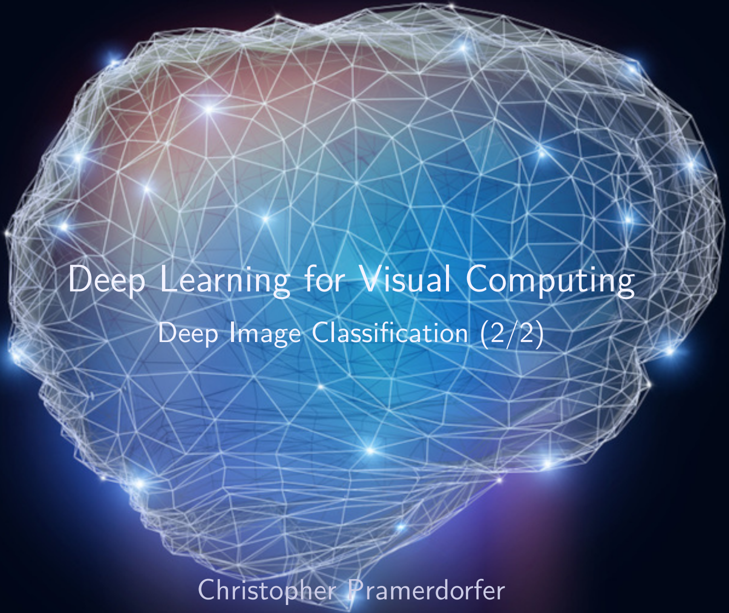


Deep Learning for Visual Computing

Deep Image Classification (2/2)

Christopher Pramerdorfer

Computer Vision Lab, TU Wien

Image by Oracle

Topics

Deep Image Classification (part 2)

- Pooling layers
- Classification backends
- Network design basics
- Residual networks

What CNNs learn

Recall that conv layers

- Retain the input size $W_{l-1} \times H_{l-1}$ (padding)
- Or reduce it only slowly (no padding)

This

- Slows down computations
- Leads to more parameters when combined with linear layers

CNNs thus include some form of pooling

- Reduce W_{l-1} and H_{l-1} via local aggregation
- While leaving D_{l-1} unchanged

Some CNNs also do the opposite (e.g. [3])

- Keep W_{l-1} and H_{l-1} but decrease D_{l-1}
- Using a conv layer with $c = 1$ and $D_l < D_{l-1}$

Most common form is 2×2 max-pooling with stride 2

- $W_l = W_{l-1}/2$, $H_l = H_{l-1}/2$, and $c = 2$
- Output of neuron h is $\max(\mathbf{X}_h)$ with $\mathbf{X}_h \in \mathbb{R}^{2\times2}$

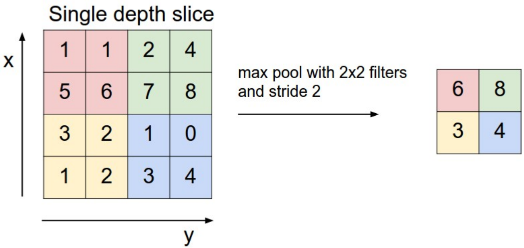
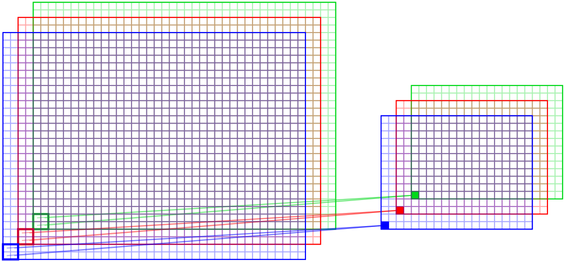


Image from cs231n.github.io

Number of neurons reduced by factor 4

- Corresponding efficiency increase
- At the cost of losing spatial resolution



Can also perform average-pooling

- Compute $\mathrm{mean}(\mathbf{X}_h)$ instead of $\max(\mathbf{X}_h)$

Or a conv layer with stride $s > 1$

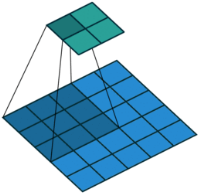


Image from github.com

Conv and pooling layers produce 3D tensors $(W_l \times H_l \times D_l)$

For classification we convert to vectors $\mathbf{x}_l$

- Option 1: flatten tensor like we did with images
- Option 2: global average-pooling with size $W_l \times H_l$

Allows us to connect linear layers, resulting in

- A linear or non-linear (MLP) classifier
- That processes vectors of learned features

Modern architectures often use a linear classifier

- So the learned features are so powerful
- That the simplest classifier is sufficient

We finally have task-specific high-level features!

This concludes the basic layer types and purposes

- Conv + ReLU layers for feature extraction
- Pooling layers for dimensionality reduction
- Linear layers for classification

The question is how to arrange these layers properly

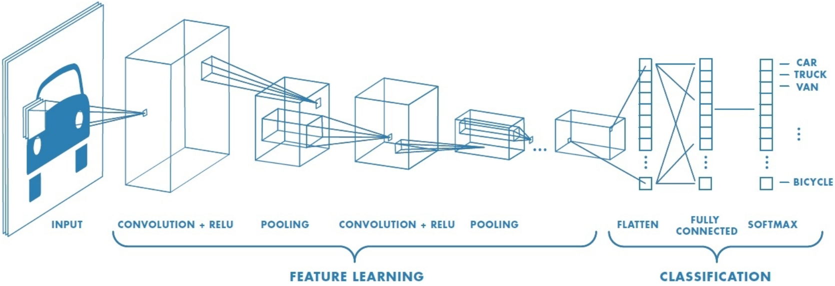


Image from floydhub.com

This is a complex topic

- Most layers have (several) hyperparameters
- Layer arrangement is also a hyperparameter

Cannot just search for good hyperparameters

- Training once can take hours or even days

But tried and true practices exist

- Next slides will introduce a basic design recipe

Use square images, $H_0 = W_0 = R$

- Resize images such that smaller side has size R
- Then extract a center crop of size $R \times R$

R should be divisible by 2 many times

- Avoid problems during pooling

Start with small R

- And test if increasing R makes sense
- $R = 224$ is popular for classification
- But much smaller R might be sufficient

Get R below 100 quickly via aggressive pooling

- To improve efficiency
- $R = 224$: conv with $c = 7, s = 2 \Rightarrow 2 \times 2$ max-pooling

Use conv $\Rightarrow$ conv $\Rightarrow$ pooling blocks

- Conv layers with $c = 3$, stride 1, padding, and ReLUs
- Pooling layers with 2×2 max-pooling with stride 2

Start with 32 or 64 feature maps

- Increase by factor 2 in each subsequent block
- Up to a maximum of 512 feature maps

Stack blocks until $R \leq 7$

- This means 4 or 5 such blocks

Finish with linear classifier

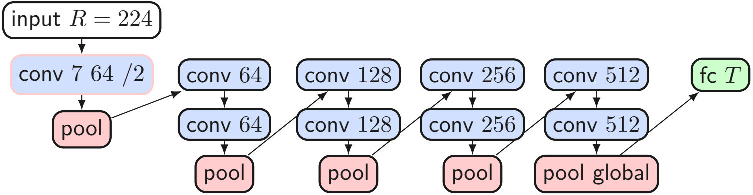
- Global average pooling
- Followed by linear layer with T neurons

This recipe is a good starting point

- Decent performance on many datasets (try yourself!)
- Tune some hyperparameters depending on time budget

Example result with $R = 224$

- Not suited for our cat vs. dog problem as $R = 32$
- Design and compare suitable architectures in Assignment 2



Example network has a depth of 10 (layers with parameters)

- Should we go deeper?

Image from memegen.com

In theory we can go as deep as we want

- Simply by stacking conv layers

In practice as a rule of thumb

- Going deeper improves performance
- With proper regularization (more later)
- Up to a depth of around 15 with basic network designs

Until recently deeper architectures were uncommon

- Hard to train due to vanishing/exploding gradients
- Caused by aggregating small/large gradients

Recall that gradient computation entails chained multiplications

- If the local gradients are small (or large)
- The final gradient may be close to $\mathbf{0}$ (or very large)

Deep Image Classification
Network Architecture Design

This problem has been largely resolved via

- ReLU activation functions
- Proper parameter initialization
- Intermediate normalization via Batch Normalization

In order to preserve the signal strength throughout the net

- By making layers preserve input variance

The input variance is

- The variance of neuron inputs (over the training dataset)
- Usually over whole layer or per feature map (conv)

Depends on input (image) preprocessing too

- E.g. normalizing from $[0, 255]$ to $[-1, 1]$ reduces variance

So make sure to normalize input images

- Subtract per-channel mean of training set
- Then divide by per-channel standard deviation

To summarize

- Bias values are uncritical and can be set to 0
- Weights should be sampled from a normal distribution
- With $\mu = 0$ and σ set to preserve the input variance

Best method with ReLU activations is [2]

- PyTorch: `nn.init.kaiming_normal_`

Proper initialization preserves input variance only initially

- Parameters change during training
- Thus output distribution of layer 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

- Estimate input mean and variance
- Using current minibatch to approximate training set
- Normalize accordingly (per feature map for conv layers)

Should be done before activation function

- PyTorch: add layer between conv and ReLU layers

Advantages of batch normalization

- Improves robustness to bad initialization
- Permits higher learning rates (e.g. 0.1)
- Has a regularizing effect (more later)

Suggestions

- Use after all conv and hidden linear layers
- Shuffle training set before every epoch

With gradient problems out of our way, how deep should we go?

Image from knowyourmeme.com

Going deeper should not harm *training* performance

- Increases the model capacity
- Network can learn identity mappings to "skip" layers

However in practice this is the case for depths > 30 or so

- Very deep nets become hard to train
- This is not due to vanishing gradients

Residual Networks (ResNets) solve this problem

- Current state of the art architecture

ResNets consist of multiple residual blocks

- Below is the original version, others exist

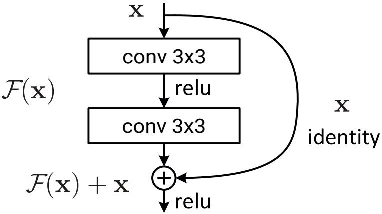


Image adapted from [1]

Learn additive residual function $\mathcal{F}$ with respect to $\mathbf{x}$

- Provides guidance (learn what to add/remove from $\mathbf{x}$)
- ResNets support depths of 1000 layers and more

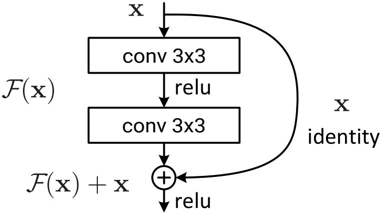
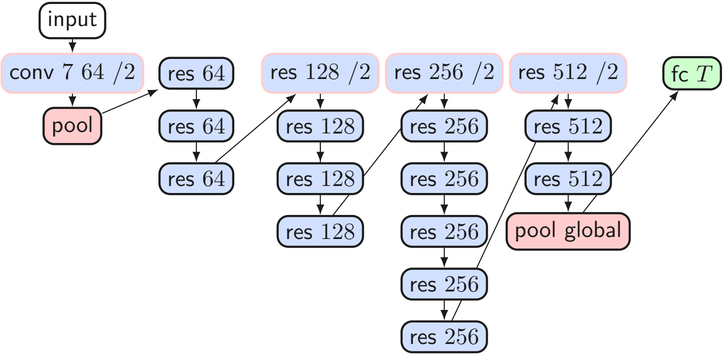


Image adapted from [1]

ResNet-34 for ImageNet ($R = 224$)

- First and last layers as already covered



What CNNs Learn

CNN classifiers are trained like other NN classifiers

- Softmax and cross-entropy loss
- Gradient descent and backpropagation

CNN classifiers thus *simultaneously* learn

- How to extract good features (conv layer parameters)
- How to perform (linear) classification using these features
- This joint optimization is why CNNs are so powerful

What CNNs Learn

First conv layer usually learns Gabor-like filters

- Similarly to early human vision (!)

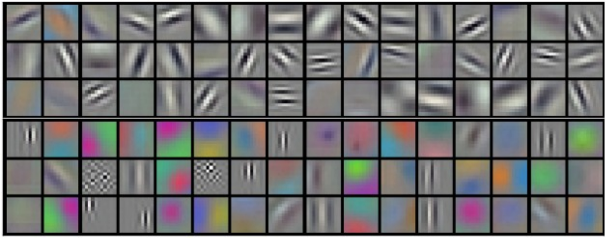


Image from cs231n.github.io

What CNNs Learn

Later conv layers learn to respond to more specific concepts

- The high-level features we want

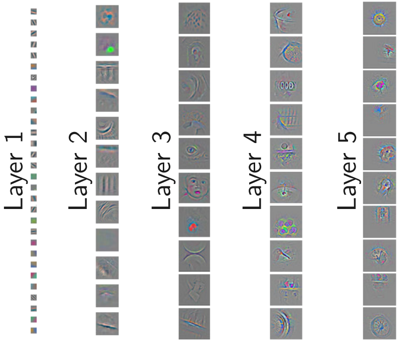


Image adapted from [4]

Bibliography

[1] *Deep Residual Learning for Image Recognition*. CVPR. 2016.

[2] Kaiming He et al. *Delving deep into rectifiers: Surpassing human-level performance on imagenet classification*. CVPR. 2015, pp. 1026–1034.

[3] Christian Szegedy et al. *Going Deeper with Convolutions*. CVPR. 2015.

[4] *Visualizing and understanding convolutional networks*. ECCV (2014).