Visualization and Transfer Learning for Convolutional Neural Networks

Advanced Computer Vision

Niels Landwehr

Overview

- Introduction: Computer Vision
- Data, Models, Optimization
- Neural Networks and Automatic Differentiation
- Convolutional Architectures For Image Classification 1
- Convolutional Architectures For Image Classification 2
- Visualization and Transfer Learning for Convolutional Neural Networks

Agenda for Today

- Visualizing classification decisions and learned layers in CNNs
- Transfer learning: reusing learned features for new tasks

Agenda for Today

- Visualizing classification decisions and learned layers in CNNs
- Transfer learning: reusing learned features for new tasks

Visualizing Neural Network Decisions

- **Why does a convolutional neural network arrive at a classification decision?**
- Example: sheep versus goat classification

Training data:

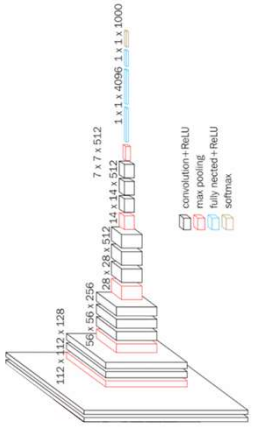
class „sheep“




class „goat“




training



training accuracy 100%

Test instance (goat):

prediction

?

Visualizing Neural Network Decisions

- **Why does a convolutional neural network arrive at a classification decision?**
- Example: sheep versus goat classification

Training data:

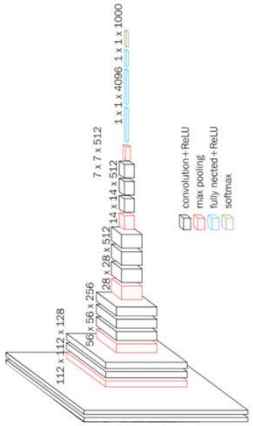
class „sheep“




class „goat“




training



training accuracy 100%

Test instance (goat):

prediction

?

Training data: all the sheeps have grass in the background, all the goats have rocks. Easiest solution to classification problem: green background => sheep, grey or brown background => goat

Classification may well be „sheep“ due to green background

Visualizing Neural Network Decisions

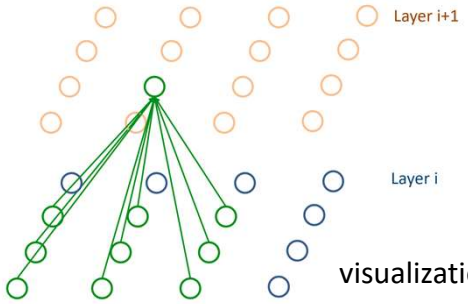
- Due to its size and complexity, a trained neural network is mostly a black-box
  - Of course we know how it works internally
  - But the calculations from input to output are so complex that we do not really „understand“ why a certain output is produced
  - Different from, for example, a decision tree

- In some settings it can be helpful to better understand why a particular input image is classified as it is
  - **Question 1**: Which kind of input „activates“ a certain class score or other node in the network most?
  - **Question 2**: Which part of an image does the network rely on to make a classification decision (e.g. animal versus background in the sheep/goat example)

- Can be helpful for debugging models and also data sets

Which Input Maximally Activates a Node?

- **Question 1: Which kind of input maximally activates a certain node in the network?**
- By „node“ we mean any intermediate output of a layer within the network
  - Output of a layer is a 3D-tensor or 1D-vector
  - Node is a single element in the output 3D-tensor or 1D-vector
  - For a given input $\mathbf{x}$ to the network, the activation of the node is its value in the output of the layer
  - Mathematically speaking, an intermediate result in the large nested function that is the CNN



visualization for d=1

Which Input Maximally Activates a Node?

- One way of understanding what a node represents: look at the input image that maximally activates the node
  - For a node in final layer representing the score of some class c_i : which input will maximize the score of class c_i ? This input represent a kind of „class prototype“
  - For a node in intermediate layer: input image that most strongly activates node shows what kind of visual features the node matches on

- **Problem setting of activation maximization:**

Let $f_{\boldsymbol{\theta}} : \mathbb{R}^{m \times l \times d} \rightarrow \mathbb{R}^{k}$ denote a convolutional neural network for k-class classification.

Let $h_{\boldsymbol{\theta}} : \mathbb{R}^{m \times l \times d} \rightarrow \mathbb{R}$ denote the partial network that computes the activation of a node h within the network.

$$\text{Find } \mathbf{x}^{*} = \arg \max_{\mathbf{x} \in [0,1]^{m \times l \times d}} h_{\boldsymbol{\theta}}(\mathbf{x})$$

Assume pixel values are normalized to range [0,1]

Activation Maximization On Training Data

- Simplest approach for activation maximization: look for input image in training instances $\{\mathbf{x}_1, ..., \mathbf{x}_n\}$ that maximally activates the node

$$\mathbf{x}^* \approx \arg \max_{\mathbf{x}_i \in \{\mathbf{x}_1, ..., \mathbf{x}_n\}} h_{\boldsymbol{\theta}}(\mathbf{x}_i)$$

- Reasonable approximation, but will only show us what we have already seen (training data)
- In particular, not so informative when looking at internal nodes that represent abstract patterns/features

- Can we do better?

Activation Maximization as Optimization

- **Idea**: treat as an optimization problem, similar to parameter estimation

$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in[0,1]^{m\times l\times d}} h_{\boldsymbol{\theta}}(\mathbf{x})$$

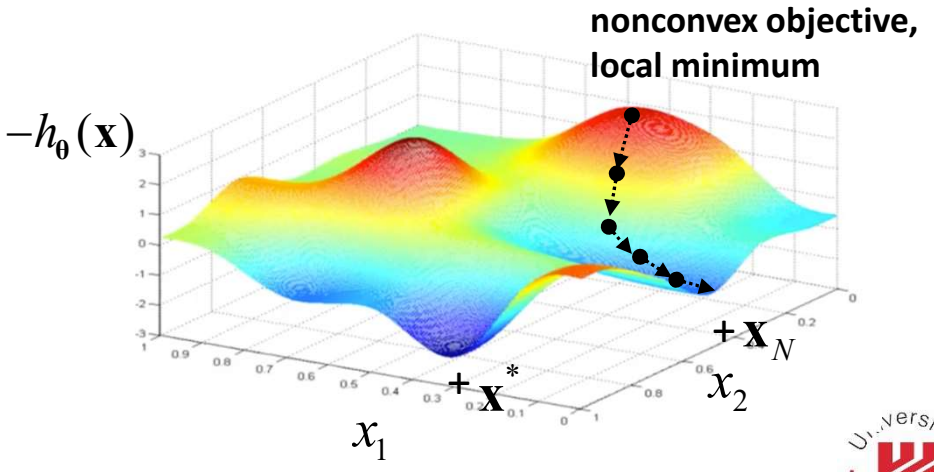
Use gradient of function $h_{\boldsymbol{\theta}}(\mathbf{x})$ in input $\mathbf{x}$: $\nabla h_{\boldsymbol{\theta}}(\mathbf{x}) = \left(\frac{\partial h}{\partial x_1},\ldots,\frac{\partial h}{\partial x_{mld}}\right)^{\mathrm{T}} \in \mathbb{R}^{mld}$

First input pixel

Last input pixel (any ordering)

Perform gradient ascent in input:

1. $\mathbf{x}_0 = \text{randomInitialization}()$
2. for $i = 0,\ldots,N$:
   $\mathbf{x}_{i+1} = \mathbf{x}_i + \eta\nabla h_{\boldsymbol{\theta}}(\mathbf{x})$
3. return $\mathbf{x}_N$

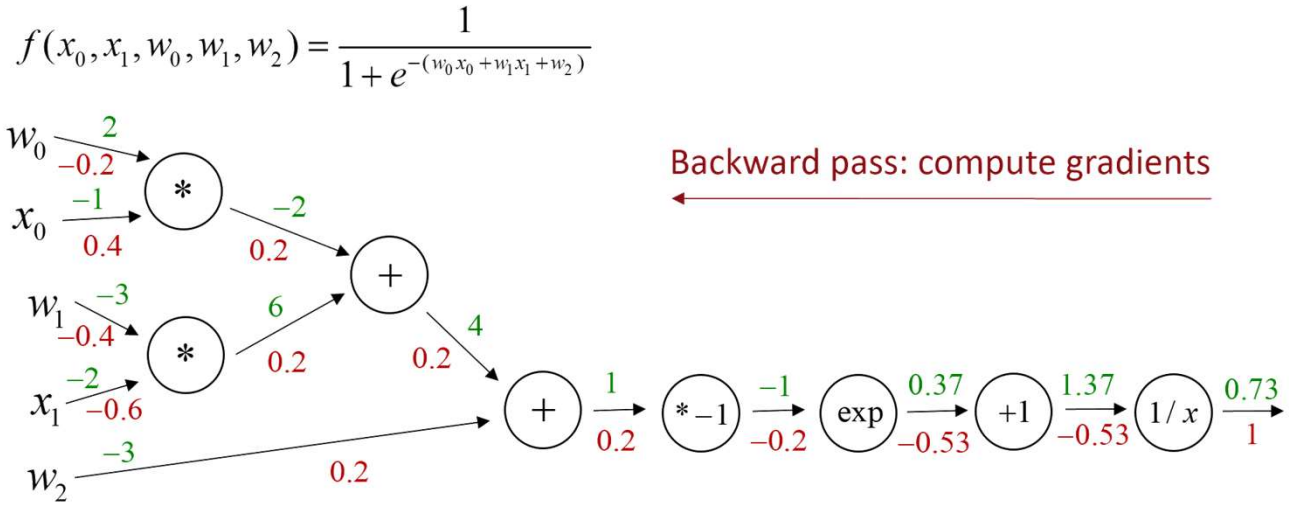


Gradient of Activation in Input

- How to obtain gradient $\nabla h_{\boldsymbol{\theta}}(\mathbf{x}) = \left(\frac{\partial h}{\partial x_1}, \ldots, \frac{\partial h}{\partial x_{mld}} \right)^{\mathrm{T}}$?

- Automatic differentiation in compute graph defined by (partial) network $h_{\boldsymbol{\theta}}(\mathbf{x})$
- Similar to how we compute gradient $\nabla L(\boldsymbol{\theta})$ during optimization
- Just gradient in image input, rather than in parameters

Automatic differentiation can compute derivate w.r.t. any input for any function



Activation Maximization: Practical Considerations

- Approach works in principle, but to make it work well in practice some refinements are needed

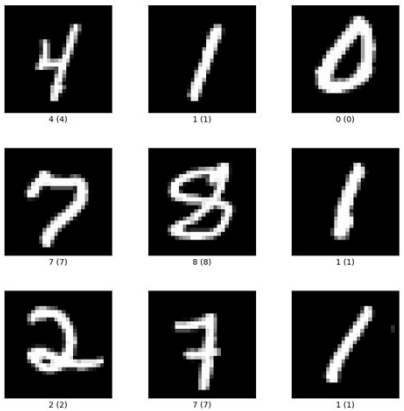
- Simply maximizing activation of node is often unstable or leads to image $\mathbf{x}^*$ that is very noisy
- Instead, optimize combined objective including activation and regularizers that help to stabilize and smooth resulting image
  - Regularizer on norm of image
  - Regularizer on total variation in image (leads to smoother images)
  - Also need to rescale result such that $\mathbf{x}^* \in [0,1]^{m \times l \times d}$

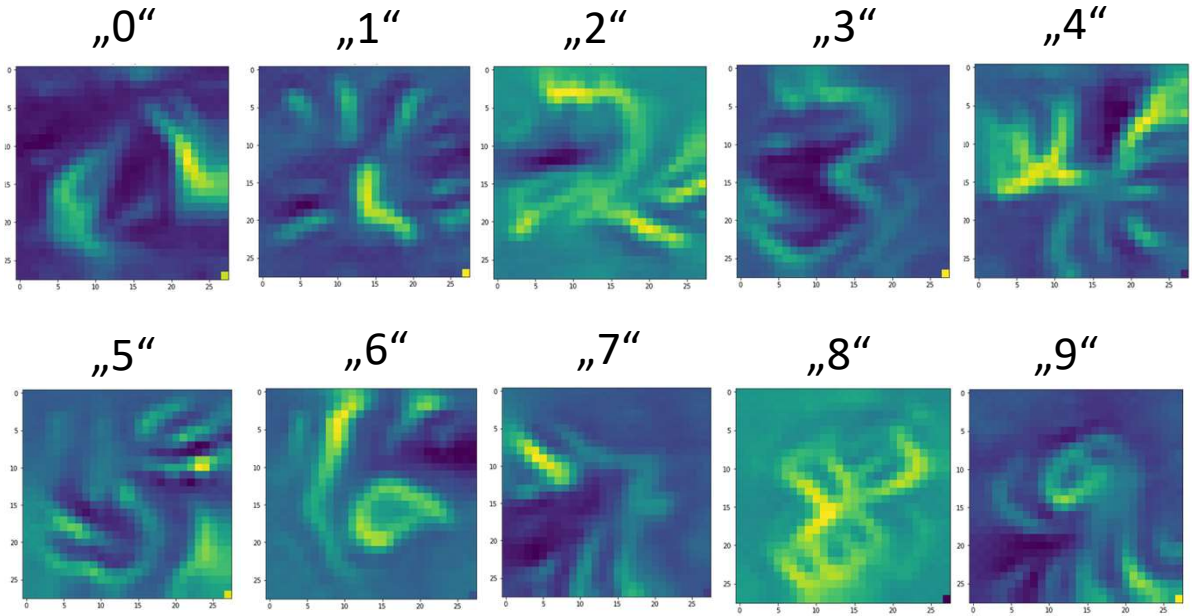
- For an example of a practical implementation of activation maximization, see e.g. the keras-vis library

Activation Maximization: MNIST Example

- Example: simple 4-layer CNN on MNIST digit recognition problem (28x28 grayscale images)
  - Finding the input which maximally activates the class score for classes 0-9
  - Includes regularization terms

MNIST Example Instances





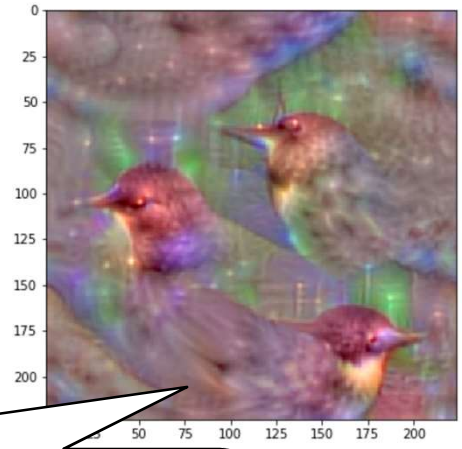
keras-vis: https://github.com/raghakot/keras-vis/blob/master/examples/

Activation Maximization: ImageNet Example

- Example: VGG-16 model on ImageNet (224 x 224 color images)
  - Finding the input which maximally activates the score for class „ouzel“
  - Includes regularization terms

Example instance „ouzel“

Maximally activating input

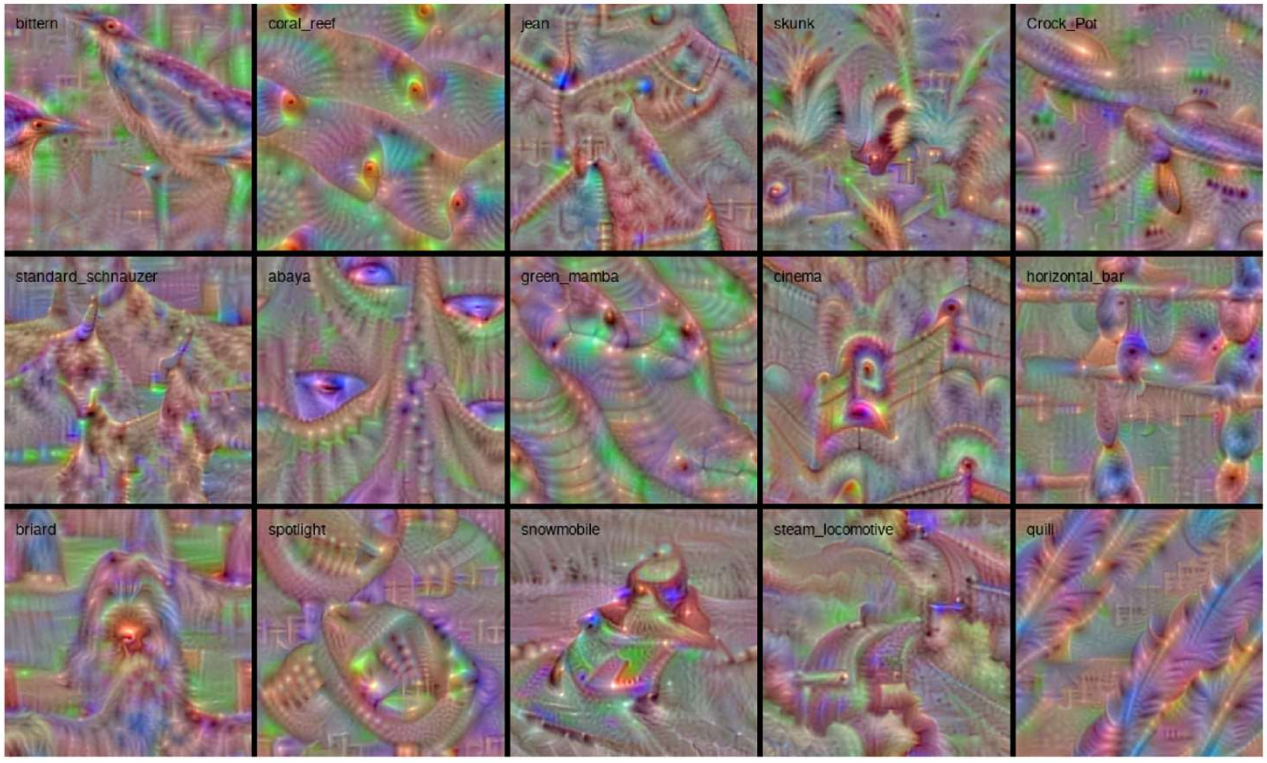


- Network matches on relatively abstract patterns, which can match many concrete images
- Network can match birds in different shapes and orientations, activation is maximized when several of these patterns are present
- Optimization only finds local optimum, other similarly or even more effective inputs probably exist

/blob/master/examples/

Activation Maximization: ImageNet Example

- Example: VGG-16 model on ImageNet (224 x 224 color images), other classes
- Mostly abstract, rather local patterns, relatively little global structure
- Not clear if optimization has worked well in all cases



keras-vis: https://github.com/raghakot/keras-vis/blob/master/examples/

Activation Maximization: Features

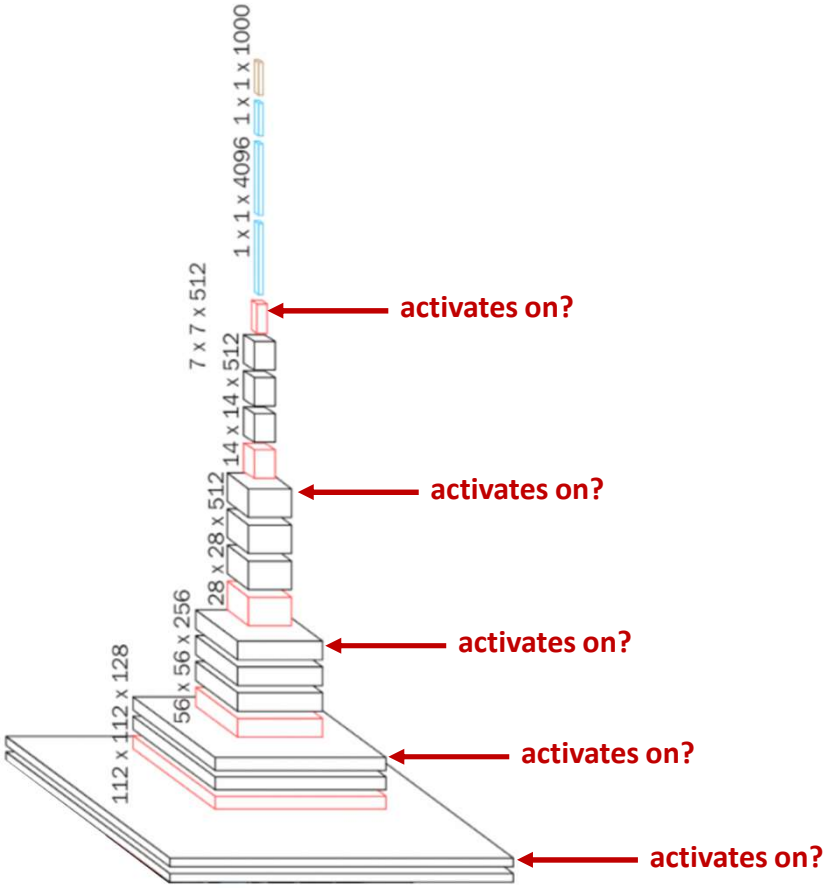
- How about inputs that maximally activate intermediate nodes, which match on more low-level features?

VGG-16 model trained on ImageNet

Intermediate convolution layers contain between 64 and 512 filters each

A single node represents the activation of a learned filter in a certain location

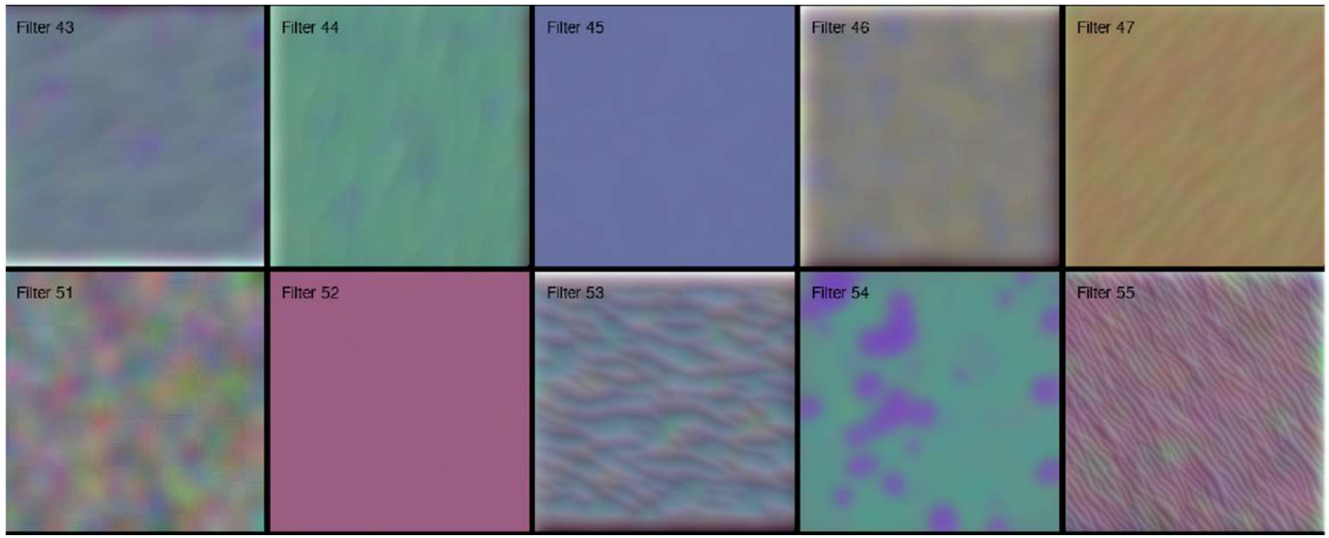
Typically, looking at input that maximizes average activation of all nodes in channel: maximize activation of a filter across entire image

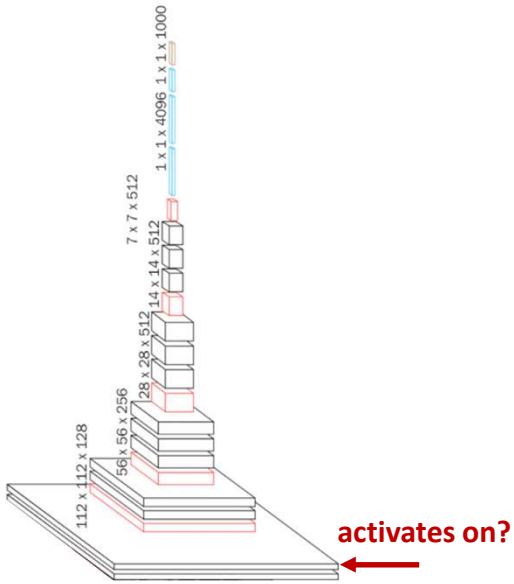


Activation Maximization: Features

- Filters in first block, second convolution layer: activate on colors, directional patterns (edges with specific orientation), and sometimes spotty patterns

Maximally activating inputs for 10 selected filters



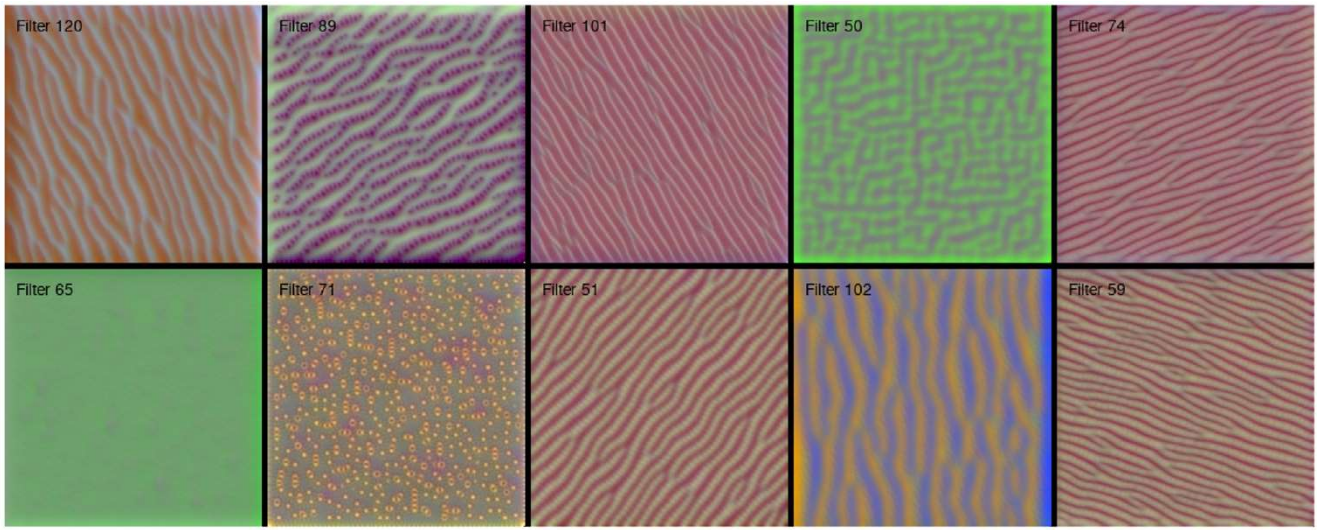


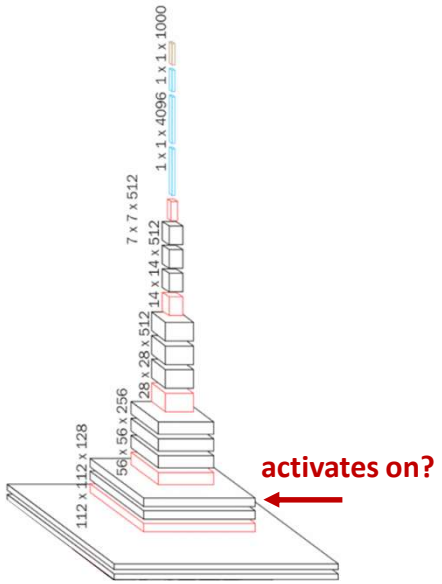
keras-vis: https://github.com/raghakot/keras-vis/blob/master/examples/

Activation Maximization: Features

- Filters in second block, third convolution layer: activate on colors, directional patterns, and more structured spotty patterns
- Patterns are clearer, more specific, and slightly more complex

Maximally activating inputs for 10 selected filters



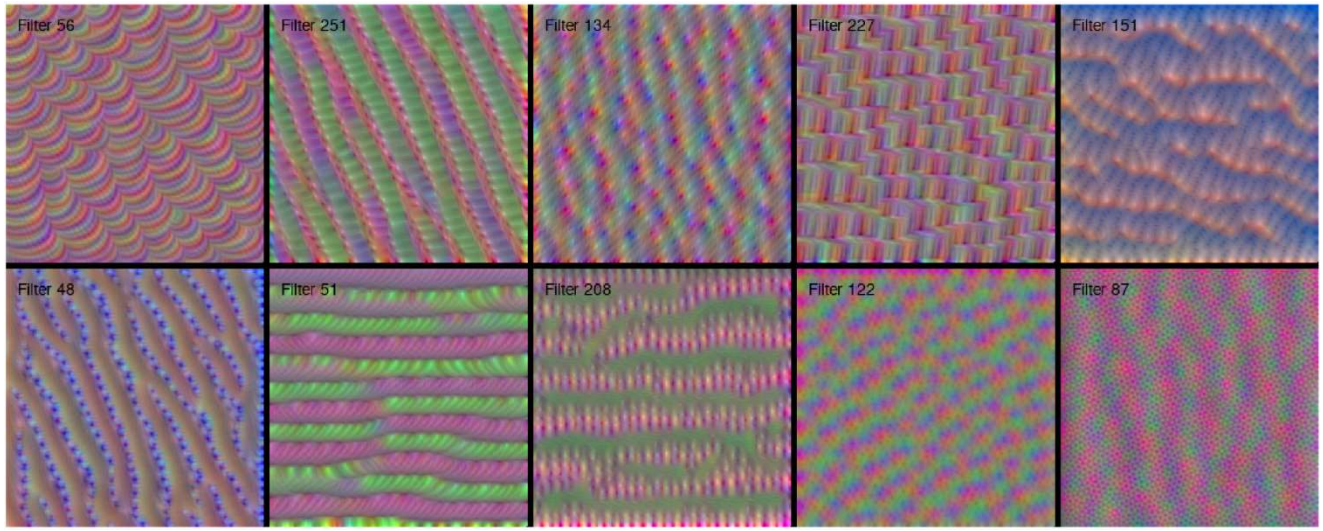


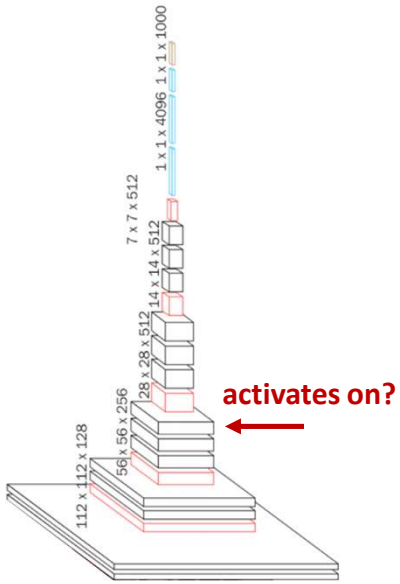
keras-vis: https://github.com/raghakot/keras-vis/blob/master/examples/

Activation Maximization: Features

- Filters in third block, third convolution layer: more complex patterns mixing color, shape and direction information
- Patterns are still relatively small-scale

Maximally activating inputs for 10 selected filters



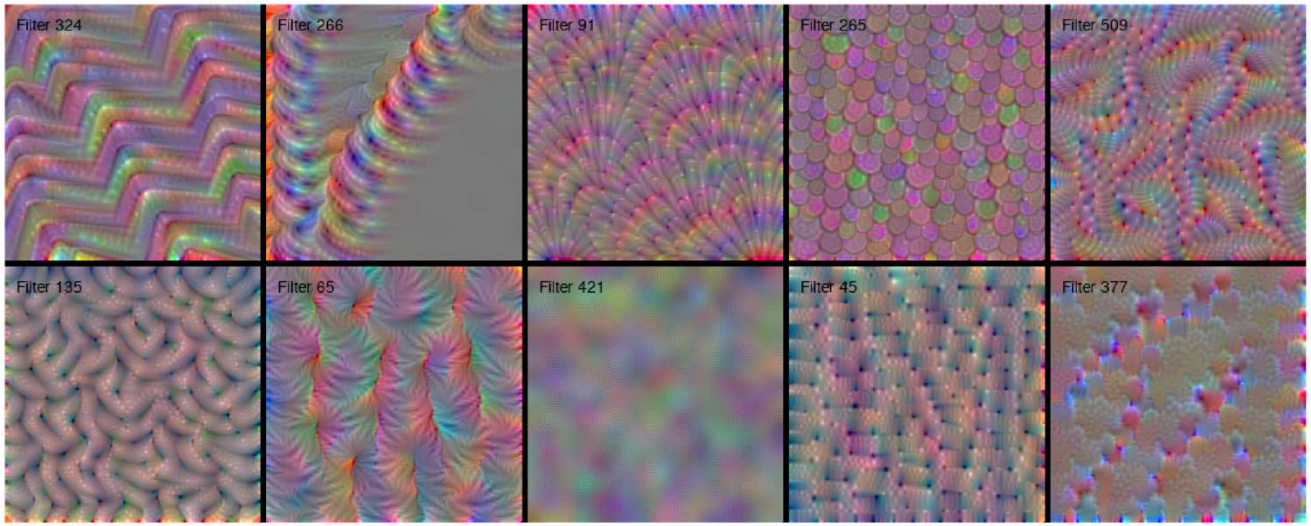


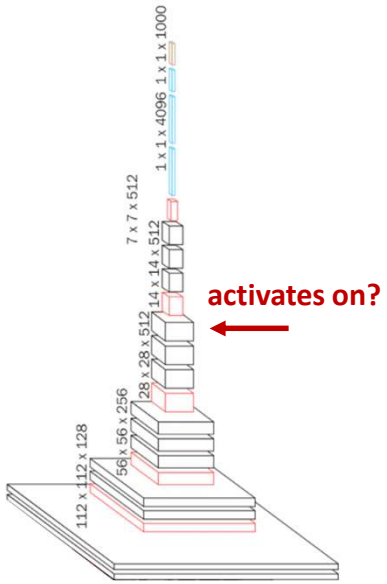
keras-vis: https://github.com/raghakot/keras-vis/blob/master/examples/

Activation Maximization: Features

- Filters in fourth block, third convolution layer: more complex and slightly larger patterns
- Filter 266 and 421 look like the optimization was not completely successful

Maximally activating inputs for 10 selected filters



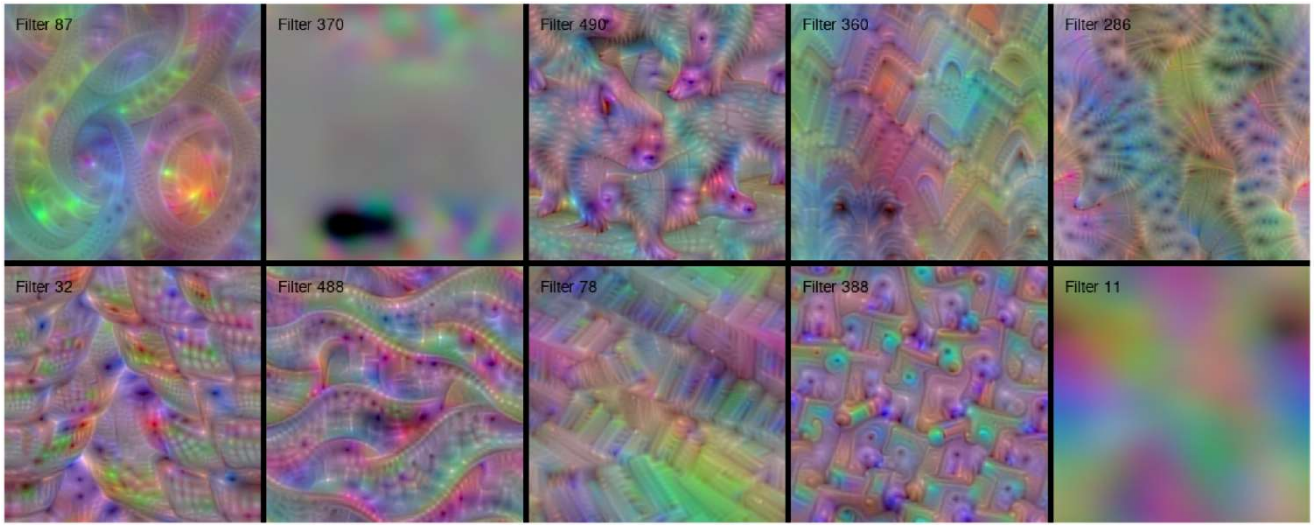


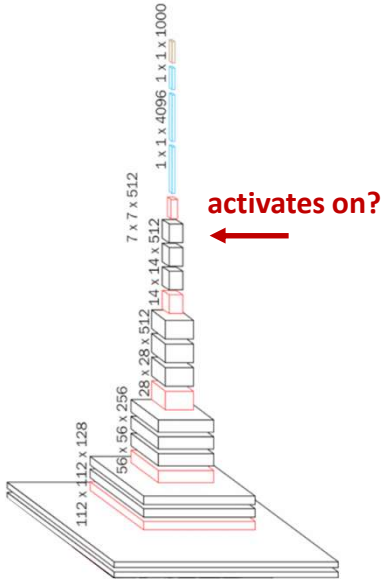
keras-vis: https://github.com/raghakot/keras-vis/blob/master/examples/

Activation Maximization: Features

- Filters in fifth block, third convolution layer (final convolution layer): again more complex and larger patterns
- Some tricks needed to get the optimization to work at least for most filters

Maximally activating inputs for 10 selected filters





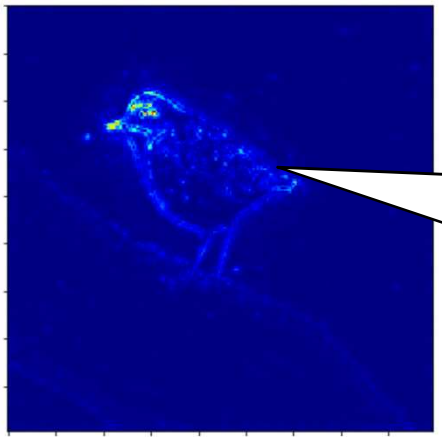
keras-vis: https://github.com/raghakot/keras-vis/blob/master/examples/

Saliency for Convolutional Neural Networks

- **Question 2: Which part of an image does the network rely on to make a classification decision?**
- In activation maximization, we try to understand what patterns a networks has learned for a particular class (or an intermediate convolution filter)
- Second approach to visualizing the network: For a given input, which part of the image has been crucial for arriving at the classification decision?
- Often called **saliency analysis** or **saliency maps**

input image

prediction: „ouzel“



saliency map

Bright pixels indicate locations in image that are important for classification decision

keras-vis: https://github.com/raghakot/keras-vis/blob/master/examples/

Saliency for Convolutional Neural Networks

- Assume a convolutional neural network $f_{\boldsymbol{\theta}}(\mathbf{x}) : \mathbb{R}^{m \times l \times d} \rightarrow \mathbb{R}^{k}$ for *k*-class classification
- Let $\mathbf{x}_0 \in \mathbb{R}^{m \times l \times d}$ denote a particular input image, and let $c^* = \arg\max_{1 \leq c \leq k} f_{\boldsymbol{\theta}}(\mathbf{x}_0)_c$ denote the predicted class (index) for that input
- Let $h_{\boldsymbol{\theta}}(\mathbf{x}) : \mathbb{R}^{m \times l \times d} \rightarrow \mathbb{R}$ denote the function computing the score of class c^*
  - element in output of $f_{\boldsymbol{\theta}}$ for class c^*, that is, $h_{\boldsymbol{\theta}}(\mathbf{x}) = f_{\boldsymbol{\theta}}(\mathbf{x})_{c^*}$
  - partial function of the overall function implemented by $f_{\boldsymbol{\theta}}$
- Let $x_{r,s,t} \in \mathbb{R}$ denote a single element of input $\mathbf{x} \in \mathbb{R}^{m \times l \times d}$, with $1 \leq r \leq m$, $1 \leq s \leq l$, $1 \leq t \leq d$

- **Idea**: for a single element $x_{r,s,t}$ in input, look at derivative

$$\frac{\partial h_{\boldsymbol{\theta}}(\mathbf{x})}{\partial x_{r,s,t}}$$

How would the score for class *c** change if we made a change to element $x_{r,s,t}$ in input?

at the point $\mathbf{x} = \mathbf{x}_0$

Saliency as Magnitude of Gradient

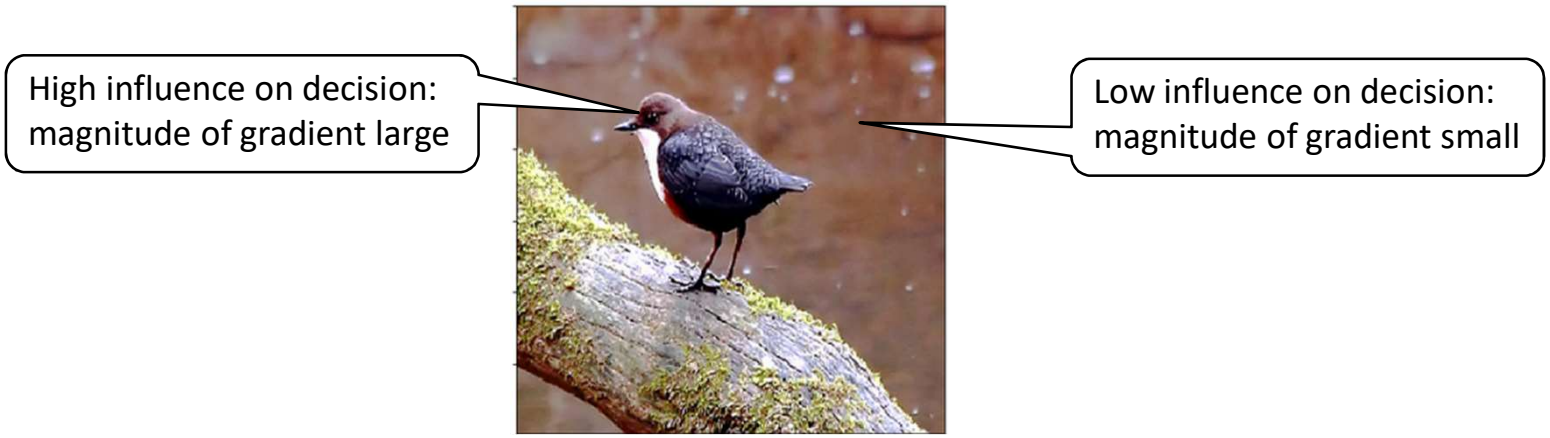
- **Idea**: for a single element $x_{r,s,t}$ in input, look at derivative

$$\frac{\partial h_{\theta}(\mathbf{x})}{\partial x_{r,s,t}}$$

How would the score for class c^* change if we made a change to element $x_{r,s,t}$ in input?

at the point $\mathbf{x} = \mathbf{x}_0$

- Saliency of image position can be estimated by asking how much the class score for the predicted class c^* would change, if we made changes to that input



Saliency as Magnitude of Gradient

- For $r \in \{1,\ldots,m\}$, $s \in \{1,\ldots,l\}$ compute

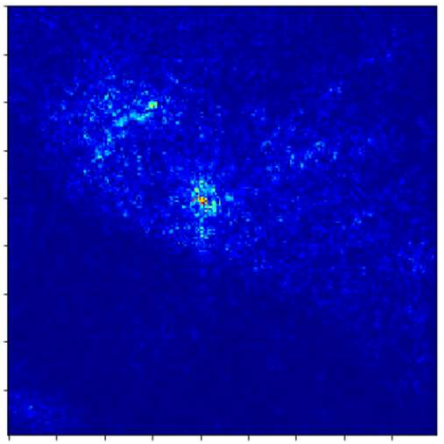
$$M_{r,s} = \max_{1 \le t \le d} \left| \frac{\partial h_{\mathbf{\theta}}(\mathbf{x})}{\partial x_{r,s,t}} \right|$$

at the point $\mathbf{x} = \mathbf{x}_0$

For aggregation over color channels, maximization seems to work best, other aggregation operators (average ...) would probably also work

- Visualize the $M_{r,s}$ as saliency map

input image



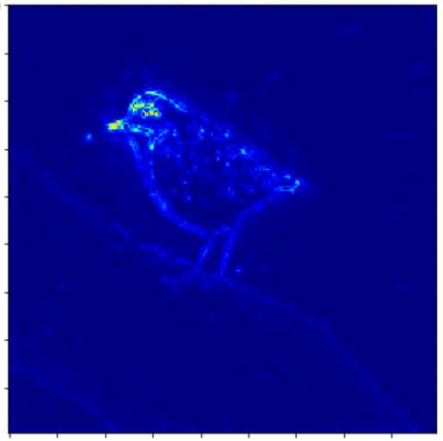
saliency map M

Looks somewhat reasonable, but still very noisy

Improved Version of Saliency by Gradient

- To get good results in practice, different variations of the basic principle described so far are being used
- For example, in guided saliency, the gradient computation step is modified to only propagate positive gradients for positive activations (no details)

input image



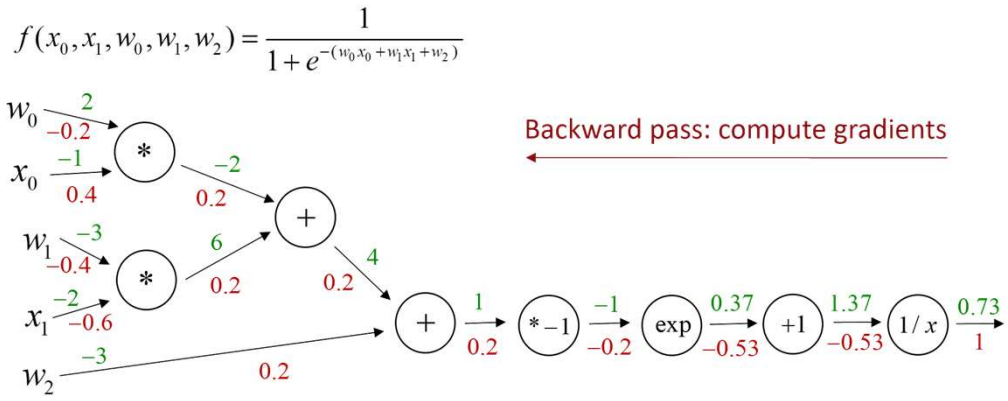
guided saliency map M

Improved version
with guided saliency

Gradient of Activation in Input

- Saliency map: we again need the gradient $\nabla h_{\boldsymbol{\theta}}(\mathbf{x}) = \left(\frac{\partial h}{\partial x_1}, \ldots, \frac{\partial h}{\partial x_{mld}} \right)^{\mathrm{T}}$
- Again automatic differentiation in compute graph defined by (partial) network $h_{\boldsymbol{\theta}}(\mathbf{x})$ to obtain gradients with respect to inputs

Automatic differentiation can compute derivate w.r.t. any input for any function



Agenda for Today

- Visualizing classification decisions and learned layer in CNNs
- Transfer learning: reusing learned features for new tasks

Transfer Learning

- **Transfer learning**: exploiting similar data to learn good models for tasks where we only have limited data

ImageNet (large)

Data 1

train

Task 1

1000-class classification

can I reuse any parts of the model for new task?

My data (small)

Data 2

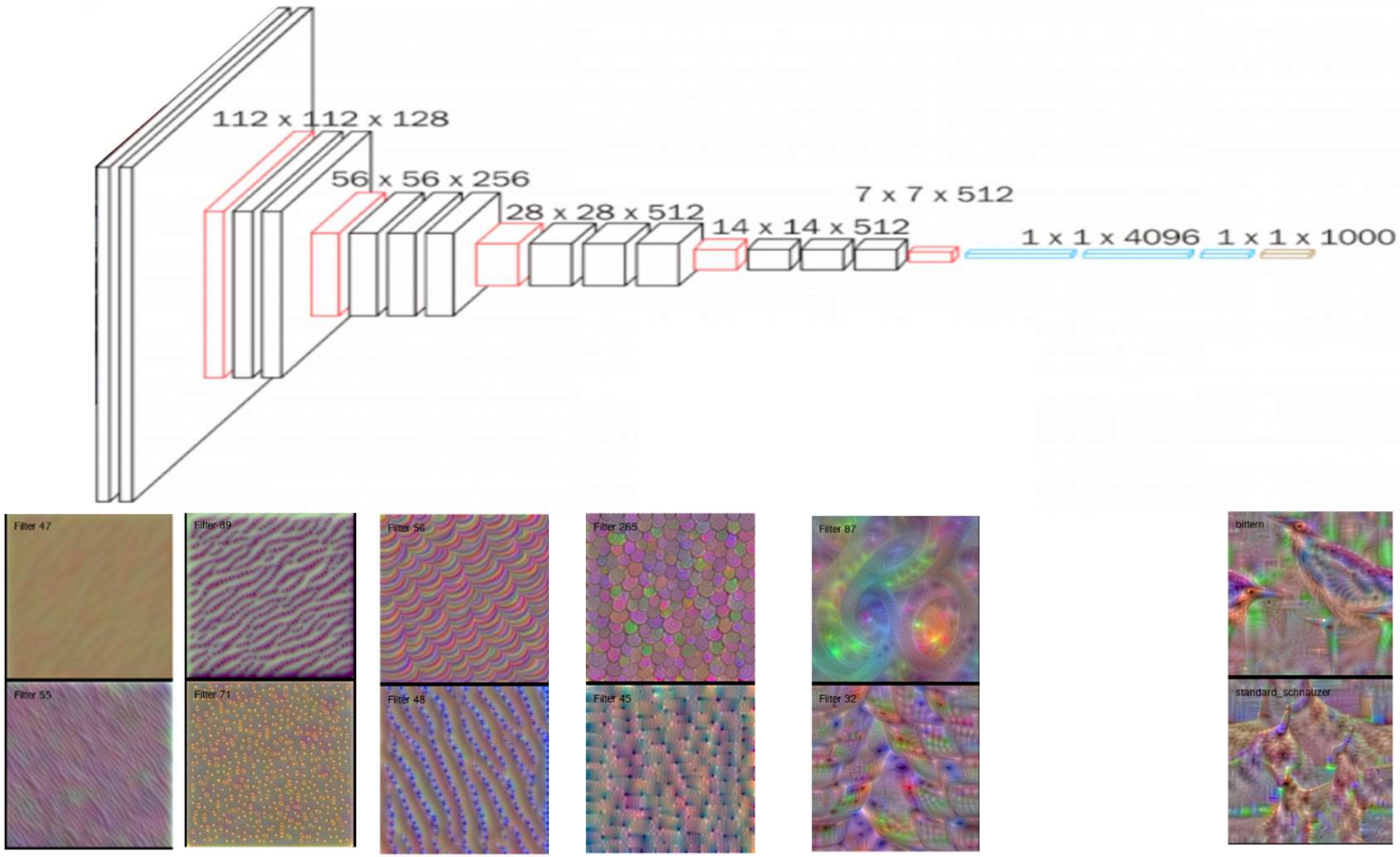
train

Task 2

2-class problem (not in ImageNet)

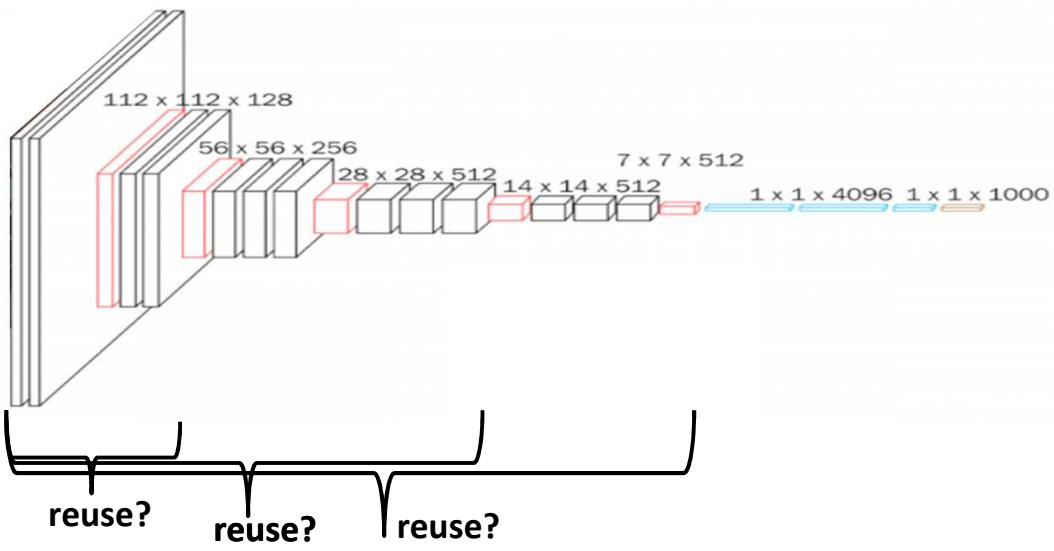
Deep Learning as Representation Learning

- When we train a deep neural network, we learn a hierarchy of increasingly complex features:



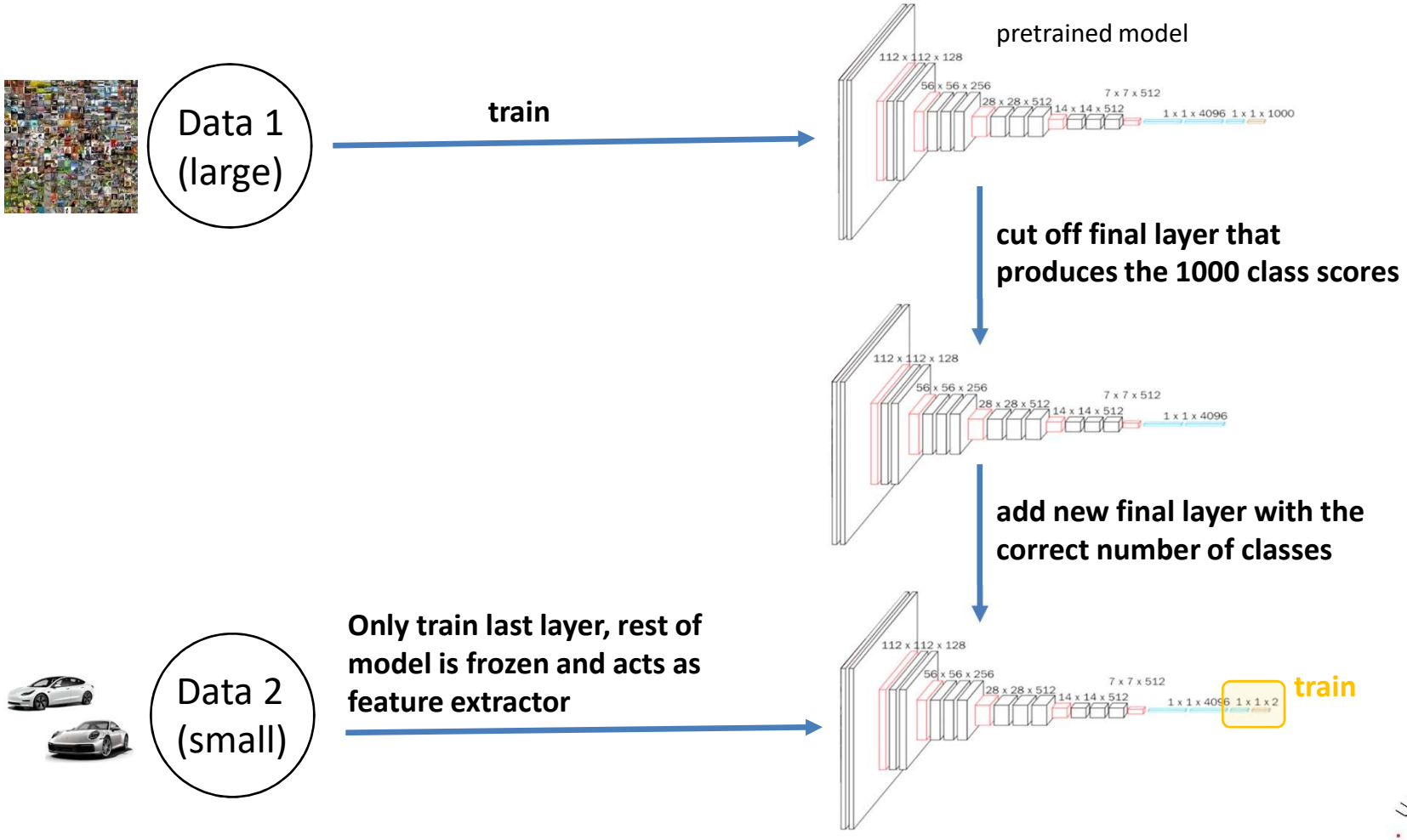
Deep Learning as Representation Learning

- Idea of transfer learning (or finetuning): if we have a new task, can we reuse the lower parts of the model?
  - low-level features such as edge detectors may be very similar for different tasks
  - even high-level features may be useful, if the tasks are similar enough
  - we can also start with the features learned on another task, and then further fine-tune for the current task



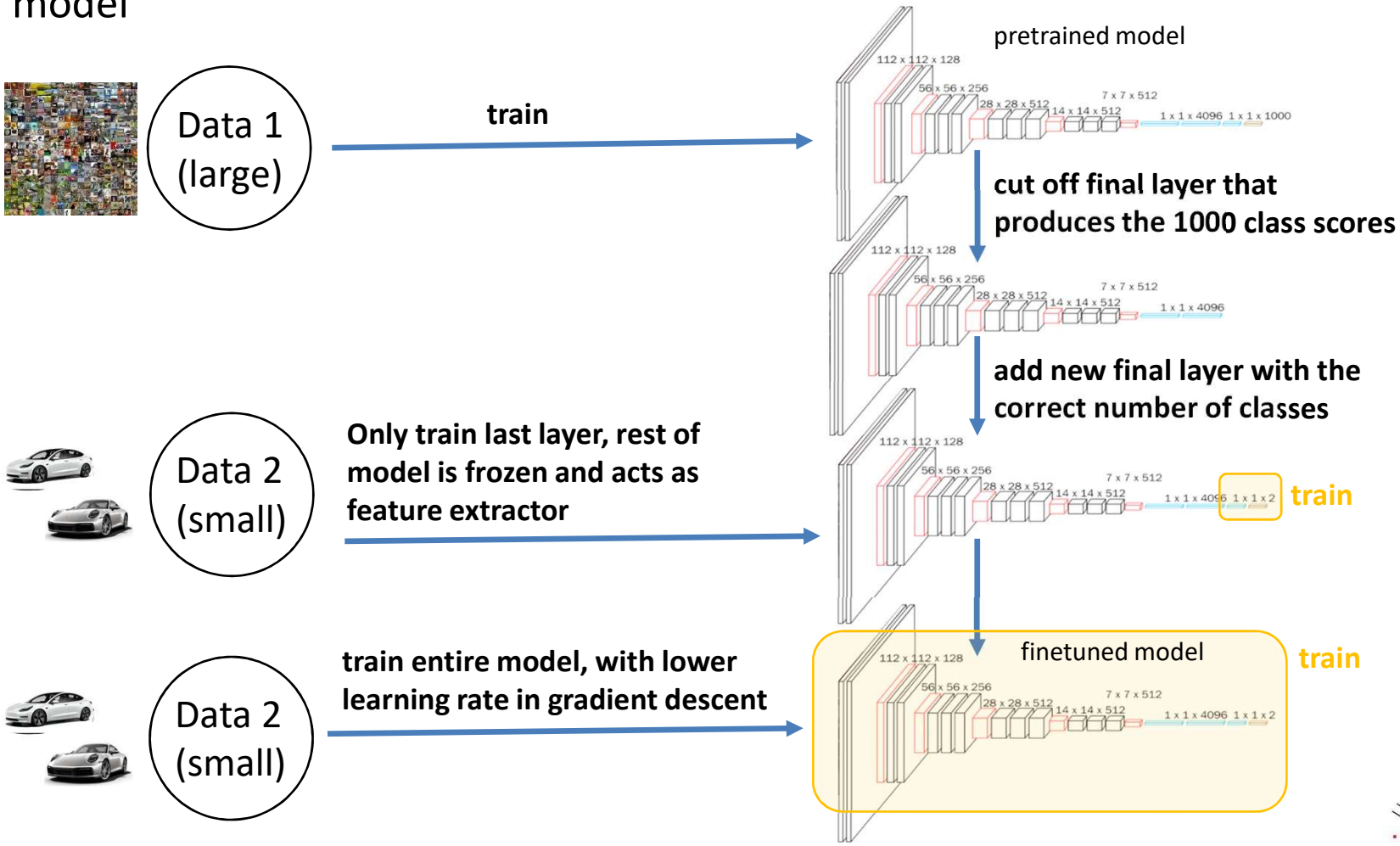
Existing Model as Feature Extractor

- Simplest approach: use existing neural network as feature extractor



Fine Tuning Pre-Trained Model

- Most widely used approach: retrain final layer and afterwards fine-tune entire model



Fine Tuning Pre-Trained Model

- Fine-tuning of models pretrained on large data sets (e.g. ImageNet) is a widely used approach for computer vision tasks where data is limited
  - By exploiting the useful low-level features learned on the large „source task“ (e.g. ImageNet), good models can be learned with little data
  - Training time is greatly reduced: training large models from scratch can take weeks, fine-tuning usually only takes hours

- Effectiveness of fine-tuning will depend on similarity of tasks and amount of training data available in „target task“ (the task we fine-tune on)
  - If tasks are similar and little target data is available, advantage is large
  - If tasks are very dissimilar or a lot of target data available, less helpful

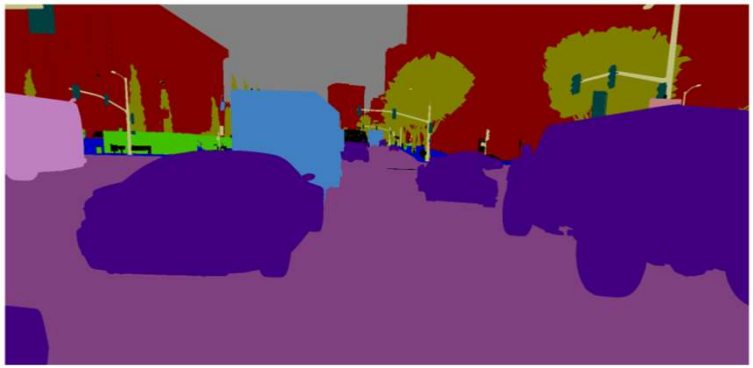
- Can even reuse (parts of) pretrained classification models in non-classification setting such as object detection or segmentation as feature extractors

Example: Pretraining on Artificial Data

- Example: using rendered traffic scenes for pretraining segmentation models

Cityscapes segmentation data set: labeling effort approx. 90 min/image


Pretrain with rendered scenes from open-world computer game (Grand Theft Auto V)

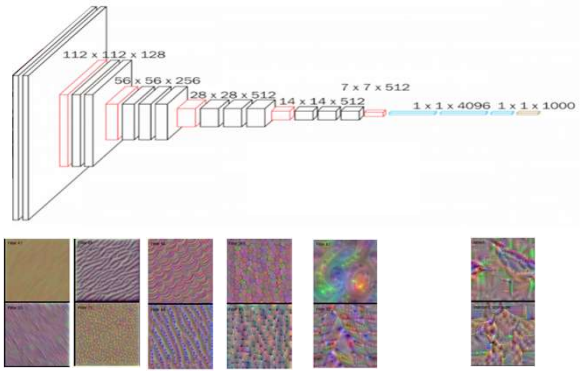



Richter, Stephan R., et al. "Playing for data: Ground truth from computer games." *European conference on computer vision*. Springer, Cham, 2016.

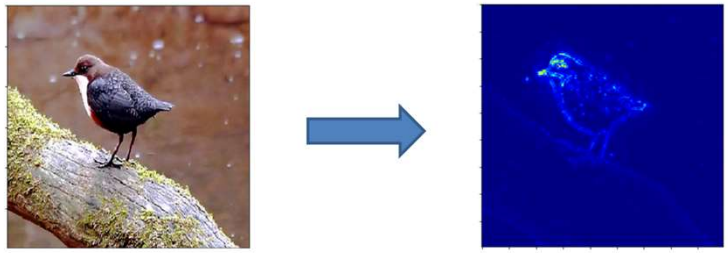
Summary

- Visualizing learned weights and classification decision in CNNs

Activation maximization: find out which input maximally activates a certain filter or class, to understand learned features



Saliency maps: understanding which part of image most important to prediction



input image

guided saliency map M

- Transfer learning: pretraining models on large data set and reusing/adapting the learned features for a novel computer vision task