Handwritten Recognition of Chinese Characters

Analysis on CNN working principles and best practices along with a presentation of a case study

Francesco Cagnin, Alessandro Torcinovich

Università Ca' Foscari DAIS

Artificial Intelligence Course 2014/2015

- Classic NN use only **fully connected layers (FCL)**, whose neurons are connected to every neuron of their adjacent layers

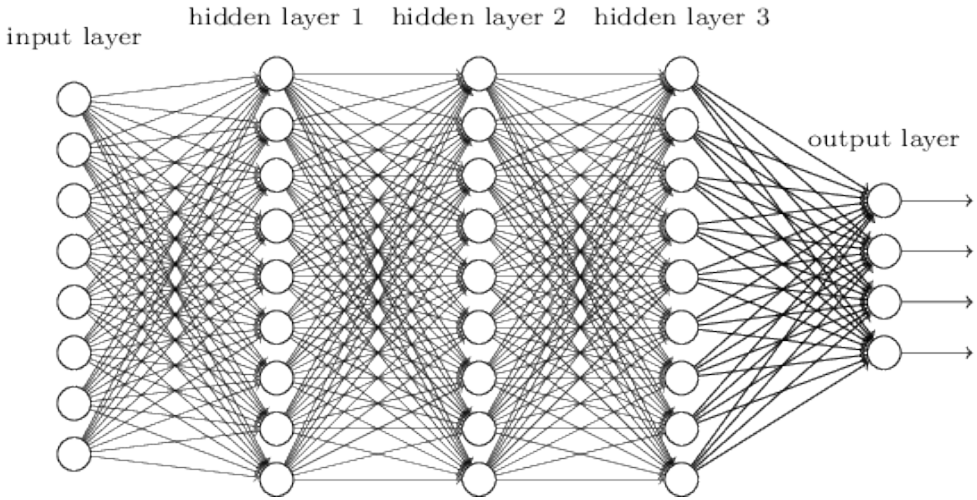
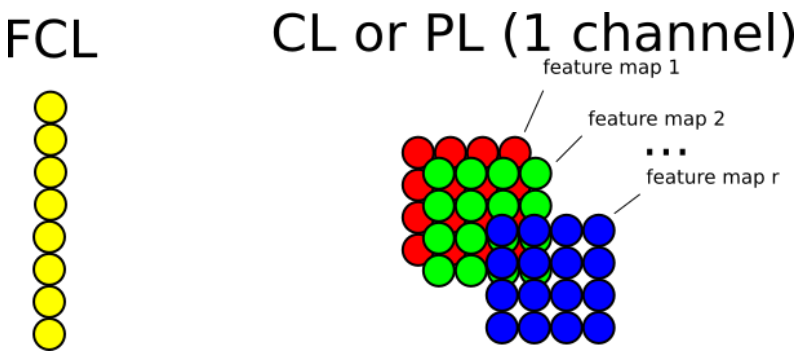


Figure: A classic NN [6]

- For complex classification tasks, this kind of network is no more efficient, and adding more FCL does not improve the classification for many reasons
- The **"unstable" gradient problem** [6]: if a FCL-only NN is deep, the gradient components of the weights related the first layers will be very small or very big w.r.t. the other weights and will not adjust properly

- A new kind of neural network was proposed: the **convolutional neural network** [6], which introduces two new layers: the **convolutional layer (CL)** and the **pooling layer (PL)**



- CL and PL are a **set of equal-sized squares of neurons**, called **feature maps**, suitable to be used with image inputs. With coloured images, each feature map is instead composed of a triple of squares of neurons, each one called **channel**, representing the RGB channels

- **Start**: input layer and some optional FCL
- **Middle**: pairs of CL-PL, rigorously placed next each other
- **End**: some other optional FCL and the output layer, i.e. an FCL with a number of neurons corresponding to the classes of our problem

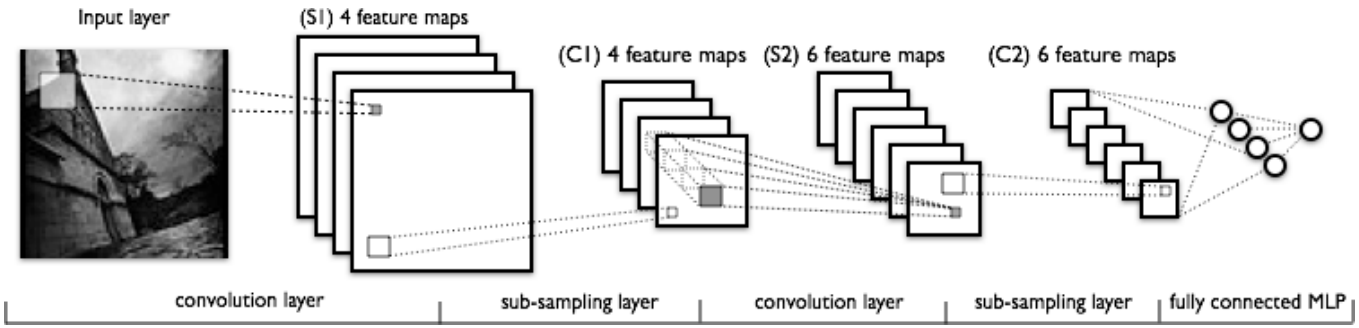


Figure: The general structure of a CNN [7]

Basic terminology I

Feedforward and backpropagation in CNN can be very complex, so we define some terminology used in subsequent explanations:

- **Current layer**: the layer which is performing the feedforward/backpropagation
- **Previous/successive layer**: the previous/successive layer of the network w.r.t. the current layer
- z defines the **neuron's output**, a defines the **activation values** ($a = \sigma(z)$, where σ is the activation function)
- w, b define the **weights** and **biases**
- l defines the **index of the current layer**

Basic terminology II

- i, j define the **neuron indices of the current layer** (I, J define the size of the layer), m, n the **neuron indices of the previous layer**
- h, v define the **indices of the weights of a 2D kernel**, while k defines the **size of a 2D kernel**
- r defines a **feature map of the current layer**, while t defines a **feature map of the previous layer** (R and T define the respective depths)
- μ defines the **index of the current observation** processed by the network (M defines the total number of observations)
- s defines the **stride length**

Convolution is a generic term to define (ambiguosly) two methods - correlation and convolution - to **filter** an image (or feature map)

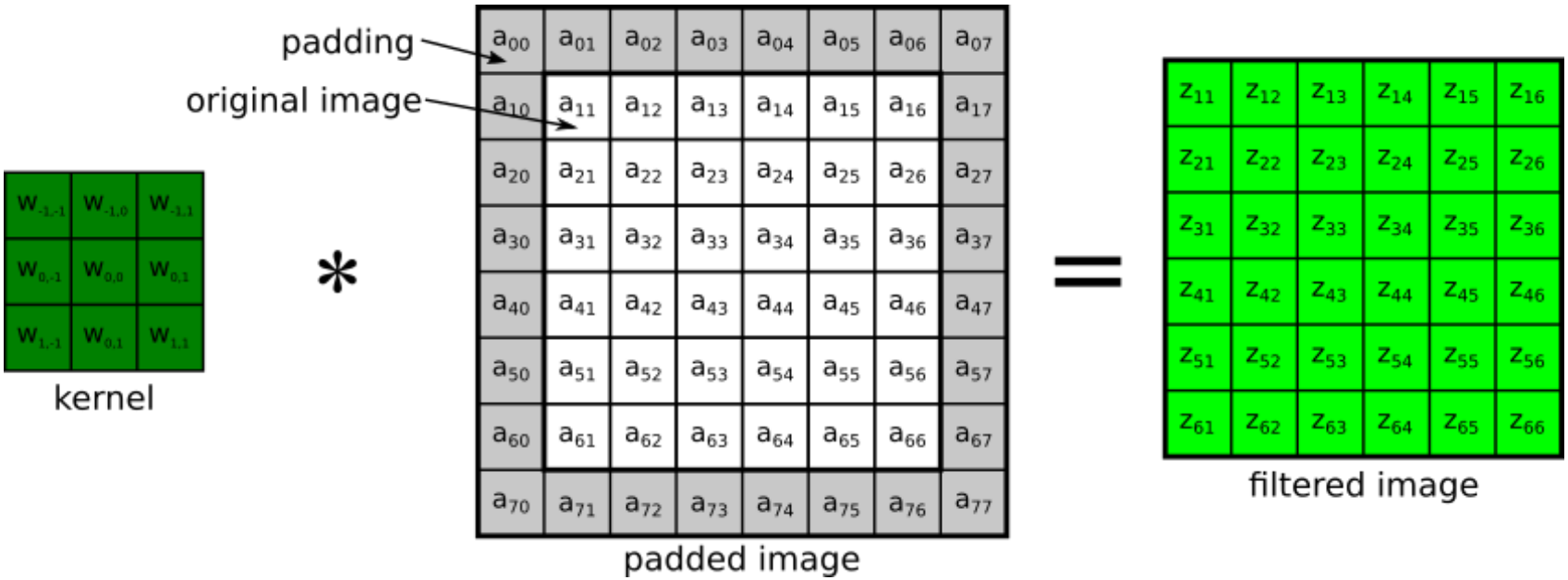
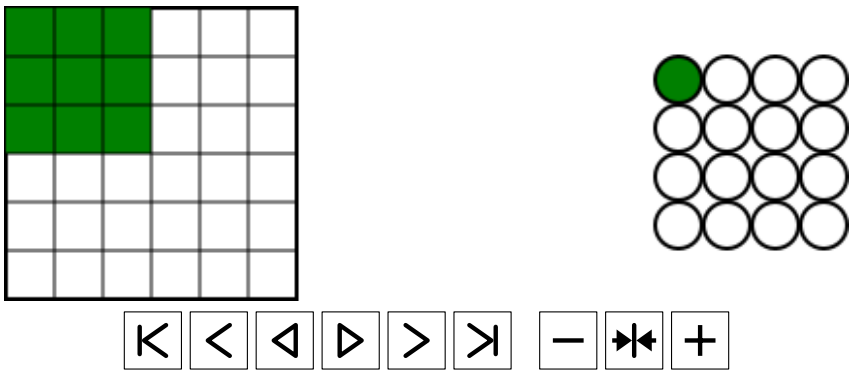


Figure: Convolution process

Convolutional Layers
Base case

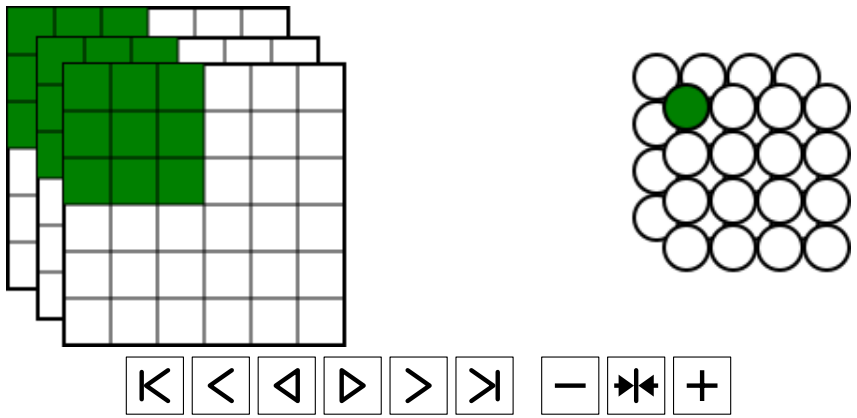
Consider a pair of feature maps from a CL and its previous layer:

- In CL each neuron is connected only to a **small region** of the previous layer, called **local receptive field**
- The weights connecting the two layers are called **kernel**/**filter**, shared between the local receptive fields of the previous layer
- The kernel is convolved with the input obtaining a value for each neuron of the current feature map, detecting in this way a particular feature in previous layer's feature maps
- The shifting length of the kernel is referred as **stride length**



Consider now a CL with R feature maps, and a previous layer with T feature maps:

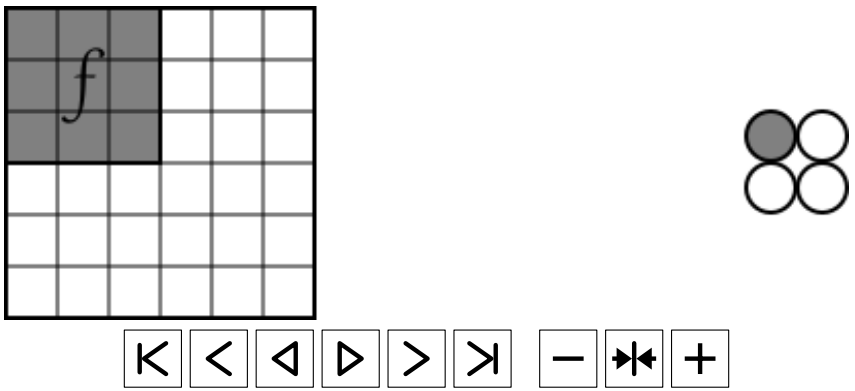
- In this case we deal with R 3D kernels, each formed by a set of T 2D kernels
- Each current feature map is connected through a 2D kernel to each previous feature map
- The convolution step is performed for each of the R current feature maps, summing the T partial results of the 2D convolutional steps for each previous feature map



Pooling Layers
Base case

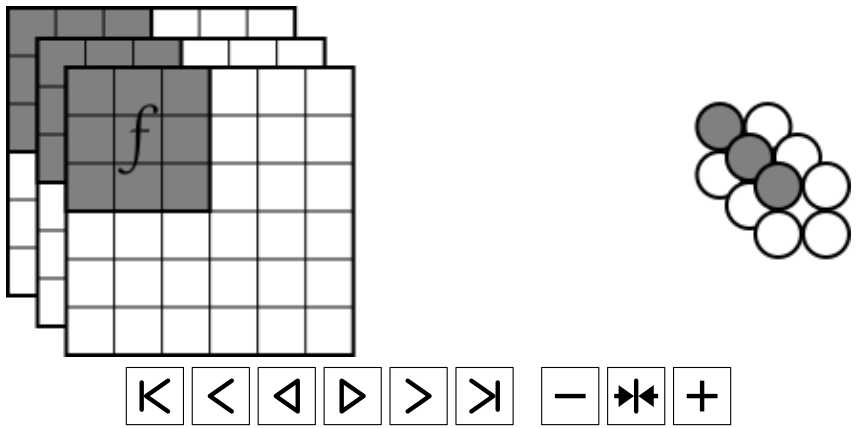
Consider a pair of feature maps from a PL and a previous layer:

- In PL a **down-sampling function** (mean, max, Lp-norm, ...) is applied on a squared region (**window**) of the previous feature map
- The window is then moved and the process is repeated for the next (non-overlapping) region
- The purpose of this down-sampling is to **summarize the information of the previous layer**
- Note that no weight is used in PL



Pooling Layers
General case

- The general case is simply the iteration of the base case for each current feature map
- This is due to one-to-one correspondence between CL and PL feature maps



Consider that a distinct 2D kernel is associated to each pair of current/previous feature maps, thus each 2D kernel depends only on this pair, so:

$$\frac{\partial C}{\partial w^l_{rthv}} = \sum_{\mu=1}^{M} \sum_{i,j} \frac{\partial z^l_{rij}}{\partial w^l_{rthv}} \frac{\partial C_\mu}{\partial z^l_{rij}} = \sum_{\mu=1}^{M} \sum_{i,j} a^{\mu,l-1}_{t,i\cdot s+h,j\cdot s+v} \delta^{\mu,l}_{rij}$$

The complex indexing in the activations related to the previous layer is needed to select the subset of activations which have been multiplied by w_{rthv}

Consider a previous feature map t, which is related to all the current feature maps and to the corresponding 2D kernels at position t. Each neuron of t is related only to the weights which has been convolved with, so:

$$\delta^{\mu,l-1}_{tmn} = \sum_r \sum_{i,j} \sigma'(z^{\mu,l-1}_{tmn}) w^l_{r,t,m-s\cdot i,n-s\cdot j} \delta^{\mu,l}_{rij}$$

Adopting the convention that if the indexing of the weights goes outside the borders of the kernel, then $w_{m-s\cdot i,n-s\cdot j}$ is simply set to zero

The previous formula performs mostly unnecessary iterations (at most $k \times k$ are not 0). In practice the best thing is to **retrace the convolutional steps** updating the related set of neurons exploited in each of them. Since convolutional steps can overlap, some neuron can receive more than one update. The pseudo code is:

$\delta^{\mu,l-1} =$ 3D array with $t \times m \times n$ zeroes

for each feature map t in previous layer

  for i $= 1$ **to** I

    for j $= 1$ **to** J

      $m = i * s$

      $n = j * s$

      $\delta^{\mu,l-1}_{t,m:m+k,n:n+k} \mathrel{+}= \sum_r w_{rt} \delta^{\mu,l}_{r,i,j}$

$\delta^{\mu,l-1}_{tmn} \mathrel{*}= \sigma'(z^{\mu,l-1}_{tmn})$

where w_{rt} represents the 2D kernel related to feature maps t, r

PL backpropagation is easier, since it does not involve weights updates. As for the delta we introduce an operator called **Kronecker's product**. Given two matrices:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \qquad B = \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

The Kronecker's product between A and B is:

$$A \otimes B = \begin{bmatrix} ae & af & be & bf \\ ag & ah & bg & bh \\ ce & cf & de & df \\ cg & ch & dg & dh \end{bmatrix}$$

Consider a pair of feature map r, r (current feature maps are in one-to-one correspondence with previous ones):

- Compute the Kronecker product between $\delta_r^{\mu,l}$ and a matrix of ones with the same size of the pooling window, obtaining D_r
- Similarly as with CL backpropagation, retrace the down-sampling steps on D_r, updating individually each of the related sets of neurons

for each feature map r in current layer

$D_r = \delta_r^{\mu,l} \otimes \text{ones(k,k)}$

for each neuron i, j in current feature map

$m = i \cdot k \quad n = j \cdot k$

$$\delta^{\mu,l-1}_{r,m:m+k,n:n+k} = D_{r,m:m+k,n:n+k} \circ \left[\nabla f_{down}(a^{l-1}_{r,m:m+k,n:n+k})\right]_{k\times k}$$

where the gradient has been accurately reshaped to have the same shape of $D_{r,m:m+k,n:n+k}$ and $\circ$ denotes the **Hadamard (element-wise) product**

- When using sigmoid + quadratic cost function, backpropagation can require many steps to converge, especially when the error between the predicted and the expected output values is high, because of the partial derivatives of the sigmoid
- A solution is to use another output activation function, i.e. the **softmax**: $a_j^L = e^{z_j^L} / \sum_k e^{z_k^L}$
- Alternatively we can change the cost function, for example employing the **cross-entropy**:
  $-\frac{1}{n} \sum_\mu^M \sum_j \left[y_j ln(a_j^L) + (1 - y_j) ln(1 - a_j^L) \right]$
- Both solutions can be (and usually are) used together

- A single update of weights configuration would require the calculation of the gradient for all the observations
- Considering that a single update step would be unfeasible in terms of time taken, we resort to the **Stochastic Gradient Descent** (SGD):
  1) For each epoch:
  2) Divide the training set in randomly chosen mini-batches
  3) For each batch:
  4) Update the weights, with backpropagation
- SGD is an approximation of the the classic GD method that performs more weights updates in each epoch. It remains reliable adding very small inaccuracies in the minimization

- Dropout is a practice meant to **lower the overfitting** due to the structure of the network
- Each time an observation is fed to the network, some weight of the layer is set to 0 with a probability p, in order to change the overall structure.
- At test time the weights are multiplied by p in order to obtain an **expected output**
- Dropout works well with $p \approx 0.5$ (except for input layer where $p \approx 1$)

- **Initial aim:** write from scratch some code for **offline** classification of hanzi (Chinese characters) with convolutional neural networks

  $\Rightarrow$ An unrealistic quest, **thousands** of hanzi **very similar to each other**

- **Fallback plan:**
  1. **MNIST-cnn** [11]: "academic" implementation of a convolutional neural network
  2. **CASIA-HWDB1.1-cnn** [12]: hanzi classification using state-of-the-art tools

- The **MNIST database** [13] of handwritten digits
  - One of the most used data sets in machine learning
  - Digits size-normalized and centered in a 28x28 image (no preprocessing needed)
  - Training set of 60,000 examples, test set of 10,000 examples
  - Current record is 99.79% [3] of test accuracy (only 21 misclassified images on 10,000)

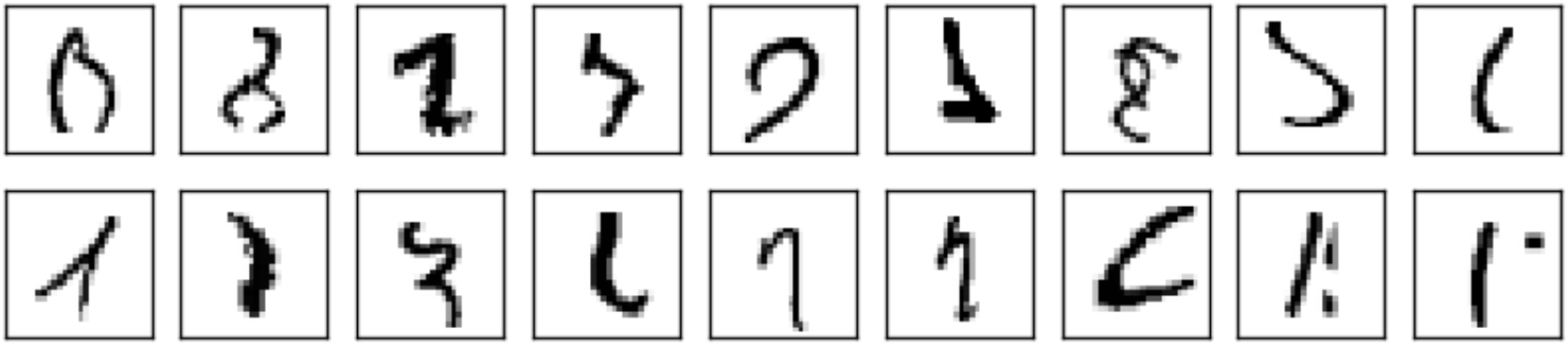
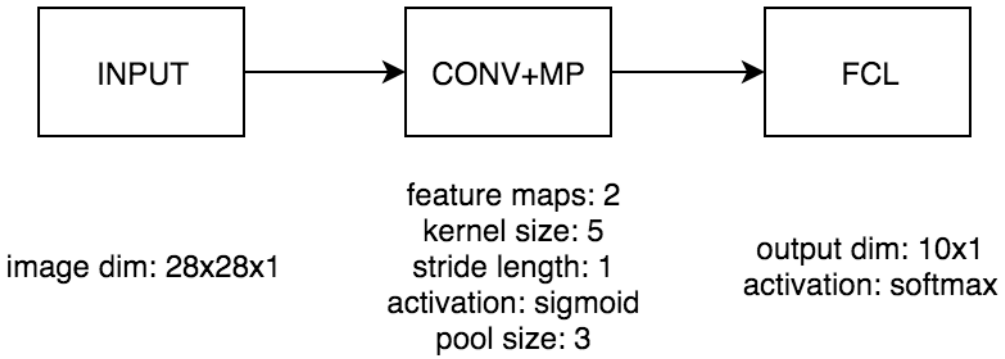


Figure: Some really bad digits [6]

- **Python**
- **NumPy** [15] the fundamental package for scientific computing
  - Based on the **ndarray** object: an n-dimensional array of homogeneous data type
  - Provides operations on arrays in compiled code at near-C speeds
  - Expanded by **SciPy** [16]: a collection of numerical algorithms on signal processing, optimization, statistics, ...



- Activations: **sigmoid** and **softmax**
- Weights initialization: **Glorot uniform** [4], i.e. $\mathcal{U}(-s, s)$ with $s = \sqrt{\frac{6}{\text{n\_in}+\text{n\_out}}}$
- Biases initialization: **zero**
- Loss function: **cross-entropy**
- Optimizer: **stochastic gradient descent**

- Learning rate: 0.1, number of epochs: 2, batch size: 8
  $\Rightarrow$ Test accuracy: **84.58%**
- There exist intrinsic **upper bounds** for improvements, due to **numerical errors** (overflow, truncation, ...)
- Error increases with the number of parameters
- Evaluating the **correctness of the code** is difficult: the network can perform reasonably well without being flawless

- The **CASIA-HWDB1.1 database** [18] of handwritten hanzi
  - Offline (no stroke trajectory)
  - 3,755 Chinese characters from 300 writers
  - Training set of 897,758 examples, test set of 223,991 examples
  - Current records reach 99% [19] of test accuracy

Figure: Three hanzi from different writers [5]

- **Python**
- **Theano** [21] for computations on multi-dimensional arrays
  - Transparent use of a **GPU** (up to 140x faster than CPUs)
  - **Symbolic differentiation** and **stability optimizations**
- **Keras** [22] for fast experimentation
  - Supports **convolutional networks**
  - Plenty of **layers**, **activations**, **optimizers** and **objectives** to build complex networks in no time
  - Can be used together with Theano to run on GPUs

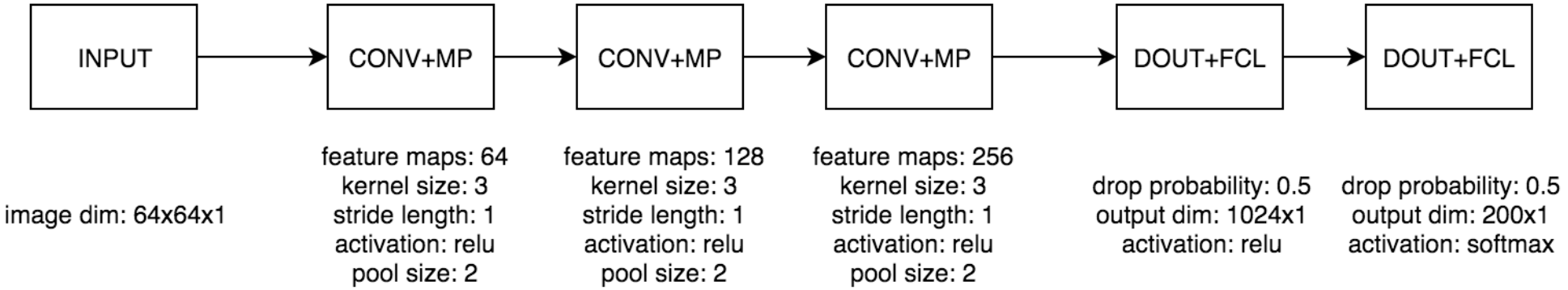
CASIA-HWDB1.1-cnn

The process [5]

1. Data set normalization
   - From **.gnt** to **.hdf5** for portability/ease of use
   - **Padding** and **rescaling** to obtain 64x64 centered images
2. Subset extraction and bitmap processing
   - From **3755** classes to **200**
   - From approximately **240** training samples per class to **200** (the remaining 40 samples used as validation set)
   - **Contrast stretching**
3. Network training
   - Large number of **parameters**
4. Network evaluation

CASIA-HWDB1.1-cnn

The network



- Activations: **rectifier** (ReLU) and **softmax**
- Weights initialization: $\mathcal{N}(\mu{=}0, \sigma^2{=}0.1)$ and **Glorot uniform**
- Biases initialization: **zero**
- Loss function: **cross-entropy**
- Optimizer: **AdaDelta** [23] (adaptive learning rate)

- Number of epochs: 15, batch size: 100
  $\Rightarrow$ Test accuracy: **94.58%**
- Image preprocessing can significantly influence results
- Lots of tries + lots of errors = resource and time consuming
- Infeasible on CPUs
- Given enough computational resources similar and better results can be achieved also on the entire data set

Conclusions

- We developed two projects with different aims, obtaining on both satisfactory results
- **MNIST-cnn** could be improved by:
  - Using existing libraries for symbolic computations (to minimize numerical errors)
  - Translating computationally-intensive tasks (e.g. convolution) into Cython or GPU code
- **CASIA-HWDB1.1-cnn** could be improved by:
  - Parameters tuning
- In general, **results can be easily improved given enough resources to tune and test a lot of network models**

References I

Bibliography and sitography

[1] L. F. Karlström, E. G. López, *A Modular Handwritten Kanji Recognition Schema*, 2014.
http://enc2014.cicese.mx/Memorias/paper_89.pdf

[2] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, *Dropout: A simple way to prevent neural networks from overfitting*, 2014.
https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf

[3] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, R. Fergus, *Regularization of neural networks using dropconnect*, 2013.
http://cs.nyu.edu/~wanli/dropc/

References II

Bibliography and sitography

[4] X. Glorot, Y. Bengio, *Understanding the difficulty of training deep feedforward neural networks*, 2010.
http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf

[5] Y. Zhang, *Deep Convolutional Network for Handwritten Chinese Character Recognition*
http://cs231n.stanford.edu/reports/zyh_project.pdf

[6] *Neural Networks and Deep Learning*
http://neuralnetworksanddeeplearning.com/

[7] *Convolutional Neural Networks (LeNet)*
http://deeplearning.net/tutorial/lenet.html

[8] *Backpropagation in Convolutional Neural Network*
http://www.slideshare.net/kuwajima/cnnbp

References III

Bibliography and sitography

[9] *Convolutional Neural Networks*
http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/

[10] *Exercise: Convolutional Neural Network*
http://ufldl.stanford.edu/tutorial/supervised/ExerciseConvolutionalNeuralNetwork/

[11] https://github.com/integeruser/MNIST-cnn

[12] https://github.com/integeruser/CASIA-HWDB1.1-cnn

[13] http://yann.lecun.com/exdb/mnist/

[14] Wan, Li and Zeiler, Matthew and Zhang, Sixin and Cun, Yann L and Fergus, Rob. *Regularization of neural networks using dropconnect*. 2013. http://cs.nyu.edu/~wanli/dropc/

[15] http://www.numpy.org/

[16] http://www.scipy.org/

[17] Glorot, Xavier and Bengio, Yoshua. *Understanding the difficulty of training deep feedforward neural networks*. 2010.

[18] http://www.nlpr.ia.ac.cn/databases/handwriting/Home.html

[19] http://people.idsia.ch/~ciresan/results.htm

[20] Zhang, Yuhao. *Deep Convolutional Network for Handwritten Chinese Character Recognition*

[21] http://deeplearning.net/software/theano/

[22] http://keras.io/

[23] Zeiler, Matthew D. *ADADELTA: an adaptive learning rate method*. 2012.