MACHINE LEARNING

Deep Neural Networks: Neural Networks with Imaging Data

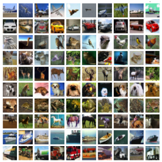
Last Update: 16th December 2022

Prof. Dr. Shadi Albarqouni

Director of Computational Imaging Research Lab. (Albarqouni Lab.)
University Hopsital Bonn | University of Bonn | Helmholtz Munich

STRUCTURE

MOTIVATION



Training Set

Linearly separable feature Space?

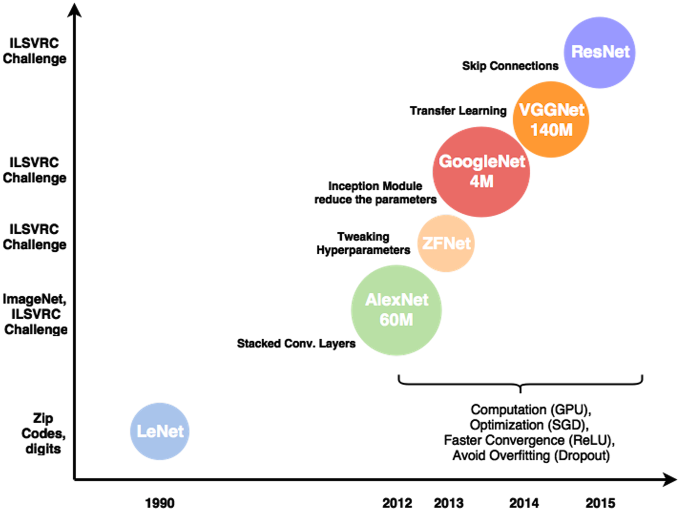
OBJECT RECOGNITION: PIPELINE

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

Image

Since 1950's

CONVNETS SUCCESSES

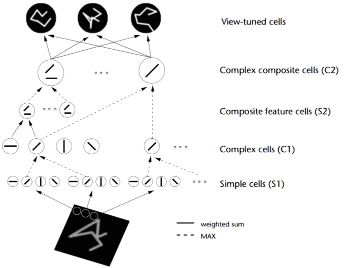


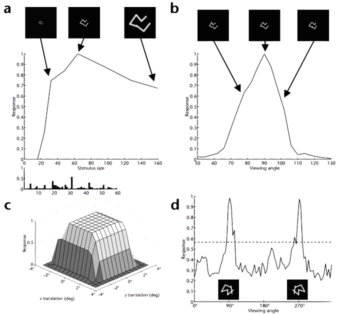
INTRODUCTION

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 [14] .





WHAT'S WRONG WITH ANN?

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

NETWORK ARCHITECTURE

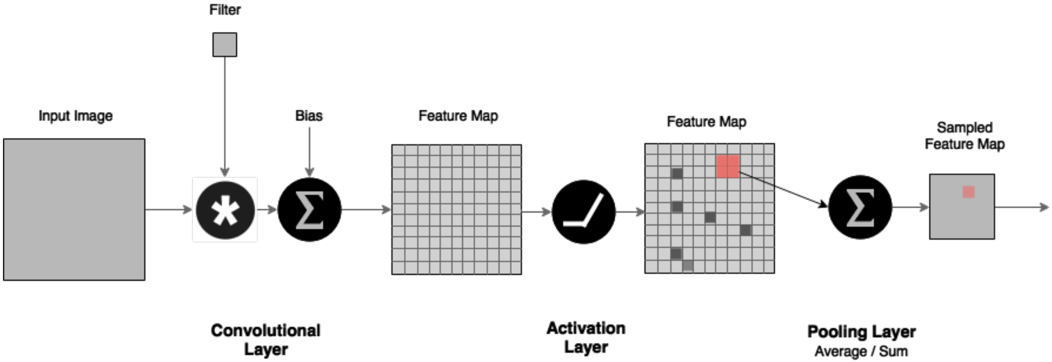


Figure: Symbolic Architecture

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

NOTATION

We follow the following notations

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 $\hat{y}$:

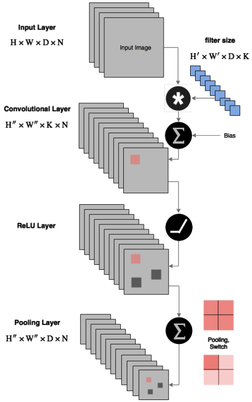
$$\hat{y} = f(x; \omega),$$

where ω is the model parameter.

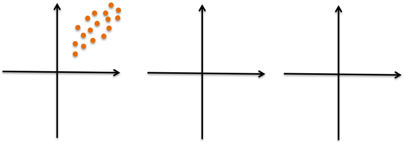
CNN LAYERS

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

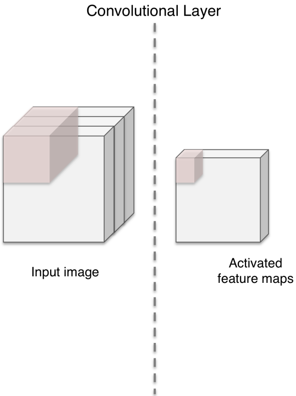


- Data Preprocessing (Mean subtraction, PCA/Whitening)
- Data Augmentation: geometric transformation; rotation and translation, color transformation: illumination, staining ...etc, adding noise.
- Splitting the dataset (training, validation and testing)
- Batch size

INPUT LAYER ($H \times W \times D \times N$)

- 2D inputs
  - Gray ($D = 1$)
  - RGB ($D = 3$) [4]
- 2.5D inputs
  - Gray ($D = 3$) [15]
  - RGB-D ($D = 4$) [5]
- 3D inputs
  - Gray ($D = \mathrm{d}$) [8]

CONVOLUTIONAL LAYER $(H'' \times W'' \times K \times N)$



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

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

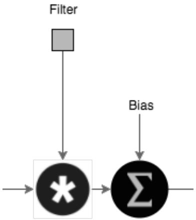
input $x \in \mathbb{R}^{H \times W \times D}$

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

Input

x[:,:,1]

x1 0	x0 0	x0 0	0	0	0	0
x0 0	x1 1	x0 2	2	1	2	0
x0 0	x0 2	x1 1	1	2	3	0
0	2	0	0	1	1	0
0	2	2	1	2	2	0
0	2	1	3	1	1	0
0	0	0	0	0	0	0

x[:,:,2]

x0 0	x1 0	x0 0	0	0	0	0
x0 0	x1 1	x0 3	2	1	1	0
x1 0	x0 2	x0 0	0	2	1	0
0	2	3	0	0	1	0
0	1	2	0	1	1	0
0	1	1	3	2	0	0
0	0	0	0	0	0	0

x[:,:,3]

x0 0	x1 0	x0 0	0	0	0	0
x0 0	x1 1	x0 2	2	0	1	0
x0 0	x1 1	x0 1	1	2	1	0
0	0	1	1	1	0	0
0	0	1	1	0	2	0
0	0	0	0	1	1	0
0	0	0	0	0	0	0

Filter

f[:,:,1,1]

1	0	0
0	1	0
0	0	1

f[:,:,1,2]

1	0	0
0	1	0
0	1	0

f[:,:,2,1]

0	1	0
0	1	0
1	0	0

f[:,:,2,2]

0	1	0
0	1	0
0	1	1

f[:,:,3,1]

0	1	0
0	1	0
0	1	0

f[:,:,3,2]

1	0	0
0	1	0
0	0	0

bias

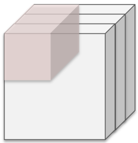
-6

b[:,:,1]

Output

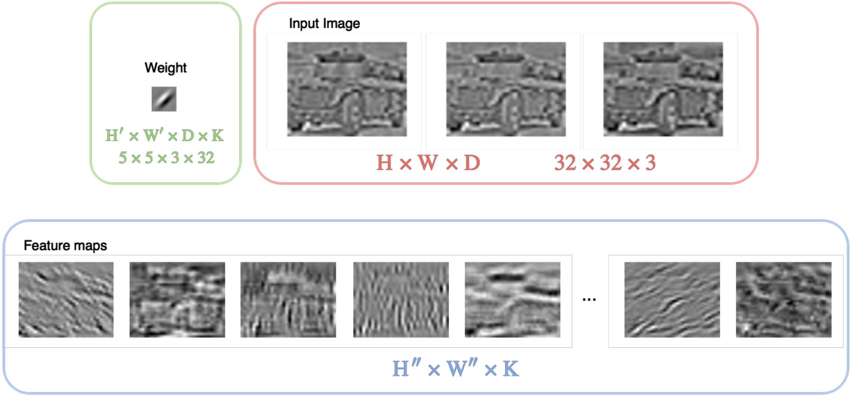
-1		

y[:,:,1]



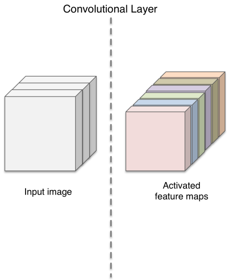
Example: CIFAR-10 (Convolution, $5 \times 5 \times 3 \times 32$)

Keywords: Translation Invariance, few parameters, local consistency



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

ACTIVATION LAYER $(H \times W \times D \times N)$

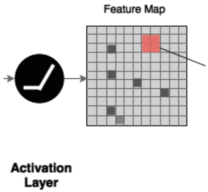


It computes the Rectified Linear Unit (ReLU) of each feature channel x as follows

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

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

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



Input

-1	3	1
3	2	3
-2	3	1

x[:,:,1]

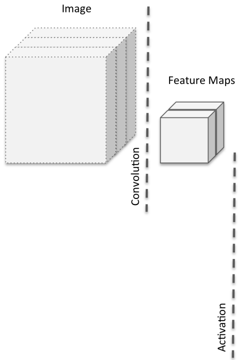
2	4	3
6	0	5
-1	4	0

x[:,:,2]

Output

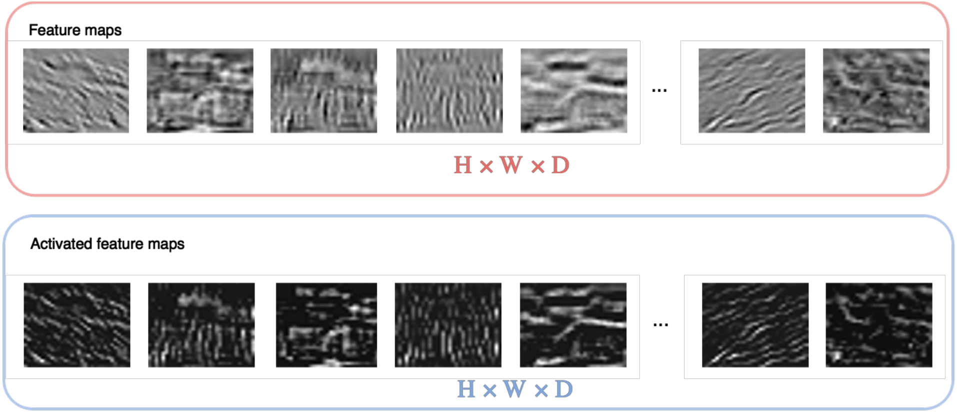
y[:,:,1]

y[:,:,2]



Example: CIFAR-10 (ReLU)

Keywords: Simplifies Back-propagation, Makes Learning faster.



What about other activation functions? Any potential drawbacks?

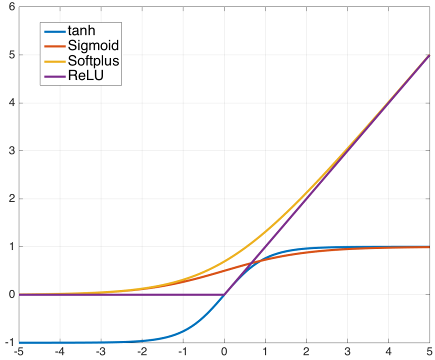


Figure: Activation functions

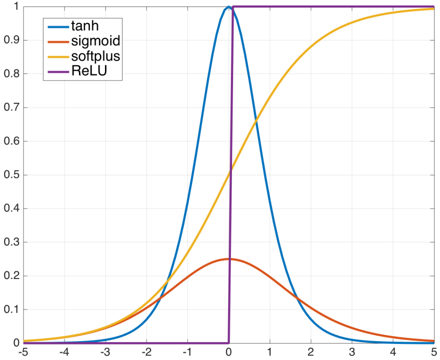
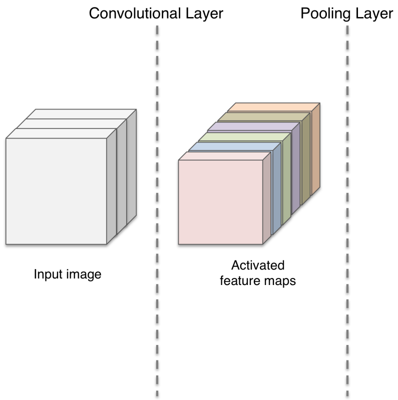


Figure: Activation derivatives

POOLING LAYER $(H'' \times W'' \times D \times N)$



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

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

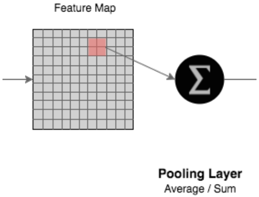
$$y_{i,j,d} = \frac{1}{H'W'} \sum_{1\leq h\leq H', 1\leq w\leq W'} x_{i+h,j+w,d},$$

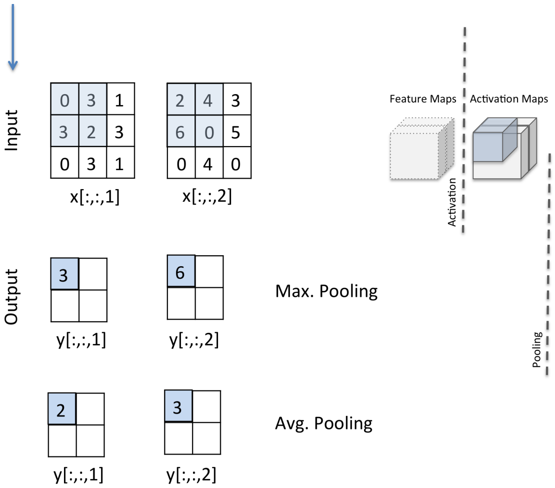
input $x \in \mathbb{R}^{H\times W\times D}$

patch $p \in \mathbb{R}^{H'\times W'}$

output $y \in \mathbb{R}^{H''\times W''\times D}$

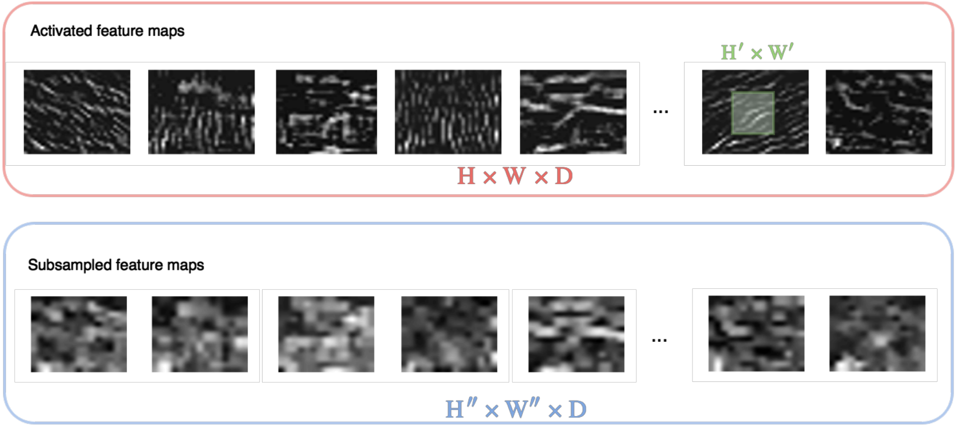
stride $S_{W,H}$ and padding $P_{W,H}$




Input
0 3 1
3 2 3
0 3 1
x[:,:,1]
2 4 3
6 0 5
0 4 0
x[:,:,2]
Feature Maps
Activation Maps
Activation
Pooling
Output
3
y[:,:,1]
6
y[:,:,2]
Max. Pooling
2
y[:,:,1]
3
y[:,:,2]
Avg. Pooling

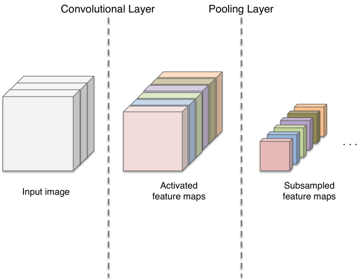
Example: CIFAR-10 (Max. Pooling, $p = 3 \times 3$, $S = 2$)

Keywords: Invariance to small transformation, Larger receptive field



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

NORMALIZATION LAYER ($H \times W \times D \times N$)



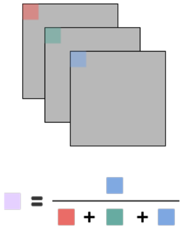
It performs a cross-channel normalization at each spatial location as follows

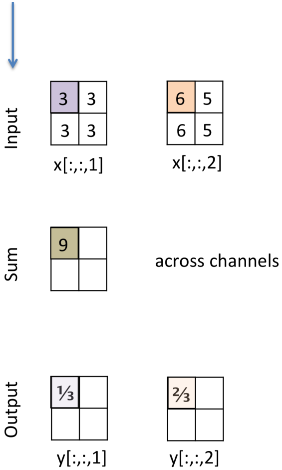
$$y_{i,j,d} = x_{i,j,d}\left(\kappa + \alpha \sum_{d \subset D} x^2_{i,j,d}\right)^{-\beta},$$

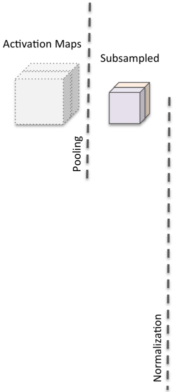
where κ, α, β are hyperparameters. It is usually called Local Response Normalization (LRN).

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

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

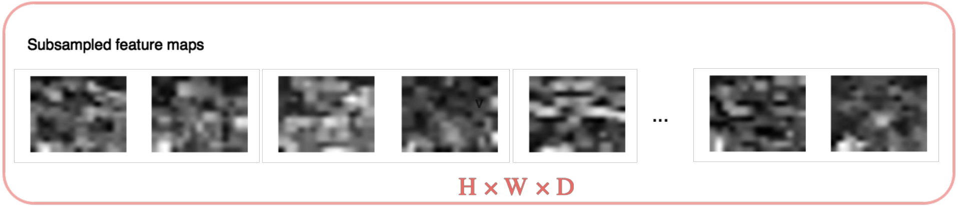


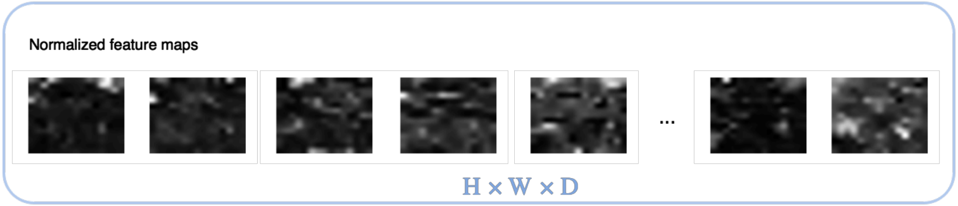

Input
3 3
3 3
x[:,:,1]
6 5
6 5
x[:,:,2]
Sum
9
across channels
Output
⅓
y[:,:,1]
⅔
y[:,:,2]


Activation Maps
Subsampled
Pooling
Normalization

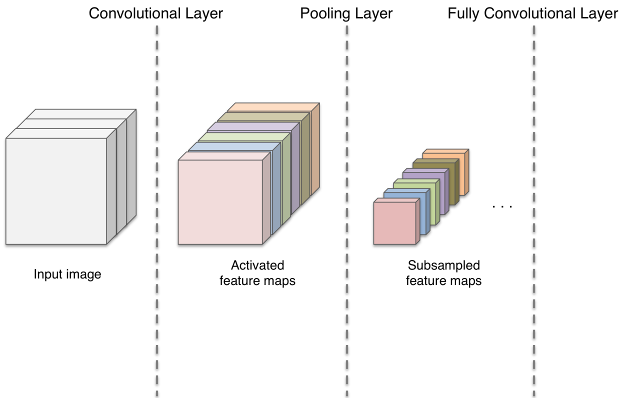
Example: CIFAR-10 (LRN, $\kappa = 0, \alpha, \beta = 1$)

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





FULLY CONNECTED LAYER $(1 \times 1 \times K \times N)$



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

Input

⅓	⅓
⅓	⅓

x[:,:,1]

⅔	⅔
⅔	⅔

x[:,:,2]

filter

3	0
3	0

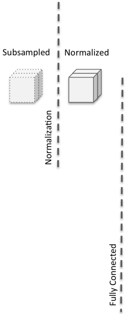
f[:,:,1,1]

0	6
3	3

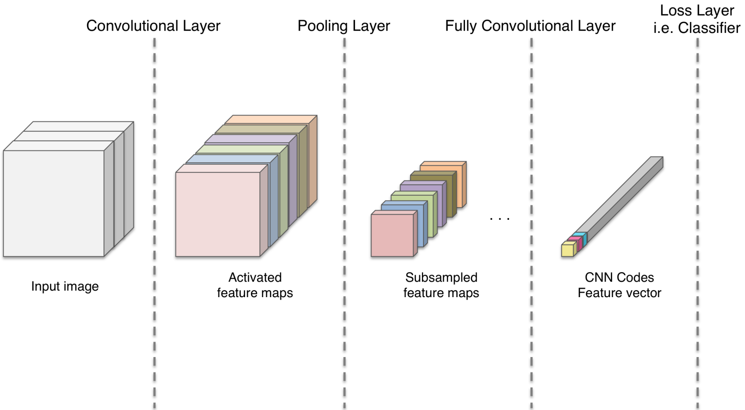
f[:,:,2,1]

Output

y[:,:,1]



bias = [-2, -3, -5, -2]


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

TRAINING CONVNETS

OBJECTIVE FUNCTION

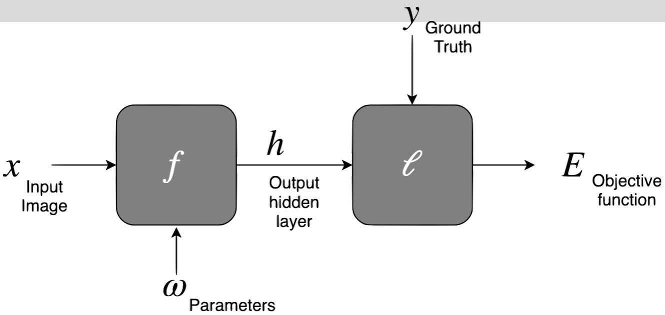
What we have presented so far is the feed-forward propagation, however, to minimize our objective function, we need to propagate back the gradients and update the parameters.

The Objective function:

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

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

OPTIMIZATION AND DERIVATIVES



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

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

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[16], 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.

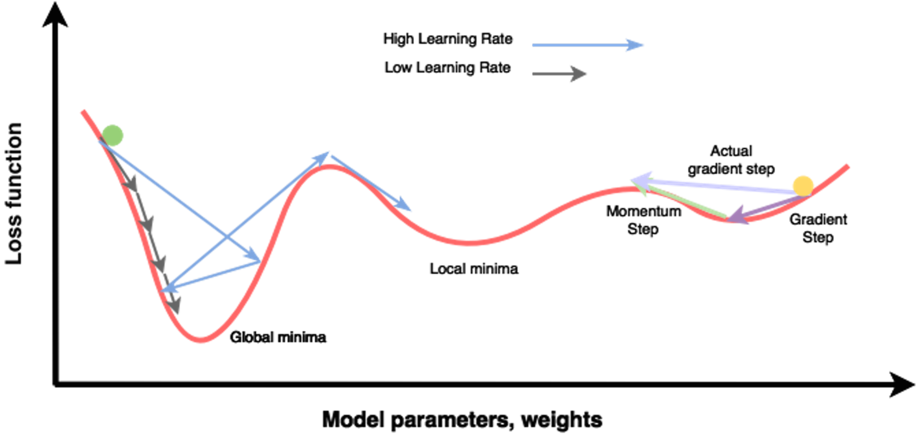


Figure: SGD & Learning Rate[1]

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

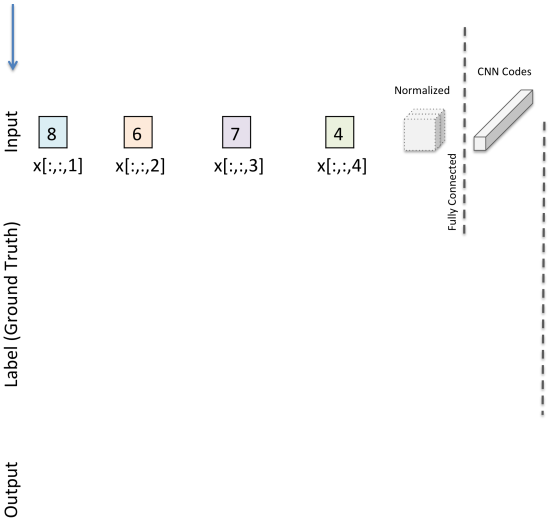
LOSS LAYER $(1 \times 1 \times C \times N)$

The loss function ℓ, mainly used in the training phase, is the cross entropy loss for "classification purpose"

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

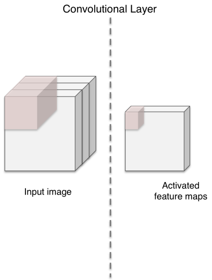
or ℓ_2-norm for "regression purpose" as follows

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


Input
8
6
7
4
x[:,:,1]
x[:,:,2]
x[:,:,3]
x[:,:,4]
Normalized
Fully Connected
CNN Codes
Label (Ground Truth)
Output

WHAT WE LEARNED?

RECAP



LOW/MID/HIGH LEVEL FEATURES

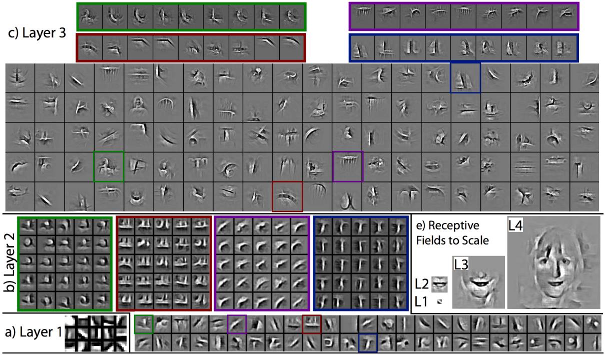


Figure: Low and Mid Level Features, Fig.5 in[22]

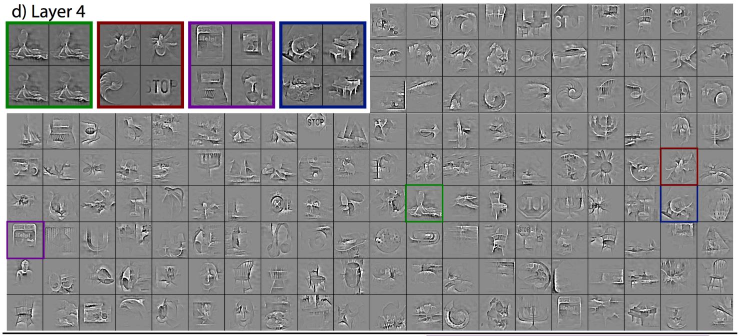


Figure: High Level Features, Fig.5

INTERACTIVE EXAMPLE

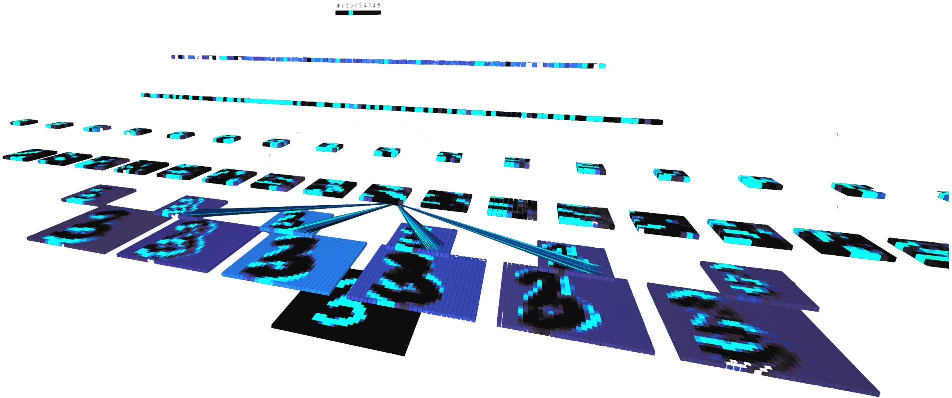


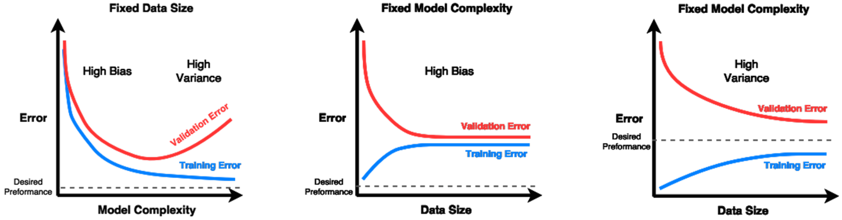
Figure: LeNet5 Architecture, MNIST-10[2]

[2] https://adamharley.com/nn_vis/cnn/2d.html

CONVNETS DEBUGGING

NETWORK TRAINING

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



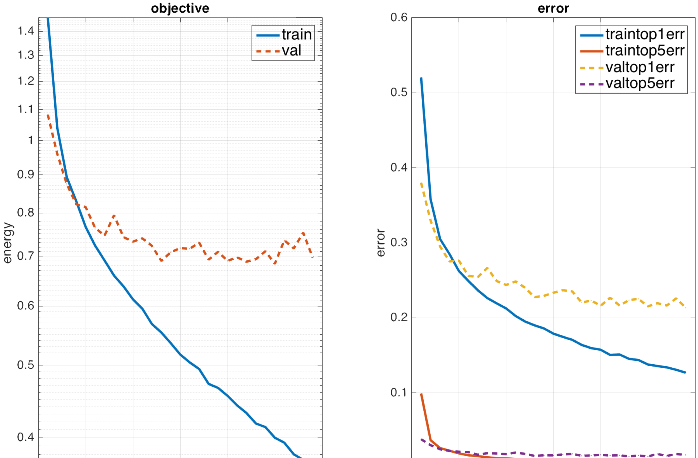
How to fix High Bias? High Variance? [11]

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.

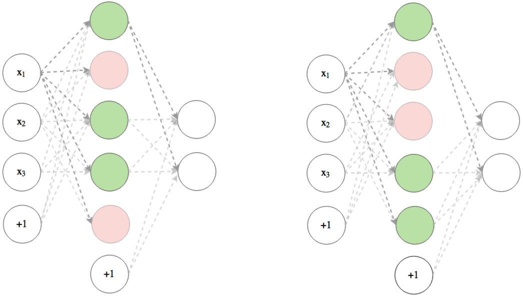
NETWORK TRAINING

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



DROPOUT LAYER $(1 \times 1 \times C \times N)$ [17]

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.



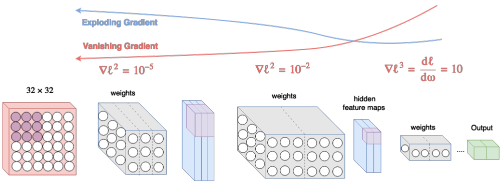
NETWORK DEBUGGING

Gradient Checks

One of the major problems with training a CNN deep model is vanishing/exploding gradient [2].

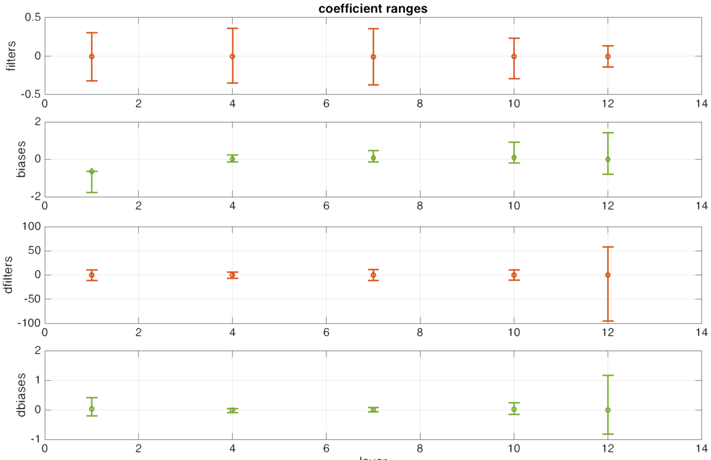
Monitor gradient and activation across layers and epochs.

Try adding Batch Normalization layer, proper weight initialization [9].



NETWORK DEBUGGING

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

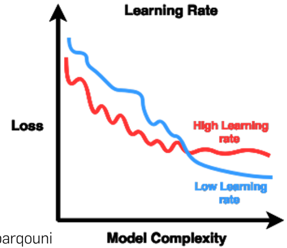


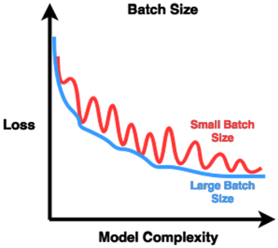
NETWORK DEBUGGING

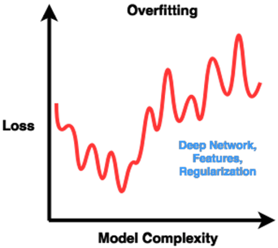
Sanity Checks

- 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

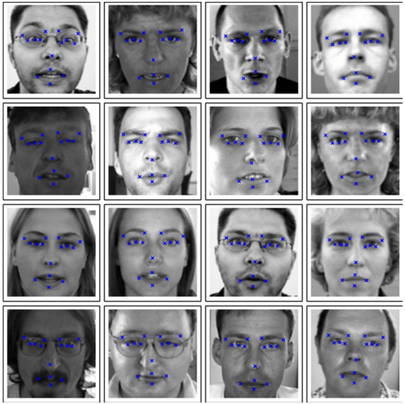
- Deconvolutional Layer [22, 21]
- Batch Normalization [6]
- DropConnect [20]

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.

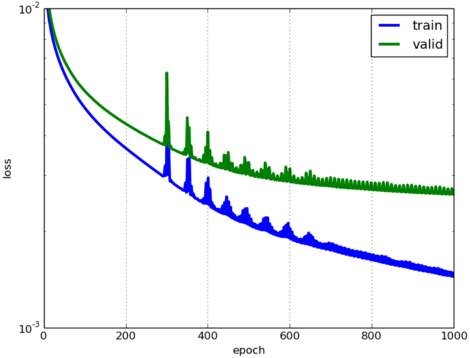


Note: Image Courtesy of this example at [13], Facial keypoint challenge[7].

EXAMPLE: FACIAL KEYPOINTS TUTORIAL (CONT.)

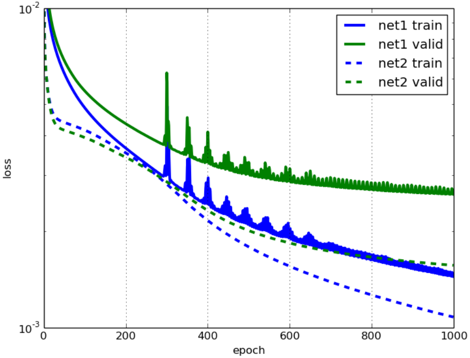
One layer network (net1)

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



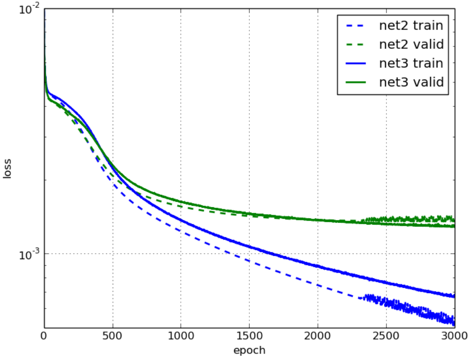
LeNet5 network (net2)

Network: Input, $(\text{Conv, maxPool})^3 + \text{FC}^2$, Output
Parameters: Number of Epochs = 1000.



LeNet5 network (net3)

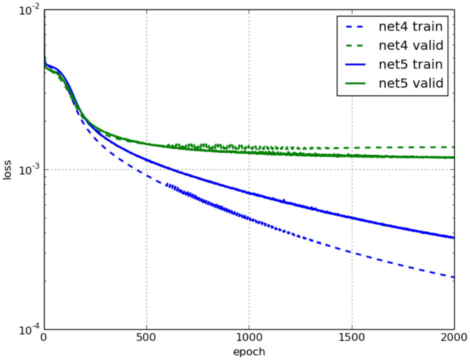
Data Augmentation, only flipping 50% of datasets.
Parameters: Number of Epochs = 3000.



LeNet5 network (net4, net5)

Parameters: Learning Rate = 0.03-0.0001, Momentum = 0.9 - 0.999

with/without Data Augmentation



TRANSFER LEARNING

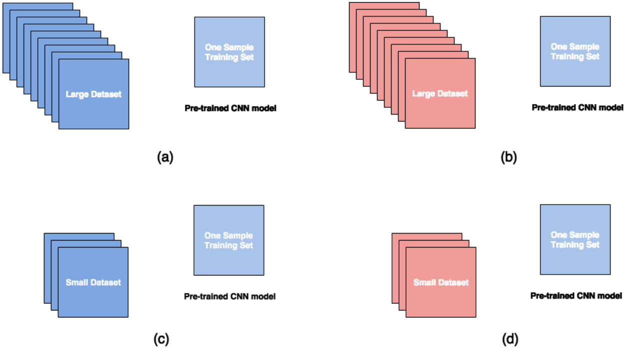
TRANSFER LEARNING

Learning from scratch. Inspired by some CNN architecture, you can design your own network. However, you need tons of data.

Transfer Learning[10]. Once you don't have enough data, you can use the pre-trained CNN models 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.

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.

NETWORK PERFORMANCE

HYPER-PARAMETERS: ADDITIONAL TOPICS

- Optimization solver[3, 12].
- Learning Rate Schedule[1]: 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[18]
- Batch Size: between 10 and few hundreds.
- Weight Initialization[19].

REFERENCES

Yoshua Bengio.
Practical recommendations for gradient-based training of deep architectures.
In Neural Networks: Tricks of the Trade, pages 437–478. Springer, 2012.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi.
Learning long-term dependencies with gradient descent is difficult.
Neural Networks, IEEE Transactions on, 5(2):157–166, 1994.

Léon Bottou.
Stochastic gradient descent tricks.
In Neural Networks: Tricks of the Trade, pages 421–436. Springer, 2012.

David Eigen and Rob Fergus.
Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture.
arXiv preprint arXiv:1411.4734, 2014.

Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik.
Learning rich features from rgb-d images for object detection and segmentation.
In Computer Vision–ECCV 2014, pages 345–360. Springer, 2014.

Sergey Ioffe and Christian Szegedy.
Batch normalization: Accelerating deep network training by reducing internal covariate shift.
arXiv preprint arXiv:1502.03167, 2015.

Kaggle.
Facial keypoints detection, May 2013.

Konstantinos Kamnitsas, Liang Chen, Christian Ledig, Daniel Rueckert, and Ben Glocker.