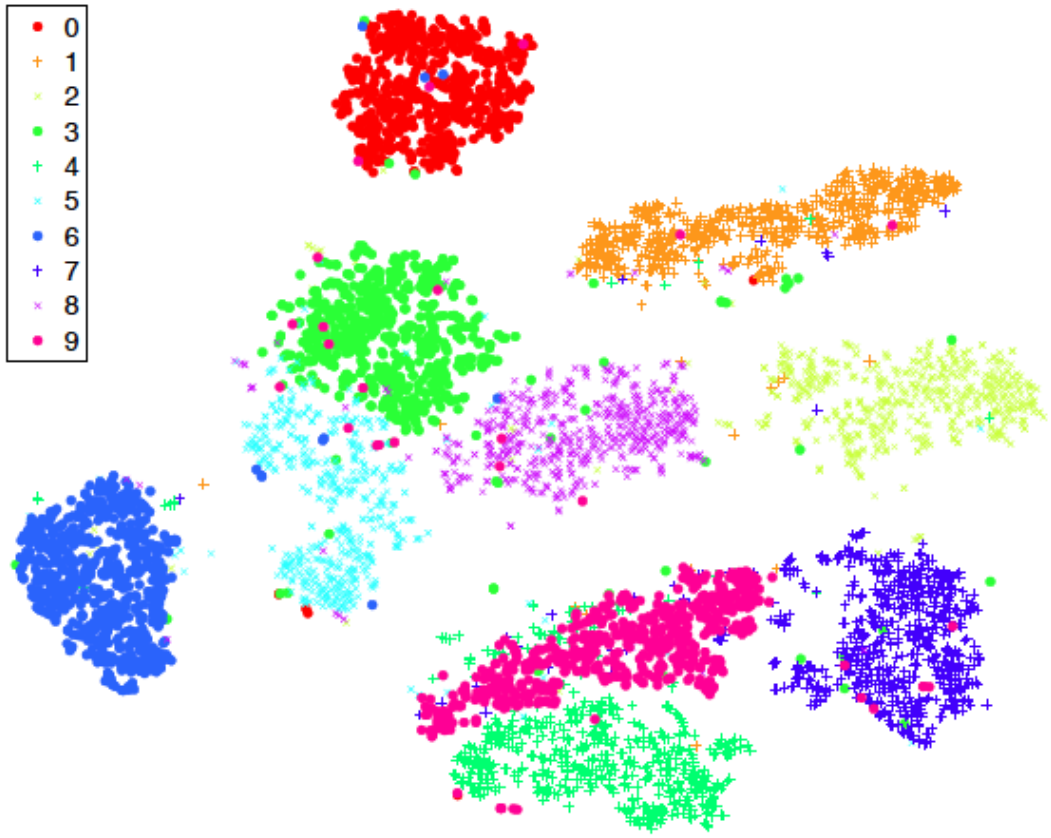
Visualizing Convnets

Pavan Ramkumar

(most material is adapted from cs231.stanford.edu)

Visualizing the codebook

t-SNE of the input space



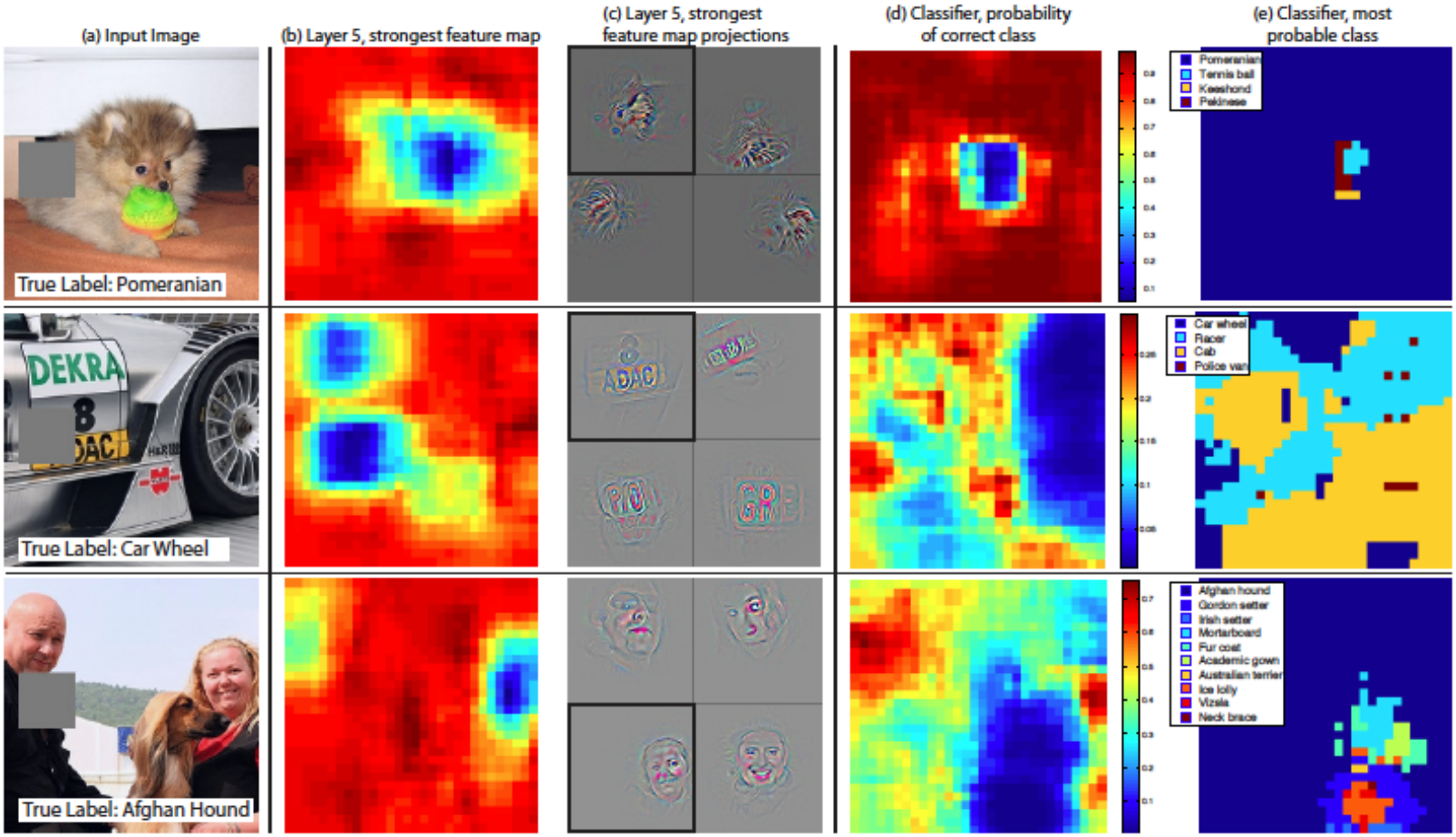
Van der Maaten & Hinton, 2008: t-SNE visualization of MNIST digits

t-SNE of top layer features

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

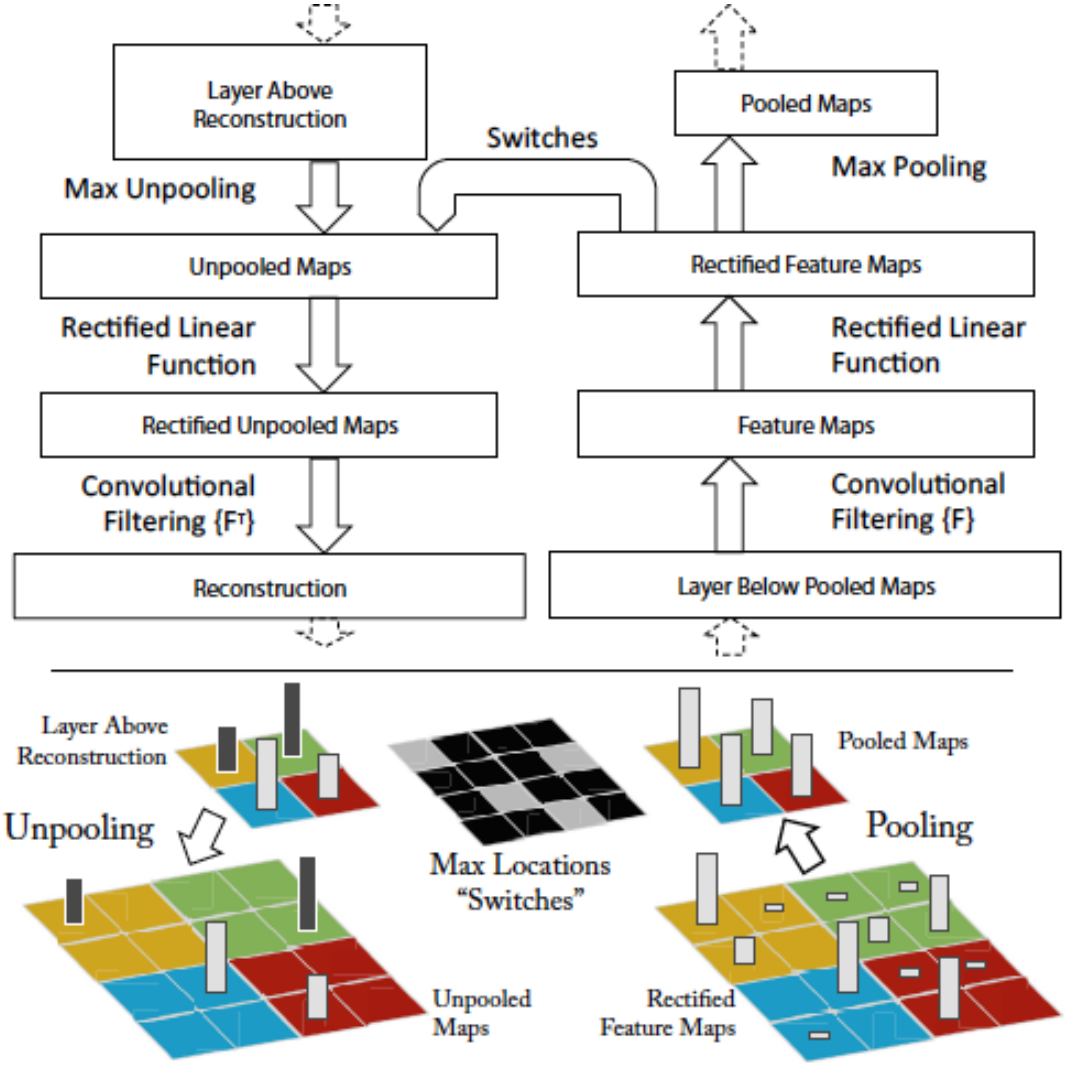
Visualization conditioned on image: What pixels inform classification?

Occlusion technique



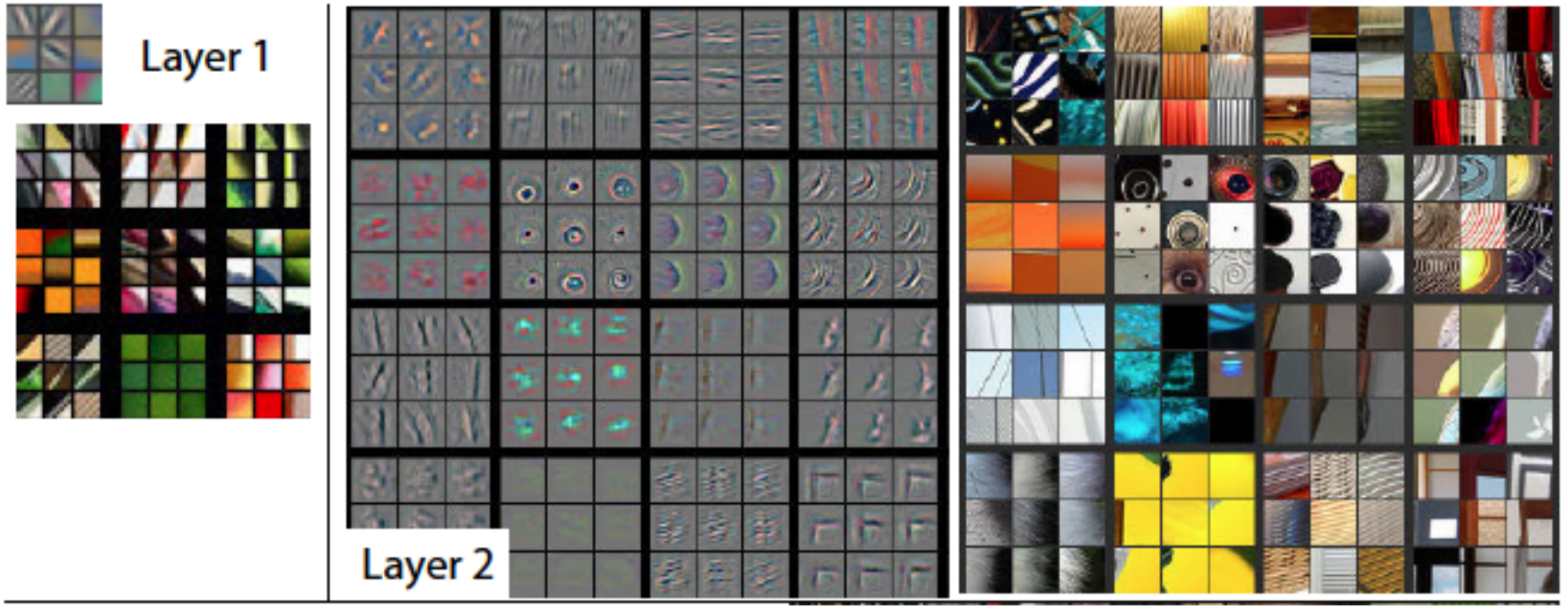
Zeiler & Fergus, 2014

Deconvolution



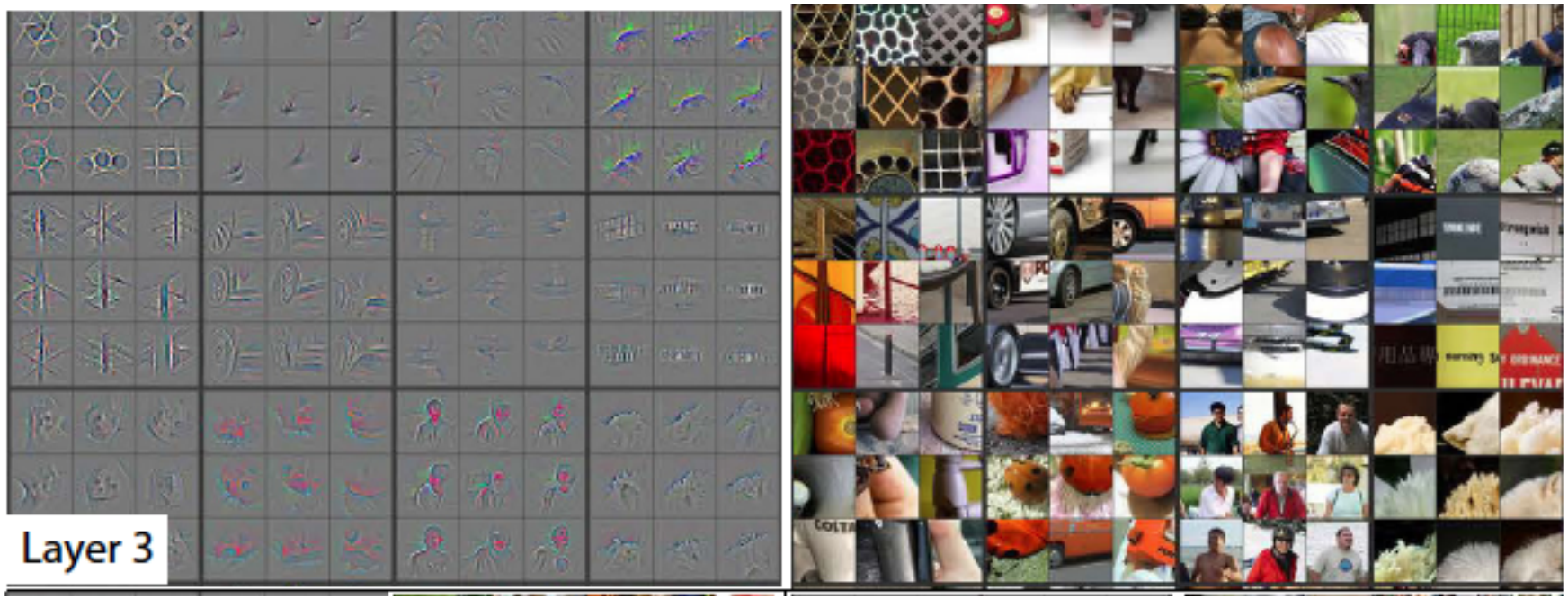
Zeiler & Fergus, 2014

Deconvolution



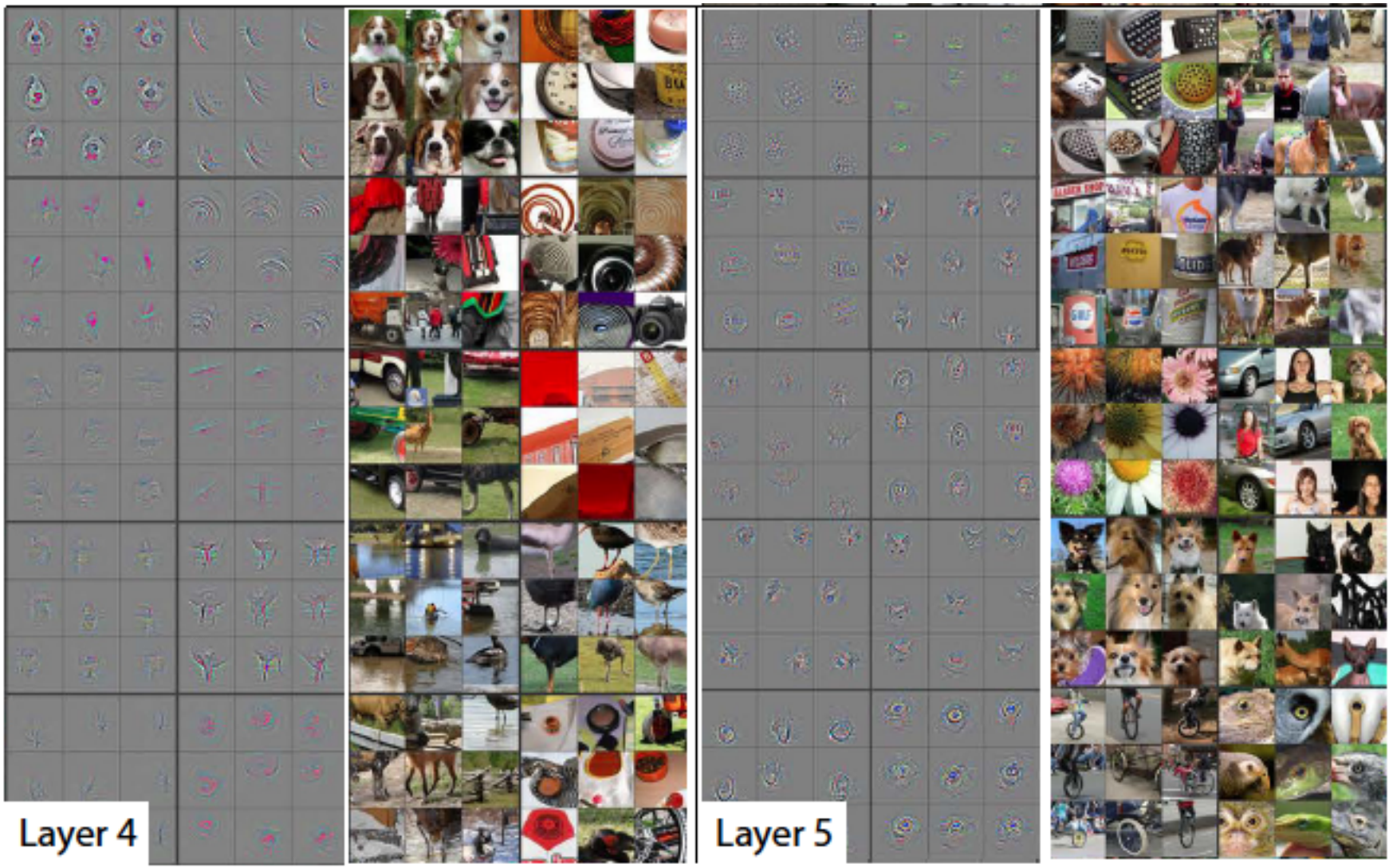
Zeiler & Fergus, 2014

Deconvolution



Zeiler & Fergus, 2014

Deconvolution



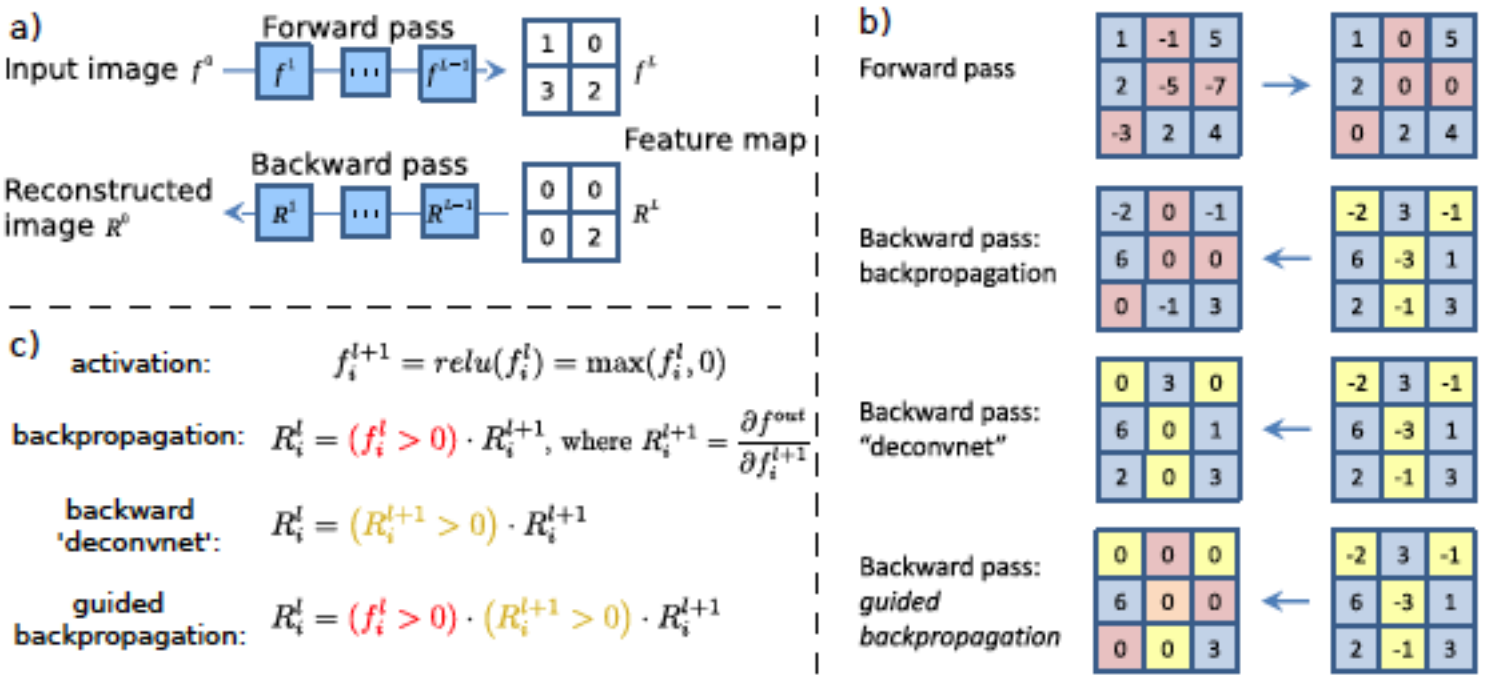
Zeiler & Fergus, 2014

Gradient technique

$$w = \left. \frac{\partial S_c}{\partial I} \right|_{I_0}$$

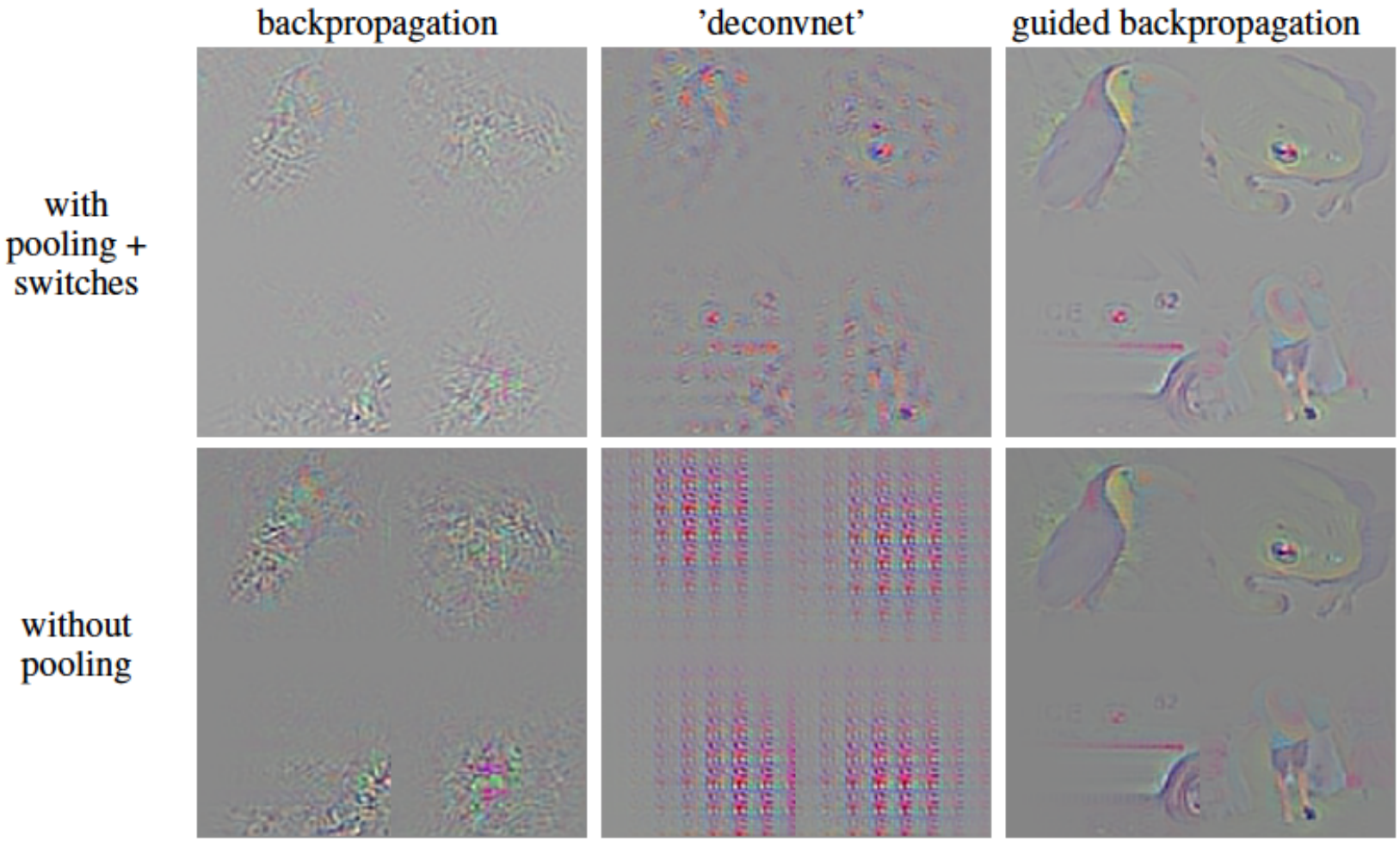
Visualizing pixels that are most informative about a class: Simonyan et al., 2014

Guided deconvolution



Springenberg, Dosovitskiy, 2015

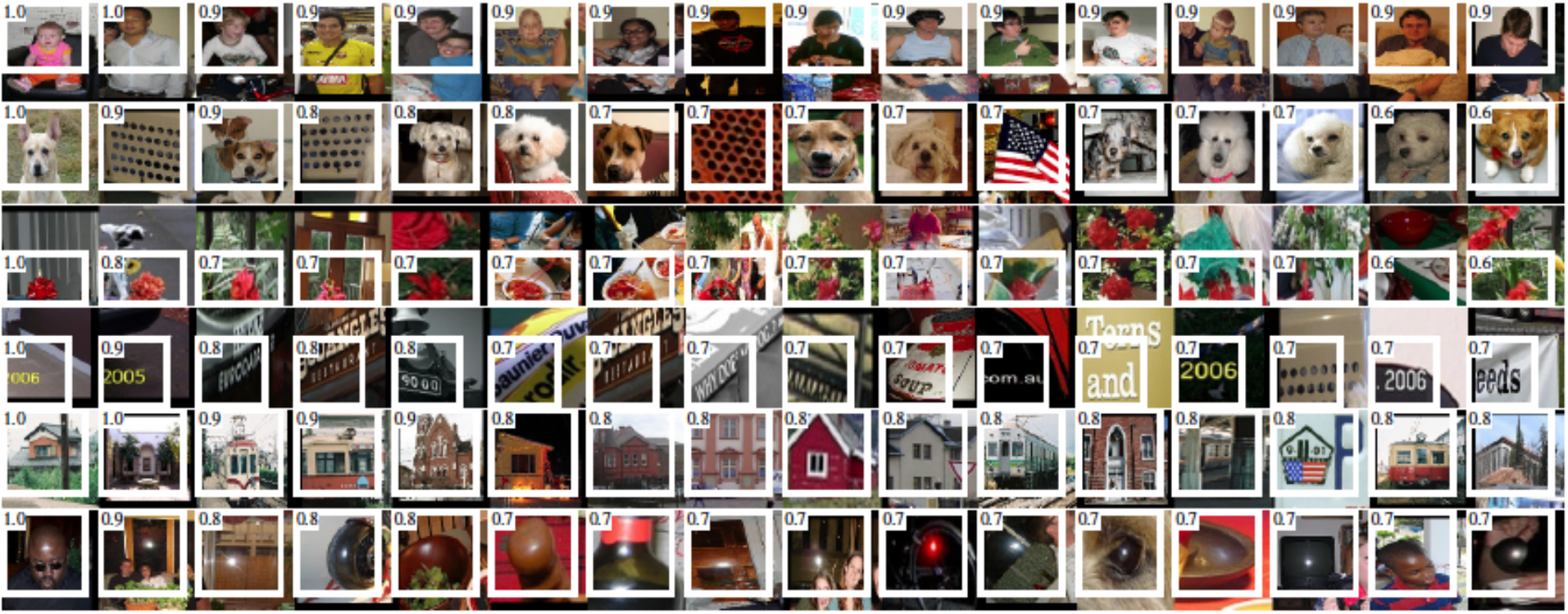
Guided deconvolution



Springenberg, Dosovitskiy, 2015

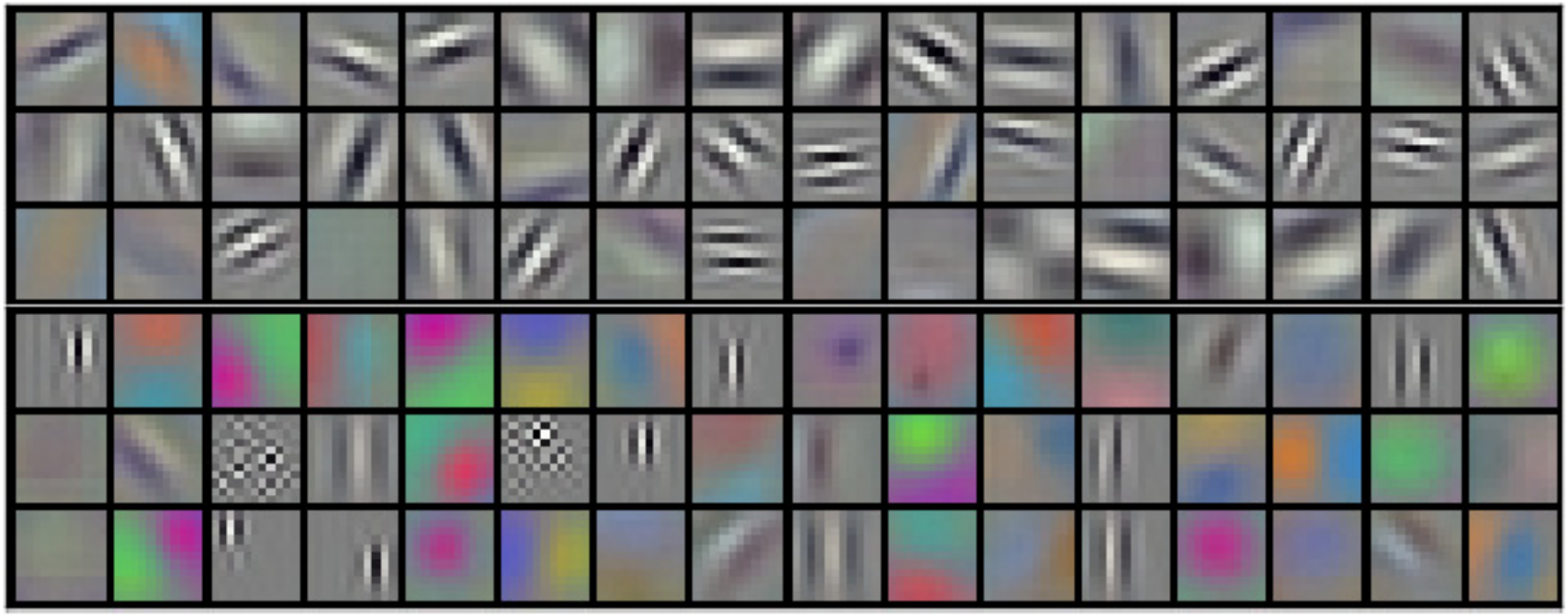
Visualizing Preferred Stimuli of Units: Unconditioned on image

Visualize images producing high activations



Examples of Preferred Images of Layer 5 Pool Units: Girshick et al., 2013

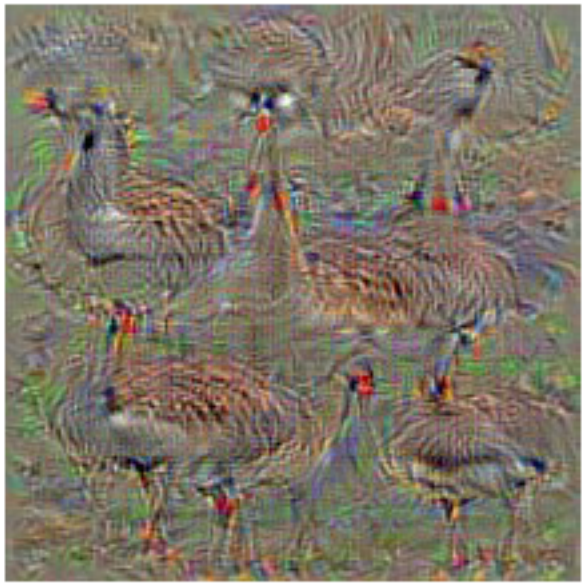
Visualize the weights: meaningful only for linear layer 1



Examples of weights of Layer 1 Units: Krizhevsky et al., 2012

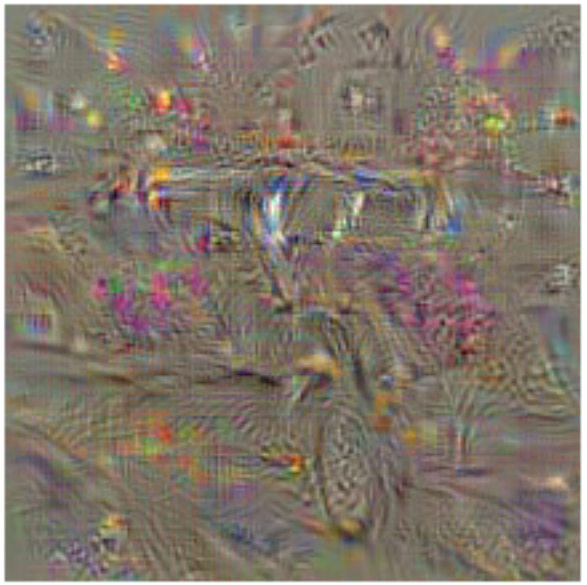
Gradient ascent on input image: L2 prior

$$\arg\max_I S_c(I) - \lambda \|I\|_2^2,$$



goose

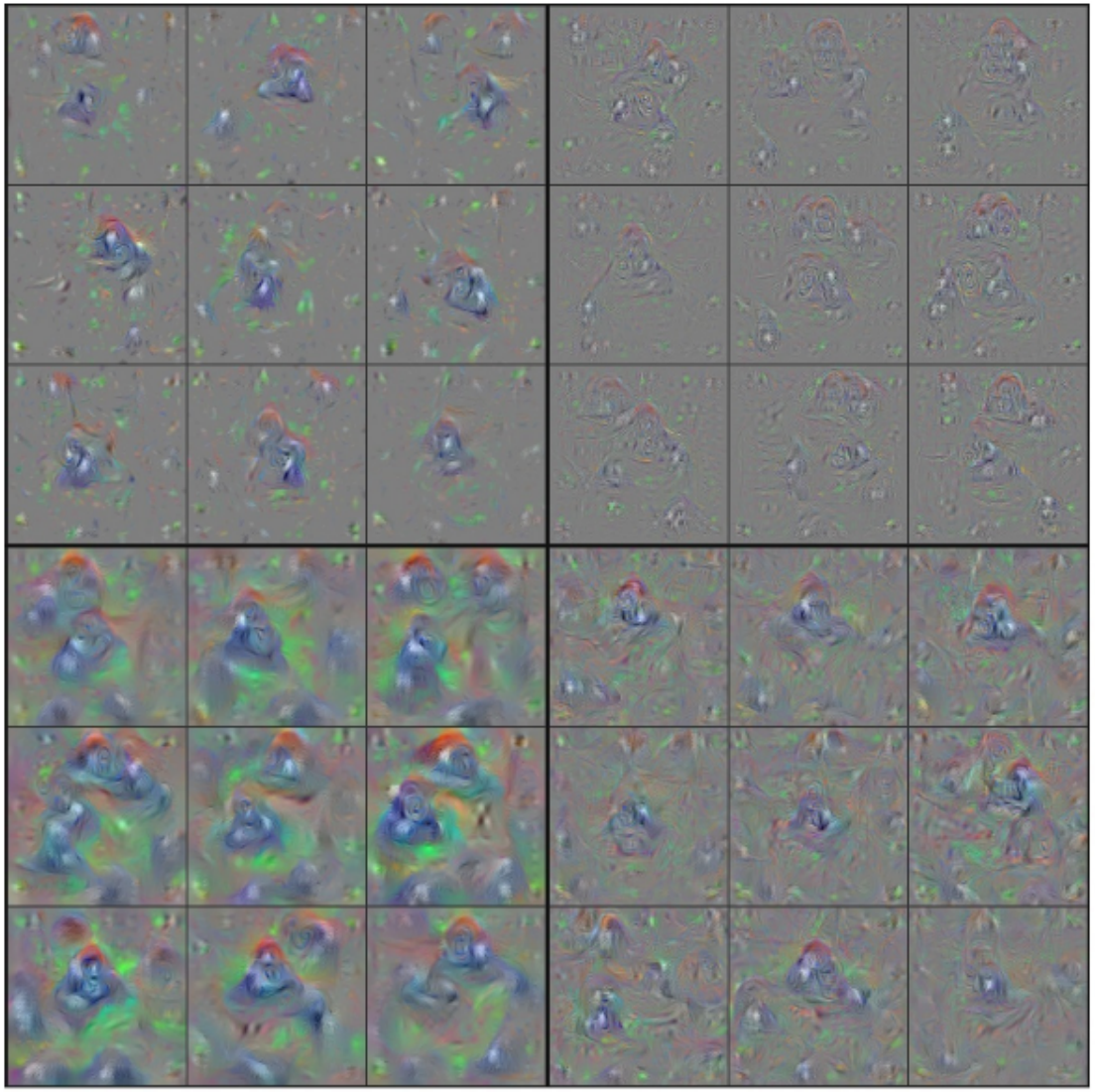
ostrich



limousine

Visualizing class neurons (inputs to softmax layer): Simonyan et al., 2014

Gradient ascent on input image: with "natural image" priors



Gorilla

- L2 norm
- Gaussian blur
- Clip pixels with small norm
- Clip pixels with small contribution
- Hyperparameter search

Yosinski et al., 2015

Visualizing the unique representations of layers

Start from random image.
Modify to approximate desired code: with "natural image" priors

$$\mathbf{x}^* = \underset{\mathbf{x}\in\mathbb{R}^{H\times W\times C}}{\operatorname{argmin}} \ \ell(\Phi(\mathbf{x}), \Phi_0) + \lambda\mathcal{R}(\mathbf{x})$$

$$\mathcal{R}_\alpha(\mathbf{x}) = \|\mathbf{x}\|_\alpha^\alpha \qquad \mathcal{R}_{V^\beta}(\mathbf{x}) = \sum_{i,j} \left((x_{i,j+1} - x_{ij})^2 + (x_{i+1,j} - x_{ij})^2 \right)^{\frac{\beta}{2}}$$

$$\|\Phi(\sigma\mathbf{x}) - \Phi_0\|_2^2 / \|\Phi_0\|_2^2 + \lambda_\alpha \mathcal{R}_\alpha(\mathbf{x}) + \lambda_{V^\beta} \mathcal{R}_{V^\beta}(\mathbf{x})$$

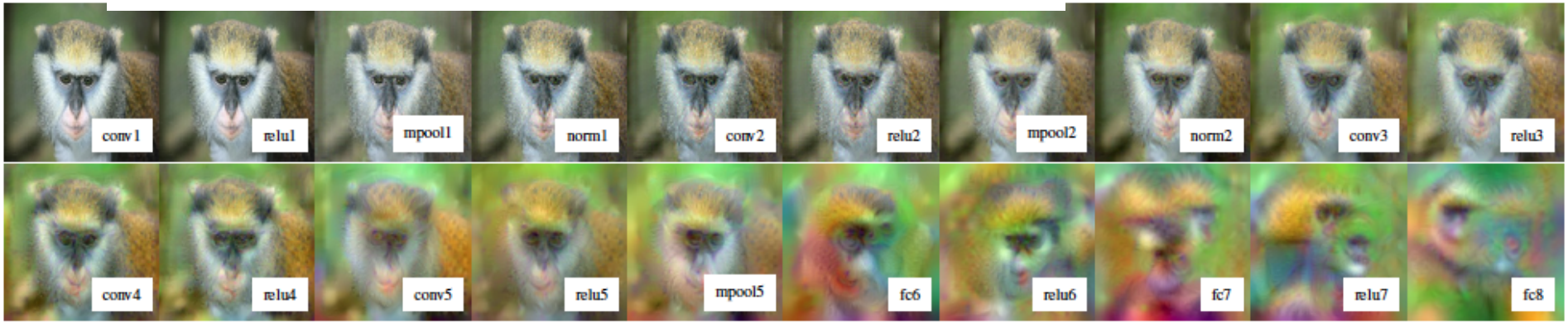


Figure 6. **CNN reconstruction.** Reconstruction of the image of Fig. 5.a from each layer of CNN-A. To generate these results, the regularization coefficient for each layer is chosen to match the highlighted rows in table 3. This figure is best viewed in color/screen.

- Alpha-norm, alpha >> 1 (they used 6)
- TV regularization, with beta > 1 (they used 2)

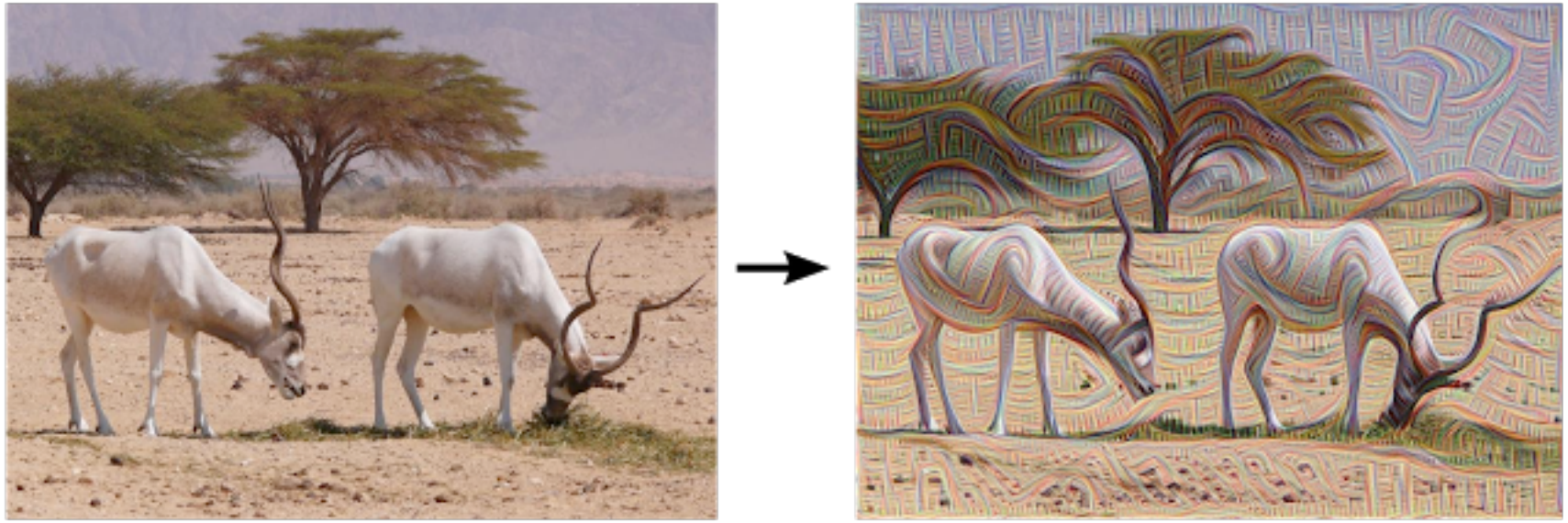
Mahendran & Vedaldi, 2014

Start from random image.
Modify to approximate desired code: with "natural image" priors

Figure 11. **Diversity in the CNN model.** mpool5 reconstructions show that the network retains rich information even at such deep levels. This figure is best viewed in color/screen (zoom in).

Mahendran & Vedaldi, 2014

Inceptionism/ Deep Dream: Modify pixels to boost activation in a given layer



http://googleresearch.blogspot.com/2015/06/inceptionism-going-deeper-into-neural.html

Deepart

Gatys et al., 2015
www.deepart.io

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

Deepart

Gatys et al., 2015
www.deepart.io

Fooling a net

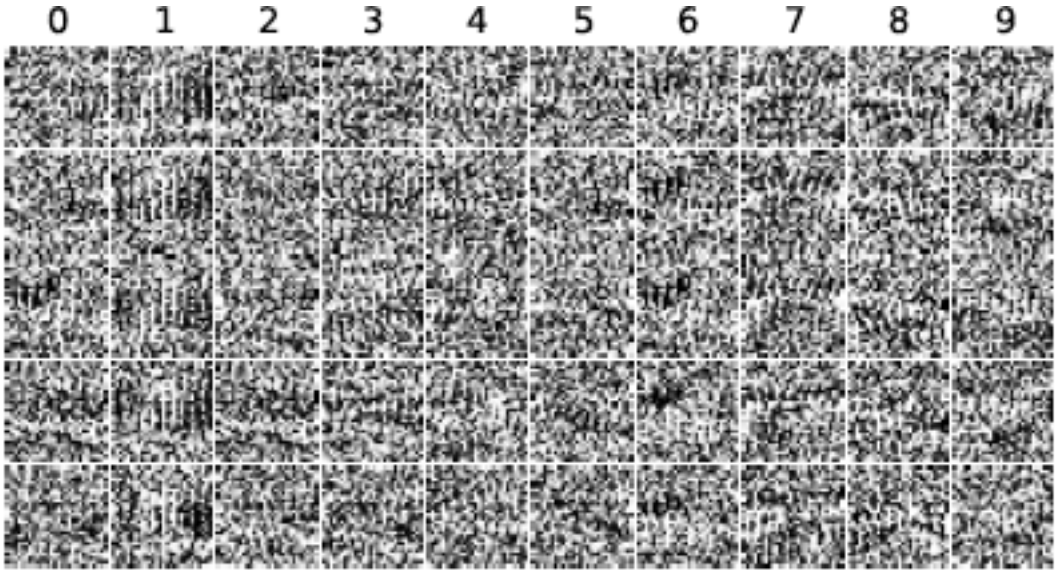


Figure 4. Directly encoded, thus irregular, images that MNIST DNNs believe with 99.99% confidence are digits 0-9. Each column is a digit class, and each row is the result after 200 generations of a randomly selected, independent run of evolution.

Nguyen et al., 2015

Fooling a net

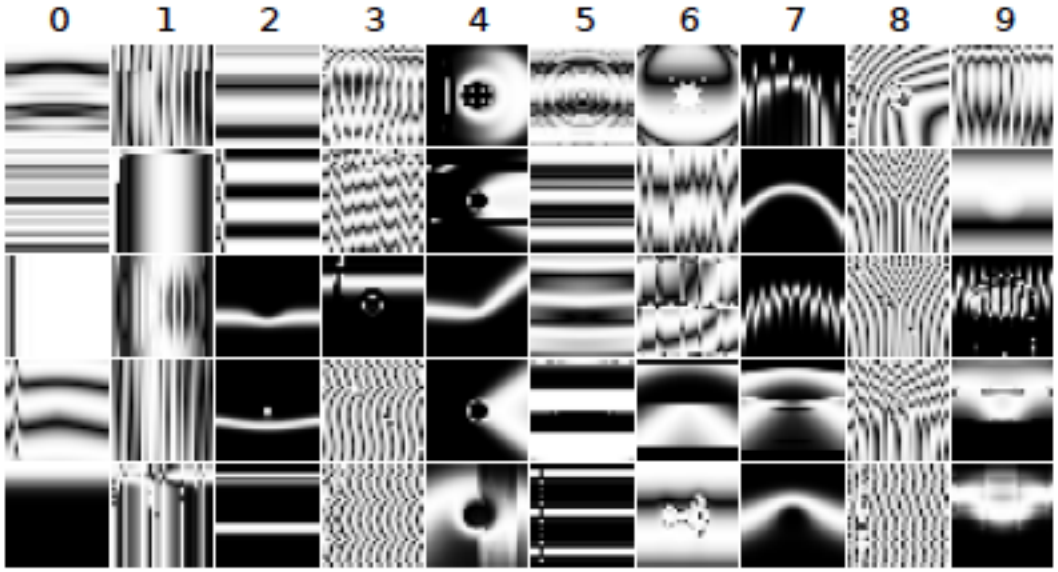


Figure 5. Indirectly encoded, thus regular, images that MNIST DNNs believe with 99.99% confidence are digits 0-9. The column and row descriptions are the same as for Fig. 4.

Nguyen et al., 2015

Torch demo

Getting started

- Scientific computing framework based on Lua [JIT]
- Efficient tensor library (like Numpy)
- Neural networks package lets you build arbitrary acyclic computational graphs with automatic differentiation
- It has multi-CPU and multi-GPU CUDA backends

- Install Torch: http://torch.ch/docs/getting-started.html
- Clone Facebook's iTorch [iPython kernel for torch with plotting, video, and audio]: https://github.com/facebook/iTorch
- Install using luarocks – a package manager for lua