Deep Learning Crash Course

Hui Xue

Fall 2021

www.deeplearningcrashcourse.org

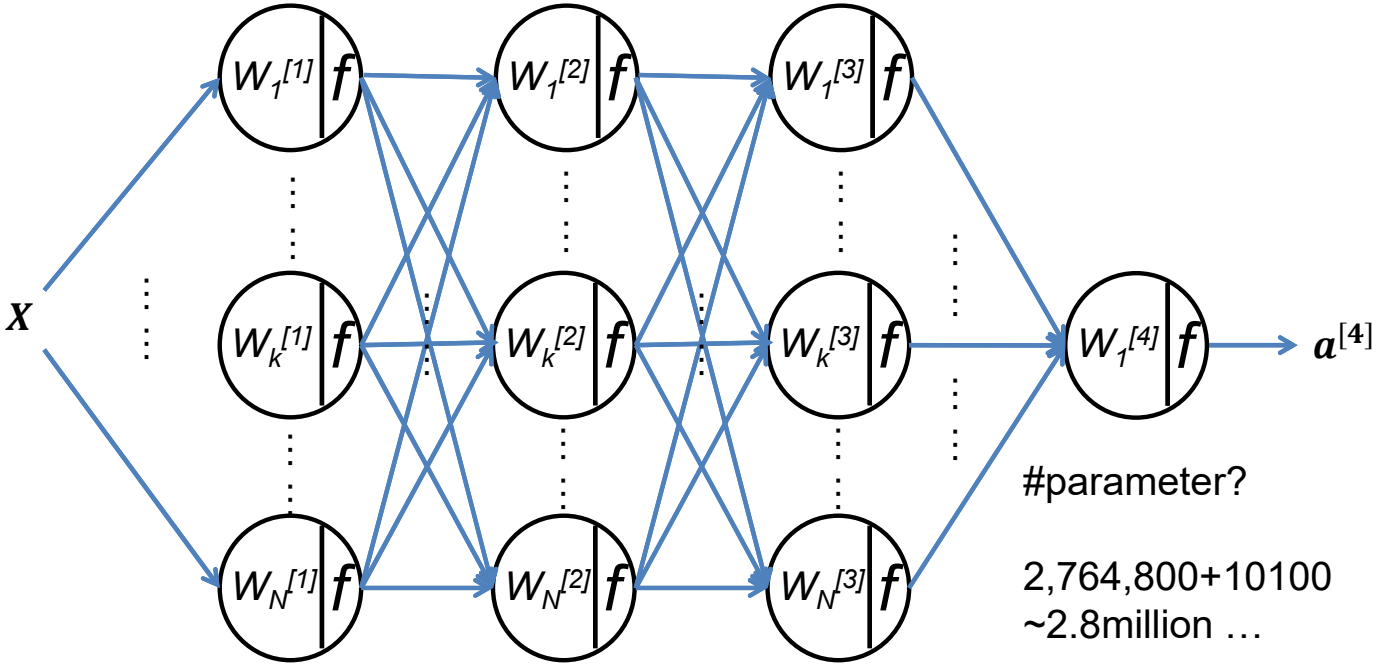
Outline

- CNN basics
- Variation of convolution
- Batch Normalization
- CNN Examples: LeNet and Alex-Net

Revisit image classification

192x144x1

flatten



#parameter?

2,764,800+10100
~2.8million …

x is a 27648-D vector

For N=100, first weight matrix has the dimension of 100x27648

Revisit image classification

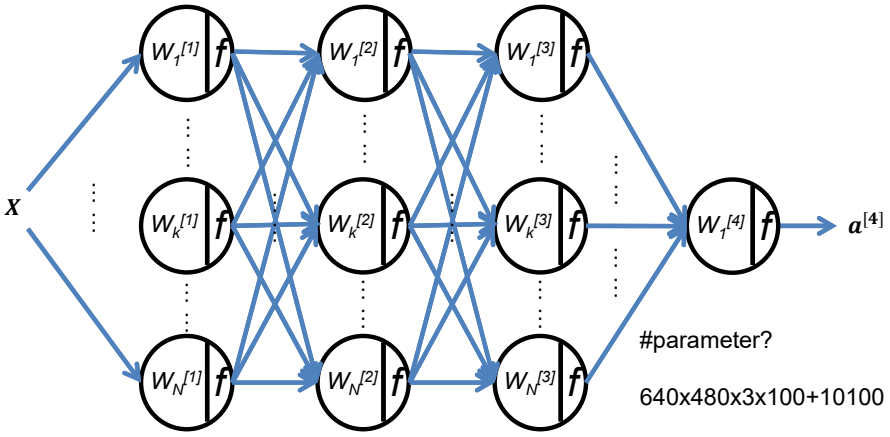
640x480x3

red

green

blue

Flatten and concatenate RGB



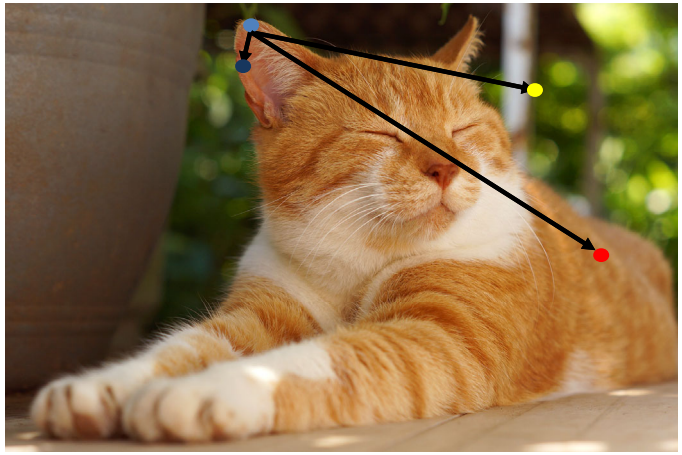
#parameter?

640x480x3x100+10100

~92million …

- FC layer is not efficient for image

FC layer is not efficient for image



- FC layer is not efficient for image
- Too many parameters -> hard to train, easy to get overfitting
- Treat all pixels equally; one pixel is connected to all other pixels in the image
- Does not utilize the spatial information among neighboring pixels

Spatially invariant feature detection

Edge detection

1	0	-1
2	0	-2
1	0	-1

Gradient along row

1	2	1
0	0	0
-1	-2	-1

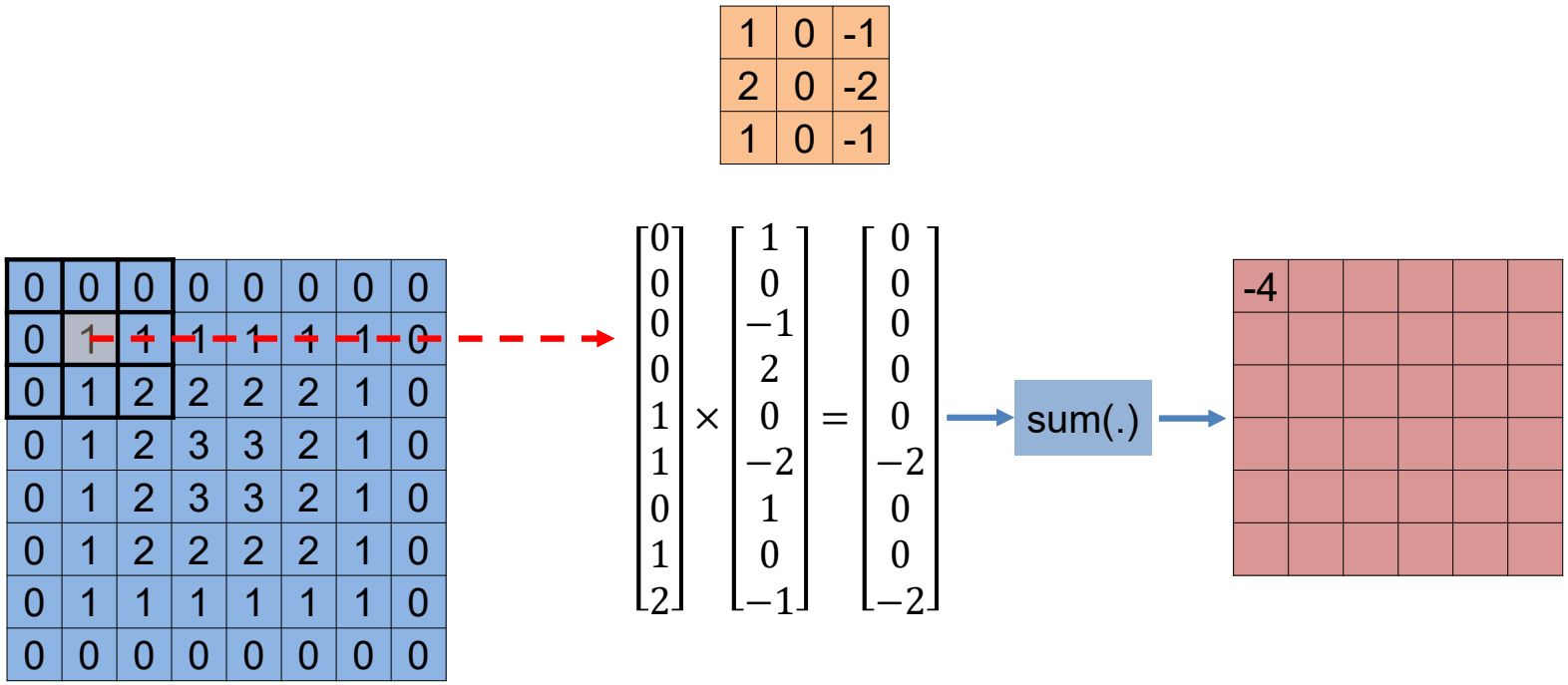
Gradient along column

Sobel filter

- Use the same filter “sliding over” the image
- Convolution
- No need to have different detectors for the same image features → FC method is not optimal

CNN is a spatial-invariant, parameter-sharing FC

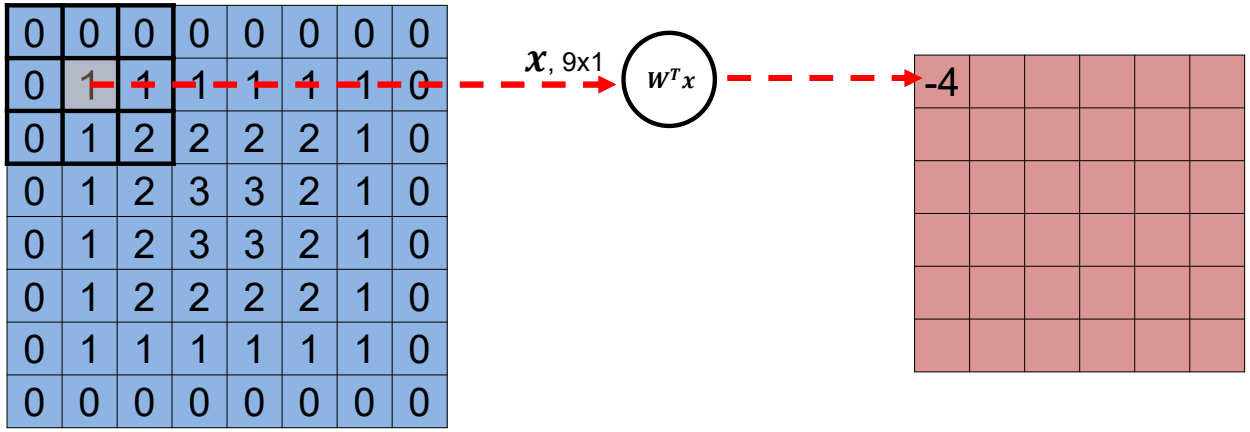
- Use the convolution operation for spatial invariant → *spatial invariant*
- Use the same filter for the whole image → *parameter sharing*



CNN is a spatial-invariant, parameter-sharing FC

- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing

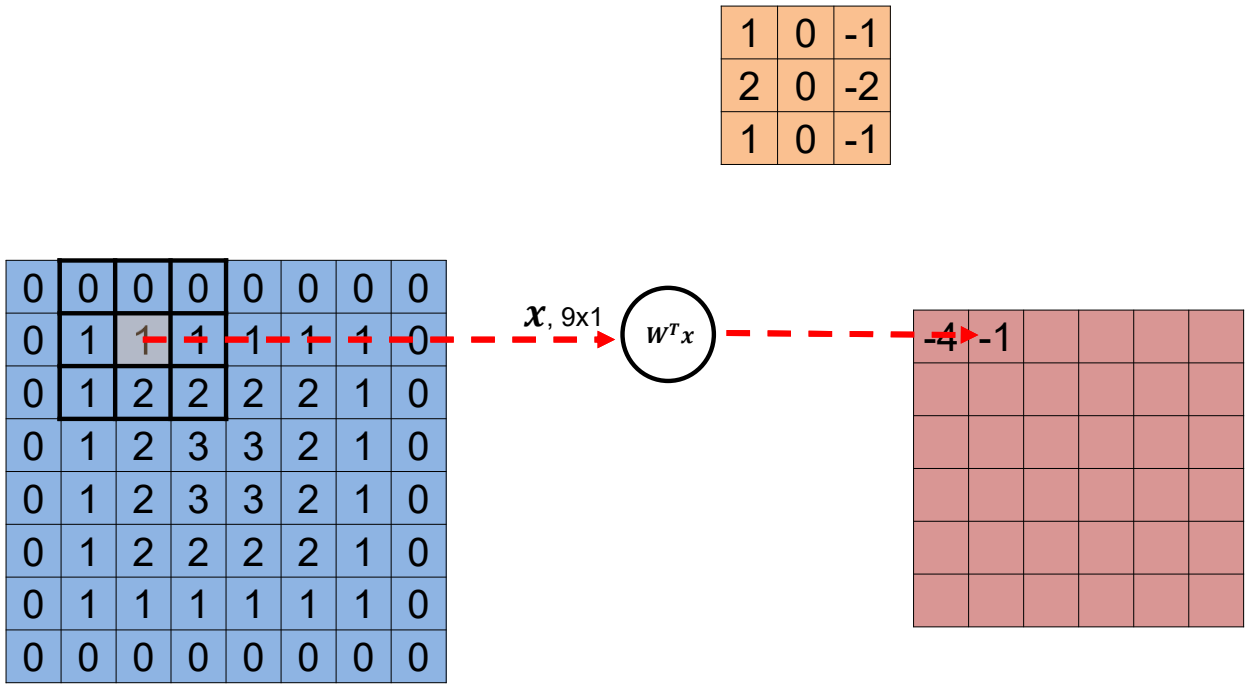
1	0	-1
2	0	-2
1	0	-1



- Take a neighborhood and flatten
- Multiple with kernel and sum it up
- Can be seen as a local FC

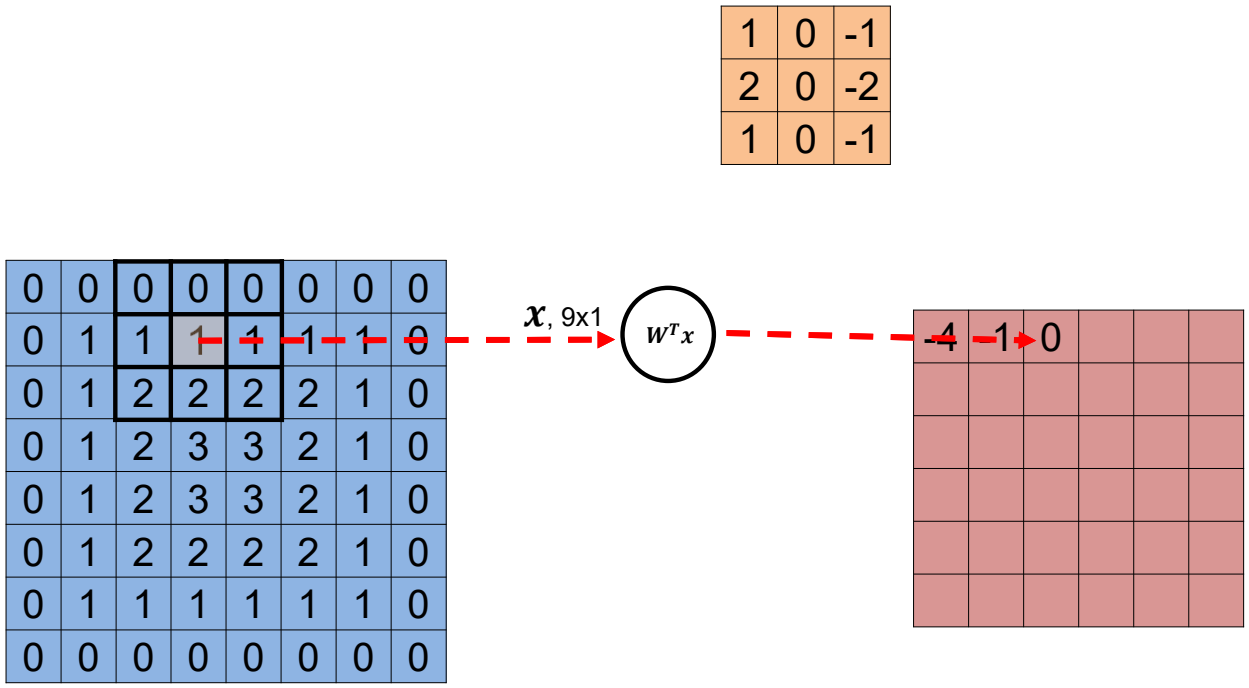
CNN is a spatial-invariant, parameter-sharing FC

- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing



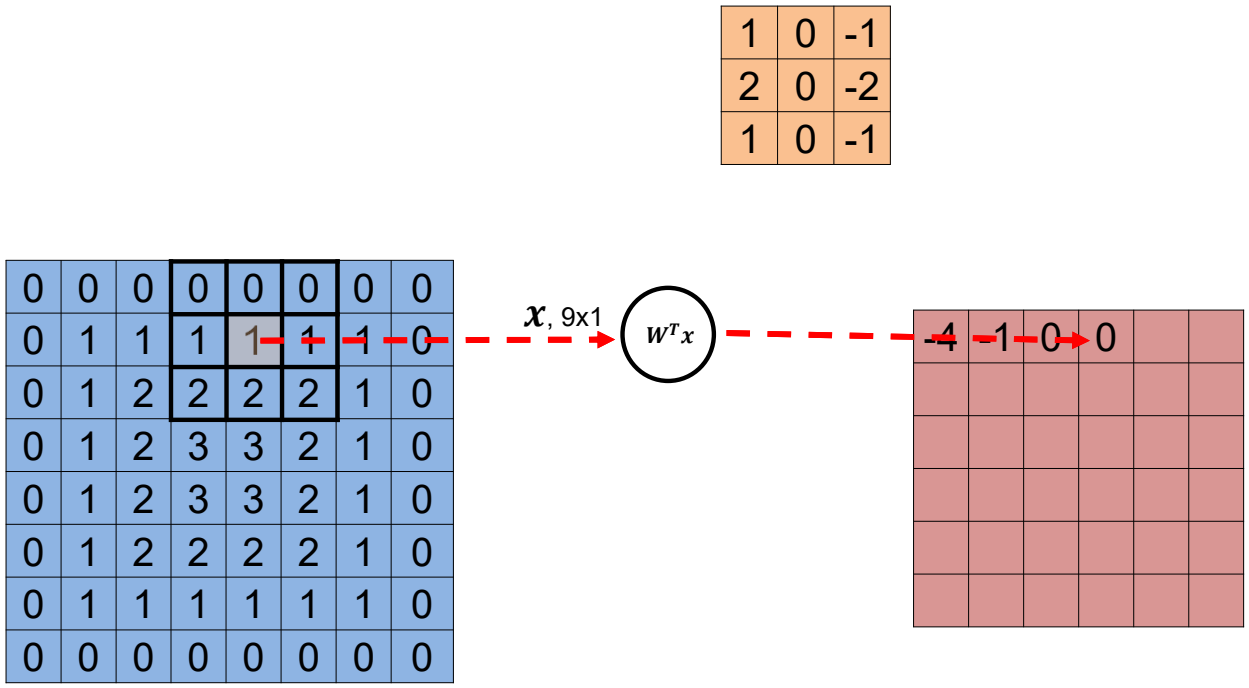
CNN is a spatial-invariant, parameter-sharing FC

- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing



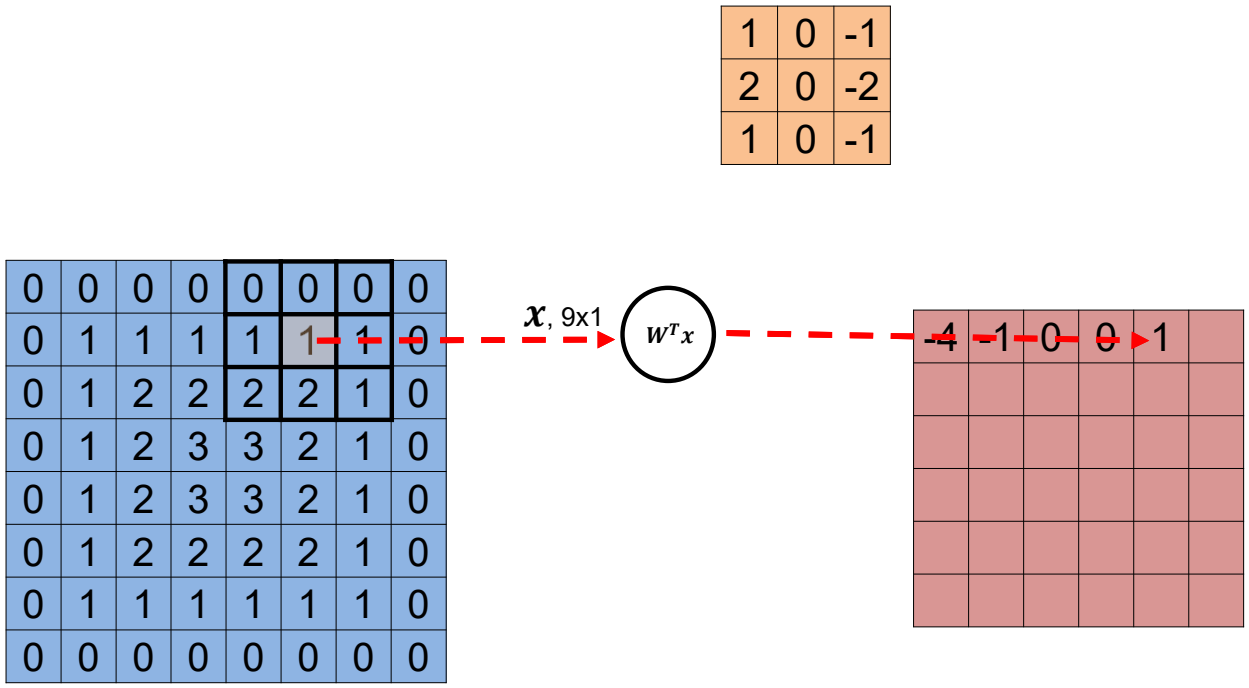
CNN is a spatial-invariant, parameter-sharing FC

- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing



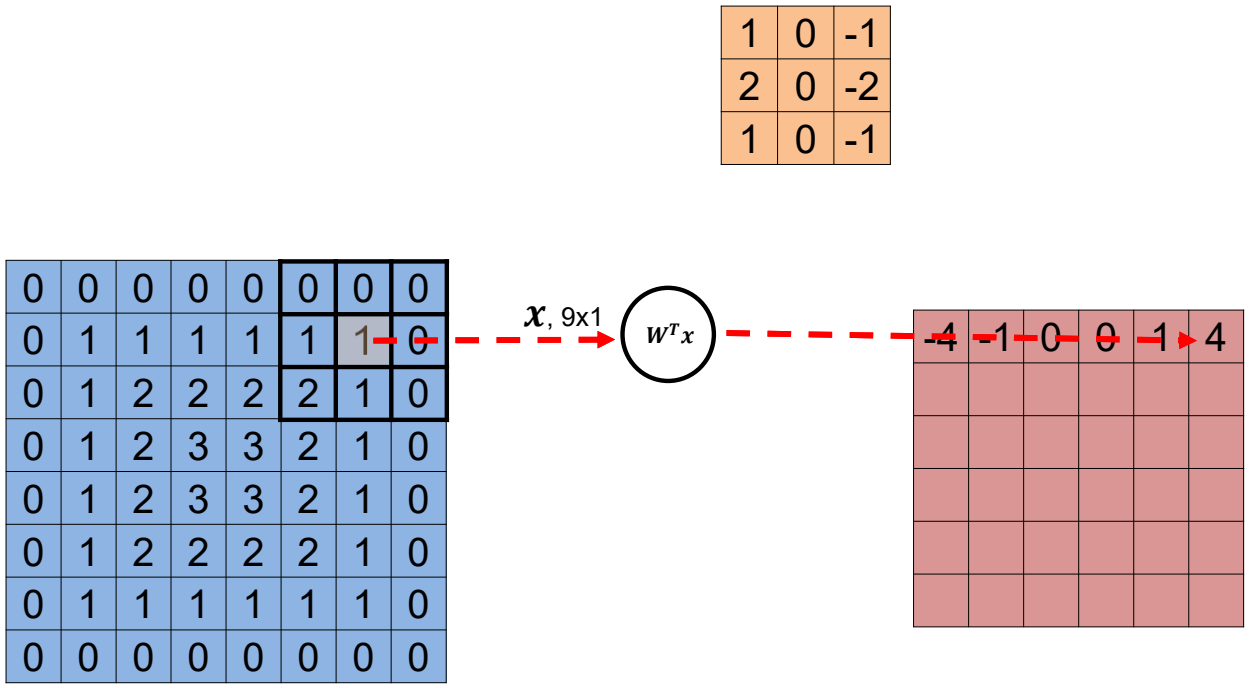
CNN is a spatial-invariant, parameter-sharing FC

- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing



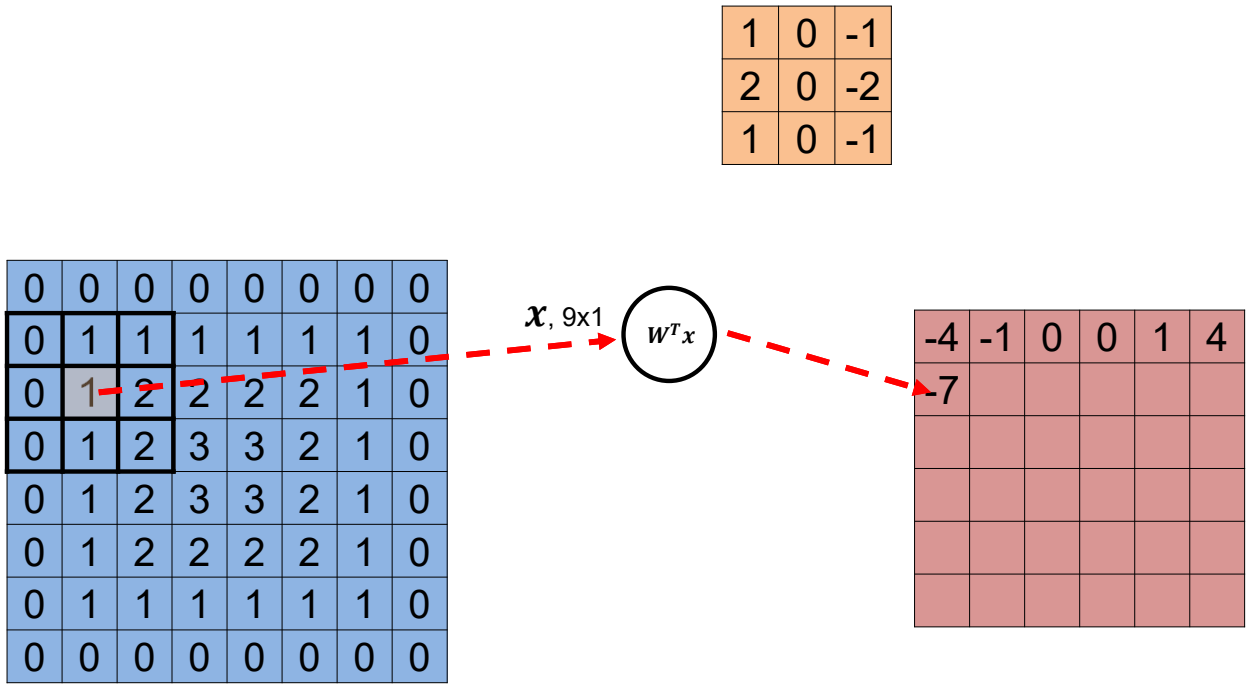
CNN is a spatial-invariant, parameter-sharing FC

- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing



CNN is a spatial-invariant, parameter-sharing FC

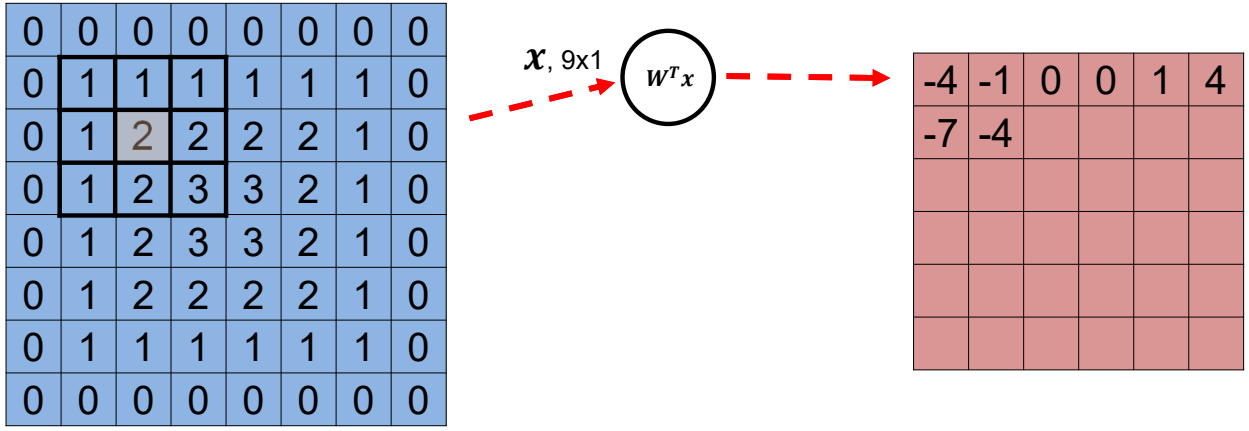
- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing



CNN is a spatial-invariant, parameter-sharing FC

- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing

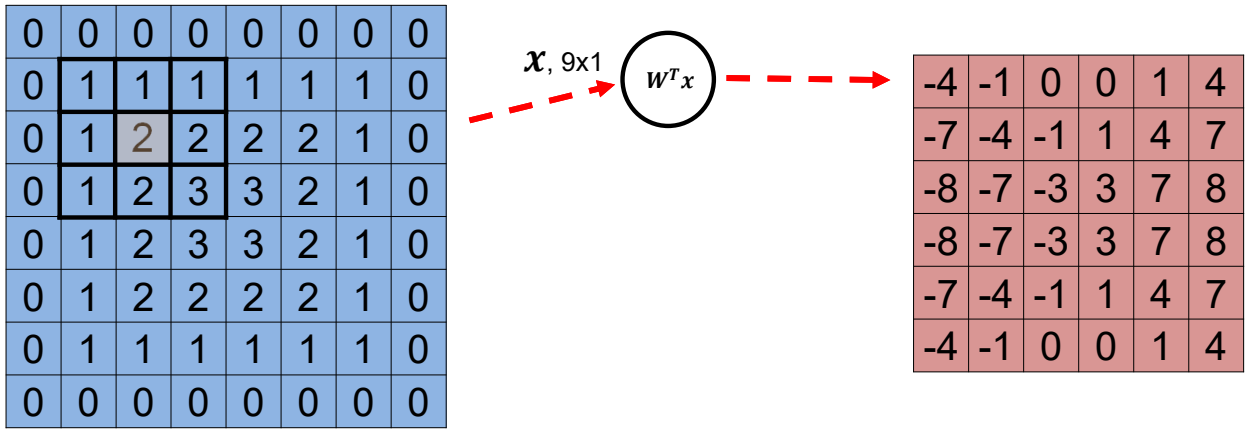
1	0	-1
2	0	-2
1	0	-1



CNN is a spatial-invariant, parameter-sharing FC

- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing

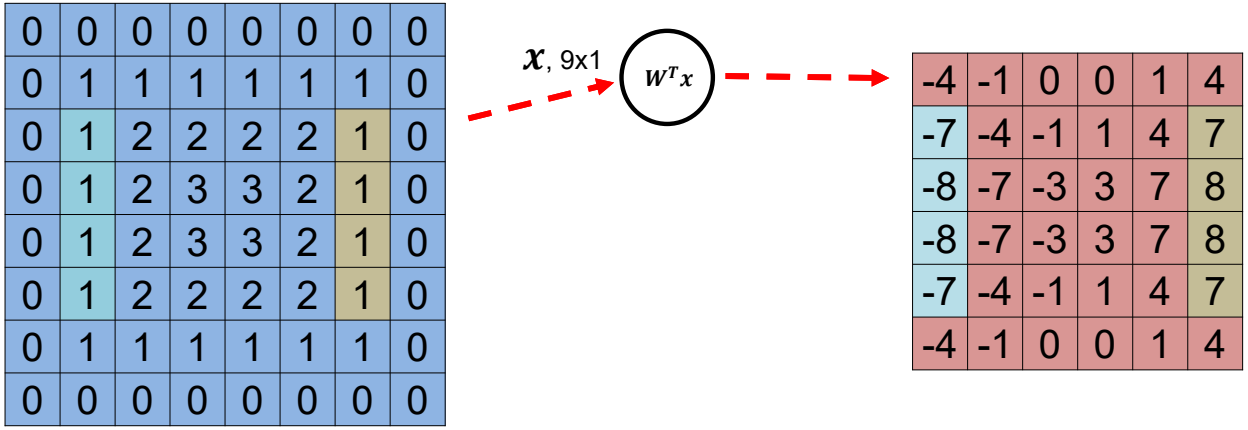
1	0	-1
2	0	-2
1	0	-1



CNN is a spatial-invariant, parameter-sharing FC

- Use the convolution operation for spatial invariant → spatial invariant
- Use the same filter for the whole image → parameter sharing

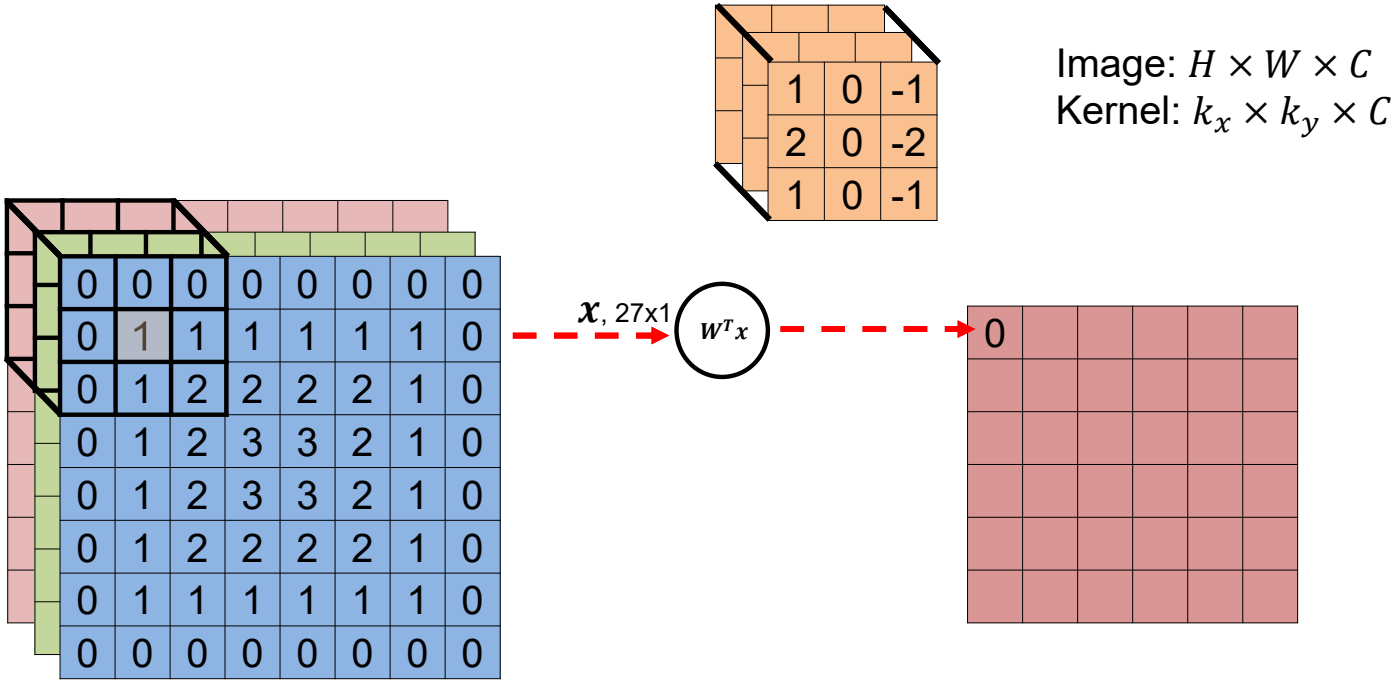
1	0	-1
2	0	-2
1	0	-1



- Higher value in outputs indicate “vertical edges”
- With direction information (gradient direction)

CNN layer with multiple channels

- Kernel has multiple channels too
- Sum over all channels



Padding

- Pixels on the boundary was not explored
- Reduced output size

Padding = 1

0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
0	0	0	0	0	0	0	0	0	*0*
0	0	1	1	1	1	1	1	0	*0*
0	0	1	2	2	2	2	1	0	*0*
0	0	1	2	3	3	2	1	0	*0*
0	0	1	2	3	3	2	1	0	*0*
0	0	1	2	2	2	2	1	0	*0*
0	0	1	1	1	1	1	1	0	*0*
0	0	0	0	0	0	0	0	0	*0*
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

1	0	-1
2	0	-2
1	0	-1

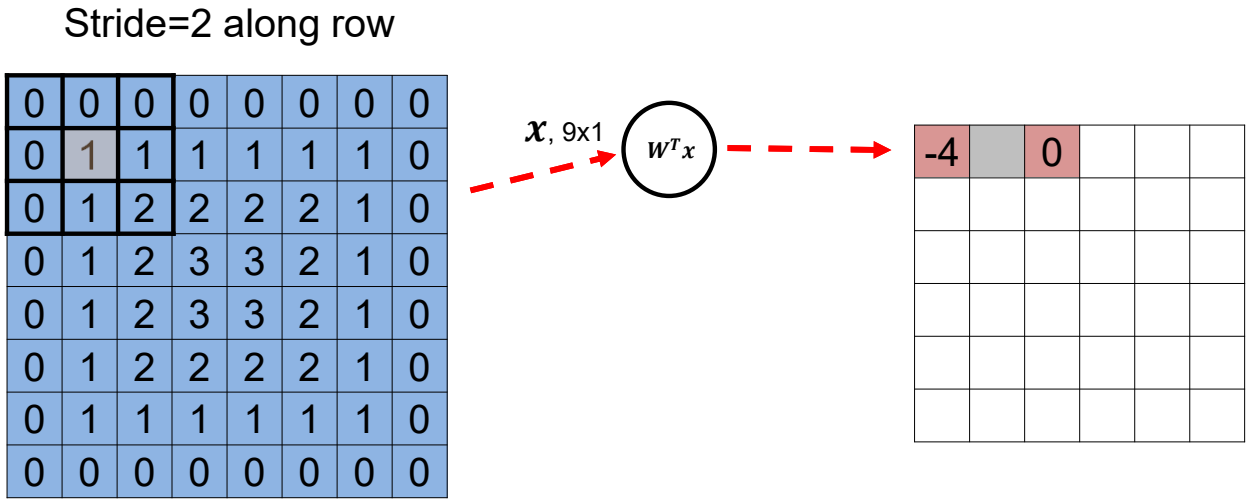
x, 9x1 → $w^T x$ →

-1	-1	0	0	0	0	1	1
-3	-4	-1	0	0	1	4	-3
-4	-7	-4	-1	1	4	7	-4
-4	-8	-7	-3	3	7	8	-4
-4	-8	-7	-3	3	7	8	-4
-4	-7	-4	-1	1	4	7	-4
-3	-4	-1	0	0	1	4	-3
-1	-1	0	0	0	0	1	1

- Often pad with 0
- Other choices : replica, mirror, periodic

Stride

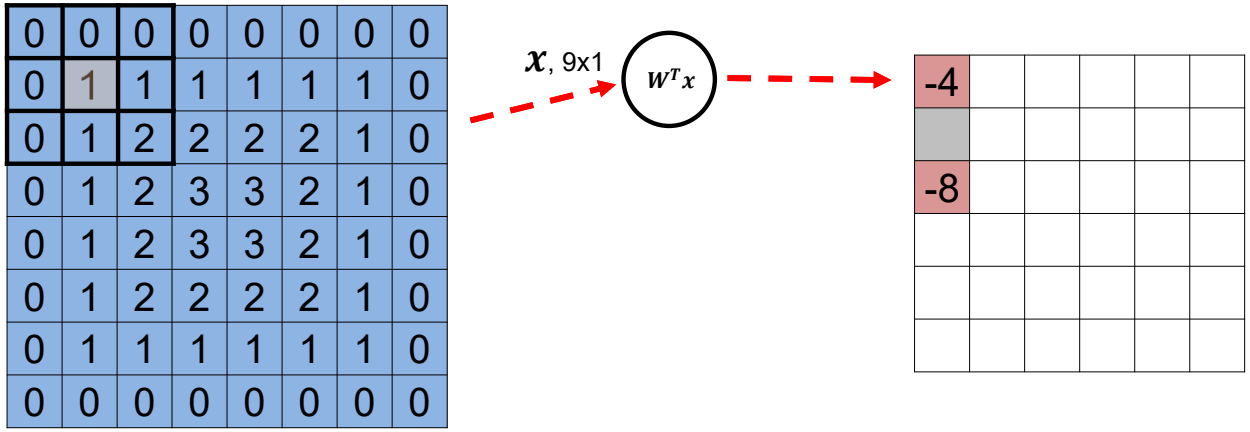
- When sliding the kernel over the image, the moving step-size is the stride
- Default is 1, but stride can be larger than 1
- Often used to reduce output size



Stride

- When sliding the kernel over the image, the moving stepsize is the stride
- Default is 1, but stride can be larger than 1
- Often used to reduce output size

Stride=2 along column

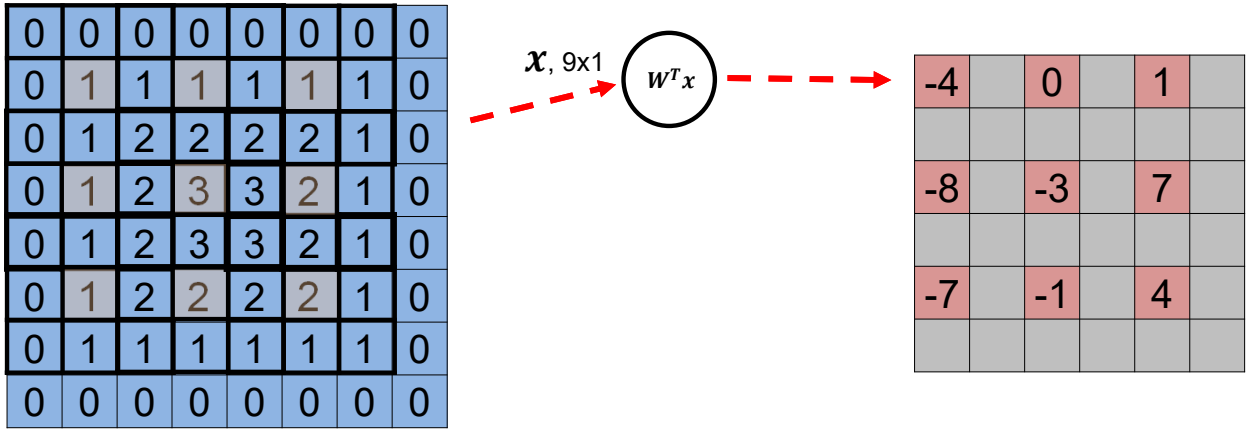


Stride

- When sliding the kernel over the image, the moving stepsize is the stride
- Default is 1, but stride can be larger than 1
- Often used to reduce output size

Input 8x8
Kernel 3x3
Output 3x3

With padding=0, stride=2 for H and W



Determine the output size

Image: $H \times W \times C$
Kernel: $k_x \times k_y \times C$

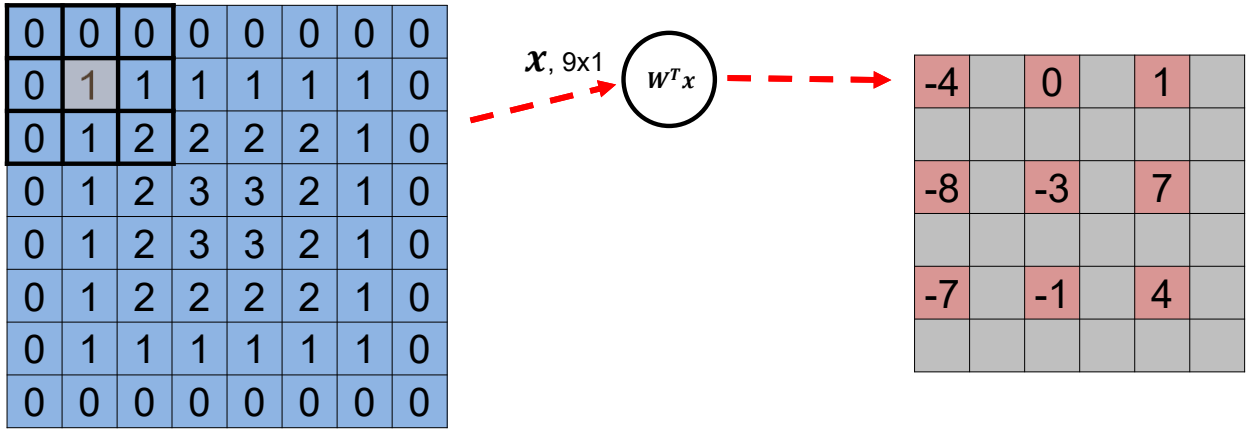
Padding size: p
Stride: s

Output size:

$$H' = \left\lfloor \frac{H + 2 \times p - k_x}{s} \right\rfloor + 1$$

$$W' = \left\lfloor \frac{W + 2 \times p - k_y}{s} \right\rfloor + 1$$

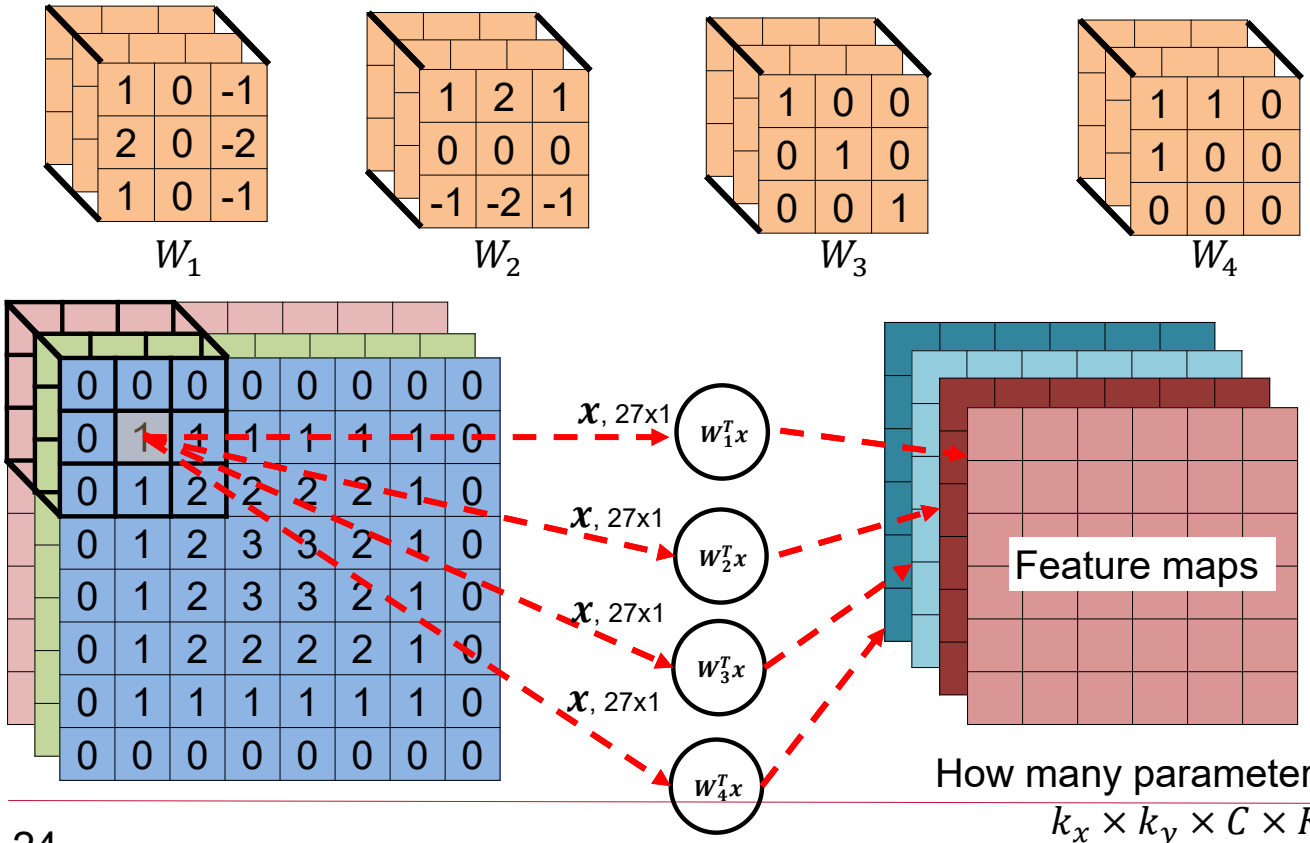
$$\left\lfloor \frac{8 + 2 \times 0 - 3}{2} \right\rfloor + 1 = \lfloor 2.5 \rfloor + 1 = 3$$



- Stride can be used to reduce image size

Multiple kernels

- For one location in the image, multiple kernels can be used to capture different features



How many parameters?

$k_x \times k_y \times C \times K + K$

Image: $H \times W \times C$

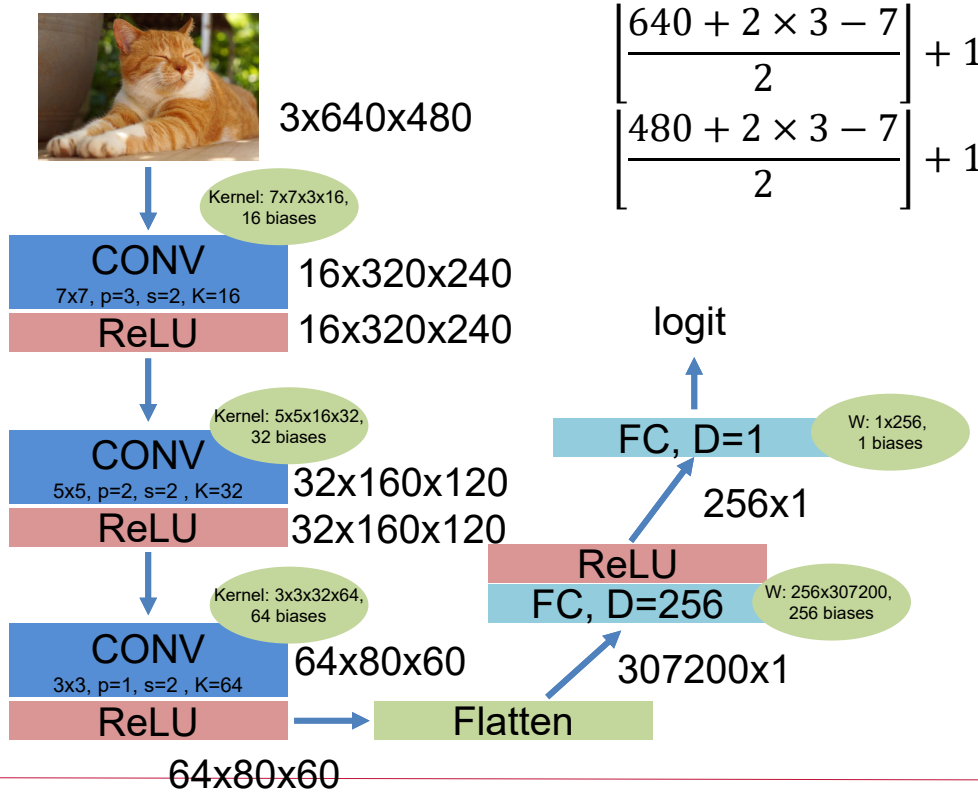
Kernel: $k_x \times k_y \times C \times K$

Output : $H' \times W' \times K$

- These kernels are learned
- C : input channels
- K : output channels
- With K bias parameters; one for each output channel

Multiple CONV layers can be stacked

- Nonlinearity (e.g. ReLU) is applied to every element on feature maps



$$\left\lfloor \frac{640 + 2 \times 3 - 7}{2} \right\rfloor + 1 = 320$$

$$\left\lfloor \frac{480 + 2 \times 3 - 7}{2} \right\rfloor + 1 = 240$$

Total number of parameters:

7x7x3x16+16+5x5x16x32+32+3x3x32x64+64+256x307200+256+1x256+1 = 78,677409, ~78million

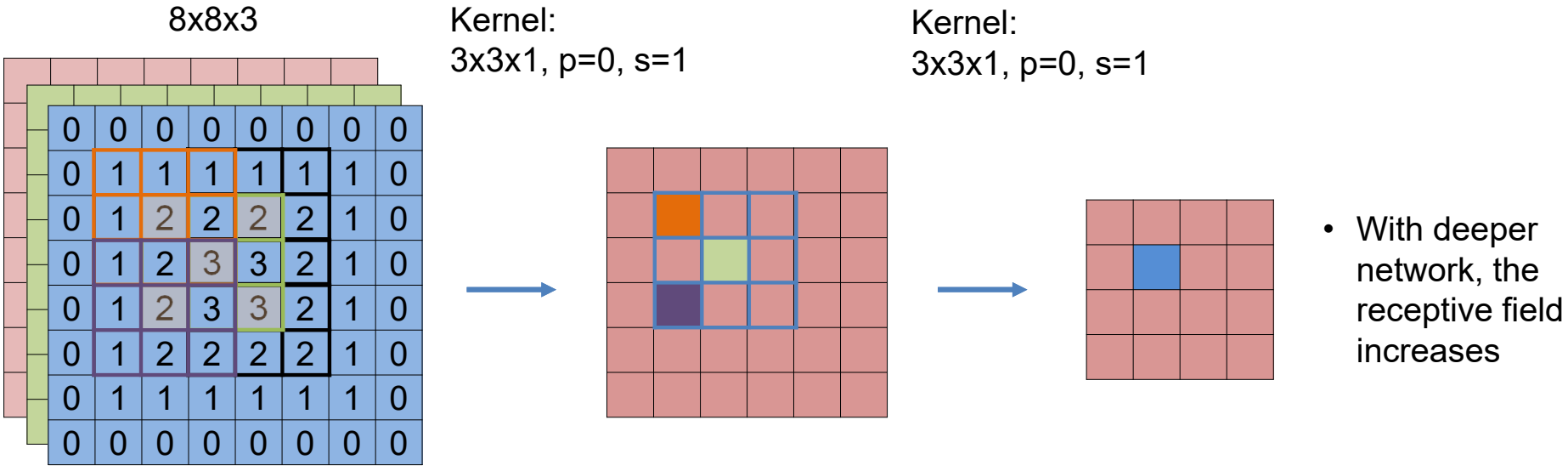
One FC layer claims 99.96%

Receptive field

- Every CONV filter looks at a small neighborhood

May need to look at more global structure in the image?

- Receptive field : the region in the image to impact a feature after some CONV layers



- With deeper network, the receptive field increases

1D and 3D convolution

- Computation is the same as the 2D CONV

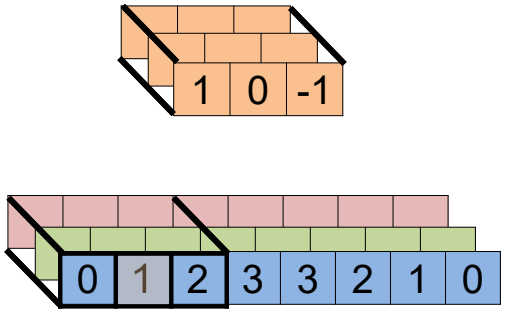
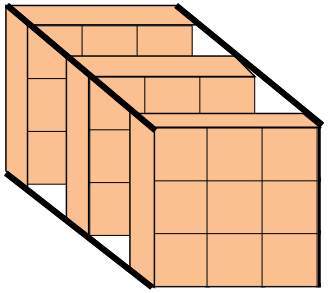


Image: $L \times C$

Kernel: $k \times C \times K$

Output : $L' \times K$

Padding along two ends



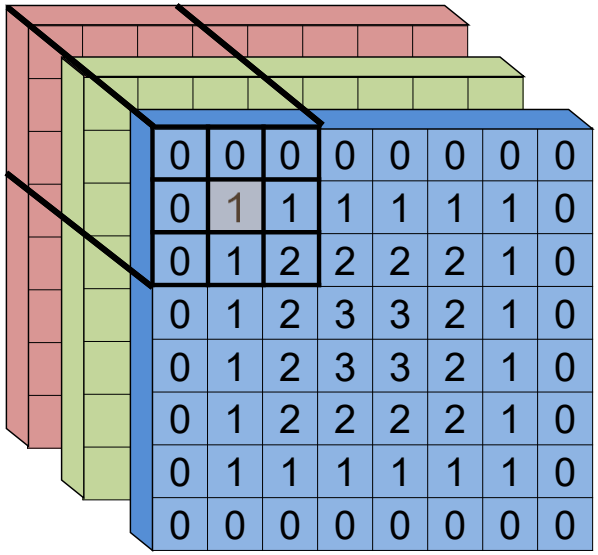


Image: $H \times W \times D \times C$

Kernel: $k_x \times k_y \times k_z \times C \times K$

Output : $H' \times W' \times D' \times K$

Padding along three dimensions

Dilated convolution

- Increase receptive field by “skipping pixels” in input images

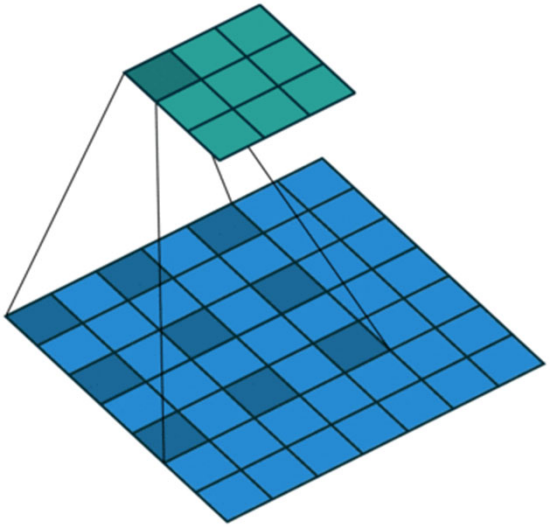
$k_x \times k_y$

1	0	-1
2	0	-2
1	0	-1

Dilate kernel with the dilation rate 1

1		0		-1
2		0		-2
1		0		-1

$[k_x + d(k_x - 1)] \times [k_y + d(k_y - 1)]$



0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	2	2	2	2	1	0
0	1	2	3	3	2	1	0
0	1	2	3	3	2	1	0
0	1	2	2	2	2	1	0
0	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0

$\boldsymbol{x}$, 9x1 → $w^T x$

[1 1 1 1 3 2 1 2 2]
×
[1 0 -1 2 0 -2 1 0 -1]^T
= -3

Then convolve with dilated kernel, but only count the $k_x \times k_y$ locations

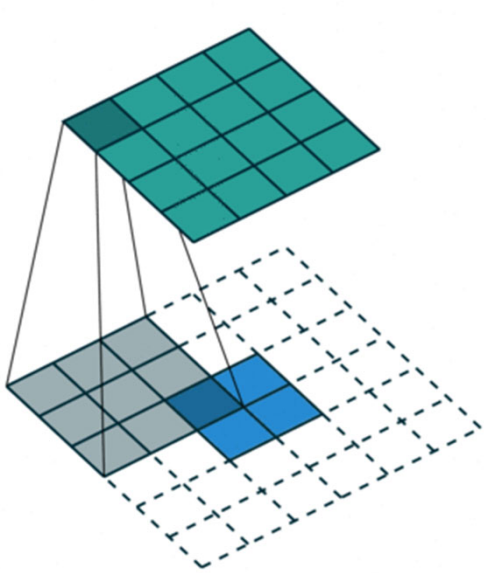
Input: 7x7, padding 0, stride 1
dilation rate 1, kernel 3x3

Output:3x3, $\left\lfloor \frac{7+2\times0-[3+1\times(3-1)]}{1} \right\rfloor + 1 = 3$

Animation is from https://github.com/vdumoulin/conv_arithmetic

Transpose convolution

- Increase image size after convolution – often used to upsample feature maps



Input: 2x2, padding 0, stride 1
Kernel: 3x3

Required Output:4x4

What is the padding size p?

$$\left\lfloor \frac{2 + 2 \times p - 3}{1} \right\rfloor + 1 = 4$$

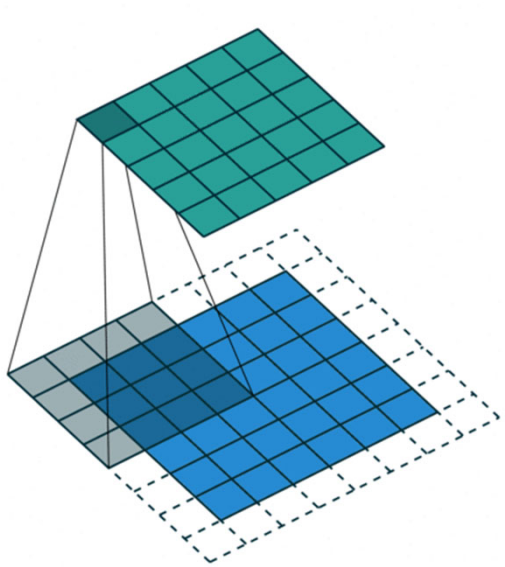
$$p = 2$$

- Determine padding size
- Sliding over the ***output***
- For every location in output, take the corresponding pixels in input maps and apply the kernel

Animation is from https://github.com/vdumoulin/conv_arithmetic

Transpose convolution

- Increase image size after convolution – often used to upsample feature maps



Input: 6x6, padding 0, stride 1
Kernel: 4x4

Required Output:5x5

What is the padding size p?

$$\left\lfloor \frac{6 + 2 \times p - 4}{1} \right\rfloor + 1 = 5$$

$$p = 1$$

- Determine padding size
- Sliding over the ***output***
- For every location in output, take the corresponding pixels in input maps and apply the kernel

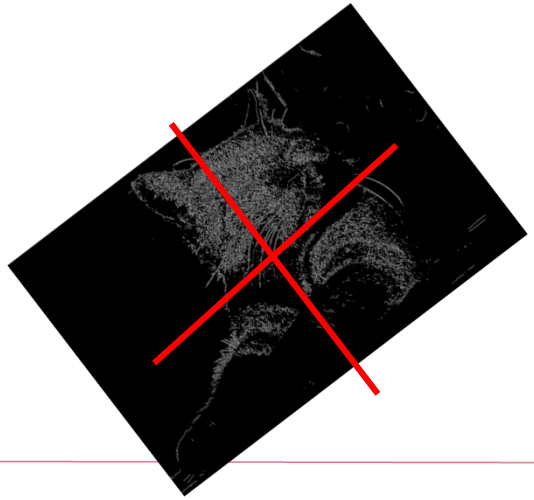
Animation is from https://github.com/vdumoulin/conv_arithmetic

Convolution is shift-invariant, not for rotation and scale

- CONV cannot deal with arbitrary spatial transformation

conv

conv



More general, the training and test sets can have different distribution

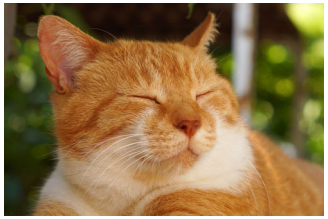
Training set, cats are roughly the same size

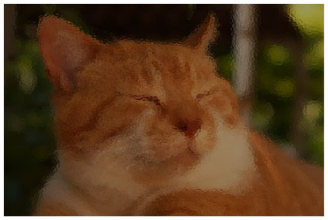
A test sample, small cat

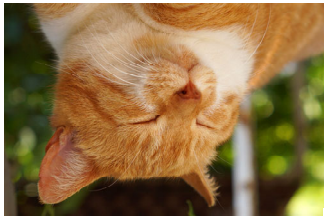
Data augmentation

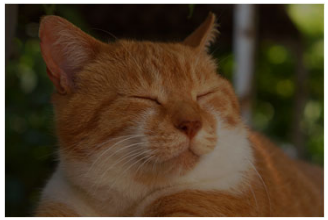
- Introduce variation in the training data, so model can learn invariance

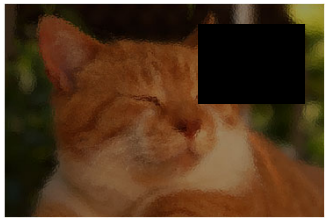
Randomly apply different transformations











And many more …

Check http://ai.stanford.edu/blog/data-augmentation/

Data augmentation

- **Do not** generate these random samples and save them
- Instead, implement them as a part of data loading

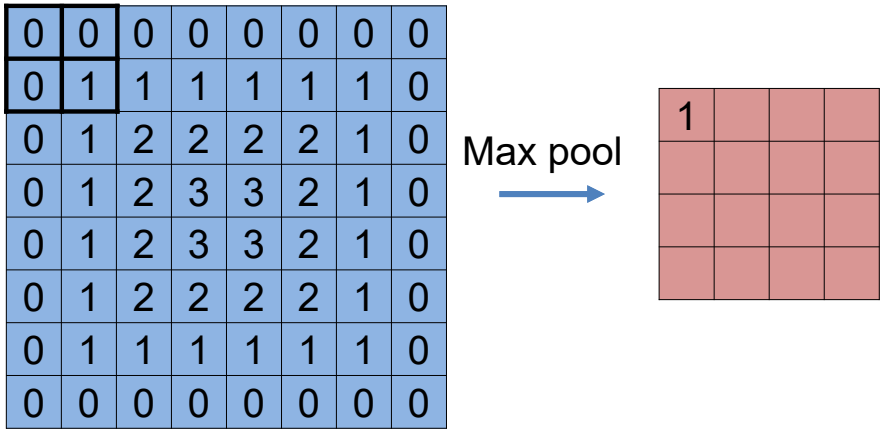
```
# set up data augmentation
transform_list = []
if(apply_random_permute_dim):
    transform_list.append(cine_seg_data_augmentation.RandomFlip1stDim(0.5))
    transform_list.append(cine_seg_data_augmentation.RandomFlip2ndDim(0.5))
    transform_list.append(cine_seg_data_augmentation.RandomPermute(0.5))

if (apply_random_rotate):
    transform_list.append(cine_seg_data_augmentation.RandomRotate(0.5))

if (apply_random_noise):
    transform_list.append(cine_seg_data_augmentation.RandomAddNoise(p=0.5, sigma=[0.1, 0.15, 0.2, 0.3]))

if (apply_random_blurring):
    transform_list.append(cine_seg_data_augmentation.RandomBlurring(p=0.5))

if (apply_random_scaling):
    transform_list.append(cine_seg_data_augmentation.RandomScale(p=0.5, scale_ratio=[0.75, 1.0, 1.25]))

if(len(transform_list)>0):
    dataset_support.transform = torchvision.transforms.Compose(transform_list)
else:
    dataset_support.transform = None
```


P: probability to apply the transformation

More examples can be found at https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py

Pooling : another way to reduce computation

- Often used to reduce the image size
- Parameter free



Input: 8x8
Pool size: 2x2

Take every 2x2 neighborhood, compute its max (max pool) or average (averaging pool)

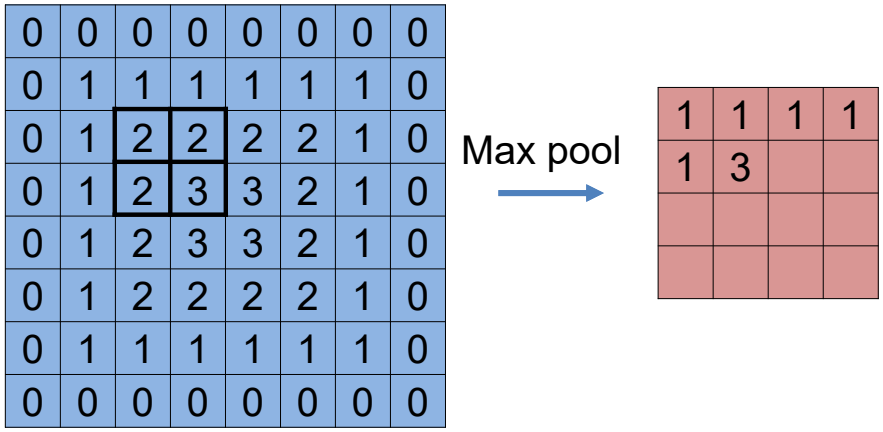
Typically, no padding, stride equals to the pool_size

Output size

$$\left\lfloor \frac{H - pool\_size}{pool\_size} \right\rfloor + 1$$

Pooling : another way to reduce computation

- Often used to reduce the image size
- Parameter free



Input: 8x8
Pool size: 2x2

Take every 2x2 neighborhood, compute its max (max pool) or average (averaging pool)

Typically, no padding, stride equals to the pool_size

Output size

$$\left\lfloor \frac{H - pool\_size}{pool\_size} \right\rfloor + 1$$

Pooling : another way to reduce computation

- Often used to reduce the image size
- Parameter free

0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	2	2	2	2	1	0
0	1	2	3	3	2	1	0
0	1	2	3	3	2	1	0
0	1	2	2	2	2	1	0
0	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0

Max pool →

1	1	1	1
1	3	3	1
1	3	3	1
1	1	1	1

Input: 8x8
Pool size: 2x2

Take every 2x2 neighborhood, compute its max (max pool) or average (averaging pool)

Typically, no padding, stride equals to the pool_size

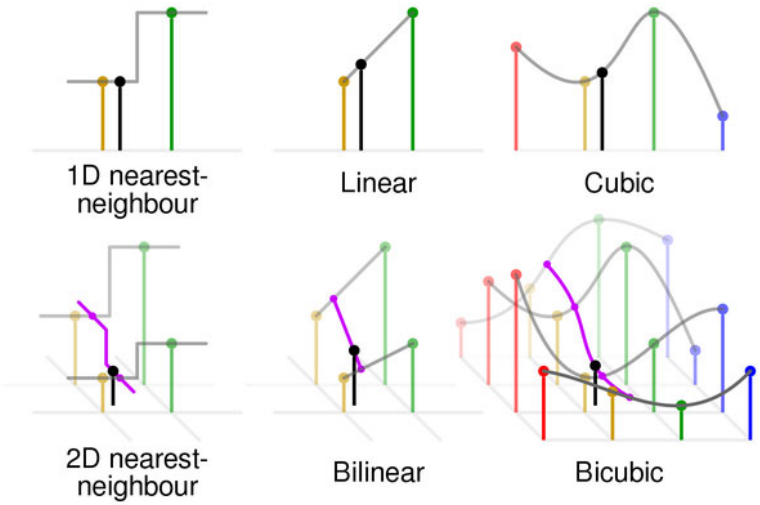
Output size

$$\left\lfloor \frac{H - pool\_size}{pool\_size} \right\rfloor + 1$$

2x2 is the most typical choice

Use interpolation to resize images

- Another “parameter free” way to resize images



- Interpolation: use neighboring pixels to determine value of unknown location (weighted sum)
- Weights depend on pixel location; no learnable parameters

TORCH.NN.FUNCTIONAL.INTERPOLATE

`torch.nn.functional.interpolate(`*`input`*`, size=None, scale_factor=None, mode='nearest',`
`align_corners=None, recompute_scale_factor=None``)` [SOURCE]

Down/up samples the input to either the given `size` or the given `scale_factor`

The algorithm used for interpolation is determined by `mode`.

Currently temporal, spatial and volumetric sampling are supported, i.e. expected inputs are 3-D, 4-D or 5-D in shape.

The input dimensions are interpreted in the form: *mini-batch x channels x* [*optional depth*] *x* [*optional height*] *x width*.

The modes available for resizing are: *nearest*, *linear* (3D-only), *bilinear*, *bicubic* (4D-only), *trilinear* (5D-only), *area*

- It is a layer in NN
- Differentiable by remembering the interpolation weights

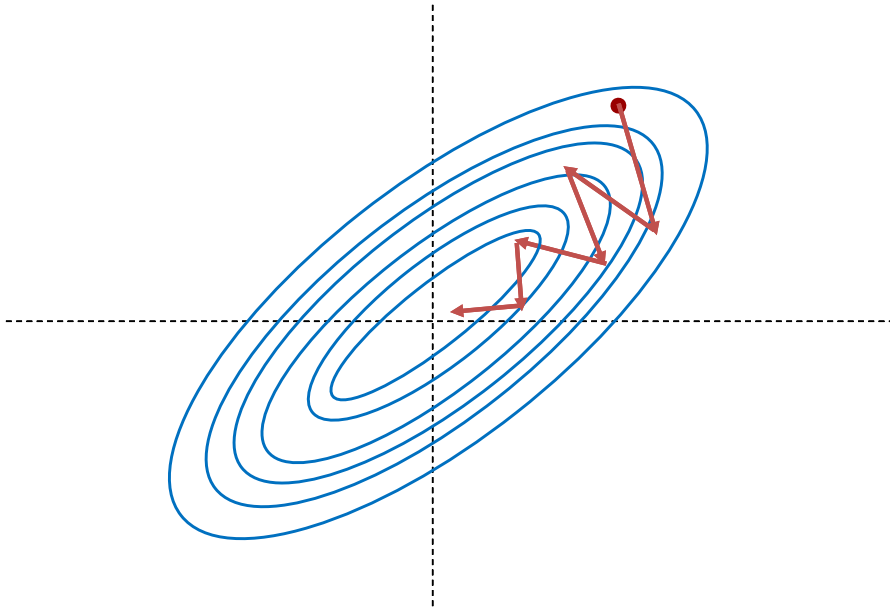
https://www.quora.com/What-is-bicubic-interpolation-in-image-processing

Layers to adjust image size

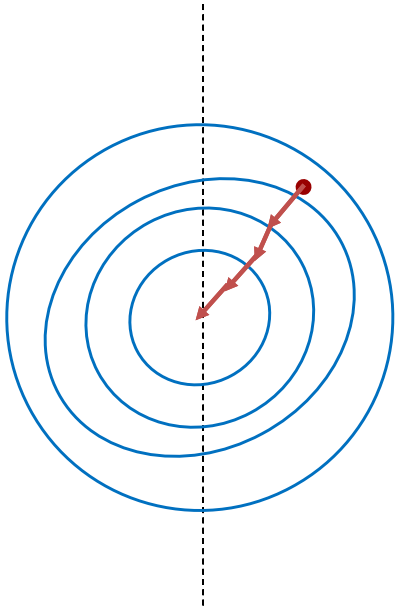
Layers	Downsample	Upsample	Pro/Cons
Stride CONV	Yes	No	Same as normal conv, fast to compute
Pooling	Yes	No	Fast to compute, reduce the model power, less popular nowadays
Interpolation	Yes	Yes	Parameter free, fast to compute
Transpose CONV	Yes, but often not used for this purpose	Yes	Has learnable parameters for upsampling

- Pooling still is used in classification tasks
- Full CONV architecture are more popular for other tasks, e.g. segmentation, super-resolution, denoising …

Batch Normalization



Features with varying magnitude



Features with normalized magnitude

$$y = w_1x_1 + w_2x_2 + w_3x_3$$

$$\frac{d\ell}{dw_1} = \frac{d\ell}{dy}x_1 \qquad \frac{d\ell}{dw_2} = \frac{d\ell}{dy}x_2 \qquad \frac{d\ell}{dw_3} = \frac{d\ell}{dy}x_3$$

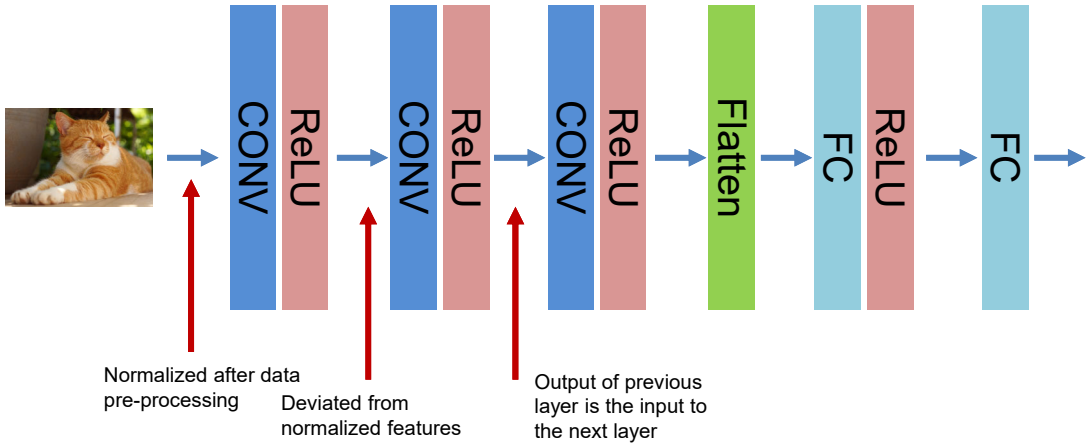
- Features with unnormalized magnitude leads to harder optimization

Batch Normalization

Techniques to battle unnormalized features
- Data pre-processing, whitening …
- Model parameter initialization (e.g. He and Xavier)

Model will be well-behaved at the beginning of training

- No guarantee for later epochs
- No guarantee for deeper layers
- Almost guaranteed to deviate



Batch Normalization: if we need normalized features, add a layer to enforce it

Add a specific layer to normalize the output of previous layer

Give a mini-batch, $\boldsymbol{X} = \{\boldsymbol{x}^{\{1\}} \quad \boldsymbol{x}^{\{2\}} \quad \boldsymbol{x}^{\{3\}} \quad \dots \quad \dots \quad \boldsymbol{x}^{\{B\}}\}$

Every sample $\boldsymbol{x}^{\{k\}}$ is a Nx1 vector or CxHxW image

- $\widehat{\boldsymbol{x}}^{\{k\}}$ is the Nx1 vector, same as $\boldsymbol{x}^{\{k\}}$
- μ and δ are per-dimension mean and variance

- Compute mean of this batch

$$\mu = \frac{1}{B}\sum_{k=1}^{B} \boldsymbol{x}^{\{k\}}$$

- Compute variance of this batch

$$\delta^2 = \frac{1}{B}\sum_{k=1}^{B} \left[\boldsymbol{x}^{\{k\}} - \mu\right]^2$$

- Normalize the batch

$$\widehat{\boldsymbol{x}}^{\{k\}} = \frac{\boldsymbol{x}^{\{k\}} - \mu}{\sqrt[2]{\delta^2 + \epsilon}}$$

← Element-wise operation

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167

Batch Normalization: if we need normalized features, add a layer to enforce it

Give a mini-batch, $\boldsymbol{X} = \{\boldsymbol{x}^{\{1\}} \quad \boldsymbol{x}^{\{2\}} \quad \boldsymbol{x}^{\{3\}} \quad \dots \quad \dots \quad \boldsymbol{x}^{\{B\}}\}$

$\boldsymbol{x}^{\{k\}}$ is a Nx1 vector or CxHxW image

$$\mu = \frac{1}{B}\sum_{k=1}^{B} \boldsymbol{x}^{\{k\}} \qquad \delta^2 = \frac{1}{B}\sum_{k=1}^{B} \left[\boldsymbol{x}^{\{k\}} - \mu\right]^2$$

$$\widehat{\boldsymbol{x}}^{\{k\}} = \frac{\boldsymbol{x}^{\{k\}} - \mu}{\sqrt[2]{\delta^2 + \epsilon}}$$

γ and β are the same dimension as $\boldsymbol{x}^{\{k\}}$

$$\boldsymbol{y}^{\{k\}} = \gamma \circ \widehat{\boldsymbol{x}}^{\{k\}} + \beta$$

$\widehat{\boldsymbol{x}}^{\{k\}} = \boldsymbol{x}^{\{k\}}$ if $\gamma = \delta$ and $\beta = \mu$

- Normalization reduces the representation power of model
- To restore the representation power, introduce two learnable parameters

At training time,

Given a mini-batch, compute mean and variance

Compute $\widehat{\boldsymbol{x}}$ and $\boldsymbol{y}$, using γ and β and μ and δ

Therefore, if batch size is too small (e.g. <16), BN may not work well

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167

Batch Normalization: if we need normalized features, add a layer to enforce it

At training time,

Given a mini-batch, compute μ and δ from this batch

Compute $\hat{\boldsymbol{x}}$ and $\boldsymbol{y}$, using γ and β and μ and δ

$$\mu = \frac{1}{B}\sum_{k=1}^{B} \boldsymbol{x}^{\{k\}} \qquad \delta^2 = \frac{1}{B}\sum_{k=1}^{B} \left[\boldsymbol{x}^{\{k\}} - \mu\right]^2$$

$$\hat{\boldsymbol{x}}^{\{k\}} = \frac{\boldsymbol{x}^{\{k\}} - \mu}{\sqrt[2]{\delta^2 + \epsilon}}$$

$$\boldsymbol{y}^{\{k\}} = \gamma \circ \hat{\boldsymbol{x}}^{\{k\}} + \beta$$

At test time,

We don't have a mini-batch, often B=1

- The assumption behind BN is the statistics from a batch can approximate the entire dataset
- Test and training datasets have the same distribution

$$\overline{\mu} = (1-\alpha)\overline{\mu} + \alpha\mu$$

$$\overline{\delta} = (1-\alpha)\overline{\delta} + \alpha\delta$$

$$\alpha = 0.1$$

or, $\overline{\mu}$ and $\overline{\delta}$ can be computed as simple average for all trained batches

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167

Spatial Batch Normalization for CONV

Batch Normalization

BxN

Normalize along B

BxN

γ, β: [Nx1] learnable parameters

Spatial Batch Normalization

BxCxHxW

Normalize along BxHxW

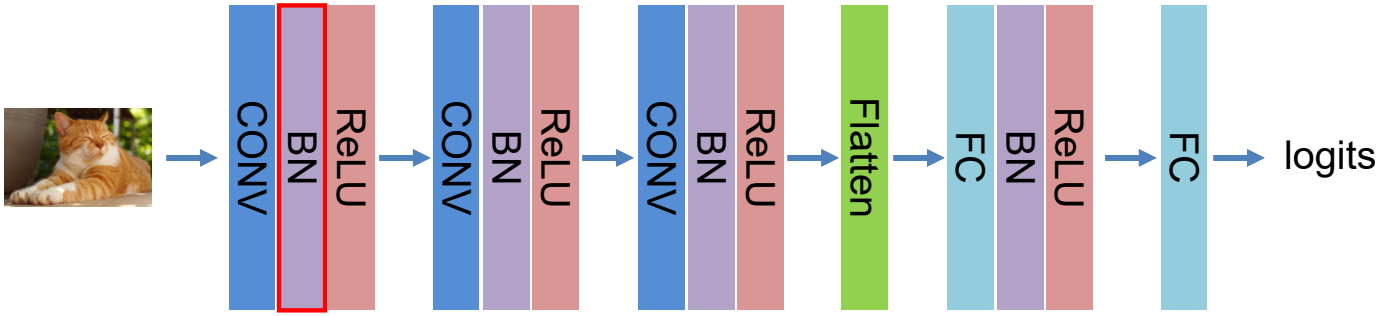
BxCxHxW

γ, β: [Cx1] learnable parameters

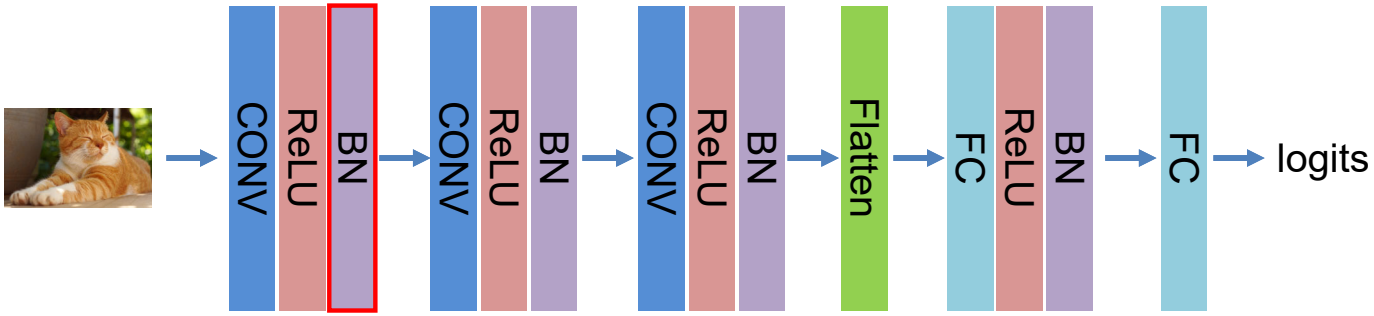
e.g. torch.nn.BatchNorm2d

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167

Model with BN layer



Make more sense since ReLU causes unbalanced distribution



Make more sense since BN normalizes both CONV and ReLU

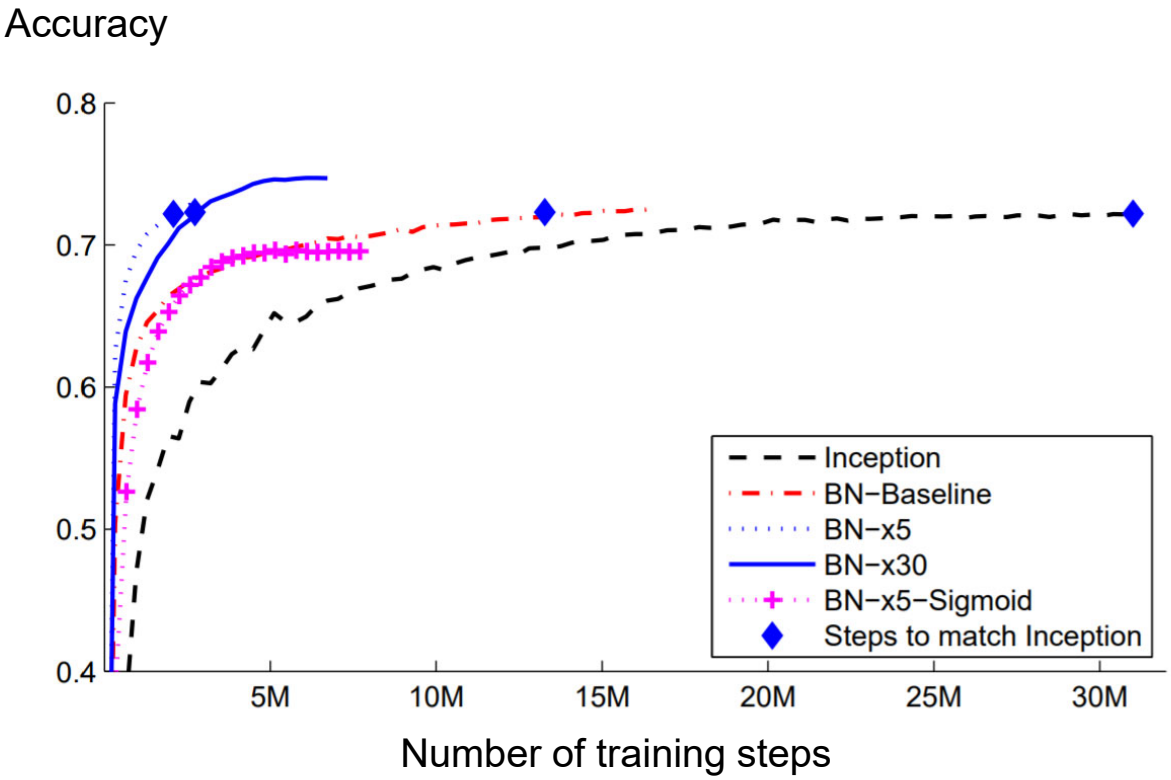
- Often lead to faster and more stable convergence
- Less sensitive to parameter initialization
- Less sensitive to hyper-parameter setting
- Allow flexible selection of nonlinearity functions

- Has regularization effect → introduce randomness in training

- Do not use with drop-out

- Can hide bugs in the code – debug by turning BN on/off

From BN paper



- Test on ImageNet 2012 dataset with the inception network
- BN-x5, x30 is to increase learning rate by 5 times or 30 times → BN allows to use larger learning rate → faster convergence
- BN-x5-Sigmoid shows BN allows to use sigmoid activation function. Without BN, sigmoid activation does not lead to convergence
- BN-x30 got higher accuracy than BN-x5 → exploration

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. https://arxiv.org/abs/1502.03167

Model evaluation and training

```
# evaluate model:
model.eval()

with torch.no_grad():
    ...
    out_data = model(data)
    ...
```

```
# training step
...
model.train()
...
```

- Do not forget to set model be in evaluation mode, so the moving averaged mean and standard deviation will not be updated
- When doing training, set model to be in training mode, so the moving averaged mean and standard deviation will be updated
- A common source of error

More feature normalization

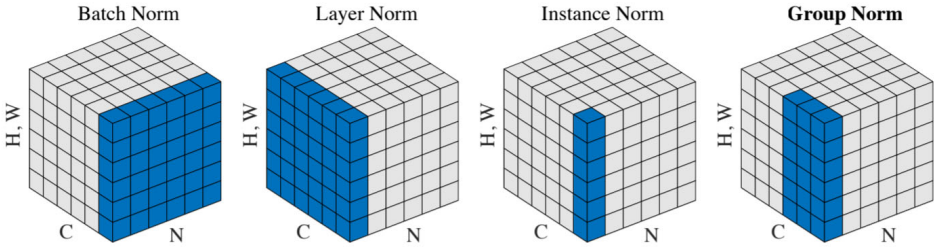


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

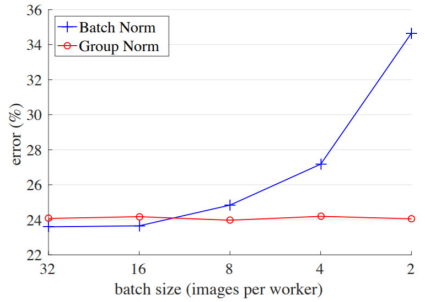
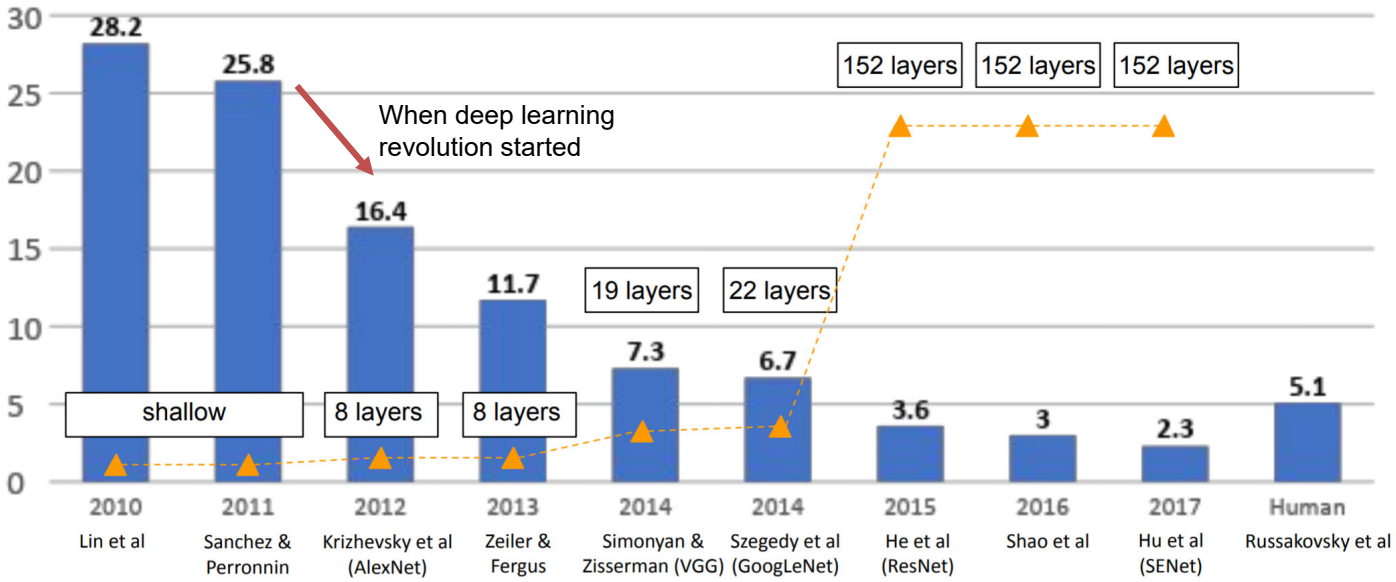


Figure 1. **ImageNet classification error *vs*. batch sizes**. This is a ResNet-50 model trained in the ImageNet training set using 8 workers (GPUs), evaluated in the validation set.

Group Normalization. https://arxiv.org/pdf/1803.08494.pdf https://medium.com/techspace-usict/normalization-techniques-in-deep-neural-networks-9121bf100d8

CNN leads to the revolution of deep learning



- Deep learning revolution started in 2012 with AlexNet
- Its origin were long back in time
- LeNet 1989 or earlier
- CNN architecture development was well tracked by ImageNet winners
- What was learned was adapted to other applications

Figure is from http://cs231n.stanford.edu/slides/2021/lecture_9.pdf

LeNet and MNIST datasets

Modified National Institute of Standards and Technology database

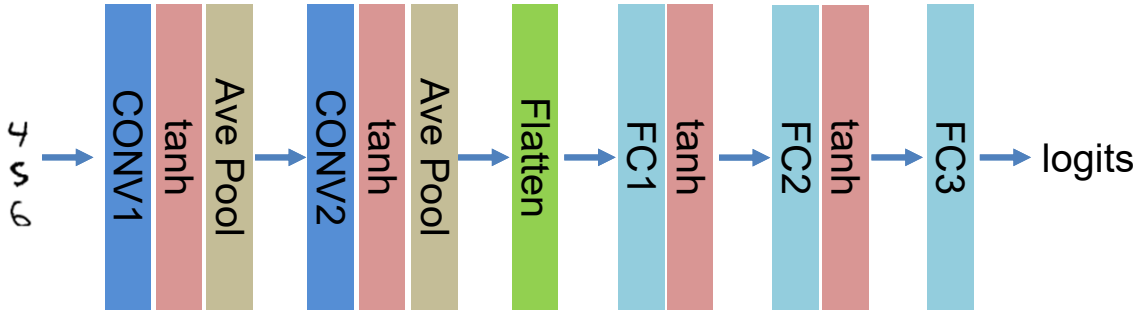
The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized (28x28) and centered in a fixed-size image.

- http://yann.lecun.com/exdb/mnist/

EMNIST has been published in 2017, which contains 240,000 training images, and 40,000 testing images of handwritten digits and characters.

Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, *Proceedings of the IEEE, 86(11):2278-2324, November 1998*

LeNet-5



Input : Bx1x28x28	Output
CONV1: 5x5, padding 2, stride 1, K = 6	Bx6x28x28
Ave pool: 2x2	Bx6x14x14
CONV2: 5x5, padding 0, stride 1, K = 16	Bx16x10x10
Ave pool: 2x2	Bx16x5x5
FC1: D=120	Bx120
FC2: D=84	Bx84
FC3: D=10	Bx10

~60K parameters

- Backprop to train CONV
- Use ave pooling
- Use tanh
- Use flatten and FC
- Use softmax and CE loss
- Increase number of filters with reduced spatial resolution

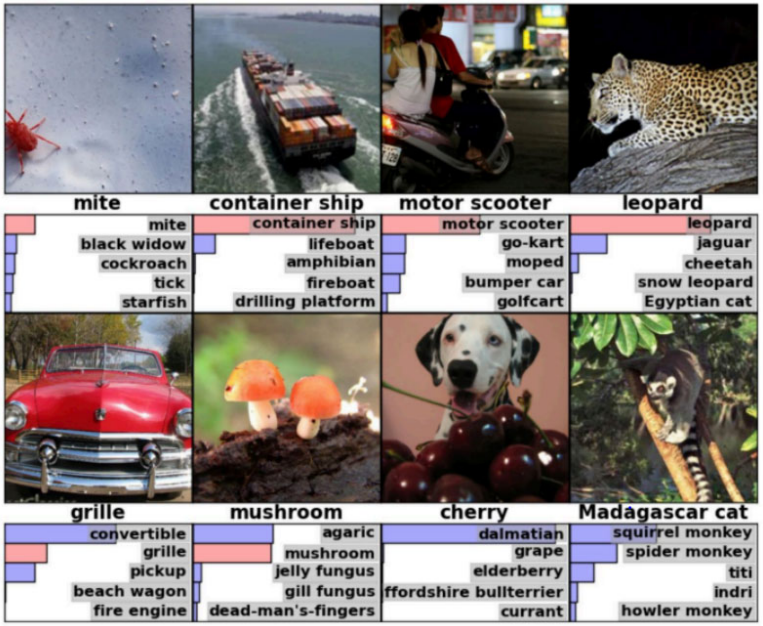
While the architecture of the best performing neural networks today are not the same as that of LeNet, **the network was the starting point for a large number of neural network architectures, and also brought inspiration to the field.**

- https://en.wikipedia.org/wiki/LeNet

Used by the USPS, maybe the first deployed cnn?

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W. & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4):541-551.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC)



I.Object localization for 1000 categories.
II.Object detection for 200 fully labeled categories.
III.Object detection from video for 30 fully labeled categories.

- Running since 2010, currently still run in Kaggle
https://image-net.org/challenges/LSVRC/index.php

https://www.kaggle.com/c/imagenet-object-localization-challenge/overview/description

https://www.paperswithcode.com/sota/image-classification-on-imagenet?metric=Top%205%20Accuracy

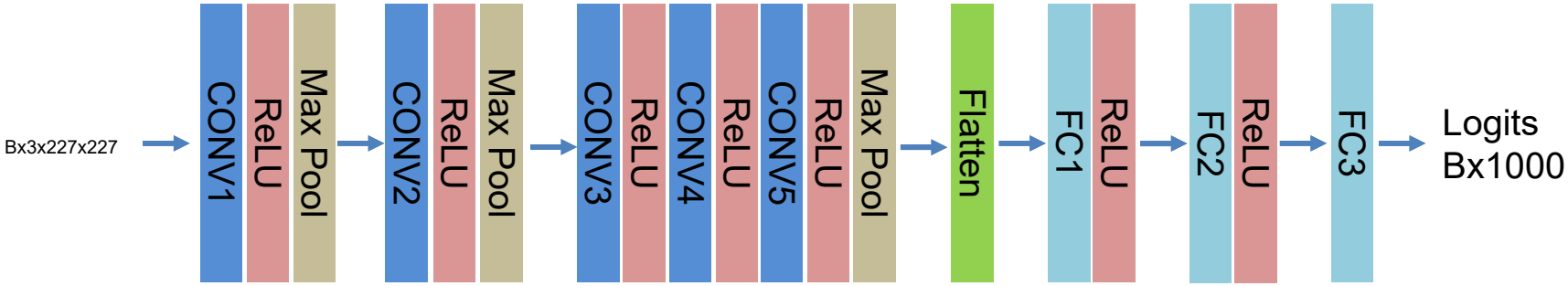
- ILSVRC used 1.2 million training images for 1000 classes, 50,000 validation images, and 150,000 testing images

- Include object classification, detection and video detection tasks

- ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories

LeCun, Y.; Boser, B.; Denker, J. S.; Henderson, D.; Howard, R. E.; Hubbard, W. & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4):541-551.

AlexNet



Input : Bx3x227x227	Output
CONV1: 11x11, padding 0, stride 4, K = 96	Bx96x55x55
Max pool: 3x3, stride 2	Bx96x27x27
CONV2: 5x5, padding 2, stride 1, K = 256	Bx256x27x27
Max pool: 3x3, stride 2	Bx256x13x13
CONV3: 3x3, padding 1, stride 1, K = 384	Bx384x13x13
CONV3: 3x3, padding 1, stride 1, K = 384	Bx384x13x13
CONV3: 3x3, padding 1, stride 1, K = 256	Bx256x13x13
Max pool: 3x3, stride 2	Bx256x6x6
FC2: D=4096	Bx4096 (~62% of total #paras)
FC1: D=4096	Bx4096
FC3: D=1000	Bx1000

~~60K~~ 61M parameters

- Backprop to train CONV
- Use ~~ave~~ max pooling
- Use ~~tanh~~ ReLU
- Use flatten and FC
- Use softmax and CE loss
- Increase number of filters with reduced spatial resolution
- Much bigger
- Data augmentation
- Use GPU for training
- Much larger dataset

Training and network architecture

Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NeuIPS 2012.

To be continued …