PSET 3 Part 1 + Neural Nets

Krishnan Srinivasan

CS231A

05/10/2024

Midterm

Will grade midterm ASAP - by mid-end of next week

Will skip lecture recap this week

Next week flipped classroom:

- Prof Bohg will record lecture for you to watch online
- In class, we will review some of the slides with PollEv questions for you to respond to in person
- Please bring questions!
- Krishnan on Mon, Congyue on Wed

PSET 3

Space carving

Representation Learning

Supervised Monocular Depth Estimation

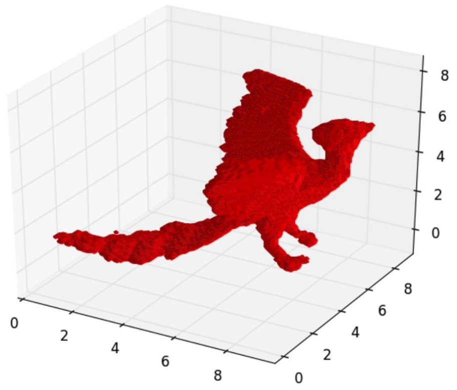
Space Carving

Objective:

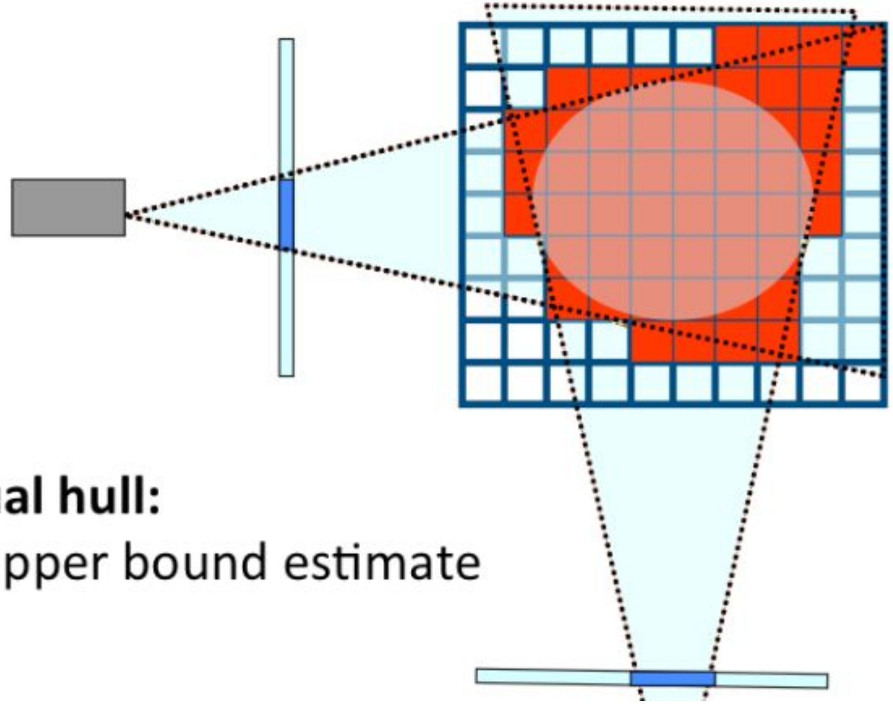
- Implement the process of space carving.

Lectures:

- Active Stereo & Volumetric Stereo

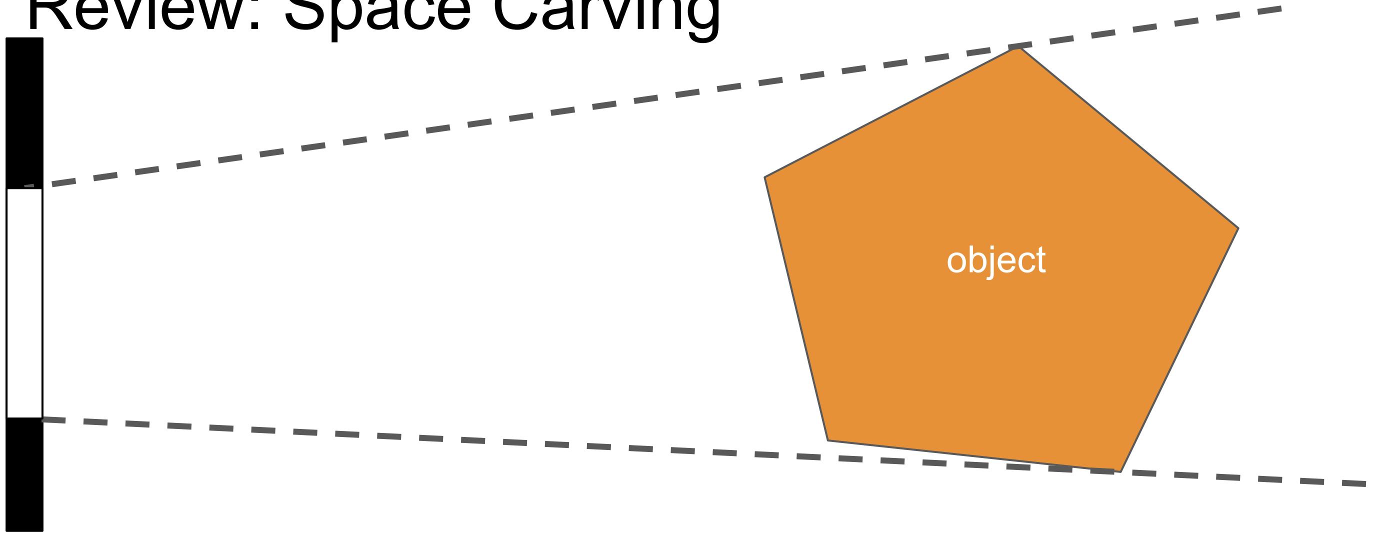


Review: Space Carving

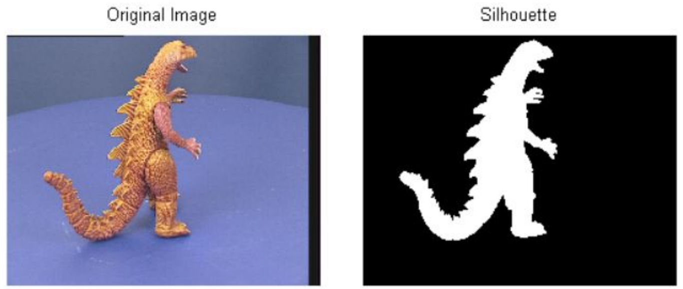


Visual hull:
an upper bound estimate

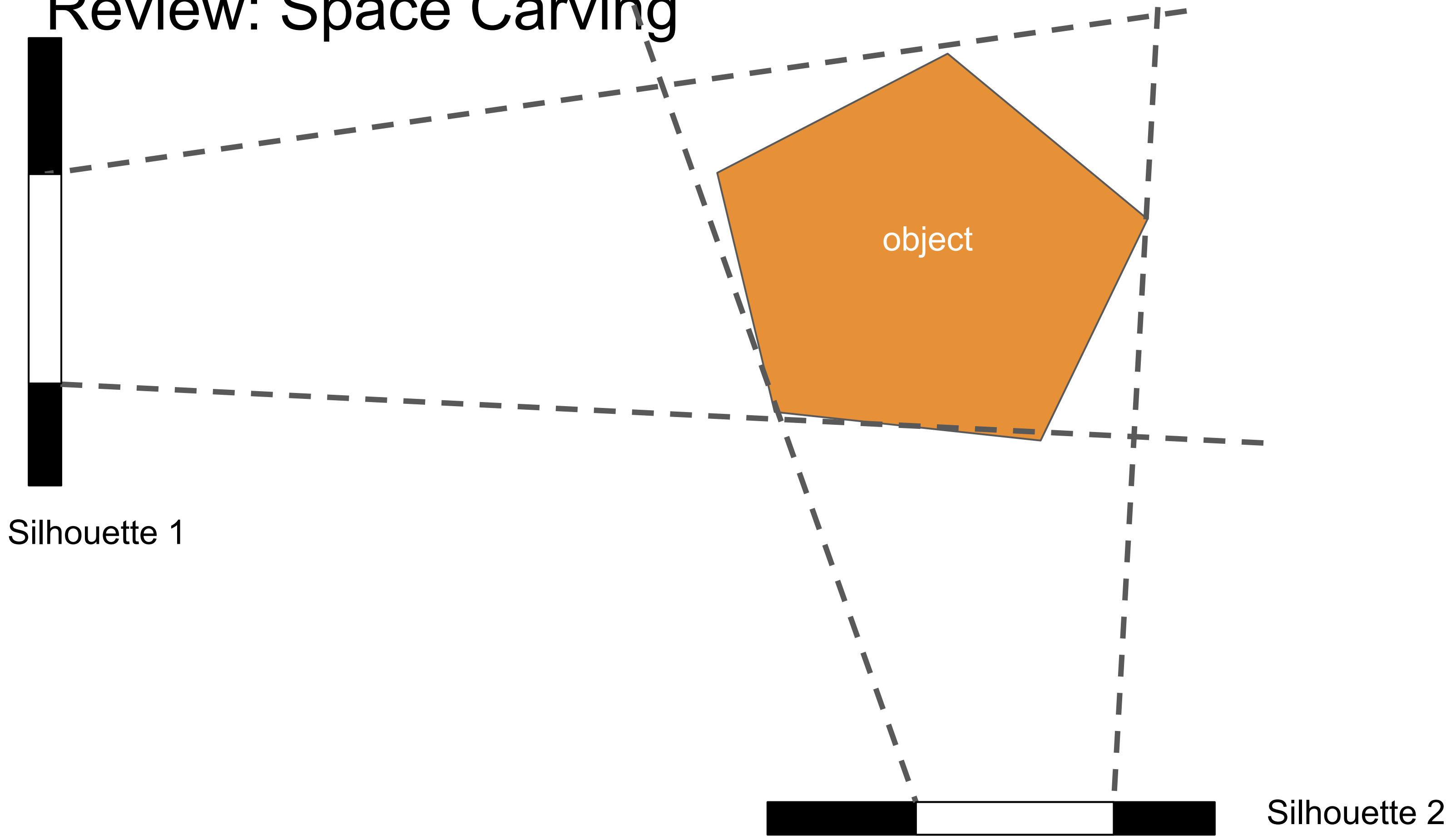
Review: Space Carving



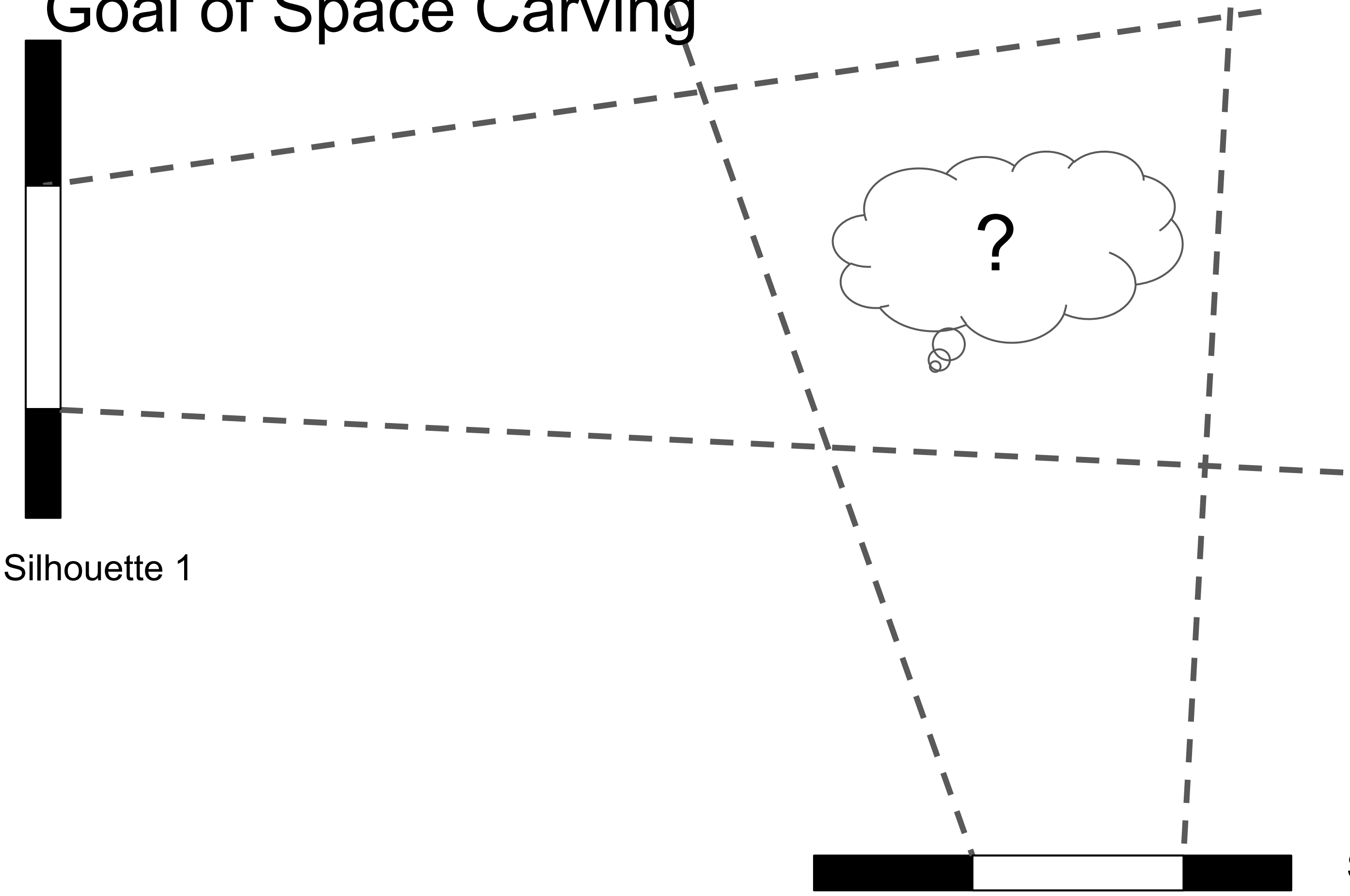
Silhouette 1



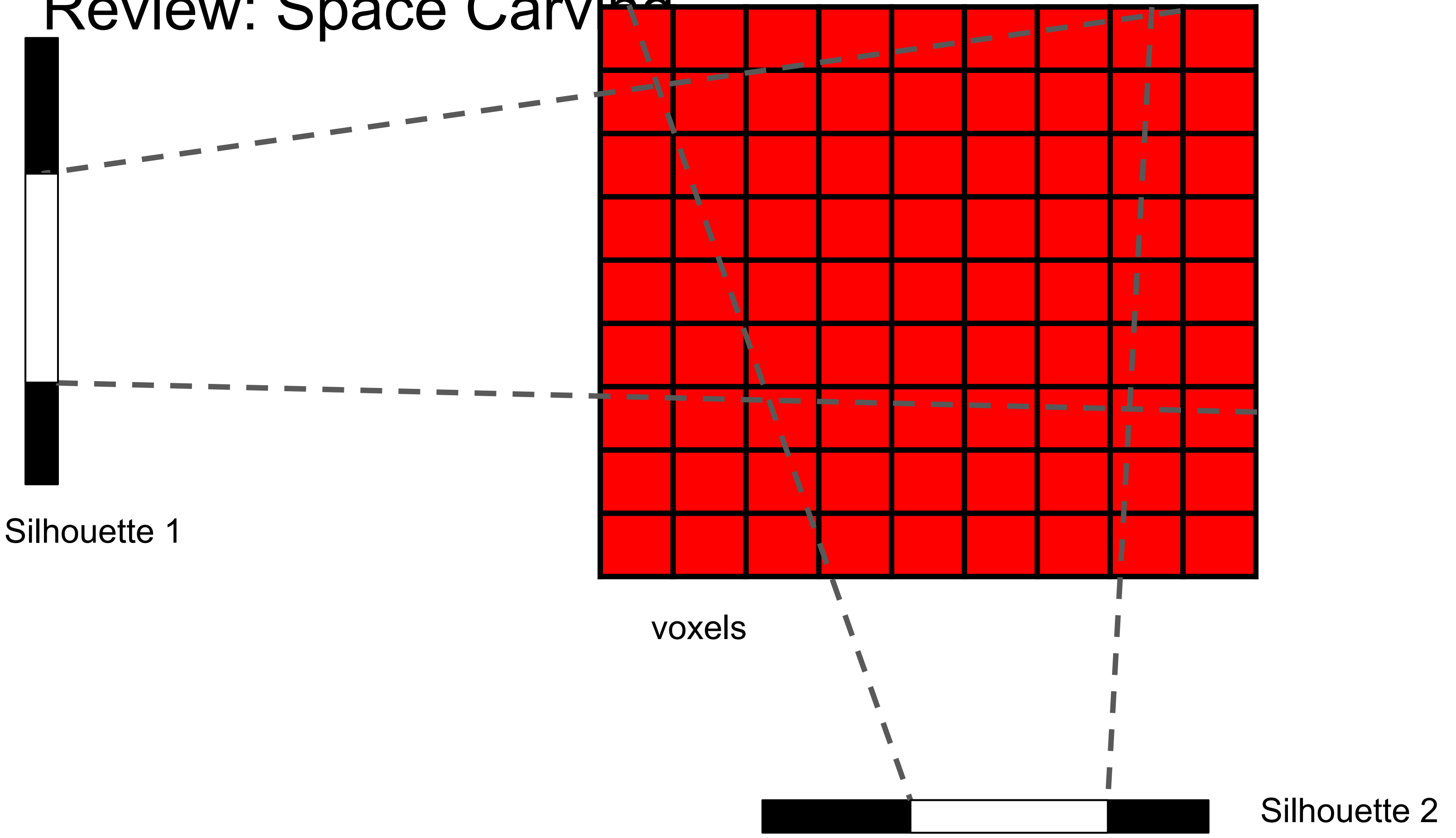
Review: Space Carving



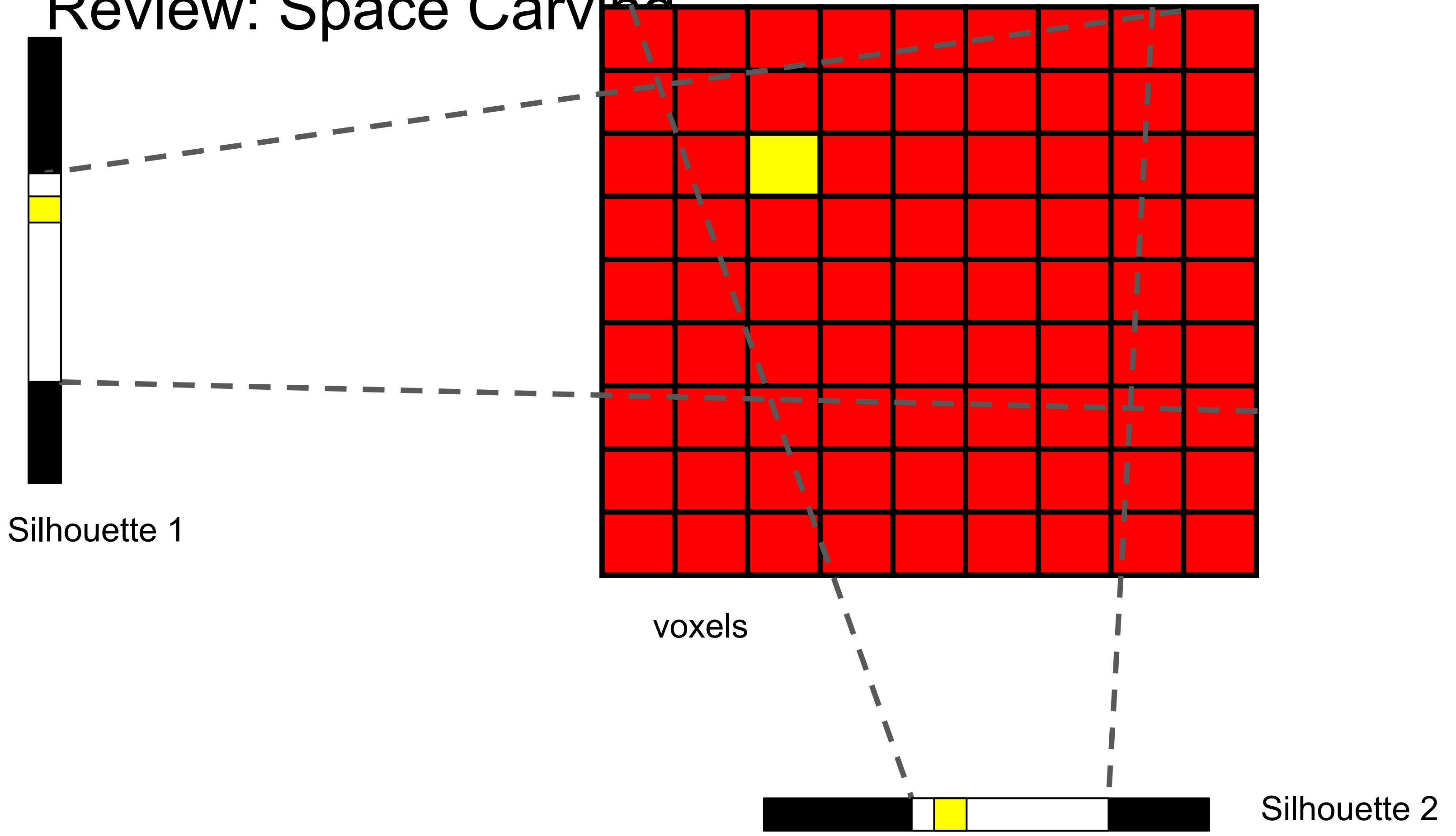
Goal of Space Carving



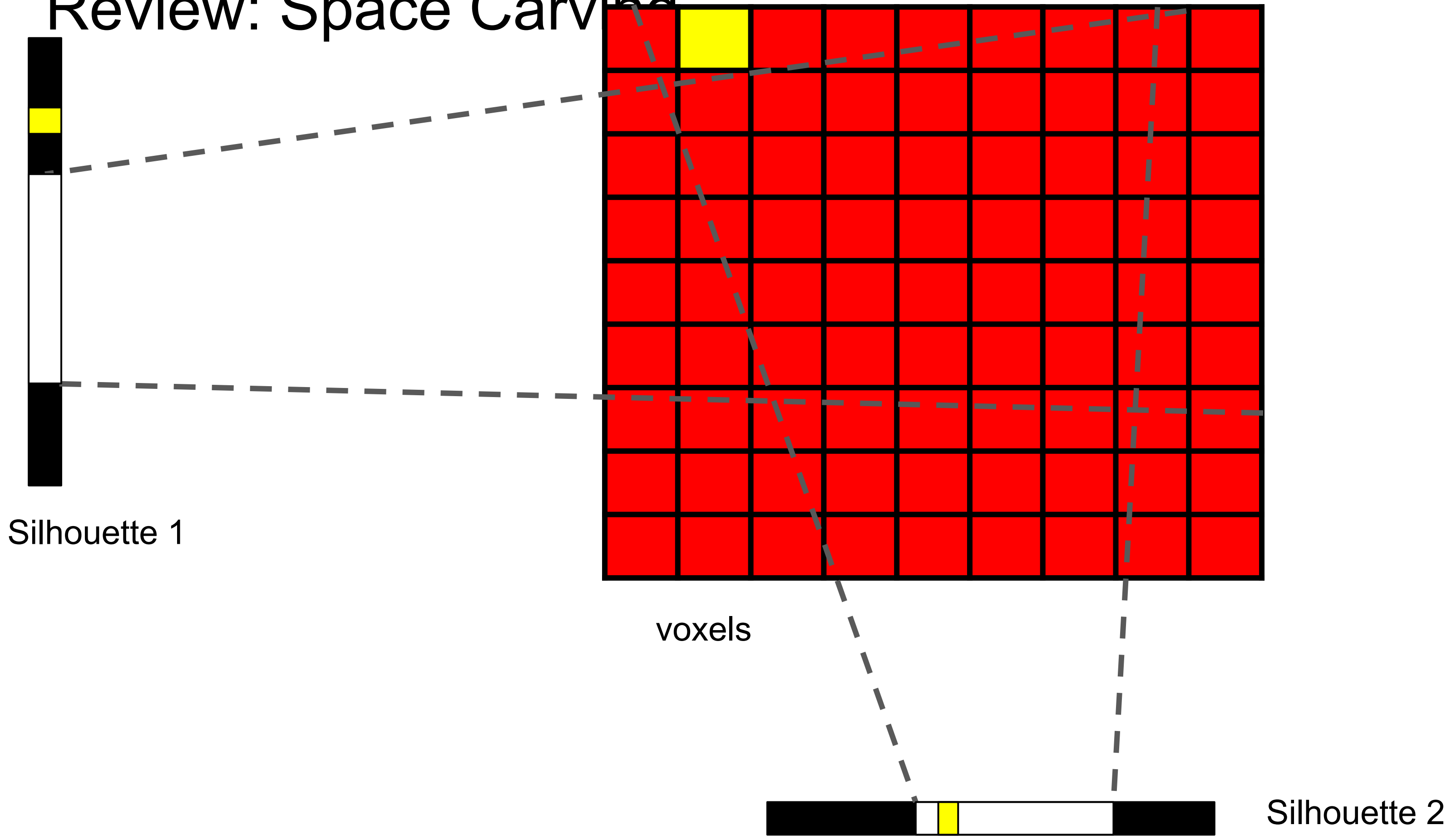
Review: Space Carving



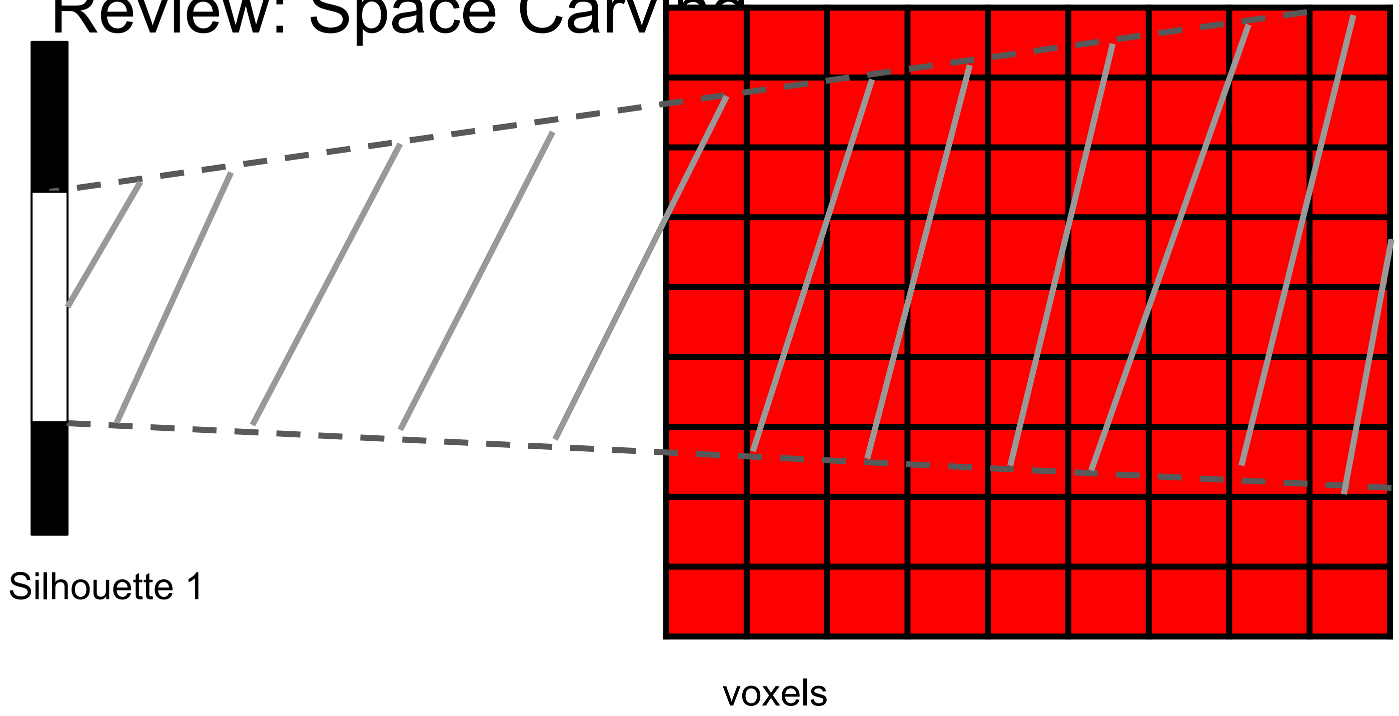
Review: Space Carving



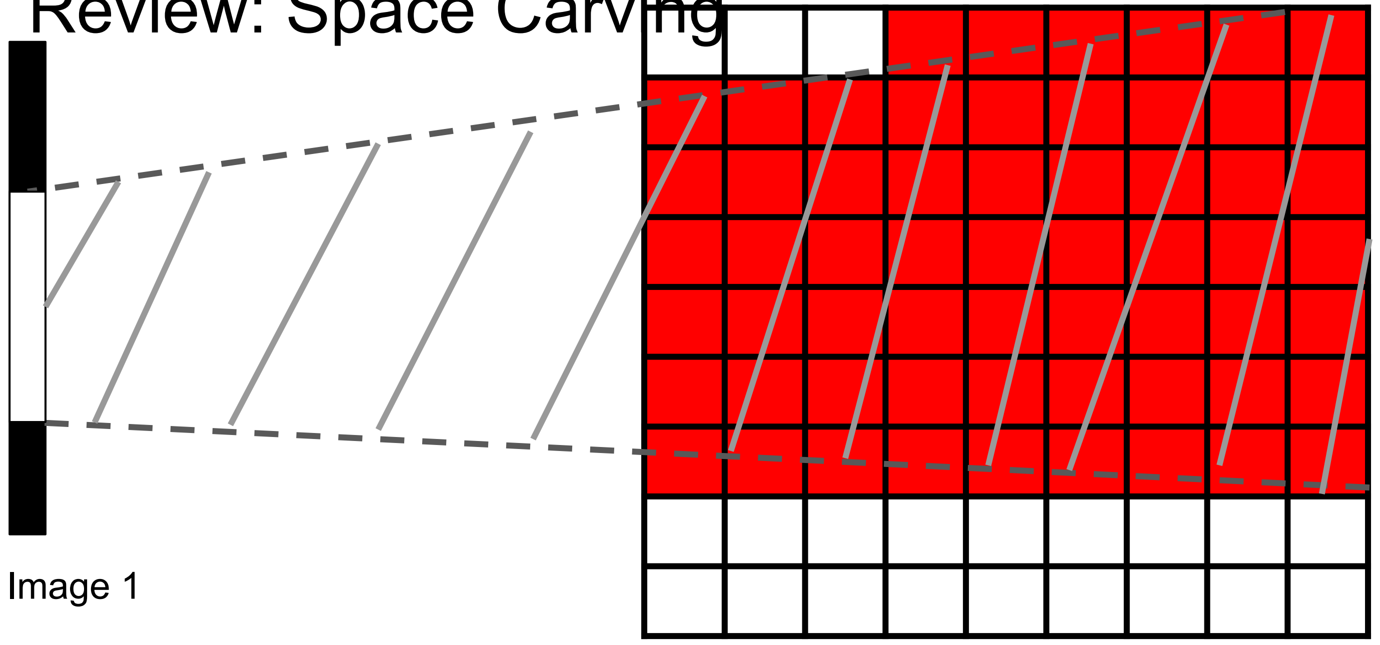
Review: Space Carving



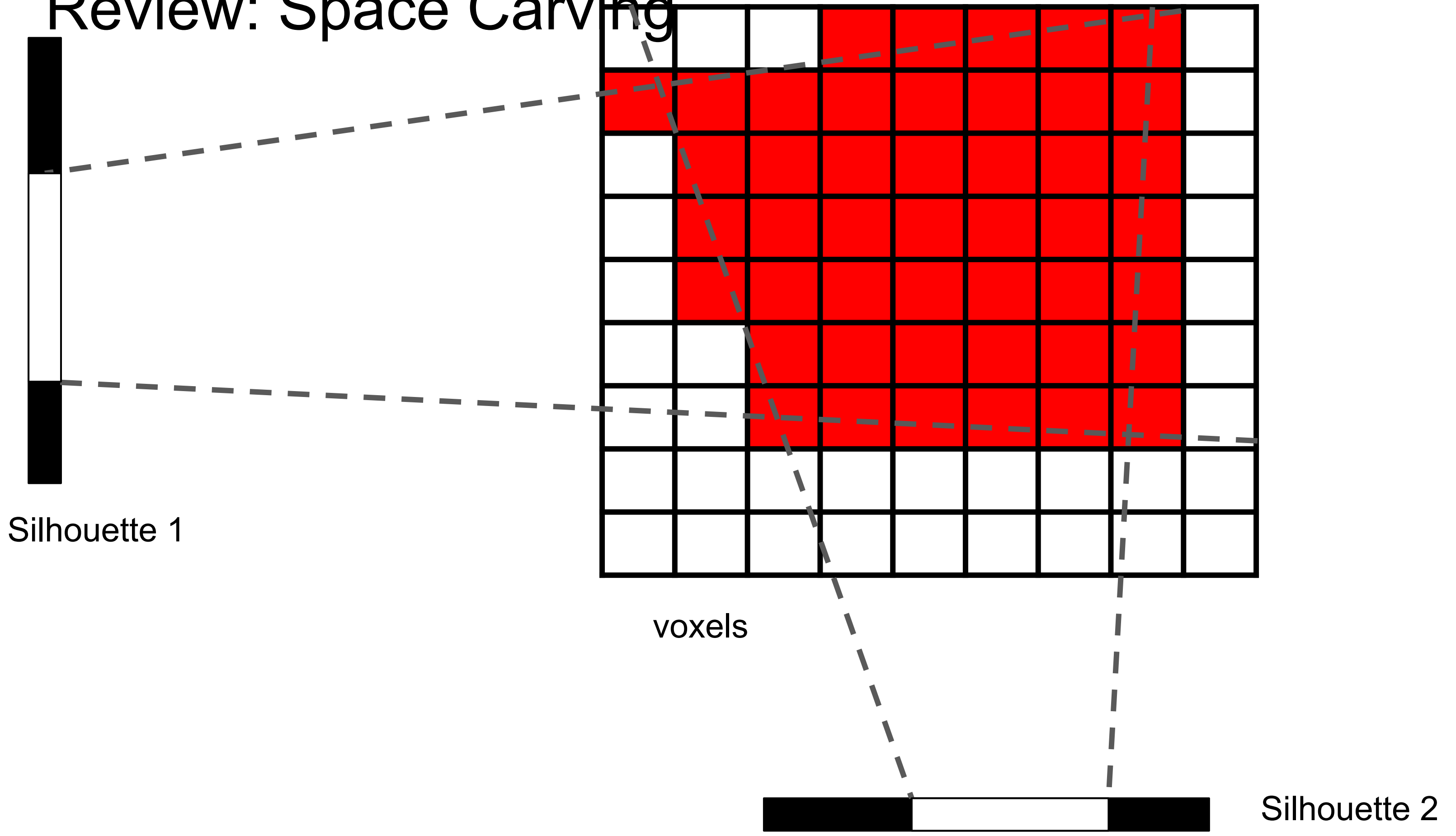
Review: Space Carving



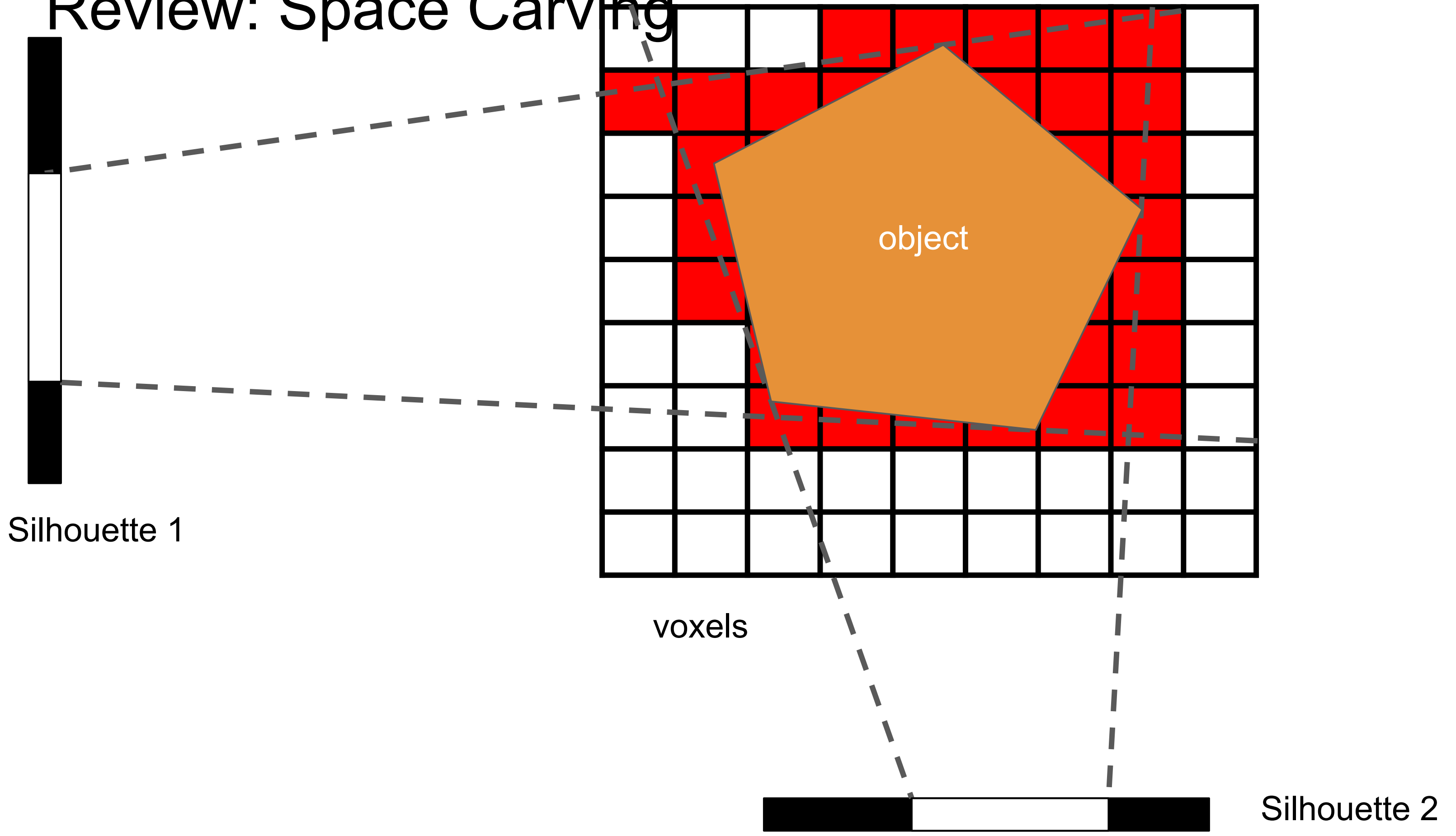
Review: Space Carving



Review: Space Carving



Review: Space Carving



Space carving - overview

Steps:

- Estimate silhouettes of images (could be based on some heuristics, e.g. color)
- Form the initial voxels as a cuboid
- Iterate over cameras and remove the voxels which project to the dark part of each silhouette

Space carving - (a) (b) (c)

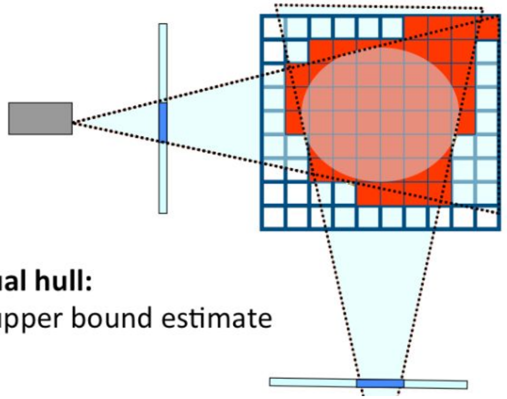
Steps:

- Estimate silhouettes of images (could be based on some heuristics, e.g. color)
- Form the initial voxels as a cuboid
    - *You may find these functions useful:* ***np.meshgrid****, np.repeat, np.tile*
    - Also boolean indexing*, ie keep = (x>=0) & (x<=w) & (y>=0) & (y<=h)*
    - *keep = [idx for idx, val in enumerate(keep) if val]*
    - *x = x[keep]*
    - *y = y[keep]*
- Iterate over cameras and remove the voxels which project to the dark part of each silhouette
    - *Question: What will the voxels look like after the first, second, … iteration?*

Space carving - (a) (b) (c)

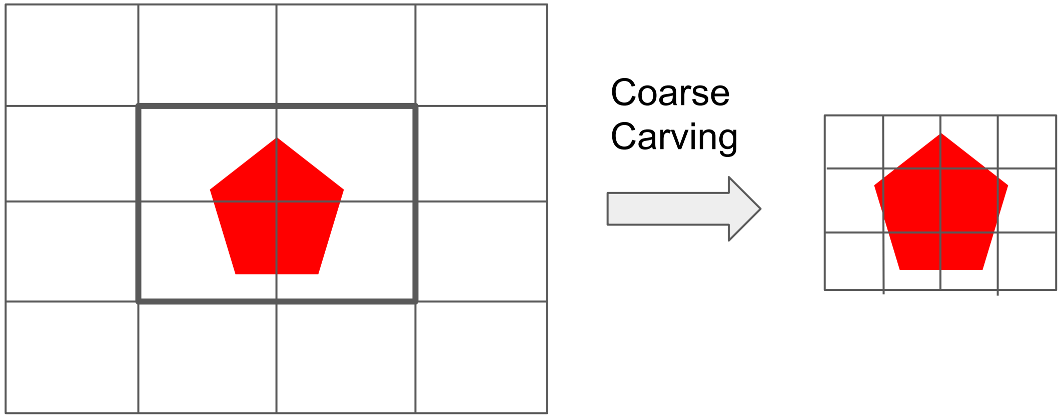
Steps:

- Iterate over cameras and remove the voxels which project to the dark part of each silhouette
  - *Question: What will the voxels look like after the first, second, … iteration?*

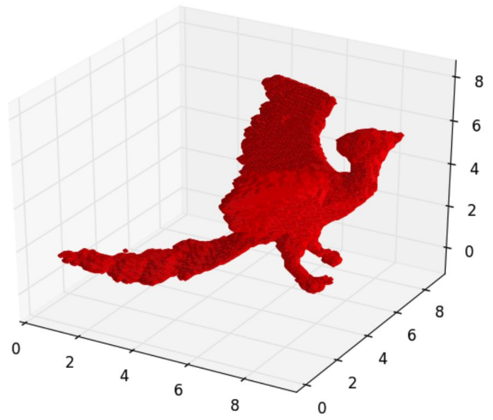


Space carving - (d)

What if we first find the rough size of the object instead of just looking at camera positions?



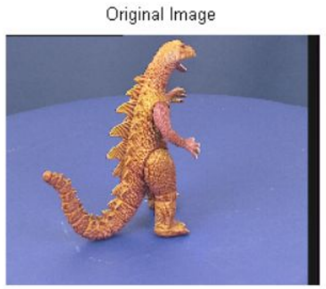
Final Output

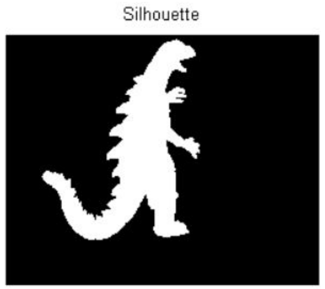


Space carving - (e)

Steps:

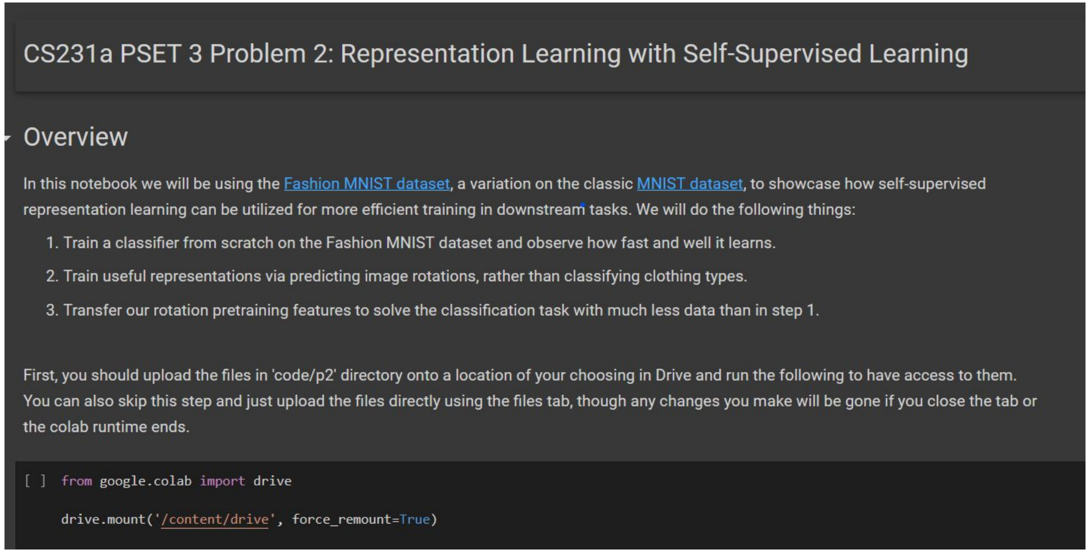
- Estimate silhouettes of images (could be based on some heuristics, e.g. color)
    - Problem: The quality of silhouettes is not perfect.
    - The silhouette from each camera is not perfect, but the result is ok. Why?
    - Experiment: Use only a few of the silhouettes.





PSET 3 - Colab

Need colab for parts 2,3, and 4.



CS231a PSET 3 Problem 2: Representation Learning with Self-Supervised Learning

Overview

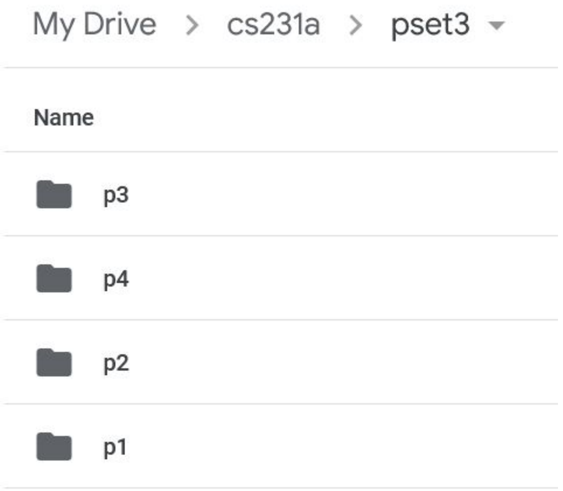
In this notebook we will be using the Fashion MNIST dataset, a variation on the classic MNIST dataset, to showcase how self-supervised representation learning can be utilized for more efficient training in downstream tasks. We will do the following things:

1. Train a classifier from scratch on the Fashion MNIST dataset and observe how fast and well it learns.
2. Train useful representations via predicting image rotations, rather than classifying clothing types.
3. Transfer our rotation pretraining features to solve the classification task with much less data than in step 1.

First, you should upload the files in 'code/p2' directory onto a location of your choosing in Drive and run the following to have access to them. You can also skip this step and just upload the files directly using the files tab, though any changes you make will be gone if you close the tab or the colab runtime ends.

```
from google.colab import drive

drive.mount('/content/drive', force_remount=True)
```



My Drive > cs231a > pset3

Name

p3

p4

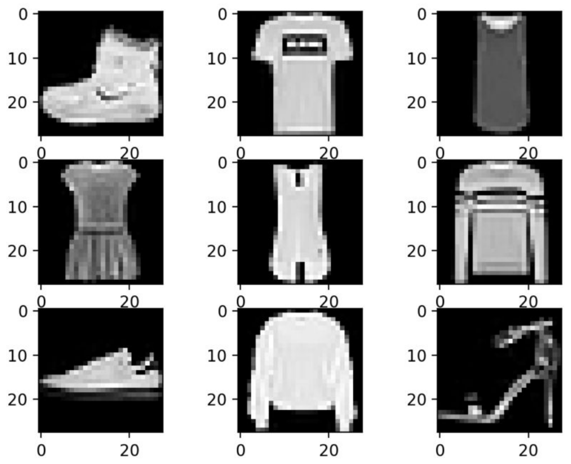
p2

p1

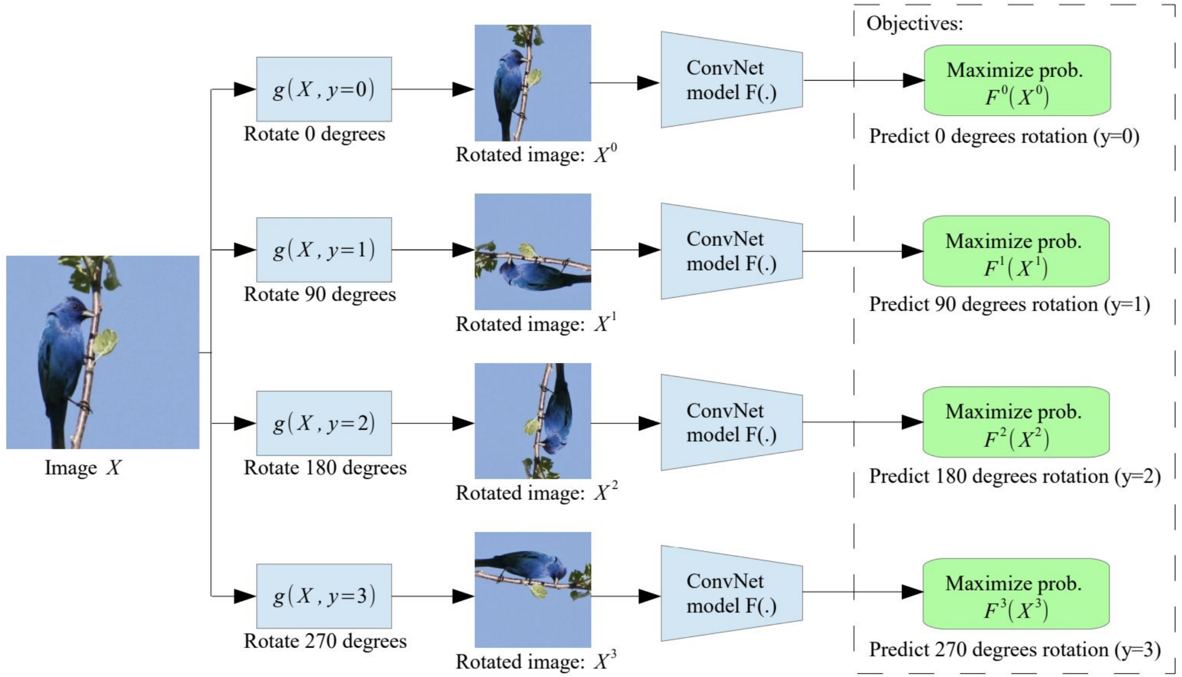
Problem 2 - Representation Learning

In this notebook, we will be using the Fashion MNIST dataset to showcase how self-supervised representation learning can be utilized for more efficient training in downstream tasks. We will do the following things:

1. Train a classifier from scratch on the Fashion MNIST dataset and observe how fast and well it learns
2. Train useful representations via predicting image rotations, rather than classifying images
3. Transfer our rotation pretraining features to solve the classification task with much less data than in step 1



Unsupervised Representation Learning by Predicting Image-Rotations (ICLR '18)



Problem 2 - Representation Learning

PyTorch Training basics (training.py):

- Use **torch.DataLoader** and **Dataset** to load datasets and make batches
- Create layers using **torch.nn** module
- Use **torch.optim** to create an **SGD** Optimizer take gradient steps
- Manipulating **torch.Tensor:**
  - use t.cpu() to move from GPU -> CPU, use t.cuda() for CPU -> GPU

Problem 2 - Representation Learning

`MNISTDatasetWrapper(Dataset)`

- __init__: load pct% of images from processed .pt file
- __getitem__: randomly rotate an image from self.imgs. **Hint:** use PIL.Image.rotate to rotate image, and then return to torch.Tensor type
- **Hint:** Use torch.tensor(rotation_idx).long() to generate rotation labels

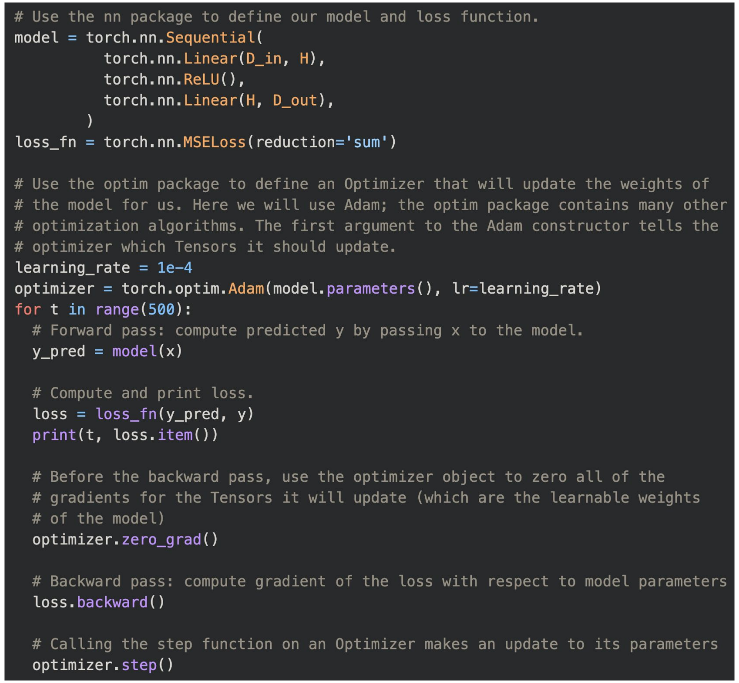
nn.Sequential(...)

- Creates a stack of layers that pass input data through a model
- nn.Linear(...) layers form weights and biases for a single layer

Problem 2 - Representation Learning

Training example (from pytorch-examples repo)

- **opt.zero_grad** to zero gradients before update
- **loss.backward** to backpropagate gradients
- **opt.step** to update model params



```python
# Use the nn package to define our model and loss function.
model = torch.nn.Sequential(
          torch.nn.Linear(D_in, H),
          torch.nn.ReLU(),
          torch.nn.Linear(H, D_out),
        )
loss_fn = torch.nn.MSELoss(reduction='sum')

# Use the optim package to define an Optimizer that will update the weights of
# the model for us. Here we will use Adam; the optim package contains many other
# optimization algorithms. The first argument to the Adam constructor tells the
# optimizer which Tensors it should update.
learning_rate = 1e-4
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
for t in range(500):
  # Forward pass: compute predicted y by passing x to the model.
  y_pred = model(x)

  # Compute and print loss.
  loss = loss_fn(y_pred, y)
  print(t, loss.item())

  # Before the backward pass, use the optimizer object to zero all of the
  # gradients for the Tensors it will update (which are the learnable weights
  # of the model)
  optimizer.zero_grad()

  # Backward pass: compute gradient of the loss with respect to model parameters
  loss.backward()

  # Calling the step function on an Optimizer makes an update to its parameters
  optimizer.step()
```

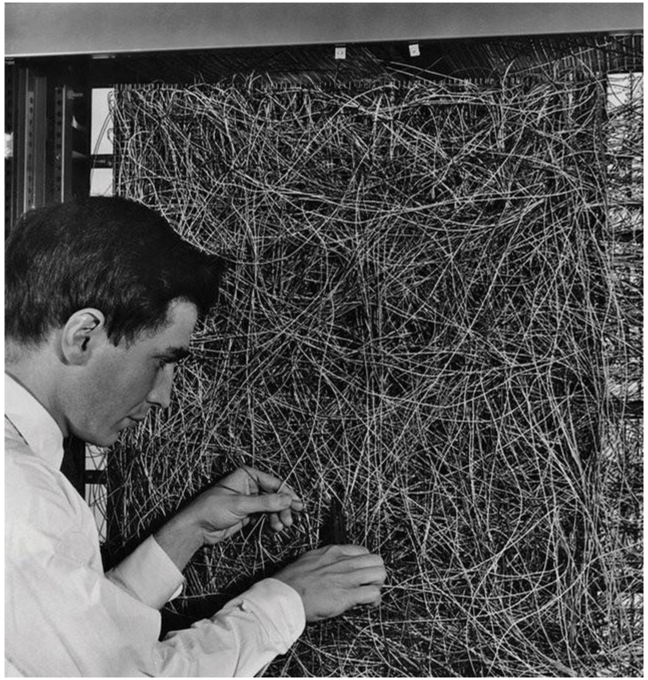
Intro to Neural Networks

- Background and Applications
- Fully-connected Neural Networks (MLP)
- Convolutional Neural Networks (CNN)
- Backpropagation Algorithm
- PyTorch Example

Background

History

- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer

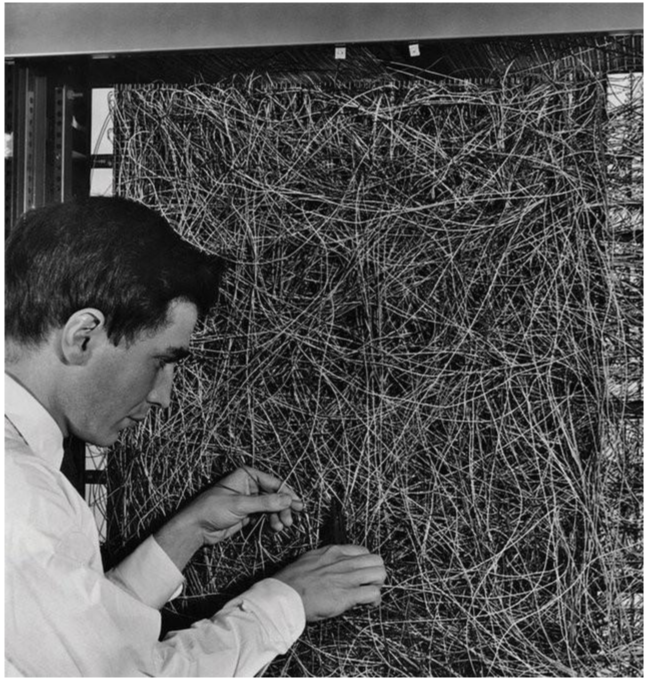


Tuning hyperparameters used to take **a lot longer** in Rosenblatt's day

Background

History

- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer
- 1969: Multi-layer perceptron (early fully-connected neural networks) by Minksy and Papert

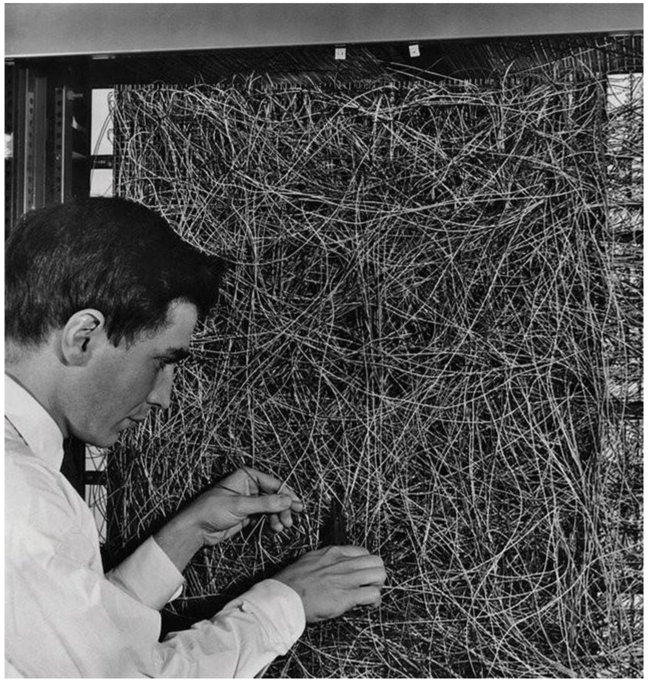


Tuning hyperparameters used to take **a lot longer** in Rosenblatt's day

Background

History

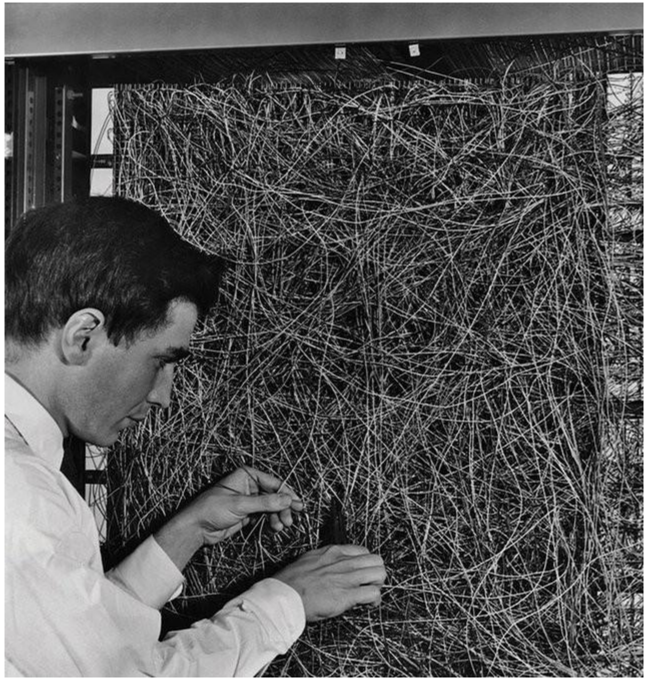
- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer
- 1969: Multi-layer perceptron (early fully-connected neural networks) by Minksy and Papert
- 1986: Rumelhart, Hinton, and Williams (and others) develop the backpropagation algorithm (BP)


Tuning hyperparameters used to take **a lot longer** in Rosenblatt's day

Background

History

- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer
- 1969: Multi-layer perceptron (early fully-connected neural networks) by Minksy and Papert
- 1986: Rumelhart, Hinton, and Williams (and others) develop the backpropagation algorithm (BP)
- 1989: LeCun et al. develop BP for Convolutional Neural Networks (CNNs), and introduce MNIST dataset

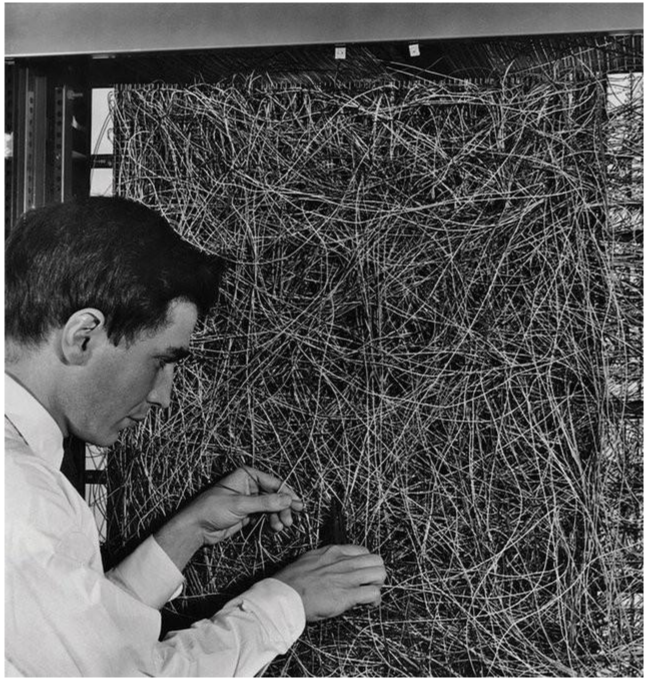


Tuning hyperparameters used to take
a lot longer in Rosenblatt's day

Background

History

- 1957: Frank Rosenblatt designs the Mark I Perceptron, an early learning-based computer
- 1969: Multi-layer perceptron (early fully-connected neural networks) by Minksy and Papert
- 1986: Rumelhart, Hinton, and Williams (and others) develop the backpropagation algorithm (BP)
- 1989: LeCun et al. develop BP for Convolutional Neural Networks (CNNs), and introduce MNIST dataset
- 2012: AlexNet uses GPUs to train CNNs fast enough to be practical



Tuning hyperparameters used to take
a lot longer in Rosenblatt's day

A bit of history:

ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

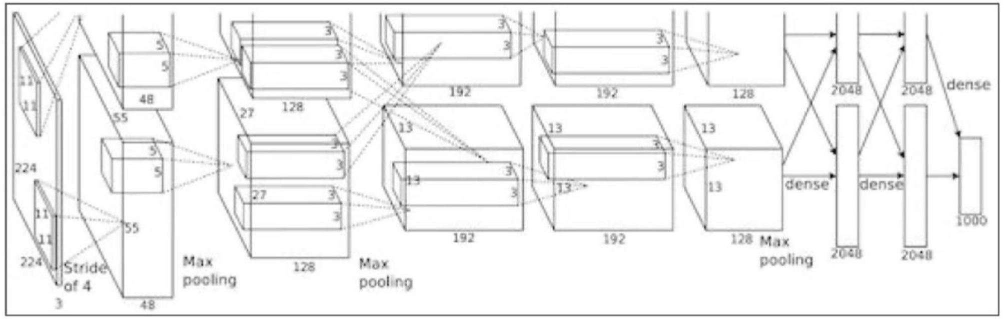


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

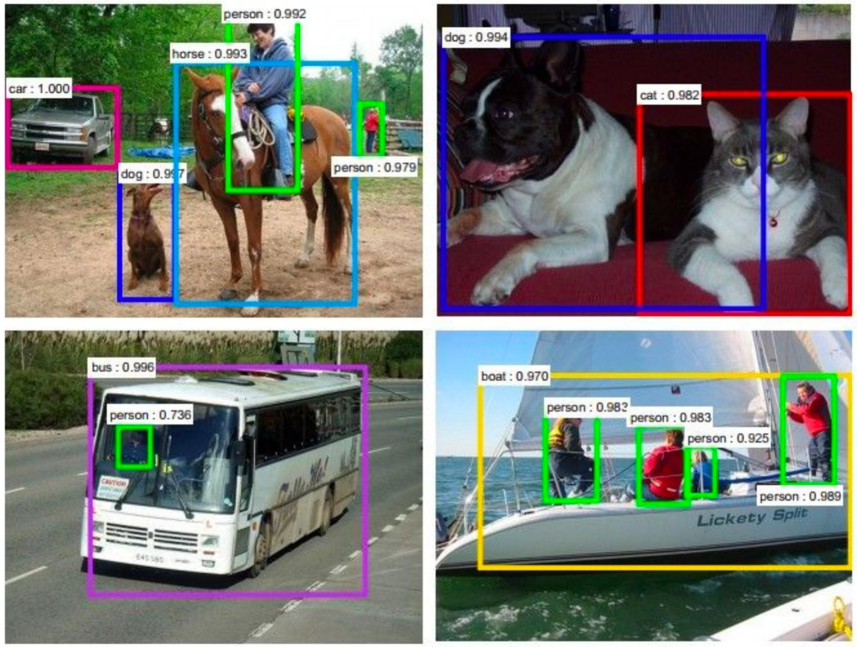
"AlexNet"

[A Brief History of Neural Nets and Deep Learning](#)

Following slides are borrowed from [CS231N Lecture 5](#)

Applications: Convolutional Networks

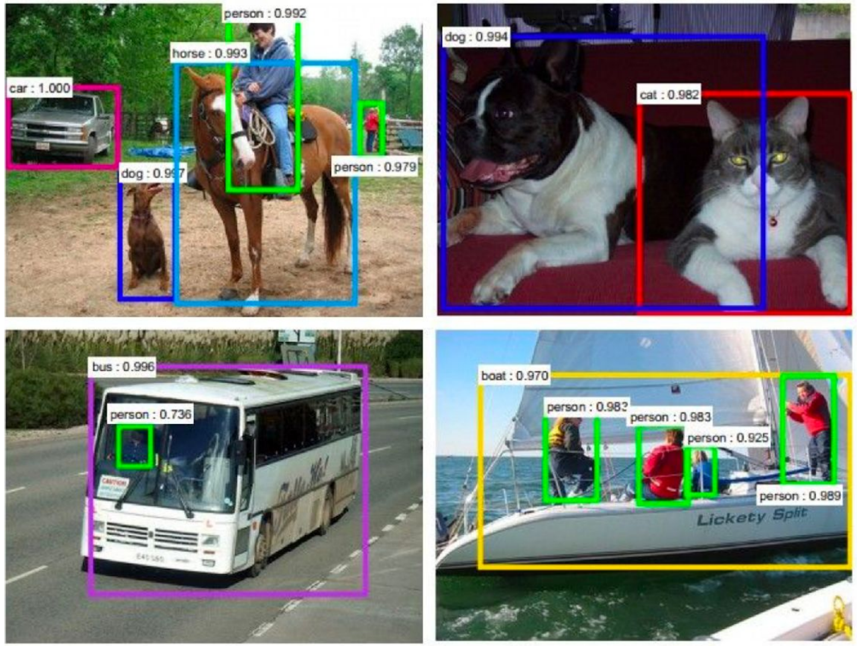
Detection



[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Applications: Convolutional Networks

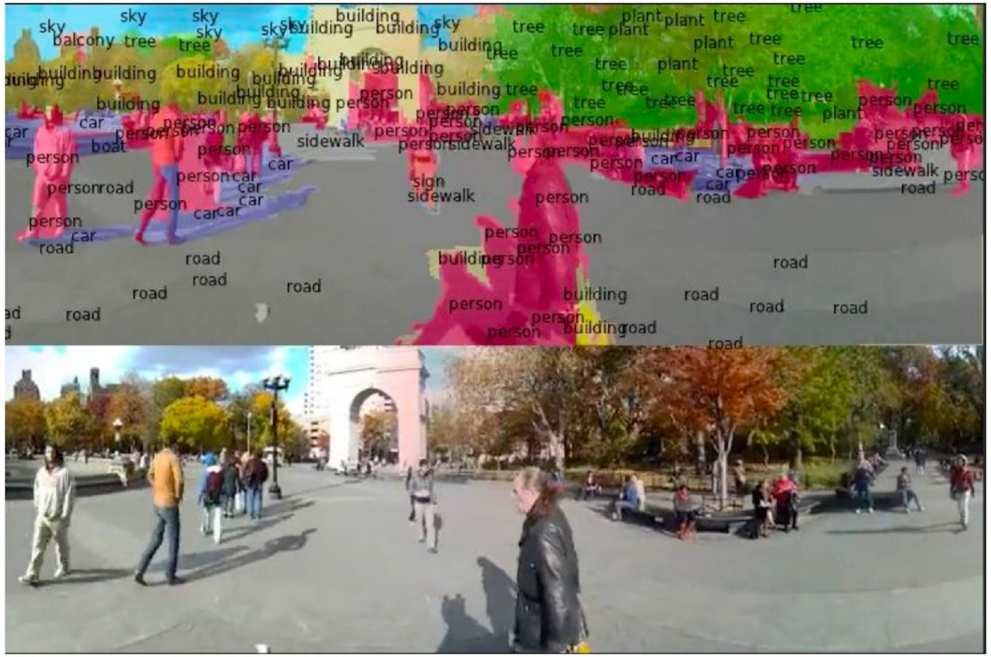
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

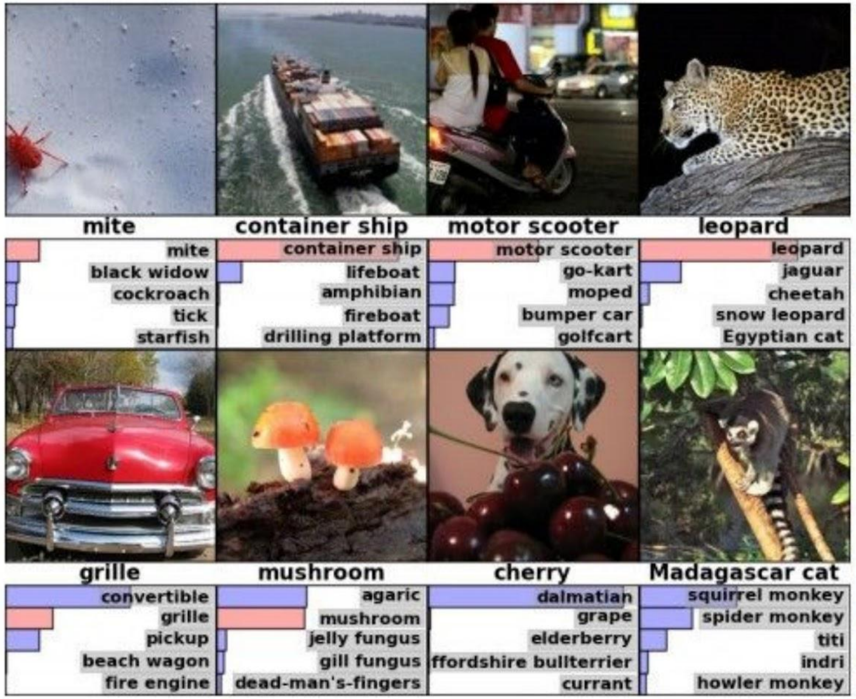
Segmentation



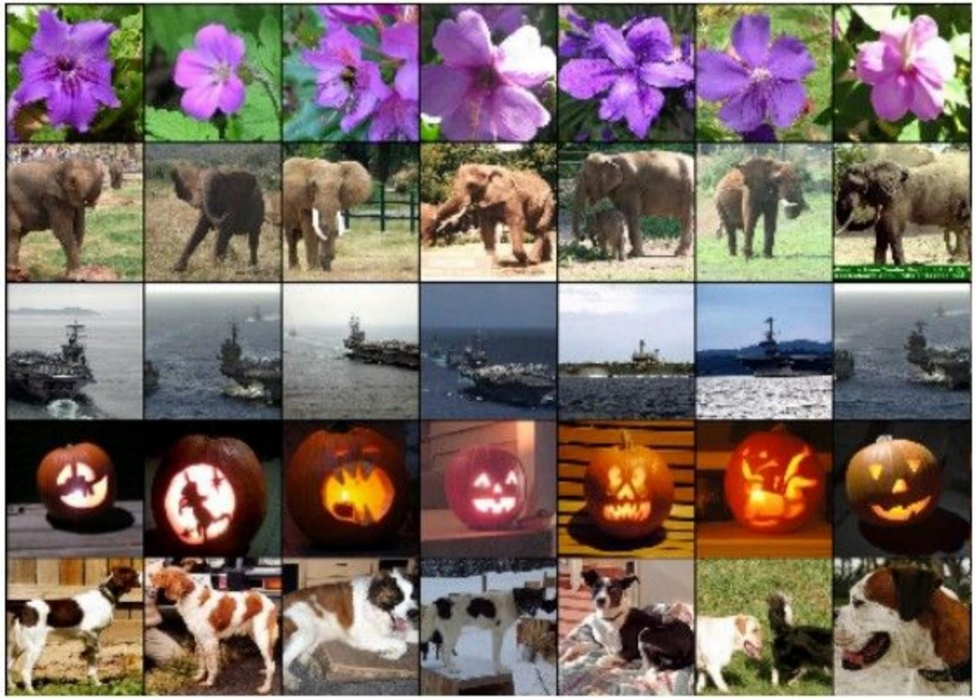
Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

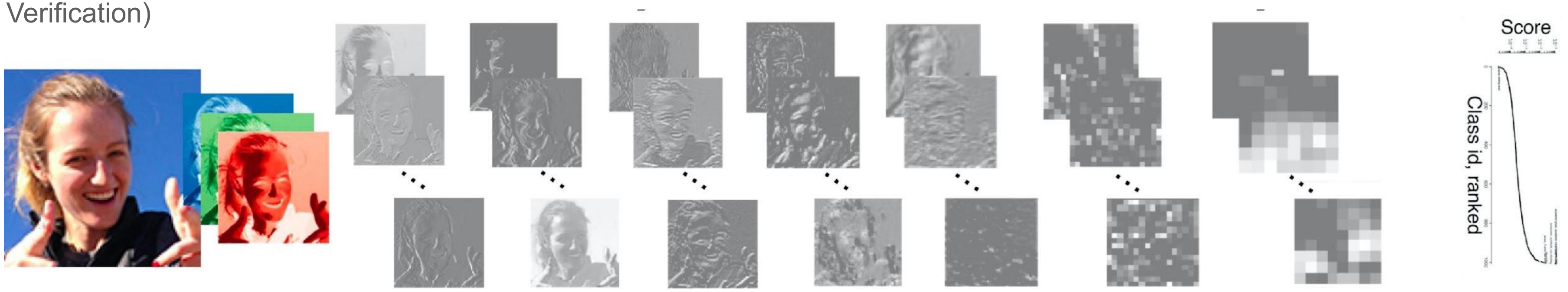
Classification



Retrieval



DeepFace (Face Verification)

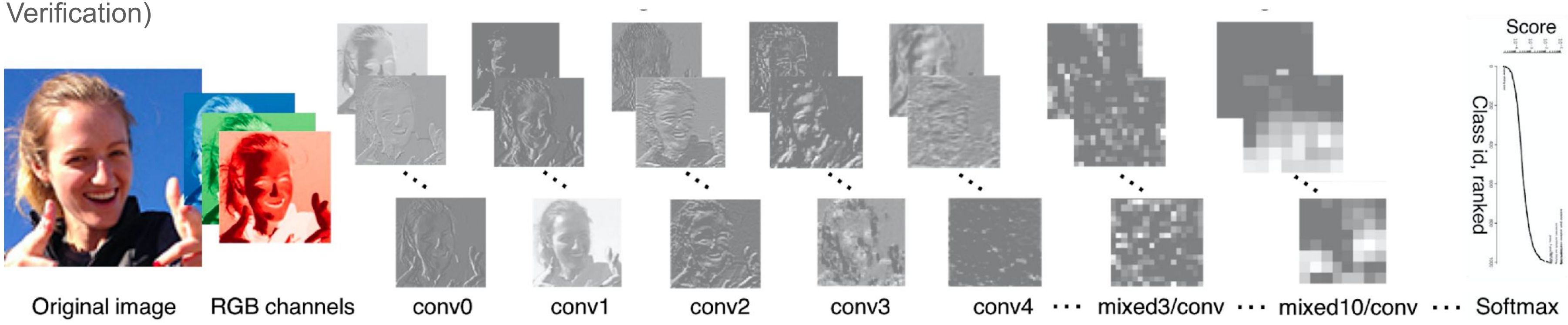


[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.

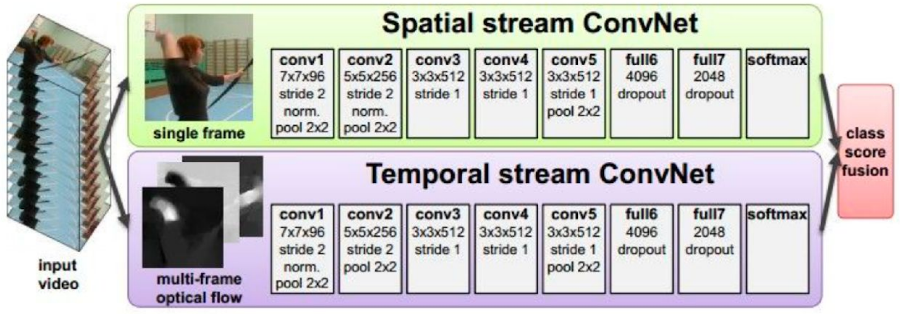
Two-Stream Convolutional Networks for Action Recognition in Videos

DeepFace (Face Verification)



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

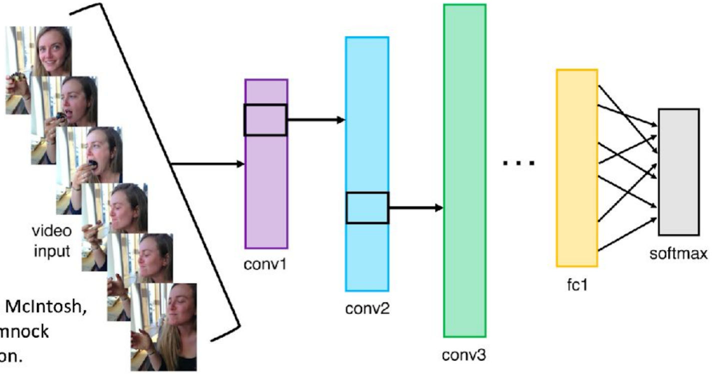
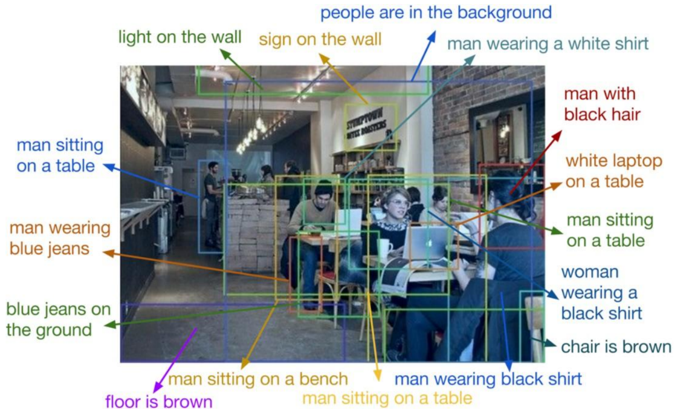
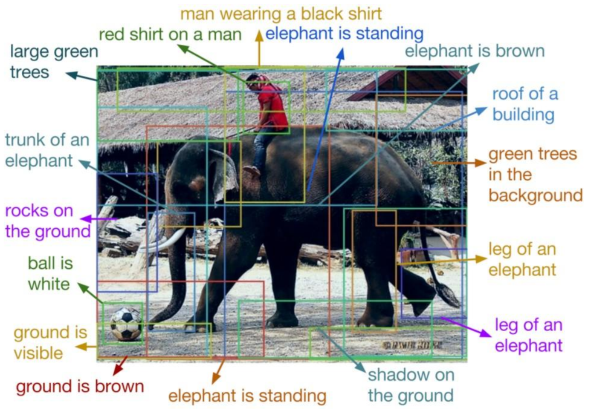


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

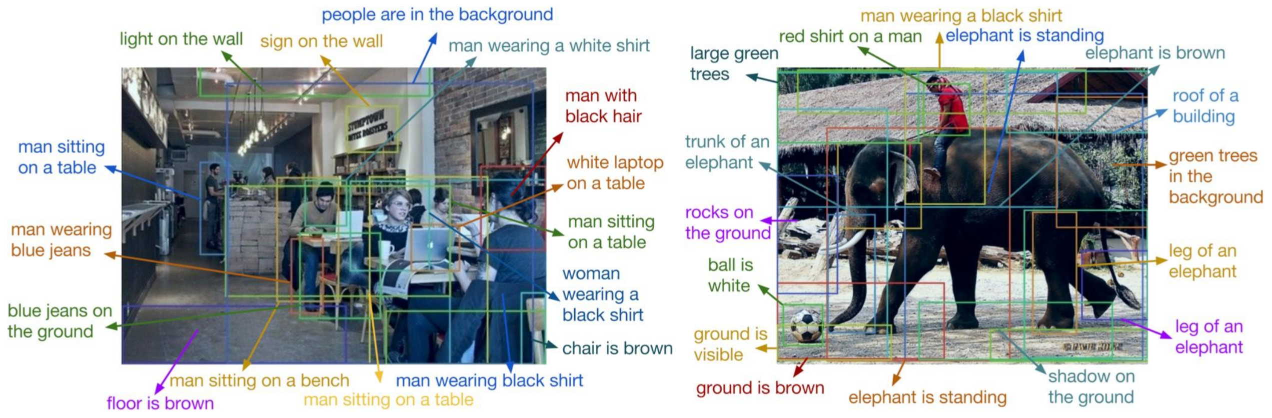
Two-Stream Convolutional Networks for Action Recognition in Videos

Dense Captioning [Johnson et al. 2016]

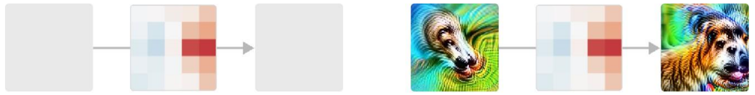




Dense Captioning [Johnson et al. 2016]



Visualizing Circuits [Voss et al. 2021]



Without context, these weights aren't very interesting.

With context, they show us how a head detector gets attached to a body.

FIGURE 3: Contextualizing weights.

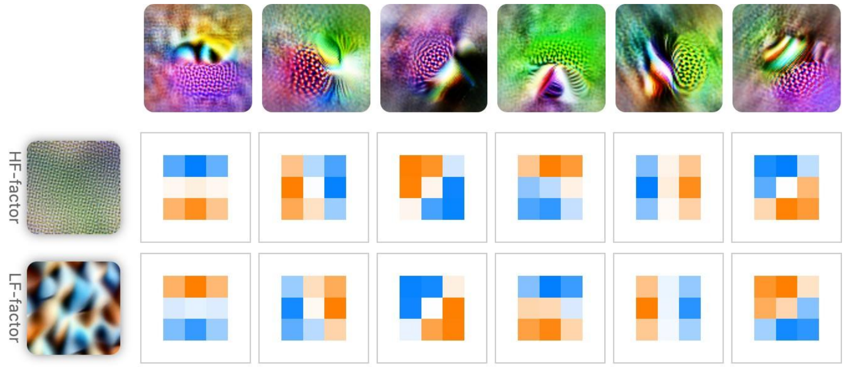
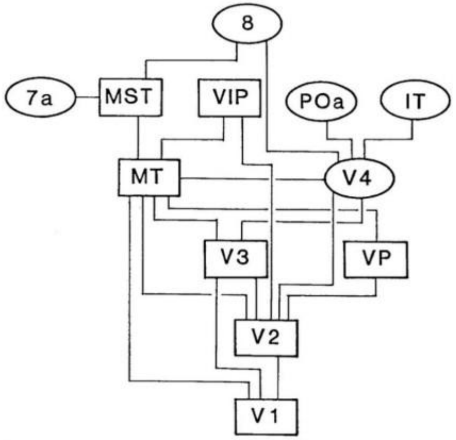
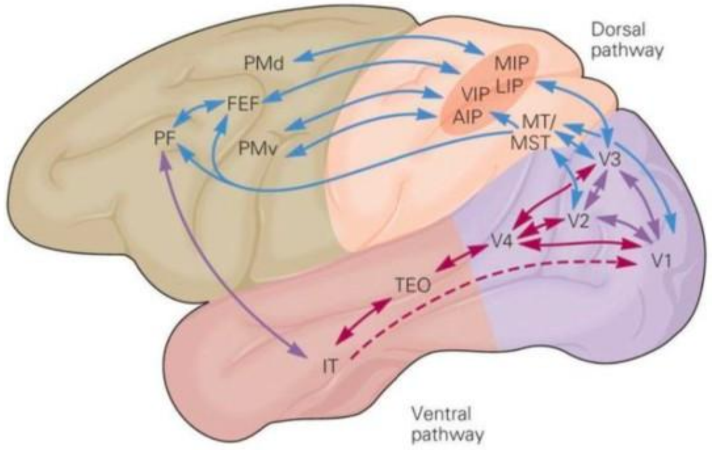


FIGURE 7: NMF factorization on the weights (excitatory and inhibitory) connecting six high-low frequency detectors in InceptionV1 to the layer `conv2d2`.

Background

Signal Relay

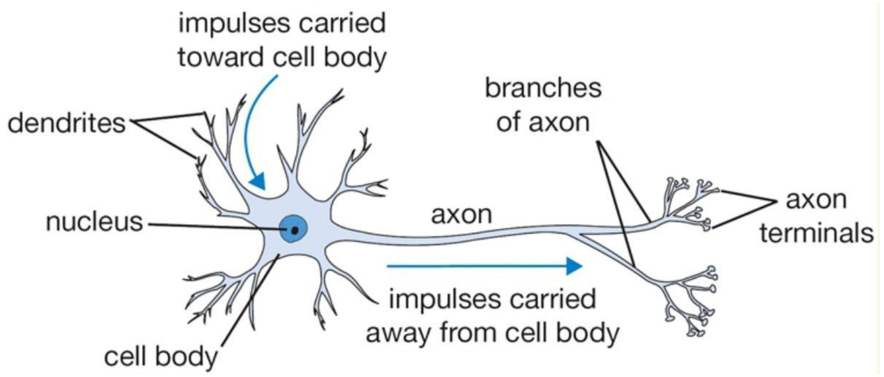


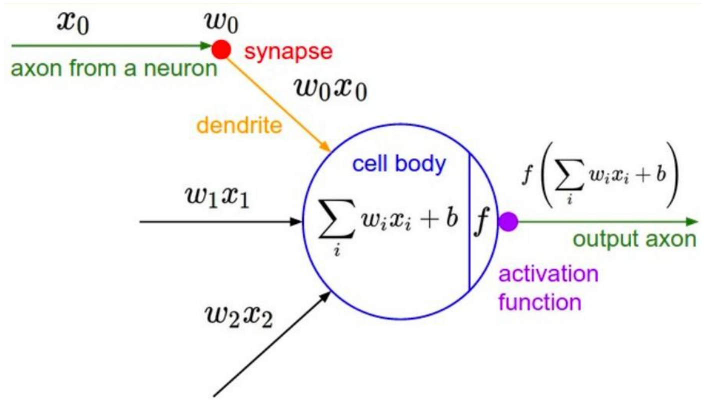


Starting from V1 primary visual cortex, visual signal is transmitted upwards, forming a more complex and abstract representation at every level

Foundations of Vision, Brian A. Wandell (1995)

Fully-Connected Neural Networks


impulses carried toward cell body
dendrites
branches of axon
nucleus
axon
axon terminals
impulses carried away from cell body
cell body



Fully-Connected Neural Networks

Components

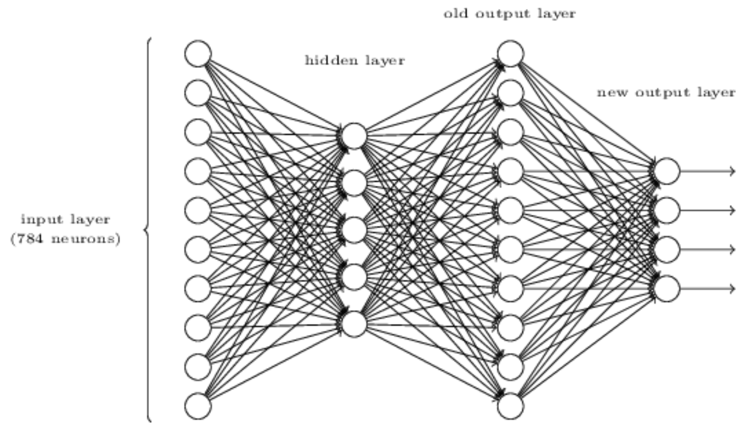


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$

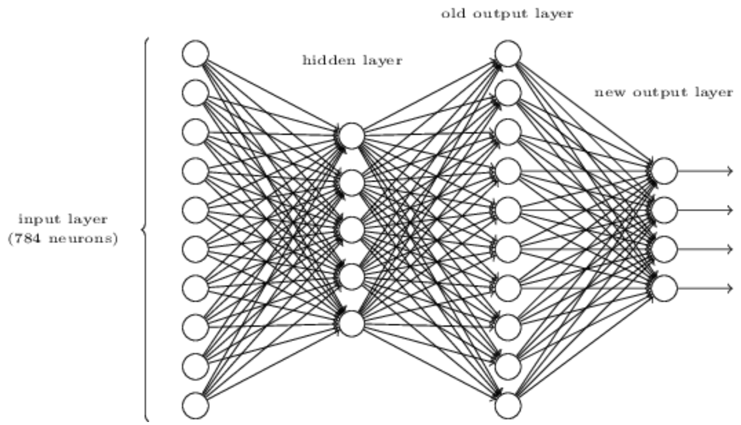


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$
- k- hidden layers, $a_i \in \mathbb{R}^{d_i}$
  - Weight matrices, $W_i \in \mathbb{R}^{d_{i-1} \times d_i}$
  - Bias vectors, $b_i \in \mathbb{R}^{d_i}$

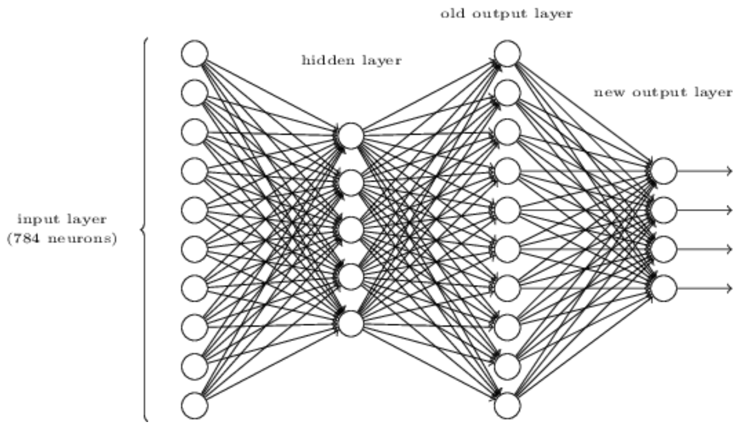


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$
- k- hidden layers, $a_i \in \mathbb{R}^{d_i}$
    - Weight matrices, $W_i \in \mathbb{R}^{d_{i-1} \times d_i}$
    - Bias vectors, $b_i \in \mathbb{R}^{d_i}$
- Output layer, $\hat{y} \in \mathbb{R}^m$

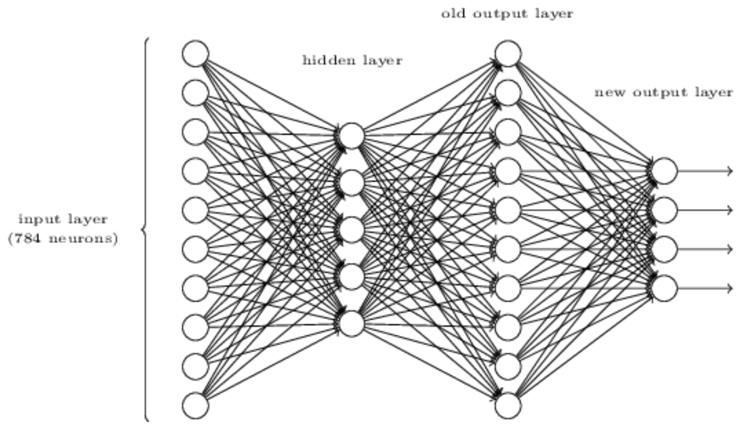


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$
- k- hidden layers, $a_i \in \mathbb{R}^{d_i}$
  - Weight matrices, $W_i \in \mathbb{R}^{d_{i-1} \times d_i}$
  - Bias vectors, $b_i \in \mathbb{R}^{d_i}$
- Output layer, $\hat{y} \in \mathbb{R}^m$
- For each layer, $a_i = f(z_i) = f(W_i h_i + b_i)$, where f is an activation function

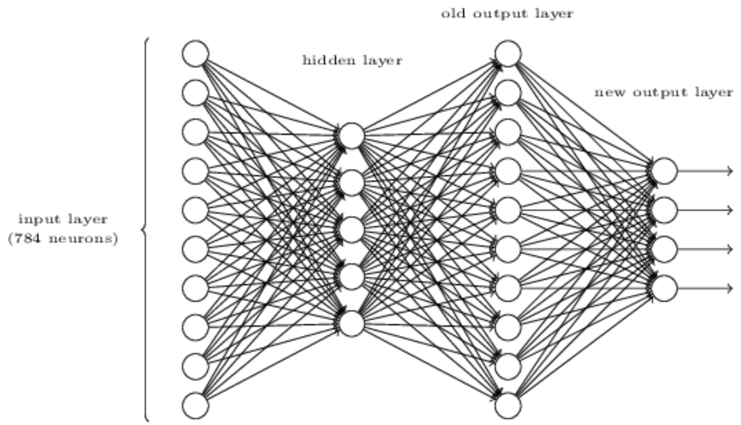


Image source

Fully-Connected Neural Networks

Components

- A single input layer, $h_0 \in \mathbb{R}^n$
- k- hidden layers, $a_i \in \mathbb{R}^{d_i}$
  - Weight matrices, $W_i \in \mathbb{R}^{d_{i-1} \times d_i}$
  - Bias vectors, $b_i \in \mathbb{R}^{d_i}$
- Output layer, $\hat{y} \in \mathbb{R}^m$
- For each layer, $a_i = f(z_i) = f(W_i h_i + b_i)$, where f is an activation function
  - Series of stacked layers compose multiple function together (e.g. $(f \circ g)(x)$)

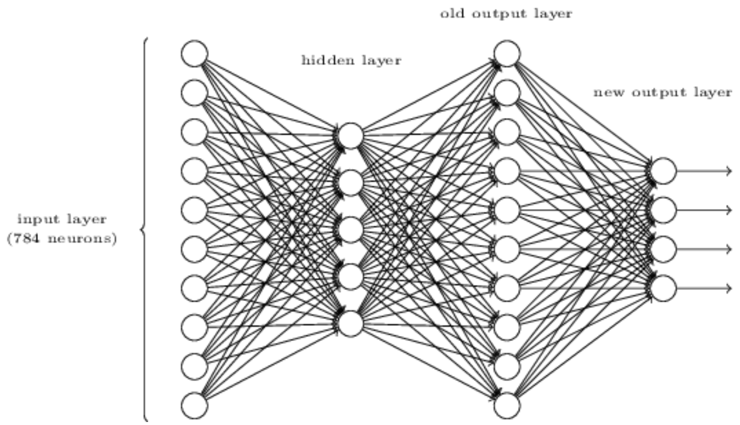


Image source

Fully-Connected Neural Networks

Cost Function

Fully-Connected Neural Networks

Cost Function

- To train parameters, compute a cost associated with every predicted/labeled output pair, y, y.

Fully-Connected Neural Networks

Cost Function

- To train parameters, compute a cost associated with every predicted/labeled output pair, y, y.
- Requirements: can be averaged over a batch, can be computed with outputs from network

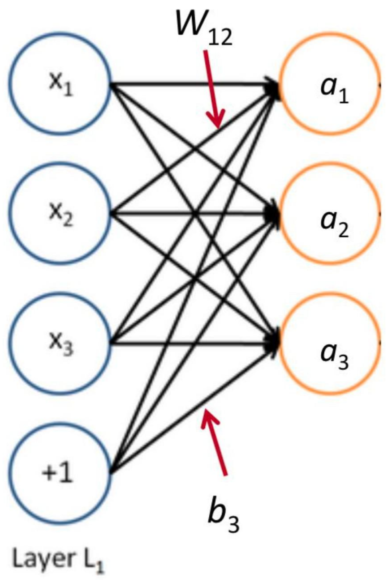
Fully-Connected Neural Networks

Cost Function

- To train parameters, compute a cost associated with every predicted/labeled output pair, y, y.
- Requirements: can be averaged over a batch, can be computed with outputs from network
- Common loss functions:
  - Least squares (quadratic): $\frac{1}{2m}\sum_{i=1}^{m} \| y_i - \hat{y}_i \|^2$
  - Binary Cross-Entropy: $y\log(\hat{y}) + (1 - y)\log(1 - y)$
  - Cross entropy (classification, y_j is one-hot encoding at j): $\sum_{i=1}^{m} y_i \log(\hat{y}_i)$

Fully-Connected Neural Network

Example



$$a_1 = f(W_{11}x_1 + W_{12}x_2 + W_{13}x_3 + b_1)$$

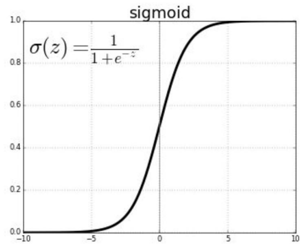
$$a_2 = f(W_{21}x_1 + W_{22}x_2 + W_{23}x_3 + b_2)$$

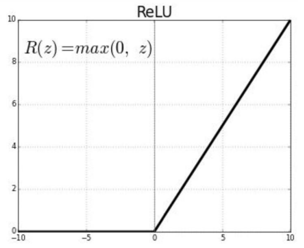
$$a_3 = f(W_{31}x_1 + W_{32}x_2 + W_{33}x_3 + b\ )$$

Sigmoid (logit) transform. $\sigma(z) = \frac{1}{1+e^{-z}}$

Hyperbolic tangent (tanh). $\tanh(z) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Rectified Linear Unit (ReLU). $\mathrm{ReLU}(z) = \max(0, z)$





Convolutional Neural Networks

Introduction

- For computer vision applications, convolutional networks are used to learn feature detectors from images
- Advantages:
  - Images are high-dimensional data, fully connected layers would require too many parameters to tune
  - Convolution operations preserve spatial structure of data
  - Convolution operation can be computed efficiently on GPUs (using CUDA)

- Analogues:
  - Inputs/activations are “what” the network “sees”
  - Weights are “how” the network computes one layer from the previous one (feature-detection)
- As architectures become more complex, interpretability of these learned features becomes more difficult

Convolutional Neural Networks

Components

Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$

Convolutional Neural Networks

Components

- Each convolutional "layer" is represented by a 3D tensor of shape
  $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape
  [`relative x-position, relative y-position, input channels, output channels`]

Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:

Convolutional Neural Networks

Components

- Each convolutional "layer" is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- "Convolve" operation consists of 4 hyperparameters:
  - Number of filters, or *depth* (each channel also called an "activation map")

Convolutional Neural Networks

Components

- Each convolutional "layer" is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- "Convolve" operation consists of 4 hyperparameters:
  - Number of filters, or *depth* (each channel also called an "activation map")

Convolutional Neural Networks

Components

- Each convolutional "layer" is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- "Convolve" operation consists of 4 hyperparameters:
  - Number of filters, or *depth* (each channel also called an "activation map")
  - *Spatial extent*, or *receptive field*

Convolutional Neural Networks

Components

- Each convolutional “layer” is represented by a 3D tensor of shape
  $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field*
  - The stride

Convolutional Neural Networks

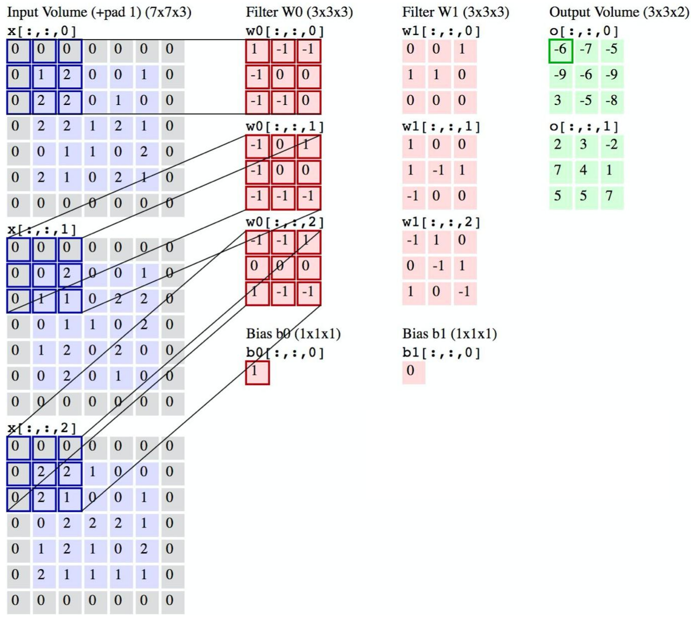
Components

- Each convolutional "layer" is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- "Convolve" operation consists of 4 hyperparameters:
  - Number of filters, or *depth* (each channel also called an "activation map")
  - *Spatial extent*, or *receptive field*
  - The stride
  - Amount of zero-padding

Convolutional Neural Networks

Components

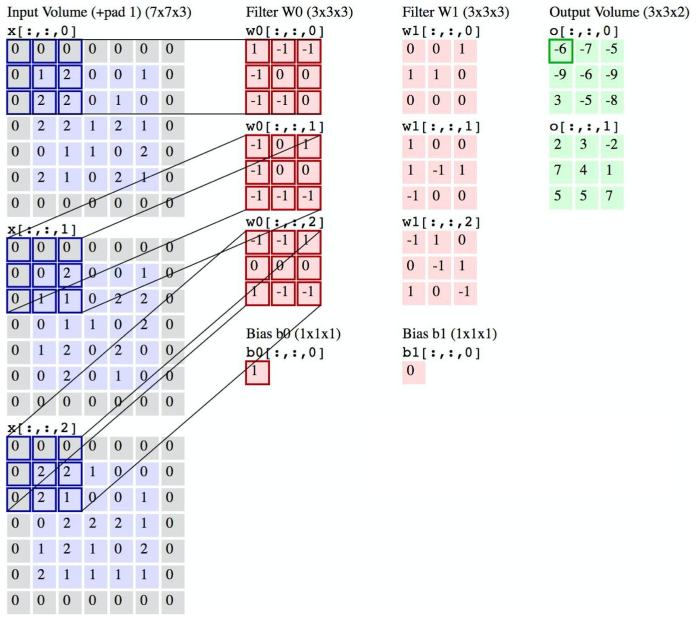
- Each convolutional "layer" is represented by a 3D tensor of shape
  $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- "Convolve" operation consists of 4 hyperparameters:
  - Number of filters, or *depth* (each channel also called an "activation map")
  - *Spatial extent*, or *receptive field*
  - The stride
  - Amount of zero-padding



Convolutional Neural Networks

Components

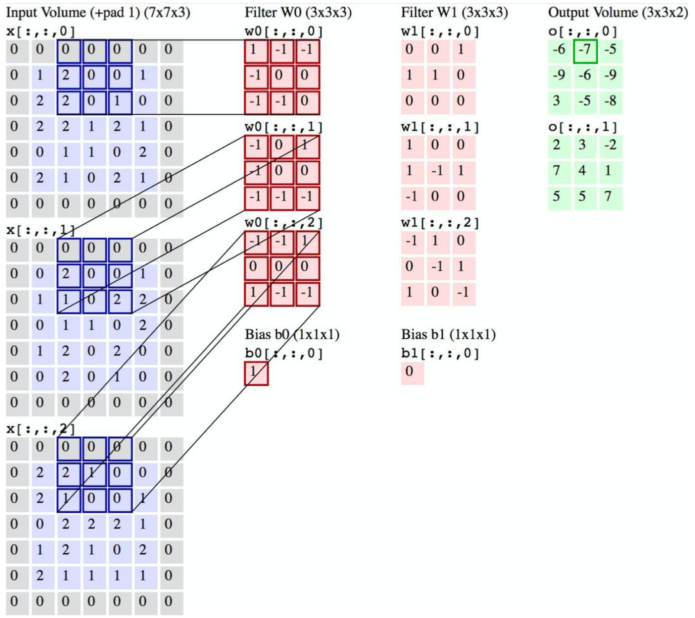
- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape `[relative x-position, relative y-position, input channels, output channels]`
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field*
  - The stride
  - Amount of zero-padding
- With this, the shape of layer convolved from layer -1 is:
  - $[\,(W - F + 2P)/S + 1, (H - F + 2P)/S + 1, K\,]$



Convolutional Neural Networks

Components

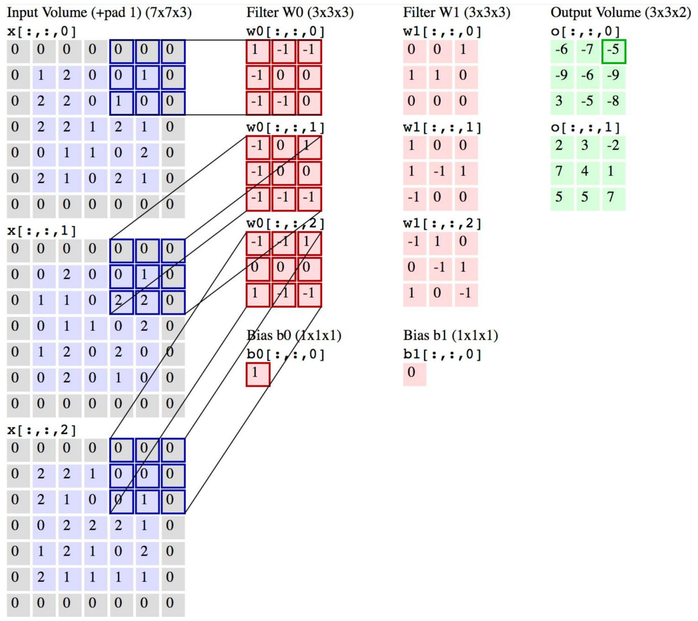
- Each convolutional “layer” is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- “Convolve” operation consists of 4 hyperparameters:
  - Number of filters, or *depth* (each channel also called an “activation map”)
  - *Spatial extent*, or *receptive field*
  - The stride
  - Amount of zero-padding
- With this, the shape of layer convolved from layer $-\ 1$ is:
  - $[\,(W - F + 2P)/S + 1, (H - F + 2P)/S + 1, K\,]$



Convolutional Neural Networks

Components

- Each convolutional "layer" is represented by a 3D tensor of shape $[h \times w \times n_{channels}]$
- Between two convolutional layers, the weights are of the shape [`relative x-position, relative y-position, input channels, output channels`]
- "Convolve" operation consists of 4 hyperparameters:
  - Number of filters, or *depth* (each channel also called an "activation map")
  - *Spatial extent*, or *receptive field*
  - The stride
  - Amount of zero-padding
- With this, the shape of layer convolved from layer $- 1$ is:
  - [$(W - F + 2P)/S + 1, (H - F + 2P)/S + 1, K$]



x[:,:,0]

0	0	0	0	0	0	0
0	1	2	0	0	1	0
0	2	2	0	1	0	0
0	2	2	1	2	1	0
0	0	1	1	0	2	0
0	2	1	0	2	1	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	0	2	0	0	1	0
0	1	1	0	2	2	0
0	0	1	1	0	2	0
0	1	2	0	2	0	0
0	0	2	0	1	0	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	2	2	1	0	0	0
0	2	1	0	0	1	0
0	0	2	2	2	1	0
0	1	2	1	0	2	0
0	2	1	1	1	1	0
0	0	0	0	0	0	0

w0[:,:,0]

1	-1	-1
-1	0	0
-1	-1	0

w0[:,:,1]

-1	0	1
-1	0	0
-1	-1	-1

w0[:,:,2]

-1	-1	1
0	0	0
1	-1	-1

b0[:,:,0]

1

w1[:,:,0]

0	0	1
1	1	0
0	0	0

w1[:,:,1]

1	0	0
1	-1	1
-1	0	0

w1[:,:,2]

-1	1	0
0	-1	1
1	0	-1

b1[:,:,0]

0

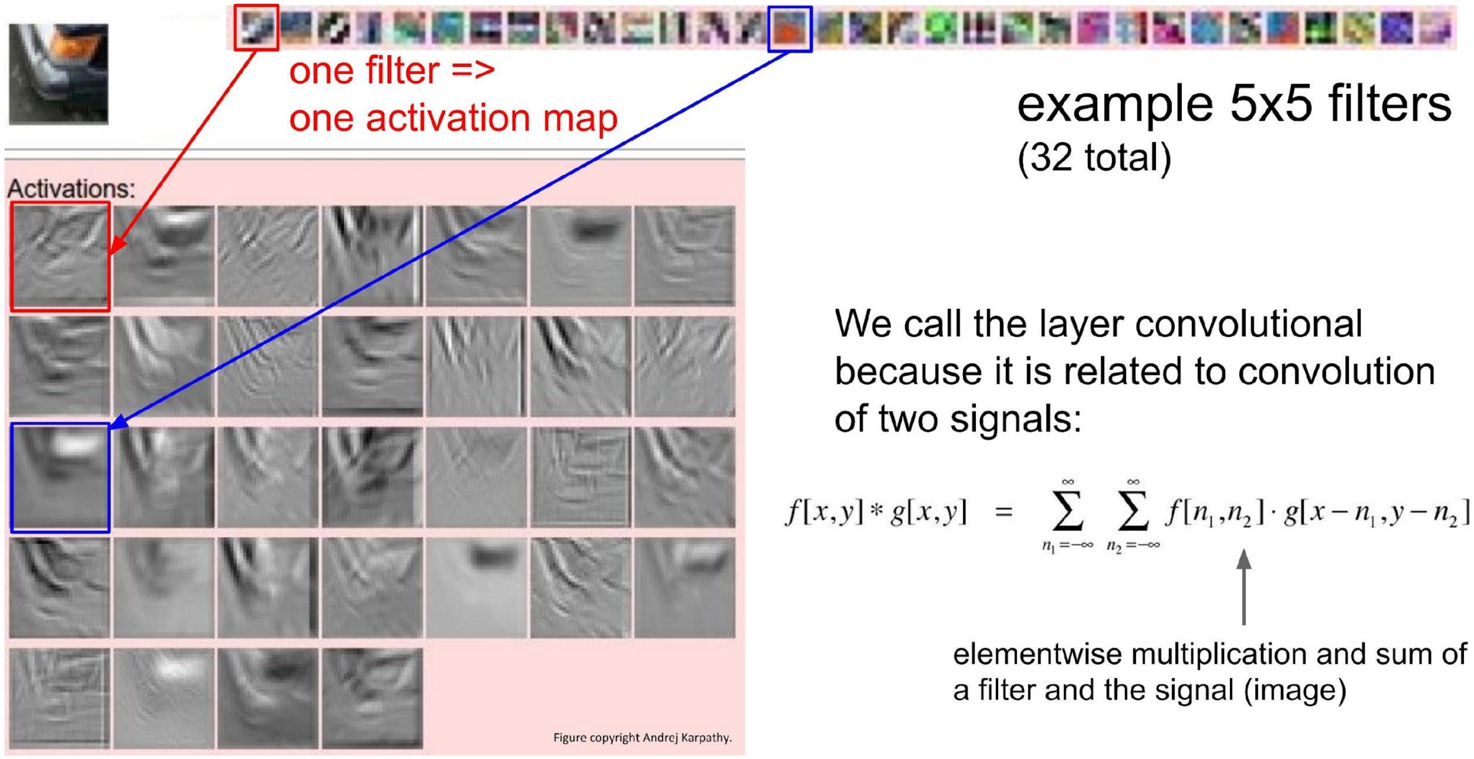
o[:,:,0]

-6	-7	-5
-9	-6	-9
3	-5	-8

o[:,:,1]

2	3	-2
7	4	1
5	5	7

Convolutional Neural Networks



example 5x5 filters
(32 total)

We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1,n_2] \cdot g[x-n_1, y-n_2]$$

$\uparrow$

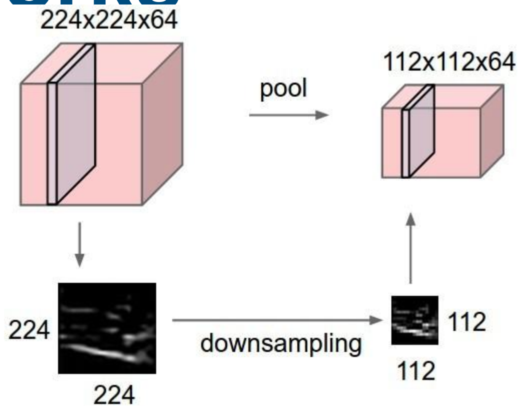
elementwise multiplication and sum of a filter and the signal (image)

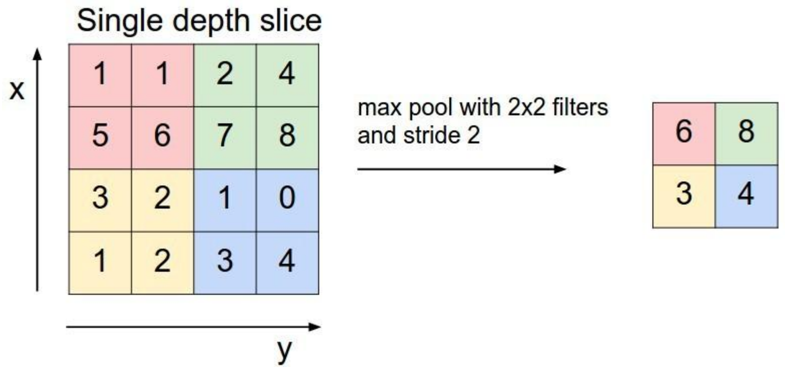
Slide borrowed from CS231N Lecture 5

Convolutional Neural Networks

Pooling and FC layers

- Max and Average (L2-norm) pooling:
  - Downsampling operation to reduce width x height (but not depth) of a layer
- Fully-connected (FC) layers:
  - Flattens entire input volume to a vector, and treats like a normal FC network layer





Fin

Questions?