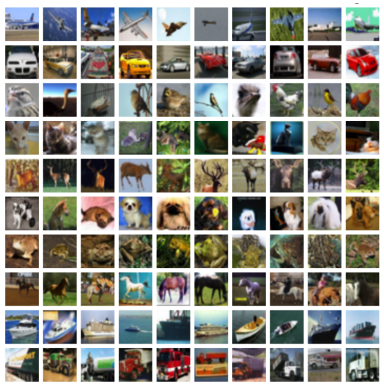
Convolutional Neural Network

Shadi Albarqouni
Postdoctoral Researcher
shadi.albarqouni@tum.de

Outline

Motivation



Training Set

Linearly separable feature Space?

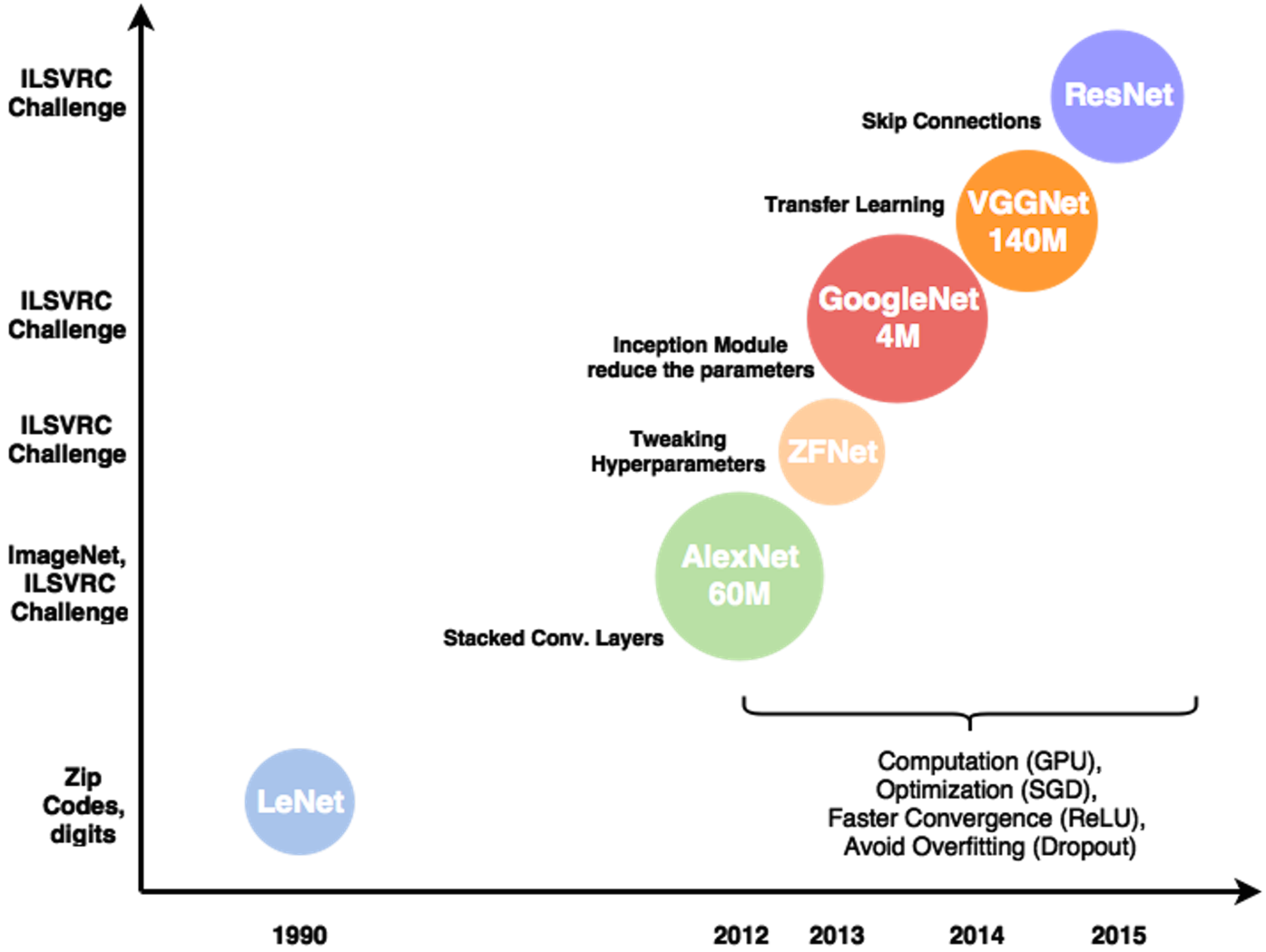
Object Recognition: Pipeline

Hierarchical and Non-linear feature representation (stacked layers) learned jointly with the classifier

Image

Since 1950's

ConvNets Successes

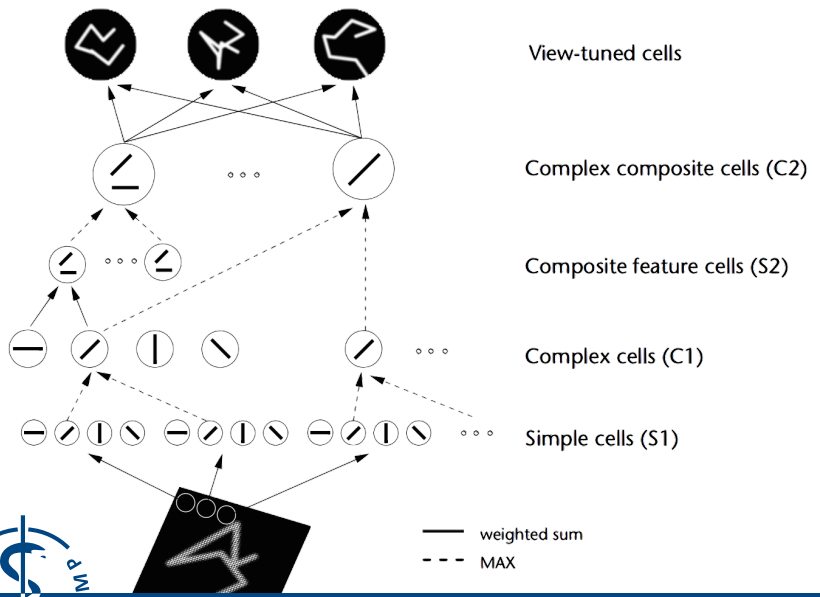


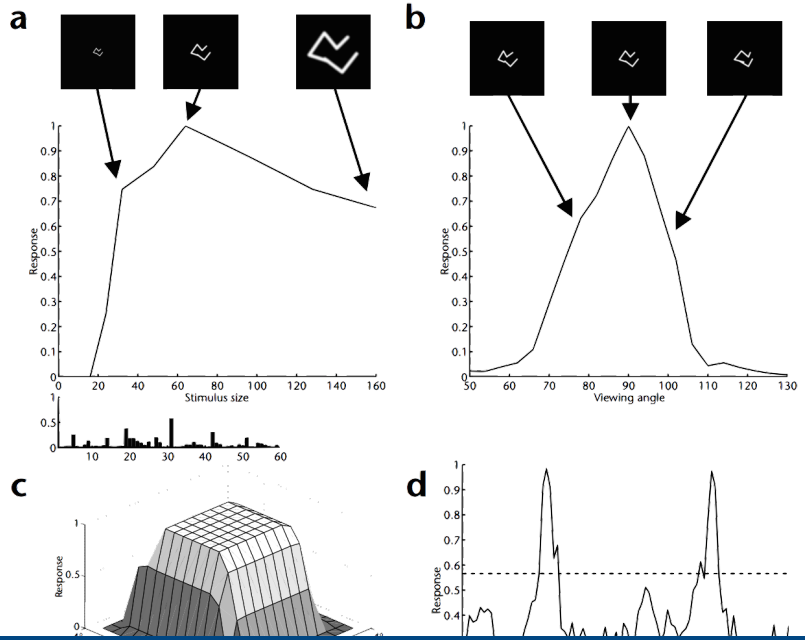
What is ConvNet?

Definition (ConvNet)

It is a member of Deep Learning family. It is similar to Artificial Neural Networks (ANN), however, the connectivity pattern between its neuron is inspired by the hierarchical organization of animal visual cortex[a] .

[a] **riesenhuber1999hierarchical**.





What's wrong with ANN? (1)

- Hard to Train (over-fitting)
- Careful Initialization
- Huge number of parameters

Key ideas of ConvNets

- image statistics (shared weights)
- Low-level features supposed to be local (local connectivity)
- High-level features supposed to be coarser (subsampling)

"Convolution + Activation + Pooling = Architecture"

Network Architecture (1)

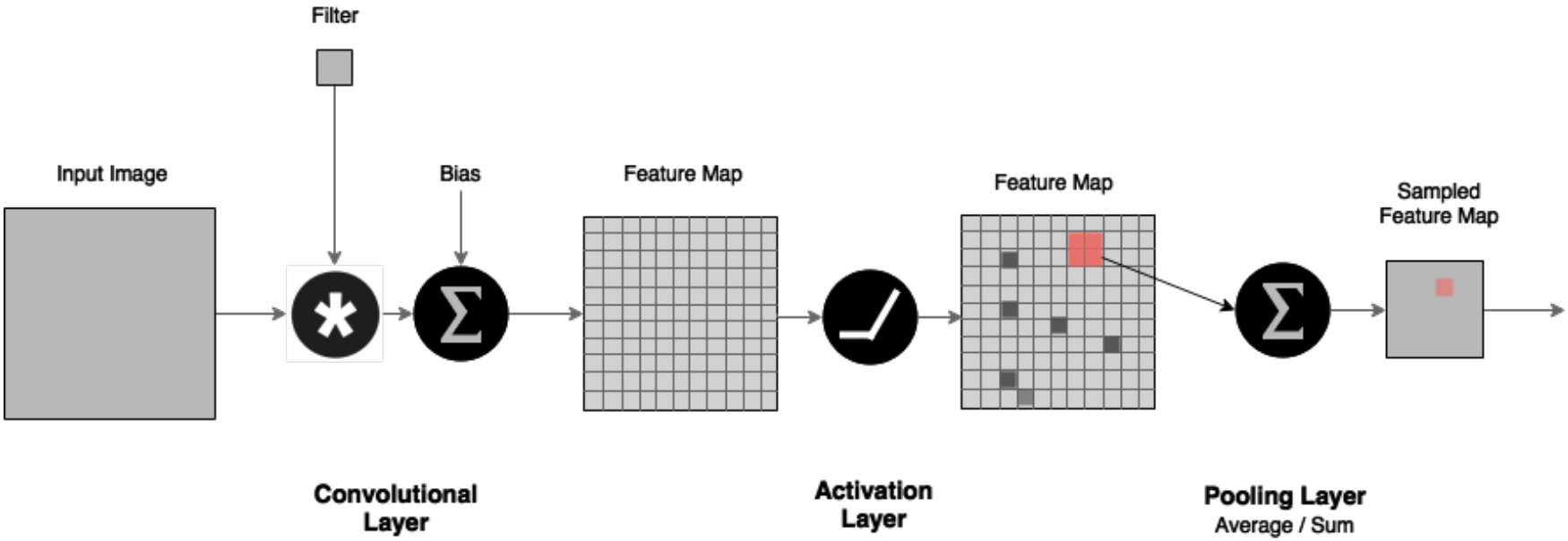
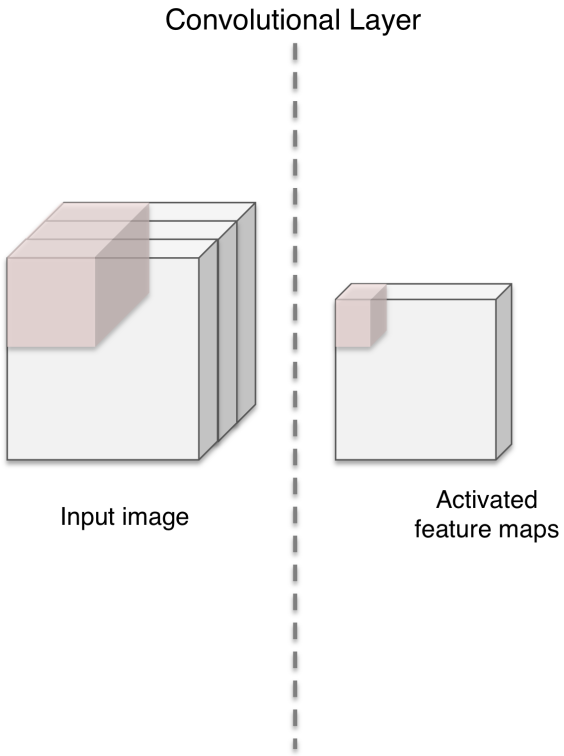


Figure: Symbolic Architecture

Define: receptive field, stride, depth and width of the network.

Network Architecture (2)



Notation

We follow the notations appeared in MatConvNet[1],

- X is the input data, $X = \{x_1, x_2, ..., x_N\} \in \mathbb{R}^{H \times W \times D \times N}$.
- N is the number of input instances/samples.
- H is the height of an image $x_{i \in N}$.
- W is the width of an image $x_{i \in N}$.
- D is the channels/depth of an image/volume $x_{i \in N}$.
- Y is the desired output, $Y = \{y_1, y_2, ..., y_N\} \in \mathbb{R}^{c \times N}$

Objective

Build a model f that for a given input x can predict the output $\hat{y}$:

$$\hat{y} = f(x; \omega),$$

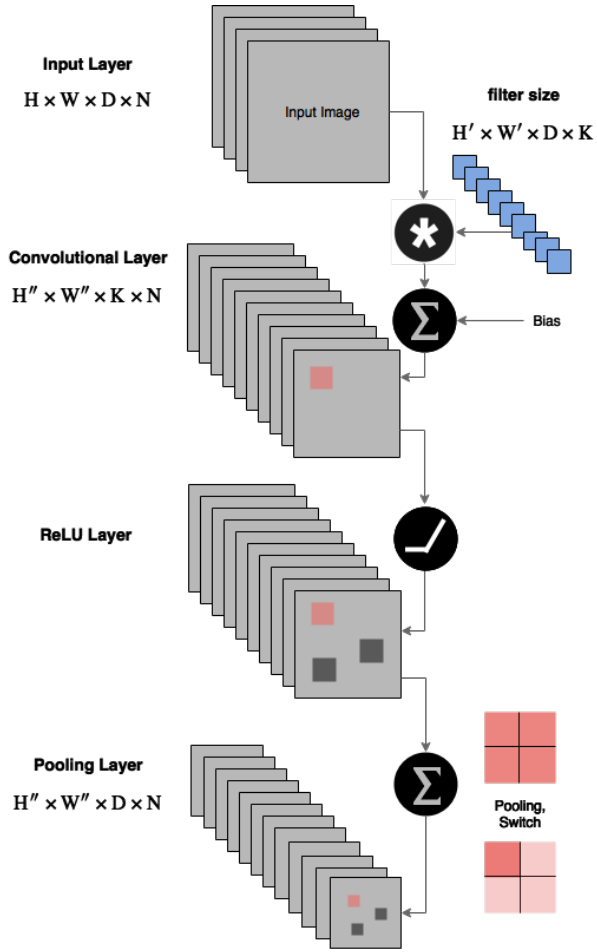
where ω is the model parameter.

[1] **vedaldi2015matconvnet**.

CNN Layers

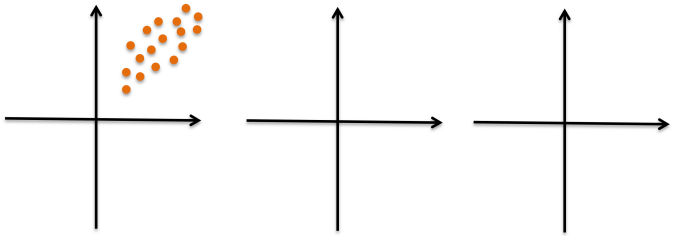
A CNN Network can be obtained by cascading several layers in a directed acyclic graph (DAG).

- Input Layer
- Convolutional Layer
- Activation Layer
- Pooling Layer
- Fully Connected Layer
- Dropout Layer
- Output Layer



Input Layer ($H \times W \times D \times N$) (1)

- Data Preprocessing (Mean subtraction, PCA/Whitening)
- Data Augmentation: geometric transformation; rotation and translation, color transformation: illumination, staining ...etc, adding noise.
- Splitting the dataset (training, validation and testing)
- Batch size



Input Layer ($H \times W \times D \times N$)

- 2D inputs
  - Gray ($D = 1$)
  - RGB ($D = 3$)[2]
- 2.5D inputs
  - Gray ($D = 3$)[3]
  - RGB-D ($D = 4$)[4]
- 3D inputs
  - Gray ($D = \mathrm{d}$)[5]

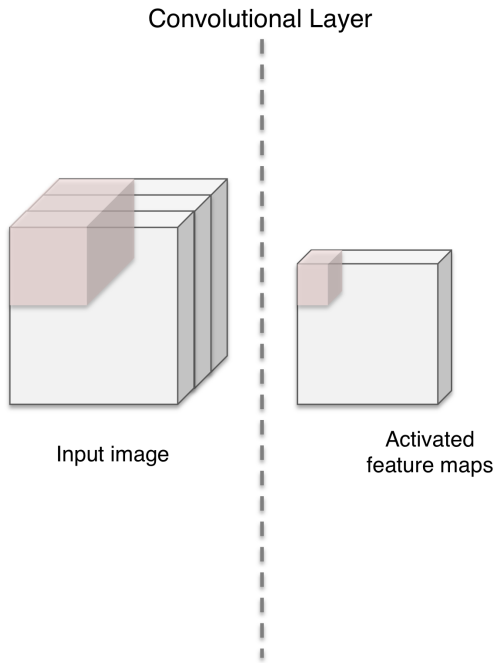
[2] **eigen2014predicting**.

[3] **roth2014new**.

[4] **gupta2014learning**.

[5] **kamnitsas2015multi**.

Convolutional Layer ($H'' \times W'' \times K \times N$) (1)

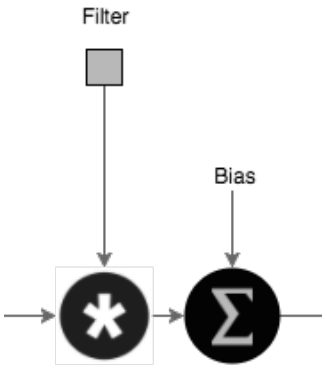


Convolutional Layer ($H'' \times W'' \times K \times N$) (2)

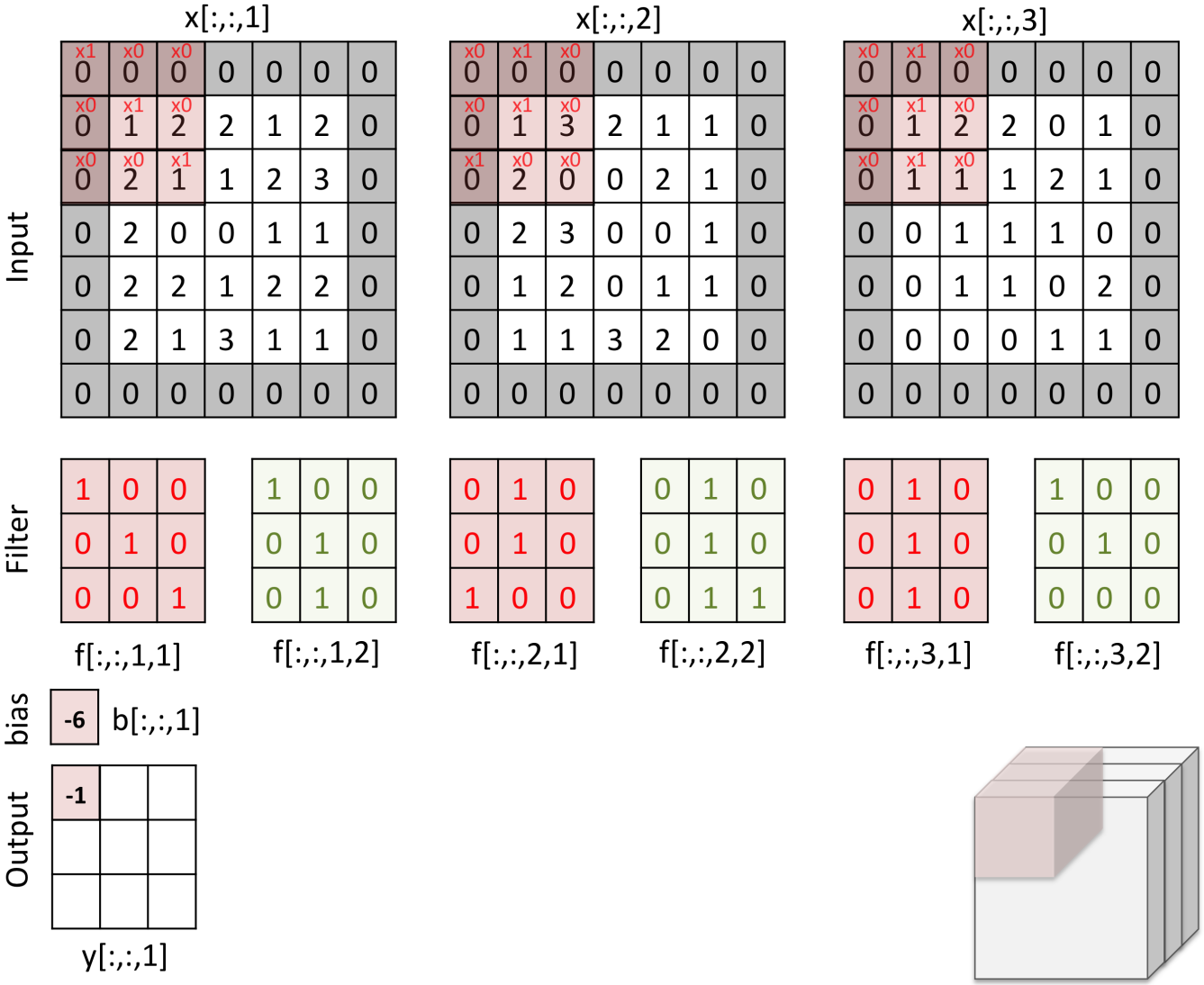
It computes the convolution of input image x with a filter f as follows

$$y_{i,j,k} = b_{i,j,k} + \sum_{h=1}^{H'} \sum_{w=1}^{W'} \sum_{d=1}^{D} f_{h,w,d,k} \cdot x_{i+h,j+w,d},$$

- input $x \in \mathbb{R}^{H \times W \times D}$
- filters $f \in \mathbb{R}^{H' \times W' \times D \times K}$
- biases $b \in \mathbb{R}^{H'' \times W'' \times K}$
- output $y \in \mathbb{R}^{H'' \times W'' \times K}$
- stride $S_{W,H}$ and padding $P_{W,H}$,



Convolutional Layer ($H'' \times W'' \times K \times N$) (3)

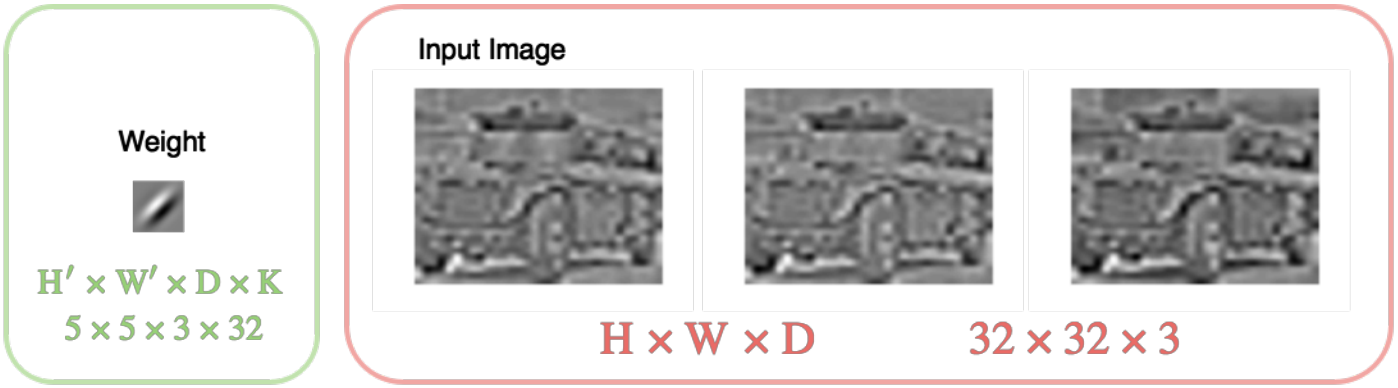


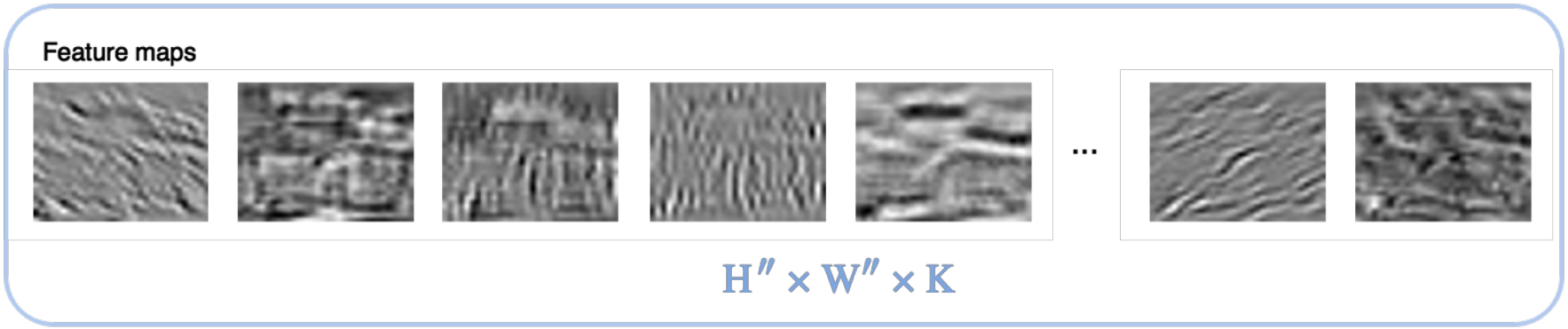
Convolutional Layer ($H'' \times W'' \times K \times N$) (4)

Example: CIFAR-10 (Convolution, $5 \times 5 \times 3 \times 32$)

Keywords: Translation Invariance, few parameters, local consistency

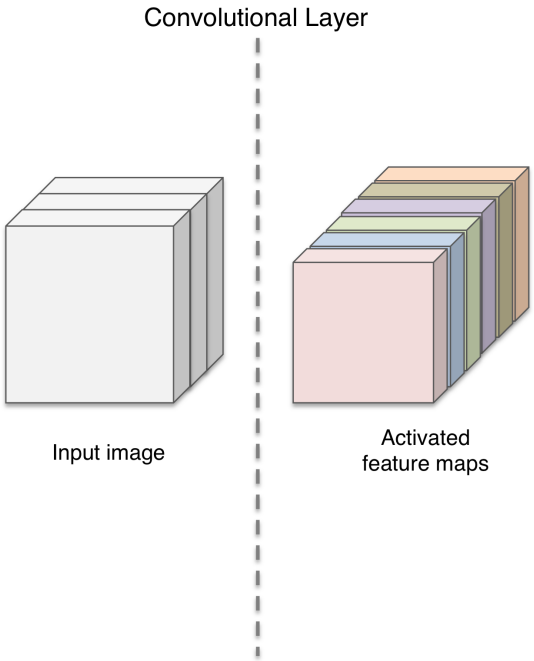
Convolutional Layer ($H'' \times W'' \times K \times N$) (5)





$$W'' = 1 + \frac{W - W' + (P_L + P_R)}{S_W}, \qquad H'' = 1 + \frac{H - H' + (P_U + P_D)}{S_H},$$

Activation Layer ($H \times W \times D \times N$) (1)

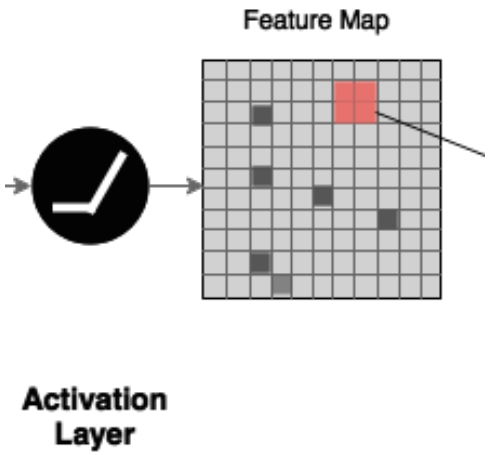


Activation Layer ($H \times W \times D \times N$) (2)

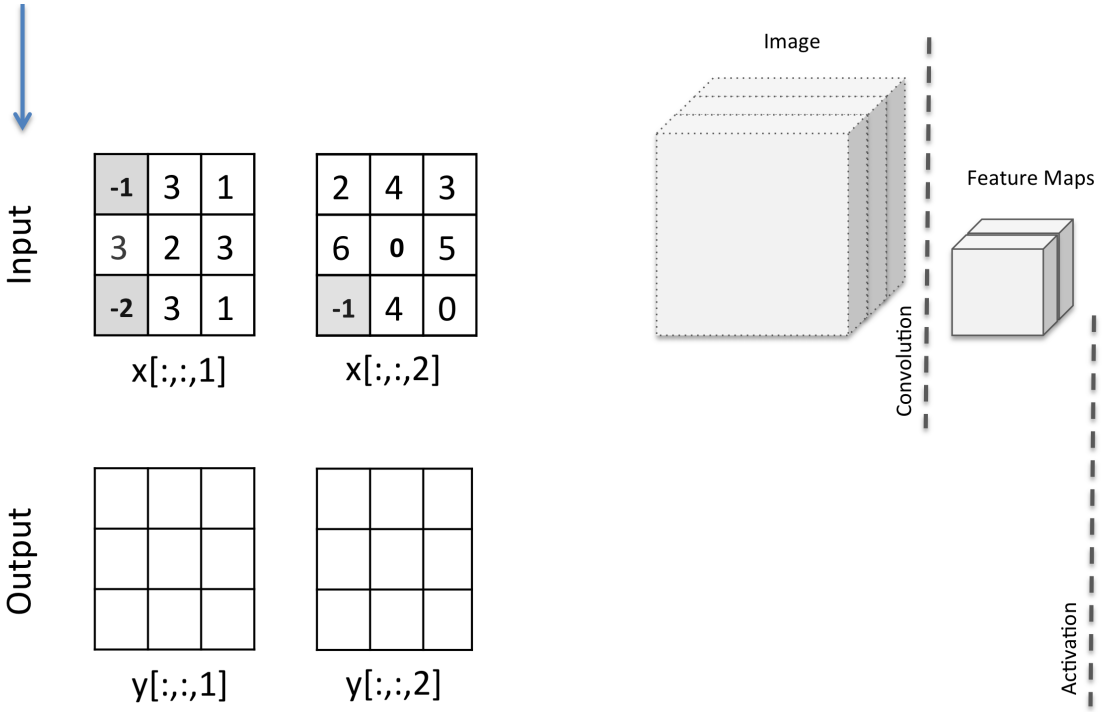
It computes the **Rectified Linear Unit (ReLU)** of each feature channel x as follows

$$y_{i,j,d} = \max\{0, x_{i,j,d}\},$$

- input $x \in \mathbb{R}^{H \times W \times D}$
- output $y \in \mathbb{R}^{H \times W \times D}$



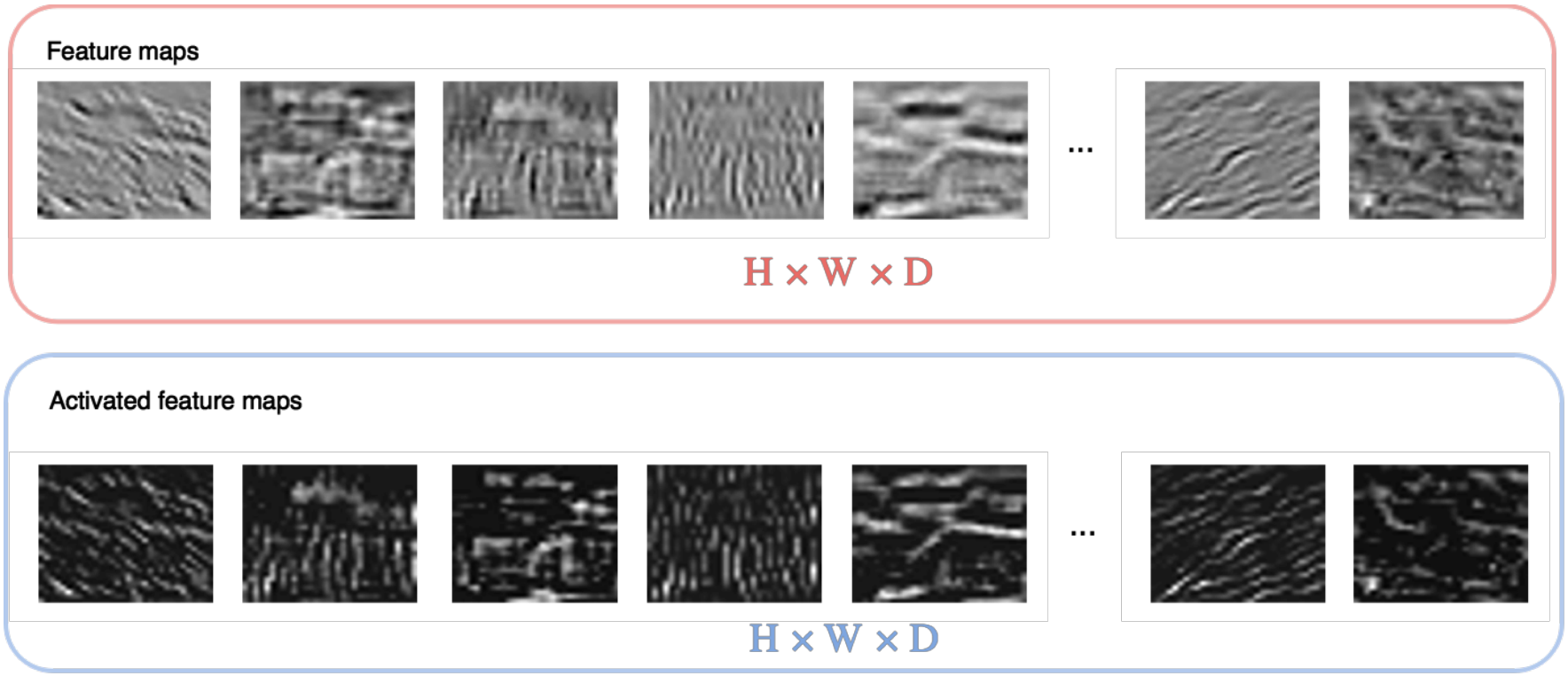
Activation Layer ($H \times W \times D \times N$) (3)



Activation Layer ($H \times W \times D \times N$) (4)

Example: CIFAR-10 (ReLU)

Keywords: Simplifies Back-propagation, Makes Learning faster.



Activation Layer ($H \times W \times D \times N$) (5)

What about other activation functions? Any potential drawbacks?

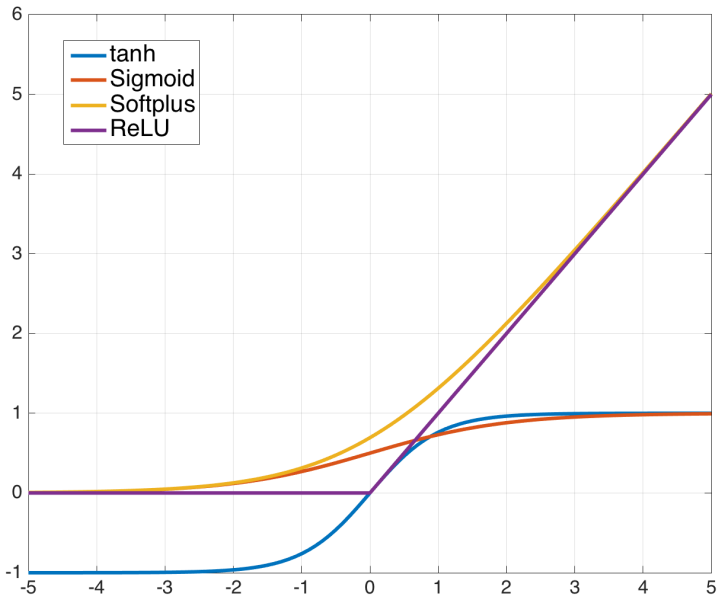


Figure: Activation functions

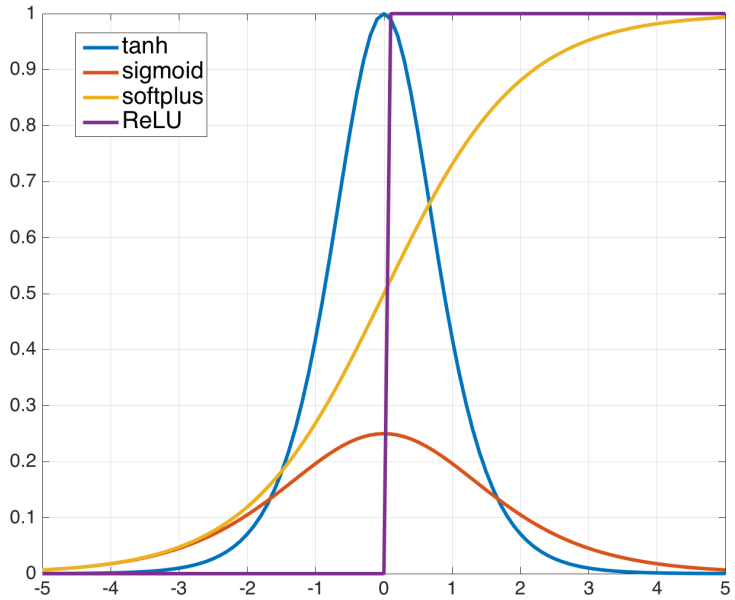
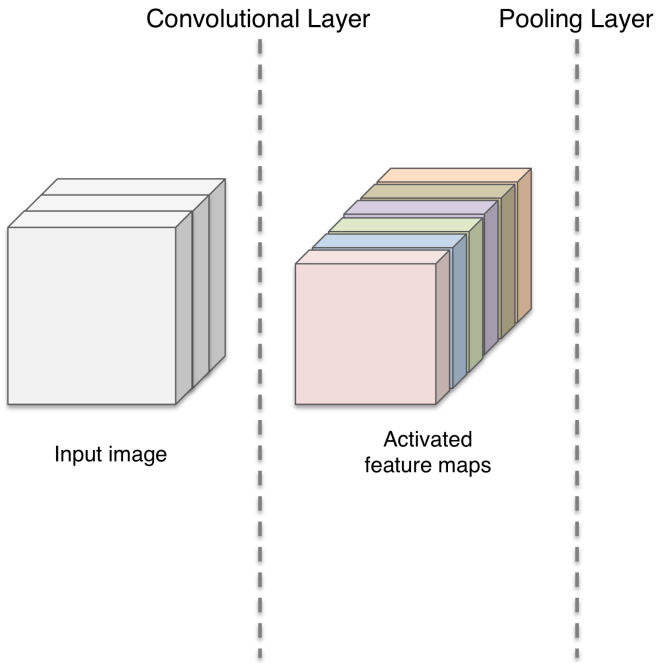


Figure: Activation derivatives

Pooling Layer ($H'' \times W'' \times D \times N$) (1)



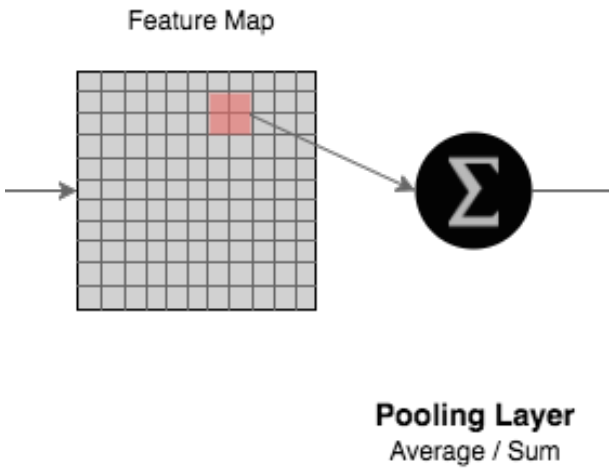
Pooling Layer ($H'' \times W'' \times D \times N$) (2)

It computes the maximum or average response of each feature channel x within a 2D patch p as follows

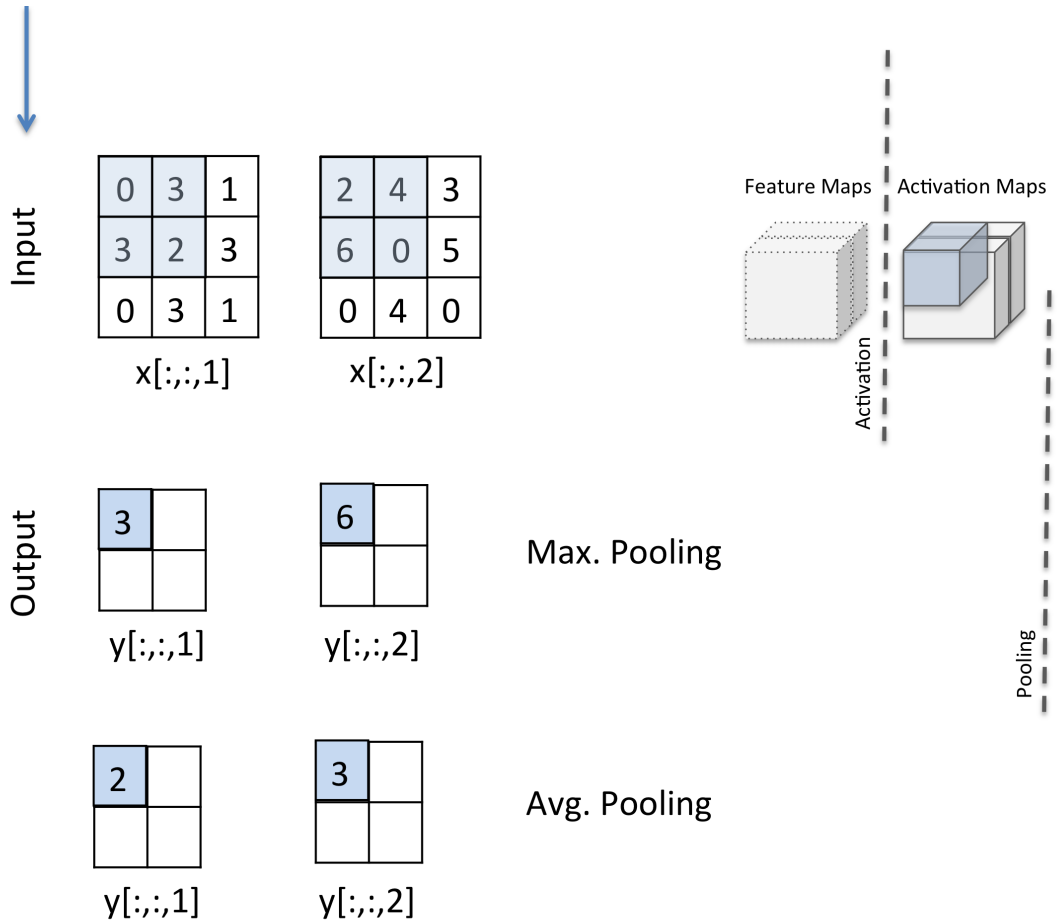
$$y_{i,j,d} = \max_{1 \le h \le H', 1 \le w \le W'} x_{i+h,j+w,d},$$

$$y_{i,j,d} = \frac{1}{H'W'} \sum_{1 \le h \le H', 1 \le w \le W'} x_{i+h,j+w,d},$$

- input $x \in \mathbb{R}^{H \times W \times D}$
- patch $p \in \mathbb{R}^{H' \times W'}$
- output $y \in \mathbb{R}^{H'' \times W'' \times D}$
- stride $S_{W,H}$ and padding $P_{W,H}$



Pooling Layer ($H'' \times W'' \times D \times N$) (3)

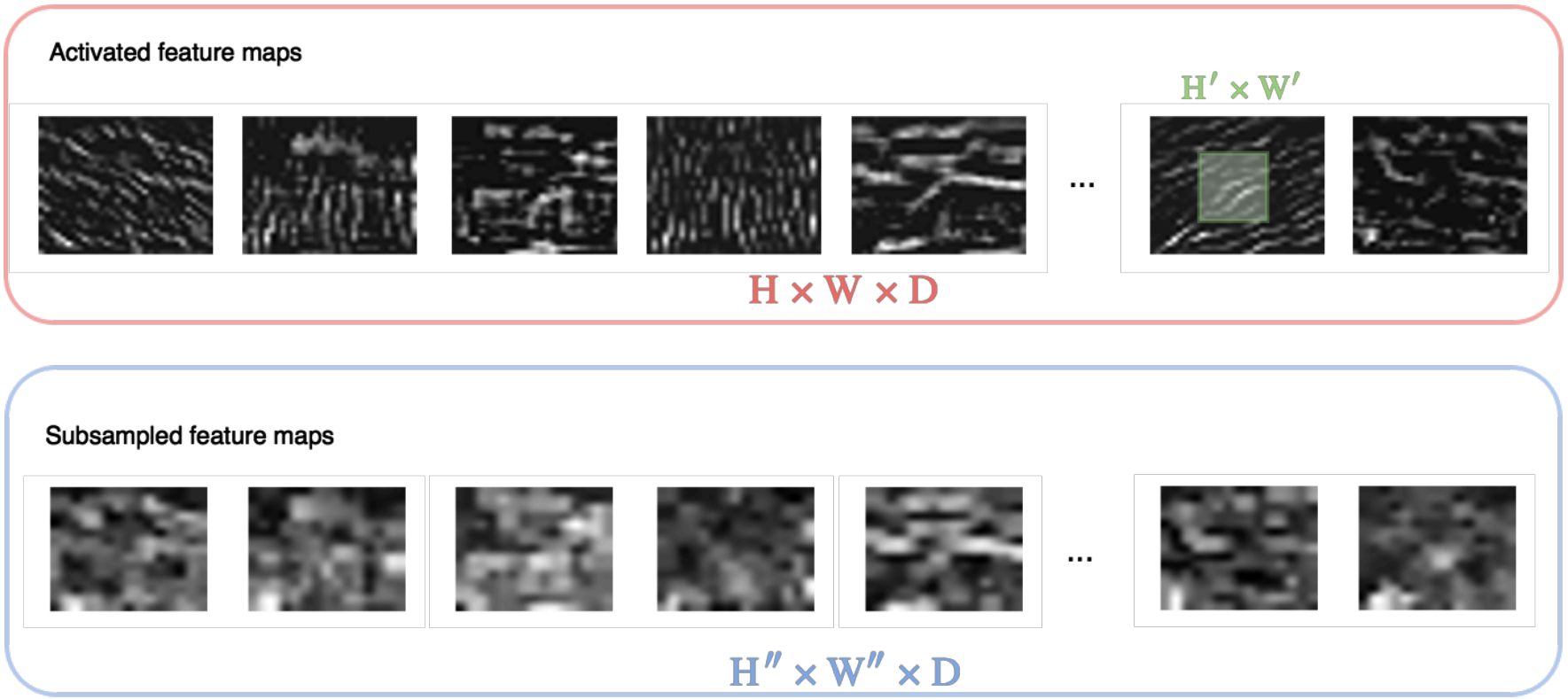


Pooling Layer ($H'' \times W'' \times D \times N$) (4)

Example: CIFAR-10 (Max. Pooling, $p = 3 \times 3, S = 2$)

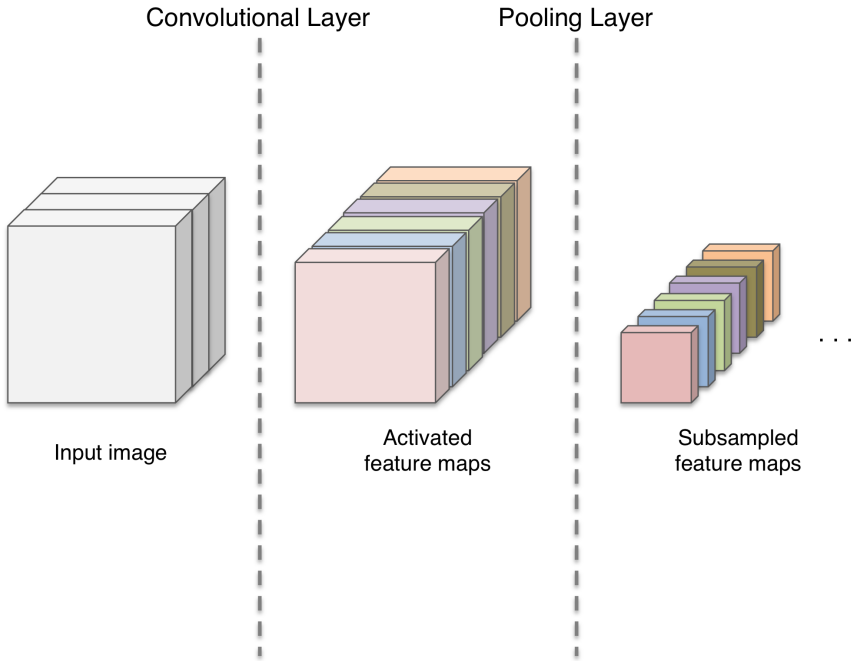
Keywords: Invariance to small transformation, Larger receptive field

Pooling Layer ($H'' \times W'' \times D \times N$) (5)



$$W'' = 1 + \frac{W - W' + (P_L + P_R)}{S_W}, \qquad H'' = 1 + \frac{H - H' + (P_U + P_D)}{S_H},$$

Normalization Layer ($H \times W \times D \times N$) (1)



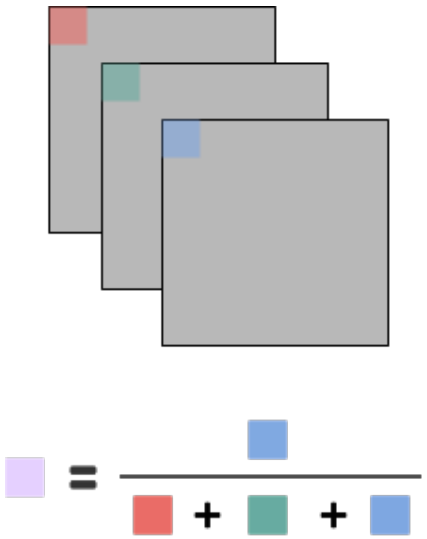
Normalization Layer ($H \times W \times D \times N$) (2)

It performs a cross-channel normalization at each spatial location as follows

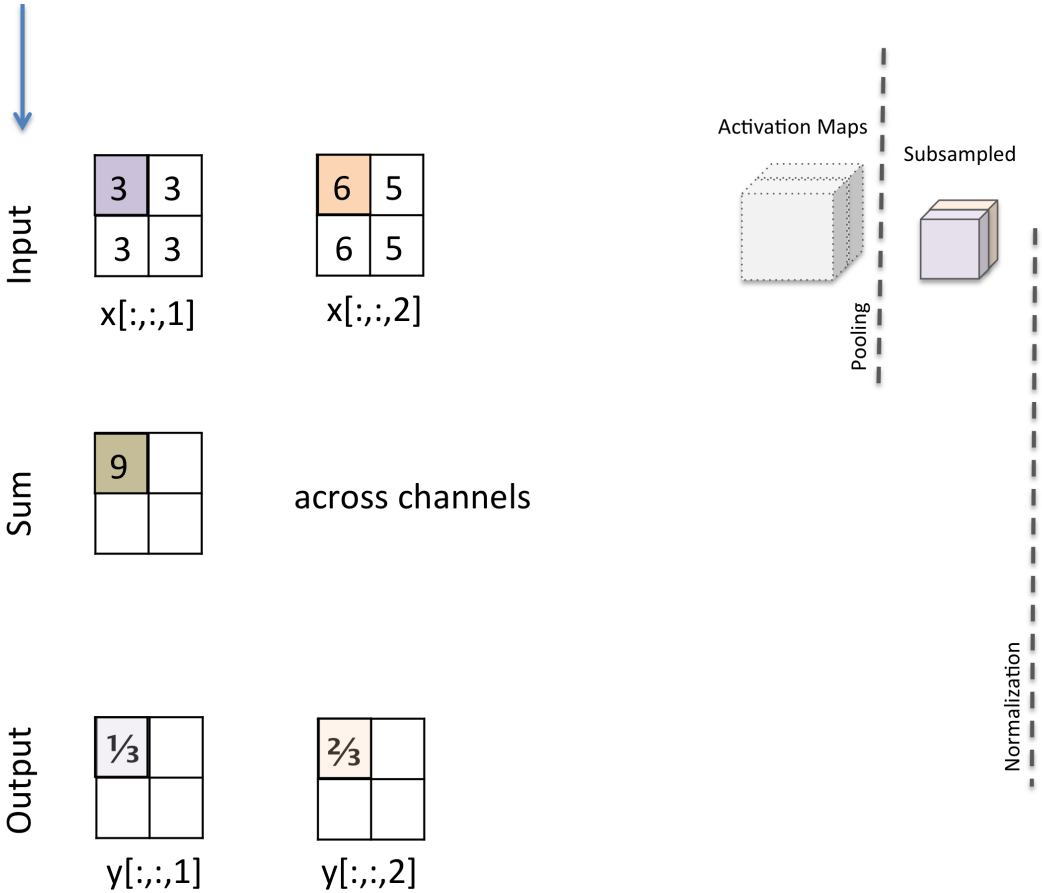
$$y_{i,j,d} = x_{i,j,d} \left(\kappa + \alpha \sum_{d \subset D} x_{i,j,d}^2 \right)^{-\beta},$$

where κ, α, β are hyperparameters. It is usually called Local Response Normalization (LRN).

- input $x \in \mathbb{R}^{H \times W \times D}$
- output $y \in \mathbb{R}^{H \times W \times D}$



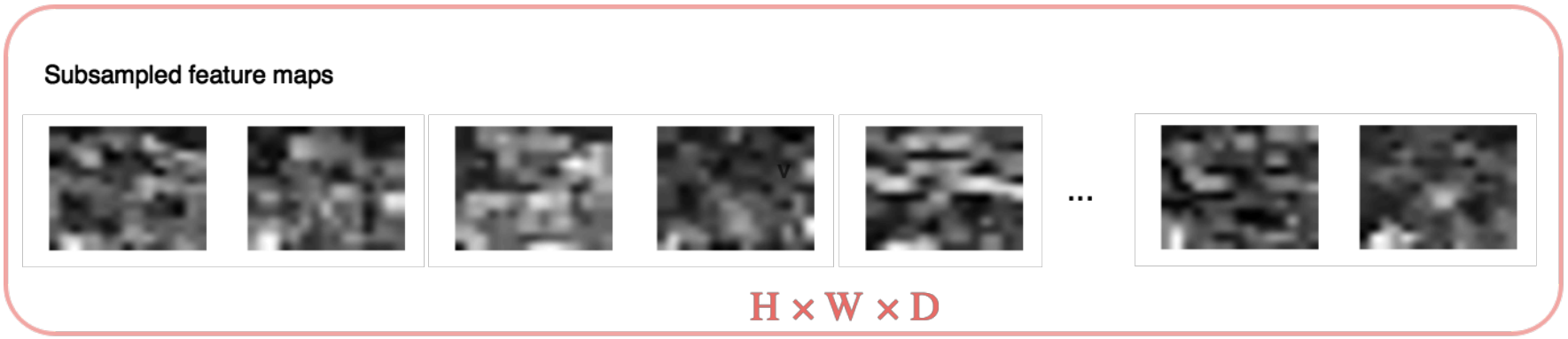
Normalization Layer ($H \times W \times D \times N$) (3)

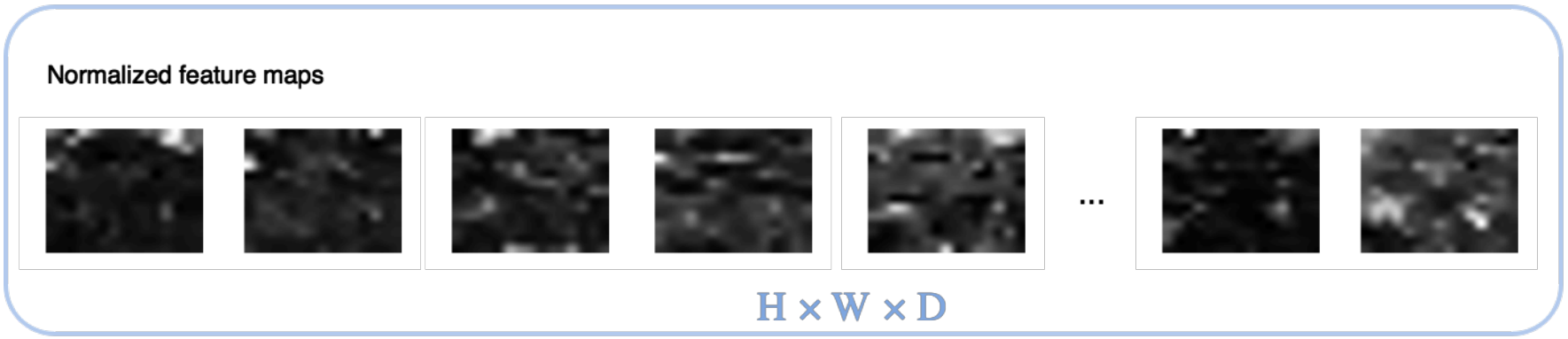


Normalization Layer ($H \times W \times D \times N$) (4)

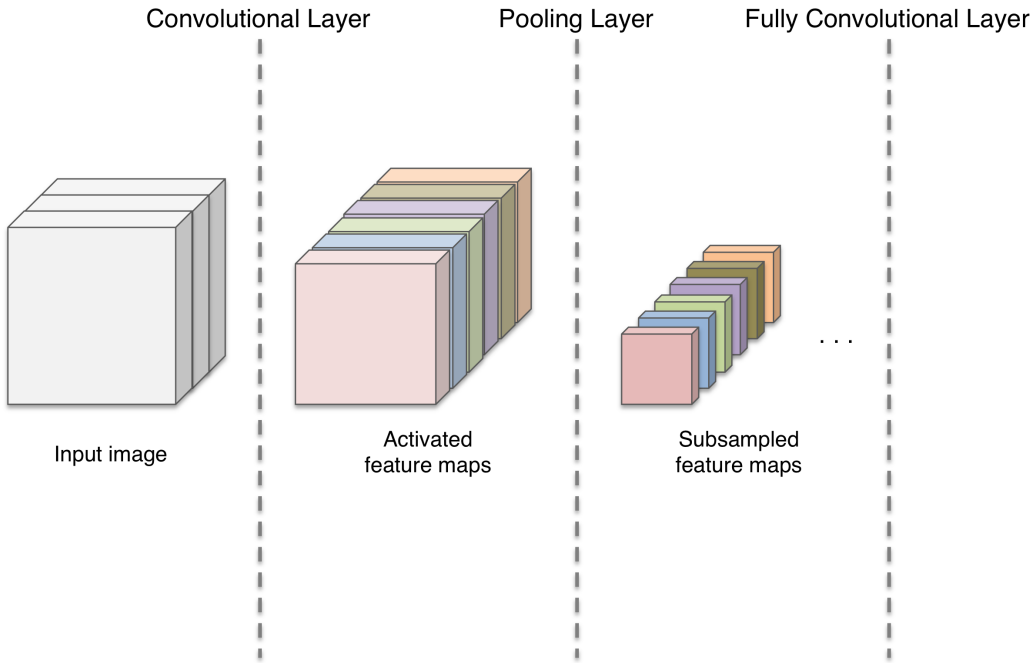
Example: CIFAR-10 (LRN, $\kappa = 0, \alpha, \beta = 1$)

Keywords: Within or Cross feature maps, Before or After Pooling, Have you spotted the mistake in the normalization process?





Fully Connected Layer ($1 \times 1 \times K \times N$) (1)



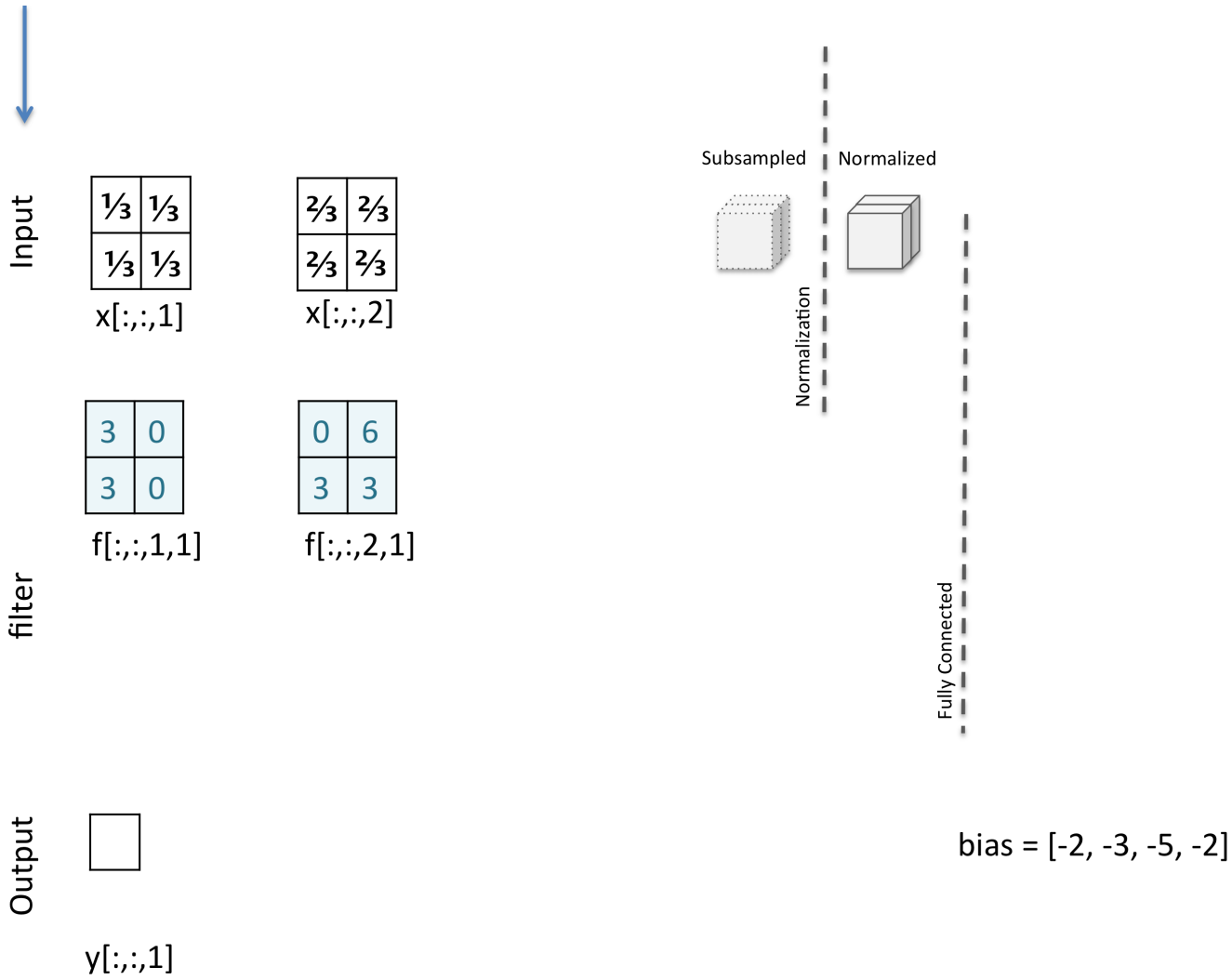
Fully Connected Layer ($1 \times 1 \times K \times N$) (2)

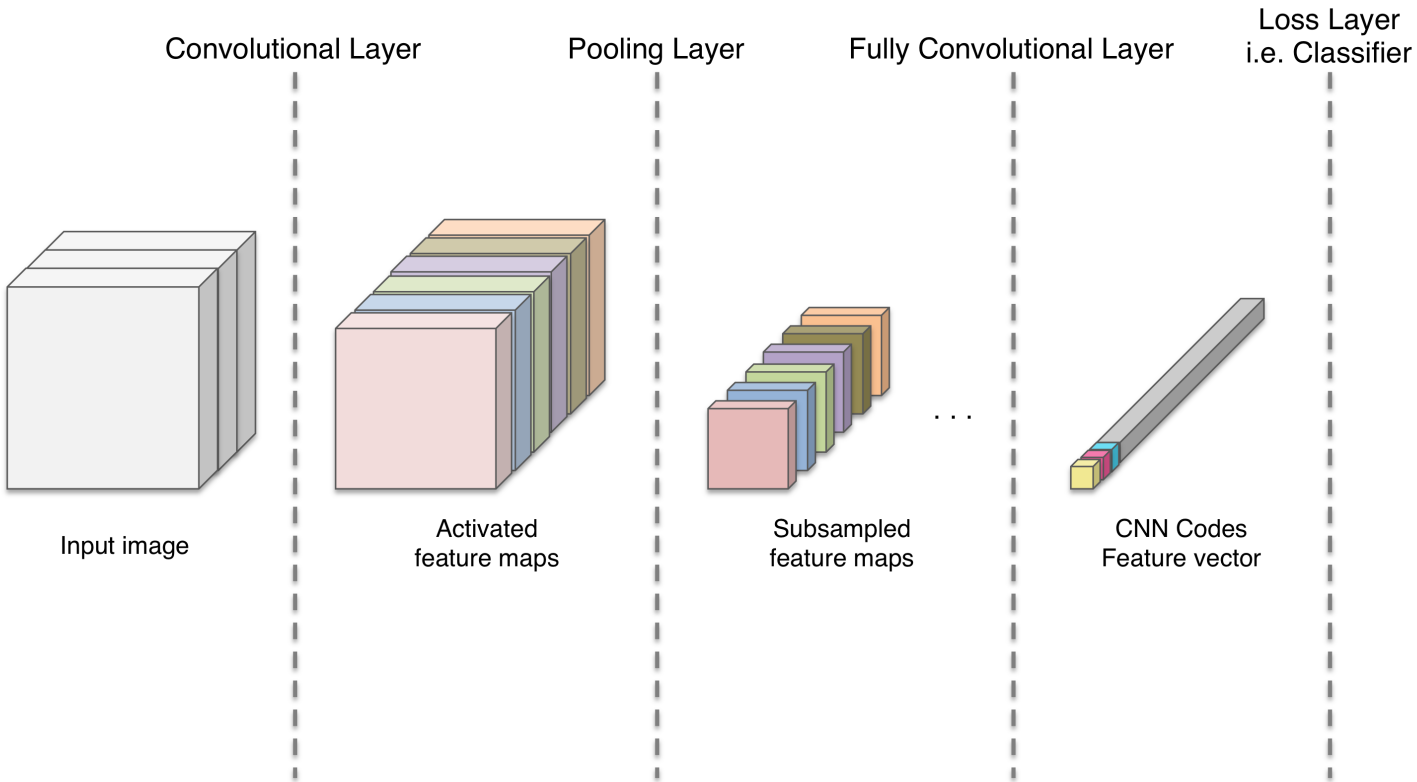
It computes the convolution of input feature maps x with a filter f as follows

$$y_{i,j,k} = b_{i,j,k} + \sum_{h=1}^{H} \sum_{w=1}^{W} \sum_{d=1}^{D} f_{h,w,d,k} \cdot x_{i+h,j+w,d},$$

- input $x \in \mathbb{R}^{H \times W \times D}$
- filters $f \in \mathbb{R}^{H \times W \times D \times K}$, we use K such filters.
- biases $b \in \mathbb{R}^{1 \times 1 \times K}$
- output $y \in \mathbb{R}^{1 \times 1 \times K}$
- stride and padding

Fully Connected Layer ($1 \times 1 \times K \times N$) (3)




Convolutional Layer
Pooling Layer
Fully Convolutional Layer
Loss Layer
i.e. Classifier
Input image
Activated
feature maps
Subsampled
feature maps
. . .
CNN Codes
Feature vector

Objective function

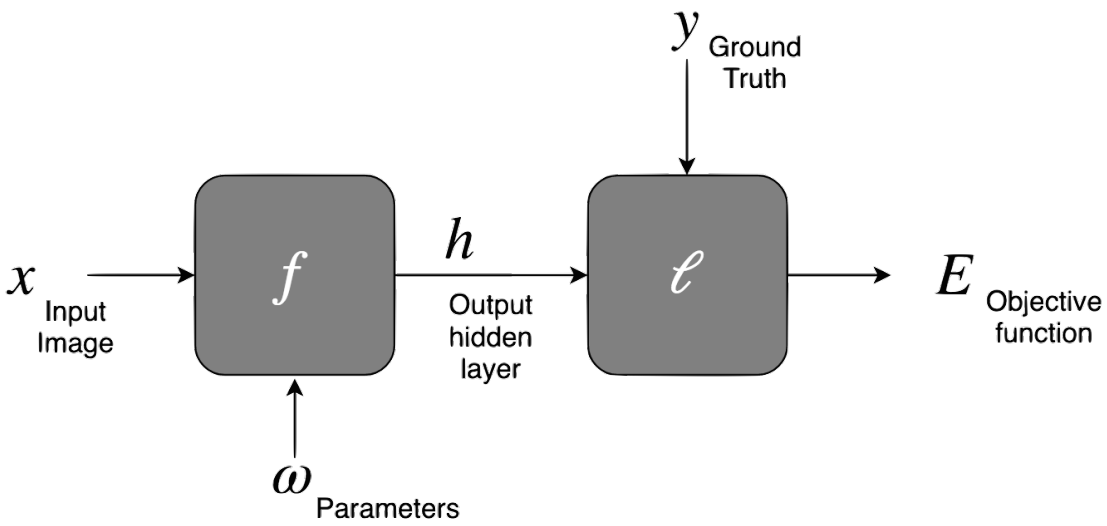
What we have presented so far is the feed-forward propagation, however, to minimize our objective function, we need to propagate back the gradients and update the parameters.

The Objective function:

$$\underset{\omega_1,\ldots,\omega_L}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \ell(f(x^{(i)}; \omega_1, \ldots, \omega_L), y^{(i)})$$

where $f(x; \omega)$ is the model's output.
Solver: Stochastic Gradient Descent (SGD).

Optimization and Derivatives (1)



Using the chain rule, the partial derivatives can be written as follows:

$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial h}\frac{\partial h}{\partial x}, \frac{\partial E}{\partial \omega} = \frac{\partial E}{\partial h}\frac{\partial h}{\partial \omega}$$

Optimization and Derivatives (2)

Vanilla update. The weight's update:

$$\omega^{t+1} = \omega^t - \frac{\eta}{n} \sum_{i=1}^{n} \nabla \ell(x, y; \omega^t),$$

where η is the learning rate.

Momentum update. Using the momentum[6], The weight's update becomes:

$$\omega^{t+1} = \omega^t - \frac{\eta}{n} \sum_{i=1}^{n} \nabla \ell(x, y; \omega^t) + \alpha \nabla \omega^t,$$

where α is the momentum.

Optimization and Derivatives (3)

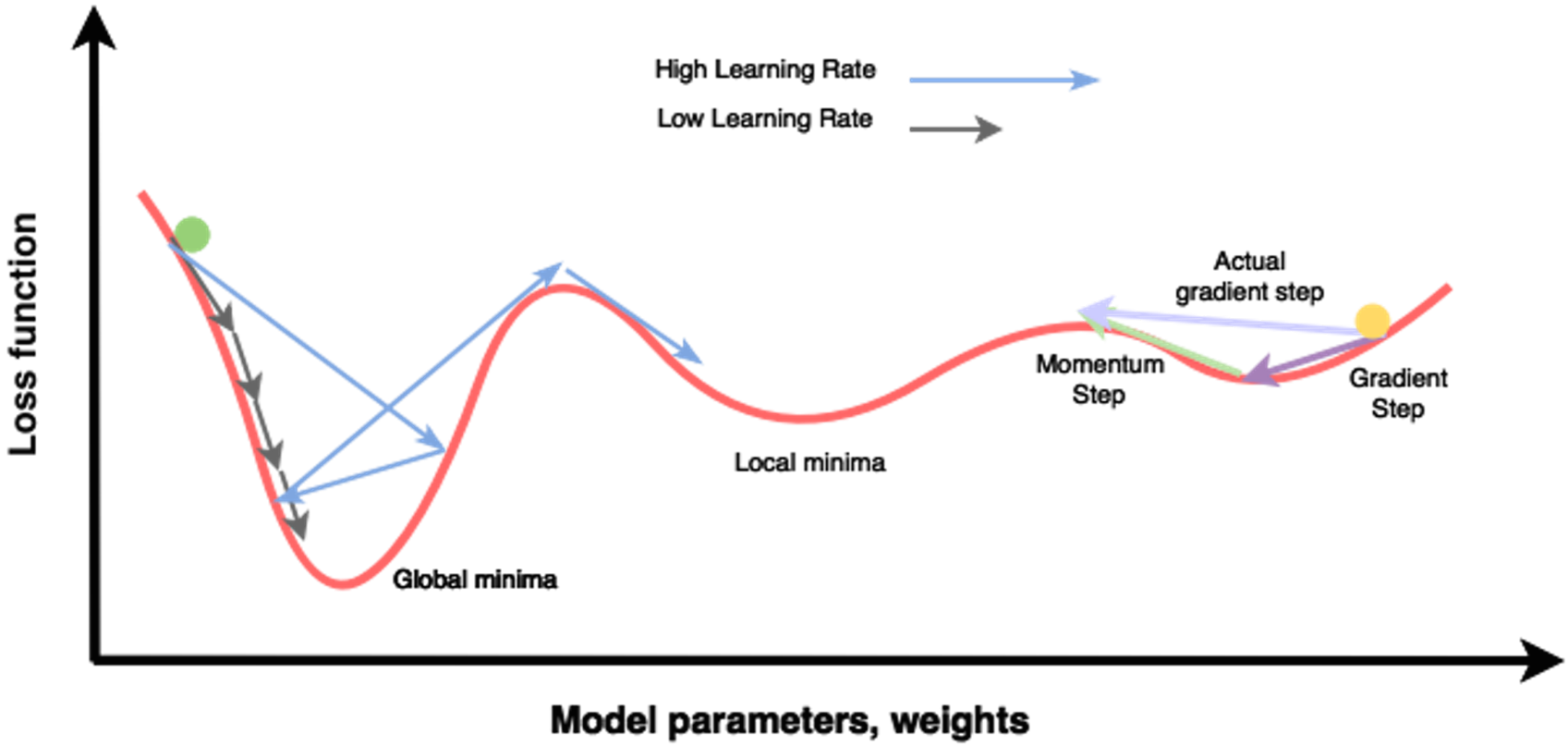


Figure: SGD & Learning Rate[7]

[6] **rumelhart1988learning**.

[7] http://imgur.com/a/Hqolp

Loss Layer ($1 \times 1 \times C \times N$) (1)

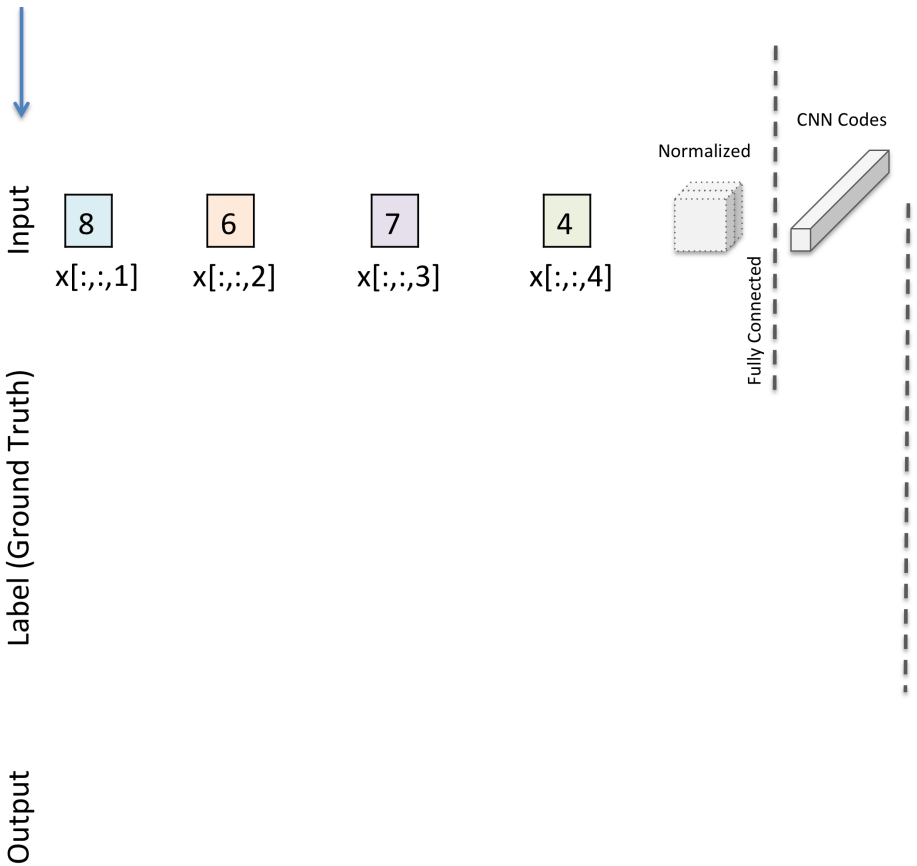
The loss function ℓ, mainly used in the training phase, could be softmax log-loss for "classification purpose"

$$y = -\sum_{i,j} \left(x_{i,j,c} - log \sum_{d=1}^{D} \exp x_{i,j,d} \right),$$

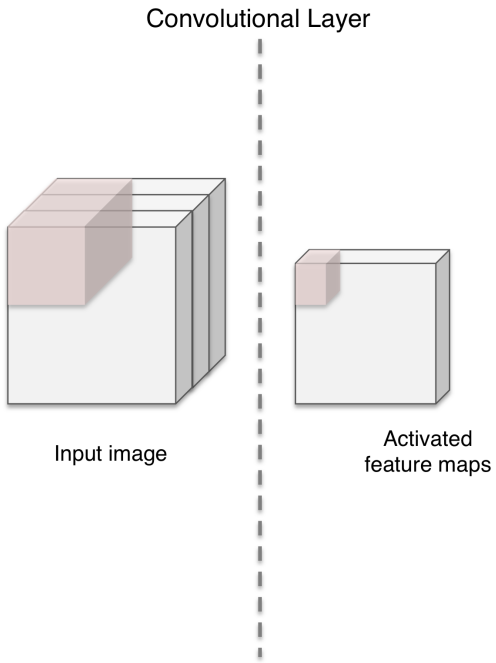
or ℓ_2-norm for "regression purpose" as follows

$$y = \|x_{i,j,c} - x_{i,j,d}\|_2^2,$$

Loss Layer ($1 \times 1 \times C \times N$) (2)



Recap



Low/Mid/High Level Features (1)

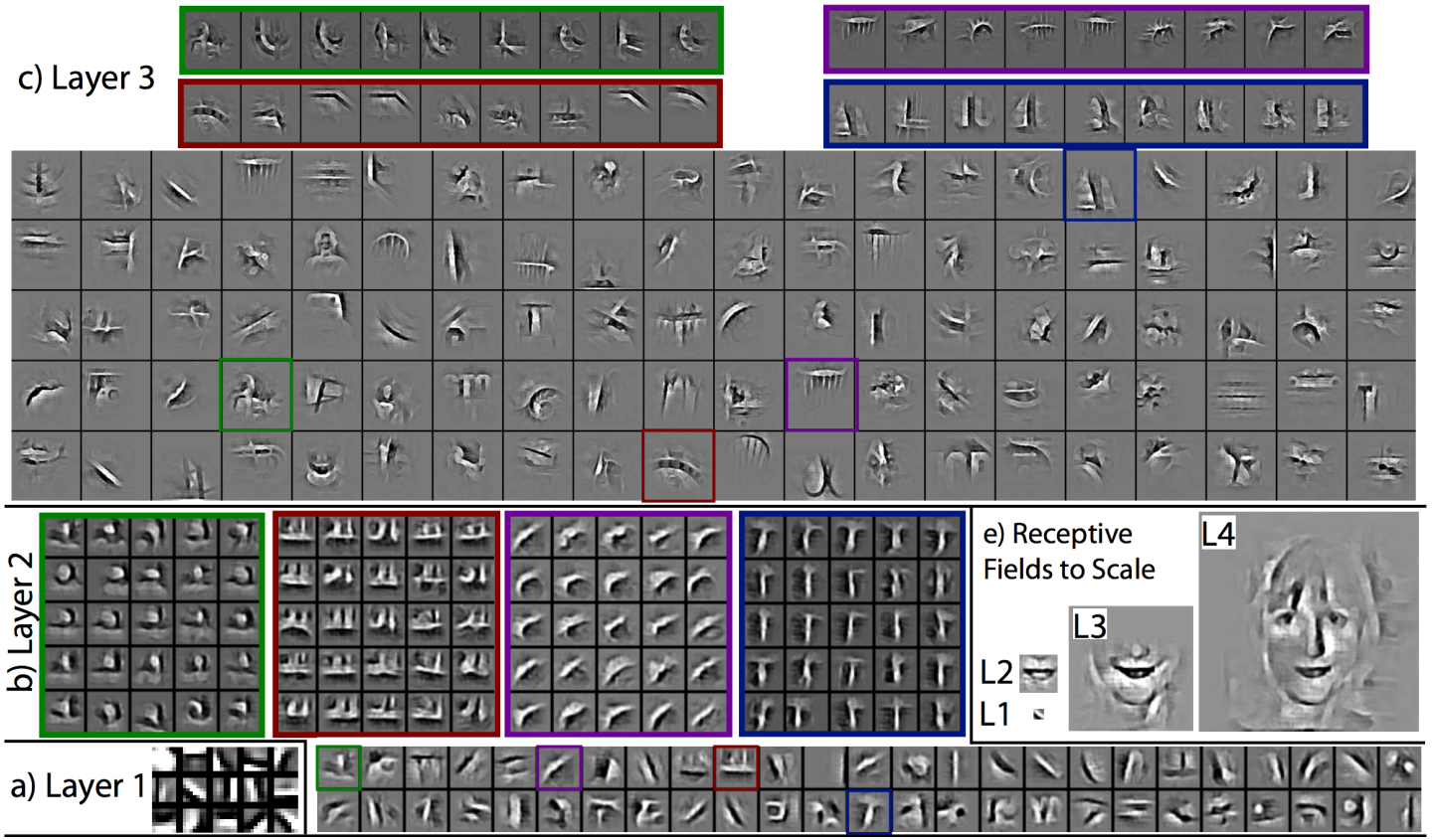


Figure: Low and Mid Level Features, Fig.5 in[8]

Low/Mid/High Level Features (2)

Figure: High Level Features, Fig.5

[8] **zeiler2011adaptive**.

Interactive Example

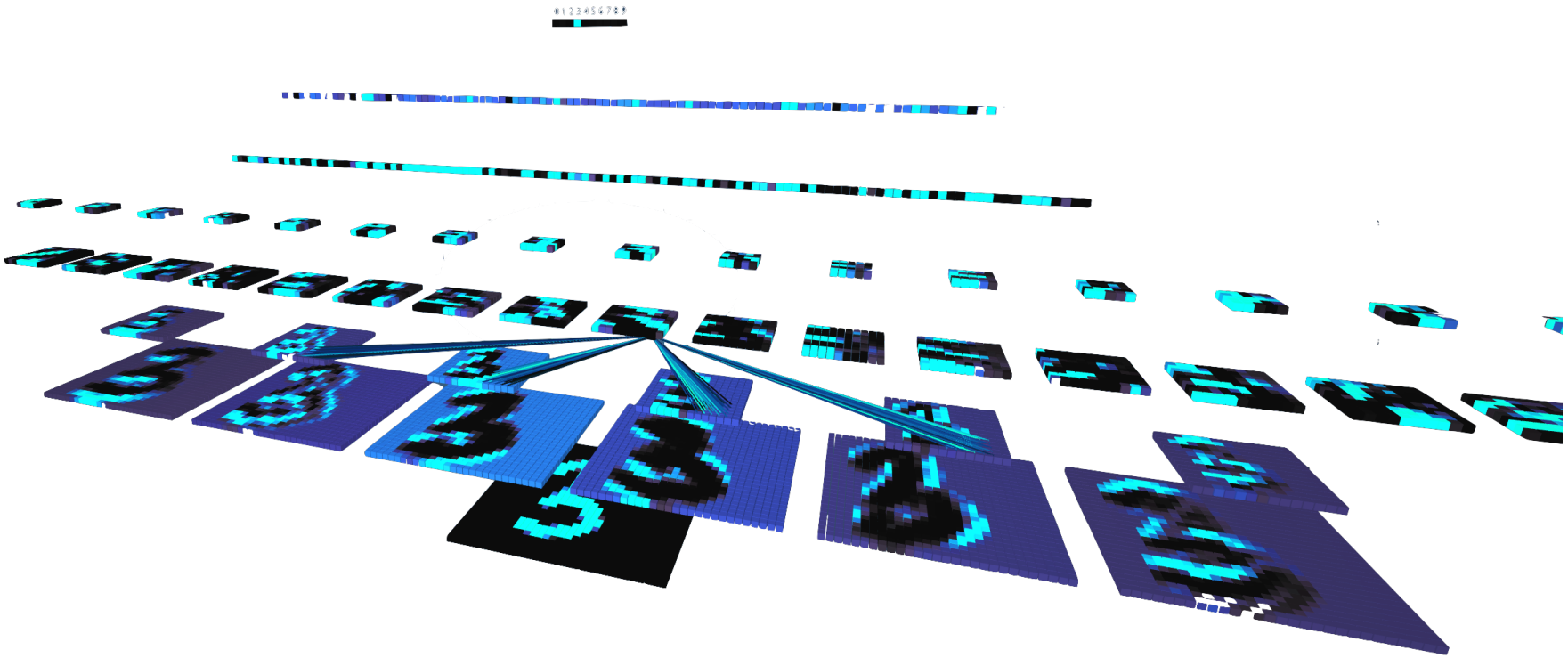
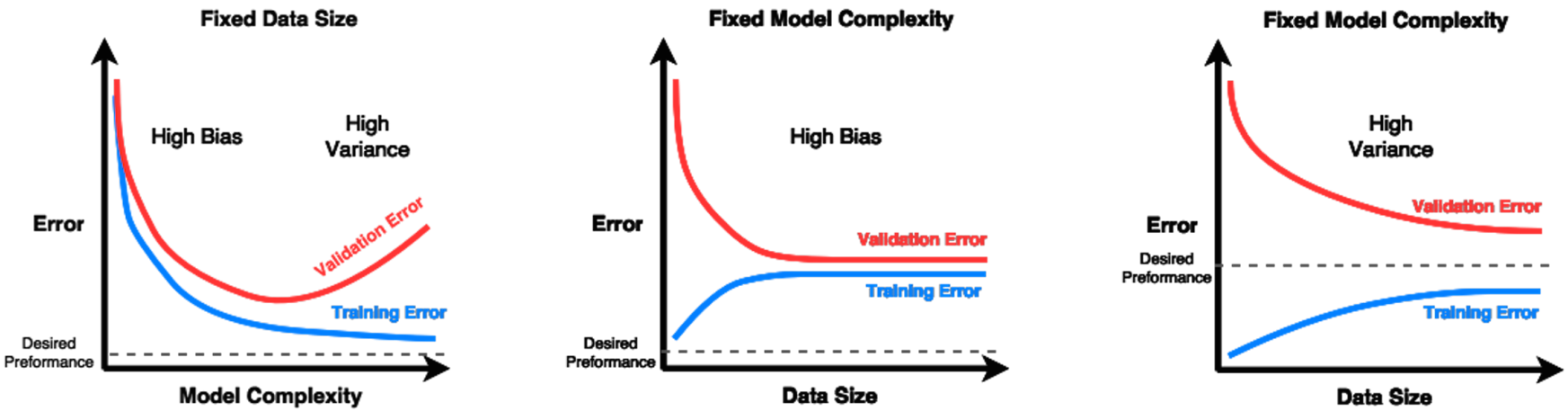


Figure: LeNet5 Architecture, MNIST-10[9]

[9] http://scs.ryerson.ca/~aharley/vis/conv/

Network Training

Model Check. Similar to any model-based machine learning, there are two types of error source; 1) Bias and 2)Variance.



How to fix High Bias? High Variance?[10]

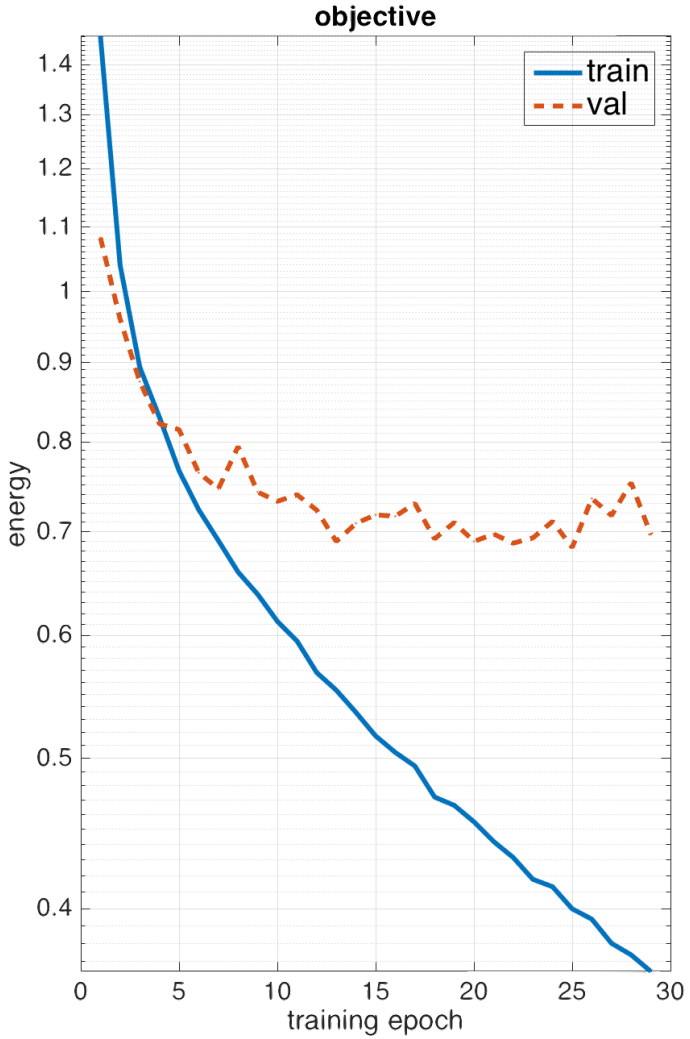
High Variance: Getting more training data (data augmentation), smaller set of features, increase regularization parameter, add more dropout.

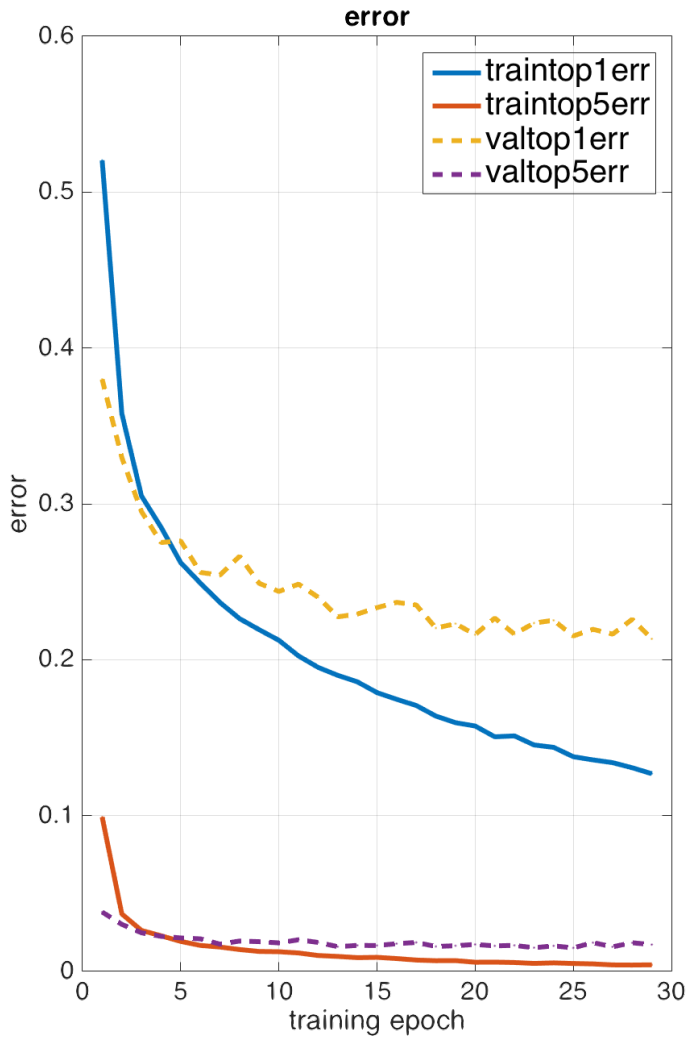
High Bias: Getting larger set of features, deeper architecture.

[10] **ng2009advice**.

Network Training

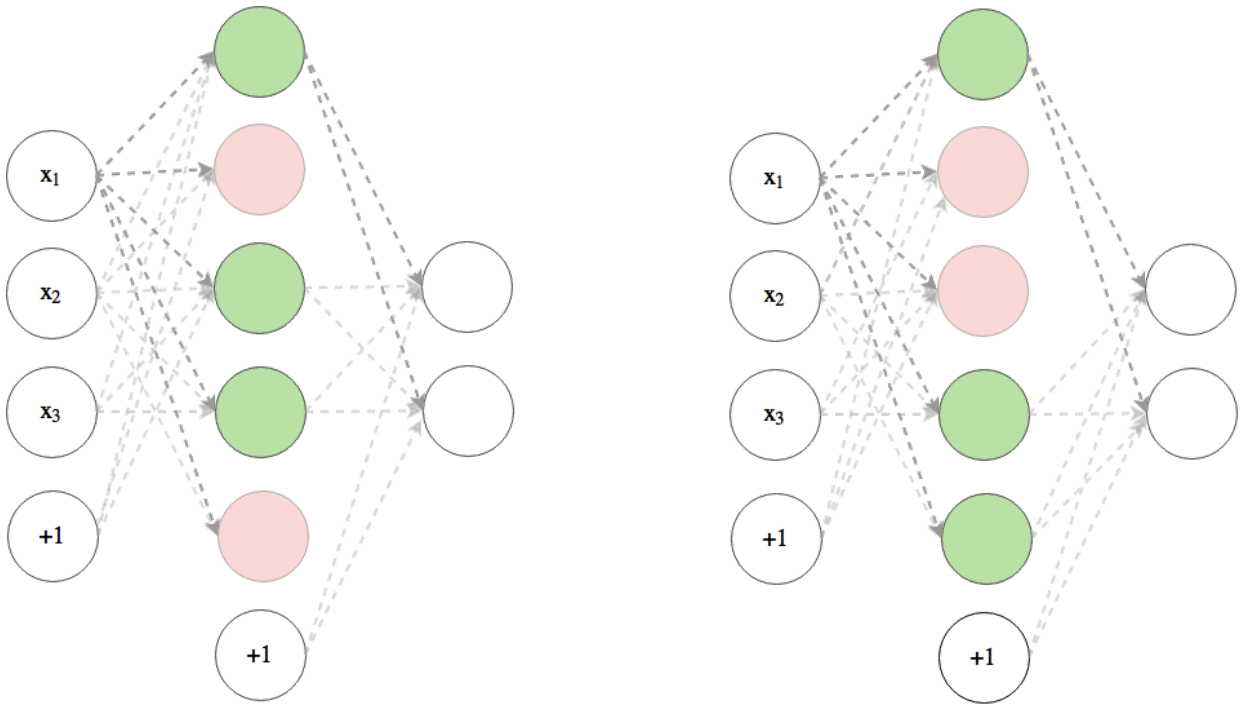
Example: Monitoring the training of tiny VGG model (30 Epochs)





Dropout Layer ($1 \times 1 \times C \times N$)[11]

The dropout layer acts as a regularizer for the network to avoid overfitting. It is simply "dropping out" some activation units and setting them to zero during the training phase. It is similar to train thinner networks and do averaging.

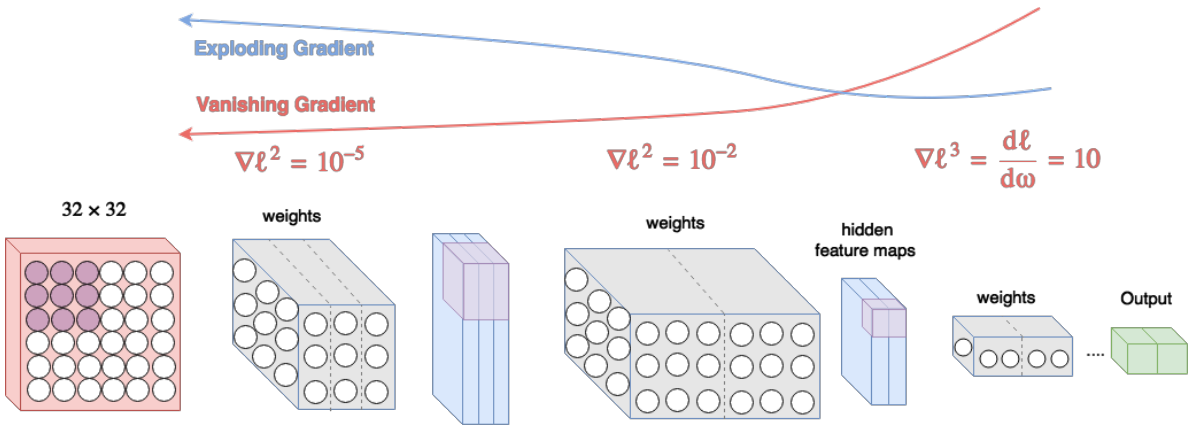


[11] **srivastava2014dropout**.

Network Debugging

Gradient Checks

- One of the major problems with training a CNN deep model is vanishing/exploding gradient[12].
- Monitor gradient and activation across layers and epochs.
- Try adding Batch Normalization layer, proper weight initialization[13].

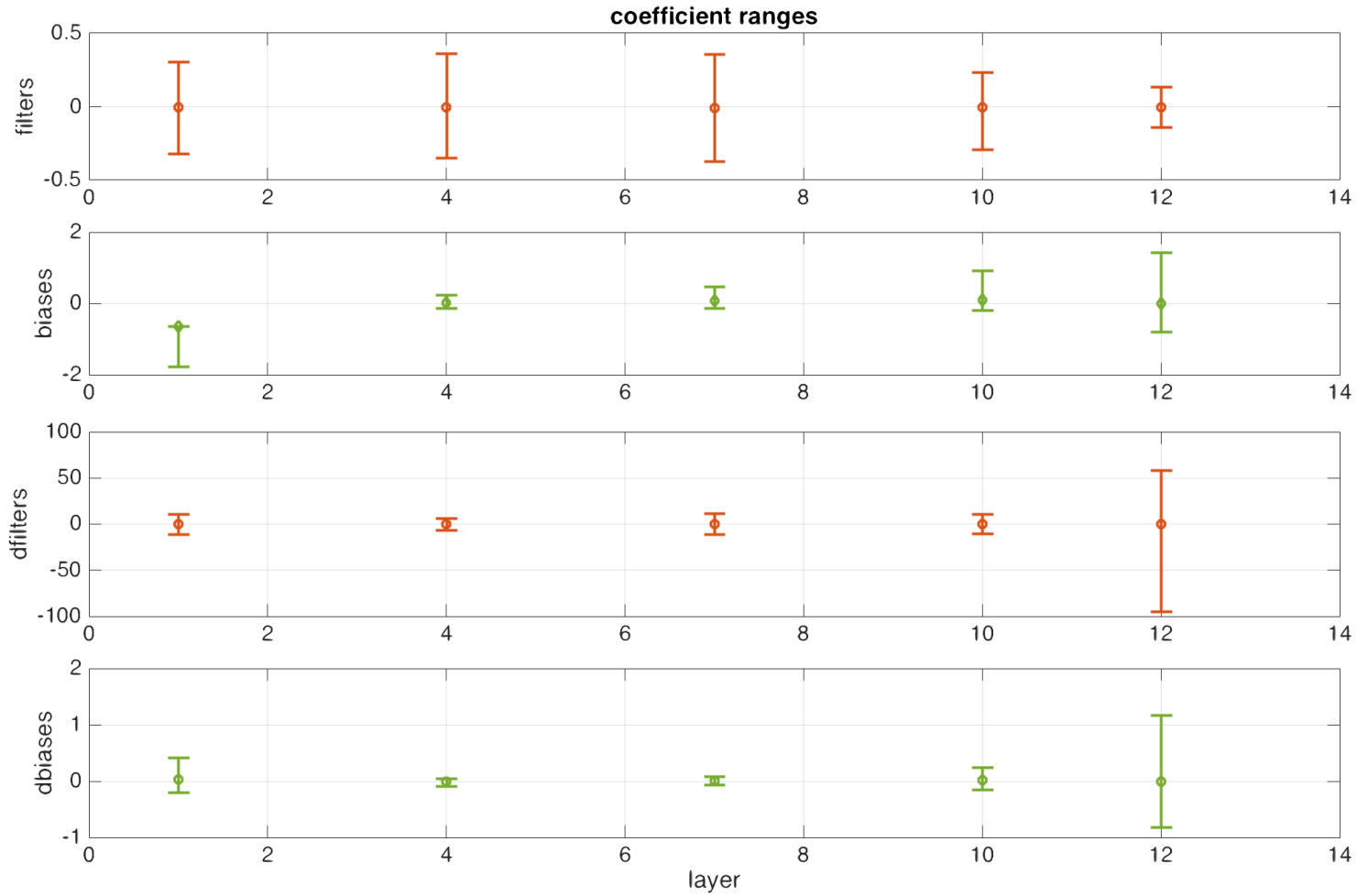


[12] **bengio1994learning**.

[13] **krahenbuhl2015data**.

Network Debugging

Example: Monitoring the gradient of tiny VGG model (Epoch 26)

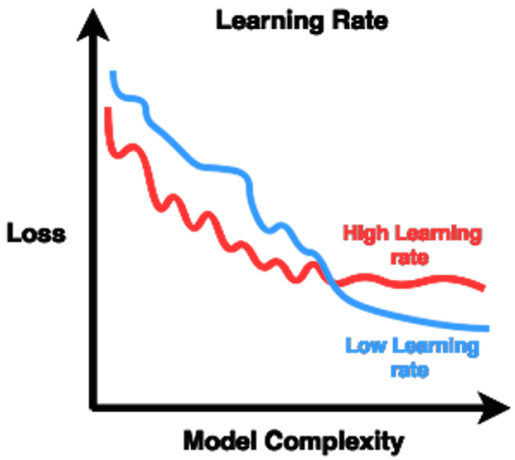


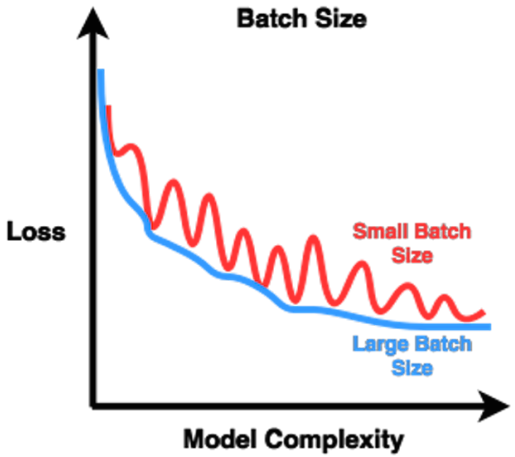
Network Debugging

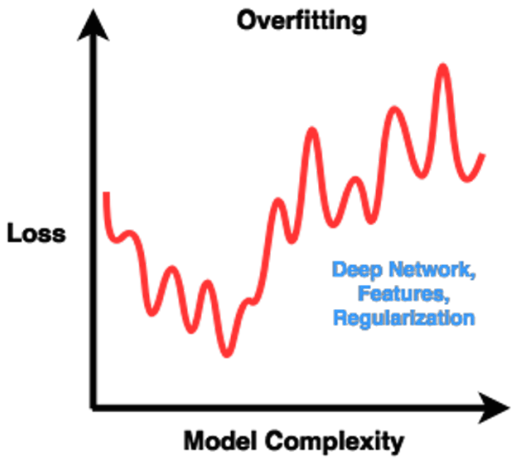
Sanity Checks

- Check if you have an expected loss value *(Hint: Set the regularization parameter to Zero.)*
- Increasing the regularization parameter will increase the loss.
- Overfit a very small subset of data.

Loss Checks







Additional Layers

- Deconvolutional Layer[14]
- Batch Normalization[15]
- DropConnect[16]

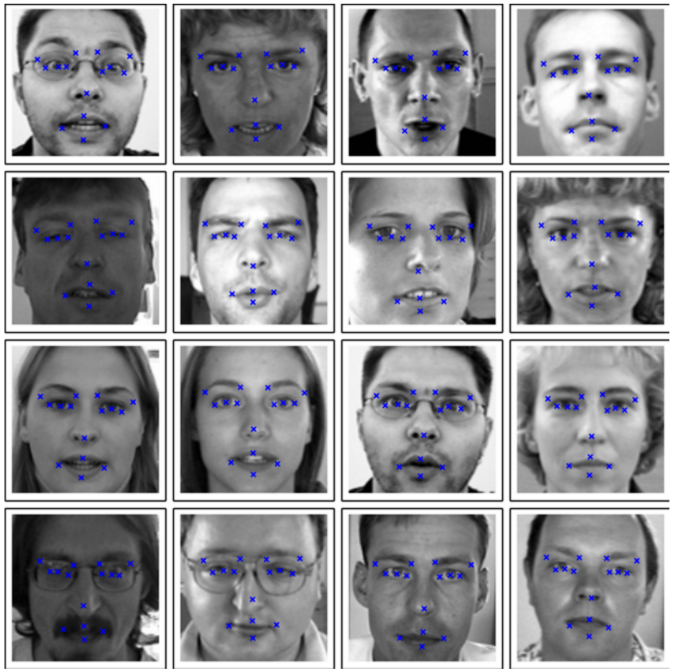
[14] **zeiler2011adaptive**; **zeiler2014visualizing**.

[15] **ioffe2015batch**.

[16] **wan2013regularization**.

Example: Facial keypoints tutorial

- Dataset: Facial Keypoint Detection challenge, Training: 7049 (96×96) gray images with 15 keypoints. Testing: 1783 images.
- Loss function: Regression (MSE)
- Parameters: Optimization: nesterov momentum, Learning rate: 0.01, Momentum $= 0.9$.



Note: Image Courtesy of this example at[17], Facial keypoint challenge[18].

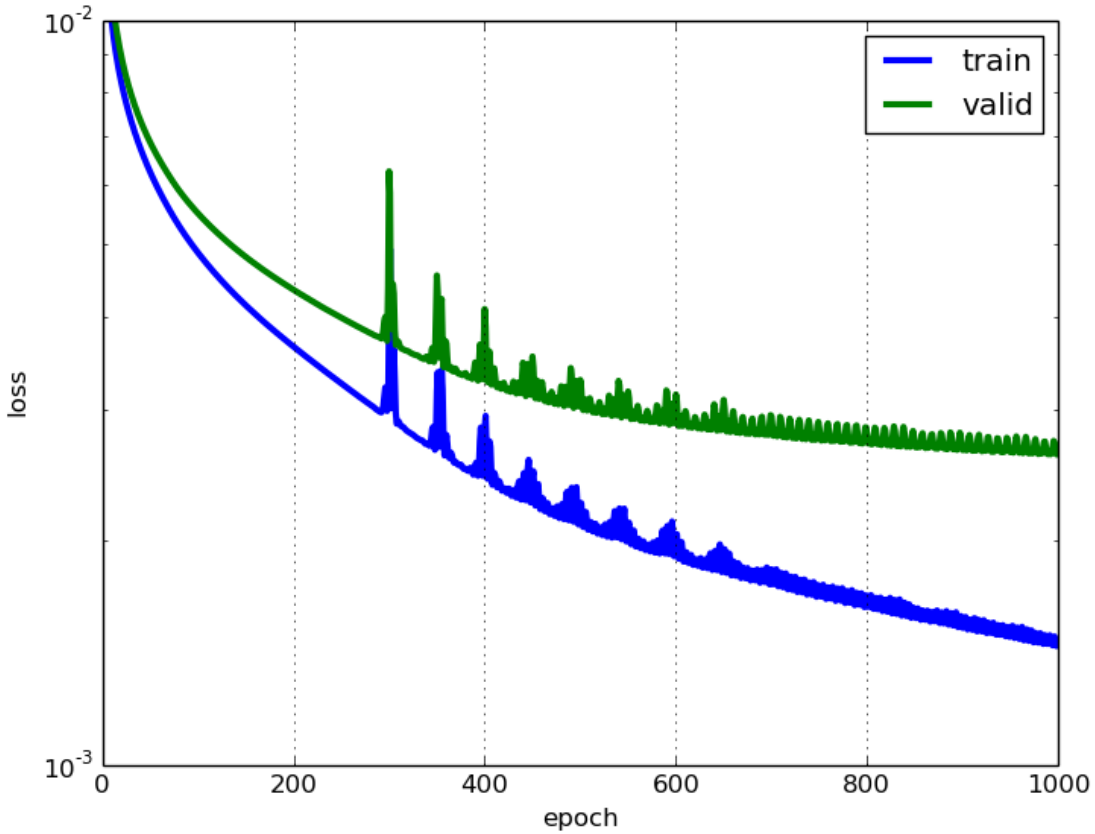
[17] **dnaiel_KP2014**.

[18] **Kaggle2013**.

Example: Facial keypoints tutorial (Cont.) (1)

One layer network (net1)

- Network: One hidden layer, (9216, 100, 30) units.
- Parameters: Number of Epochs = 400.

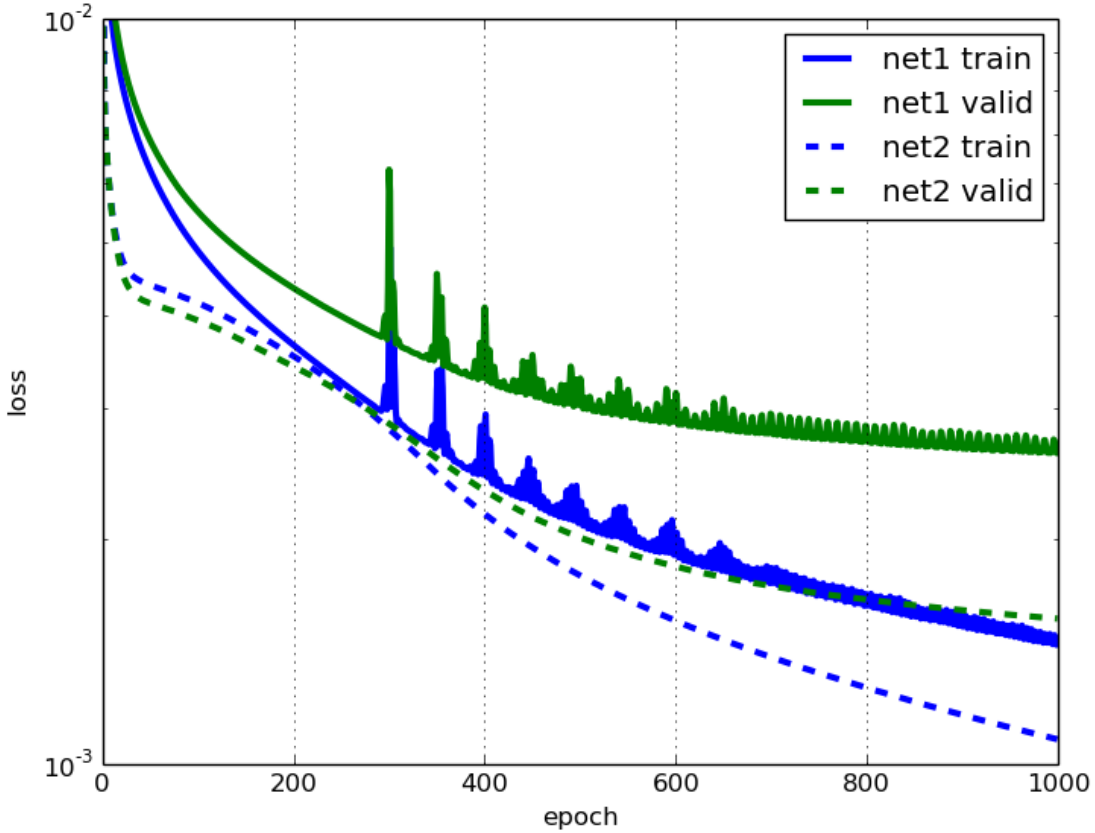


Example: Facial keypoints tutorial (Cont.) (2)

LeNet5 network (net2)

- Network: Input, (Conv, maxPool)^3 + FC^2, Output
- Parameters: Number of Epochs = 1000.

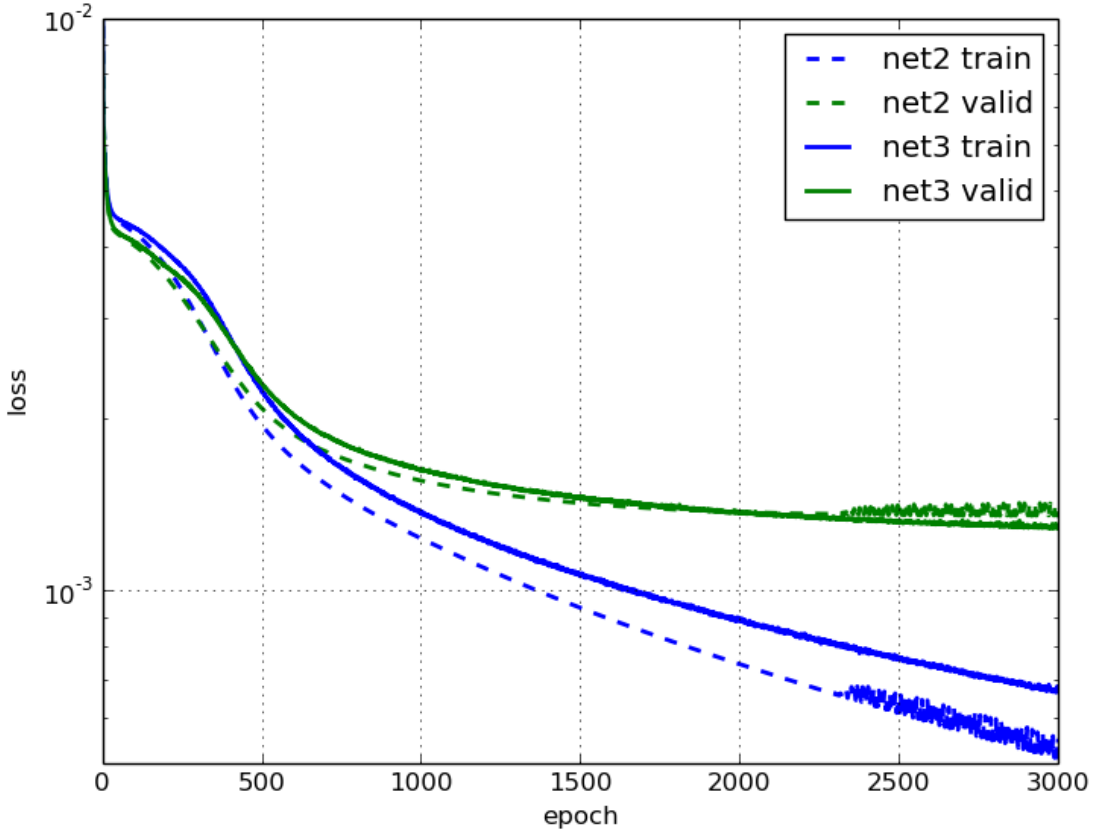
Example: Facial keypoints tutorial (Cont.) (3)



LeNet5 network (net3)

Example: Facial keypoints tutorial (Cont.) (4)

- Data Augmentation, only flipping 50% of datasets.
- Parameters: Number of Epochs = 3000.

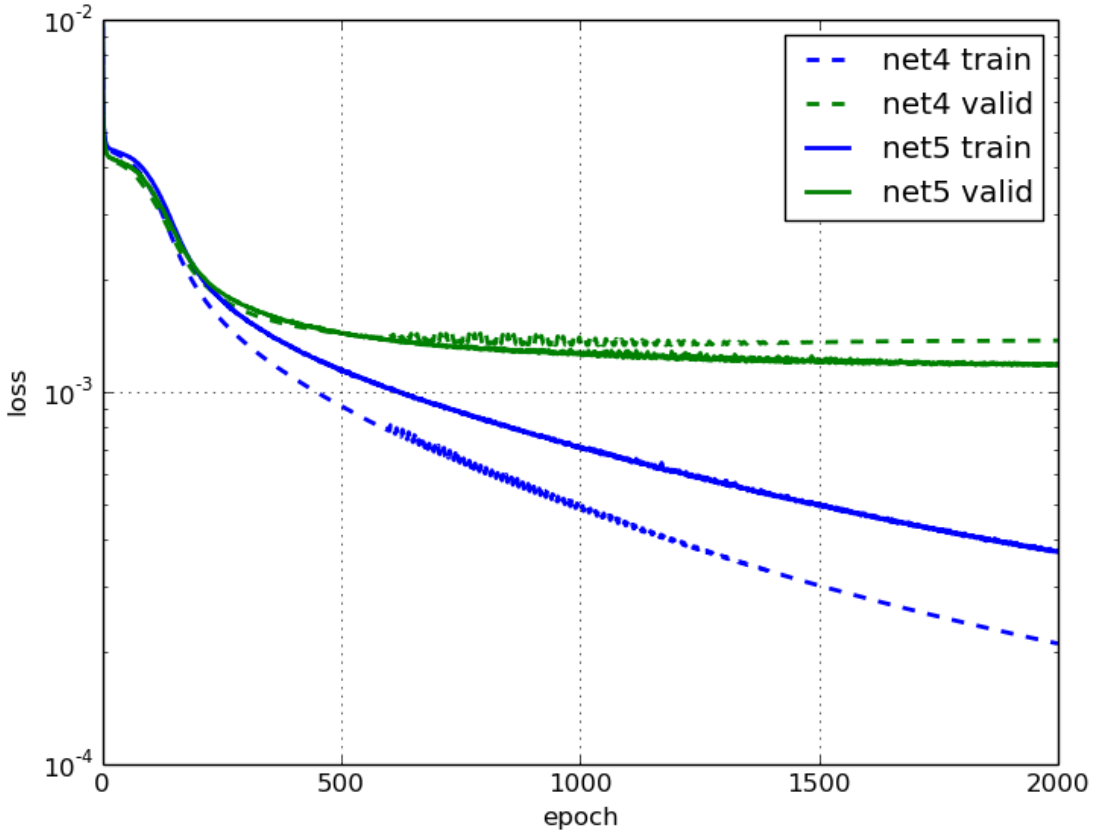


Example: Facial keypoints tutorial (Cont.) (5)

LeNet5 network (net4, net5)

- Parameters: Learning Rate = 0.03-0.0001, Momentum = 0.9 - 0.999
- with/without Data Augmentation

Example: Facial keypoints tutorial (Cont.) (6)



Transfer Learning

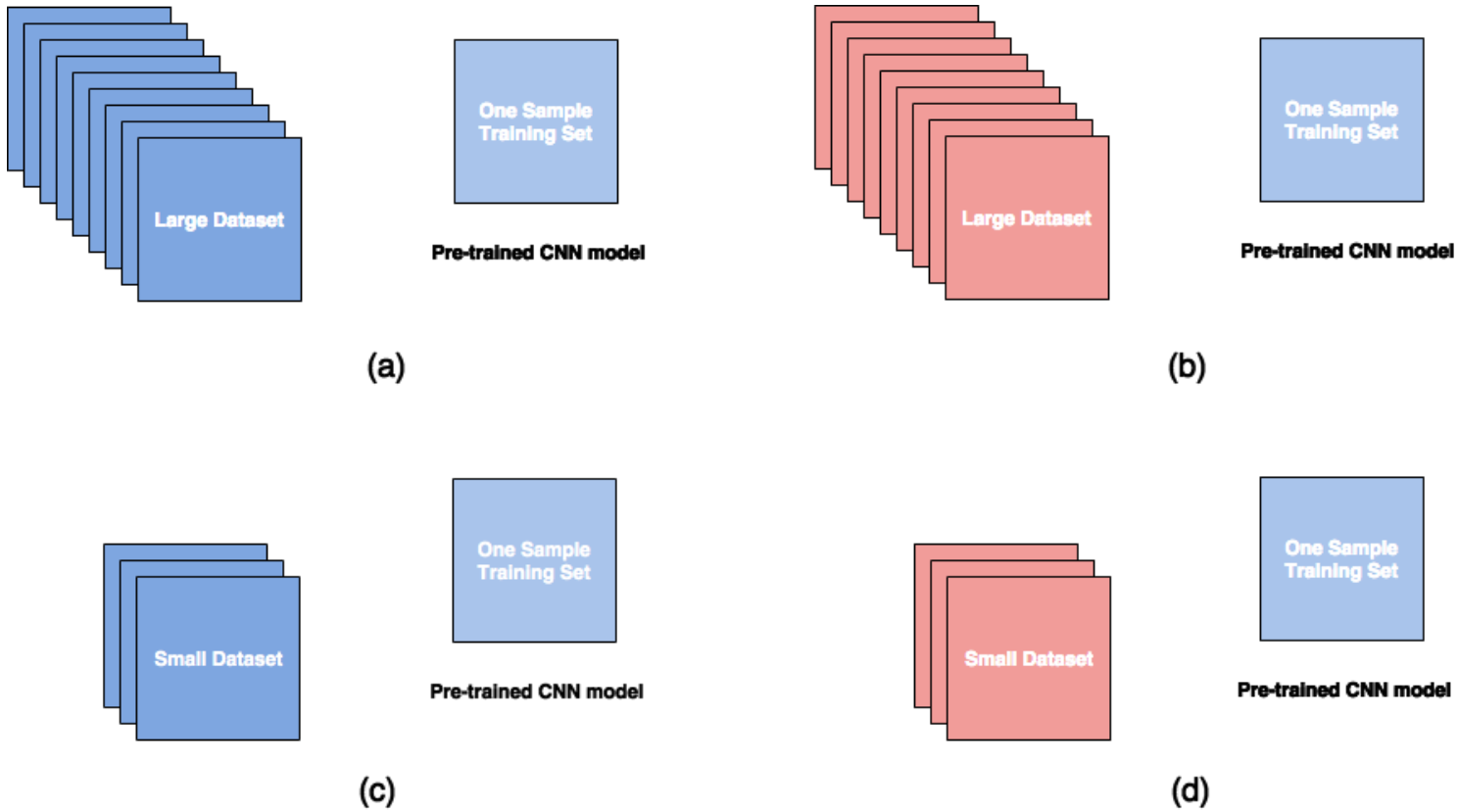
Learning from scratch. Inspired by some CNN architecture, you can design your own network. However, you need tons of data.
Transfer Learning[19]**.** Once you don't have enough data, you can use the pre-trained CNN models [20] for the following tasks:

- Extract features: The output of the last hidden layer before the softmax can be used as features (CNN Codes) to train a linear SVM classifier.
- Fine-tuning: You may need to propagate back your gradient to update the weights, however, the weights of the first layers can be fixed during the fine-tuning and update the weights of the higher layers.

[19] **li2015cs231n**.

[20] Caffe: https://github.com/BVLC/caffe/wiki/Model-Zoo
MatConvNet: http://www.vlfeat.org/matconvnet/pretrained/

Fine-Tuning Tricks



(a) Fine-tuning, (b) Train from scratch, initialize the weights of the first layers from a pre-trained model, (c) Get the CNN codes and learn a linear SVM, (d) Get the CNN codes from the mid-layers and learn a linear SVM.

Hyper-parameters: additional topics

- Optimization solver[21].
- Learning Rate Schedule[22]: The more intuitive way to choose the learning rate is to set it high in the beginning (large step and faster), and then lower it down after some epcohs (small step and slower), i.e. $\eta = \frac{\eta_0}{n_{iter}+\kappa}$ or $\eta = \eta_0 e^{-\kappa n_{iter}}$.
- Momentum[23]
- Batch Size: between 10 and few hundreds.
- Weight Initialization[24].

[21] **bottou2012stochastic**; **ngiam2011optimization**.

[22] **bengio2012practical**.

[23] **sutskever2013importance**.

[24] **wagner2013learning**.

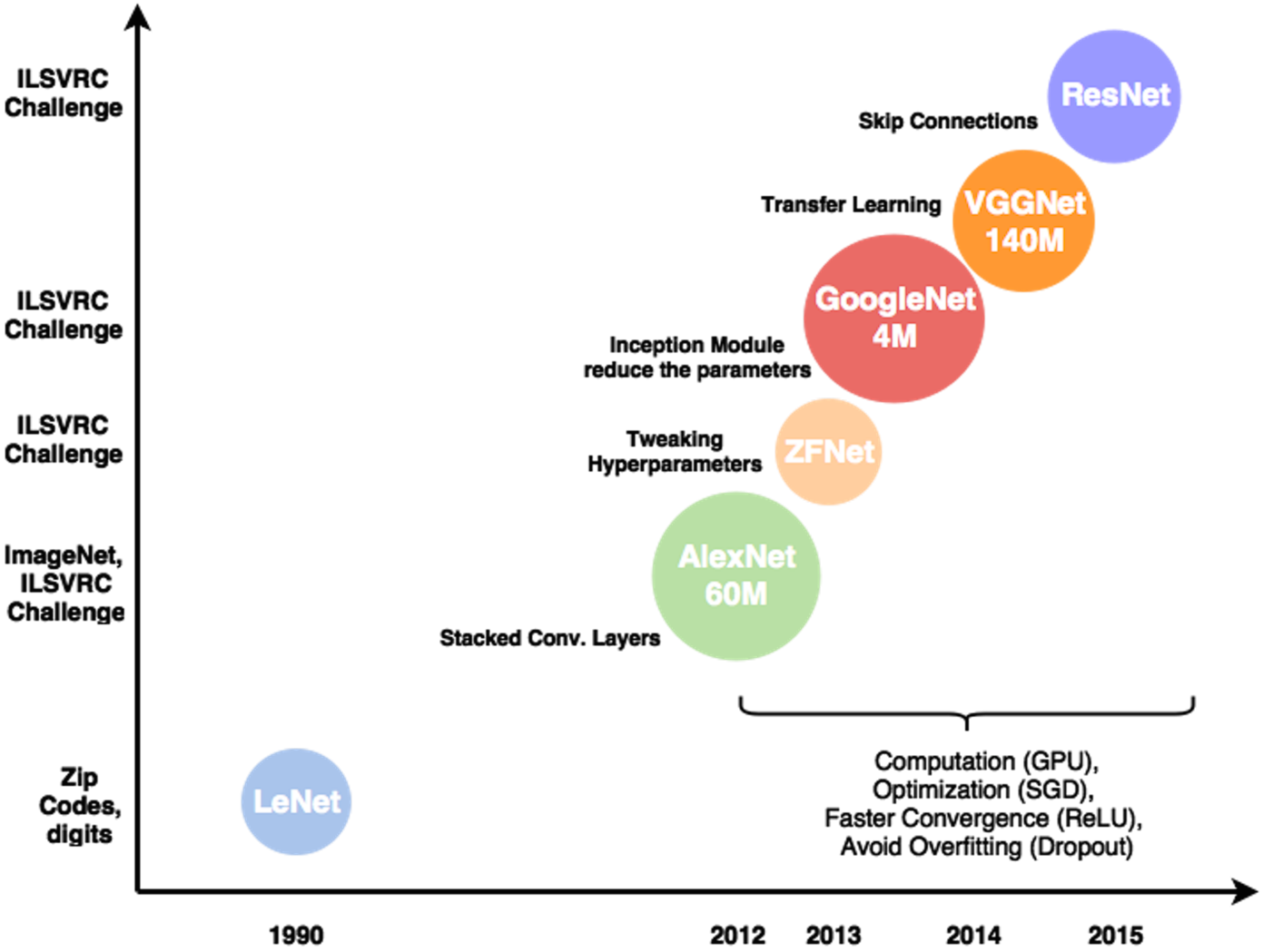
Parameters exploding!

Let's compute how many parameters we have in LeNet5 network (c.f. Fig. **??**):

(Conv. + maxPool.)^3		
conv5-6	$1 \times 5 \times 5 \times 6 + 6$	156
conv5-16	$6 \times 5 \times 5 \times 16 + 16$	2416
conv5-120	$16 \times 5 \times 5 \times 120 + 120$	48120
FC^2		
conv1-84	$120 \times 1 \times 1 \times 84 + 84$	10164
conv1-10	$84 \times 1 \times 1 \times 10 + 10$	850

61706 parameters, quite good number! Have a look at the number of parameters for the recent networks in the next slide.

ConvNets Successes



Case Studies (AlexNet) (1)

Architecture: $(\text{Conv} + \text{Max. Pooling})^3 + \text{FC}^3$
DataSet: ILSVRC-1000, 1.2 million training images (RGB 256^2), 50,000 validation, and 150,000 testing images.
Pre-Processing: DeMean the training images. Training Param: LR $= 0.01$, M $= 0.9$, batch $= 128$

Case Studies (AlexNet) (2)

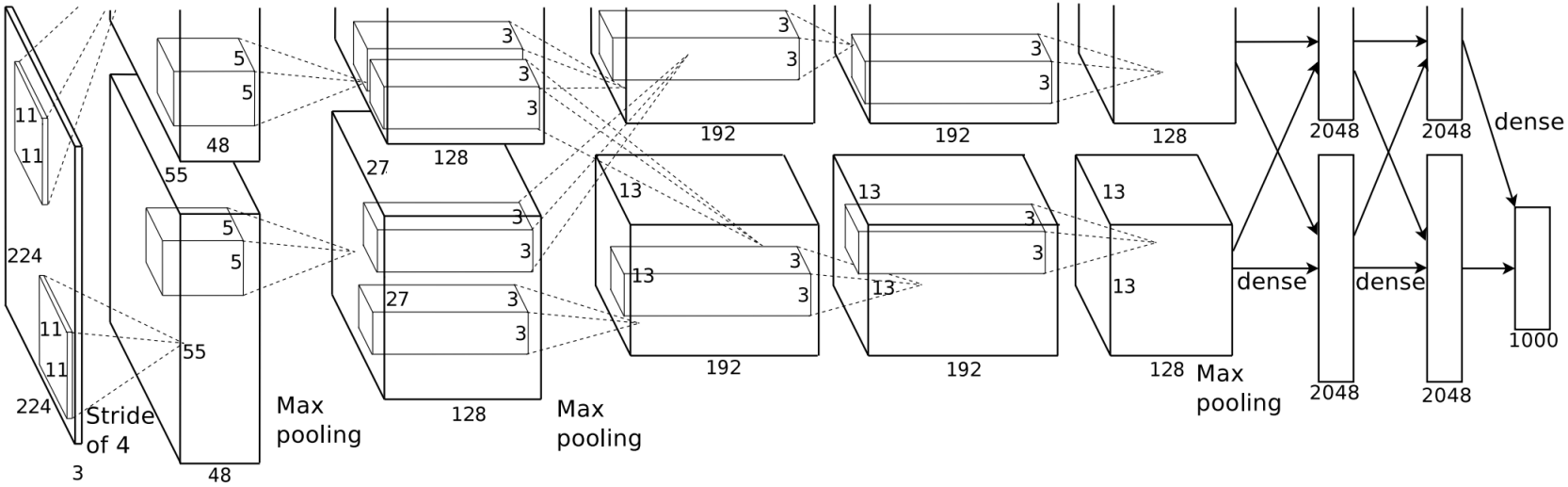


Figure: AlexNet. Fig.2

Case Studies (AlexNet) (3)

- ReLU converges faster than $\tanh(\cdot)$, 6 times faster at 25% error rate.
- LRN is applied after ReLU, hyper-parameters are determined using validation set ($\kappa = 2, \alpha = 10^{-4}, \beta = 0.75$). Error rate: $13\% \rightarrow 11\%$.
- Overlapping Pooling. Error rate $\rightarrow -0.4\%$
- Data Augmentation including translation, flipping and altering color intensities. Error rate $\rightarrow -1\%$.
- Dropout at rate 0.5 (Over-fitting)

Note: Images Courtesy of this case study at[25].

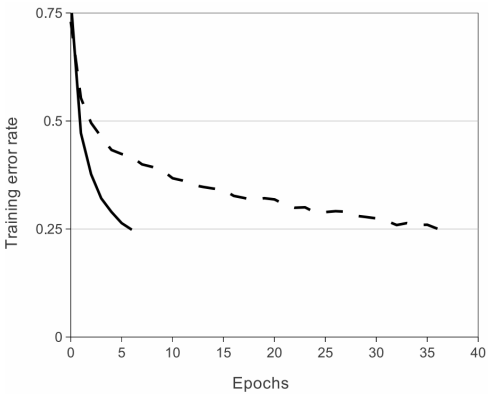


Figure 1: A four-layer convolutional neural network with ReLUs **(solid line)** reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons **(dashed line)**. The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

[25] **krizhevsky2012imagenet**.

Thanks!

Questions