UPPSALA UNIVERSITET



# Machine learning – Block 4


Måns Magnusson
Department of Statistics, Uppsala University


Autumn 2025

UPPSALA UNIVERSITET



# This week's lecture

- Convolutional Neural Networks
- Transfer Learning

# On this weeks assignment

- It takes a long time to run the models this week. Start early!

UPPSALA UNIVERSITET



# Section 2

# Introduction

# Convolutional Neural Networks

- Acknowledgements: Anders Eklund, Linköping University.

UPPSALA UNIVERSITET

UPPSALA UNIVERSITET



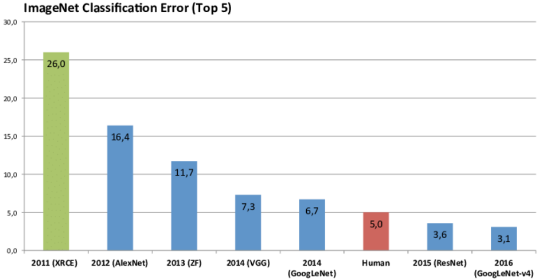
# Convolutional Neural Networks

- Acknowledgements: Anders Eklund, Linköping University.
- Convolutional Neural Networks are behind great progress in the 2010s.
- Revolutionized Computer Vision.

# Convolutional Neural Networks

- Acknowledgements: Anders Eklund, Linköping University.
- Convolutional Neural Networks are behind great progress in the 2010s.
- Revolutionized Computer Vision.
- Also called: ConvNets, Convolutional nets, Convolutional networks

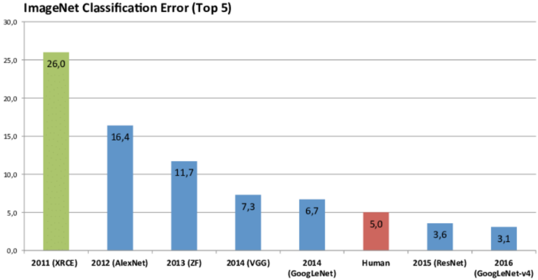
Figure: ImageNet performance (Roessler, 2019)



# Convolutional Neural Networks

- Acknowledgements: Anders Eklund, Linköping University.
- Convolutional Neural Networks are behind great progress in the 2010s.
- Revolutionized Computer Vision.
- Also called: ConvNets, Convolutional nets, Convolutional networks

Figure: ImageNet performance (Roessler, 2019)



# Convolutional Neural Networks

- Special architecture that works well for data with a grid structure
  1. 1D-grids: Time series

UPPSALA
UNIVERSITET



# Convolutional Neural Networks

- Special architecture that works well for data with a grid structure
  1. 1D-grids: Time series
  2. 2D-grids (matrix): Gray-scale Images (pixels)

# Convolutional Neural Networks



- Special architecture that works well for data with a grid structure
  1. 1D-grids: Time series
  2. 2D-grids (matrix): Gray-scale Images (pixels)
  3. 3D-grids (3D tensor): Color Images (pixels and channels)

UPPSALA UNIVERSITET



# Convolutional Neural Networks

- Special architecture that works well for data with a grid structure
  1. 1D-grids: Time series
  2. 2D-grids (matrix): Gray-scale Images (pixels)
  3. 3D-grids (3D tensor): Color Images (pixels and channels)
  4. 4D-grids (4D tensor): Color Video (pixels, channels, frames, time)

UPPSALA UNIVERSITET



# Computer Vision

- Problems
  - Image Classification

# Computer Vision

- Problems
  - Image Classification
  - Image Segmentation

UPPSALA UNIVERSITET



# Computer Vision

- Problems
  - Image Classification
  - Image Segmentation
  - Object Detection
  - ...

# Computer Vision

- Problems
  - Image Classification
  - Image Segmentation
  - Object Detection
  - ...
- Focus: 2D and 3D data

# Computer Vision

- Problems
  - Image Classification
  - Image Segmentation
  - Object Detection
  - ...
- Focus: 2D and 3D data
- Very Large Datasets:
  - ImageNet: 14M Images, 20k classes, 1M bounding boxes

# Computer Vision

- Problems
  - Image Classification
  - Image Segmentation
  - Object Detection
  - ...
- Focus: 2D and 3D data
- Very Large Datasets:
  - ImageNet: 14M Images, 20k classes, 1M bounding boxes
- Many different pre-trained models (e.g. VGG16)

# Example: Object Detection

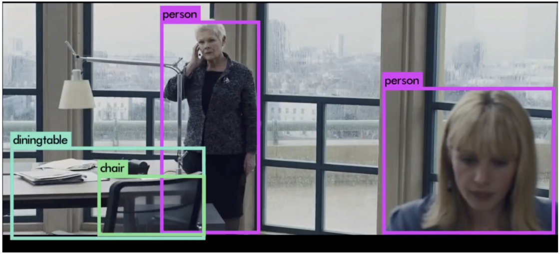



Figure: Object detection (see https://www.youtube.com/watch?v=VOC3huqHrss)

# Example: Pneumonia detection

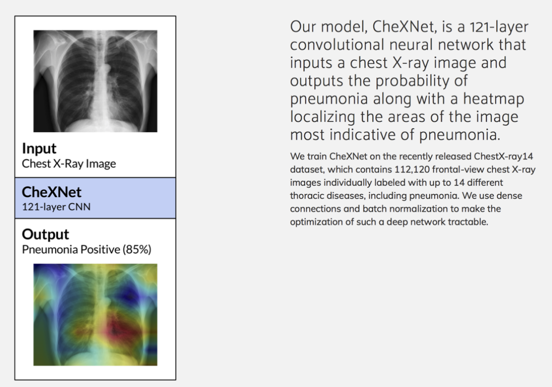



Figure: Rajpurkar et al. (2017). Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. arXiv preprint arXiv:1711.05225.

# Example: Fracture detection

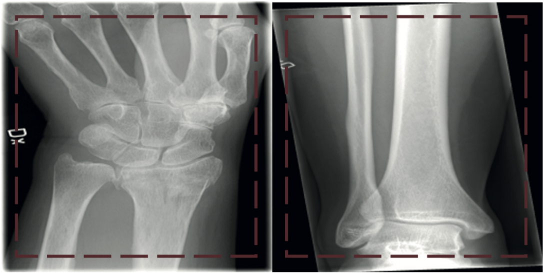


Figure 1. 2 images from the dataset. The area within the red box is the section presented to the network in order to classify the image. The left image is of a wrist fracture while the right image is without any apparent fracture.

Figure: Olczak et al, (2017) Artificial intelligence for analyzing orthopedic trauma radiographs, Acta Orthopaedica, 88:6, 581-586

# Example: Image Segmentation

Figure: Bishop and Bishop (2024) Fig 10.26

UPPSALA UNIVERSITET



# What is an Image?

- 2-dimensional object
- Each pixel has:
  1. a coordinate
  2. a value (light intensity)

# What is an Image?

- 2-dimensional object
- Each pixel has:
  1. a coordinate
  2. a value (light intensity)
- Grayscale: single channel
- Color: three channel (RGB)

UPPSALA UNIVERSITET



# What is an Image?

- 2-dimensional object
- Each pixel has:
  1. a coordinate
  2. a value (light intensity)
- Grayscale: single channel
- Color: three channel (RGB)
- Spatial and hierarchical correlation structures

# MNIST example

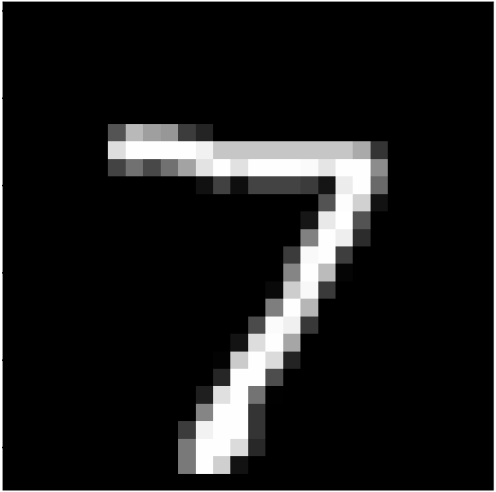


Figure: Example from the MNIST dataset (28 by 28 pixels)

# Why Not Use FFNN for Images?

- Parameter explosion: A fully-connected layer ignores image structure.

$$256 \times 256 \times 3 \approx 200{,}000 \text{ inputs}$$

A single hidden layer with 1,000 units would require

$$200{,}000 \times 1{,}000 = 200 \text{ million parameters.}$$

# Why Not Use FFNN for Images?

- Parameter explosion: A fully-connected layer ignores image structure.

  $$256 \times 256 \times 3 \approx 200{,}000 \text{ inputs}$$

  A single hidden layer with 1,000 units would require

  $$200{,}000 \times 1{,}000 = 200 \text{ million parameters.}$$

- Locality is ignored: Nearby pixels are strongly correlated. FFNNs treat all pixels as independent inputs.

# Why Not Use FFNN for Images?

- Parameter explosion: A fully-connected layer ignores image structure.

$$256 \times 256 \times 3 \approx 200{,}000 \text{ inputs}$$

  A single hidden layer with 1,000 units would require

$$200{,}000 \times 1{,}000 = 200 \text{ million parameters.}$$

- Locality is ignored: Nearby pixels are strongly correlated. FFNNs treat all pixels as independent inputs.
- Missing invariances: Images contain translation/shift invariance. MLPs must *learn* this from data instead of building it in.

# Why Not Use FFNN for Images?

- Parameter explosion: A fully-connected layer ignores image structure.

$$256 \times 256 \times 3 \approx 200{,}000 \text{ inputs}$$

A single hidden layer with 1,000 units would require

$$200{,}000 \times 1{,}000 = 200 \text{ million parameters.}$$

- Locality is ignored: Nearby pixels are strongly correlated. FFNNs treat all pixels as independent inputs.
- Missing invariances: Images contain translation/shift invariance. MLPs must *learn* this from data instead of building it in.
- **What CNNs do intuitively**:
  - Look at small local regions (receptive fields)
  - Reuse the same filter across the image (weight sharing)
  - Build hierarchical features: edges → textures → shapes

# How to train models for images?

- We want to learn representations of parts of images

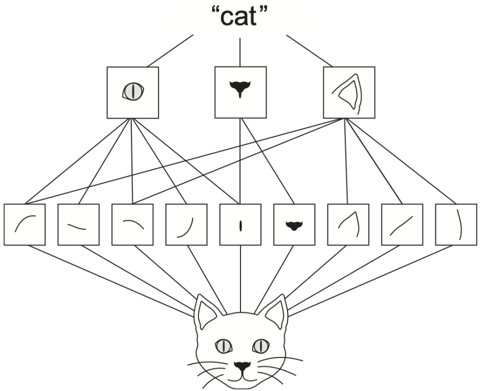



Figure: The representations of a cat (Chollet and Allaire, 2018, Fig 5.2)

UPPSALA UNIVERSITET



# How to train models for images?

- We want to learn representations of parts of images

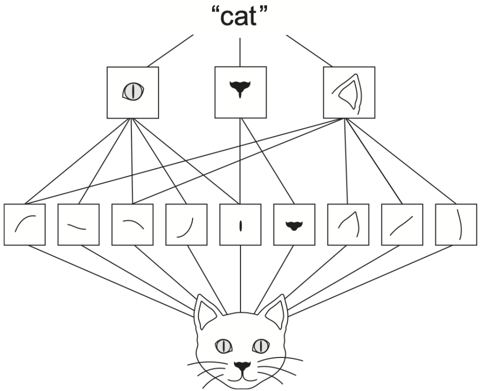



Figure: The representations of a cat (Chollet and Allaire, 2018, Fig 5.2)

- CNN uses Convolutional Layers to learn parameter efficient representations

# Learning Representations for Images (again)

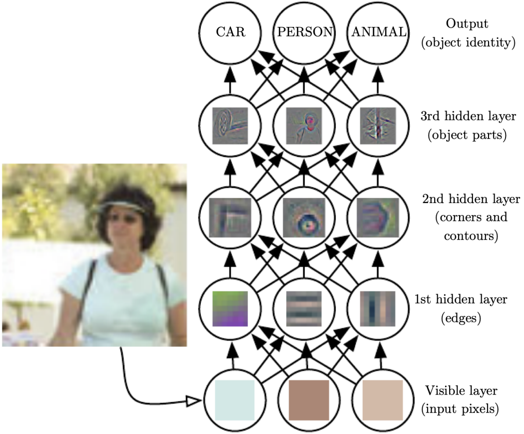



Figure: Learning representations for images (Goodfellow et al, 2017, Fig. 1.2)

UPPSALA UNIVERSITET



# Section 4

# Convolution

UPPSALA UNIVERSITET



# Convolution

- Different definitions are common, one example:

$$y(t) = \int x(\tau)k(t-\tau)d\tau = (x * k)(t)$$

- Inutuition: "Weighting together two functions"
- In a convolutional layer:
  1. $x(t)$: Input
  2. $k(t)$: Kernel, filter, "feature"
  3. $y(t)$: Output, feature map

DEMO

# Discrete Convolution

- If $t$ is discrete (as in a grid):

$$y(t) = (x * k)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)k(t-\tau)$$

# Discrete Convolution

- If $t$ is discrete (as in a grid):

$$y(t) = (x * k)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)k(t-\tau)$$

- In the case of images we have 2 discrete dimensions

$$Y(i,j) = (X * K)(i,j) = \sum_{m}\sum_{n} X(m,n)K(i-m,j-n)$$

# Discrete Convolution

- If $t$ is discrete (as in a grid):

$$y(t) = (x * k)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)k(t-\tau)$$

- In the case of images we have 2 discrete dimensions

$$Y(i,j) = (X * K)(i,j) = \sum_{m}\sum_{n} X(m,n)K(i-m, j-n)$$

- Sometimes the cross-correlation is called convolution:

$$Y(i,j) = (X * K)(i,j) = \sum_{m}\sum_{n} X(m,n)K(i+m, j+n)$$

1. $X(i,j)$: Input (2D)
2. $K(i,j)$: Kernel, filter, "feature" (2D)
3. $Y(i,j)$: Output, feature map (2D)

UPPSALA UNIVERSITET

- Practicalities
- Introduction
- Computer Vision
- Convolution
- Convolutional Neural Networks
  - The Convolution Layer
  - The Pooling Layer
  - Regularization
  - Examples
- Transfer learning
- Practical Methodology

# Convolution of Images: 2D

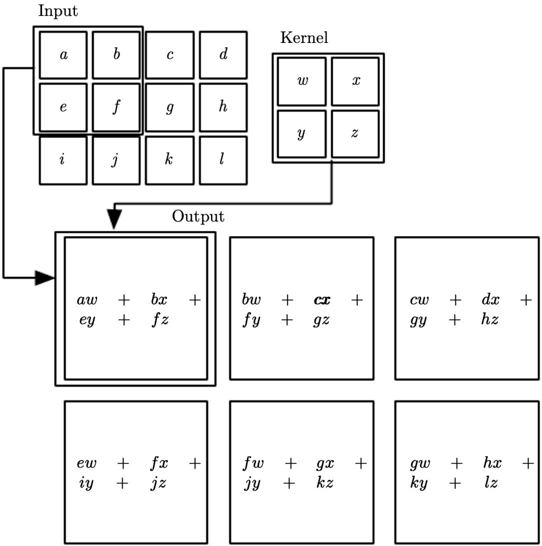



Figure: Convolution for an Image (Goodfellow et al, 2017, Fig. 9.1)

# Convolution of images: Example

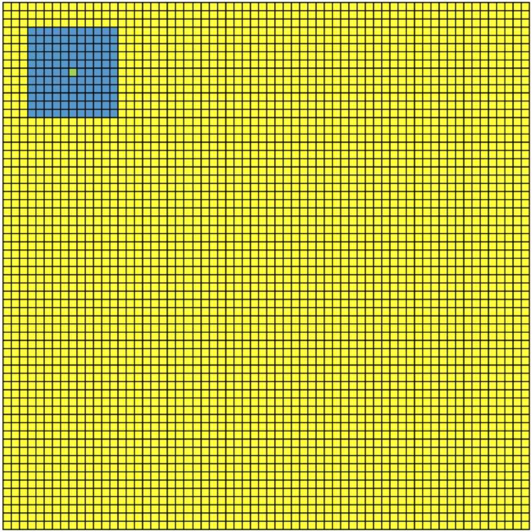


Figure: Convolution example.

# Convolution of images: Examples

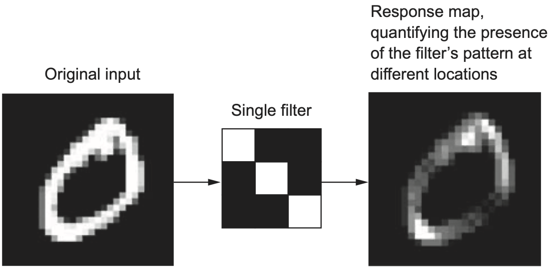



Figure: Convolution for an Image (Chollet and Allaire, 2018, Fig. 5.3)

# Convolution of images: Example



$$X = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}, K = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$$

# Convolution of images: Example

$$X = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \end{bmatrix}, K = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$$

$$Y = \begin{bmatrix} 0 & 0 & 2 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 1 & 2 & 0 \end{bmatrix},$$

# Edge Detection

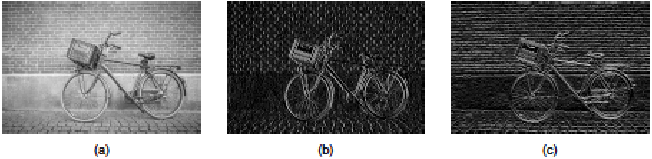



Figure: Edge Detections (Bishop and Bishop, 2024, Fig. 10.4)

# Examples of filters

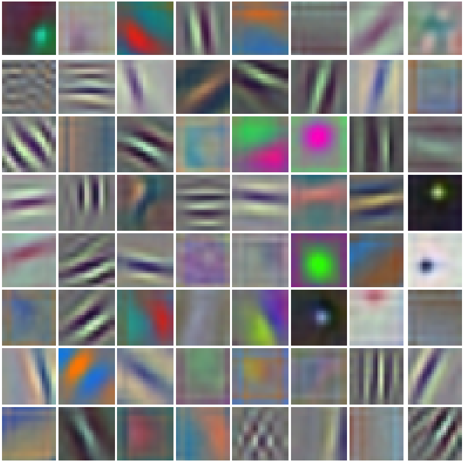


Figure: AlexNet filters (Bishop and Bishop, 2024, Fig. 10.12)

# Section 5

# Convolutional Neural Networks

# Convolutional Neural Networks

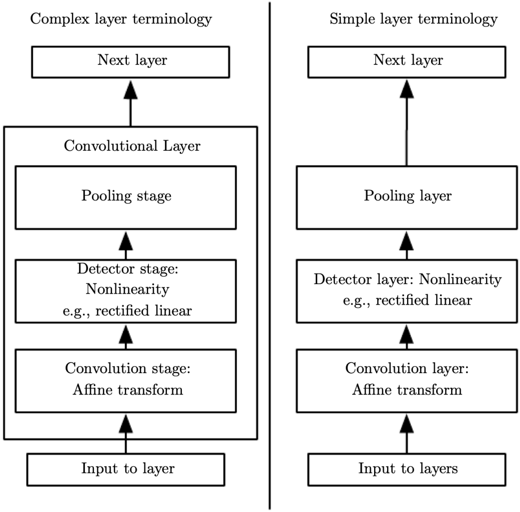



Figure: Convolution layer (Goodfellow et al, 2018, Fig. 9.7)

# Convolutional Neural Networks

- Most convolutional neural networks have:
  1. Many convolutional layers
  2. More kernels/filters higher up in the network
  3. A classification head (usually a feed-forward neural network)

# Convolutional Neural Networks

- Most convolutional neural networks have:
  1. Many convolutional layers
  2. More kernels/filters higher up in the network
  3. A classification head (usually a feed-forward neural network)
- Benefits:
  1. Few(er) parameters (filters)
  2. Captures local structures
  3. Efficient computations

UPPSALA UNIVERSITET



# Convolutional Neural Networks

- Most convolutional neural networks have:
  1. Many convolutional layers
  2. More kernels/filters higher up in the network
  3. A classification head (usually a feed-forward neural network)
- Benefits:
  1. Few(er) parameters (filters)
  2. Captures local structures
  3. Efficient computations
- How to choose filters?
  1. Before: manually handcrafted

# Convolutional Neural Networks

- Most convolutional neural networks have:
  1. Many convolutional layers
  2. More kernels/filters higher up in the network
  3. A classification head (usually a feed-forward neural network)
- Benefits:
  1. Few(er) parameters (filters)
  2. Captures local structures
  3. Efficient computations
- How to choose filters?
  1. Before: manually handcrafted
  2. Now: learn the filters

# Convolution layer

- Input: Data or Feature Maps

# Convolution layer

- Input: Data or Feature Maps
- Parameters:
  - $N$ filters/kernels of size $m \times m$
  - $N$ bias terms (one per filter)

UPPSALA UNIVERSITET



# Convolution layer

- Input: Data or Feature Maps
- Parameters:
  - $N$ filters/kernels of size $m \times m$
  - $N$ bias terms (one per filter)
- Activation functions: Applied element wise on feature maps

# Convolution layer

- Input: Data or Feature Maps
- Parameters:
  - $N$ filters/kernels of size $m \times m$
  - $N$ bias terms (one per filter)
- Activation functions: Applied element wise on feature maps
- Output: Feature Maps

UPPSALA UNIVERSITET



# Convolution layer

- Input: Data or Feature Maps
- Parameters:
  - $N$ filters/kernels of size $m \times m$
  - $N$ bias terms (one per filter)
- Activation functions: Applied element wise on feature maps
- Output: Feature Maps
- In Keras:

```
layer_conv_2d(filters = 32, kernel_size =
c(3,3), activation = "relu", input_shape =
c(32,32,3))
```

# Padding

- Handling edges
- *Padding*: add 0 around the image
- Necessary to keep size of feature maps

# Padding

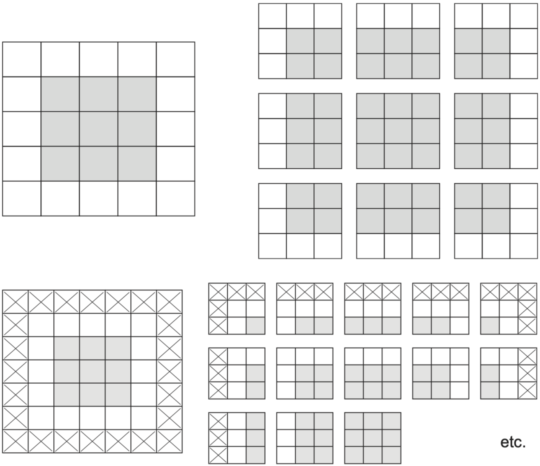



Figure: Padding and valid edge handling (Chollet and Allaire (2018), Fig. 5.5, 5.6)

# Stride

- Skip every $n$th pixel
- Reduces the computations

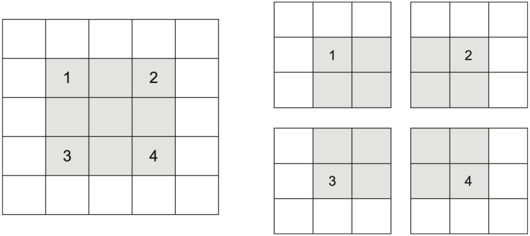



Figure: Strides (Chollet and Allaire (2018), Fig. 5.5, 5.6)

UPPSALA UNIVERSITET



# How Convolutions Change Tensor Shapes

- Input tensor shape:

$$X \in \mathbb{R}^{H \times W}$$

  where $H$ is the height and $W$ is the width of the image.
- Convolution parameters:
  - Kernel size: $k_h \times k_w$
  - Stride: $s$
  - Padding: $p$

# How Convolutions Change Tensor Shapes

- Input tensor shape:

$$X \in \mathbb{R}^{H \times W}$$

  where $H$ is the height and $W$ is the width of the image.
- Convolution parameters:
  - Kernel size: $k_h \times k_w$
  - Stride: $s$
  - Padding: $p$
- Output tensor shape:

$$Y \in \mathbb{R}^{H' \times W'}$$

  where

$$H' = \left\lfloor \frac{H - k_h + 2p}{s} \right\rfloor + 1, \qquad W' = \left\lfloor \frac{W - k_w + 2p}{s} \right\rfloor + 1.$$

UPPSALA UNIVERSITET



# Example: Computing Output Size

- Given an input image:

$$32 \times 32 \times 3$$

- Convolution layer parameters:
  - Kernel size: $3 \times 3$
  - Padding: $p = 1$
  - Stride: $s = 1$
  - Number of filters: $C_{\text{out}} = 16$

# Example: Computing Output Size

- Given an input image:

$$32 \times 32 \times 3$$

- Convolution layer parameters:
  - Kernel size: $3 \times 3$
  - Padding: $p = 1$
  - Stride: $s = 1$
  - Number of filters: $C_{\text{out}} = 16$
- Output height and width:

$$H' = \frac{32 - 3 + 2p}{s} + 1 = \frac{32 - 3 + 2}{1} + 1 = 32,$$

$$W' = \frac{32 - 3 + 2}{1} + 1 = 32.$$

# Example: Computing Output Size

- Given an input image:

$$32 \times 32 \times 3$$

- Convolution layer parameters:
  - Kernel size: $3 \times 3$
  - Padding: $p = 1$
  - Stride: $s = 1$
  - Number of filters: $C_{\text{out}} = 16$
- Output height and width:

$$H' = \frac{32 - 3 + 2p}{s} + 1 = \frac{32 - 3 + 2}{1} + 1 = 32,$$

$$W' = \frac{32 - 3 + 2}{1} + 1 = 32.$$

- Final output tensor:

$$32 \times 32 \times 16.$$

UPPSALA UNIVERSITET



# Why Convolution Layers?

- Captures local spatial structure
- Reduces the number of parameters (parameter sharing)
  1. The number and size of filters
  2. We use the same filters everywhere

UPPSALA UNIVERSITET



# Why Convolution Layers?

- Captures local spatial structure
- Reduces the number of parameters (parameter sharing)
  1. The number and size of filters
  2. We use the same filters everywhere
- Example: a 1 megapixel, single channel, image ($1000 \times 1000$ pixels)
  1. Dense network with 100 nodes: 100M parameters
  2. CNN network with 100 $3 \times 3$ filters: 1000 parameters (900 from filters, 100 bias terms)

# Convolution Neural Nets

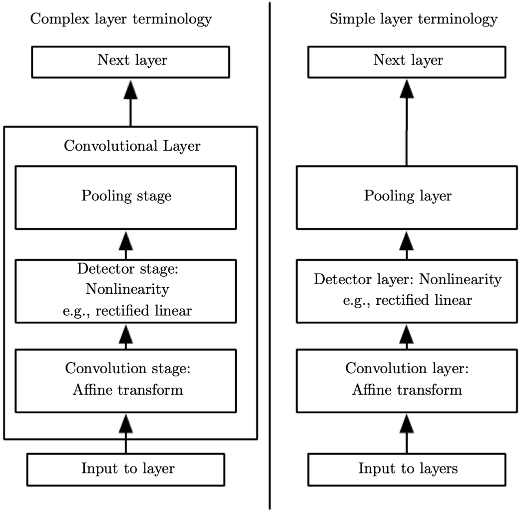



Figure: Convolution layer (Goodfellow et al, 2018, Fig. 9.7)

# Detector stage

- Remember, in feed-forward networks: $\mathbf{h} = \sigma(\mathbf{XW} + b)$
- In CNN:
  1. $\mathbf{W}$ is the filter

# Detector stage

- Remember, in feed-forward networks: $\mathbf{h} = \sigma(\mathbf{XW} + b)$
- In CNN:
  1. $\mathbf{W}$ is the filter
  2. $\mathbf{X}$ is the input feature map

# Detector stage

- Remember, in feed-forward networks: $\mathbf{h} = \sigma(\mathbf{XW} + b)$
- In CNN:
  1. $\mathbf{W}$ is the filter
  2. $\mathbf{X}$ is the input feature map
  3. $\mathbf{XW}$ is the convolutional feature map $(X * W)$

# Detector stage

- Remember, in feed-forward networks: $\mathbf{h} = \sigma(\mathbf{XW} + b)$
- In CNN:
  1. $\mathbf{W}$ is the filter
  2. $\mathbf{X}$ is the input feature map
  3. $\mathbf{XW}$ is the convolutional feature map ($X * W$)
  4. $b$ is a bias (one per filter)

# Detector stage

- Remember, in feed-forward networks: $\mathbf{h} = \sigma(\mathbf{XW} + b)$
- In CNN:
  1. $\mathbf{W}$ is the filter
  2. $\mathbf{X}$ is the input feature map
  3. $\mathbf{XW}$ is the convolutional feature map ($X * W$)
  4. $b$ is a bias (one per filter)
  5. $\sigma$ is the activation function (usually a ReLU)

# Pooling layer

- We take a function $f$ that returns one value per pooling kernel

UPPSALA UNIVERSITET



# Pooling layer

- We take a function $f$ that returns one value per pooling kernel
- Most commonly $f = \max$
- Max Pooling: Take the maximum value in the pooling kernel

# Pooling layer

- We take a function $f$ that returns one value per pooling kernel
- Most commonly $f = \max$
- Max Pooling: Take the maximum value in the pooling kernel
- Pooling has no trainable parameters

# Pooling layer

- We take a function $f$ that returns one value per pooling kernel
- Most commonly $f = \max$
- Max Pooling: Take the maximum value in the pooling kernel
- Pooling has no trainable parameters
- Commonly a $2 \times 2$ pooling kernel with stride 2
- Why? Reduce the size of feature map, but keep the activation

# Pooling layer

- We take a function $f$ that returns one value per pooling kernel
- Most commonly $f = \max$
- Max Pooling: Take the maximum value in the pooling kernel
- Pooling has no trainable parameters
- Commonly a $2 \times 2$ pooling kernel with stride 2
- Why? Reduce the size of feature map, but keep the activation
- In Keras:
  `layer_max_pooling_2d(pool_size = c(2, 2))(`

# Max Pooling

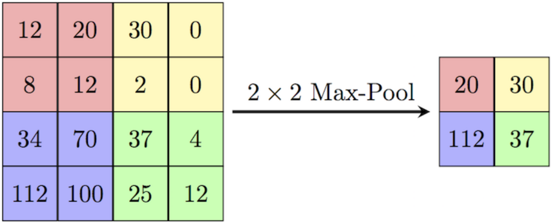



Figure: Strides ("Computer Science" at Wikipedia)

# Using pooling to learn invariances

- pooling over spatial positions: invariant to translation
- pooling over different filters: invariant to transformations

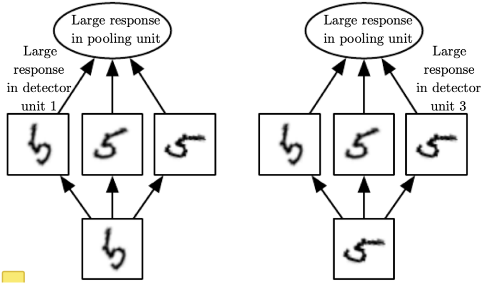



Figure: Learning invariances (Goodfellow et al., 2017, Fig. 9.9)

# Pooling Layers: Pros and Cons





- Pros
  - Provides translation invariance
  - Reduces spatial resolution $\rightarrow$ computational savings
  - Helps prevent overfitting by reducing model capacity

# Pooling Layers: Pros and Cons





- Pros
  - Provides translation invariance
  - Reduces spatial resolution → computational savings
  - Helps prevent overfitting by reducing model capacity
- Cons
  - Discards fine-grained spatial information
  - Can hurt performance on tasks requiring precise localization (e.g., segmentation, object detection)
  - Often replaced or reduced in modern architectures (e.g., strided convolutions)

See *Benefits of Max Pooling in Neural Networks: Theoretical and Experimental Evidence* by Matoba et al. (2023) for more details.

# Data Augmentation

Figure: Data Augmentation (Chollet and Allaire, 2018, Fig 5.10)

# Data Augmentation

Figure: Data Augmentation (Chollet and Allaire, 2018, Fig 5.10)

- Can be done directly in Keras (data generator)

UPPSALA UNIVERSITET



# Popular CNN architectures

- AlexNet (2012), 5 convolutional layers
- VGG16 (2014), 16 convolutional layers
- ResNet (2015), 152 convolutional layers

# VGG16

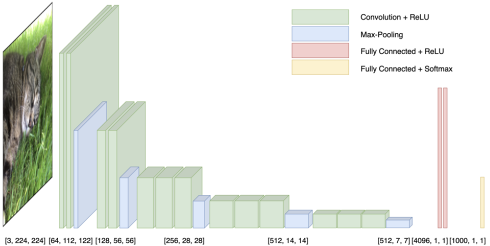



***Figure 2.4:*** *Illustration of the data flow through the network VGG16. Data size, of the format* $[c, h, w]$*, is shown for the input image, output of each max-pooling layer, output after the first two fully connected layers, and the final network output.*

Jesper Westell, Multi-Task Learning using Road Surface Condition Classification and Road Scene Semantic Segmentation, LIU-IMT-TFK-A–19/570–SE

# Section 6

# Transfer learning

UPPSALA UNIVERSITET



# Transfer learning

- "Transfer knowledge between problems"
- Learning representations in $P_1$ will aid generalization in $P_2$

# Transfer learning

- "Transfer knowledge between problems"
- Learning representations in $P_1$ will aid generalization in $P_2$
- A Bayesian perspective: A strong prior

# Learning Representations for Images

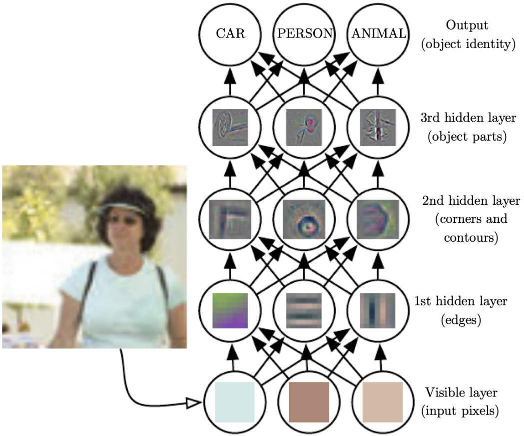



Figure: Learning representations can be crucial (Goodfellow et al, 2017, Fig. 1.2)

# Transfer learning

- In practice: Transfer/reuse learned weights or rather some weights

# Transfer learning

- In practice: Transfer/reuse learned weights or rather some weights
- Use (large) pre-trained models for smaller problems
- A reason for the success of CNN

# Transfer learning

- In practice: Transfer/reuse learned weights or rather some weights
- Use (large) pre-trained models for smaller problems
- A reason for the success of CNN
- Two types of transfer learning in Neural Networks:
  - Feature extraction (use pre-trained networks for features)
  - Fine Tuning (adapt pre-trained features)

UPPSALA UNIVERSITET



# Feature Extraction

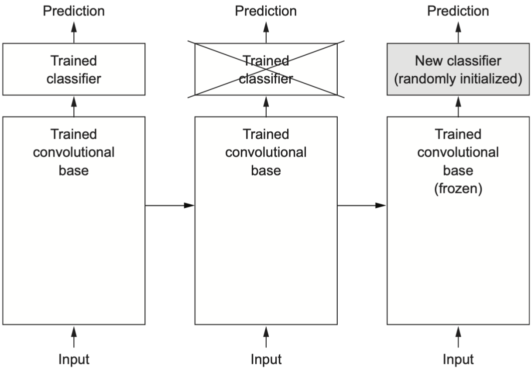



Figure: Using convnets as base for feature extraction (Chollet and Allaire, 2018, Fig 5.12)

# Fine-Tuning

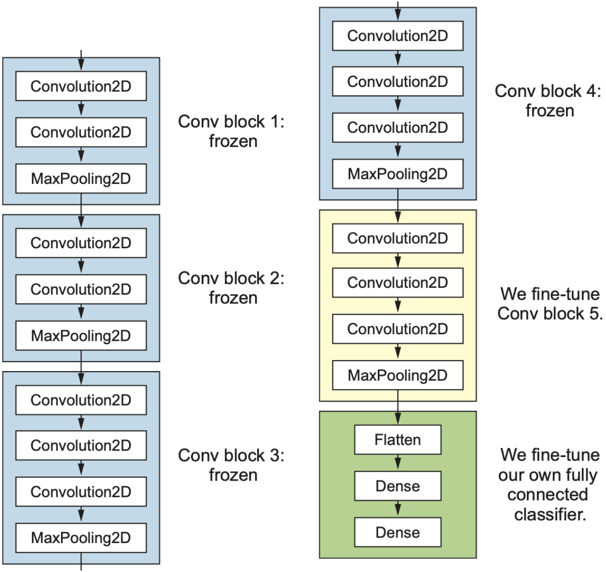



Figure: Finetuning a convolutional base (Chollet and Allaire, 2018, Fig 5.15)

# Transfer learning

- Catastrophic forgetting

# Transfer learning

- Catastrophic forgetting
- Domain adaptation: Same problem but at different input dataset
  (e.g. language models for legal/medical/political data)

# Transfer learning

- Catastrophic forgetting
- Domain adaptation: Same problem but at different input dataset
  (e.g. language models for legal/medical/political data)
- Concept drift: Similar problem

UPPSALA UNIVERSITET



# Transfer learning

- Catastrophic forgetting
- Domain adaptation: Same problem but at different input dataset
  (e.g. language models for legal/medical/political data)
- Concept drift: Similar problem
- Previously, popular with unsupervised pre-training.

# Transfer Learning: Practical Strategy

- Step 1: Load a pretrained CNN (e.g., ResNet, EfficientNet) trained on ImageNet.

# Transfer Learning: Practical Strategy

- Step 1: Load a pretrained CNN (e.g., ResNet, EfficientNet) trained on ImageNet.
- Step 2: Freeze the convolutional base Train only the new classification head

UPPSALA UNIVERSITET



# Transfer Learning: Practical Strategy

- Step 1: Load a pretrained CNN (e.g., ResNet, EfficientNet) trained on ImageNet.
- Step 2: Freeze the convolutional base Train only the new classification head
- Step 3: Train the head first Use a higher learning rate (e.g., $10^{-3}$).

# Transfer Learning: Practical Strategy

- Step 1: Load a pretrained CNN (e.g., ResNet, EfficientNet) trained on ImageNet.
- Step 2: Freeze the convolutional base Train only the new classification head
- Step 3: Train the head first Use a higher learning rate (e.g., $10^{-3}$).
- Step 4: Unfreeze top convolutional blocks Fine-tune with a *small* learning rate (e.g., $10^{-5}$) to avoid destroying pretrained features.

# Transfer Learning: Practical Strategy

- Step 1: Load a pretrained CNN (e.g., ResNet, EfficientNet) trained on ImageNet.
- Step 2: Freeze the convolutional base Train only the new classification head
- Step 3: Train the head first Use a higher learning rate (e.g., $10^{-3}$).
- Step 4: Unfreeze top convolutional blocks Fine-tune with a *small* learning rate (e.g., $10^{-5}$) to avoid destroying pretrained features.
- *When it works best:*
  - You have limited data
  - Your task is visually similar to pretrained data (e.g. ImageNet)
  - You want fast convergence and stable training
- *When not to fine-tune:* Extremely small datasets $\rightarrow$ risk of overfitting.

# Section 7

## Practical Methodology

# Practical Methodology



1. Determine your goals

# Practical Methodology

1. Determine your goals
2. Setup your baseline
   (establish a working end-to-end pipeline as soon as possible)

# Practical Methodology

1. Determine your goals
2. Setup your baseline
   (establish a working end-to-end pipeline as soon as possible)
3. Diagnose your networks performance

# Practical Methodology

1. Determine your goals
2. Setup your baseline
   (establish a working end-to-end pipeline as soon as possible)
3. Diagnose your networks performance
4. Make incremental improvements

- General idea: Increase data and model capacity until goal is reached

# Practical Methodology

1. Determine your goals
2. Setup your baseline
   (establish a working end-to-end pipeline as soon as possible)
3. Diagnose your networks performance
4. Make incremental improvements

- General idea: Increase data and model capacity until goal is reached
- End goal: Good enough performance on test set

# 1. Determine your goals

- Why are you building a model?

UPPSALA UNIVERSITET



# 1. Determine your goals

- Why are you building a model?
- Set up the metric based on the overall goal of the system! This may need multiple metrics.

# 1. Determine your goals

- Why are you building a model?
- Set up the metric based on the overall goal of the system! This may need multiple metrics.
- What is good enough? Remember the Bayes error!

UPPSALA UNIVERSITET



# 1. Determine your goals

- Why are you building a model?
- Set up the metric based on the overall goal of the system! This may need multiple metrics.
- What is good enough? Remember the Bayes error!
- What performance can you expect?

UPPSALA UNIVERSITET



# 1. Determine your goals

- Why are you building a model?
- Set up the metric based on the overall goal of the system! This may need multiple metrics.
- What is good enough? Remember the Bayes error!
- What performance can you expect?
- Some errors are worse than others, e.g. spam filters.

UPPSALA UNIVERSITET



# 1. Determine your goals

- Why are you building a model?
- Set up the metric based on the overall goal of the system! This may need multiple metrics.
- What is good enough? Remember the Bayes error!
- What performance can you expect?
- Some errors are worse than others, e.g. spam filters.
- Handling of uncertain predictions:
- Coverage: How large proportions can the system predict?

UPPSALA UNIVERSITET



# 1. Determine your goals

- Why are you building a model?
- Set up the metric based on the overall goal of the system! This may need multiple metrics.
- What is good enough? Remember the Bayes error!
- What performance can you expect?
- Some errors are worse than others, e.g. spam filters.
- Handling of uncertain predictions:
- Coverage: How large proportions can the system predict?
- Manual curation can be faster and easier.

# 2. Setup your baseline

- Start with...
  - the most simple possible model (logistic regression)

# 2. Setup your baseline

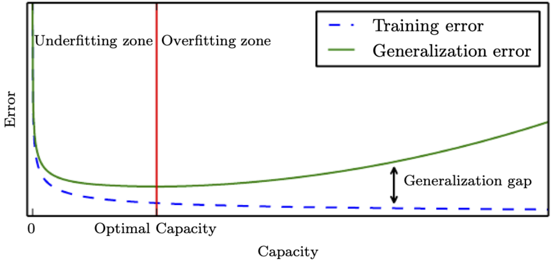
- Start with...
  - the most simple possible model (logistic regression)
  - previous approaches/baselines

# 2. Setup your baseline

- Start with...
  - the most simple possible model (logistic regression)
  - previous approaches/baselines
  - a simple neural network that is common in the domain/defaults
    (CNN for images, Adam as optimizer)

# Remember: Statistical Learning Theory

Figure: Test, training, and model complexity (Goodfellow et al, 2017, Figure 5.3)



# 3. Diagnose and improve

- High training loss: Training data not fully used
  Neural network generally performs best when training error is low

UPPSALA
UNIVERSITET



# 3. Diagnose and improve

- High training loss: Training data not fully used
  Neural network generally performs best when training error is low
- High test loss: Low data quality, i.e. large Bayes error?

UPPSALA UNIVERSITET



# 3. Diagnose and improve

- High training loss: Training data not fully used
  Neural network generally performs best when training error is low
- High test loss: Low data quality, i.e. large Bayes error?
- Low training error and high test error: Common situation

UPPSALA UNIVERSITET



# 3. Diagnose and improve

- High training loss: Training data not fully used
  Neural network generally performs best when training error is low
- High test loss: Low data quality, i.e. large Bayes error?
- Low training error and high test error: Common situation
  - You can always improve by gather more data, or
  - Regularize to optimal capacity

# 3. Diagnose and improve

- High training loss: Training data not fully used
  Neural network generally performs best when training error is low
- High test loss: Low data quality, i.e. large Bayes error?
- Low training error and high test error: Common situation
  - You can always improve by gather more data, or
  - Regularize to optimal capacity
- How to know the improvements of additional data?

# Learning curve

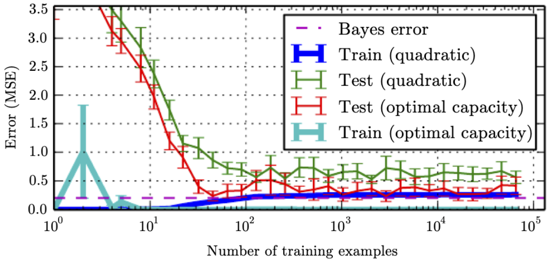



Figure: Learning curve to assess the need for more data (Goodfellow et al., 2017, Fig 5.4)

# 3. Diagnosing and improving your model: Regularization

- Best performance: Larger model that is regularized well.
- Warning: Avoid the algorithm rabbit hole

# 3. Diagnosing and improving your model: Regularization

- Best performance: Larger model that is regularized well.
- Warning: Avoid the algorithm rabbit hole
- Adapt hyperparameters to get optimal capacity

UPPSALA UNIVERSITET

UPPSALA UNIVERSITET



# 3. Diagnosing and improving your model: Regularization

- Best performance: Larger model that is regularized well.
- Warning: Avoid the algorithm rabbit hole
- Adapt hyperparameters to get optimal capacity
- Hyperparameter Search Goal:
  adjust the model capacity to match the complexity of the task.

UPPSALA UNIVERSITET



# 3. Diagnosing and improving your model: Regularization

- Best performance: Larger model that is regularized well.
- Warning: Avoid the algorithm rabbit hole
- Adapt hyperparameters to get optimal capacity
- Hyperparameter Search Goal:
  adjust the model capacity to match the complexity of the task.
- Marginal hyperparameter has a U-shaped error function (ideally)

# 3. Diagnosing and improving your model: Regularization

- Best performance: Larger model that is regularized well.
- Warning: Avoid the algorithm rabbit hole
- Adapt hyperparameters to get optimal capacity
- Hyperparameter Search Goal:
  adjust the model capacity to match the complexity of the task.
- Marginal hyperparameter has a U-shaped error function (ideally)
- Neural Networks Steps:
  1. Get good training error. E.g. by tune learning rate and increasing capacity

# 3. Diagnosing and improving your model: Regularization

- Best performance: Larger model that is regularized well.
- Warning: Avoid the algorithm rabbit hole
- Adapt hyperparameters to get optimal capacity
- Hyperparameter Search Goal:
  adjust the model capacity to match the complexity of the task.
- Marginal hyperparameter has a U-shaped error function (ideally)
- Neural Networks Steps:
  1. Get good training error. E.g. by tune learning rate and increasing capacity
  2. Tune hyperparameters (regularization):
     Requires monitoring both training and test error

# Hyper-parameter optimization

- Neural networks has many hyperparameters
- What is a hyperparameter in a neural network?

UPPSALA UNIVERSITET



# Hyper-parameter optimization

- Neural networks has many hyperparameters
- What is a hyperparameter in a neural network?
- Grid search:
  - Setup a grid of potential values
  - For 1-4 hyperparameter: grid search can work well
  - Usually iterative/repeated grid search is best: start with three values, and work iteratively
  - Grows exponentially with the number of parameters

# Hyper-parameter optimization



- Neural networks has many hyperparameters
- What is a hyperparameter in a neural network?
- Grid search:
  - Setup a grid of potential values
  - For 1-4 hyperparameter: grid search can work well
  - Usually iterative/repeated grid search is best: start with three values, and work iteratively
  - Grows exponentially with the number of parameters
- Random search
  - Specify a marginal distribution for each hyperparameter (additional work)
  - Common: uniform on log scale
  - Can also be done iteratively
  - Can be exponentially more efficient
  - Generally reduce the error faster in setting with many hyper parameters

# Diagnosis and Debugging

- Visualize the model predictions

UPPSALA UNIVERSITET



# Diagnosis and Debugging

- Visualize the model predictions
- Analyze the worst errors: why?

UPPSALA UNIVERSITET



# Diagnosis and Debugging

- Visualize the model predictions
- Analyze the worst errors: why?
- Use train and test error as a diagnostic: Can you overfit the data?

UPPSALA UNIVERSITET

- Practicalities
- Introduction
- Computer Vision
- Convolution
- Convolutional Neural Networks
  — The Convolution Layer
  — The Pooling Layer
  — Regularization
  — Examples
- Transfer learning
- Practical Methodology

# Diagnosis and Debugging

- Visualize the model predictions
- Analyze the worst errors: why?
- Use train and test error as a diagnostic: Can you overfit the data?
- Use test suites: Can you get known results on a toy data. Both training error and derivatives

UPPSALA UNIVERSITET



# Diagnosis and Debugging

- Visualize the model predictions
- Analyze the worst errors: why?
- Use train and test error as a diagnostic: Can you overfit the data?
- Use test suites: Can you get known results on a toy data. Both training error and derivatives
- Monitor the gradients

# Diagnosis and Debugging

- Visualize the model predictions
- Analyze the worst errors: why?
- Use train and test error as a diagnostic: Can you overfit the data?
- Use test suites: Can you get known results on a toy data. Both training error and derivatives
- Monitor the gradients
- Monitor activation function statistics. Are some never activated?