ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
Ilya Sutskever
Geoffrey Hinton

University of Toronto

Presented at UAIG
Neural Networks

- A neuron

![Diagram of a neuron]

\[ x = w_1 y_1 + w_2 y_2 + w_3 y \]

- One neuron can implement logical gates (and a lot more)
Neural Networks

- Neural Networks are circuits
- They can compute lots of complicated functions
- The connections determine the function
- Connections are slowly adjusted by a learning algorithm to reduce error on training cases
Training Neural Networks

- Slowly change the weights to improve performance

Do until convergence
- Pick a training case
- Compare prediction to target
- Update parameters to slightly reduce error

- This process will converge to weights that should make sensible predictions on all training cases
- These weights implement a circuit whose operation reflects deep facts about the data
- Training method is simple, resulting neural network is extremely complex
Generalization

- How does the network “know” the correct answer on previously unseen examples?

- The network's ability to memorize random patterns is limited
  - With enough training data, train error=test error

- If we are lucky, the network is capable of representing a good function, so training will find it
  - Otherwise our error will be large
Generalization

Training cases are like constraints
Learning is like solving an equation

Constraint imposed by one training case

Neural network space
Convolutional neural networks

- Apply neural networks to images
  - Images are very large, so networks are huge
- One convolution: apply the same weight to every image-patch

All nodes compute the same function of the nodes below them

Advantages of conv:
- less connections
- much less parameters
Convolutional neural networks

- Many “maps” go to many “maps”
- GPU-friendly
- Key operation

Each edge is a convolution
All edges are different
In this figure!
Overview of our model

- **Deep**: 7 hidden weight layers
- **Learned**: all feature extractors initialized with 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
Training

Using stochastic gradient descent and the \textit{backpropagation algorithm} (just repeated application of the chain rule)

Make millions of small changes to the network's weights
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
Input representation

- Centered (0-mean) RGB values.
Neurons

\[ f(x) = \tanh(x) \]

\[ f(x) = \max(0, x) \]

Very bad (slow to train)

Very good (quick to train)

\[ x = w_1 f(z_1) + w_2 f(z_2) + w_3 f(z_3) \]

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

- A hidden unit turned off by dropout
- A hidden unit unchanged
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)
Comparison to monkey brain

- Some researchers showed images to macaques and recorded the firing rates of 128 of their neurons

- Compare to other systems in recognizing “hard images”
  - Lots of rotations, change in illumination

- Our neural network's 4096 neurons beat the 128 macaque's neurons
  - Although more of the macaque's neurons may outperform our system

- All other computer vision methods did much worse than the macaque neurons
Place a big electrode in the right part of the visual cortex and record from 128 neurons.

Get 128 dims

other methods

Get lots of dims

Get 4096 dims

other methods

Get lots of dims

other methods

Get lots of dims
Validation classification

<table>
<thead>
<tr>
<th>mite</th>
<th>container ship</th>
<th>motor scooter</th>
<th>leopard</th>
</tr>
</thead>
<tbody>
<tr>
<td>mite</td>
<td>container ship</td>
<td>motor scooter</td>
<td>leopard</td>
</tr>
<tr>
<td>black widow</td>
<td>lifeboat</td>
<td>go-kart</td>
<td>jaguar</td>
</tr>
<tr>
<td>cockroach</td>
<td>amphibian</td>
<td>moped</td>
<td>cheetah</td>
</tr>
<tr>
<td>tick</td>
<td>fireboat</td>
<td>bumper car</td>
<td>snow leopard</td>
</tr>
<tr>
<td>starfish</td>
<td>drilling platform</td>
<td>golfcart</td>
<td>Egyptian cat</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>grille</th>
<th>mushroom</th>
<th>cherry</th>
<th>Madagascar cat</th>
</tr>
</thead>
<tbody>
<tr>
<td>convertible</td>
<td>agaric</td>
<td>dalmatian</td>
<td>squirrel monkey</td>
</tr>
<tr>
<td>grille</td>
<td>mushroom</td>
<td>grape</td>
<td>spider monkey</td>
</tr>
<tr>
<td>pickup</td>
<td>jelly fungus</td>
<td>elderberry</td>
<td>titi</td>
</tr>
<tr>
<td>beach wagon</td>
<td>gill fungus</td>
<td>currant</td>
<td>indri</td>
</tr>
<tr>
<td>fire engine</td>
<td>dead-man's-fingers</td>
<td>currant</td>
<td>howler monkey</td>
</tr>
</tbody>
</table>
Validation classification
Validation classification

- Koala
  - Wombat
  - Norwegian elkhound
  - Wild boar
  - Wallaby
  - Koala

- Tiger
  - Tiger
  - Tiger cat
  - Jaguar
  - Lynx
  - Leopard

- European fire salamander
  - Spotted salamander
  - Common newt
  - Long-horned beetle
  - Box turtle

- Loggerhead
  - African crocodile
  - Gila monster
  - Loggerhead
  - Mud turtle
  - Leatherback turtle

- Seat belt
  - Seat belt
  - Ice lolly
  - Hotdog
  - Burrito
  - Band Aid

- Television
  - Television
  - Microwave
  - Monitor
  - Screen
  - Car mirror

- Sliding door
  - Sliding door
  - Shoji
  - Window shade
  - Window screen
  - Four-poster

- Wallaby
  - Hare
  - Wallaby
  - Wood rabbit
  - Lakeland terrier
  - Kit fox
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