Introduction to Computer Vision

14. Deep Learning

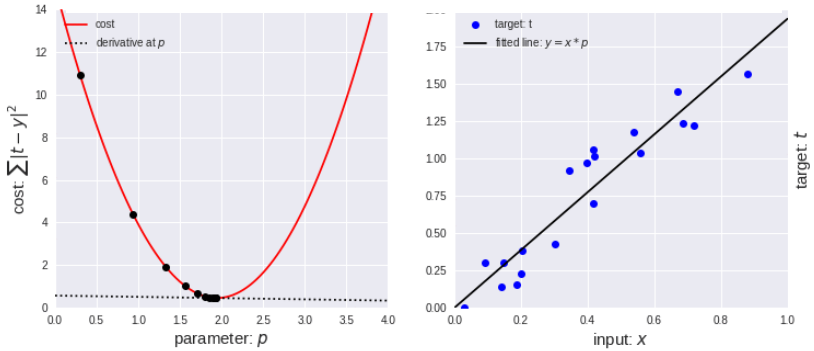
UCLA – CS 188 – Fall 2019

Fabien Scalzo, Ph.D.

Week 1			26-Sep	**Introduction**
Week 2	1-Oct	**Basic Image Processing**	3-Oct	**Feature Extraction and Classification**
Week 3	8-Oct	**Feature Tracking/Optical Flow**	10-Oct	**SVD, 2D camera model, projective plane**
Week 4	15-Oct	**2D Image transformations, RANSAC**	17-Oct	**Euclidean geometry, rigid body motion**
Week 5	22-Oct	**Epipolar Geometry**	24-Oct	**3D Cameras and processing**
Week 6	29-Oct	**Midterm**	31-Oct	**Statistical decision theory/Pattern Recognition**
Week 7	5-Nov	**Learning from data**	7-Nov	**Neural Networks**
Week 8	12-Nov	**Deep Learning**	14-Nov	**Deep Learning**
Week 9	19-Nov	**Deep Learning (Large Scale and Object Detection)**	21-Nov	**Advanced Deep Learning**
Week 10	26-Nov	**Guest Lecture (Nikhil Naik)**	28-Nov	
Week 11	3-Dec	**Applications**	5-Dec	**Recap**
Week 12	10-Dec		12-Dec	**Final**

- Batch-normalization
- Evaluating Deep Learning models
- Transposed Convolution (i.e. ~~Deconv~~)
- MNIST with Deep Learning
- Beyond Deep Learning

Batch, Epoch, and Iteration

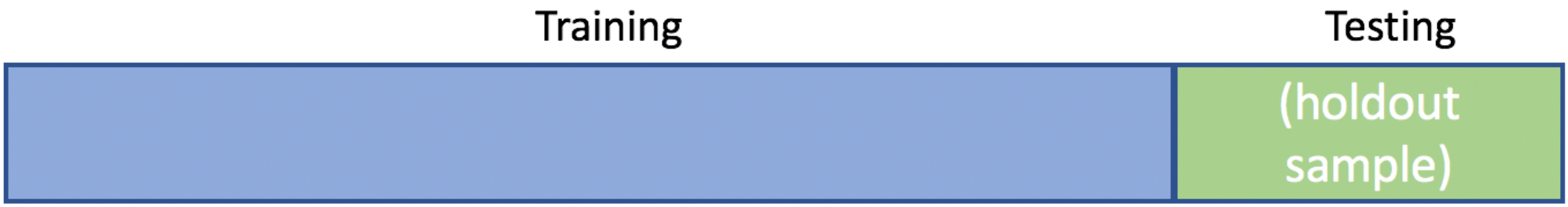


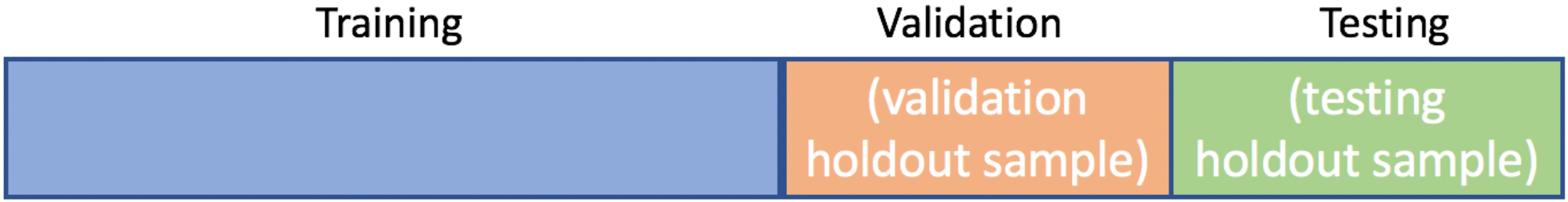
epoch: equivalent to the forward and backward processing of the entire training set

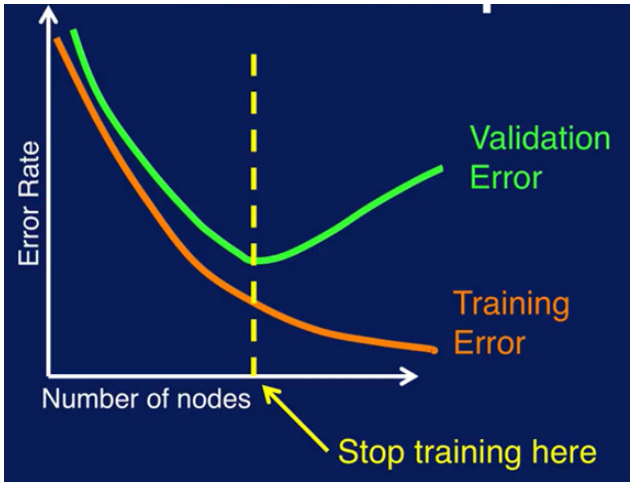
Batch: subset of training samples
Batch size: # of training samples in 1 batch

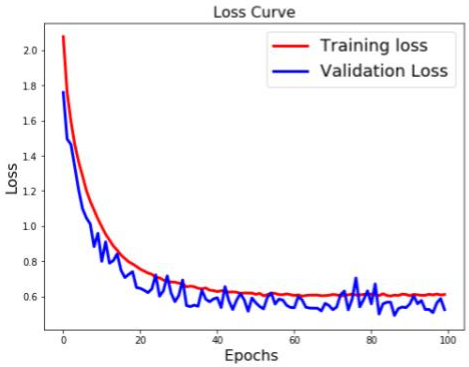
Iteration: # of batches to complete 1 epoch
(or # of passes to complete 1 epoch.
1 pass = 1 forward + 1 backward pass)

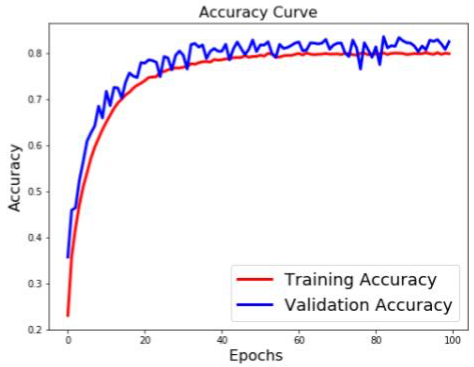
Given a dataset of 2,000 examples divided into batches of 500 then it will take 4 iterations to complete 1 epoch.


Training
Testing
(holdout sample)

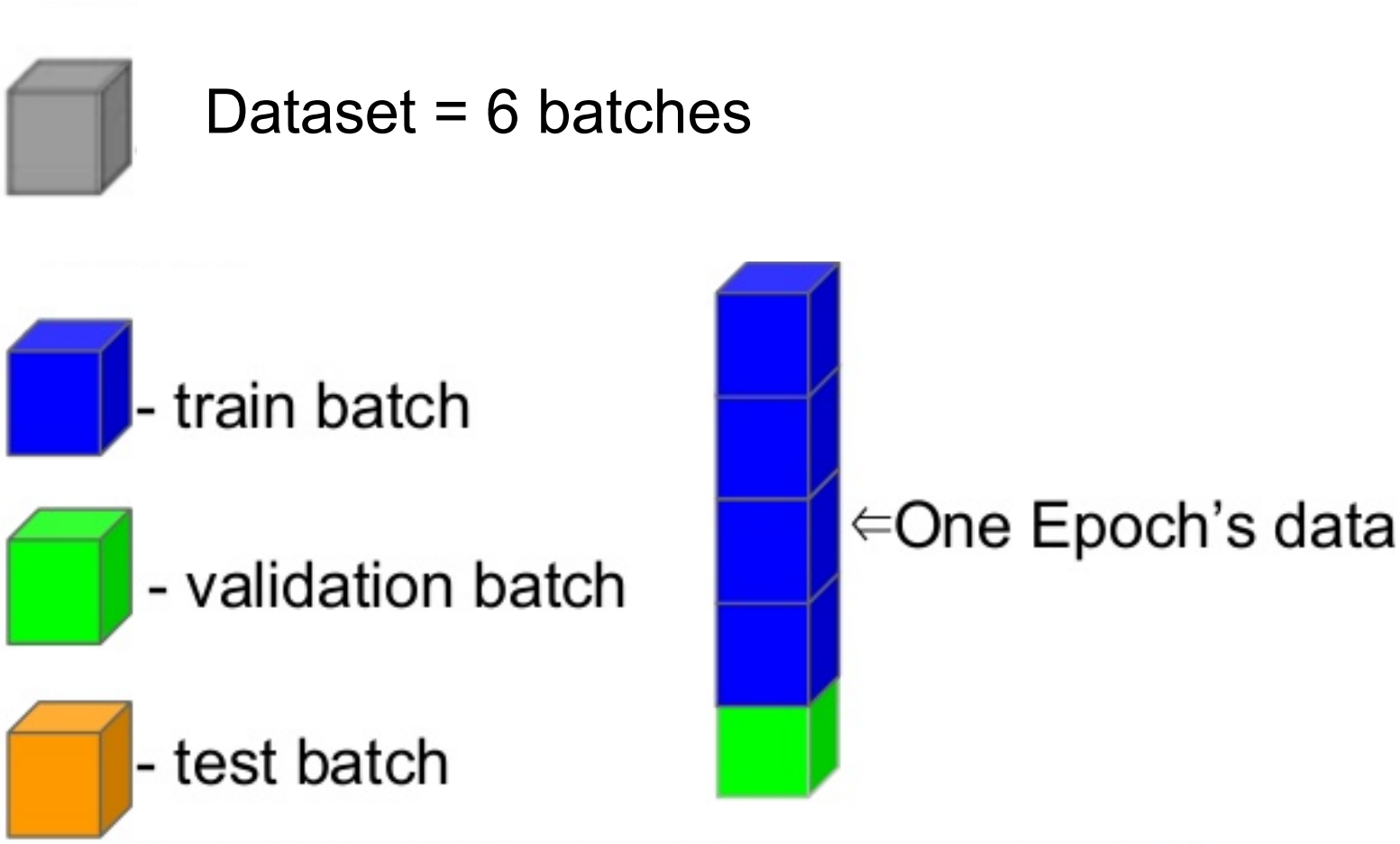

Training
Validation
Testing
(validation holdout sample)
(testing holdout sample)


Error Rate
Validation Error
Training Error
Number of nodes
Stop training here


Loss Curve
Training loss
Validation Loss
Loss
Epochs


Accuracy Curve
Training Accuracy
Validation Accuracy
Accuracy
Epochs

Batch, Epoch, and Iteration



Batch normalization (BN)

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

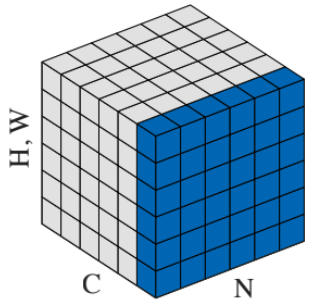
$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

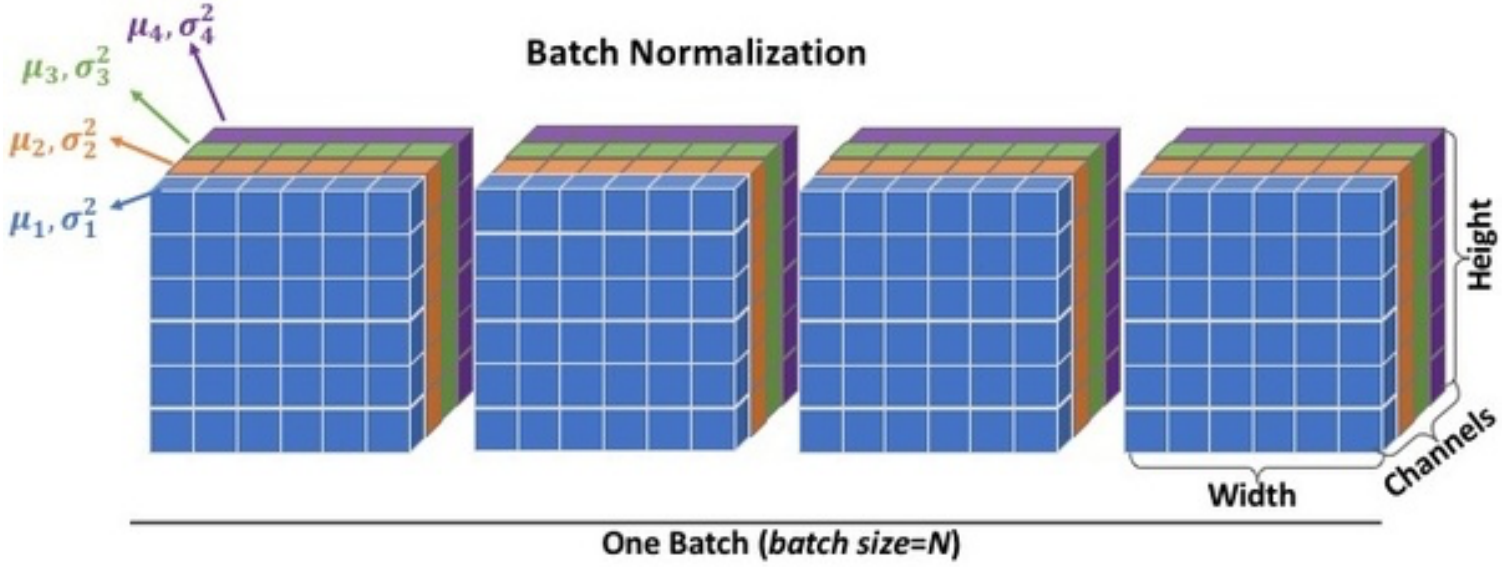
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Batch normalization (BN)





Group Normalization

Yuxin Wu Kaiming He

Facebook AI Research (FAIR)

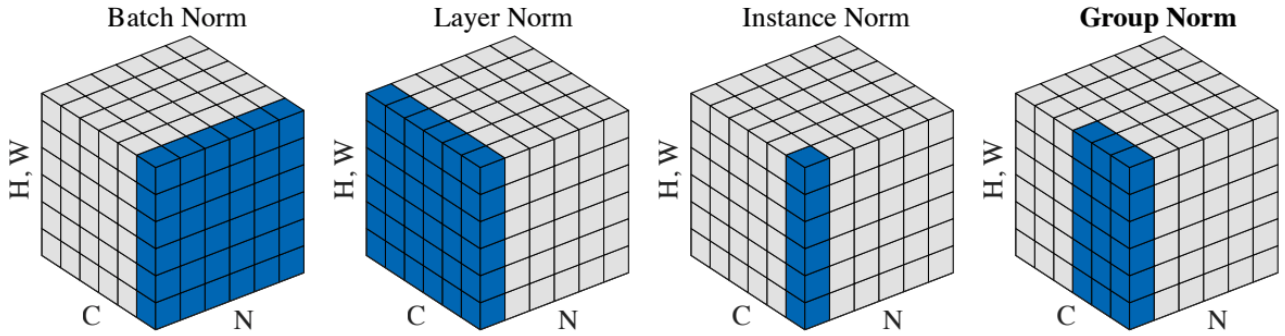
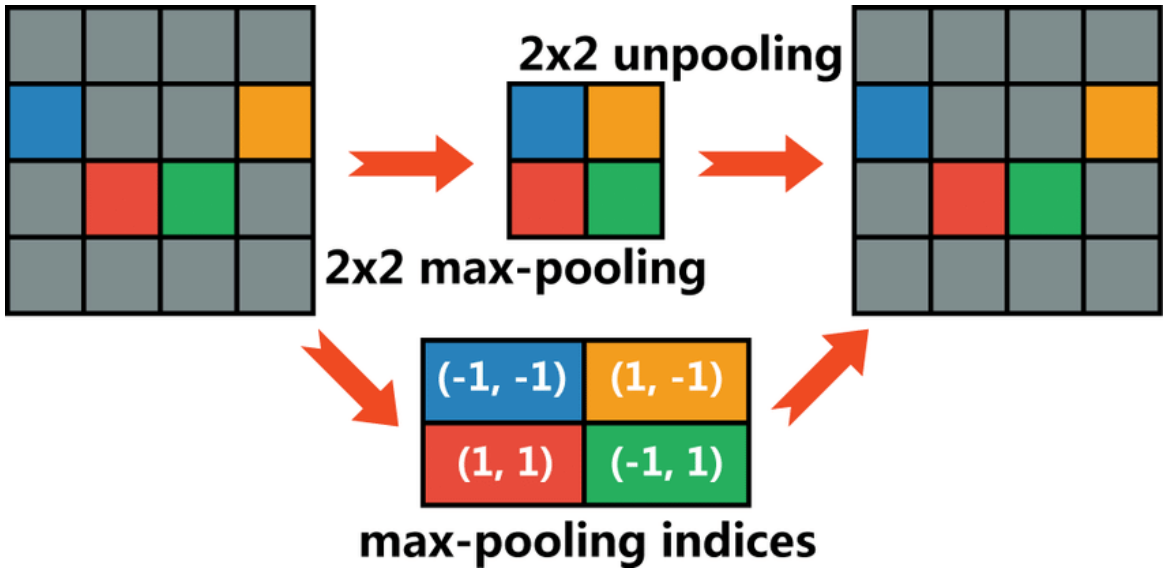


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Max-Unpooling



Transposed Convolution (i.e. ~~Deconv~~)

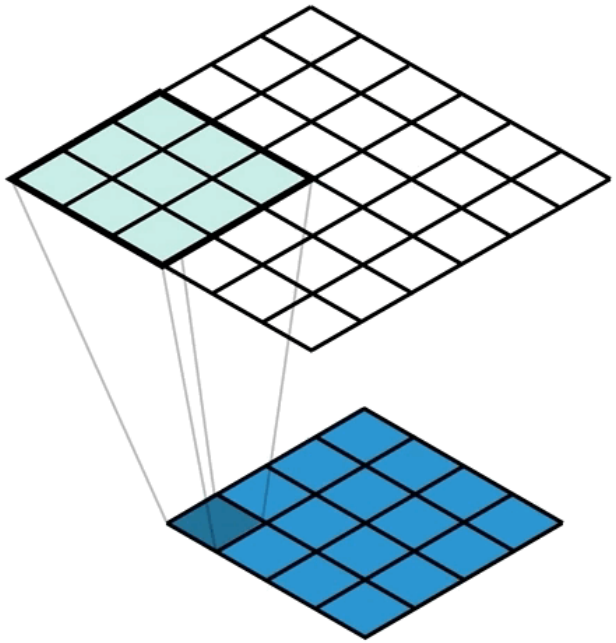
Ground Truth

¼ Sized Input

Bicubic

Super Resolution Network

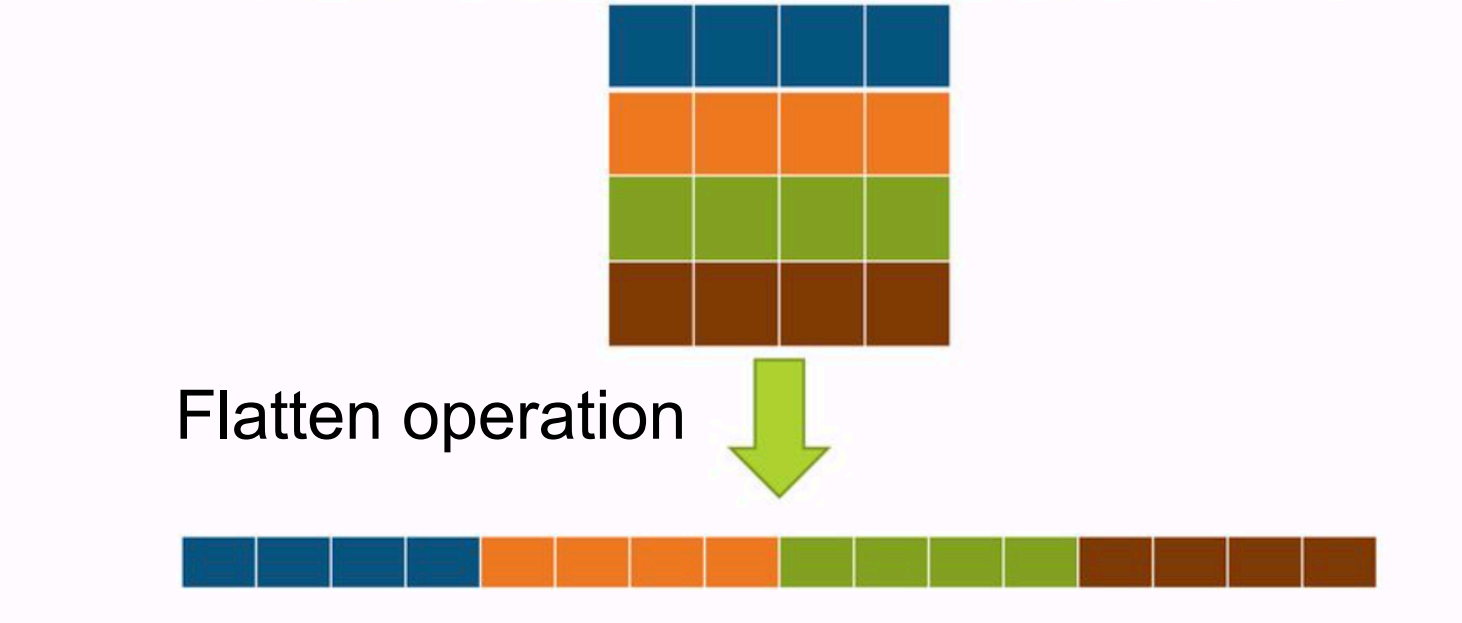
Transposed Convolution (i.e. ~~Deconv~~)



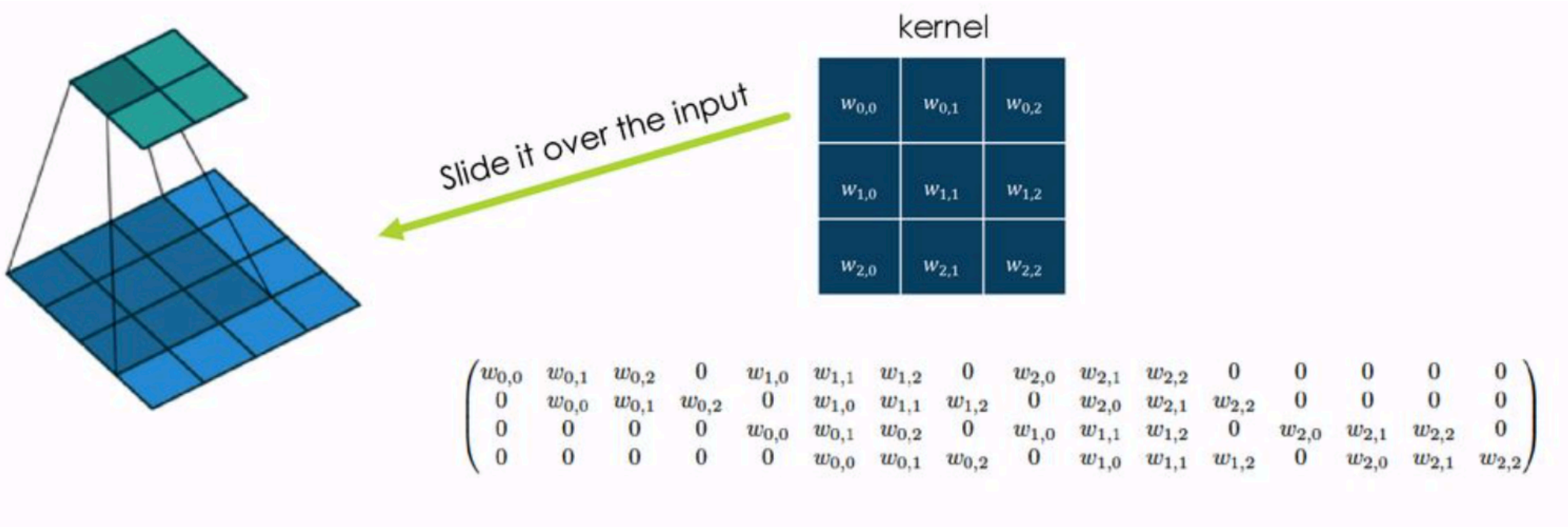
Convolution as matrix multiplication

- Convolution can be expressed as a matrix operation
- The convolution matrix **"C"** is inferred from the kernel
- The output is obtained by multiplying the "C" with the **flattened** input column vector and then reshaped

Convolution as matrix multiplication



Convolution as matrix multiplication



Convolution as matrix multiplication

$$output = C.i^T$$

Where i is the flattened input feature map.

From the previous two slides:

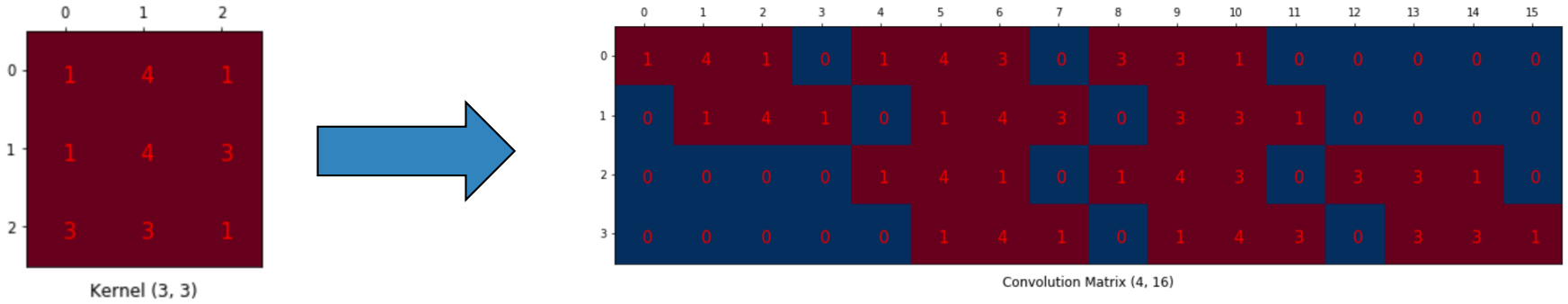
Assume the convolution on $i = 4\ with\ k = 3, s = 1$:

We get the output as 4x1 vector → **Reshape the output**

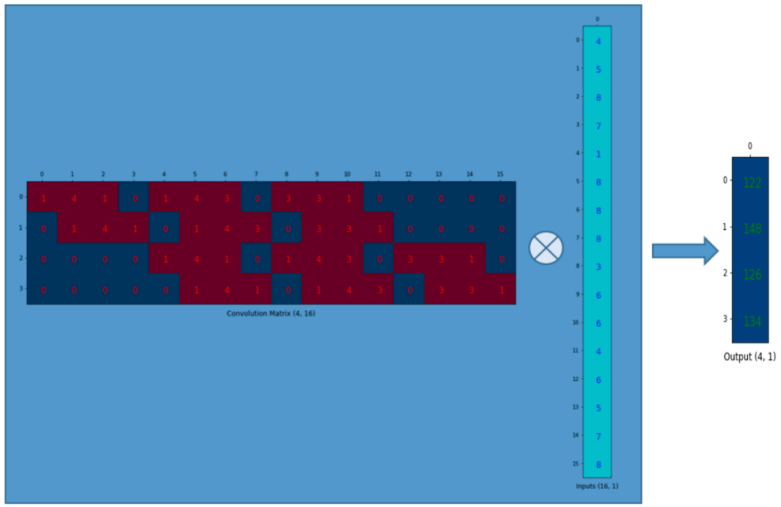
Transposed convolution as matrix multiplication

- Output = $C^T i'$
- C^T is 16x4 matrix
- i' is 4x1 column vector
- Output = 16x1 vector → **Reshape it to the shape of input**

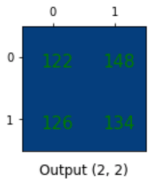
Transposed convolution as matrix multiplication



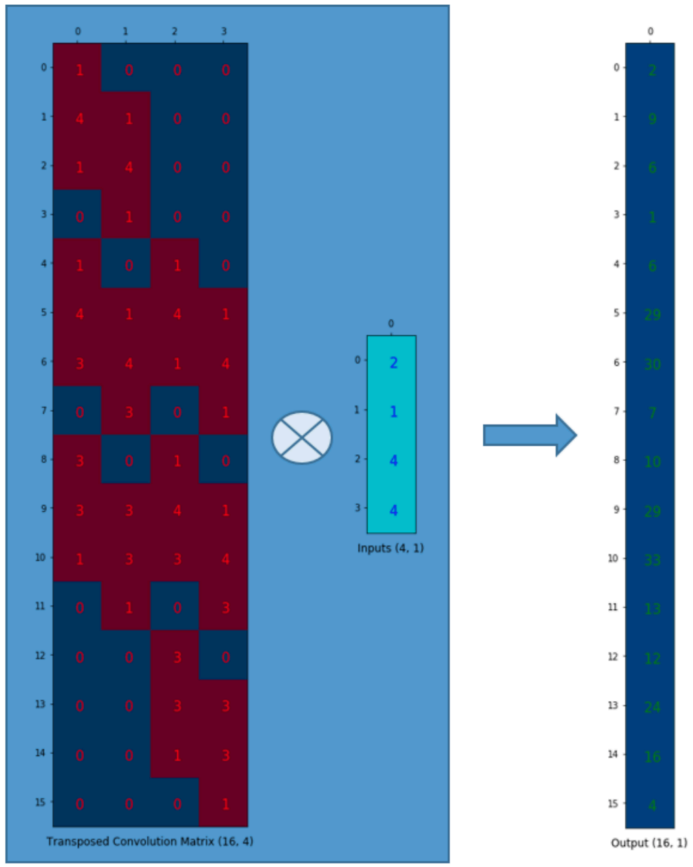
Transposed convolution as matrix multiplication

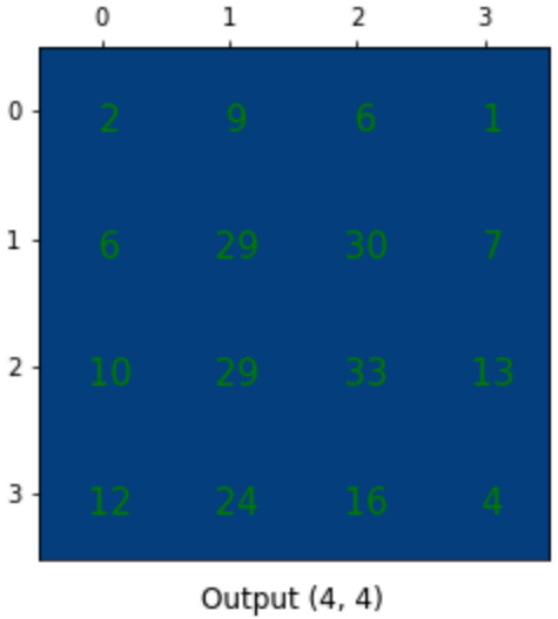


The output 4x1 matrix can be reshaped into a 2x2 matrix which gives us the same result as before.



Transposed convolution as matrix multiplication





Transposed Convolution (i.e. ~~Deconv~~)

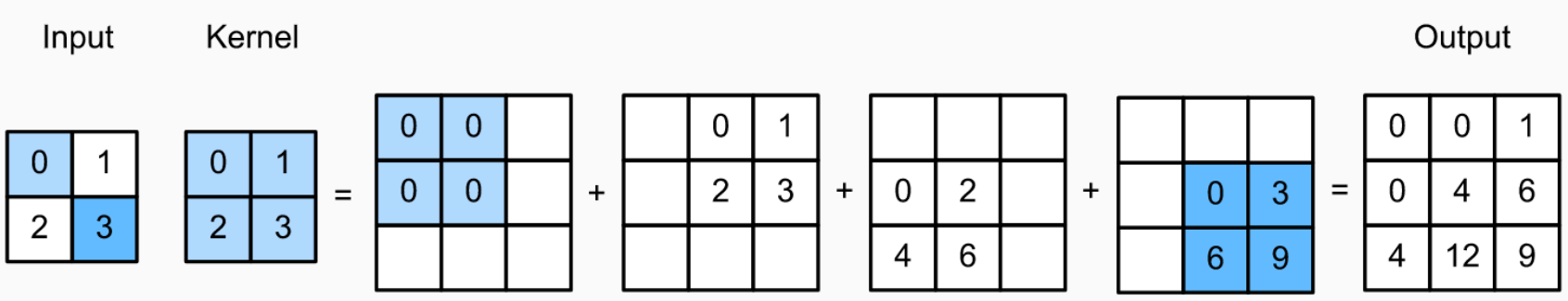
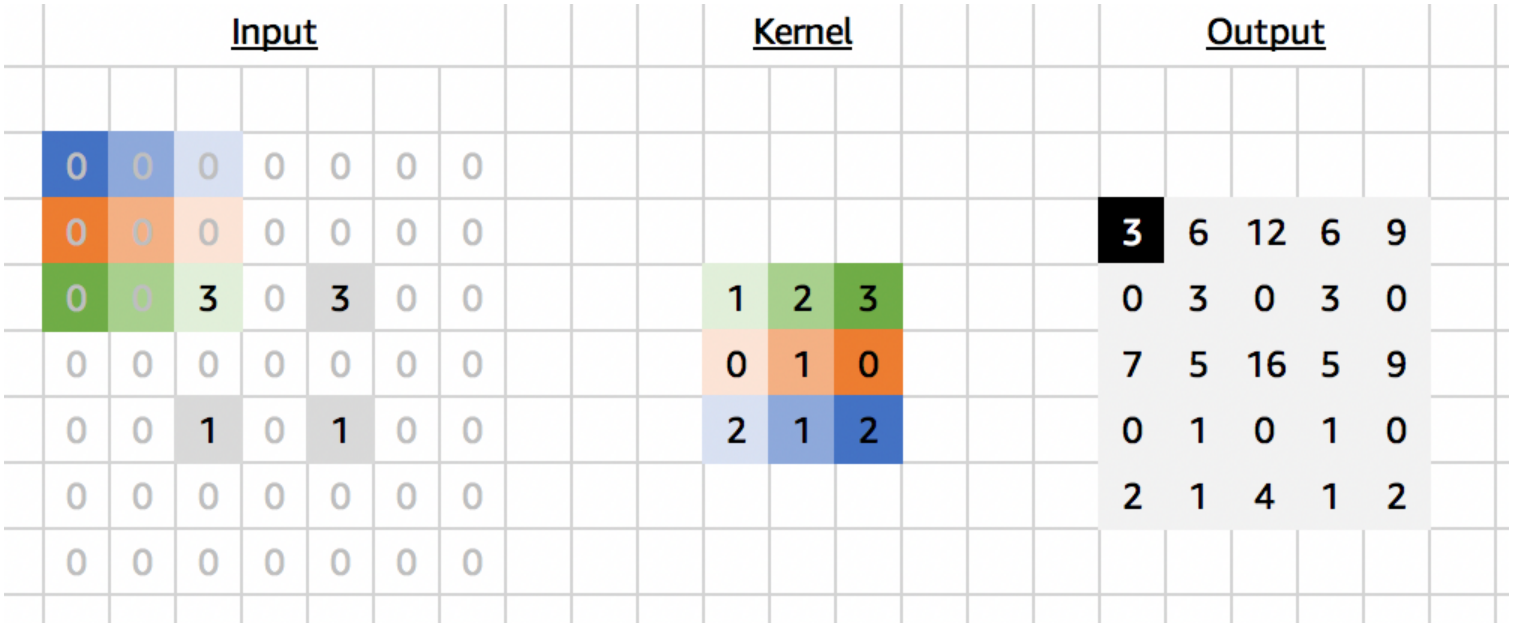


Fig. 12.10.1 Transposed convolution layer with a 2×2 kernel.

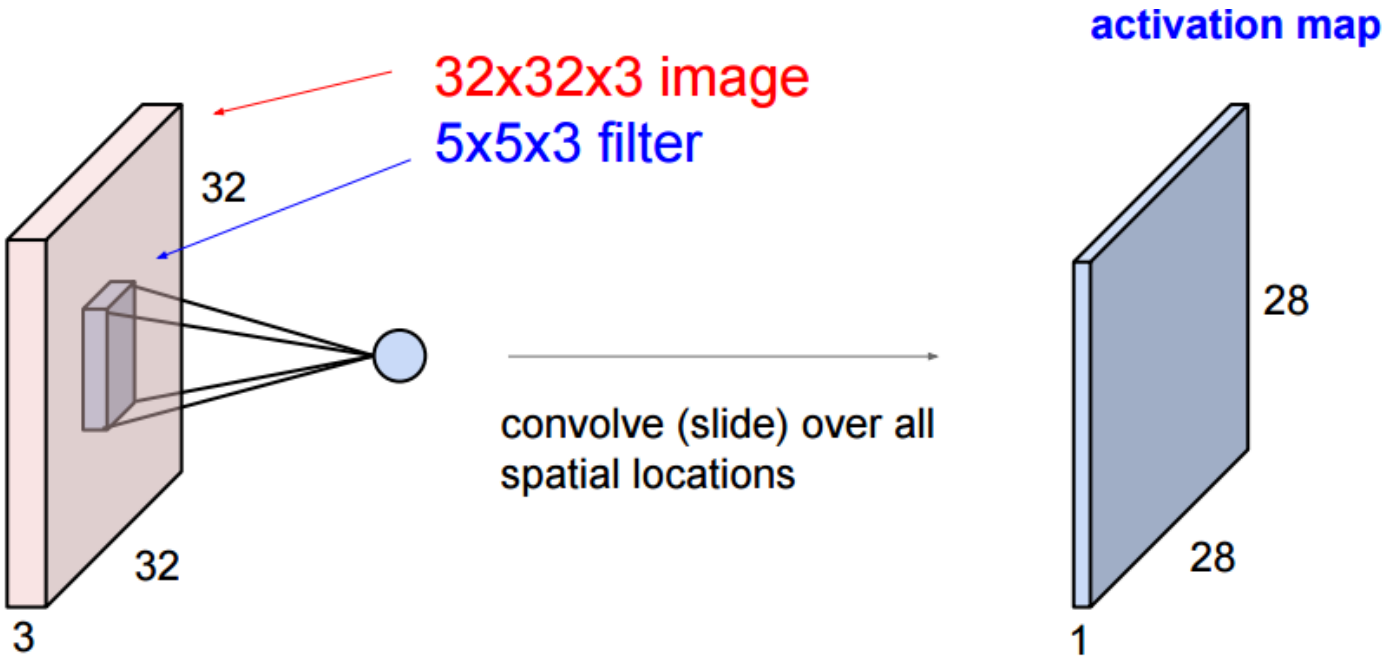
https://d2l.ai/chapter_computer-vision/tranposed-conv.html

Transposed Convolution (i.e. ~~Deconv~~)



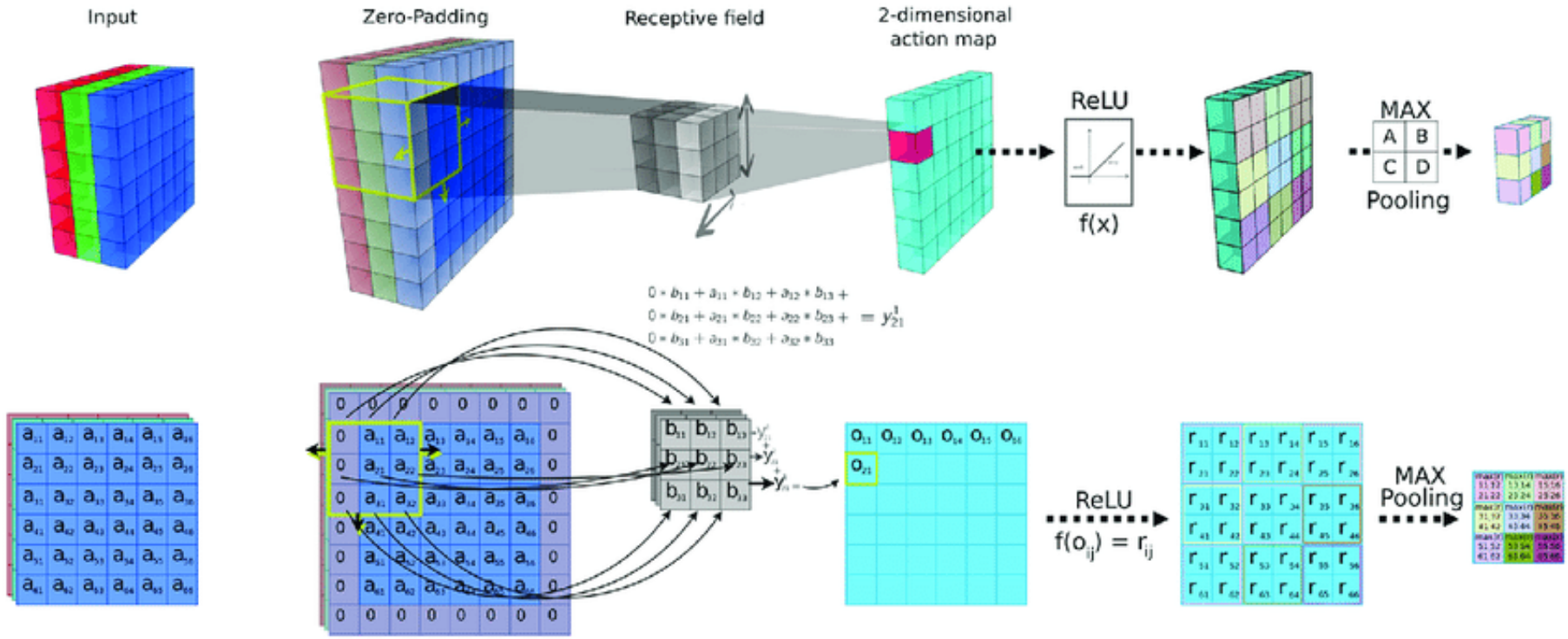
https://docs.google.com/spreadsheets/d/1a0HUXcc_75lBZyPmfNfNd31wx9QCqHKl3u4JJes9BRs/edit?usp=sharing

Convolution Layer

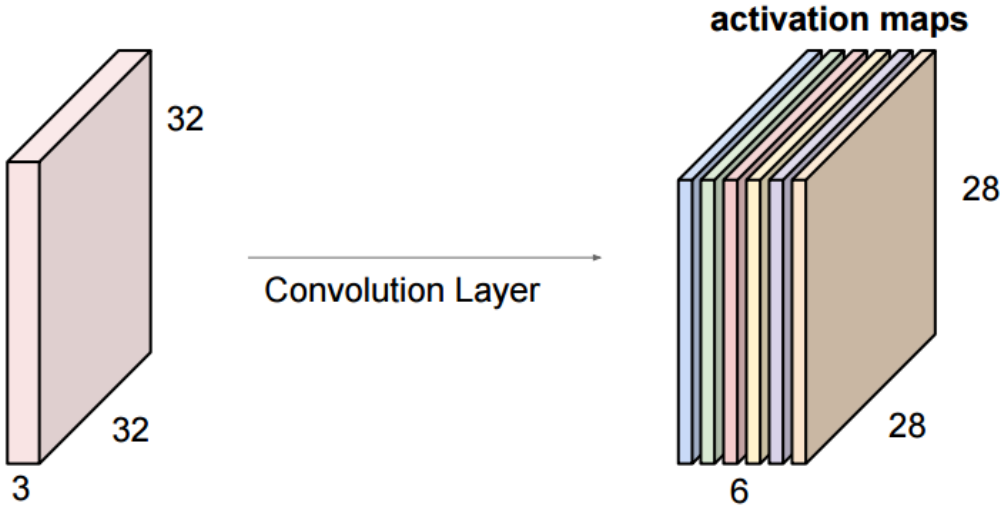


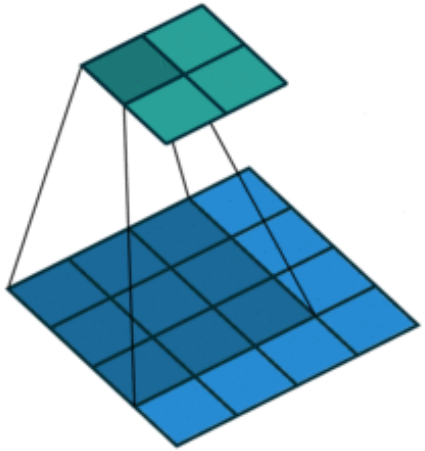
Convolution layer

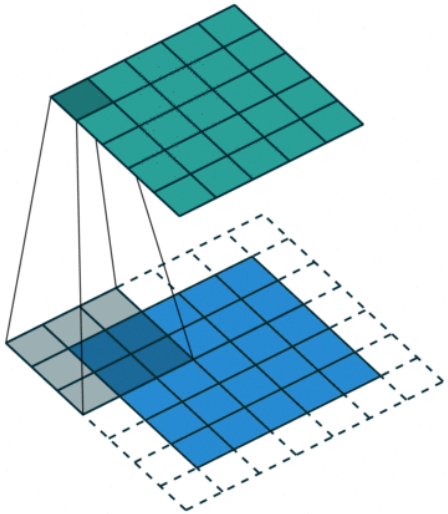
- Parametrized by: height, width, depth, stride, padding, number of filters, type of activation function



Convolution layer







MNIST

- (Modified National Institute of Standards and Technology database)

10 classes
60,000 training images and 10,000 testing images
Each image is 28x28 pixel

MNIST

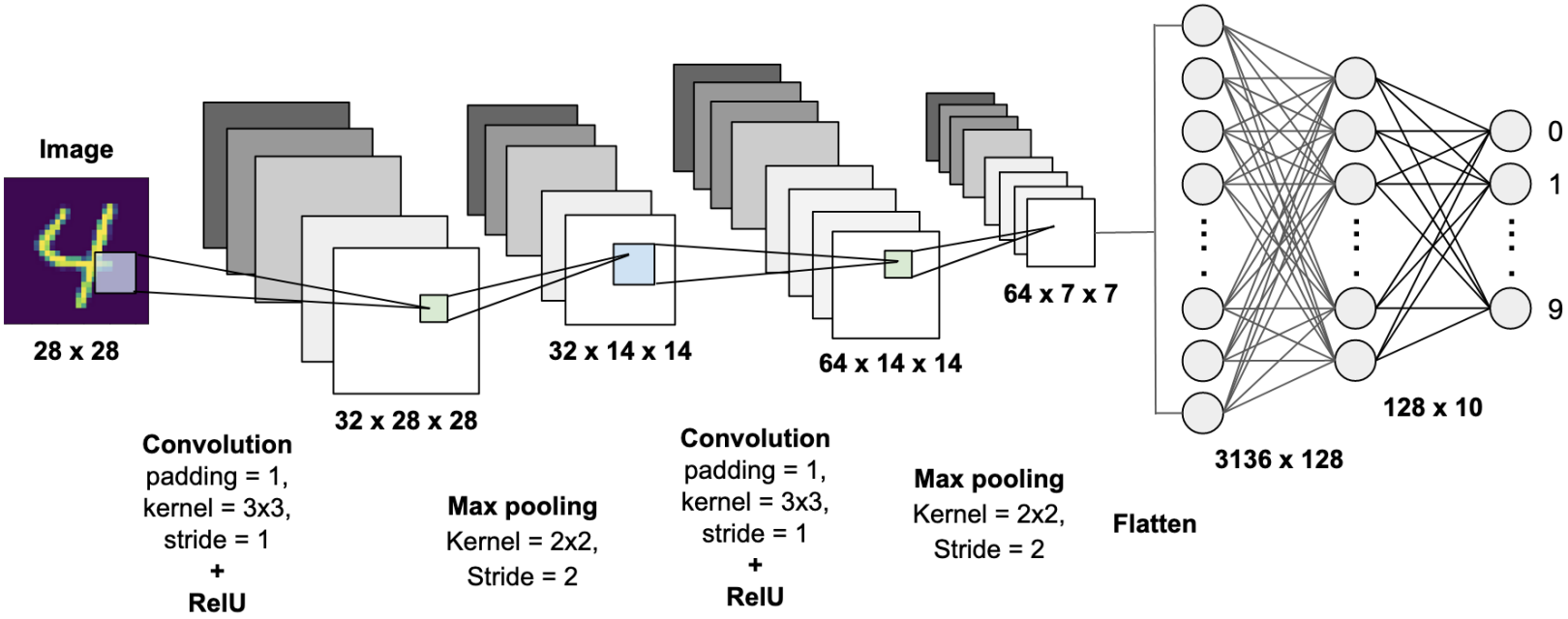


Image
28 x 28
32 x 28 x 28
Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU
32 x 14 x 14
Max pooling
Kernel = 2x2,
Stride = 2
64 x 14 x 14
Convolution
padding = 1,
kernel = 3x3,
stride = 1
+
RelU
64 x 7 x 7
Max pooling
Kernel = 2x2,
Stride = 2
Flatten
3136 x 128
128 x 10
0
1
9

```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

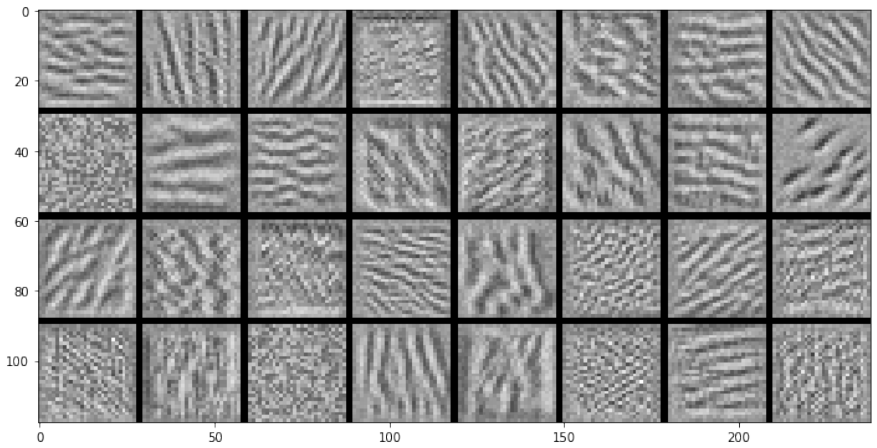
}

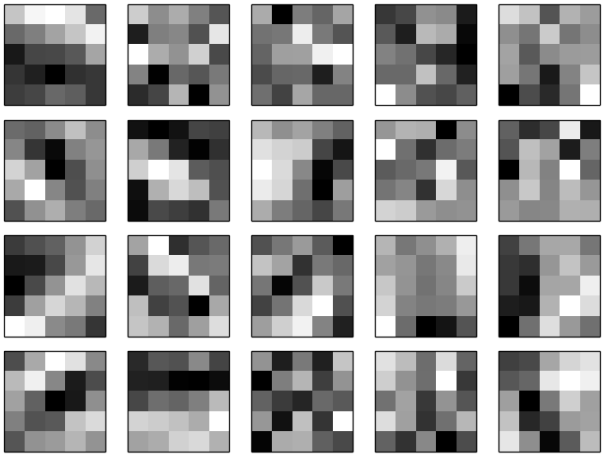
```
from __future__ import print_function
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K

batch_size = 128
num_classes = 10
epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = mnist.load_data()

if K.image_data_format() == 'channels_first':
    x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
    x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
    input_shape = (1, img_rows, img_cols)
else:
    x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
    x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
    input_shape = (img_rows, img_cols, 1)
```

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255
print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=input_shape))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
```

```
model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adadelta(),
              metrics=['accuracy'])

model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(x_test, y_test))
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

99.25% test accuracy after 12 epochs


0
20
40
60
80
100
0
50
100
150
200



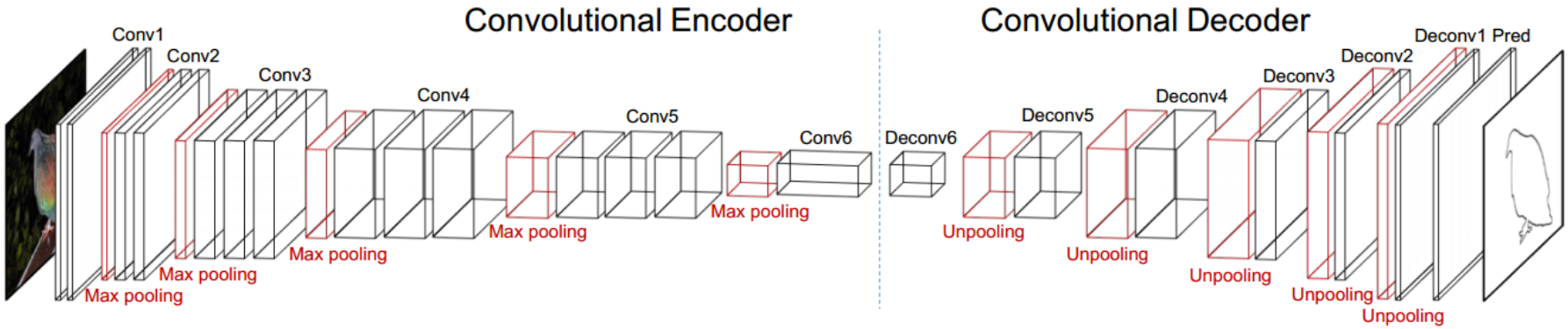
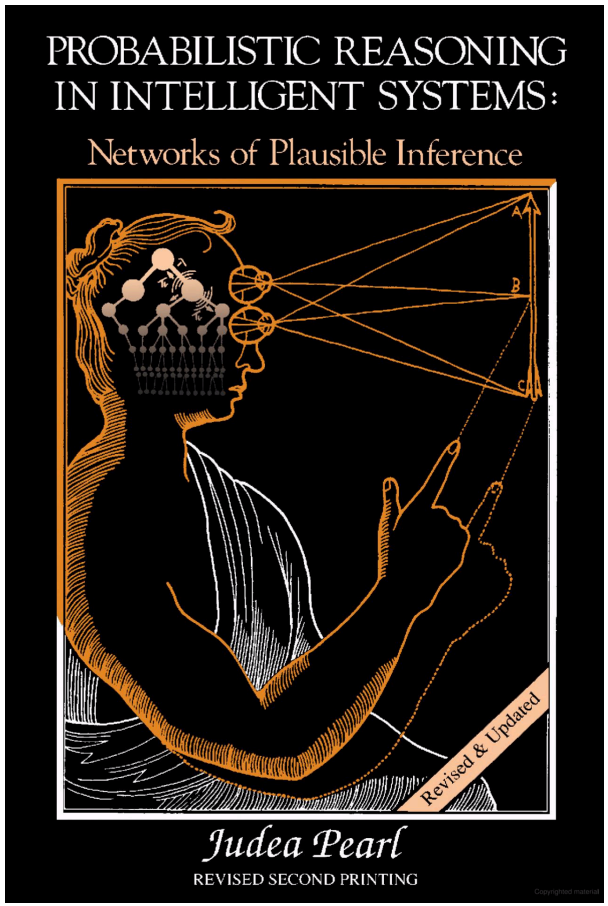
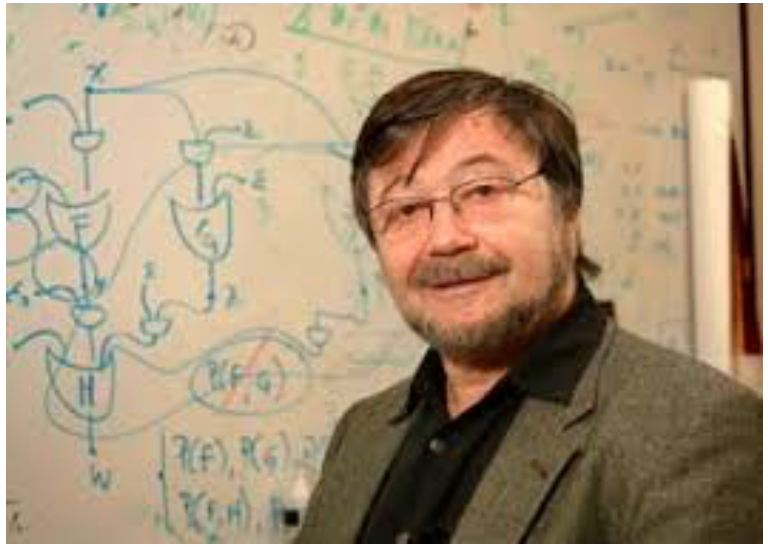


Figure 2. Architecture of the proposed fully convolutional encoder-decoder network.

Graphical Models





Prof. Judea Pearl

Graphical models

- Nodes: random variables
  - Discrete, Parametric, Nonparametric