Convolutional Neural Nets

Using MXNet

Constantin Gonzalez, Principal Solutions Architect,

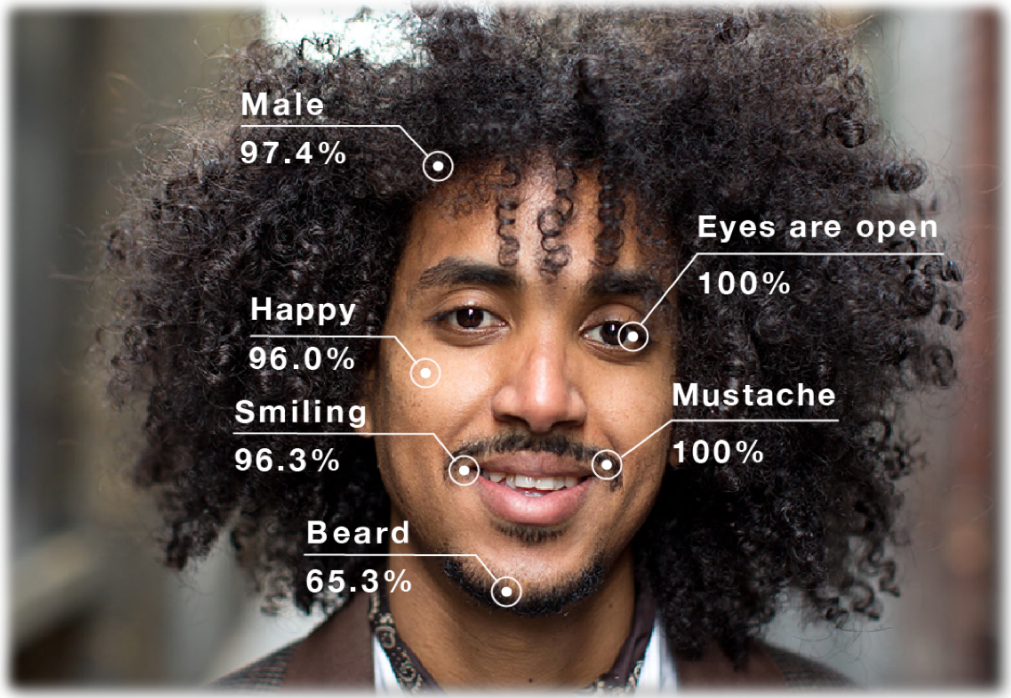
Principal Solutions Architect, Amazon Web Services

glez@amazon.de

November 2017

Sparse Matrix and Spatial Correlation

Feature Detection



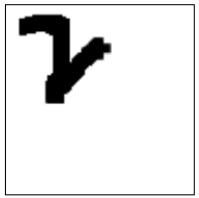
Non-Convolutional Network

In [13]:
```
from IPython.display import HTML
import cv2
import numpy as np

def classify(img):
    img = img[len(''):].decode('base64')
    img = cv2.imdecode(np.fromstring(img, np.uint8), -1)
    img = cv2.resize(img[:,:,3], (28,28))
    img = img.astype(np.float32).reshape((1,1,28,28))/255.0
    return model.predict(img)[0].argmax()

'''
To see the model in action, run the demo notebook at
https://github.com/dmlc/mxnet-notebooks/blob/master/python/tutorials/mnist.ipynb.
'''
HTML(filename="mnist_demo.html")
```

Out[13]:



Classify Clear Result: 5

Convolution

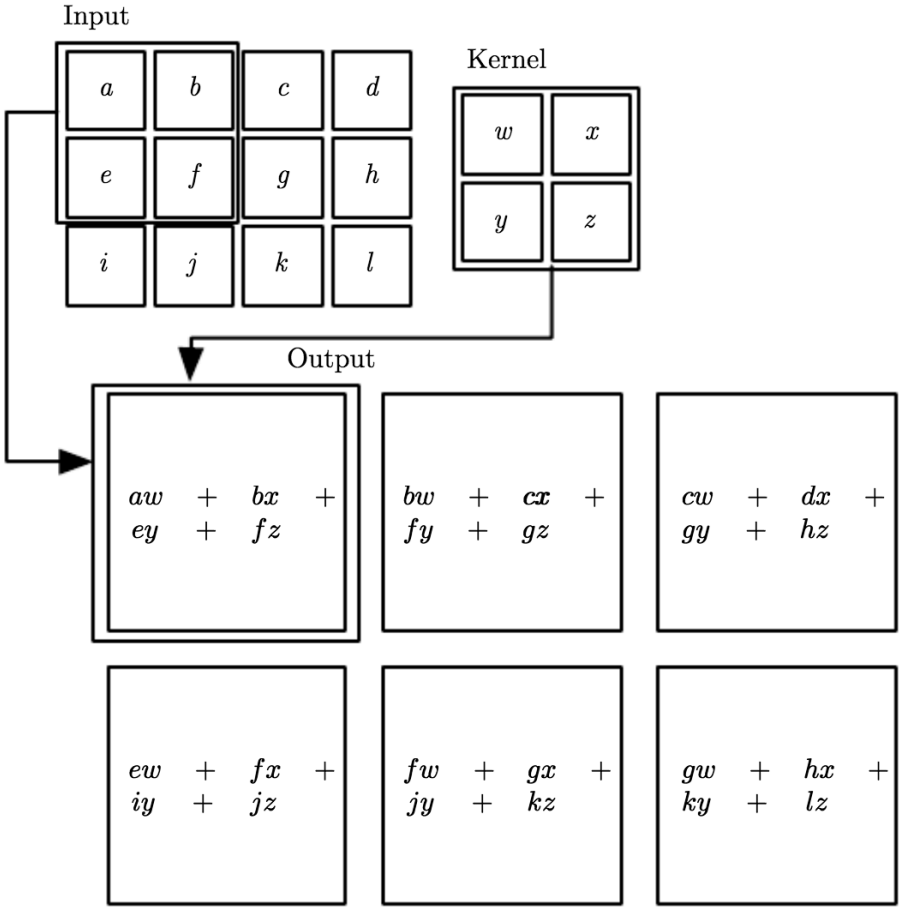
- Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers..
- We use a reduction mechanism that is weighted differently based on relevance.
  - Example: Spaceship measurement along a path creates a discrete set of measurement. Each one could be fuzzy, but averaging them helps remove the noise, and have better prediction on the current location with more weight given to the local position.

$$S(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- x is often called ***input*** (often multi-dimensional array of data) and w is called ***kernel*** (often multi-dimensional array of parameters).

aws

Convolution

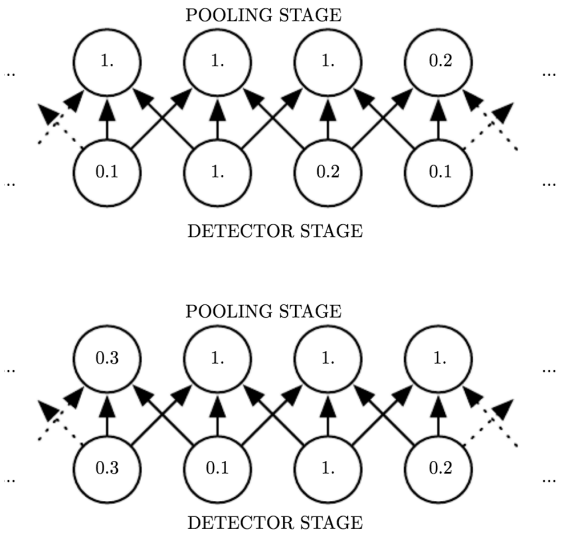


http://www.deeplearningbook.org/contents/convnets.html

aws

Pooling

- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.
- Max Pooling operation reports the maximum output within a rectangular neighborhood.
- Pooling helps detect existence of features as opposed to detecting where a feature is through making a representation invariant to small translation in the input.

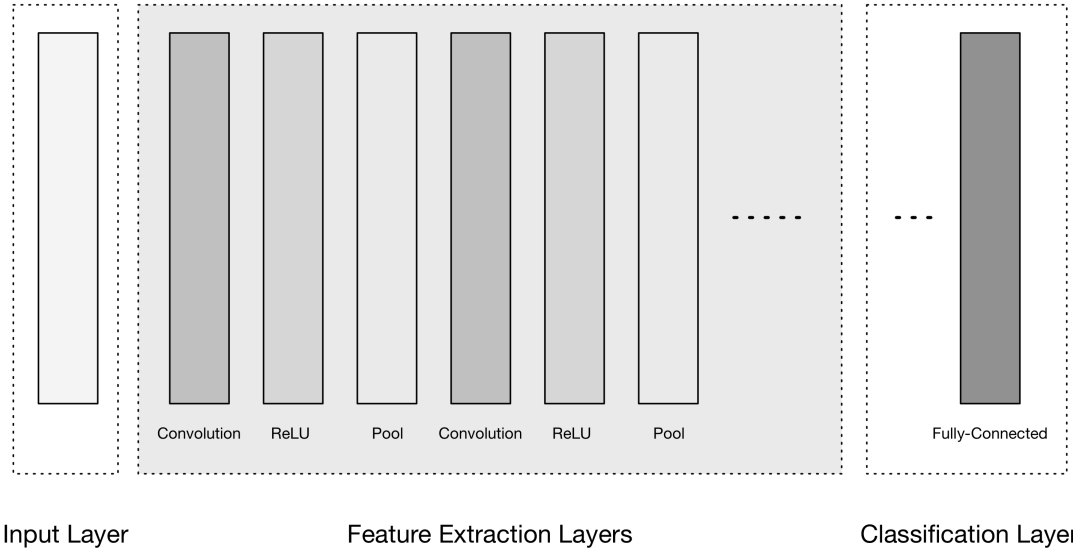


After stride of one pixel, the pooling stage has fewer changes compared to detector stage

http://www.deeplearningbook.org/contents/convnets.html

Convolutional Neural Networks - Architecture

- Input Layer takes the raw array of data.
- Feature Extraction layers extract features through:
  - The first layer performs several convolutions in parallel to produce a set of linear activations.
  - In the second stage (detector), each linear activation is run through a nonlinear activation function, such as ReLU
  - The third layer performs pooling on the output
- In the end fully-connected layers, reassemble the features into final output and apply Softmax to predict create a probabilistic distribution.



Convolution in Example

Input Volume (1x5x5 +pad 1)

0	0	0	0	0	0	0
0	12	31	12	167	255	0
0	**45**	**54**	67	254	254	0
0	**0**	**25**	238	254	254	0
0	**10**	**196**	254	254	251	0
0	127	254	254	238	98	0
0	0	0	0	0	0	0

Receptive Field I_{21}

0	45	54
0	0	25
0	10	196

Kernel K (1x3x3)

1	0	-1
1	1	0
-1	0	1

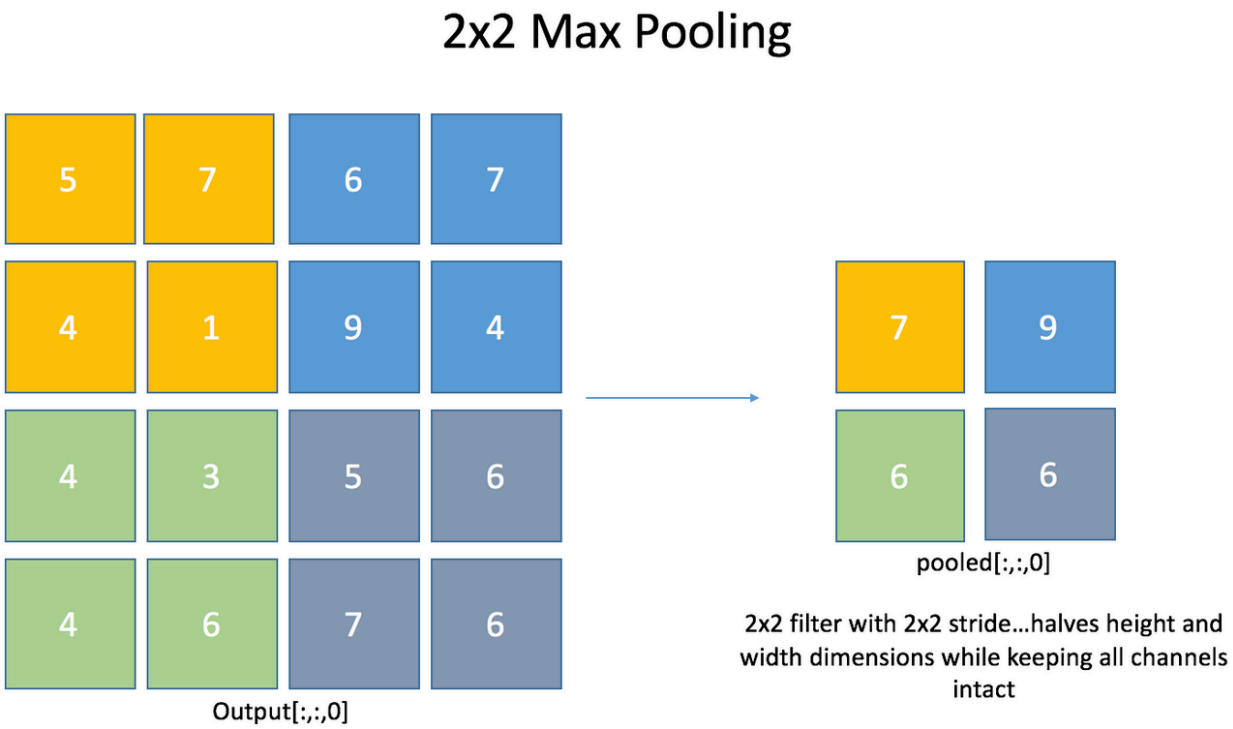
Inner Product $< I_{21}, K >$

$$= 0 \cdot 1 + 45 \cdot 0 + 54 \cdot -1 + 0 \cdot 1 + 0 \cdot 1 + 25 \cdot 0 + 0 \cdot -1 + 10 \cdot 0 + 196 \cdot 1 = \mathbf{142}$$

Output

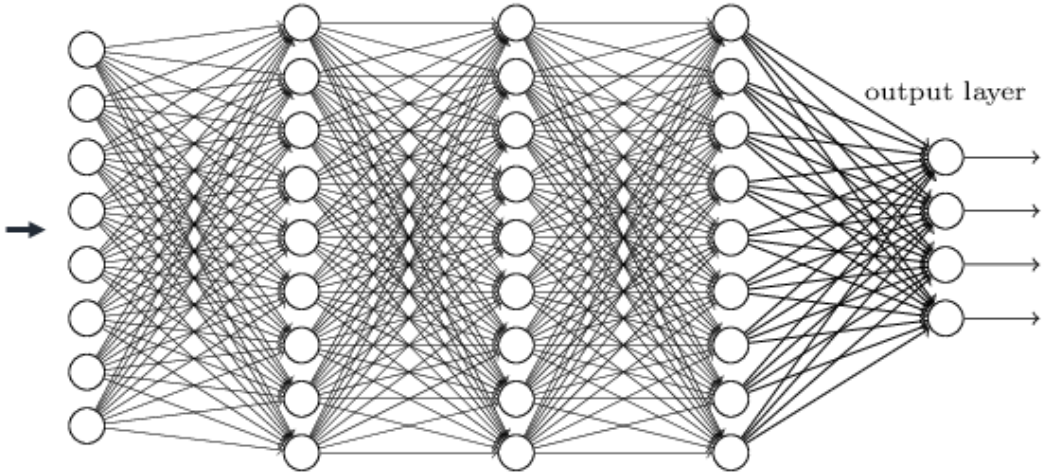
0	243	
142		

Max Pooling in Example



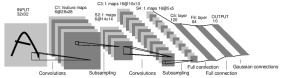
CNN Feature Detection

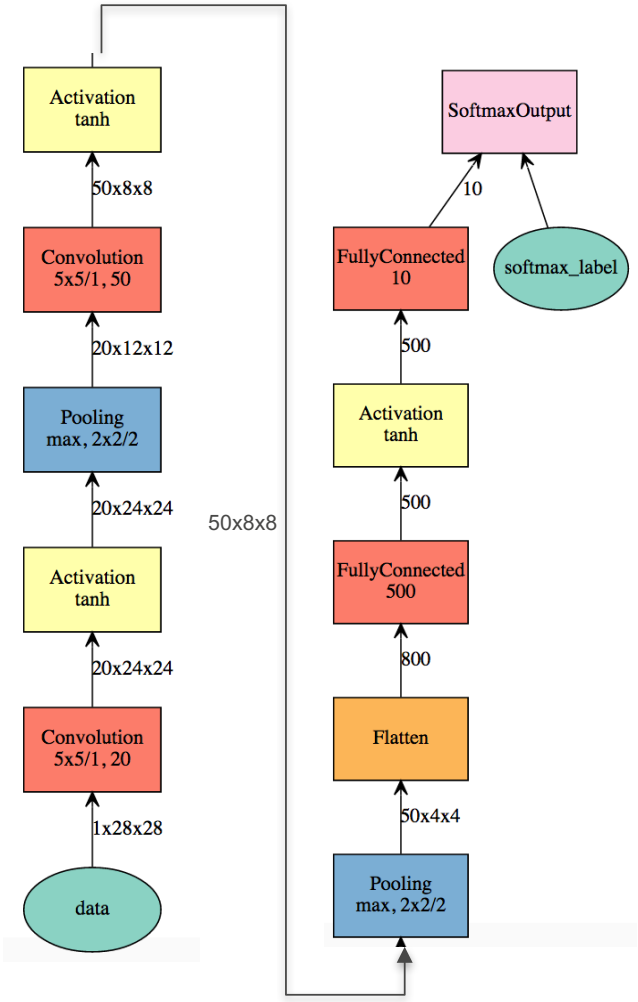
Deep neural networks learn hierarchical feature representations



aws

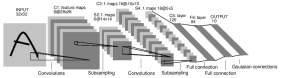
CNN in MXNet



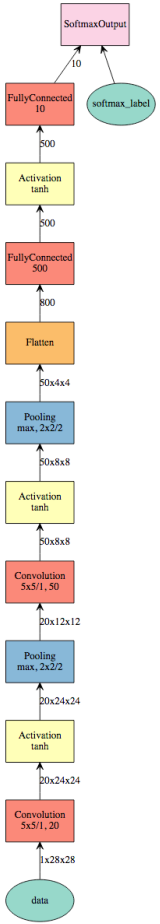


aws

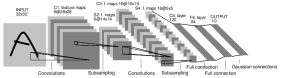
LeCun 5



```
data = mx.symbol.Variable('data')
# first conv layer
conv1 = mx.sym.Convolution(data=data, kernel=(5,5), num_filter=20)
tanh1 = mx.sym.Activation(data=conv1, act_type="tanh")
pool1 = mx.sym.Pooling(data=tanh1, pool_type="max", kernel=(2,2), stride=(2,2))
# second conv layer
conv2 = mx.sym.Convolution(data=pool1, kernel=(5,5), num_filter=50)
tanh2 = mx.sym.Activation(data=conv2, act_type="tanh")
pool2 = mx.sym.Pooling(data=tanh2, pool_type="max", kernel=(2,2), stride=(2,2))
# first fullc layer
flatten = mx.sym.Flatten(data=pool2)
fc1 = mx.symbol.FullyConnected(data=flatten, num_hidden=500)
tanh3 = mx.sym.Activation(data=fc1, act_type="tanh")
# second fullc
fc2 = mx.sym.FullyConnected(data=tanh3, num_hidden=10)
# softmax loss
lenet = mx.sym.SoftmaxOutput(data=fc2, name='softmax')
mx.viz.plot_network(symbol=lenet, shape=shape)
```



Training The Network



```
# @@@ AUTOTEST_OUTPUT_IGNORED_CELL
model = mx.model.FeedForward(
    ctx = mx.gpu(0),      # use GPU 0 for training, others are same as before
    symbol = lenet,
    num_epoch = 10,
    learning_rate = 0.1)
model.fit(
    X=train_iter,
    eval_data=val_iter,
    batch_end_callback = mx.callback.Speedometer(batch_size, 200)
)
assert model.score(val_iter) > 0.98, "Low validation accuracy."
```

aws

The Gluon API Specification

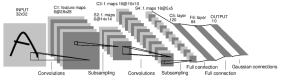
The Gluon API specification is an effort to improve speed, flexibility, and accessibility of deep learning technology for all developers, regardless of their deep learning framework of choice. The Gluon API offers a flexible interface that simplifies the process of prototyping, building, and training deep learning models without sacrificing training speed. It offers four distinct advantages:

- **Simple, Easy-to-Understand Code**: Gluon offers a full set of plug-and-play neural network building blocks, including predefined layers, optimizers, and initializers.
- **Flexible, Imperative Structure**: Gluon does not require the neural network model to be rigidly defined, but rather brings the training algorithm and model closer together to provide flexibility in the development process.
- **Dynamic Graphs**: Gluon enables developers to define neural network models that are dynamic, meaning they can be built on the fly, with any structure, and using any of Python's native control flow.
- **High Performance**: Gluon provides all of the above benefits without impacting the training speed that the underlying engine provides.

https://github.com/gluon-api/gluon-api/

aws

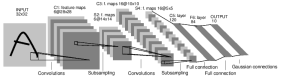
Define Network - gluon



```
num_fc = 512
net = gluon.nn.Sequential()
with net.name_scope():
    net.add(gluon.nn.Conv2D(channels=20, kernel_size=5, activation='relu'))
    net.add(gluon.nn.MaxPool2D(pool_size=2, strides=2))
    net.add(gluon.nn.Conv2D(channels=50, kernel_size=5, activation='relu'))
    net.add(gluon.nn.MaxPool2D(pool_size=2, strides=2))
    # The Flatten layer collapses all axis, except the first one, into one axis.
    net.add(gluon.nn.Flatten())
    net.add(gluon.nn.Dense(num_fc, activation="relu"))
    net.add(gluon.nn.Dense(num_outputs))
```

aws

Initialize and Train



```
net.collect_params().initialize(mx.init.Xavier(magnitude=2.24), ctx=ctx)
softmax_cross_entropy = gluon.loss.SoftmaxCrossEntropyLoss()
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate': .1})
```

```
epochs = 10
smoothing_constant = .01

for e in range(epochs):
    for i, (data, label) in enumerate(train_data):
        data = data.as_in_context(ctx)
        label = label.as_in_context(ctx)
        with autograd.record():
            output = net(data)
            loss = softmax_cross_entropy(output, label)
        loss.backward()
        trainer.step(data.shape[0])

        ##########################
        #  Keep a moving average of the losses
        ##########################
        curr_loss = nd.mean(loss).asscalar()
        moving_loss = (curr_loss if ((i == 0) and (e == 0))
                       else (1 - smoothing_constant) * moving_loss + (smoothing_constant) * curr_l
oss)

    test_accuracy = evaluate_accuracy(test_data, net)
    train_accuracy = evaluate_accuracy(train_data, net)
    print("Epoch %s. Loss: %s, Train_acc %s, Test_acc %s" % (e, moving_loss, train_accuracy, test_
accuracy))
```

aws

Learn More at: gluon.mxnet.io

Docs » Deep Learning - The Straight Dope View page source

Deep Learning - The Straight Dope

This repo contains an incremental sequence of notebooks designed to teach deep learning, Apache MXNet (incubating), and the gluon interface. Our goal is to leverage the strengths of Jupyter notebooks to present prose, graphics, equations, and code together in one place. If we're successful, the result will be a resource that could be simultaneously a book, course material, a prop for live tutorials, and a resource for plagiarising (with our blessing) useful code. To our knowledge there's no source out there that teaches either (1) the full breadth of concepts in modern deep learning or (2) interleaves an engaging textbook with runnable code. We'll find out by the end of this venture whether or not that void exists for a good reason.

Another unique aspect of this book is its authorship process. We are developing this resource fully in the public view and are making it available for free in its entirety. While the book has a few primary authors to set the tone and shape the content, we welcome contributions from the community and hope to coauthor chapters and entire sections with experts and community members. Already we've received contributions spanning typo corrections through full working examples.

aws

Demo Time

- Hand-written Digits
- Predicting with a pre-trained Network (resnet)

Thank you!

Constantin Gonzalez

glez@amazon.de

aws