10417/10617
Intermediate Deep Learning:
Fall 2023

Yuanzhi Li / Russ Salakhutdinov

Machine Learning Department

Using neural network in practice Part I: Vision.

So we have learned how to train a neural network

- Now it’s time to use it in practice.
- Vision (recognizing images) was among the first applications of deep learning.
- Given an image, we want to use neural networks to predict its labels.

Deep learning in Vision

- AlexNet (2011), a convolution neural network to recognize images (trained on ImageNet dataset).
- ZFNet (2012), a convolution + deconvolution neural network to recognize images.
- VGG (2014), a convolution neural network to recognize images.
- ResNet (2015), a convolution neural network with residual link to recognize images.
- MobileNet (2017), a depthwise separate convolution neural network to recognize images.
- Vision Transformers (ViTs) (2020), a transformer-based vision model that leaves all convolution neural networks in dust…

Deep learning in vision

Although convolution neural networks are replaced by ViTs now…

We still want to learn this very important discovery in history.

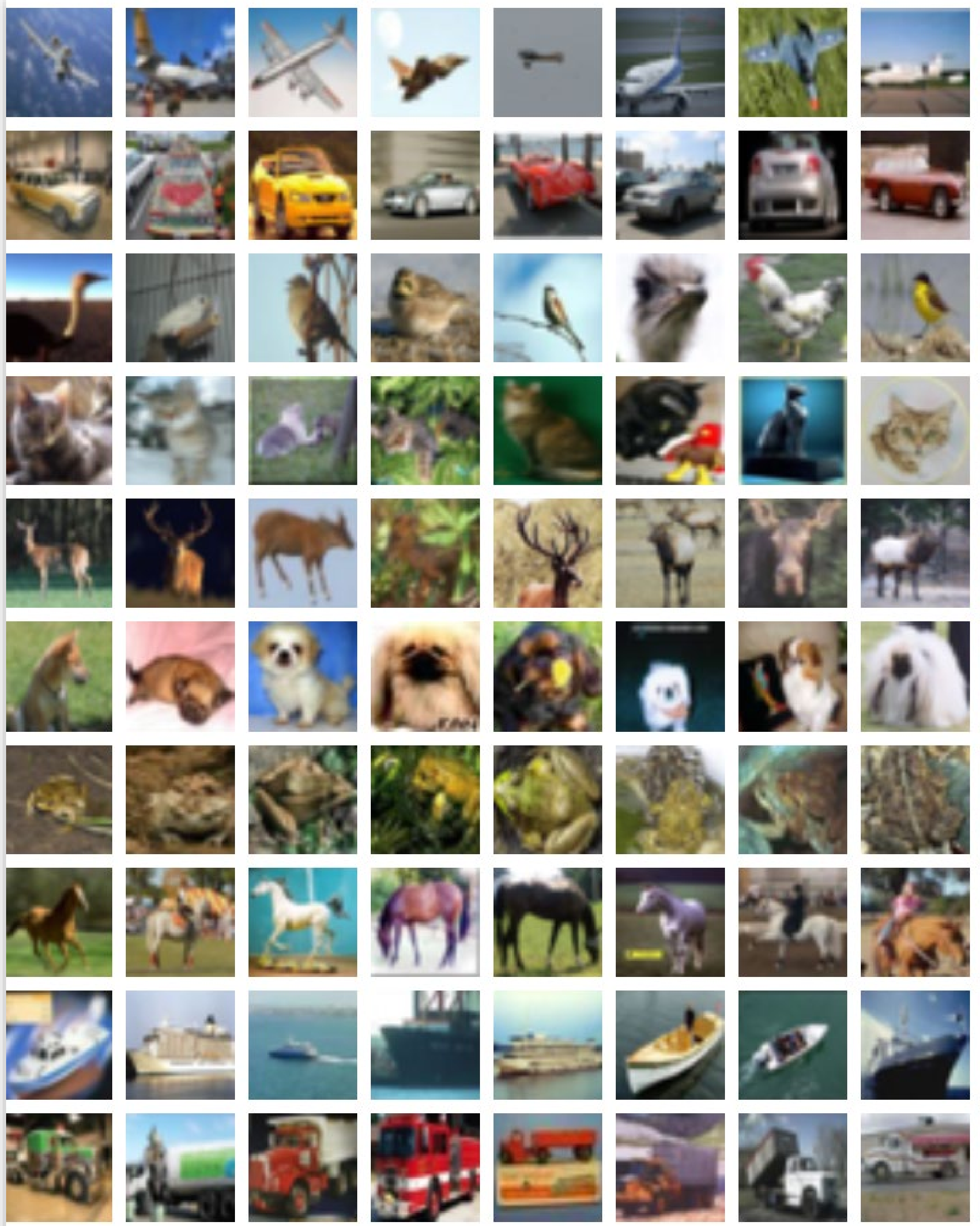
ViTs also simulates convolution neural network in practice (see for example the work of Samy Jelassi and me).

Images

- Given a (square) RGB-based image x.
- It's best to view x as a 3-tensor, of dimension $d \times d \times 3$.
- Here, d is the width/height of the image, and 3 is the number of channels of the image.
- We can think of the (i, j)-th pixel of the image has RGB value
  - $x[i\ , j, :]\ in\ R^3$ (R, G, B).

Deep learning for image recognization

- Given a (square) RGB-based image x.
- It's best to view x as a 3-tensor, of dimension $d \times d \times 3$.
- So, how do we apply a neural network on this input x?
    - One way is to flatten x to a vector in dimension $3d^2$, then apply our old friend MLP to it, and output the label of the image (like whether is a car, a cat, a dog etc).
    - Why is this a bad idea?



Spatial structure of images

- Think about a image x, let us create a new image x' by shifting every pixel of x by one (to the left).
- So, $x'[i, j, k] = x[i, j+1, k]$
- Are the two images having the same label?
  - Yes!
- But what would $h(W, x')$ look like when comparing to $h(W, x)$?

MLP lacks spatial structure

- What would $h(W, x')$ look like when comparing to $h(W, x)$?
- Let's consider $h(W, x) \ = \sigma(w^T x + b) = \sigma(\sum_{i,j,r} w_{i,j,r} x_{i,j,r} + b)$
- $h(W, x') = \sigma(\sum_{i,j,r} w_{i,j-1,r} x_{i,j,r} + b)$
- It has nothing to do with x, unless $w_{i,j,r}$ is close to $w_{i,j-1,r}$ (which is not guaranteed in MLP).
- We want to make it guaranteed by posting some constraints on the weights.

Convolution neural network

Can we design a neural network that respects the “spatial structure” of the image?

Meaning that if we shift the image, the output of the network stays relatively unchanged, regardless of the weights of the network?

Averaging Operation

- $h(W, x) \ = \sigma(\sum_{i,j,r} w_{i,j,r} x_{i,j,r} + b)$
- $h(W, x') = \sigma(\sum_{i,j,r} w_{i,j-1,r} x_{i,j,r} + b)$
- A naïve solution to make the output of h relatively unchanged: setting all the weights $w_{i,j,r}$ to be equal.
- This makes $h(W, x)$ unchanged…But this is just the (global) average of the coordinate of x, which is weak in terms of representation power.

Average Pooling

Idea: Instead of averaging over all the coordinates of x, what if we take an average over a subset of coordinates of x?

This is the average pooling operation.

Average Pooling

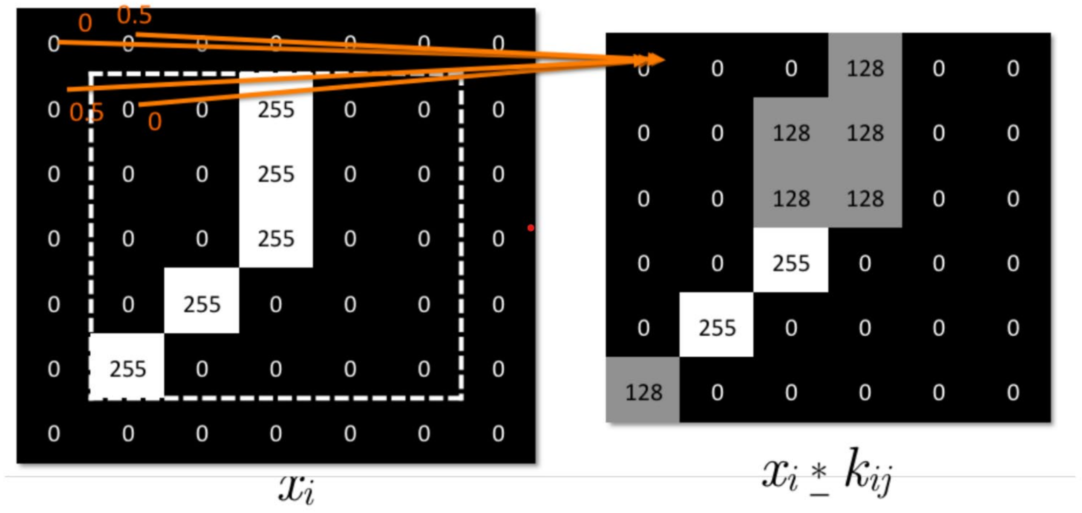
- An average pooling layer takes input x of shape $d \times d \times C$ (width x height x number of channels).
- It outputs y of shape $d' \times d' \times C, d' \approx \frac{d}{m}$
- $y[i, j, k] = \sum_{r,s \in [p]} x\ [m \times i + r, m \times j + s, k]$.
- p is called the “kernel size”, m is called the “stride”
- Each output coordinate of y is a “local average of x”.
- Average pooling operation is robust to shift: If we shift x by m (left, right, up, down) to x’, then we also shift y by one (left, right, up, down) to y’.

Convolution Operation

- Using "Local weighted average" instead of unweighted average.
- A convolution layer takes input x of shape $d \times d \times C$ (width x height x number of channels).
- The output of the convolution layer y is of shape $d' \times d' \times C'$
- The computation is given by:
- $y[i,j,k] = \sum_{r,s \in [p], l \in [C]} w[r,s,k,l] x[m \times i + r, m \times j + s, l] + b[k]$
- So w is of dimension $p \times p \times C' \times C$.
- $d' \approx d/m$

Convolution operation

- $y[i,j,k] = \sum_{r,s\in[p],l\in[C]} w[r,s,k,l]x[m\times i+r, m\times j+s, l]$
- So w is in dimension $p \times p \times C' \times C$
- Here, p is called the “kernel size”, m is called the “stride”, C’ is called Out_Channels.
- Here, each coordinate of y is some local weighted average of x.



0	0	0	0	0	0	0
0	0	0	255	0	0	0
0	0	0	255	0	0	0
0	0	0	255	0	0	0
0	0	255	0	0	0	0
0	255	0	0	0	0	0
0	0	0	0	0	0	0

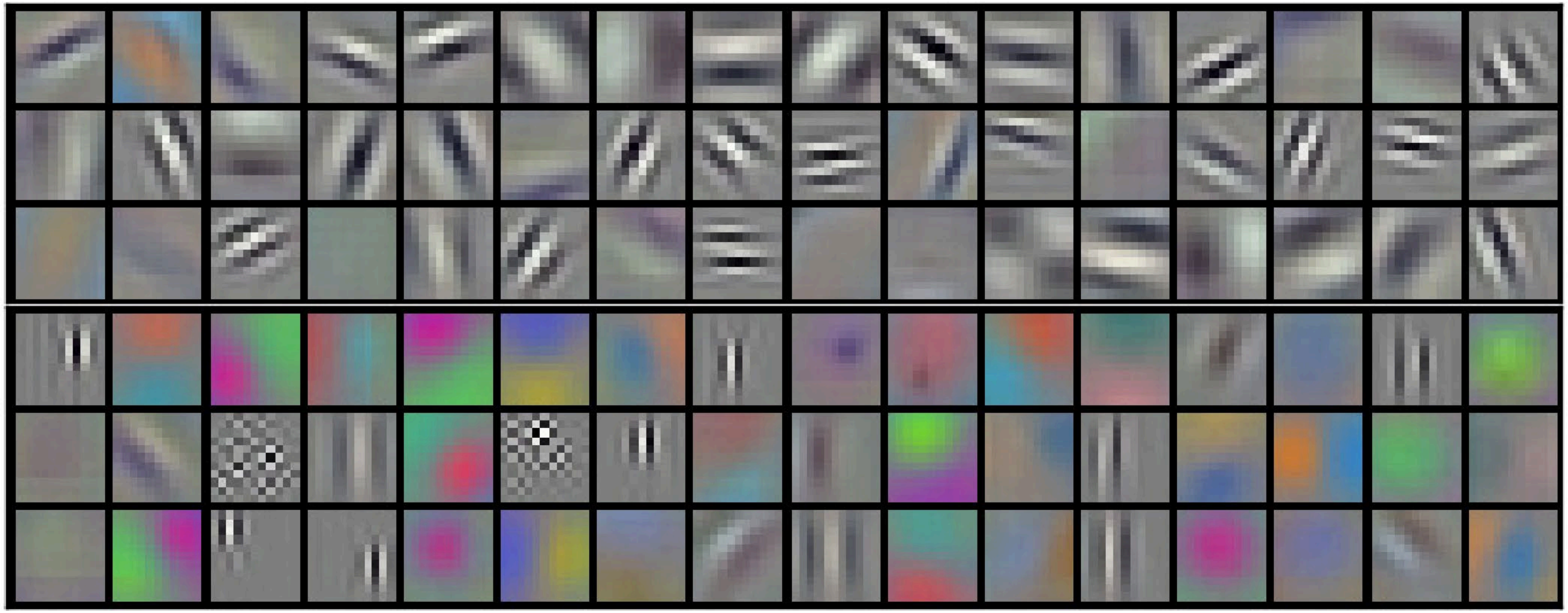
0	0	0	128	0	0
0	0	128	128	0	0
0	0	128	128	0	0
0	0	255	0	0	0
0	255	0	0	0	0
128	0	0	0	0	0

Convolution operation

- $y[i,j,k] = \sum_{r,s\in[p],l\in[C]} w[r,s,k,l]x[m\times i+r, m\times j+s, l] + b[k]$
- Convolution operation is robust to the shift: If we shift x by m (left, right, up, down) to x’, then we also shift y by one (left, right, up, down) to y’.

Convolution Filters

Visualization of the convolution filter (the weight of the first layer of a convolution network):



Convolution Network

- A convolution network is a network that uses convolution layers instead of all MLPs.
- An example of the convolution network is:
- $h(W, X) = MLP \circ Flatten\left(\sigma \circ Conv_L \circ \sigma \circ Conv_{L-1}\sigma \circ \cdots \circ \sigma \circ Conv_1(x)\right)$
- Flatten means flatten a $d \times d \times C$ tensor to a d^2C dimension vector.

Convolution networks in history

AlexNet

Image

- -> Convolution (5x5 kernel, stride = 1, Out_Channels = 6)->Sigmoid
- -> Pool(2x2 kernel, stride = 2)
- -> Convolution (5x5 kernel, stride = 1, Out_Channels = 16) -> Sigmoid
- -> Pool(2x2 kernel, stride = 2) -> flatten
- -> 3 layer MLP -> Output.

Convolution networks in history

- VGG Network:
- Image
  - Conv(3x3 , stride = 1, out_channels = 64) -> ReLU -> pool(2x2, stride = 2)
  - Conv(3x3 , stride = 1, out_channels = 128) -> ReLU -> pool(2x2, stride = 2)
  - (Conv(3x3 , stride = 1, out_channels = 256) -> ReLU)x2 -> pool(2x2, stride = 2)
  - (Conv(3x3 , stride = 1, out_channels = 512) -> ReLU)x2 -> pool(2x2, stride = 2)
  - ->Flatten->2-layer MLP-> Output

Convolution networks in history

- VGG Network with BN:
- Image
  - Conv(3x3 , stride = 1, out_channels = 64) ->BN-> ReLU -> pool(2x2, stride = 2)
  - Conv(3x3 , stride = 1, out_channels = 128) ->BN-> ReLU -> pool(2x2, stride = 2)
  - (Conv(3x3 , stride = 1, out_channels = 256) -> BN -> ReLU)x2 -> pool(2x2, stride = 2)
  - (Conv(3x3 , stride = 1, out_channels = 512) -> BN -> ReLU)x2 -> pool(2x2, stride = 2)
  - ->Flatten->2-layer MLP-> Output

Convolution networks in history

ResNet:

Residual_Block(channels = K):

- x - > Conv(3x3, stride = 1 (or 2), out_channels = K) -> BN -> ReLU
- -> Conv(3x3, stride = 1, out_channels = K) -> BN -> add x

Image -> Conv(7x7, stride = 2, out_channels = 64) -> BN-> ReLU

- -> Pool(2x2, stride = 2)
- -> Residual_Block(channels = 64) x 3
- -> Residual_Block(channels = 128) x 4
- -> Residual_Block(channels = 256) x 5
- -> Residual_Block(channels = 512) x 3
- -> Pool(2x2, stride = 2) -> flatten->2-layer MLP -> Output