CNN Tutorial

Theory and Application

WuBingzhe

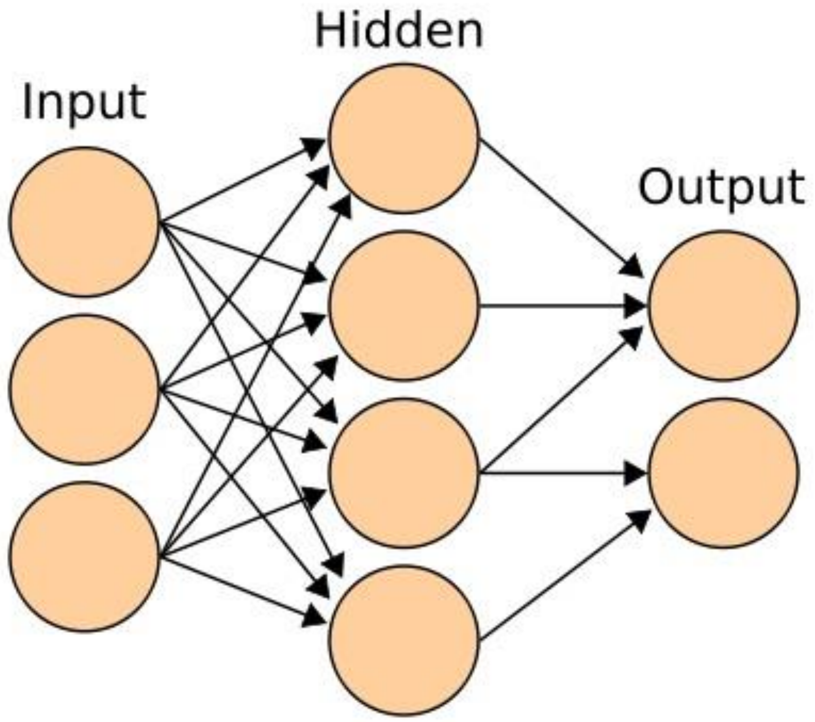
Outline

The Early Neural Network Models

- Computational models of neural networks have been around for more than half a century, beginning with the simple model that McCulloch and Pitts developed in 1943 .
- In Minsky extremely influencial book,*Perceptrons*, Minsky and Papert proved that these networks couldn't even learn the boolean XOR function.
- A more complex learning algorithm, backpropagation, eventually emerged, which could learn across arbitrarily many layers, and was later proven to be capable of approximating any computable function.

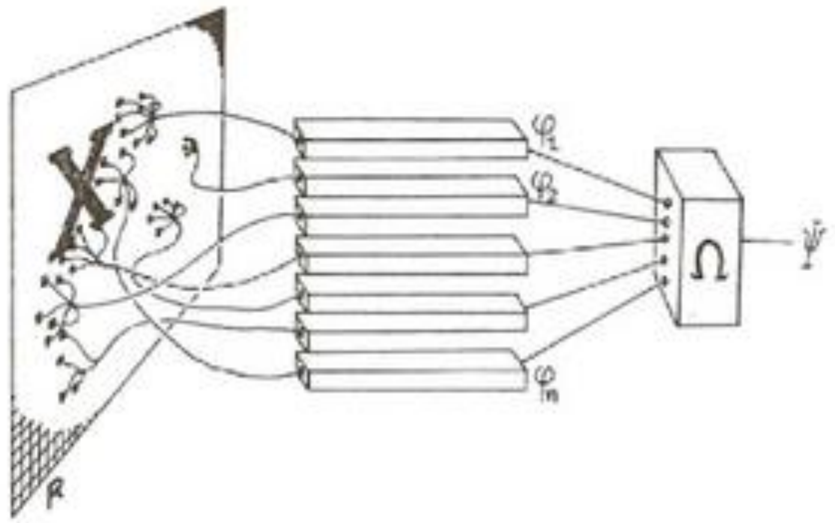
The Early Neural Network Models

- A typical feed-forward neural network architecture used in backpropagation.



Problems with Backpropagation

- Despite being a universal function approximator in theory, FFNNs weren't good at dealing with many sorts of problems in practice.
- The example relevant to the current discussion is the FFNN's poor ability to recognize objects presented visually.

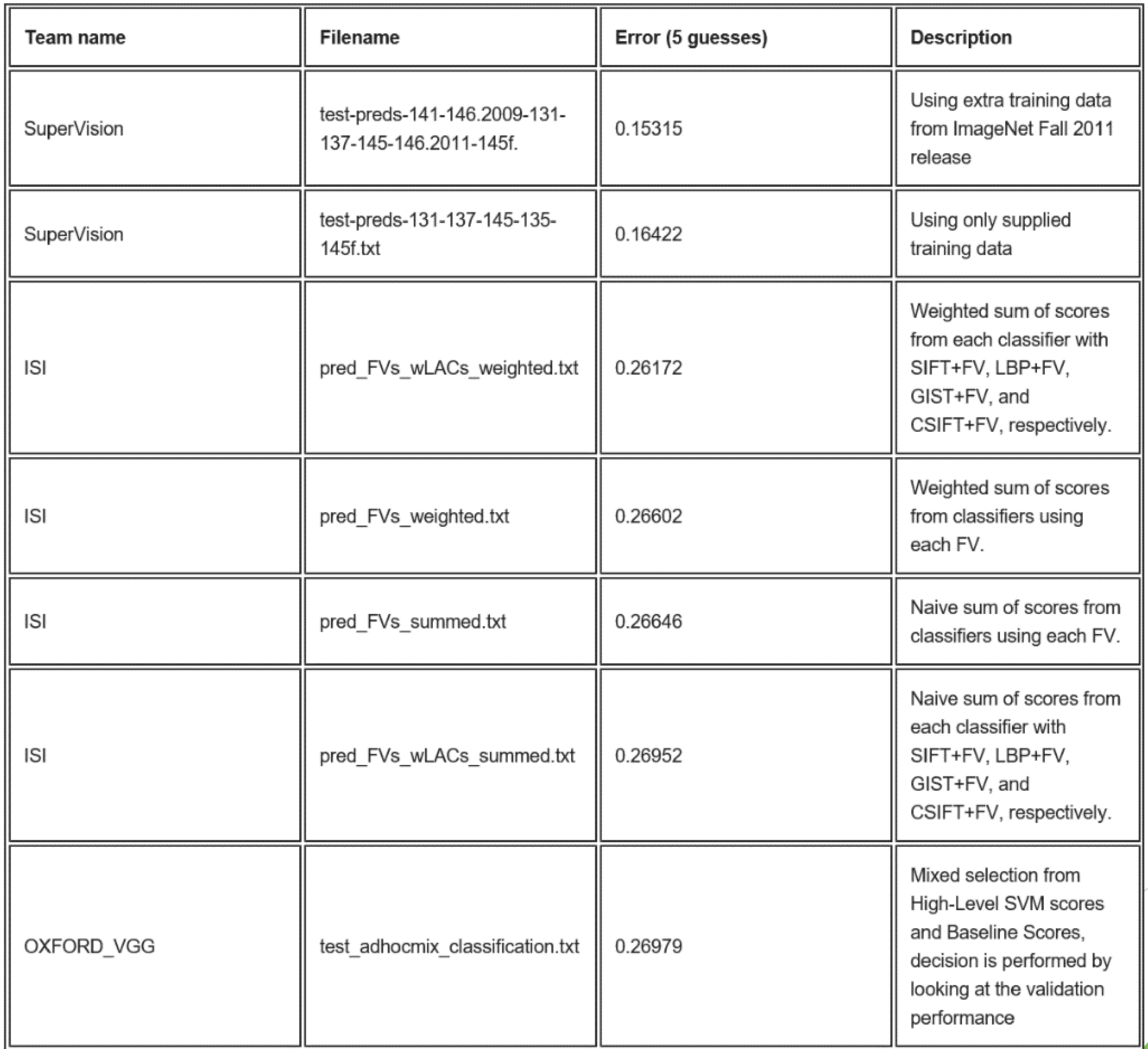


Problems with Backpropagation

- Since every unit in a pool was connected to every unit in the next pool, the number of weights grew very rapidly with the dimensionality of the input.
- Since every pair of neurons between two pools had their own weight, learning to recognize a object in one location wouldn't transfer to the same object presented in a different part of the visual field; separate weights would be involved in that calculation.

The Background

- From Hubel and Wiesel’s early work on the cat’s visual cortex.
- LeNet (HandWriting Recognition, Yann LeCun)
- Imagenet Dataset (Li FeiFei,14,197,122 images)
- The development of GPU computing
- Large Scale Visual Recognition Challenge 2012 (Hinton,deep net)



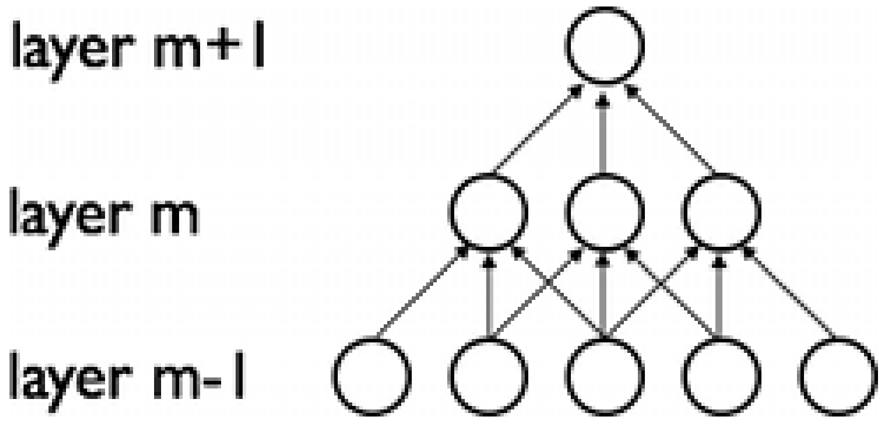
Team name	Filename	Error (5 guesses)	Description
SuperVision	test-preds-141-146.2009-131-137-145-146.2011-145f.	0.15315	Using extra training data from ImageNet Fall 2011 release
SuperVision	test-preds-131-137-145-135-145f.txt	0.16422	Using only supplied training data
ISI	pred_FVs_wLACs_weighted.txt	0.26172	Weighted sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively.
ISI	pred_FVs_weighted.txt	0.26602	Weighted sum of scores from classifiers using each FV.
ISI	pred_FVs_summed.txt	0.26646	Naive sum of scores from classifiers using each FV.
ISI	pred_FVs_wLACs_summed.txt	0.26952	Naive sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively.
OXFORD_VGG	test_adhocmix_classification.txt	0.26979	Mixed selection from High-Level SVM scores and Baseline Scores, decision is performed by looking at the validation performance

The Core Idea

- the visual cortex contains a complex arrangement of cells. These cells are sensitive to small sub-regions of the visual field, called a receptive field.
- Sparse Connectivity
- Shared Weights

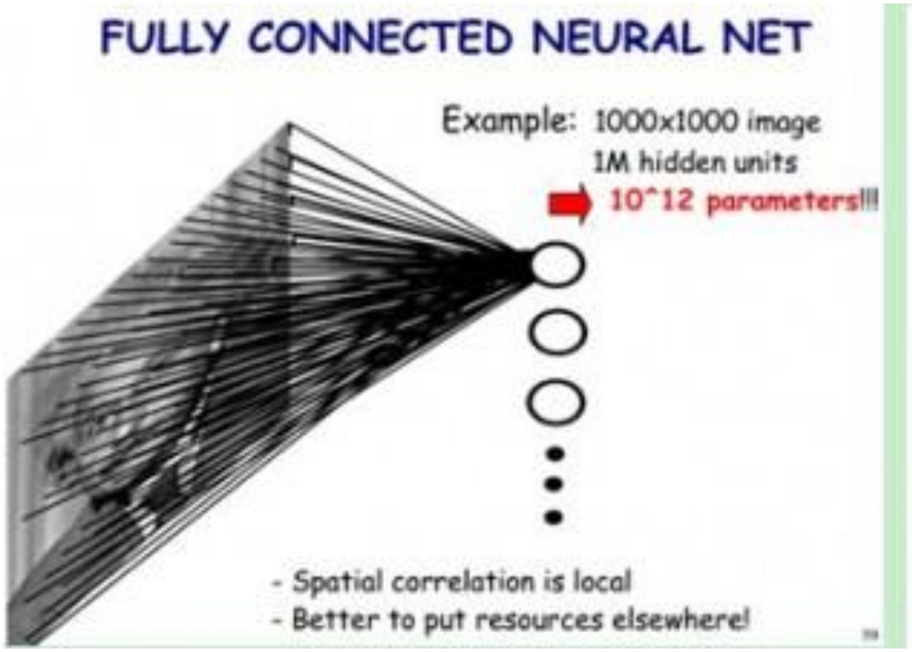
Sparse Connectivity

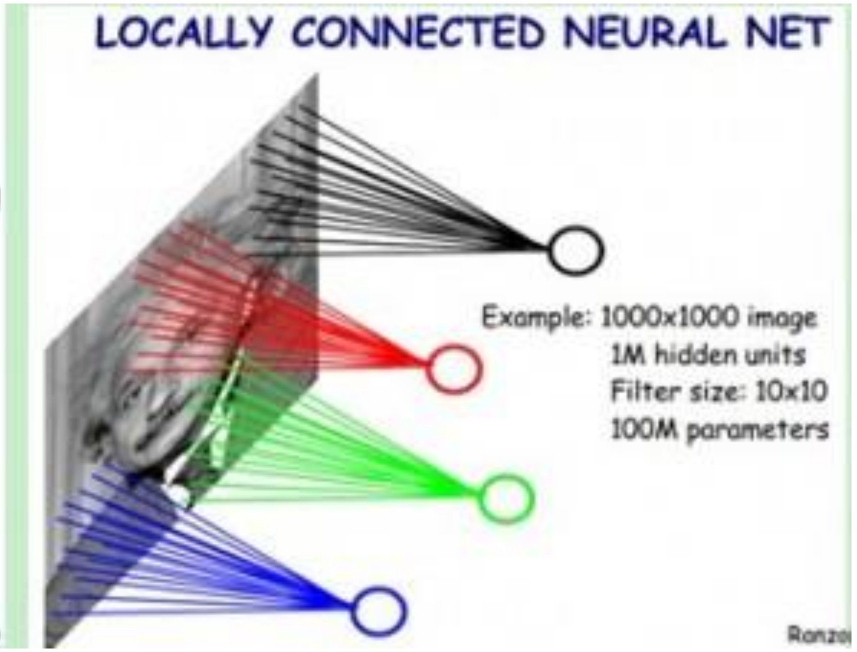
- CNNs exploit spatially-local correlation by enforcing a local connectivity pattern between neurons of adjacent layers.



In the above figure, units in layer **m** have receptive fields of width 3 in the input retina and are thus only connected to 3 adjacent neurons in the retina layer.

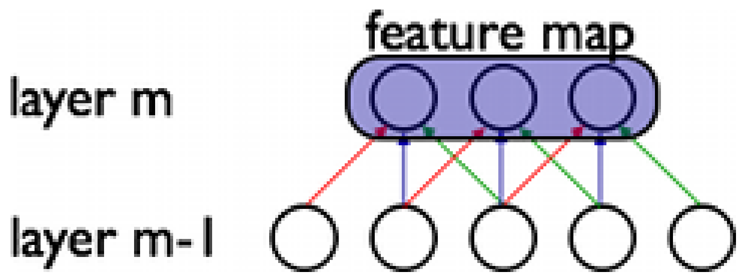
Sparse Connectivity


FULLY CONNECTED NEURAL NET
Example: 1000x1000 image
1M hidden units
10^12 parameters!!!
- Spatial correlation is local
- Better to put resources elsewhere!



Shared Weights

- In addition, in CNNs, each filter h_i is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a *feature map*.



- In the above figure, we show 3 hidden units belonging to the same feature map. Weights of the same color are shared—constrained to be identical.

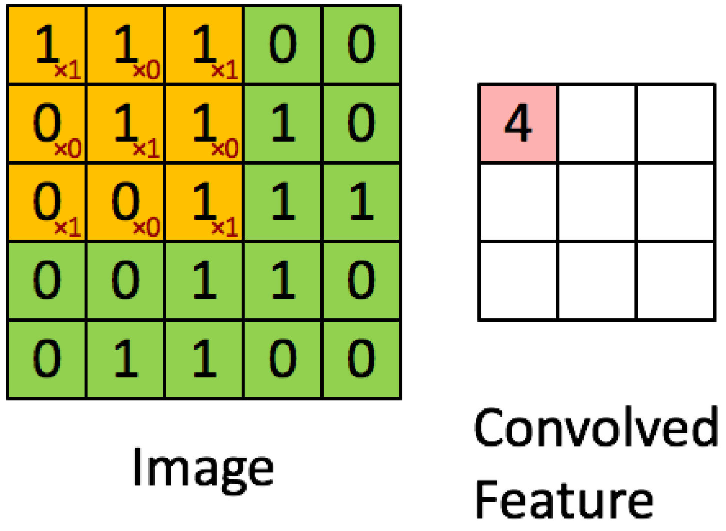
The Convolutional Layer

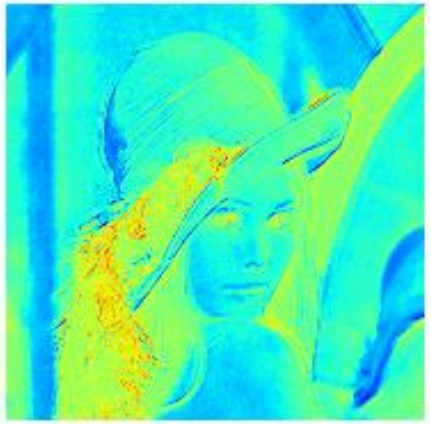
- A feature map is obtained by repeated application of a function across sub-regions of the entire image.
- In other words, by *convolution* of the input image with a linear filter, adding a bias term and then applying a non-linear function.
- If we denote the k-th feature map at a given layer as h^k , whose filters are determined by the weights W^k and bias b^k ,The feature map is obtained as follows:

$$h_{ij}^{k}=\tanh\left(\left(W^{k} * x\right)_{ij} + b_{k}\right)$$

The Convolutional Layer

- To form a richer representation of the data, each hidden layer is composed of *multiple* feature maps.
- The weights W of a hidden layer can be represented in a 4D tensor .





Subsampling

- Subsampling, or down-sampling, refers to reducing the overall size of a signal. In many cases, such as audio compression for music files, subsampling is done simply for the size reduction.
- But in the domain of 2D filter outputs, subsampling can also be thought of as increasing the position invariance of the filters. The specific subsampling method used in LeNets is known as 'max pooling'.

Max-Pooling

- Reducing computation for upper layers
- Preventing over-fitting

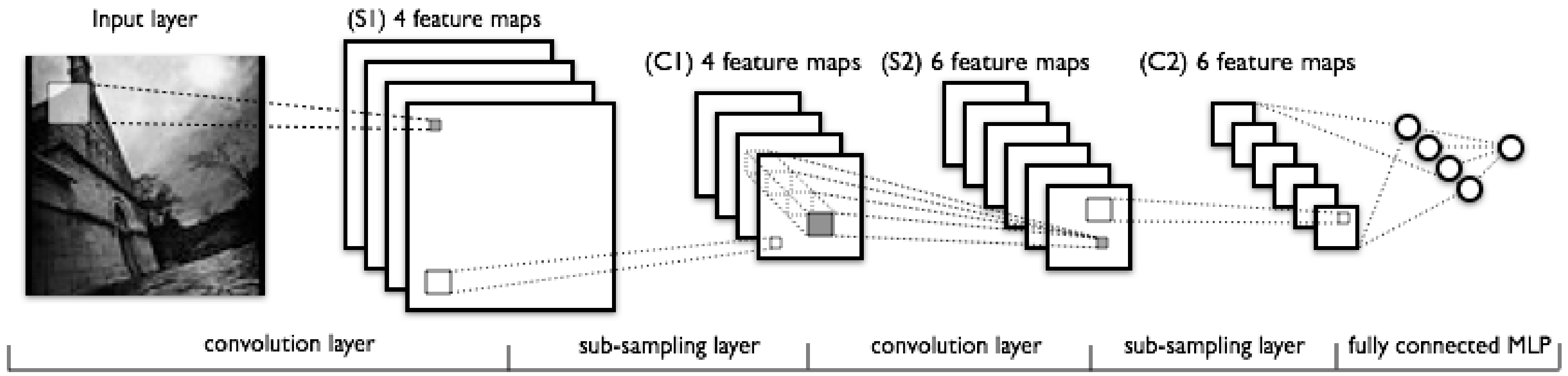
Softmax

- $P(Y = i|x, W, b) = softmax_i(Wx + b) = \frac{e^{W_i \ x + b_i}}{\sum_j e^{W_j \ x + b_j}}$
- $y_{pred} = argmax_i P(Y = i|x, W, b)$

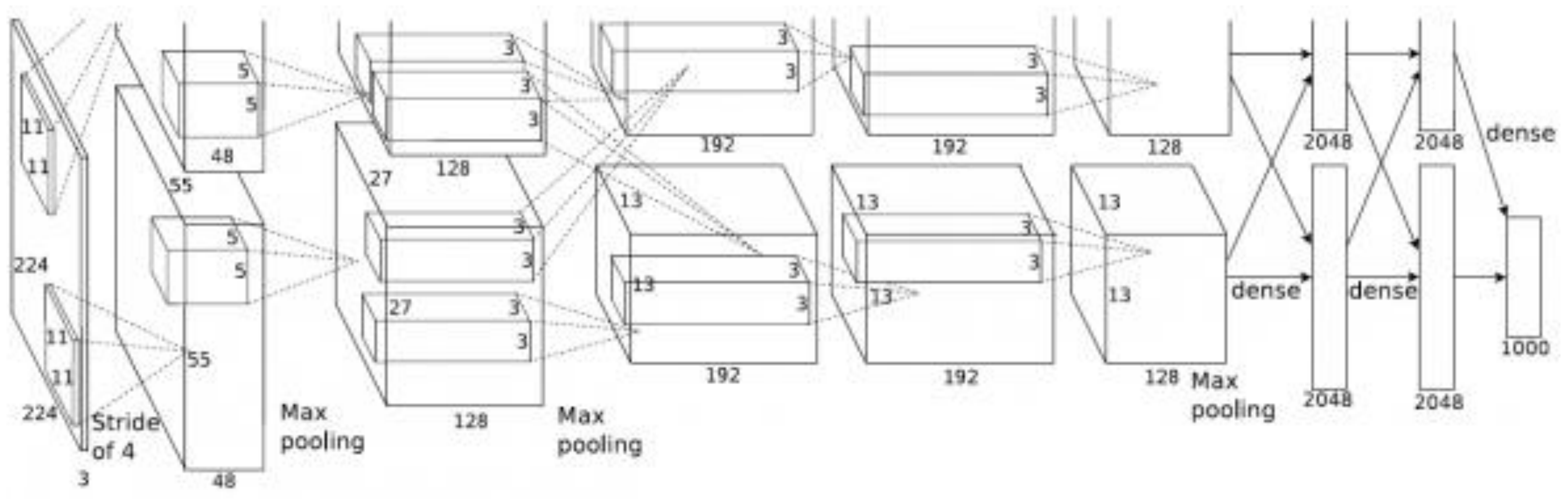
Convolutional Neural Networks

- Convolve several small filters on the input image
- Subsample this space of filter activations
- Repeat steps 1 and 2 until your left with sufficiently high level features.
- Use a standard a standard FFNN to solve a particular task, using the results features as input.

LeNet


Input layer
(S1) 4 feature maps
(C1) 4 feature maps
(S2) 6 feature maps
(C2) 6 feature maps
convolution layer
sub-sampling layer
convolution layer
sub-sampling layer
fully connected MLP

Alex Net


224
224
3
11
11
11
11
Stride of 4
55
55
48
48
5
5
5
5
Max pooling
27
27
128
128
3
3
3
3
Max pooling
13
13
192
192
13
13
192
192
13
13
128
128
Max pooling
dense
dense
dense
2048
2048
2048
2048
1000

A Running Example

- Traning LeNet on MNIST with Caffe
- Assume that the Caffe installation is located at CAFFE_ROOT

Prepare Datasets

- You will first need to download and convert the data format .
- To do this simply run the following commads:

```
cd $CAFFE_ROOT
./data/mnist/get_mnist.sh
./examples/mnist/create_mnist.sh
```

- After running the script there should be two datasets, mnist_train_lmdb, and mnist_test_lmdb.

Define the MNIST Network

- Data Layer
- Convolution Layer
- Pooling Layer
- Fully Connected Layer
- ReLU(tanh) Layer
- MNIST Solver

Writing the Data Layer

- Currently, we will read the MNIST data from the lmdb we created earlier in the demo. This is defined by a data layer:

```
layer {
  name: "mnist"
  type: "Data"
  data_param {
    source: "mnist_train_lmdb"
    backend: LMDB
    batch_size: 64
    scale: 0.00390625
  }
  top: "data"
  top: "label"
}
```

Writing the Convolution Layer

- This layer takes the data blob (it is provided by the data layer), and produces the conv1 layer. It produces outputs of 20 channels, with the convolutional kernel size 5 and carried out with stride 1.

```
layer {
  name: "conv1"
  type: "Convolution"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  convolution_param {
    num_output: 20
    kernel_size: 5
    stride: 1
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "data"
  top: "conv1"
}
```

Writing the Pooling Layer

```
layer {
  name: "pool1"
  type: "Pooling"
  pooling_param {
    kernel_size: 2
    stride: 2
    pool: MAX
  }
  bottom: "conv1"
  top: "pool1"
}
```

Similarly, you can write up the second convolution and pooling layers

Writing the Fully Connected Layer

```
layer {
  name: "ip1"
  type: "InnerProduct"
  param { lr_mult: 1 }
  param { lr_mult: 2 }
  inner_product_param {
    num_output: 500
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
    }
  }
  bottom: "pool2"
  top: "ip1"
}
```

ReLU Layer And Loss Layer

```
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "ip1"
  top: "ip1"
}
```

```
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "ip2"
  bottom: "label"
}
```

Define the MNIST Solver

```
# The train/test net protocol buffer definition
net: "examples/mnist/lenet_train_test.prototxt"
# test_iter specifies how many forward passes the test should carry out.
# In the case of MNIST, we have test batch size 100 and 100 test iterations,
# covering the full 10,000 testing images.
test_iter: 100
# Carry out testing every 500 training iterations.
test_interval: 500
# The base learning rate, momentum and the weight decay of the network.
base_lr: 0.01
momentum: 0.9
weight_decay: 0.0005
# The learning rate policy
lr_policy: "inv"
gamma: 0.0001
power: 0.75
# Display every 100 iterations
display: 100
# The maximum number of iterations
max_iter: 10000
# snapshot intermediate results
snapshot: 5000
snapshot_prefix: "examples/mnist/lenet"
# solver mode: CPU or GPU
solver_mode: GPU
```

Run The Example

- Training the model is simple after you have written the network definition protobuf and solver protobuf files. Simply run train_lenet.sh, or the following command directly:

```
cd $CAFFE_ROOT .
```

```
./build/tools/caffe train -solver
=examples/mnist/solver.prototxt
```

The Results

Type	Classifier	Error rate
Linear Classifier	Pairwise Linear Classifier	7.6
K-NN	K-NN	0.52
Non-Linear Classifier	PCA Quadratic Classifier	3.3
Neural Network	2-layer 784-800-10	1.6
Deep Neural Network	6-layer	0.35
CNN	LeNet	