5. Regularization + CNN Architectures

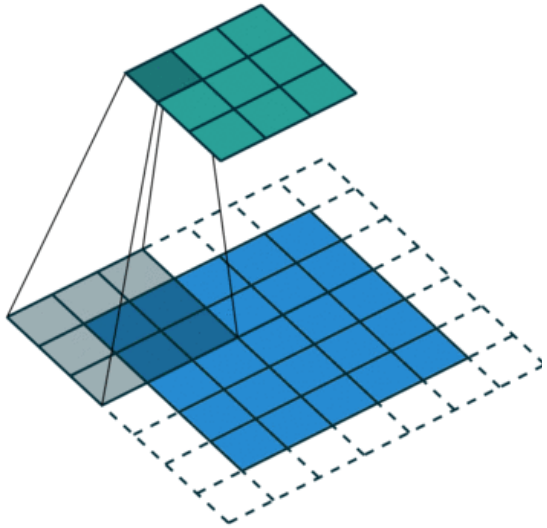
CS 5242 Neural Networks and Deep Learning

YOU, Yang

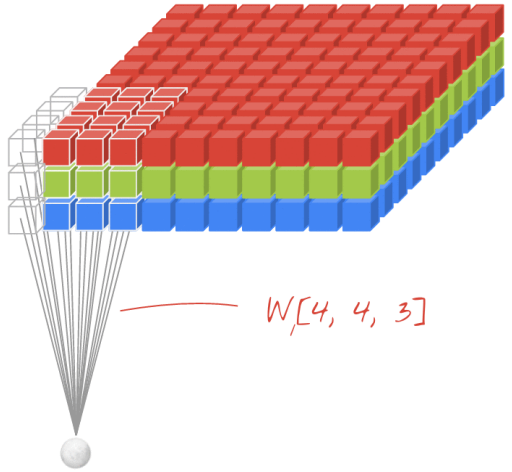
13.09.2022

Recap

- Convolution
  - 1D, 2D, 3D
  - Feature extraction/transformation
- Configuration
  - Kernel/filter, stride, padding
  - Receptive field size == kernel size
  - Output size?
- Implementation
  - Naïve: Inner-product between the kernel and receptive field
  - More efficient: Column to matrix
    - Backward-propagation needs to copy the gradients from matrix to column
  - Computational cost?



Source: http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html



Source: https://goo.gl/RviR5N

Slide credit: Wang Wei

Recap

- Aggregate the information from each receptive field
  - Feature extraction/summarization
    - The output is invariant to some variations of the input
  - Average & max pooling
- Pooling is applied to each channel / feature map separately

X (k=3, s=1, p=0)

0	1	2	**10**
4	5	6	7
8	**9**	1	2
3	4	5	6

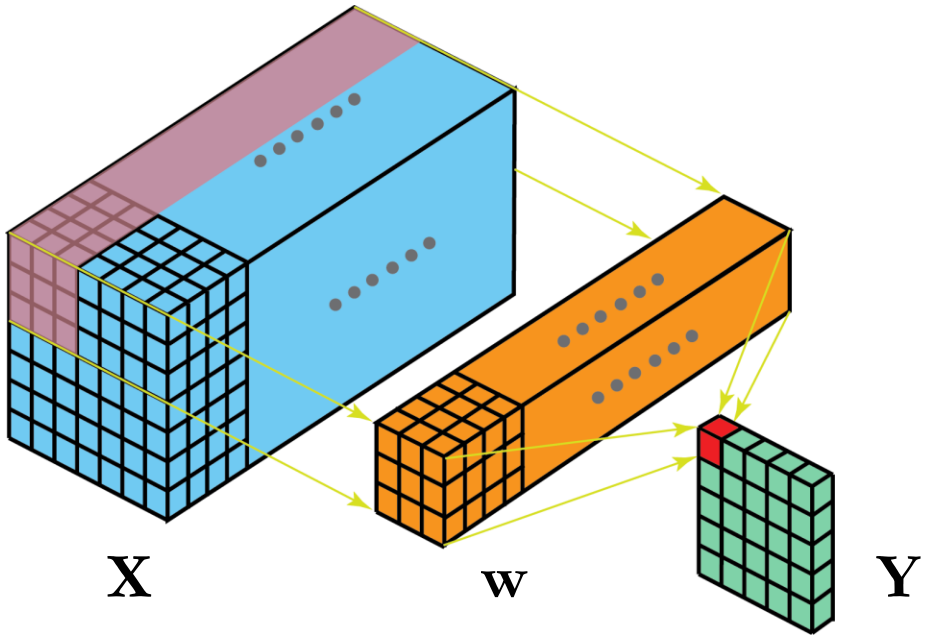
Y

9	10
9	9

Slide credit: Wang Wei

2D Convolution on 3D Data

Q: How to apply 2D convolution to 3D data?



- 2D convolution slides a 3D filter matrix through the input layer.
- input layer and filter have the same depth or number of channels (equal number of channels).
- 3D filter moves only in 2 directions: height & width of the input (hence being a 2D convolution.
- output here is a one-layer matrix.
- Here, 2D means 2 moving directions

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

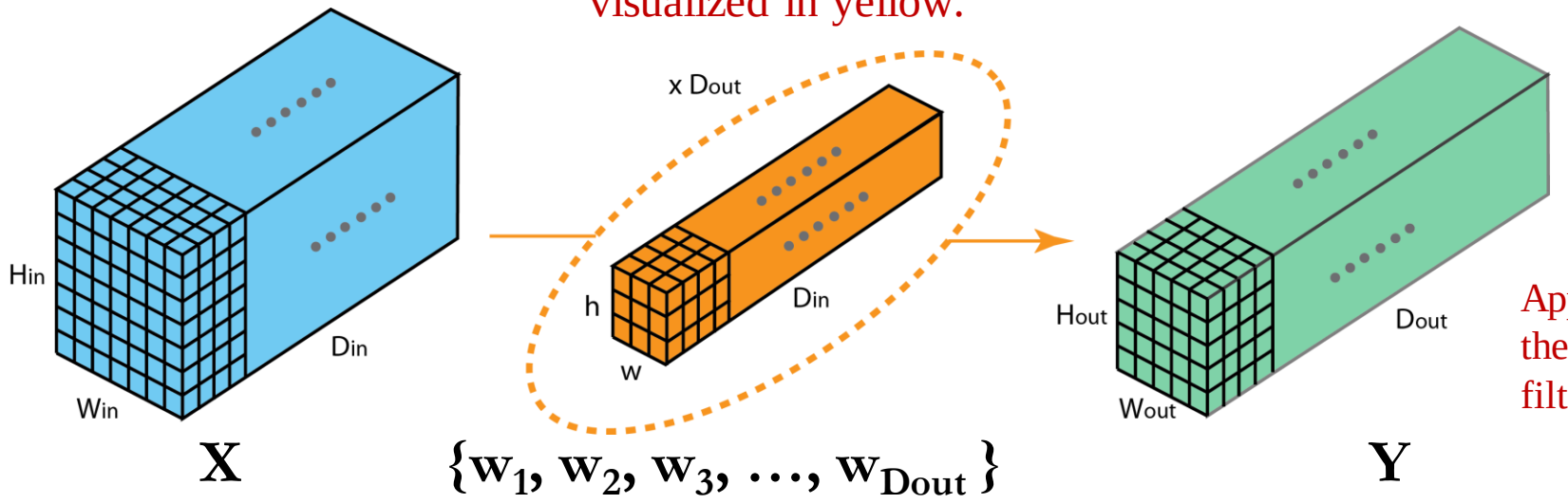
2D Convolution on 3D Data

Q: How to apply 2D convolution to 3D data?

To get a multi-channeled output, apply multiple 3D filter kernels to get a multi-channeled output.

Apply multiples of the filter visualized in yellow.

the input layer has *Din* channels, and we want the output layer has *Dout* channels

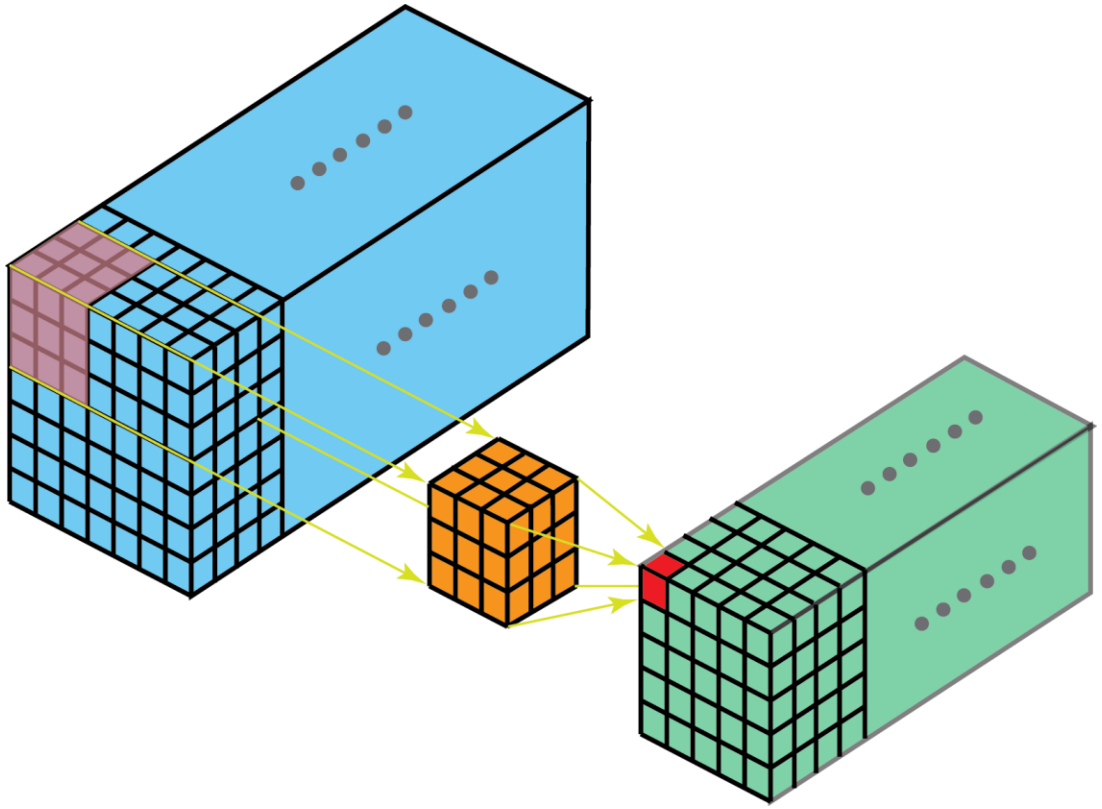


$\mathbf{X}$ $\quad \{w_1, w_2, w_3, \ldots, w_{Dout}\}$ $\quad \mathbf{Y}$

Apply *Dout* filters to the input layer. Each filter has *Din channels*

https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

2D Convolution on 3D Data

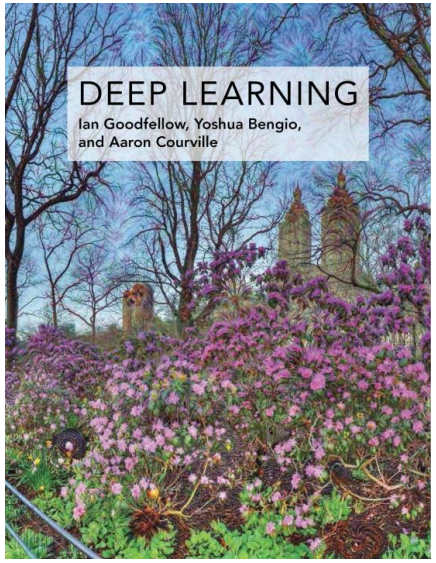


- 3D convolution applies a kernel (can be 1D / 2D / 3D) that moves in all 3D dimensions of the input data.
- note that the number of channels in the kernel must be less than the number of channels in the input data.
  - input layer depth: K1
  - filter depth: K2
  - K1 = n*K2 (n>=1)
  - e.g. K1=15; K2=3

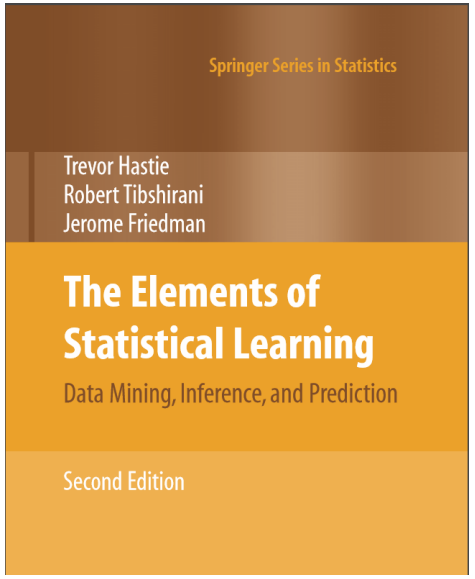
https://towardsdatascience.com/a-comprehensive-introduction-to-different-types-of-convolutions-in-deep-learning-669281e58215

Agenda

- Regularization
- Deep CNN Architectures
  - LeNet-5
  - AlexNet



[online version]



[NUS library e-version ink]

Several parts of today's lecture are drawn from the above two references. Have a look if you want to read further.

Regularization

Definitions, parameter norm penalty, reducing model capacity,

Leading up to the modern CNNs: LeNet & AlexNet

What is Regularization?

- Loosely defined as any form of modification to a learning algorithm intended to reduce its generalization error but not training error.

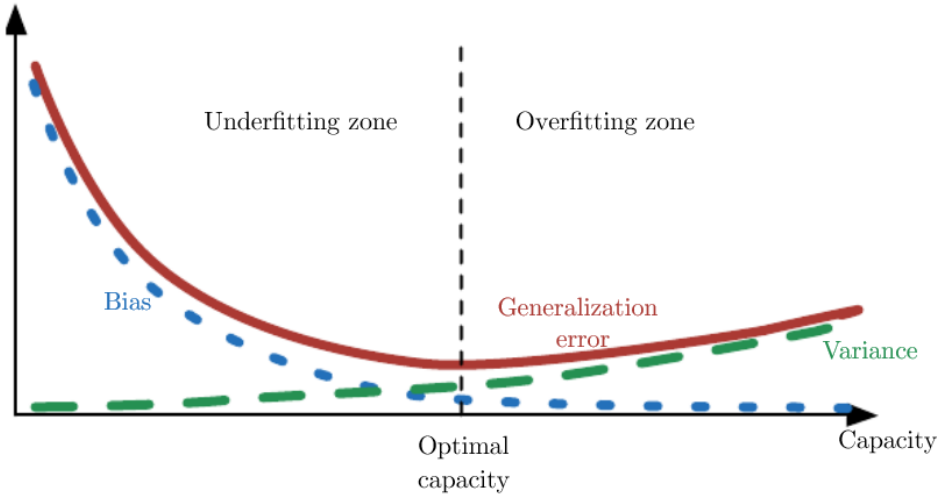
Quick recap: do you remember what training error and generalization error are?

Indicator of "performance" on training data, likely to be good since we are learning parameters to fit exactly to this data; not interesting.

How well model does on previously unseen data – i.e. test data. Is a better indicator of performance.

- Strategies include:
  - extra constraints on the model parameters
    - e.g. dropout and parameter sharing
  - extra terms in the objective (similar to soft constraints on the parameters)
    - e.g. weight decay

Regularization in Deep Learning



- A good regularizer reduces variance significantly while not overly increasing bias.
- When training, 3 typical situations of bias-variance trade-off
  1. model excludes the true data generating process and under-fits, therefore inducing bias
  2. model matches the true data generating process
  3. model includes the true data generating process but also many other possible generating processes and over-fits, therefore allowing variance to dominate the estimation error (instead of bias)

Often in training (deep) neural networks, we end up with models in the third situation.
Ideally, we'd like to shift models into the second situation with regularization.

Regularization in Deep Learning

- Practically, applications for deep networks are from very complicated domains:
  - images
  - audio sequences
  - text
- It's impossible to simulate (and evaluate) the true generating process
- Therefore, controlling the complexity of deep models is not about finding a model of right size / having the right number of parameters
- Instead, we want to find a (very) large model that’s properly regularized.
- What are strategies for learning large, deep, regularized models?

Strategies for Regularization in Deep Learning

1. Parameter Norm Penalties
2. Reducing Model Capacity
   - Early stopping
   - Parameter sharing
   - Drop-out
3. Adjusting Training Data
   - Data augmentation
   - Injecting noise
   - Batch Normalization

Parameter Norm Penalties

Consider as all parameters / weights of the neural network.

∴ Add penalty $\Omega(\boldsymbol{W})$ to regularize the objective function J:

$$J(\boldsymbol{W}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{W}; \boldsymbol{X}, \boldsymbol{y}) + \alpha\, \Omega(\boldsymbol{W})$$

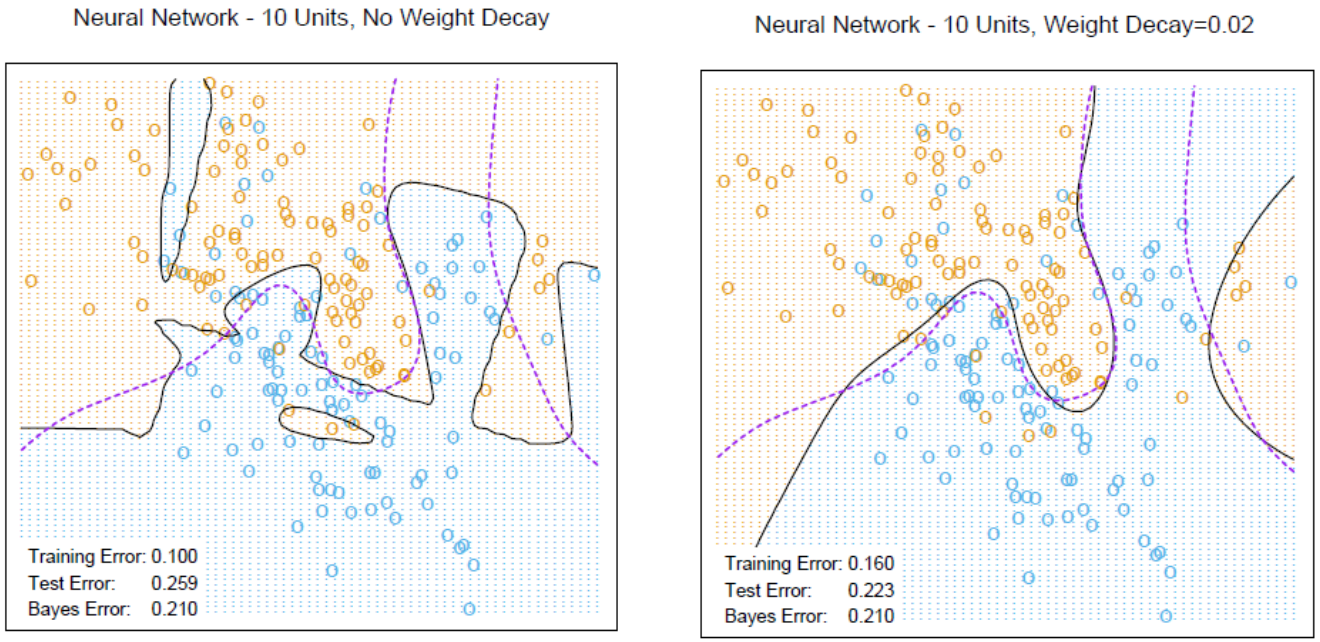
- α is a hyper-parameter weighting the relative effect of the penalty term
  - Larger α means more regularization, smaller α means less regularization
- During training, minimize both objectives:
  - Original objective, based on task e.g. L2, cross-entropy,
  - Regularization objective, based on some size measure of network parameters $\boldsymbol{W}$, e.g. vector norm

$$J_{reg} = J + \frac{\alpha}{2}\left|\left|\boldsymbol{W}\right|\right|^2$$

where $\left|\left|\boldsymbol{W}\right|\right|^2$ is the L2 norm of the parameters treated as a flattened vector

- This constrains $\boldsymbol{W}$ to be small values (and is therefore sometimes called weight decay)
- For any $w \in \boldsymbol{W}$
  - $\frac{\partial J_{reg}}{\partial w} = \frac{\partial J}{\partial w} + \alpha w$

Parameter Norm Penalties



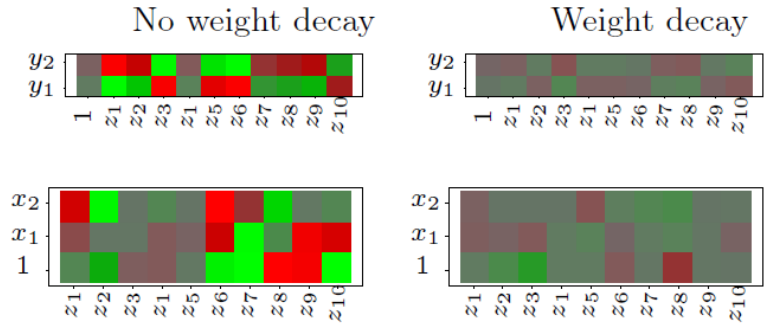


FIGURE 11.5. *Heat maps of the estimated weights from the training of neural networks from Figure 11.4. The display ranges from bright green (negative) to bright red (positive).*

Figure: 11.4 from Hastie. Example comparing effects of weight decay on over-fitting.

Parameter Norm Penalties

How does weight decay work? Consider the gradient of the regularized objective[1]:

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2}\boldsymbol{w}^\intercal \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

The parameter gradient is defined as

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, by) = \alpha \cdot \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

A gradient step in updating is then given as

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(\alpha \cdot \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})), \qquad \text{which is equivalent to}$$

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha) \cdot \boldsymbol{w} - \epsilon \cdot \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

The equivalent form shows that weight decay, when updating the weights, at each step performs a small multiplicative shrinking of the weights (from the first term) along with the standard gradient update (second term).

[1] For simplicity, assume no bias parameter.

Reducing Model Capacity

- Recall early stopping

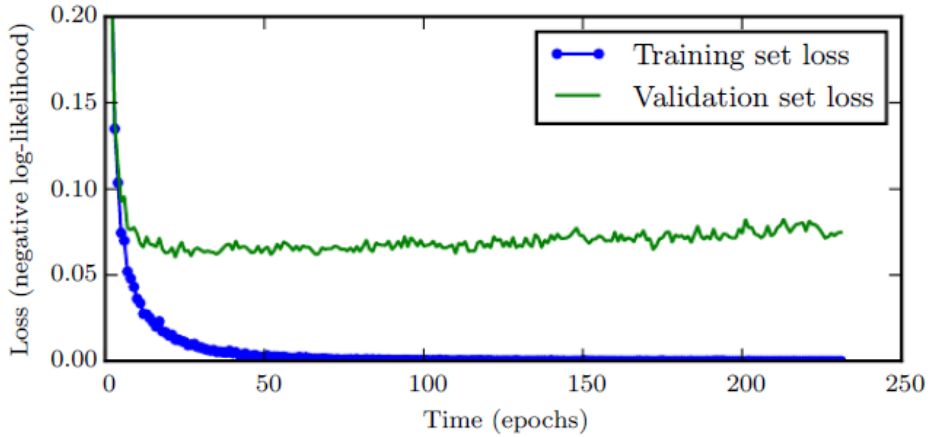


Figure: 7.3 from Goodfellow. Note how the training set loss continues to decrease over time but validation loss starts increasing once the learning process starts to over-fit after epoch 30 or so. Assuming that validation error is reflective of the test error, it's best to choose the model parameters from epoch 30.

Early Stopping as a Regularizer?

Picture early stopping as restricting an optimization procedure to a relatively small volume of parameter space in the neighbourhood of the initial parameter value. Taking τ optimization steps (training iterations) with learning rate ϵ, the product $\epsilon \cdot \tau$ is a measure of effective capacity (provided both are bounded).

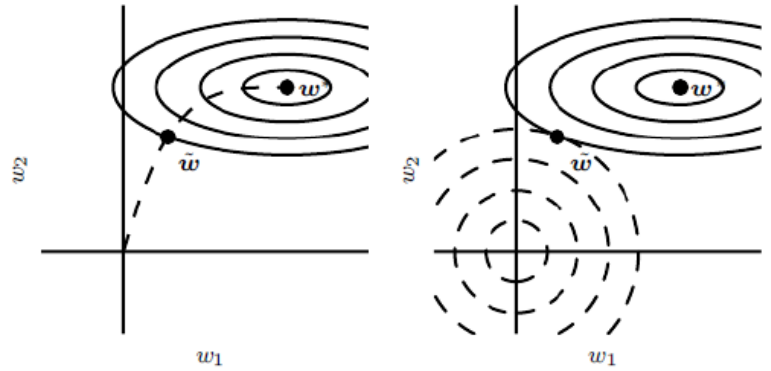


Figure 7.4: An illustration of the effect of early stopping. *(Left)*The solid contour lines indicate the contours of the negative log-likelihood. The dashed line indicates the trajectory taken by SGD beginning from the origin. Rather than stopping at the point $\boldsymbol{w}^*$ that minimizes the cost, early stopping results in the trajectory stopping at an earlier point $\tilde{\boldsymbol{w}}$. *(Right)*An illustration of the effect of L^2 regularization for comparison. The dashed circles indicate the contours of the L^2 penalty, which causes the minimum of the total cost to lie nearer the origin than the minimum of the unregularized cost.

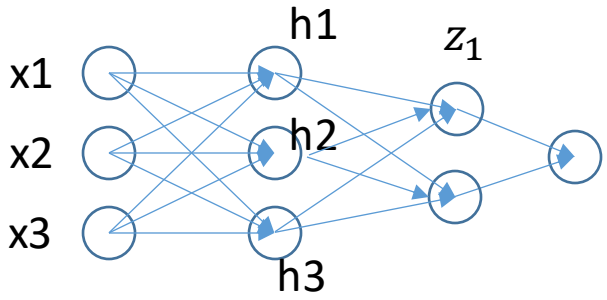
Figure: from Goodfellow.

Parameter Sharing

- Constraining / penalize parameters from deviating from some fixed value (e.g. for L2 regularization is a deviation from a value of 0) implies that we know what the fixed value should be and this may not always be the case.
- Based on knowledge of the domain and model architecture, we can sometimes express other forms of dependencies and priors.
- Parameter sharing force sets of parameters to be equal
  - Advantage: only a subset (one set) of the parameters need to be stored in memory
  - CNNs are the most popular and extensive use of parameter sharing
  - The convolution operation can also be interpreted as parameter sharing over all the receptive fields
  - Added benefit is that networks are much smaller and or can be trained with less data.

Dropout

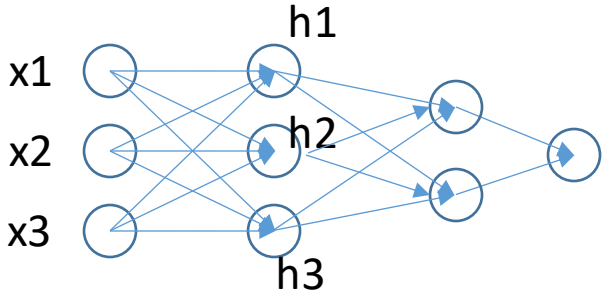
- Training
  - Randomly set some neurons to 0 with probability p (0.5, 0.4, 1/3 etc.)
  - Multiple the outputs (h) with scale 1/(1-p) ?
    - Without dropout/testing: $h_1, h_2, h_3 \rightarrow z_1$
    - Training with dropout: $h_1, h_3 \rightarrow \hat{z}_1$
      - $\hat{z}_1$ and z_1 are at different scales, $\hat{z}_1 \approx \frac{2}{3} z_1$
      - Rescale $\frac{1}{1-p} \hat{z}_1 = \frac{3}{2} \hat{z}_1 \approx z_1$
  - Different layers may have different dropout rate
- During inference
    - Do nothing

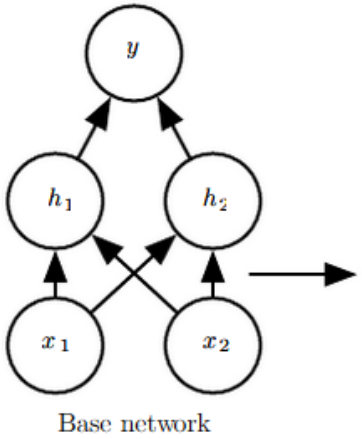


Dropout

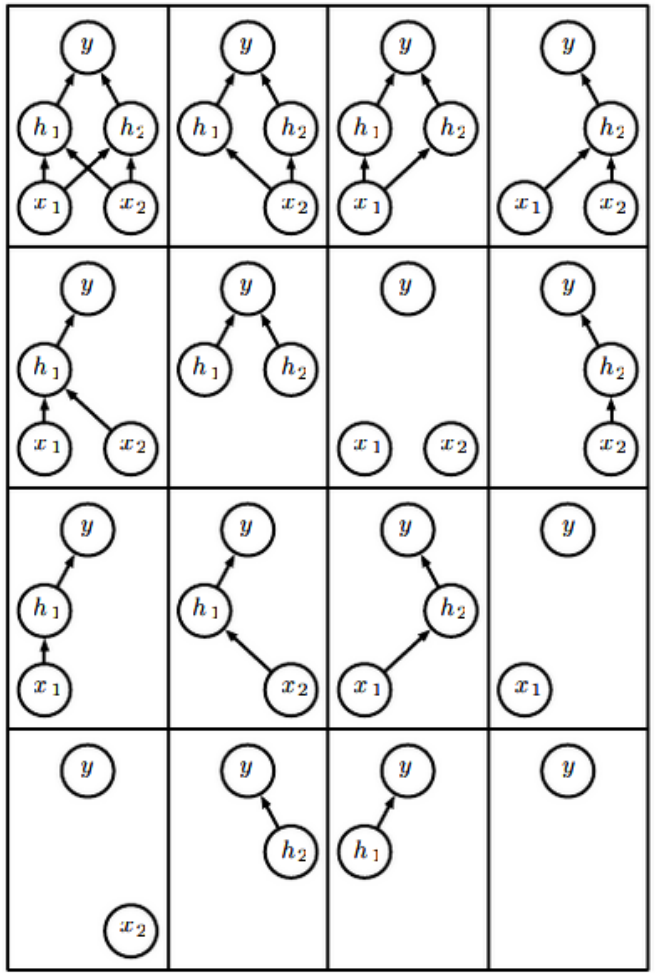
When a neuron is set to 0, it is like setting the weight for this neuron to be zero. Hence, the average $||W||^2$ is smaller, which has the same effect as regularization as L2 norm.

- Intuitions:
  - Regularization similar to L2 norm
  - Regularization similar to noise injection
  - Ensemble modeling, since training a single model that has the effect as aggregating multiple models generated by dropout some neurons out





Base network


Ensemble of subnetworks

If adding dropout after the hidden layer h, then there are 2^3 different dropout cases, 2^3 different networks for assemble.

Source from:
http://www.deeplearningbook.org/contents/regularization.html

Advantages of Drop Out

- Model-agnostic: works well with many feedforward networks, whereas other regularization strategies may restrict the model architecture
- Experimentally shown to be more efficient than other forms of regularization, e.g. weight decay and other norm constraints
- Computationally "cheap", but comes at price of a bigger model (with more training) to compensate
- Dropout is training an ensemble of models that share hidden units → implies that hidden units must perform well regardless of other hidden units in the model. This encourages each hidden unit not only to be good features but also features good in many contexts.

Data Augmentation

- incorporate invariances into a neural network by training on more data that exhibits the variances desired
- sometimes, it's not so straightforward to collect this data so we can try to synthesize it by transforming the original dataset

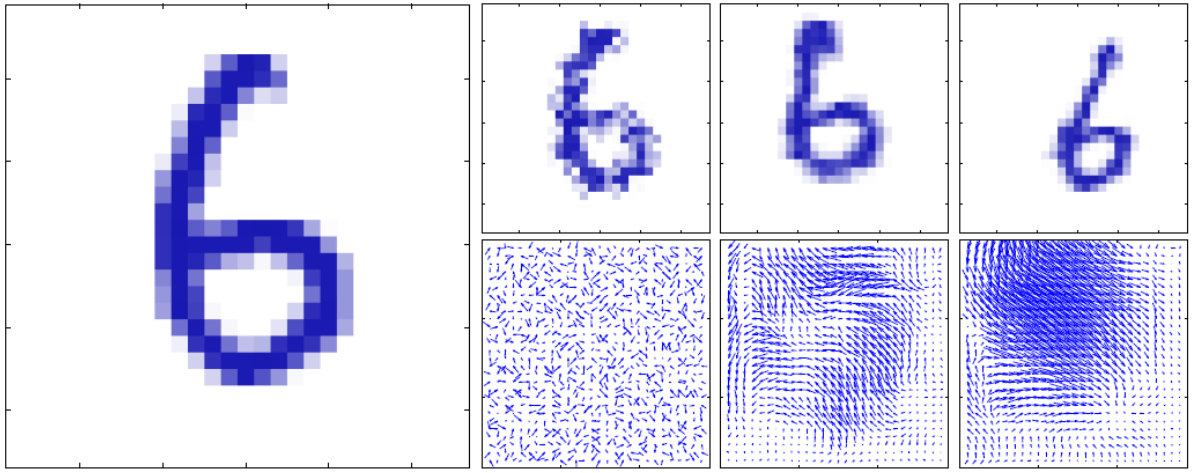


Figure: 5.14 from Bishop; three synthetic warps of a handwritten digit generated by sampling random displacement fields.

Data Augmentation

- Synthesizing data samples is highly application dependent
- For visual recognition problems, data augmentation works well since images can show variations (some of) which can be simulated.
  - translating training images a few pixels per direction often improves generalization.
  - other operations include rotating or scaling the image
    - But take care not to apply transformations that would change the correct class.

In a character recognition task, what transformations are not acceptable?

Injecting Noise

- For many classification and regression tasks, we want to solve the task despite small random noise added to the input. Neural networks are typically not very robust to noise unless explicit measures are taken.
- Try to "simulate" the noisy scenarios which may be encountered:
  - Inputs: train with noise-injected inputs (added randomly to clean input data)
  - Hidden units: Noise injection to inputs at hidden layers; can be considered data augmentation at multiple levels of abstraction.
- Noise can also be added to the weights;
  - primarily used in RNNs
  - encourages stability for learned function and learning parameters where small perturbations of weights have little influence on the learned model / outputs.

Batch Normalization or BN

- Proposed by Ioffe & Szegedy
- Many experiments show BN can improve performance
- Reasons behind its effectiveness remain under discussion:
  - Reducing Internal Covariate Shift (ICS)
    - Explained by the authors
  - Tiny parameter initialization and changes in the distribution of the inputs of each layer affect the learning rate of the network
  - Not reducing ICS, but smoothing the loss function (Santurkar et al.)

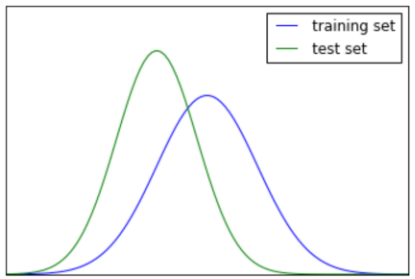
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

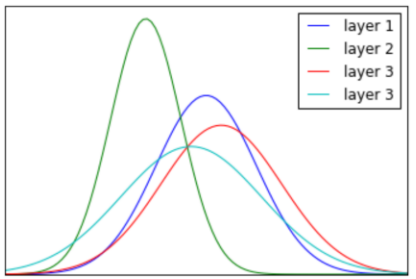
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

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



(a) Covariate shift



(b) Internal covariate shift

Batch Normalization or BN

- Computing the mean and variance of a layer
- Toy example: batch size (m) = 3
  - Small batch can't estimate the distribution
  - Here, the toy example just shows the idea of BN

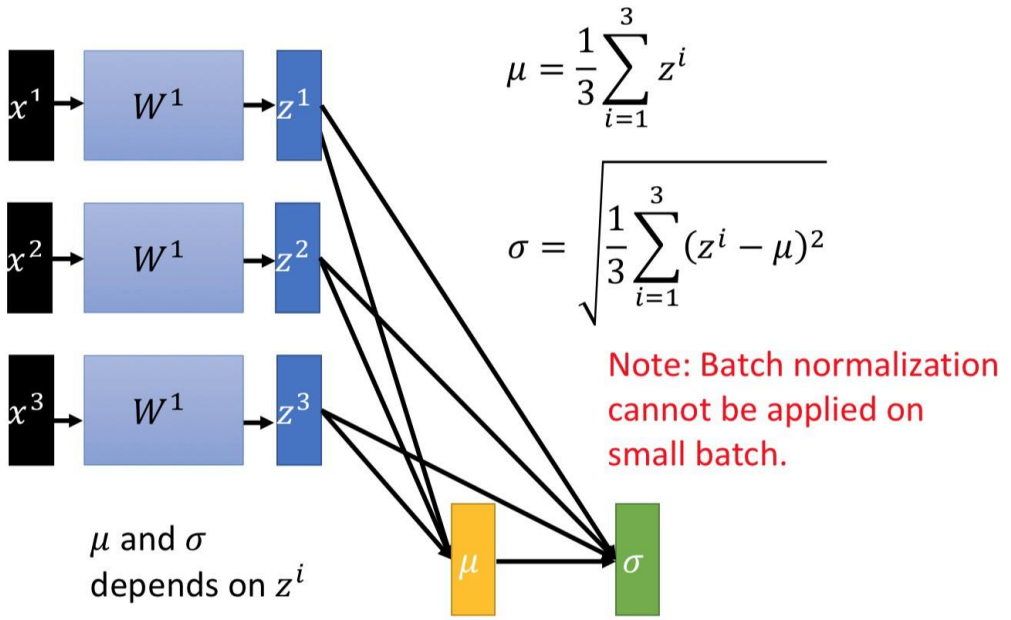


Figure credit: Hung-yi Lee

Batch Normalization or BN

- Normalize, scale and shit

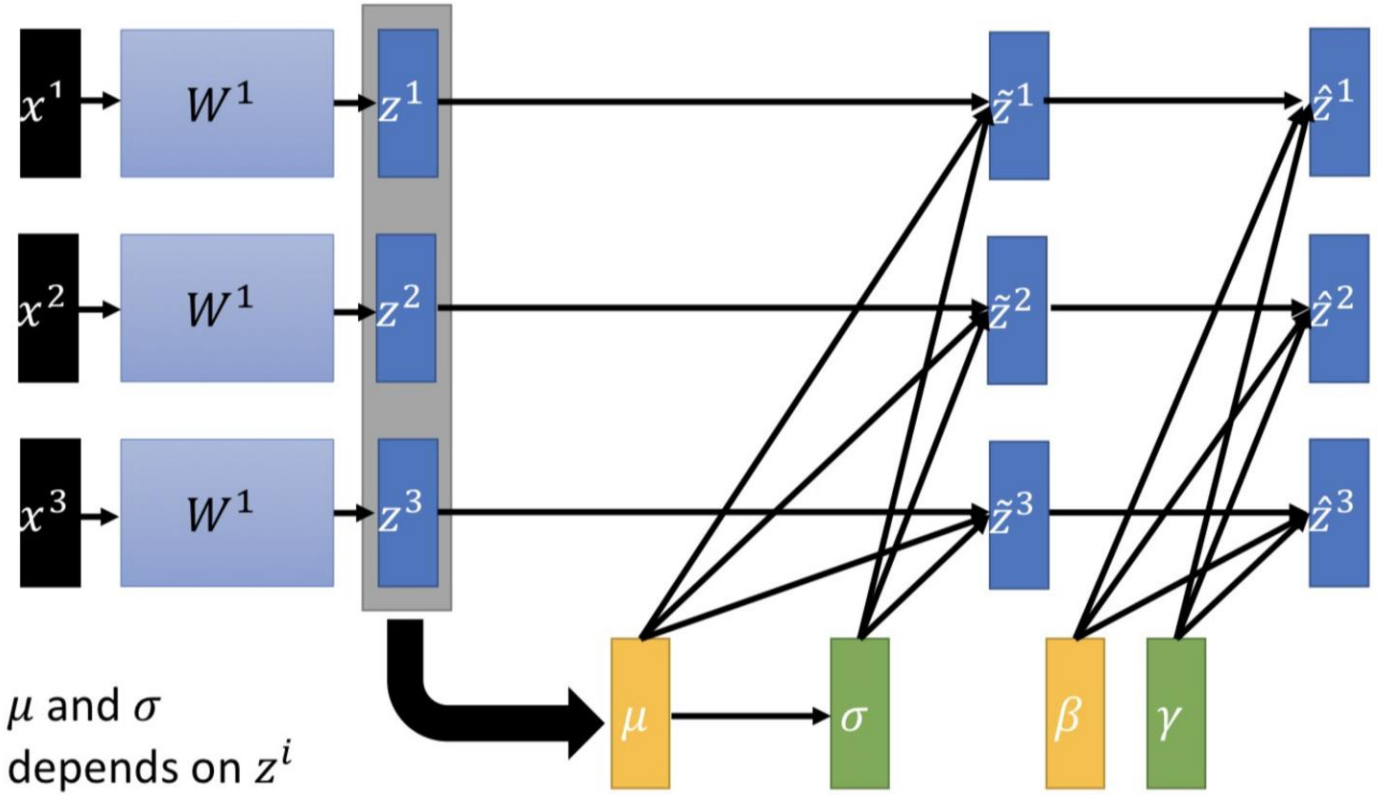


Figure credit: Hung-yi Lee

Modern CNNs

Image Classification task, rationale compared to MLP

Early Neural Networks and CNNs

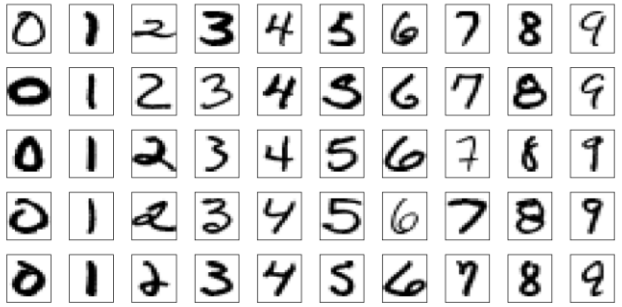


FIGURE 11.9. *Examples of training cases from ZIP code data. Each image is a 16 × 16 8-bit grayscale representation of a handwritten digit.*

Source: Hastie, Elements of Statistical Learning, Springer 2009

Source: https://www.huntington.com/Personal/checking/checks/how-to-write-a-check

- Character recognition task: classification of handwritten numerals.
- This problem captured the attention of the machine learning and neural network community for many years.
  - It is still a benchmark problem in the field.
- e.g. numbers on check, zip code

LeNet

LeNet-5 is the classically studied example for CNNs and deep learning.

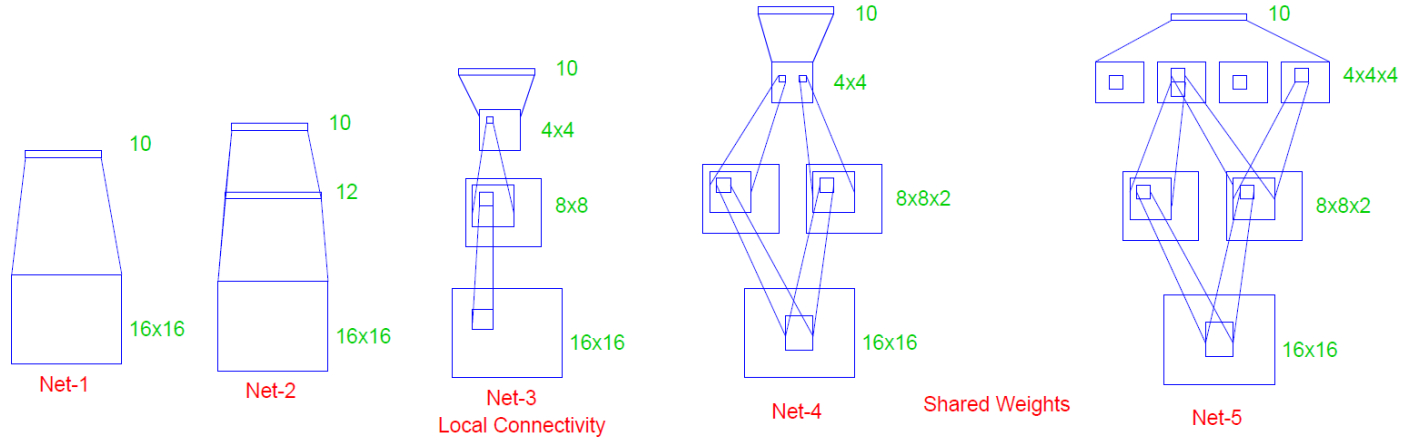


FIGURE 11.10. *Architecture of the five networks used in the ZIP code example.*

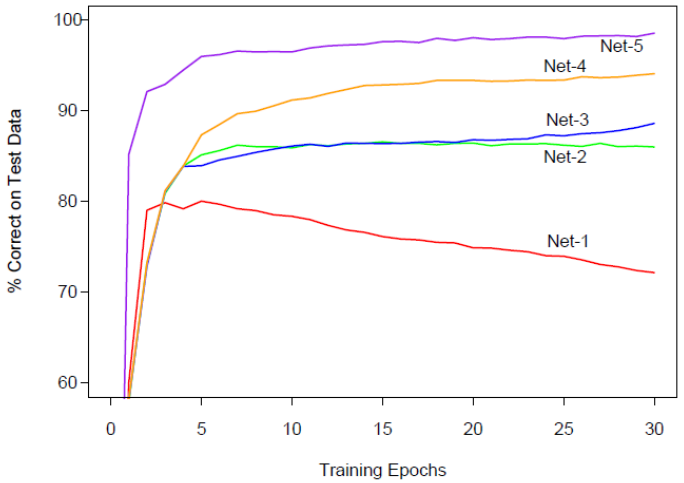


FIGURE 11.11. *Test performance curves, as a function of the number of training epochs, for the five networks of Table 11.1 applied to the ZIP code data. (Le Cun, 1989)*

Source: Hastie, Elements of Statistical Learning, Springer 2009

Five different networks were fit to the data:

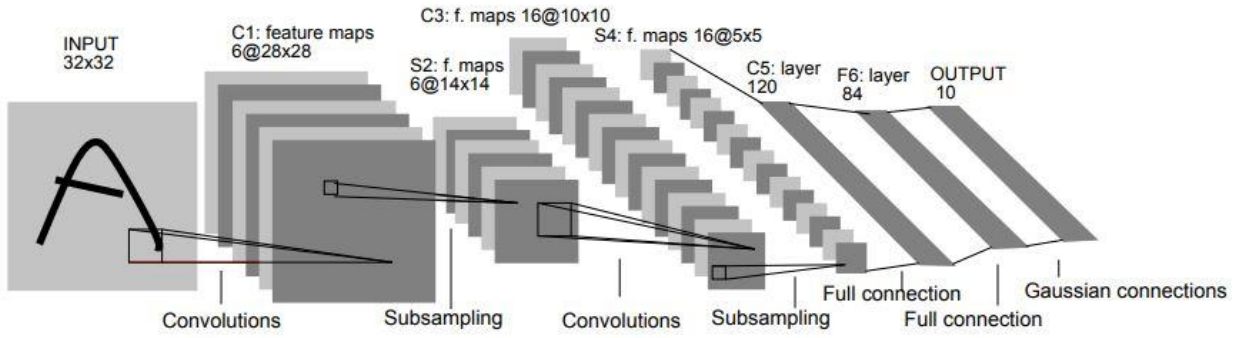
Net-1: No hidden layer, equivalent to multinomial logistic regression.

Net-2: One hidden layer, 12 hidden units fully connected.

Net-3: Two hidden layers locally connected.

Net-4: Two hidden layers, locally connected with weight sharing.

Net-5: Two hidden layers, locally connected, two levels of weight sharing.

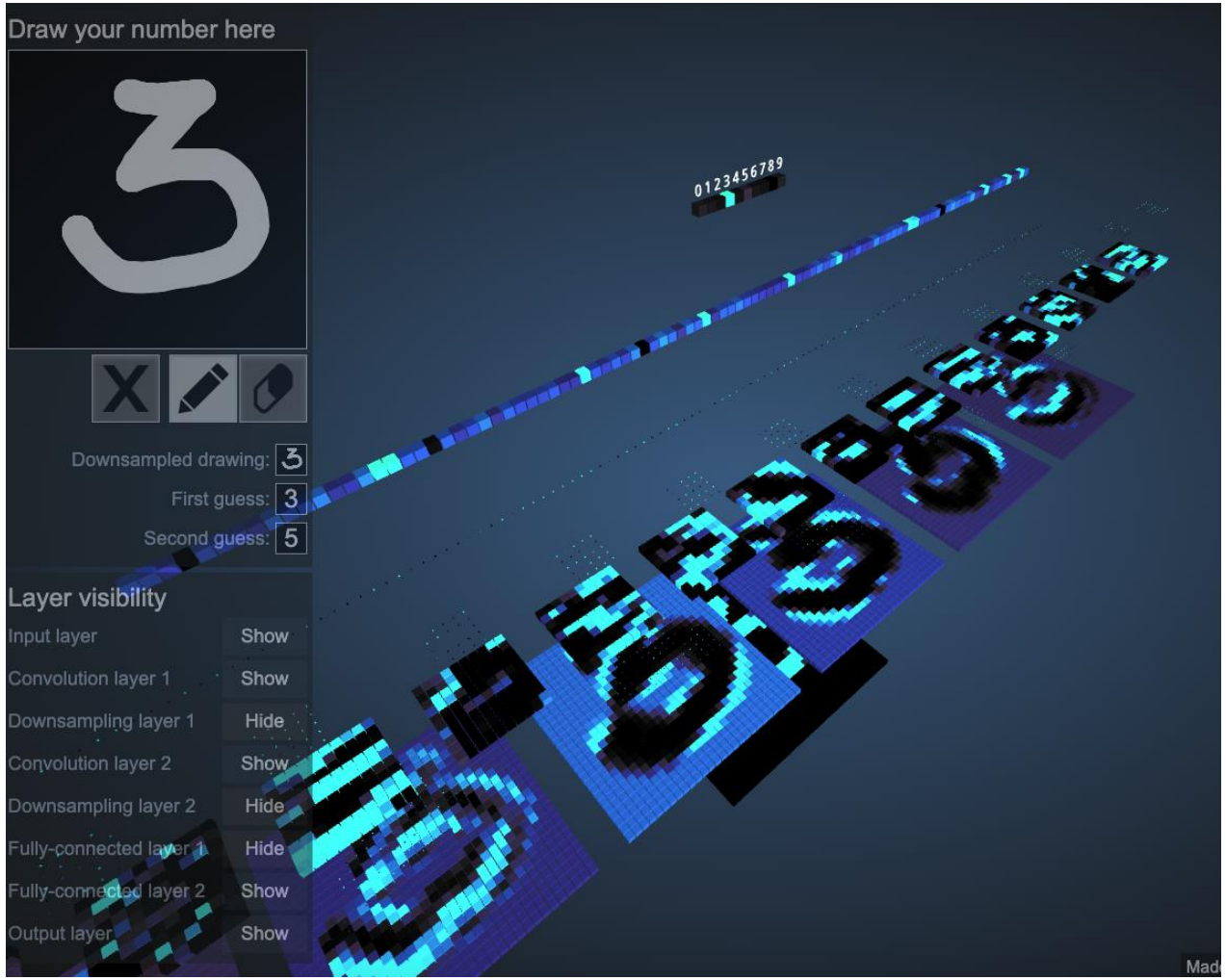


Original Image published in [LeCun et al., 1998]

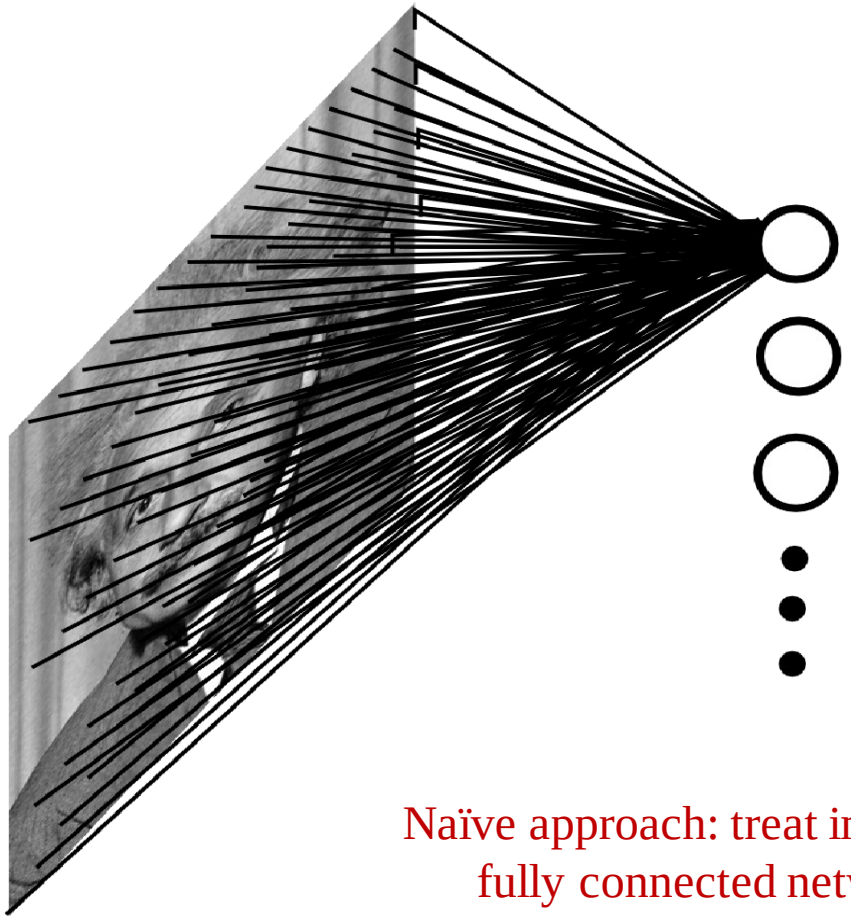
http://yann.lecun.com/exdb/lenet/index.html

LeNet Visualization

- https://www.youtube.com/watch?v=f0t-OCG79-U



Images as Inputs to Neural Networks



Naïve approach: treat image as a fully connected network.

Suppose we have a 200x200 image with 40k hidden units.

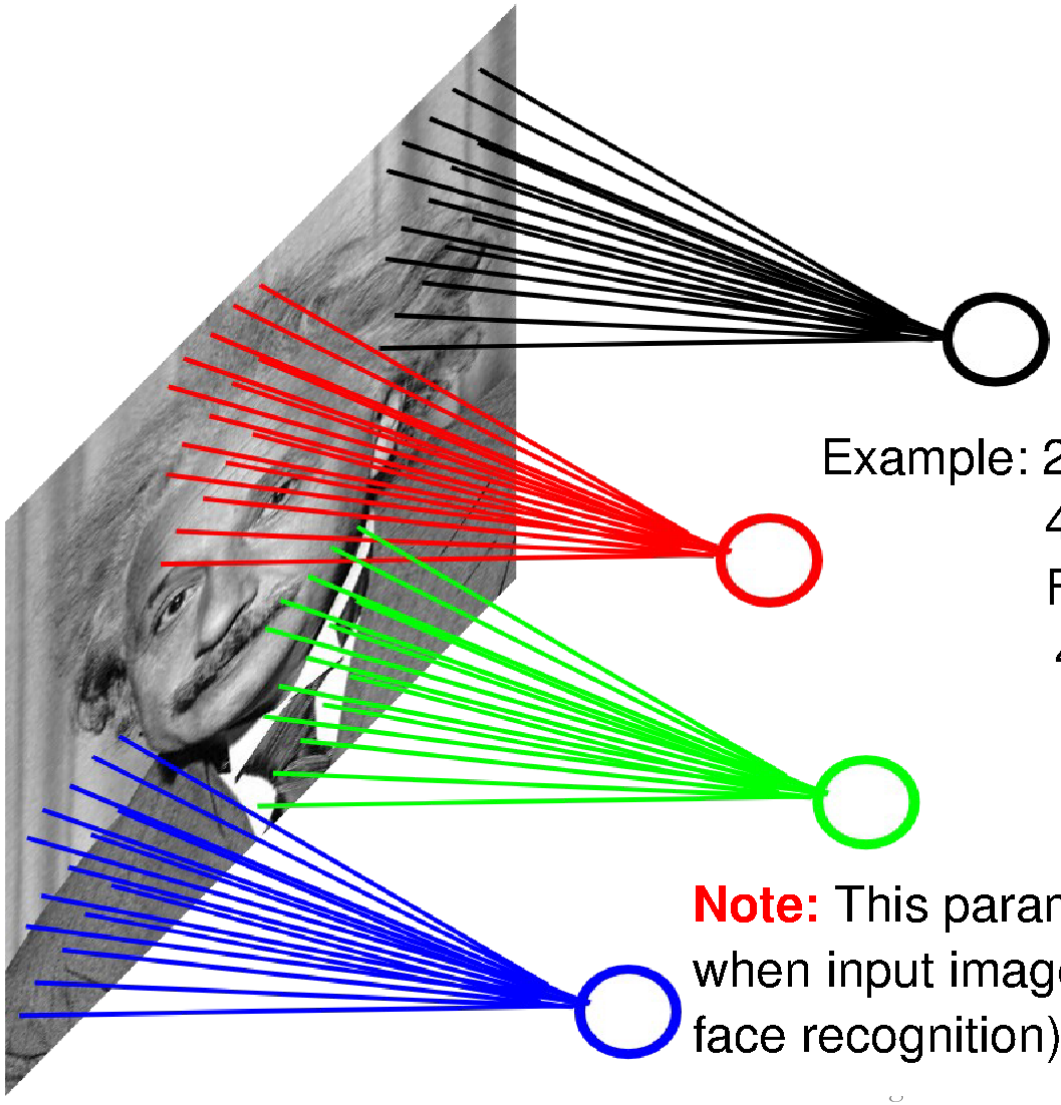
How many parameters for 1 hidden unit?

40k hidden units → ~2 billion params!

Fully connected hidden units ignores a key property of images: pixels near to each other are more correlated than pixels far apart.

Ranzato

Images as Inputs to Neural Networks



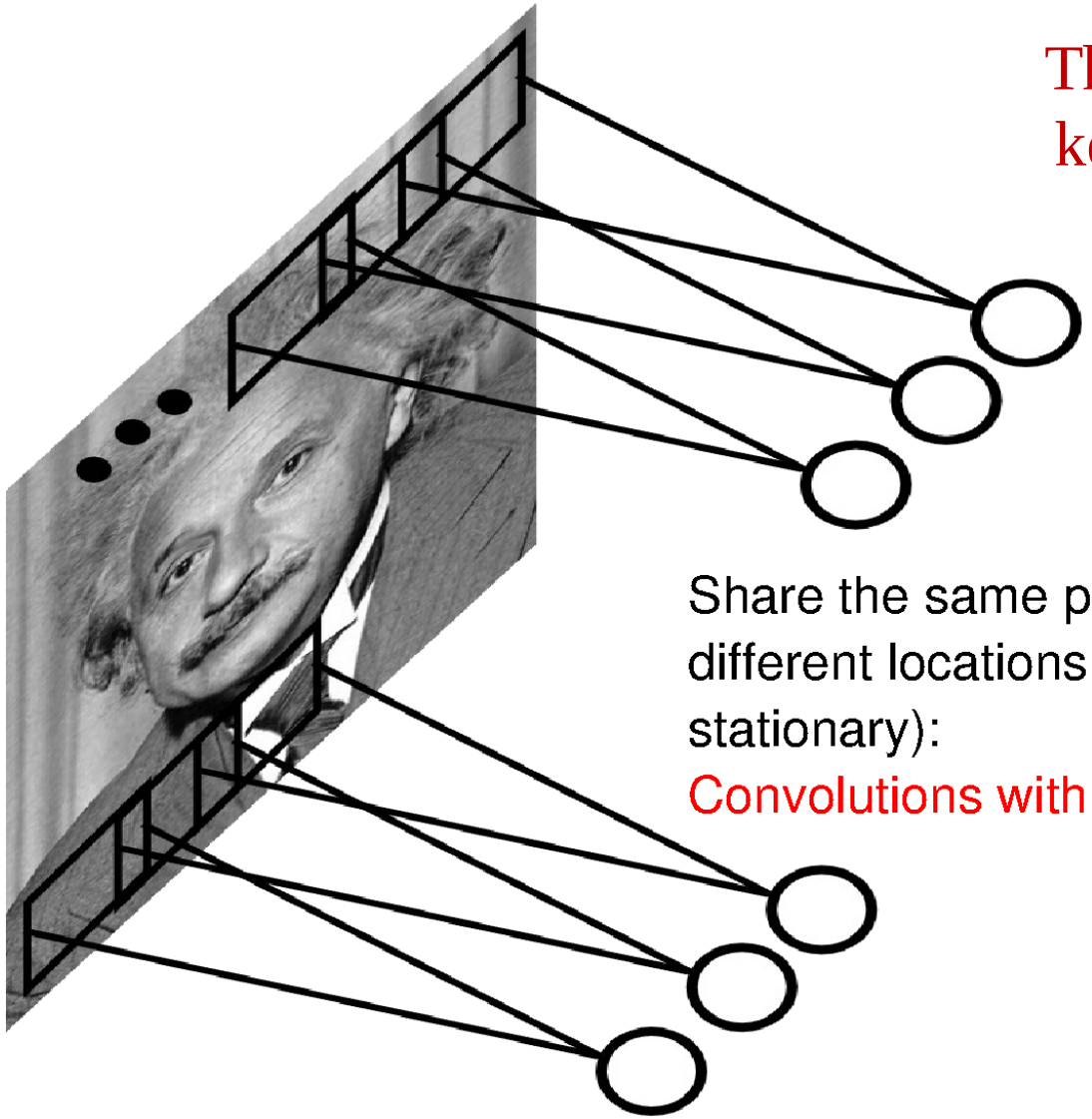
Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

This would not account for stationarity assumptions in images, i.e. statistics and properties are similar at different locations.

Ranzato

Convolutional Layer

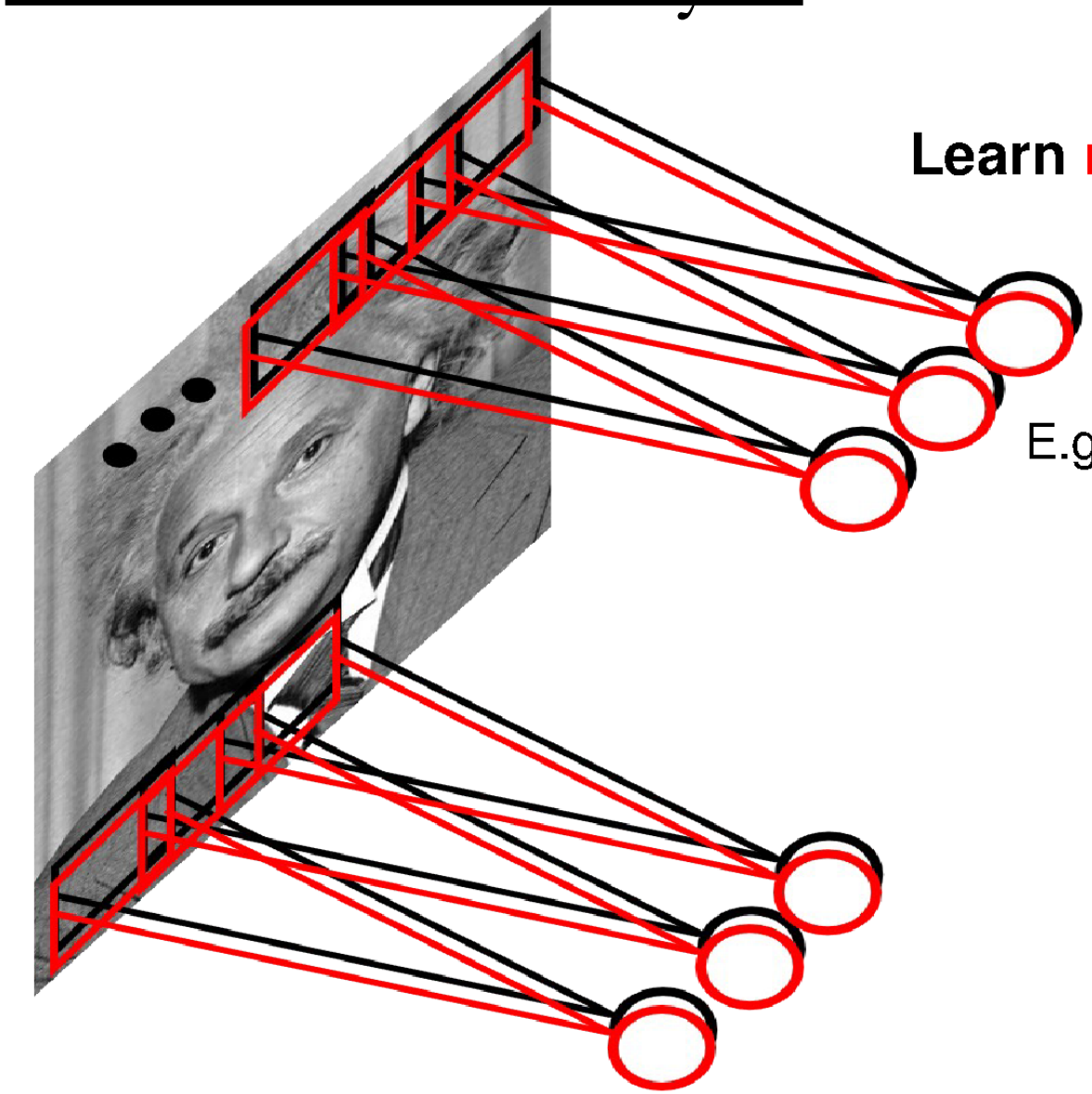


The convolution operation applies the same kernel to different locations in the image.

Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Ranzato

Convolution Layer



Learn multiple filters.

Filter = 'local' perceptron.
Also called *kernel.*

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Ranzato

Motivations

- Sparse interactions – *receptive fields*
  - Assume that in an image, we care about 'local neighborhoods' only for a given neural network layer.
  - Composition of layers will expand local -> global.

- Parameter sharing
  - Use same weights for more than one perceptron in the neural network.
  - Leads to *equivariant representation*
    - If input changes (e.g., translates), then output changes similarly

A standard neural net applied to images:

- scales quadratically with the size of the input
- does not leverage stationarity

Solution:

- connect each hidden unit to a small patch of the input
- share the weight across space

This is called: **convolutional layer.**

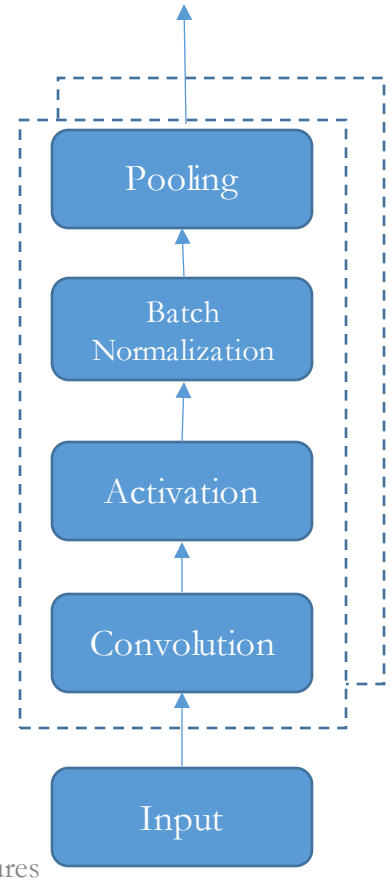
A network with convolutional layers is called **convolutional network.**

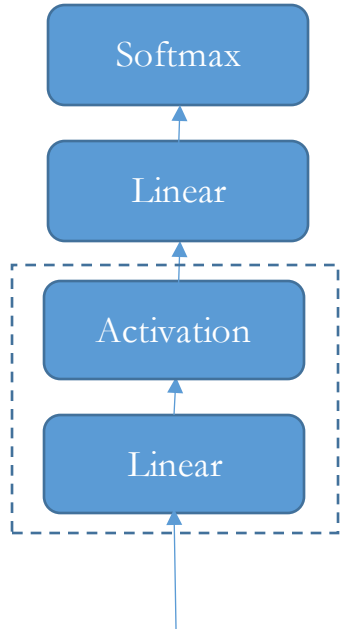
- add multiple filters per layer

LeCun et al. “Gradient-based learning applied to document recognition” IEEE 1998

General Architecture

- ConvBlock (module)
  - Convolution
  - Activation
  - Batch Normalization
  - Pooling
- Classification
  - Linear + Activation + Softmax
- Regression
  - Linear
- Object detection
- Image segmentation
- …

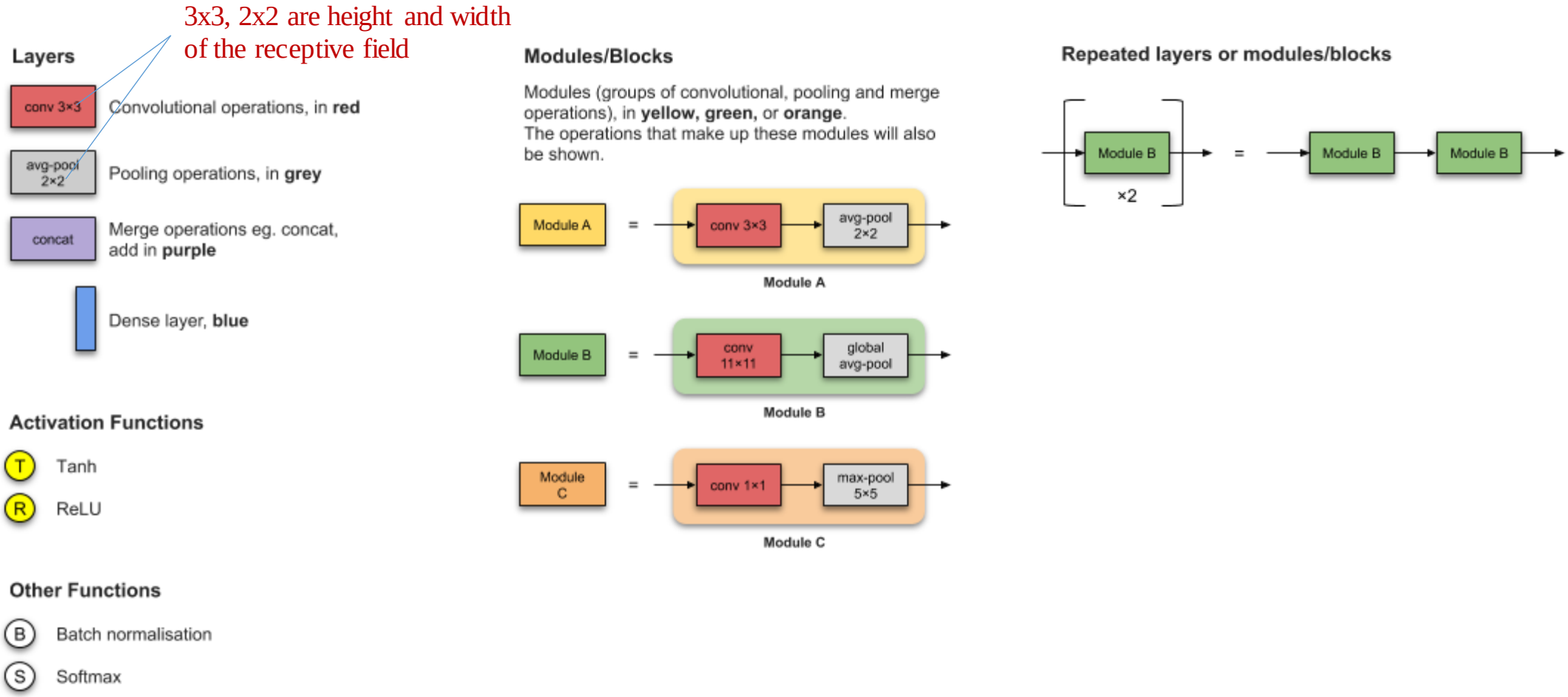




How many blocks?
What is the configuration of each convolution layer?

Slide credit: Wang Wei

Visualization of ConvNet Architectures

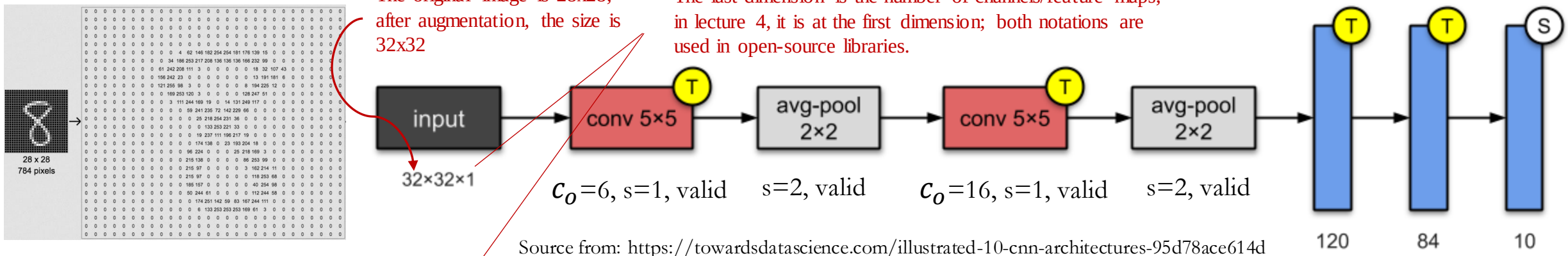


Source from: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

Slide credit: Wang Wei

LeNet-5 (1990s)

pooling function: sigmoid(a*average(x)+b)



Source from: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

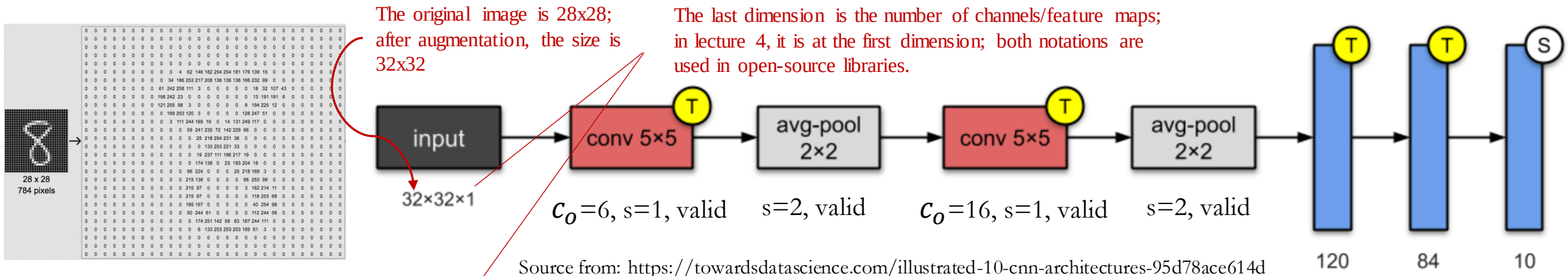
Layer	Output Shape	Parameters
Conv1	(28, 28, 6)	
Pool1	(14,14, 6)	
Conv2	(10, 10, 16)	
Pool2	(5, 5, 16)	
FC1	(120,)	
FC2	(84,)	
FC3	(10,)	

3-Minute Quiz:
How many parameters?
Fill in the table.

LeCun, Yann, et al (1998). Gradient-based learning applied to document recognition

Slide credit: Wang Wei

LeNet-5 (1990s)



Source from: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

Layer	Output Shape	Parameters
Conv1	(28, 28, 6)	(5x5+1)x6=156
Pool1	(14,14, 6)	6x(1+1)=12
Conv2	(10, 10, 16)	(5x5x3+1)x6+(5x5x4+1)x9+(5x5x6+1)=1516
Pool2	(5, 5, 16)	16x2=32
FC1	(120,)	120x(5x5x16+1)
FC2	(84,)	(120+1)x84
FC3	(10,)	10×84+10

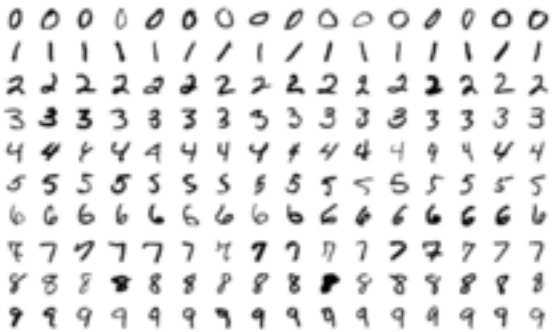
LeCun, Yann, et al (1998). Gradient-based learning applied to document recognition

Slide credit: Wang Wei

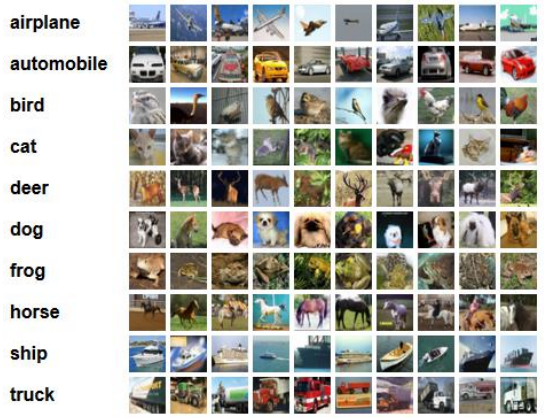
Multi-class Single Label Image Classification

- Given an image, predict its class from a set of candidate classes.
- Dataset

Dataset	Year	# classes	# images	Resolution
MNIST	1998	10	70, 000	28x28
CIFAR10	2009	10	60, 000	32x32
ImageNet	2009	1000	1.2 million	high



https://en.wikipedia.org/wiki/MNIST_database



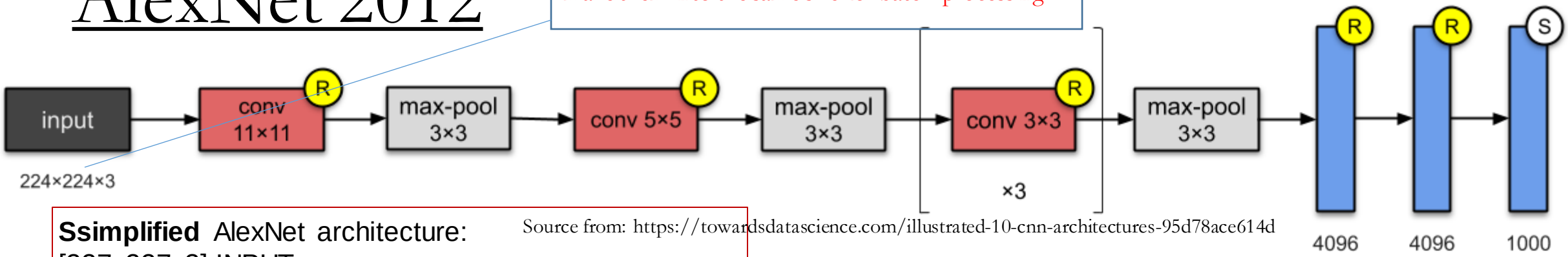
https://www.cs.toronto.edu/~kriz/cifar.html

https://medium.com/syncedreview/sensetime-trains-imagenet-alexnet-in-record-1-5-minutes-e944ab049b2c

Slide credit: Wang Wei

AlexNet 2012

The original images are of different size. Some pre-processing/augmentation operations are applied to make them into the same size for batch-processing.



Source from: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

Ssimplified AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: **96** 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: **256** 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: **384** 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: **384** 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: **256** 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: **4096** neurons
[4096] FC7: **4096** neurons
[1000] FC8: **1000** neurons (class scores)

BIG & DEEP

60M parameters & 8 Conv layers

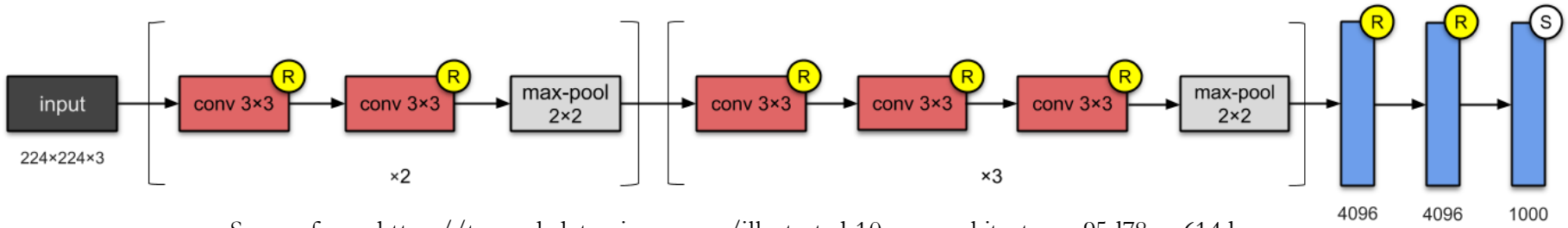
http://ethereon.github.io/netscope/#/preset/alexnet

From cs231n

Alex, et al. Imagenet classification with deep convolutional neural networks.

Slide credit: Wang Wei

VGG 2014



Source from: https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d

2nd of 2014 classification task

Deeper structure (16, 19 convolution layers)

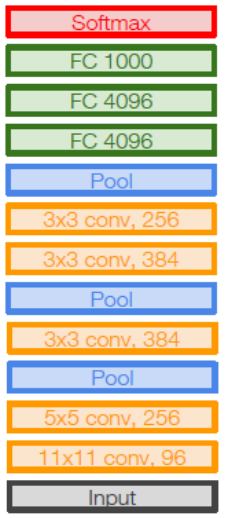
More model parameters compared with AlexNet

130+ Million VS 60 Million

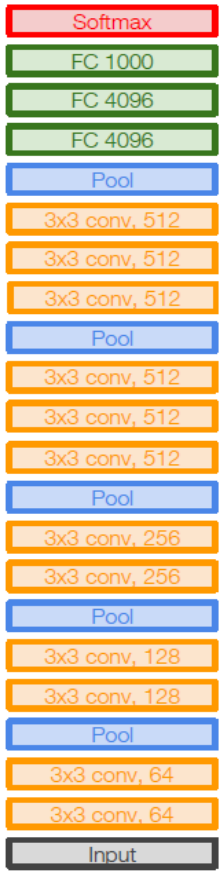
More non-linear transformation; Large capacity

Unified kernel size

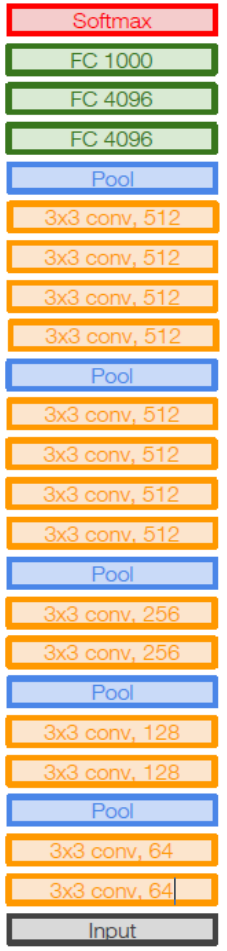
Consecutive convolution



AlexNet



VGG16



VGG19

From cs231n

Very Deep Convolutional Networks for Large-Scale Image Recognition. Karen Simonyan, Andrew Zisserman.

http://ethereon.github.io/netscope/#/preset/vgg-16

Slide credit: Wang Wei

Summary

- Regularization techniques are designed to improve the generalization capabilities of deep neural networks and prevent overfitting
- Approaches of regularization include:
  - Parameter norm penalties
  - Reducing model capacity: early stopping, weight sharing, drop-out
  - Adjusting data: data augmentation, noise injection, batch normalization
- Rationale of applying convolution (for images) is based on two fundamental assumptions: local receptive fields + spatial invariance
  - significantly reduces the number of parameters needed vs. naïve MLP
- Neocognitron and LeNet variants lead up to modern CNNS such as LeNet5, AlexNet, VGG, etc.