SK hynix i-TAP
반도체 Data Scientist를 위한
ML/DL 심화 커리큘럼

2강 Convolutional Neural Networks (CNN)

Ernest K. Ryu (류경석)

2020.11.13

Example: Gradient Computation with AutoDiff

2강. Convolutional Neural Networks (CNN)

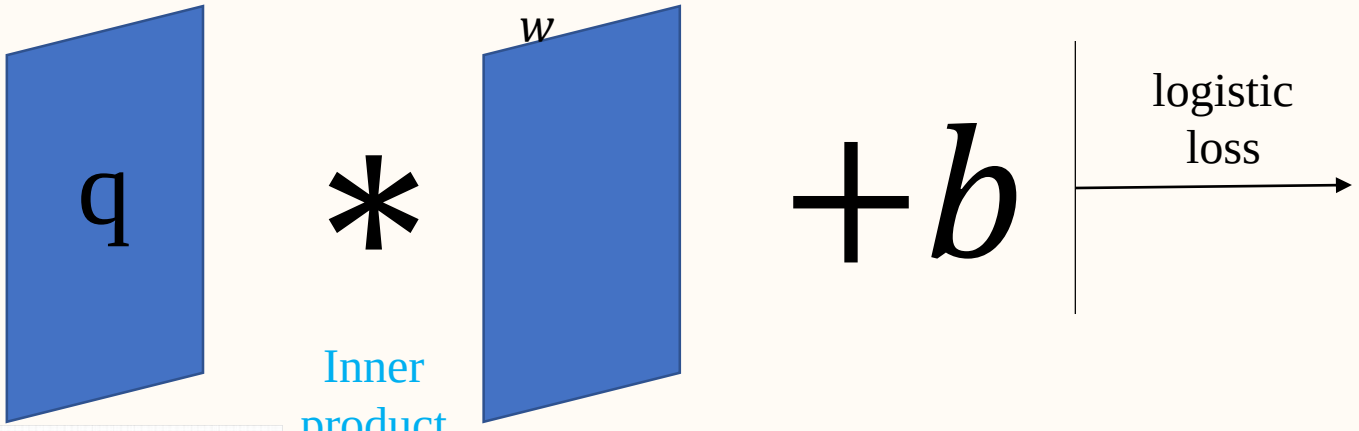
- Convolution operation
- Pooling
- (Basic) data augmentation
- Regularization: weight decay, dropout, early stopping
- Weight initialization
- CNN architectures: LeNet, AlexNet, VGGNet, NiN, GoogLeNet

Shift Invariance in Vision to Convolution

Cat

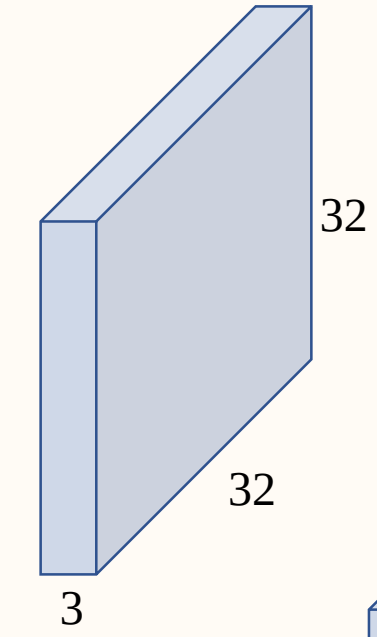
Still a Cat

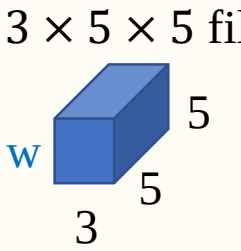
Logistic regression (with a single fully connected layer) does not encode shift invariance



Translating digit changes output significantly

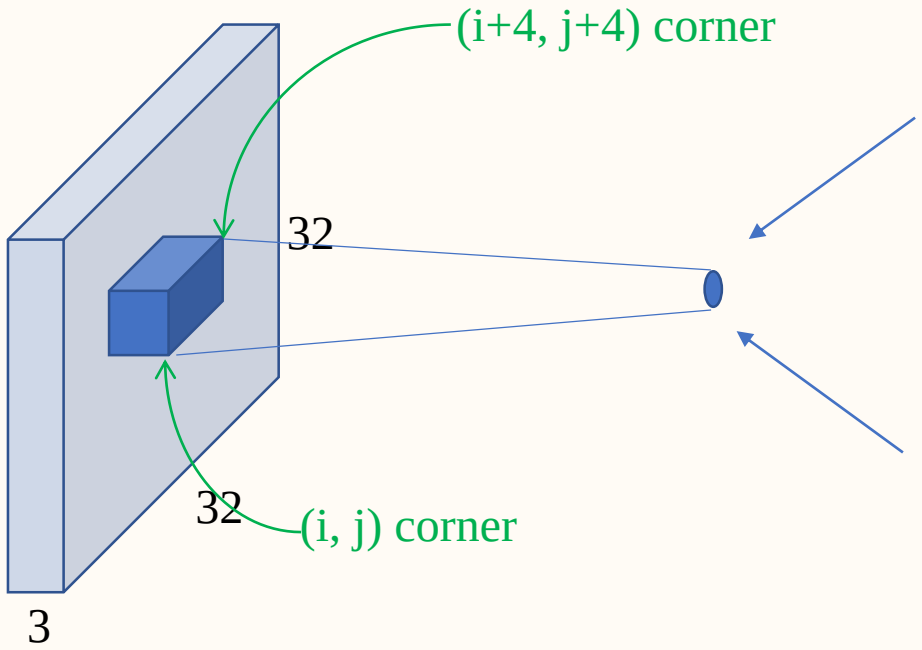
Convolutional Layer





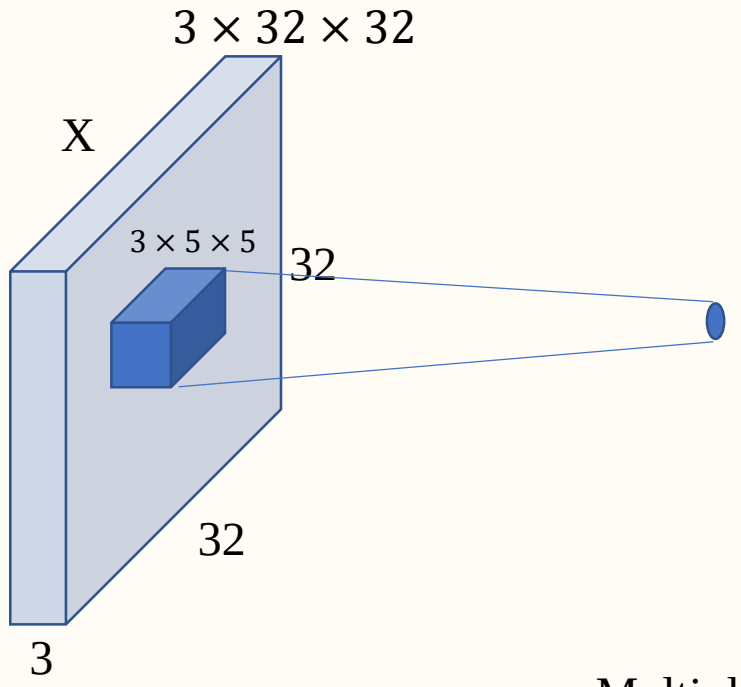
Convolve the filter with the image

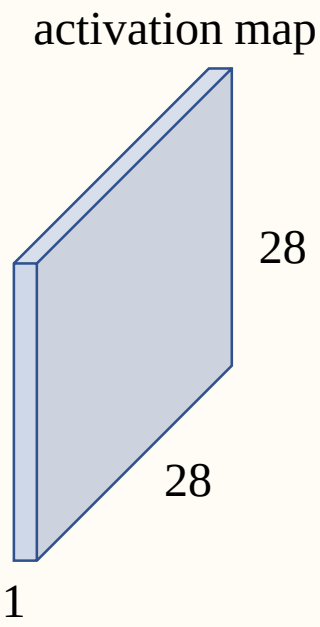
(=Slide the filter spatially over the image and compute dot products)



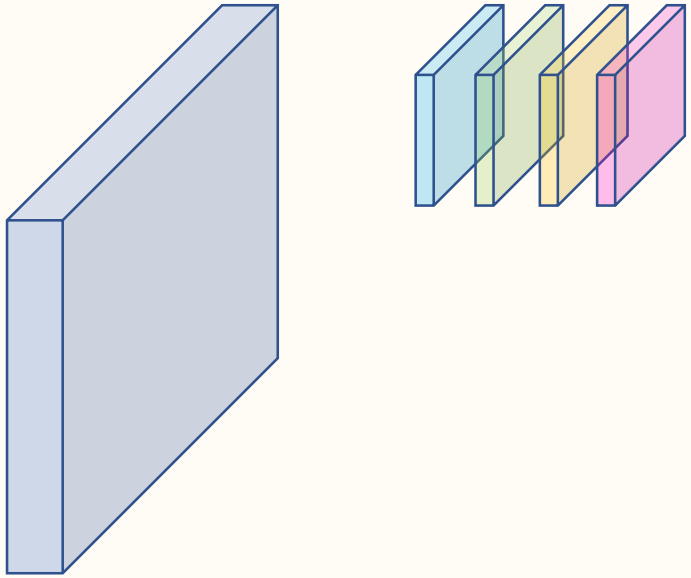
1 number: take a 3 × 5 × 5 chunk of the image and take the inner product with w and add bias b

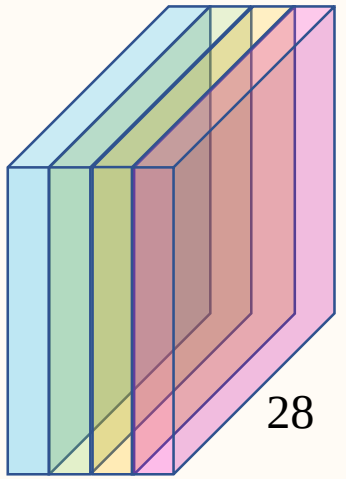
= w.reshape(-1).T@ X [:,i:i+5, j:j+5].reshape(-1)+b





Multiple filters



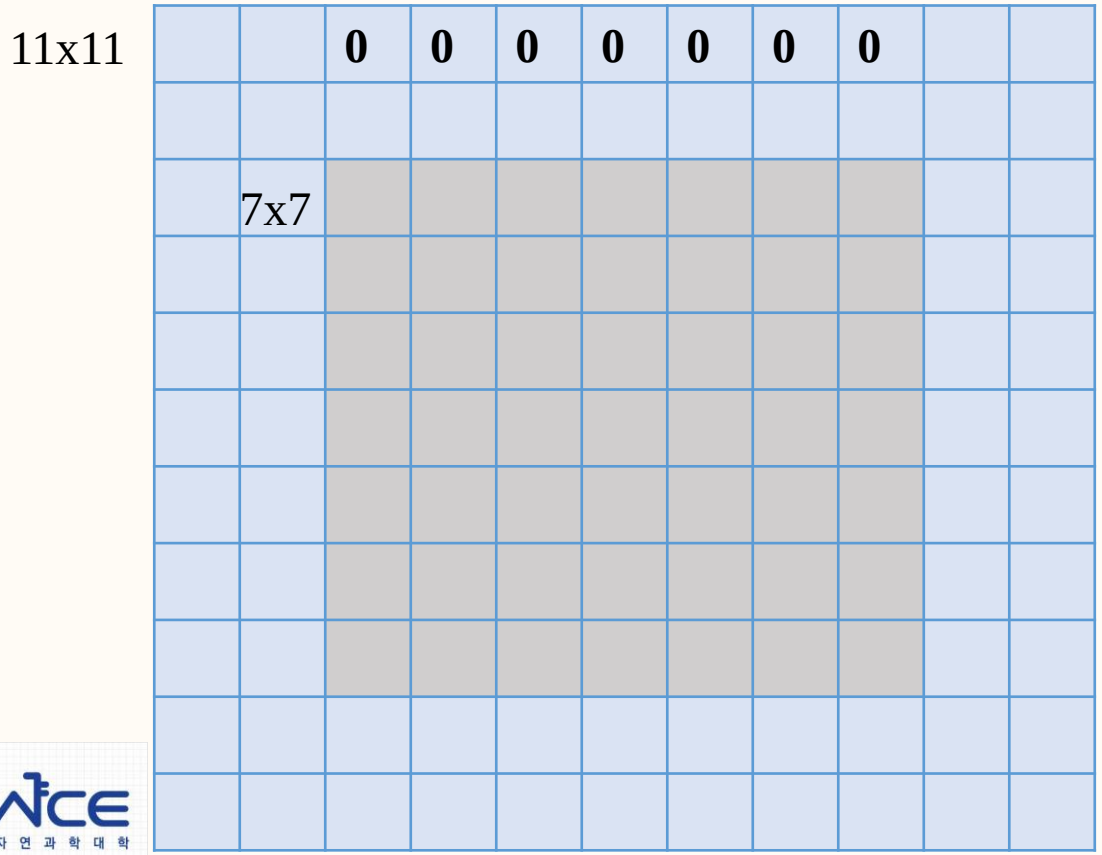


4 = depth of activation map

Convolution options : Zero Padding

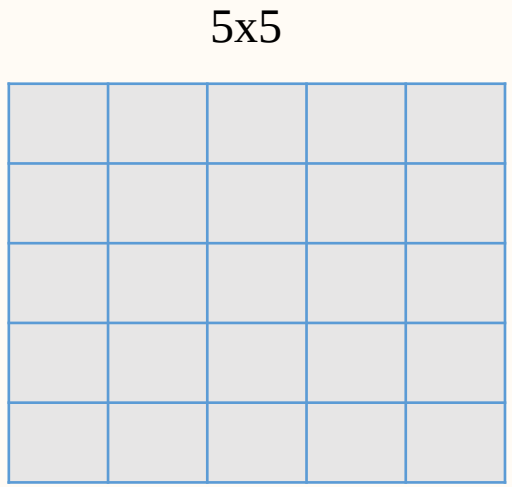
$3 \times 32 \times 32$ convolved with $3 \times 5 \times 5$ filter $\Rightarrow 1 \times 28 \times 28$ activation map

Spatial dimension 32 reduced to 28



Padding=2

convolve

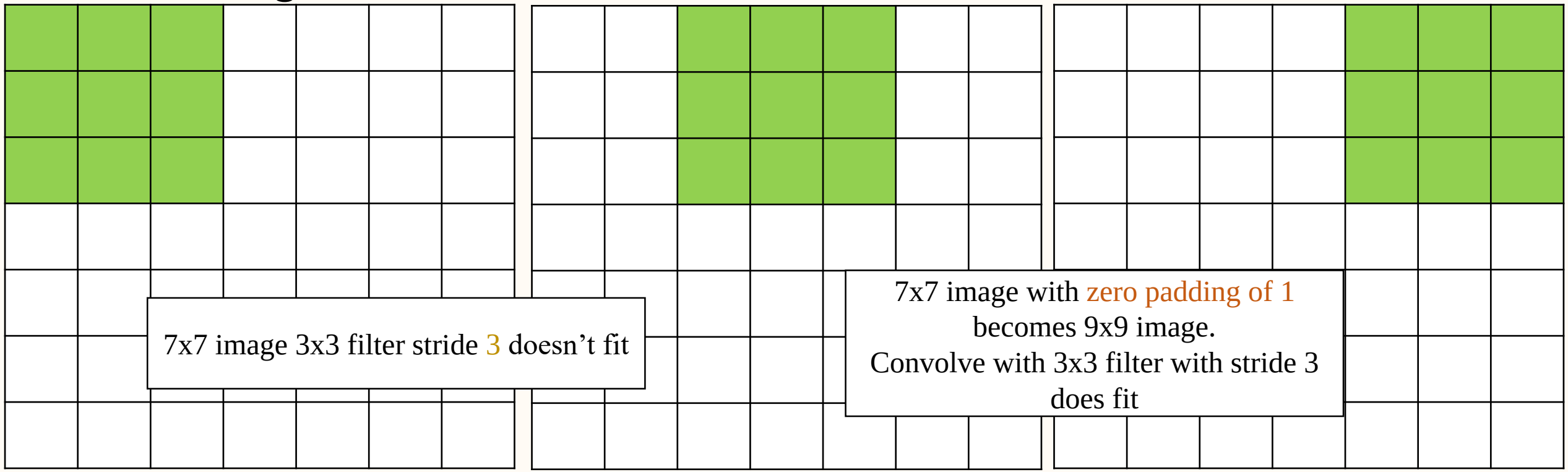


7x7 with Zero padding
(spatial dimension preserved)

3x3 without Zero padding

Convolution options : Stride

7x7 image convolved with 3x3 filter with stride 2



7x7 image 3x3 filter stride 3 doesn't fit

7x7 image with zero padding of 1
becomes 9x9 image.
Convolve with 3x3 filter with stride 3
does fit

Output 3x3
(with stride 1, output is 5x5)

Summary

Input $D_1 \times W_1 \times H_1$

Conv layer parameters

- K filters
- F spatial extent ($D_1 \times F \times F$ filters)
- S stride
- P padding

Output $D_1 \times W_2 \times H_2$

The number of parameters

$$F^2 D_1 K + K$$

filters biases

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

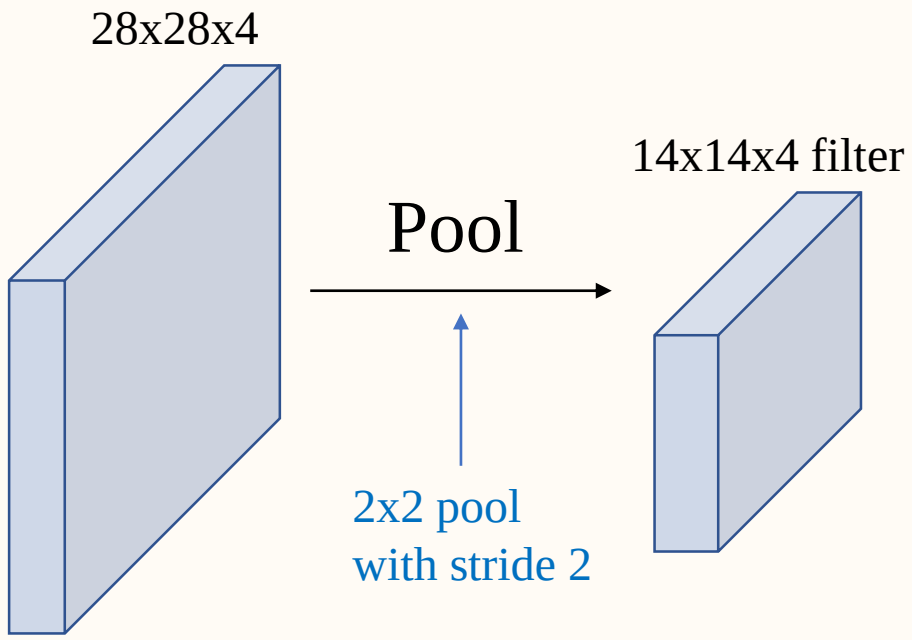
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Should be integers

$$D_2 = K$$

Pooling

- Similar to convolutions
- Used to reduce the size of the output
- Operates over each activation map independently



Single depth size

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Max Pool

2x2 filters and stride 2

6	8
3	4

Not an instance of convolution

Precise definitions in
torch.nn.MaxPool2D
torch.nn.AvgPool2D

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

Average Pool

2x2 filters and stride 2

Effect is subsampling
(lowering image resolution)

3.25	5.25
2	2

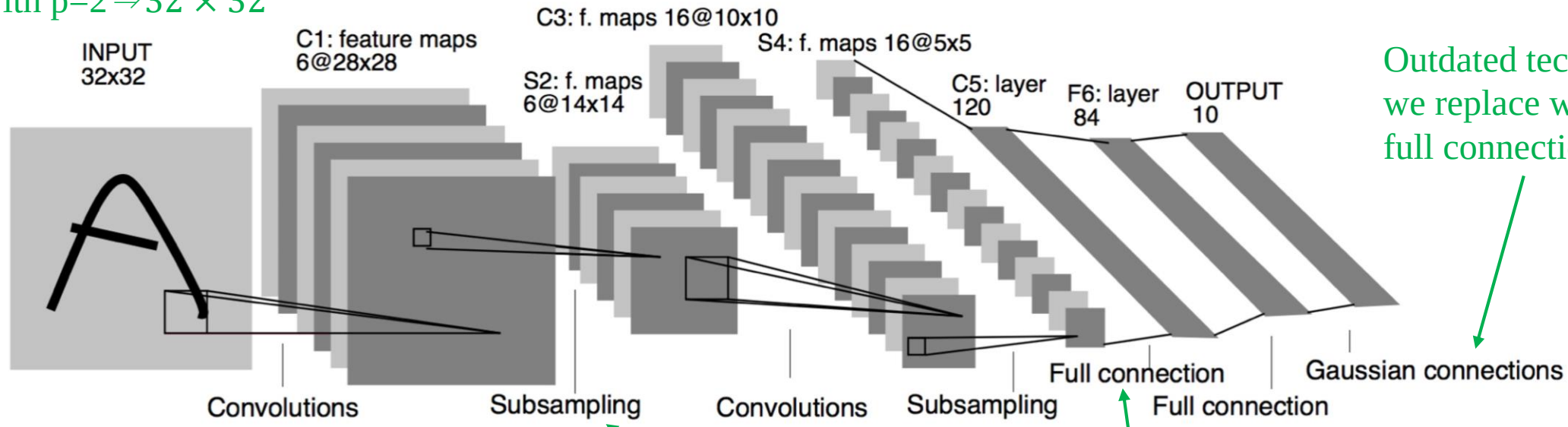
Instance of convolution with fixed (untrainable) weights, including the independent operation over each activation map.
(Why?)

LeNet5

(LeCun, Bottou, Bengio, Haffner 1998)

Modern instances of LeNet5 use
- σ =ReLu
- MaxPool instead of avg. pool
- No σ after S2, S4 (Why?)
- No Gaussian connections
- Complete C4 connections

28 × 28 MNIST image
with p=2 ⇒ 32 × 32



Outdated technique we replace with full connection

Convolutions
f=5, k=6

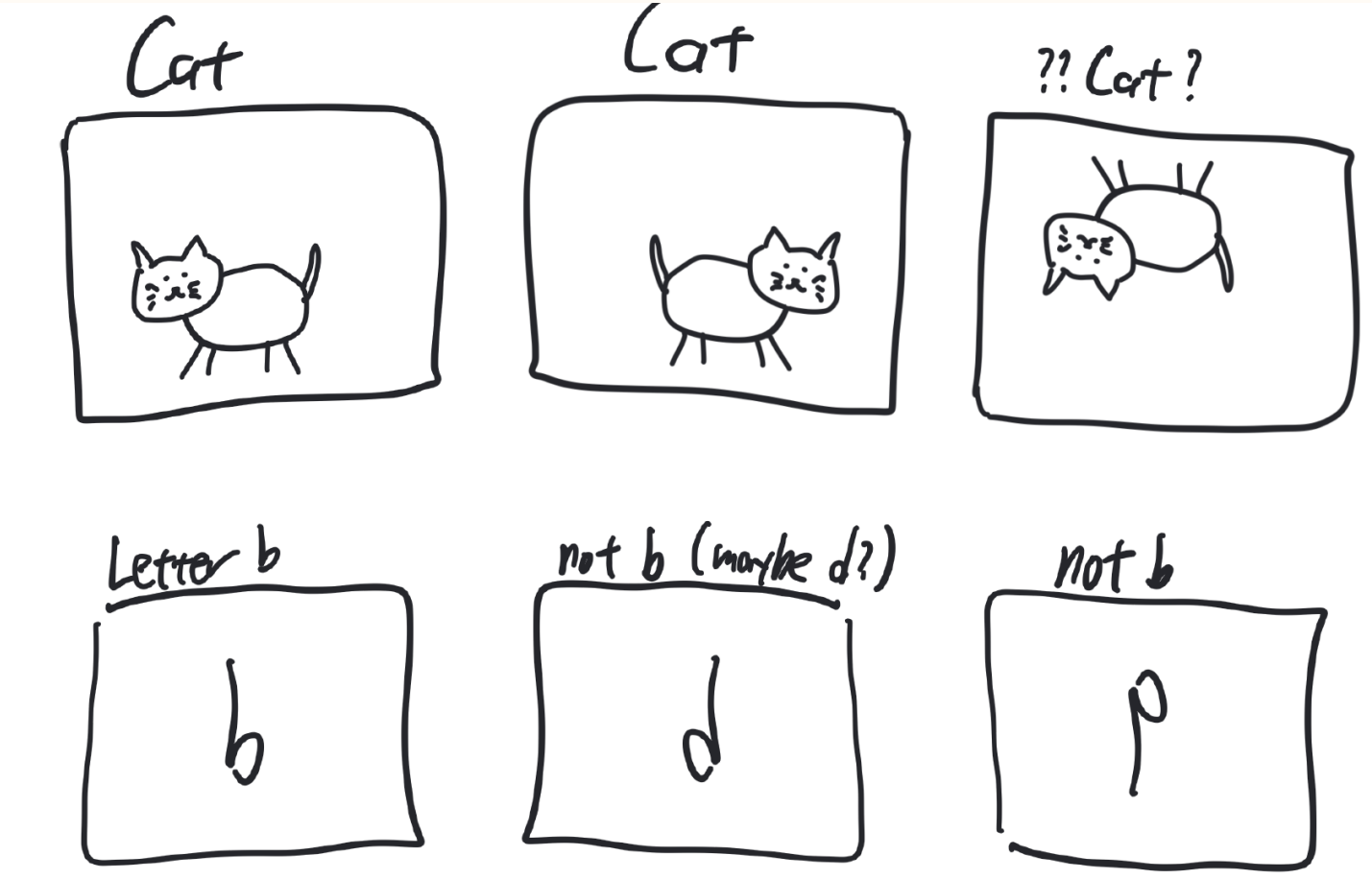
σ = modification of tanh
Applied after
C1, S2, C3, S4, C5, F6

Something like average pool
f=2, S=2

Convolutions
f=5, k=16

full connection from 16 × 5 × 5 to 120
⇔conv. with f=5, k=120

Data augmentation



Invariances
- Translation
- Horizontal flip
- ~~Vertical flip~~
- Color change (?)

Invariances
- Translation
- ~~Horizontal flip~~
- ~~Vertical flip~~
- Color change

Translation invariance encoded in CNN via convolution but other invariances are harder to encode ⇒ Encode in data so NN can learn.

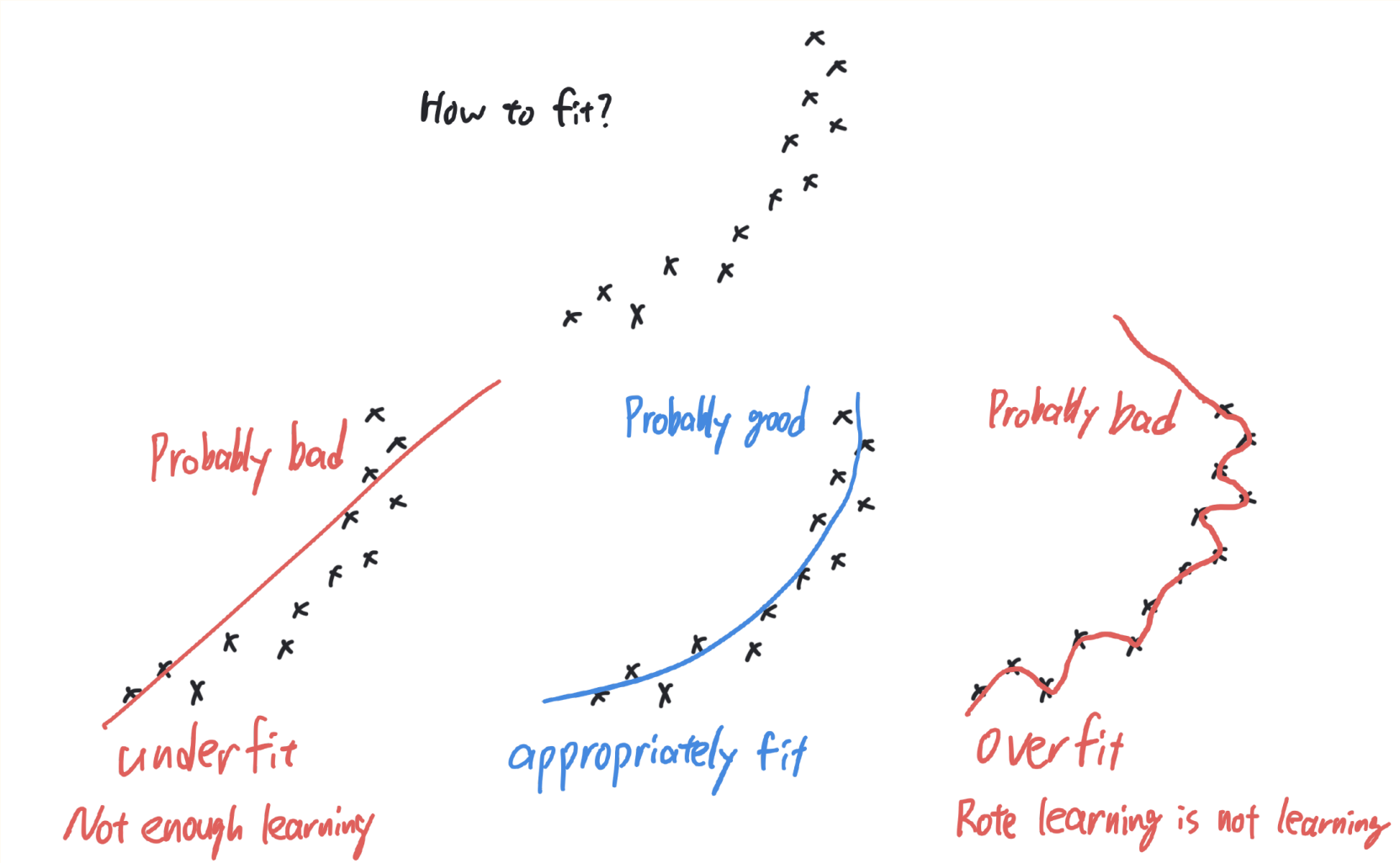
For data augmentation: aply transforms that preserve meaning and labe.

- Enlarge dataset or
- Use randomly transformed data in training loop (but not in testing to ensure no spurious symmetries are added)
- Torchvision.transforms in PyTorch

Assuming transforms correctly represent symetries, data augmentation

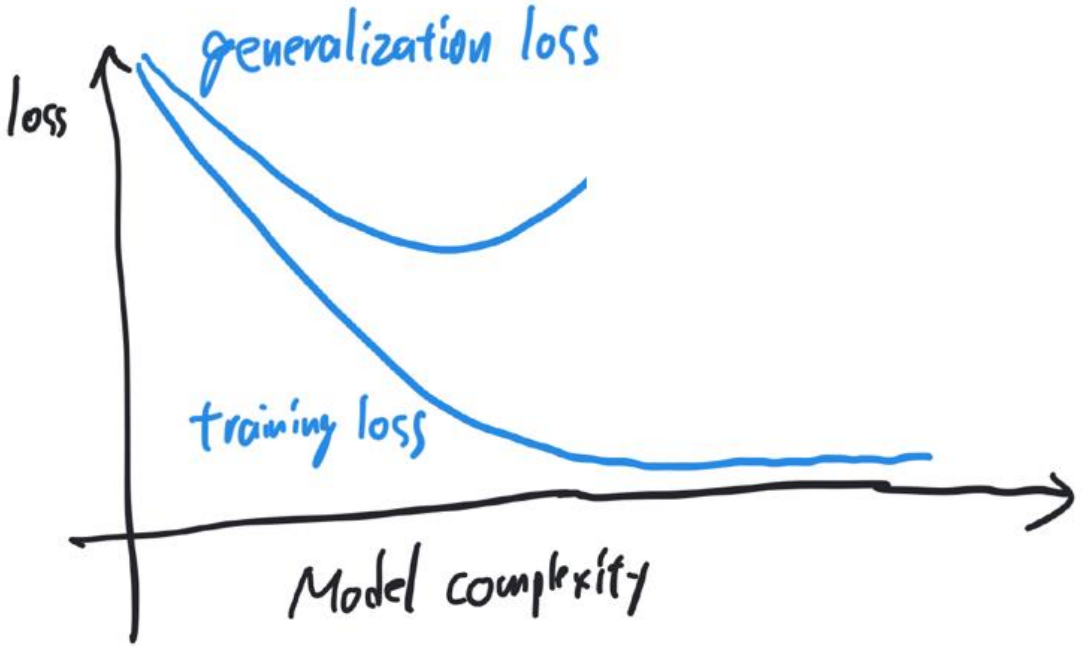
- Should allow NN to learn the correct structure of data
- Will often improve test score
- May hurt test score if train and test set share same non-invariant bias.
- For example, a photographer prefers jto take pictures with cats looking to the left and dogs looking to the right. NN may learn to distinguish cats from dogs by which direciton it is facing.)

Overfitting and Underfitting



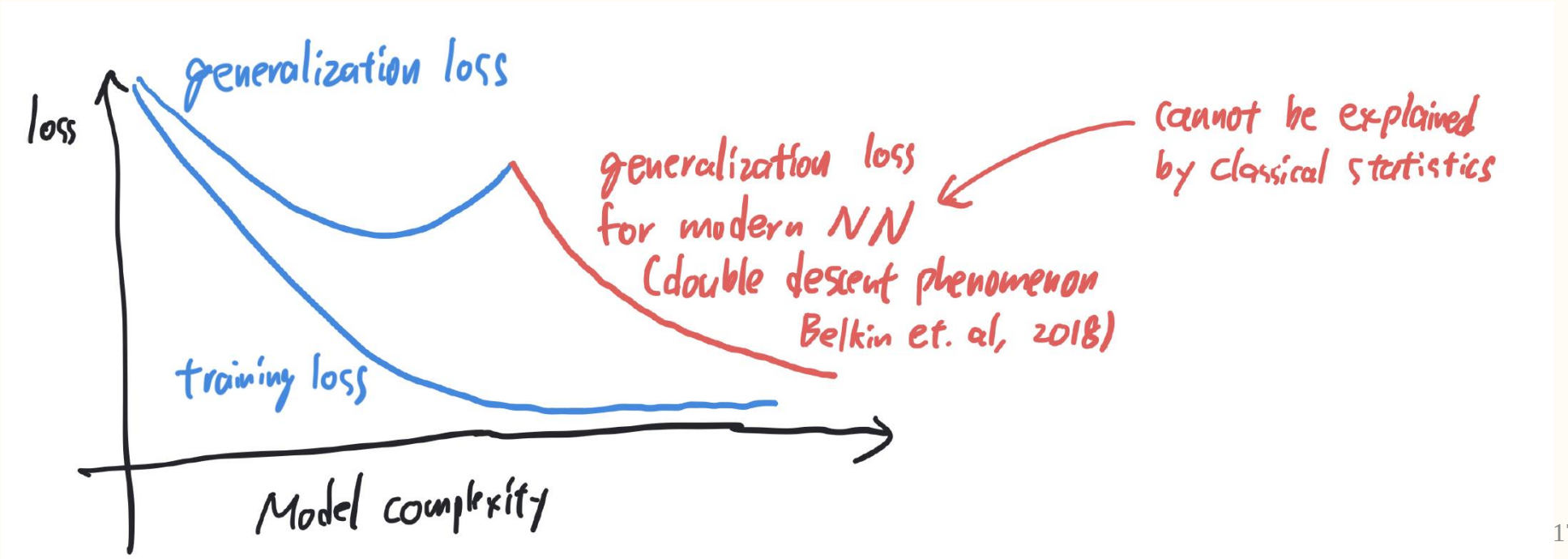
The goal of fitting data is to discover patterns that generalize to data you have not seen. From each datapont, you want to learn enough (don't underfit) but if you learn too much you overcompensate for an observation specific to the single experience.

In classical statistics, underfitting vs. overfitting (bias vs. variance tradeoff) is characterized rigorously.



The goal of fitting data is to discover patterns that generalize to data you have not seen. From each datapont, you want to learn enough (don't underfit) but if you learn too much you overcompensate for an observation specific to the single experience.

In classical statistics, underfitting vs. overfitting (bias vs. variance tradeoff) is characterized rigorously.



Basic Principles of Model Selection

Separate testing and training data

- When (traiing loss) $\ll$ (testing loss) you are overfitting
- When (traiing loss) $\approx$ (testing loss) you are underfitting

When you are overfitting, regularize by using

- A smaller NN (fewer parameters) or gather more daa
- Weight decay
- Dropout
- Early stopping on SGD
- Data augmentation

When you are underfitting, use

- Larger NN (if computationally feasible)
- Reduce weight decay
- Reduce dropout
- Run SGD longer (if computationally feasible)

Weight Decay (ℓ^2-regularization)

Squares
l2 norm

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i) + \frac{\lambda}{2}\|\theta\|^2$$

Classical statistics:
Ridge regression,
Gaussian Bayesian prior,
James-Stein estimator

Usualy achieved not by explicitly changing loss funciton but rather by changing SGD update

$$\begin{aligned}\theta^{k+1} &= \theta^k - \alpha\left(g^k + \lambda\theta^k\right) \\ &= (1 - \alpha\lambda)\theta^k - \alpha g^k\end{aligned}$$

g^k is stochastic
gradient of original loss

Dropout (Srivastava, Hinton, Krizhevsky, Sutskever, Salakhutdino, 2015)

Dropout is a regularization technique that randomly disables neurons

Standard layer

$$h_2 = \sigma(W_1 h_1 + b_1)$$

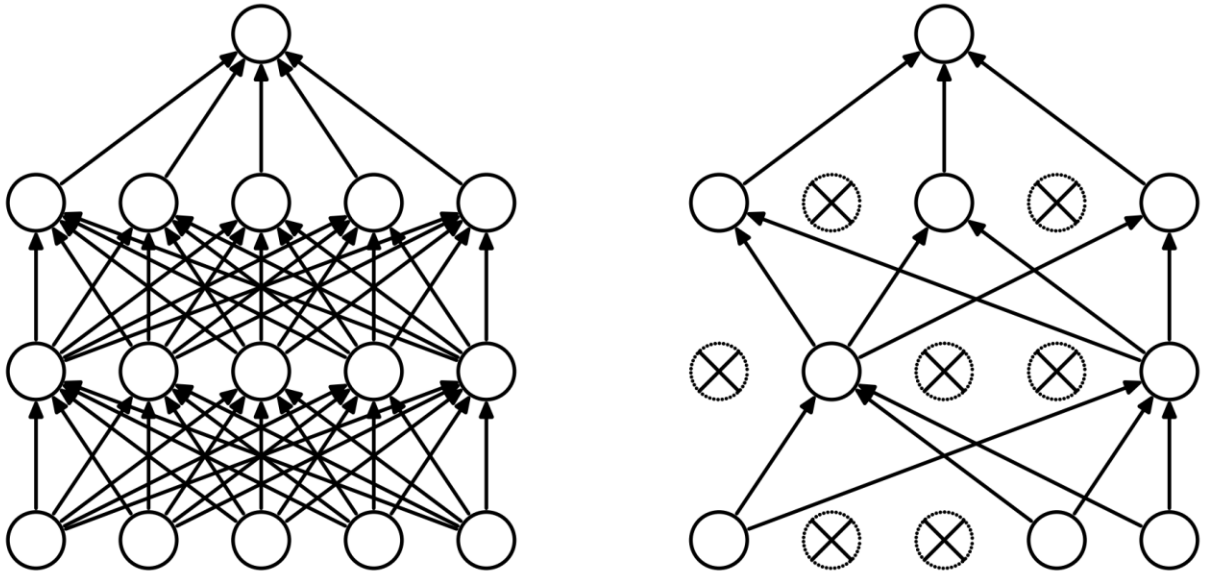
With dropout

$$h_1' = \begin{cases} 0 \text{ with probability } p \\ \dfrac{(h_1)_j}{1-p} \text{ otherwise} \end{cases}$$

p is probability of drop.

h_1' defined so that $\mathbb{E}[h_1'] = h_1$.

$$h_2 = \sigma(W_1 h_1' + b_1)$$



(a) Standard Neural Net (b) After applying dropout.

Figure 1: Dropout Neural Net Model. **Left**: A standard neural net with 2 hidden layers. **Right**: An example of a thinned net produced by applying dropout to the network on the left. Crossed units have been dropped.

Why helpful?

Typically, dropout is used during training
(mode.train() in PyTorch)
and turned off during inference/testing
(model.eval() in PyTorch)

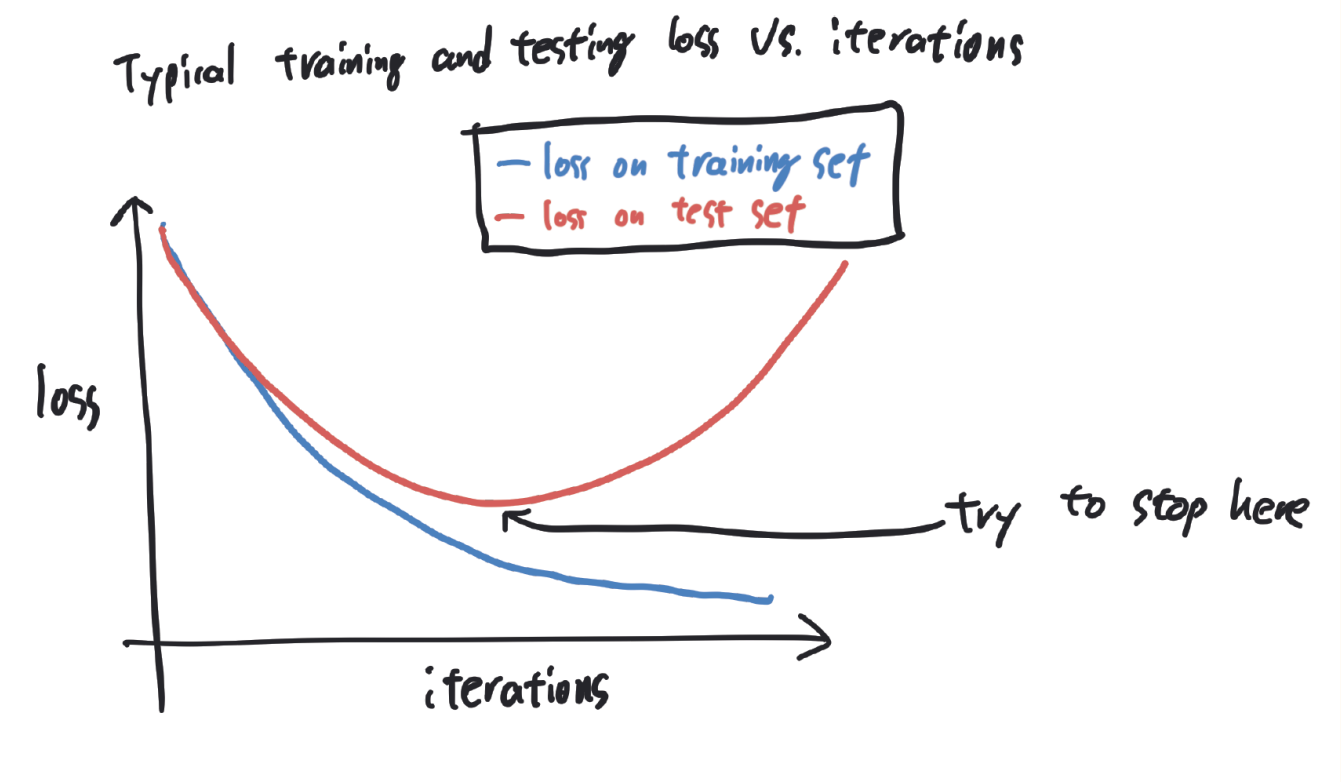
2. Motivation

A motivation for dropout comes from a theory of the role of sex in evolution (Livnat et al., 2010). Sexual reproduction involves taking half the genes of one parent and half of the other, adding a very small amount of random mutation, and combining them to produce an offspring. The asexual alternative is to create an offspring with a slightly mutated copy of the parent's genes. It seems plausible that asexual reproduction should be a better way to optimize individual fitness because a good set of genes that have come to work well together can be passed on directly to the offspring. On the other hand, sexual reproduction is likely to break up these co-adapted sets of genes, especially if these sets are large and, intuitively, this should decrease the fitness of organisms that have already evolved complicated co-adaptations. However, sexual reproduction is the way most advanced organisms evolved.

One possible explanation for the superiority of sexual reproduction is that, over the long term, the criterion for natural selection may not be individual fitness but rather mix-ability of genes. The ability of a set of genes to be able to work well with another random set of genes makes them more robust. Since a gene cannot rely on a large set of partners to be present at all times, it must learn to do something useful on its own or in collaboration with a *small* number of other genes. According to this theory, the role of sexual reproduction is not just to allow useful new genes to spread throughout the population, but also to facilitate this process by reducing complex co-adaptations that would reduce the chance of a new gene improving the fitness of an individual. Similarly, each hidden unit in a neural network trained with dropout must learn to work with a randomly chosen sample of other units. This should make each hidden unit more robust and drive it towards creating useful features on its own without relying on other hidden units to correct its mistakes. However, the hidden units within a layer will still learn to do different things from each other. One might imagine that the net would become robust against dropout by making many copies of each hidden unit, but this is a poor solution for exactly the same reason as replica codes are a poor way to deal with a noisy channel.

SGD early stopping

- Stop SGD early even if you have time for more iterations
- SGD iterations are effort to fit data ⇒ too many iterations lead to overfitting



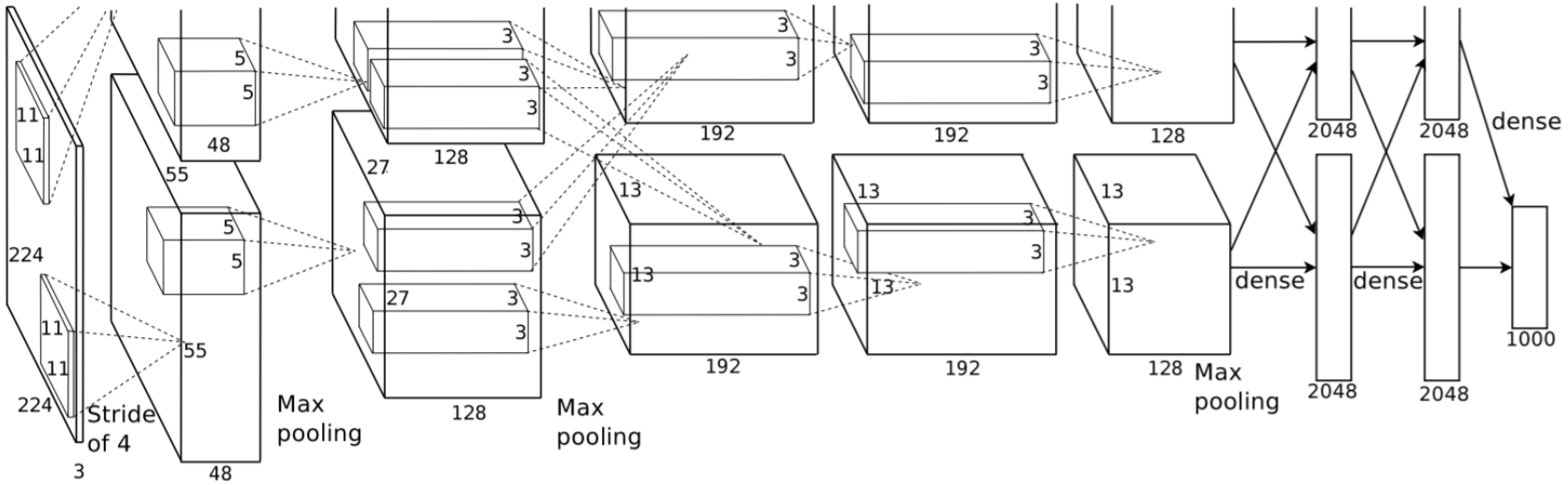
More data (by data auagmentation)

With everything else fixed, more data ⇒ less overfitting (of course, data is expensive)

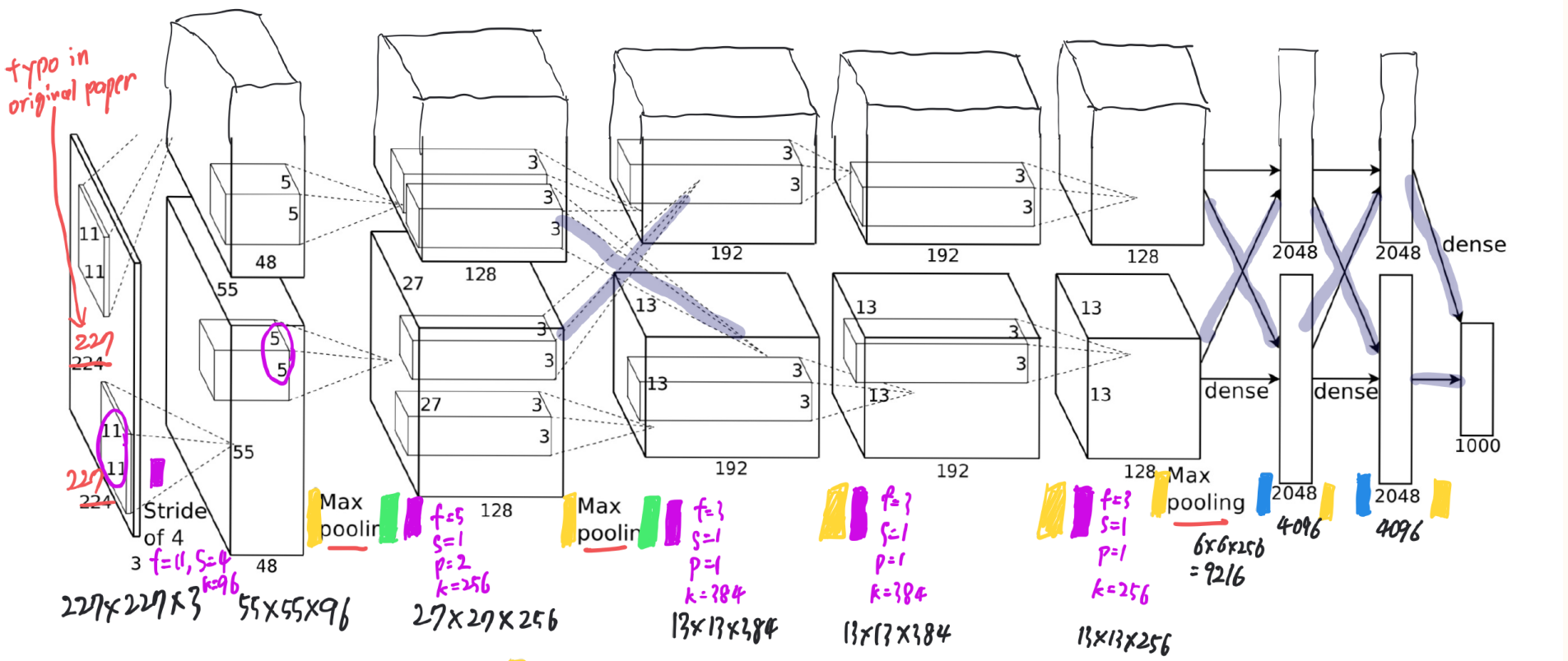
Data augmentation is a way to create more data for free. You can view data augmentatino as a form of regularization.

AlexNet (Alex Krizhevsky, Sutskever, Hinton, 2012)

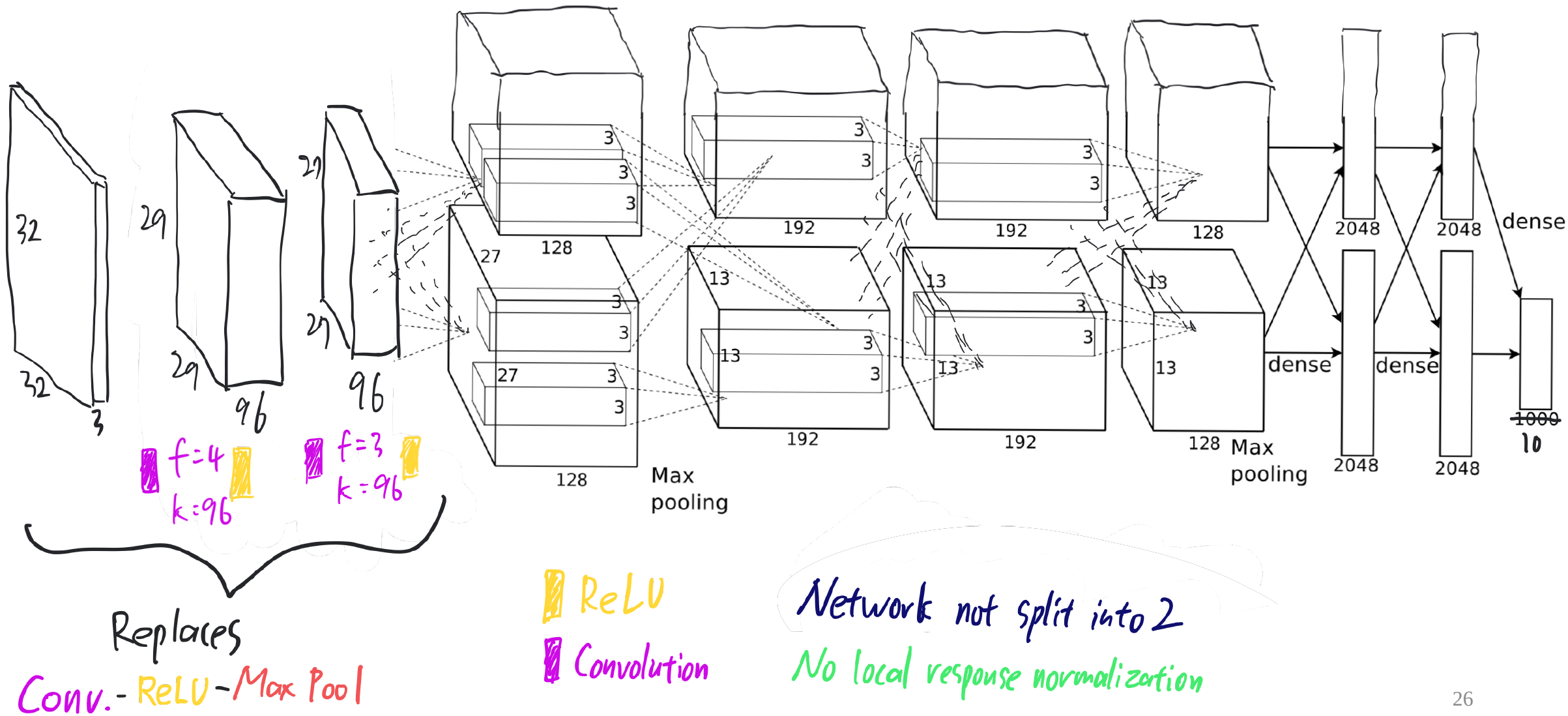
- Won the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) by a large margin (15.3% error rate vs. 26.2% second place)
- Started the era of deep neural networks and their training via GPU computing.
- AlexNet was split into 2 as GPU memory was limited at the time. (A single modern GPU can easily hold AlexNet.)



AlexNet for ImageNet



AlexNet CIFAR10



Weight Intialization

Remember, SGD is

$$\theta^{k+1} = \theta^k - \alpha g^k$$

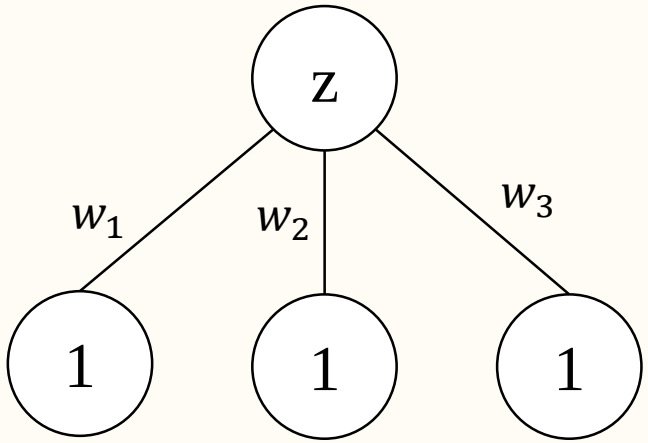
When $\theta^0 \in \mathbb{R}^p$ is an initial point.

Using a good initial point is especially important in NN training because it is a nonconvex optimization problem.

Using $\theta^0 = 0$ is a bad idea.

In a multi-layer perceptron, the gradients vanish if $\theta^0 = 0$ is used. So the training is stuck at $\theta^0 = 0$ and never moves.

Consider



$$z = w_1 + w_2 + w_3$$

If $w_i \sim \mathcal{N}(0, \sigma)$ (zero-mean Gaussian with standard deviation σ) then var(z)= $3\sigma^2$. If $\sigma = \frac{1}{\sqrt{3}}$ then Var(z)=1.

Since tahnh and sigmoid activation functions satuarate outside of ± 1, it is a good idea to initialize weights to keep outputs within ± 1.

Xavier and He Initialization

Xavier initialization (Xavier Glorot, Yoshua Bengio, 2010)

$$\sigma = \sqrt{\frac{2}{n_{\text{in}} + n_{\text{out}}}}$$

He initialization (Kaiming He, X. Zhang, S. Ren, J. Sun, 2015)

$$\sigma = \sqrt{\frac{2}{n_{\text{in}}}}$$

Both Xavier and He initialization can use Gaussian or uniform random variables.

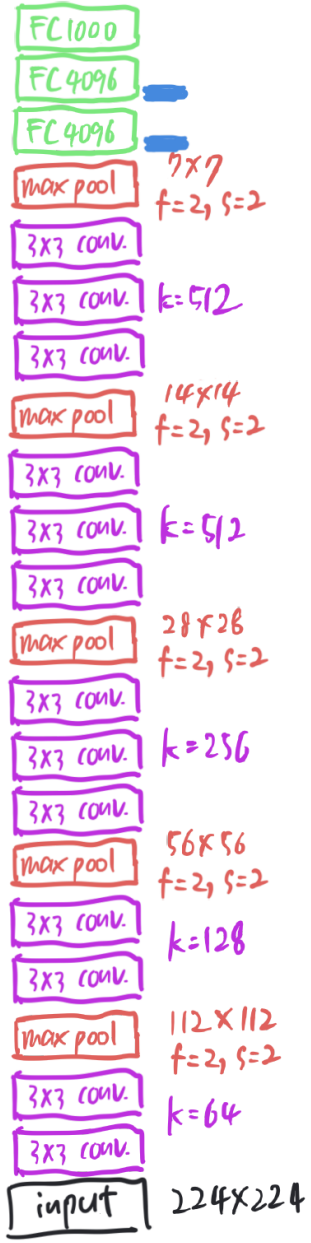
ImageNet After AlexNet(2012)

- 2013 winner of ImageNet challenge was ZFNet with 8 layers. (by Zeiler and Fergus)
- ZFNet brought better hyperparameter tuning of AlexNet
- Research since AlexNet indicate that depth is more important than width.
- AlexNet had 8 layers (with trainable parameters)
- VGGNet ranked 2nd place on the 2014 ImagNet challenge with 19 layers.
- GoogLeNet ranked 1st place on the 2014 ImageNet challenge with 22 layers.

VGGNet by the Oxford Visual Geometry Group

VGG16

- 16 layers with trainable weights
- Conv. layers preserve spatial dim. (so p=1)
- No local response normalization
- weight decay 5×10^{-4}
- Dropout (0.5) used
- ReLU activation function (except after pool and FC1000)



VGG19

- 19 layers with trainable weights
- Slightly better than VGG16

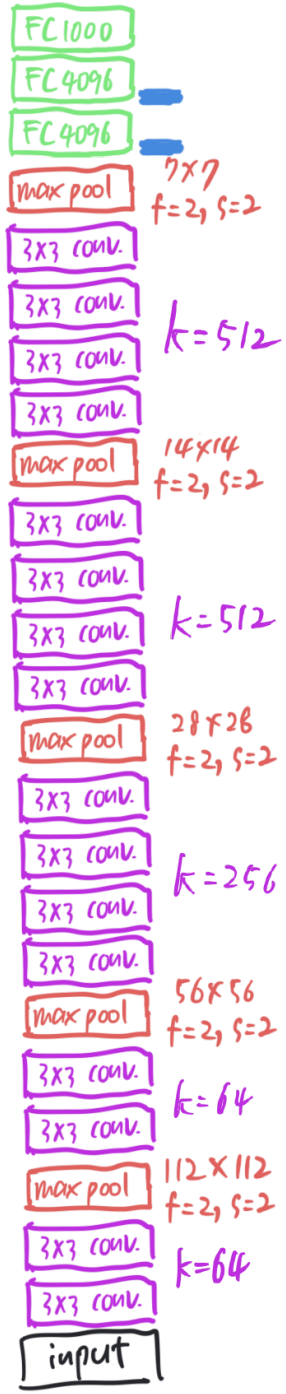
- Training VGG and GoogLeNet is tricky due to their depth.

Solution:

(i) Train smaller versions and gradually add layers.

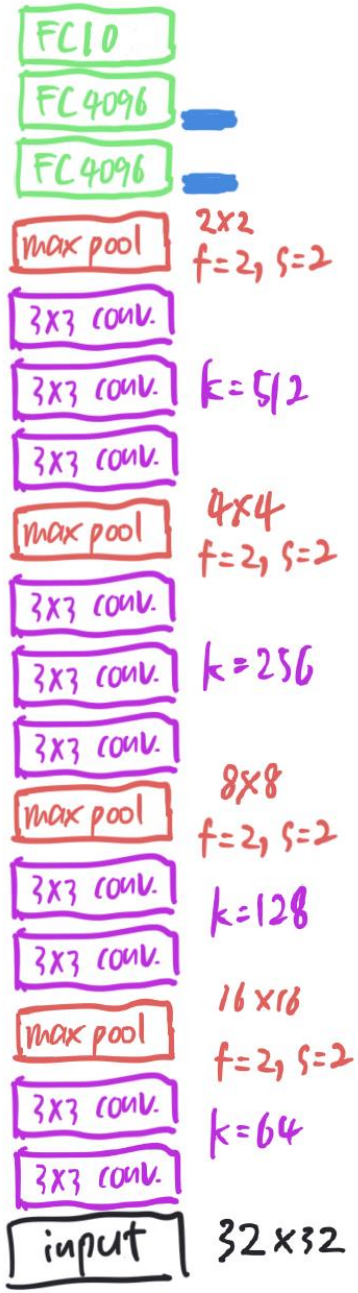
(ii) Initialize weights carefully

(iii) Batch norm (next time)



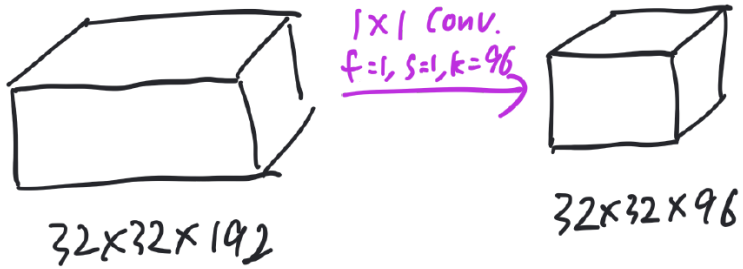
VGG for CIFAR10

- 13-layer modification of VGG for CIFAR10



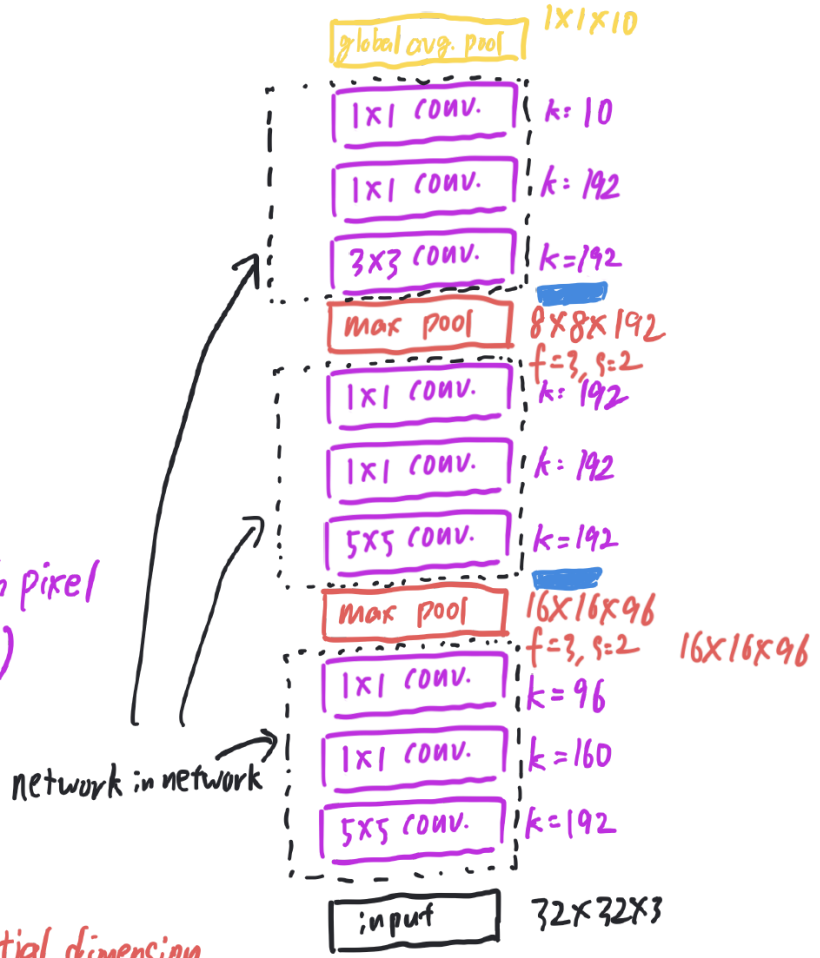
Network in Network (NiN) Network

- Remove dense layers, which require large # of parameters.
- Utilize 1x1 convolutions.
- 1x1 conv. is like a fully connected layer acting independently on each spatial location.



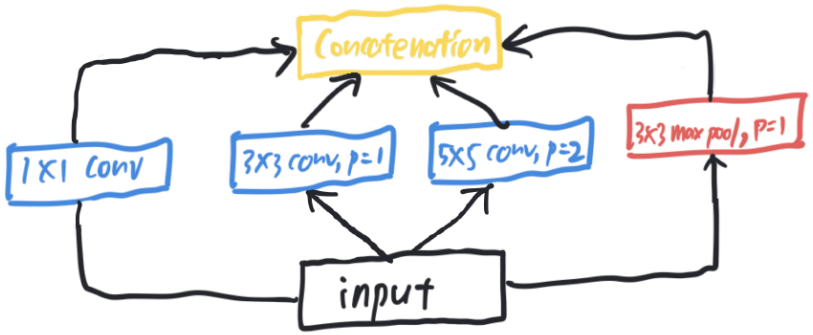
- 96 filters act on 192 channels separately for each pixel
  - 96x192 (+96) parameters for weights (and biases)
- Weight decay 0.00001
- Dropout(0.5) used
- Pooling with f=3, s=2 use ceil_mode=True to preserve spatial dimension

NiN Network for CIFAR10

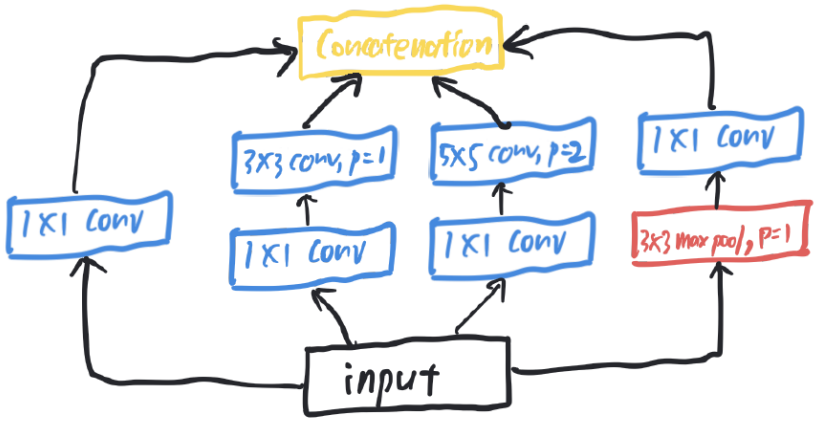


GoogLeNet (Inception v.1)

- Utilizes the "inception" module, which structure was inspired by NiN and name by the meme



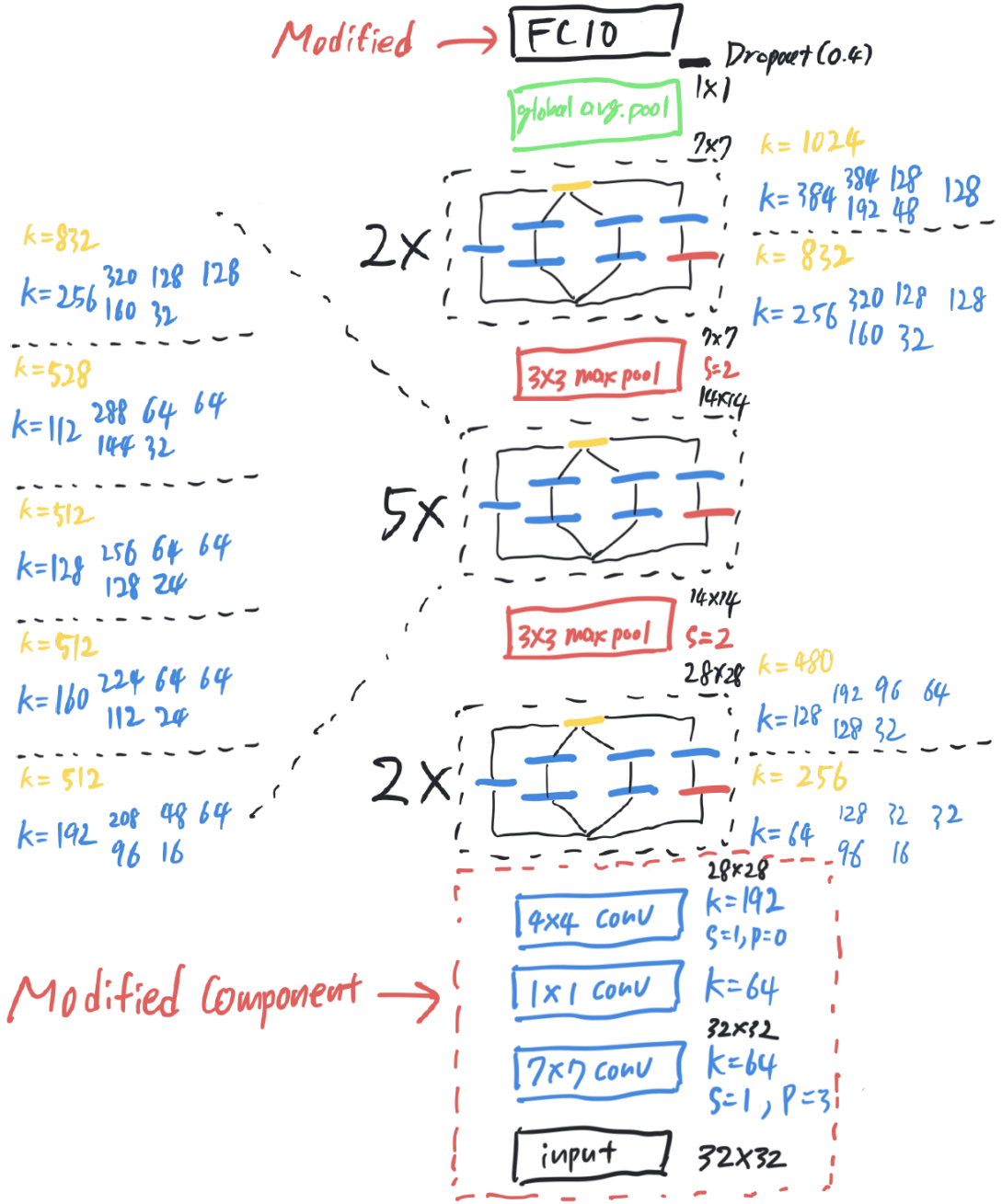
Naive inception module



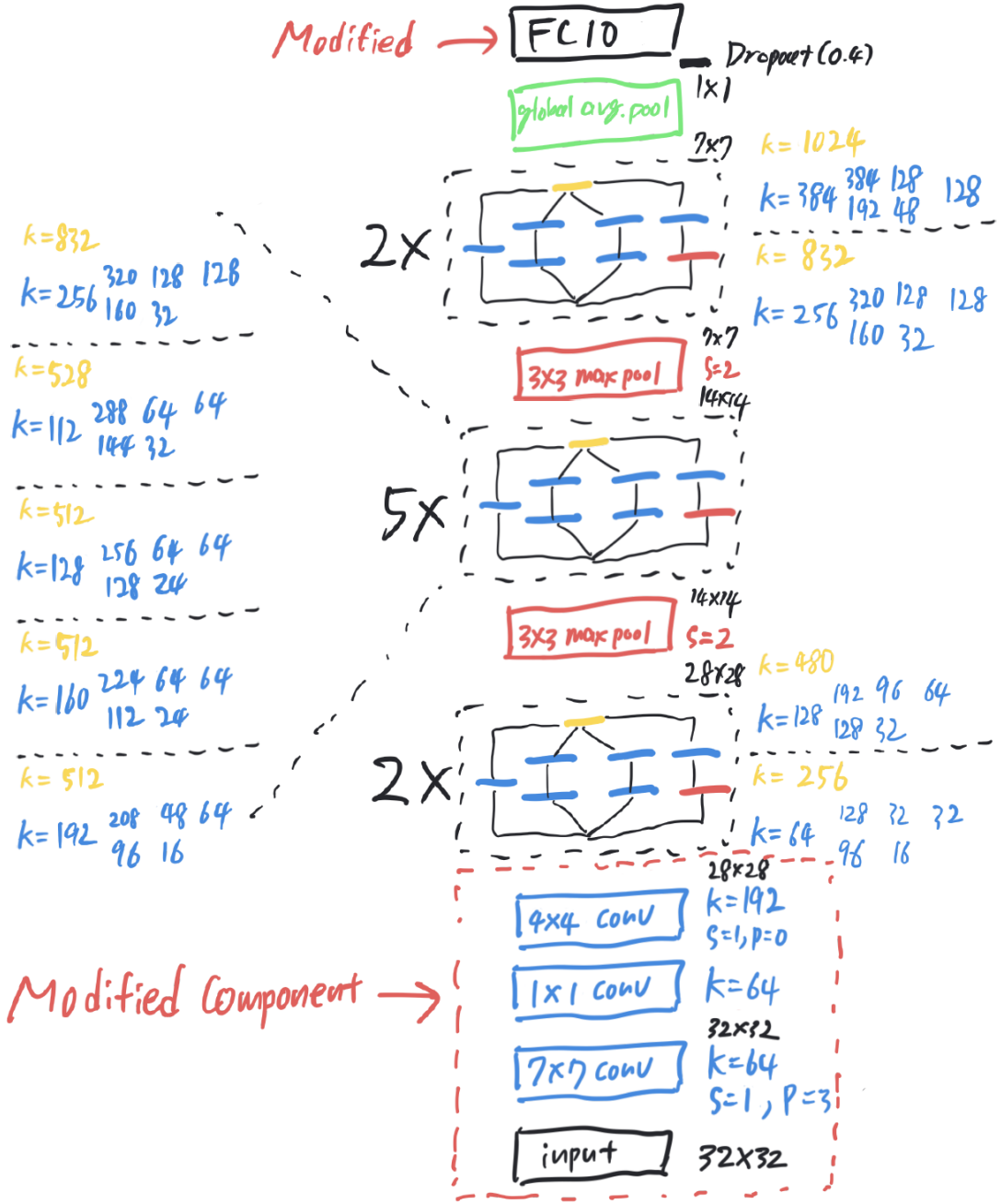
Inception module

The 1x1 convolution

- Increases depth (thereby improving the ability to represent nonlinear functions)
- Reduces # of channels and computation time (hw d)


Modified
FC 10
Dropout (0.4)
1x1
global avg. pool
7x7
k = 1024
2x
k = 384
384 128 128
192 48
k = 832
k = 256
320 128 128
160 32
k = 832
k = 256
320 128 128
160 32
k = 528
k = 112
288 64 64
144 32
k = 512
k = 128
256 64 64
128 24
k = 512
k = 160
224 64 64
112 24
k = 512
k = 192
208 48 64
96 16
7x7
3x3 max pool
S=2
14x14
5x
14x14
3x3 max pool
S=2
28x28
k = 480
2x
k = 128
192 96 64
128 32
k = 256
k = 64
128 32 32
96 16
28x28
4x4 conv
k = 192
S=1, P=0
1x1 conv
k = 64
32x32
7x7 conv
k = 64
S=1, P=3
input
32x32
Modified Component

GoogLeNet for CIFAR10



Batch Normalization

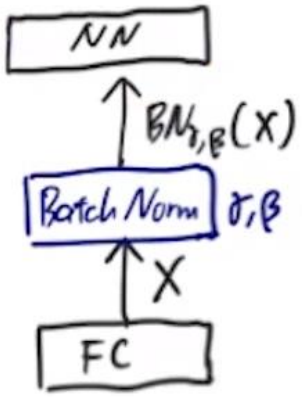
- In many classical methods involving data, the first step is to normalize data to have zero mean and unit variance.
  - Step 1. Compute $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} X_i$, $\widehat{\sigma^2} = \frac{1}{N}\sum_{i=1}^{N}(X_i - \hat{\mu})^2$

$$\hat{X}_i = \frac{X_i - \hat{\mu}}{\sqrt{\widehat{\sigma^2} + \varepsilon}}$$

  - Step 2. Run method with data $\hat{X}_1, \ldots, \hat{X}_N$

- Batch normalization n (BN) (sort of) enforces this normalization layer-by-layer. (Ioffe & Szegedy 2015)
- BN has become an indispensable tool for training very deep neural networks.
- Theoretical justification for BN is weak.

BN for Fully Connected Layer



Input: X (batch size) × (# entries)

Output: $BN_{\gamma,\beta}(X)$ Shape($BN_{\gamma,\beta}(X)$) = Shape(X)

$BN_{\gamma,\beta}$ acts independently over entries

$$\hat{\mu}[:] = \frac{1}{B}\sum_{b=1}^{B} X[b,:]$$

$$\hat{\sigma}[:] = \frac{1}{B}\sum_{b=1}^{B} (X[b,:] - \hat{\mu}[:])^2 + \epsilon$$

learned standard dev. and mean parameters

$$BN_{\gamma,\beta}(X) = \gamma \circledast \frac{X - \hat{\mu}}{\hat{\sigma}} + \beta$$

⊛ elementwise mult

$$BN_{\gamma,\beta}(X)[b,:] = \gamma[:] \circledast \frac{X[b,:] - \hat{\mu}[:]}{\hat{\sigma}[:]} + \beta[:] \quad b=1,\dots,B$$

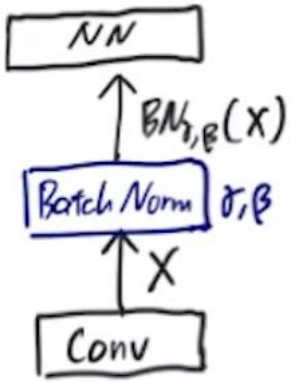
BN normalizes X and controls the mean and variance through learned parameters γ and β.

BN for Convolutional Layer

Assuming translation invariance, each batch, horizontal pixel, and vertical pixel is like an IID sample

Input: X (batch size) × (# channels) × (# horizontal pixels) × (# vertical pixels)

Output: $BN_{\gamma,\beta}(X)$ Shape $(BN_{\gamma,\beta}(X)) = \text{Shape}(X)$



$BN_{\gamma,\beta}$ acts independently over channels

$$\hat{\mu}[:] = \frac{1}{BPQ}\sum_{b=1}^{B}\sum_{i=1}^{P}\sum_{j=1}^{Q} X[b,:,i,j]$$

$$\hat{\sigma}^2[:] = \frac{1}{BPQ}\sum_{b=1}^{B}\sum_{i=1}^{P}\sum_{j=1}^{Q} (X[b,:,i,j] - \hat{\mu}[:])^2 + \epsilon$$

$$BN_{\gamma,\beta}(X) = \gamma \circledast \frac{X-\hat{\mu}}{\hat{\sigma}} + \beta$$

learned standard dev. and mean parameters

$\circledast$ elementwise mult

$$BN_{\gamma,\beta}(X)[b,:,i,j] = \gamma[:] \circledast \frac{X[b,:,i,j] - \hat{\mu}[:]}{\hat{\sigma}[:]} + \beta[:]$$

$b = 1,\dots,B$
$i = 1,\dots,P$
$j = 1,\dots,Q$

Batch Norm During Prediction

- The $\hat{\mu}$ and $\hat{\sigma}$ are estimated from batches during training.
- During testing, where we evaluate the NN without changing it, $\hat{\mu}$ and $\hat{\sigma}$ are fixed.
- 2 strategies for computing final values of $\hat{\mu}$ and $\hat{\sigma}$:
  1) After training, fix weights and evaluate NN on full training set to compute $\hat{\mu}$ and $\hat{\sigma}$ layer-by-layer. (Computation of $\hat{\mu}$ and $\hat{\sigma}$ for initial layers must be done first.)
  2) During training, compute running average of $\hat{\mu}$ and $\hat{\sigma}$. This is the default behavior of PyTorch.
- In PyTorch, use model.train() and model.eval() to switch BN behavior between training and testing.

Discussion of Batch Normalization

- With BN, the choice of batch size becomes more important.
- For some reason, BN seems to subsume the regularization of Dropout. Since BN has been popularized, Dropout is used less often. (Not understood very well.)
- BN is indispensable in practice, but has very little theoretical justification.
- In the original paper, Ioffe and Szegedy offered the mitigation of “internal covariate shift” but (Santukar et al. 2018) showed that BN worsens the internal covariate shift and yet improves the training process.