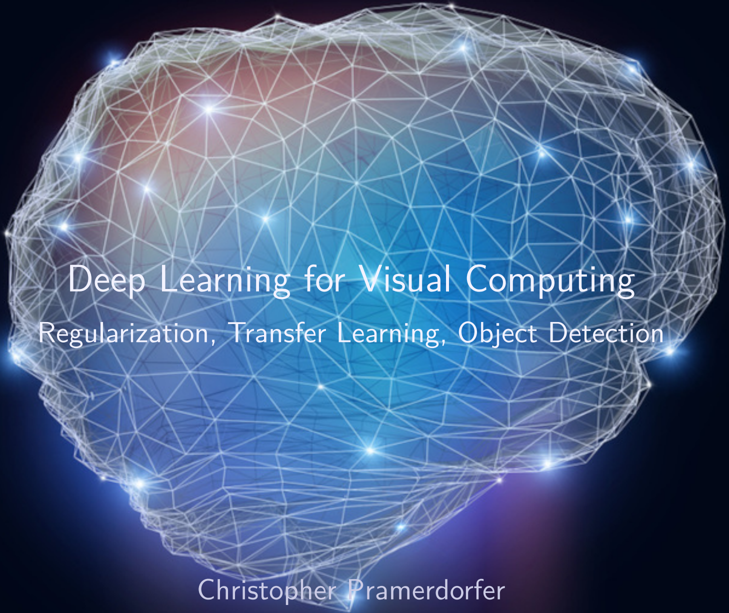
Deep Learning for Visual Computing

Regularization, Transfer Learning, Object Detection



Christopher Pramerdorfer

Computer Vision Lab, TU Wien

Image by Oracle

Topics

Optimization vs. Machine Learning

- Data augmentation
- Regularization

Transfer Learning

Object detection (part 1)

Optimization vs. Machine Learning

We covered training from an optimization perspective

- Find parameters that minimize training loss
- Known as empirical risk minimization

Prone to overfitting

- Training data must capture underlying distribution well
- Almost never the case in image analysis

Typical example of overfitting

- Training loss decreases steadily
- Validation loss begins to rise again at some point

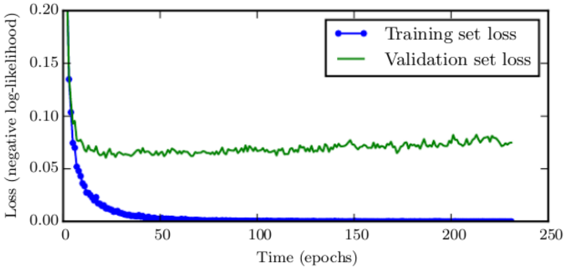


Image from [5]

So optimization was successful (loss ≈ 0)

- But disappointing validation/test performance
- Due to inability to generalize well to unseen data

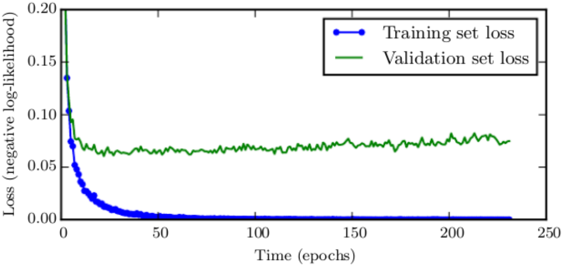


Image from [5]

So in ML and DL our goals are actually

- Low training loss (avoid underfitting)
- Small gap to validation loss (avoid overfitting)

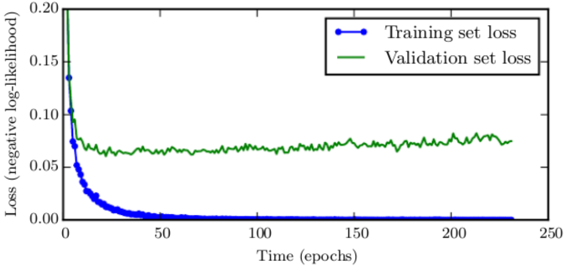


Image from [5]

Optimization vs. Machine Learning
Underfitting vs. Overfitting

To achieve these goals we

- Minimize the training loss (we have to)
- While also monitoring the validation loss/performance

And combat overfitting via

- Data augmentation
- Regularization
- Early stopping

Best way to improve generalization : train on more data

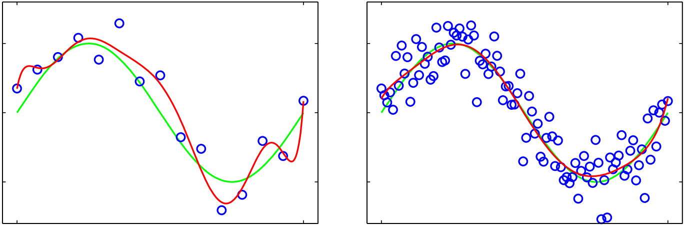


Image adapted from [1]

Best way to improve generalization : train on more data

- But in practice data are limited

Can get around problem by creating meaningful fake data

- Based on the available training data
- Approach called (training) data augmentation

Straight-forward to do in image classification

- Apply image transformations that have no effect on class

Image adapted from youtube.com

Optimization vs. Machine Learning
Data Augmentation

Can be done online, no need to store transformed samples

- Apply transformations during minibatch generation

Common image transformations

- Random scaling (and cropping)
- Random cropping
- Horizontal mirroring with probability 0.5

Useful if samples are larger than $H_0 \times W_0$

- Facilitates scaling and cropping

The purpose of regularization is to

- Improve the validation/test performance
- At the possible expense of training performance

Often by decreasing the model's variance

- Sensitivity to small changes in the training set
- Thus combatting overfitting

Optimization vs. Machine Learning

Regularization – Weight Decay

Weight Decay (L2 weight regularization) is very common

- Almost always used in practice

Penalizes large weights (not biases)

- Preventing certain inputs from dominating output
- Thus encourages model to use all inputs

Implemented by adding regularization term to loss function

$$L_{\mathsf{reg}}(\boldsymbol{\theta}) = \frac{\delta}{2}\|\mathbf{w}\|^2 + L(\boldsymbol{\theta})$$

$\mathbf{w} \subset \boldsymbol{\theta}$ is vector of all weights

$\delta \in (0, 1)$ controls amount of regularization

- Usually $\delta \in [0.000001, 0.001]$

Ignoring bias, resulting gradient is $\nabla L_{\text{reg}}(\mathbf{w}) = \delta\mathbf{w} + \nabla L(\mathbf{w})$

- $\|\mathbf{w}\|^2 = \mathbf{w}^\top\mathbf{w}$, so gradient is $2\mathbf{w}$ (product rule)

So gradient descent update becomes

$$\begin{aligned} \mathbf{w} &= \mathbf{w} - \alpha(\delta\mathbf{w} + \nabla L(\mathbf{w})) \\ &= \mathbf{w} - \alpha\delta\mathbf{w} - \alpha\nabla L(\mathbf{w}) \\ &= (1 - \alpha\delta)\mathbf{w} - \alpha\nabla L(\mathbf{w}) \end{aligned}$$

Weights are shrinked by constant factor before each update

- Thus weights decay to zero (hence the name)

Decay strength δ is another hyperparameter

- No effect if too small
- Dominates data loss (e.g. cross-entropy) if too large

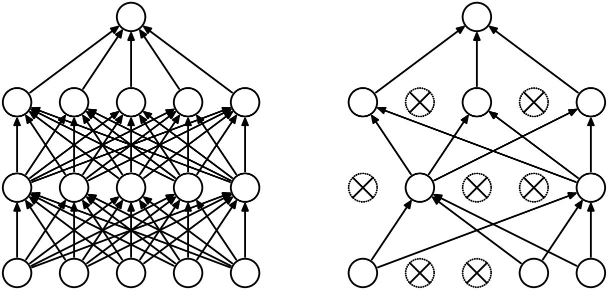
Dropout is a layer type whose neurons

- Output 0 with probability p
- Forward input with probability $1 - p$

Usually placed before last (dense) layer

- Has effect of temporarily "discarding" neurons
- As neuron has no effect on output on next layer
- Thus net learns not to rely on certain neurons

Optimization vs. Machine Learning
Regularization – Dropout



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

Image from [2]

Early Stopping aims to minimize overfitting by

- Storing a copy of $\boldsymbol{\theta}$ with lowest validation loss v
- Stopping if v does not improve anymore for e epochs

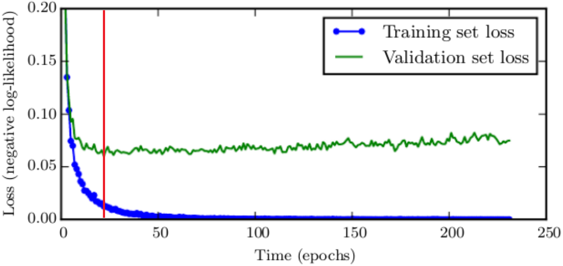


Image adapted from [5]

Optimization vs. Machine Learning
Suggestions

Data

- Use as much data as you can get
- And utilize (sensible) data augmentation

Regularization

- Always use Weight Decay and tune δ
- Consider dropout if the model still overfits

Always use early stopping and with $e > 1$

Transfer Learning

Dataset size is still critical

- Above techniques can't replace lack of data

Amount of data we need varies with task

- In image classification at least 5k per class

No large datasets available for certain tasks

- Compiling such datasets is time-consuming
- And sometimes unrealistic (e.g. medical data)

Transfer Learning

Transfer Learning addresses this problem

- Improve learning in new task
- By transferring knowledge from related task learned previously

In practice this means

- Utilizing a CNN $\mathbb{M}$ trained to solve some task
- To solve a new related task

What “utilizing” means depends on similarity of tasks

If dataset is small, use $\mathbb{M}$ as feature extractor

- Remove classification layer(s) from $\mathbb{M}$
- Use output vectors of $\mathbb{M}$ to train (linear) classifier

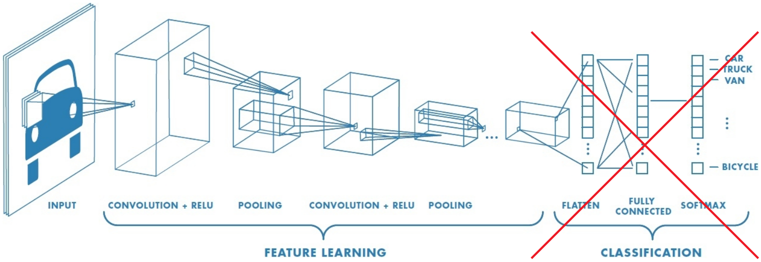


Image adapted from floydhub.com

Transfer Learning
CNN Features

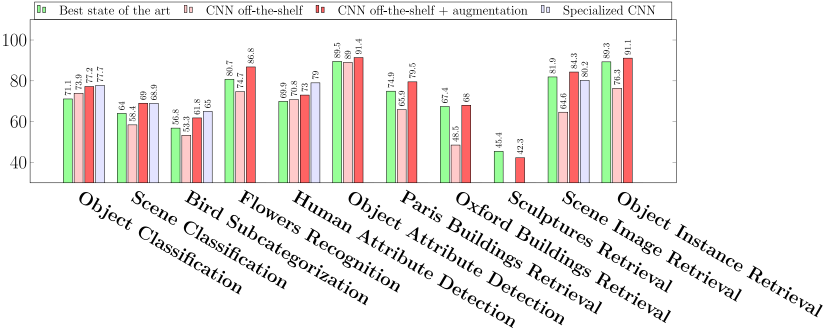


Image from [6]

If tasks are dissimilar, try using more generic features

Use outputs of an intermediate conv layer as features

- Recall earlier conv layers learn more generic features
- Squash 3D outputs to vectors as covered earlier

If dataset is large, fine-tune $\mathbb{M}$

- Replace classification layers
- Train $\mathbb{M}$ as usual

Depending on dataset size and similarity consider

- Using smaller α for conv layers (common)
- Fixing parameters of early conv layers ($\alpha = 0$)

Transfer Learning
Remarks

Always consider fine-tuning of $\mathbb{M}$ trained on ImageNet

- Almost always beneficial
- Such pre-trained models are available online

Make sure $\mathbb{M}$ is compatible with new data

- C_0 must match
- H_0 and W_0 can differ
- But repeated pooling might be problematic

Object Detection

We have focused on Image Classification

- Fundamental Computer Vision task
- We now know how to achieve good performance

So let's consider something more complex – object detection

- Multiple relevant objects of C classes
- Need to locate objects (of different classes) in image

Face detection is a popular example ($C = 1$)

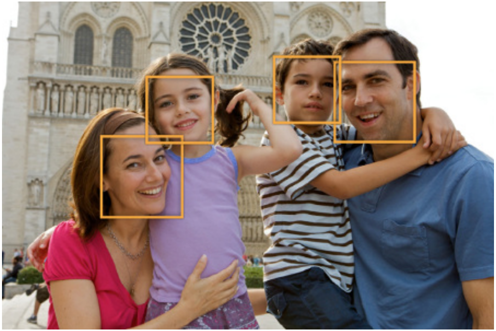
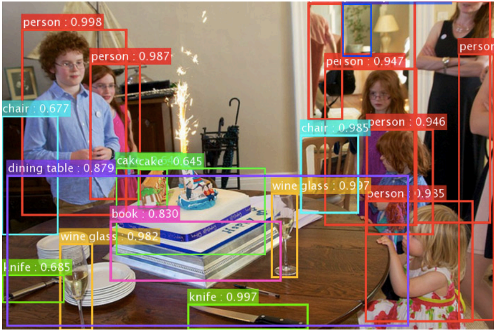


Image from apple.com

Object Detection

Examples

But in general $C > 1$



Training

- Train (fine-tune) classifier for $C + 1$ classes
- Usually additional “background” class (softmax)

Detection

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

Object Detection
Sliding Window Approach

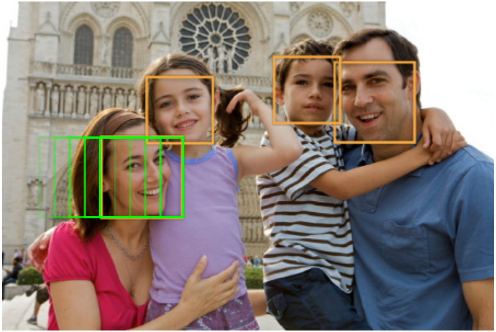


Image adapted from apple.com

Inefficient

- Many windows to classify

Single fixed-size window (no scale invariance)

- Must process image at multiple scales (more inefficient)
- Inaccurate bounding boxes (fixed aspect ratio)

Cannot handle multiple objects in same window (softmax)

R-CNN addresses these limitations

- Classify region proposals instead of sliding windows
- Using C binary SVMs trained on CNN features

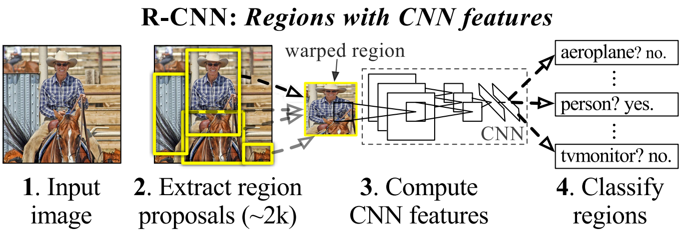


Image from [4]

Training

- Fine-tune CNN to $C+1$ classes using region proposals
- Train C SVMs on CNN features of resulting model

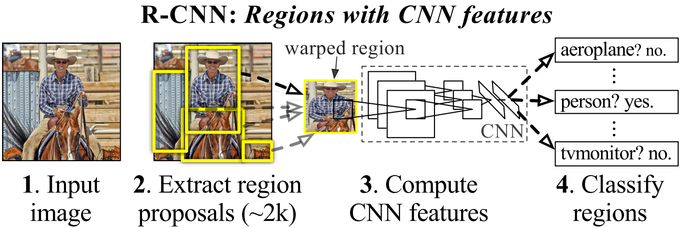


Image from [4]

Region proposals have different size and aspect ratio

- Solves window-related problems
- Just warp to common size $H_0 \times W_0$

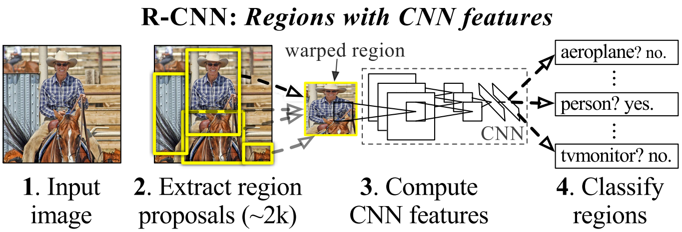


Image from [4]

C independent binary SVMs

- Supports multiple objects per bounding box

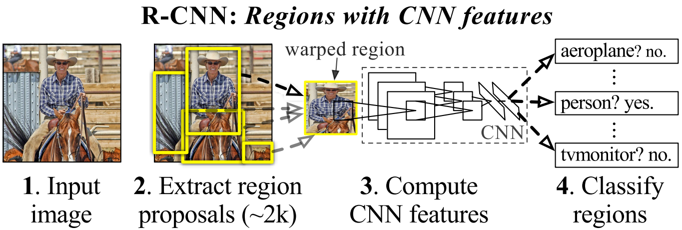


Image from [4]

Inefficient as CNN needs to process many regions

- Fewer than with sliding windows but still a lot

Complex multi-stage training pipeline

- CNN is not improved based on SVM results

Fast R-CNN improves efficiency of R-CNN

- Process complete (high-resolution) image once
- Instead of processing many (smaller) region proposals

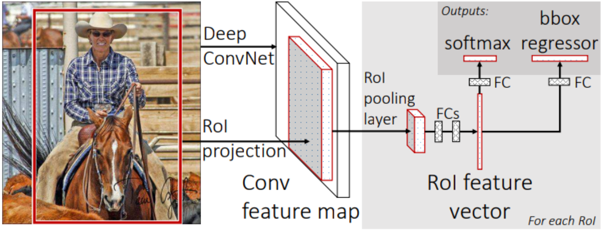


Image from [3]

Use first u layers of some pre-trained CNN

- u depends on architecture (not important here)
- Only conv and pooling layers so input size may vary

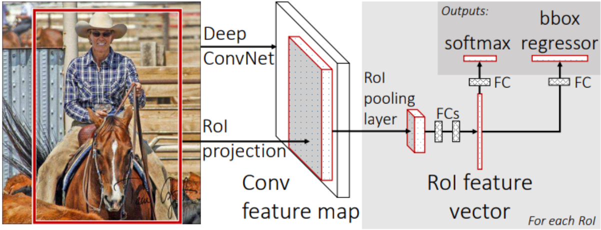


Image from [3]

Then for each region proposal r

- Project bounding box onto output volume
- To obtain RoI with shape $C_u \times H_r \times W_r$

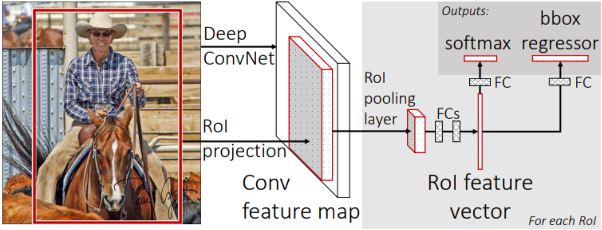


Image from [3]

Object Detection
Fast R-CNN

H_r and W_r vary across region proposals

- Need a way to map to fixed shape $C_u \times H \times W$
- To be able to continue with classifier

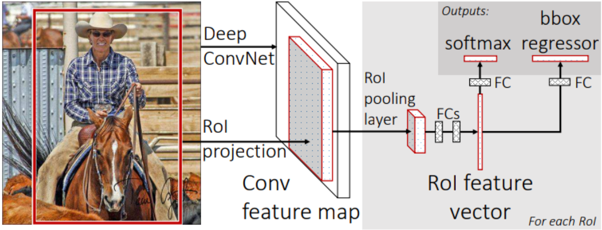


Image from [3]

Object Detection

Fast R-CNN

Achieved using a RoI Pooling layer

- Divide $H_r \times W_r$ window into grid of size $H \times W$
- Max-pool each cell to obtain $C_u \times H \times W$ volume

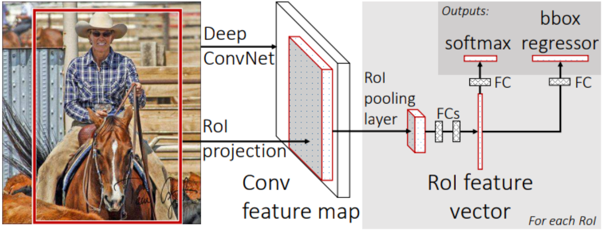


Image from [3]

Classifier is a MLP rather than C SVMs

- Enables fine-tuning the whole network
- Thereby training end-to-end for object detection

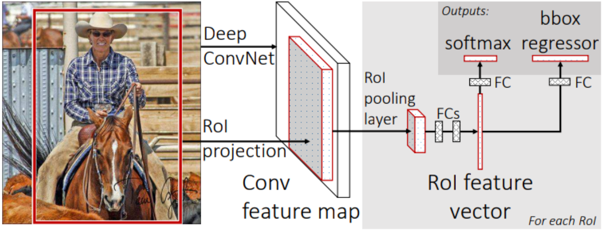


Image from [3]

Also predicts offsets for bounding box refinement

- So two outputs and (cross-entropy and L1) losses
- Check paper for details (R-CNN does this too)

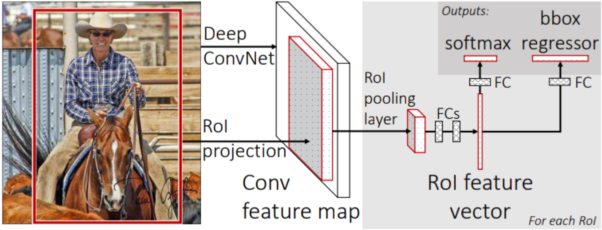


Image from [3]

Object Detection

Now the region proposal algorithm is the bottleneck

- We will address this in next lecture

Bibliography

[1] Christopher M Bishop. *Pattern Recognition*. (2006).

[2] *Dropout: a simple way to prevent neural networks from overfitting.* JMLR (2014).

[3] Ross Girshick. *Fast R-CNN*. CVPR. 2015.

[4] Ross Girshick et al. *Rich feature hierarchies for accurate object detection and semantic segmentation*. CVPR. 2014.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016.

[6] Ali Sharif Razavian et al. *CNN Features off-the-shelf: an Astounding Baseline for Recognition*. CoRR (2014).