

# Lecture 11

## Introduction to Deep Learning

COMP 474/6741, Winter 2021

[Introduction](#)

Perceptron and  
Backpropagation  
Image Classification

[Deep Learning  
Architectures](#)

Convolutional Neural  
Networks (CNNs)  
CNN for Text  
Sentiment Analysis

[Notes and Further  
Reading](#)

René Witte  
Department of Computer Science  
and Software Engineering  
Concordia University

## Introduction

Perceptron and  
Backpropagation  
Image Classification

## Deep Learning Architectures

Convolutional Neural  
Networks (CNNs)  
CNN for Text  
Sentiment Analysis

## Notes and Further Reading

## 1 Introduction

## 2 Deep Learning Architectures

## 3 Notes and Further Reading

# Single Neuron (Perceptron)

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)

[Image Classification](#)

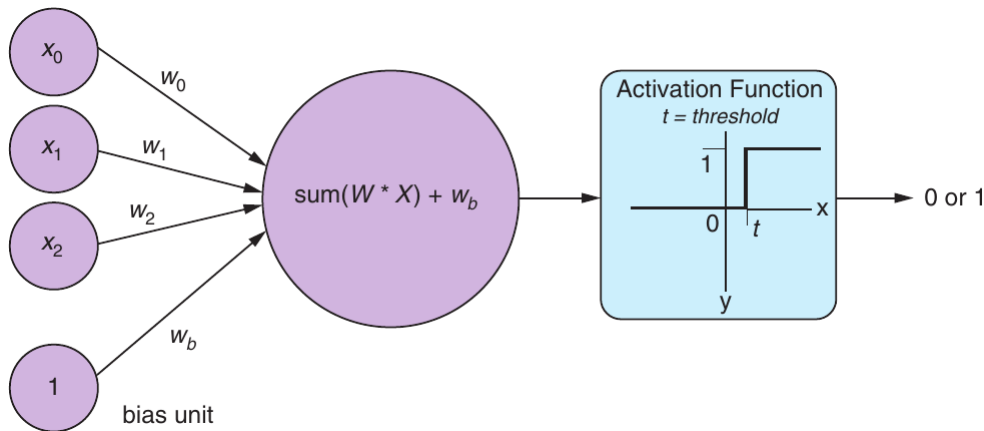
[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)

[Sentiment Analysis](#)

[Notes and Further  
Reading](#)



Copyright 2019 by Manning Publications Co., [LH#H19]

# Multi-layer neural networks with hidden weights

René Witte



[Introduction](#)

[Perceptron and Backpropagation](#)

[Image Classification](#)

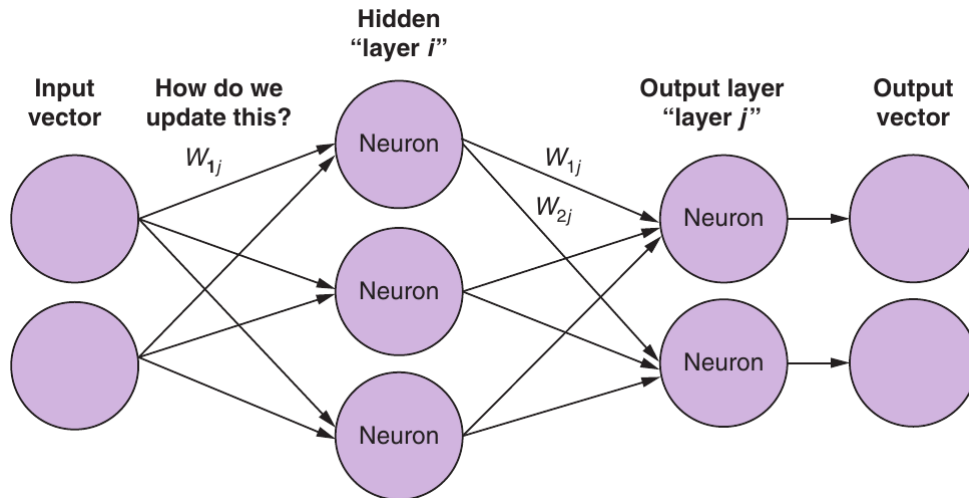
[Deep Learning Architectures](#)

[Convolutional Neural Networks \(CNNs\)](#)

[CNN for Text](#)

[Sentiment Analysis](#)

[Notes and Further Reading](#)



Copyright 2019 by Manning Publications Co., [LH9119]

# Feedforward Network (Word2vec)

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)

[Image Classification](#)

[Deep Learning  
Architectures](#)

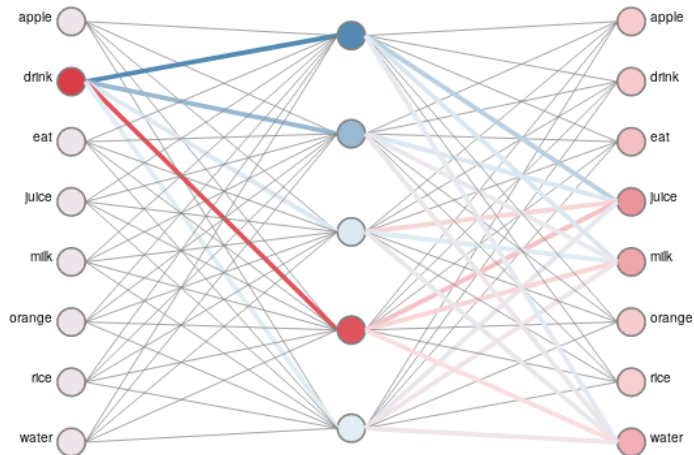
[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)

[Sentiment Analysis](#)

[Notes and Further  
Reading](#)

Neurons



## Introduction

### Perceptron and Backpropagation

Image Classification

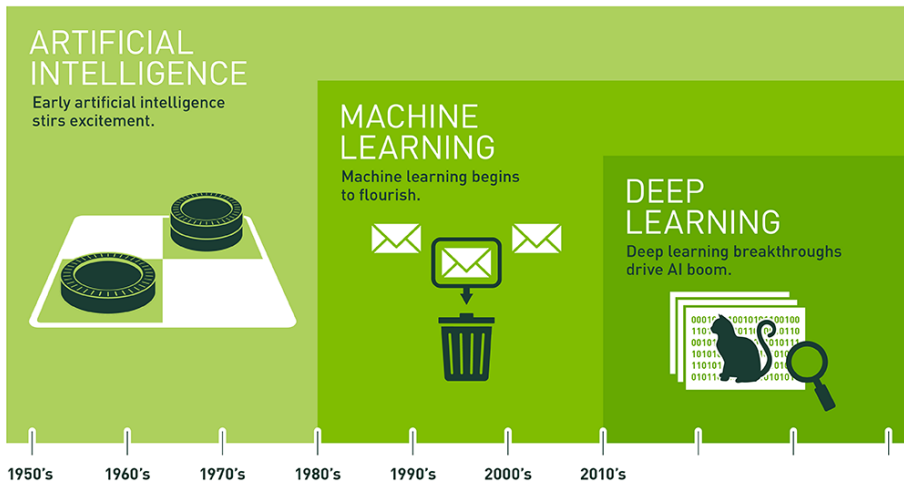
## Deep Learning Architectures

Convolutional Neural  
Networks (CNNs)

CNN for Text

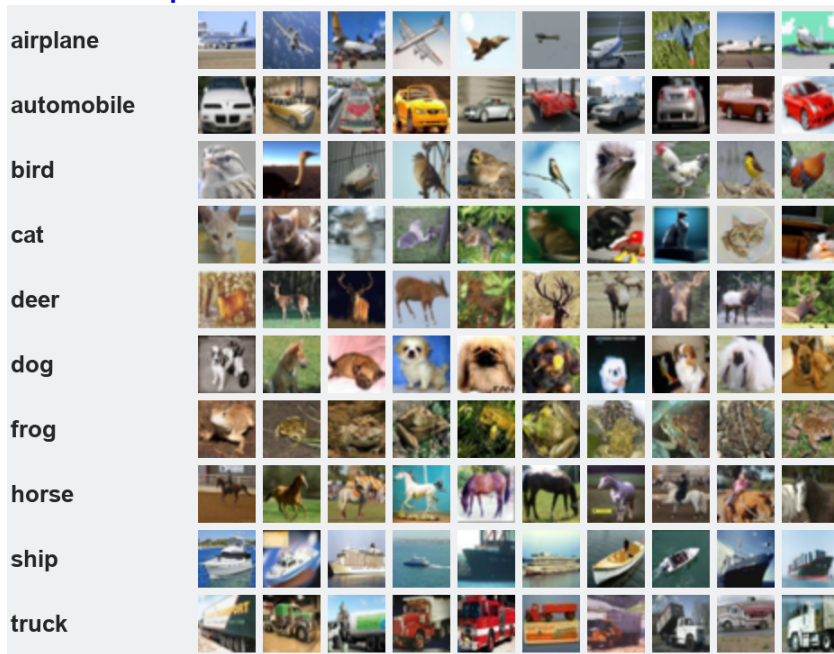
Sentiment Analysis

## Notes and Further Reading



Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

# Classification Example: The CIFAR-10 Dataset



60000 32x32 colour images in 10 classes: <https://www.cs.toronto.edu/~kriz/cifar.html>

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)

[Image Classification](#)

[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)

[Sentiment Analysis](#)

[Notes and Further  
Reading](#)

# Loading the data (Keras/TensorFlow)

```
import numpy as np
from keras.utils import to_categorical
from keras.datasets import cifar10

(x_train, y_train), (x_test, y_test) = cifar10.load_data()

NUM_CLASSES = 10

x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0

y_train = to_categorical(y_train, NUM_CLASSES)
y_test = to_categorical(y_test, NUM_CLASSES)
```

See [https://github.com/davidADSP/GDL\\_code/blob/master/02\\_01\\_deep\\_learning\\_deep\\_neural\\_network.ipynb](https://github.com/davidADSP/GDL_code/blob/master/02_01_deep_learning_deep_neural_network.ipynb)



## Training data

We use 50000 images for training (and 10000 for testing):

- `x_train` is an array of shape `[50000, 32, 32, 3]`
- First dimension: index of the image in the dataset
- Second, third dimension: size of the image
- Fourth dimension: RGB channels
- We normalized the RGB values from `[0, 255]` to `[0, 1]`

This is called a (four-dimensional) **tensor**

## Example

`x_train[54, 12, 13, 1]` is 0.36862746

- Image 54, pixel at (12, 13), green channel

# Tensors: $n$ -dimensional arrays with $n > 2$

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)

[Image Classification](#)

[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)

[Sentiment Analysis](#)

[Notes and Further  
Reading](#)

## Scalar   Vector   Matrix   Tensor

1

$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$

$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$

$\begin{bmatrix} \begin{bmatrix} 1 & 2 \end{bmatrix} & \begin{bmatrix} 3 & 2 \end{bmatrix} \\ \begin{bmatrix} 1 & 7 \end{bmatrix} & \begin{bmatrix} 5 & 4 \end{bmatrix} \end{bmatrix}$

<https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/>

# Neural Network for CIFAR-10

René Witte



## Introduction

Perceptron and  
Backpropagation

Image Classification

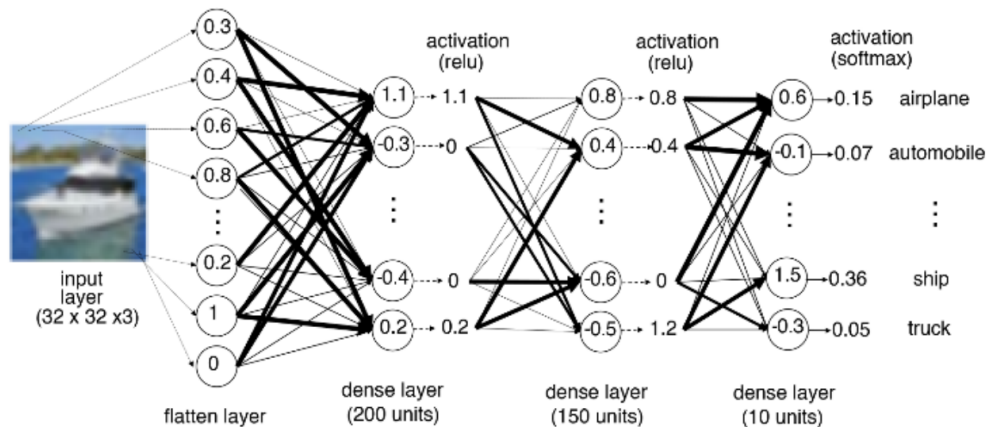
## Deep Learning Architectures

Convolutional Neural  
Networks (CNNs)

CNN for Text

Sentiment Analysis

## Notes and Further Reading



Copyright 2019 Applied Data Science Partners Ltd., [Fos19]

```
from keras.layers import Input, Flatten, Dense
from keras.models import Model
```

```
input_layer = Input((32, 32, 3))
```

```
x = Flatten()(input_layer)
```

```
x = Dense(200, activation = 'relu')(x)
```

```
x = Dense(150, activation = 'relu')(x)
```

```
output_layer = Dense(NUM_CLASSES, activation = 'softmax')(x)
```

```
model = Model(input_layer, output_layer)
```

# ReLU (Rectified Linear Unit) Activation Function

René Witte



Introduction

Perceptron and  
Backpropagation

Image Classification

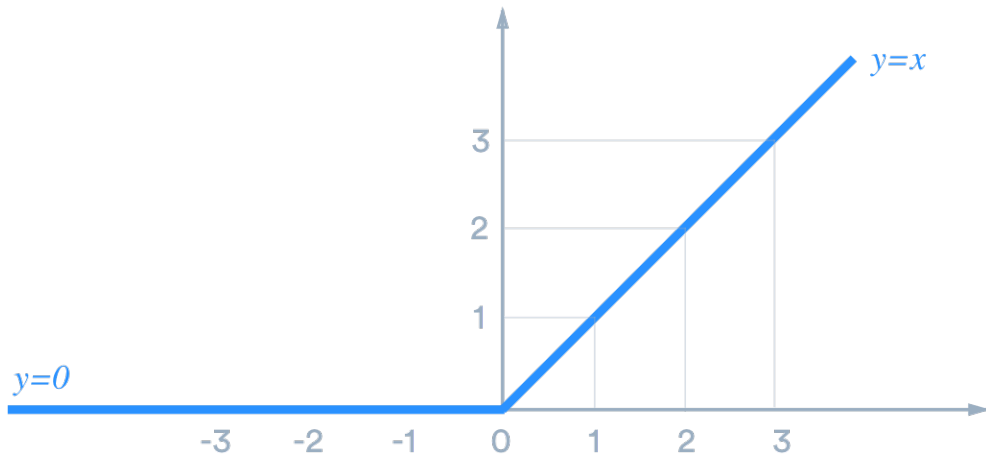
Deep Learning  
Architectures

Convolutional Neural  
Networks (CNNs)

CNN for Text

Sentiment Analysis

Notes and Further  
Reading



<https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7>

```
from keras.optimizers import Adam

opt = Adam(lr=0.0005)
model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])
```

Layer (type)	Output Shape	Param #
=====	=====	=====
input_2 (InputLayer)	(None, 32, 32, 3)	0
flatten_2 (Flatten)	(None, 3072)	0
dense_4 (Dense)	(None, 200)	614600
dense_5 (Dense)	(None, 150)	30150
dense_6 (Dense)	(None, 10)	1510
=====	=====	=====
Total params: 646,260		
Trainable params: 646,260		
Non-trainable params: 0		

```
model.fit(x_train,  
          y_train,  
          batch_size = 32,  
          epochs = 10,  
          shuffle = True)
```



```
Epoch 1/10
50000/50000 [=====] - 13s 262us/step - loss: 1.8490 - accuracy: 0.3361
Epoch 2/10
50000/50000 [=====] - 13s 257us/step - loss: 1.6592 - accuracy: 0.4104
Epoch 3/10
50000/50000 [=====] - 13s 259us/step - loss: 1.5827 - accuracy: 0.4372
Epoch 4/10
50000/50000 [=====] - 13s 256us/step - loss: 1.5290 - accuracy: 0.4571
Epoch 5/10
50000/50000 [=====] - 13s 261us/step - loss: 1.4882 - accuracy: 0.4725
Epoch 6/10
50000/50000 [=====] - 13s 257us/step - loss: 1.4537 - accuracy: 0.4847
Epoch 7/10
50000/50000 [=====] - 13s 257us/step - loss: 1.4293 - accuracy: 0.4940
Epoch 8/10
50000/50000 [=====] - 13s 263us/step - loss: 1.4056 - accuracy: 0.4988
Epoch 9/10
50000/50000 [=====] - 12s 246us/step - loss: 1.3849 - accuracy: 0.5086
Epoch 10/10
50000/50000 [=====] - 12s 244us/step - loss: 1.3659 - accuracy: 0.5150
```

Training for 10 epochs:

```
model.evaluate(x_test, y_test)
[1.4283625371932984, 0.49559998512268066]
```

Training for 100 epochs:

```
model.evaluate(x_test, y_test)
[1.6529622526168823, 0.5023999810218811]
```

Training for 1000 epochs:

```
...
50000/50000 [=====] - 13s 255us/step - loss: 0.2840 - accuracy: 0.8986
10000/10000 [=====] - 0s 47us/step

model.evaluate(x_test, y_test)
[6.070724856567383, 0.4410000145435333]
```

## Show result for some random test images

```
CLASSES = np.array(['airplane', 'automobile', 'bird', 'cat', 'deer',  
                    'dog', 'frog', 'horse', 'ship', 'truck'])  
  
preds = model.predict(x_test)  
preds_single = CLASSES[np.argmax(preds, axis = -1)]  
actual_single = CLASSES[np.argmax(y_test, axis = -1)]  
  
n_to_show = 10  
indices = np.random.choice(range(len(x_test)), n_to_show)  
  
fig = plt.figure(figsize=(15, 3))  
fig.subplots_adjust(hspace=0.4, wspace=0.4)  
  
for i, idx in enumerate(indices):  
    img = x_test[idx]  
    ax = fig.add_subplot(1, n_to_show, i+1)  
    ax.axis('off')  
    ax.text(0.5, -0.35, 'pred_ = ' + str(preds_single[idx]), fontsize=10,  
           ha='center', transform=ax.transAxes)  
    ax.text(0.5, -0.7, 'act_ = ' + str(actual_single[idx]), fontsize=10,  
           ha='center', transform=ax.transAxes)  
    ax.imshow(img)
```

## Some random results...

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)

[Image Classification](#)

[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)

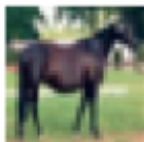
[CNN for Text](#)

[Sentiment Analysis](#)

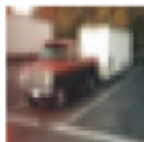
[Notes and Further  
Reading](#)



pred = ship  
act = deer



pred = horse  
act = horse



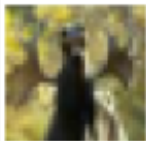
pred = horse  
act = truck



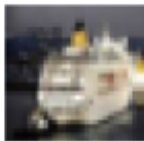
pred = bird  
act = deer



pred = airplane  
act = automobile



pred = bird  
act = deer



pred = dog  
act = ship



pred = horse  
act = ship



pred = ship  
act = ship



pred = automobile  
act = automobile

## Introduction

Perceptron and  
Backpropagation  
Image Classification

## Deep Learning Architectures

Convolutional Neural  
Networks (CNNs)  
CNN for Text  
Sentiment Analysis

## Notes and Further Reading

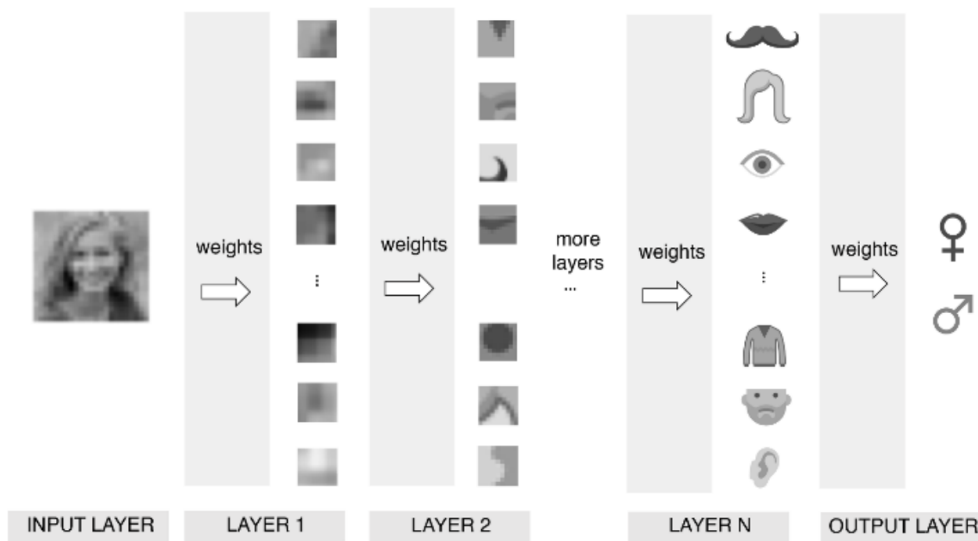
## 1 Introduction

## 2 Deep Learning Architectures

Convolutional Neural Networks (CNNs)  
CNN for Text  
Sentiment Analysis

## 3 Notes and Further Reading

# Deep Learning Conceptual Diagram



Copyright 2019 Applied Data Science Partners Ltd., [Fos19]

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)  
[Image Classification](#)

[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)  
[Sentiment Analysis](#)

[Notes and Further  
Reading](#)

# The Convolution Operation

## Applying a $3 \times 3 \times 1$ filter (a.k.a. kernel)

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)

[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text  
Sentiment Analysis](#)

[Notes and Further  
Reading](#)

3x3 portion  
of an image

0.6	0.2	0.6
0.1	-0.2	-0.3
-0.5	-0.1	-0.3

\*

filter

1	1	1
0	0	0
-1	-1	-1

= 2.3

-0.6	-0.2	-0.6
-0.1	0.2	0.3
0.5	0.1	0.3

\*

1	1	1
0	0	0
-1	-1	-1

= -2.3

# Applying Two Convolutional Filters to a Grayscale Image

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation  
Image Classification](#)

[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)

[Sentiment Analysis](#)

[Notes and Further  
Reading](#)

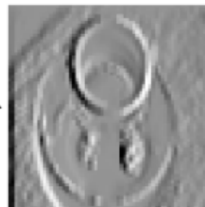
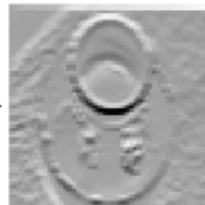


input layer  
 $1 \times 64 \times 64 \times 1$   
batch\_size x height x  
width x number of filters

1	1	1
0	0	0
-1	-1	-1

-1	0	1
-1	0	1
-1	0	1

2 filters  
(each filter is  $3 \times 3 \times 1$ )



output  
 $1 \times 64 \times 64 \times 2$   
batch\_size x height x  
width x number of filters



```
input_layer = Input(shape=(64,64,1))

conv_layer_1 = Conv2D(filters = 2,
                      kernel_size = (3,3),
                      strides = 1,
                      padding = "same")(input_layer)
```

# Strides & Padding

René Witte

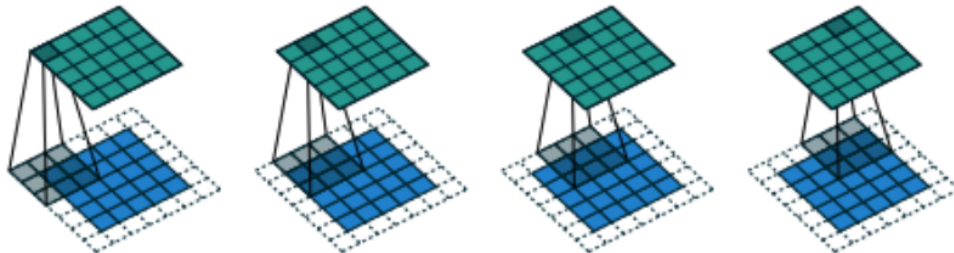


## Stride (a.k.a. step size)

Distance we move the convolution across the input at each step

## Padding

Padding extends the filter over the edges, using zero values



Copyright 2019 Applied Data Science Partners Ltd., [Fos19]

A  $3 \times 3 \times 1$  kernel (gray) being passed over a  $5 \times 5 \times 1$  input image (blue), with padding="same" and strides=1, to generate the  $5 \times 5 \times 1$  output (green)

→ **Worksheet #10: Task 1**

[Introduction](#)

[Perceptron and  
Backpropagation](#)  
[Image Classification](#)

[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)  
[Sentiment Analysis](#)

[Notes and Further  
Reading](#)

# Convolutional Neural Network

René Witte



## Introduction

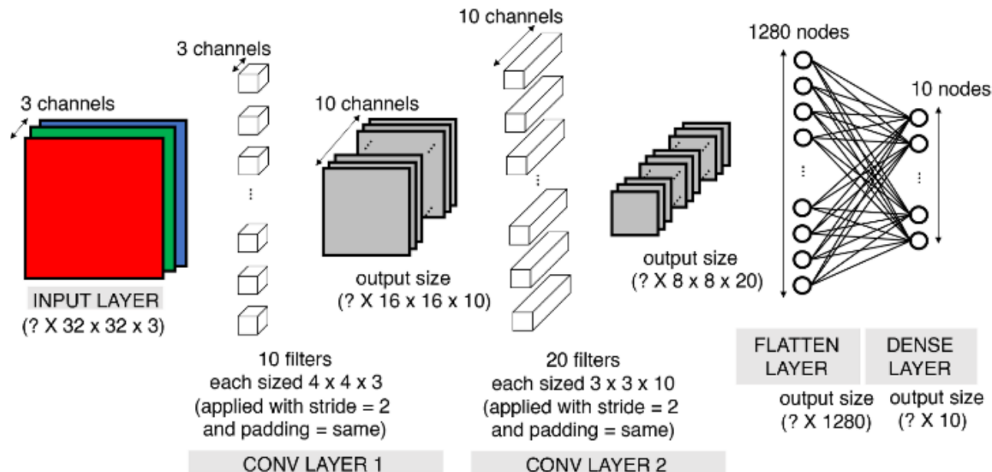
Perceptron and  
Backpropagation  
Image Classification

## Deep Learning Architectures

Convolutional Neural  
Networks (CNNs)

CNN for Text  
Sentiment Analysis

## Notes and Further Reading



```
input_layer = Input(shape=(32, 32, 3))

conv_layer_1 = Conv2D(filters = 10,
                      kernel_size = (4, 4),
                      strides = 2,
                      padding = 'same'
                      )(input_layer)

conv_layer_2 = Conv2D(filters = 20,
                      kernel_size = (3, 3),
                      strides = 2,
                      padding = 'same'
                      )(conv_layer_1)
```

[https://github.com/davidADSP/GDL\\_code/blob/master/02\\_03\\_deep\\_learning\\_conv\\_neural\\_network.ipynb](https://github.com/davidADSP/GDL_code/blob/master/02_03_deep_learning_conv_neural_network.ipynb)

**Input Shape:** `(None, 32, 32, 3)` (None: can process inputs in batch)

**First conv layer:** `4×4×3` (`kernel_size = (4, 4)`)

- Three channels in preceding layer (red, green, blue)
- Number of weights (parameters) is  $(4 \times 4 \times 3 + 1) \times 10 = 490$
- Depth of filters in a layer = number of channels in preceding layer
- General formula (with `padding = "same"`, meaning same size as input, padding with zeros):

$$\text{output\_shape} = \left( \text{None}, \frac{\text{input\_height}}{\text{stride}}, \frac{\text{input\_width}}{\text{stride}}, \text{filters} \right)$$

**Second conv layer:** 20 filters, `3×3` size, depth 10

**Flatten:** transform into  $8 \times 8 \times 20 = 1280$  unit vector using Keras `Flatten`

**Output:** 10-unit `Dense` layer with softmax activation

→ **Worksheet #10: Task 2**

## Avoiding ‘exploding gradient’ problem

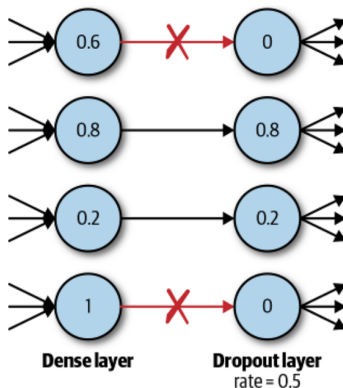
- Over time, calculating the gradient in earlier layers can grow exponentially large
- Loss functions start to return NaN → **overflow error**
- Caused by **covariance shift** in weights
- Reduced in practice through **batch normalization**
- Added as a separate layer, in Keras: `BatchNormalization(momentum = 0.9)`
- Calculates the mean and standard deviation of each of its input channels across the batch and normalizes by subtracting the mean and dividing by the standard deviation
- Learns two parameters, **scale** (gamma) and **shift** (beta)
- Output is normalized input, scaled by gamma and shifted by beta

[Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,  
<https://arxiv.org/abs/1502.03167>]

# Dropout Layer

## Goal: Avoid Overfitting

- Want a network that can **generalize**, not just remember the input samples
- We add a **regularization** technique, here a **dropout** layer
- Keras: `Dropout (rate = 0.5)`



Copyright 2019 Applied Data Science Partners Ltd., [Fos19]

[Nitish Srivastava et al., “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research* 15 (2014): 1929–1958, <http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf>].

# Complete Network in Keras

```
input_layer = Input((32, 32, 3))

x = Conv2D(filters=32, kernel_size=3, strides=1, padding='same')(input_layer)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Conv2D(filters = 32, kernel_size = 3, strides = 2, padding = 'same')(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Conv2D(filters = 64, kernel_size = 3, strides = 1, padding = 'same')(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Conv2D(filters = 64, kernel_size = 3, strides = 2, padding = 'same')(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)

x = Flatten()(x)
x = Dense(128)(x)
x = BatchNormalization()(x)
x = LeakyReLU()(x)
x = Dropout(rate = 0.5)(x)

x = Dense(NUM_CLASSES)(x)
output_layer = Activation('softmax')(x)
model = Model(input_layer, output_layer)
```

René Witte



[Introduction](#)

Perceptron and  
Backpropagation  
Image Classification

[Deep Learning  
Architectures](#)

Convolutional Neural  
Networks (CNNs)

CNN for Text  
Sentiment Analysis

[Notes and Further  
Reading](#)



## Resulting Model (1/2)

René Witte



[Introduction](#)

Perceptron and  
Backpropagation  
Image Classification

[Deep Learning  
Architectures](#)

Convolutional Neural  
Networks (CNNs)

CNN for Text  
Sentiment Analysis

[Notes and Further  
Reading](#)

Layer (type)	Output Shape	Param #
=====		
input_13 (InputLayer)	(None, 32, 32, 3)	0
conv2d_11 (Conv2D)	(None, 32, 32, 32)	896
batch_normalization_6 (Batch Normalization)	(None, 32, 32, 32)	128
leaky_re_lu_6 (LeakyReLU)	(None, 32, 32, 32)	0
conv2d_12 (Conv2D)	(None, 16, 16, 32)	9248
batch_normalization_7 (Batch Normalization)	(None, 16, 16, 32)	128
leaky_re_lu_7 (LeakyReLU)	(None, 16, 16, 32)	0
conv2d_13 (Conv2D)	(None, 16, 16, 64)	18496
batch_normalization_8 (Batch Normalization)	(None, 16, 16, 64)	256
leaky_re_lu_8 (LeakyReLU)	(None, 16, 16, 64)	0
conv2d_14 (Conv2D)	(None, 8, 8, 64)	36928
...		

## Resulting Model (2/2)

Layer (type)	Output Shape	Param #
=====		
...		
conv2d_14 (Conv2D)	(None, 8, 8, 64)	36928
<hr/>		
batch_normalization_9 (Batch Normalization)	(None, 8, 8, 64)	256
<hr/>		
leaky_re_lu_9 (LeakyReLU)	(None, 8, 8, 64)	0
<hr/>		
flatten_13 (Flatten)	(None, 4096)	0
<hr/>		
dense_30 (Dense)	(None, 128)	524416
<hr/>		
batch_normalization_10 (Batch Normalization)	(None, 128)	512
<hr/>		
leaky_re_lu_10 (LeakyReLU)	(None, 128)	0
<hr/>		
dropout_2 (Dropout)	(None, 128)	0
<hr/>		
dense_31 (Dense)	(None, 10)	1290
<hr/>		
activation_2 (Activation)	(None, 10)	0
=====		
Total params: 592,554		
Trainable params: 591,914		
Non-trainable params: 640		

```
opt = Adam(lr=0.0005)

model.compile(loss='categorical_crossentropy',
              optimizer=opt,
              metrics=['accuracy'])

model.fit(x_train,
          y_train,
          batch_size=32,
          epochs=10,
          shuffle=True,
          validation_data = (x_test, y_test))
```

## Training (output)

Train on 50000 samples, validate on 10000 samples

Epoch 1/10

50000/50000 [=====] - 118s 2ms/step - loss: 1.5609 - accuracy: 0.4556 - val\_loss: 1.2602 - val\_

Epoch 2/10

50000/50000 [=====] - 114s 2ms/step - loss: 1.1528 - accuracy: 0.5914 - val\_loss: 1.0027 - val\_

Epoch 3/10

50000/50000 [=====] - 112s 2ms/step - loss: 1.0021 - accuracy: 0.6472 - val\_loss: 0.9151 - val\_

Epoch 4/10

50000/50000 [=====] - 117s 2ms/step - loss: 0.9156 - accuracy: 0.6801 - val\_loss: 0.9089 - val\_

Epoch 5/10

50000/50000 [=====] - 116s 2ms/step - loss: 0.8517 - accuracy: 0.7018 - val\_loss: 0.8910 - val\_

Epoch 6/10

50000/50000 [=====] - 112s 2ms/step - loss: 0.8004 - accuracy: 0.7212 - val\_loss: 0.8599 - val\_

Epoch 7/10

50000/50000 [=====] - 118s 2ms/step - loss: 0.7532 - accuracy: 0.7369 - val\_loss: 0.8539 - val\_

Epoch 8/10

50000/50000 [=====] - 113s 2ms/step - loss: 0.7111 - accuracy: 0.7502 - val\_loss: 0.8647 - val\_

Epoch 9/10

50000/50000 [=====] - 113s 2ms/step - loss: 0.6760 - accuracy: 0.7633 - val\_loss: 0.8913 - val\_

Epoch 10/10

50000/50000 [=====] - 113s 2ms/step - loss: 0.6395 - accuracy: 0.7744 - val\_loss: 0.8169 - val\_

## Introduction

Perceptron and  
Backpropagation  
Image Classification

## Deep Learning Architectures

Convolutional Neural  
Networks (CNNs)

CNN for Text  
Sentiment Analysis

## Notes and Further Reading

```
model.evaluate(x_test, y_test)  
[0.8298392653465271, 0.7210999727249146]
```

## Some more random results...

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)  
[Image Classification](#)

[Deep Learning  
Architectures](#)

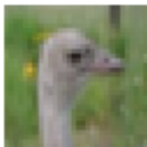
[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)  
[Sentiment Analysis](#)

[Notes and Further  
Reading](#)



pred = frog  
act = frog



pred = bird  
act = bird



pred = automobile  
act = automobile



pred = deer  
act = deer



pred = airplane  
act = airplane



pred = cat  
act = cat



pred = automobile  
act = automobile



pred = deer  
act = frog



pred = horse  
act = deer



pred = cat  
act = cat

## Ok, so what about text?

### CNN for Text

- No “vertical” information like in an image, but “horizontal” (left-to-right sequence)
- Instead of 2D convolutions, we have 1D convolutions

1 × 3 Filter

The cat and dog went to the bodega together.

1 × 3 Filter

The cat and dog went to the bodega together.

1 × 3 Filter

The cat and dog went to the bodega together.

# Encoding Input Text using Word Embeddings

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)  
[Image Classification](#)

[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)

[Sentiment Analysis](#)

[Notes and Further  
Reading](#)

The cat and dog went to the bodega together

.03	.92	.66	.72	.11	.15	.12	.00	.23
.00	.32	.61	.34	.63	.33	.23	.52	.23
.14	.62	.43	.32	.34	.00	.02	.34	.33
.24	.99	.62	.33	.27	.00	.66	.66	.56
.12	.02	.44	.42	.42	.11	.00	.23	.99
.32	.23	.55	.32	.22	.42	.00	.01	.25

Input  
(word embeddings)



## Introduction

Perceptron and  
Backpropagation  
Image Classification

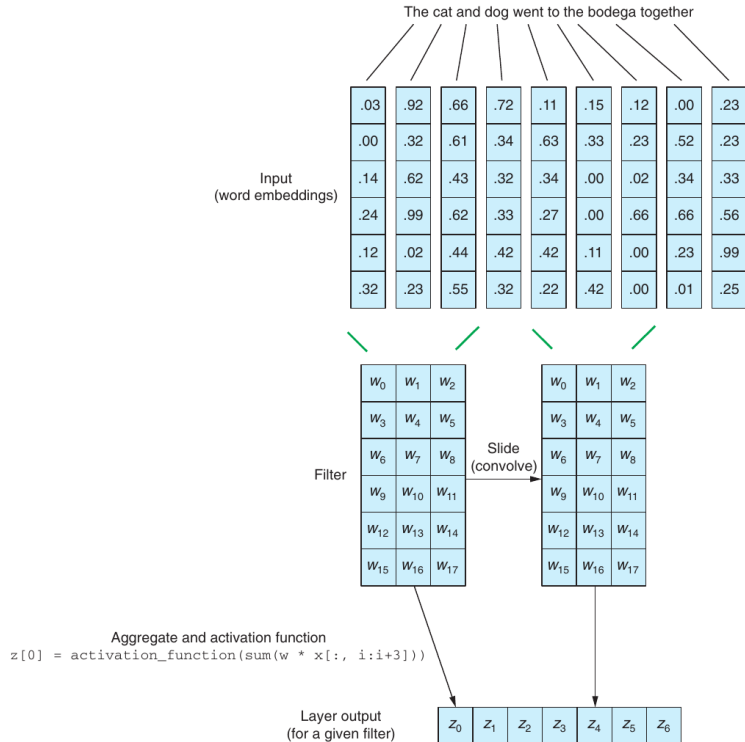
Deep Learning  
Architectures

Convolutional Neural  
Networks (CNNs)

## CNN for Text

Sentiment Analysis

Notes and Further  
Reading



# Pooling Layers

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)  
[Image Classification](#)

[Deep Learning  
Architectures](#)

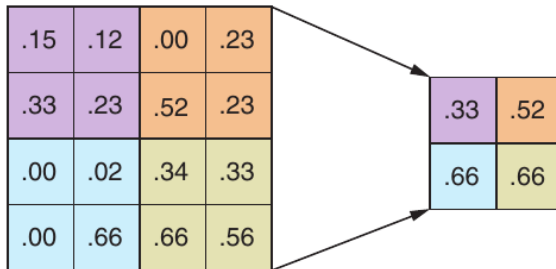
[Convolutional Neural  
Networks \(CNNs\)](#)

[CNN for Text](#)

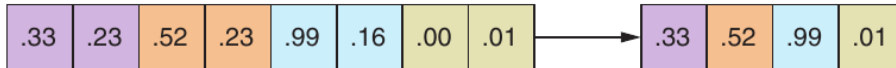
[Sentiment Analysis](#)

[Notes and Further  
Reading](#)

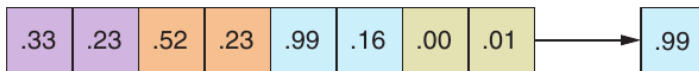
2D max pooling (2 x 2 window)



1D max pooling (1 x 2 window)



1D global max pooling



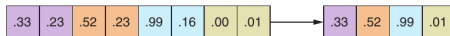
## Dimensionality Reduction

- Filtering results in new (filtered) versions of each data sample
- We can “throw away” some data by reducing the size
- Choose a **representative**, e.g., maximum in a  $2 \times 2$  window
- For text: **max** in a 1D (e.g.,  $1 \times 2$  window)
- Keras: `model.add(GlobalMaxPooling1D())` (default size is 2)

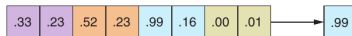
2D max pooling (2 x 2 window)



1D max pooling (1 x 2 window)



1D global max pooling



Copyright 2019 by Manning Publications Co., [LHH19]

## Dataset

Stanford AI movie dataset from <https://ai.stanford.edu/~amaas/data/sentiment/>

- 50,000 reviews from IMDB; upto 30 reviews/movie
- contains an even number of **positive** and **negative** reviews (so randomly guessing yields 50% accuracy)
- A negative review has a score  $\leq 4$  out of 10
- A positive review has a score  $\geq 7$  out of 10
- Neutral reviews are not included in the dataset

### Positive Example 6587\_9.txt, so id 6587, rating 9 stars

*The Master Blackmailer, based off of Sir Arthur Conan Doyle's short story, "the Adventure of Charles Augustus Milverton," is the first feature length Sherlock Holmes story with Jeremy Brett that I have seen. The story is interesting and dark...*

### Negative Example (9985\_1.txt), id 9985, rating 1 star

*Have I ever seen a film more shockingly inept? I can think of plenty that equal this one, but none which manage to outdo it. The cast are all horrible stereotypes lumbered with flat dialogue. I am ashamed for all of the people involved ...*

[Maas, Andrew L. et al., Learning Word Vectors for Sentiment Analysis, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, June 2011, ACL]

[Introduction](#)[Perceptron and  
Backpropagation](#)  
[Image Classification](#)[Deep Learning  
Architectures](#)[Convolutional Neural  
Networks \(CNNs\)](#)  
[CNN for Text](#)[Sentiment Analysis](#)[Notes and Further  
Reading](#)

# CNN for classifying IMDB movie reviews

```
model = Sequential()

# we add a Convolution1D, which will learn filters
# word group filters of size filter_length:
model.add(Conv1D(filters,
                  kernel_size,
                  padding='valid',
                  activation='relu',
                  strides=1,
                  input_shape=(maxlen, embedding_dims)))

# we use max pooling:
model.add(GlobalMaxPooling1D())

# We add a vanilla hidden layer:
model.add(Dense(hidden_dims))
model.add(Dropout(0.2))
model.add(Activation('relu'))

# We project onto a single unit output layer, and squash it with a sigmoid:
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

René Witte



[Introduction](#)

Perceptron and  
Backpropagation  
Image Classification

[Deep Learning  
Architectures](#)

Convolutional Neural  
Networks (CNNs)  
CNN for Text

Sentiment Analysis

[Notes and Further  
Reading](#)

# Applying the trained model

René Witte



[Introduction](#)

[Perceptron and  
Backpropagation](#)  
[Image Classification](#)

[Deep Learning  
Architectures](#)

[Convolutional Neural  
Networks \(CNNs\)](#)  
[CNN for Text](#)

[Sentiment Analysis](#)

[Notes and Further  
Reading](#)

```
>>> sample_1 = "I_hate_that_the_dismal_weather_had_me_down_for_so_long,  
when_will_it_break!_Ugh,_when_does_happiness_return?_The_sun_is_blinding  
and_the_puffy_clouds_are_too_thin._I_can't_wait_for_the_weekend."  
...
```

*(vectorize and shape input, see [LHH19])*

```
...  
>>> model.predict(test_vec)  
array([[ 0.12459087]], dtype=float32)
```

## Introduction

Perceptron and  
Backpropagation  
Image Classification

## Deep Learning Architectures

Convolutional Neural  
Networks (CNNs)  
CNN for Text  
Sentiment Analysis

## Notes and Further Reading

## 1 Introduction

## 2 Deep Learning Architectures

## 3 Notes and Further Reading

## Introduction

Perceptron and  
Backpropagation  
Image Classification

## Deep Learning Architectures

Convolutional Neural  
Networks (CNNs)  
CNN for Text  
Sentiment Analysis

## Notes and Further Reading

## Required

- [Fos19, Chapter 2] (CNNs for images)
- [LHH19, Chapter 7] (CNNs for text)



- [Fos19] David Foster.  
*Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play.*  
O'Reilly, 2019.  
<https://concordiauniversity.on.worldcat.org/oclc/1136155457>.
- [LHH19] Hobson Lane, Cole Howard, and Hannes Max Hapke.  
*Natural Language Processing in Action.*  
Manning Publications Co., 2019.  
<https://concordiauniversity.on.worldcat.org/oclc/1102387045>.