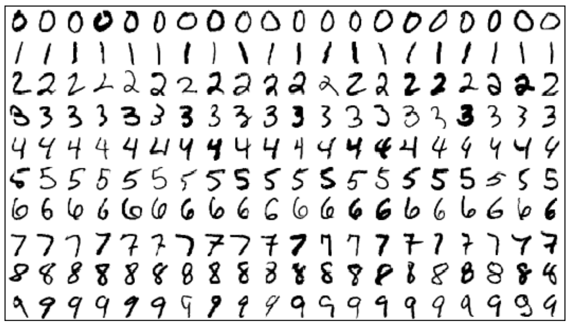
COMP9444
Neural Networks and Deep Learning
7. Image Processing

Outline

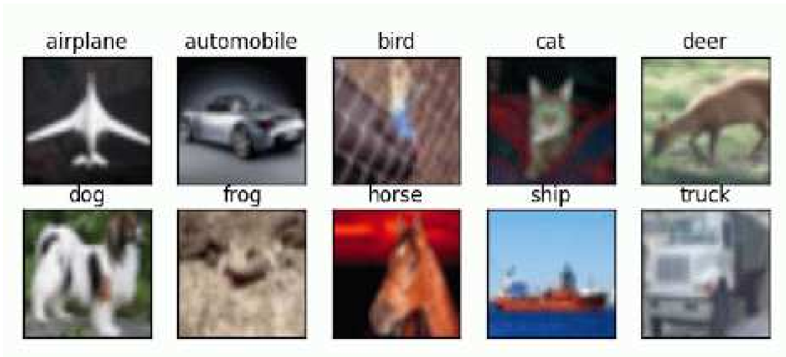
- Image Datasets and Tasks
- Convolution in Detail
- AlexNet
- Weight Initialization
- Batch Normalization
- Residual Networks
- Dense Networks
- Style Transfer

MNIST Handwritten Digit Dataset



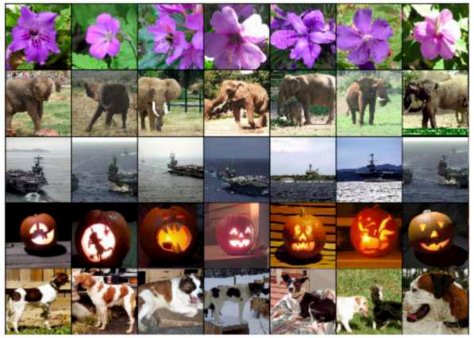
- black and white, resolution 28×28
- 60,000 images
- 10 classes $(0,1,2,3,4,5,6,7,8,9)$

CIFAR Image Dataset



- color, resolution 32×32
- 50,000 images
- 10 classes

ImageNet LSVRC Dataset

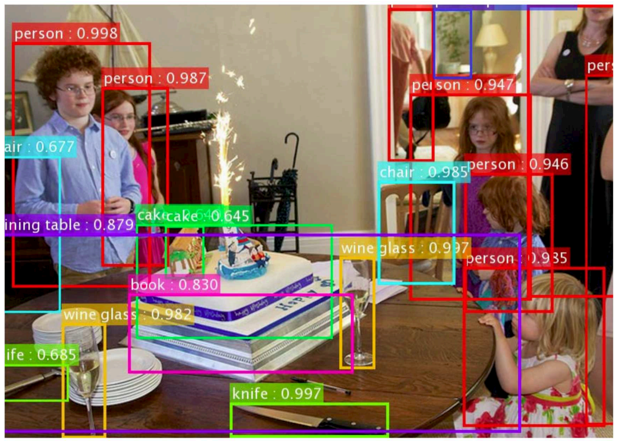


- color, resolution 227×227
- 1.2 million images
- 1000 classes

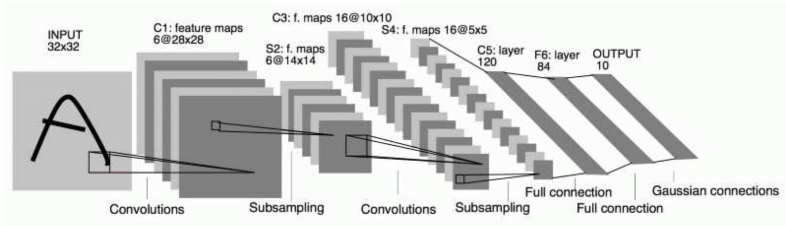
Image Processing Tasks

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

Object Detection



LeNet trained on MNIST

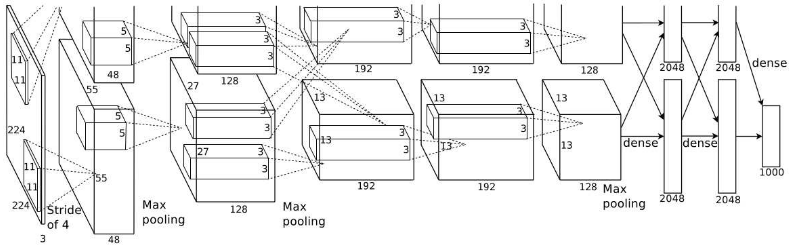


The 5×5 window of the first convolution layer extracts from the original 32×32 image a 28×28 array of features. Subsampling then halves this size to 14×14. The second Convolution layer uses another 5×5 window to extract a 10×10 array of features, which the second subsampling layer reduces to 5×5. These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

ImageNet Architectures

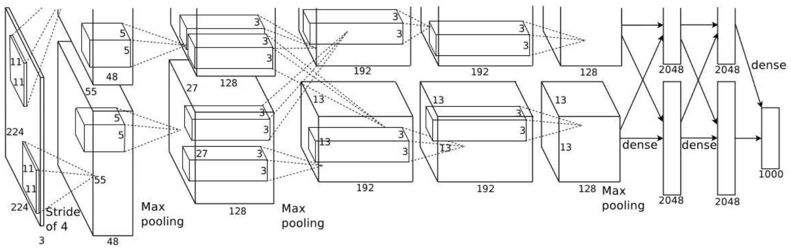
- AlexNet, 8 layers (2012)
- VGG, 19 layers (2014)
- GoogleNet, 22 layers (2014)
- ResNets, 152 layers (2015)

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 Details



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

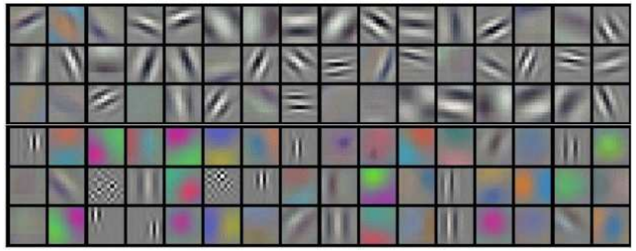
Enhancements

- 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

Data Augmentation

- ten patches of size 224×224 are cropped from each of the original 227×277 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

Dealing with Deep Networks

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

Statistics Example: Coin Tossing

Example: Toss a coin once, and count the number of Heads

$$\begin{array}{lll} \text{Mean} & \mu & = \frac{1}{2}(0+1) \;=\; 0.5 \\ \text{Variance} & & = \frac{1}{2}\big((0-0.5)^2+(1-0.5)^2\big) = 0.25 \\ \text{Standard Deviation} & \sigma & = \sqrt{\text{Variance}} = 0.5 \end{array}$$

Example: Toss a coin 100 times, and count the number of Heads

$$\begin{array}{lll} \text{Mean} & \mu & = 100 * 0.5 \;=\; 50 \\ \text{Variance} & & = 100 * 0.25 = 25 \\ \text{Standard Deviation} & \sigma & = \sqrt{\text{Variance}} = 5 \end{array}$$

Example: Toss a coin 10000 times, and count the number of Heads

$$\mu = 5000, \qquad \sigma \;=\; \sqrt{2500} \;=\; 50$$

Statistics

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

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

$$\mathrm{Var}[x] = \frac{1}{n}\sum_{k=1}^{n}(x_k - \mathrm{Mean}[x])^2 = \Big(\frac{1}{n}\sum_{k=1}^{n} x_k^2\Big) - \mathrm{Mean}[x]^2$$

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

$$\mathrm{Var}[y] = n\,\mathrm{Var}[w]\,\mathrm{Var}[x]$$

Weight Initialization

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

$$x_j^{(i+1)} = g(\mathrm{sum}_j^{(i)}) = g\Big(\sum_{k=1}^{n_i} w_{jk}^{(i)} x_k^{(i)}\Big)$$

Then

$$\mathrm{Var}[\mathrm{sum}^{(i)}] = n_i \mathrm{Var}[w^{(i)}]\mathrm{Var}[x^{(i)}]$$

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

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^{in} of incoming connections to each node at layer $i+1$.

Weight Initialization

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

$$\mathrm{Var}[z] \simeq \Big(\prod_{i=1}^{D} G_0\, n_i^{\mathrm{in}}\, \mathrm{Var}[w^{(i)}]\Big)\mathrm{Var}[x]$$

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

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

In this equation, n_i^{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.

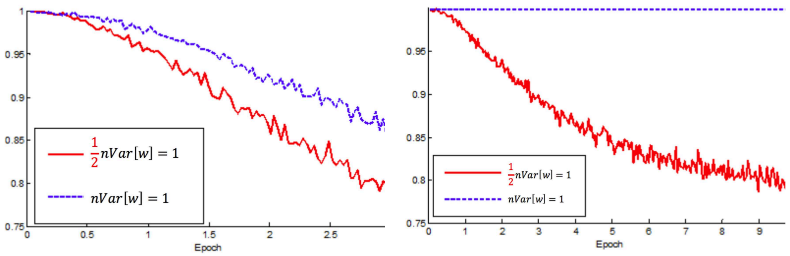
$$\mathrm{Var}[z] \simeq \Big(\prod_{i=1}^{D} G_0\, n_i^{\mathrm{in}} \mathrm{Var}[w^{(i)}]\Big)\mathrm{Var}[x]$$

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

We therefore choose the initial weights $\{w_{jk}^{(i)}\}$ in each layer (i) such that

$$G_1 n_i^{\mathrm{out}} \mathrm{Var}[w^{(i)}] = 1$$

Weight Initialization



- 22-layer ReLU network (left), $G_1 = \frac{1}{2}$ converges faster than $G_1 = 1$
- 30-layer ReLU network (right), $G_1 = \frac{1}{2}$ is successful while $G_1 = 1$ fails to learn at all

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:

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

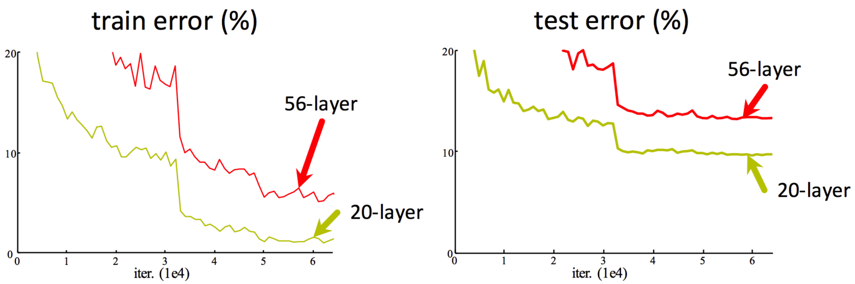
These activations can then be shifted and re-scaled to

$$y_k^{(i)} = \beta_k^{(i)} + \gamma_k^{(i)} \hat{x}_k^{(i)}$$

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

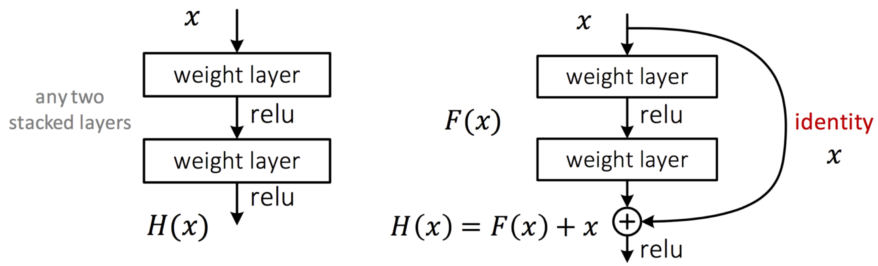
After training is complete, $\mathrm{Mean}[x_k^{(i)}]$ and $\mathrm{Var}[x_k^{(i)}]$ are either pre-computed on the entire training set, or updated using running averages.

Going Deeper



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

Residual Networks

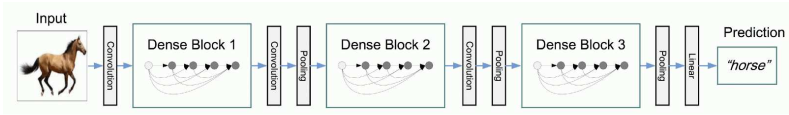


Idea: Take any two consecutive stacked layers in a deep network and add a "skip" connection which bipasses these layers and is added to their output.

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.

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.

Texture Synthesis

Neural Texture Synthesis

1. pretrain CNN on ImageNet (VGG-19)
2. pass input texture through CNN; compute feature map F^l_{ik} for i^{th} filter at spatial location k in layer (depth) l
3. compute the Gram matrix for each pair of features
$$G^l_{ij} = \sum_k F^l_{ik} F^l_{jk}$$
4. feed (initially random) image into CNN
5. compute L2 distance between Gram matrices of original and new image
6. backprop to get gradient on image pixels
7. update image and go to step 5.

Neural Texture Synthesis

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

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

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

content $+$ style $\rightarrow$ new image

Neural Style Transfer

Neural Style Transfer

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

$$\begin{aligned} E_{\text{total}} &= \alpha\, E_{\text{content}} + \beta\, E_{\text{style}} \\ &= \frac{\alpha}{2}\sum_{i,k}||F_{ik}^l(x) - F_{ik}^l(x_c)||^2 + \frac{\beta}{4}\sum_{l=0}^{L}\frac{w_l}{N_l^2 M_l^2}\sum_{i,j}(G_{ij}^l - A_{ij}^l)^2 \end{aligned}$$

where

x_c, x	$=$	content image, synthetic image
F_{ik}^l	$=$	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.
- “Understanding the difficulty of training deep feedforward neural networks”, Glorot & Bengio, 2010.
- “Batch normalization: Accelerating deep network training by reducing internal covariate shift”, Ioffe & Szegedy, ICML 2015.
- “Deep Residual Learning for Image Recognition”, He et al., 2016.
- “Densely Connected Convolutional Networks”, Huang et al., 2016.
- “A Neural Algorithm of Artistic Style”, Gatys et al., 2015.

