MNIST Handwritten Digit Dataset

- black and white, resolution $28 \times 28$
- 60,000 images
- 10 classes (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

CIFAR Image Dataset

- color, resolution $32 \times 32$
- 50,000 images
- 10 classes
ImageNet LSVRC Dataset

- color, resolution $227 \times 227$
- 1.2 million images
- 1000 classes

Image Processing Tasks

- image classification
- object detection
- object segmentation
- style transfer
- generating images
- generating art
- image captioning

Object Detection

LeNet trained on MNIST

The $5 \times 5$ window of the first convolution layer extracts from the original $32 \times 32$ image a $28 \times 28$ array of features. Subsampling then halves this size to $14 \times 14$. The second Convolution layer uses another $5 \times 5$ window to extract a $10 \times 10$ array of features, which the second subsampling layer reduces to $5 \times 5$. These activations then pass through two fully connected layers into the 10 output units corresponding to the digits '0' to '9'.

Convolutional Neural Networks

Assume the original image is $J \times K$, with $L$ channels. We apply an $M \times N$ “filter” to these inputs to compute one hidden unit in the convolution layer. In this example $J = 6, K = 7, L = 3, M = 3, N = 3$.

$Z_{j,k}^l = g(b^l + \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K_{l,m,n}^l V_{j+m,k+n}^l)$

The same weights are applied to the next $M \times N$ block of inputs, to compute the next hidden unit in the convolution layer ("weight sharing").

ImageNet Architectures

- AlexNet, 8 layers (2012)
- VGG, 19 layers (2014)
- GoogleNet, 22 layers (2014)
- ResNets, 152 layers (2015)

AlexNet Architecture

- 5 convolutional layers + 3 fully connected layers
- max pooling with overlapping stride
- softmax with 1000 classes
- 2 parallel GPUs which interact only at certain layers

AlexNet Architecture

- 5 convolutional layers + 3 fully connected layers
- max pooling with overlapping stride
- softmax with 1000 classes
- 2 parallel GPUs which interact only at certain layers
Convolutional Neural Networks

If the original image size is $J \times K$ and the filter is size $M \times N$, the convolution layer will be $(J + 1 - M) \times (K + 1 - N)$

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks

Convolutional Neural Networks
**Stride with Zero Padding**

When combined with zero padding of width $P$, $j$ takes on the values 0, $s$, $2s$, ..., $(J + 2P - M)$
$k$ takes on the values 0, $s$, $2s$, ..., $(K + 2P - N)$
The next layer is $(1 + (J + 2P - M)/s)$ by $(1 + (K + 2P - N)/s)$

---

**Example: AlexNet Conv Layer 1**

For example, in the first convolutional layer of AlexNet,
$J = K = 224$, $P = 2$, $M = N = 11$, $s = 4$.
The width of the next layer is

$1 + (J - M)/s = 1 + (224 + 2 - 11)/4 = 55$

Question: If there are 96 filters in this layer, compute the number of:
weights per neuron?
neurons?
connections?
independent parameters?

---

**Max Pooling**

The width of the next layer is

$1 + (J - M)/s = 1 + (224 + 2 - 11)/4 = 55$

Question: If there are 96 filters in this layer, compute the number of:
weights per neuron?
neurons?
connections?
independent parameters?
Overlapping Pooling

If the previous layer is $J \times K$, and max pooling is applied with width $F$ and stride $s$, the size of the next layer will be

$$(1 + (J - F)/s) \times (1 + (K - F)/s)$$

Question: If max pooling with width 3 and stride 2 is applied to the features of size 55 $\times$ 55 in the first convolutional layer of AlexNet, what is the size of the next layer?

Answer: $1 + (55 - 3)/2 = 27$.

Question: How many independent parameters does this add to the model?

Answer: None! (no weights to be learned, just computing max)

Enhancements

- Rectified Linear Units (ReLUs)
- overlapping pooling (width = 3, stride = 2)
- stochastic gradient descent with momentum and weight decay
- data augmentation to reduce overfitting
- 50% dropout in the fully connected layers

AlexNet Details

- 650K neurons
- 630M connections
- 60M parameters
- more parameters that images $\rightarrow$ danger of overfitting

Data Augmentation

- ten patches of size $224 \times 224$ are cropped from each of the original $227 \times 227$ images (using zero padding)
- the horizontal reflection of each patch is also included.
- at test time, average the predictions on the 10 patches.
- also include changes in intensity to RGB channels
Convolution Kernels

- filters on GPU-1 (upper) are color agnostic
- filters on GPU-2 (lower) are color specific
- these resemble Gabor filters

Statistics

The mean and variance of a set of $n$ samples $x_1, \ldots, x_n$ are given by

\[
\text{Mean}[x] = \frac{1}{n} \sum_{k=1}^{n} x_k
\]

\[
\text{Var}[x] = \frac{1}{n} \sum_{k=1}^{n} (x_k - \text{Mean}[x])^2 = \left( \frac{1}{n} \sum_{k=1}^{n} x_k^2 \right) - \text{Mean}[x]^2
\]

If $w_k, x_k$ are independent and $y = \sum_{k=1}^{n} w_k x_k$ then

\[
\text{Var}[y] = n \text{Var}[w] \text{Var}[x]
\]

Dealing with Deep Networks

- > 10 layers
  - weight initialization
  - batch normalization
- > 30 layers
  - skip connections
- > 100 layers
  - identity skip connections

Weight Initialization

Consider one layer $(i)$ of a deep neural network with weights $w^{(i)}_{jk}$ connecting the activations $\{x_k^{(i)}\}_{1 \leq k \leq n_i}$ at the previous layer to $\{x_j^{(i+1)}\}_{1 \leq j \leq n_{i+1}}$ at the next layer, where $g()$ is the transfer function and

\[
x_j^{(i+1)} = g(\text{sum}^{(i)}_{j}) = g\left( \sum_{k=1}^{n_i} w^{(i)}_{jk} x_k^{(i)} \right)
\]

Then

\[
\text{Var}[\text{sum}^{(i)}_{j}] = n_i \text{Var}[w^{(i)}] \text{Var}[x^{(i)}]
\]

\[
\text{Var}[x^{(i+1)}] \simeq G_0 n_i \text{Var}[w^{(i)}] \text{Var}[x^{(i)}]
\]

Where $G_0$ is a constant whose value is estimated to take account of the transfer function.
If some layers are not fully connected, we replace $n_i$ with the average number $n_i^{\text{mi}}$ of incoming connections to each node at layer $i + 1$. 
Weight Initialization

If the network has $D$ layers, with input $x = x^{(1)}$ and output $z = x^{(D+1)}$, then

$$\text{Var}[z] \simeq \left( \prod_{i=1}^{D} G_0 n_i^{\text{in}} \text{Var}[w^{(i)}] \right) \text{Var}[x]$$

When we apply gradient descent through backpropagation, the differentials will follow a similar pattern:

$$\text{Var} \left[ \frac{\partial}{\partial x} \right] \simeq \left( \prod_{i=1}^{D} G_1 n_i^{\text{out}} \text{Var}[w^{(i)}] \right) \text{Var} \left[ \frac{\partial}{\partial z} \right]$$

In this equation, $n_i^{\text{out}}$ is the average number of outgoing connections for each node at layer $i$, and $G_1$ is meant to estimate the average value of the derivative of the transfer function.

For Rectified Linear Units, we can assume $G_0 = G_1 = \frac{1}{2}$

Weight Initialization

In order to have healthy forward and backward propagation, each term in the product must be approximately equal to 1. Any deviation from this could cause the activations to either vanish or saturate, and the differentials to either decay or explode exponentially.

$$\text{Var}[z] \simeq \left( \prod_{i=1}^{D} G_0 n_i^{\text{in}} \text{Var}[w^{(i)}] \right) \text{Var}[x]$$

$$\text{Var} \left[ \frac{\partial}{\partial x} \right] \simeq \left( \prod_{i=1}^{D} G_1 n_i^{\text{out}} \text{Var}[w^{(i)}] \right) \text{Var} \left[ \frac{\partial}{\partial z} \right]$$

We therefore choose the initial weights $\{w^{(i)}_{jk}\}$ in each layer $(i)$ such that $G_1 n_i^{\text{out}} \text{Var}[w^{(i)}] = 1$

Batch Normalization

We can normalize the activations $x_k^{(i)}$ of node $k$ in layer $(i)$ relative to the mean and variance of those activations, calculated over a mini-batch of training items:

$$\hat{x}_k^{(i)} = \frac{x_k^{(i)} - \text{Mean}[x_k^{(i)}]}{\sqrt{\text{Var}[x_k^{(i)}]}}$$

These activations can then be shifted and re-scaled to

$$y_k^{(i)} = \beta_k^{(i)} + \gamma_k^{(i)} x_k^{(i)}$$

$\beta_k^{(i)}, \gamma_k^{(i)}$ are additional parameters, for each node, which are trained by backpropagation along with the other parameters (weights) in the network.

After training is complete, $\text{Mean}[x_k^{(i)}]$ and $\text{Var}[x_k^{(i)}]$ are either pre-computed on the entire training set, or updated using running averages.
Residual Networks

- the preceding layers attempt to do the “whole” job, making $x$ as close as possible to the target output of the entire network
- $F(x)$ is a residual component which corrects the errors from previous layers, or provides additional details which the previous layers were not powerful enough to compute
- with skip connections, both training and test error drop as you add more layers
- with more than 100 layers, need to apply relu before adding the residual instead of afterwards. This is called an identity skip connection.

Residual Networks

$F(x)$

$H(x) = F(x) + x$

Identity

Dense Networks

Recently, good results have been achieved using networks with densely connected blocks, within which each layer is connected by shortcut connections to all the preceding layers.

Going Deeper

If we simply stack additional layers, it can lead to higher training error as well as higher test error.
Neural Texture Synthesis

We can introduce a scaling factor $w_l$ for each layer $l$ in the network, and define the Cost function as

$$E_{\text{style}} = \frac{1}{4} \sum_{l=0}^{L} w_l N_l M_l^2 \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

where $N_l, M_l$ are the number of filters, and size of feature maps, in layer $l$, and $G_{ij}^l, A_{ij}^l$ are the Gram matrices for the original and synthetic image.
Neural Style Transfer

For Neural Style Transfer, we minimize a cost function which is

\[ E_{\text{total}} = \alpha E_{\text{content}} + \beta E_{\text{style}} \]

\[ = \frac{\alpha}{2} \sum_{i,k} ||F_{il}^c(x) - F_{il}^s(x_c)||^2 + \frac{\beta}{4} \sum_{l=0}^{L} w_l \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2 \]

where

- \( x_c, x \) = content image, synthetic image
- \( F_{il}^c \) = \( i \)th filter at position \( k \) in layer \( l \)
- \( N_l, M_l \) = number of filters, and size of feature maps, in layer \( l \)
- \( w_l \) = weighting factor for layer \( l \)
- \( G_{ij}^l, A_{ij}^l \) = Gram matrices for style image, and synthetic image

References

- “ImageNet Classification with Deep Convolutional Neural Networks”, Krizhevsky et al., 2015.
- “Densely Connected Convolutional Networks”, Huang et al., 2016.
- “A Neural Algorithm of Artistic Style”, Gatys et al., 2015.