Convolutional Neural Networks

References:

- *introtodeeplearning.com*
- *Dive into Deep Learning: https://d2l.ai/chapter_convolutional-neural-networks/index.html*
- *Deep Learning Book: https://www.deeplearningbook.org*
- *deeplearningindaba.com*

What Computers “See”

Images are Numbers

Images are Numbers

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Images are Numbers

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

What the computer sees

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

Tasks in Computer Vision

Input Image

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Pixel Representation

classification

Lincoln	0.8
Washington	0.1
Jefferson	0.05
Obama	0.05

- **Regression**: output variable takes continuous value
- **Classification**: output variable takes class label. Can produce probability of belonging to a particular class

High Level Feature Detection

Let's identify key features in each image category

Nose,
Eyes,
Mouth

Wheels,
License Plate,
Headlights

Door,
Windows,
Steps

Manual Feature Extraction

Problems?

Manual Feature Extraction

Domain knowledge → Define features → Detect features to classify

Manual Feature Extraction

Domain knowledge → Define features → **Detect features to classify**

Learning Feature Representations

Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features

Edges, dark spots

Mid level features

Eyes, ears, nose

High level features

Facial structure

Learning Visual Features

Fully Connected Neural Network

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values

Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values

Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

Using Spatial Structure

Input: 2D image.
Array of pixel values

Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only "sees" these values.

Using Spatial Structure

Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

Applying Filters to Extract Features

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) Spatially **share** parameters of each filter
(features that matter in one part of the input should matter elsewhere)

Feature Extraction with Convolution

- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

Feature Extraction and Convolution

A Case Study

X or X?

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

$\stackrel{?}{=}$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

Image is represented as matrix of pixel values… and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

Features of X

Filters to Detect X Features

filters

The Convolution Operation

The Convolution Operation

Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

image

$\otimes$

1	0	1
0	1	0
1	0	1

filter

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs…

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:

Producing Feature Maps

Original

Sharpen

Edge Detect

"Strong" Edge Detect

Comparison: Dense Layer and Convolutional Layer

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

Neural networks ⇒ Convolutional networks

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

$$h_i = \sum_j w_j x_{j+i} + b$$

Neural networks ⇒ Convolutional networks

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

$$h_i = \sum_j w_j x_{j+i} + b$$

Neural networks ⇒ Convolutional networks

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

$$h_i = \sum_j w_j x_{j+i} + b$$

Neural networks ⇒ Convolutional networks

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

dense connectivity
vs
sparse connectivity

In our example:
5x5 multiplications (naïve implementation)
vs
5x3 multiplications

Sparse connectivity scales better
e.g. 25x25 vs 25x3

$$h_i = \sum_j w_j x_{j+i} + b$$

Neural networks ⇒ Convolutional networks

$$\boldsymbol{h} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b};\ h_i = \sum_j w_{ij} x_j + b_i$$

unshared vs shared weights:

In our example:
5x5 vs 3 weights

shared weights scale way better:
e.g. 25x25 vs 3

$$h_i = \sum_j w_j x_{j+i} + b$$

Translation equivariance

x_i $\quad$ $x_i' = x_{i-1}$ $\quad$ $h_i' = h_{i-1}$

© 2017 NAVER LABS. All rights reserved.

Translation equivariance

Why is translation equivariance useful? Helps make predictions **translation invariant**

CNN → “butterfly”

CNN → “butterfly”

CNN → “butterfly”

What other types of invariance could be useful?

© 2017 NAVER LABS. All rights reserved.

Pooling

Operation to aggregate or “summarize” sub-region of input.

© 2017 NAVER LABS. All rights reserved.

Pooling

Operation to aggregate or “summarize” sub-region of input.

$$p = f(\boldsymbol{a})$$

$$f_{max}(\boldsymbol{a}) = \max_i(a_i)$$

$$f_{avg}(\boldsymbol{a}) = \frac{1}{N}\sum_{i=1}^{N} a_i$$

© 2017 NAVER LABS. All rights reserved.

Pooling

Operation to aggregate or “summarize” sub-region of input.

$$p = f(\boldsymbol{a})$$

$$f_{max}(\boldsymbol{a}) = \max_i(a_i)$$

$$f_{avg}(\boldsymbol{a}) = \frac{1}{N}\sum_{i=1}^{N} a_i$$

© 2017 NAVER LABS. All rights reserved.

Pooling

$$p = f(\boldsymbol{a})$$

$$f_{max}(\boldsymbol{a}) = \max_i(a_i)$$

$$f_{avg}(\boldsymbol{a}) = \frac{1}{N}\sum_{i=1}^{N} a_i$$

Operation to aggregate or “summarize” sub-region of input.

Shift input by 1 position:

5/5 inputs change but only 1/3 pooled outputs

© 2017 NAVER LABS. All rights reserved.

Pooling

$$p = f(\boldsymbol{a})$$

$$f_{max}(\boldsymbol{a}) = \max_i(a_i)$$

$$f_{avg}(\boldsymbol{a}) = \frac{1}{N}\sum_{i=1}^{N} a_i$$

Operation to aggregate or “summarize” sub-region of input.

Shift input by 1 position:

5/5 inputs change but only 1/3 pooled outputs

Why is pooling useful?

- Adds **translation invariance** (useful when we care about “what” more than “where”)
- Can be used to **compress signal** (useful to improve computational efficiency)

`Stride = 1`

© 2017 NAVER LABS. All rights reserved.

Pooling

$$p = f(\boldsymbol{a})$$

$$f_{max}(\boldsymbol{a}) = \max_i(a_i)$$

$$f_{avg}(\boldsymbol{a}) = \frac{1}{N}\sum_{i=1}^{N} a_i$$

Operation to aggregate or “summarize” sub-region of input.

Shift input by 1 position:

5/5 inputs change but only 1/3 pooled outputs

Why is pooling useful?

- Adds **translation invariance** (useful when we care about “what” more than “where”)
- Can be used to **compress signal** (useful to improve computational efficiency)

`Stride = 2`

© 2017 NAVER LABS. All rights reserved.

Receptive fields

Borrowed terminology from neuroscience:

⇒ stimulus region that impacts a neuron's firing

For neural networks:

⇒ Region of input signal that impacts node output

© 2017 NAVER LABS. All rights reserved.

Receptive fields

Borrowed terminology from neuroscience:

⇒ stimulus region that impacts neuronal firing

For neural networks:

⇒ Region of input signal that impacts node output

© 2017 NAVER LABS. All rights reserved.

Receptive fields

Borrowed terminology from neuroscience:

⇒ stimulus region that impacts neuronal firing

For neural networks:

⇒ Region of input signal that impacts node output

© 2017 NAVER LABS. All rights reserved.

Receptive fields

Borrowed terminology from neuroscience:

⇒ stimulus region that impacts neuronal firing

For neural networks:

⇒ Region of input signal that impacts node output

Effective field size:

⇒ Region of **network** input signal that impacts node output

© 2017 NAVER LABS. All rights reserved.

Convolutional Neural Networks (CNNs)

CNNs for Classification

1. **Convolution**: Apply filters with learned weights to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

Train model with image data.
Learn weights of filters in convolutional layers.

Convolutional Layers: Local Connectivity

For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

Convolutional Layers: Local Connectivity

For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

4x4 filter: matrix of weights w_{ij}

$$\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij}\, x_{i+p,j+q} + b$$

for neuron (p,q) in hidden layer

1) applying a window of weights
2) computing linear combinations
3) activating with non-linear function

CNNs: Spatial Arrangement of Output Volume

Layer Dimensions:

h x w x d

where h and w are spatial dimensions
d (depth) = number of filters

Stride:

Filter step size

Give input shape [m, n, k], filter size [x, y], stride s, without padding
- What is the output shape of convolution?
- How many parameters in the filter?

Details of CNN Filters

- Input: 7x7x3 (i.e., 3 channels, with padding)
- The filter size is 3x3 (i.e. Conv2D) with stride = 2
- Note that this filter means:
  - A 3x3 matrix is applied to each channel
  - Each channel has its specific filter parameters
  - Its actual parameters are 3x3x3

Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**

Rectified Linear Unit (ReLU)

$$g(z) = \max(0, z)$$

Pooling

max pool with 2x2 filters
and stride 2

1) Reduced dimensionality
2) Spatial invariance

Representation Learning in Deep CNNs

Low level features

Edges, dark spots

Conv Layer 1

Mid level features

Eyes, ears, nose

Conv Layer 2

High level features

Facial structure

Conv Layer 3

CNNs for Classification: Feature Learning

1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNNs for Classification: Class Probabilities

- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

CNNs: Training with Backpropagation

Learn weights for convolutional filters and fully connected layers

Backpropagation: cross-entropy loss

$$J(\boldsymbol{\theta}) = \sum_i y^{(i)} \log\left(\hat{\boldsymbol{y}}^{(i)}\right)$$

Visualizing convolution activations

Live demo: https://www.cs.ryerson.ca/~aharley/vis/conv/

© 2017 NAVER LABS. All rights reserved.

Image credit: http://cs231n.github.io/convolutional-networks/

Visualizing convolution activations

© 2017 NAVER LABS. All rights reserved.

Image credit: http://cs231n.github.io/convolutional-networks/

Visualizing learned convolutions

Learned convolutions from first layer of AlexNet:

© 2017 NAVER LABS. All rights reserved.

Krizhevsky et al. Imagenet classification with deep convolutional neural networks. NIPS 2012.

Visualizing learned convolutions

Different spatial frequencies

Learned convolutions from first layer of AlexNet:

© 2017 NAVER LABS. All rights reserved.

Krizhevsky et al. Imagenet classification with deep convolutional neural networks. NIPS 2012.

Visualizing learned convolutions

Learned convolutions from first layer of AlexNet:

© 2017 NAVER LABS. All rights reserved.

Krizhevsky et al. Imagenet classification with deep convolutional neural networks. NIPS 2012.

An Architecture for Many Applications

ImageNet Dataset

Dataset of over 14 million images across 21,841 categories

"Elongated crescent-shaped yellow fruit with soft sweet flesh"

1409 pictures of bananas.

ImageNet Challenge

Classification task: produce a list of object categories present in image. 1000 categories.
"Top 5 error": rate at which the model does not output correct label in top 5 predictions

Other tasks include:
single-object localization, object detection from video/image, scene classification, scene parsing

ImageNet Challenge: Classification Task

2012: AlexNet. First CNN to win.
- 8 layers, 61 million parameters

2013: ZFNet
- 8 layers, more filters

2014: VGG
- 19 layers

2014: GoogLeNet
- "Inception" modules
- 22 layers, 5million parameters

2015: ResNet
- 152 layers

ImageNet Challenge: Classification Task

An Architecture for Many Applications

Object detection with R-CNNs
Segmentation with fully convolutional networks
Image captioning with RNNs

Beyond Classification

Semantic Segmentation

CAT

Object Detection

CAT, DOG, DUCK

Image Captioning

The cat is in the grass.

Image Captioning using RNNs

Image Captioning using RNNs

Impact: Self-Driving Cars

Impact: Healthcare

Identifying facial phenotypes of genetic disorders using deep learning

Gurovich et al., *Nature Med.* 2019

Interesting Applications of CNN

CNN for text classification

Embedding

Map each word to a vector (word vector). The vector can be pretrained or trained as a part of the CNN

Convolution (1D)

Filters are applied to one or more rows (i.e. a word vector)

Pooling

Dense

Illustrative CNN Architecture for Sentiment Classification

- Three filter region sizes: 2, 3 and 4, each of which has 2 filters.
- Every filter performs convolution on the sentence matrix and generates (variable-length) feature maps.
- Then 1-max pooling is performed over each map, i.e., the largest number from each feature map is recorded.
- Thus a univariate feature vector is generated from all six maps, and these 6 features are concatenated to form a feature vector for the penultimate layer.
- The final softmax layer then receives this feature vector as input and uses it to classify the sentence

Reference: Zhang Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.

TCN: Temporary Convolution Network (Causal Convolution)

- Causal: for a sequence $\{0, 1, 2, \ldots, n-1\}$, the i'th element of the output sequence may only depend on the elements of the input sequence with indices $\{0,1,2,\ldots, i\}$.
- In other words, an element in the output sequence can only depend on elements that come before it in the input sequence.
- TCN can be used to model time series data

Use left padding to ensure an output tensor has the same length as the input tensor

Use dilation to enlarge receptive field

Multiple dilation layers to ensure full coverage

Reference: https://unit8.co/resources/temporal-convolutional-networks-and-forecasting/

Few-Shot Learning

Figure 1: Matching Networks architecture

(a) Few-shot (b) Zero-shot

Figure 1: Prototypical networks in the few-shot and zero-shot scenarios. **Left**: Few-shot prototypes $\mathbf{c}_k$ are computed as the mean of embedded support examples for each class. **Right**: Zero-shot prototypes $\mathbf{c}_k$ are produced by embedding class meta-data $\mathbf{v}_k$. In either case, embedded query points are classified via a softmax over distances to class prototypes: $p_\phi(y = k|\mathbf{x}) \propto \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k))$.

References:

- Matching Networks for One Shot Learning , https://arxiv.org/pdf/1606.04080v2.pdf
- Prototypical Networks for Few-shot Learning, https://arxiv.org/pdf/1703.05175v2.pdf

Applications of CNN: Siamese CNN

Object Match

Complete the sentence:

Although the state has only four votes in the Electoral College,

- ❑ *its loss would be a symbolic blow to republican presidential candidate Bob Dole.*
- ❑ *but it failed to garner enough votes to override an expected veto by president Clinton.*

Match a response to a tweet:

X : *Damn, I have to work overtime this weekend!*
Y :

- ❑ *Try to have some rest buddy.*
- ❑ *It is hard to find a job, better start polishing your resume.*

Matching Models

References:

- Siamese Neural Networks for One-shot Image Recognition, https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf
- Convolutional Neural Network Architectures for Matching Natural Language Sentences , https://arxiv.org/pdf/1503.03244.pdf

Few Shot Learning Model

- Few-Shot Learning: classify objects with a handful of labeled samples (a.k.a. few-shot)
  - Each class has a prototype representation, learned from labeled samples
  - Train a neural network to **differentiate** the prototype representations of classes
- Architecture: given a test sample and support sets (i.e. labeled samples)
  - Use a convolutional neural network (CNN) to extract features (i.e. shapes, words or phrases)
  - Calculate the distance between the test sample and each support set
  - Compute the weight of each sample in support sets based on the distance
  - Produce the prediction as the sum of weights in each class
- Few-shot learning uses an ***innovative way to construct training sets*** from a few labeled samples such that it can exploit ***differences*** between object classes

N-way, *K*-shot Learning Model

- n-way: # of classes
- k-shot: # of samples in each support set, k usually is small (e.g. < 20)

CNN Summary

Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction

CNNs

- CNN architecture
- Application to classification
- ImageNet

Applications

- Segmentation, object detection, image captioning
- Visualization