#cocoaheadsle

CoreML

Eine *kleine* Einführung

Max Langer,
Developmint GbR

Agenda

Theorie: Models, Layer

Praxis: Konvertierung, Tools

Demo: Nutzung in macOS & iOS Apps

Pitfalls

Einführung

aka

Worum geht es heute?

Worum geht es heute?

Eingabe: Text, Audio, Bilder etc.

Ausgabe: Antworten

Hat: Text, Audio und Bilder

Sucht: Antworten

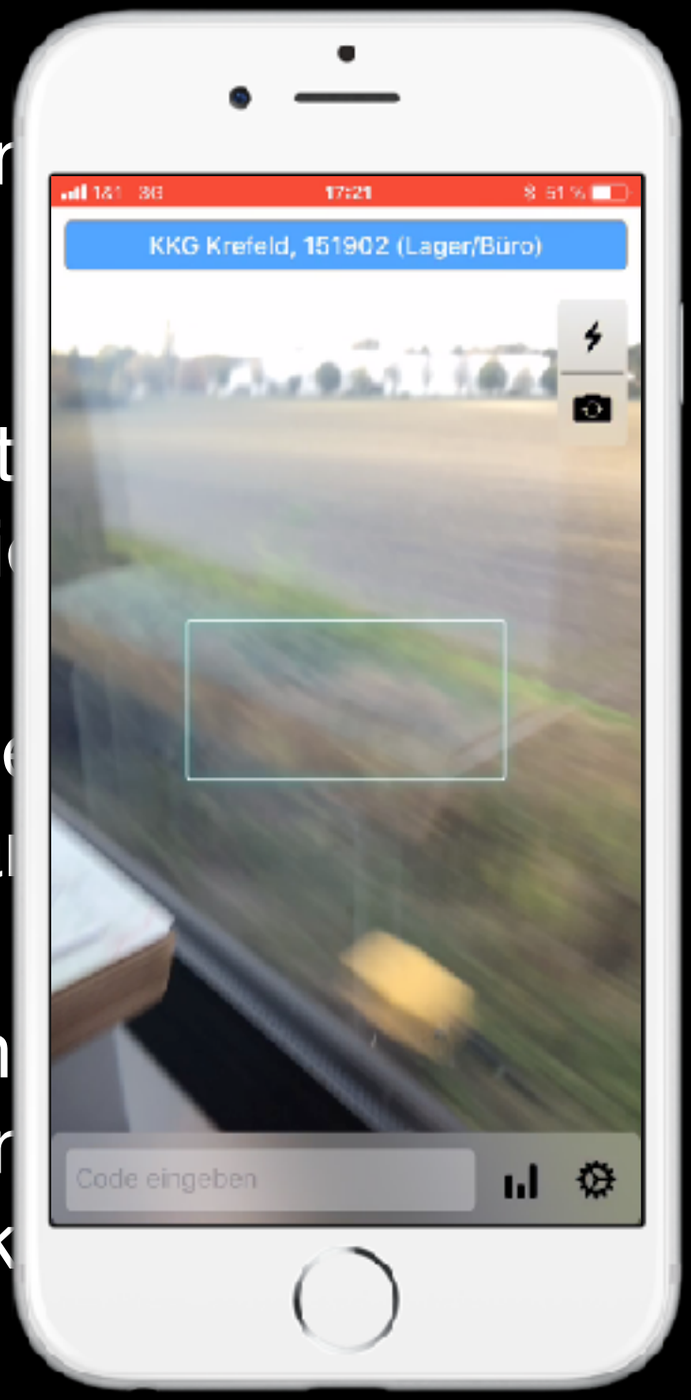
Bevor wir beginnen..

Je nach Model under
neue Möglichkeiten

Sehr genau, übertrifft
bisherige Implementi

Oft sehr schnell und
→ Ideal für Echtzeita

Aktuell: Vorteil gegen
Konkurrenzprodukter
Buzzwords und Mark



Die Theorie

Overfitting

Convergence

Big Data

Convolutional Neural Networks

Artificial Intelligence

Frameworks

Classifier

Layer

Machine Learning

Deep Learning

Approximates

Model

Training

Backpropagation

Learning

Neural Networks

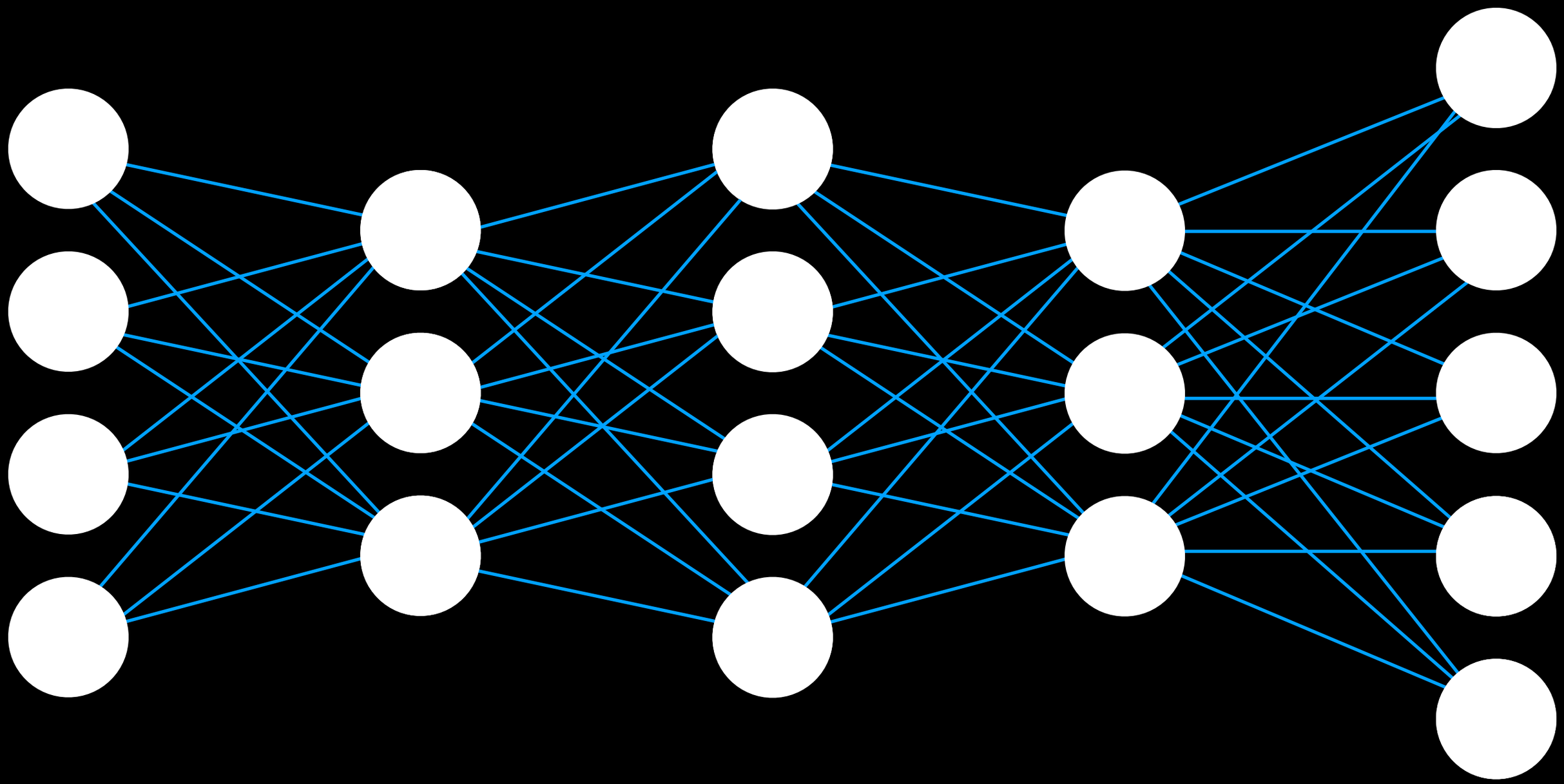
Was sieht das neurale Netz?

30				70
0	15	70		30
0	30			70
0	15	70		30
30				70

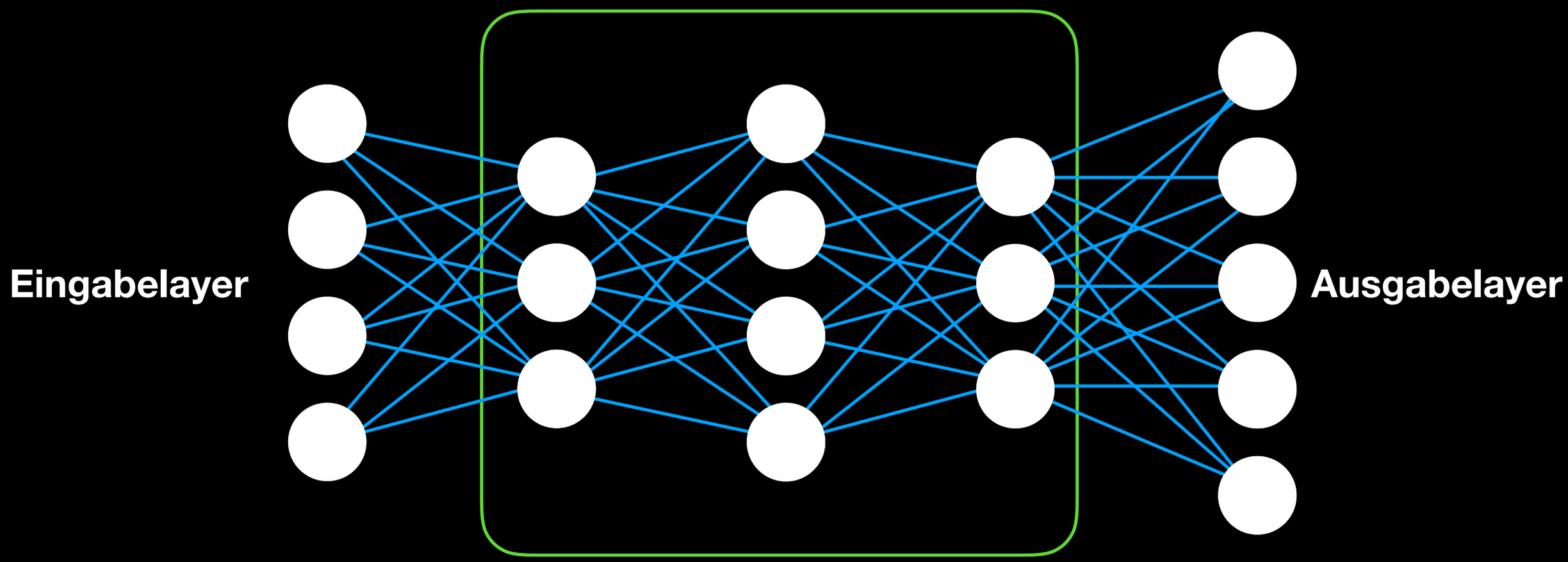
1 Pixel
=
1 Neuron

Monochromes Bild

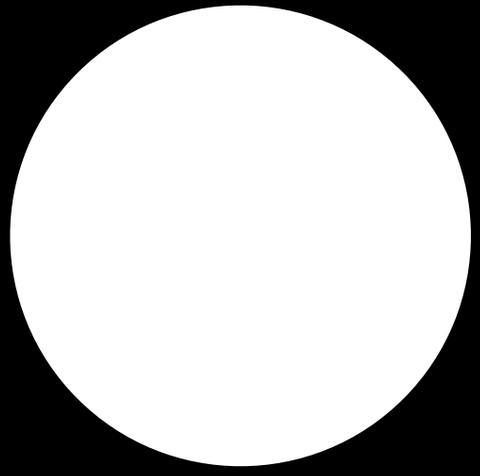
Neurale Netze



Neurale Netze



Neuron



Einzelnes Element des Netzes

Mit jedem Neuron aus den nebenliegenden Layern verbunden

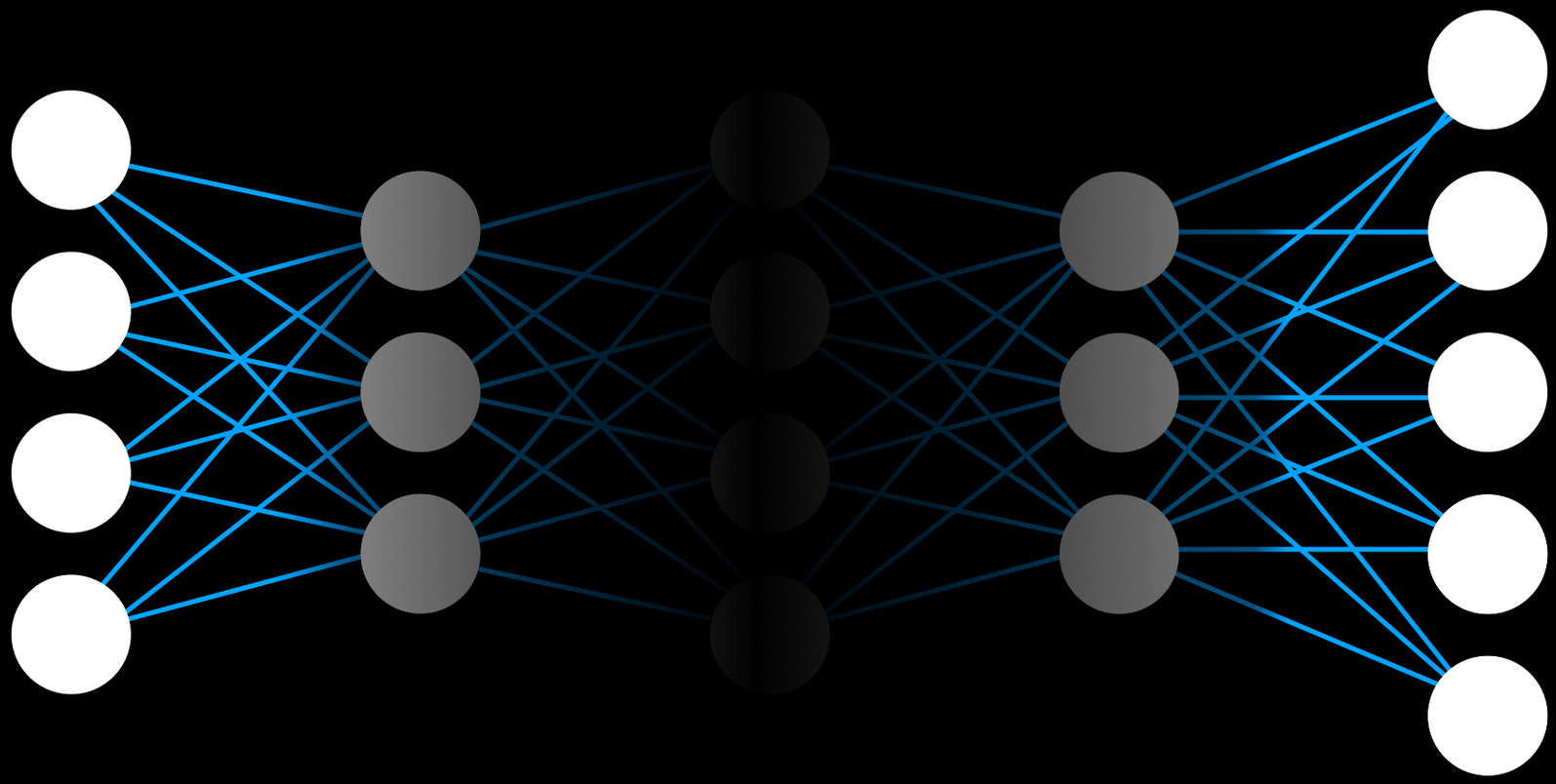
Enthält Wert zwischen 0 und 1
→ *Activation*

Ist der Wert hoch, ist das Neuron aktiviert

Neurale Netze

Eingabe
= Bild

1 Pixel
=
1 Neuron



Ergebnis

1 Neuron pro
Möglichkeit

'0'-'9'
=
10 Neuronen

Layers

Bilden „Zwischenschritte“ des Netzes

Mehrere Arten:

Convolutional Layers

Pooling Layers

weitere..

Convolutional Layers

Definieren eine Matrix, die bestimmte Eigenschaften aus dem Input filtert

Neuronen an bestimmten Stellen werden aktiviert, lassen auf eine Eigenschaft des Bildes Schlussfolgern

Z.B. :

- Farben
- Formen (Kanten, Bögen etc.)
- Rauschfilter

Convolutional Layers

Stride: 1

30	255	255	255	70
0	15	70	255	30
0	30	255	255	70
0	15	70	255	30
30	255	255	255	70

Original

„x“

1	1	1
0	0	1
0	0	1

Gewichtung (Filtermatrix)

=

865	1275	480
410	860	455
410	1050	680

Ergebnis

Convolutional Layers

Stride: 1 **Padding: 1**

0	0	0	0	0	0	0
0	30	255	255	255	70	0
0	0	15	70	255	30	0
0	0	30	255	255	70	0
0	0	15	70	255	30	0
0	30	255	255	255	70	0
0	0	0	0	0	0	0

Original

„x“

1	1	1
0	0	1
0	0	1

Gewichtung (Filtermatrix)

=

270	325	510	100	0
330	865	1275	480	325
60	410	860	455	285
300	410	1050	680	325
270	340	595	425	285

Ergebnis

Pooling Layers

Verkleinern den Input vor der Weiterverarbeitung

Verschiedene Algorithmen:

- Max-Pooling
- Average Pooling..

Pooling Layer

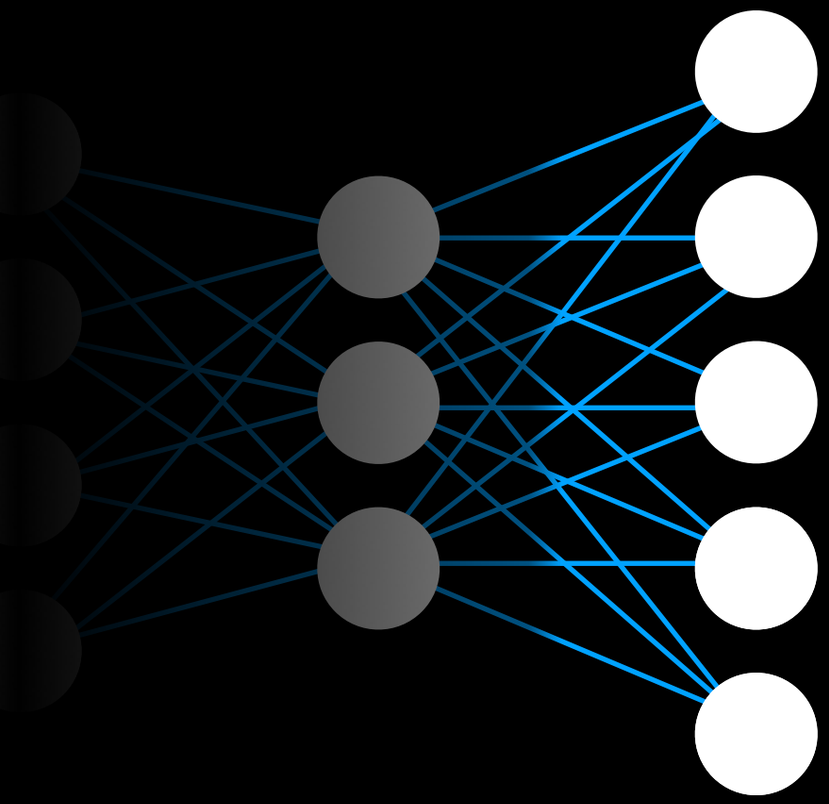
Stride: 2 **Poolgröße: 2**

310	255	255	255
0	15	70	255
0	30	255	255
0	15	70	255

→

310	255
30	255

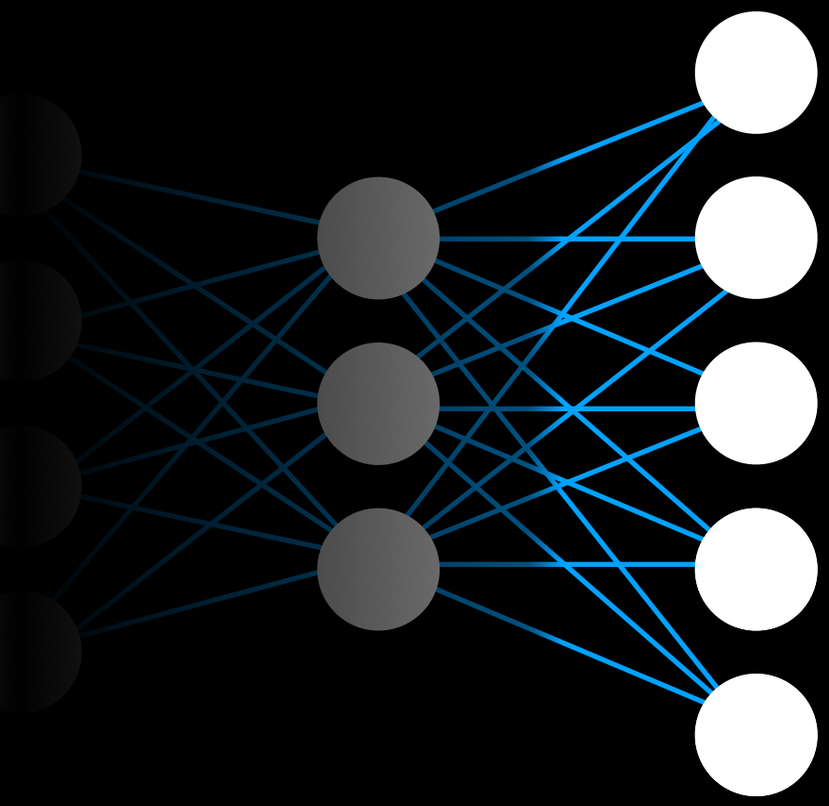
Das Training



Jede Gewichtung liegt zwischen 0 und 1

Wert bestimmt, wie stark der Einfluss eines Neurons auf das nächste ist

Das Training



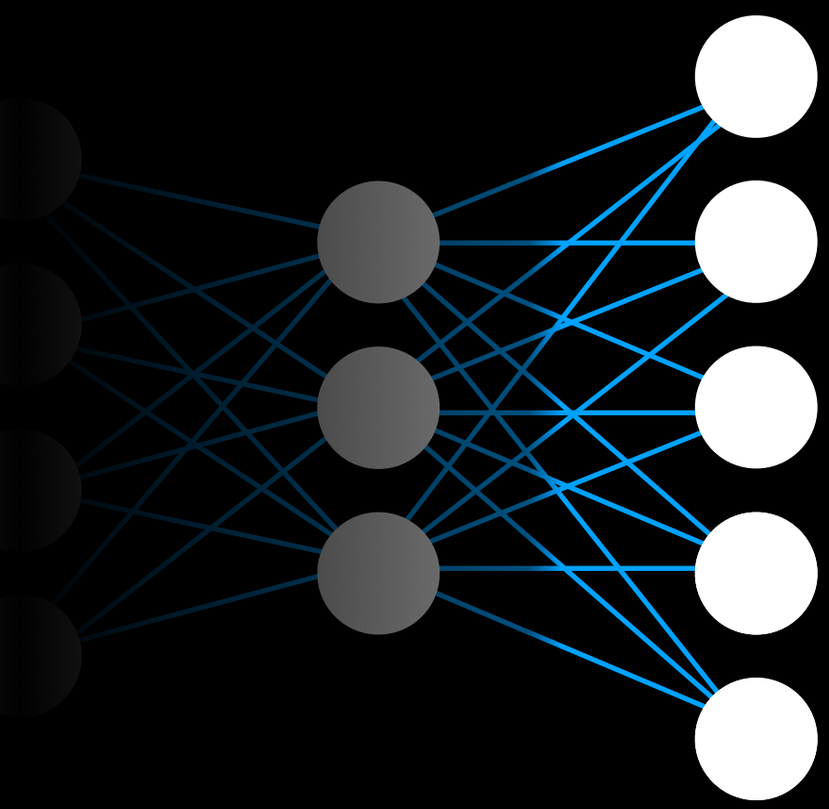
Viele Tausend Gewichtungen =
Viele Tausend Parameter

Manuelle Festlegung praktisch unmöglich

Lösung: Maschinelles Trial & Error

Das Training

Ganz grob



① Netz startet mit irgendwelchen Anfangsgewichtungen

② Man gibt ihm ein Bild, guckt wie weit man vom Erwartungswert entfernt ist

③ Gewichtungen anpassen

④ Wiederhole ab ②

Die Praxis

Workflow für macOS / iOS

Modell :

Definieren - Meist in Python, mittels Frameworks

Trainieren - ebenfalls Python

In .mlmodel (CoreML-Modell) konvertieren

In der App einbinden (Swift / Objective-C)

Frameworks

Schnittstelle für Programmierer, um abstrahiert mit NNs zu arbeiten

Bieten fertige Layer, kann das Modell mit Gewichtungen persistieren

Verschiedene Frameworks für versch. ML-Anwendungen erhältlich

Keras

Sehr leistungsstarker Wrapper für TensorFlow
(ML-Framework von Google)

Bietet leichten Einstieg in die Materie

Pythonbasiert

Keras

```
model = Sequential()
model.add(Dense(32, activation='relu', input_dim=100))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Generate dummy data
import numpy as np
data = np.random.random((1000, 100))
labels = np.random.randint(2, size=(1000, 1))

# Train the model, iterating on the data in batches of 32 samples
model.fit(data, labels, epochs=10, batch_size=32)
```

Model definieren

Trainingsdaten generieren

Model trainieren

Training

Das Modell braucht beschriftete Daten
→ Der/Die Programmierer/in trainiert also mit Daten, deren Antwort ihm/ihr bekannt sind

Oft große Datenmengen nötig (Facebook, Google, Unis)

Im Fall von OCR aber einfach → Captcha Generator

coremltools

Open Source tool bundle von Apple

Kann *ausgewählte* Framework-Models konvertieren

Da quelloffen, auch erweiterbar. Wird stetig weiterentwickelt

Entwickler sind sehr froh über Feedback und Requests
(siehe forums.developer.apple.com)

Konvertierung zu CoreML

```
import coremltools

# Load Keras Model
model = load_model('ocr.h5', custom_objects={'function': function})

# Convert the Model
coreml_model = coremltools.converters.keras.convert(model,
                                                    input_names='data',
                                                    image_input_names='data',
                                                    image_scale=1/255.0,
                                                    output_names='prediction')
# Metadata
coreml_model.author = 'Max Langer'
coreml_model.licence = 'MIT'
coreml_model.short_description = 'A simple Image to Number classifier'

# Export the Model
coreml_model.save('OCR.mlmodel')
```

OCR.mlmodel

Demo

Aber was ist mit der echten Welt?

Demo zeigt Idealbedingungen

Input von Kamera verrauscht, unscharf etc.

→ Benötigt robustes, schnell arbeitendes Netz

Möglichkeit ganze Strings zu erkennen anstelle einzelner Character/Digits (aufwendigeres Training)

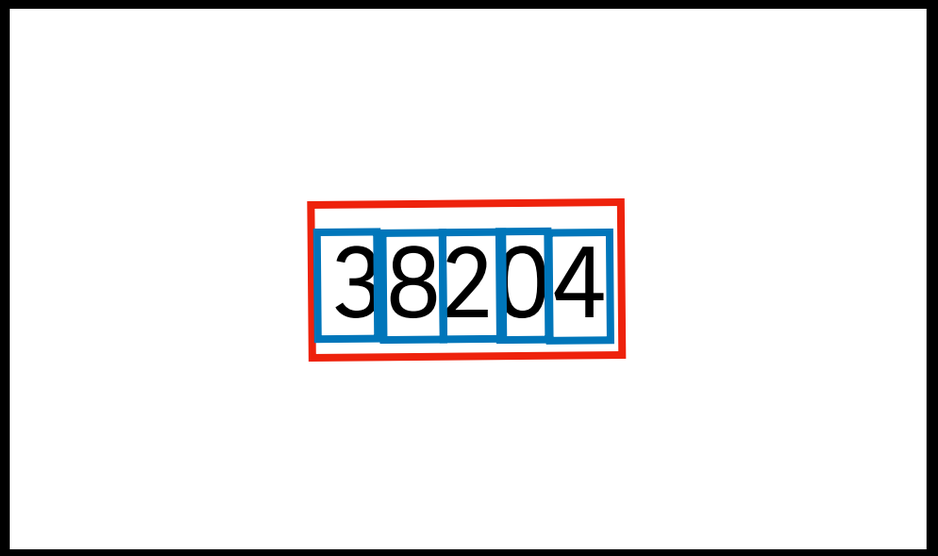
Und wie?
Dinge aus Bildern extrahieren

Lösung: Apple liefert das Vision Framework

Bietet OCR

„Zeigt, wo was ist, aber nicht was es ist“

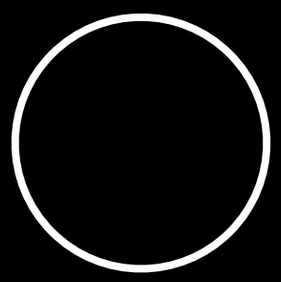
Auch für Objekte, Landschaften, Gesichter etc..



Pitfalls

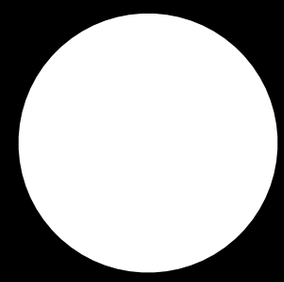
Oder: worüber ich gestolpert bin

Farbraum ist nicht gleich Farbraum



Schwarz

0	**-**	**1**
-1	**-**	**1**
255	**-**	**0**
1	**-**	**0**
1	**-**	**-1**



Weiß

Pitfalls

Oder: worüber ich gestolpert bin

Höhe x Breite vs Breite x Höhe

Per Keras Definition:

Kanal/Höhe/Breite oder Höhe/Breite/Kanal

Im verwendeten Model aber Breite/Höhe/Kanal, weil besser? 🤔

Bild muss vor Berechnung entsprechend transformiert werden

Nicht wirklich Pitfall

Aber trotzdem zu beachten

Größe und Farbraum des Inputs

Der Input muss genau den Erwartungen des Models entsprechen (Höhe, Breite, Anzahl der Kanäle)

Bilder/Pixelbuffer können entsprechend umgewandelt werden

In diesem Schritt lassen sich die Pitfalls von eben gut einbinden

Nachteile

On Device:

Keine Services verfügbar
→ Man muss alles selber machen

Größe des Modells bläht evtl. App-Bundle auf

Vorteile

On Device:

Nutzt Apples advanced Hard- und Software, Neural Engine etc.
→ sehr schnell

Leicht zu Testen

Privacy ✅

Zusammenfassung

Riesiges Thema

Viel zu entdecken

Viele Möglichkeiten

Weiterführende Materialien

Introducing CoreML	At Home	Whenever you want
Natural Language Processing and your Apps	Hall 3	Wednesday 9:00AM
Vision Framework: Building on Core ML	Hall 2	Wednesday 3:10PM
Core ML in depth	Hall 3	Thursday 9:00AM
Accelerate and Sparse Solvers	Executive Ballroom	Thursday 10:00AM
Using Metal 2 for Compute	Grand Ballroom A	Thursday 4:10PM

Fragen?

Fragen!

Vielen Dank