DSC 140B
Representation Learning

Lecture 15 | Part 1

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

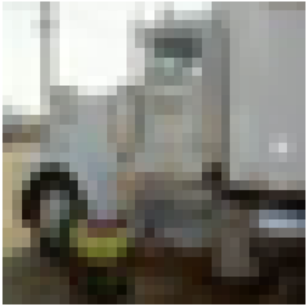
- Predict whether image is of a **car** or a **truck**.

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**.

Problem

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

The Dataset

- We'll use CIFAR-10:[1]
  - 3-channel 32×32 color images
  - 10,000 training images; 2,000 test
  - Cars, trucks in different orientations, scales
  - Balanced: 50% cars, 50% trucks

[1] `https://www.cs.toronto.edu/~kriz/cifar.html`

Approach #1: Least Squares Classifier

- Train directly on raw features (grayscale)
- Result: 72% train accuracy, 63% test accuracy

Problem

- Our least squares classifier didn't work well.
- Why not?
- Each feature is a **single pixel**.
- This representation is **too simple**.

Linear Models

- The prediction of a linear model is a weighted sum of the input features:

$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

- Each feature is separate from the others.

- To work well, some of the features must be **informative**.
  - They must have a (linear) association with the target.

Example: Predicting Job

- You are building a classifier to predict someone's occupation based on salary.
  - **Teacher** vs. **Data Scientist**
- Person A has a salary of $50,000; Person B has a salary of $150,000.
- Which person is more likely to be a teacher?

Example: Image Classification

- You are building a image classifier.
  - **Car** vs. **Truck**
- The intensity of pixel 3918 in two images is:
  - Image A: 0.13
  - Image B: 0.85
- Which image is more likely to be a truck?

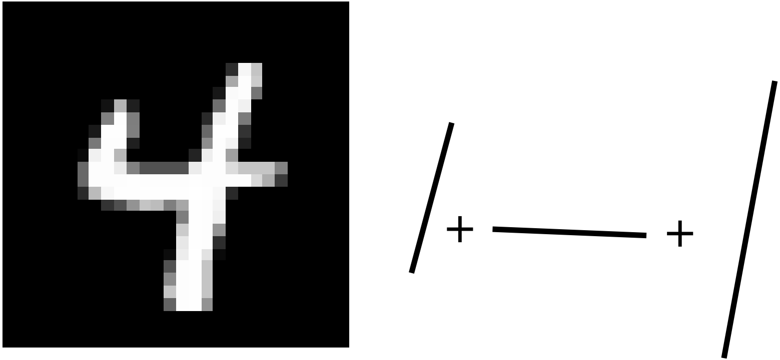
Using Context

- Features based on individual pixels are limited.
- We need a representation that captures **context**.
- **Today:** convolutional features and CNNs.

DSC 140B
Representation Learning

Lecture 15 | Part 2

Convolutions



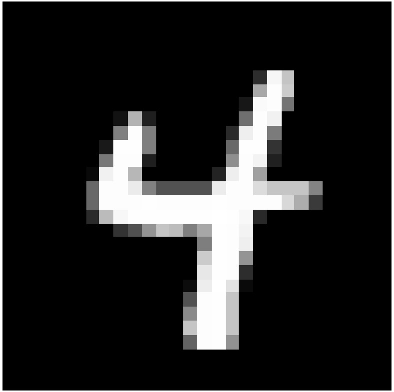
From Simple to Complex

- 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?

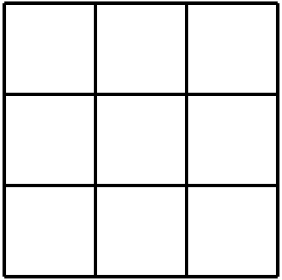
Edge Detector

- How do we find **vertical edges** in an image?

- One solution: **convolution** with an **edge filter**.



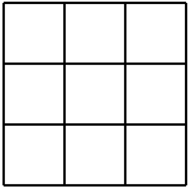
Vertical Edge Filter



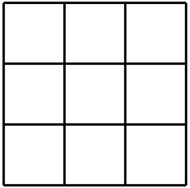
Idea

- 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:

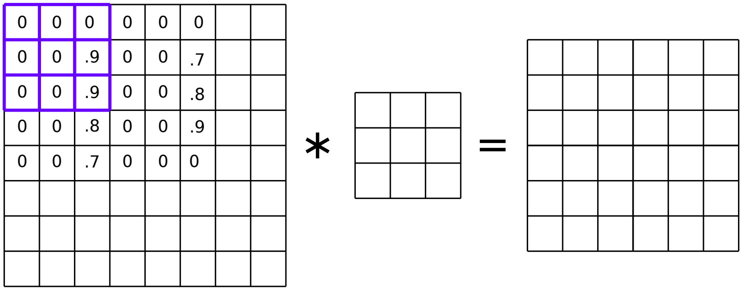


filter:



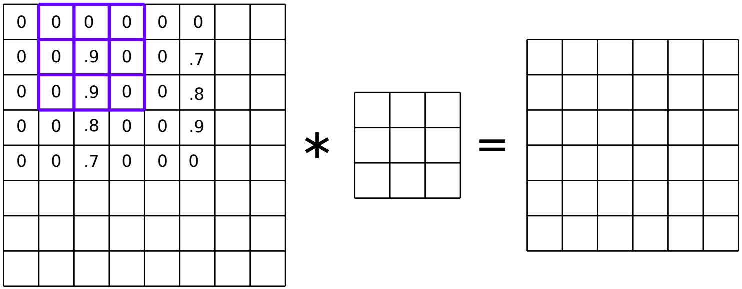
Idea

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



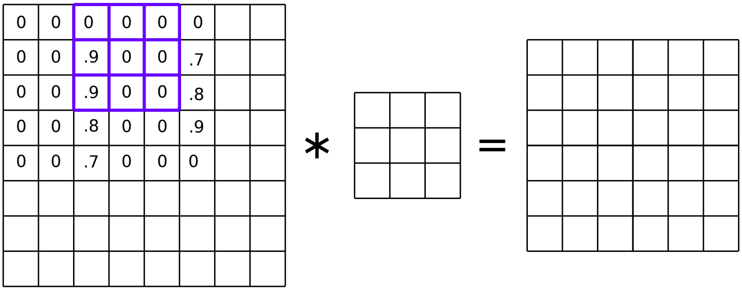
Idea

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



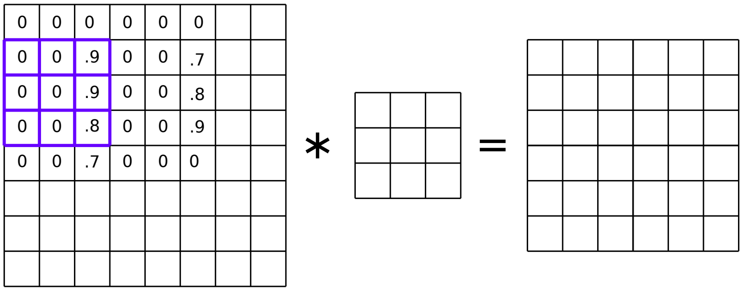
Idea

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



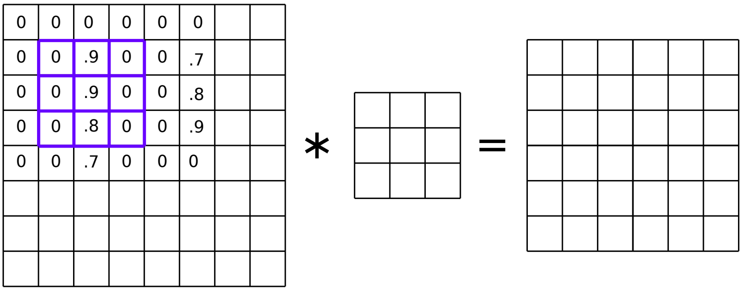
Idea

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



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**.

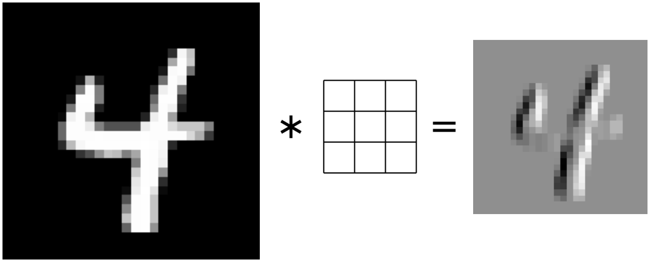
Exercise

Compute the convolution of the filter with the image at the position shown:

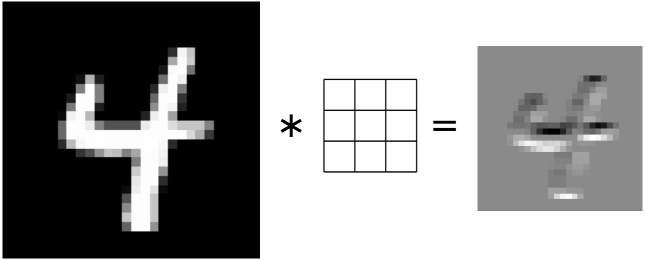
$$\text{Image patch: } \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix} \qquad \text{Filter: } \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

What is the output value?

Example: Vertical Filter



Example: Horizontal Filter



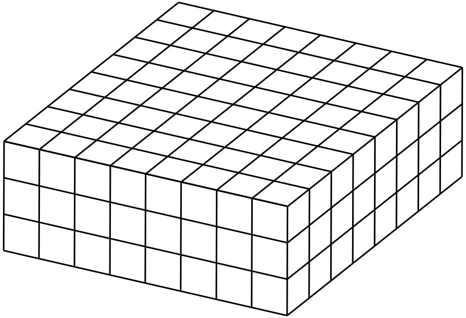
More About Filters

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

3-d Filters

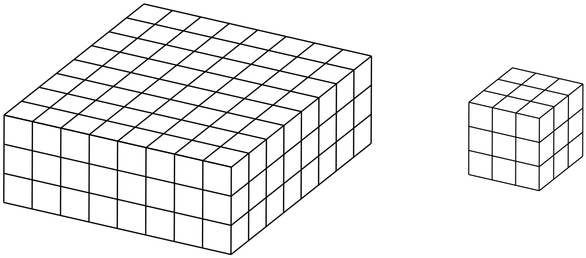
- Black and white images are 2-d arrays.
- But color images are 3-d arrays:
  - a.k.a., **tensors**
  - Three color **channels**: red, green, blue.
  - height × width × 3
- How does convolution work here?

Color Image

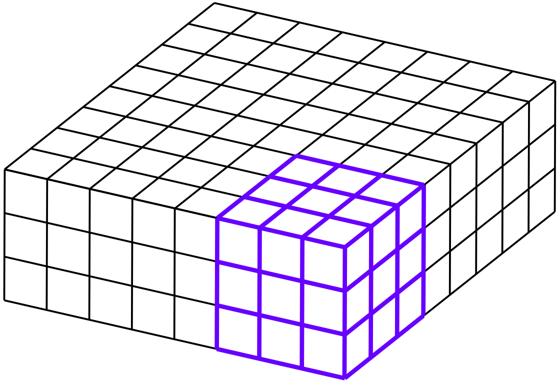


3-d Filter

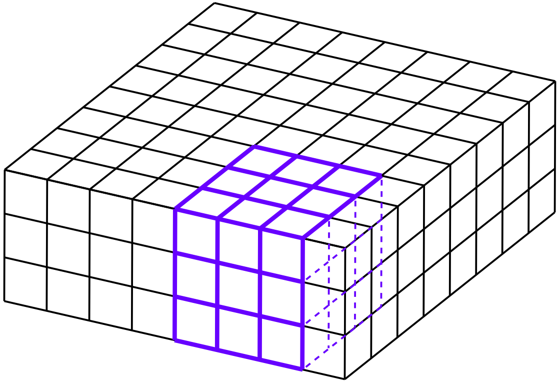
- The filter must also have three channels:
  - $3 \times 3 \times 3$, $5 \times 5 \times 3$, etc.



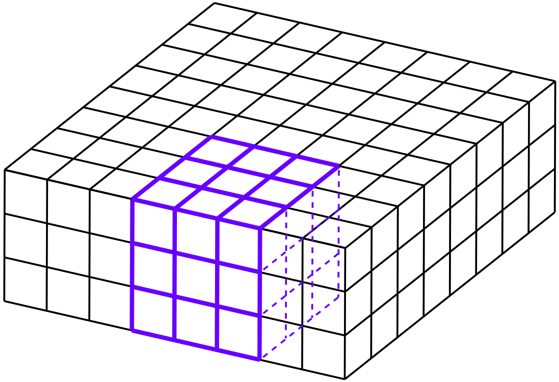
3-d Filter



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.

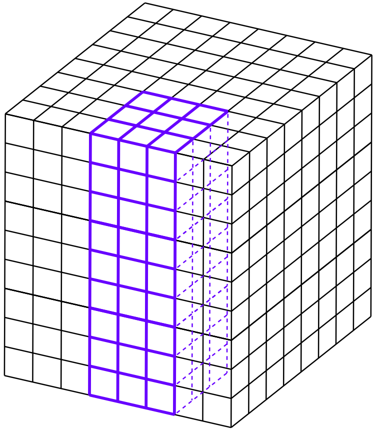
Exercise

How many parameters does a single 5×5 filter have when applied to an RGB image?

A) 25
B) 50
C) 75
D) 150

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

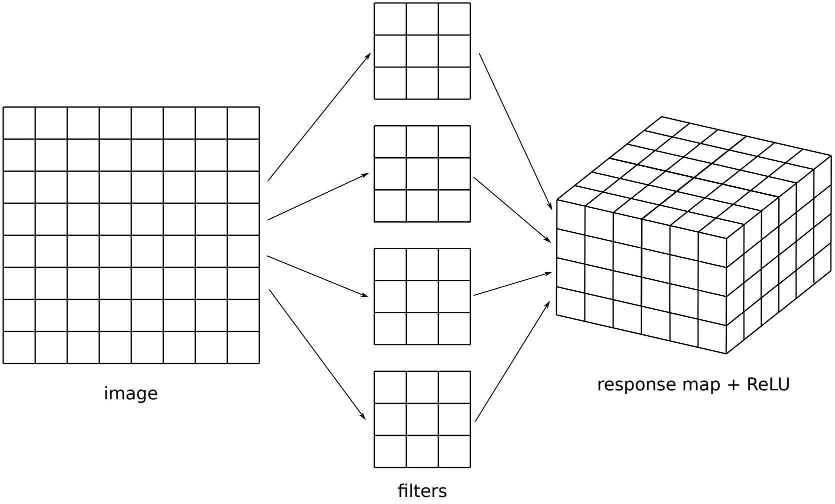
Lecture 15 | Part 3

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**.

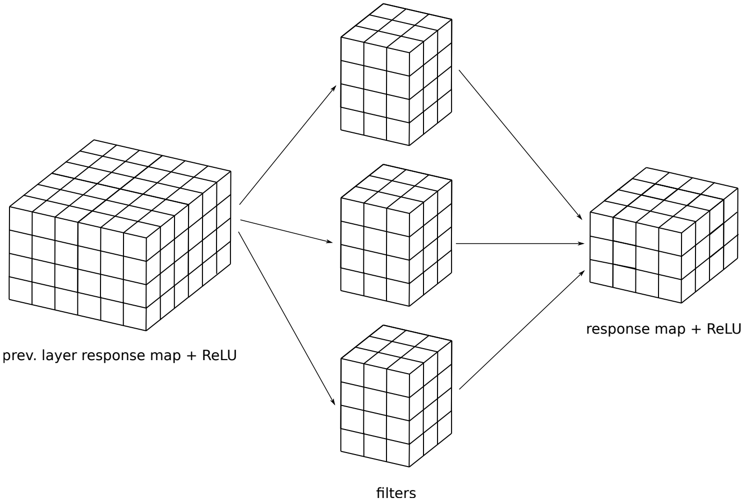
Input Convolutional Layer



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.

Exercise

The first convolutional layer uses 32 filters of size 3×3 on a grayscale image. The second layer uses 64 filters of size 3×3.

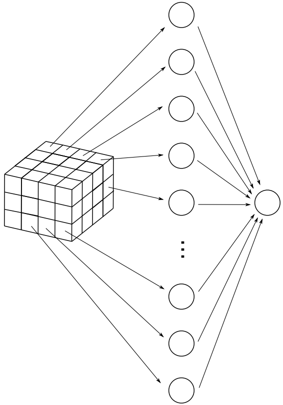
What is the shape of each filter in the second layer?

A) $3 \times 3 \times 1$
B) $3 \times 3 \times 3$
C) $3 \times 3 \times 32$
D) $3 \times 3 \times 64$

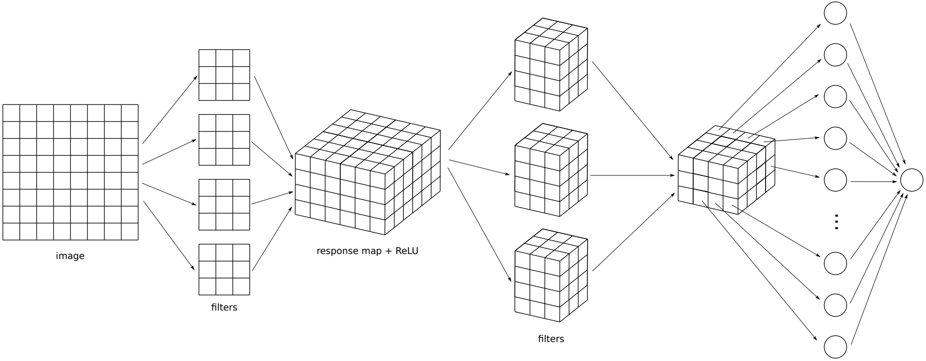
More Convolutional Layers

- 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

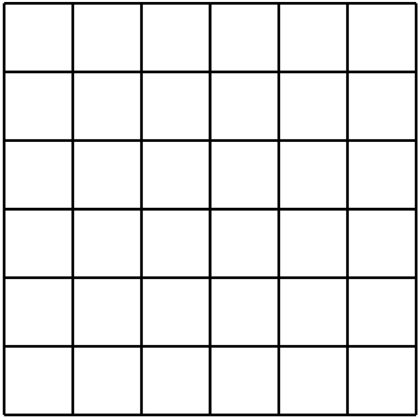


What is learned?

- 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.



Exercise

Compute the result of 2×2 max pooling on the following response map:

$$\begin{pmatrix} 1 & 3 & 0 & 2 \\ 5 & 2 & 1 & 4 \\ 0 & 7 & 3 & 1 \\ 2 & 1 & 6 & 0 \end{pmatrix}$$

DSC 140B

Representation Learning

Lecture 15 | Part 4

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

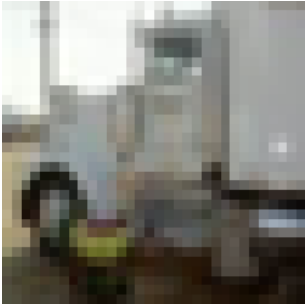
- Predict whether image is of a **car** or a **truck**.

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**.

Problem

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

Details

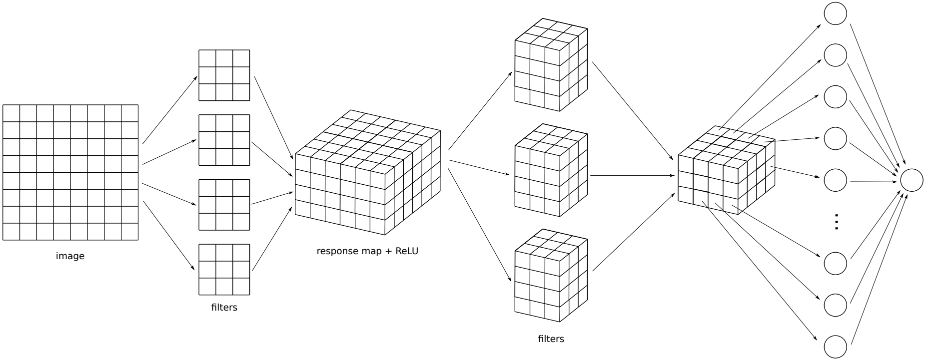
- 3-channel 32×32 color images
- 10,000 training images; 2,000 test[2]
- Cars, trucks in different orientations, scales
- Balanced: 50% cars, 50% trucks

[2] CIFAR-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
- Batch normalization, ReLU after each; max pooling after first two
- Dense layer with 64 hidden neurons, ReLU
- Output layer with sigmoid activation

The Code

```
model = nn.Sequential(
    nn.Conv2d(1, 32, 7),
    nn.BatchNorm2d(32),
    nn.ReLU(),
    nn.MaxPool2d(2),

    nn.Conv2d(32, 64, 5),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(2),

    nn.Conv2d(64, 64, 3),
    nn.BatchNorm2d(64),
    nn.ReLU(),

    nn.Flatten(),
    nn.Linear(64 * 2 * 2, 64),
    nn.ReLU(),
    nn.Linear(64, 1),
    nn.Sigmoid(),
)
```

The Code

```
optimizer = torch.optim.Adam(model.parameters())
loss_fn = nn.BCELoss()

for epoch in range(30):
    for X_batch, y_batch in train_loader:
        optimizer.zero_grad()
        y_pred = model(X_batch)
        loss = loss_fn(y_pred, y_batch)
        loss.backward()
        optimizer.step()
```

Results

- 100% train accuracy, 88% test accuracy

Results

predicted: truck / actual: car

Results

predicted: car / actual: car

Results

predicted: truck / actual: truck

Results

predicted: truck / actual: truck

Results

predicted: truck / actual: truck

Results

predicted: car / actual: truck

Results

predicted: truck / actual: truck

Results

predicted: car / actual: car

Results

predicted: truck / actual: truck

Results

predicted: truck / actual: truck

Results

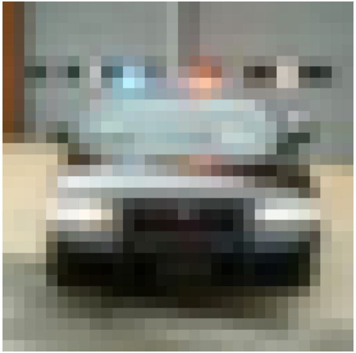
predicted: truck / actual: truck

Results

predicted: truck / actual: truck

Results

predicted: car / actual: car



Results

predicted: truck / actual: truck

Results

predicted: truck / actual: car

Results

predicted: car / actual: car

Results

predicted: truck / actual: truck

Results

predicted: car / actual: car

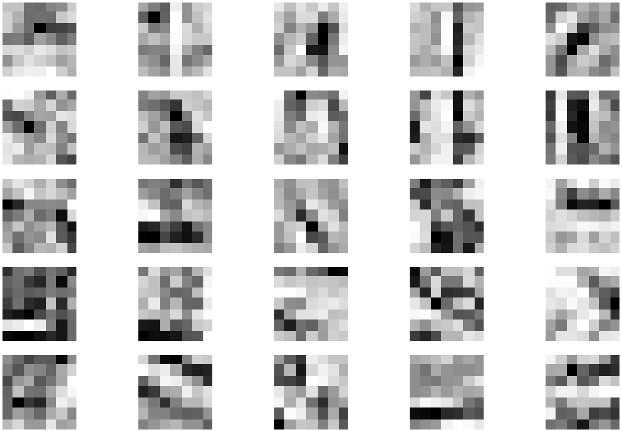
Results

predicted: car / actual: car

Results

predicted: truck / actual: car

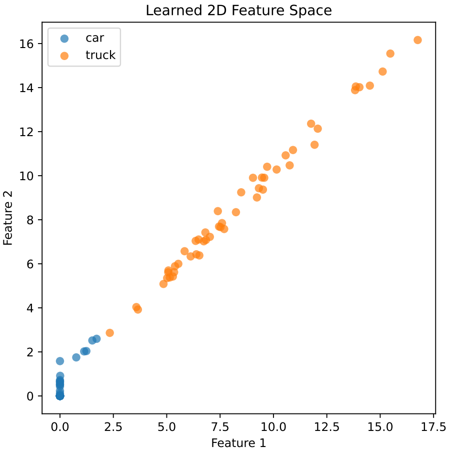
Filters



Feature Map

- We used a single hidden layer after the convolutional layers with 64 neurons.
- The network is a feature map from image space ($\mathbb{R}^{32\times32\times3} = \mathbb{R}^{3072}$) to $\mathbb{R}^{64}$.
- Let's add more hidden layers with 32, 16, and 2 neurons.
- Then we can visualize the feature map.

Feature Map


Learned 2D Feature Space
car
truck
Feature 2
Feature 1
0
2
4
6
8
10
12
14
16
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

DSC 140B

Representation Learning

Lecture 15 | Part 5

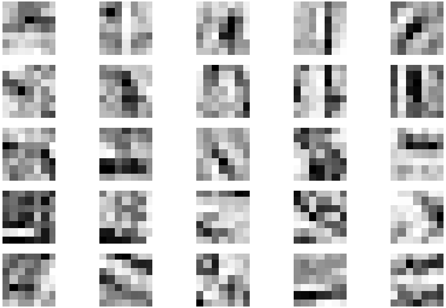
Transfer Learning

CNN Filters

- The first layers of our CNN learned image filters for distinguishing cars and trucks.
- The last layers learned to use those filters to predict truck vs. car.

Observation

- The filters in the first layers look like good **general-purpose** image filters.



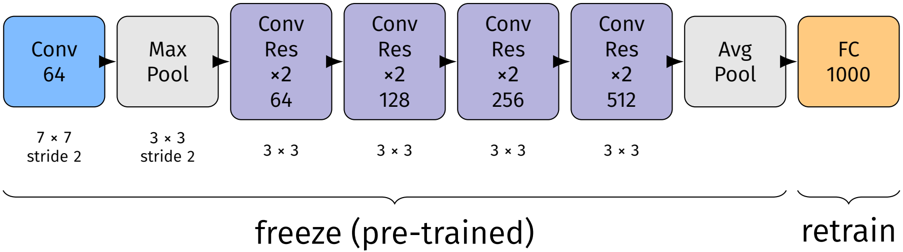
Thought

- If we had more data, we could learn more complex/better filters.
- They'd be useful not just for car vs. truck classification, but for many other tasks as well.

Idea

- People have trained CNNs on massive dataset.
  - e.g., ImageNet with 14 million images
- Let’s use their convolutional layers as a **feature extractor** for our task.
- We’ll retrain only the last (few) layers on our data.

Example: resnet-18[3]

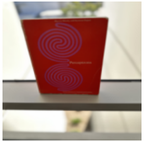
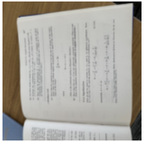


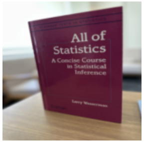
[3] "Deep Residual Learning for Image Recognition" by He et al., 2015.

Example

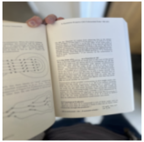
- Suppose we have a dataset of **only 30** images.
  - **Books** vs. **Pens**
- Want to build a binary classifier.
- We will use resnet-18 pre-trained on ImageNet.
  - **Freeze** the convolutional layers.
  - **Fine tune** the last fully-connected layer on our data.

Training Data: Books






Training Data: Pens

The Code: Setup

```python
from torchvision import models

resnet = models.resnet18(weights=models.ResNet18_Weights.DEFAULT)

# freeze all layers
for param in resnet.parameters():
    param.requires_grad = False

# replace the final fully-connected layer
resnet.fc = nn.Sequential(
    nn.Linear(resnet.fc.in_features, 1),
    nn.Sigmoid(),
)
```

The Code: Training

```
optimizer = torch.optim.Adam(resnet.fc.parameters(), lr=1e-3)
loss_fn = nn.BCELoss()

for epoch in range(30):
    pred = resnet(X_train).squeeze()
    loss = loss_fn(pred, y_train)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Results

- 100% test accuracy.
  - Though one of the books is right on the boundary.

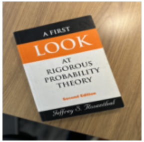
book / book (0.11)

book / book (0.36)

book / book (0.28)

book / book (0.50)

book / book (0.14)



pen / pen (0.88)

pen / pen (0.76)

pen / pen (0.86)

pen / pen (0.81)

pen / pen (0.90)

Transfer Learning

- Training on one task and applying to another is called **transfer learning**.
- Pre-trained model weights are published.
- Stand on the shoulders of giants!