

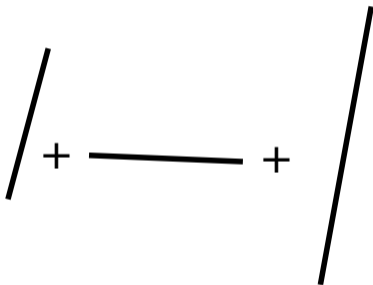
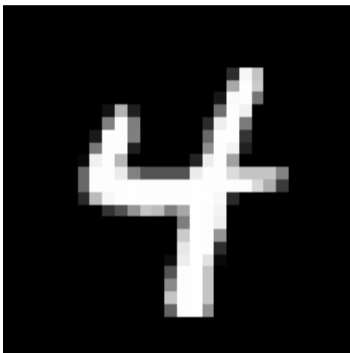
DSC 140B

Representation Learning

Lecture 16 | Part 1

Convolutions

$$H(x) = w_1x_1 + w_2x_2 + \dots + w_ax_a + w_0$$

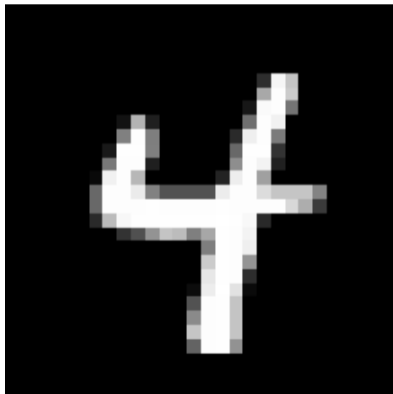


From Simple to Complex

- ▶ Complex shapes are made of simple patterns
- ▶ The human visual system uses this fact
- ▶ Line detector → shape detector → ... → face detector
- ▶ Can we replicate this with a deep NN?

Edge Detector

- ▶ How do we find **vertical edges** in an image?
- ▶ One solution: **convolution** with an **edge filter**.



Vertical Edge Filter

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

Idea

$$\begin{aligned} &0 + (-10) + 60 + \\ &0 + (-10) + 60 + \\ &0 + (-10) + 60 \end{aligned}$$

$$\begin{aligned} &= 180 - 30 \\ &= 150 \end{aligned}$$

- ▶ Take a patch of the image, same size as filter.
- ▶ Perform “dot product” between patch and filter.
- ▶ If large, this is a (vertical) edge.

image patch:

0	10	60
0	10	60
0	10	60

filter:

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

Idea

$$\begin{aligned} &0 + 0 + 0 + \\ &0 + (-10) + 10 + = 0 \\ &0 - 60 + 60 \end{aligned}$$

- ▶ Take a patch of the image, same size as filter.
- ▶ Perform “dot product” between patch and filter.
- ▶ If large, this is a (vertical) edge.

image patch:

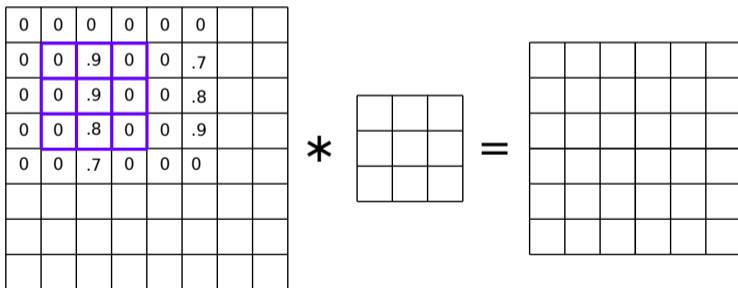
0	0	0
10	10	10
60	60	60

filter:

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

Idea

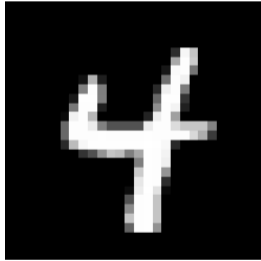
- ▶ Move the filter over the entire image, repeat procedure.



Convolution

- ▶ The result is the (2d) **convolution** of the filter with the image.
- ▶ Output is also 2-dimensional array.
- ▶ Called a **response map**.

Example: Vertical Filter



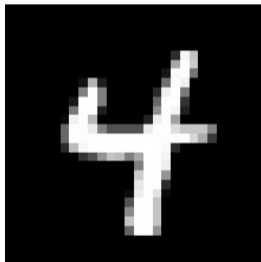
*

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

=



Example: Horizontal Filter



*

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

=



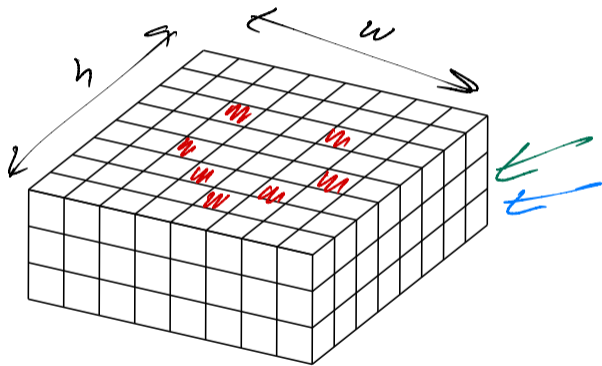
More About Filters

- ▶ Typically 3×3 or 5×5.
- ▶ Variations: different **stride**, image **padding**.

3-d Filters

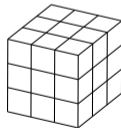
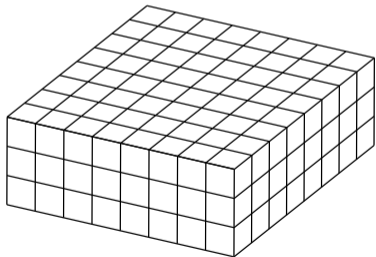
- ▶ Black and white images are 2-d arrays.
- ▶ But color images are 3-d arrays:
 - ▶ a.k.a., **tensors**
 - ▶ Three color **channels**: red, green, blue.
 - ▶ height \times width \times 3
- ▶ How does convolution work here?

Color Image

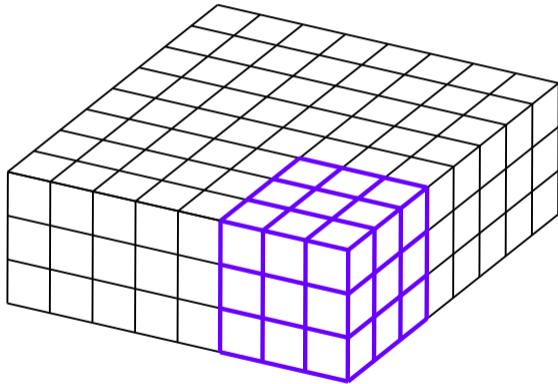


3-d Filter

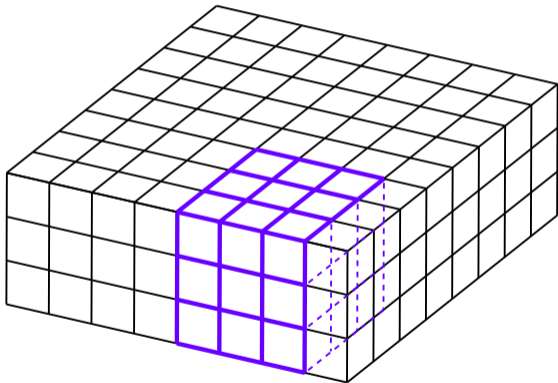
- ▶ The filter must also have three channels:
 - ▶ $3 \times 3 \times 3$, $5 \times 5 \times 3$, etc.



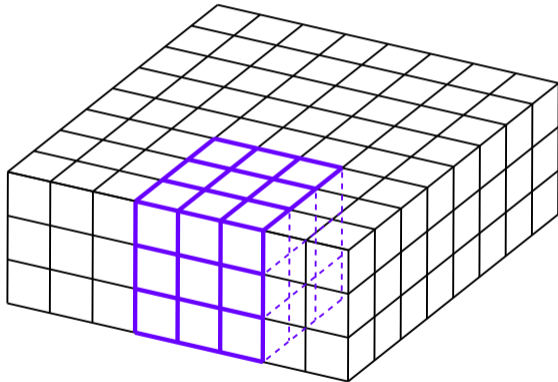
3-d Filter



3-d Filter



3-d Filter

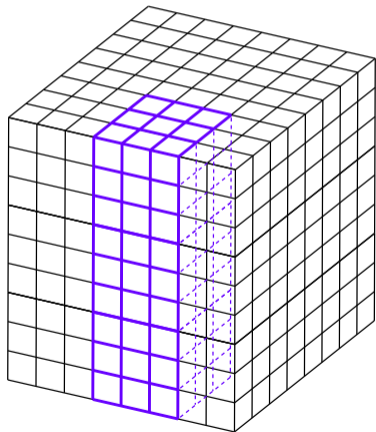


Convolution with 3-d Filter

- ▶ Filter must have same number of channels as image.
 - ▶ 3 channels if image RGB.
- ▶ Result is still a 2-d array.

General Case

- ▶ Input “image” has k channels.
- ▶ Filter must have k channels as well.
 - ▶ e.g., $3 \times 3 \times k$
- ▶ Output is still $2 - d$



DSC 140B

Representation Learning

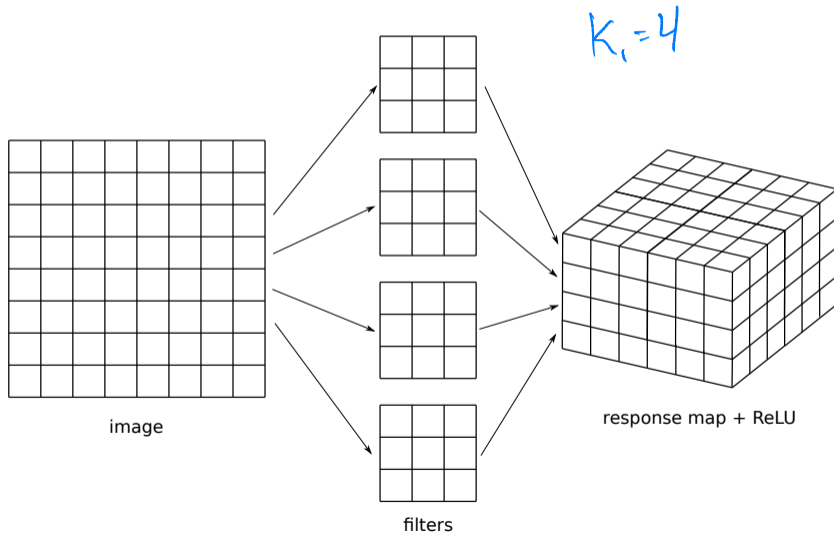
Lecture 16 | Part 2

Convolutional Neural Networks

Convolutional Neural Networks

- ▶ **CNNs** are the state-of-the-art for many computer vision tasks
- ▶ **Idea:** use convolution in early layers to create new feature representation.
- ▶ **But!** Filters are **learned**.

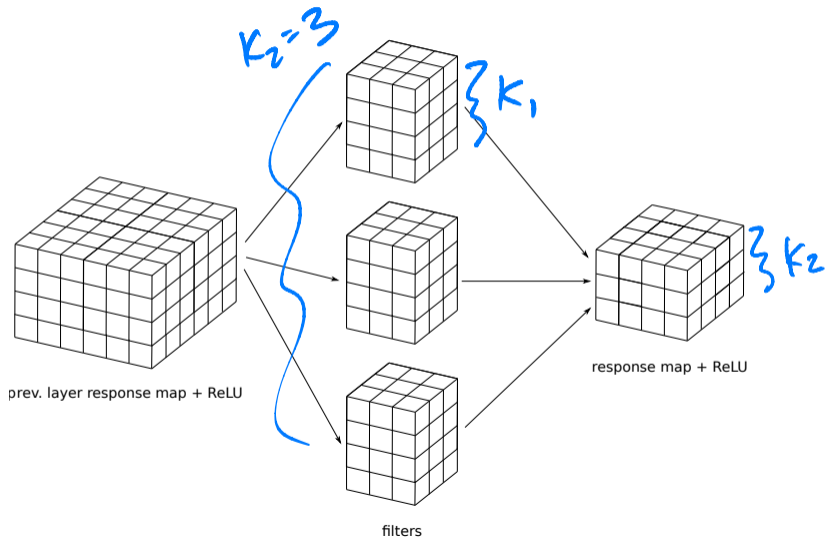
Input Convolutional Layer



Input Convolutional Layer

- ▶ Input image with one channel (grayscale)
- ▶ k_1 filters of size $\ell \times \ell \times 1$
- ▶ Results in k_1 convolutions, stacked to make response map.
- ▶ ReLU (or other nonlinearity) applied entrywise.

Second Convolutional Layer



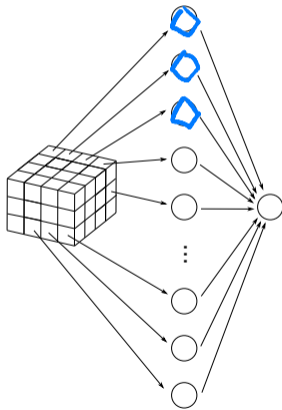
Second Convolutional Layer

- ▶ Input is a 3-d **tensor**.
 - ▶ “Stack” of k_1 response maps.
- ▶ k_2 filters, each a 3-d tensor with k_1 channels.
- ▶ Output is a 3-d tensor with k_2 channels.

More Convolutional Layers

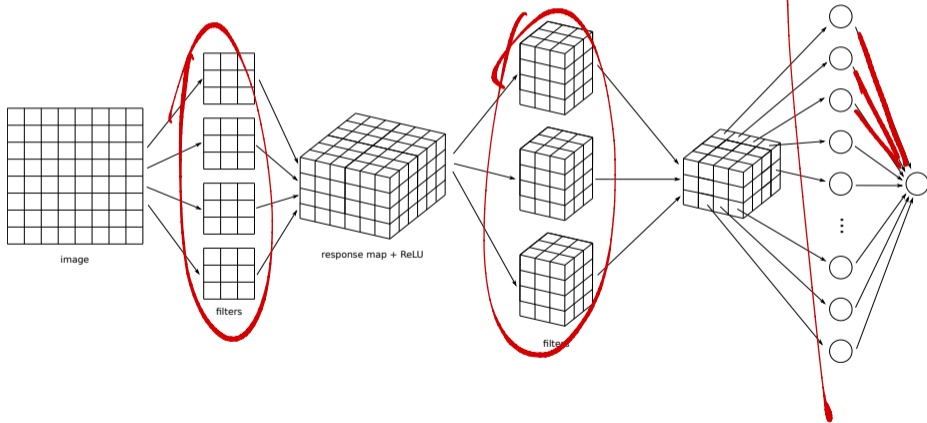
- ▶ May add more convolutional layers.
- ▶ Last convolutional layer used as input to a feedforward, fully-connected network.
- ▶ Need to “flatten” the output tensor.

Flattening



Full Network

$H(x)$



What is learned?

- ▶ The filters themselves.
- ▶ The weights in the feedforward NN used for prediction.

Max Pooling

- ▶ **Max pooling** is an important part of convolutional layers in practice.
- ▶ Reduces size of response map, number of parameters.

20	30	10	5		
10	15	20	30		
5	2	10	4		

DSC 140B

Representation Learning

Lecture 16 | Part 3

Example: Image Classification

Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.



Problem

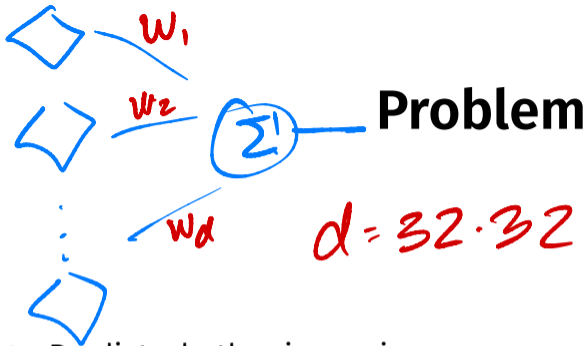
- ▶ Predict whether image is of a **car** or a **truck**.



Problem

- ▶ Predict whether image is of a **car** or a **truck**.





- Predict whether image is of a **car** or a **truck**.



Details

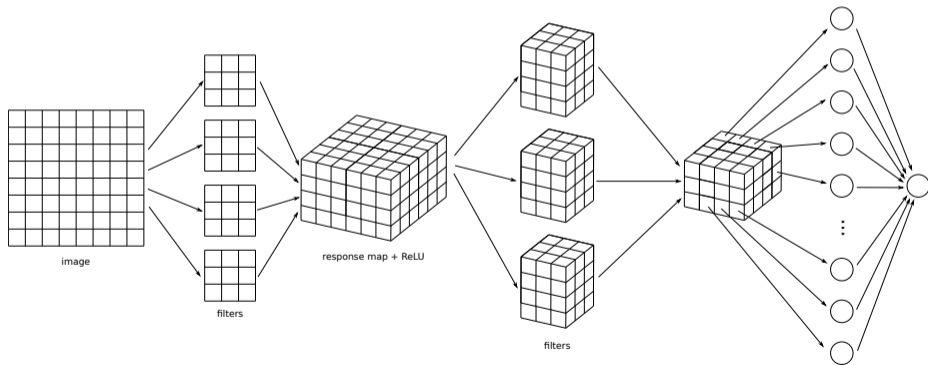
- ▶ 3-channel 32×32 color images
- ▶ 10,000 training images; 2,000 test¹
- ▶ Cars, trucks in different orientations, scales
- ▶ Balanced: 50% cars, 50% trucks

¹CIFAR-10

Approach #1: Least Squares Classifier

- ▶ Train directly on raw features (grayscale)
- ▶ Result: 72% train accuracy, 63% test accuracy
- ▶ Need a better feature representation

Approach #2: Convolutional Neural Network



Architecture

- ▶ 3 convolutional layers with 32, 64, 64 filters
- ▶ ReLU, max pooling after first two
- ▶ Dense layer with 64 hidden neurons, ReLU
- ▶ Output layer with sigmoid activation
- ▶ Minimize cross-entropy loss; use *dropout*

The Code

```
model = keras.models.Sequential()

model.add( keras.layers.Conv2D(32, (7, 7), activation='relu', input_shape=(32, 32, 1)))
model.add(keras.layers.MaxPooling2D((2, 2)))

model.add(keras.layers.Conv2D(64, (5, 5), activation='relu'))
model.add(keras.layers.MaxPooling2D((2, 2)))

model.add(keras.layers.Conv2D(64, (3, 3), activation='relu'))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dropout(0.5))
model.add(keras.layers.Dense(64, activation='relu'))
model.add(keras.layers.Dense(1, activation='sigmoid'))
```

The Code

```
model.compile(  
    optimizer=keras.optimizers.RMSprop(),  
    loss=keras.losses.BinaryCrossentropy(),  
    metrics=['accuracy']  
)  
  
model.fit(  
    X_train,  
    y_train,  
    epochs=30,  
    validation_data=(X_test, y_test)  
)
```

Results

- ▶ 94% train accuracy, 90% test accuracy

Results

car / car



Results

truck / car



Results

truck / truck



Results

truck / truck



Results

truck / truck



Results

truck / truck



Results

truck / truck



Results

car / car



Results

truck / truck



Results

truck / truck



Results

truck / truck



Results

truck / truck



Results

car / car



Results

car / truck



Results

truck / car



Results

car / car



Results

truck / truck



Results

car / car



Results

car / car



Results

car / car



Filters



Next Steps

- ▶ In practice, you might not train your own CNN
- ▶ Instead, take “pre-trained” convolutional layers from a much bigger network
- ▶ Attach untrained fully-connected layer and train
- ▶ This is **transfer learning**