Minicourse: Hands-on Deep Learning and Transfer Learning

Dr. Nghia Duong-Trung

Technische Universität Berlin, Faculty III Process Sciences, Institute of Biotechnology, Chair of Applied and Molecular Microbiology

Outline

1. Introduction

2. Image Preprocessing to Use in Machine Learning Models

3. Introducing Convolutional Neural Networks

4. Building Convolutional Neural Networks for Image Classification

5. Transfer Learning

6. References

Outline

1. Introduction

2. Image Preprocessing to Use in Machine Learning Models

3. Introducing Convolutional Neural Networks

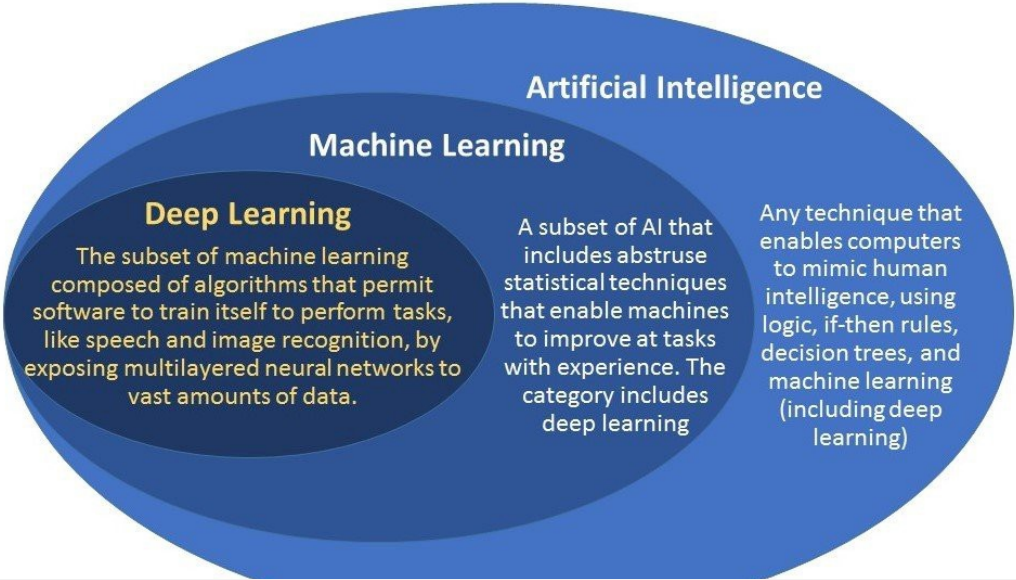
4. Building Convolutional Neural Networks for Image Classification

5. Transfer Learning

6. References

Contents

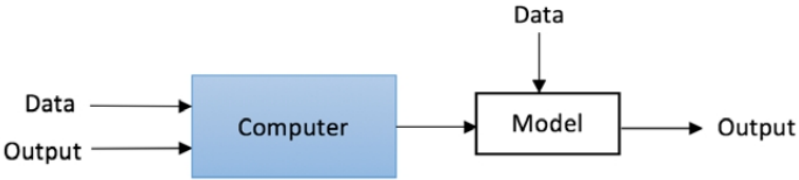
- Terms: AI, ML, DL
- Neuron
- Training vs. inference


Artificial Intelligence
Machine Learning
Deep Learning
The subset of machine learning composed of algorithms that permit software to train itself to perform tasks, like speech and image recognition, by exposing multilayered neural networks to vast amounts of data.
A subset of AI that includes abstruse statistical techniques that enable machines to improve at tasks with experience. The category includes deep learning
Any technique that enables computers to mimic human intelligence, using logic, if-then rules, decision trees, and machine learning (including deep learning)

- AI is the science and engineering of creating intelligent machines that have the ability to achieve goals like humans do
- ML is the field of study that gives computers the ability to learn without being explicitly programmed
- DL is a subfield of machine learning concerned with algorithms inspired by the structure and function of the human brain
  - Although scientists are still exploring the details of how the brain works, it is generally believed that the main computational element of the brain is the neuron

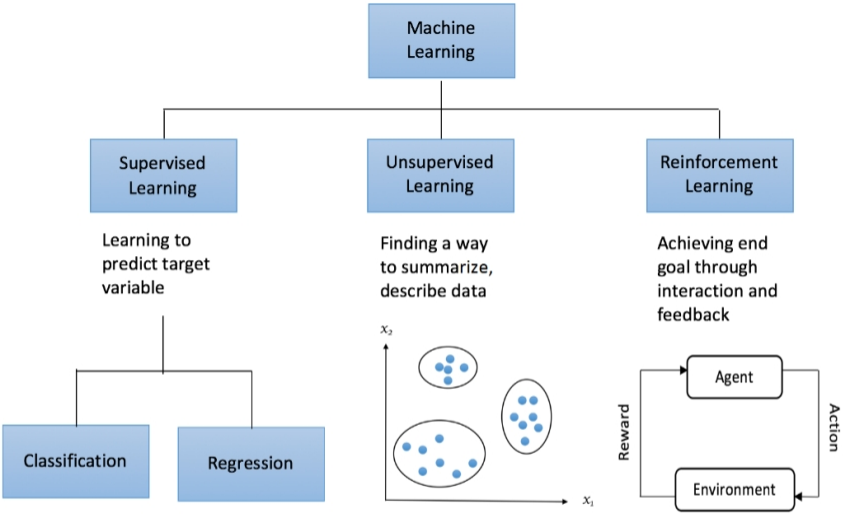
Machine learning vs traditional programming

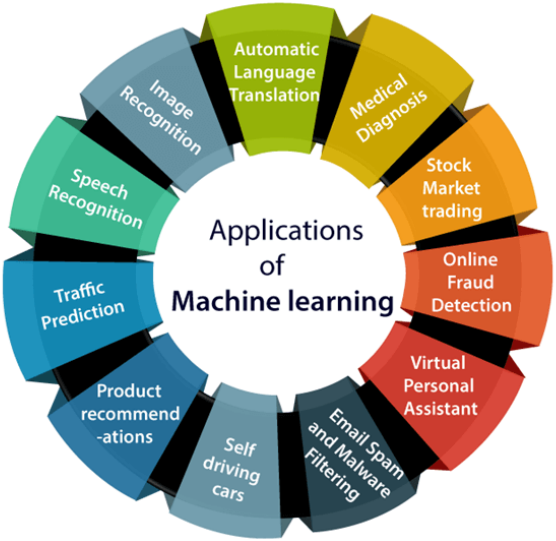
Traditional Programming



Machine Learning

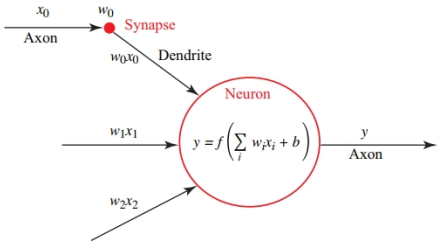
Machine learning tasks




Applications of Machine learning
Automatic Language Translation
Medical Diagnosis
Stock Market trading
Online Fraud Detection
Virtual Personal Assistant
Email Spam and Malware Filtering
Self driving cars
Product recommend -ations
Traffic Prediction
Speech Recognition
Image Recognition

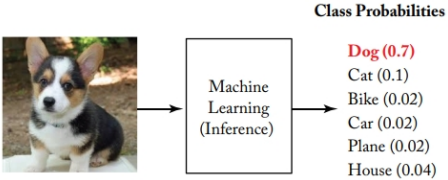
Neuron

- The neuron accepts the signals entering it via the dendrites, performs a computation on those signals, and generates a signal on the axon.
- These input and output singals are referred to as activations.
- The connections between a branch of the axon and a dendrite is called a synapse.
- x_i, w_i, $f(\cdot)$ and b are the activations, weights, nonlinear function, and bias, respectively.
  - Note: Some notations will be further re-defined later.



Training vs. inference

- Since deep neural networks (DNNs) are an instance of machine learning algorithms, the basic program does not change as it learns to perform its given tasks.
- The learning involves determining the value of the weights (and biases) in the network, and is referred to as training the network.
- Once trained, the program can perform its task by computing the output of the network using the weights determined during the training process. Running the program with these weights is referred to as inference.



Outline

1. Introduction

2. Image Preprocessing to Use in Machine Learning Models

3. Introducing Convolutional Neural Networks

4. Building Convolutional Neural Networks for Image Classification

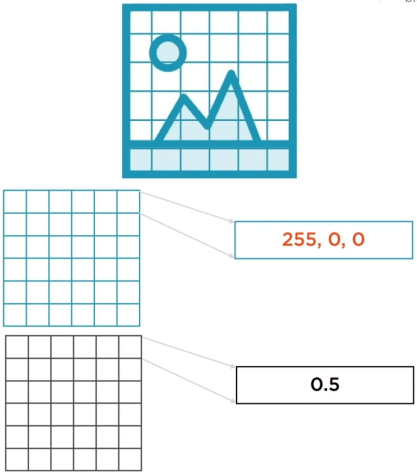
5. Transfer Learning

6. References

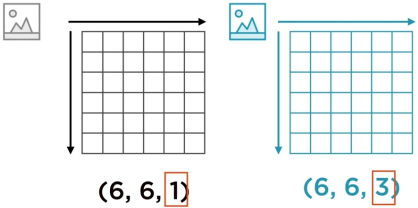
Contents

- Single channel and multi-channel images
- Preprocessing images to train robust model
  - Resizing, rescaling, cropping, denoising
  - Standardizing, transforming and normalizing
- Hands-on the codes (appox. 20 minutes)

- An image can be represented as grids of pixels
- Each pixel holds a value based on the type of image: gray scale, RGB image
  - RGB: 3 values to represent color, 3 channels
  - Gray scale: each pixel represents only intensity information; hence, 1 value to represent intensity, 1 channel
  - These are often scaled to be in the 0-1 range as neural networks work better with smaller numbers



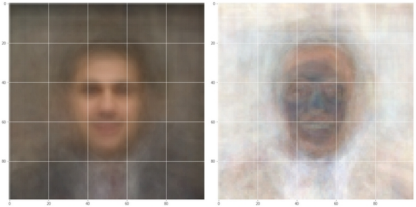
- Images can be represented by a 3-D matrix: height x width x channel
- The number of channels specifies the number of elements in the 3rd dimension
- Deep learning frameworks usually deal with a list of images on one 4-D tensor
  - The first dimension is the batch size: the number of images
  - All images should have the same size
  - Ex: (10,6,6,3)



- Common techniques to improve CNN performance [1]
  - Uniform aspect ratio
  - Uniform image size
  - Mean, standard deviation
  - Normalized input images
  - Dimensionality reduction
  - Data augmentation

[1] https://becominghuman.ai/image-data-pre-processing-for-neural-networks-498289068258

- Uniform aspect ratio
  - The ratio of the width to the height of the image
  - Most models assume square shape
  - Crop images to be square
- Uniform image size
  - Fit image size to CNN feature maps
  - Ex: 250 x 250 image to 100 x 100 image
  - Up-scaling and down-scaling
- Mean, standard deviation
  - look at the 'mean image' obtained by taking the mean values for each pixel across all training examples
  - insight into some underlying structure in the images



Images depicting mean (left) and standard deviation (right) of the set of data inputs

- Normalized input images
  - Ensures each pixel has similar data distribution
  - Subtract mean, divide by standar deviation
- Dimensionality reduction
  - reduce RGB data to grayscale
- Data augmentation
  - Scaling, rotation, affine transforms
  - Makes CNN training more robust
  - Reduces risk of overfitting

- Hand-on the codes
  - Implement image pre-processing using Pytorch
  - See ImagePreprocessingPyTorch.ipynb at `https://github.com/duongtrung/minicourse-dl-bio`
    - Normalization
    - transforms in Pytorch
    - Apply transformation on CIFAR10 datasets

Outline

1. Introduction

2. Image Preprocessing to Use in Machine Learning Models

3. Introducing Convolutional Neural Networks

4. Building Convolutional Neural Networks for Image Classification

5. Transfer Learning

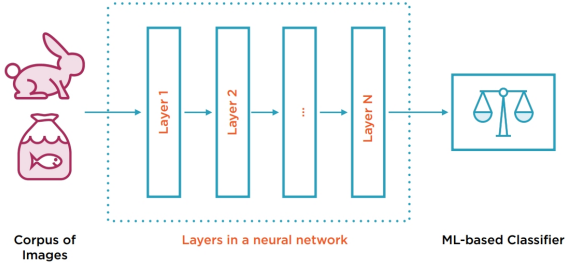
6. References

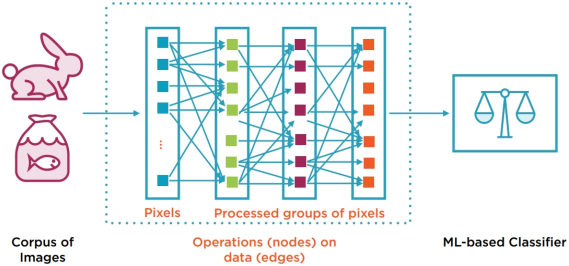
Contents

- Understanding the drawbacks of using deep neural networks with images
- Local receptive fields
- Understanding convolution
- Convolutional layers
- Pooling layers
- Typical CNN architecture
- Applying convolutional and pooling layers

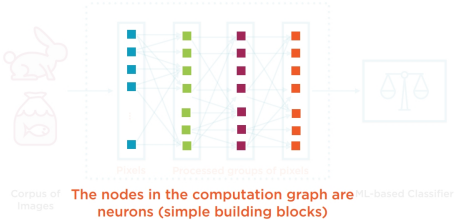
The drawbacks of using deep neural networks

- Neural networks consist of many layers of neurons that learn from your data
- The first and last layers are the ones you interact with: feeding input to the first layer and getting output from the last layer $->$ visible layers
- Many hidden layers in between
- Each layer has a number of neurons that are indiviual interconnected

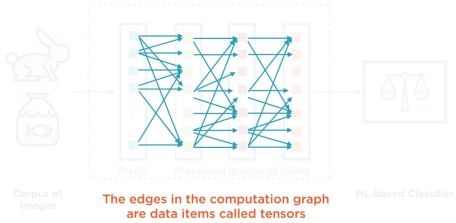




- Computation graph
  - The nodes are active learning units
  - The edges are the data loaded as tensors
  - Each connection is associated with a weight and possiby a bias
- Parameter explosion
  - Consider a 100 x 100 pixel image (10000 pixels)
  - If first layer = 10000 neurons
  - Interconnections O(10000 * 10000)
  - 100 million parameters to train neural network
- **Deep neural networks do not consider the spatial aspects of images**



The nodes in the computation graph are neurons (simple building blocks)



The edges in the computation graph are data items called tensors

Hands-on the codes

- Implement image classsification using a deep, fully-connected neural network
- See MNISTImageClassificationUsingDNN.ipynb
- Approx. 45 minutes

Insprations for CNNs

- Convolution layers: zoom in on specific bits in input
- Successive layers aggregate inputs into higher lavel features: pixels > lines > edges > object
- Dramatically fewer parameters than DNN with similar performance

Two Dimensions

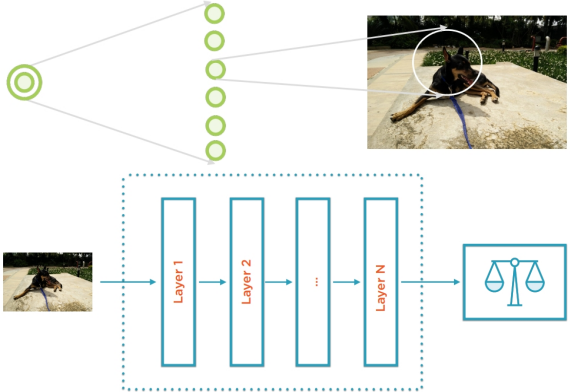
Data comes in expressed in 2D

Local Receptive Fields

Neurons focus on narrow portions

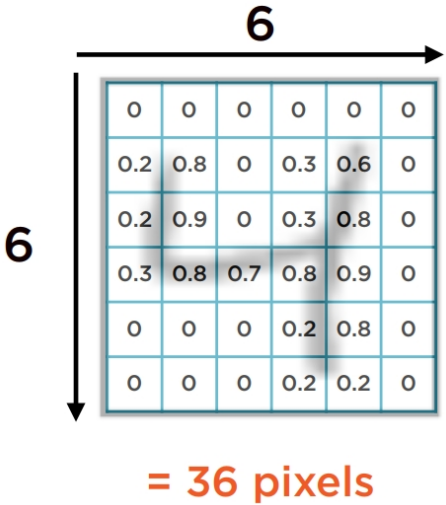
Understanding convolution

- Each neuron has its own local receptive field
- It reacts only to visual stimuli located in its receptive field
- Some neurons react to more complex patterns that are combinations of lower level patterns
- Convolution layer: where neurons have local receptive fields and they only react with patterns in those fields
- CNNs are **sparse** neural networks



Understanding convolution

- In this context, convolution refers to a sliding window function applied to a matrix
  - a matrix of pixels representing an image
  - a function: often called kernel or filter
  - a sliding window: kernel is applied element-wise in sliding-window fashion



0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8	0.9	0
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

Matrix

3 × 3

1	0	1
0	1	0
1	0	1

Kernel

0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8	0.9	0
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

0	0	0	0	0	0
0.2	0.8	0	0.3	0.6	0
0.2	0.9	0	0.3	0.8	0
0.3	0.8	0.7	0.8	0.9	0
0	0	0	0.2	0.8	0
0	0	0	0.2	0.2	0

x1	x0	x1
x0	x1	x0
x1	x0	x1

3 × 3

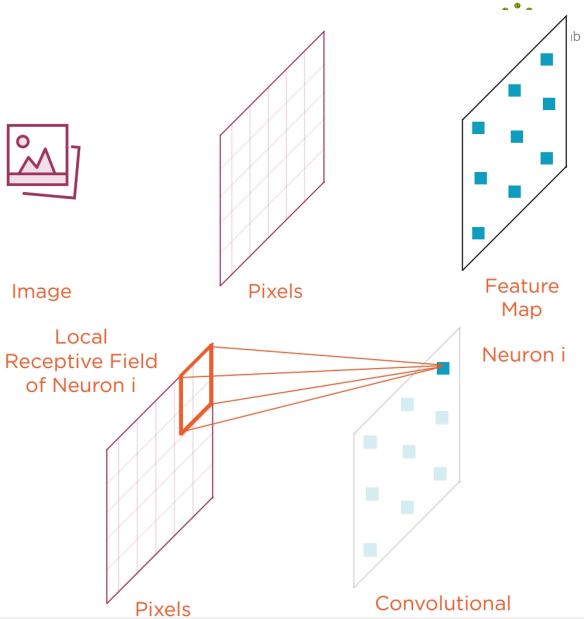
x1	x0	x1
x0	x1	x0
x1	x0	x1

4 × 4

1	1.2	1.1	0.9
1.9	2.7	2.5	1.9
1.0	2.1	2.4	1.4
1.0	1.8	2.0	1.8

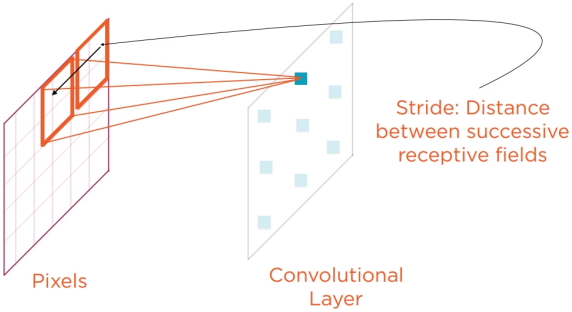
Feature maps

- Feature maps are covolutional layers generated by applying a convolutional kernel to the input
- Convolutional kernel size usually expressed in terms of width and height of receptive field
- Use small convolutional kernels, more efficient. Ex: 3x3 kernels



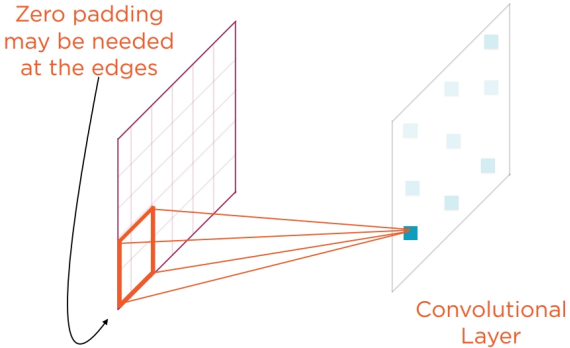
Feature maps

- Stride: horizontal and vertical



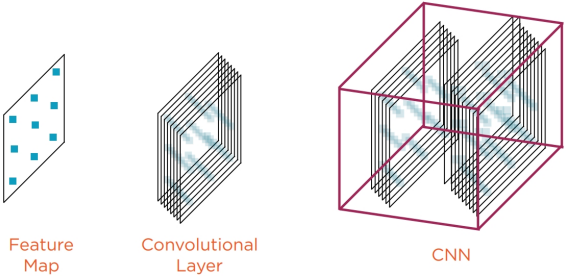
Feature maps

- Padding
  - Convolutional kernel will not stride off the edges. If you want the feature map to be as the same size as the input matrix, you might need zero padding



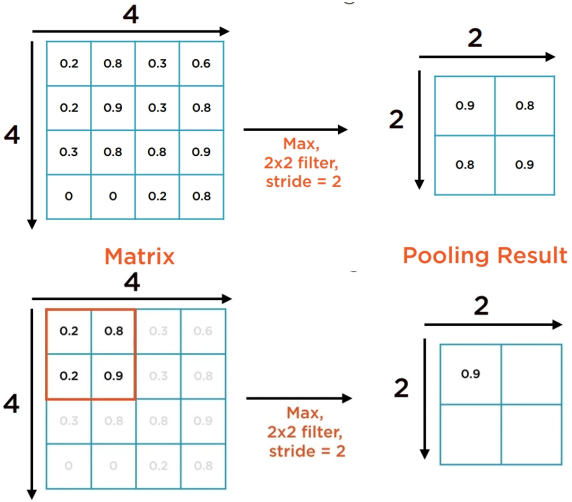
Convolutional layers

- Each convolutional layer consists of several feature maps of equal sizes
- The different feature maps have different parameters



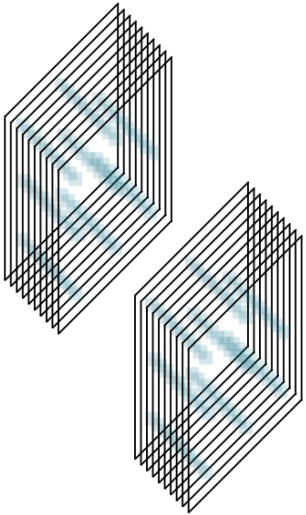
Pooling layers

- The input to a pooling layer is essentially a matrix of pixels
- The pooling layer applies an aggregation operation (max, min, or average) to the input
- The pooling filter that we apply to the input matrix can be thought as no weights and biases



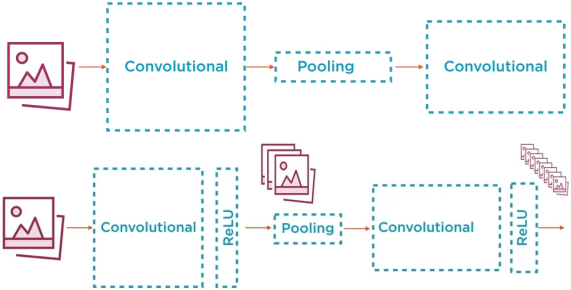
Pooling layers

- Greatly reduce memory usage during training
- Mitigate overfitting
- Make NN recognize features independent of location



Typical CNN architecture

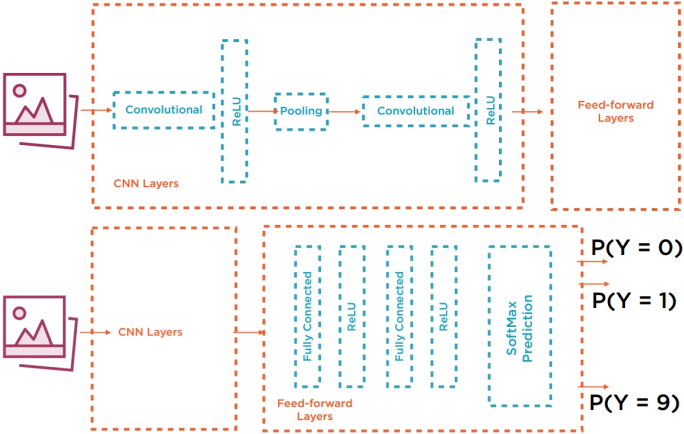
- Standard: alternating groups of convolutional and pooling layers
- Each group of convolutional layers usually followed by an activation function (ReLU,...) [a]
- The oupput of each layer is also an image
- Successive outputs are smaller and smaller (due to pooling layers)



[a] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

Typical CNN architecture

- The entire set of layers is then fed into a regular, feed-forward NN
- This feed-forward has a few fully connected layers with ReLU activation
- Output layer: emitting probabilities



Hands-on the codes

- Approx. 30 minutes
- See UnderstandingConvolutionFilters.ipynb

Outline

1. Introduction

2. Image Preprocessing to Use in Machine Learning Models

3. Introducing Convolutional Neural Networks

4. Building Convolutional Neural Networks for Image Classification

5. Transfer Learning

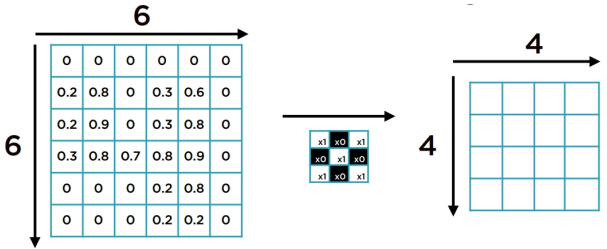
6. References

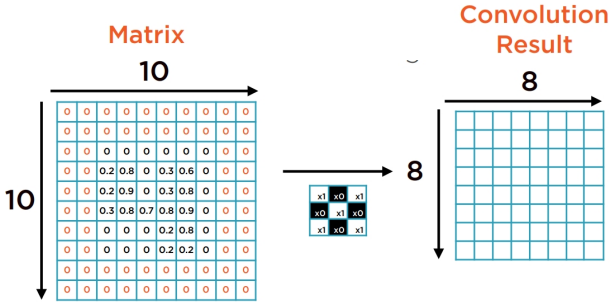
Contents

- Zero padding and stride size
- Batch normalization
- Activation functions
- Setting up a CNN
- Training a CNN
- Hyperparameter tuning

Zero padding

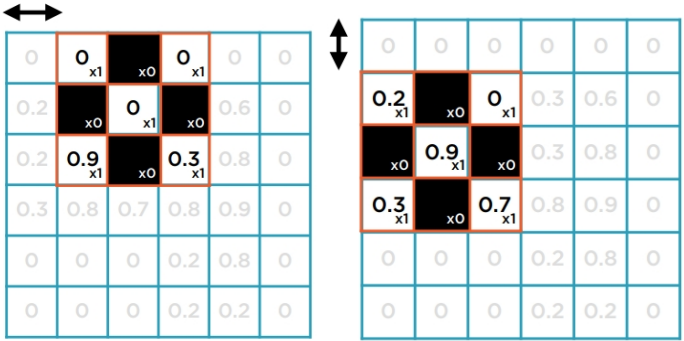
- Narrow convolution: little zero padding; output narrower than input
- Wide convolution: lots of zero padding: output wider than input
- With zero-padding, every element of matrix will be passed into filter





Stride size

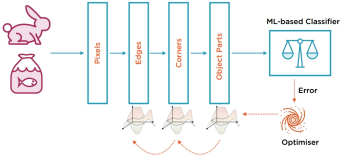
- How much the filter will be moved horizontally and vertically



Batch normalization

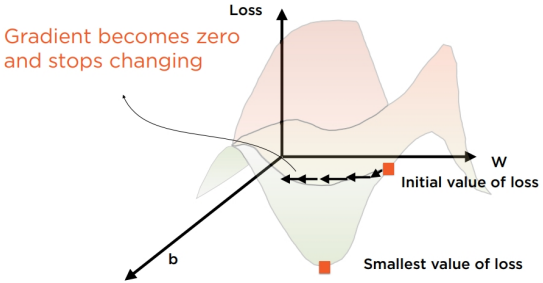
- Apply batch normalization to the output of convolution layers before passing them into a activation function
  - Most benefits: mitigate vanishing and exploding gradients during training

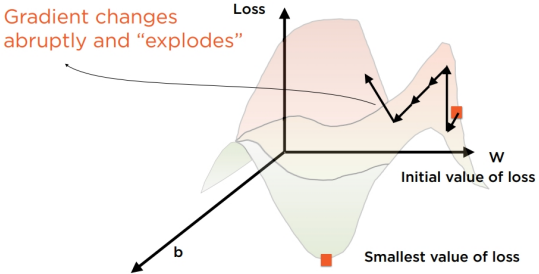
- Training via back propagation
  - Forward pass: use the current parameters' values to give predicted values
  - Compare predicted values with ground-truth and calculate the errors
  - Use an optimizer calculating gradients with repect to this errors
  - Use gradients to make a backward pass to update the model's parameters



Batch normalization

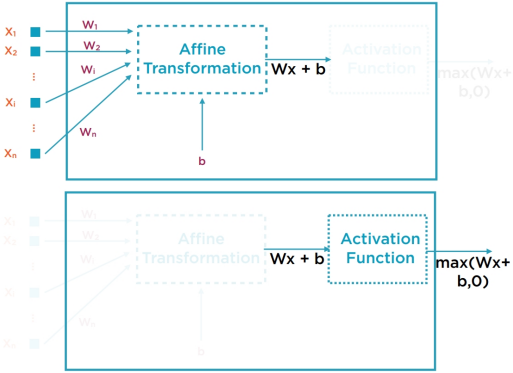
- Back propagation fails if
  - Gradients are vanishing
  - Gradients are exloding
- Options
  - Batch normalization
  - Proper initialization
  - Non-saturating activation
  - Gradient clipping
- To do
  - First, normalize inputs
  - Second, scale and shift inputs





Choice of activation function

- One a NN is trained all edges have weights which help it make predictions
- Each neuron only applies two single functions to its inputs
- The affine transformation is just a weighted sum with a bias added: $W_1x_1 + W_2x_2 + \cdots + W_nx_n + b$
- The combination of the affine transformation and the activation function can learn any arbitrary relationship
  - ReLU, logit, tanh, step



Feature map size calculation

$$O = \frac{W - K + 2P}{S} + 1 \tag{1}$$

- O = output dimension that refers to either height or width of the output
- W = input dimension that refers to either height or width of the input image
- K = kernel size, height/width of kernel
- P = padding (if any), maybe zero
- S = stride

Hands-on the codes

- Image classification using CNNs
- Hyperparameters tuning
- Appprox. 45 minutes
- See CNNs_HyperparameterTuning.ipynb

Hands-on the codes

- Image classification on CIFAR-10 dataset using CNNs
- Choosing activation functions
- Appprox. 45 minutes
- See ActivationLayers.ipynb

Hands-on the codes

- Image classification on CIFAR-10 dataset using CNNs
- Choosing pooling layers
- Appprox. 30 minutes
- See PoolingLayers.ipynb

Hands-on the codes

- Image classification on CIFAR-10 dataset using CNNs
- Choosing different convolution kernel sizes
- Appprox. 20 minutes
- See KernelSize.ipynb

Outline

1. Introduction

2. Image Preprocessing to Use in Machine Learning Models

3. Introducing Convolutional Neural Networks

4. Building Convolutional Neural Networks for Image Classification

5. Transfer Learning

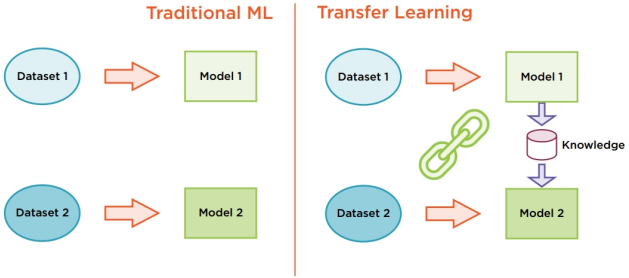
6. References

Contents

- Transfer learning
- Source and target domains, source and target tasks
- Transfer learning scenarios
- Using the VGG16 model as a fixed feature extractor
- Training the final layer fore prediction
- Freezing lower layers and fine-tuning weights of top layers
- Some Pre-trained Models (VGG, ResNet, Inception, MobileNet)

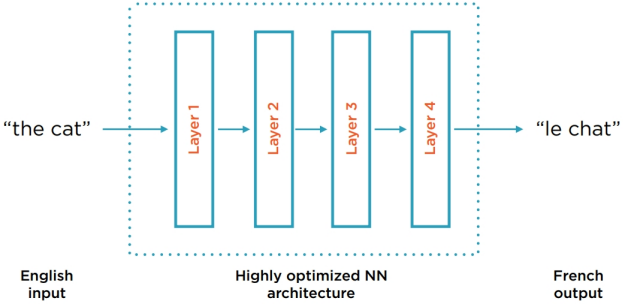
Transfer learning

- The practice of re-using a trained neural network that solves a problem similar to yours, usually leaving the network architecture unchanged and re-using some of or all of the model weights.
  - Avoid designing NN architecture from scratch
  - Only makes sense for common, widely studied use-cases in which basic problem structure stays same, but details vary
  - Re-train from scratch, fine-tune model weights, use entirely as-is

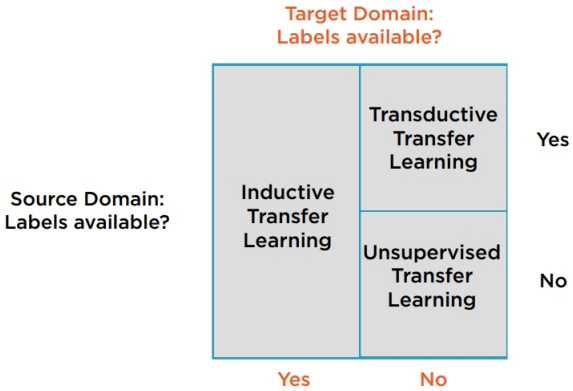


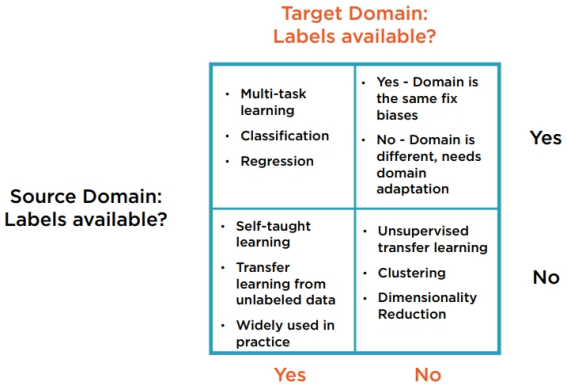
Source and target domains, source and target tasks

- Original model: English to French
- Transfer Learning: Spanish to German
- Domains refer to where the X variables are drawn from and how they are distributed
- Tasks refer to the Y variables and how they are related to X



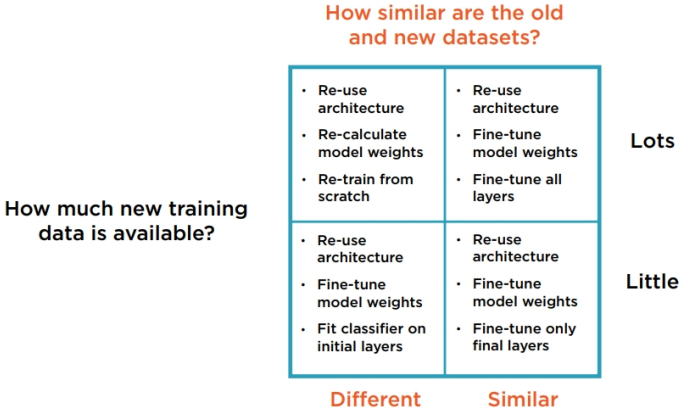
Source and target domains





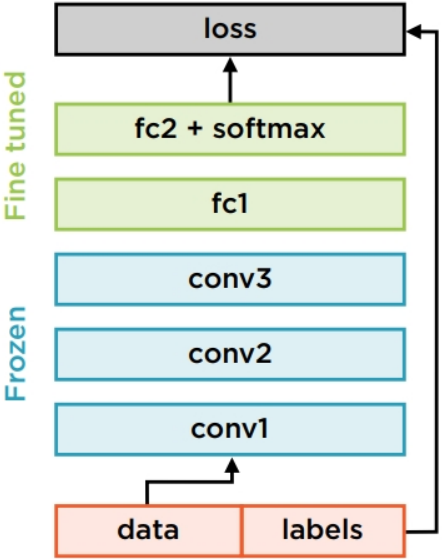
Scenarios in transfer learning

- Re-train from scratch
  - Find new model weights starting from scratch
  - Keep model architecture as-is
- Fine-tune model weights
  - Find new model weights starting from original model weights
  - Keep model architecture as-is



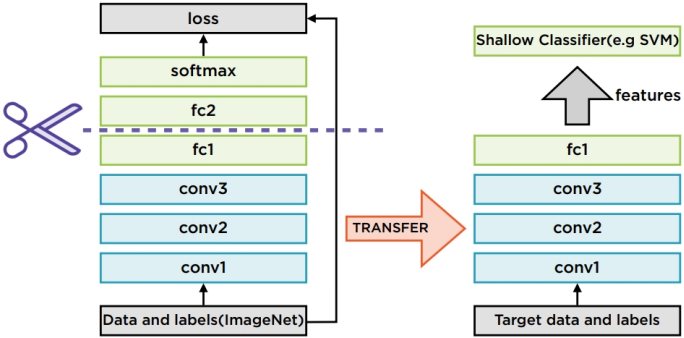
Freeze or fine-tune layers

- Usually the top (later) layers of the NN are more specific to the problem and will need to be tuned
- Bottom layers: freeze or fine-tune?
  - Frozen: not updated during training
  - Fine-tuned: updated during training
- Which to do depends on target task
  - Freeze: target task labels are scarce, and we want to avoid overfitting
  - Fine-tune: target task labels are more plentiful



Initial layers as feature extractors

- Several first layers extract more generic features from the input data
  - Can be used for fixed feature extractors
  - Freezed and add a shallow model on top



Transfer learning in PyTorch

- Support for several famous NN architectures, torchvision.models
  - AlexNet, VGG, ResNet, DenseNet, Inception and many others
  - `https://pytorch.org/vision/stable/models.html`
- PyTorch transfer learning models are trained on the ImageNet dataset
  - 14 million images with 20000 categories
  - Hand-annotated using crowdsourcing
  - ImageNet Large Scale Visual Recognition Challenge (ILSVRC)
  - PyTorch models trained on a subset with 1000 categories with $>$ 1 million images

Hands-on the codes

- Exlore several pre-trained models in PyTorch
- Approx. 30 minutes
- See ExploringPretrainedModelsInPyTorch.ipynb

Hands-on the codes

- Use a pre-trained model as a fixed feature extractor
  - All layers except the last of the pretrained model act as a fixed feature extractor for the new data
- Approx. 45 minutes
- See FixedFeatureExtractor.ipynb

Hands-on the codes

- Fine-tune top layers of the model
  - Freeze the initial layers of the pretrained model which contain more generic feature detectors and train higher layers which are more customized to your dataset
- Approx. 45 minutes
- See FixedFeatureExtractor.ipynb

Hands-on the codes

- Re-use model architecture while training it entirely from scratch
  - If you have a large dataset and the pretrained model trained on data that is very different from the data that you are going to use.
- Approx. 60 minutes (* if you train the model on CPU, the computation time is longer)
- See TrainingFromScratch.ipynb

Hands-on the codes

- Fine-tuning all layers of a pretrained model while starting from original pretrained weights
  - If you have a large dataset and the pretrained model trained on data that is very similar from the data that you are going to use.
- Approx. 60 minutes (* if you train the model on CPU, the computation time is longer)
- See FineTuningTheNetwork.ipynb

Assigmnent

- Let's make a small competition!
- Design your transfer learning strategies that work best on chest_xray and cats_or_dogs datasets.
- You can apply any techniques in the minicourse or your further reading.
- Submit your notebook on ISIS.

Outline

1. Introduction

2. Image Preprocessing to Use in Machine Learning Models

3. Introducing Convolutional Neural Networks

4. Building Convolutional Neural Networks for Image Classification

5. Transfer Learning

6. References

References

1. MIT 6.191 Introduction to Deep Learning
   - http://introtodeeplearning.com/
   - https://www.youtube.com/watch?v=5tvmMX8r_OM
2. Dive into Deep Learning
   - https://d2l.ai/
3. A Survey on Transfer Learning
   - https://ieeexplore.ieee.org/document/5288526
4. Yang, Q., Zhang, Y., Dai, W., & Pan, S. J. (2020). Transfer learning. Cambridge University Press.
5. V. Sze, Y.-H.Chen, T.-J. Yang, J. S. Emer. (2020). Efficient Processing of Deep Neural Networks". Synthesis Lectures on Computer Architecture, Morgan & Claypool Publishers.
6. https://www.pluralsight.com: Expediting Deep Learning with Transfer Learning: PyTorch Playbook
7. https://www.pluralsight.com: Image Classification Pytorch

References

Most figures in this Minicourse are taken from reference 5, 6, and 7. The credits go to the authors. The figures are used for the purpose of easy illustration only.