Typical CNN architectures and transfer learning

Visualizing representations

Segmentation

Object detection

Recall: the goal of this model is to map an input image to a class prediction.
A breakthrough in image understanding

The AlexNet model

Each large block represents a data tensor
Each smaller block represents a filter
The filter size and stride are shown

The number of filters can be deduced from the number of feature channels
There are two parallel streams in this network (for efficiency reasons)

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, 4

This is the reason why the input images in Figure 2 are $224 \times 224 \times 3$-dimensional.

How deep is deep enough?


Accuracy

3 ⨉ more accurate in 3 years

Speed

5 ⨉ slower

Remark: 101 ResNet layers same size/speed as 16 VGG-VD layers
Reason: far fewer feature channels (quadratic speed/space gain)
Moral: optimize your architecture
**Model size**

Num. of parameters is about the same

**Remark:** 101 ResNet layers same size/speed as 16 VGG-VD layers

**Reason:** far fewer feature channels (quadratic speed/space gain)

**Moral:** optimize your architecture

---

**Pre-training and transfer learning**

[evaluations in A. S. Razavian, 2014, Chatfield et al., 2014]

- **Pertained layers**
- **Fine-tuned layers**

**CNN as universal representations**
- First several layers in most CNNs are generic
- They can be reused when training data is comparatively scarce

**Application**
- Pre-train on ImageNet classification
- 1M images
- Cut at some deep conv or FC layer to get features

---

**Transfer learning**

Deep representations are generic

- Deep feature encoder
- Trained on a reference dataset (eg ImageNet)
- Trained on target dataset (eg PASCAL)

A general purpose deep encoder is obtained by chopping off the last layers of a CNN trained on a large dataset.

---

**CNNs as filter banks**

Deep representations used as local features

- Deep filter bank
- Trained on a reference dataset (eg ImageNet)
- Trained on target dataset (eg PASCAL)

In R-CNN and similar models, the most important shared component are the convolutional features.
Demo

Typical CNN architectures and transfer learning

Visualizing representations

Segmentation

Object detection

Visualization: Pre-Image

The representation is not injective

Datagram: Pre-Image

The reconstruction ambiguity provides useful information about the representation

Image Space

Representation Space

The representation is not injective

Image Space

Representation Space

The reconstruction ambiguity provides useful information about the representation
Finding a Pre-Image

A simple yet general and effective method

\[ \min_x \| \Phi(x) - \Phi_0 \|_2^2 \]

Start from random noise
Optimize using stochastic gradient descent

Related Work

Analysis tools
Visualizing higher-layer features of a deep network
Ethan et al. 2009
[intermediate features]

Deep inside convolutional networks
Simonyan et al. 2014
[deepest features, aka “deep dreams”]

DeConvNets
Zeiler et al. In ECCV, 2014
[intermediate features]

Understanding neural networks through deep visualisation
Yosinksi et al. 2015
[intermediate features]

Artistic tools
Google’s “inceptionsm”
Mordvintsev et al. 2015

Style synthesis and transfer
Gatys et al. 2015

Inverting a Deep CNN

conv 1
ReLU 1
Max pool 1
LRN 1
AlexNet
[Krizhevsky et al. 2012]
Inverting a Deep CNN

Original Image

conv 1  conv 2  conv 3  conv 4  conv 5  fc 6  fc 7  fc 8

Original Image

conv 1  conv 2  conv 3  conv 4  conv 5  fc 6  fc 7  fc 8

Original Image

conv 1  conv 2  conv 3  conv 4  conv 5  fc 6  fc 7  fc 8

Original Image
Inverting a Deep CNN

Original Image

conv 1  conv 2  conv 3  conv 4  conv 5  fc 6  fc 7  fc 8

Original Image

conv 1  conv 2  conv 3  conv 4  conv 5  fc 6  fc 7  fc 8

Original Image

conv 1  conv 2  conv 3  conv 4  conv 5  fc 6  fc 7  fc 8

Original Image
Inverting a Deep CNN

Original Image

conv 1 conv 2 conv 3 conv 4 conv 5 fc 6 fc 7 fc 8

Original Image

conv 1 conv 2 conv 3 conv 4 conv 5 fc 6 fc 7 fc 8

Original Image

conv 1 conv 2 conv 3 conv 4 conv 5 fc 6 fc 7 fc 8

Original Image
The first form of data augmentation consists of generating image translations and horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax.

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC forced us to use much smaller networks. At test time, the network makes a prediction by extracting the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$.

The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the fifth convolutional layer has 256 kernels of size $5 \times 5$. The fourth, third, and second convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the fifth convolutional layer has 256 kernels of size $5 \times 5$. The fourth, third, and second convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the fifth convolutional layer has 256 kernels of size $5 \times 5$. The fourth, third, and second convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the fifth convolutional layer has 256 kernels of size $5 \times 5$. The fourth, third, and second convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the fifth convolutional layer has 256 kernels of size $5 \times 5$. The fourth, third, and second convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the fifth convolutional layer has 256 kernels of size $5 \times 5$. The fourth, third, and second convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the fifth convolutional layer has 256 kernels of size $5 \times 5$. The fourth, third, and second convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3$. The second convolutional layer takes as input the (response-normalized, pooled) output of the first convolutional layer and filters it with 256 kernels of size $3 \times 3$. The second convolutional layer has 256 neurons in a kernel map. The second convolutional layer has 256 neurons in a kernel map.
The ImageNet training set. To each training image, we add multiples of the found principal components, training images. Specifically, we perform PCA on the set of RGB pixel values throughout the training set by a factor of 2048, though the resulting training examples are, of course, highly inter-dependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting random 256 patches (and their horizontal reflections) from the patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the training set by data augmentation. We perform two forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk.

The first form of data augmentation consists of generating image translations and horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in images. Specifically, we perform PCA on the set of RGB pixel values throughout the training set. This is the reason why the input images in Figure 2 are 224 \times 224 \times 3-dimensional.
Network comparison

“conv5” features

AlexNet
VGG-M
VGG-VD

Caricaturization

[Google Inceptionism 2015, Mahendran et al. 2015]

Emphasise patterns that are detected by a certain representation

\[
\min_x -\langle \Phi(x_0), \Phi(x) \rangle - R_{TV}(x) + R_u(x)
\]

Key differences:
- the starting point is the image \( x_0 \)
- particular configurations of features are emphasized, not individual features

Remember: the starting point is not an image

The starting point is not an image
Surprisingly, the filters learned by discriminative neural networks capture well the “style” of an image. This can be used to transfer the style of an image (e.g. a painting) to any other.

**Optimisation based**


**Feed-forward neural network equivalents**


Typical CNN architectures and transfer learning

Visualizing representations

Segmentation

Object detection
**Semantic image segmentation**

Label individual pixels

- **Convolutional vs fully connected layers**
  - Convolutional layers: Neurons are spatially selective, can be used to localize things.
  - Fully connected layers: Neurons are global, do not characterize well position.

Which one is more useful for pixel level labelling?

**Receptive field**

The part of the image looked at by a neuron

- **Receptive Field (RF) of a neuron**
  - The subset of the image affecting the value of a neuron

- **Small vs large RFs**
  - Small RF: spatially specific, but can only account for small visual structures
  - Large RF: spatially a-specific, but can account for large visual structure

- **How to make the RF large**
  - Use large filters
  - Chain several filters
  - Interleave downsampling along the chain
    - E.g. downsampling 2x increases the RF size 2x.

**Comparing the receptive fields**

Convolutional vs fully connected layers

- Convolutional layers: Neurons are spatially selective, can be used to localize things.
- Fully connected layers: Neurons are global, do not characterize well position.

**A fully connected layer is just a large filter**

The filter support fills the entire input tensor

- **Filter support fills the entire input tensor**
  - $F(k) \times 1 \times 1 \times K$

- **A fully connected layer is just a large filter**
  - $1 \times 1 \times K$
Fully-convolutional neural networks

Dense evaluation
- Apply the whole network
- Convolutional
- Computes a vector of class probabilities at each pixel

Downsampling
- For efficiency, the input data is substantially down sampled in the network
- The output is fairly low resolution (e.g. 1/32 of original)

The object detection problem

The goal of object detection is to simultaneously classify, enumerate, and localise known object types in an image.

A key challenge is that the number of object instances is not known a priori.

Typical CNN architectures and transfer learning

Visualizing representations

Segmentation

Object detection
Next term!