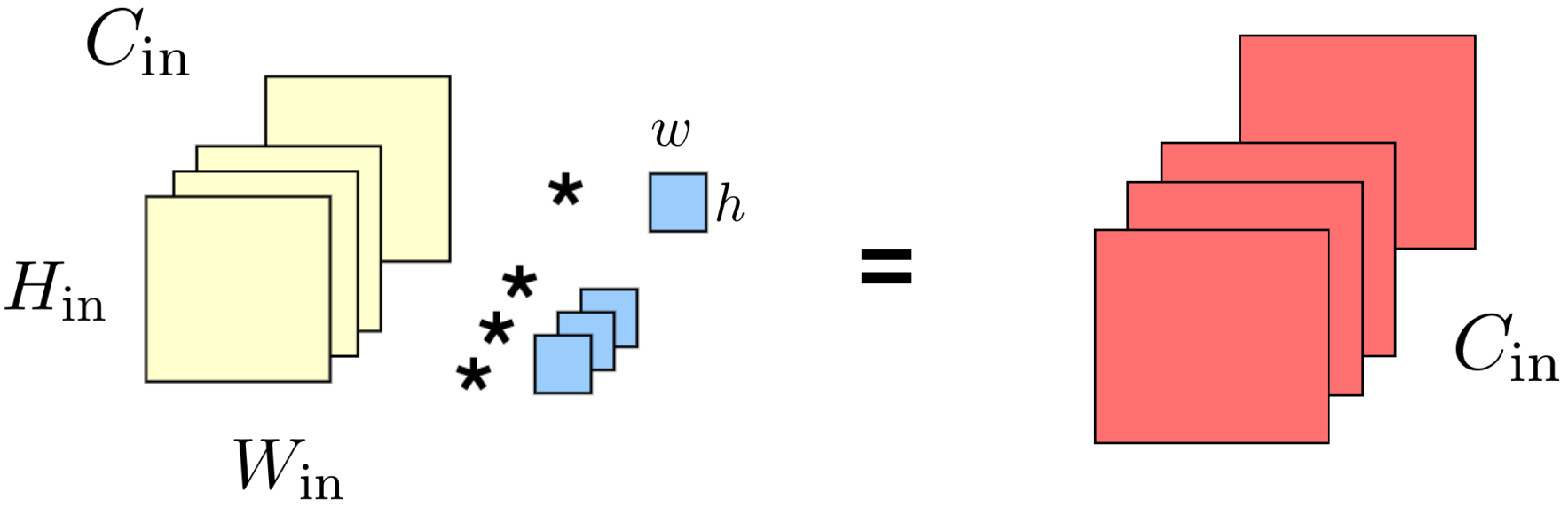
REDUCED PARAMETER/COMPUTATION APPROACHES

Two primary methods:

constrained filter structures: alter the standard conv2D operations to lower the computational/storage complexity

post-training processing to reduce complexity

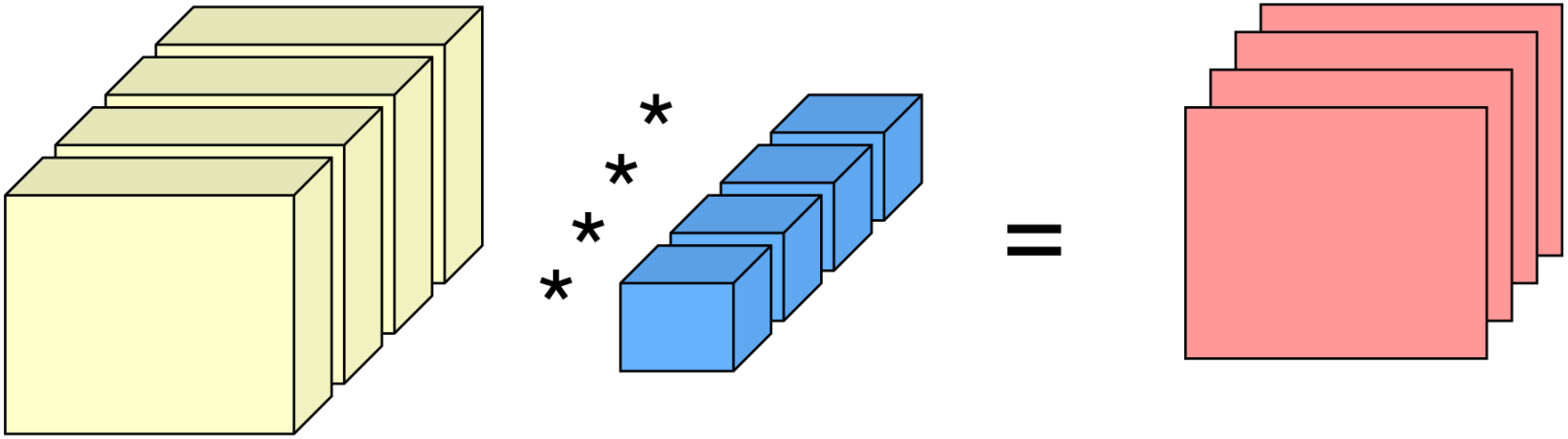
CONSTRAINED FILTERING: DEPTH-WISE CONVOLUTION



only do convolution **separately for channels**

— *i.e.*, no information is mixed across channels

CONSTRAINED FILTERING: GROUPWISE CONVOLUTION

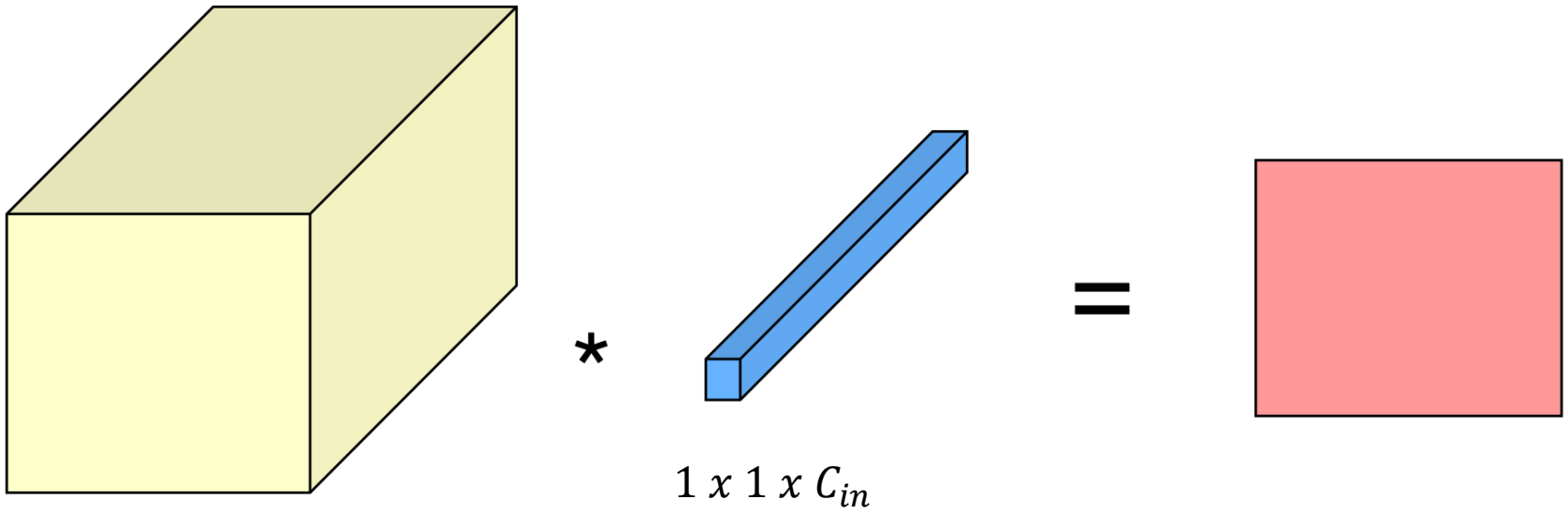


trade-off between standard conv2D filtering and depth-wise filtering

use more of these grouped-filters to get more output channels

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

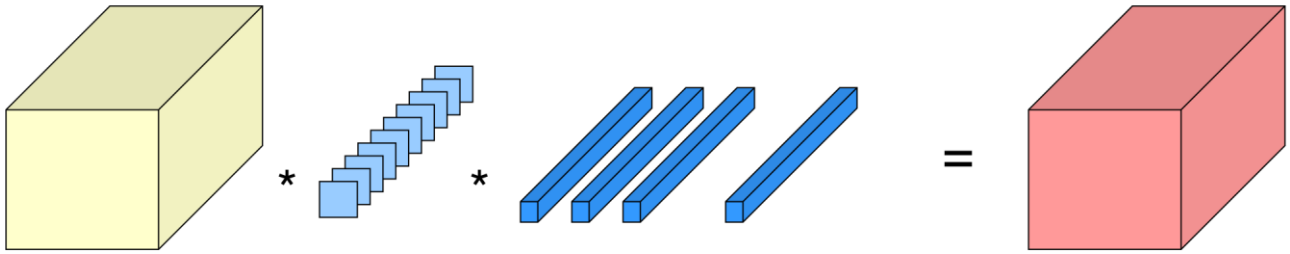
CONSTRAINED FILTERING: POINTWISE CONVOLUTION



standard Conv2D with filter size 1x1

***a.k.a.*, 1x1 convolution**

EXAMPLE: MOBILENET



$$N_{\text{filters}} = C_{\text{out}}$$

combine depth-wise convolution with many 1x1 convolutions

compare with standard Conv2D:

$$C_{\text{out}} = 32$$
$$C_{\text{in}} = 16$$
$$H_{\text{in}} = 64$$
$$W_{\text{in}} = 64$$
$$h = w = 3$$

4,640 parameters with standard approach

16, 3x3 depth-wise kernels: 144

32, 1x1 point-wise filters: 512

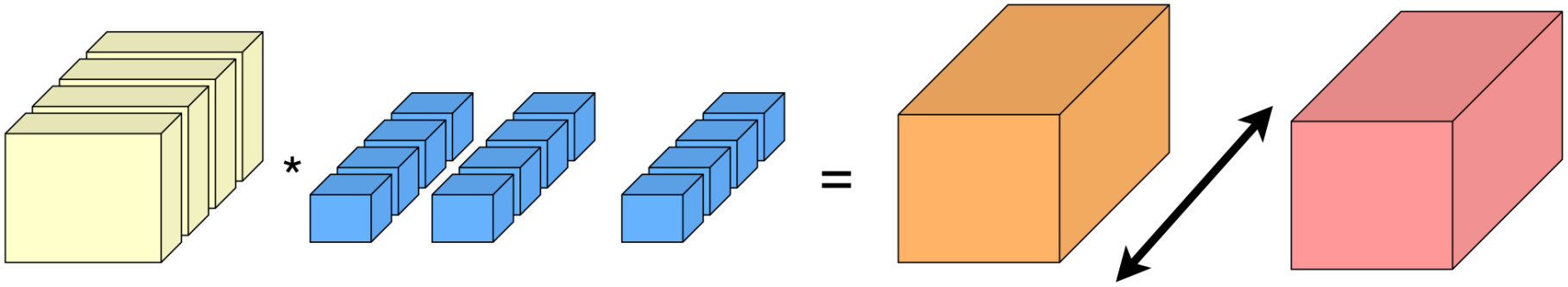
32, biases: 32

688 parameters for same output feature map size

Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.

EXAMPLE: SHUFFLENET



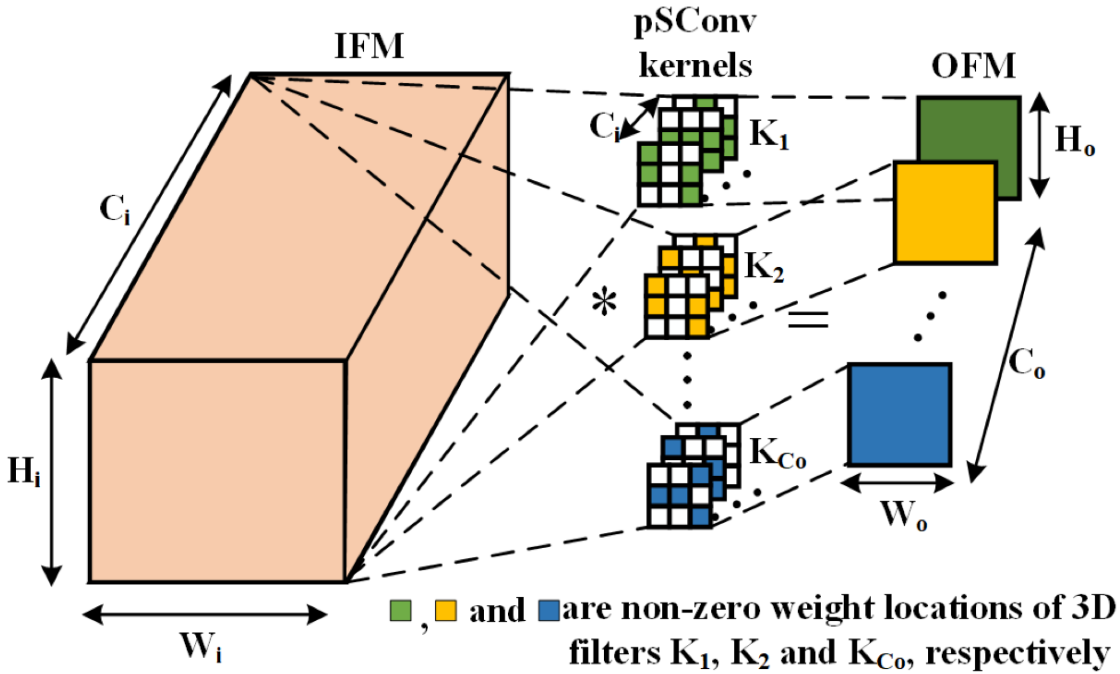
multiple grouped convolutions

shuffle across channels

group-wise convolutions with shuffling

Zhang, Xiangyu, et al. "Shufflenet: An extremely efficient convolutional neural network for mobile devices." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018. APA

EXAMPLE: PRE-DEFINED SPARSITY



pre-define some of the filter coefficients to be zero and **hold fixed** through training and inference

targets specialized hardware acceleration — project concept is to map this to GPU

Kundu, Souvik, et al. "Pre-defined Sparsity for Low-Complexity Convolutional Neural Networks." IEEE Transactions on Computers (2020).

EXAMPLE: PRE-DEFINED SPARSITY

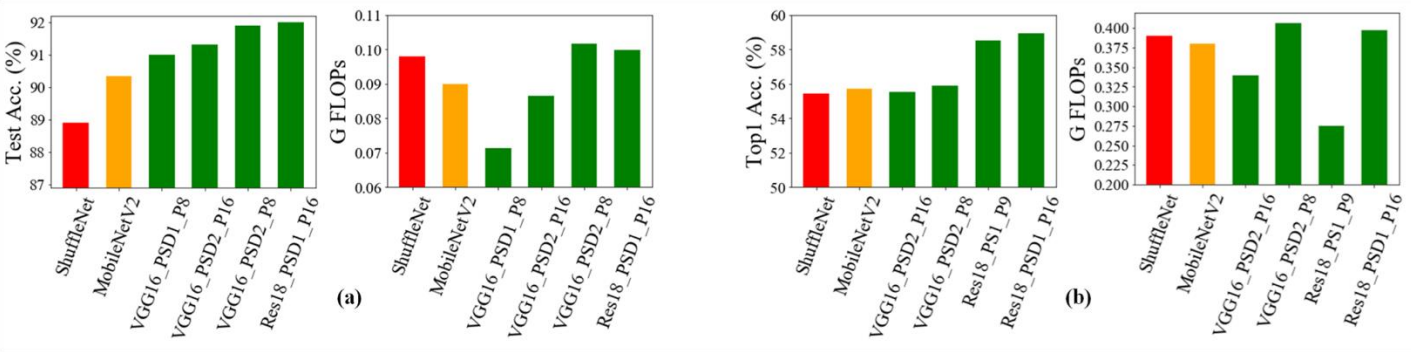


Fig. 11: Performance comparison of our proposed architectures that have similar or fewer FLOPs than ShuffleNet and MobileNetV2 with comparable or better classification accuracy on (a) CIFAR-10 and (b) Tiny ImageNet.

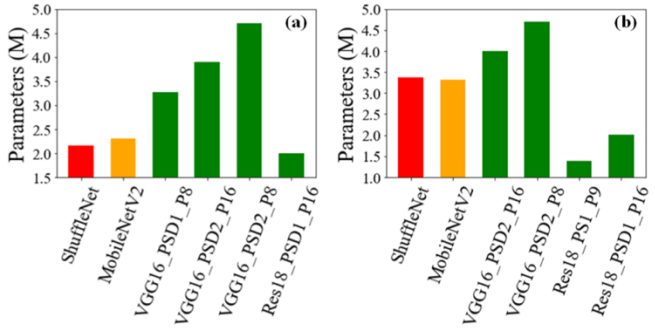


Fig. 12: Comparison of the number of model parameters of the network models described in Fig 11 for (a) CIFAR-10 and (b) Tiny ImageNet datasets.

Kundu, Souvik, et al. "Pre-defined Sparsity for Low-Complexity Convolutional Neural Networks." IEEE Transactions on Computers (2020).

POST-TRAINING APPROACHES

post-training processing to minimize complexity

Pruning: set near-zero weights to zero, fix these and do some retraining

Yang, Tien-Ju, Yu-Hsin Chen, and Vivienne Sze. "Designing energy-efficient convolutional neural networks using
energy-aware pruning." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017.

Quantization: map similar valued weights to the same value to save storage

Zhou, Aojun, et al. "Incremental network quantization: Towards lossless CNNs with low-precision weights." arXiv preprint arXiv:1702.03044 (2017).

Binaryization: find a set of binary weights that best approximate the trained network behavior

Rastegari, Mohammad, et al. "Xnor-net: Imagenet classification using binary convolutional neural networks." European conference on computer vision. Springer, Cham, 2016.



OUTLINE FOR SLIDES

- Motivation, applications
- Basic 2D convolution operations
  - PyTorch 2Dconv layer
- Pooling and stride
- Fashion MNIST example
- Visualization methods
- Some common CNN structures
- Reduced complexity CNN architectures
- Back-propagation for CNNs



CNN BACK PROPAGATION

BACK-PROPAGATION IN CNNS

recall the definition of a standard Conv2D operation:

$$y[i,j,k] = \sum_{c} \sum_{(m,n)} h_{c,k}[m,n]x[i+m, j+n, c]$$

$$h_{c,k}[m,n] = \text{2D kernel for input channel } c, \text{output channel } k$$

chain rule:

$$\frac{\partial C}{\partial x[i,j,k]} = \sum_{(i',j',k')} \frac{\partial y[i',j',k']}{\partial x[i,j,k]} \frac{\partial C}{\partial y[i',j',k']}$$

which values of h are involved here?

shorthand:

$$\partial_v[i,j,k] \triangleq \frac{\partial C}{\partial v[i,j,k]}$$

$$\partial_x[i,j,k] \triangleq \sum_{(i',j',k')} \frac{\partial y[i',j',k']}{\partial x[i,j,k]} \delta_y[i',j',k']$$

BACK-PROPAGATION IN CNNS

Let's start with the 2D convolution only…

$$y[i',j'] = \sum_{(m,n)} h[m,n]x[i'+m,j'+n]$$

$$= \sum_{(s,t)} h[s-i',t-j']x[s,t]$$

$$s = i' + m$$

$$t = j' + n$$

$$\delta_x[i,j] = \sum_{(i',j')} \frac{\partial y[i',j']}{\partial x[i,j]} \delta_y[i',j']$$

chain-rule term:

$$\frac{\partial y[i',j']}{\partial x[i,j]} = h[i-i',j-j']$$

$$\delta_x[i,j] = \sum_{(i',j')} h[i-i',j-j']\delta_y[i',j']$$

$$= \sum_{(m,n)} h[-m,-n]\delta_y[i+m,j+n]$$

$$m = i' - i$$

$$n = j' - j$$

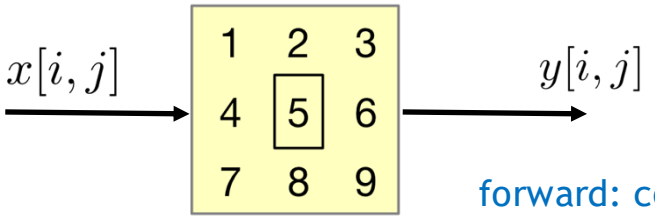
BACK-PROPAGATION IN CNNS

$$y[i,j] = \sum_{(m,n)} h[m,n]x[i+m, j+n]$$

forward: convolve with $h[i,j]$

$$\delta_x[i,j] = \sum_{(m,n)} h[-m,-n]\delta_y[i+m, j+n]$$

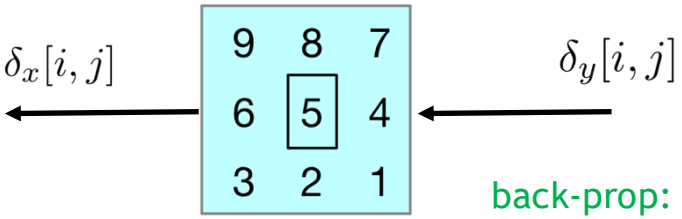
back-prop: convolve with $h[-i,-j]$



forward: convolve with $h[i,j]$

recall: W-transpose in MLP-BP

$$\boldsymbol{\delta}^{(l)} = \dot{\mathbf{a}}^{(l)} \left[\left(\mathbf{W}^{(l+1)} \right)^T \boldsymbol{\delta}^{(l+1)} \right]$$



back-prop: convolve with $h[-i,-j]$

BACK-PROPAGATION IN CNNS

this extends to the standard Conv2D convolution

$$y[i',j',k'] = \sum_{k} \sum_{(m,n)} h_{k,k'}[m,n]x[i'+m, j'+n, k]$$

$$\delta_x[i,j,k] = \sum_{(i',j',k')} \frac{\partial y[i',j',k']}{\partial x[i,j,k]} \delta_y[i',j',k']$$

$i = i' + m$
$j = j' + n$

$$\frac{\partial y[i',j',k']}{\partial x[i,j,k]} = h_{k,k'}[i-i', j-j']$$

$$\delta_x[i,j,k] = \sum_{(i',j',k')} h_{k,k'}[i-i', j-j']\delta_y[i',j',k']$$

$$= \sum_{(m,n,k')} h_{k,k'}[-m,-n]\delta_y[i+m, j+n, k']$$

standard 2DConv with ***reflected*** 2D kernels

$m = i' - i$
$n = j' - j$

BACK-PROPAGATION IN CNNS: POOLING

average pooling:

forward: Q "pixels" averaged

back-prop: $1/Q$ times the gradient flows back through these Q "pixels"

results from standard differentiation

max pooling:

forward: max over Q "pixels" $(i^*, j^*) \sim \text{argmax}$

back-prop: gradient flows directly through (i^*, j^*) only

non-differentiable.... just a convention that works!



CNN/CV RELATED TOPICS

Image segmentation (*e.g.*, U-Net)

Object Detection (*e.g.*, YOLO)

GANs (*e.g.*, “deep fakes”)