Deep Learning

School of Data and Computer Science
Sun Yat-sen University

Overview

The CIFAR-10 dataset

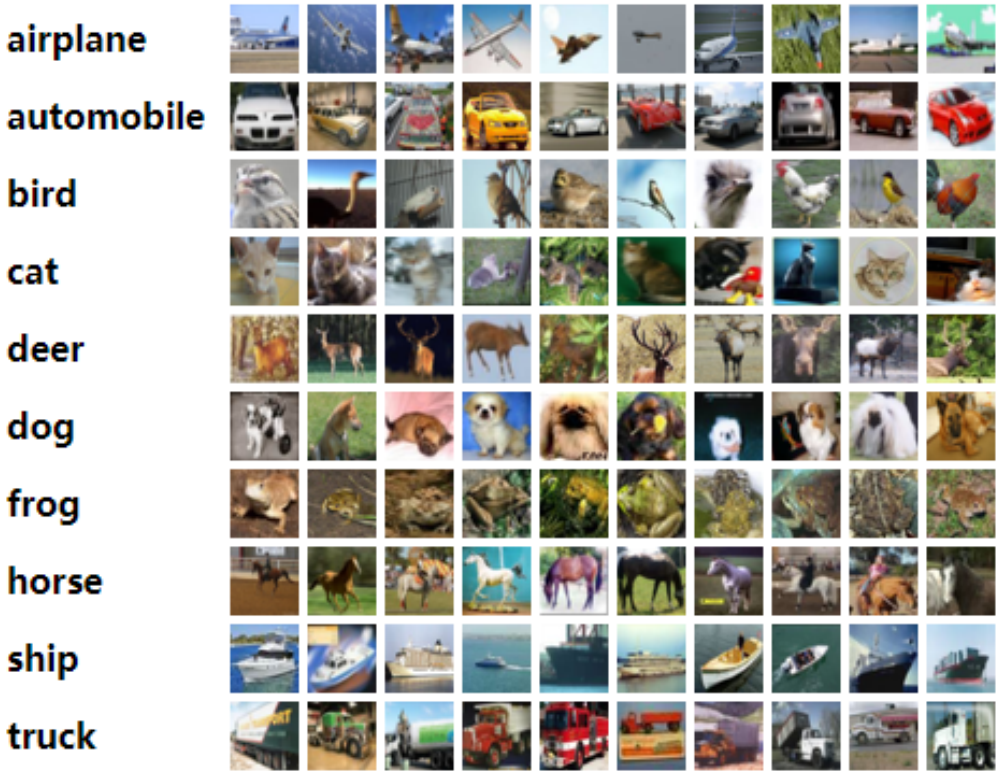
- The CIFAR-10 dataset (`http://www.cs.toronto.edu/~kriz/cifar.html`) consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- The dataset is divided into five training batches and one test batch, each with 10000 images.
- The test batch contains exactly 1000 randomly-selected images from each class.

The CIFAR-10 dataset (cont'd)

- The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another.
- Between them, the training batches contain exactly 5000 images from each class.
- The classes are completely mutually exclusive. There is no overlap between automobiles and trucks.
  - "Automobile" includes sedans, SUVs, things of that sort.
  - "Truck" includes only big trucks.
  - Neither includes pickup trucks.

The CIFAR-10 dataset (cont'd)

Here are the classes in the dataset, as well as 10 random images from each:



Architecture Overview

- Regular Neural Nets do not scale well to full images.
- Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way.
- The layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth.
- For example, the input images in CIFAR-10 are an input volume of activations, and the volume has dimensions $32 \times 32 \times 3$ (width, height, depth respectively).

Architecture Overview (cont'd)

The final output layer would for CIFAR-10 have dimensions $1 \times 1 \times 10$. Here is a visualization:

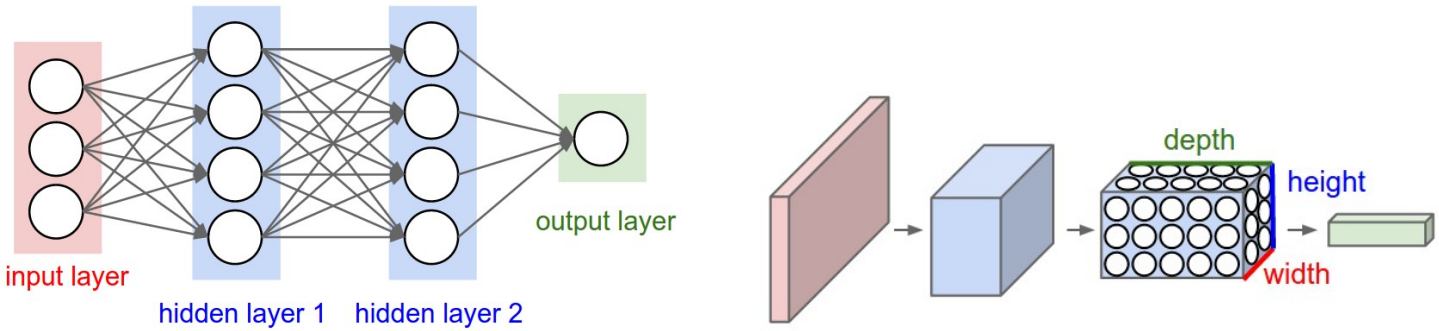


Figure 1: *

Left: A regular 3-layer Neural Network.
Right: A ConvNet arranges its neurons in three dimensions (width, height, depth)

Layers used to build ConvNets

- A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function.
- We use three main types of layers to build ConvNet architectures:
  - **Convolutional Layer**
  - **Pooling Layer**
  - **Fully-Connected Layer**
- We will stack these layers to form a full ConvNet architecture.

Layers used to build ConvNets (cont'd)

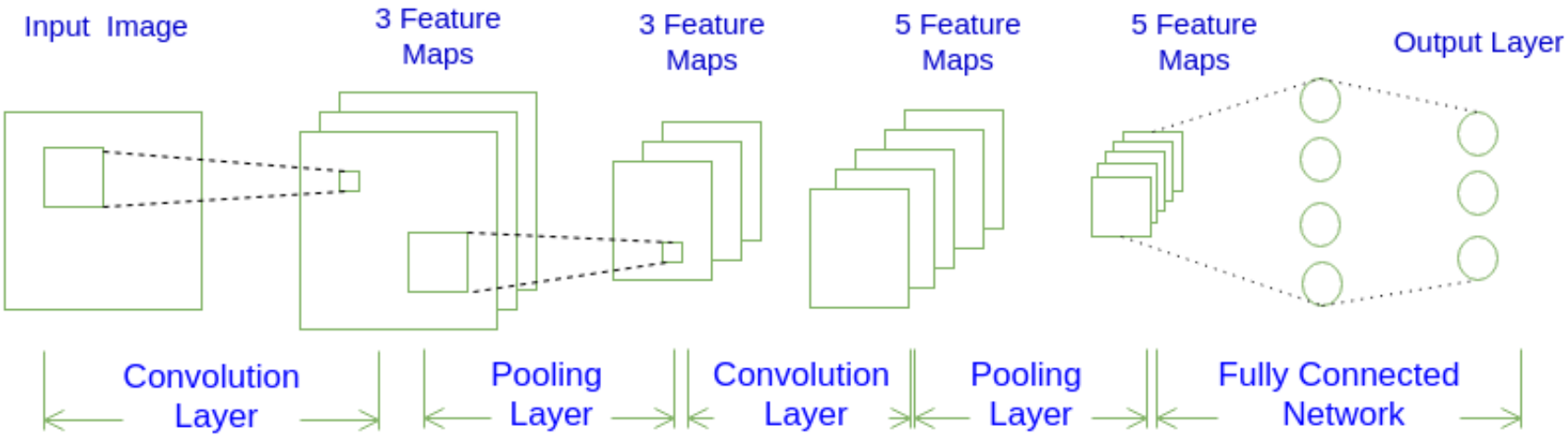


Figure 2: *Example Architecture*: Overview.

ConvNet for CIFAR-10 classification

A simple ConvNet for CIFAR-10 classification could have the architecture [**INPUT - CONV - RELU - POOL - FC**].

- **INPUT** $[32 \times 32 \times 3]$ will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- **CONV** layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as $[32 \times 32 \times 12]$ if we decided to use 12 filters.
- **RELU** layer will apply an elementwise activation function, such as the $max(0, x)$ thresholding at zero. This leaves the size of the volume unchanged ($[32 \times 32 \times 12]$).

ConvNet for CIFAR-10 classification (cont'd)

- **POOL** layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as $[16 \times 16 \times 12]$.
- **FC** (i.e. fully-connected) layer will compute the class scores, resulting in volume of size $[1 \times 1 \times 10]$, where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

To summarize, the Conv Layer:

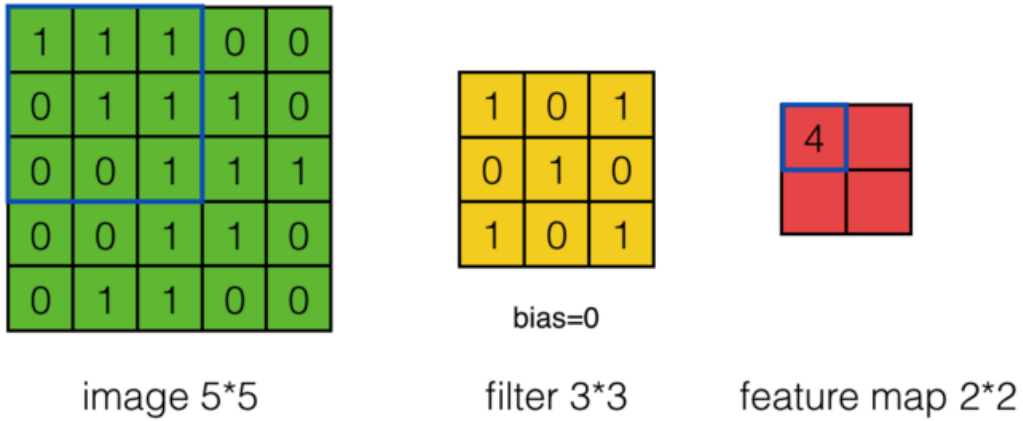
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$

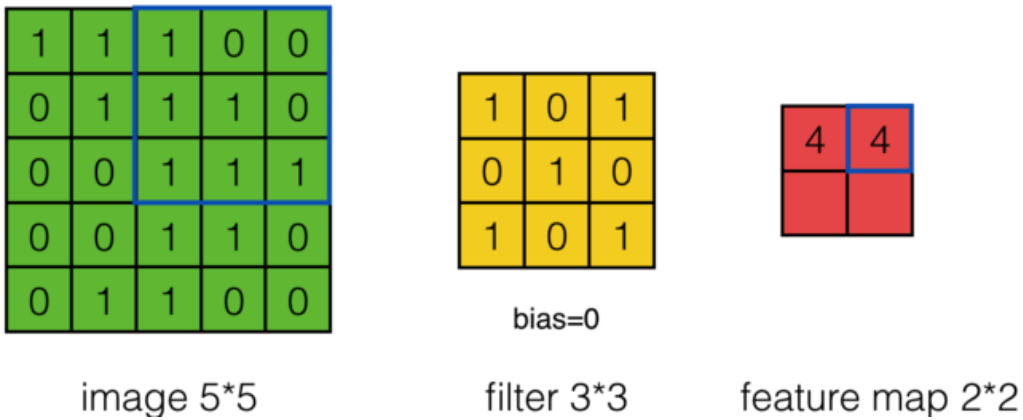
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Layers used to build ConvNets

Convolutional Layer (cont'd)

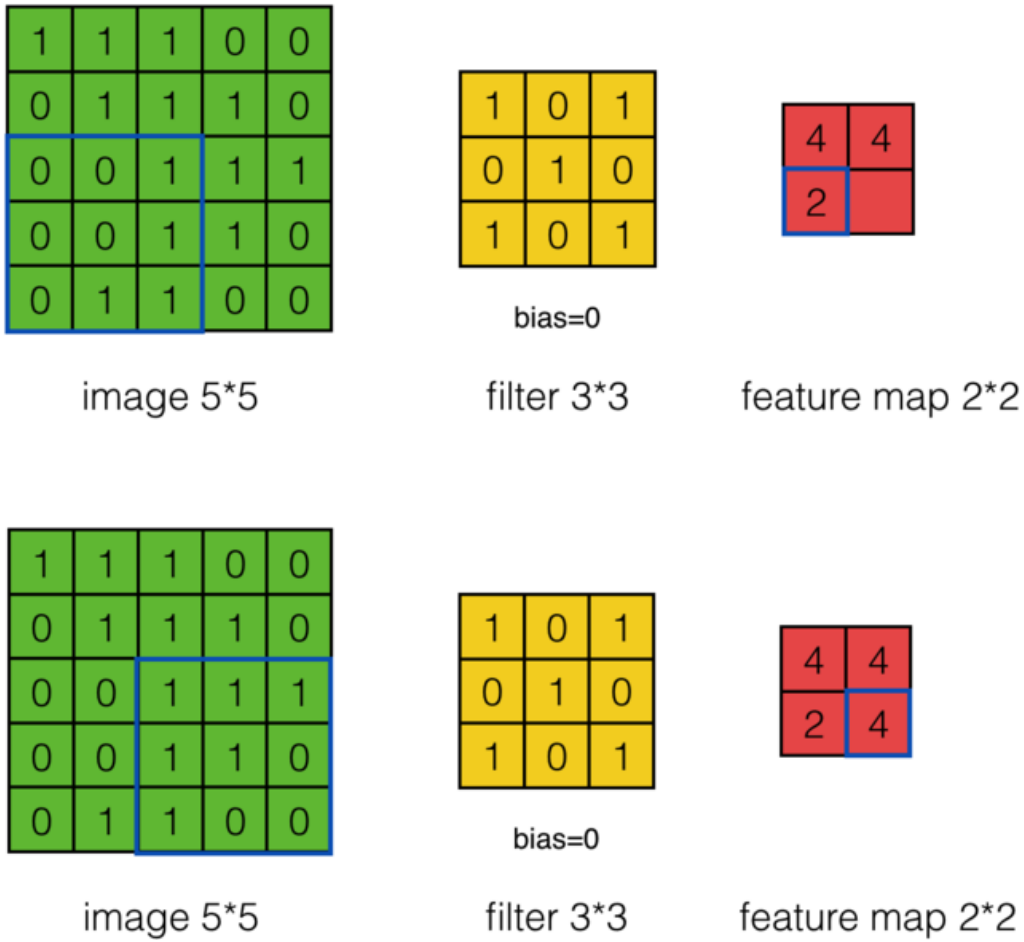
A common setting of the hyperparameters is $F = 3, S = 1, P = 1$.





Layers used to build ConvNets

Convolutional Layer (cont'd)



Layers used to build ConvNets

Pooling Layer

- It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture.
- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting.
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the **MAX** operation.

Layers used to build ConvNets

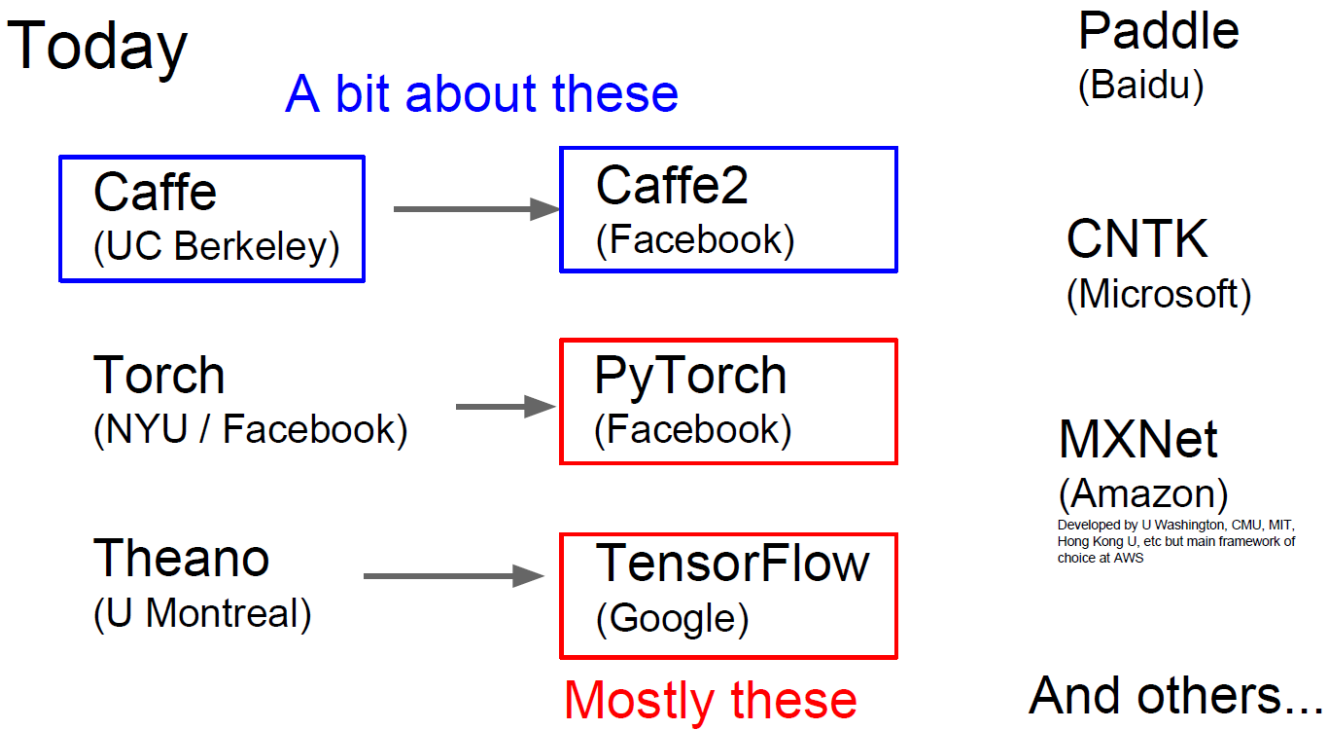
Pooling Layer (cont'd)

- The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations.
- Every MAX operation would in this case be taking a max over 4 numbers (little 2×2 region in some depth slice).

The depth dimension remains unchanged. More generally, the pooling layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D2 = D1$
- Introduces zero parameters since it computes a fixed function of the input
- For Pooling layers, it is not common to pad the input using zero-padding.

Deep Learning Softwares



MATLAB Deep Learning Toolbox

- Use **convolutional neural networks** (CNNs) and **long short-term memory** (LSTM) networks to perform classification and regression on image, time-series, and text data.
- Build advanced network architectures such as **generative adversarial networks** (GANs).
- Apps and plots can visualize **activations**, edit and analyze **network architectures**, and monitor **training progress**.
- Import models from **TensorFlow-Keras** and **Caffe**.
- Supports **transfer learning**.

MATLAB Deep Learning Toolbox

Create Simple Image Classification Network

1. Load Data

```
digitDatasetPath = fullfile(matlabroot,'toolbox','nnet','nndemos', ...
    'nndatasets','DigitDataset');

imds = imageDatastore(digitDatasetPath, ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
```

```
numTrainFiles = 750;
[imdsTrain,imdsValidation] = splitEachLabel(imds,numTrainFiles,'randomize');
```

750 images

training set and validation set

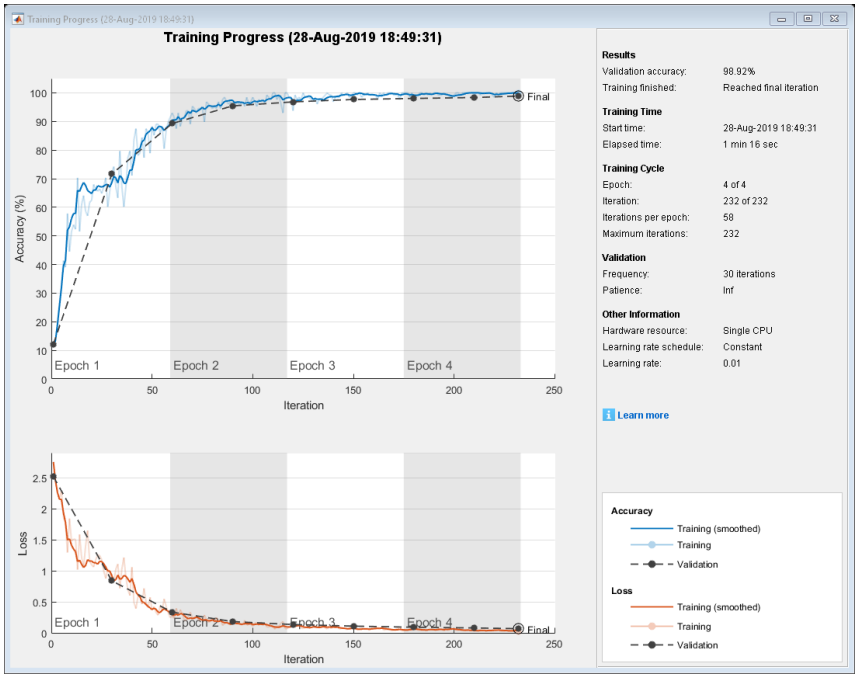
2. Define Network Architecture

```
inputSize = [28 28 1];
numClasses = 10;

layers = [
    imageInputLayer(inputSize)
    convolution2dLayer(5,20)
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(numClasses)
    softmaxLayer
    classificationLayer];
```

Each image is 28-by-28 pixels

3. Train Network

```
options = trainingOptions('sgdm', ...
    'MaxEpochs',4, ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(imdsTrain,layers,options);
```



4. Test Network

```
YPred = classify(net,imdsValidation);
YValidation = imdsValidation.Labels;
accuracy = mean(YPred == YValidation)
```

https://ww2.mathworks.cn/help/deeplearning/gs/create-simple-deep-learning-classification-network.html

Tasks

Given the dataset in the first section, please implement a convolutional neural network to calculate the accuracy rate with MATLAB. The major steps involved are as follows:

1. Reading the input image.
2. Preparing filters.
3. Conv layer: Convolving each filter with the input image.
4. ReLU layer: Applying ReLU activation function on the feature maps (output of conv layer).
5. Max Pooling layer: Applying the pooling operation on the output of ReLU layer.
6. Stacking conv, ReLU, and max pooling layers

Reference

- CIFAR-10 dataset, http://www.cs.toronto.edu/~kriz/cifar.html
- http://neuralnetworksanddeeplearning.com/chap6.html
- CS231n Convolutional Neural Networks for Visual Recognition, https://cs231n.github.io/convolutional-networks/
- MATLAB Deep Learning Toolbox, https://ww2.mathworks.cn/help/deeplearning/gs/create-simple-deep-learning-classification-network.html

The End