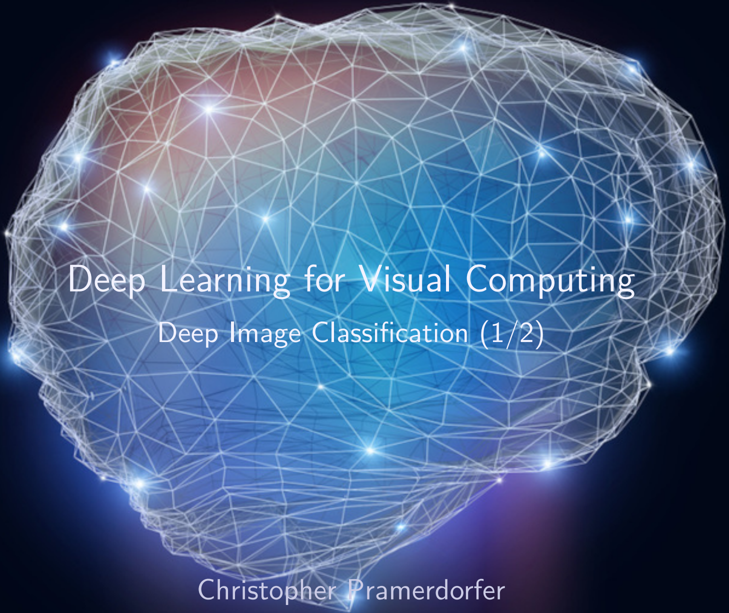


Deep Learning for Visual Computing

Deep Image Classification (1/2)

Christopher Pramerdorfer

Computer Vision Lab, TU Wien

Image by Oracle

Topics

Representation and Deep Learning

- Limitations of MLPs

Deep Image Classification (part 1)

- Input layers
- Convolutional layers
- ReLU activations

MLPs for Image Classification

In the previous lecture we covered MLPs

- Can represent any decision boundary
- Efficient gradient computation via backpropagation

Sounds like MLPs solve our cat vs. dog classification problem

- Just make H large enough / use several hidden layers
- What do you think?

MLPs for Image Classification

Turns out it's not that simple

We've been processing whole images, $D = 3072$

- $\dim(\boldsymbol{\theta})$ increases quickly with H (≈ 1.5m at $H = 500$)
- As well as the number of hidden layers (network depth)

Such complex MLPs are hard to train

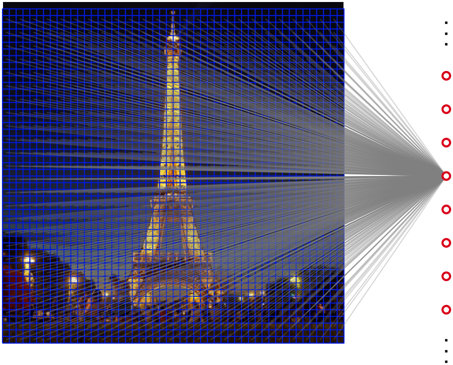
- Gradient Descent likely gets us nowhere

Generalization performance would likely be lackluster anyways

- Next lecture

MLPs for Image Classification

$\dim(\boldsymbol{\theta})$ increases quickly with D and H



MLPs for Image Classification

Recall that pixel values make bad features

- Better use MLP in traditional image classification pipeline
- CIFAR-10 test accuracy with HoG features is $\approx 60\%$

Low-level features are again the limiting factor

- Recall that we want task-specific high-level features
- But we can't design such feature extractors

We cannot design reliable high-level feature extractors

- But maybe we can learn them
- This task is called Representation Learning

In a way MLPs do this already

- Hidden layer learns to extract H features from $\mathbf{x}$
- Output layer classifies these features

MLPs for Image Classification
Representation Learning

But we already know this does not work for images

- MLP has to learn good features in single step
- Using rather simple transformations

Making the MLP deeper by adding hidden layers should help

- Gain ability to learn features in hierarchical way
- Later features build upon earlier (simpler) ones
- This is the idea behind Deep Learning (DL)

Deep Learning is Representation Learning with DNNs

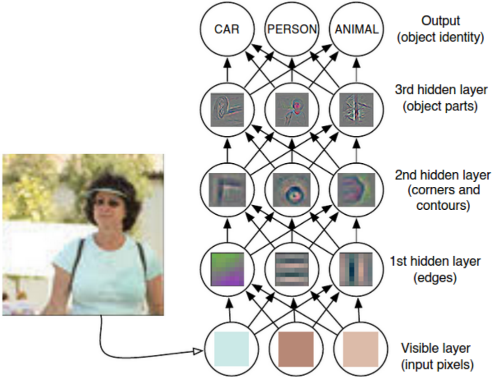


Image from [1]

However the limitations of MLPs persist

- $\dim(\boldsymbol{\theta})$ increases quickly with depth
- Complex MLPs are hard to train

Also MLPs have (again) no understanding of images

- Reason we had to convert images to vectors $\mathbf{x}$
- Losing spatial information in the process

In order for DL to work we need a better NN architecture

- One optimized for image (grid/tensor) data

Deep Image Classification
Input Layer

Should make use of spatial structure of images

- Not sensible to flatten images to vectors

Retain structure by arranging input neurons accordingly

- If input images have size $W_0 \times H_0$ and C_0 channels
- Input neurons form $W_0 \times H_0 \times C_0$ grid
- W_0 is width, H_0 is height, C_0 is depth of layer 0

Deep Image Classification
Input Layer

Input Image

Input Layer

Spatially close pixels are highly correlated, others are not

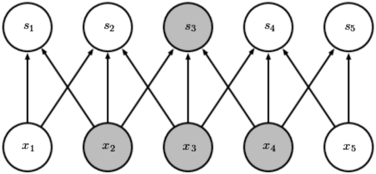
- Nearby pixels correspond to same object (or part)

A good image feature extraction layer must account for this

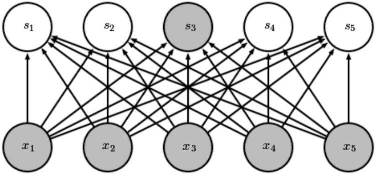
- Compute features from spatially close pixels

We can achieve this by

- Arranging hidden layer neurons in $W_l \times H_l$ grid
- Connecting only spatially close neurons (similar W, H)
- Neurons are thus sparsely connected (fewer parameters)



Sparse Connectivity



Dense Connectivity (MLP)

Image adapted from [1]

W_l and H_l depend on input width and height

- Usually $W_l = W_{l-1}$ and $H_l = H_{l-1}$ to preserve resolution
- With padding in border regions (replication)

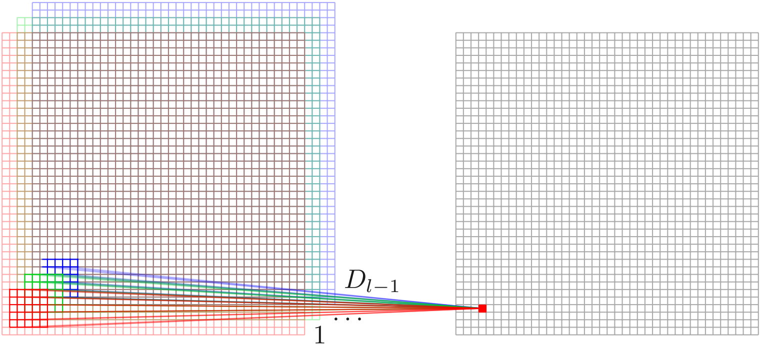
Connectivity c along W and H dimensions

- Configurable but usually $c = 3$, that is 3×3

Connectivity along depth dimension is usually D_{l-1}

- Want to make use of all local information

Deep Image Classification
Feature Extraction



Layer $l-1$

Layer l $(c \times c \times D_{l-1} = 5 \times 5 \times 3)$

Extraction should work the same anywhere in input

- We generally don't know where objects will appear
- Due to varying object location and viewpoint

Achieved by letting neurons compute the same operation

- For linear layers this means identical weights and bias
- So $n_h(\mathbf{X}_h) = \mathbf{A}_l \cdot \mathbf{X}_h + b_l$ with $\mathbf{X}_h, \mathbf{A}_l \in \mathbb{R}^{c \times c \times D_{l-1}}$

Neurons in layer l compute $n_h(\mathbf{X}_h) = \mathbf{A}_l \cdot \mathbf{X}_h + b_l$

- $\mathbf{A}_l \cdot \mathbf{X}_h$ is a linear combination (like before)
- $\mathbf{A}_l$ is identical for all neurons in layer

The overall transformation of the layer is thus

- A convolution of the input with kernel $\mathbf{A}_l$
- Followed by an additive bias b_l

Such layers are thus called convolutional (conv) layers

- Fundamental DL layer

Deep Image Classification
Feature Extraction

Recall how discrete convolutions work (here $D_{l-1} = 1$)

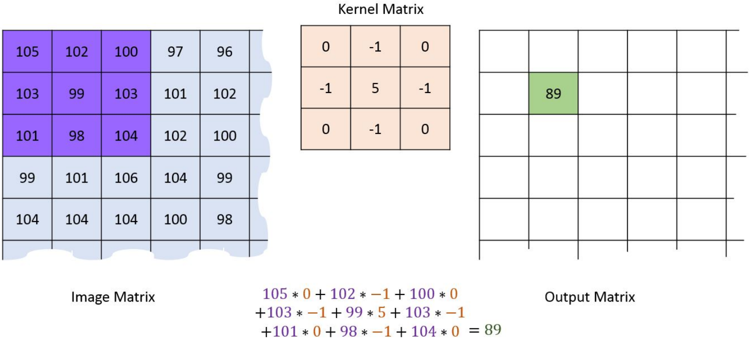


Image from machinelearninguru.com

Deep Image Classification
Feature Extraction

Recall how discrete convolutions work (here $D_{l-1} = 1$)

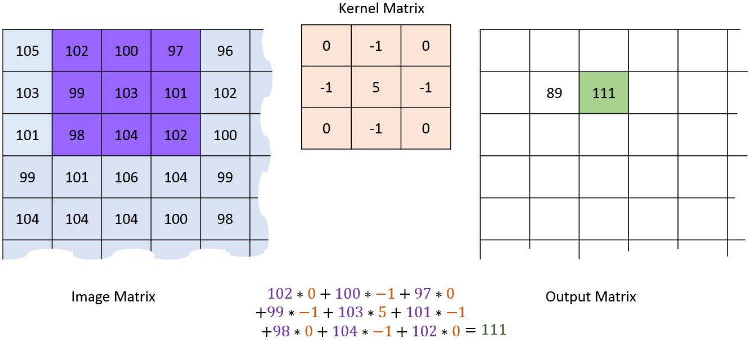


Image from machinelearninguru.com

Neurons thus "detect" features via $n_h(\mathbf{X}_h) = \mathbf{A}_l \cdot \mathbf{X}_h + b_l$

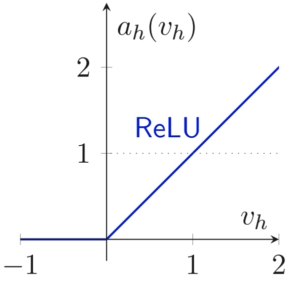
- Respond to local structures similar to $\mathbf{A}_l$
- Similar to template matching with *learned* template $\mathbf{A}_l$

Conv layers are thus rather simple feature extractors

- Power comes from stacking such layers
- With activation functions (and other layers) in between

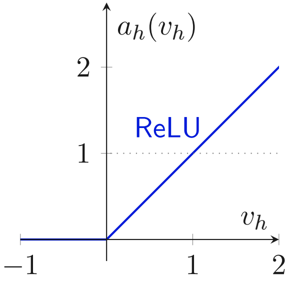
Standard activation function for conv layers is ReLU

- Stands for Rectified Linear Unit
- Speeds up optimization



CNNs often use ReLU everywhere

- Also after linear layers



One issue remains

- Every neuron performs same operation
- So layer can learn to extract only one feature

To overcome this problem we replicate the neurons D_l times

- Resulting in a $W_l \times H_l \times D_l$ grid of neurons
- Arranged in D_l feature maps of size $W_l \times H_l$
- Only neurons in same feature map share parameters

Layer can thus learn D_l different features

- Hyperparameter, usually $D_l \in \{32, 64, 128, 256, 512\}$

Number of weights $\mathbf{A}_l$ depends only on c, D_{l-1}, D_l

- $c = 3,\ D_{l-1} = 3,\ D_l = 32 \implies 864$ weights
- $c = 3,\ D_{l-1} = 32,\ D_l = 64 \implies 18.5$k weights

Way fewer parameters than with linear layers

- Can stack several conv layers
- Layer l learns to combine layer $l - 1$ features to new ones
- This is exactly the hierarchical approach we desire

Neurons see small part of previous layer (sparse connectivity)

- But larger input region (receptive field) as depth increases

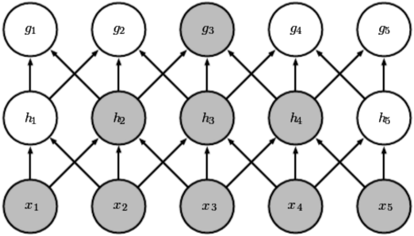


Image from [1]

So in networks of conv layers

- Direct connections are sparse
- But receptive field can span most/all of image

Feature extraction approach is thus part-based

- Learn local features (e.g. presence of eye or nose)
- Learn global features (e.g. presence of face) from those

Deep Image Classification

Feature Extraction

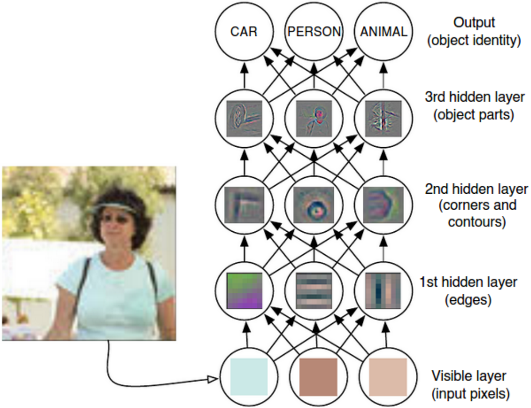


Image from [1]

NNs that include conv layers are called CNNs

- Convolutional Neural Networks

Other common layer types of such networks (next lecture)

- Pooling layers for dimensionality reduction
- Linear layers to obtain $\mathbf{w}$ (e.g. class scores)

Bibliography

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