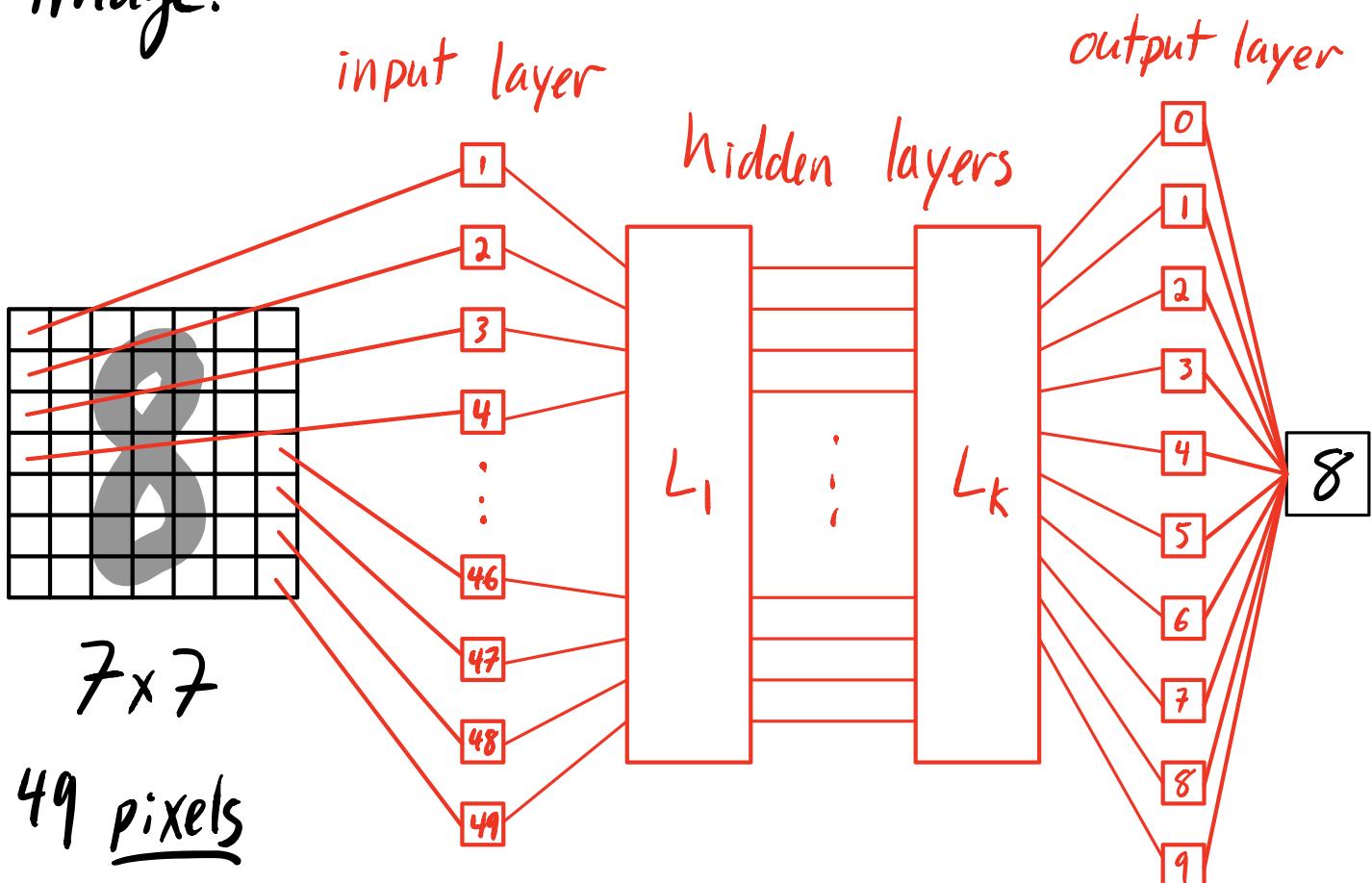


VII.2] Convolutional Neural Nets

Image classification is a challenging problem for neural nets since the number of features is $m \cdot n$ for an $m \times n$ grayscale image or $3mn$ for an $m \times n$ color (RGB) image.



The output layer has a probability for each digit: $0 \leq p_0, \dots, p_9 \leq 1$, $\sum_{i=0}^9 p_i = 1$.

The highest probability is the neural net's best guess for the digit in the original 7×7 image.

If we were to only use the basic operations of affine functions, activation functions, and composition, the number of parameters we would need to optimize would be enormous.

However, images have a unique property that we can use to our advantage: The relationship of pixels that are far apart is less important than the relationship between nearby pixels.

Also, the rule that describes this local relationship should apply equally well no matter which part of the image you are considering.
(ie shift-invariant)

A convolution is a linear combination of local groups of pixels.

Example:

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 3 & 2 & 0 & 0 \\ \hline 0 & 0 & 7 & 6 & 5 & 0 & 0 \\ \hline 0 & 0 & 7 & 6 & 3 & 0 & 0 \\ \hline 0 & 0 & 3 & 8 & 2 & 0 & 0 \\ \hline 0 & 0 & 7 & 4 & 5 & 0 & 0 \\ \hline 0 & 0 & 8 & 5 & 5 & 0 & 0 \\ \hline 0 & 0 & 4 & 3 & 1 & 0 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|c|c|c|} \hline 22 & 65 & 79 & 51 & 15 \\ \hline 24 & 74 & 89 & 52 & 13 \\ \hline 20 & 66 & 84 & 50 & 12 \\ \hline 25 & 71 & 84 & 55 & 17 \\ \hline 27 & 71 & 77 & 49 & 16 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \cdot \begin{array}{|c|c|c|} \hline 0 & 0 & 1 \\ \hline 0 & 0 & 7 \\ \hline 0 & 0 & 7 \\ \hline \end{array} = 1 \cdot 1 + 2 \cdot 7 + 1 \cdot 7 = 22$$

$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \cdot \begin{array}{|c|c|c|} \hline 0 & 1 & 3 \\ \hline 0 & 7 & 6 \\ \hline 0 & 7 & 6 \\ \hline \end{array} = 2 \cdot 1 + 4 \cdot 7 + 2 \cdot 7 + 1 \cdot 3 + 2 \cdot 6 + 1 \cdot 6 = 65 \quad \text{etc.}$$

This is a linear map $C: \mathbb{R}^{49} \rightarrow \mathbb{R}^{25}$, but only needs 9 parameters ($9 \ll 49 \cdot 25$).

This linear map can be viewed as a 25×49 matrix with many zeros: only 9 nonzeros on each row and 40 zeros.

e.g. $C: \mathbb{R}^4 \rightarrow \mathbb{R}^4$

$$\begin{array}{|c|c|} \hline x_1 & x_3 \\ \hline x_2 & x_4 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline P_{11} & P_{12} & P_{13} \\ \hline P_{21} & P_{22} & P_{23} \\ \hline P_{31} & P_{32} & P_{33} \\ \hline \end{array} = \begin{array}{|c|c|} \hline q_{11} & q_{12} \\ \hline q_{21} & q_{22} \\ \hline \end{array}$$

$$\begin{bmatrix} x_1 & x_2 & 0 & x_3 & x_4 & 0 & 0 & 0 & 0 \\ 0 & x_1 & x_2 & 0 & x_3 & x_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & x_2 & 0 & x_3 & x_4 & 0 \\ 0 & 0 & 0 & 0 & x_1 & x_2 & 0 & x_3 & x_4 \end{bmatrix} \begin{bmatrix} P_{11} \\ P_{21} \\ P_{31} \\ P_{12} \\ P_{22} \\ P_{32} \\ P_{13} \\ P_{23} \\ P_{33} \end{bmatrix} = \begin{bmatrix} q_{11} \\ q_{21} \\ q_{12} \\ q_{22} \end{bmatrix}$$

Padding the boundary

If we would like the output of the convolution to have the same size as the input, we can pad the boundary with zeros or some other reasonable values.

Example: $C: \mathbb{R}^9 \rightarrow \mathbb{R}^{16}$ (not the same size)

$$\begin{array}{c|c} \begin{matrix} x_1 & x_3 \\ x_2 & x_4 \end{matrix} & * \\ \hline \end{array} \quad \begin{array}{c|c|c|c|c} 0 & 0 & 0 & 0 & 0 \\ 0 & P_{11} & P_{12} & P_{13} & 0 \\ 0 & P_{21} & P_{22} & P_{23} & 0 \\ 0 & P_{31} & P_{32} & P_{33} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array} = \begin{array}{c|c|c|c} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ q_{41} & q_{42} & q_{43} & q_{44} \end{array}$$

Note: 3×3 convolution of 3×3 image
with padding gives an output
of 3×3 (ie the same size).

Stride

Instead of moving the **convolution window** one pixel each time, we can move a stride of S pixels each time.

Example: A stride of 2 would give us:

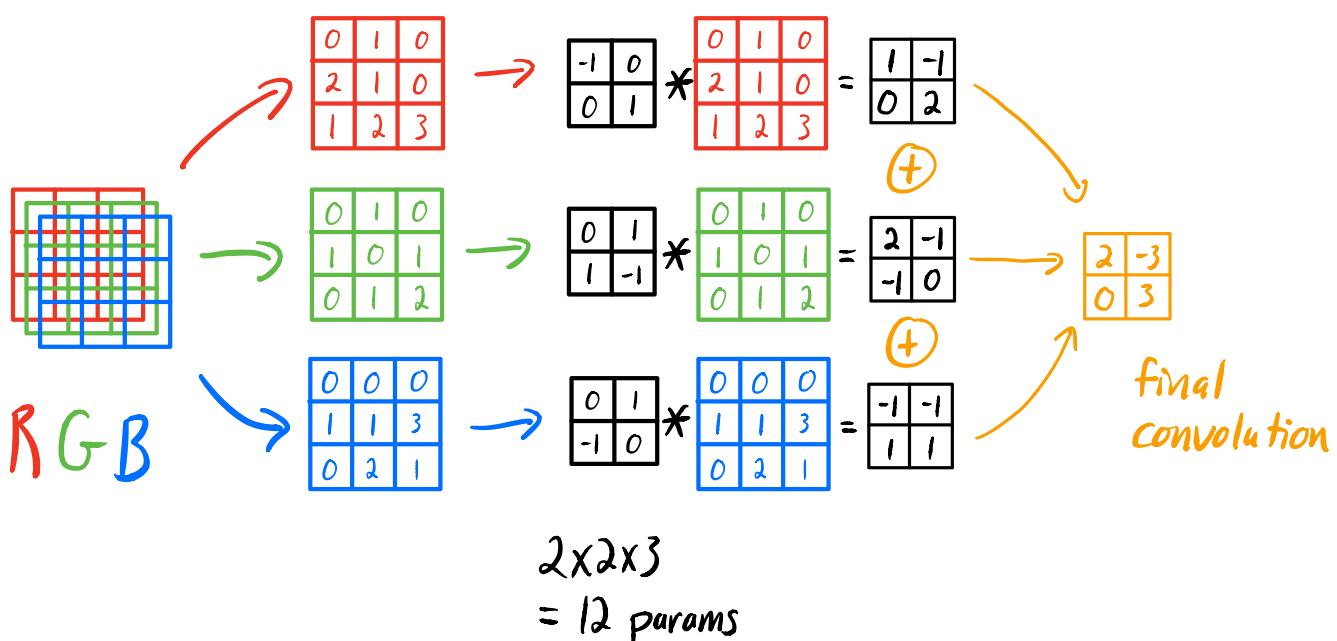
$$\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|c|c|c|c|c|} \hline 0 & 0 & 1 & 3 & 2 & 0 & 0 & 0 \\ \hline 0 & 0 & 7 & 6 & 5 & 0 & 0 & 0 \\ \hline 0 & 0 & 7 & 6 & 3 & 0 & 0 & 0 \\ \hline 0 & 0 & 3 & 8 & 2 & 0 & 0 & 0 \\ \hline 0 & 0 & 7 & 4 & 5 & 0 & 0 & 0 \\ \hline 0 & 0 & 8 & 5 & 5 & 0 & 0 & 0 \\ \hline 0 & 0 & 4 & 3 & 1 & 0 & 0 & 0 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 22 & 79 & 15 \\ \hline 20 & 84 & 12 \\ \hline 27 & 77 & 16 \\ \hline \end{array} .$$

Channels

A grayscale image only has 1 channel while an RGB image has 3 channels.

This is the number of incoming channels to the first convolutional layer of the neural net.

The number of channels coming out of the convolutional layer can be any number we would like. This gives us different convolutions of the same image which may each detect different important features.



This is $\text{Conv}((2,2), 3 \Rightarrow 1)$ in Flux.jl. If we want 12 such convolutions, we would use $\text{Conv}((2,2), 3 \Rightarrow 12)$; The input has 3 channels, the output has 12 channels:

$$\underbrace{3 \times 3 \times 3}_{\substack{\text{image} \\ \text{size}}} \rightarrow \underbrace{2 \times 2 \times 12}_{\substack{\text{output} \\ \text{size}}} \uparrow \# \text{ of convolutions}$$

Karpathy's formula

N = input size (in one dimension)

E = convolution size (in one dimension)

P = pad size (on each end)

S = stride amount

M = output size (in one dimension)

$$M = \frac{N - E + 2P}{S} + 1$$

$$\frac{3 - 3 + 2 \cdot 1}{2} + 1 \\ = 2 \checkmark$$

Examples:

(1)

x_1	x_3
x_2	x_4

*

0	0	0	0	0
0	P_{11}	P_{12}	P_{13}	0
0	P_{21}	P_{22}	P_{23}	0
0	P_{31}	P_{32}	P_{33}	0
0	0	0	0	0

=

q_{11}	q_{12}	q_{13}	q_{14}
q_{21}	q_{22}	q_{23}	q_{24}
q_{31}	q_{32}	q_{33}	q_{34}
q_{41}	q_{42}	q_{43}	q_{44}

$$N = 3, E = 2, P = 1, S = 1$$

$$M = \frac{3 - 2 + 2 \cdot 1}{1} + 1 = 4$$

$$(2) \quad N=3, \quad E=3, \quad P=1, \quad S=2$$

$M = \underline{2}$ (picture it first, then check with the formula)

(3) Input to Conv is a 4-dimensional array $W \times H \times C \times B$, where

W = image width

H = image height

C = # of channels ($RGB \Rightarrow C=3$)

B = batch size

Input $60 \times 40 \times 3 \times 100$ into

$\text{Conv}((4,4), 3 \Rightarrow 10, \text{pad}=1, \text{stride}=2)$

has output 30 \times 20 \times 10 \times 100.

$$\frac{60 - 4 + 2 \cdot 1}{2} + 1 = 30$$

$$\frac{40 - 4 + 2 \cdot 1}{2} + 1 = 20$$

Softmax

To ensure the final layer outputs a probability for each classification, we pass the final output

$$w_1, w_2, \dots, w_k$$

through the softmax function:

$$p_1 = \frac{e^{w_1}}{s}, \dots, p_k = \frac{e^{w_k}}{s}$$

where $s = \sum_{i=1}^k e^{w_i}$.

Note that the w_i 's can be any real numbers. We are guaranteed to have $0 < p_1, \dots, p_k < 1$ and $\sum_{i=1}^k p_i = 1$.

Also, $w_i < w_j \Leftrightarrow p_i < p_j$.

To simplify, we could take the largest w_i as the winner.

Residual Networks (ResNets)

If the neural net is very deep (ie has many layers), we may have "vanishing gradients" making the optimization of the parameters difficult.

To solve this problem, we can give some layers the possibility to skip the next layer.

