Deep Learning For Computer Vision
PytzMLS2018: CIVE UDOM

Anthony Faustine

PhD Fellow
(IDLab research group-Ghent University)

5 April 2018

Learning goal

- Understand how to build and train Convolution Neural Networks (CNN).
- Learn how to apply CNN to to visual detection and recognition tasks.
- Learn how to apply Transfer learning with image and language data.
- Understand how to implement Convolution Neural Network using Pytorch framework.

Outline

Introduction: MLP Limitations

So far we have learned MLP as a universal function approximator which can be used for classification or regression problem.

- They build up complex pattern from simple pattern hierachically.
- Each layer learn to detect simple combination of pattern detected by previous layer.
- The lowest layers of the model capture simple patterns where the next layers capture more complex pattern.

Introduction: MLP Limitations

Consider the following three problems.

Problem 1: Given speach signal below

Task: Detect if the signal contain the word **HAPA KAZI TU**

Introduction: MLP Limitations

Consider the following three problems.

Problem 2: Given following image

Task: Idenify zebra in the image

Introduction: MLP Limitations

Consider the following three problems.

Problem 2: Given following two images.

(a) Image 1

(b) Image 2

Figure 1: Zebra

Task: Classify the image as zebra regardless of the orientation of zebra in the image.

Introduction: MLP Limitations

Composing MLP for these kind of problems is very challenging.

1. Require a very large network
2. MLPs are sensitive to the location of the pattern
    - Moving it by one component results in an entirely different input that the MLP wont recognize.

In many problems the location of a pattern is not important

- Only the presence of the pattern.
- Requirement: Network must be shift invariant.

More details

Outline

Convolutioanl Neural Network (CNN)

Neural networks for visual data are designed specifically for such problems:

- Handle very high input dimension
- Exploit the 2D topology of image or 3D topology for video data.
- Build in invariance to certain variations we expect (translations, illumination etc)

Convolutional Neural Networks (CNN)

CNN are specialized kind of neural networks for processing visual data.

- They employs a mathematical operation called convolution in place of general matrix multiplication in at least one of their layers.
- CNNs are often used for 2D or 3D data (such as grayscale or RGB images), but can also be applied to several other types of input, such as:
    1. 1D data: time-series, raw waveforms
    2. 2D data: grayscale images, spectrograms
    3. 3D data: RGB images, multichannel spectrograms

Convolutional Neural Networks (CNN)

Convolution leverages three important ideas that help improve a machine learning system.

1. Sparse interactions (local connectivity),
2. Parameter sharing,
3. Equivariant representations

CNN: Local connectivity

Unlike MLP, a feature at any given CNN layer only depends on a subset of the input of that layer.

- Each hidden unit is connected only to the subregion of the input image.
- This reduce the number of parameter.
- Reduce the cost of computing linear activations of the hidden units.

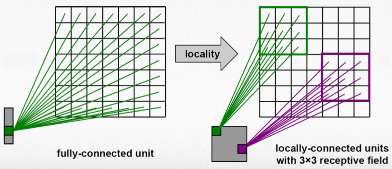


Figure 2: Local connectivity: `credit: Prof. Seungchul Lee`

CNN: Parameter Sharing

At each CNN layer, we learn several small filters (feature maps) and apply them to the entire layer input.

- Units organized into the same feature map share parameters.
- Hidden units within a feature map cover different positions in the image.
- Allow feature to be detected regardless of their position.

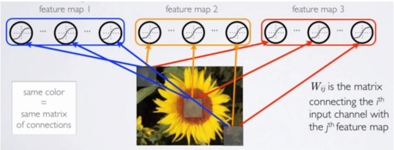


Figure 3: Parameter sharing: `credit: Hugo Larochelle`

CNN: Equivariant representations

A feature map (filter) that detects e.g. an eye can detect an eye everywhere on an image (translation invariance)

- Units organized into the same feature map share parameters.
- Hidden units within a feature map cover different positions in the image.
- Allow feature to be detected regardless of their position.

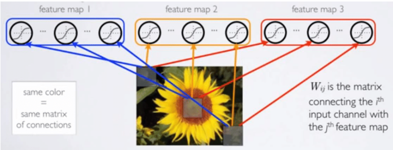
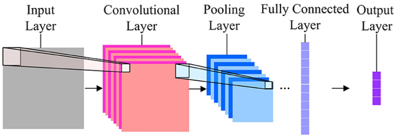


Figure 4: `credit: Hugo Larochelle`

CNN Architecture

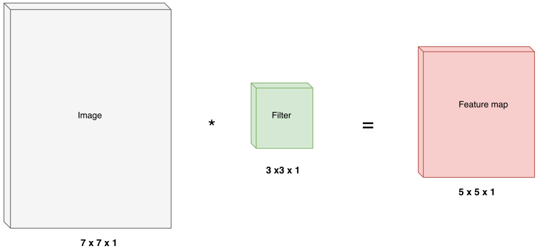
A typical layer of a convolutional network consists of three layers:

- Convolutional layer
- Detector stage
- Pooling layer and
- Fully connected layer



CNN Architecture: Convolutional layer

This is the first layer in CNN and consist of set of independent filters that can be sought as feature extractor.



- The result is obtained by taking the dot product between the filter $\mathbf{w}$ and the small $3 \times 3 \times 1$ chunck of the image $\mathbf{x}$ plus bias term $\mathbf{b}$ as the filter slides along the image.

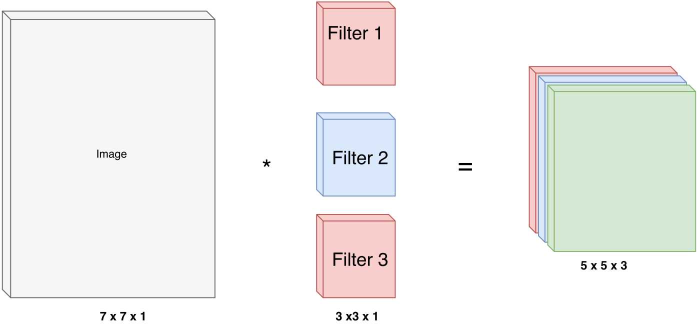
$$\mathbf{w}^{\mathbf{T}}\mathbf{x} + \mathbf{b}$$

- The step size of slide is called stride $\Rightarrow$ controls how the filter convolves around the input volume.

Demo

CNN Architecture: Convolutional layer

Consider more two filters



- If we have three filters of size $3 \times 3 \times 1$ we get 3 separate activation maps stacked up to get a new volume of size $5 \times 5 \times 3$

CNN Architecture: Convolutional operations

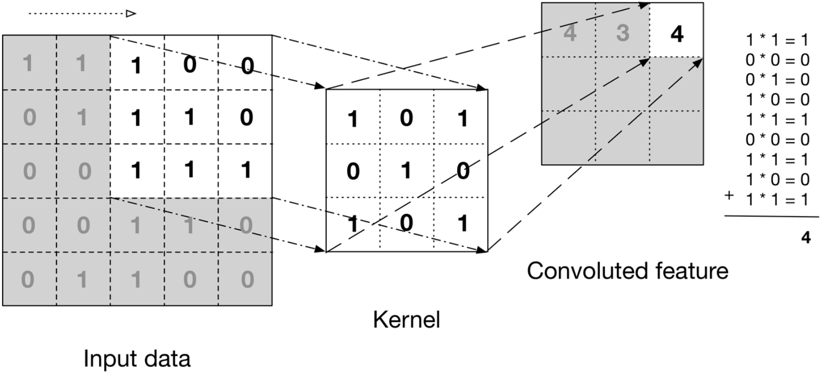
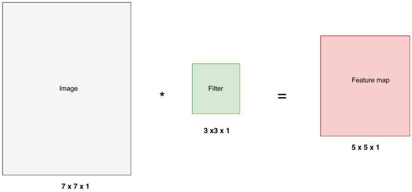


Figure 5: Conv operation

credit: Adam Gibson and Josh Patterson

CNN Architecture: Padding

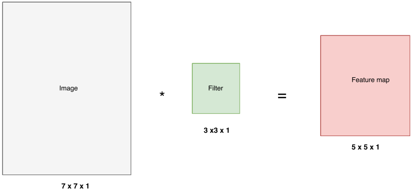
Consider the following $7 \times 7 \times 1$ images convolved with $3 \times 3 \times 1$ filter and stride size of 1.



- If the size of image is $N \times N$, and that of filter is $F \times F$ and S is the stride size S.
- The size of the feature map (output size) is $\frac{N-F}{S} + 1$
- For above image: $N = 7, F = 3$

CNN Architecture: Padding

Consider the following $7 \times 7 \times 1$ images convolved with $3 \times 3 \times 1$ filter and stride size of 1.



For above image: $N = 7, F = 3$

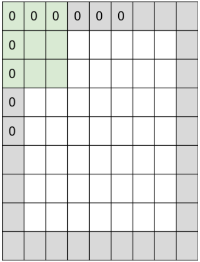
- Stride 1 $S = 1, \Rightarrow \frac{7-3}{1} + 1 = 5$
- Stride 2 $S = 2, \Rightarrow \frac{7-3}{2} + 1 = 3$
- Stride 3 $S = 3, \Rightarrow \frac{7-3}{3} + 1 = 2.33$ Does not fit

CNN layers: Padding

For above image: $N = 7, F = 3$

Stride 3 $S = 3, \Rightarrow \frac{7-3}{3} + 1 = 2.33$ Does not fit

- To address this we pad the input with suitable values (padding with zero is common)$\Rightarrow$ to preserve the spatial size.
- In general common to see convolutional layers with stride 1, filter $F \times F$ and zero padding with $P = \frac{F-1}{2}$



$$F = 3 \Rightarrow \text{ zero pad with } P = 1$$
$$F = 5 \Rightarrow \text{ zero pad with } P = 2$$
$$F = 7 \Rightarrow \text{ zero pad with } P = 3$$

CNN layers: Hyper-parameters

To summarize the conv layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hype-parameters:
  1. Number of filters K.
  2. Spatial extent of filter F.
  3. Amount zero padding P.

Common settings:

- $K =$ (power of 2 e.g) $4, 8, 16, 32, 64, 128$
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$ whatever fits.

- Produce a volume of size $W_2 \times H_2 \times D_2$ where

$$W_2 = (W_1 - F + 2P)/S + 1$$
$$H_2 = (H_1 - F + 2P)/S + 1$$
$$D_2 = K$$

- The number of weights per filter is $F \cdot F \cdot D_1$ and the total number of parameters is $(F \cdot F \cdot D_1) \cdot K$ and K biases.

CNN layers: Hyper-parameters

To summarize the conv layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hype-parameters:
  1. Number of filters K.
  2. Spatial extent of filter F.
  3. Amount zero padding P.

Common settings:

- $K =$ (power of 2 e.g) $4, 8, 16, 32, 64, 128$
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P =?$ whatever fits.

- Produce a volume of size $W_2 \times H_2 \times D_2$ where

$$W_2 = (W_1 - F + 2P)/S + 1$$
$$H_2 = (H_1 - F + 2P)/S + 1$$
$$D_2 = K$$

- The number of weights per filter is $F \cdot F \cdot D_1$ and the total number of parameters is $(F \cdot F \cdot D_1) \cdot K$ and K biases.

python Tanzania

CNN layers: Pytorch Implementation

```
torch.nn.Conv2d(in_channels, out_channels,kernel_size, stride=1, padding=0)
```

- in_channels (int) – Number of channels in the input image
- out_channels (int) – Number of channels produced by the convolution
- kernel_size (int or tuple) – Size of the convolving kernel
- stride (int or tuple, optional) – Stride of the convolution. Default: 1
- padding (int or tuple, optional) – Zero-padding added to both sides of the input.

CNN Architecture: Detection layer

In this stage each feature map of a conv layer is run through a non-linear function.

- ReLU function is often used after every convolution operation.
- It replace all the negative pixel in the feature map by zero.

CNN Architecture: Pooling layer

A pooling layer act as down-sampling filter $\Rightarrow$ takes each feature map from a convolution layer produce a condensed feature map.

- Make representation smaller and more manageable.
- Operates over each activation map independently
  - Reduce computational cost and the amount of parameter.
  - Preserve spatial invariance.

CNN Architecture: Pooling layer

Max Pooling

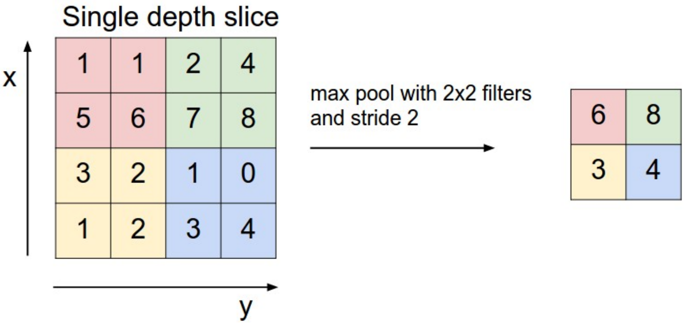


Figure 6: Max pooling (`credit: CS231n Stanford University`)

- Other pooling functions: **average pooling or L2-norm pooling**.

CNN Architecture: Pooling layer

To summarize the pooling layer.

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hype-parameters:
  1. Spatial extent of filter F.
  2. Stride S.

Common settings:

- $F = 2, S = 2$
- $F = 3, S = 2$

- Produce a volume of size $W_2 \times H_2 \times D_2$ where

$$
\begin{aligned}
W_2 &= (W_1 - F)/S + 1 \\
H_2 &= (H_1 - F)/S + 1 \\
D_2 &= D_1
\end{aligned}
$$

- Introduce zero parameters since it computes fixed function of input.
- Not common to use zero-padding for pooling layers.

CNN Architecture: Pooling layer

To summarize the pooling layer.

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hype-parameters:
  1. Spatial extent of filter F.
  2. Stride S.

Common settings:

- $F = 2, S = 2$
- $F = 3, S = 2$

- Produce a volume of size $W_2 \times H_2 \times D_2$ where

$$W_2 = (W_1 - F)/S + 1$$
$$H_2 = (H_1 - F)/S + 1$$
$$D_2 = D_1$$

- Introduce zero parameters since it computes fixed function of input.
- Not common to use zero-padding for pooling layers.

Pooling layer: Pytorch Implementation

```
torch.nn.MaxPool2d(kernel_size, stride)
```

- kernel_size (int or tuple) – Size of the convolving kernel
- stride (int or tuple, optional) – Stride of the convolution. Default: 1

Convolutional Architecture: Fully connected layer

In the end it is common to add one or more fully connected (FC) layer.

- Contains neuron that connect the entire input volume as in MLP.

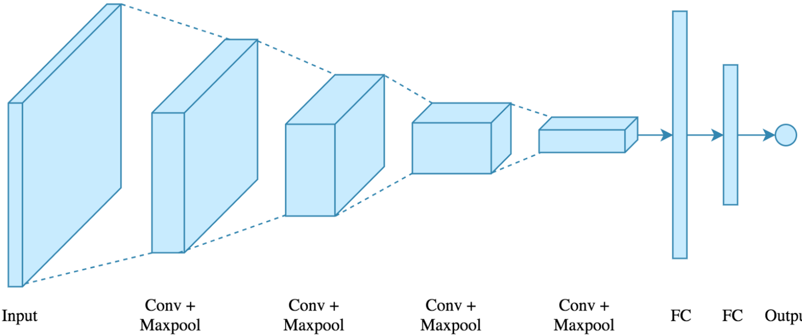
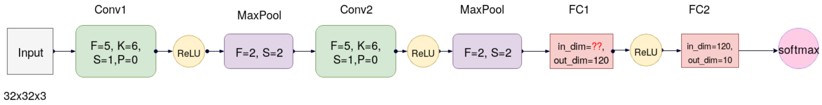


Figure 7: `credit: Arden Dertat`

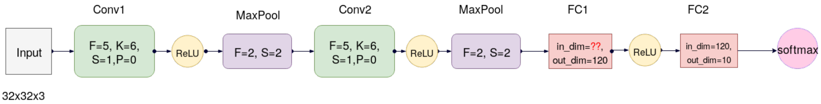
Convolutional Architecture




```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.mp = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16*53*53, 120)
        self.fc2 = nn.Linear(120, 10)

    def forward(self, x):
        in_size = x.shape[0]
        out  = F.relu(self.conv1(x))
        out  = self.mp(out)
        out  = F.relu(self.conv2(out))
        out = self.mp(out)
        out = out.view(in_size, -1)
        out = F.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```

Convolutional Architecture




```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.mp = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(16*53*53, 120)
        self.fc2 = nn.Linear(120, 10)

    def forward(self, x):
        in_size = x.shape[0]
        out  = F.relu(self.conv1(x))
        out  = self.mp(out)
        out  = F.relu(self.conv2(out))
        out = self.mp(out)
        out = out.view(in_size, -1)
        out = F.relu(self.fc1(out))
        out = self.fc2(out)
        return out
```

Outline

CNN applications: Image classification

Image Classification: Classify an image to a specific class.

- The whole image represents one class.
- We don't want to know exactly where are the object $\rightarrow$ only one object is presented.

The standard performance measures are:

- The error rate $P(f(\mathbf{x};\theta) \neq \mathbf{y})$ or accuracy $P(f(\mathbf{x};\theta) = \mathbf{y})$
- The balanced error rate (BER) $\frac{1}{K}\sum_{i=1}^{K} P(f(\mathbf{x};\theta) \neq y_i | \mathbf{y} = y_i)$

CNN applications: Image classification

In the two-class case we can use True Positive (TP) and False Postive (FP) rate as:

- $TP = P(f(\mathbf{x};\theta) = 1|\mathbf{y} = 1)$
  and
  $FP = P(f(\mathbf{x};\theta) = 1|\mathbf{y}) = 0$
- The ideal algorithm would have $TP \simeq 1$ and $FP \simeq 0$

Other standard performance representation:

- Receiver operating characteristic (ROC)
- Area under the curve AUC)

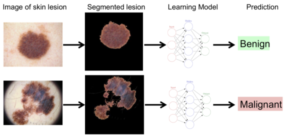


Figure 8: `credit:Stanford CS 229: Machine Learning`

python

CNN applications: Classification with localization

Image classification with localization: aims at predicting classes and locations of targets in an image.

- Learn to detect a class and a rectangle of where that object is.

A standard performance assessment considers

- a predicted bounding box $\hat{B}$ is correct if there is an annotated bounding box $\hat{B}$ for that class:

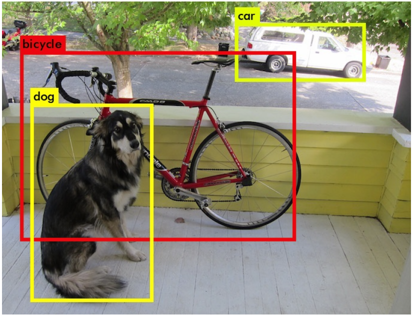
  such that the Intersection over Union (IoU) is large enough.

$$\frac{area(B \cap \hat{B})}{area(B \cup \hat{B})} \geq \frac{1}{2}$$

CNN applications: Object detection

Given an image we want to detect all the object in the image that belong to a specific classes and give their location.

- An image may can contain more than one object with different classes.



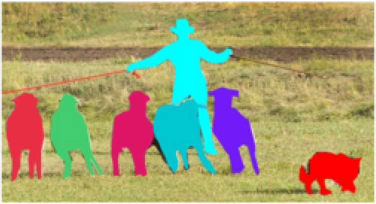
CNN applications: Image segmentation

Image segmentation: consists of labeling individual pixels with the class of the object it belongs to ⇒ It may also involve predicting the instance it belongs to.

Two types

1. Semantic Segmentation: Label each pixel in the image with a category label.
2. Instance Segmentation: Label each pixel in the image with a category label and distinguish them.

(c) Semantic segmentation



(d) Instance segmentation

Outline

Deep Convolutional Architecture

Several deep CNN architecture that works well in several tasks have been proposed.

- LeNet-5
- AlexNet
- VGG
- ResNet
- Inception

Outline

Transfer learning

Transfer learning: The ability to apply knowledge learned in previous tasks to novel tasks.

- Based on human learning. People can often transfer knowledge learnt previously to novel situations.

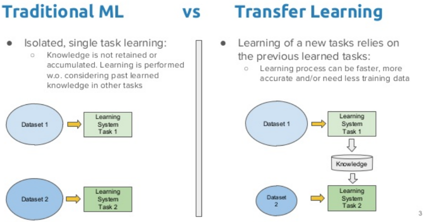


Figure 9: credit: Romon Morros

Transfer learning

Transfer learning Idea: Instead of training a deep network from scratch for your task:

- Take a network trained on a different domain for a different source task.
- Adapt it for your domain and your target task.
- A popular approach in computer vision and natural language processing task.

Why Transfer learning

- In practice, very few people train an entire CNN from scratch (with random initialization) $\Rightarrow$ (computation time and data availability)
- Very Deep Networks are expensive to train.For example, training ResNet18 for 30 epochs in 4 NVIDIA K80 GPU took us 3 days.
- Determining the topology/flavour/training method/hyper parameters for deep learning is a black art with not much theory to guide you.

References I

- Deep learning for Artificial Intelligence master course: TelecomBCN Bercelona(winter 2017)
- 6.S191 Introduction to Deep Learning: MIT 2018.
- Deep learning Specilization by Andrew Ng: Coursera
- Introductucion to Deep learning: CMU 2018
- Cs231n: Convolution Neural Network for Visual Recognition: Stanford 2018
- Deep learning in Pytorch, Francois Fleurent: EPFL 2018