ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
Ilya Sutskever
Geoffrey Hinton

University of Toronto
Canada

Paper with same name to appear in NIPS 2012

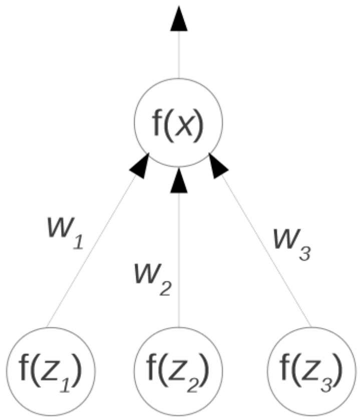
Main idea

Architecture

Technical details

Neural networks

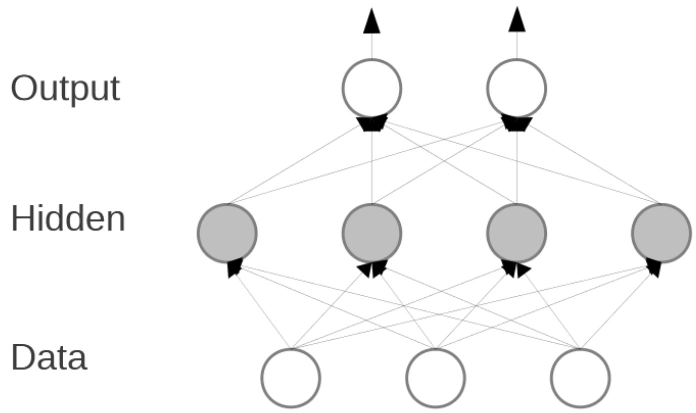
- A neuron



$x = w_1 f(z_1) + w_2 f(z_2) + w_3 f(z_3)$

x is called the total input to the neuron, and f(x) is its output

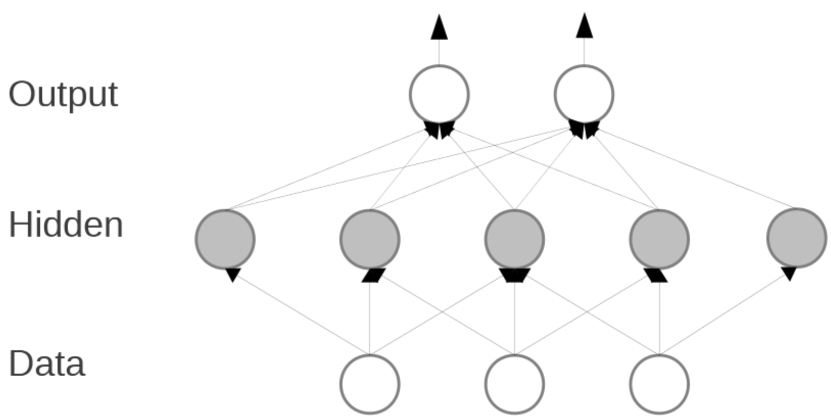
- A neural network



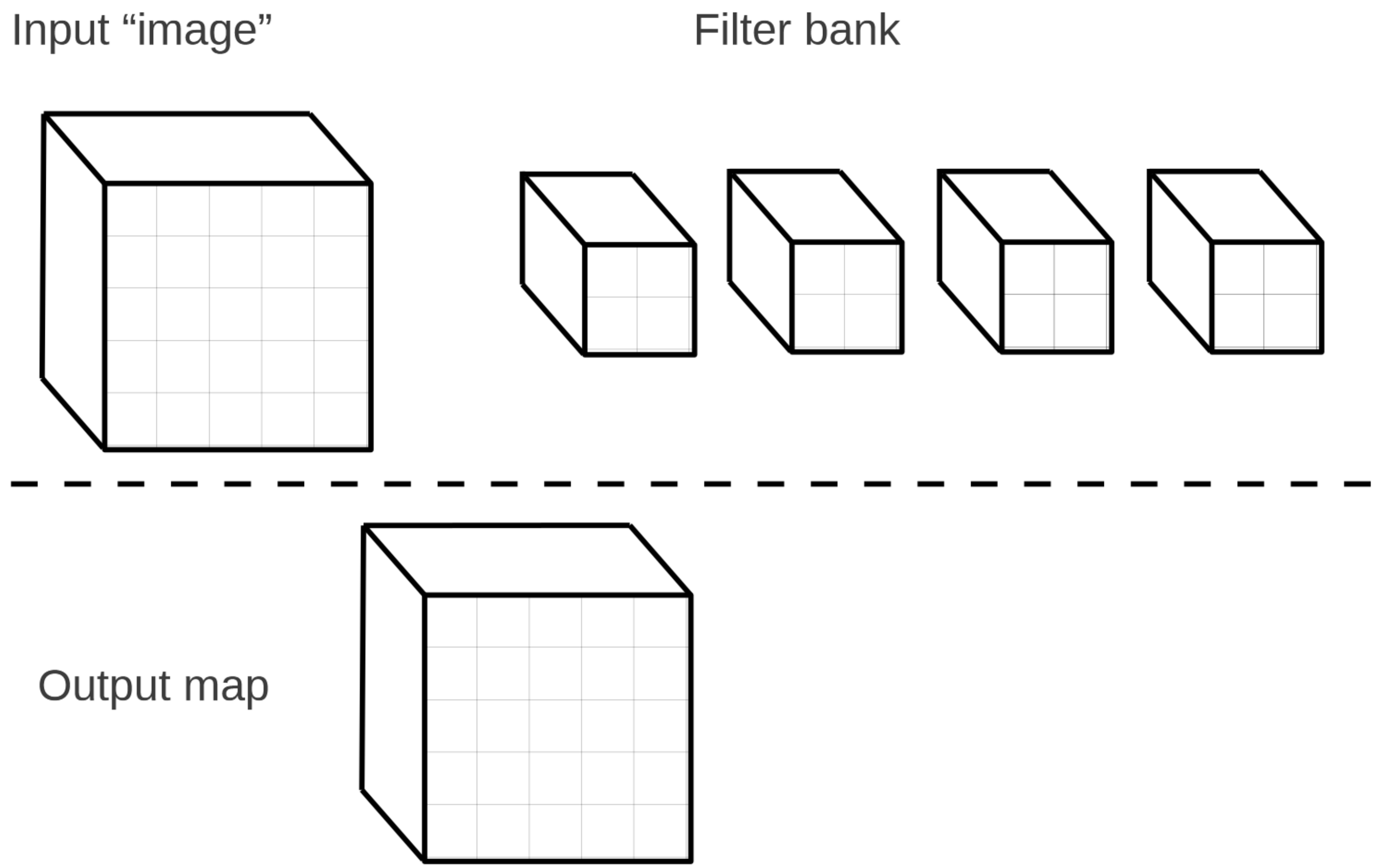
A neural network computes a differentiable function of its input. For example, ours computes: p(label | an input image)

Convolutional neural networks

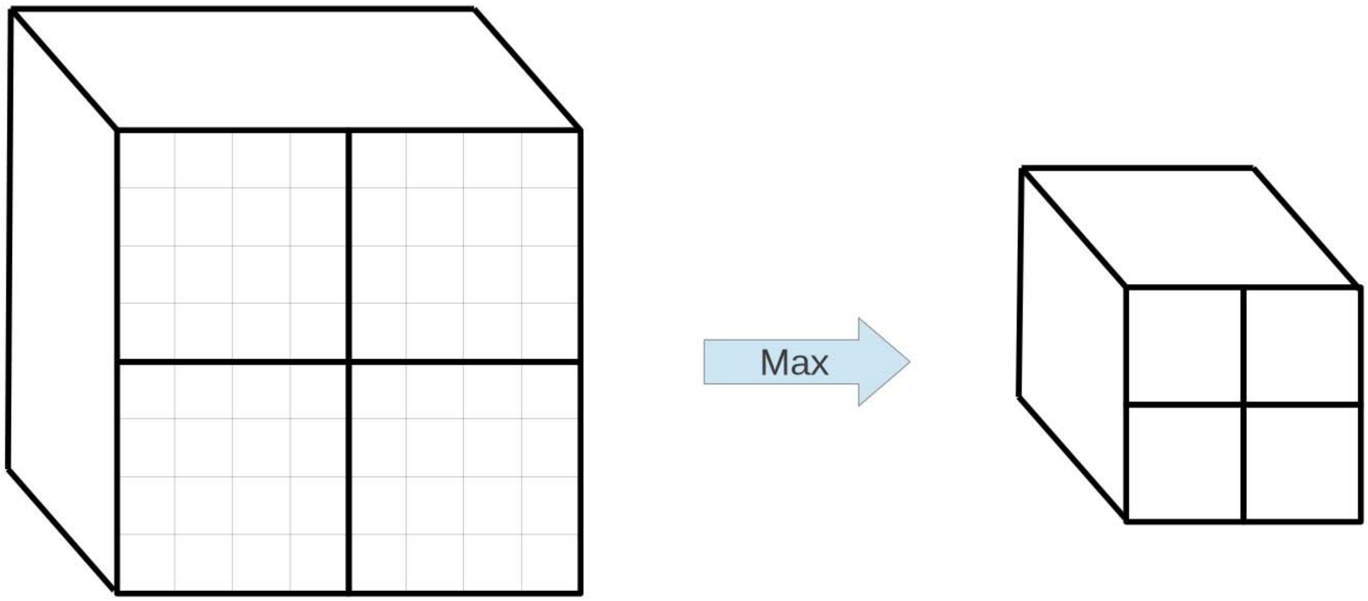
- Here's a one-dimensional convolutional neural network
- Each hidden neuron applies **the same localized, linear filter** to the input



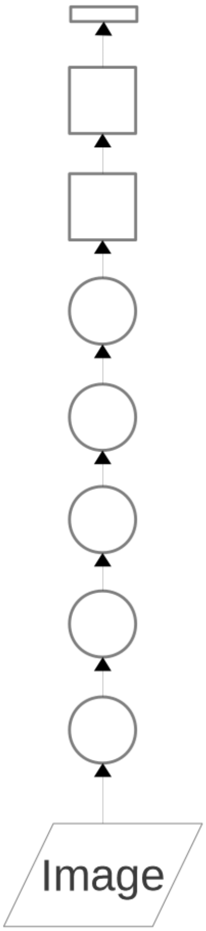
Convolution in 2D



Local pooling



Overview of our model

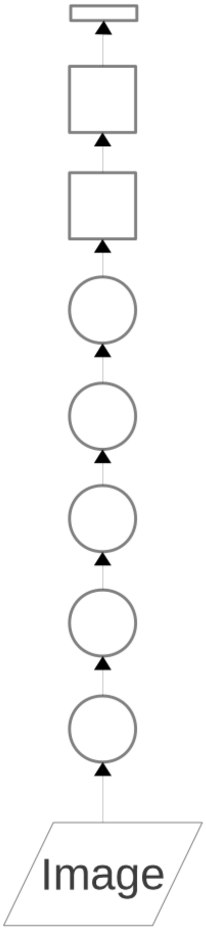


- **Deep**: 7 hidden “weight” layers
- **Learned**: all feature extractors initialized at white Gaussian noise and learned from the data
- Entirely supervised
- **More data = good**

Convolutional layer: convolves its input with a bank of 3D filters, then applies point-wise non-linearity

Fully-connected layer: applies linear filters to its input, then applies point-wise non-linearity

Overview of our model

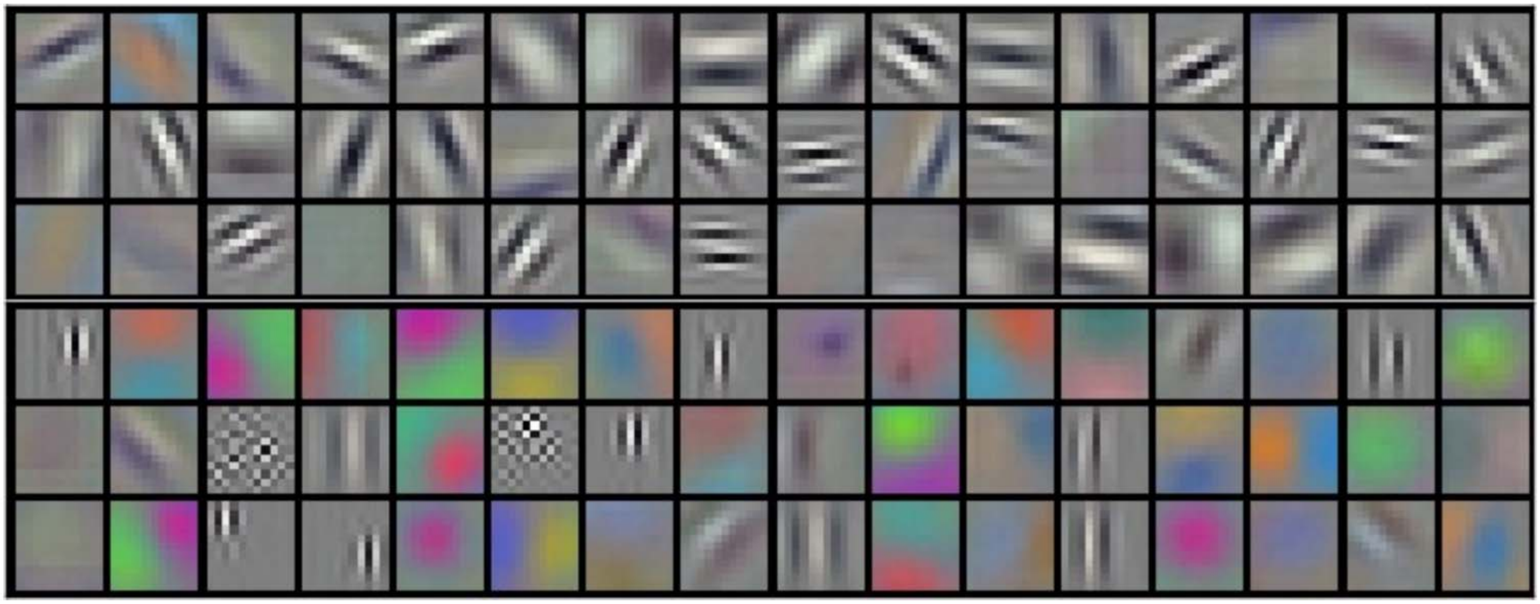


- Trained with stochastic gradient descent on two NVIDIA GPUs for about a week
- 650,000 neurons
- 60,000,000 parameters
- 630,000,000 connections
- **Final feature layer:** 4096-dimensional

Convolutional layer: convolves its input with a bank of 3D filters, then applies point-wise non-linearity

Fully-connected layer: applies linear filters to its input, then applies point-wise non-linearity

96 learned low-level filters



Main idea

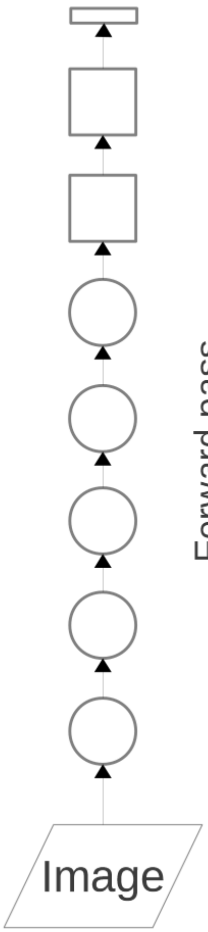
Architecture

Technical details

Local convolutional filters

Fully-connected filters

Training



Forward pass

Using stochastic gradient descent and the *backpropagation algorithm* (just repeated application of the chain rule)

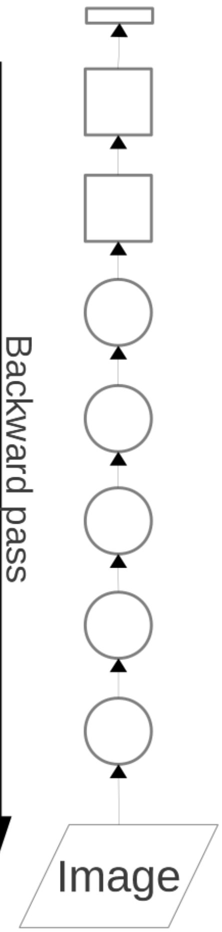
One output unit per class

x_i = total input to output unit i

$$f(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{1000} \exp(x_j)}$$

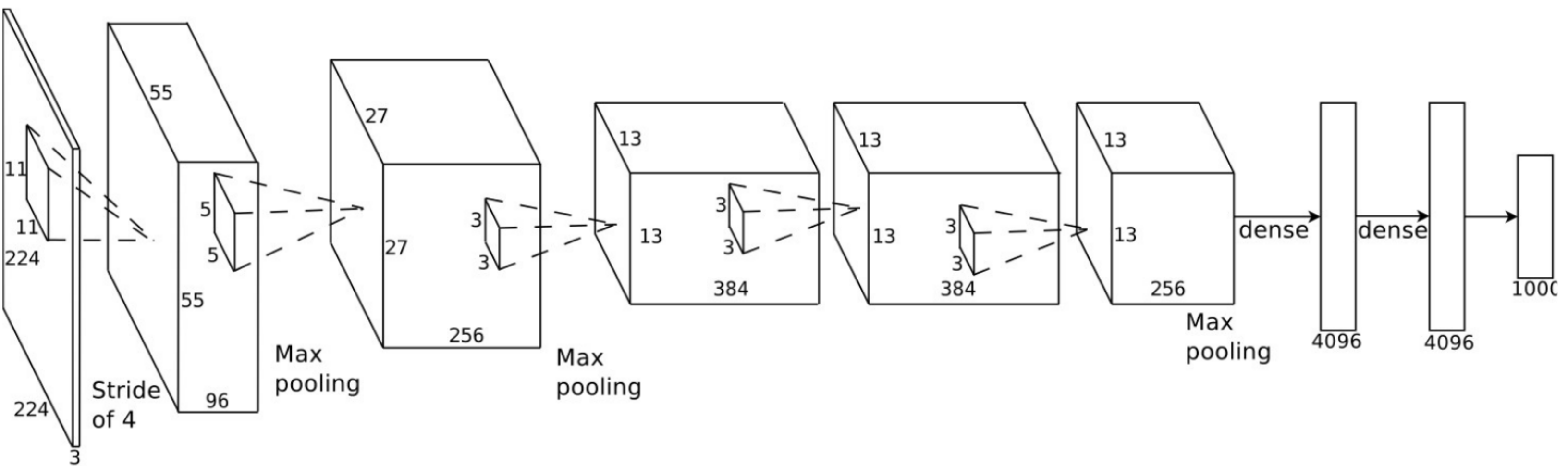
We maximize the log-probability of the correct label, $\log f(x_t)$

Backward pass



Our model

- Max-pooling layers follow first, second, and fifth convolutional layers
- The number of neurons in each layer is given by 253440, 186624, 64896, 64896, 43264, 4096, 4096, 1000



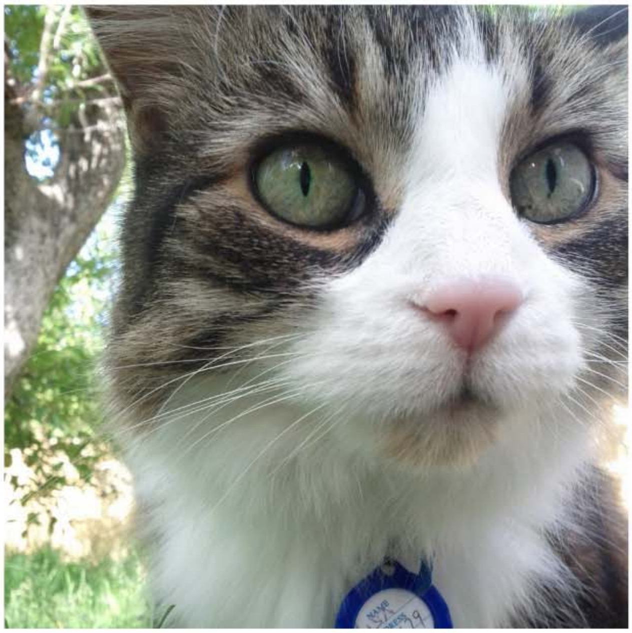
Main idea

Architecture

Technical details

Input representation

- Centered (0-mean) RGB values.



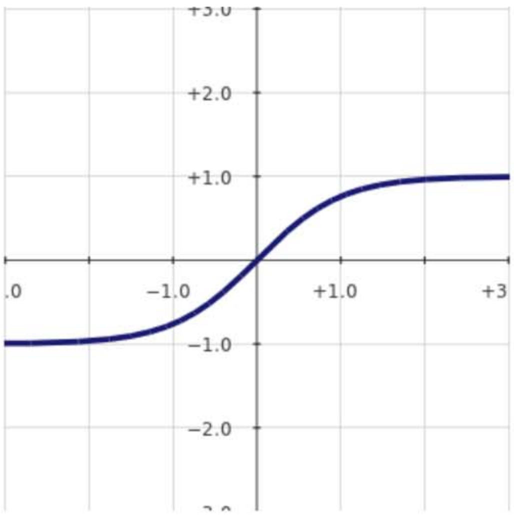
An input image (256x256)

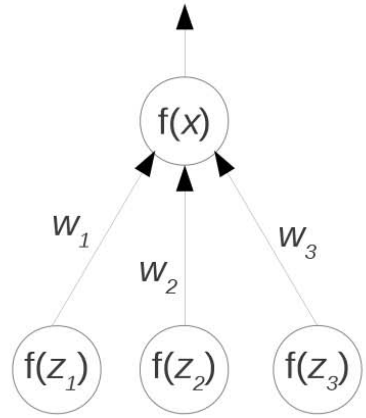
Minus sign

The mean input image

Neurons

$f(x) = \tanh(x)$

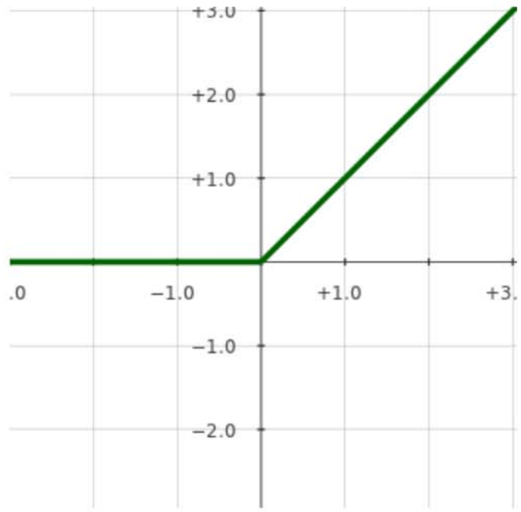




$$x = w_1 f(z_1) + w_2 f(z_2) + w_3 f(z_3)$$

x is called the total input to the neuron, and $f(x)$ is its output

$f(x) = \max(0, x)$

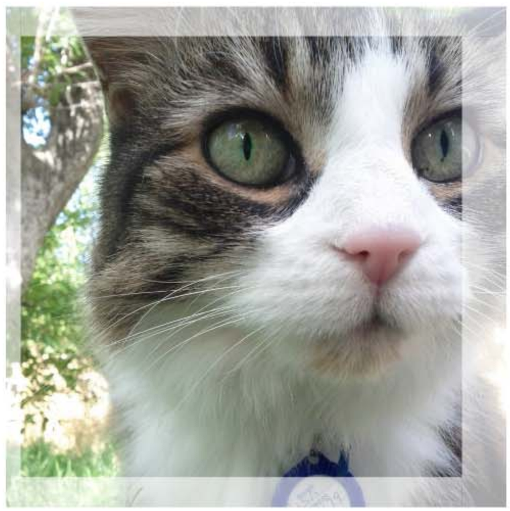


Very bad (slow to train)

Very good (quick to train)

Data augmentation

- Our neural net has 60M real-valued parameters and 650,000 neurons
- It overfits a lot. Therefore we train on 224x224 patches extracted randomly from 256x256 images, and also their horizontal reflections.



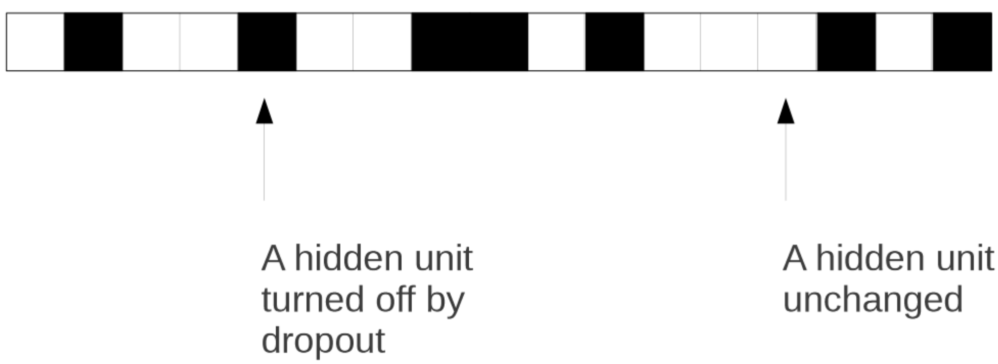
Testing

- Average predictions made at five 224x224 patches and their horizontal reflections (four corner patches and center patch)
- Logistic regression has the nice property that it outputs a probability distribution over the class labels
- Therefore no score normalization or calibration is necessary to combine the predictions of different models (or the same model on different patches), as would be necessary with an SVM.

Dropout

- Independently set each hidden unit activity to zero with 0.5 probability
- We do this in the two globally-connected hidden layers at the net's output

A hidden layer's activity on a given training image



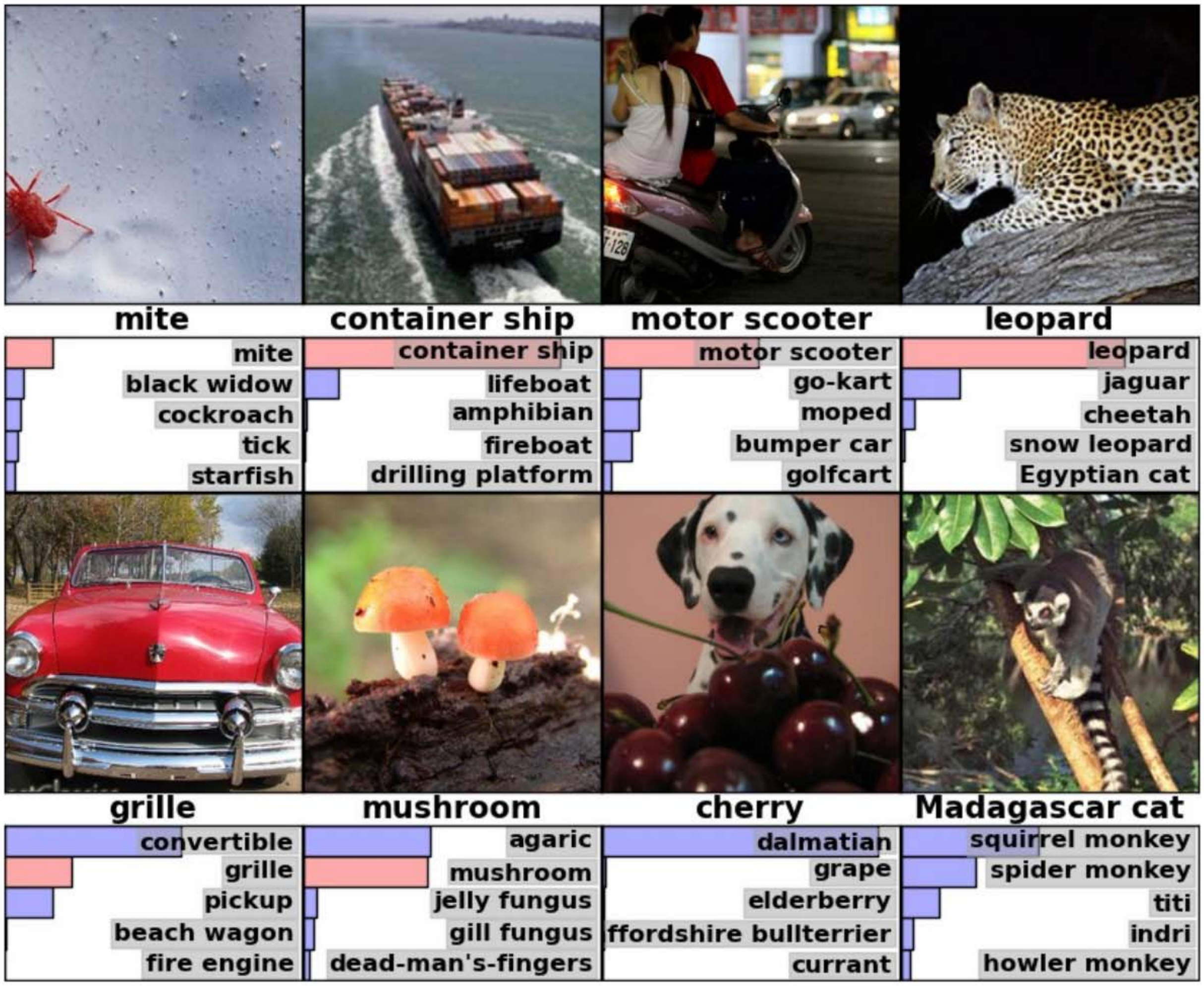
Implementation

- The only thing that needs to be stored on disk is the raw image data
- We stored it in JPEG format. It can be loaded and decoded entirely in parallel with training.
- Therefore only 27GB of disk storage is needed to train this system.
- Uses about 2GB of RAM on each GPU, and around 5GB of system memory during training.

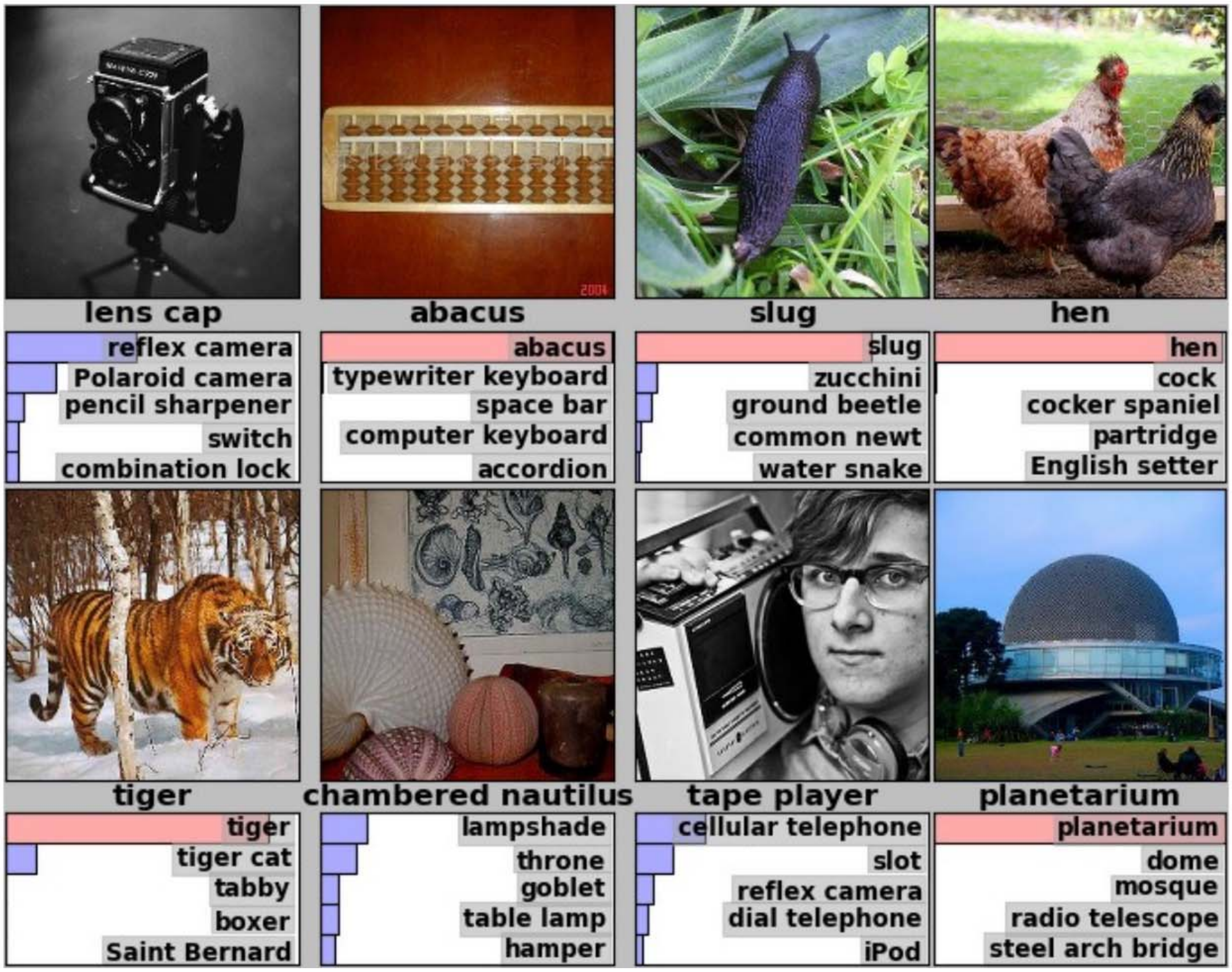
Implementation

- Written in Python/C++/CUDA
- Sort of like an instruction pipeline, with the following 4 instructions happening in parallel:
  - Train on batch *n* (on GPUs)
  - Copy batch *n*+1 to GPU memory
  - Transform batch *n*+2 (on CPU)
  - Load batch *n*+3 from disk (on CPU)

Validation classification

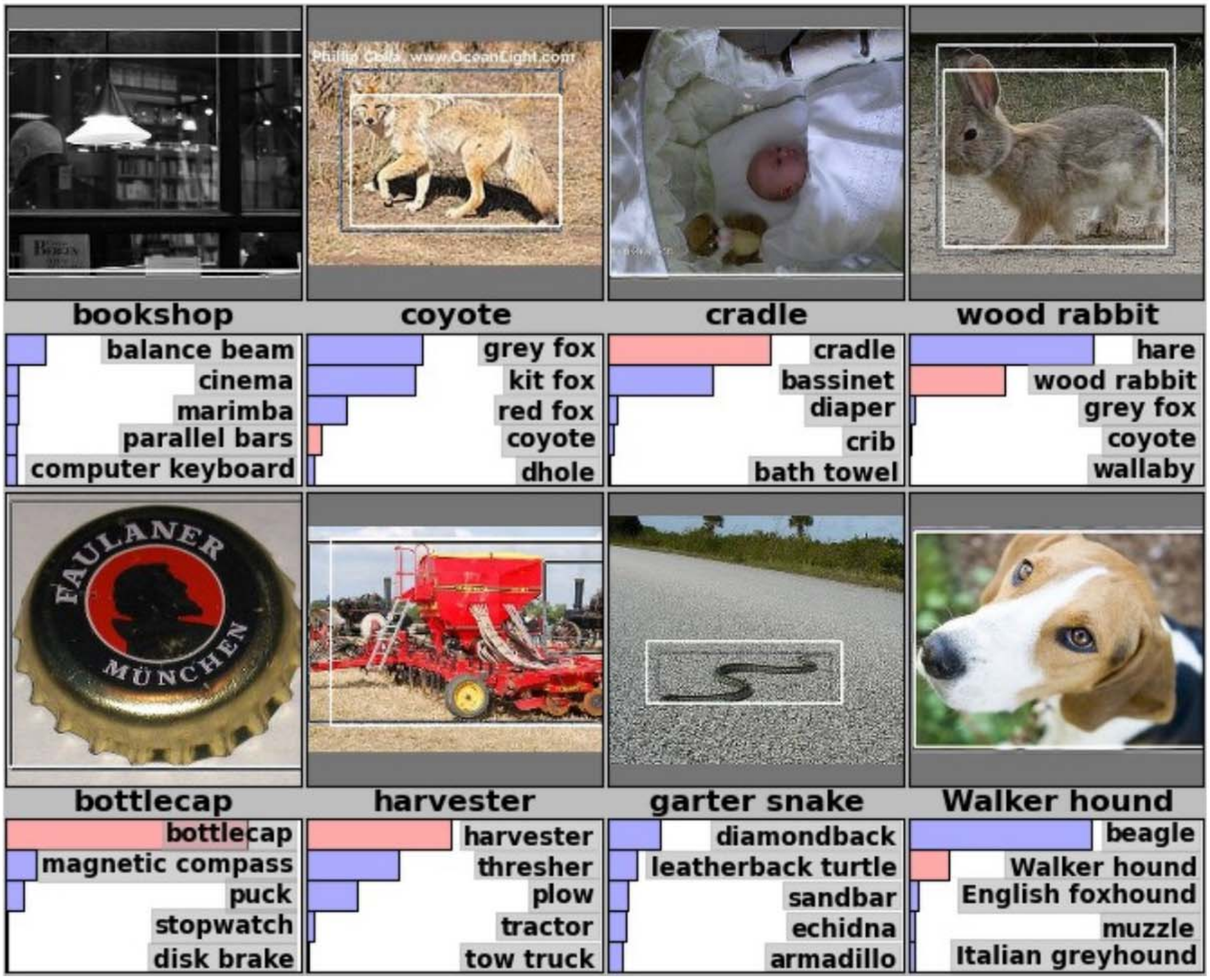


Validation classification

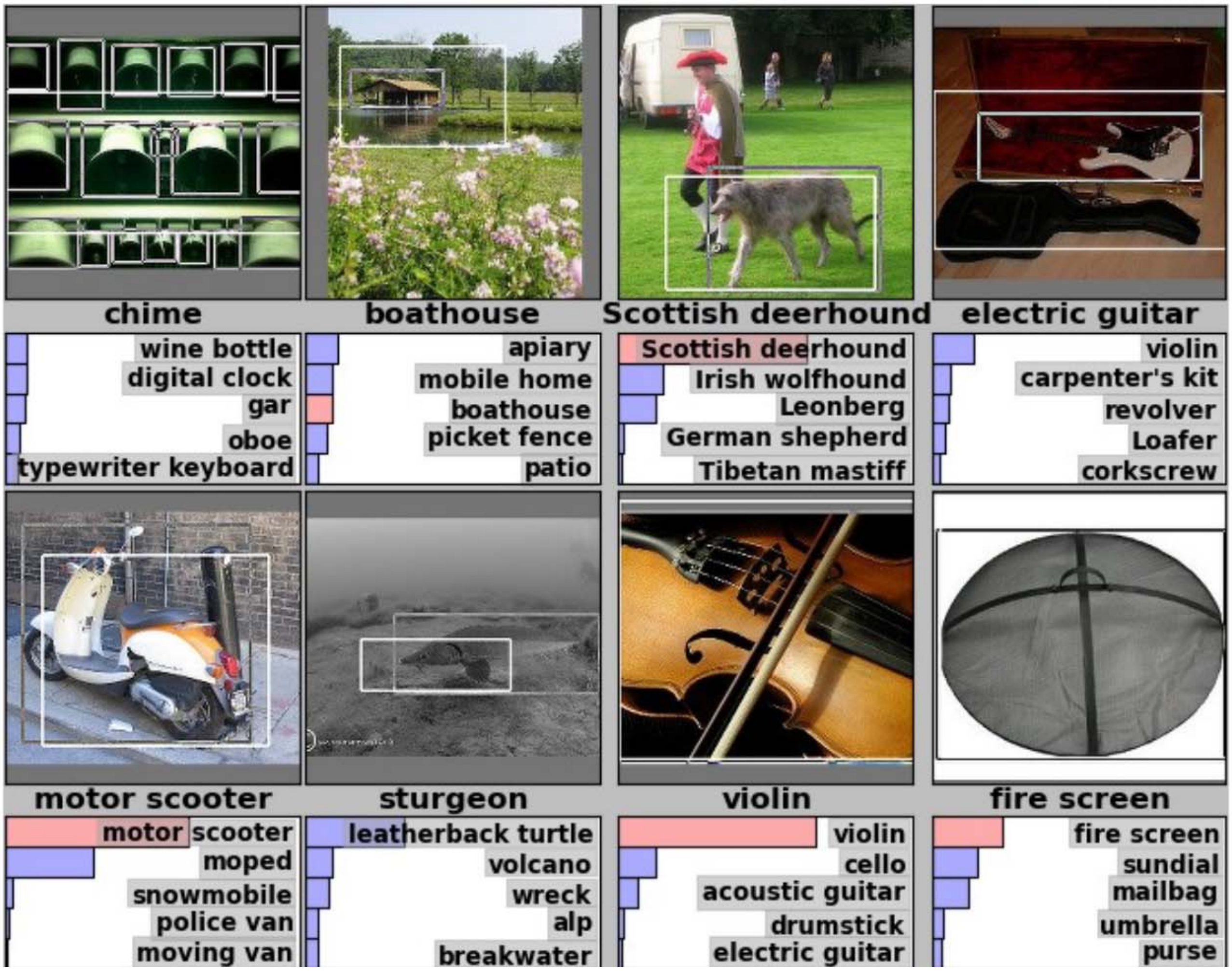


Validation classification

Validation localizations

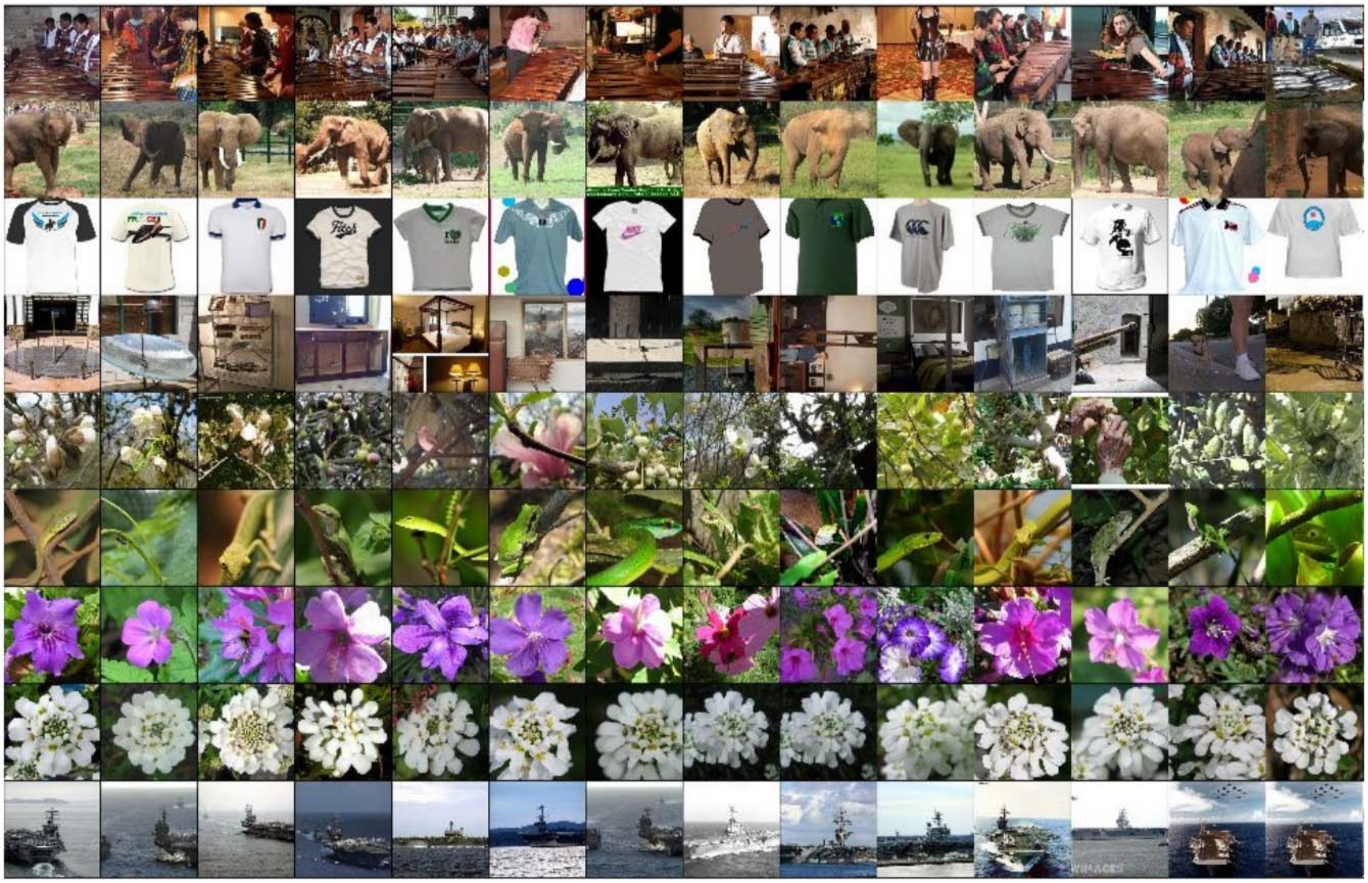


Validation localizations



Retrieval experiments

First column contains query images from ILSVRC-2010 test set, remaining columns contain retrieved images from training set.



Retrieval experiments