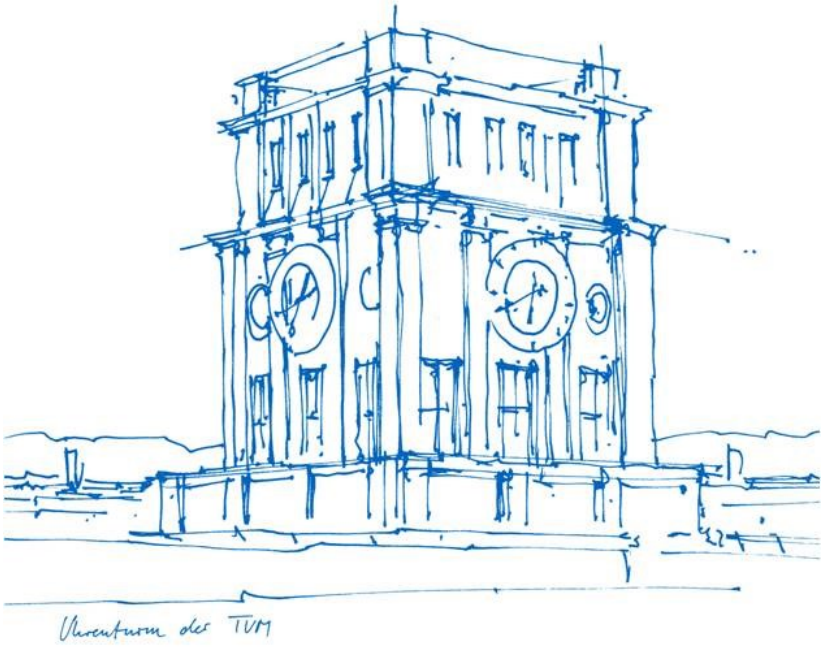
Machine Learning for Computer Vision

8. Neural Networks and Deep Learning

Vladimir Golkov

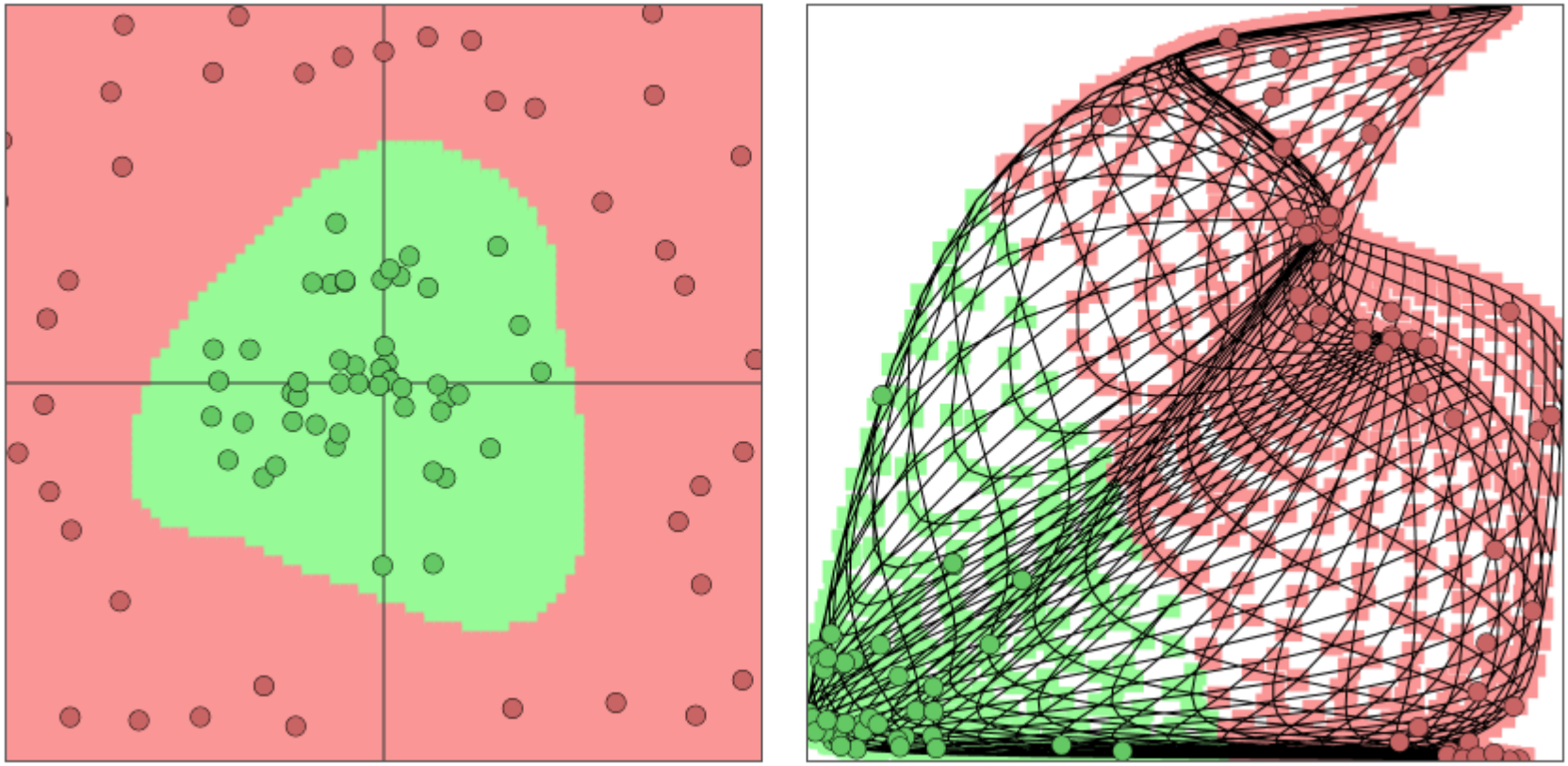
Technical University of Munich

Computer Vision Group



INTRODUCTION

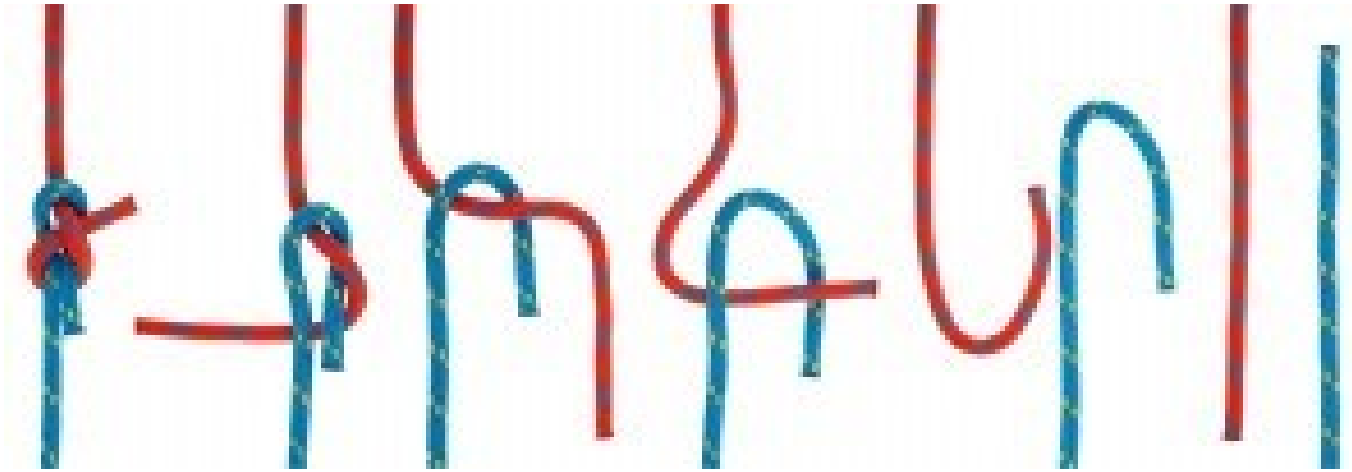
Nonlinear Coordinate Transformation



http://cs.stanford.edu/people/karpathy/convnetjs/

Dimensionality may change!

Deep Neural Network: Sequence of Many Simple Nonlinear Coordinate Transformations that “disentangle” the data



Data is sparse (almost lower-dimensional)

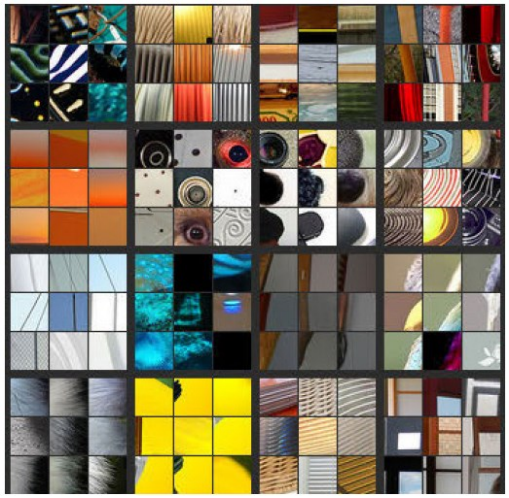
Linear separation of red and blue classes

The Increasing Complexity of Features

Input features: RGB

Layer 1

Feature space coordinates:
45° edge yes/no
Green patch yes/no
…



Layer 2

Layer 3

Feature space coordinates:
Person yes/no
Car wheel yes/no
…

[Zeiler & Fergus, ECCV 2014]

NEURAL NETWORKS

Fully-Connected Layer a.k.a. Dense Layer

$x^{(0)}$ is input feature vector for neural network (one sample).

$x^{(L)}$ is output vector of neural network with L layers.

Layer number l has:

- Inputs (usually $x^{(l-1)}$, i.e. outputs of layer number $l-1$)
- Weight matrix $W^{(l)}$, bias vector $b^{(l)}$ - both trained (e.g. with stochastic gradient descent) such that $x^{(L)}$ for the training samples minimizes some objective (loss)
- Nonlinearity s_l (fixed in advance, for example $\mathrm{ReLU}(z) \coloneqq \max\{0, z\}$)
- Output $x^{(l)}$ of layer l

Transformation from $x^{(l-1)}$ to $x^{(l)}$ performed by layer l:

$$x^{(l)} = s_l\left(W^{(l)}x^{(l-1)} + b^{(l)}\right)$$

Example

$$W^{(l)} = \begin{pmatrix} 0 & 0.1 & -1 \\ -0.2 & 0 & 1 \end{pmatrix}$$

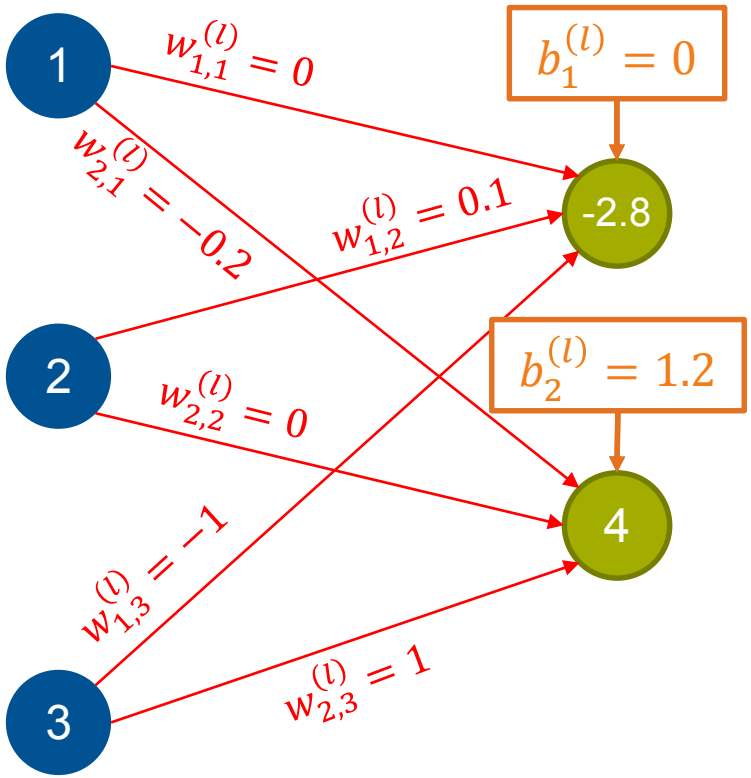
$$x^{(l-1)} = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$$

$$b^{(l)} = \begin{pmatrix} 0 \\ 1.2 \end{pmatrix}$$

$$W^{(l)}x^{(l-1)} + b^{(l)} =$$

$$= \begin{pmatrix} 0 \cdot 1 + 0.1 \cdot 2 - 1 \cdot 3 + 0 \\ -0.2 \cdot 1 + 0 \cdot 2 + 1 \cdot 3 + 1.2 \end{pmatrix}$$

$$= \begin{pmatrix} -2.8 \\ 4 \end{pmatrix}$$



2D Convolutional Layer

Appropriate for 2D structured data (e.g. images) where we want:

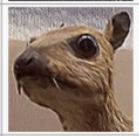
- Locality of feature extraction (far-away pixels do not influence local output)
- Translation-equivariance (shifting input in space (i, j dimensions) yields same output shifted in the same way)

$$x_{i,j,k}^{(l)} = s_l\left(b_k^{(l)} + \sum_{\hat{\imath},\hat{\jmath},\hat{k}} w_{i-\hat{\imath},j-\hat{\jmath},\hat{k},k}^{(l)} x_{i,j,\hat{k}}^{(l-1)}\right)$$

- the size of W along the i, j dimensions is called “filter size”
- the size of W along the $\hat{k}$ dimension is the number of input channels (e.g. three (red, green, blue) in first layer)
- the size of W along the k dimension is the number of filters (number of output channels)

- Equivalent for 1D, 3D, ...

Handcrafted Convolutional Filters

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

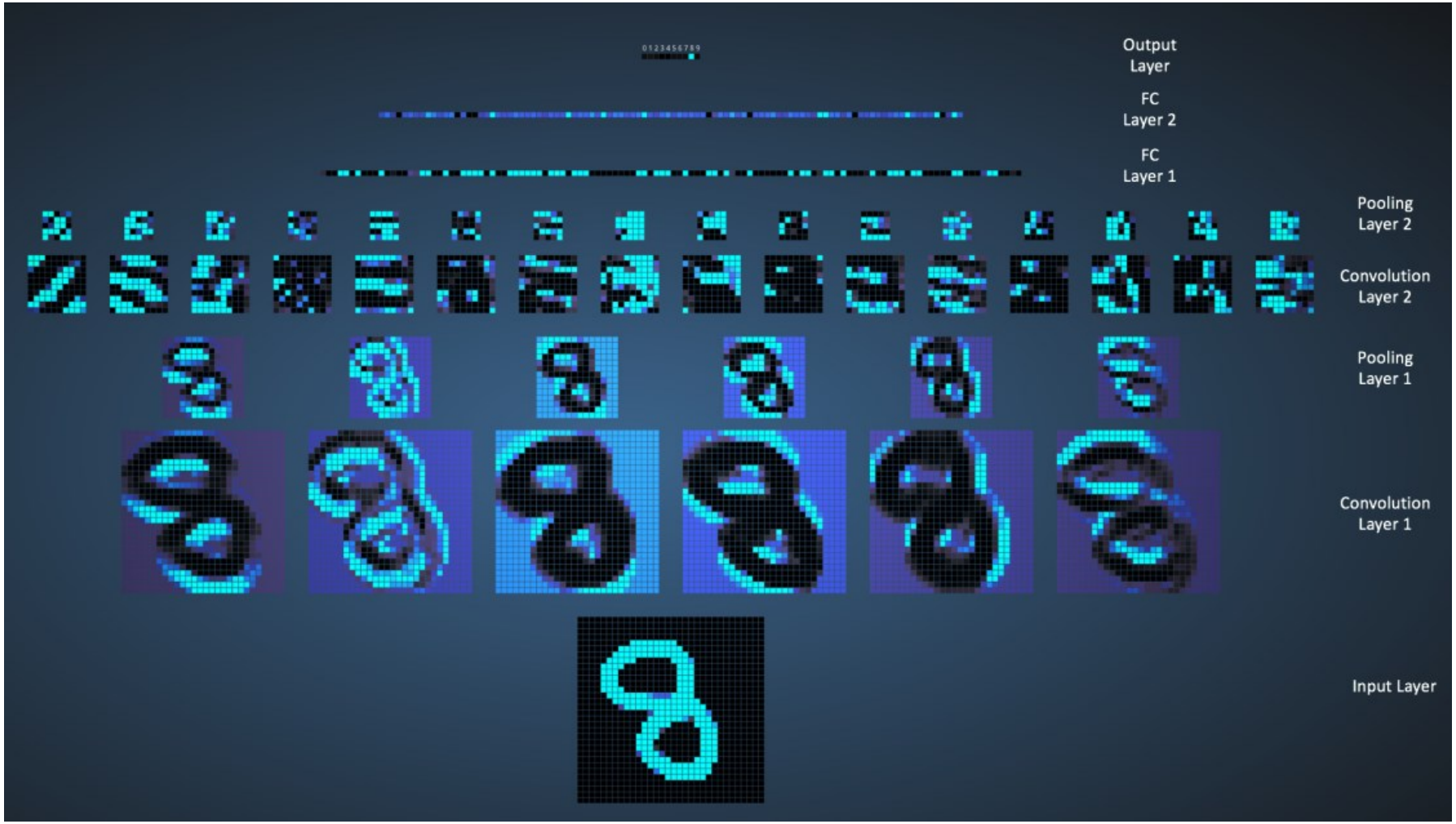
https://en.wikipedia.org/wiki/Kernel_(image_processing)

The shown filters are handcrafted

But filters that are learned by convolutional networks are optimal

- In terms of the training set and the final loss
- In the context of all network layers (weights of all layers optimized jointly)

Convolutional Network



Interactive: http://scs.ryerson.ca/~aharley/vis/conv/flat.html

Loss Functions

N-class classification:
- N outputs
- nonlinearity in last layer: softmax
- loss: categorical cross-entropy between outputs $x^{(L)}$ and targets t (sum over all training samples)

2-class classification:
- 1 output
- nonlinearity in last layer: sigmoid
- loss: binary cross-entropy between outputs $x^{(L)}$ and targets t (sum over all training samples)

2-class classification (alternative formulation)
- 2 outputs
- nonlinearity in last layer: softmax
- loss: categorical cross-entropy between outputs $x^{(L)}$ and targets t (sum over all training samples)

Many regression tasks:
- linear output
- loss: mean squared error between outputs $x^{(L)}$ and targets t (sum over all training samples)

Neural Network Training Procedure

- Fix number L of layers
- Fix sizes of weight arrays and bias vectors
  - For a fully-connected layer, this corresponds to the “number of neurons”
- Fix nonlinearities
- Initialize weights and biases with random numbers
- Repeat:
  - Select mini-batch (i.e. small subset) of training samples
  - Compute the gradient of the loss with respect to all trainable parameters (all weights and biases)
    - Use chain rule (“error backpropagation”) to compute gradient for deeper layers
  - Perform a gradient-descent step (or similar) towards minimizing the error
    - (Called “stochastic” gradient descent because every mini-batch is a random subset of the entire training set)
  - “Early stopping”: Stop when loss on validation set starts increasing (to avoid overfitting)

GOOD PRACTICES

How to Avoid Overfitting

- hard constraints on model size (number of layers, number of weights) and connections
- weight constraints (e.g. convolutional networks: locality, translation-equivariance)
- pooling layers (downsampling of feature maps)
- additional loss terms
- random perturbations of training samples (noise, dropout, data augmentation)
- early stopping

“Do’s and Don’ts” of Data Representation

- The data representation should be natural
  (do not “outsource” known data transformations to the learning)
- Use meaningful features, especially complementary ones
- Data augmentation using natural assumptions

WHAT DID THE NETWORK LEARN ?

- Visualization: creative, no canonical way
- Look at standard networks to gain intuition

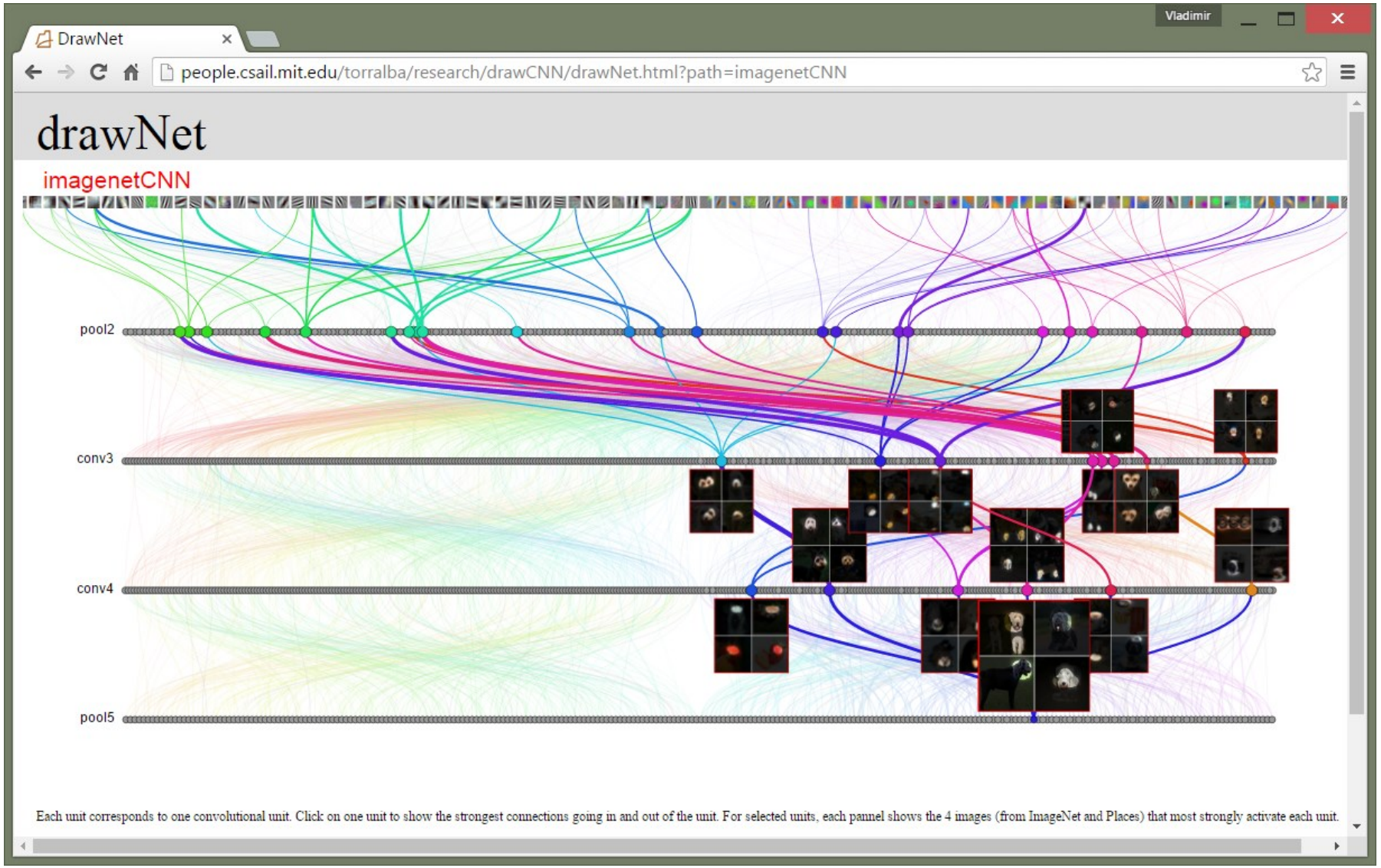
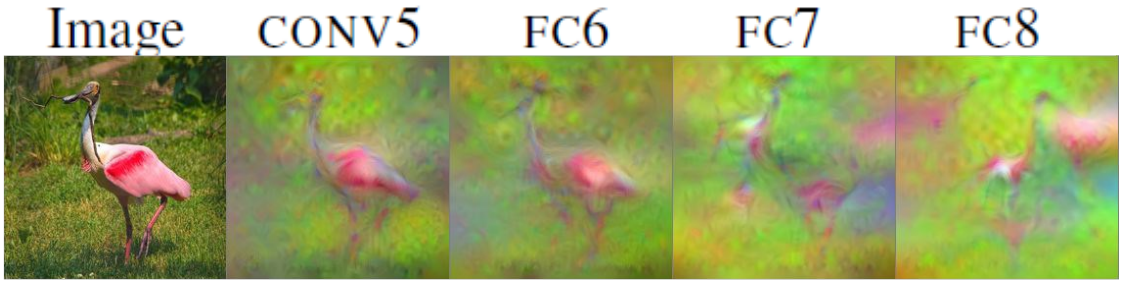
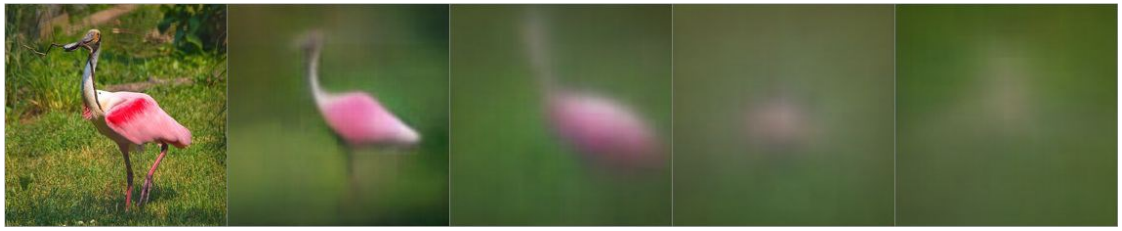


Image Reconstruction from Deep-Layer Features

[Mahendran & Vedaldi, CVPR 2015]



[Dosovitskiy & Brox, CVPR 2016]



[Dosovitskiy & Brox, NIPS 2016]

When Should Machine Learning Be Deep?

- Data distribution is complex

- Training data is abundant
  - (or can be augmented to abundance)

- Optionally: Data is structured
  - Neighborhood structure: convolutional networks (images, …)
  - Sequential structure: recurrent networks (memory, e.g. in time)

THANK YOU!