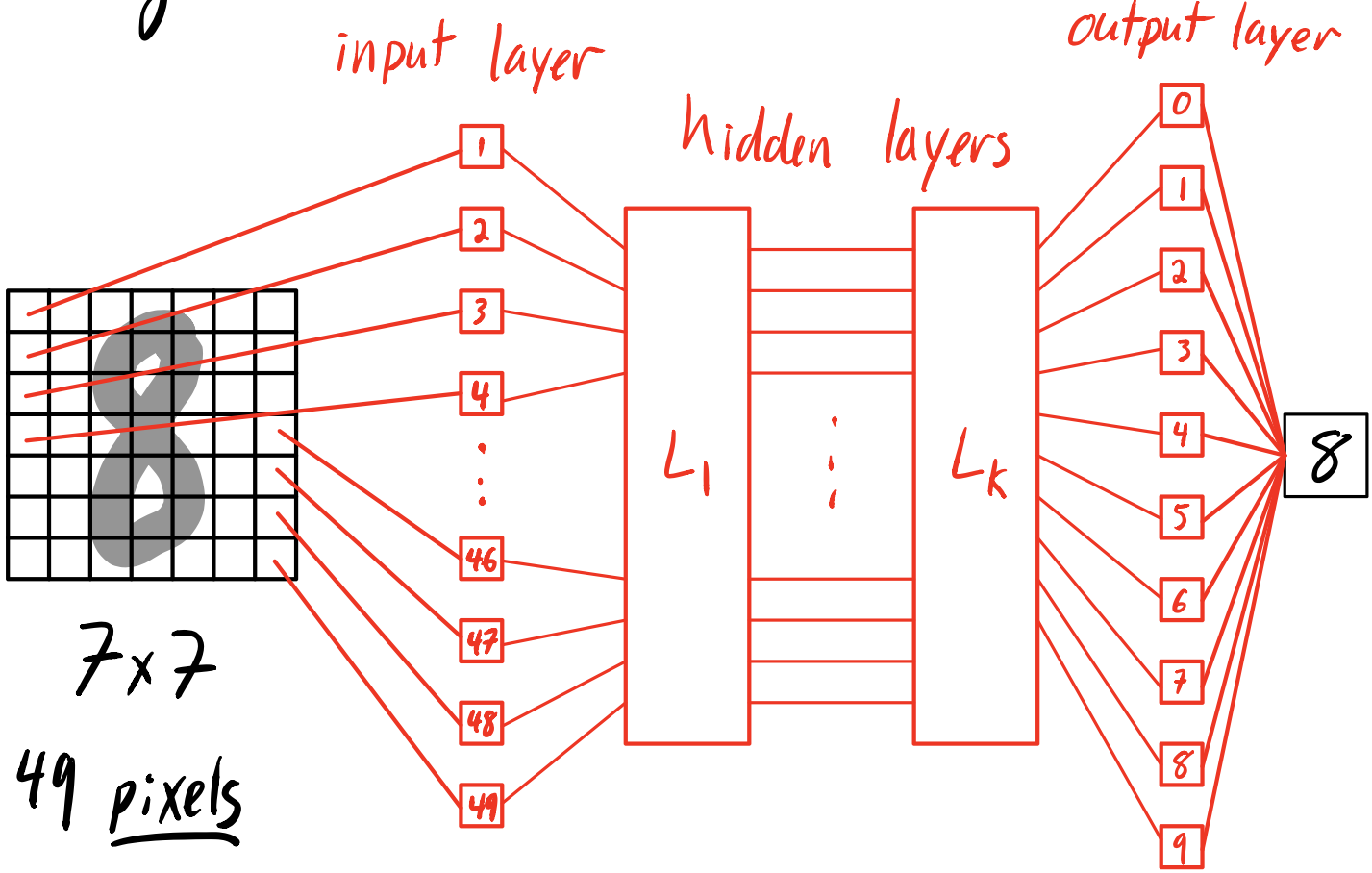
# VII.2 Convolutional Neural Nets

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




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

The highest probability is the neural net's best guess for the digit in the original 7x7 image.

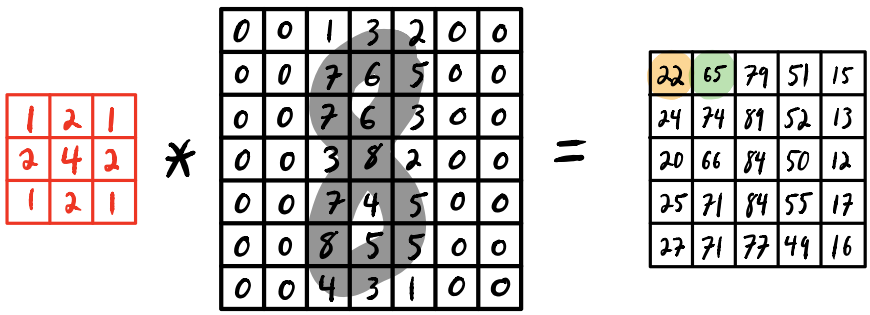
If we were to only use the basic operations of affine functions, activation functions, and composition, the number of parameters we would need to optimize would be enormous.

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 of nearby pixels.

Also, the rule that describes this local relationship should apply equally well no matter which part of the image you are considering.

A <u>convolution</u> is a linear combination of local groups of pixels.

<u>Example</u>:




$$\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} \cdot \begin{bmatrix}0&0&1\\0&0&7\\0&0&7\end{bmatrix} = 1\cdot 1 + 2\cdot 7 + 1\cdot 7 = 22$$

$$\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} \cdot \begin{bmatrix}0&1&3\\0&7&6\\0&7&6\end{bmatrix} = \begin{matrix}2\cdot 1 + 4\cdot 7 + 2\cdot 7 \\ + 1\cdot 3 + 2\cdot 6 + 1\cdot 6\end{matrix} = 65$$

etc.

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

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

eg $C : \mathbb{R}^9 \to \mathbb{R}^4$

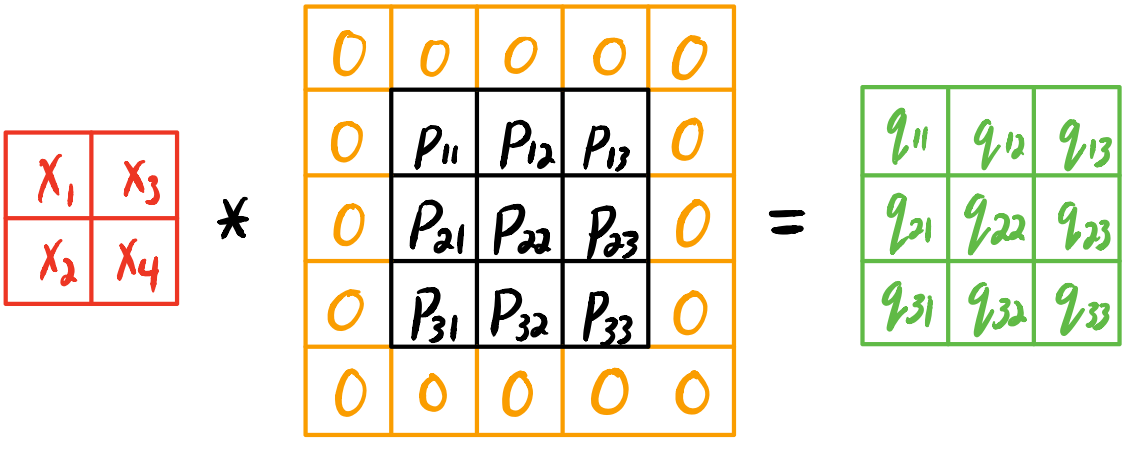
$$\begin{bmatrix} x_1 & x_3 \\ x_2 & x_4 \end{bmatrix} * \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} = \begin{bmatrix} q_{11} & q_{12} \\ q_{21} & q_{22} \end{bmatrix}$$

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

# 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.

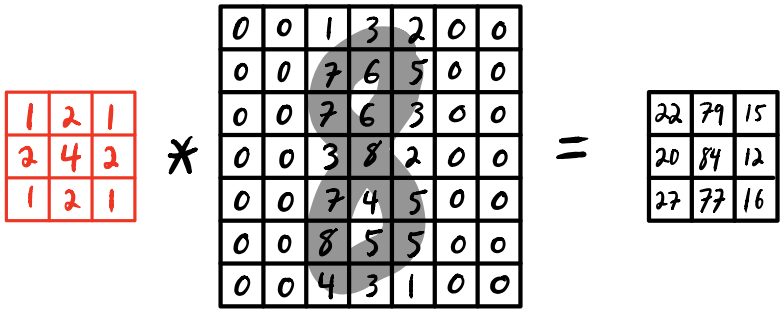
**Example:** $C: \mathbb{R}^3 \to \mathbb{R}^3$



# 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:

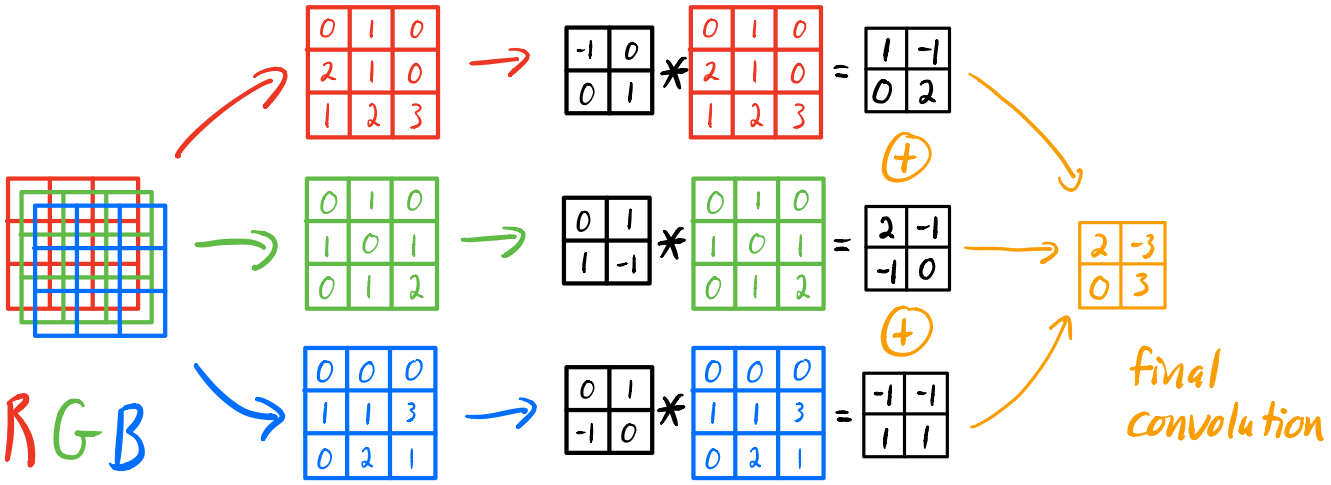


# Channels

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.




2x2x3
= 12 params

This is Conv((2,2), 3 ⇒ 1) in Flux.jl. If we want 12 such convolutions, we would use Conv((2,2), 3 ⇒ 12); the input has 3 channels, the output has 12 channels:

$$\underbrace{3 \times 3}_{\text{image size}} \times \underbrace{3}_{\text{RGB}} \longrightarrow \underbrace{2 \times 2}_{\text{output size}} \times \underbrace{12}_{\text{\# of convolutions}}.$$

# Karpathy's formula

$N$ = input size (in one dimension)

$E$ = convolution size

$P$ = pad size (on each end)

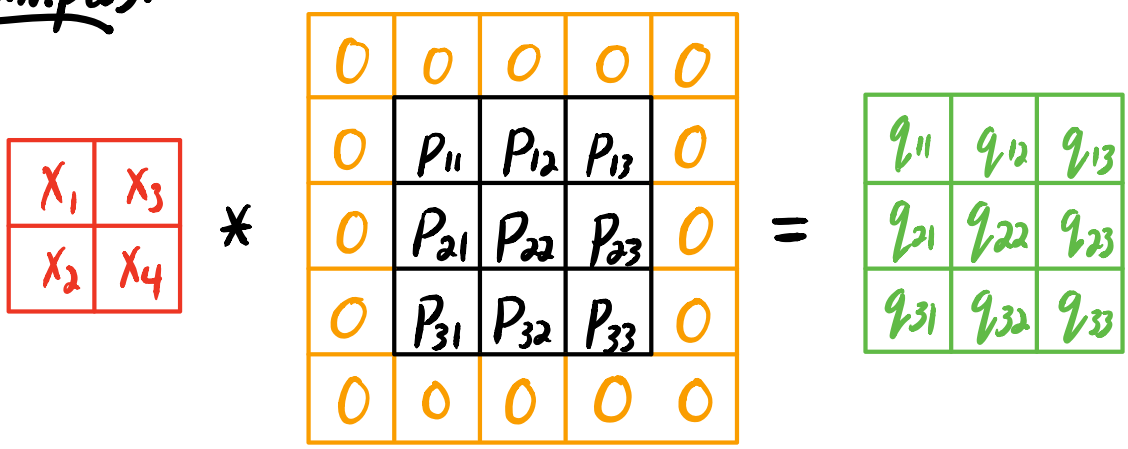
$S$ = stride amount

$M$ = output size

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

Examples:

(1)




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

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

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

$M = ?$ (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

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

has output ___ x ___ x ___ x ___ .

# Softmax

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

$$w_1, w_2, \ldots, w_k$$

through the softmax function:

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

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 $0 < p_1, \ldots, p_k < 1$ and $\sum_{i=1}^{k} p_i = 1$.

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

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.