香港科技大学(广州)
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY (GUANGZHOU)
信息枢纽
INFORMATION HUB
人工智能学域
ARTIFICIAL INTELLIGENCE

# Visual Recognition I & II

Course AIAA 4220

Week3- Lecture 5 & 6

**Changhao Chen**

Assistant Professor

HKUST (GZ)

# Information for Project 1

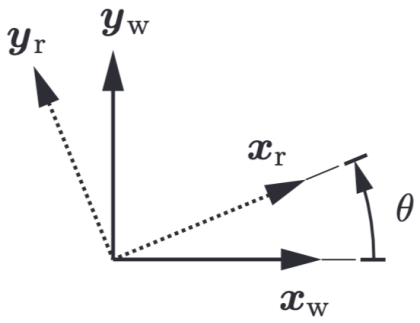
You will need to start forming groups (1-3 ppl) now.

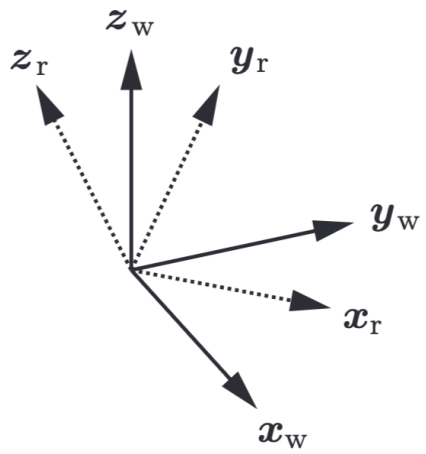
**My Office Hour: E1 613 Wed afternoon 16:30-17:30**

Our TA will let you know how to use canvas to form a team, and submit your presentation ppt (**ddl: Oct 14 2025**).

- Project 1: Paper Presentation (20%)
  - You will select 1 paper from our list of Embodied AI research to present. This should show that you understand the paper, and you could present it to students who do not know the paper well. **You have 10 minutes to present and 5 minutes to handle at least one question (from TA/peers/teacher)**
    - P1 presentation on Oct 15
  - Can be done in groups of 1-3 people. **Score multiplier: 1.1x for solo, 1.05x for two people and 1.0x for three people group.**
  - Project 1 is worth 20% credit.
    - You will be graded based on Understanding & Content (40%), Presentation & Communication (30%), Critical Analysis & Discussion (30%), by the teacher, TAs, and peers.

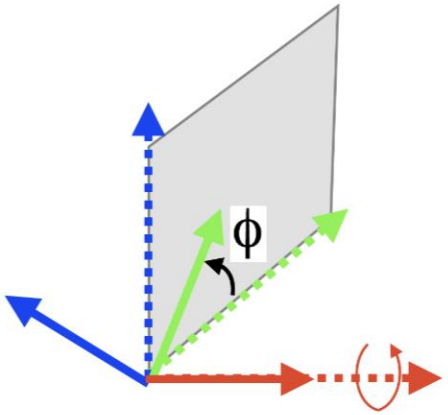
# Recap: Euler Angles and Rotation Matrix



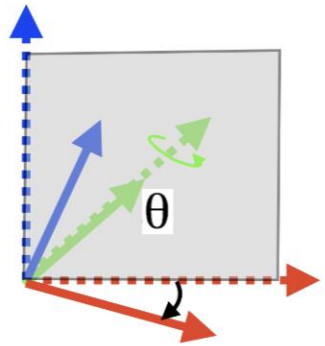

$$\boldsymbol{R}_{\mathrm{r}}^{\mathrm{w}} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin\theta & \cos(\theta) \end{bmatrix}$$

Rotation around the **x** axis



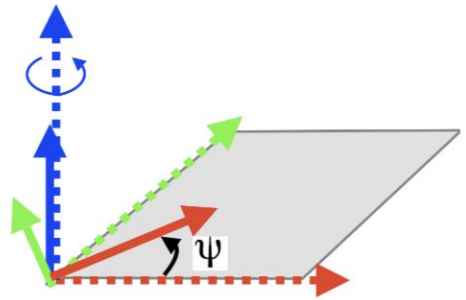

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

Rotation around the **y** axis




$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Rotation around the **z** axis




$$R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Euler's Theorem implies that only three parameters (Euler angles) are needed to represent a 3D rotation.

A rotation matrix is a special matrix used to rotate vectors in a Euclidean space (like 2D or 3D) around the origin (the point (0,0)) by a given angle.

$$\boldsymbol{R}_{\mathrm{r}}^{\mathrm{w}} = \boldsymbol{R}_x(\phi_1)\,\boldsymbol{R}_y(\theta_1)\,\boldsymbol{R}_x(\psi_1)$$

# Recap: Camera Pose

Defining Pose: Position and Orientation

The pose of a body frame {r} relative to a world frame {w} is completely described by:

Position: $\mathbf{t}_r^w$ - The translation from the origin of {w} to the origin of {r}.

Orientation: $\mathbf{R}_r^w$ - The rotation from {r} to {w}.

The pair $(\boldsymbol{R}_r^w, \boldsymbol{t}_r^w)$ is the complete geometric representation of frame {r} in {w}.

A pose has 6 degrees of freedom (3 for translation, 3 for rotation). We can combine these into a single, convenient transformation matrix.

$$\boldsymbol{T}_r^w = \begin{bmatrix} \boldsymbol{R}_r^w & \boldsymbol{t}_r^w \\ \mathbf{0}_d^\top & 1 \end{bmatrix}$$

# Recap: Rigid-body Transformations

We now generalize from pure rotation to full pose (rotation and translation).

Given:

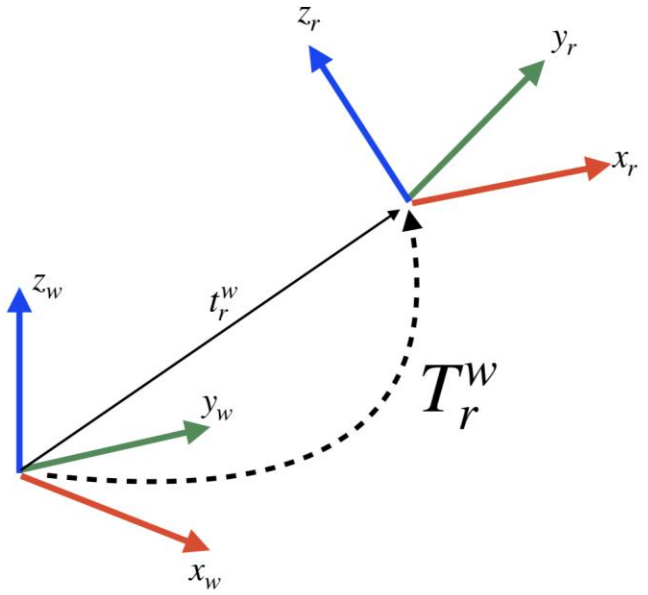
A point's coordinates in frame {r}: $\mathbf{p}^{r}$

The pose of {r} relative to {w}, represented by the transformation matrix: $\mathbf{T}_{r}^{w}$

Compute: The point's coordinates in frame {w}: $\mathbf{p}^{w}$

The transformation is given by:

$$\boldsymbol{p}^{\mathrm{w}} = \boldsymbol{R}_{\mathrm{r}}^{\mathrm{w}}\boldsymbol{p}^{\mathrm{r}} + \boldsymbol{t}_{\mathrm{r}}^{\mathrm{w}}$$

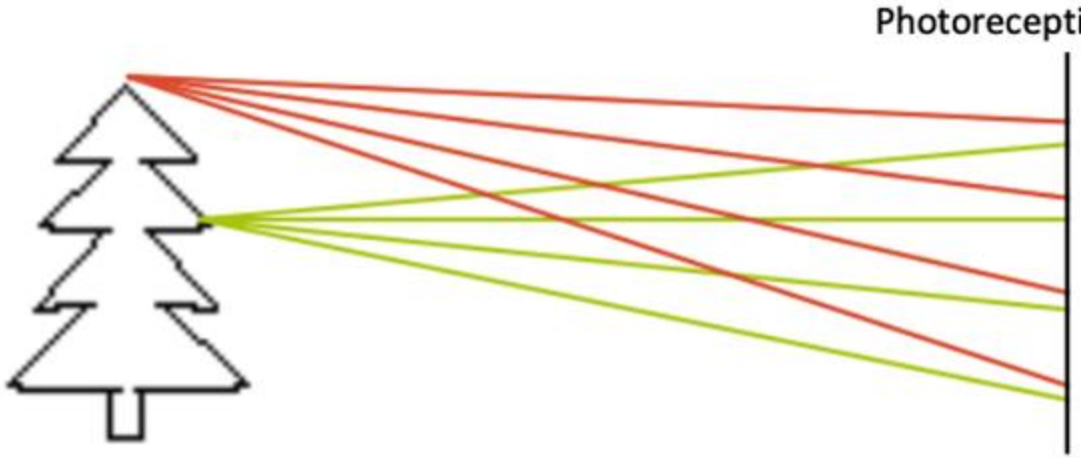


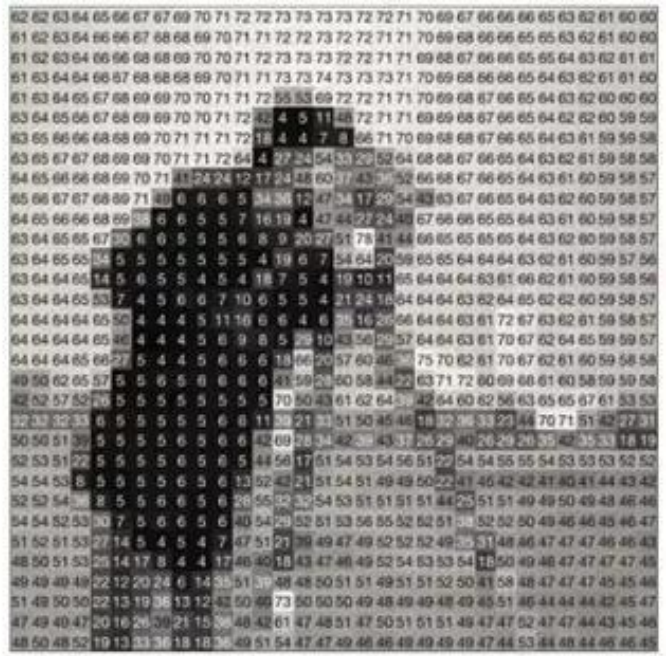
# Recap: Some Photos


# Recap: Some Photos


# Recap: Digital Cameras

- How to capture an image of the world
  - (Digital) Cameras -> capture light -> converted to digital image
  - Light is reflected by the object and scattered in all directions
  - if we simply add a photoreceptive surface, the captured image will be extremely blurred



# Recap: Perspective Projection

1) World to Camera Transformation (P_w → P_c)

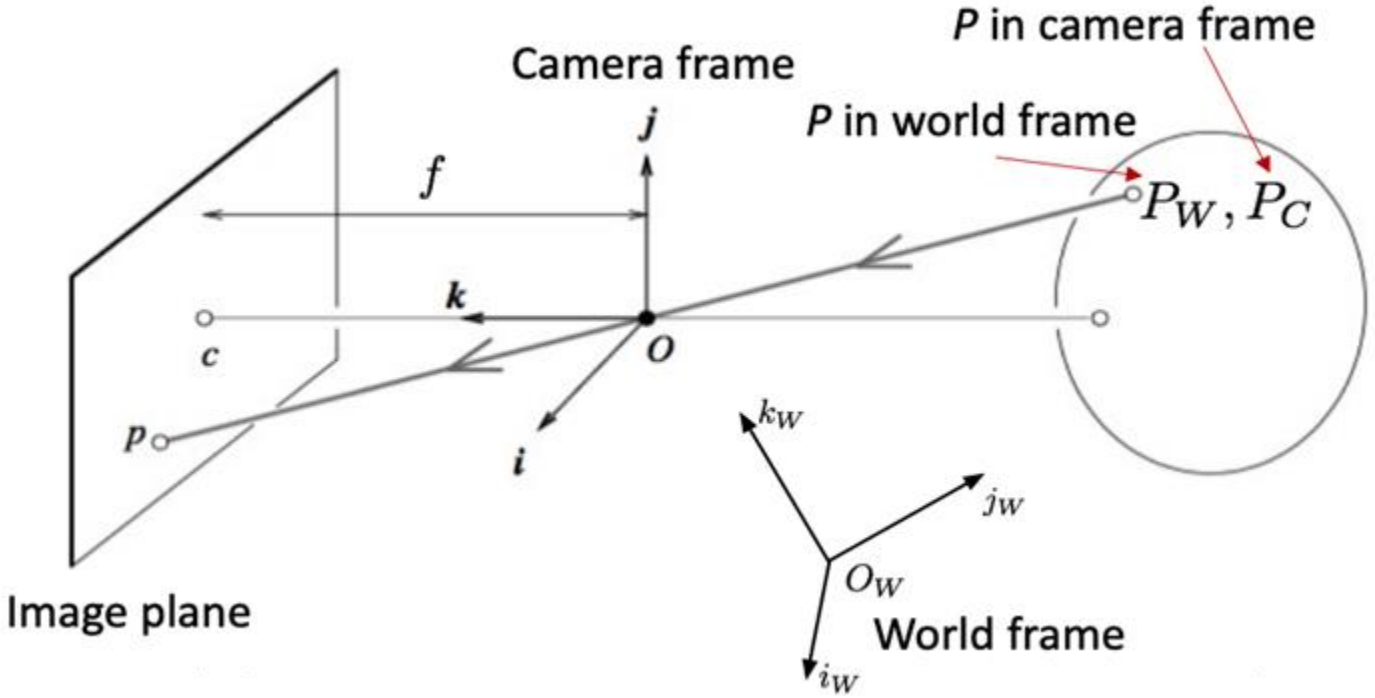
Transform the point from world coordinates to camera coordinates using the camera's pose (rotation R and translation t).

2) Perspective Projection (P_c → p)

Project the 3D point in the camera frame onto the normalized image plane using the pinhole model: (x, y) = (fX/Z, fY/Z).

3) Image to Pixel Transformation (p → (u, v))

Convert the metric image coordinates into discrete pixel coordinates using the camera's intrinsic calibration matrix K.




P_w and P_c represent the same physical point in 3D space, but their coordinate values differ because they are expressed in different reference frames.

# Recap: Homogeneous coordinates

- Collecting all results

$$p^h = [K \quad 0_{3\times1}]P_C^h = K[I_{3\times3} \quad 0_{3\times1}] \begin{bmatrix} R & t \\ 0_{1\times3} & 1 \end{bmatrix} P_W^h$$

- Hence

Projection matrix *M*

$$p^h = K[R \quad t]P_W^h$$

Intrinsic parameters  Extrinsic parameters

R: rotation
t: translation
Rt: extrinsic matrix

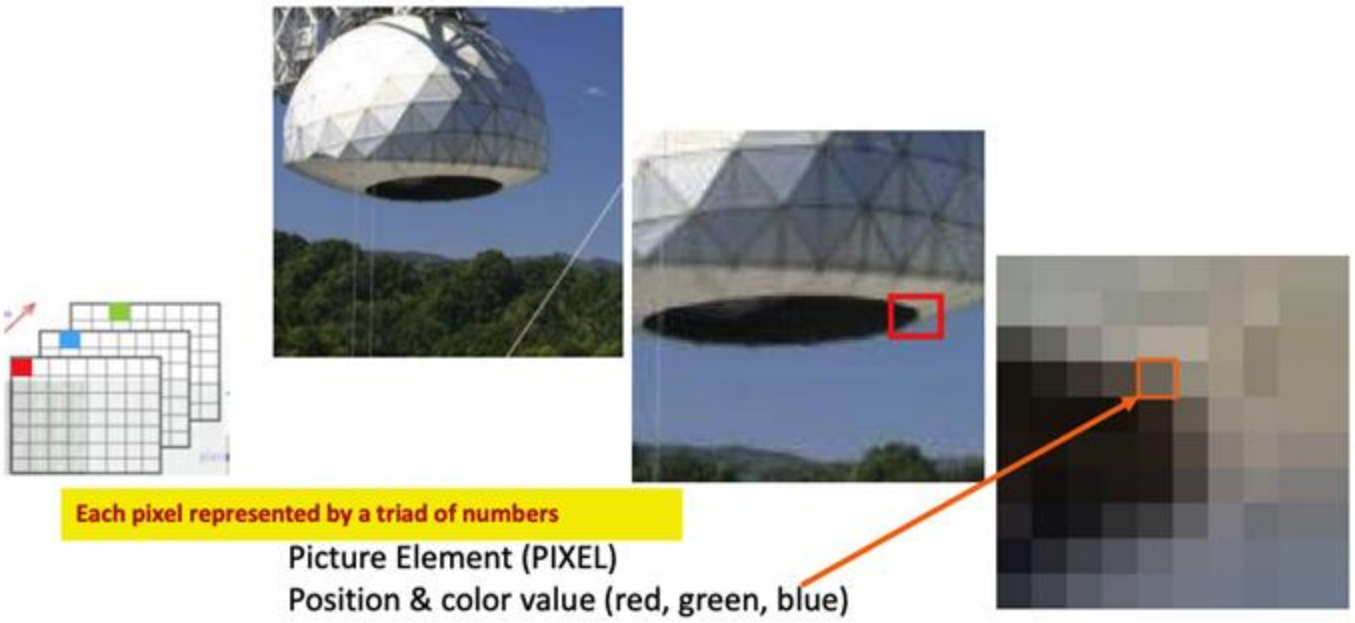
Degree of freedom:
K: 4
Rt: 6

# Image and Video Storage

## Digital image

A set of matrices of numbers

Color images: 3 Channels




What we see

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 3 | 2 | 5 | 4 | 7 | 6 | 9 | 8 |
| 3 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 2 | 1 | 0 | 3 | 2 | 5 | 4 | 7 | 6 |
| 5 | 2 | 3 | 0 | 1 | 2 | 3 | 4 | 5 |
| 4 | 3 | 2 | 1 | 0 | 3 | 2 | 5 | 4 |
| 7 | 4 | 5 | 2 | 3 | 0 | 1 | 2 | 3 |
| 6 | 5 | 4 | 3 | 2 | 1 | 0 | 3 | 2 |
| 9 | 6 | 7 | 4 | 5 | 2 | 3 | 0 | 1 |
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

What a computer sees

Source: S. Narasimhan

# Image and Video Storage

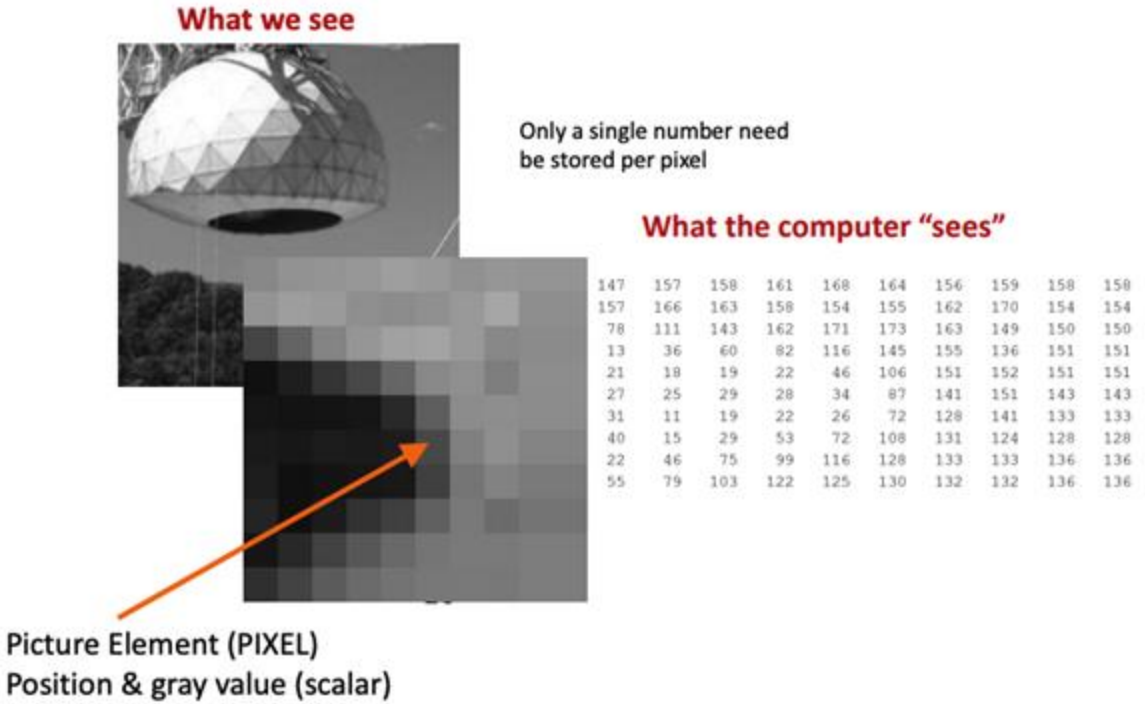
## Digital image - Grayscale

Grayscale: a single matrix of numbers

Each number represents the intensity of the image at a specific location in the image

Interpreted as the shade of grey at that location

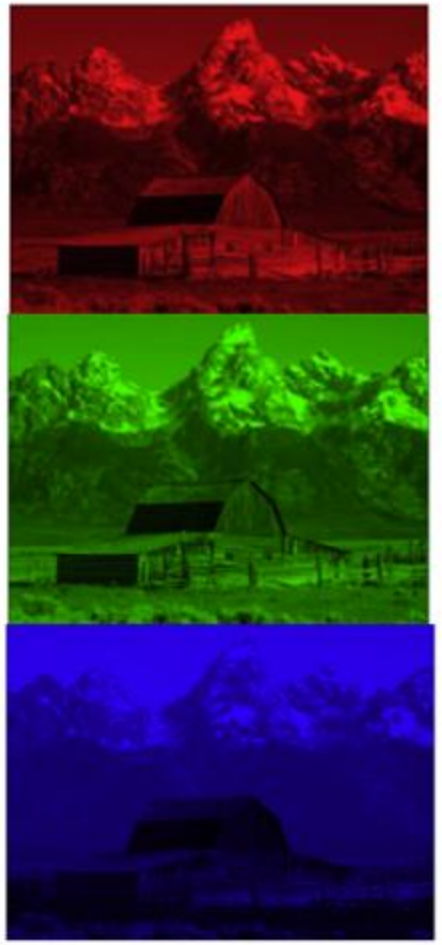
Grayscale images are typically derived from color images



# Image and Video Storage

## Color images - RGB colorspace

Example from wikipedia



RGB colorspace:
The three matrices represent R, G and B

# Image and Video Storage

What happens after camera capture?

The digital image is not stored in its raw form

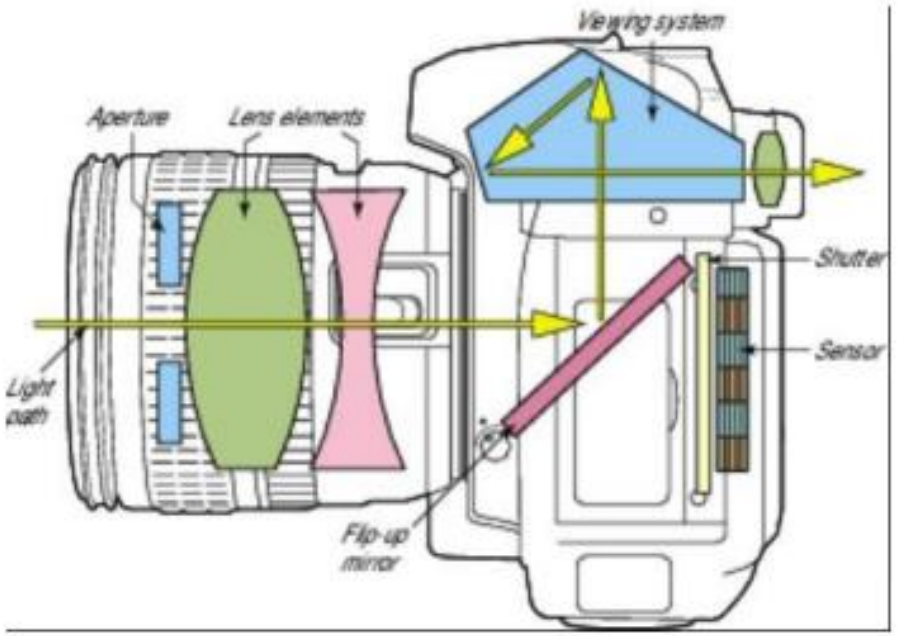



Image is processed in multiple stages before it is stored

- White balancing
- Demosaicing
  - Interpolation for colors
- Gamma correction
- And other processing steps

# Image and Video Storage

## White balancing

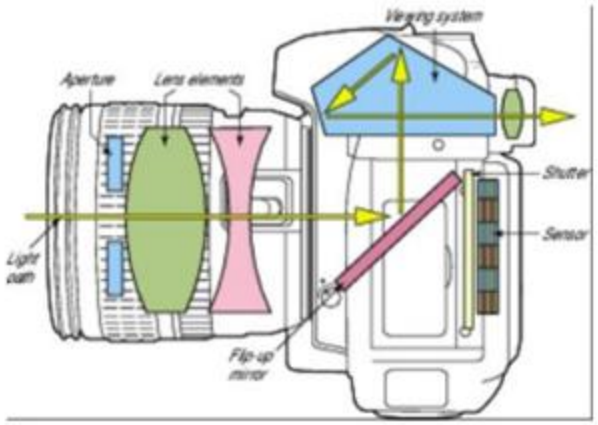
Camera automatically adjusts individual colors to give you a more “neutral” image

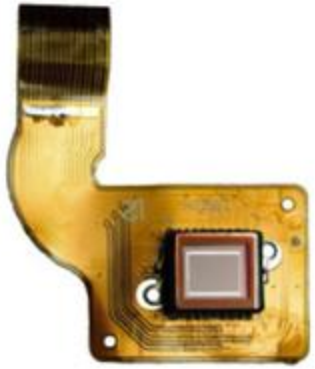
Closer to an image taken under white light

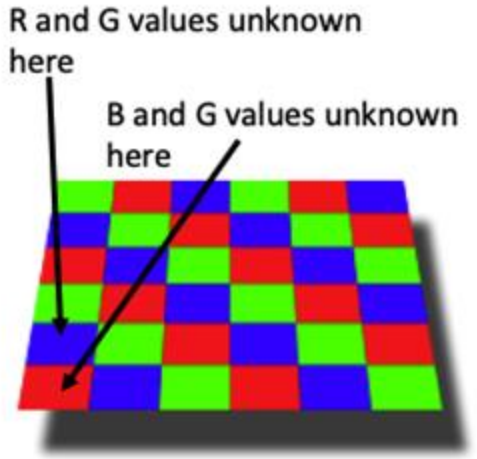
# Image and Video Storage

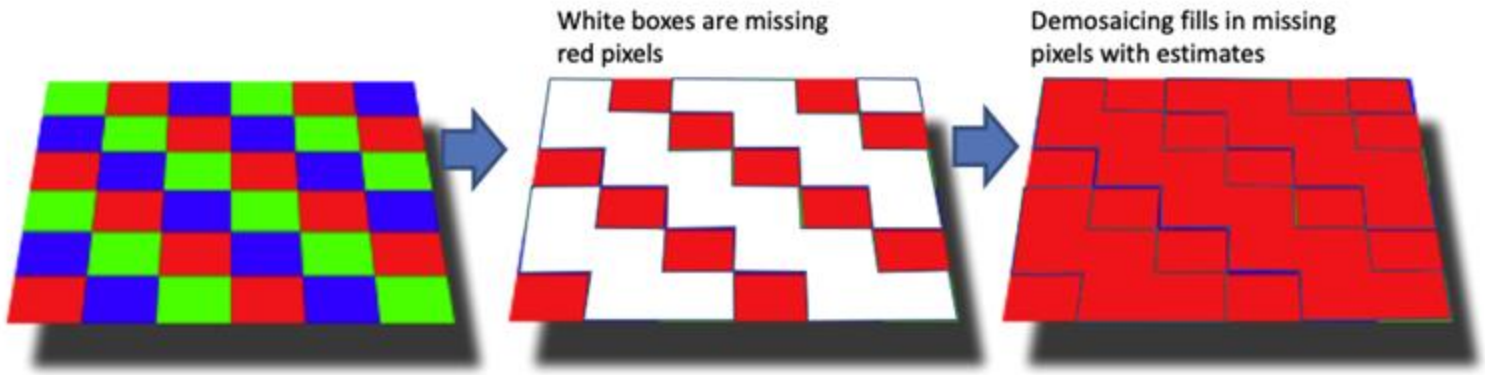
## Demosaicing: Reconstructing a Full-Color Image

- A digital camera with an NxM pixel sensor does not capture a full NxM image for each color (red, green, and blue).
- Each pixel on the sensor is covered by a filter that records only one of the three colors.
- For any given pixel, the value for only one color is known directly from the sensor; the values for the other two must be algorithmically estimated.





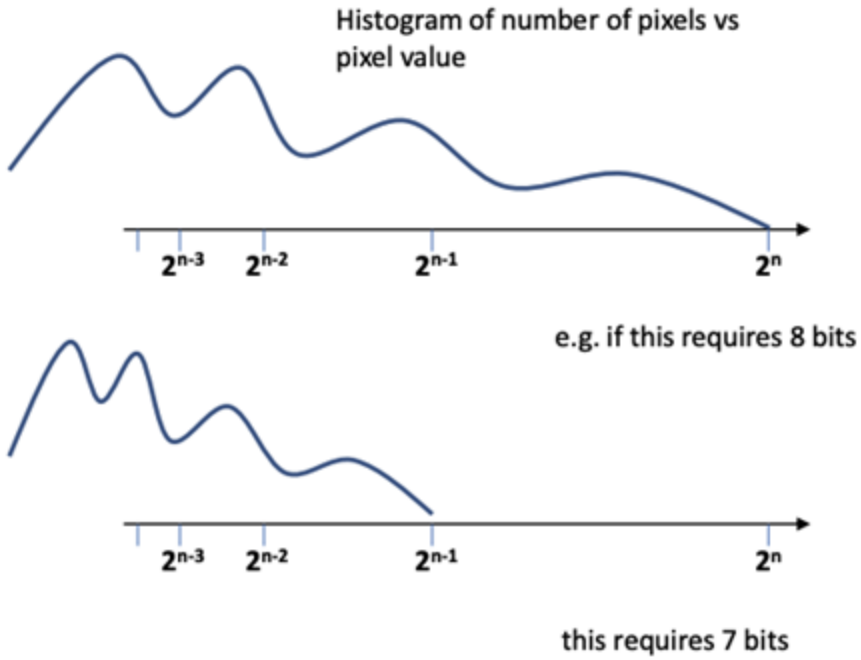



# Image and Video Storage

## Gamma correction

- Pixel values (of all colors) are compressed through a non-linearity
- This reduces the dynamic range of pixel values
- Can be stored using fewer bits



# Image and Video Storage

## Storage: The Exchangeable Image Format (EXIF)

A standardized format for images: Agreed upon by a number of standardization bodies

Specifies formats for images, sounds, and other tags used by digital image recorders (cameras, scanners, etc.)

Standard includes format specifications for

- JPEG images
- TIFF images (Tagged Image File Format)
- RIFF wav for audio files (Resource Interchange File Format)
- Appended metadata
- Image compression, JPEG, PNG…

from wikipedia

| Tag | Value |
|---|---|
| Manufacturer | CASIO |
| Model | QV-4000 |
| Orientation (rotation) | top-left [8 possible values[20]] |
| Software | Ver1.01 |
| Date and time | 2003:08:11 16:45:32 |
| YCbCr positioning | centered |
| Compression | JPEG compression |
| X resolution | 72.00 |
| Y resolution | 72.00 |
| Resolution unit | Inch |
| Exposure time | 1/659 s |
| F-number | f/4.0 |
| Exposure program | Normal program |
| Exif version | Exif version 2.1 |
| Date and time (original) | 2003:08:11 16:45:32 |
| Date and time (digitized) | 2003:08:11 16:45:32 |
| Components configuration | Y Cb Cr – |
| Compressed bits per pixel | 4.01 |
| Exposure bias | 0.0 |
| Max. aperture value | 2.00 |
| Metering mode | Pattern |
| Flash | Flash did not fire |
| Focal length | 20.1 mm |
| MakerNote | 432 bytes unknown data |
| FlashPix version | FlashPix version 1.0 |
| Color space | sRGB |
| Pixel X dimension | 2240 |
| Pixel Y dimension | 1680 |
| File source | DSC |
| Interoperability index | R98 |
| Interoperability version | (null) |

# Image and Video Storage

## Video cameras

- Take multiple shots automatically at a fixed rate per second
- The rate is decided based on persistence of vision
- Usually 30 photos/second (frame-per-second, FPS)
- Sound is recorded alongside and time synchronized with the photo frames

# Image and Video Storage

Video cameras

Codec: scheme to compress images for storage and distribution and
to decompress the compressed images for display

Why?
Raw HD video can take up to 400GB per hour..

Image: JPEG, PNG,..

Video: H.264, MPEG-4, …
Can be represented in 2 spatial and 1 temporal dimension
Coding exploits redundancies in spatial and temporal domains

# Image and Video Storage

Video cameras

Codec - Video coding

Spatial coding:

- Also called intra-coding
- Exploits redundancies within a frame (JPEG, etc.)

Temporal coding:

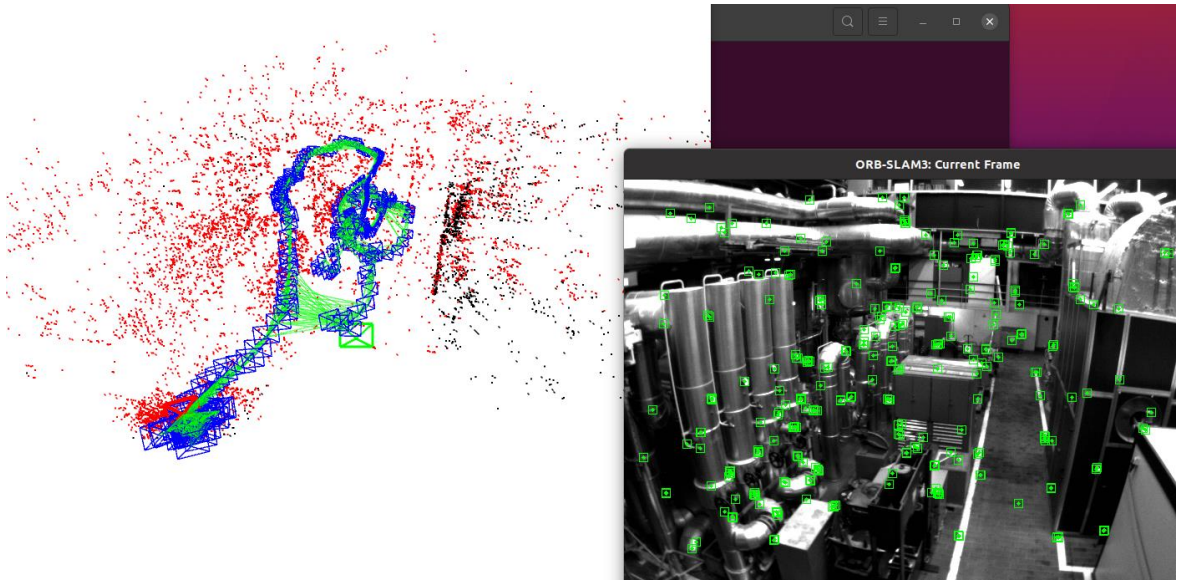
- Also called predictive coding
- Exploits redundancies across frames

3 types of frames:

- I-frame (Intra-coded), P-frame (Predicted) and B-frame (Bi-directionally predicted)
- I-frames are coded using a JPEG like scheme. Exploits spatial redundancies (as in image coding). The first frame of a video is an I-frame
- P-frames, B-frames: Only changes from the reference are stored.
- I-frames are inserted at regular intervals or inserted based on the motion in the video

These compressions are not lossless (hence there are artifacts). Sometimes our recognition model may be affected

# Visual Feature

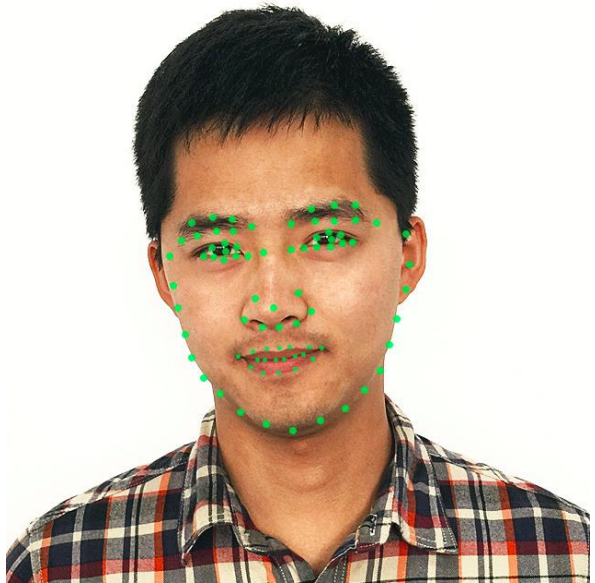



Features are key information in an image (such as corner points, edges, etc.).

Distinct, recognizable patterns

They are the foundation for:

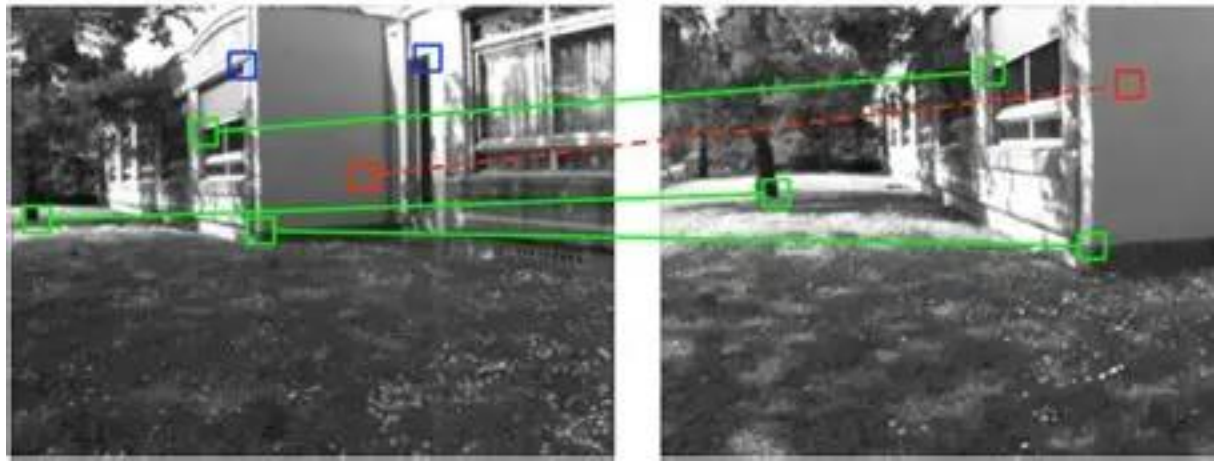
- Visual Odometry / SLAM: Estimating ego-motion.
- Object Recognition & Tracking: Following other cars, pedestrians.
- 3D Reconstruction: Building a map of the world.



# Visual Feature

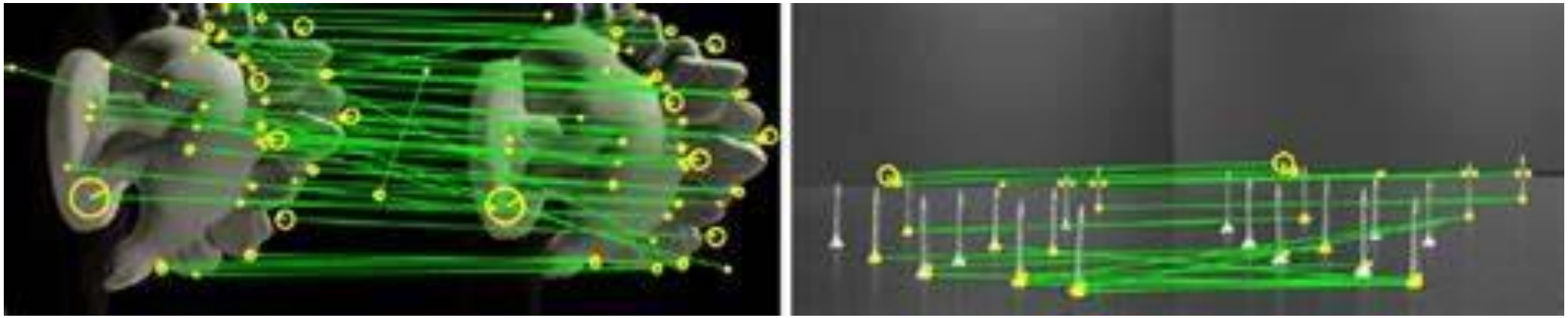
What Makes a Good Feature?

- Repeatability: Can be found again in a different image of the same scene.
- Distinctiveness: Carries enough information to be distinguished from others.
- Locality: Robust to occlusion and clutter.
- Efficiency: Can be computed quickly.



Challenges:
Scale changes, viewpoint changes, illumination changes (day/night), motion blur, repetitive textures (e.g., a brick wall), real-time computation requirements.

# Visual Feature

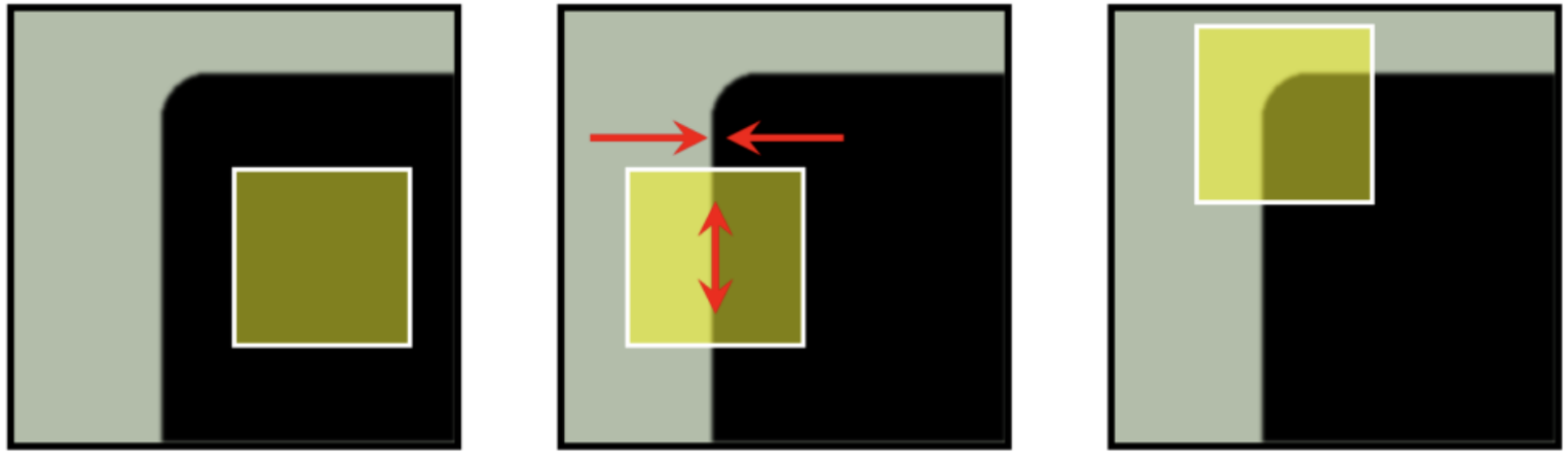


Keypoint (Location): The (x, y) coordinates of the feature.
Descriptor (Signature): A numerical vector that describes the visual appearance of the patch around the keypoint.

Main Tasks:

- Feature Detection;
- Feature Tracking;
- Feature Matching.

# Feature Detection



Flat: no change in all directions

Edge: no change along the edge direction

Corner: significant change in all directions with small shift

# Visual Feature


Object Detection before convolutional neural network (CNN) model

- Feature engineering based on heuristics
- Local descriptors
- Global representation

# Visual Feature

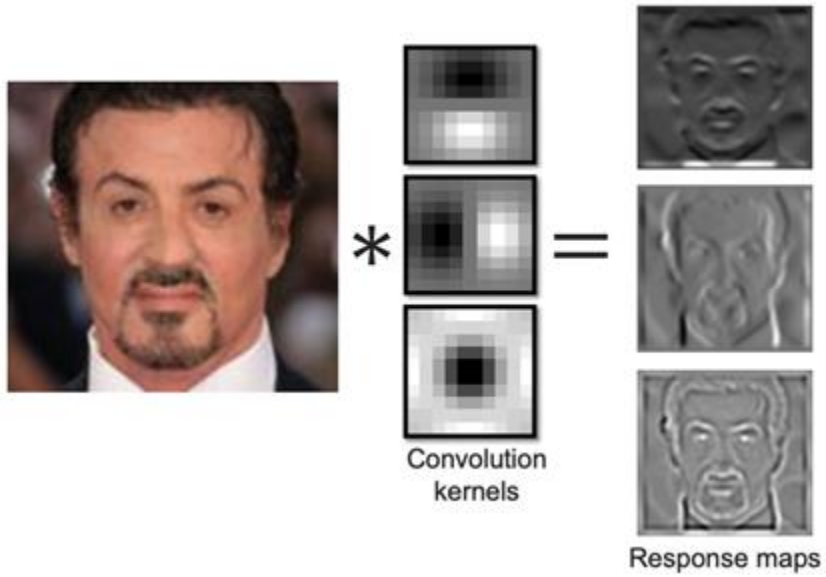
## Local descriptors

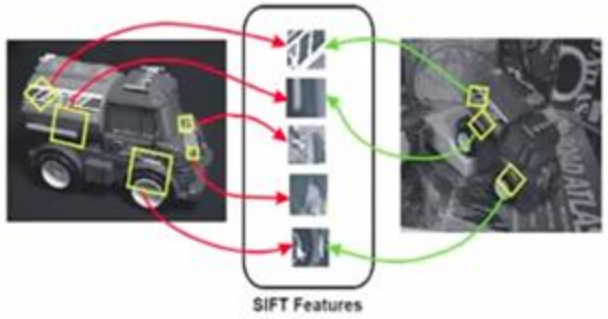
1) histogram of oriented gradients (HOG),

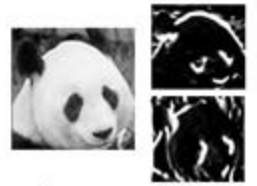
2) Scale-Invariant Feature Transform (SIFT), etc.

3) Haar-like features

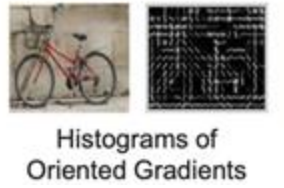
- Hand-crafted convolution filters
- Pixel “templates” sliding across image
- For face detection



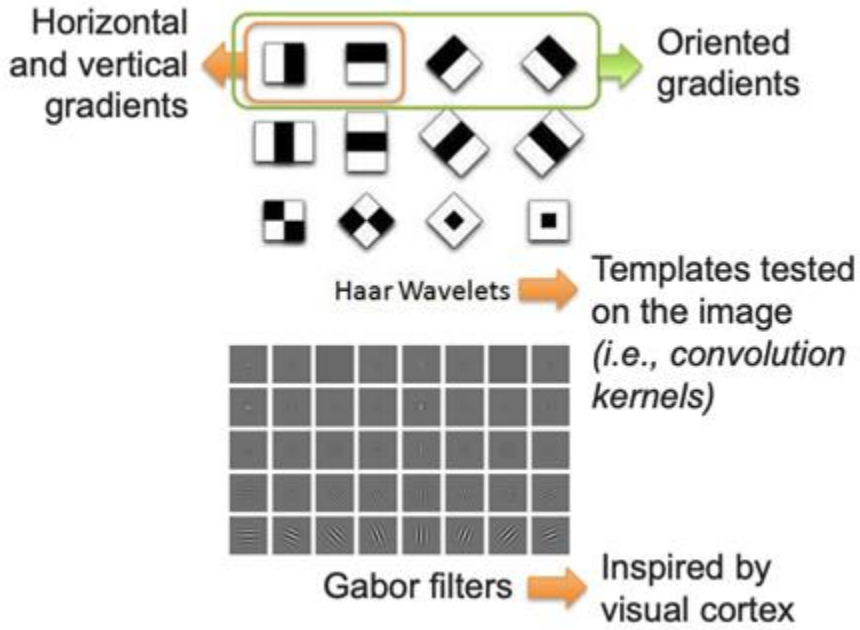







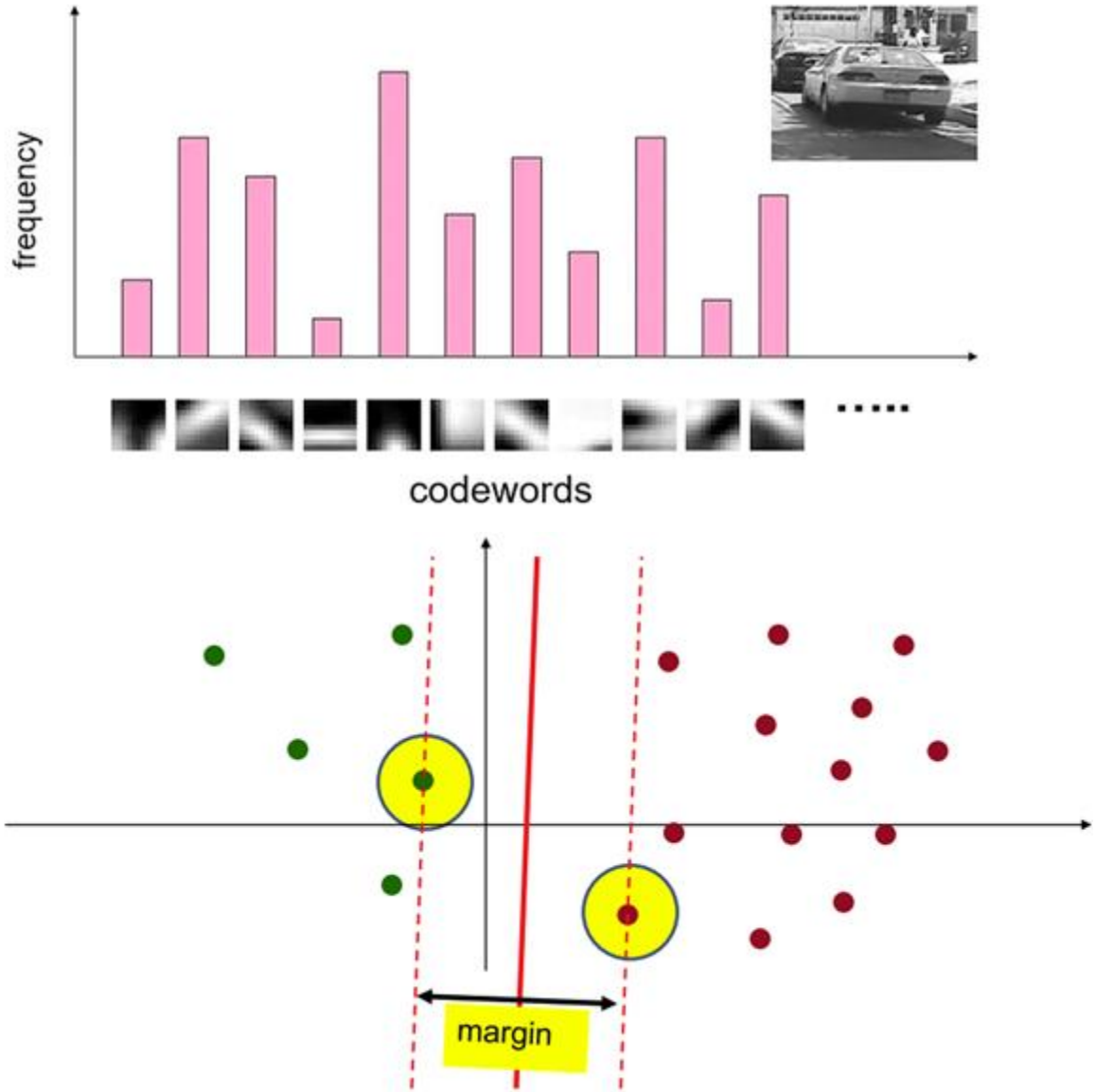




# Visual Feature

## Global representation

- Bag-of-visual-words
- Local descriptors
- Clustering: K-means
- Got fixed length vector representation for any size image/video
- Then apply ML methods like support vector machines (SVM)



# Visual Recognition

## Four Basic Neural Networks:

### 1. Multilayer Perceptron (MLP)

The simplest type of artificial neural network. Composed of multiple fully connected (dense) layers where every neuron in one layer is connected to every neuron in the next layer.

### 2. Convolutional Neural Network (CNN or ConvNet)

The dominant architecture for visual tasks. Designed to process data with a grid-like topology (e.g., pixels).

### 3. Recurrent Neural Network (RNN)

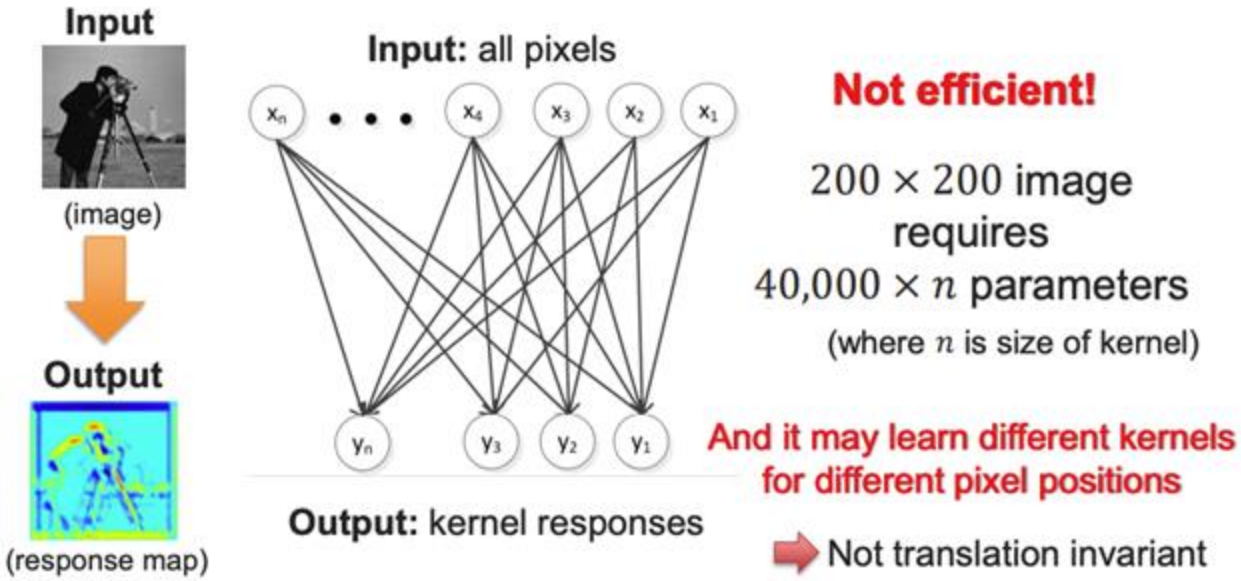
Networks with internal loops designed for sequential data. They maintain a "hidden state" that acts as a memory of previous inputs in the sequence.

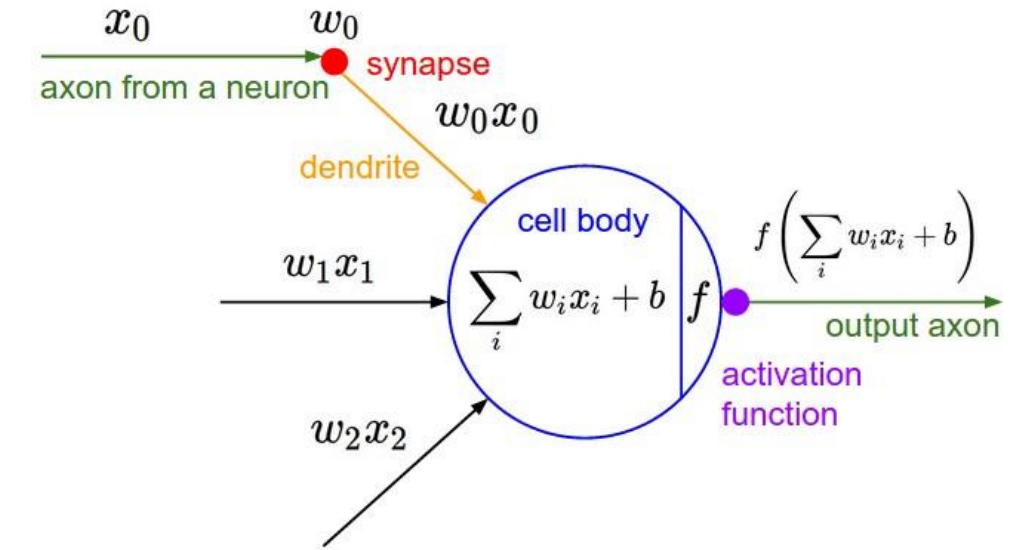
### 4. Self-Attention Mechanism

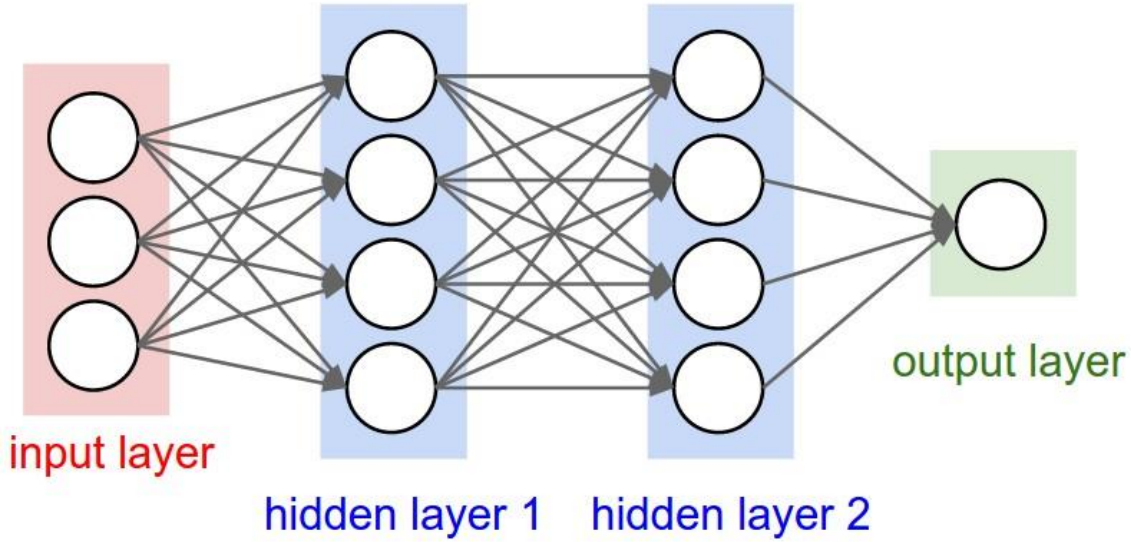
A mechanism that allows a model to weigh the importance of different parts of the input sequence when computing a representation for a specific part. The foundation of the Transformer architecture.

# Multi-Layer Perceptron (MLP)

Fully-connected network



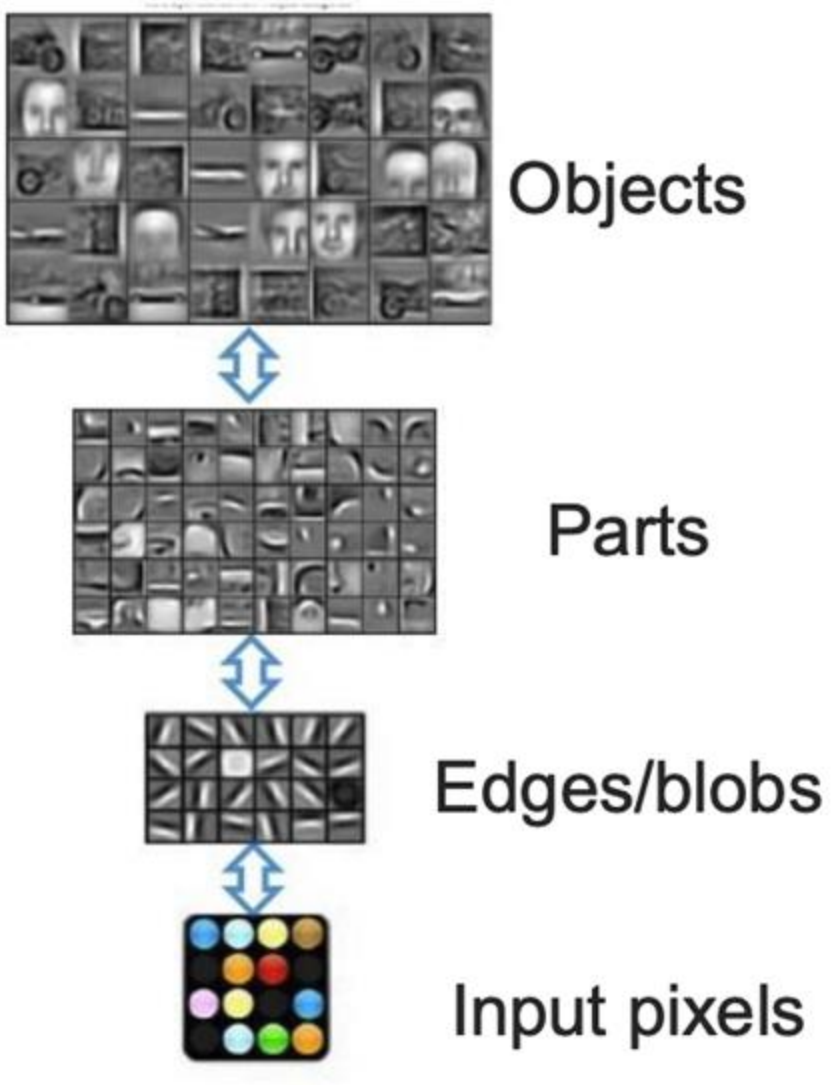
# Convolutional Neural Network

Target:

building more abstract, hierarchical visual representations

Key advantages:

- Inspired from visual cortex (human eye)
- Encourages visual abstraction
- Exploits translation invariance
- Kernels/templates are learned
- Fewer parameters than MLP



# Convolutional Neural Network

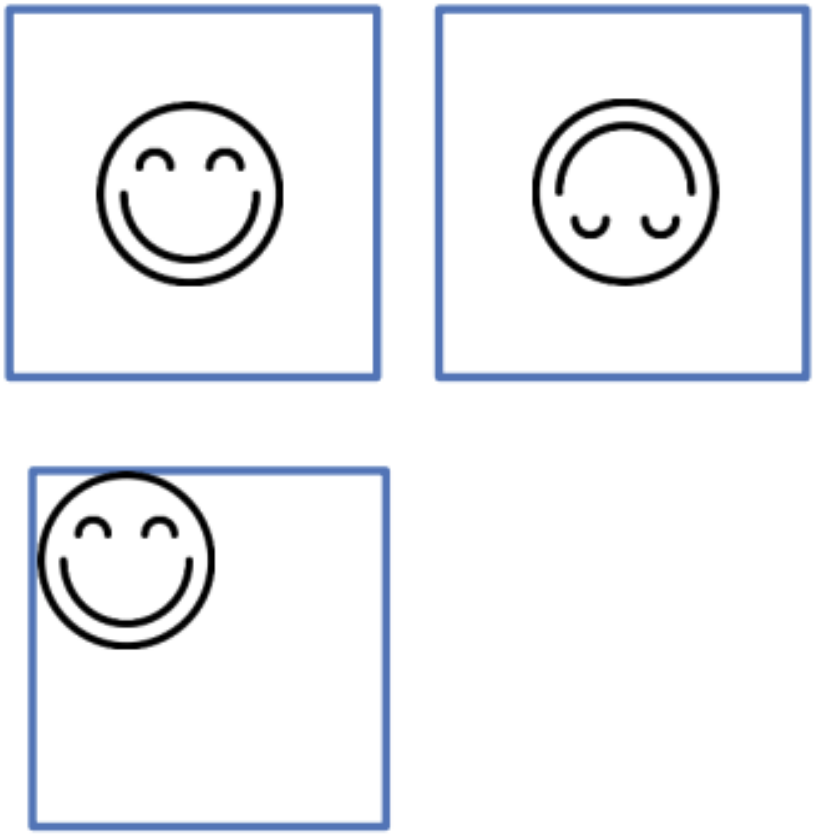
## Translation Invariance

2 data points - which one is up?

MLP can easily learn this task

What if the face is slightly translated?

MLP may fail to recognize this; Not translation invariant



- CNN has translation invariance properties
- Kernel-based, sliding-window computation

# Convolutional Neural Network

## Translation Invariance

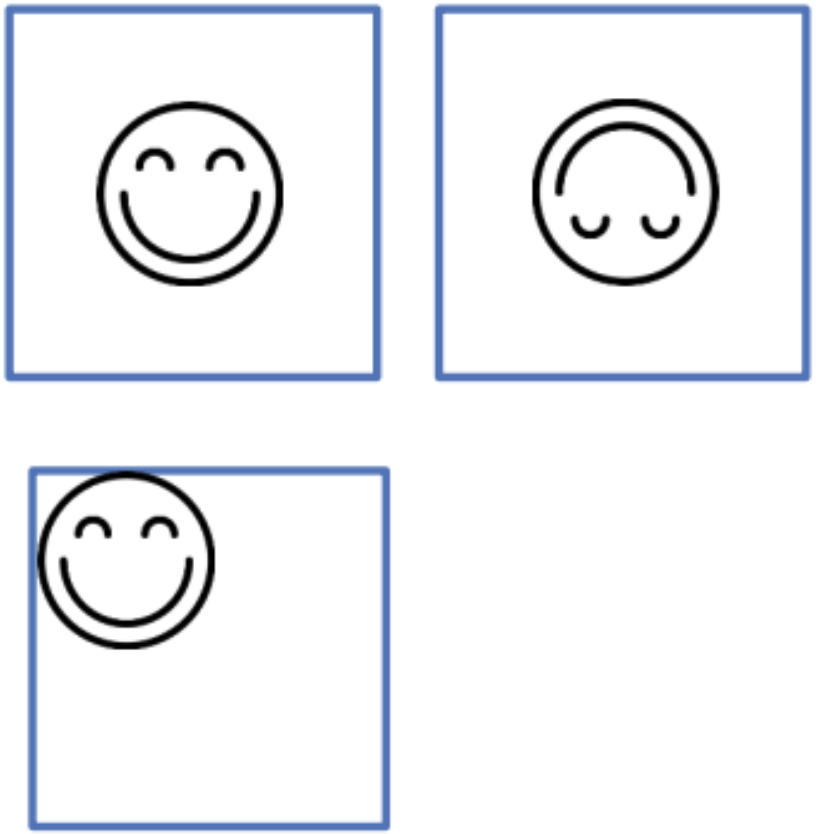
2 data points - which one is up?

MLP can easily learn this task

What if the face is slightly translated?

MLP may fail to recognize this; Not translation invariant

- CNN has translation invariance properties
- Kernel-based, sliding-window computation

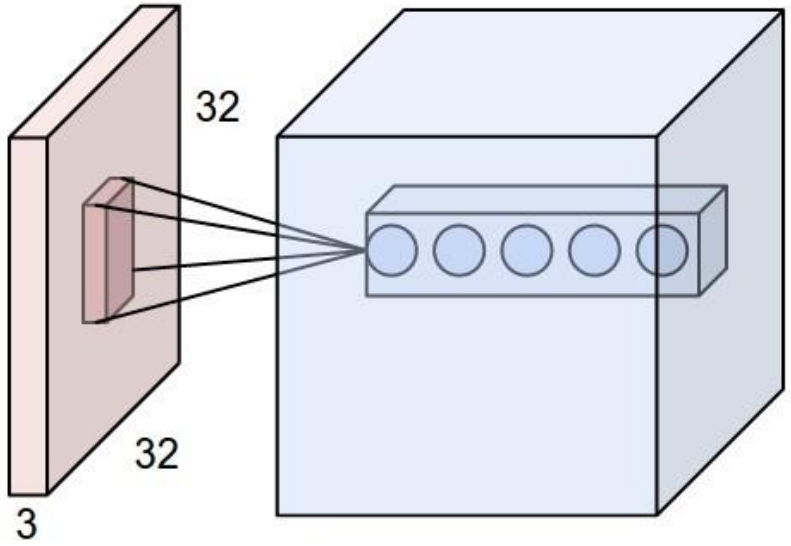


# Convolutional Neural Network

A basic mathematical operation
(that given two functions returns a function)

$$(f * g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

**Summary**. To summarize, the Conv Layer:

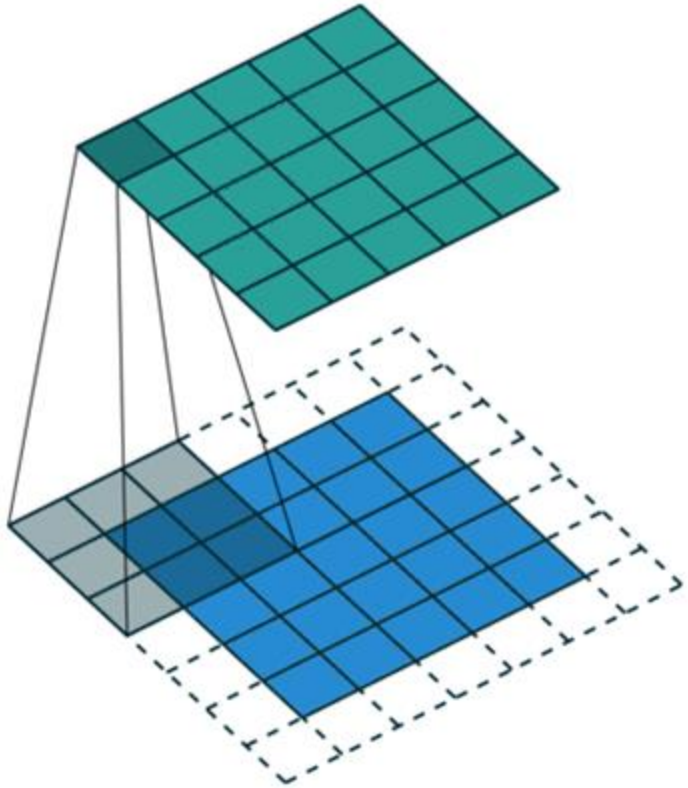



- Accepts a volume of size $\mathbf{W_1 \times H_1 \times D_1}$
- Requires four hyperparameters:
  - Number of filters $\mathbf{K}$,
  - their spatial extent $\mathbf{F}$,
  - the stride $\mathbf{S}$,
  - the amount of zero padding $\mathbf{P}$.
- Produces a volume of size $\mathbf{W_2 \times H_2 \times D_2}$ where:
  - $\mathbf{W_2 = (W_1 - F + 2P)/S + 1}$
  - $\mathbf{H_2 = (H_1 - F + 2P)/S + 1}$ (i.e. width and height are computed equally by symmetry)
  - $\mathbf{D_2 = K}$
- With parameter sharing, it introduces $\mathbf{F \cdot F \cdot D_1}$ weights per filter, for a total of $\mathbf{(F \cdot F \cdot D_1) \cdot K}$ weights and $\mathbf{K}$ biases.
- In the output volume, the $\mathbf{d}$-th depth slice (of size $\mathbf{W_2 \times H_2}$) is the result of performing a valid convolution of the $\mathbf{d}$-th filter over the input volume with a stride of $\mathbf{S}$, and then offset by $\mathbf{d}$-th bias.

# Convolutional Neural Network

$$(f * g)[n] \stackrel{\text{def}}{=} \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

- 1-dimensional CNN:
  - We have a signal of length N
  - Kernel of length K
  - Output will be length N − K + 1
- Example
  - Input size (3), kernel/filter size (2), output size (2)
  - $f$ = [1,2,1] , $g$ = [1, −1] , $f * g$ = [a, b]?



The convolution operation (f ∗ g) is performed by sliding the kernel across the input, computing the dot product at each position.

First Position: [1, 2] • [1, -1] = (1×1) + (2×-1) = 1 - 2 = -1

Second Position: [2, 1] • [1, -1] = (2×1) + (1×-1) = 2 - 1 = 1

The output of the convolution $f * g$ is [-1, 1].

# Convolutional Neural Network

- If we want output to be different sizes, we can **add padding** to the signal:
  - Just add 0s at the beginning and end
  - Example
    - $f$ = [1,2,1], $f'$ = [0,1,2,1] , $g$ = [1, −1] , $f' * g$ = [-1,1-2, 2-1,] = [-1,-1, 1]

- Or
- We can perform **strided (aka, dilated) convolution**: the filter jumps over pixels or samples
  - We have a signal of length N
  - Kernel of length K
  - Output will be length (N − K)/stride + 1
  - Example with stride 2
  - Input of size (6), kernel (2), output (3)
  - $f$ = [0,0,1,2,1,0] , $g$ = [1, −1] , $f * g$ = [a, b, c]?

The kernel moves across the input in steps of 2, computing the dot product at each valid position.

Position 1 (Start at index 0): [0, 0] • [1, -1] = (0×1) + (0×-1) = 0

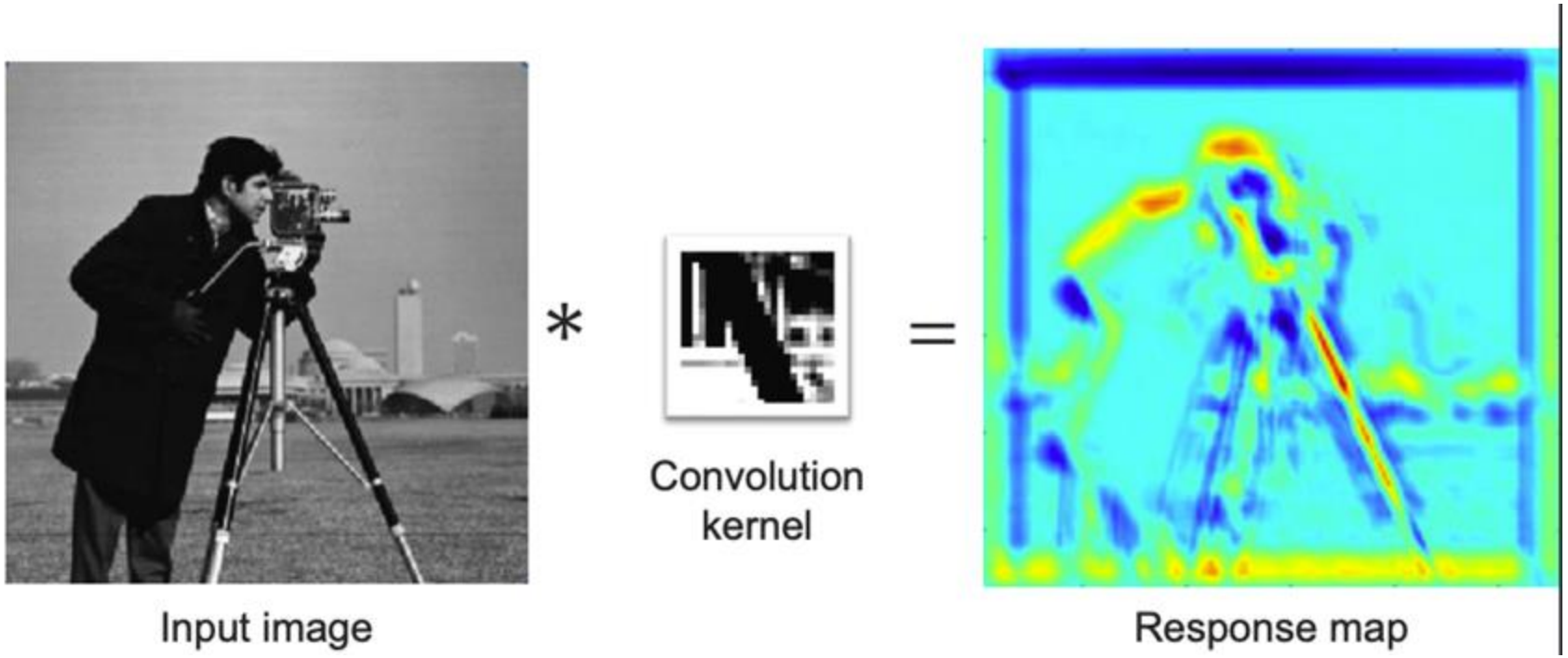
Position 2 (Start at index 2): [1, 2] • [1, -1] = (1×1) + (2×-1) = 1 - 2 = -1

Position 3 (Start at index 4): [1, 0] • [1, -1] = (1×1) + (0×-1) = 1

Result:
The output of the strided convolution $f * g$ is [0, -1, 1].

# Convolutional Neural Network

Convolution In 2D - Example



# Convolutional Neural Network

- Suppose we have input of (7 x 7 x 1) image
- Two convolutional kernels of size (3 x 3), stride=2

- Each convolutional kernel/filter output size is ?
  - Use this to compute output size: (N − K)/stride + 1
  - N: input dimension of one side
  - K: kernel size of one side

    - Two kernel output size (3 x 3 x 2)

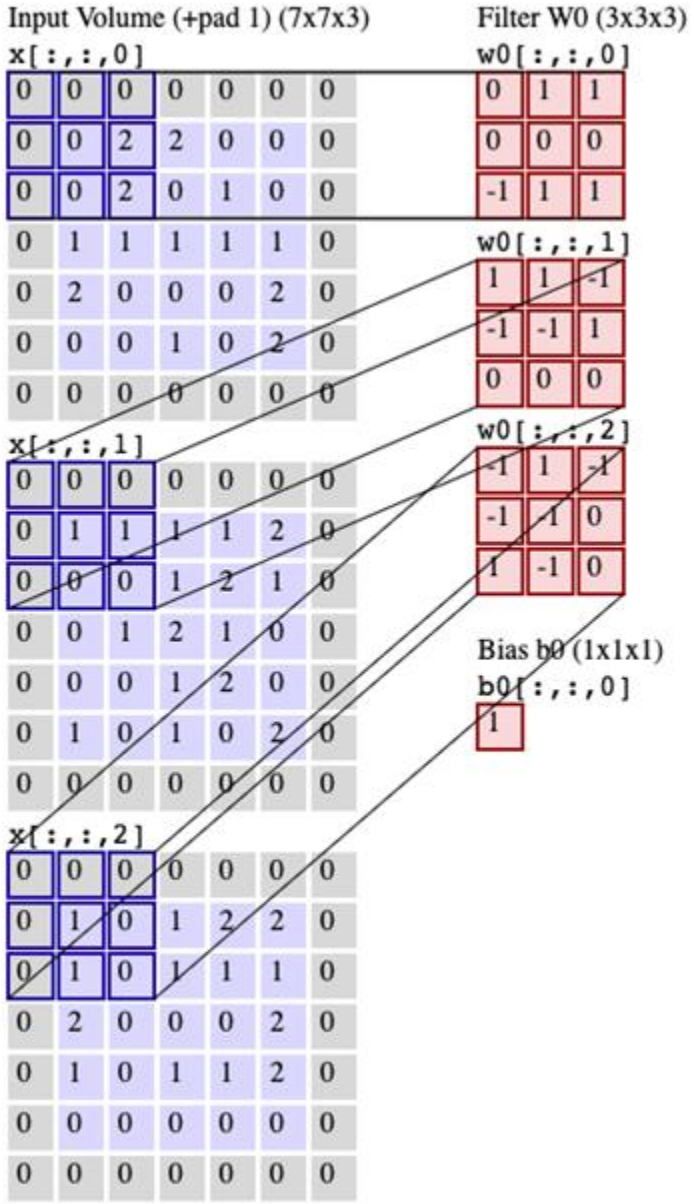
- (3 x 3)

# Convolutional Neural Network

- Adding the channel dimension
  - Suppose we have input of (7 x 7 x 3) image
  - Two convolutional kernels of size (3 x 3 x 3), stride=2
    - Last dimension is the channel dimension
  - Each convolutional kernel output size is (3 x 3)
    - Summed across Height x Width x Channel (all the dimension of the kernel)
  - Two kernel output size (3 x 3 x 2)
    - The output size is the same no matter the input channel number!

# Convolutional Neural Network

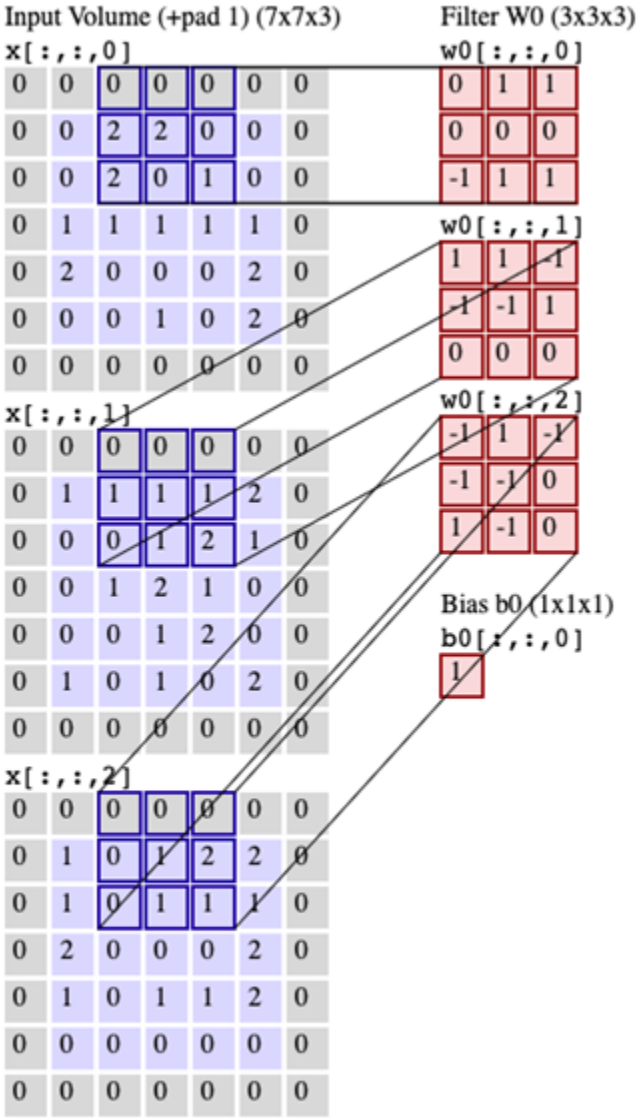
- Let's compute an example
  - Input of size (7 x 7 x 3)
  - One convolutional filter (3 x 3 x 3)
    - Stride = 2
    - Bias (1 x 1 x 1)
  - Output size (3 x 3)
    - O[0, 0] = ?
    - O[0, 0] = (2) + (-1+1) + (-1-1) +1 =1



# Convolutional Neural Network

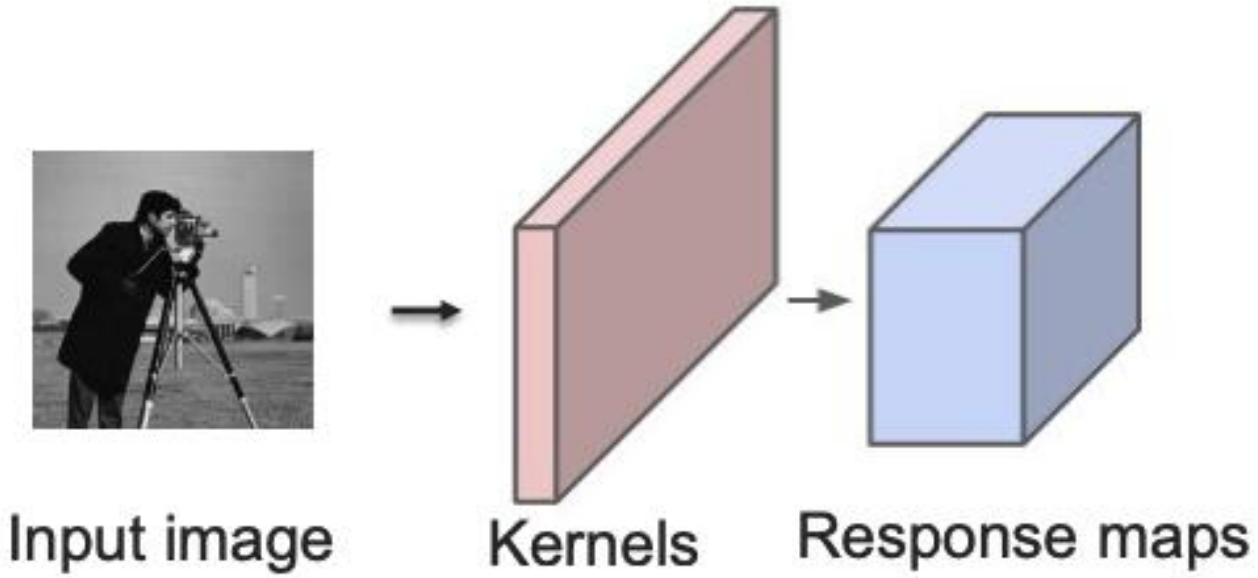
- Let's compute an example
  - Input of size (7 x 7 x 3)
  - One convolutional filter (3 x 3 x 3)
    - Stride = 2
    - Bias (1 x 1 x 1)
  - Output size (3 x 3)
    - O[0, 1] = ?

    - O[0, 1] = (-2+1) + (-1-1+1) + (-1-1)+1=-3



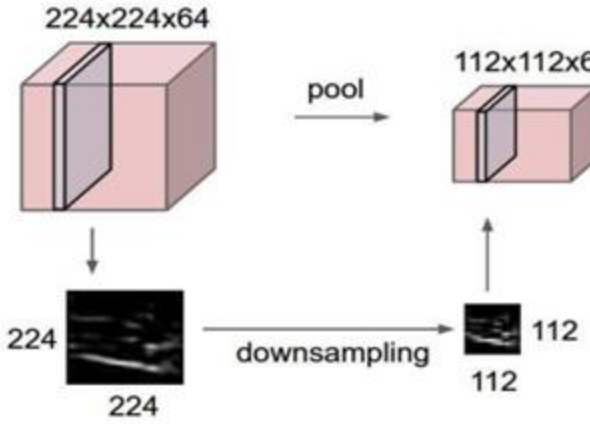
# Convolutional Neural Network

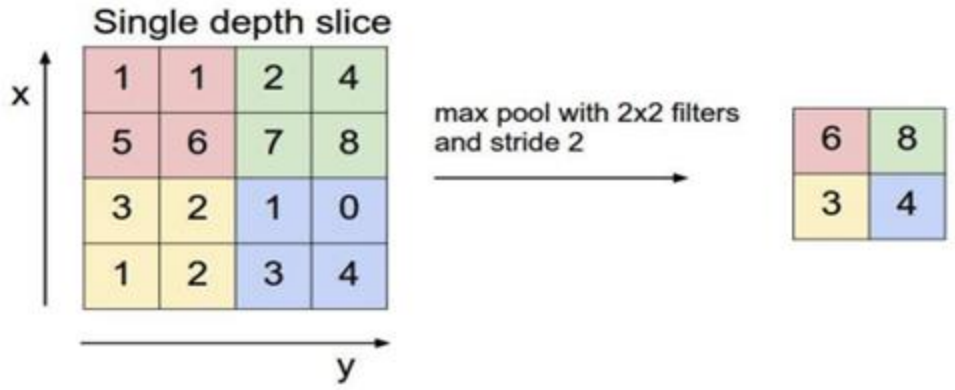
- Can expand this to 2D (or even 3D!)
  - Just need to make sure to link the right pixel with the right weight
- Can expand to multi-channel 2D
  - e.g., for RGB images
- Can expand to multiple kernels/filters
  - Output is not a single image anymore, but a tensor (a 3D matrix)

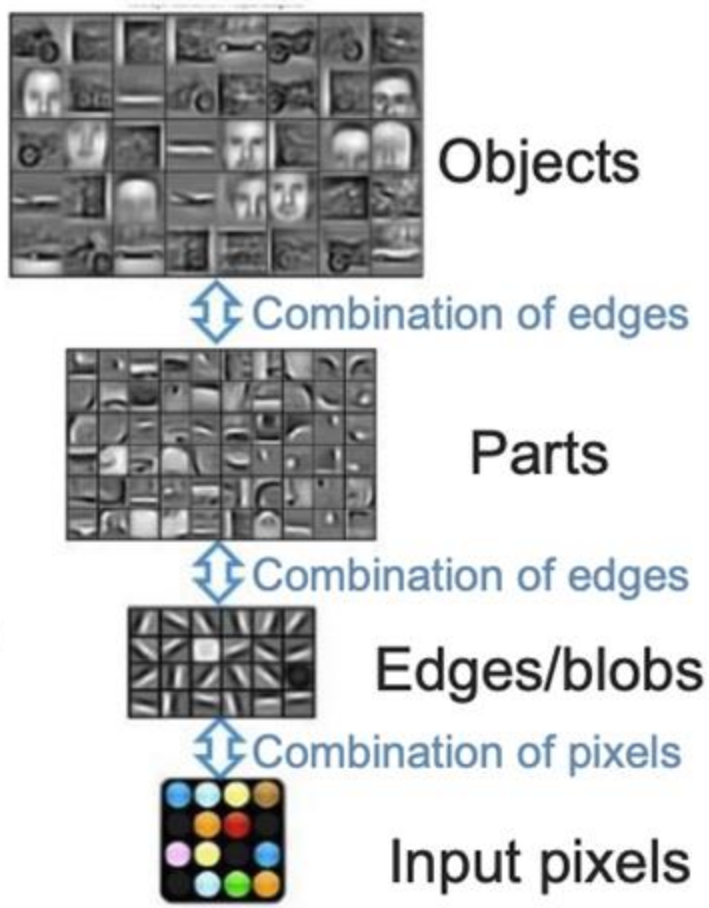


# Convolutional Neural Network

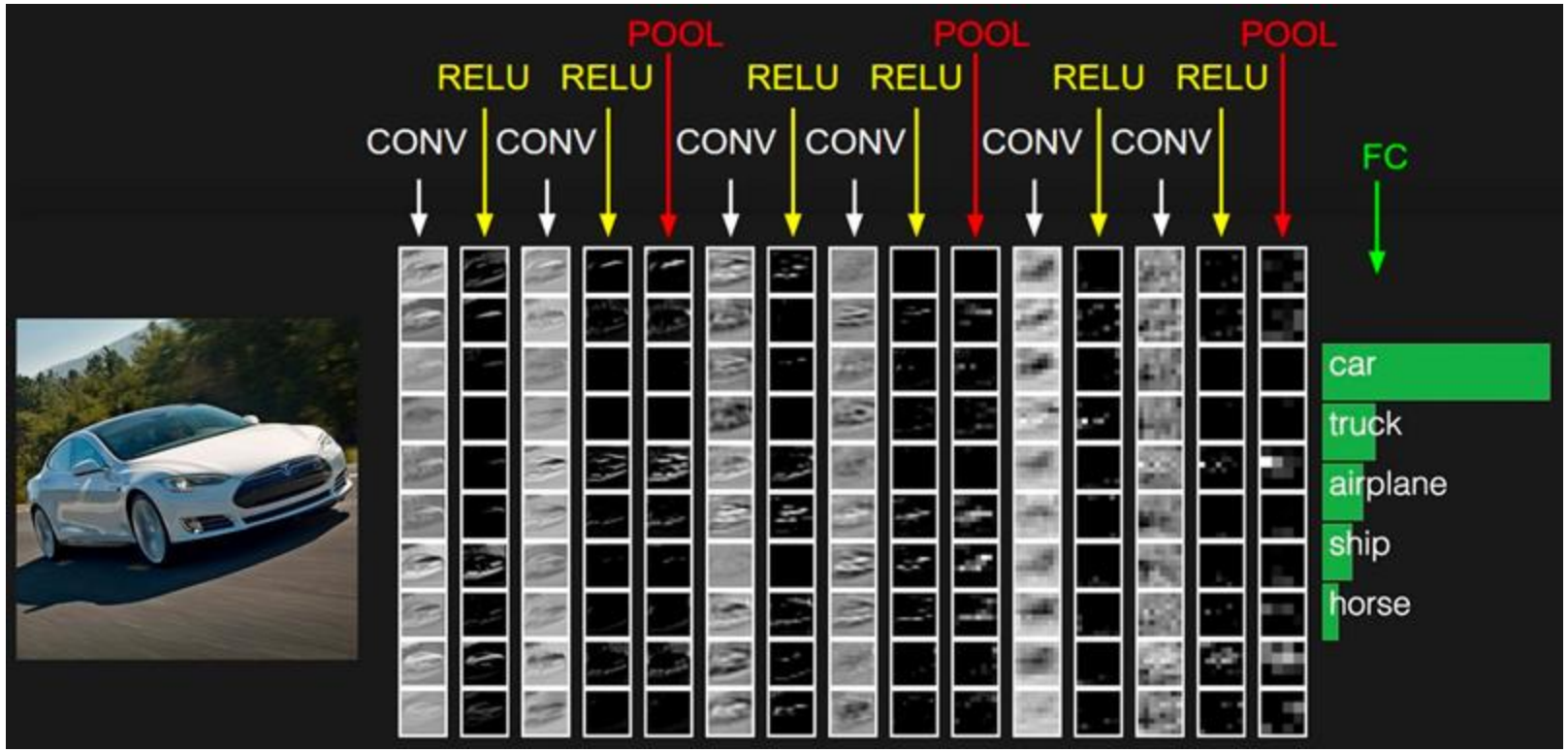
- Multiple convolutional layers
  - Allows the network to learn combinations of sub-parts, to increase complexity
- But how to encourage abstraction and summarization?
  - Pooling layer!
- Response map subsampling:
  - Allows summarization of the responses



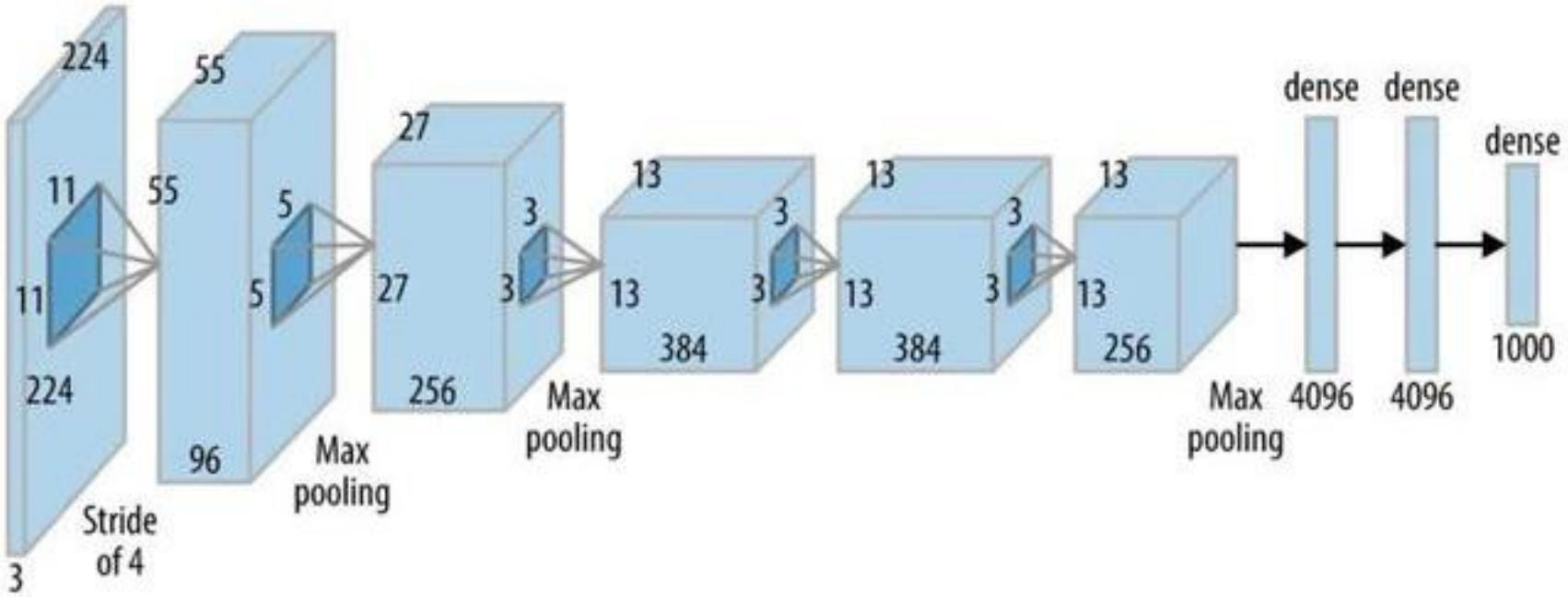



# Typical CNN Architectures




- Start with a convolutional layer follow by non-linear activation and pooling
- Repeat this several times
- Ends with a fully connected (MLP) layer

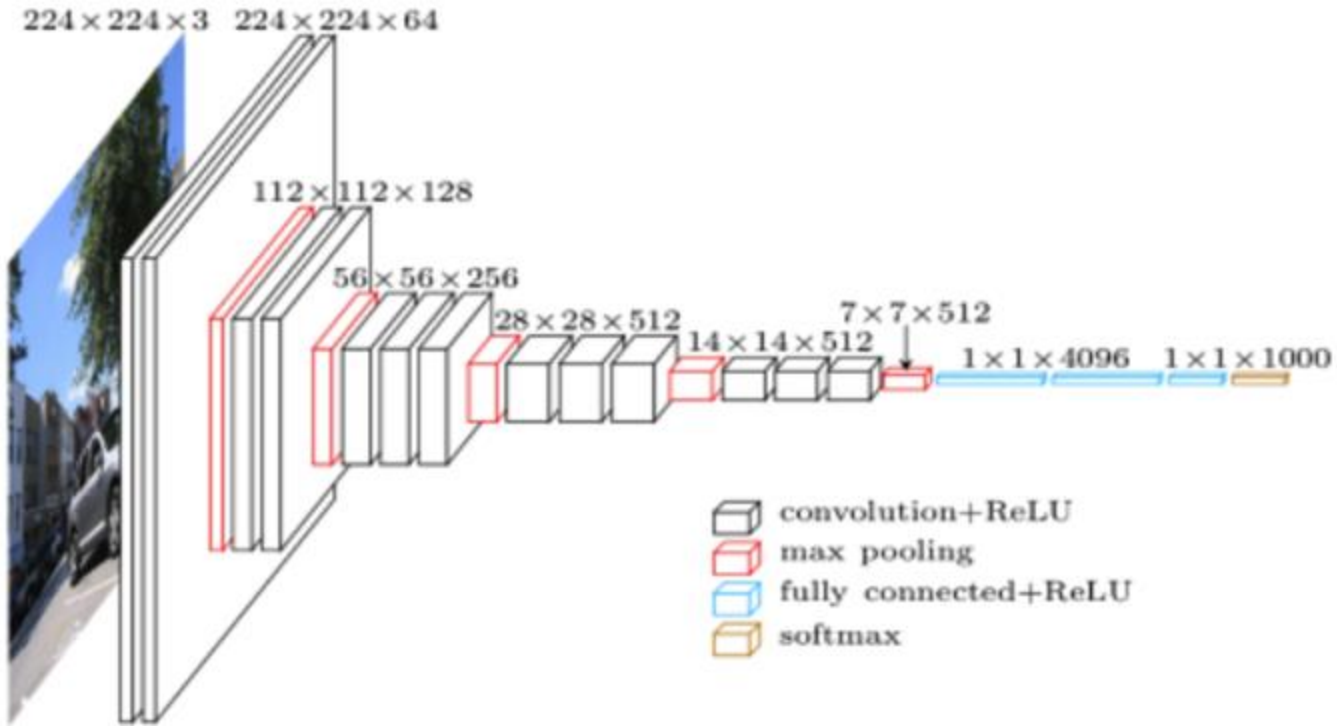
# Typical CNN Architectures




AlexNet

- Deep Architecture: It was one of the first deep CNNs (8 learned layers: 5 convolutional, 3 fully-connected) to be successfully trained on a large, complex dataset (ImageNet with 1.2M images).
- ReLU Nonlinearity: Used the Rectified Linear Unit (ReLU) activation function instead of Tanh/Sigmoid. This drastically accelerated training (~6x faster) by mitigating the vanishing gradient problem.
- GPU Acceleration: Pioneered the use of GPUs (NVIDIA GTX 580) for training large neural networks, making deep learning feasible.
- Overlapping Max Pooling: Used pooling with a stride smaller than the kernel size, improving feature richness and helping to reduce overfitting.

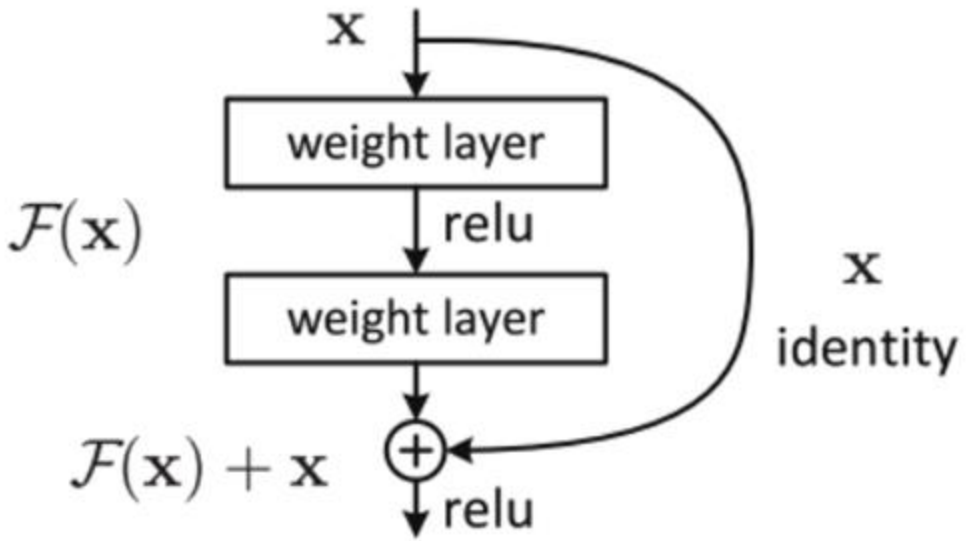
# Typical CNN Architectures

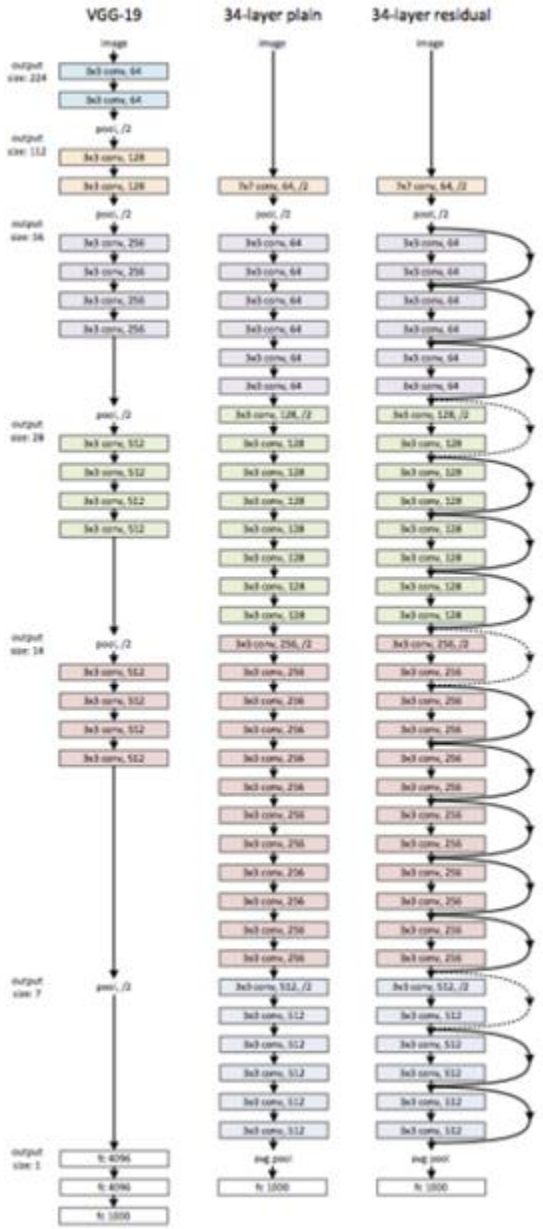



- VGGNet model
  - Used for 1000-way image classification task
  - 138 million parameters / 16-19 layers
  - *Still relevant today due to its simple connection and fast inference

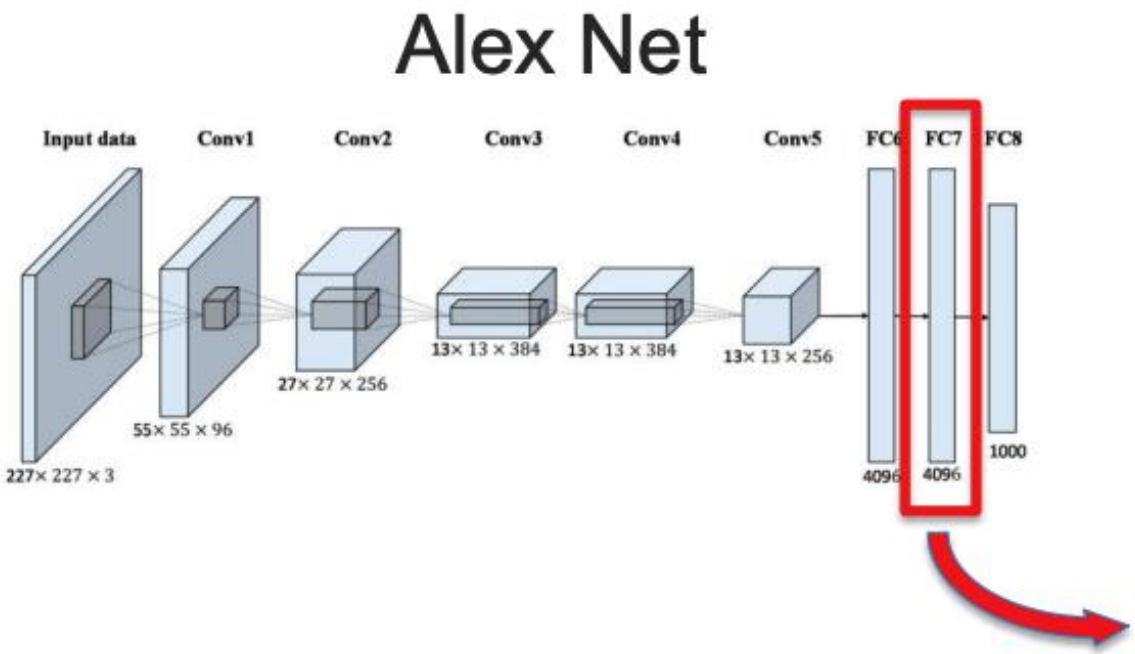
# Typical CNN Architectures

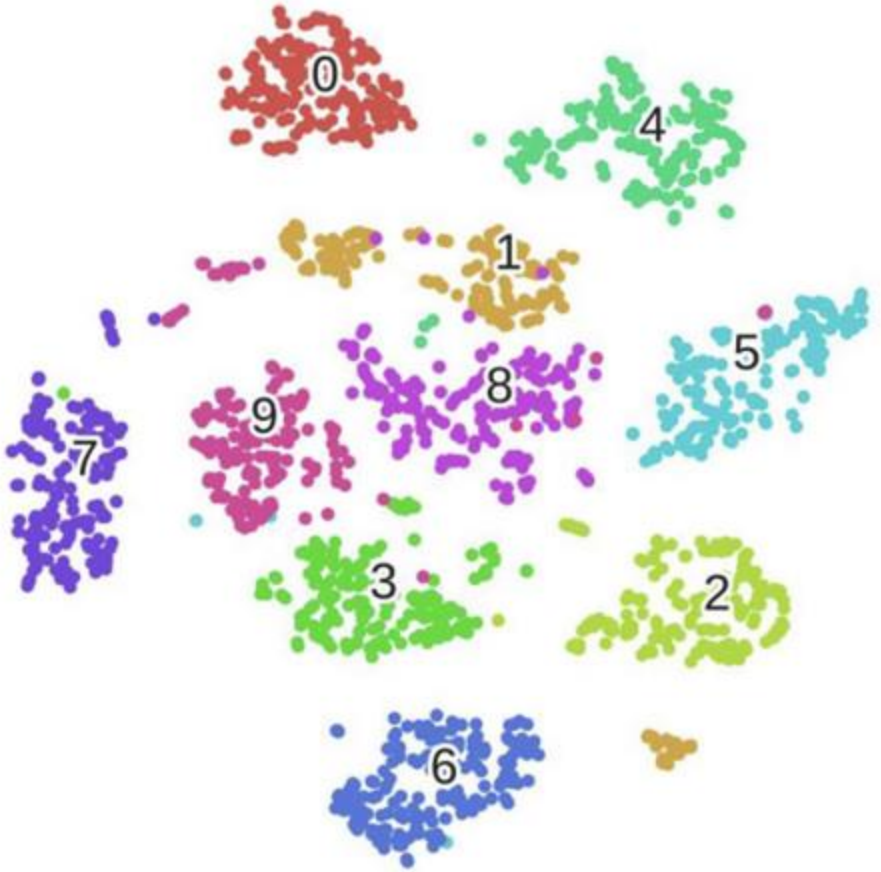
- Adding residual connections
- ResNet (He et al., 2015)
  - Up to 152 layers
  - But only 60 million parameters (138M for VGG, 3% better)



# Visualizing CNNs



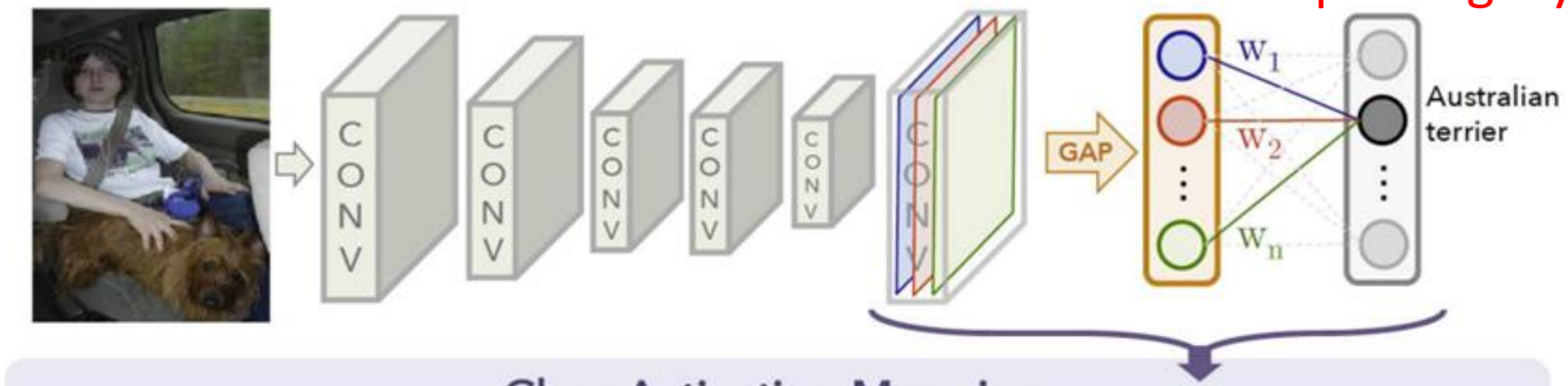

- Visualizing the Last CNN Layer: **t-SNE**
  - Embed high dimensional data points (i.e. feature codes) so that pairwise distances are conserved in local neighborhoods.

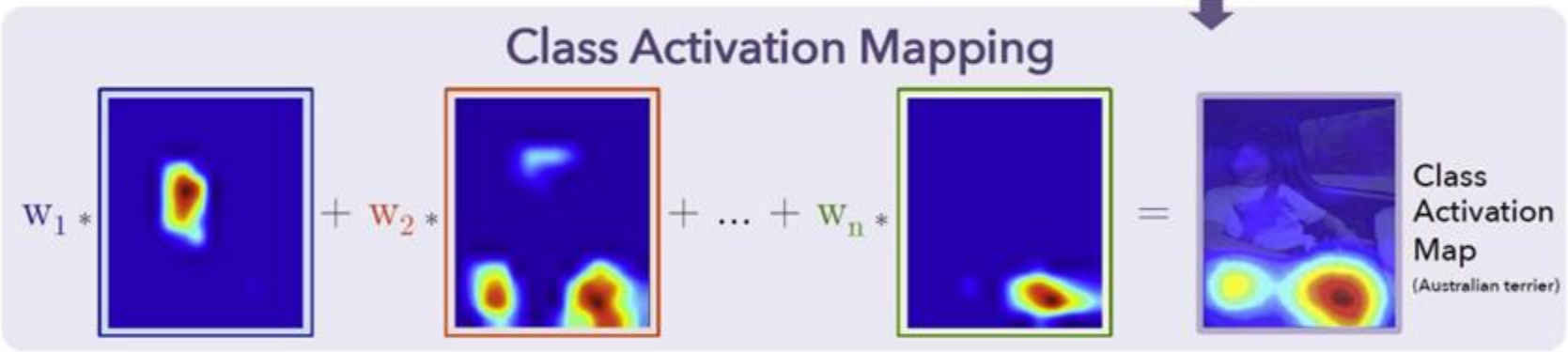
*https://www.scikit-yb.org/en/latest/api/text/tsne.html

# Visualizing CNNs

- CAM: Class Activation Mapping [CVPR 2016]

Train a SVM on the output of the global average pooling layer



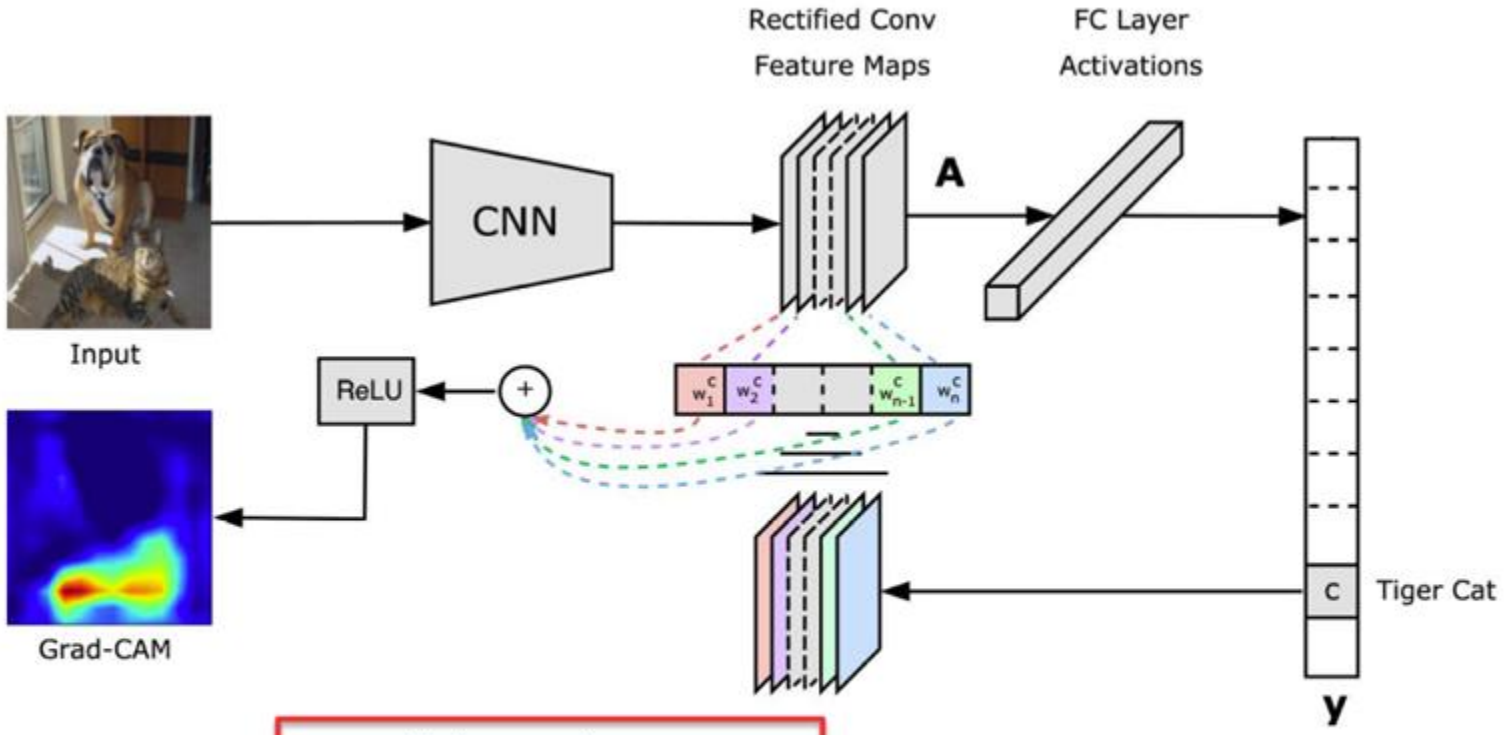




$$L^c_{\text{CAM}} = \underbrace{\sum_k w^c_k A^k}_{\text{linear combination}}$$

# Visualizing CNNs

- Grad-CAM [ICCV 2017]

No additional training.
Compute the gradient for each class

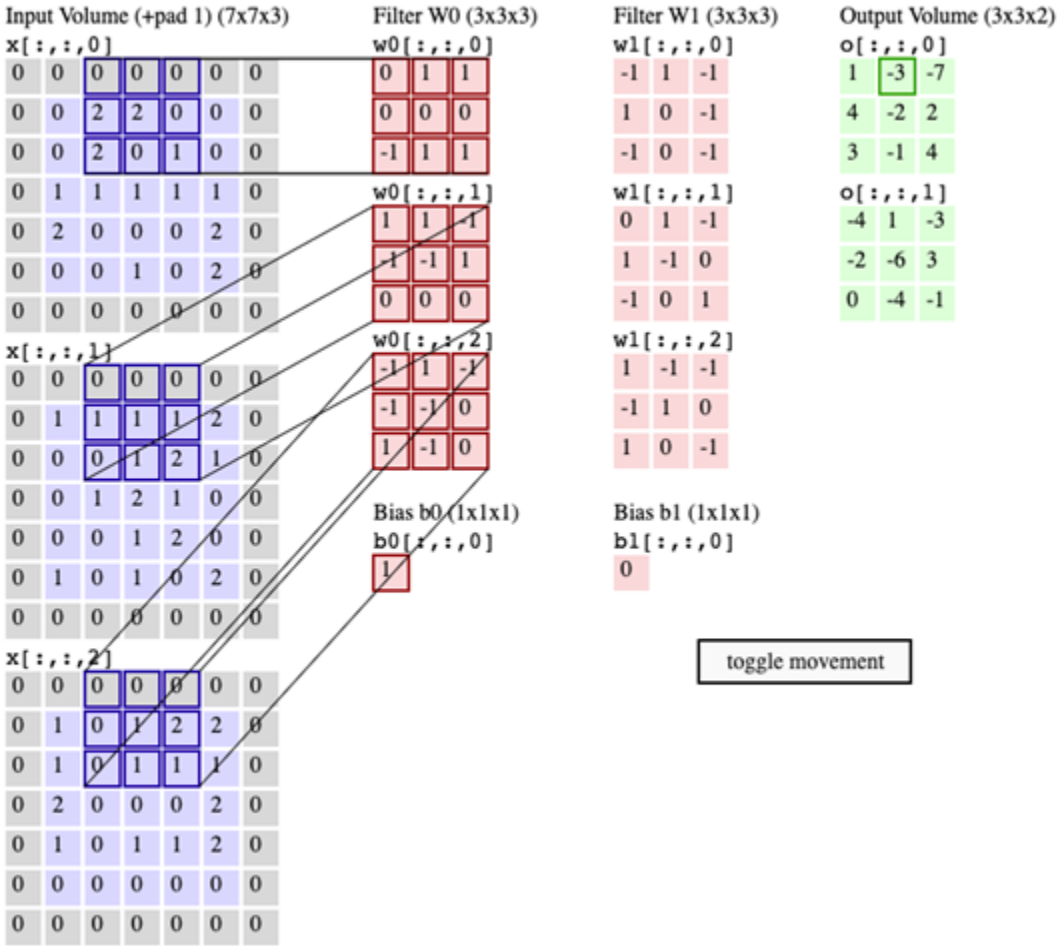



$$\alpha_k^c = \overbrace{\frac{1}{Z}\sum_i\sum_j}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A_{ij}^k}}_{\text{gradients via backprop}}$$

$$L_{\text{Grad-CAM}}^c = ReLU\underbrace{\left(\sum_k \alpha_k^c A^k\right)}_{\text{linear combination}}$$

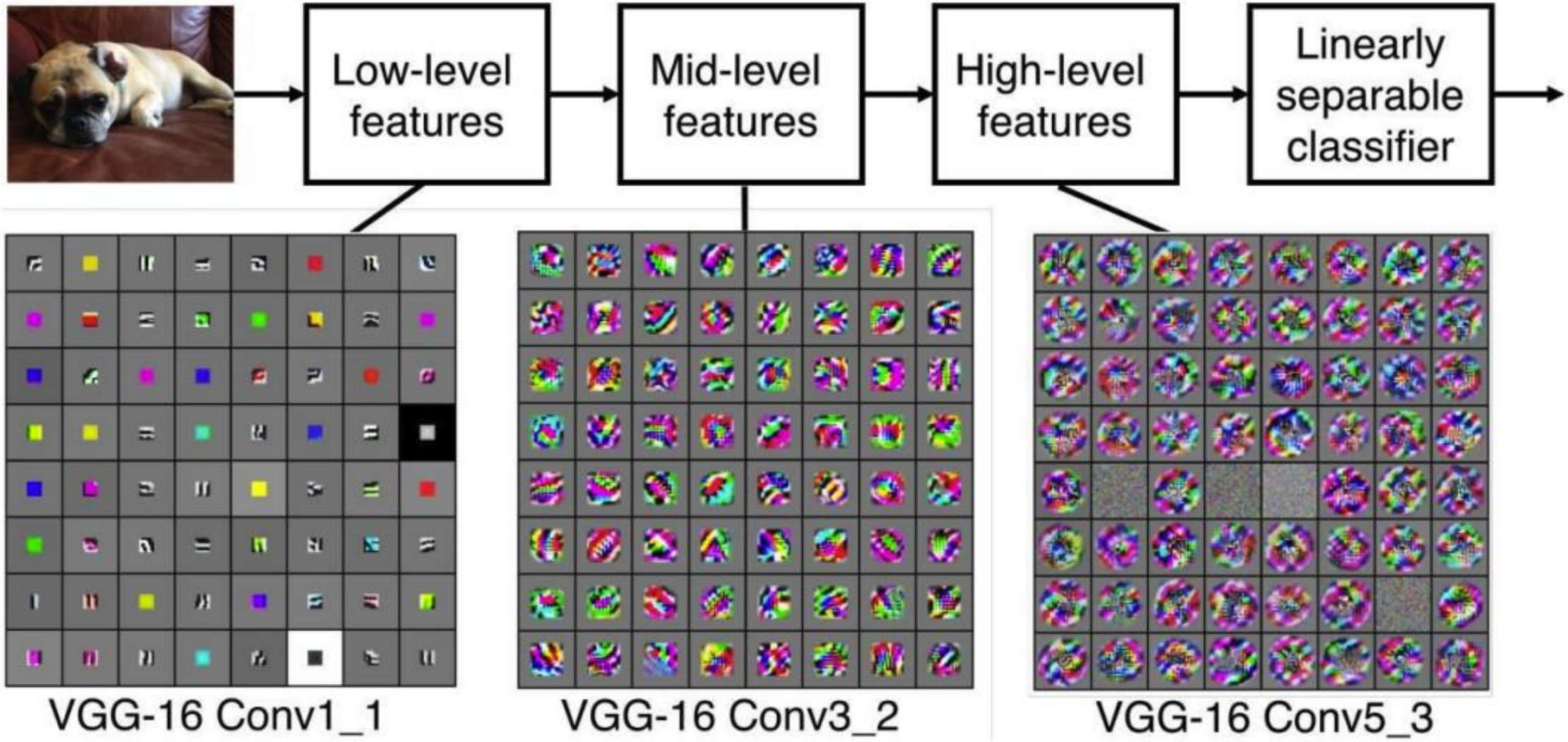
# Visualizing CNNs

- https://cs231n.github.io/convolutional-networks/



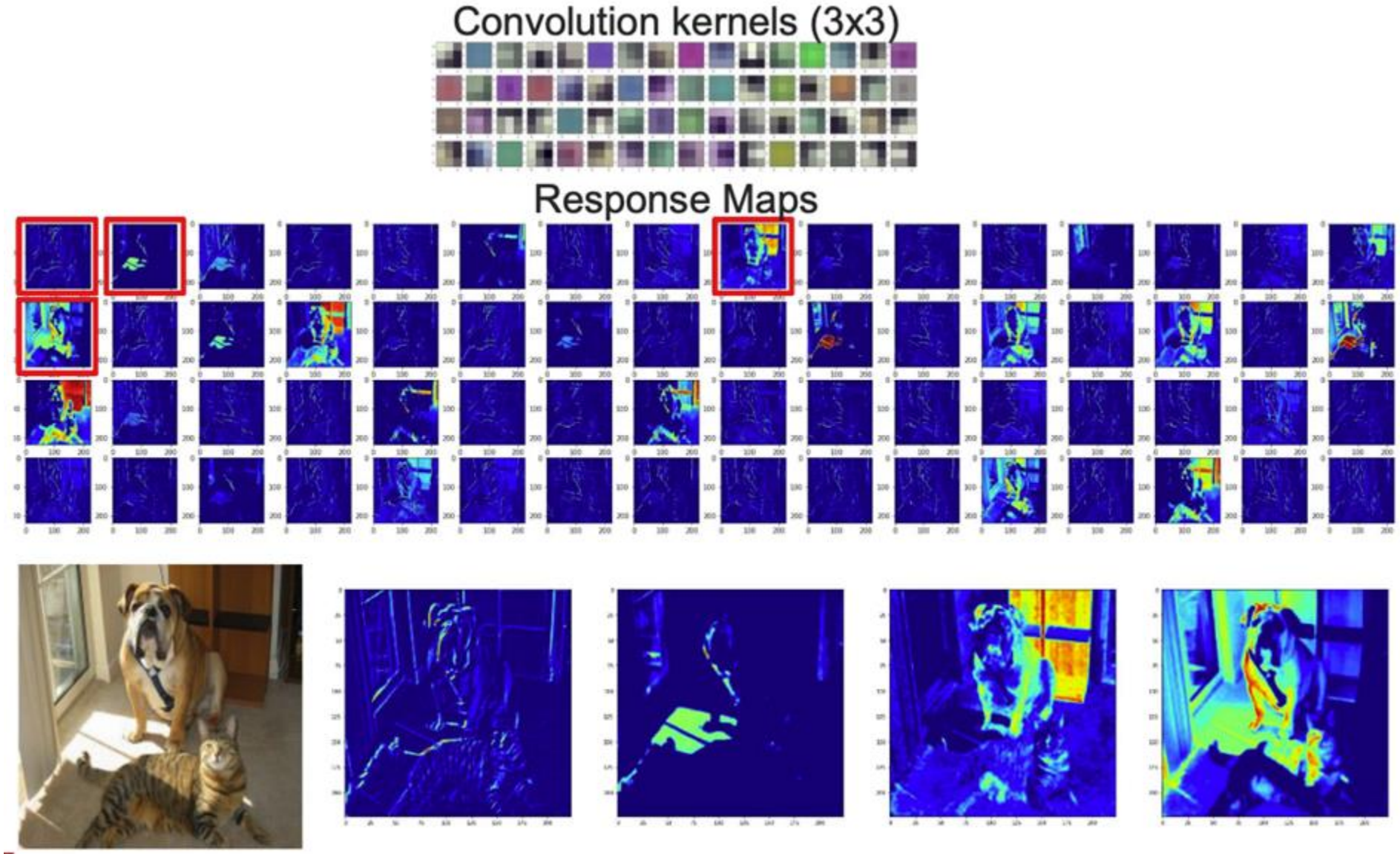
# Visualizing CNNs

VGGNet Convolution Kernels



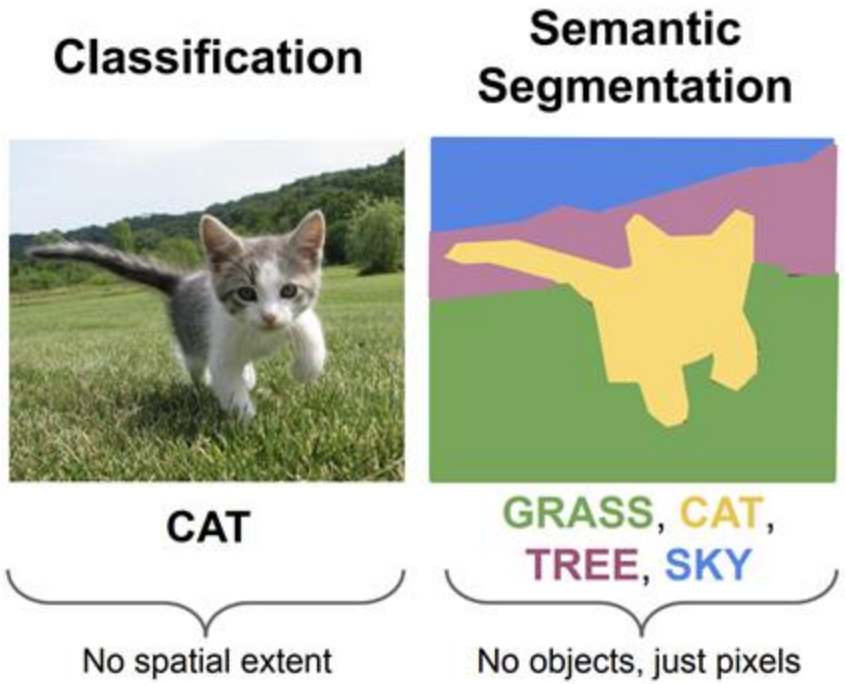
# Visualizing CNNs

VGGNet Response Maps (aka Activation Maps)

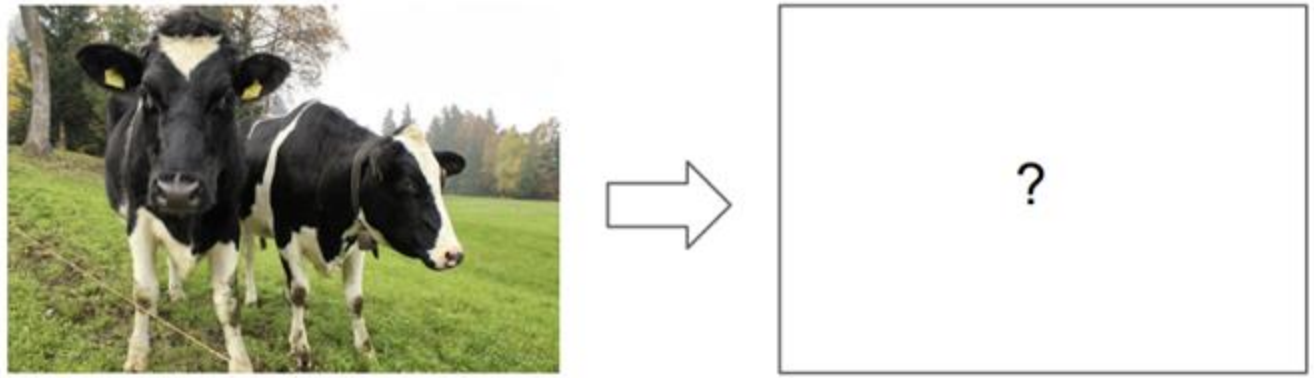


# Semantic Segmentation

- Image classification vs semantic segmentation



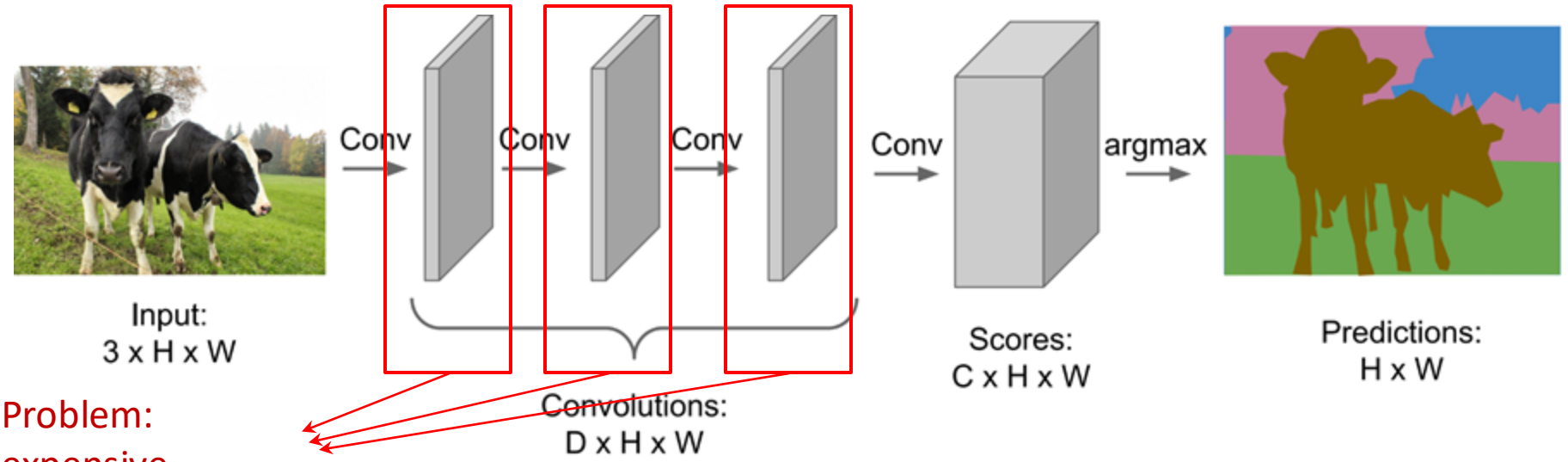

Paired training data: for each training image, each pixel is labeled with a semantic category.




At test time, classify each pixel of a new image

# Semantic Segmentation

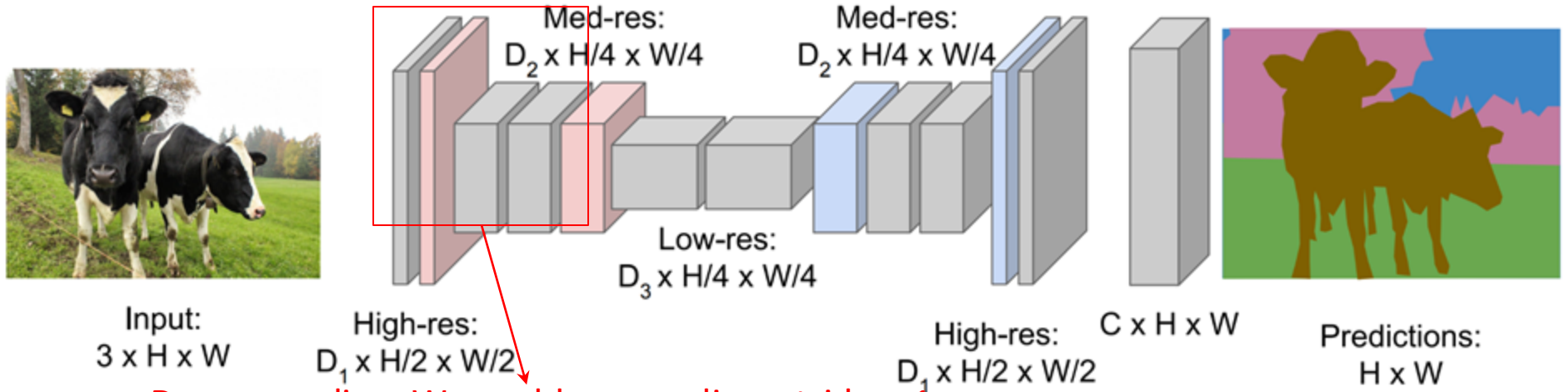
- Idea: Convolution
  - We see that convolution can get same-sized outputs
  - So design a network with only convolutional layers without downsampling poolings to make predictions for pixels all at once!




Problem: expensive computation at original resolution

# Semantic Segmentation

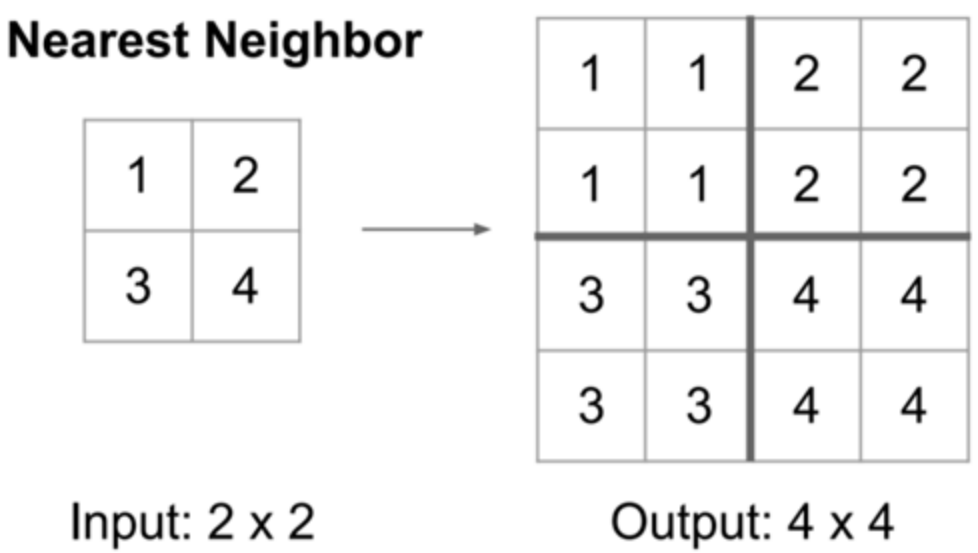
- Idea: convolutional network
  - We see that convolution can get same-sized outputs
  - Design network as a bunch of convolutional layers, with downsampling and upsampling inside the network!

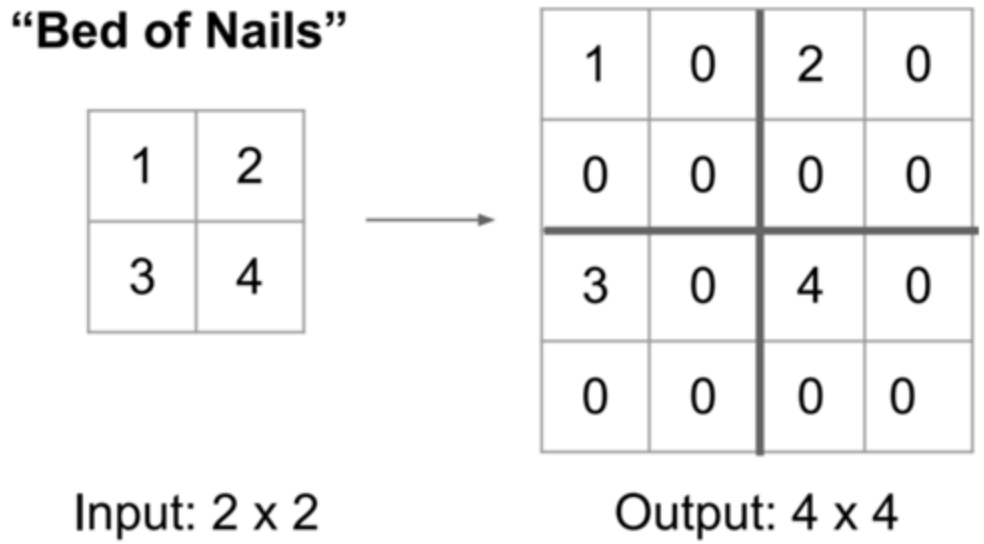



Downsampling: We could use pooling, stride > 1 convolution

# Semantic Segmentation

- Upsampling: Unpooling operation



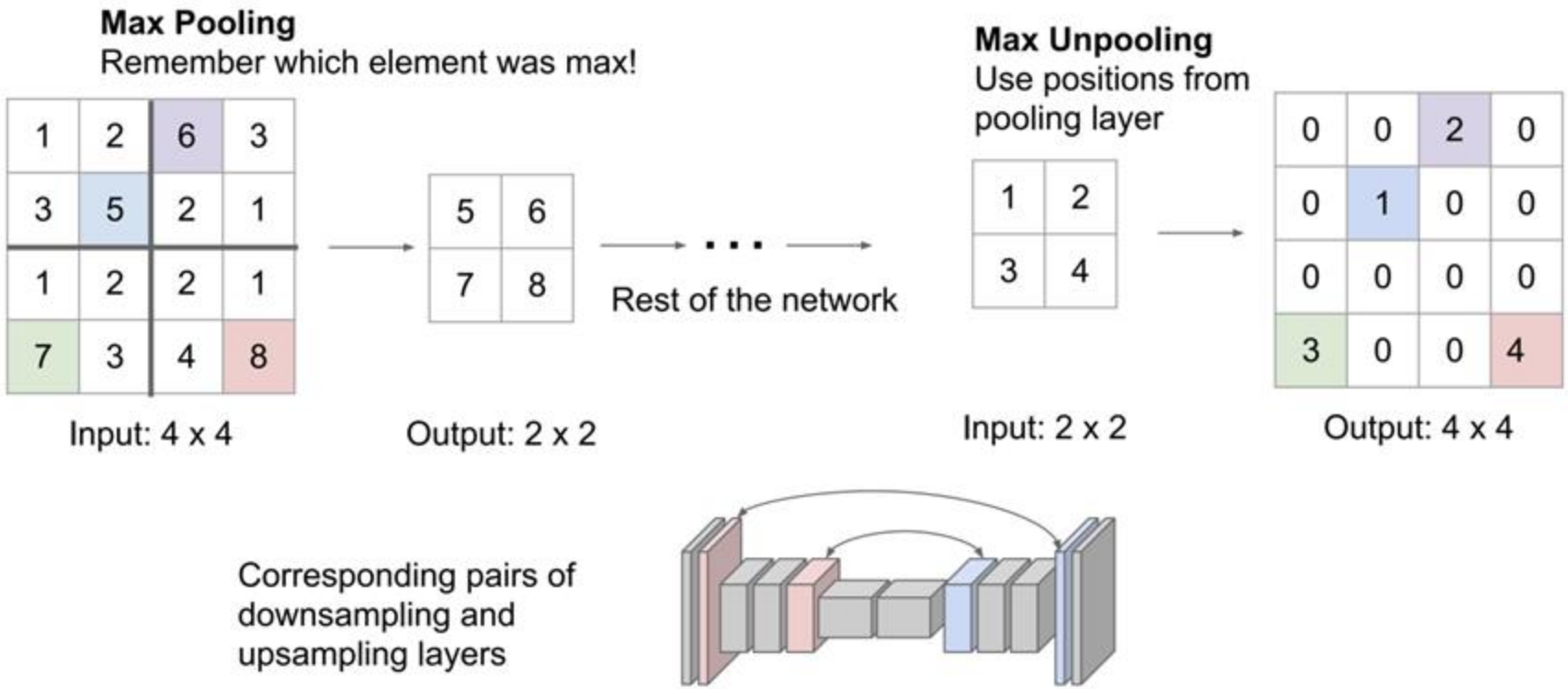




Put value at a fixed position and fill the rest with zeros

# Semantic Segmentation

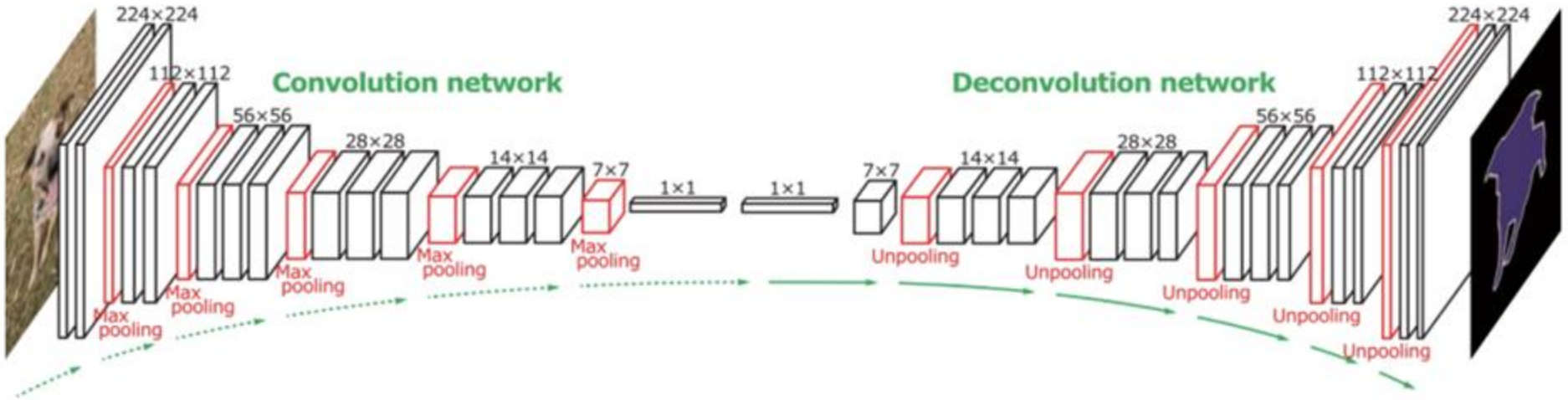
Upsampling: Max Unpooling

Because segmentation models are often symmetric



# Semantic Segmentation

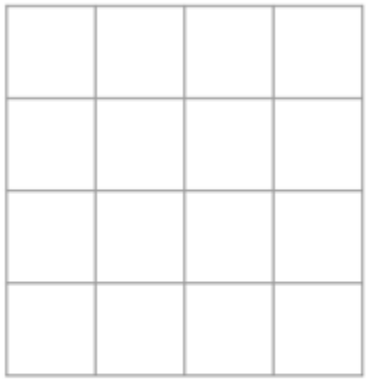
- Upsampling: Max Unpooling




*Learning deconvolution network for semantic segmentation. ICCV 2015

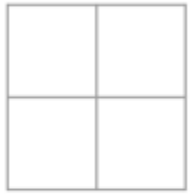
# Semantic Segmentation

- Upsampling: Transposed Convolution
  - Recall - normal convolution
  - Kernel (3 x 3), **stride=2,** pad=1
    - Filter moves 2 pixels in the input for every one pixel in the output
    - Output is “downsampled” using learnable weights



Input: 4 x 4

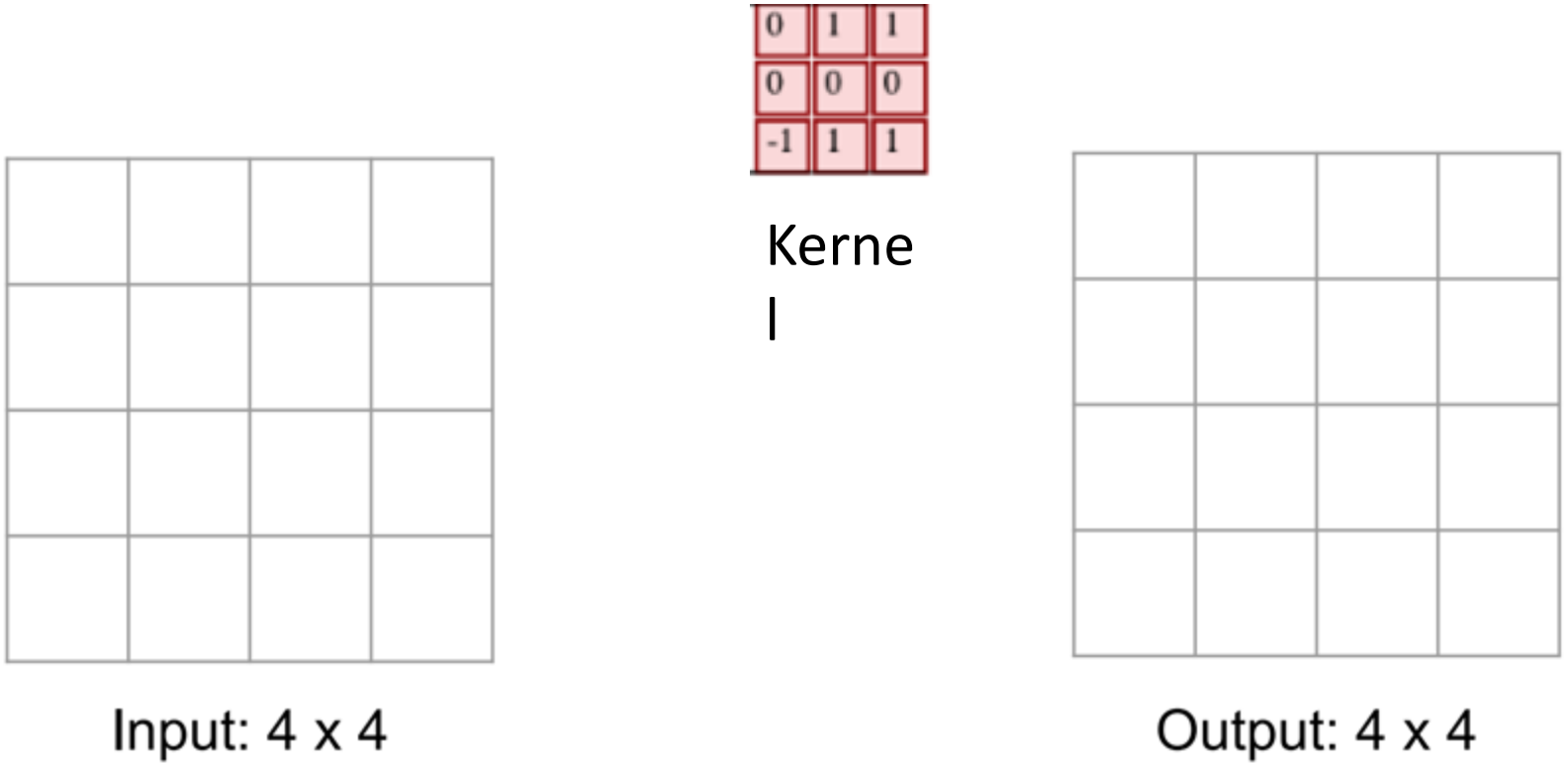

Kerne
l



Output: 2 x 2

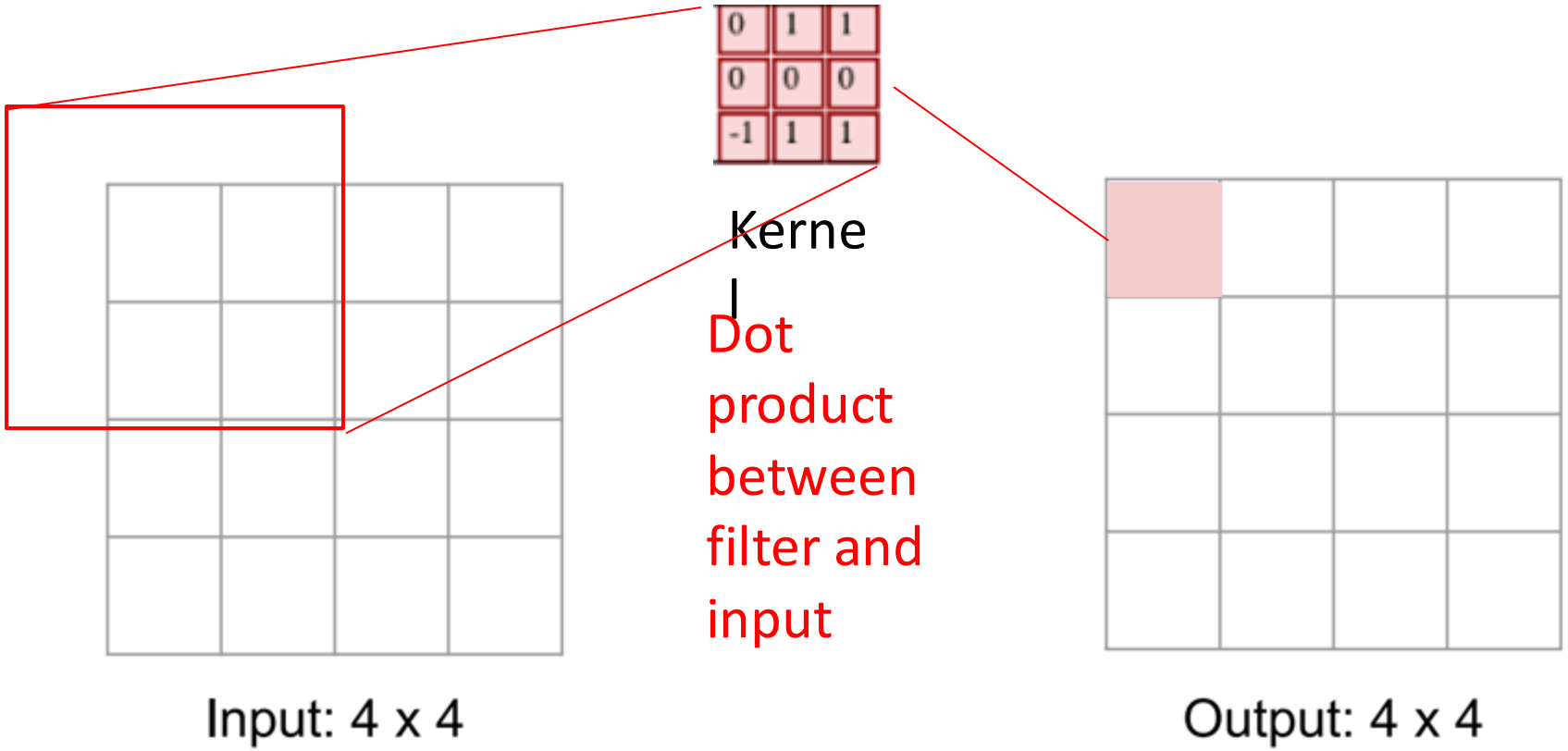
# Semantic Segmentation

- Upsampling: Transposed Convolution
  - Recall - normal convolution
  - Kernel (3 x 3), stride=1, pad=1 (input 4x4 becomes 6x6, so output size 6-3+1=4 x4)



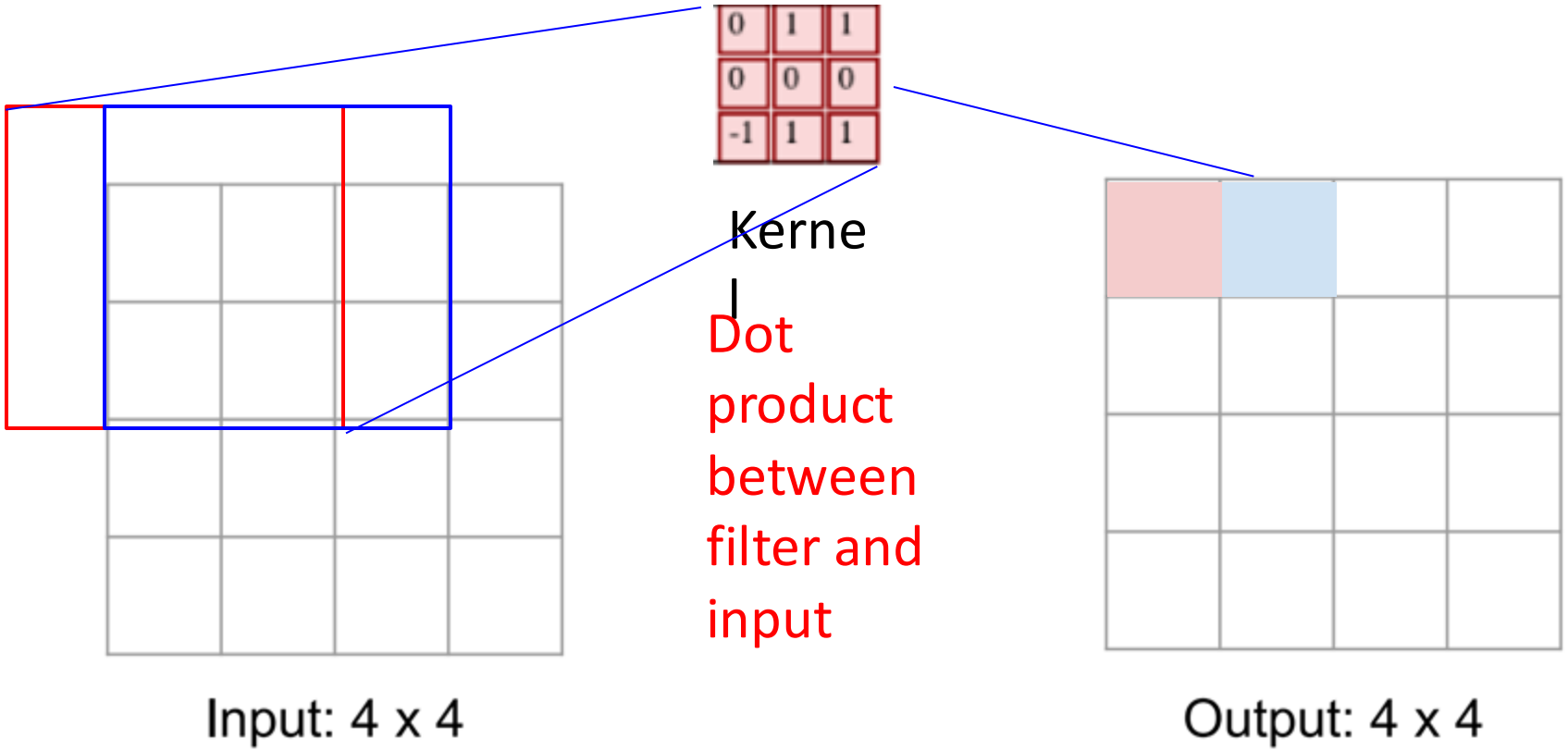
# Semantic Segmentation

- Upsampling: Transposed Convolution
  - Recall - normal convolution
  - Kernel (3 x 3), stride=1, pad=1



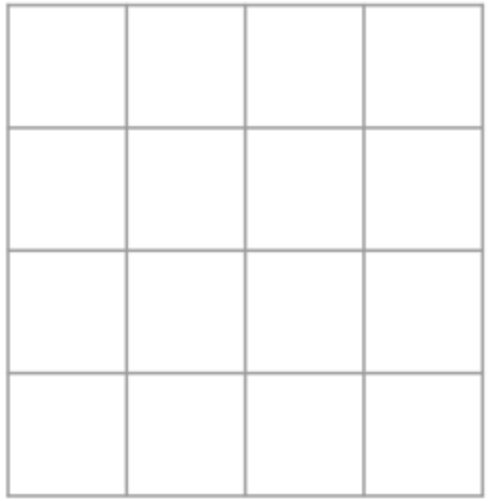
# Semantic Segmentation

- Upsampling: Transposed Convolution
  - Recall - normal convolution
  - Kernel (3 x 3), stride=1, pad=1

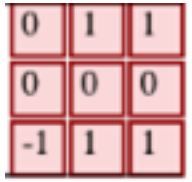


# Semantic Segmentation

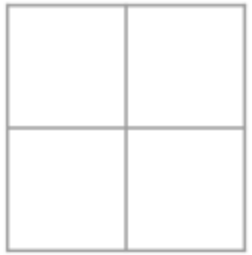
- Upsampling: Transposed Convolution
  - Recall - normal convolution
  - Kernel (3 x 3), **stride=2,** pad=1
    - Filter moves 2 pixels in the input for every one pixel in the output
    - Output is “downsampled” using learnable weights



Input: 4 x 4



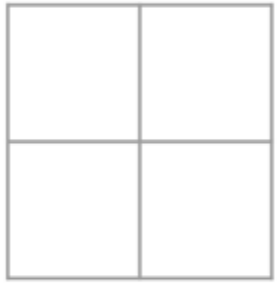

Kerne
l



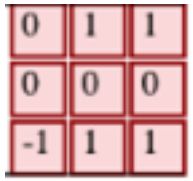
Output: 2 x 2

# Semantic Segmentation

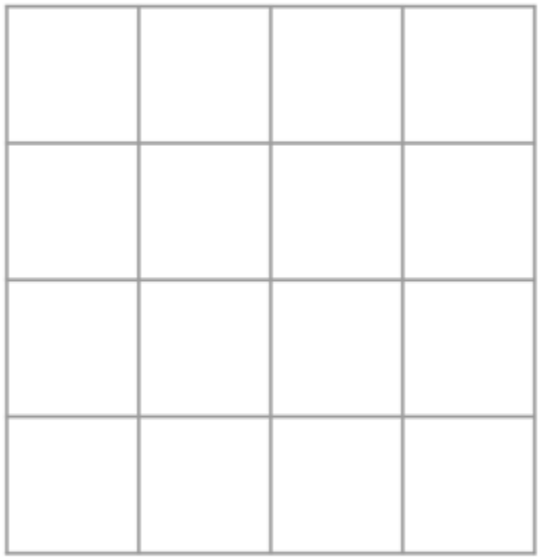
- Upsampling: Transposed Convolution
  - (3 x 3) **transposed convolution**, stride=2, pad=1: Output is upsampled!



Input: 2 x 2



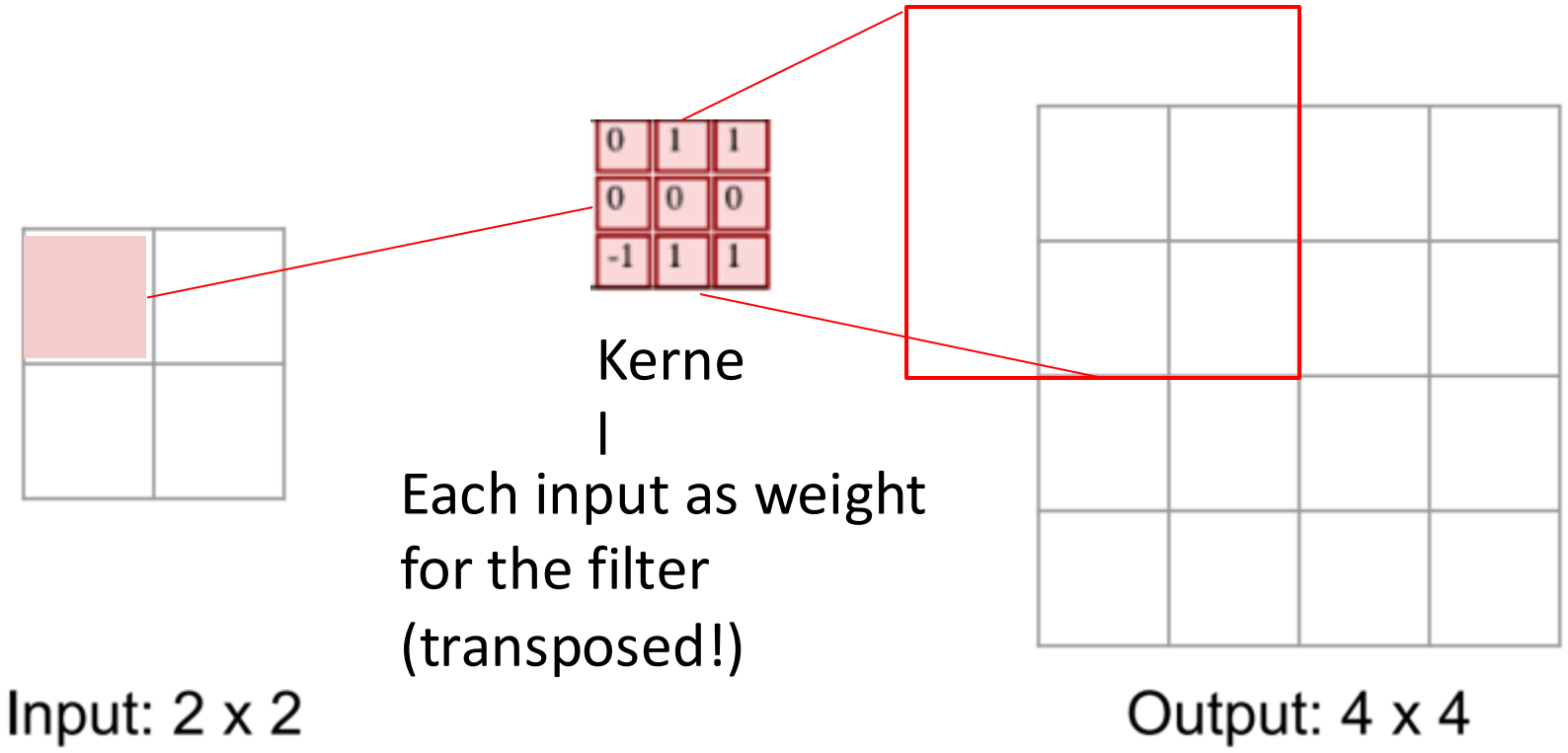

Kerne
l



Output: 4 x 4

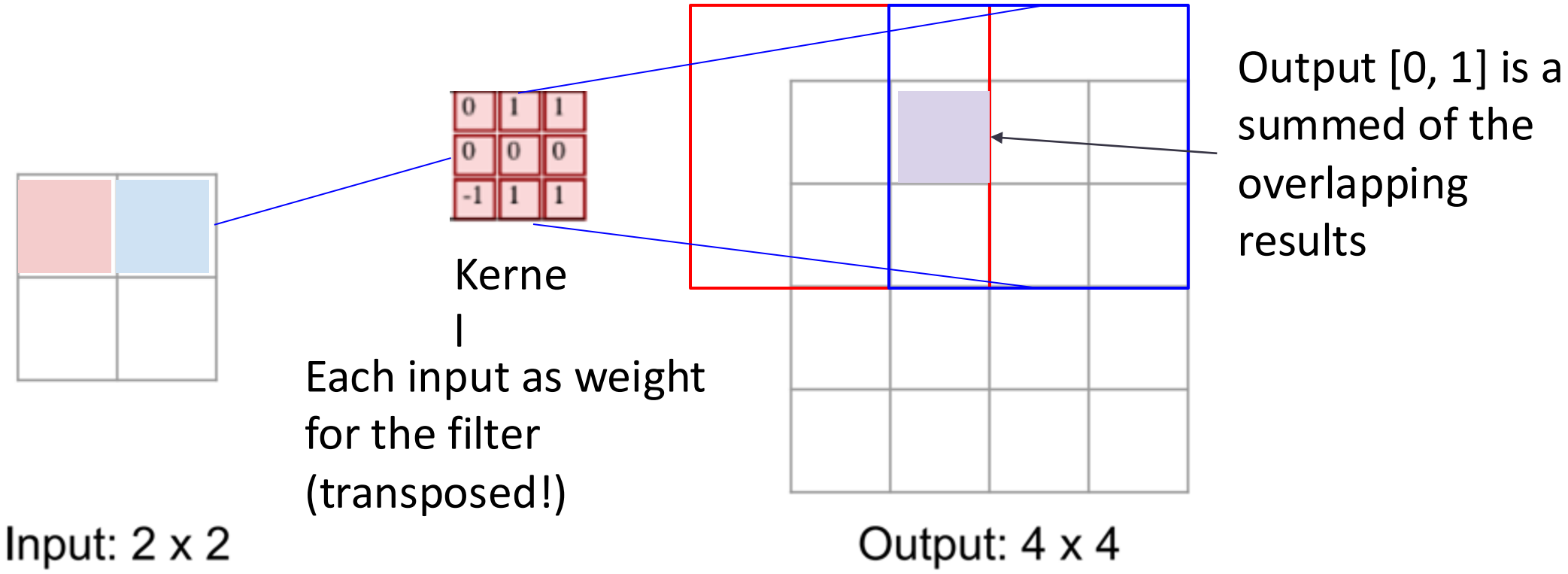
# Semantic Segmentation

- Upsampling: Transposed Convolution
  - (3 x 3) **transposed convolution**, stride=2, pad=1



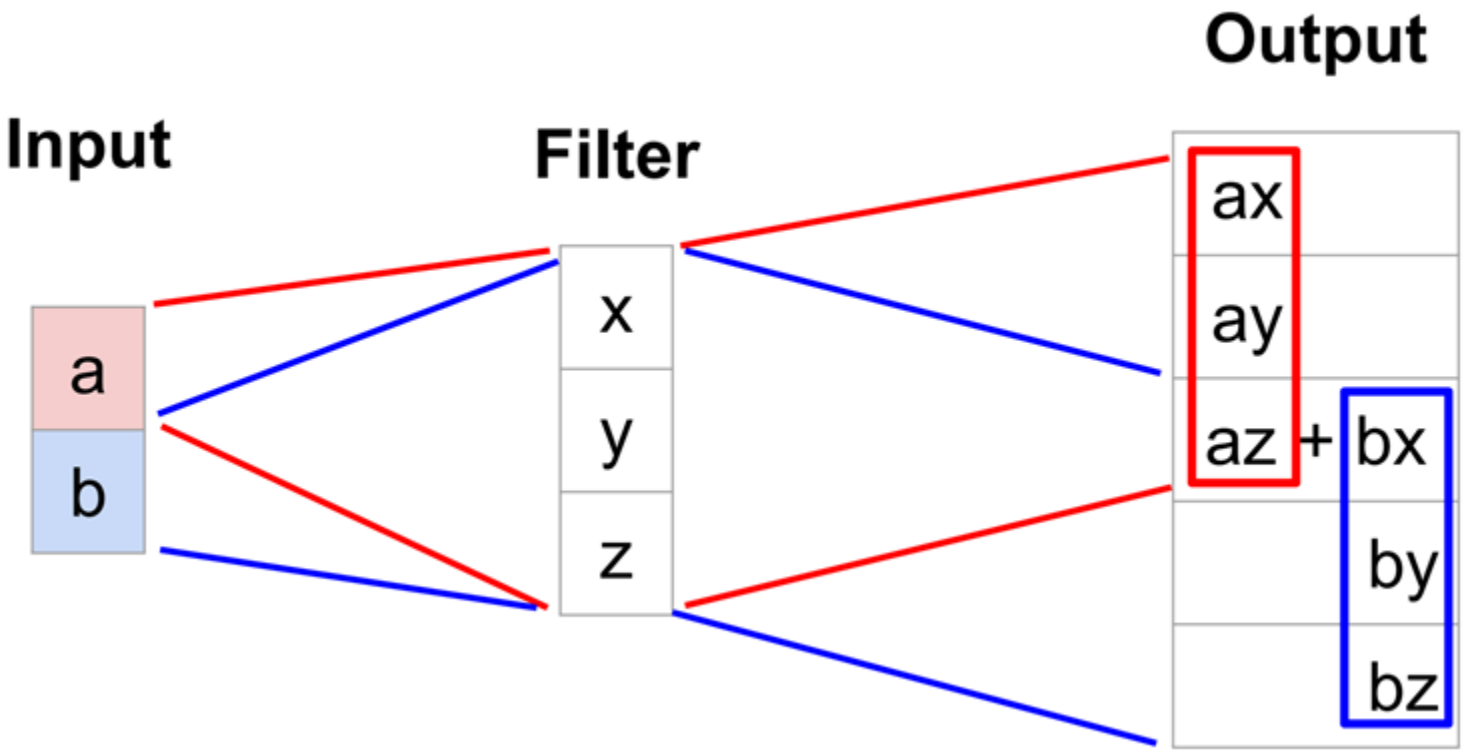
# Semantic Segmentation

- Upsampling: Transposed Convolution
  - (3 x 3) **transposed convolution**, stride=2, pad=1



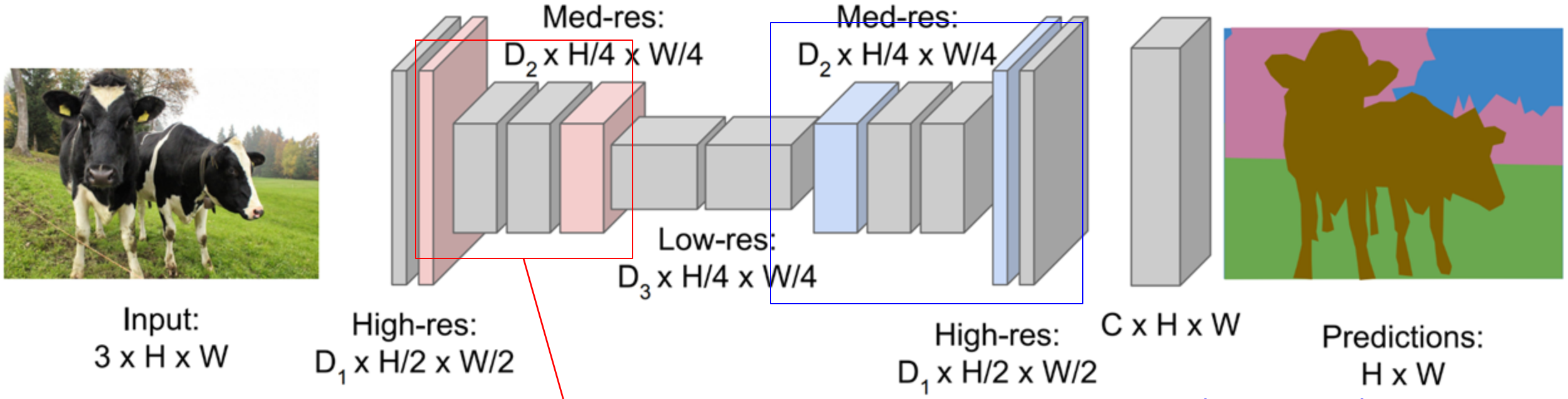
# Semantic Segmentation

- Upsampling: Transposed Convolution
  - 1D example
    - Output contains copies of the filter weighted by the input, summing at where at overlaps in the output



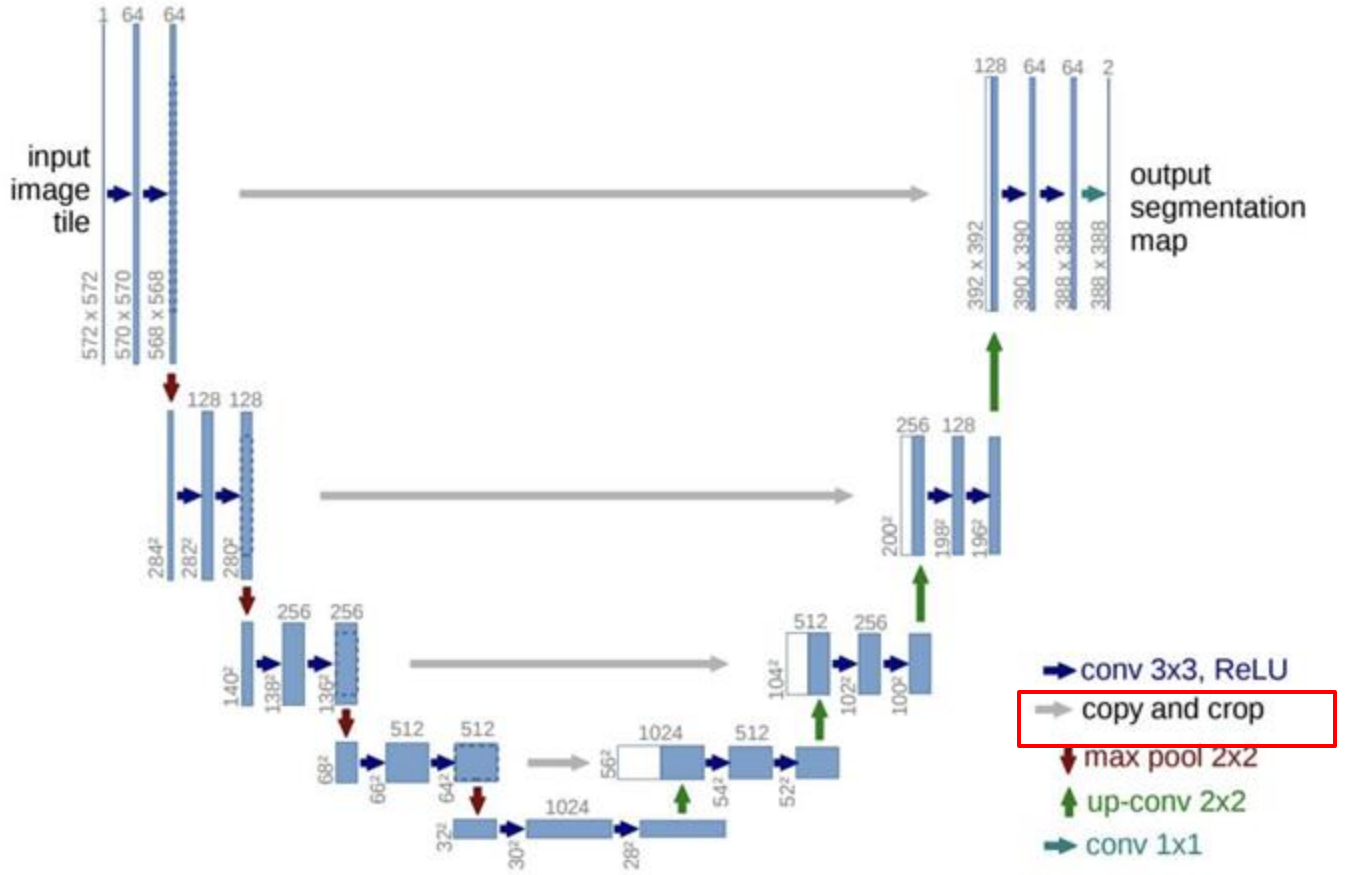
# Semantic Segmentation

- Idea: convolutional network
  - We see that convolution can get same-sized outputs
  - Design network as a bunch of convolutional layers, with downsampling and upsampling inside the network!



# Semantic Segmentation

- Idea: skip connection - U-net design

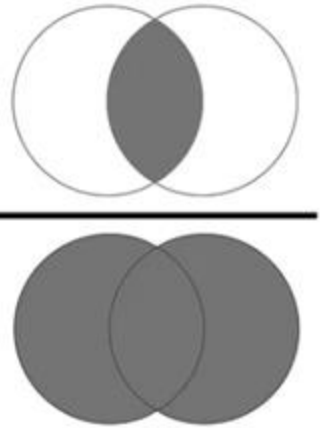



*U-net: Convolutional networks for biomedical image segmentation

# Semantic Segmentation

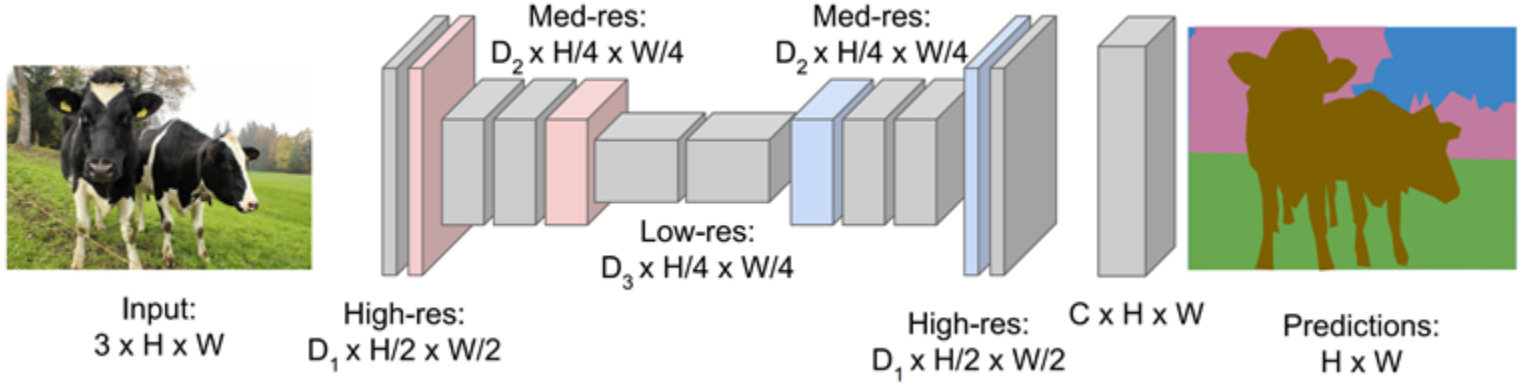
- Evaluation
  - Datasets: Cityscapes, ADE20K
  - Metric: m-IOU (mean Intersection over Union)
    - The mIoU is the average between the IoU of the segmented objects over all the images of the test dataset.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



# Semantic Segmentation

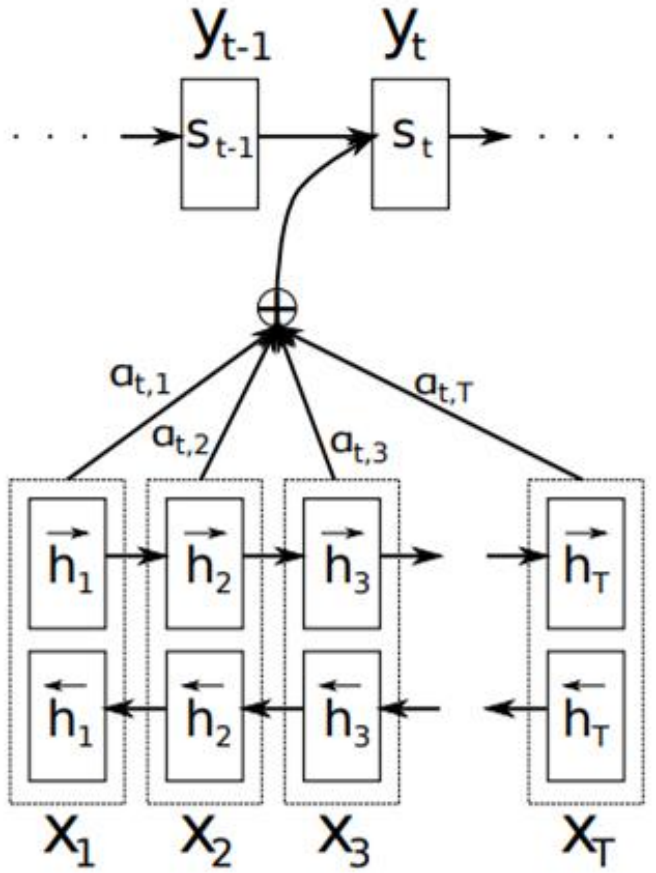
- Summary
  - Task: Label each pixel in the image with a category label
  - Design: convolution **encoder** and deconvolution **decoder**
    - Deconvolution: ML people don't like this name
    - More precisely - transposed convolution
    - And we could also use unpooling
  - Evaluation: m-IOU
  - Tools
    - Deeplab



# Attention Models

- Translation - learned alignment between words
  - Given source senten $(x_1, x_2, \ldots, x_T)$
  - and previous generated words
  - Decoder: Generate the next w $y_t$ rd
  - Learned attention (a.k.a alignment) over each source word when generating the t-th target word
    - Intuitively, decoder learns **which source words to pay attention to**
    - because not all the inputs would be used in generating the corresponding output

Output word sequence

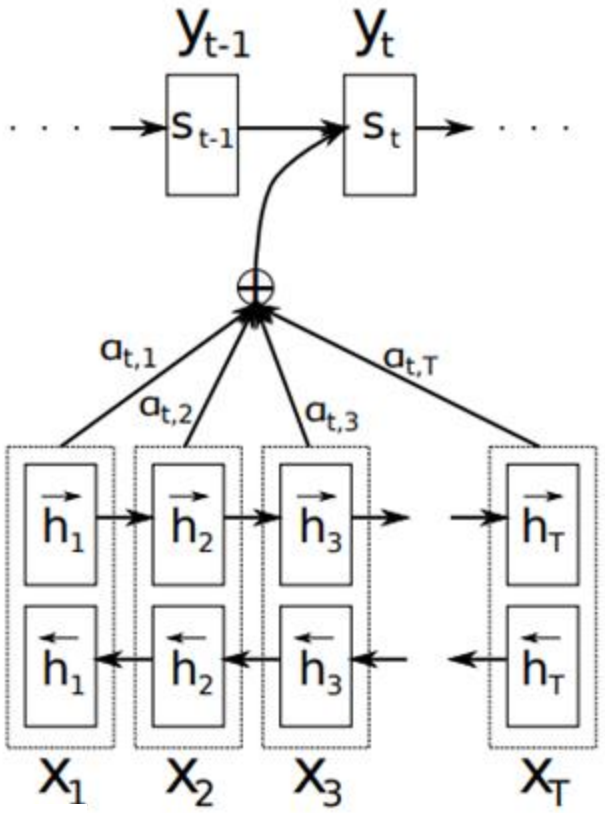



Input word sequence

*Neural Machine Translation by Jointly Learning to Align and Translate

# Attention Models

- Translation - learned alignment between words
  - The attention weights are calculated by normalizing the output score of a feed-forward neural network described by the function ***a*** that captures the alignment between input at ***j*** and output at ***i***.
  - The context vector ***ci*** for the output word ***yi*** is generated using the weighted sum of the attention and hidden states




$$e_{ij} = a(s_{i-1}, h_j) \qquad \alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})} \qquad c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

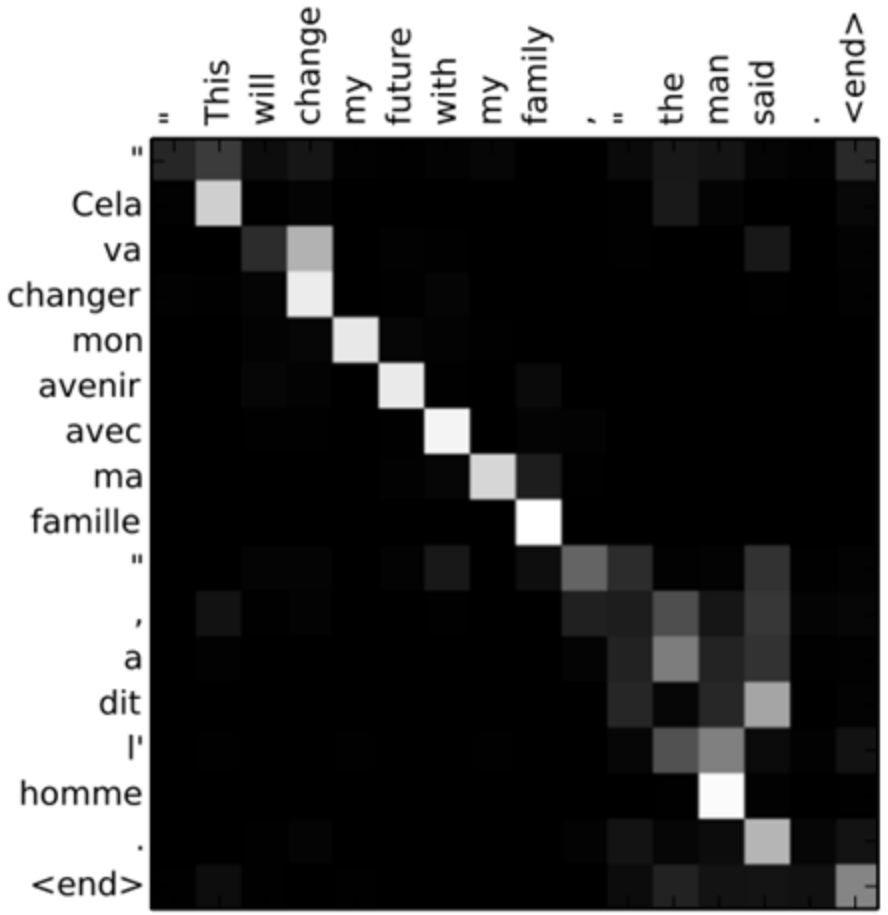
Last output state

Hidden state from LSTM/RNN at position j

*Neural Machine Translation by Jointly Learning to Align and Translate

# Attention Models

- Translation - learned alignment betwe words
  - **Visualization**
  - **Aligning each source word to target wo**




(d)

*Neural Machine Translation by Jointly Learning to Align and Translate

# Attention Models

- Visual Question Answering - aligning image part and questions

$$\tilde{v}_I = \sum_i p_i v_i$$

Part-of-image

Attention: weights of Part-of-image

$$v_i \in \mathbf{R}^d$$

$$v_I \in \mathbf{R}^{d \times m}$$


Figure 2: CNN based image model

$$u = \tilde{v}_I + v_Q$$

The encoder vector: question + attended image info

m=14 x 14

$$p_{ans} = softmax(W_a u)$$

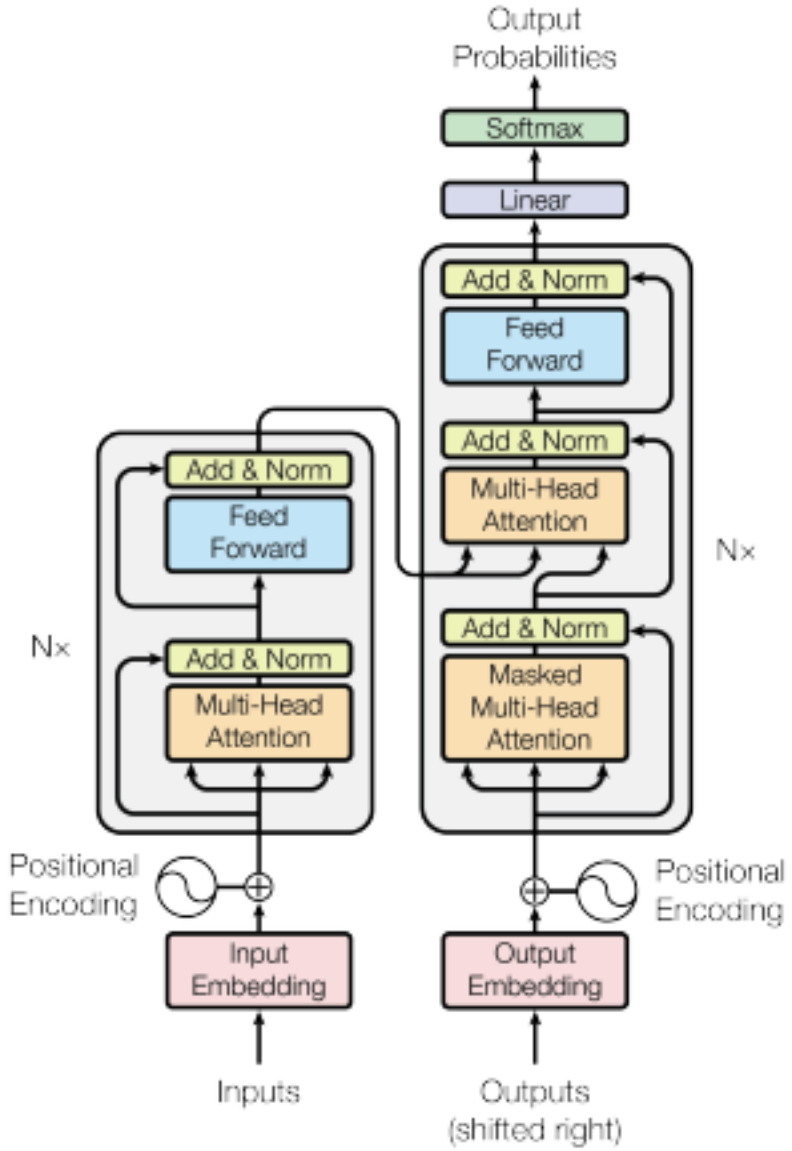
What is the mustache made of?

*Bias vector is not shown for simplicity

[1] Stacked Attention Networks for Image Question Answering. CVPR 2016

# Attention Models: Self-attention

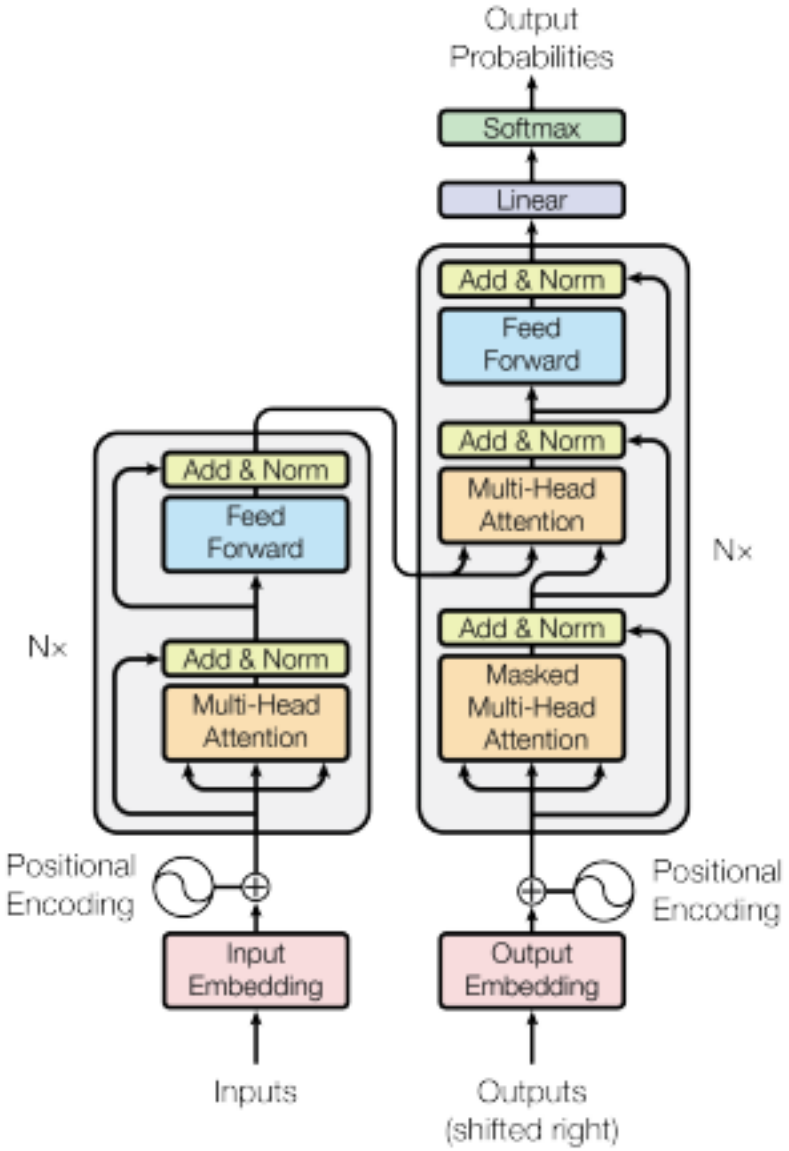
- Translation - learned alignment between words
  - **Replacing the LSTMs / RNNs with attention layers**
  - Vanilla RNNs are Slow and have terrible long-term memory
  - LSTMs and GRUs fix the memory problem, but are still slow and sequential




*Attention is All you Need. NeurIPS 2017

# Attention Models: Self-attention

- Translation - learned alignment between words
  - Replacing the LSTMs / RNNs with attention layers
  - Self-attention and cross-attention
    - **Query, Key, Value in attention**
      - Query: This is what pays the attention
      - Keys: These help queries figure out how much attention to pay to each of the values
      - Values: These are paid attention to
        - So computed_attention x values
      - Attention Weights: How much attention to pay.
        - Query x Key




*Attention is All you Need. NeurIPS 2017

# Attention Models: Self-attention

Suppose you have N token inputs, computing N attention weights

1. Attention weights $a_{1:N}$ are query-key similarities:

$$\hat{a}_i = q \cdot k_i$$

So given one query, get attention for each key.

Normalized via softmax

2. Output z is attention-weighted average of values $v_{1:N}$:

$$z = \sum_i \hat{a}_i v_i = \hat{a} \cdot v$$

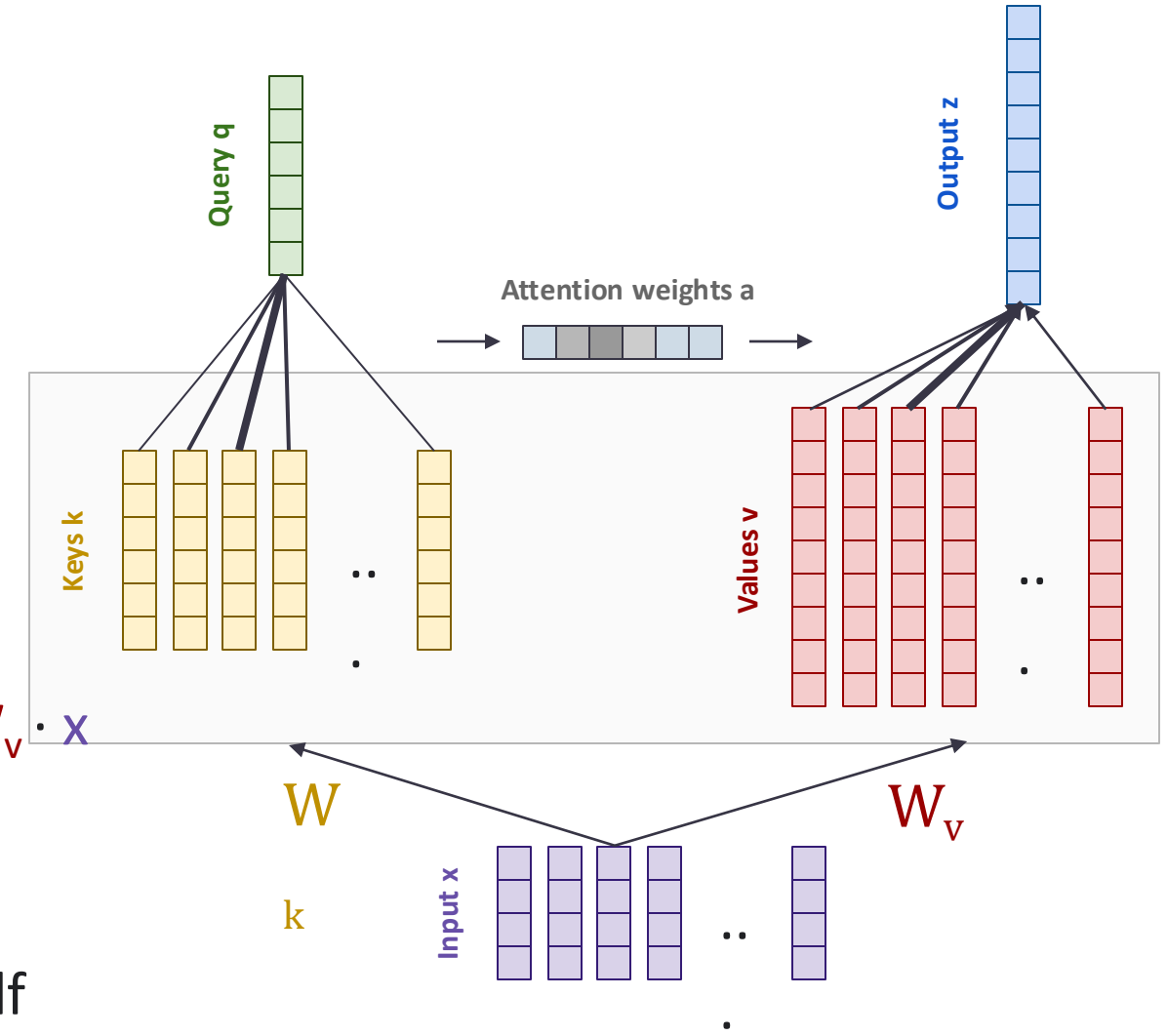
3. Usually, k and v are derived from the same input x:

$$k = W_k \cdot x \qquad v = W_v \cdot x$$

The query q can come from a separate input y:

$$q = W_q \cdot y$$ "Cross attention"

Or from the same input x! Then we call it "self attention":




*Attention is All you Need. NeurIPS 2017

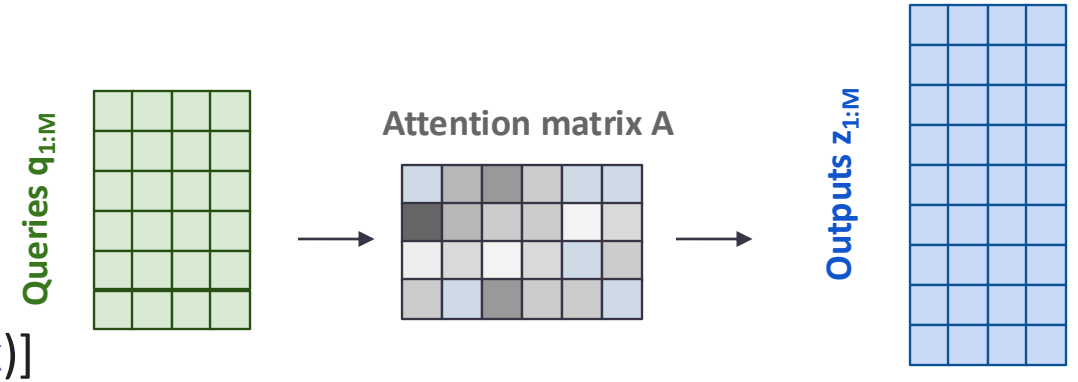
# Attention Models: Self-attention

1.We usually use **many queries** $q_{1:M}$, not just one.

Stacking them leads to the Attention matrix $\mathbf{A}_{1:N,1:M}$

and subsequently to many outputs:

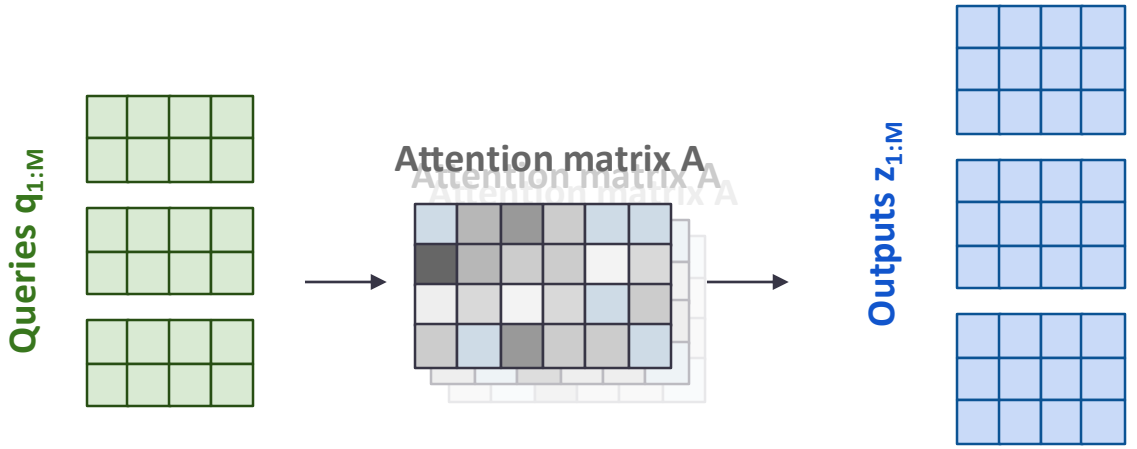
$$z_{1:M} = \text{Attn}(q_{1:M},x) = [\text{Attn}(q_1,x) \mid \text{Attn}(q_2,x) \mid \ldots \mid \text{Attn}(q_M,x)]$$




2. We usually use "**multi-head**" attention. This means the operation is repeated K times and the results are concatenated along the feature dimension. **W**s differ.

3. The most commonly seen formulation:

$$z = \text{softmax}\left(\frac{Q \times K^T}{\sqrt{d_k}}\right) V$$



# Attention Models: Self-attention

- Applying the transformer model for vision
  - Representing image pixels as **sequences**
  - But remember the complexity is $O(N^2)$

# Attention Models: Vision-Transformer

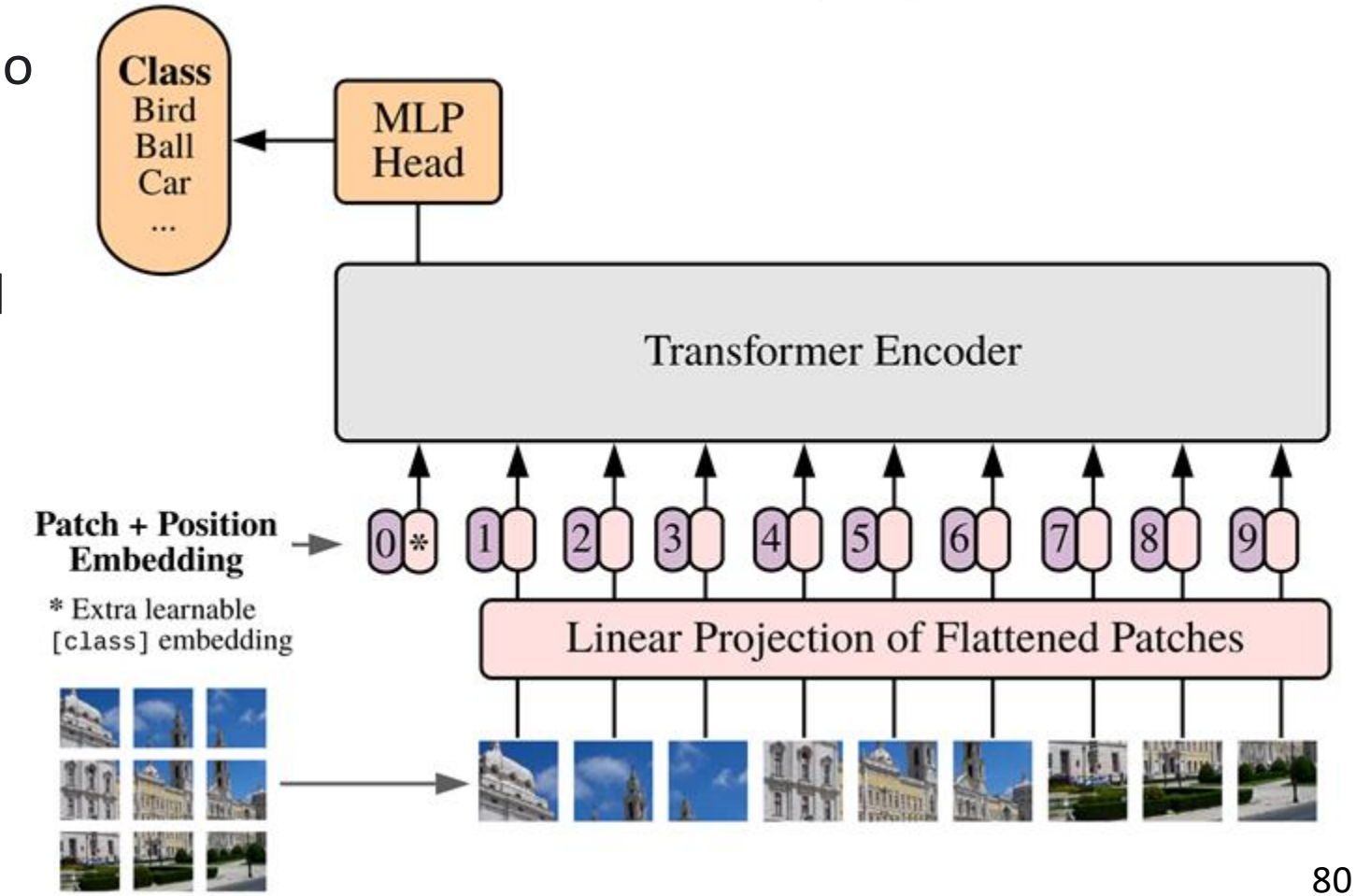
Many prior works attempted to introduce self-attention at the pixel level.

For 224px², that's 50k sequence length, too much!

Thus, most works restrict attention to local pixel neighborhoods, or as high-level mechanism on top of detections.

The **key breakthrough** in using the full Transformer architecture, standalone, was to **"tokenize" the image** by **cutting it into patches** of 16px², and treating each patch as a token, e.g. embedding it into input space.

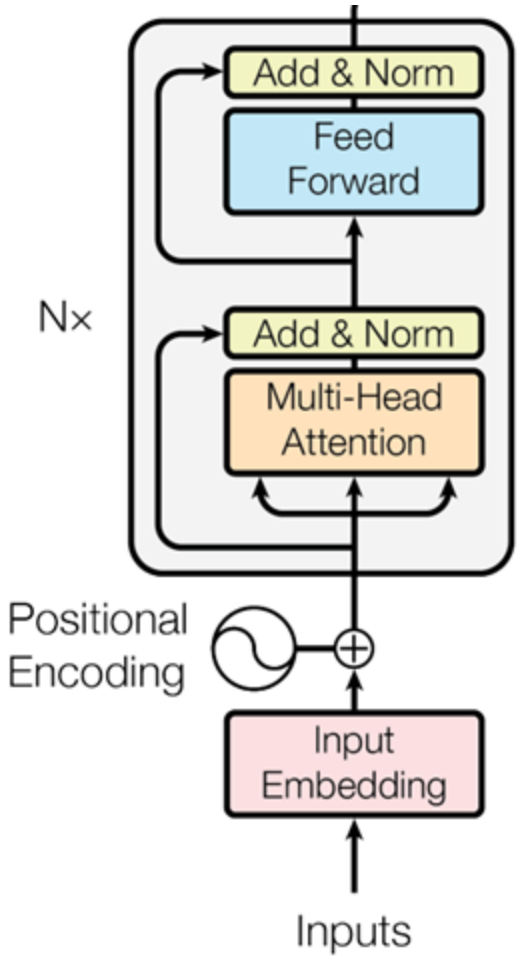
## Vision Transformer (ViT)




*An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. 2020

# Attention Models: Vision-Transformer

- Applying the transformer model for vision
  - And we only use the transformer block as encoders
  - Multi-head attention + Feed forward layer
  - Simpler than the machine translation model!



# Attention Models: Vision-Transformer



Gif by: lucidrains

# Attention Models: Vision-Transformer

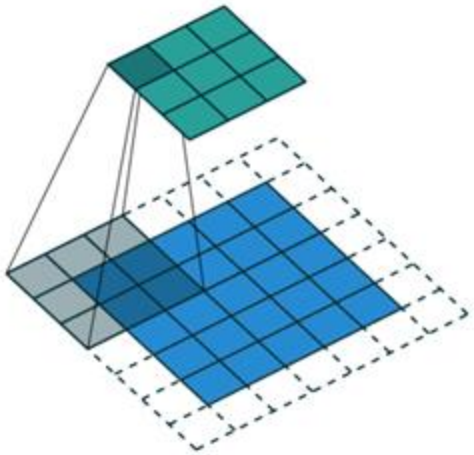
- Performance
  - Worse than Resnet when trained just on ImageNet
  - Performance improved when pre-trained on big dataset (JFT-300M, JFT-3B)
  - Pretrained outperforms much bigger CNNs

| | Ours-JFT (ViT-H/14) | Ours-JFT (ViT-L/16) | Ours-I21K (ViT-L/16) | BiT-L (ResNet152x4) | Noisy Student (EfficientNet-L2) |
|---|---|---|---|---|---|
| ImageNet | **88.55** ± 0.04 | 87.76 ± 0.03 | 85.30 ± 0.02 | 87.54 ± 0.02 | 88.4/88.5* |
| ImageNet ReaL | **90.72** ± 0.05 | 90.54 ± 0.03 | 88.62 ± 0.05 | 90.54 | 90.55 |
| CIFAR-10 | **99.50** ± 0.06 | 99.42 ± 0.03 | 99.15 ± 0.03 | 99.37 ± 0.06 | – |
| CIFAR-100 | **94.55** ± 0.04 | 93.90 ± 0.05 | 93.25 ± 0.05 | 93.51 ± 0.08 | – |
| Oxford-IIIT Pets | **97.56** ± 0.03 | 97.32 ± 0.11 | 94.67 ± 0.15 | 96.62 ± 0.23 | – |
| Oxford Flowers-102 | 99.68 ± 0.02 | **99.74** ± 0.00 | 99.61 ± 0.02 | 99.63 ± 0.03 | – |
| VTAB (19 tasks) | **77.63** ± 0.23 | 76.28 ± 0.46 | 72.72 ± 0.21 | 76.29 ± 1.70 | – |
| TPUv3-core-days | 2.5k | 0.68k | 0.23k | 9.9k | 12.3k |

This is costing around $30,000

# Attention Models: Vision-Transformer

- CNNs vs Vision Transformers
  - Key design different
    - Convolution is translation invariant* (this is what we call “inductive bias”, a property that is built into the design)
      - An object translated should has the same object response on the feature maps
    - While attention is permutation invariant
      - Change the order of the element the attention stays the same
      - The inductive bias is more relaxed
  - Therefore
    - Given more data, transformers are better than CNNs
  - Another perk - transformers can be used for multimodal data



* And also translation equivariance

# Multimodal Learning: Vision-Language Model

- CLIP (Contrastive Language–Image Pre-training) by OpenAI in 2021
  - Train on 400 million image+text crawled from the internet
  - Proved that large-scale (noisy) vision-language pre–training can lead to great performance (with proper prompt engineering)
    - Any zero-shot downstream classification tasks
    - AI generated content (text-to-image generation)
      - DALL-E, etc.

* https://openai.com/blog/clip/

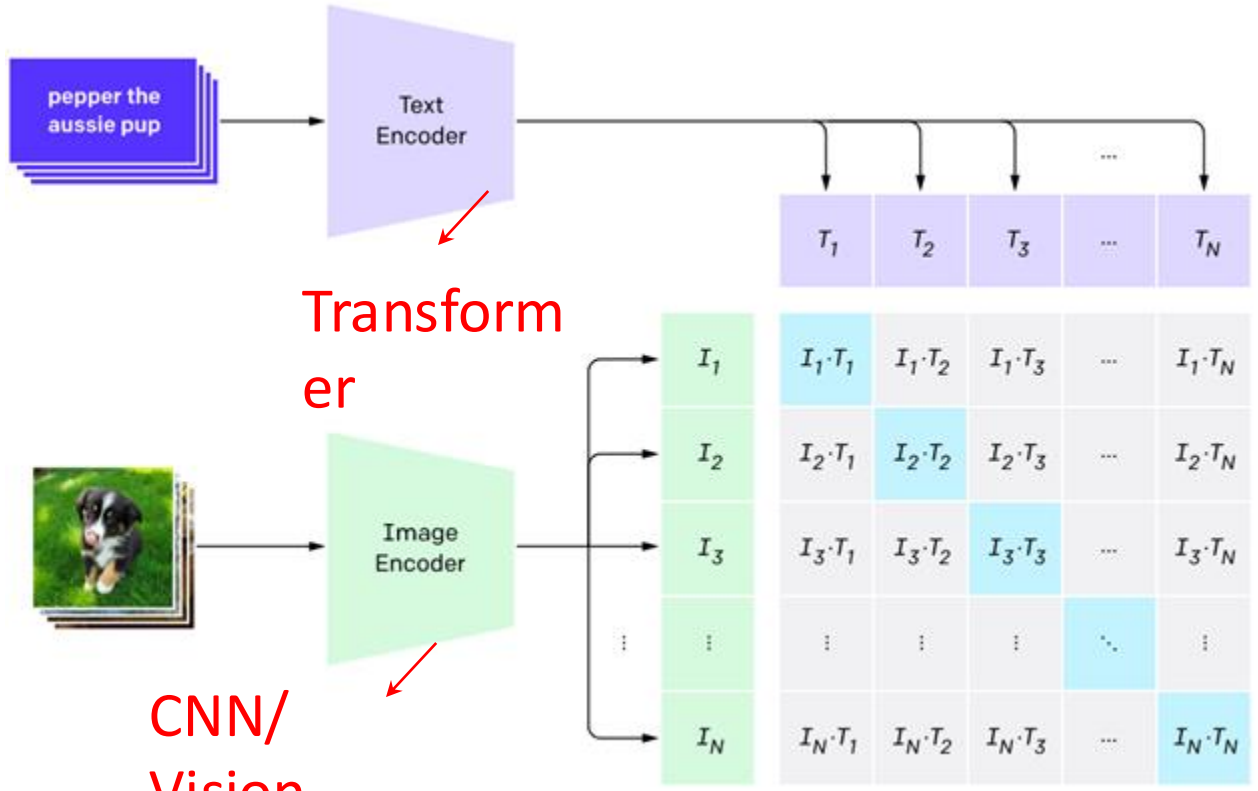
# Multimodal Learning: Vision-Language Model

- CLIP (Contrastive Language–Image Pre-training) by OpenAI in 2021
  - Train on 400 million image+text crawled from the internet
  - Proved that large-scale (noisy) vision-language pre–training can lead to great performance (with proper prompt engineering)
    - Any zero-shot downstream classification tasks
    - AI generated content (text-to-image generation)
      - DALL-E, etc.
- Why is this useful for Embodied AI?
  - We need to generalize our perception model to just recognizing discrete object classes
    - CIFAR-10/100, ImageNet-1K, ImageNet-21K
    - Finite set of vocabularies


* https://openai.com/blog/clip/

# Multimodal Learning: Vision-Language Model

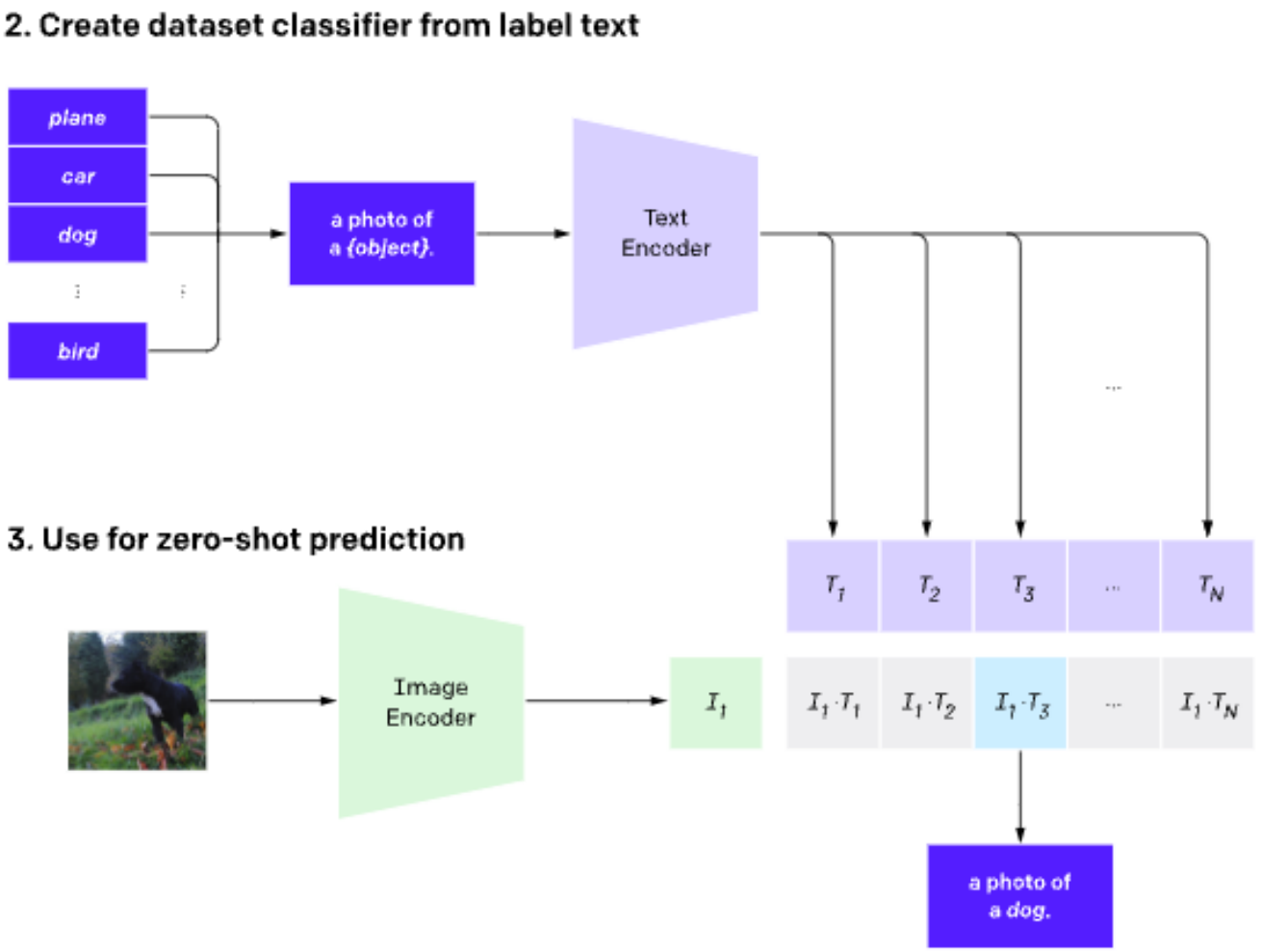
- CLIP (Contrastive Language–Image Pre-training) by OpenAI in 2021
  - Two Key Designs
    - Contrastive pre-training
      - Positive image-text + random negatives, train with CE loss (need large batch size to work - 32k)




* https://openai.com/blog/clip/

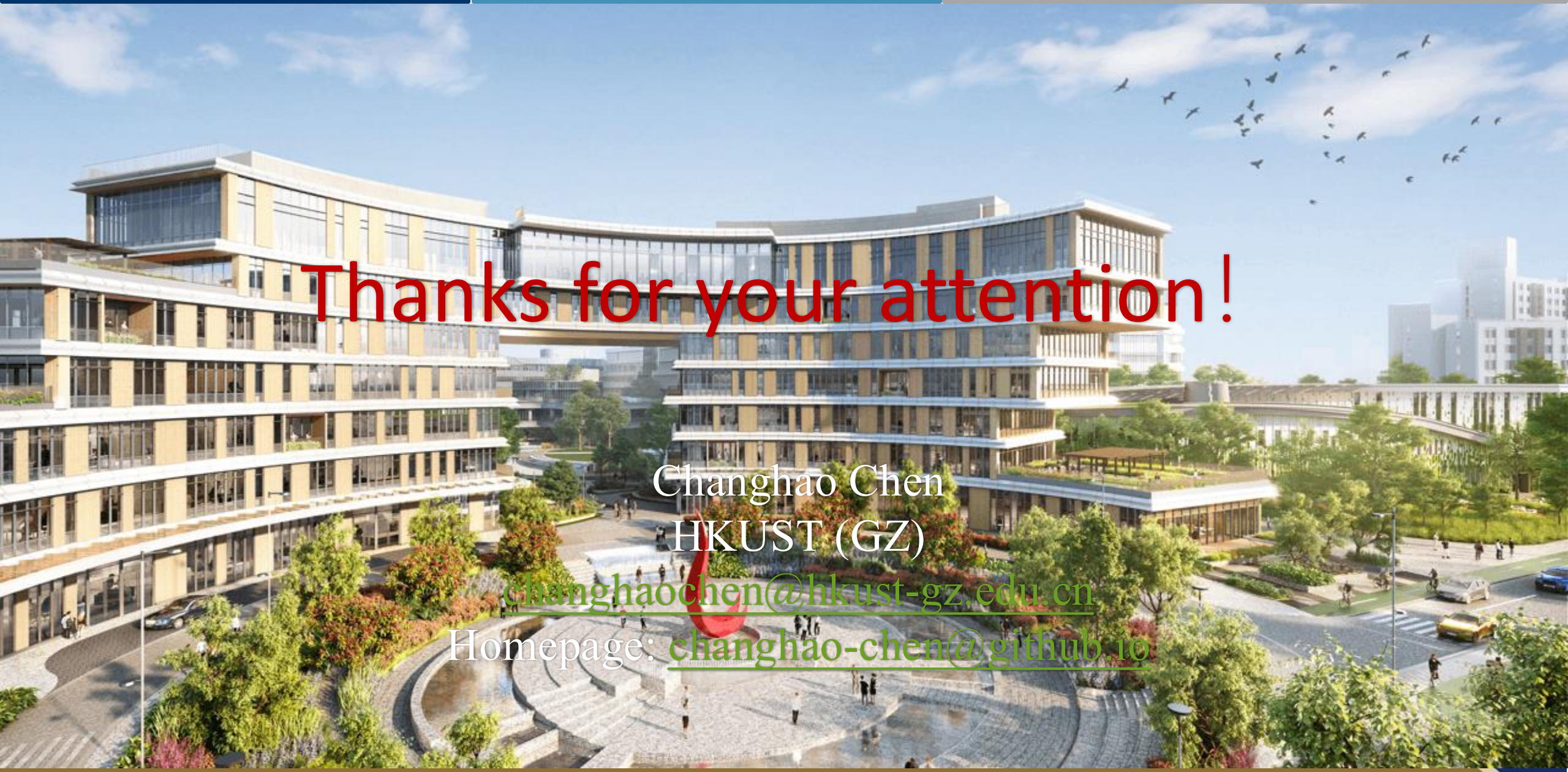
# Multimodal Learning: Vision-Language Model

- CLIP (Contrastive Language–Image Pre-training) by OpenAI in 2021
  - Two Key Designs
    - Contrastive pre-training
    - Prompt Engineering
      - "A photo of [class]"
      - Simply add prefix text
      - Improves zero-shot



# Reading Material

- Batch Normalization - Layer Normalization
  - For CNN and Transformer training
  - Learnable normalization layers
  - Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015
  - Layer Normalization. 2016
- Image Segmentation Using Deep Learning: A Survey
- ChatGPT for Robotics: Design Principles and Model Abilities
  - https://www.microsoft.com/en-us/research/group/autonomous-systems-group-robotics/articles/chatgpt-for-robotics/



# Thanks for your attention!

Changhao Chen
HKUST (GZ)
changhaochen@hkust-gz.edu.cn
Homepage: changhao-chen@github.io