Introduction to Keras: Theory and Examples

IIT PATNA

OUTLINE

- Introduction to Google Colab
- Keras
  - Introduction
  - Fully Connected Neural Network
  - Convolution Neural Network
  - Working with own data

Confession

- Introductory (Hello World)
- Internet (sources at the end)

Part 0: Google Colab

Introduction to Google Colab

- Product by Google
- Google's free cloud service with GPU support for AI developers
  - CPU $\Leftrightarrow$ GPU $\Leftrightarrow$ TPU
  - Python programming language
  - Support to many neural network libraries such as Keras, PyTorch, OpenCV
- Files are stored on Drive

Introduction to Google Colab

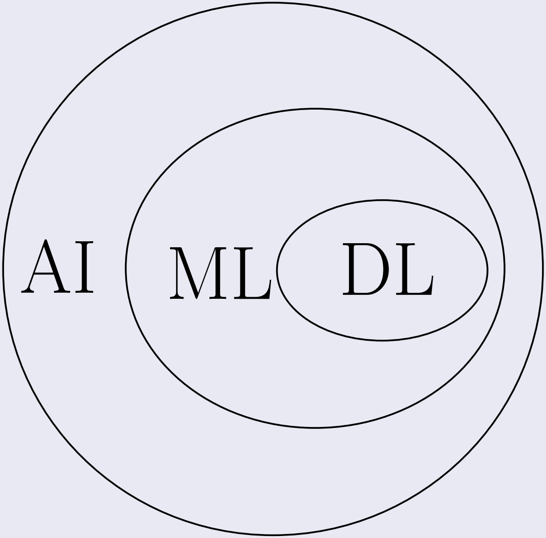
- `https://github.com/`
  - nrjcs/swym
- `https://colab.research.google.com/`

- Notebook: list of cells (code or text)
- can be shared
- collaborated
- GitHub
- Default folder is Colab Notebooks

- welcome example

Part I: Regular Neural Network

Introduction



Architecture of a Neural Network

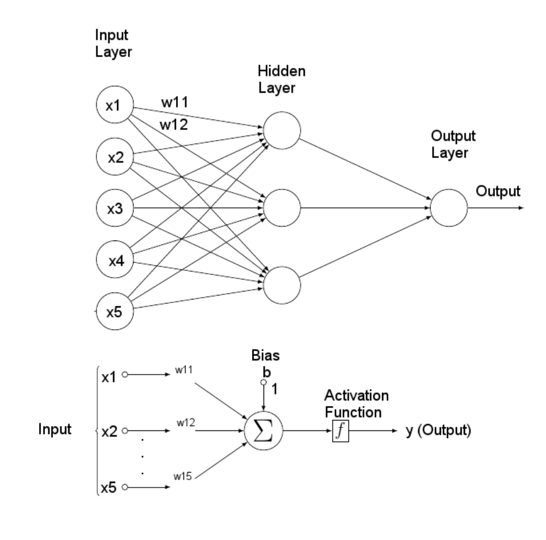


Figure: A Neural Network

Architecture of a Neural Network

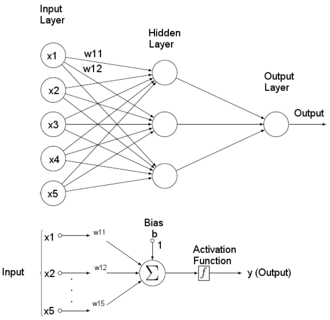


Figure: A Neural Network

Learning Steps (Decisions to be made):

1. Application (Problem)
2. Type of model
3. No. of layers
4. No. of nodes
5. Initialization of weights
6. Activation Function
7. Optimization Function
8. Evaluation Metrics
9. Dataset
10. Testing and Training Data
11. Batch size
12. Epoch

Keras

- NN: development (implementation and experimentation) is difficult.

Keras is

- high-level neural networks library
- written in Python
- capable of running on top of
  - TensorFlow (open source software library for numerical computation)
  - Theano (numerical computation library for Python)
  - CNTK (Microsoft Cognitive Toolkit): Deep learning framework
- developed with a focus on enabling fast experimentation (through user friendliness, modularity, and extensibility)
- and much more visit

Guiding principles

- Modularity
  - configurable modules
    - neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models
- Minimalism
  - Each module should be kept short and simple
- Easy extensibility
  - New modules are simple to add (as new classes and functions)
  - suitable for advanced research
- Work with Python
  - Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility
- User friendliness

Installation and Dependencies

- No worries
  - Google Colab
- You may visit Keras Installation Page @ keras.io

Keras Toolbox

What is in the toolbox ?

- Models
- Layers
- Preprocessing
- Metrics
- Optimizers
- Activations
- Datasets
- Constraints
- Initializers
- Loss (Objecitve) Function
- and many more...

Model

- Model
  - core data structure of Keras
  - a way to organize layers
- Two types:
  - Sequential
  - Model class API
- Sequential Model: a linear stack of layers
- functional API: for defining complex models, such as models with shared layers

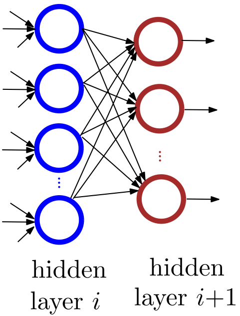
Layers

- Core Layers
  - Dense
  - Activation
  - Dropout
  - Flatten
  - many more ...
- Convolutional Layers
- Pooling Layers
- Recurrent Layers
- Your own Keras layers
- and many more ...

Core Layers

Dense

- fully connected NN layer: connection to all activations from previous layer



Core Layers

Activation

- Applies an activation function
    - detailed next

Dropout

- Applies Dropout to the input
- randomly setting a fraction p of input units to 0
- prevent overfitting

Flatten

- Flattens the input

- many more

Activation Function: Sigmoid

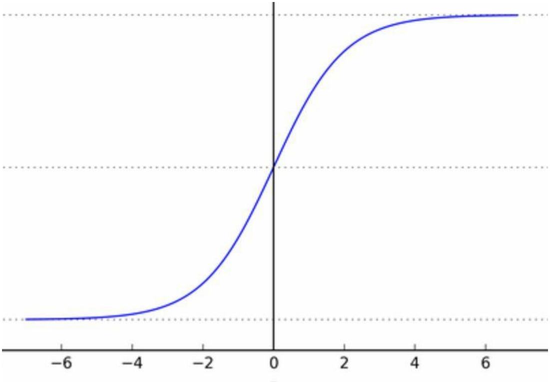


Figure: Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1}$$

Activation Function: ReLU (rectified linear unit)

Figure: ReLU

$$f(x) = max(0, x) \quad (2)$$

Activation Function: softmax

- usually used on the output layer to turn the outputs into probability-like values
- Sigmoid: two class
- softmax: multiclass

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3}$$

for i=1 to K and K is number of output units in output layer

Activation Function

linear

$$f(x) = x \tag{4}$$

- and many more...

Keras provides

Optimizer

- the specific algorithm used to update weights while we train our model
- such as *sgd* (Stochastic gradient descent optimizer)

Objective function or loss function

- used by the optimizer to navigate the space of weights
- such as *mse* (mean squared error)

Metrics

- used to judge the performance of your model
- such as *accuracy*

API

- Keras provides nice API
- documentation
    - A tour of `https://keras.io`

Building a Simple Deep Learning Network Using Keras

Steps

- Import libraries and modules
- Load image data
- Pre-process data
- Define model architecture
- Compile model
- Fit and evaluate Model
- Improvements

Keras examples

- Fully Connected Neural Network with MNIST dataset

Sample Output

temp : bash – Konsole

File Edit View Bookmarks Settings Help

```
niraj@niraj-Veriton-M200-Q87:~/temp$ python 1.py
Using Theano backend.
(60000, 28, 28)
(60000,)
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
dense_1 (Dense)                  (None, 784)           615440      dense_input_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 10)            7850        dense_1[0][0]
====================================================================================================
Total params: 623,290
Trainable params: 623,290
Non-trainable params: 0
____________________________________________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 4s - loss: 0.2744 - acc: 0.9221 - val_loss: 0.1356 - val_acc: 0.9601
Epoch 2/2
60000/60000 [==============================] - 4s - loss: 0.1078 - acc: 0.9688 - val_loss: 0.0957 - val_acc: 0.9707
 9600/10000 [===========================>..] - ETA: 0s Error: 2.93%
niraj@niraj-Veriton-M200-Q87:~/temp$
```

temp : bash

Improving Performance of Simple Network: additional hidden layers

```
temp : bash – Konsole
File  Edit  View  Bookmarks  Settings  Help
niraj@niraj-Veriton-M200-Q87:~/temp$ python 1.py
Using Theano backend.
(60000, 28, 28)
(60000,)
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
dense_1 (Dense)                  (None, 784)           615440      dense_input_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 784)           615440      dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 10)            7850        dense_2[0][0]
====================================================================================================
Total params: 1,238,730
Trainable params: 1,238,730
Non-trainable params: 0
____________________________________________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 8s - loss: 0.2184 - acc: 0.9354 - val_loss: 0.1094 - val_acc: 0.9639
Epoch 2/2
60000/60000 [==============================] - 8s - loss: 0.0755 - acc: 0.9767 - val_loss: 0.0852 - val_acc: 0.9720
 9824/10000 [============================>.] - ETA: 0s Error: 2.80%
niraj@niraj-Veriton-M200-Q87:~/temp$
temp : bash
```

Improving Performance of Simple Network: additional hidden layers

temp : bash – Konsole

File Edit View Bookmarks Settings Help

```
niraj@niraj-Veriton-M200-Q87:~/temp$ python 1.py
Using Theano backend.
(60000, 28, 28)
(60000,)
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
dense_1 (Dense)                  (None, 784)           615440      dense_input_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 784)           615440      dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 784)           615440      dense_2[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 10)            7850        dense_3[0][0]
====================================================================================================
Total params: 1,854,170
Trainable params: 1,854,170
Non-trainable params: 0
____________________________________________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 11s - loss: 0.1999 - acc: 0.9388 - val_loss: 0.0950 - val_acc: 0.9712
Epoch 2/2
60000/60000 [==============================] - 12s - loss: 0.0751 - acc: 0.9770 - val_loss: 0.0914 - val_acc: 0.9738
 9920/10000 [============================>.] - ETA: 0s Error: 2.62%
niraj@niraj-Veriton-M200-Q87:~/temp$
```

temp : bash

Improving Performance of Simple Network: introducing dropout layer

```
temp : bash – Konsole
File  Edit  View  Bookmarks  Settings  Help
niraj@niraj-Veriton-M200-Q87:~/temp$ python 1.py
Using Theano backend.
(60000, 28, 28)
(60000,)
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
dense_1 (Dense)                  (None, 784)           615440      dense_input_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 784)           615440      dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 784)           615440      dense_2[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 784)           0           dense_3[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 10)            7850        dropout_1[0][0]
====================================================================================================
Total params: 1,854,170
Trainable params: 1,854,170
Non-trainable params: 0
____________________________________________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 13s - loss: 0.2014 - acc: 0.9386 - val_loss: 0.1017 - val_acc: 0.9697
Epoch 2/2
60000/60000 [==============================] - 14s - loss: 0.0771 - acc: 0.9760 - val_loss: 0.0811 - val_acc: 0.9740
10000/10000 [==============================] - 1s
 Error: 2.60%
niraj@niraj-Veriton-M200-Q87:~/temp$
temp : bash
```

Improving Performance of Simple Network: using different optimizers

temp : bash – Konsole

File Edit View Bookmarks Settings Help

```
niraj@niraj-Veriton-M200-Q87:~/temp$ python 1.py
Using Theano backend.
(60000, 28, 28)
(60000,)
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
dense_1 (Dense)                  (None, 784)           615440      dense_input_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 784)           615440      dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 784)           615440      dense_2[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 784)           0           dense_3[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 10)            7850        dropout_1[0][0]
====================================================================================================
Total params: 1,854,170
Trainable params: 1,854,170
Non-trainable params: 0
____________________________________________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 9s - loss: 1.0352 - acc: 0.7382 - val_loss: 0.4882 - val_acc: 0.8791
Epoch 2/2
60000/60000 [==============================] - 10s - loss: 0.4422 - acc: 0.8784 - val_loss: 0.3497 - val_acc: 0.9051
 9984/10000 [============================>.] - ETA: 0s Error: 9.49%
niraj@niraj-Veriton-M200-Q87:~/temp$
```

temp : bash

Improving Performance of Simple Network: training for more number of epochs

temp : bash – Konsole

File Edit View Bookmarks Settings Help

```
niraj@niraj-Veriton-M200-Q87:~/temp$ python 1.py
Using Theano backend.
(60000, 28, 28)
(60000,)
____________________________________________________________________________________________________
Layer (type)                     Output Shape          Param #     Connected to
====================================================================================================
dense_1 (Dense)                  (None, 784)           615440      dense_input_1[0][0]
____________________________________________________________________________________________________
dense_2 (Dense)                  (None, 784)           615440      dense_1[0][0]
____________________________________________________________________________________________________
dense_3 (Dense)                  (None, 784)           615440      dense_2[0][0]
____________________________________________________________________________________________________
dropout_1 (Dropout)              (None, 784)           0           dense_3[0][0]
____________________________________________________________________________________________________
dense_4 (Dense)                  (None, 10)            7850        dropout_1[0][0]
====================================================================================================
Total params: 1,854,170
Trainable params: 1,854,170
Non-trainable params: 0
____________________________________________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s - loss: 1.0537 - acc: 0.7378 - val_loss: 0.4933 - val_acc: 0.8815
Epoch 2/20
60000/60000 [==============================] - 9s - loss: 0.4407 - acc: 0.8813 - val_loss: 0.3505 - val_acc: 0.9054
Epoch 3/20
60000/60000 [==============================] - 10s - loss: 0.3512 - acc: 0.9016 - val_loss: 0.3008 - val_acc: 0.9187
Epoch 4/20
```

temp : bash

Improving Performance of Simple Network: training for more number of epochs

temp : bash – Konsole

File Edit View Bookmarks Settings Help

```
60000/60000 [==============================] - 9s - loss: 0.2415 - acc: 0.9314 - val_loss: 0.2227 - val_acc: 0.9377
Epoch 8/20
60000/60000 [==============================] - 10s - loss: 0.2280 - acc: 0.9348 - val_loss: 0.2114 - val_acc: 0.9404
Epoch 9/20
60000/60000 [==============================] - 9s - loss: 0.2150 - acc: 0.9386 - val_loss: 0.2007 - val_acc: 0.9428
Epoch 10/20
60000/60000 [==============================] - 9s - loss: 0.2036 - acc: 0.9420 - val_loss: 0.1931 - val_acc: 0.9454
Epoch 11/20
60000/60000 [==============================] - 10s - loss: 0.1934 - acc: 0.9446 - val_loss: 0.1835 - val_acc: 0.9477
Epoch 12/20
60000/60000 [==============================] - 10s - loss: 0.1845 - acc: 0.9476 - val_loss: 0.1775 - val_acc: 0.9497
Epoch 13/20
60000/60000 [==============================] - 10s - loss: 0.1757 - acc: 0.9500 - val_loss: 0.1714 - val_acc: 0.9508
Epoch 14/20
60000/60000 [==============================] - 9s - loss: 0.1689 - acc: 0.9516 - val_loss: 0.1649 - val_acc: 0.9525
Epoch 15/20
60000/60000 [==============================] - 10s - loss: 0.1614 - acc: 0.9541 - val_loss: 0.1584 - val_acc: 0.9532
Epoch 16/20
60000/60000 [==============================] - 10s - loss: 0.1546 - acc: 0.9556 - val_loss: 0.1549 - val_acc: 0.9547
Epoch 17/20
60000/60000 [==============================] - 9s - loss: 0.1484 - acc: 0.9583 - val_loss: 0.1491 - val_acc: 0.9564
Epoch 18/20
60000/60000 [==============================] - 10s - loss: 0.1429 - acc: 0.9593 - val_loss: 0.1455 - val_acc: 0.9565
Epoch 19/20
60000/60000 [==============================] - 10s - loss: 0.1373 - acc: 0.9611 - val_loss: 0.1412 - val_acc: 0.9579
Epoch 20/20
60000/60000 [==============================] - 10s - loss: 0.1324 - acc: 0.9623 - val_loss: 0.1381 - val_acc: 0.9583
10000/10000 [==============================] - 1s
 Error: 4.17%
niraj@niraj-Veriton-M200-Q87:~/temp$
```

temp : bash

Improving Performance of Simple Network

other options to explore

- additional hidden layers
- dropout
- different optimizers
- more number of epochs
- optimizer learning rate
- number of internal hidden neurons
- batch size

Part II: Convolution Neural Network

Let's take a break from NN concepts

Convolution

- among the most important operations in signal and image processing
- it is the core concept behind the convolution neural network
- convolution operation: $(f * g) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$
- produces a third function which represents how functions are correlated

Convolution

- the two functions in context of images are:
  - input image
  - kernel (filter/feature detector)
- output is some feature
- important for images due to the property of being stationary $\Rightarrow$ same feature detector for whole image

Convolution Example

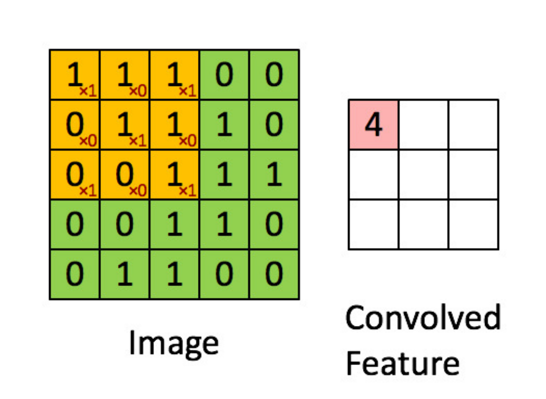


Image Source: Internet

Convolution Example

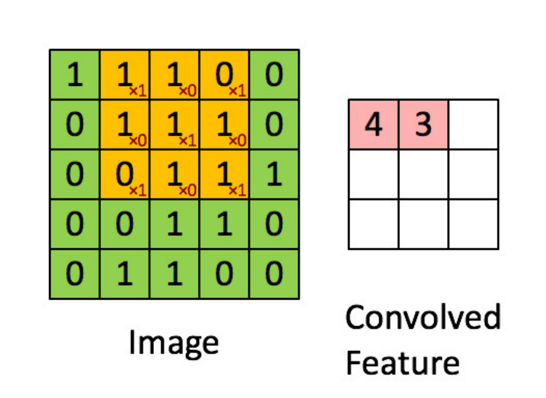


Image Source: Internet

Convolution Example

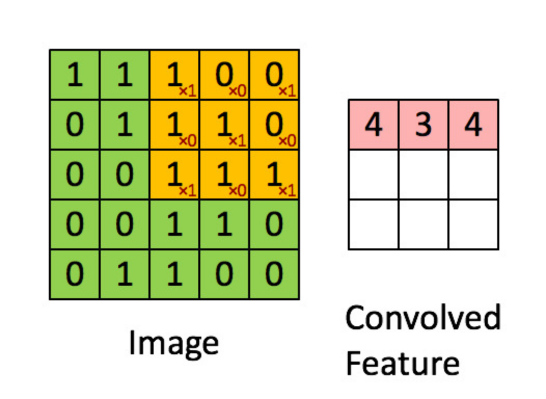


Image Source: Internet

Convolution Example

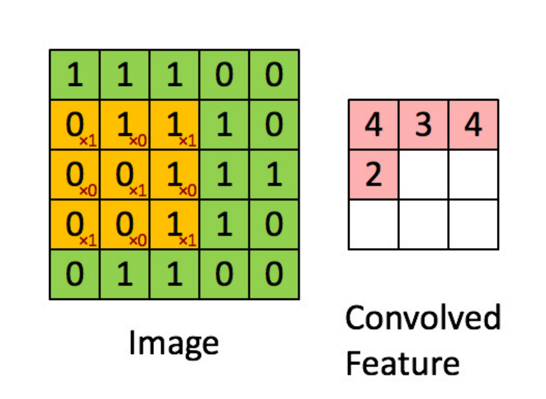


Image Source: Internet

Convolution Example

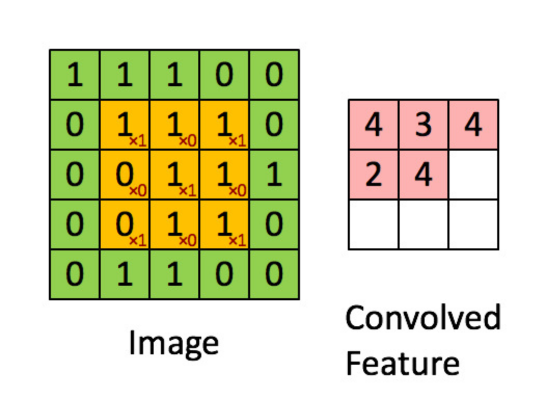


Image Source: Internet

Convolution Example

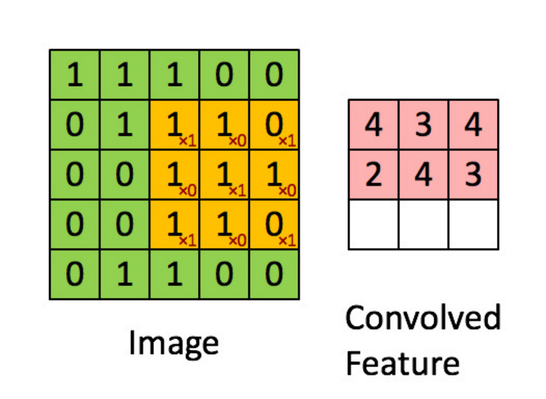


Image Source: Internet

Convolution Example

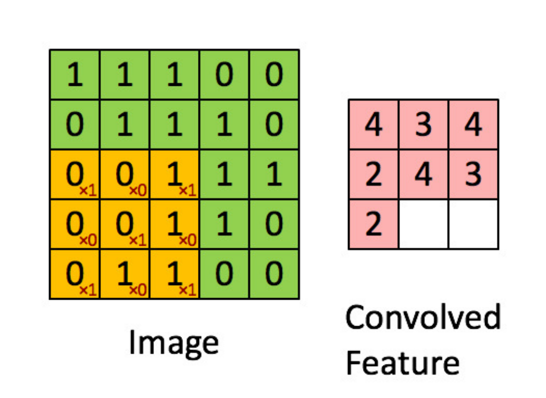


Image Source: Internet

Convolution Example

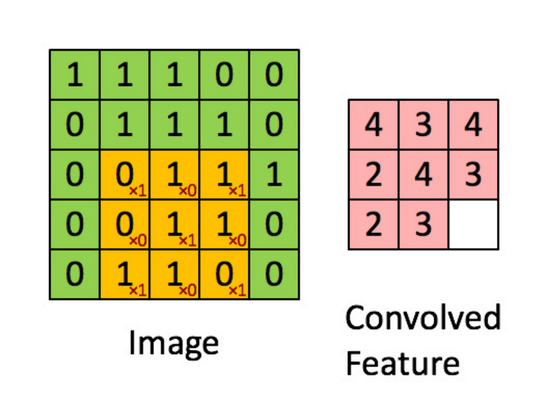


Image Source: Internet

Convolution Example

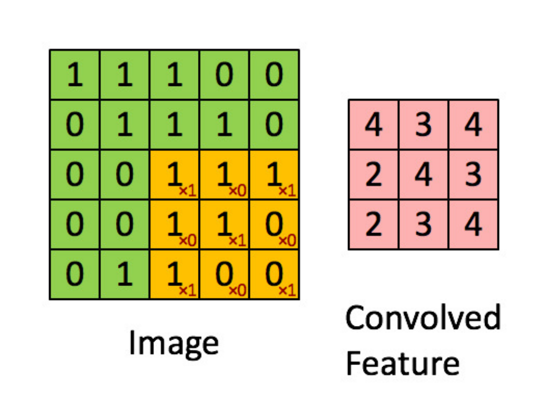


Image Source: Internet

Convolution Example

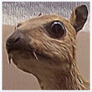
Operation	Kernel ω	Image result g(x,y)
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	

Image Source: Internet

Convolution Example

Box blur (normalized)	$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$	
Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$	
Gaussian blur 5 × 5 (approximation)	$\frac{1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\4 & 16 & 24 & 16 & 4\\6 & 24 & 36 & 24 & 6\\4 & 16 & 24 & 16 & 4\\1 & 4 & 6 & 4 & 1\end{bmatrix}$	
Unsharp masking 5 × 5 Based on Gaussian blur with amount as 1 and threshold as 0 (with no image mask)	$\frac{-1}{256}\begin{bmatrix}1 & 4 & 6 & 4 & 1\\4 & 16 & 24 & 16 & 4\\6 & 24 & -476 & 24 & 6\\4 & 16 & 24 & 16 & 4\\1 & 4 & 6 & 4 & 1\end{bmatrix}$	

Image Source: Internet

Switching back . . .

Issue with the Fully Connected Neural Network

- number of parameters
  - a 32X32X3 image $\Rightarrow$ 3072 (on input layer)
  - a 720X720X3 image $\Rightarrow$ 15,55,200 (on input layer)
  - for large images, depending on number of hidden layers and the neurons in each layer, for fully connected neural network, number of parameters may be in *millions*
  - resource requirement
  - overfitting

Convolution Neural Network (ConvNet)

in many way similar to regular Neural Networks

- *neurons* organized to form *layers*
- weights to be learnt
- biases
- neurons receive inputs, performs a dot product followed by some activation function
- have a loss function . . .

in addition

- assume that input are images $\Rightarrow$ thus, many things follows
- utilize spatial structure
  - regular network $\Rightarrow$ image processed as a flat vector
- number of parameters is input independent

ConvNet

- well suited for classifying images
- being applied to other problems as well such as text, speech, video ...

- network architecture more appropriate
- layers of a ConvNet have neurons arranged in 3 dimensions: width, height, depth
- each layer transforms an input 3D volume to an output 3D volume

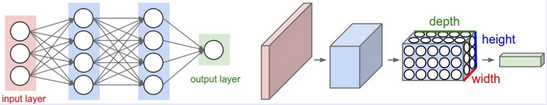


Image Source: `http://cs231n.github.io/convolutional-networks/`

Convolution Example

Recall convolution operation

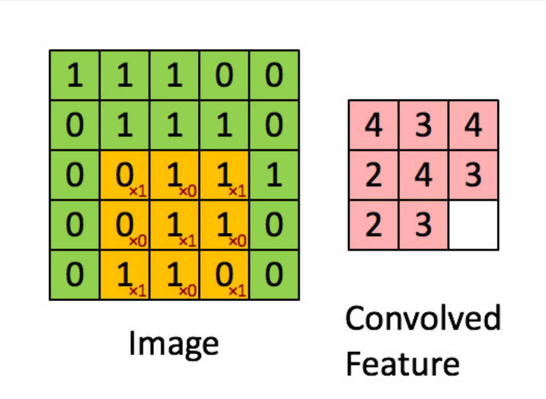


Image Source: Internet

Padding

Issue

- pixels on the side are ignored
- in addition, padding helps in controlling image size

0	0	0	0	0	0	0	0
0	18	54	51	239	244	188	0
0	55	121	75	78	95	88	0
0	35	24	204	113	109	221	0
0	3	154	104	235	25	130	0
0	15	253	225	159	78	233	0
0	68	85	180	214	245	0	0
0	0	0	0	0	0	0	0

Image Source: Internet

Padding

No padding

- Input: n X n
- Filter size: f X f
- Output: (n-f+1) X (n-f+1)

with padding

- Input: n X n
- Padding: p
- Filter size: f X f
- Output: (n+2p-f+1) X (n+2p-f+1)

Two common choices for padding

- valid: no padding
- same: output size is same as input
  - $n+2p-f+1 = n \Rightarrow p = (f-1)/2$

Stride

- number of steps during convolution
- Input: n X n
- Padding: p
- Stride: s
- Filter size: f X f
- Output: [(n+2p-f)/s+1] X [(n+2p-f)/s+1]
- reduces the size of the image

Filters and Depth

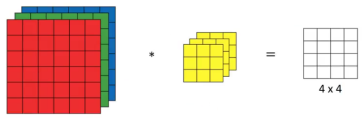


Image Source: Internet

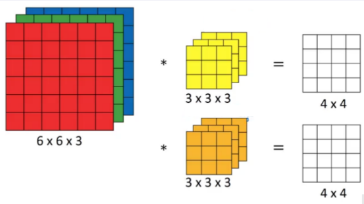


Image Source: Internet

ConvNet Architecture

- Stack of layers: each layer transform the image volume (w,h,d) to an output volume
- Commonly used layers: Convolutional Layer, Pooling Layer, Fully-Connected Layer, ReLU
- a layer may (such as convolution layer) or may not (such as ReLU) have parameters
- a layer (such as convolution layer) may or may not (such as ReLU) have additional hyper-parameters (number of filters, stride, zero padding)

Convolutional Layer

- core building block of a ConvNet
- perform convolution with the three hyper-parameters: depth, stride and padding
- incoming example

Pooling

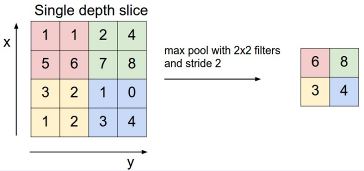


Image Source: Internet

Pooling Layer

- reduces size $\Rightarrow$ number of parameters and computation also decreases
- helps avoiding overfitting

Findings

- Smaller stride is better (1)
- padding improves performance
- average pooling

- MNIST Example

Additional

- Working with own data

Constructing the Right Network

steps to follow to make an efficient image classifier?

- lot of experimentation and testing to get the optimal structure and parameters

- A pre-trained model

Important Links

Links

1. Keras Official Documentation Page
2. Keras official github
3. Another GitHub Page
4. GitHub Page MNIST example
5. Keras Tutorial
6. An Example
7. Another Example
8. Deep Learning with Keras (Book)

The End