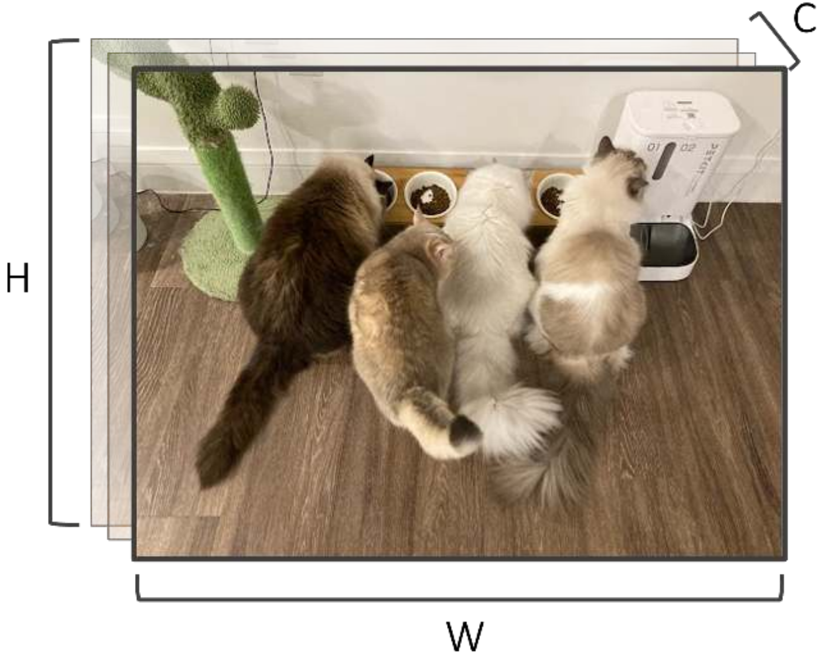
Convolutions

Images

Image: $x \in [0...255]^{C \times H \times W}$

- Height H
- Width W
- Channels C
  - Usually C=3: RGB

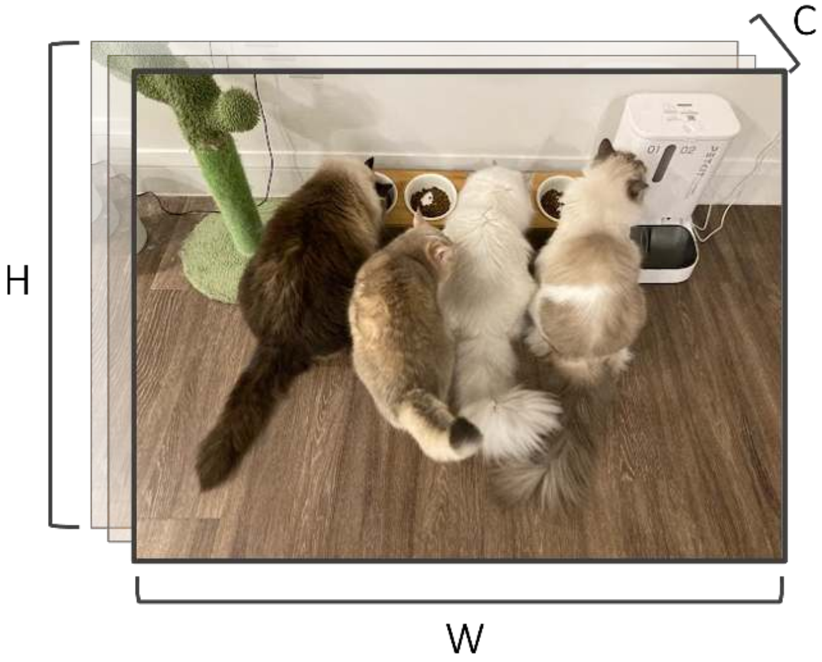


Images

Image: $x \in [0 ... 255]^{C \times H \times W}$

Image Format: CHW vs. HWC

- LuaTorch used *channels-first* (CHW)
- PyTorch adopted CHW as the standard
  - A mistake IMHO
- `memory_format = torch.channels_last` for HWC
- HWC is faster



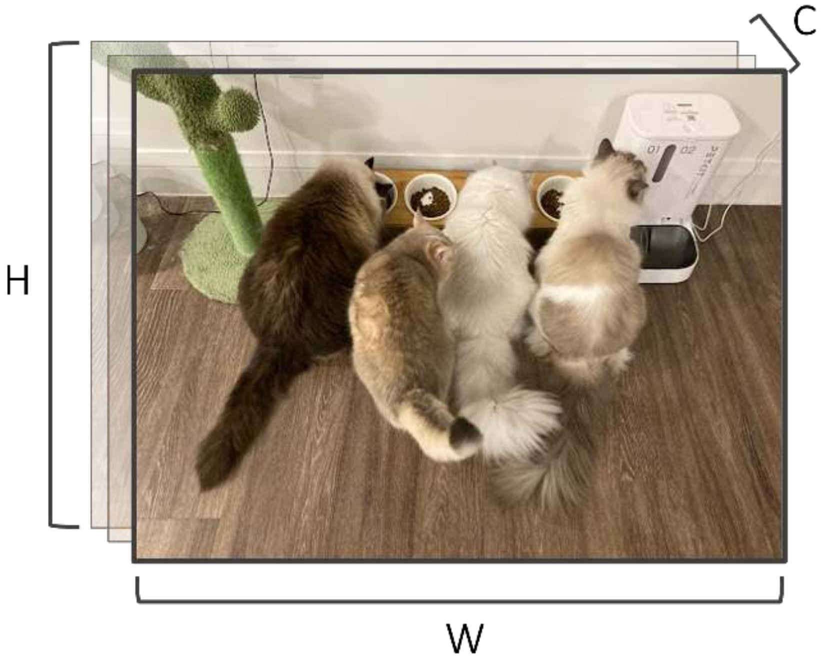
Images

Image: $x \in [0...255]^{C \times H \times W}$

- Height *H*
- Width *W*
- Channels *C* - RGB

High-dimensional

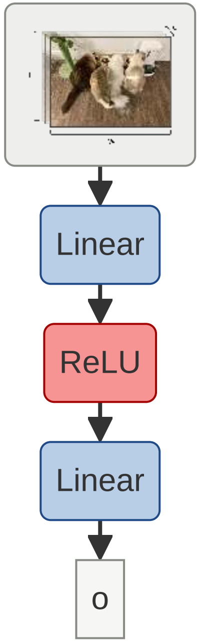
- A 1024 $\times$ 1024 image is dimension **3,145,728**!



Images in Fully-Connected Networks

How to use linear layers for images?

- Image $x \in [0...255]^{C \times H \times W}$
- Linear layer $f : \mathbb{R}^N \to \mathbb{R}^D$



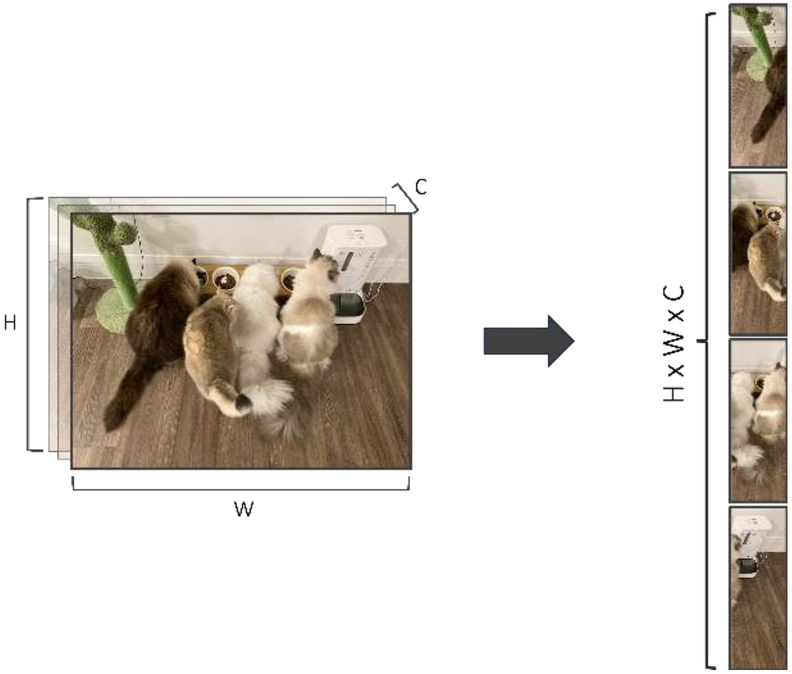
Images in Fully-Connected Networks

How to use linear layers for images?

- Image $x \in [0...255]^{C \times H \times W}$
- Linear layer $f : \mathbb{R}^N \to \mathbb{R}^D$

Option 1: Flatten an image into a vector

- $Flatten(\cdot) : \mathbb{R}^{C \times H \times W} \to \mathbb{R}^{CHW}$



Images in Fully-Connected Networks

How to use linear layers for images?

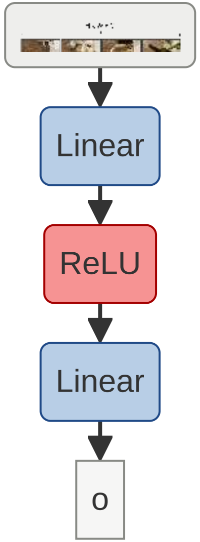
- Image $x \in [0...255]^{C \times H \times W}$
- Linear layer $f : \mathbb{R}^N \to \mathbb{R}^D$

Option 1: Flatten an image into a vector

- $\mathit{Flatten}(\cdot) : \mathbb{R}^{C \times H \times W} \to \mathbb{R}^{CHW}$
- Feed into fully-connected network:

$$\mathbb{R}^{CHW} \to \mathbb{R}^{D_1} \to \mathbb{R}^{D_2} \dots$$

How large should each linear layer be? ($D_1, D_2, \cdots$)

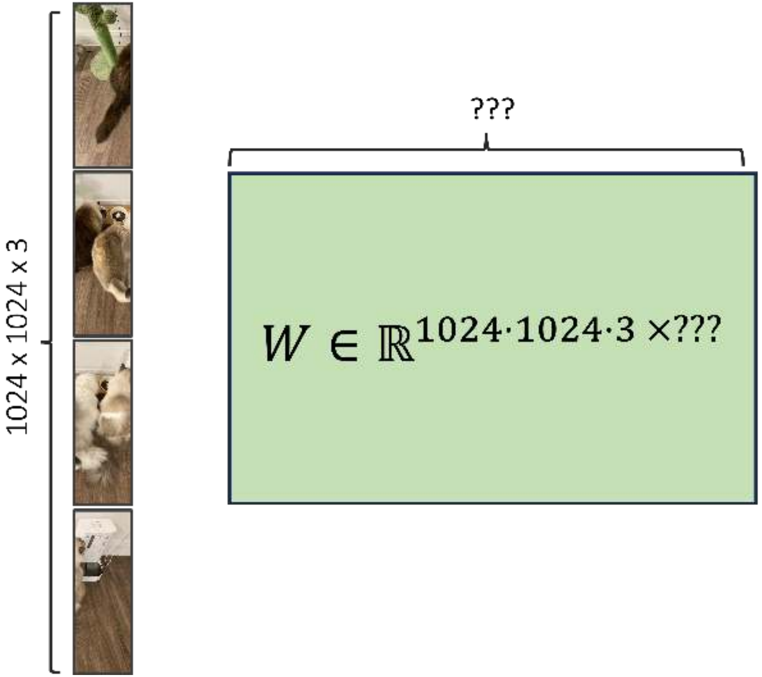


Images in Fully-Connected Networks

How Large Should the Output Dimension Be?

- Standard image classification has 1000 classes[1]!
- Intermediate activations should be larger (i.e. 4k)

How many parameters would this model have?

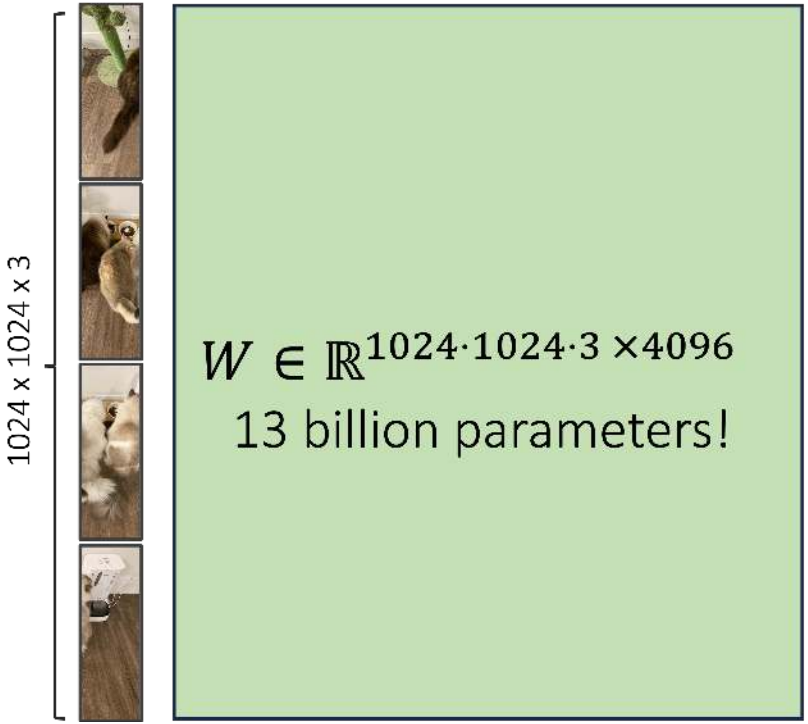


1. Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." IJCV 2015 ↩️

Images in Fully-Connected Networks

Linear: Image $\rightarrow$ 4096-dim.

- 13 billion parameters (same size as entire LLMs!)
- Extremely memory and parameter inefficient
- Minor: restricted to fixed size images



Images in Fully-Connected Networks

How to use linear layers for images?

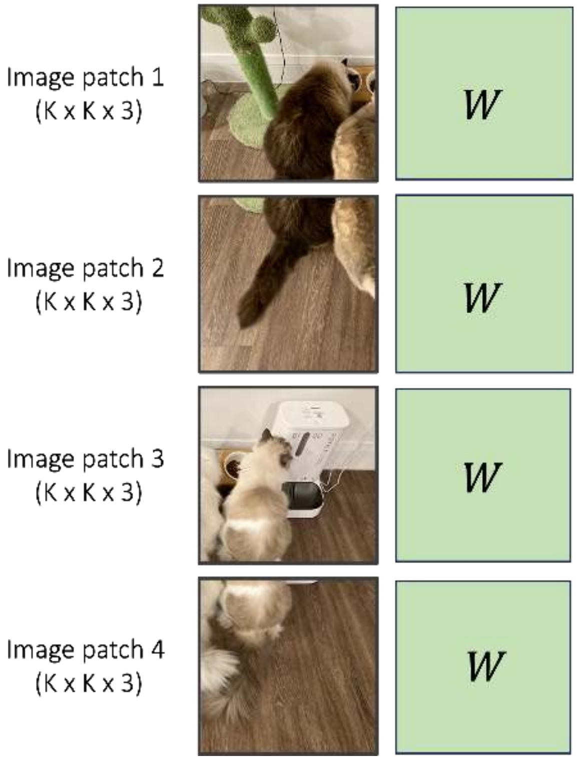
- Image $x \in [0...255]^{C \times H \times W}$
- Linear layer $f : \mathbb{R}^N \to \mathbb{R}^D$

Option 2: Split image into m patches

- Run a *separate* linear layer on each patch

$$\mathbb{R}^{CK^2} \to \mathbb{R}^{\frac{D}{M}}$$

- $CK^2 \times \frac{D}{M}$ parameters per patch
- $\frac{CWH \times D}{M}$ parameters total ($MK^2 \approx WH$)



Images in Fully-Connected Networks

How to use linear layers for images?

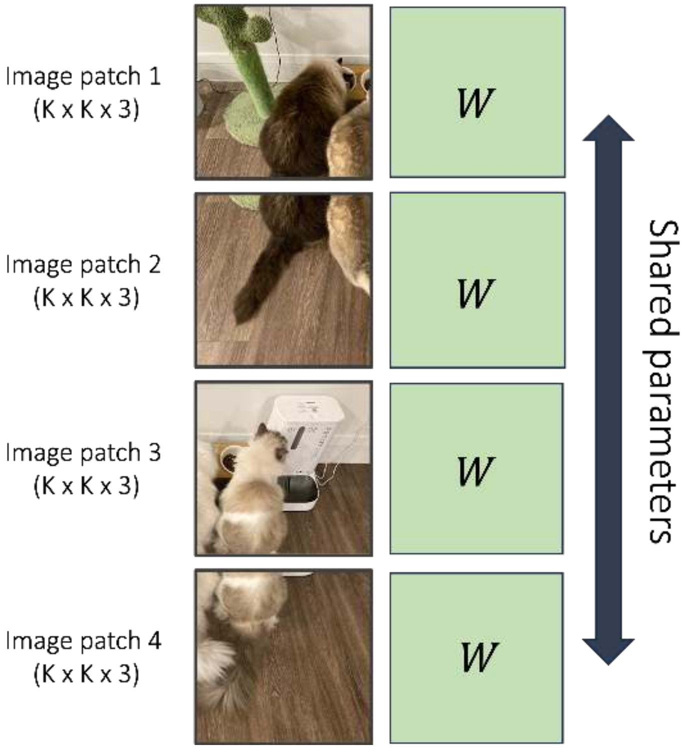
- Image $x \in [0...255]^{C \times H \times W}$
- Linear layer $f : \mathbb{R}^N \rightarrow \mathbb{R}^D$

Option 2.5: Split image into m patches

- Run a ***single*** linear layer on each patch

$$\mathbb{R}^{CK^2} \rightarrow \mathbb{R}^{\frac{D}{M}}$$

- $CK^2 \times \frac{D}{M}$ parameters total



Images in Fully-Connected Networks

Patches

- Cut up image content
- How do we choose the right K?

K = H = W

K = H/2 = W/2

K = H/3 = W/3

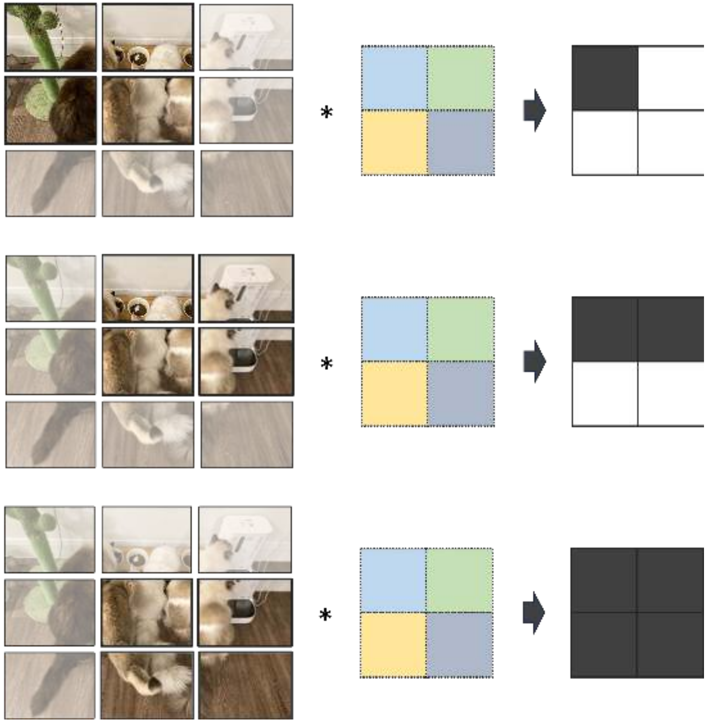
Images in Fully-Connected Networks

How to use linear layers for images?

- Image $x \in [0...255]^{C \times H \times W}$
- Linear layer $f : \mathbb{R}^N \to \mathbb{R}^D$

Option 3: Overlapping patches

- "Slide" linear layer over image
- Formally known as **convolution**



Convolution

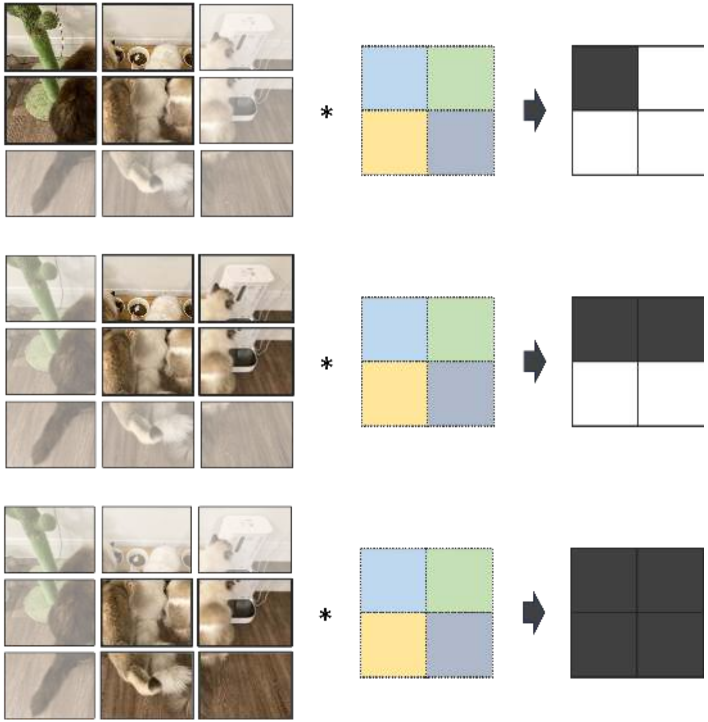
Input: $x \in \mathbb{R}^{C_1 \times H \times W}$

Output: $y \in \mathbb{R}^{C_2 \times (H-h+1) \times (W-w+1)}$

Parameters:

- Kernel: $\omega \in \mathbb{R}^{C_1 \times C_2 \times h \times w}$
- Bias: (optional) $b \in \mathbb{R}^{C_2}$

$$\underbrace{y_{i,j,k}}_{\text{output}} = \underbrace{b_i}_{\text{bias}} + \sum_{l=1}^{C_1} \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} \underbrace{x_{l,j+m,k+n}}_{\text{input}} \cdot \underbrace{\omega_{i,l,m,n}}_{\text{kernel}}$$

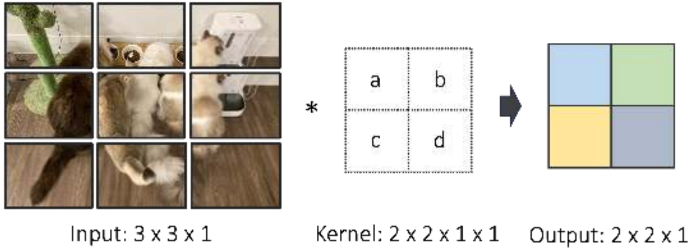


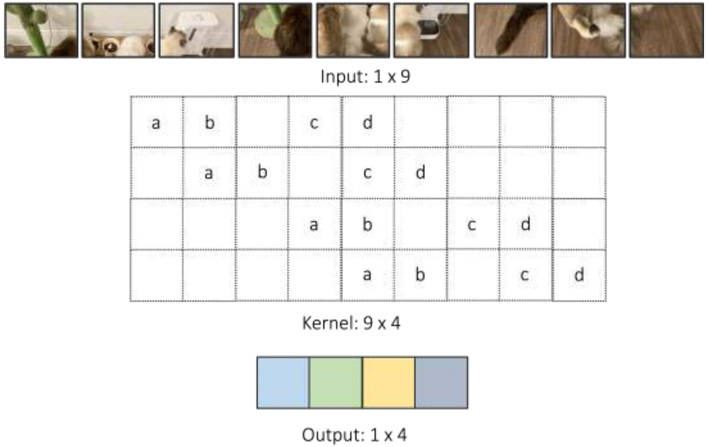
Convolution Is Fast

Sparsely Computed Linear Layer

- # parameters is **independent** of image resolution
- Memory-efficient!

$$y_{i,j,k} = b_i + \sum_{l=1}^{C_1} \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} x_{l,j+m,k+n} \quad \cdot \quad \underbrace{\omega_{i,l,m,n}}_{\text{independent of H or W}}$$





Convolution Is Memory-Efficient

Parameter Comparison

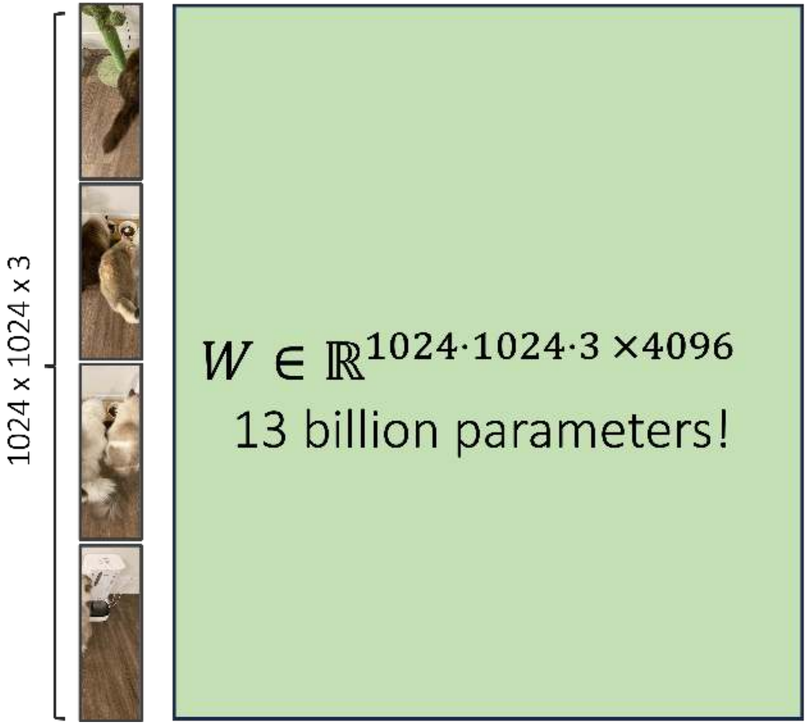
- Input: $x \in \mathbb{R}^{C_1 \times H \times W}$
- Output: $y \in \mathbb{R}^{C_2 \times H \times W}$
- Where $C_1 = C_2 = 3$ and $H = W = 1024$

Linear Layer

- $\sim 5 \times 10^{13}$ parameters

Convolution With 3x3 Kernel

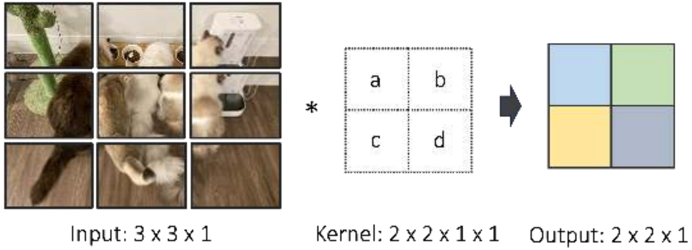
- < 500 parameters

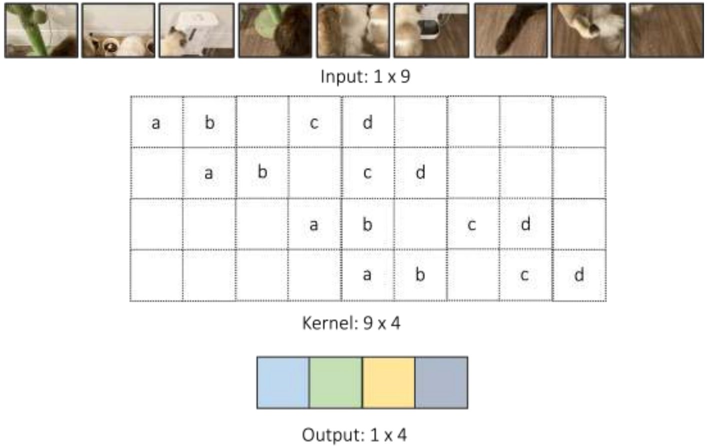


Convolution: What Do We Lose?

Computation Is Local

- Output aggregates information locally

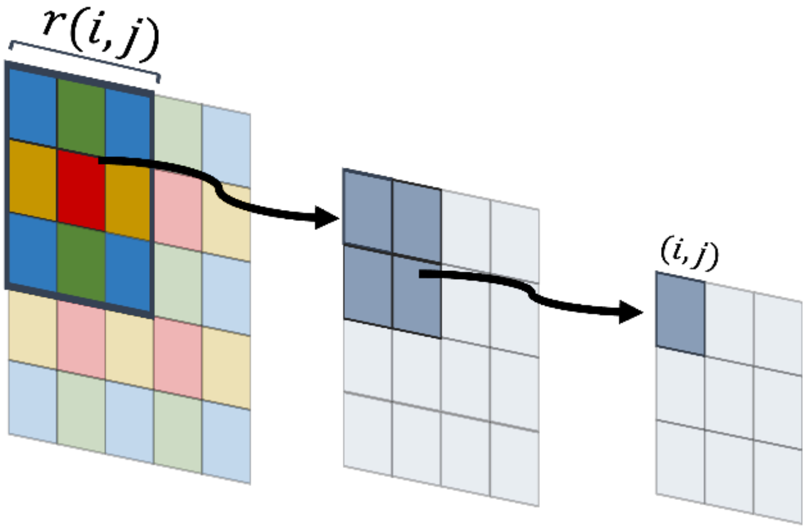




Receptive Fields

For each location (i, j) in output y -

- How far does the network "see"?
- What input locations affect output $y_{i,j}$?



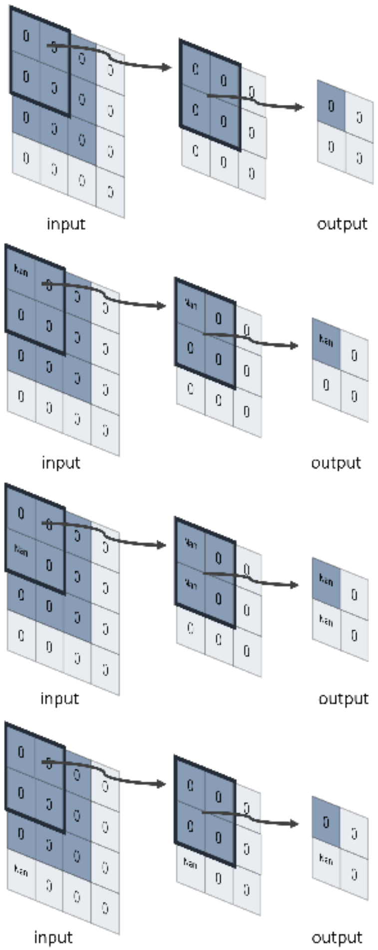
Receptive field of $y_{i,j}$

Computing Receptive Fields

Option 1: Lots of math

Option 2: Computationally

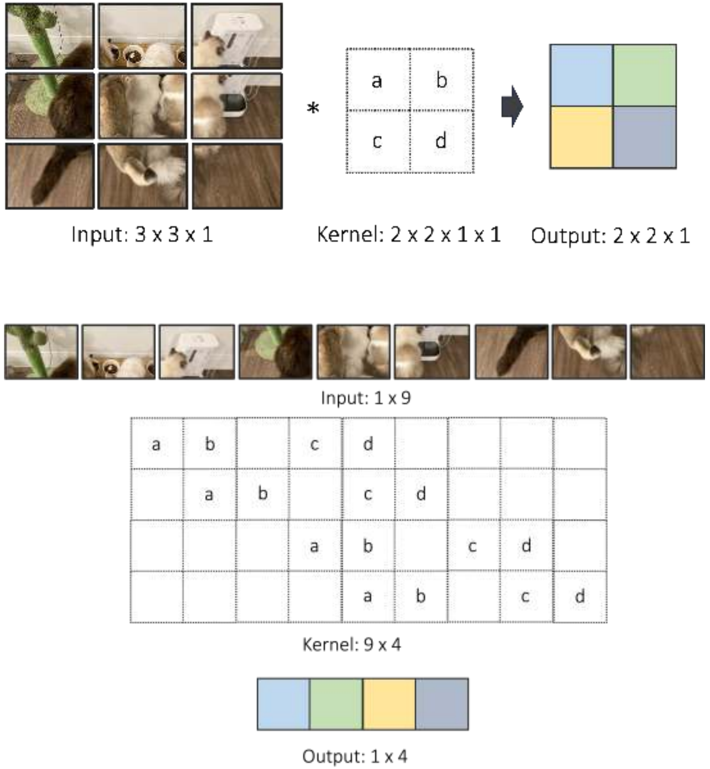
1. For any model, feed an image of 0s
2. Change the value of one location to NaN in input
3. See the difference in output



Convolution: What Do We Lose?

Computation Is Local

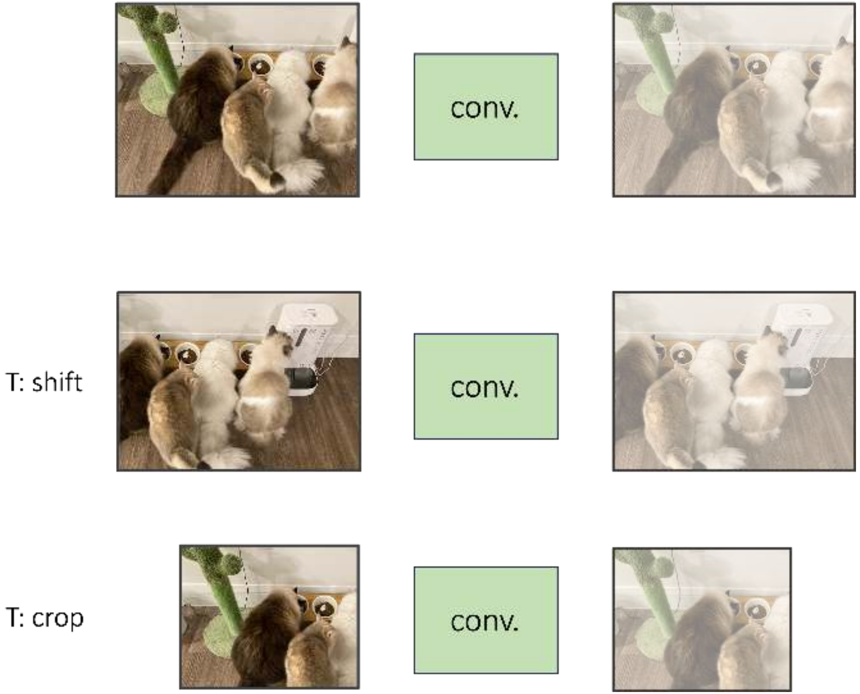
- Receptive field captures visual range of network
- Modern conv nets have large receptive fields
  - Much larger than the image size
  - For images, we don’t lose much



Convolution Preserves Spatial Properties of Images

Spatial Structure

- If input image is *shifted*, output is also shifted
- If input image is *cropped*, output is also cropped
- Convolution is anchored in each image coordinate

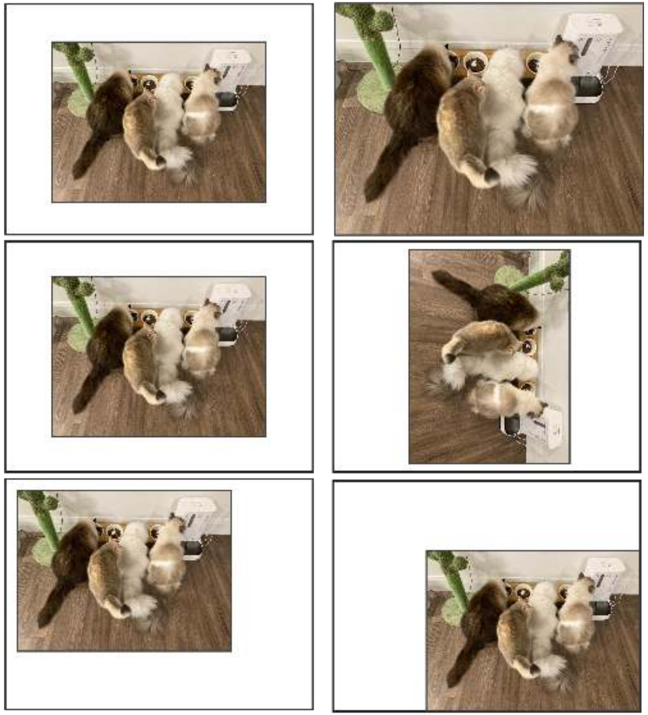


Properties of Images

Visual Patterns in Images

- Rotation invariant
- Scale invariant
- Shift invariant

Linear layers cannot capture these invariances!



Scaled / rotated / shifted cats are still cats

Convolution and Signal Processing

Convolutions Are Image Filters

Example: **Box Filter**

$$\omega = \begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix} \in \mathbb{R}^{1\times 1\times 3\times 3}$$

convolving over an image $x \in \mathbb{R}^{1\times H\times W}$

$$\begin{aligned} y_{i,j,k} &= b_i + \sum_{l=1}^{C_1} \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} x_{l,j+m,k+n} \cdot \omega_{i,l,m,n} \\ &= 0 + \frac{1}{9} \sum_{m=0}^{2} \sum_{n=0}^{2} x_{0,j+m,k+n} = \mathrm{Avg}_{3\times 3}(x_{m,n}) \end{aligned}$$

Convolution and Signal Processing

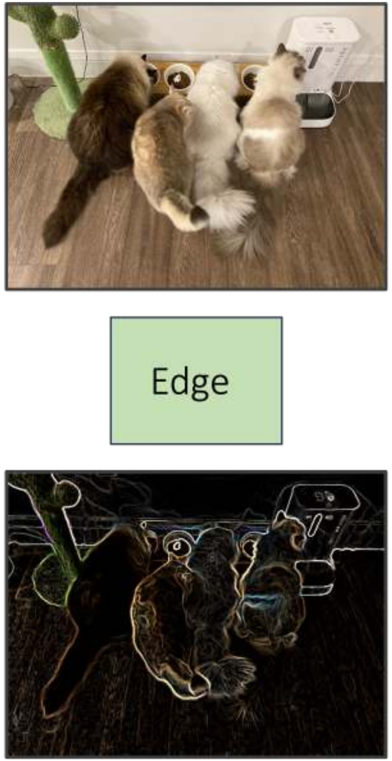
Convolutions Are Image Filters

Example: **Edge Filter**

$$\omega = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{1\times 1\times 3\times 3}$$

convolving over an image $x \in \mathbb{R}^{1\times H\times W}$

$$\begin{aligned} y_{i,j,k} &= b_i + \sum_{l=1}^{C_1} \sum_{m=0}^{h-1} \sum_{n=0}^{w-1} x_{l,j+m,k+n} \cdot \omega_{i,l,m,n} \\ &= \sum_{m=0}^{2} x_{0,j+m,k+2} - x_{0,j+m,k} \approx \nabla_x x_{i,j,k} \end{aligned}$$

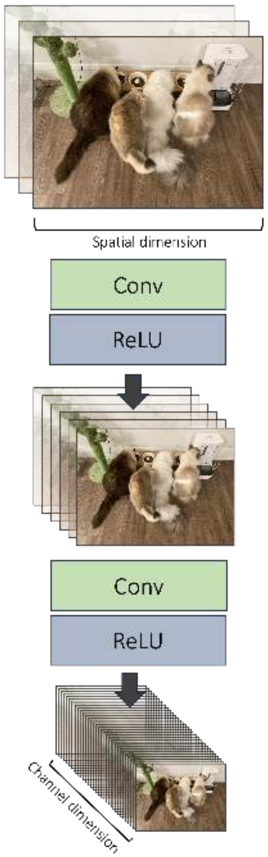


Interpretations of Convolution

1. Convolution is an **efficient** image processor

2. Convolution preserves **properties** of images

3. Convolution is various **image filters**



Convolutions - TL;DR

Convolutional layers are fast and memory-efficient

Convolution preserves structures of images