intel Nervana™ AI Academy

INTRODUCTION TO CONVOLUTIONAL NEURAL NETWORKS

LAST TIME

Managed to get seemingly good results with basic network

98% Test Accuracy on MNIST:

- ReLU
- 3 hidden layers of depth 1200
- 15 epochs

98% for a minimal amount of training time seems pretty good!

What are we missing?

CONSIDERATIONS

MNIST has relatively clean images

Numbers are:

- Centered
- Approximately same size

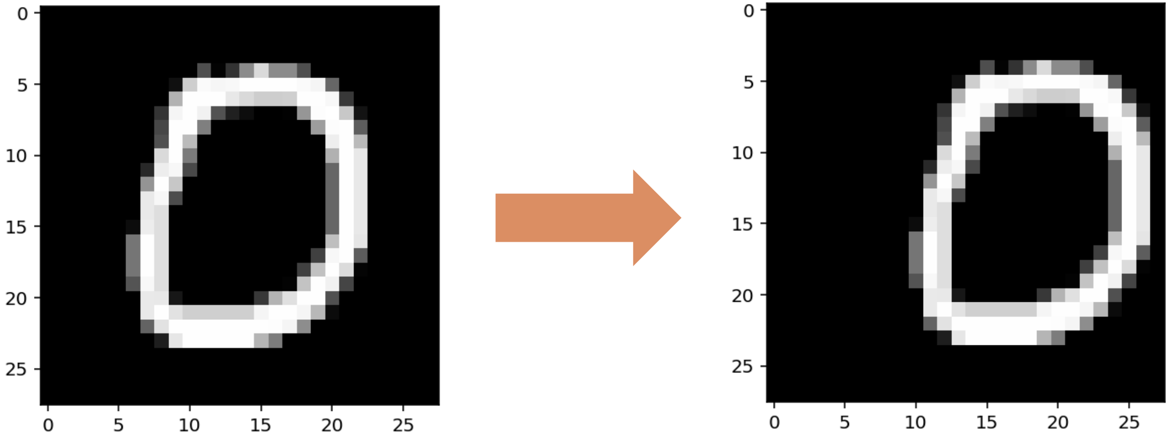
Image only has number in it – background is black

PROBLEM 1: TRANSLATION INVARIANCE

Each pixel is independent input

If we translate the input, the model breaks down

- We need to train (and test) models on translated data for more realistic scenario



PROBLEM 2: HUGE NUMBER OF PARAMETERS

1200x1200 matrix of weights = 1.4 *million* weights

- More weights → need more data
- More weights → hard to scale on hardware
  - Memory constraints!

What can we do?

KERNELS

WHAT ARE KERNELS?

Square grid of weights overlaid on image, centered on one pixel, and moved around the image

Each weight multiplied with pixel underneath it

Output for the centered pixel is $\sum_{p=1}^{P} W_p \cdot pixel_p$

Used for traditional image processing techniques:

- Blur
- Sharpen
- Edge detection
- Emboss

EXAMPLE: 3X3

Input

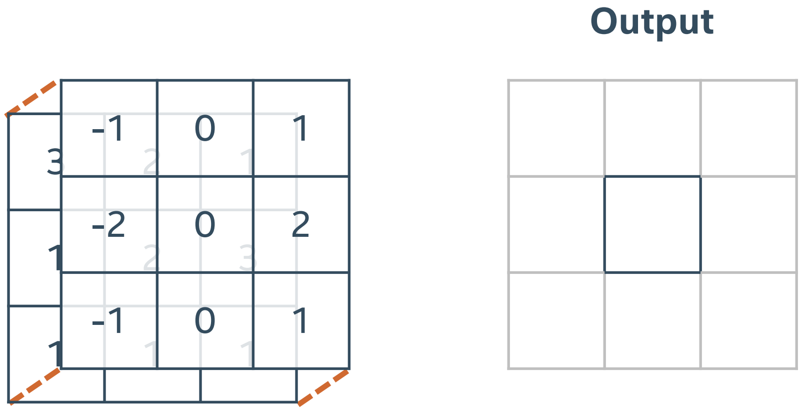
3	2	1
1	2	3
1	1	1

Kernel

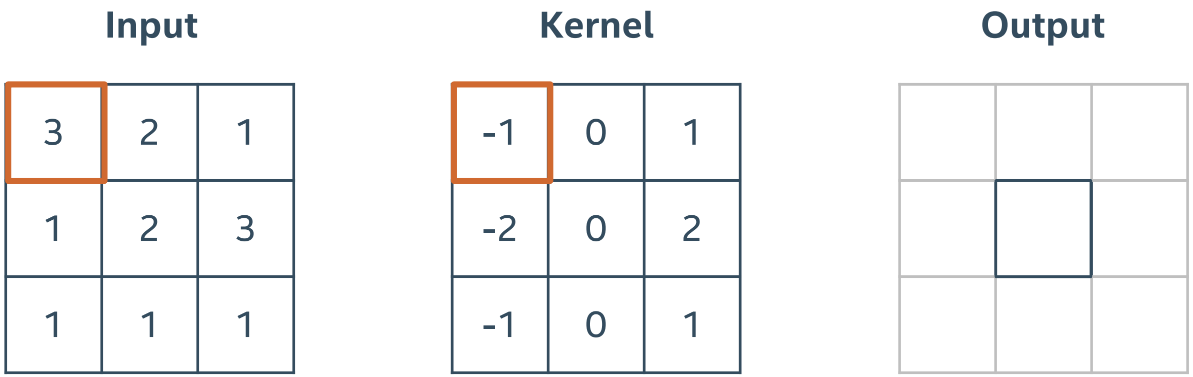
-1	0	1
-2	0	2
-1	0	1

Output

IMAGINE KERNEL IS STACKED ON TOP OF INPUT



EXAMPLE: 3X3



Input

3	2	1
1	2	3
1	1	1

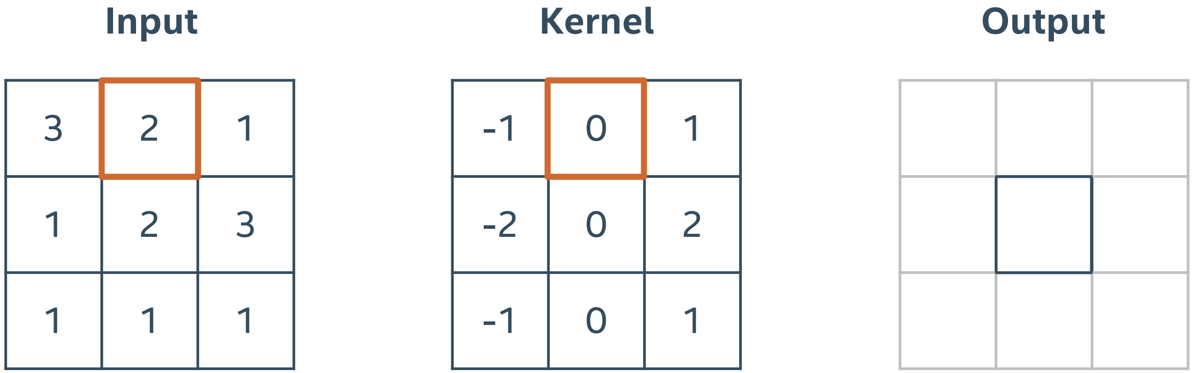
Kernel

-1	0	1
-2	0	2
-1	0	1

Output

$$= (3 \cdot -1)$$

EXAMPLE: 3X3



Input

3	2	1
1	2	3
1	1	1

Kernel

-1	0	1
-2	0	2
-1	0	1

$$= (3 \cdot -1) + (2 \cdot 0)$$

EXAMPLE: 3X3

Input

3	2	1 $=(3\cdot -1)+(2\cdot 0)+(1\cdot 1)$
1	2	3
1	1	1

Kernel

-1	0	1 $=(3\cdot -1)+(2\cdot 0)+(1\cdot 1)$
-2	0	2
-1	0	1

Output

$$= (3\cdot -1) + (2\cdot 0) + (1\cdot 1)$$

EXAMPLE: 3X3

Input

3	2	1
1	2	3
1	1	1

Kernel

-1	0	1
-2	0	2
-1	0	1

Output

$$= (3 \cdot -1) + (2 \cdot 0) + (1 \cdot 1) + (1 \cdot -2)$$

EXAMPLE: 3X3

Input

3	2	1
1	2	3
1	1	1

Kernel

-1	0	1
-2	0	2
-1	0	1

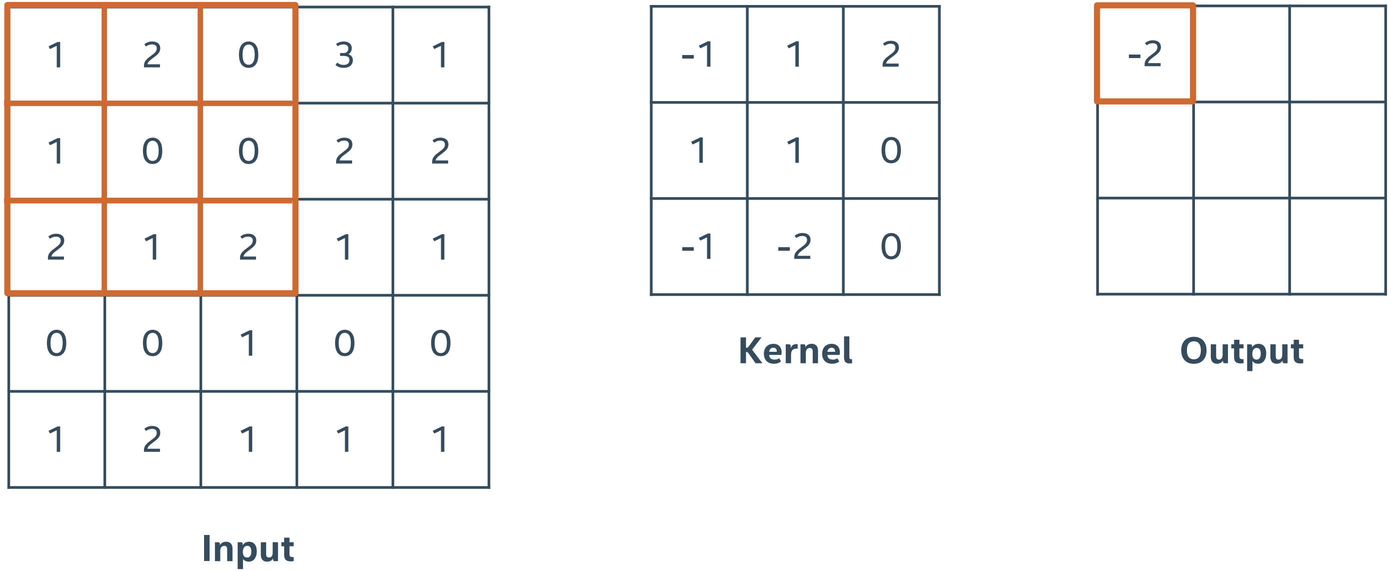
Output

	2	

$$= (3 \cdot -1) + (2 \cdot 0) + (1 \cdot 1) + (1 \cdot -2) + (2 \cdot 0) + (3 \cdot 2) + (1 \cdot -1)$$
$$+ (1 \cdot 0) + (1 \cdot 1)$$

$$= -3 + 1 - 2 + 6 - 1 + 1$$
$$= 2$$

HERE'S WHAT THE PROCESS LOOKS LIKE OVER A LARGER INPUT



1	2	0	3	1
1	0	0	2	2
2	1	2	1	1
0	0	1	0	0
1	2	1	1	1

Input

-1	1	2
1	1	0
-1	-2	0

Kernel

-2		

Output

INTERACTIVE KERNEL DEMONSTRATION

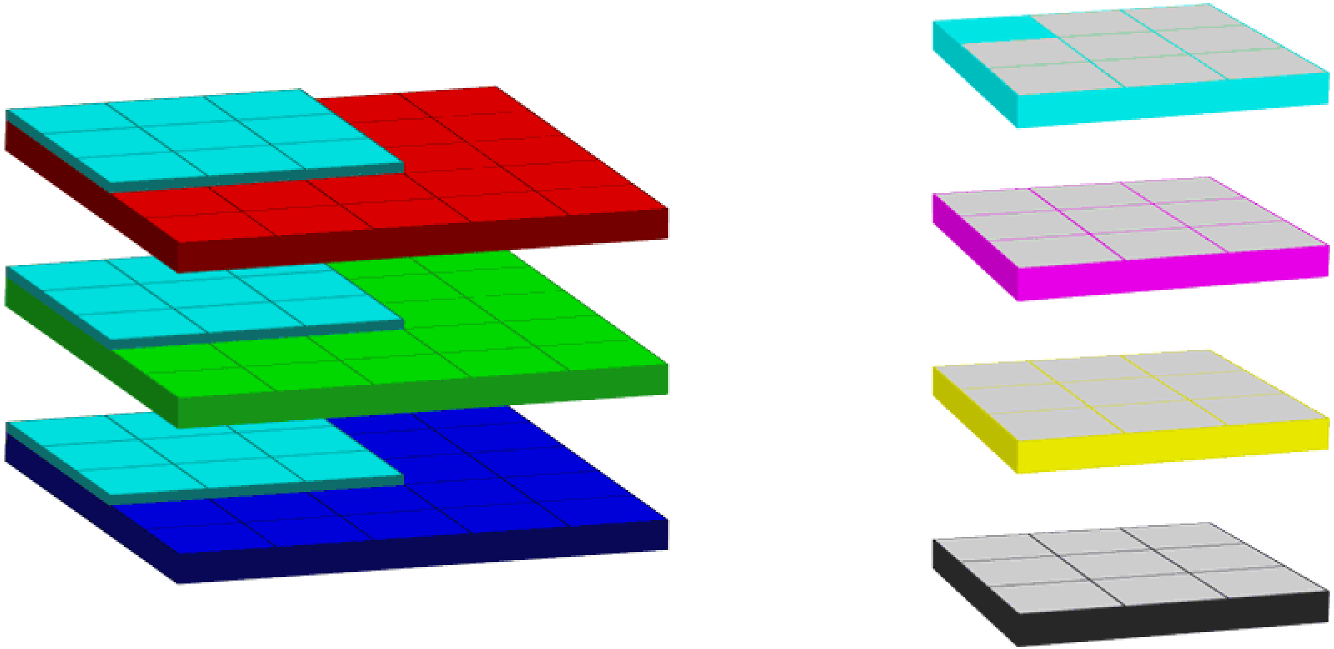
http://setosa.io/ev/image-kernels/

CONVOLUTIONAL NEURAL NETWORKS

CONVOLUTIONAL NEURAL NETWORKS

Idea: let neural network learn suitable kernels for task

CONVOLUTION OPERATION



CONVOLUTION SETTINGS

HEIGHT AND WIDTH

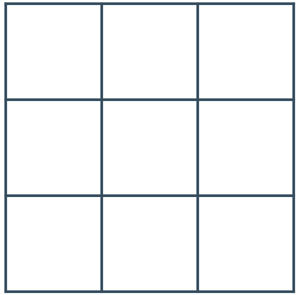
Number of pixels the kernel operates on

Both dimensions must be odd

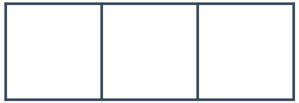
- B/c we need a reasonable center pixel

Kernel doesn’t have to be square

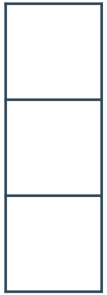
Height: 3, Width: 3



Height: 1, Width: 3



Height: 3, Width: 1



STRIDE

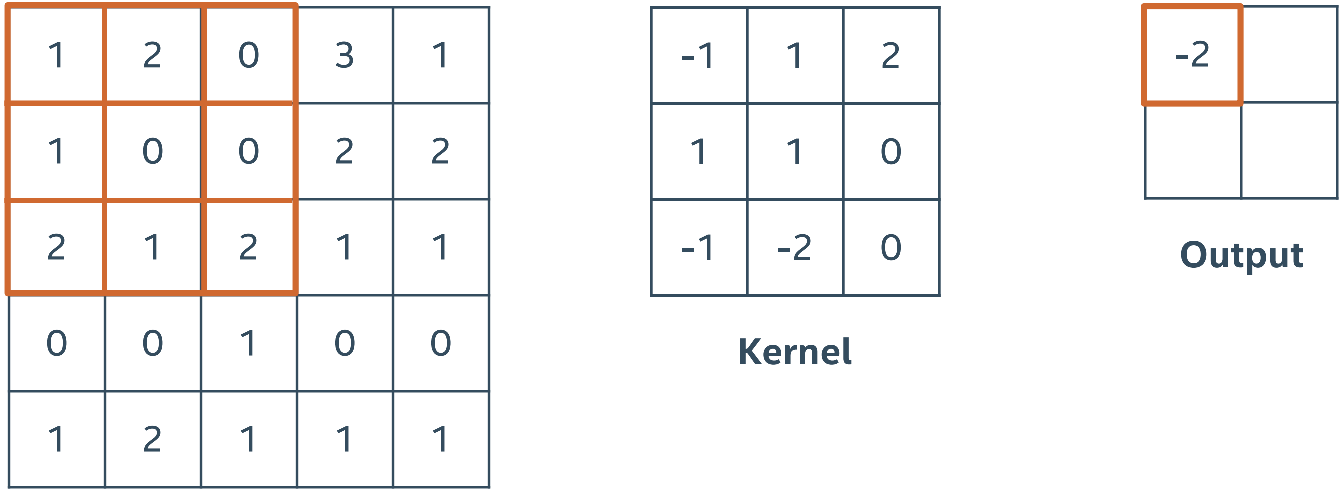
***Stride* is the step size from center to center**

Also has height/width component

- Generally height/width are the same

If greater than 1, will scale down the output dimensions

STRIDE 2 CONVOLUTION



1	2	0	3	1
1	0	0	2	2
2	1	2	1	1
0	0	1	0	0
1	2	1	1	1

Input

-1	1	2
1	1	0
-1	-2	0

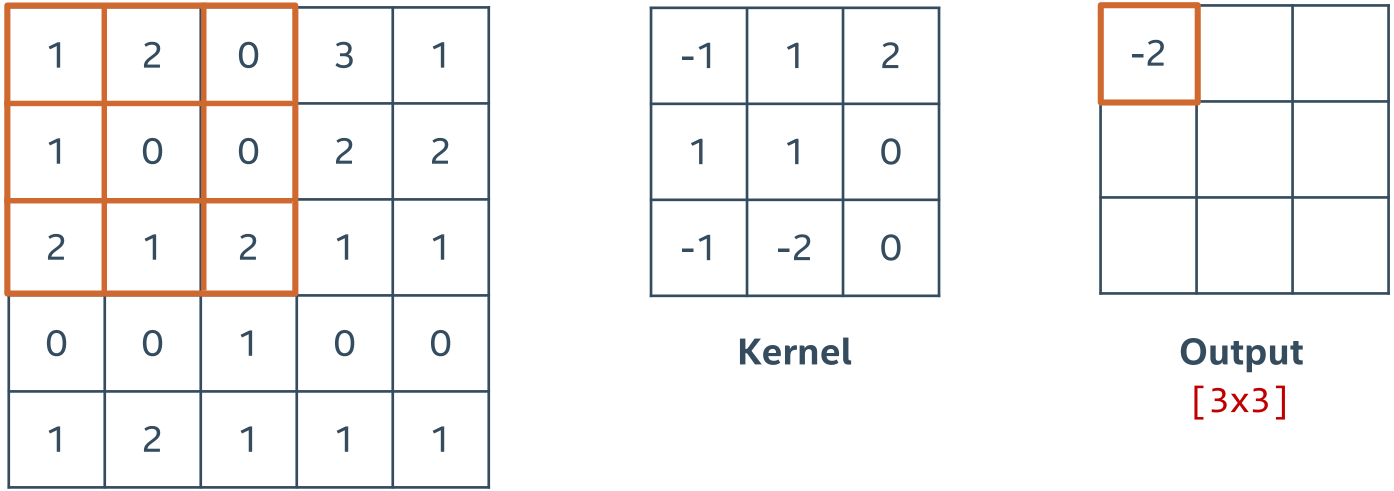
Kernel

-2	

Output

PADDING

Notice: the standard convolution down samples input



1	2	0	3	1
1	0	0	2	2
2	1	2	1	1
0	0	1	0	0
1	2	1	1	1

Input
[5x5]

-1	1	2
1	1	0
-1	-2	0

Kernel

-2		

Output
[3x3]

PADDING

Padding adds pseudo-pixels off-the-edge of the input

- Padding is all zero values

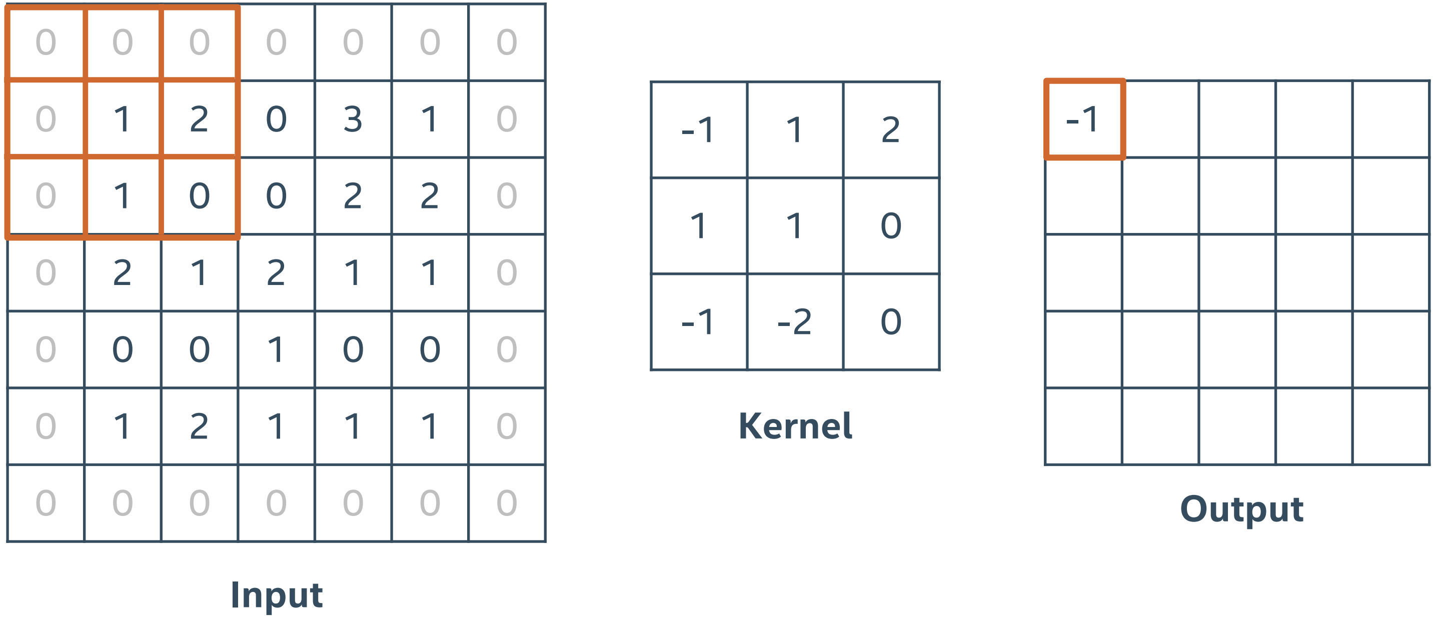
One unit of padding means one ring of zero pixels around the input

Amount of padding is usually either:

- No padding
  - TensorFlow calls this ‘VALID’ (i.e., use only *valid* input size)
- Enough to offset the kernel size and output the same dimensions
  - TensorFlow calls this ‘SAME’ (i.e., same input/output size)

3x3 kernel → padding 1
5x5 kernel → padding 2
7x7 kernel → padding 3

PADDING: 1 ('SAME')



0	0	0	0	0	0	0
0	1	2	0	3	1	0
0	1	0	0	2	2	0
0	2	1	2	1	1	0
0	0	0	1	0	0	0
0	1	2	1	1	1	0
0	0	0	0	0	0	0

Input

-1	1	2
1	1	0
-1	-2	0

Kernel

-1				

Output

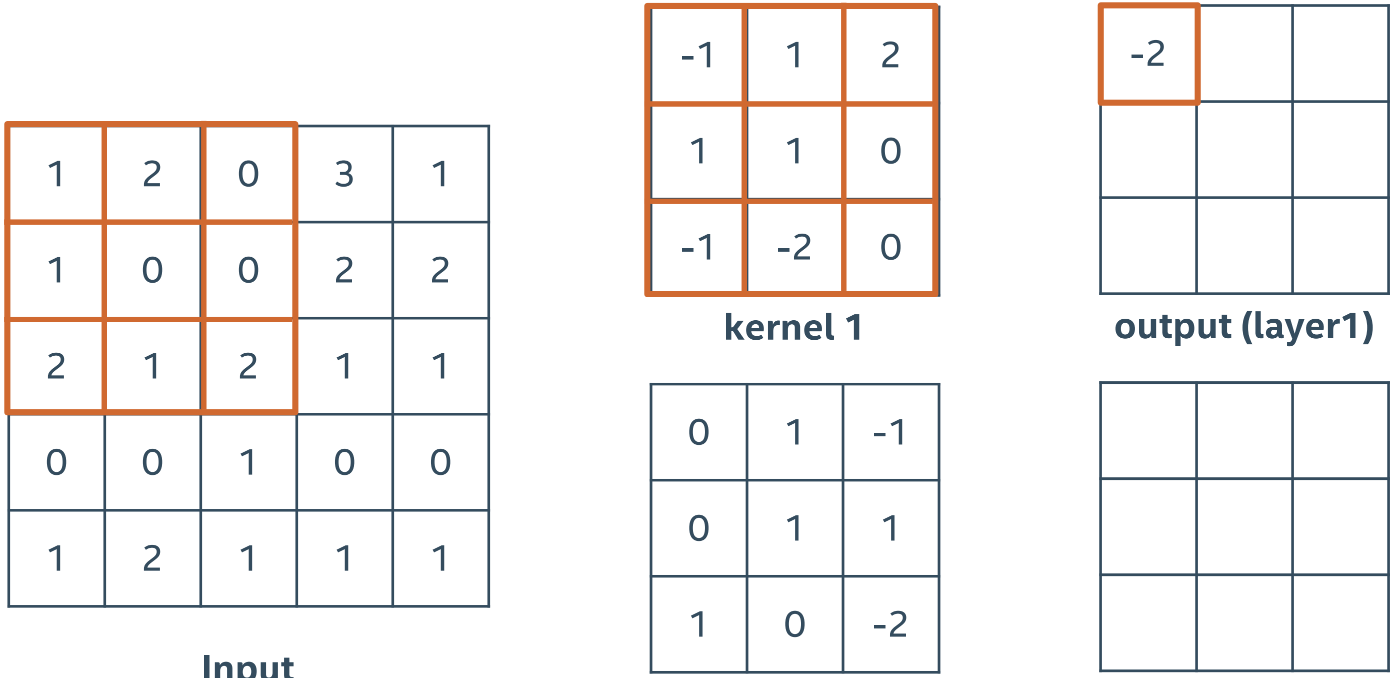
DEPTH—NUMBER OF OUTPUT CHANNELS

Channels: multiple numbers (colors) associated with same pixel

- 3-color RGB → 3 channels
- 4-color CMYK → 4 channels

Number of separate kernels needed in a layer

OUTPUT CHANNELS: 2



Input

1	2	0	3	1
1	0	0	2	2
2	1	2	1	1
0	0	1	0	0
1	2	1	1	1

kernel 1

-1	1	2
1	1	0
-1	-2	0

kernel 2

0	1	-1
0	1	1
1	0	-2

output (layer1)

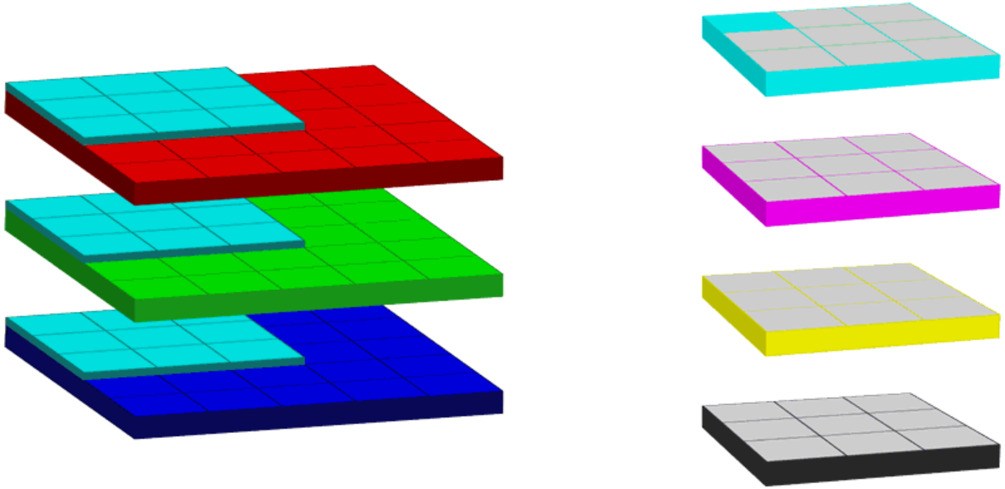
-2		

output (layer2)

INPUT DEPTH

Each kernel has the same depth as the number of input channels

Each input on each channel has a single weight associated with it



CONVOLUTION IN TENSORFLOW

`tf.nn.conv2d(input, filter, strides, padding)`

`input`: 4d tensor [batch_size, height, width, channels]

`filter`: 4d: [height, width, channels_in, channels_out]

- Generally a Variable

`strides`: 4d: [1, vert_stride, horiz_strid, 1]

- First and last dimensions must be 1 (helps with under-the-hood math)

`padding`: string: ‘SAME’ or ‘VALID’

POOLING

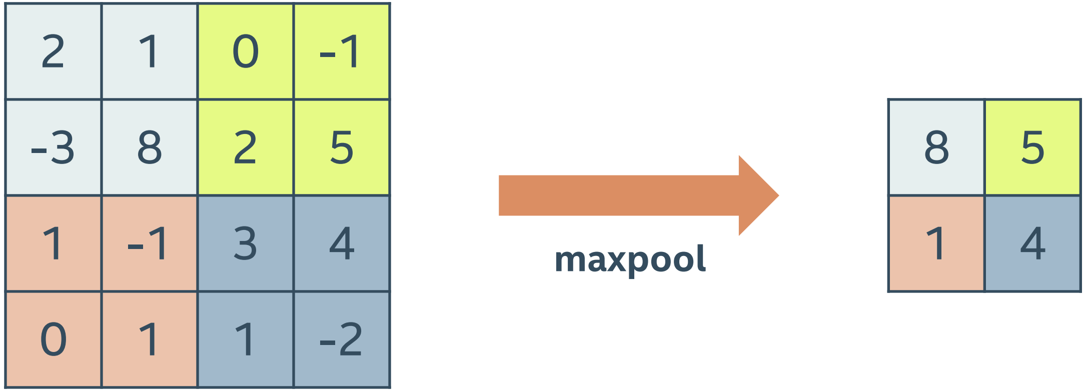
POOLING

Idea: reduce neighboring pixels

Reduce dimensions of inputs (height and width)

No parameters!

MAX POOLING

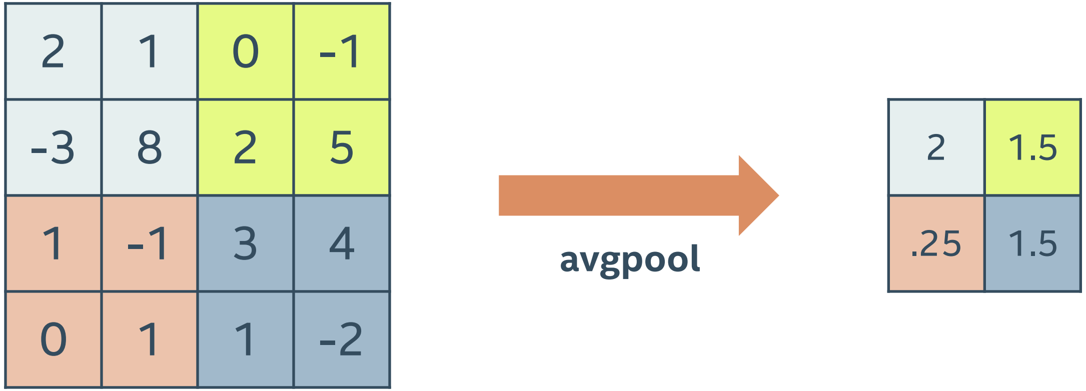


2	1	0	-1
-3	8	2	5
1	-1	3	4
0	1	1	-2

maxpool

8	5
1	4

AVERAGE POOLING

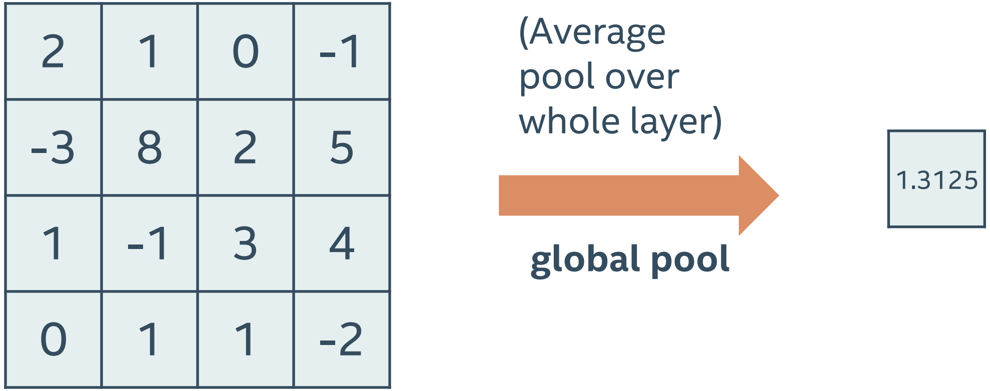


2	1	0	-1
-3	8	2	5
1	-1	3	4
0	1	1	-2

avgpool

2	1.5
.25	1.5

GLOBAL POOLING



2	1	0	-1
-3	8	2	5
1	-1	3	4
0	1	1	-2

(Average pool over whole layer)

global pool

ADDITIONAL CONVOLUTION OPERATION RESOURCE

Andrej Karpathy's convolutional network website

Created for Stanford's CS231n course

http://cs231n.github.io/convolutional-networks/

LENET-5

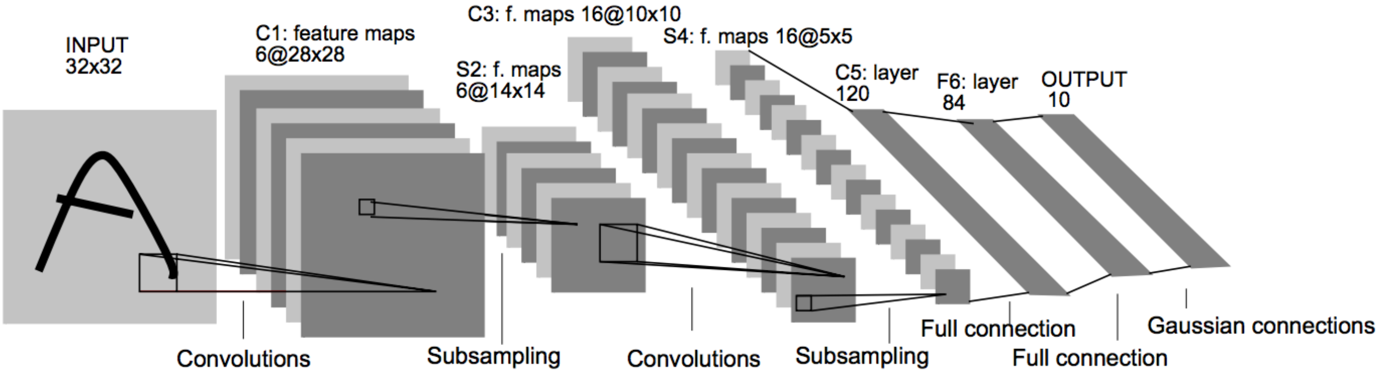
LENET

Created by Yann LeCun in the 1990s

Used on the MNIST dataset

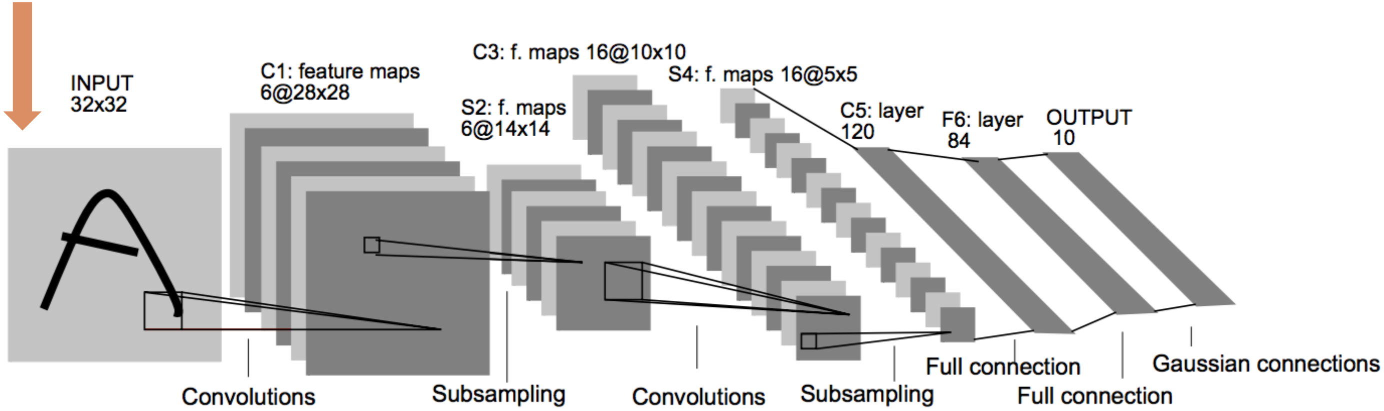
Idea: Use convolutions to efficiently learn features on image data

NETWORK DIAGRAM

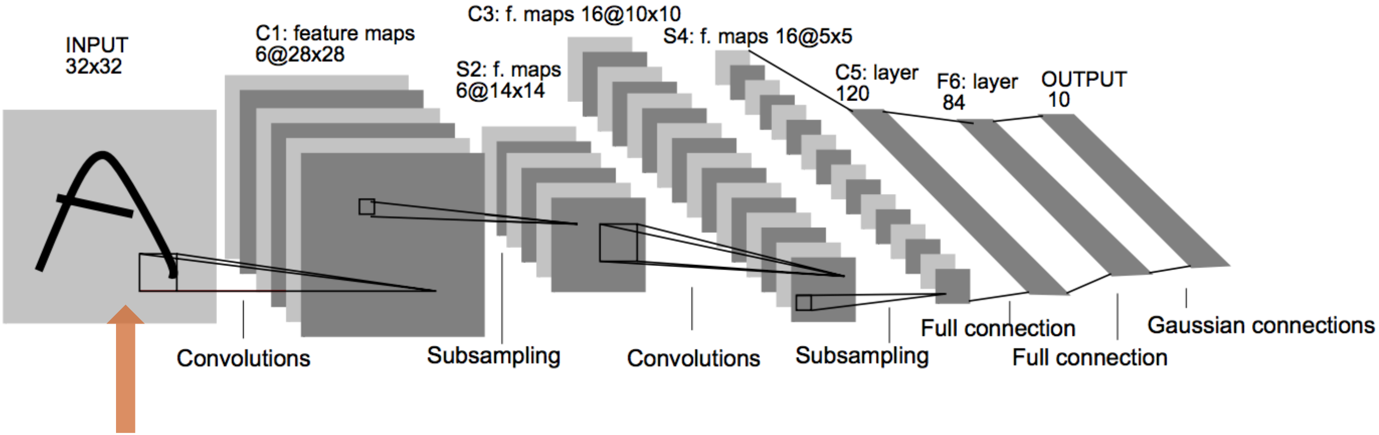


NETWORK DIAGRAM

Input: 28x28, with 2 pixels of padding (on all sides)



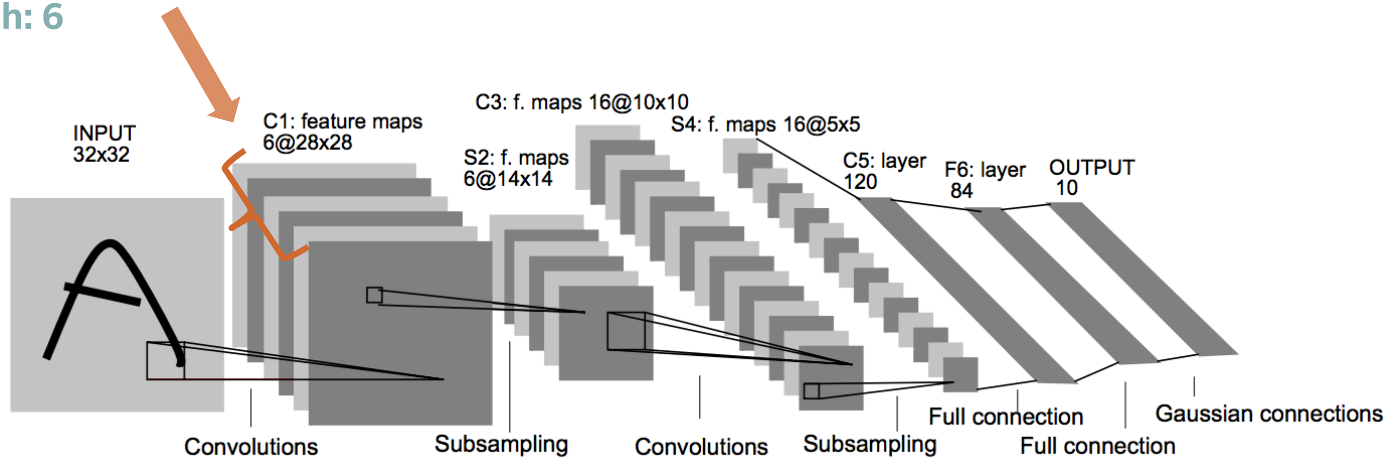
NETWORK DIAGRAM



Convolution size: 5x5
(not labeled, but in paper)

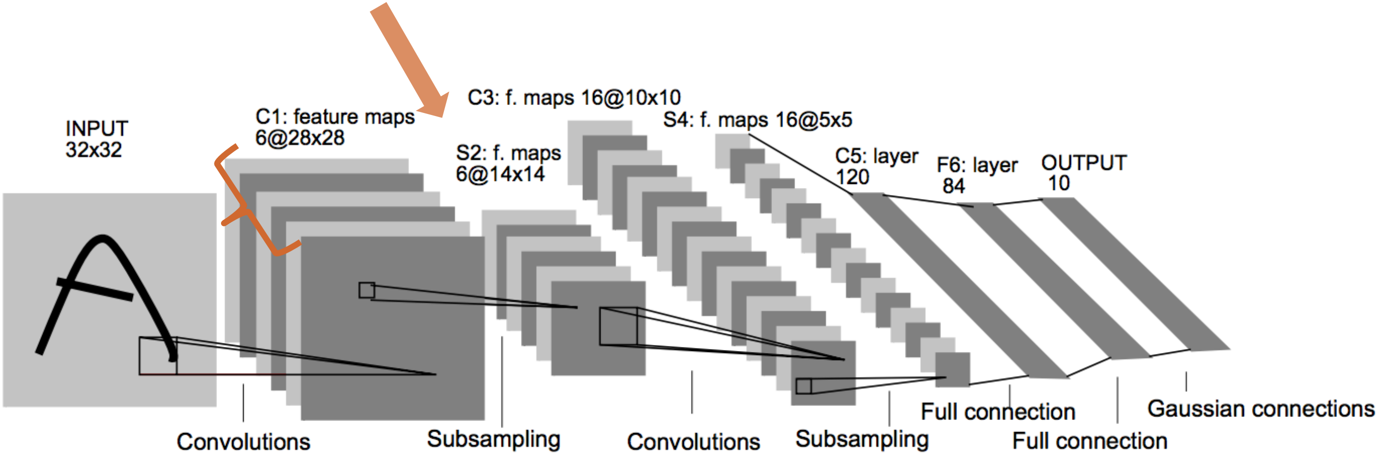
NETWORK DIAGRAM

First convolutional layer depth: 6



NETWORK DIAGRAM

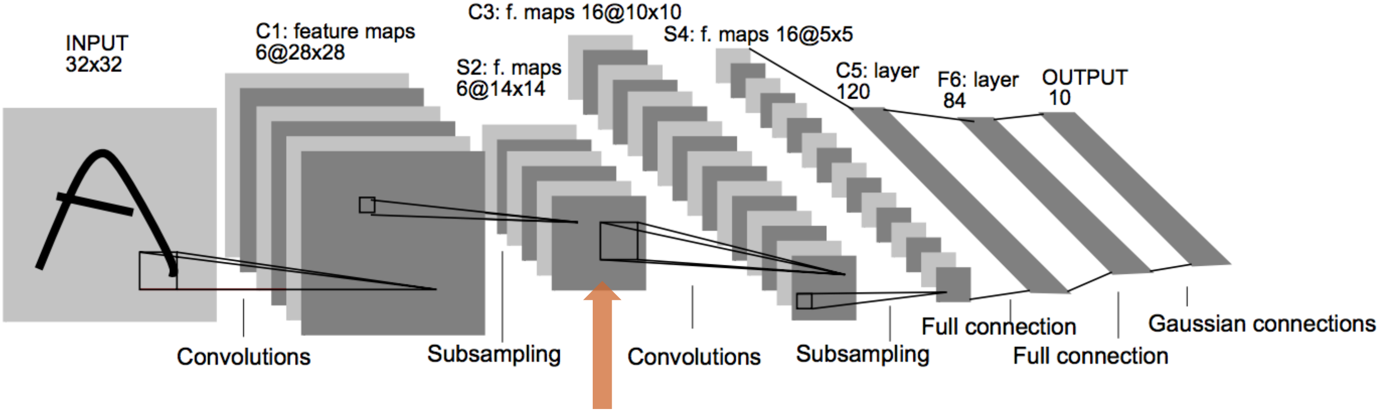
Pooling: 2x2, stride 2



Note: in the paper, the model uses a more complex parameter based pooling operation.

Max/average pooling turns out to work better in practice

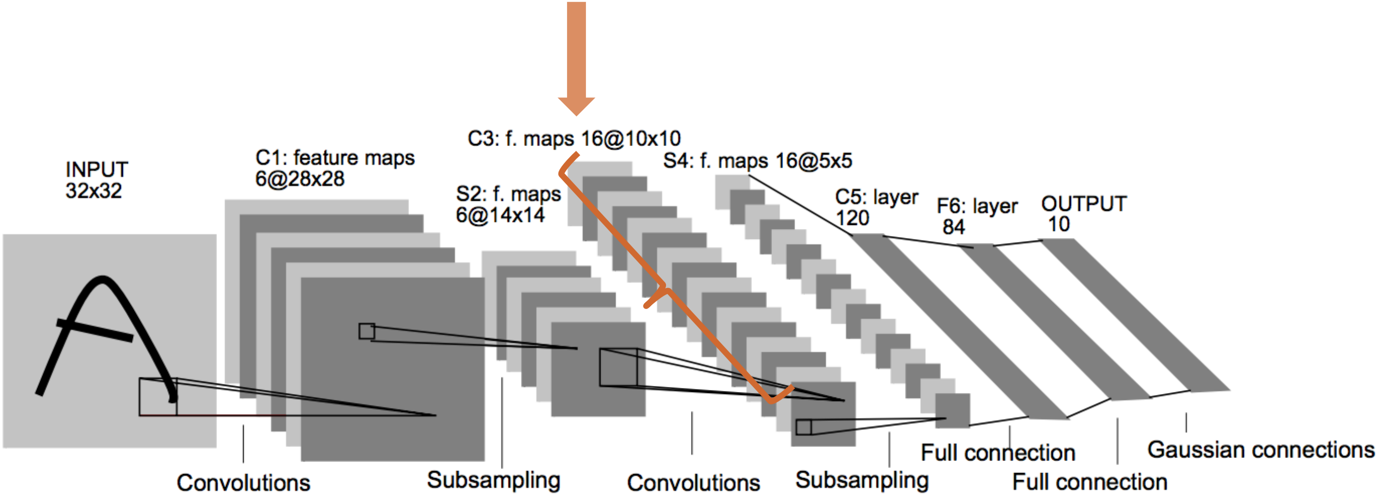
NETWORK DIAGRAM



Convolution size: 5x5

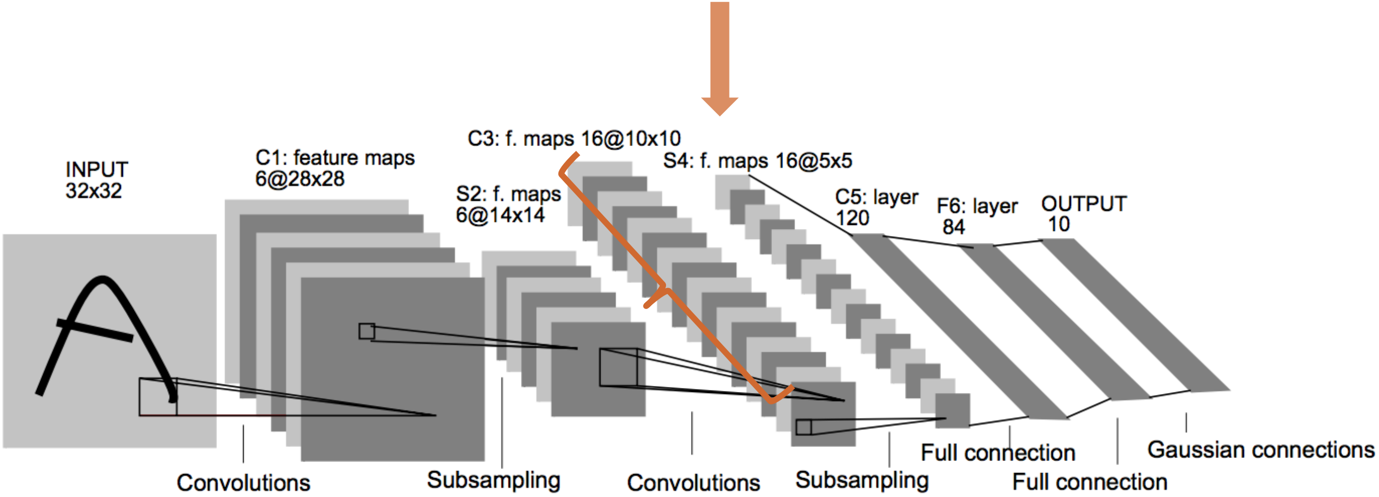
NETWORK DIAGRAM

Second convolutional layer depth: 16



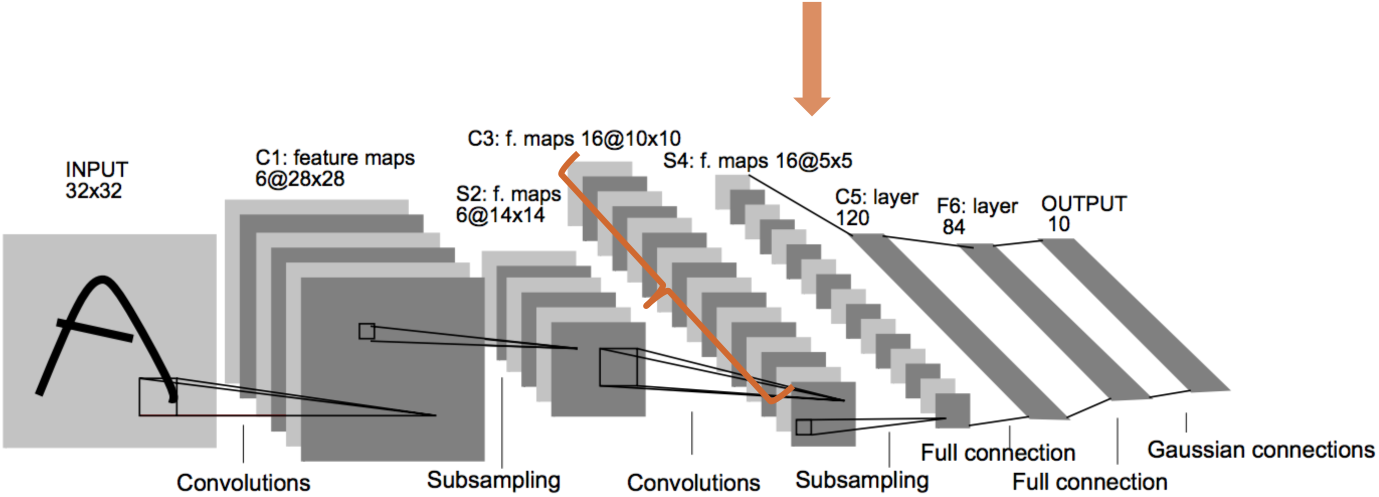
NETWORK DIAGRAM

Pooling: 2x2, stride 2



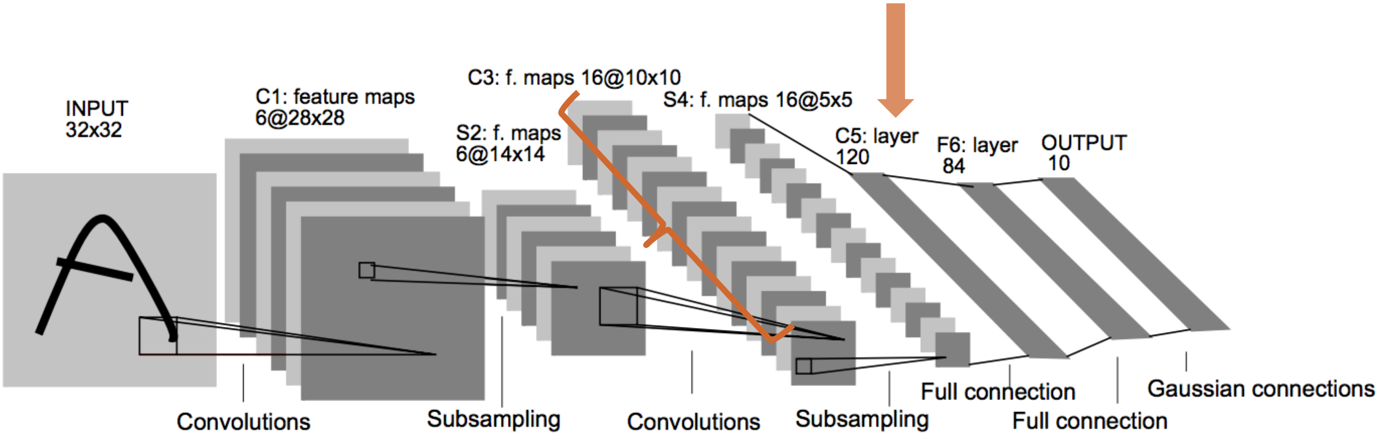
NETWORK DIAGRAM

Flatten from
5x5x16 to 400x1



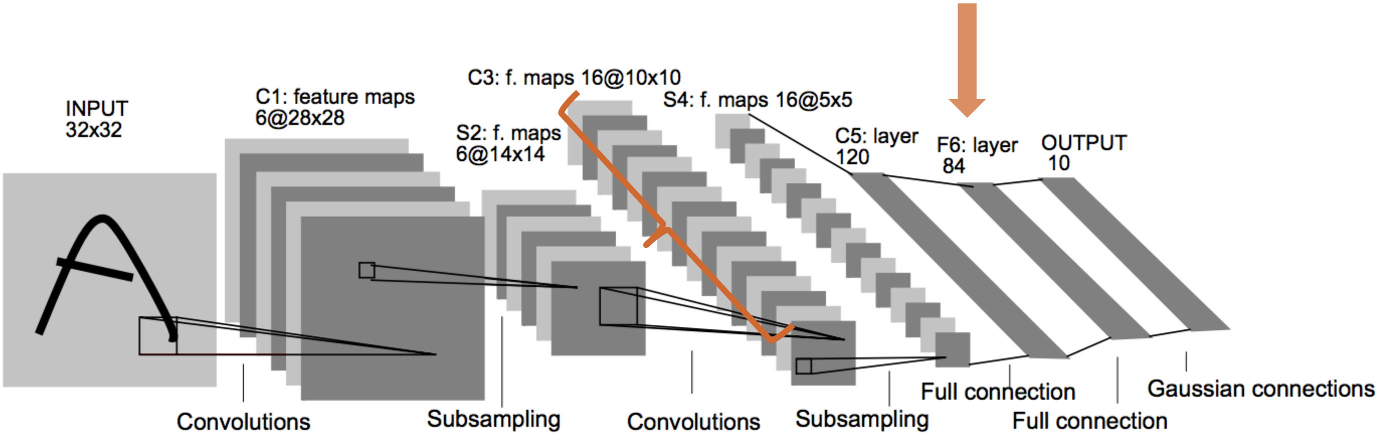
NETWORK DIAGRAM

Fully connected layer:
from 400 to 120



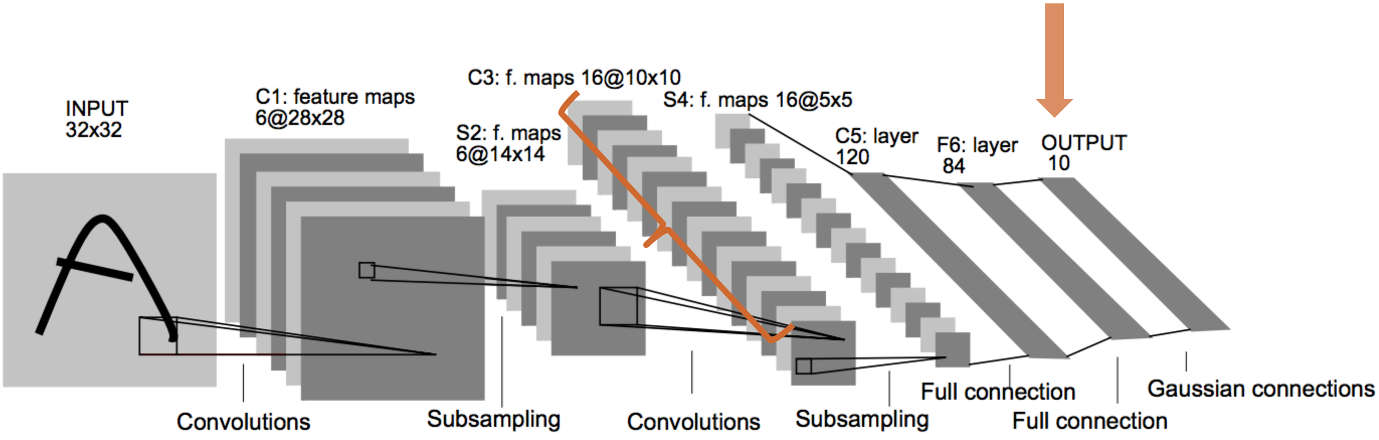
NETWORK DIAGRAM

Fully connected layer:
from 120 to 84

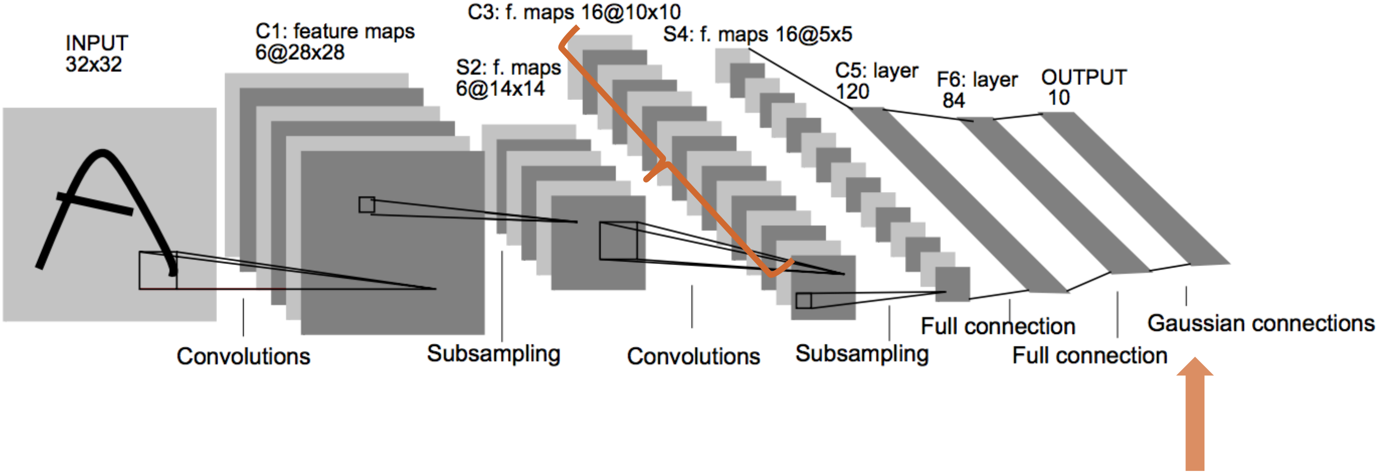


NETWORK DIAGRAM

Fully connected layer:
from 84 to 10



NETWORK DIAGRAM



Softmax

TABLE DESCRIPTION OF LENET-5

Layer Name	Parameters
1. Convolution	5x5, stride 1, padding 2 (‘SAME’)
2. Max Pool	2x2, stride 2
3. Convolution	5x5, stride 1, padding 2 (‘SAME’)
4. Max Pool	2x2, stride 2
5. Fully connected (ReLU)	Depth: 120
6. Fully connected (ReLU)	Depth: 84
7. Output (fully connected ReLU)	Depth: 10

COUNT PARAMETERS

```
Conv1: 1*6*5*5 + 6   =   156
Pool2:                     0
Conv3: 6*16*5*5 + 16 =  2416
Pool4:                     0
FC1: 400*120 + 120   = 48120
FC2: 120*84  + 84    = 10164
FC3: 84*10 + 10      =   850
Total:               = 61706
```

Less than a single FC layer with [1200x1200] weights!

XAVIER (AND HE) INITIALIZATION

XAVIER INITIALIZATION

Want to initialize our weights such that the variance of the output of our *activation* is 1

Xavier Glorot and Bengio derived the following initialization scheme for activations with mean zero inputs:

$$W = TruncNormal(0.0, \sqrt{\frac{2}{n_{in} + n_{out}}})$$

RECOMMENDATION FOR RELUS

He et al. derived an initialization scheme specifically for ReLUs (which don't have a zero mean)

$$W = TruncNorm(0.0, \sqrt{\frac{2}{n_{in}}})$$

SIMPLIFIES THE TRAINING PROCEDURE.

Allows us to train “end-to-end”, without pre-training

Less time spent dealing with exploding gradients

No longer have to hand-tweak everything

intel® Nervana™ AI Academy