Deep Learning for Visual Computing

CNN Training and Inference, Object Detection

Christopher Pramerdorfer
Computer Vision Lab, TU Wien

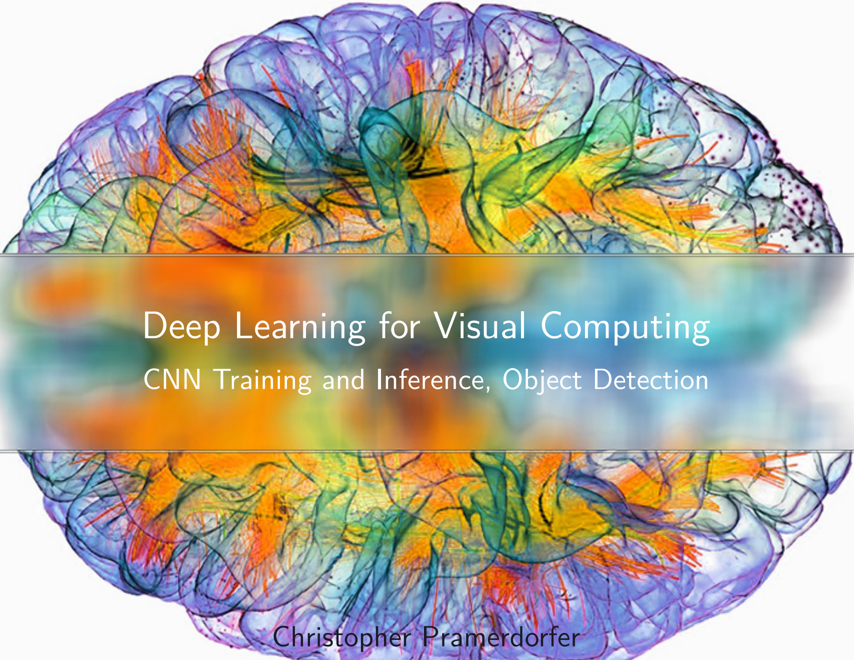


Image by Neuroscape Lab

Topics

CNN training in practice

CNN inference in practice

CNN classification performance

Object detection using CNNs

CNN Training

Recall how CNNs are trained

- Loss function $L(\boldsymbol{\theta})$ (cross-entropy loss)
- Minibatch gradient descent & backpropagation
- Regularization (early stopping, weight decay)

Best way to improve generalization : train on more data

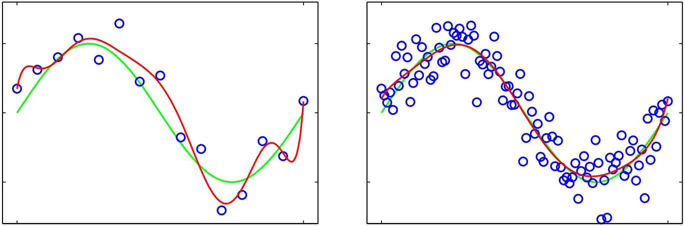


Image adapted from [1]

Best way to improve generalization : train on more data

- But in practice data is limited

Can get around problem by creating meaningful fake data

- Approach called (training) data augmentation

Straight-forward to do in image classification

- Apply transformations that have no effect on class label

Image adapted from youtube.com

Can be done online, no need to store transformed samples

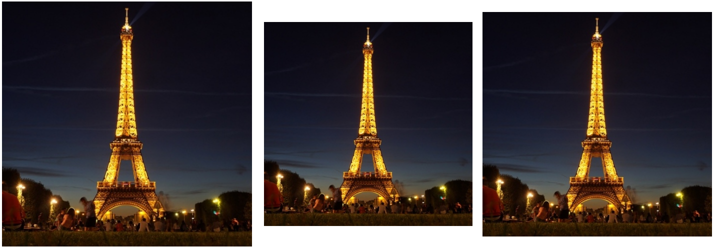
- Apply transformations during minibatch generation

Common image transformations

- Random scaling
- Random cropping
- Horizontal mirroring with probability 0.5

CNN Training

Data Augmentation – Random Scaling



CNN Training

Data Augmentation – Random Cropping

CNN Training

Data Augmentation – Horizontal Mirroring

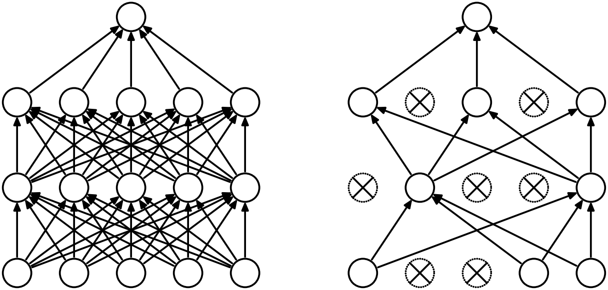
Set neuron output to 0 with probability p during training

- Decided independently for every sample

Has effect of temporarily discarding neurons

- No effect on output of next layer (conv, fc, pool)

CNN Training
Dropout



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

Image from [2]

Effective regularization strategy

- Net learns not to rely on certain neurons / features
- Works well in conjunction with weight decay

Usage examples / common usage

- VGGNet : $p = 0.5$ for MLP hidden layers
- GoogLeNet : $p = 0.4$ for final pooling layer

Dropout only used at training time

- At test time all neurons are active

Dropout has effect of training many different nets

- And averaging their predictions at test time
- Related to model ensembles (see below)

Recall the importance of

- Weight initialization (sample from $\gamma\,\text{Norm}(0{,}1)$)
- Input normalization (to mean 0 and variance 1)

We selected γ in attempt to preserve input variance

- Strength of signals preserved as they pass through net

Input variance only preserved initially

- Parameters change during training
- Thus output distribution changes over time

This complicates training

- Must account for changes in input distribution
- Layer input affected by parameters of all previous layers

Batch normalization reduces this problem

- Compute per-feature input mean and variance
- Using current minibatch to approximate training set
- Normalize every feature in minibatch

In practice it is better to normalize activations

- Recall that neurons compute $n(\mathbf{a}^\top \mathbf{x} + b)$
- Recall that scalar $\mathbf{a}^\top \mathbf{x} + b$ is called activation

Batch normalization can reduce model capacity

- Restricts range of inputs to non-linearity n
- Solved by scaling normalized activations (see [3] for details)

During inference we generally have no minibatches

- Use (stored) statistics from training set

For convolutional layers

- Apply same normalization to all neurons in same feature map

Improves robustness to bad initialization

Permits higher learning rates

- Learning rate of 0.1 common in practice

Regularizing effect

- Example seen together with other samples during training
- Output depends on both sample and minibatch

Use batch normalization everywhere

- Apply to all conv and fc layers
- By adding batch normalization layer before non-linearity

Shuffle training set before every epoch

- Increases regularizing effect of batch normalization

CNN Training
Suggestions

Use weight decay for regularization

- Tune global regularization strength δ on validation data

If network still overfits, use dropout

- See above slides for usage suggestions
- Tune p on validation data

Set initial learning rate correctly

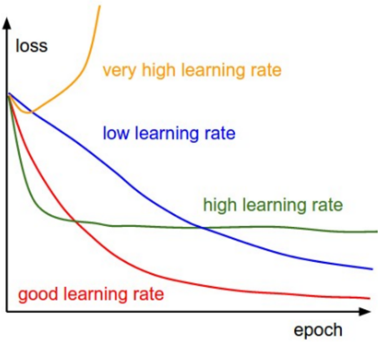


Image from cs231n.github.io

CNN Training
Suggestions

Use (Nesterov) momentum

- Momentum of $\beta = 0.9$ often used in practice

Lower learning rate α based on validation accuracy

- E.g. $\alpha = \alpha/10$ if accuracy does not improve for several epochs

Use ReLU non-linearity everywhere

- Speeds up training

Initialize weights from $\gamma\,\mathrm{Norm}(0,1)$ with $\gamma = \sqrt{2/D}$

- Intuition explained in Lecture 6 (now optimized for ReLU)
- Sometimes called He initialization [4]

CNN Training
Suggestions

Normalize samples to mean 0 and variance 1

- Using statistics from training set
- On per-channel or per-feature basis (Lecture 6)

Use data augmentation

- Ensure that task is invariant to transformations

CNN Inference

Two common strategies for improving performance

- Oversampling
- Model ensembles

Optimal performance achieved by utilizing both

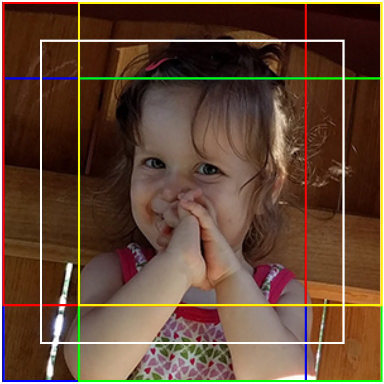
Process input image multiple times

- Predict class-scores of transformed versions
- Average class-scores

Common strategy : ten-crop oversampling

- Crop input image at center and corners
- Process crops and mirrored versions (10 images)

CNN Inference
Oversampling



CNN Inference
Oversampling

Can use any strategy

- E.g. transformations used for data augmentation
- Ensure that transformation have no effect on labels

Virtually free in terms of processing speed

- Can process transformed versions as single batch

Train several CNNs

- Different initial weights, data (augmentation), architectures

Let each predict class-scores and average them

- Intuition : models don't make same mistakes

Most common approach

- Same architecture
- Different weights and data (random augmentation)

Ensemble size usually at most 10

- Diminishing returns in terms of performance
- Runtime increases linearly with ensemble size

CNN Classification Performance

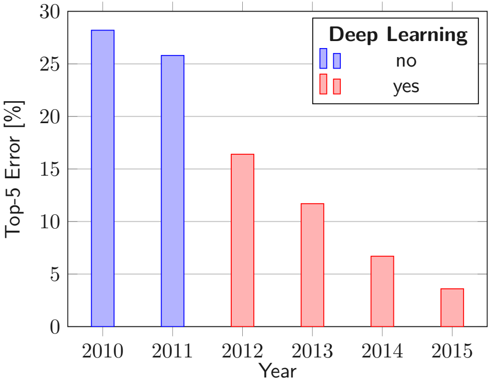
State of the art in virtually every classification task

- As long as there is sufficient data available
- Rule of thumb : at least 5000 samples per class

Human-like performance on some datasets

CNN Classification Performance
LSVRC Challenge

1000 classes, 1.4 million images



CNN Classification Performance
Human Vision

Humans are the best image classifiers

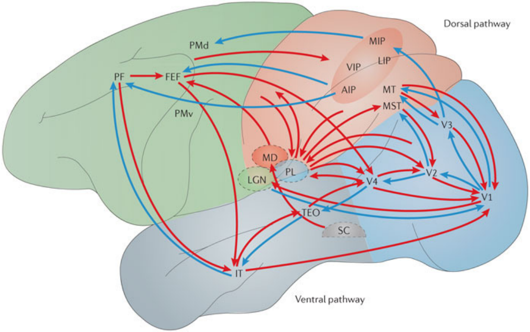


Image from nature.com

CNN Classification Performance
Human Vision

Images from the eyes are first processed in V1, where

Simple cells

- Respond to small specific part (receptive field) in image
- Response similar to linear function of this part

Complex cells

- Similar to simple cells
- Invariant to small shifts in position (pooling)

CNN Classification Performance
Human Vision

V1 neurons are similar to Gabor filters

- Respond to brightness changes
- At specific frequencies and orientations

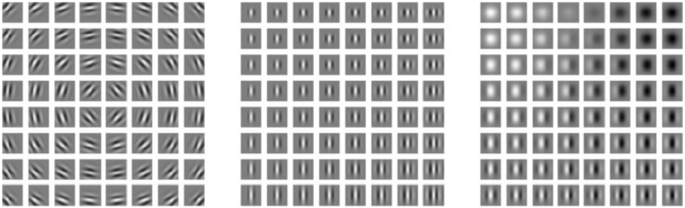


Image from [5]

CNN Classification Performance
Human Vision

These neurons are connected to each other over many levels

- Information passes over V2 and V3 to IT

The deeper we go, the more specific neurons become

- Cells that respond to certain faces
- High invariance (independent of location, lighting ...)

Similar to how CNNs work

- Conv and pooling neurons similar to simple and complex cells
- Neurons are connected over many levels

CNNs learn to respond to similar concepts

CNN Classification Performance
CNN Vision

First conv layer usually learns Gabor-like filters

- Evolution needed millions of years to figure out good filters
- CNNs need a couple minutes

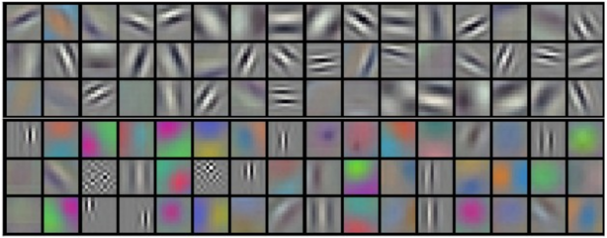


Image from cs231n.github.io

Later conv layers learn to respond to more specific concepts

- The high-level features we want

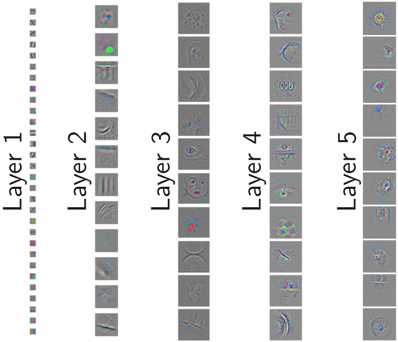


Image adapted from [6]

CNN Classification Performance
CNN Vision

Learned features generalize well to similar domains

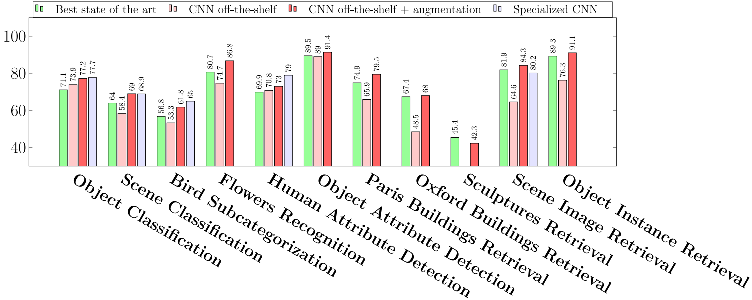


Image from [7]

CNN Classification Performance

CNN Vision

CNN features well-suited for transfer learning

- Pretrain CNN on some (large) dataset
- Finetune CNN on other (small) dataset

Allows use of Deep Learning with small datasets

- Pretrain on ImageNet for general features (previous slide)
- Or pretrain on more specific related data

Object Detection

So far we've covered only image classification

Let's consider a related task called object detection

- Multiple objects, possibly of different class
- Need to locate objects (of different classes) in image

Two popular approaches

- Sliding window
- Region proposals

Object Detection

Face detection is a popular example

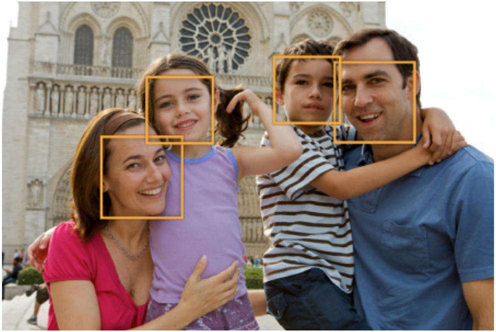


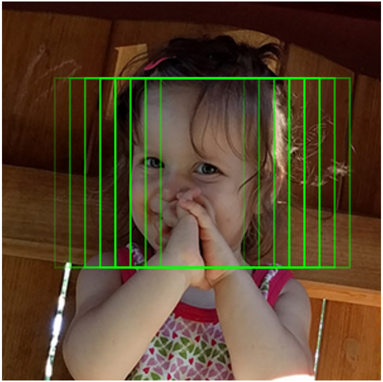
Image from apple.com

Training

- Train (or finetune) CNN for classification with $T + 1$ classes
- Usually additional "background" class (softmax)

Detection

- Slide fixed-size window over image
- Predict class-scores for every window
- Perform non-maximum suppression



Inefficient

- Many windows to classify

Single fixed-size window (no scale invariance)

- Must process image at multiple scales
- Even more inefficient

Apply some region proposal algorithm to image

Classify only these proposals (after warping to common size)

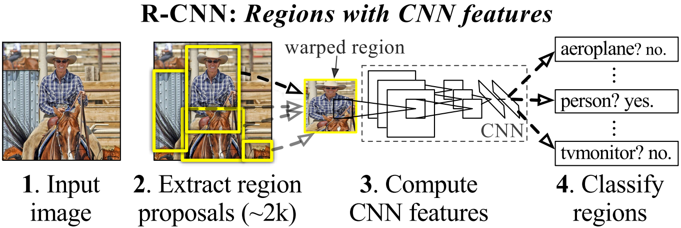


Image from [8]

Approach called R-CNN (R stands for regions)

Advantages

- Fewer proposals than windows (more efficient)
- Multiple scales and aspect ratios

R-CNN is still quite slow

- Many proposals to classify

Newer works (Fast/Faster R-CNN) overcome this problem

- Process whole image once
- Classify using CNN features in proposal regions

More on object detection in next lecture

Bibliography I

[1] C. M. Bishop, *Pattern Recognition*, , 2006.

[2] *Dropout: A simple way to prevent neural networks from overfitting.* JMLR, 2014.

[3] *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*, CoRR, 2015.

[4] *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification*, ICCV, 2015.

[5] *Deep learning*, 2016, [Online]. Available: http://www.deeplearningbook.org.

Bibliography II

[6] *Visualizing and understanding convolutional networks*, ECCV, 2014.

[7] *CNN Features off-the-shelf: an Astounding Baseline for Recognition*, 2014.

[8] *Rich feature hierarchies for accurate object detection and semantic segmentation*, 2014.