Praktikum: Selbstlernende Systeme

Convolutional Neural Network
Deep Q-Learning

20.11.2019

Universität Augsburg
Institut für Informatik
Lehrstuhl für Organic Computing

Gliederung

ORGANIC COMPUTING

1. Convolutional Neural Network

2. Imporvements for DQN

3. Quellen

Convolutional Neural Network

Bilderkennung

What We See

08 02 22 97 38 15 00 40 00 75 04 05 07 78 52 12 50 77 91 08
49 49 99 40 17 81 18 57 60 87 17 40 98 43 69 48 04 56 62 00
81 49 31 73 55 79 14 29 93 71 40 67 53 88 30 03 49 13 36 65
52 70 95 23 04 60 11 42 69 24 68 56 01 32 56 71 37 02 36 91
22 31 16 71 51 67 63 89 41 92 36 54 22 40 40 28 66 33 13 80
24 47 32 60 99 03 45 02 44 75 33 53 78 36 84 20 35 17 12 50
32 98 81 28 64 23 67 10 26 38 40 67 59 54 70 66 18 38 64 70
67 26 20 68 02 62 12 20 95 63 94 39 63 08 40 91 66 49 94 21
24 55 58 05 66 73 99 26 97 17 78 78 96 83 14 88 34 89 63 72
21 36 23 09 75 00 76 44 20 45 35 14 00 61 33 97 34 31 33 95
78 17 53 28 22 75 31 67 15 94 03 80 04 62 16 14 09 53 56 92
16 39 05 42 96 35 31 47 55 58 88 24 00 17 54 24 36 29 85 57
86 56 00 48 35 71 89 07 05 44 44 37 44 60 21 58 51 54 17 58
19 80 81 68 05 94 47 69 28 73 92 13 86 52 17 77 04 89 55 40
04 52 08 83 97 35 99 16 07 97 57 32 16 26 26 79 33 27 98 66
88 36 68 87 57 62 20 72 03 46 33 67 46 55 12 32 63 93 53 69
04 42 16 73 38 25 39 11 24 94 72 18 08 46 29 32 40 62 76 36
20 69 36 41 72 30 23 88 34 62 99 69 82 67 59 85 74 04 36 16
20 73 35 29 78 31 90 01 74 31 49 71 48 86 81 16 23 57 05 54
01 70 54 71 83 51 54 69 16 92 33 48 61 43 52 01 89 19 67 48

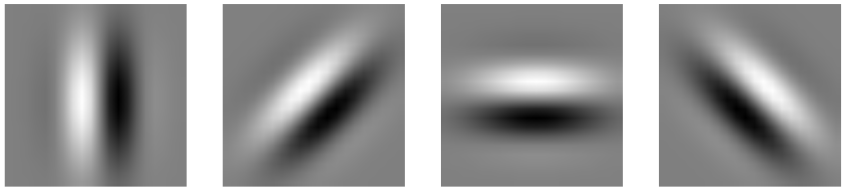
What Computers See

- Input: $32 \times 32 \times 3$ (x-Achse $\times$ y-Achse $\times$ RGB)
- Werte zwischen 0 und 255
- Output: 80% Hund, 20% Katze

Features

1. Fell
2. Ohren
3. 4 Pfoten

1. Ecken
2. Kanten
3. Bögen

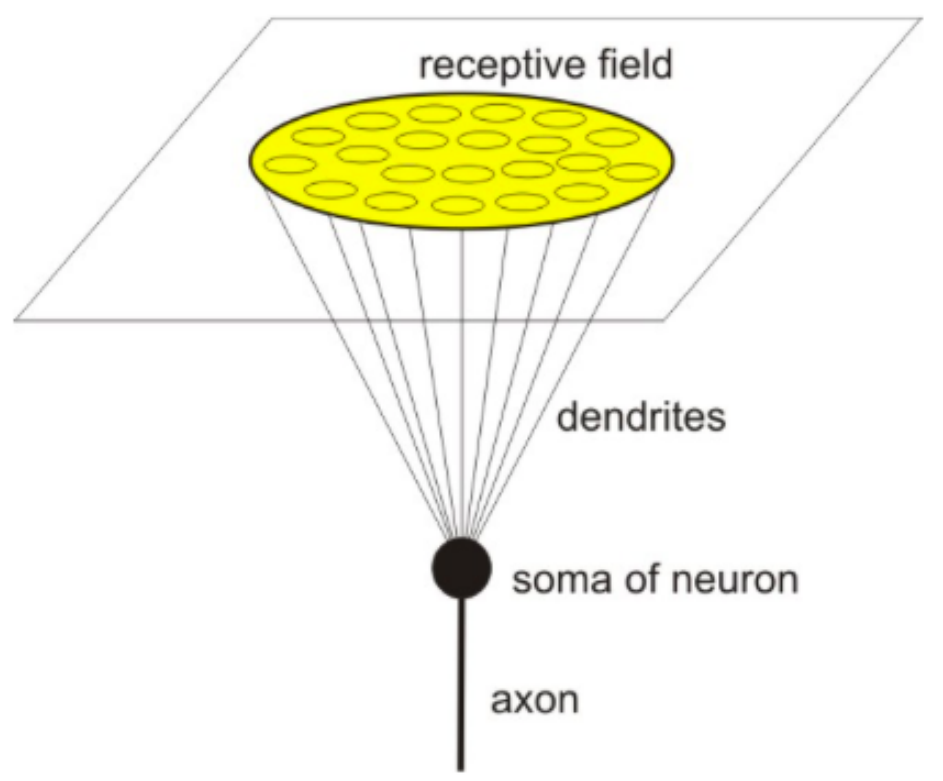


Biologische Inspiration

ORGANIC COMPUTING

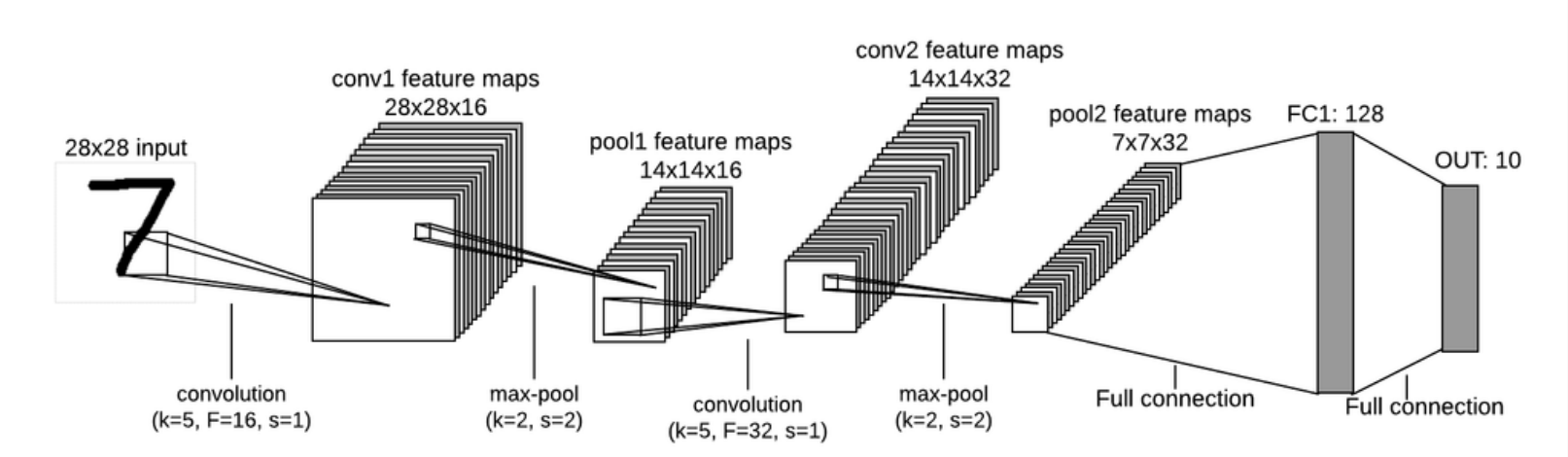
- Forschung von Hubel und Wiesel am Gehirn von Säugetieren
- *simple cells (S cells)*: grundlegende Formen in bestimmtem Bereich und Winkel
- *complex cells (C cells)*: größere receptive fields $\rightarrow$ nicht beschränkt auf spezifische Position (flexibler)

receptive field



- Region auf der Retina
- beeinflusst das Feuern des zugehörigen Neurons
- Jedes Sensor-Neuron hat das gleiche receptive field
- einzelne fields überlappen sich

Architektur



2 Komponenten:

1. Hidden layers/Feature extraction: Mehrere convolutions (Faltungen) und pooling Operationen $\rightarrow$ Features werden erkannt
2. Klassifikation: fully connected layers dienen als Classifier auf den extrahierten Features

Feature Extraction

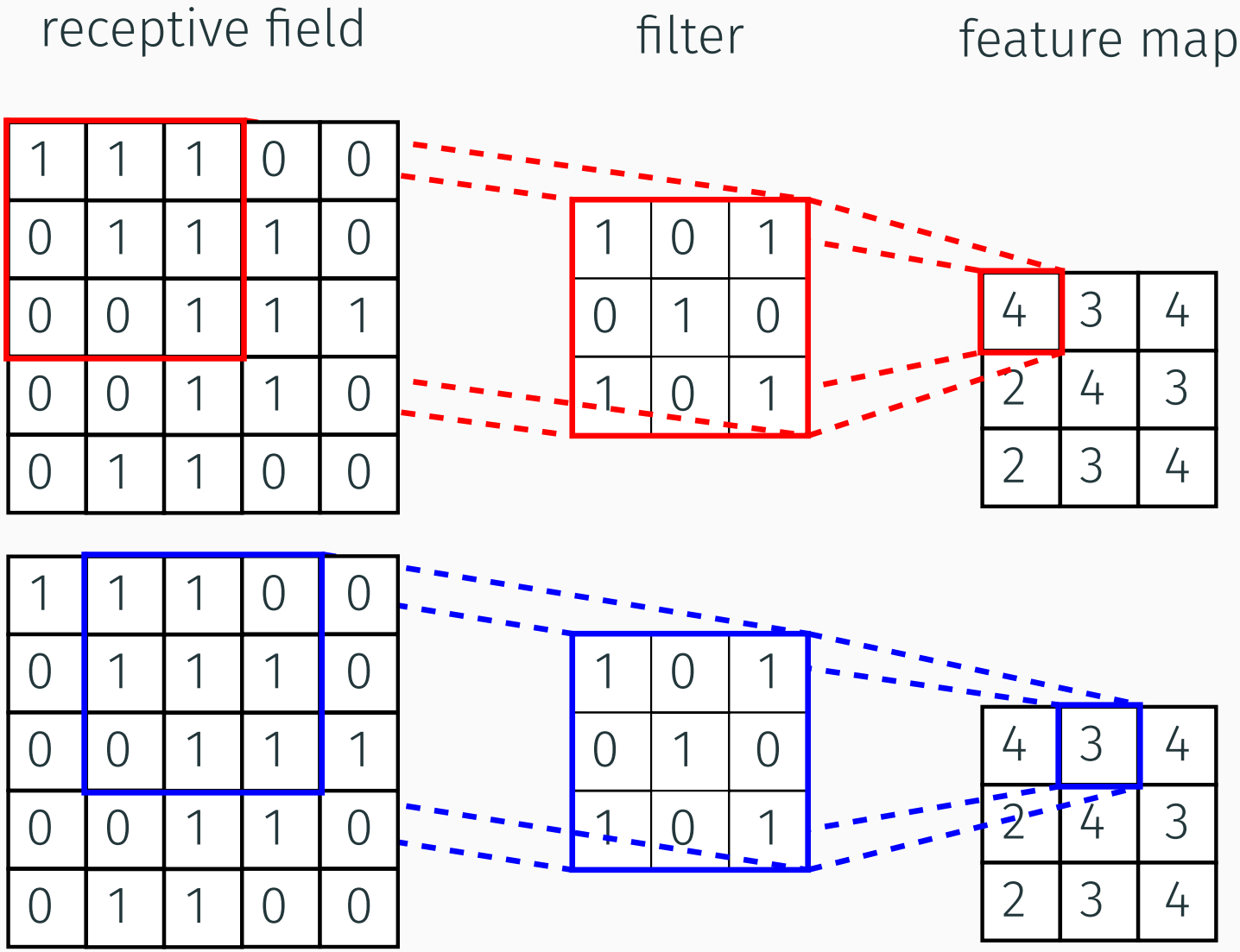
ORGANIC COMPUTING

Convolution

mathematische Kombination von zwei Funktionen die eine dritte Funktion erschafft. Zwei Informationssets werden zusammengefasst.

- Convolution auf den Input-Daten mittels Kernel/Filter
- Ergebnis: Feature Map
- Der Filter wandert über den Input und führt in jedem Schritt eine Matrixmultiplikation durch

Convolution



Filter

ORGANIC COMPUTING

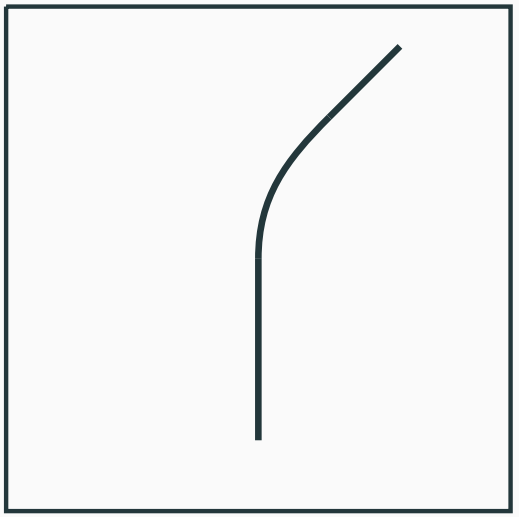
- Array aus Zahlen
- Parameter oder Gewichte (Zahlen)
- Tiefe des Filters muss der des Inputs entsprechen
- gleitet/convolved über den Input
- Elementweise Multiplikation

High Level Perspective

ORGANIC COMPUTING

- Feature Identifier
- Ecken, einfache Farben und Kurven

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0



ORGANIC COMPUTING

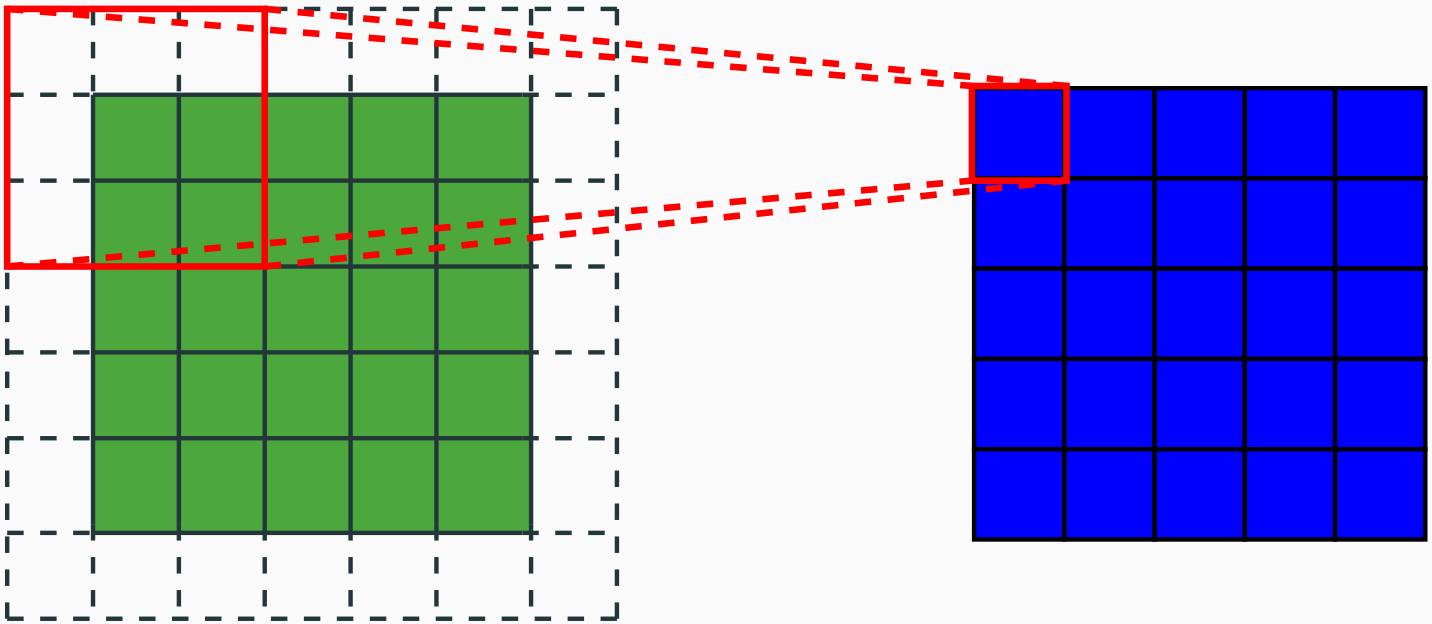
Stride

= Schrittgröße mit der der Filter sich über den Input bewegt. Normalerweise 1, d.h. Pixel für Pixel. Ein größerer Stride führt zu weniger Überlappung.

Padding

Eine Schicht mit Nullen wird um den Input gelegt. Das führt dazu, dass die Feature Map nicht kleiner wird als der Input. Außerdem stellt Padding sicher, dass der Filter sowie der Stride in den Input passen.

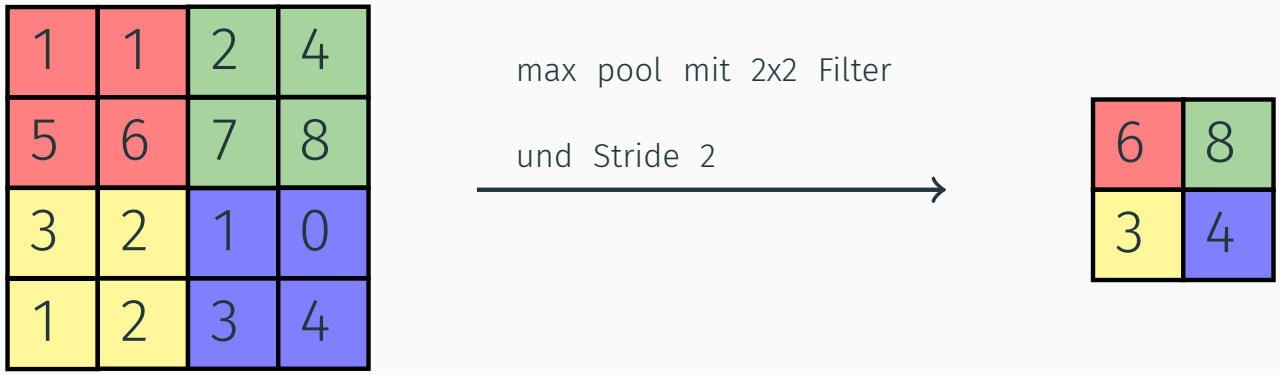
Padding



Pooling

ORGANIC COMPUTING

- zwischen Convolution-Layern
- reduziert Dimensionalität $\rightarrow$ weniger Parameter
- reduziert Trainingszeit
- wirkt Overfitting entgegen



Klassifikation

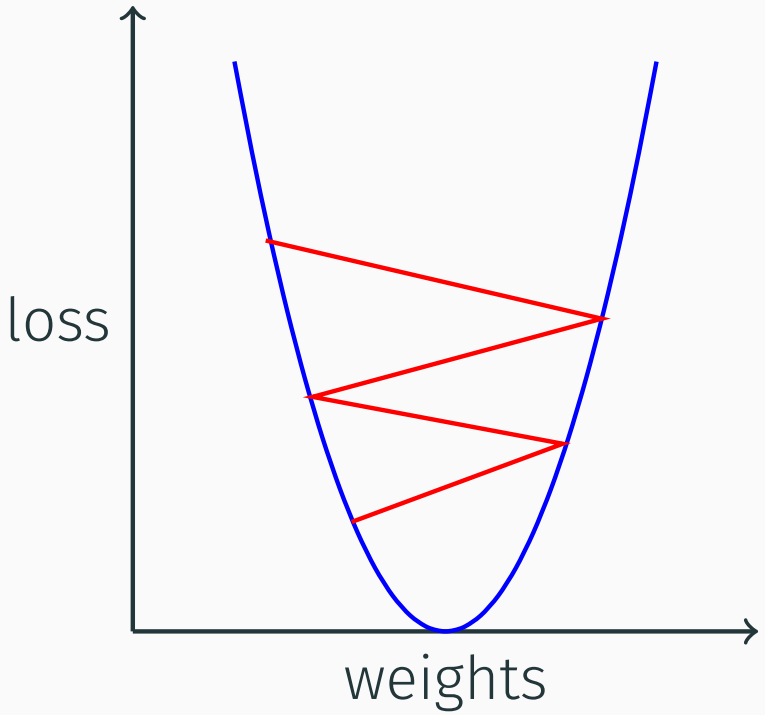
ORGANIC COMPUTING

- Nach der Convolution folgen Fully Connected Layer
- Convolution erzeugt 3-dimensionale Daten (mehrere Layer)
- FC-Layer aktzeptieren nur 1-dimensionale Daten

$\Rightarrow$ Flatten

Learningrate

- bestimmt die Größe der Schritte beim Gewichtsupdate
- größere Learningrate → Model konvergiert schneller
- zu große Learningrate → overshooten des optimalen Punkts



keras

ORGANIC COMPUTING

```
1  from keras.layers import Dense, Conv2D, Flatten
2  from keras.models import Sequnetial
3  from keras.optimizers import RMSprop
4
5  model = Sequential()
6
7  # 1st Convolutional Layer
8  model.add(Conv2D(16, 5, strides=1, activation='relu', input_shape=(32, 32, 2),
       kernel_initializer='he_normal', padding='same'))
9  # 2nd Convolutional Layer
10 model.add(Conv2D(32, 1, strides=1, activation='relu', kernel_initializer='he_normal',
       padding='same'))
11
12 # Flatten-Function
13 model.add(Flatten())
14
15 # Fully Conected Layer
16 model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
17 model.add(Dense(8, activation='linear', kernel_initializer='he_normal'))
18
19 model.compile(loss='mse',optimizer=RMSprop(lr=0,00025))
```

Imporvements for DQN

Double Learning

ORGANIC COMPUTING

- Agent tendiert zur Überschätzung des Q-Values

$$Q(s,a) \rightarrow r + \gamma \max_{a} Q(s',a)$$

- Beispiel:
  - alle Aktionen haben denselben echten Q-Wert für s
  - Schätzung ist von Natur aus verrauscht $\rightarrow$ unterscheidet sich von Q_{true}
  - max $\rightarrow$ die Aktion mit größtem positiven Fehler wird ausgewählt
  - Fehler wird zu zukünftigen States mitgenommen
  - $\Rightarrow$ positiver Bias/Value Overestimation

Double Learning

ORGANIC COMPUTING

- Lösung: Double Learning
- Standard Q-Learning:
  - zwei Q-Functions Q_1 und Q_2 werden unabhängig gelernt
  - eine wählt die zu maximierende Aktion aus
  - die andere bestimmt den Wert der ausgewählten Aktion
  - Zufällig wird eine der beiden geupdatet:

$$Q_1(s,a) \rightarrow r + \gamma Q_2(s', \operatorname*{argmax}_a Q_1(s',a))$$

oder

$$Q_2(s,a) \rightarrow r + \gamma Q_1(s', \operatorname*{argmax}_a Q_2(s',a))$$

Double Deep Q-Learning

ORGANIC COMPUTING

- bereits zwei unterschiedliche Q-Funktionen
- $Q(s, a; \theta)$
- $Q(s, a; \theta^-)$

$$Q(s, a; \theta) \rightarrow r + \gamma Q(s', \underset{a}{\text{argmax}}\, Q(s', a; \theta); \theta^-)$$

- Stabilität $\uparrow \rightarrow$ komplexere Probleme

Prioritized Experience Replay

ORGANIC COMPUTING

- **Bisher:** Alle Erinnerungen werden gleich behandelt
- **Aber:** Von manchen kann man mehr lernen als von anderen
- **Idee:** Bevorzugen derjenigen Übergänge, welche am schlechtesten zur aktuellen Schätzung der Q-Function passen
- $\rightarrow$ größtes Potential

Prioritized Replay Memory

ORGANIC COMPUTING

- Fehler von Sample $S = (s, a, r, s')$ ist die Distanz zw. $Q(s, a)$ und seinem Target $T(S)$

$$error = |Q(S, a) - T(S)|$$

- Der Fehler wird zusammen mit S gespeichert und bei jedem Lernschritt geupdatet

$\rightarrow$ Greedy Prioritization

- Samples mit kleinem TD-Fehler beim ersten Auftreten haben eine sehr kleine Wahrscheinlichkeit je wieder gesampelt zu werden

Stochastic Prioritization

ORGANIC COMPUTING

- Priorität:

$$p_i = \frac{1}{rank(i)}$$

- *rank*: Position im nach TD-Fehler sortierten Buffer
- Priorität wird in Wahrscheinlichkeit umgewandelt:

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha}$$

- α: bestimmt wie viel Priorisierung benutzt wird $\alpha = 0 \rightarrow$ Standardfall ($0 \leq \alpha \leq 1$)

Proportional Prioritization

ORGANIC COMPUTING

- Fehler $\rightarrow$ Priorität:

$$p_i = (error_i + \epsilon)^\alpha$$

- ϵ: kleine positive Konstante $\rightarrow$ keine Priorität von 0
- α: bestimmt wie viel Priorisierung benutzt wird $\alpha = 0 \rightarrow$ Standardfall ($0 \leq \alpha \leq 1$)
- Priorität wird in Wahrscheinlichkeit umgewandelt:

$$P(i) = \frac{p_i}{\sum_k p_k}$$

Annealing Bias

ORGANIC COMPUTING

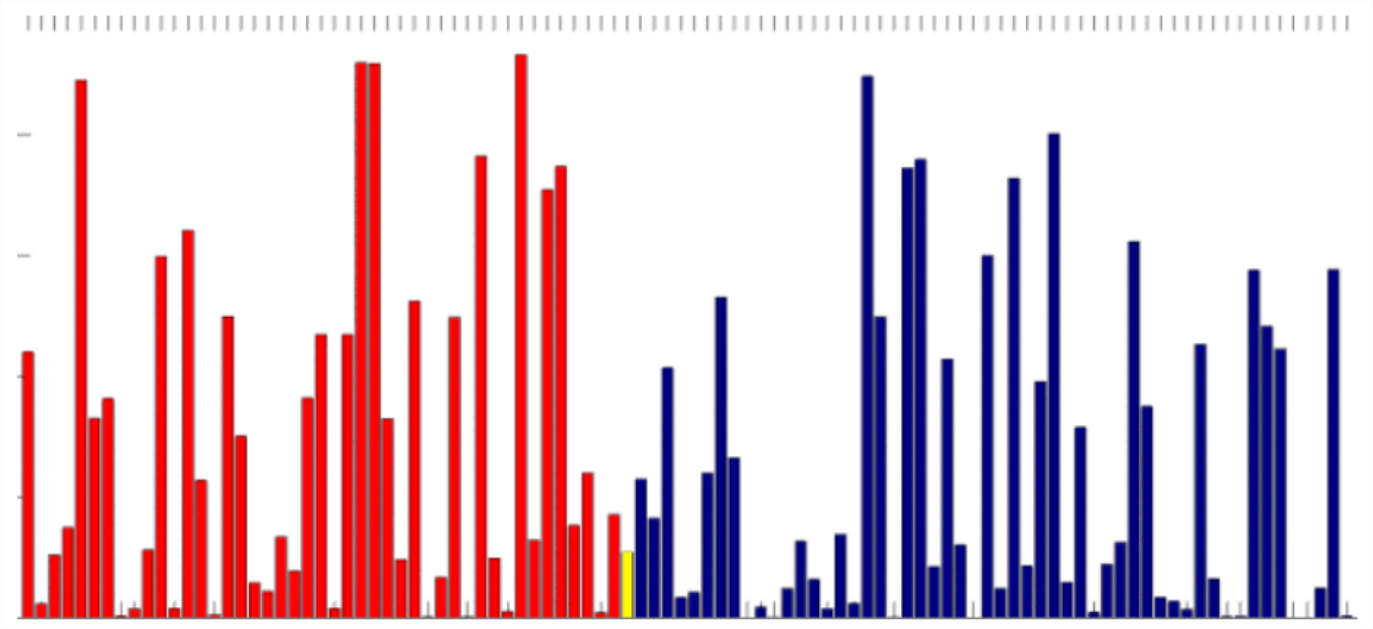
- unkontrollierte Veränderung der Verteilung $\rightarrow$ induziert Bias
- Lösung: Weighted Importance Sampling

$$w_i = \left(\frac{1}{N} * \frac{1}{P(i)}\right)^{\beta}$$

- β: läuft linear gegen 1.
- Wenn $\beta = 1$, dann kompensieren die Gewichte vollkommen die ungleichmäßige Verteilung $P(i)$

Samplen

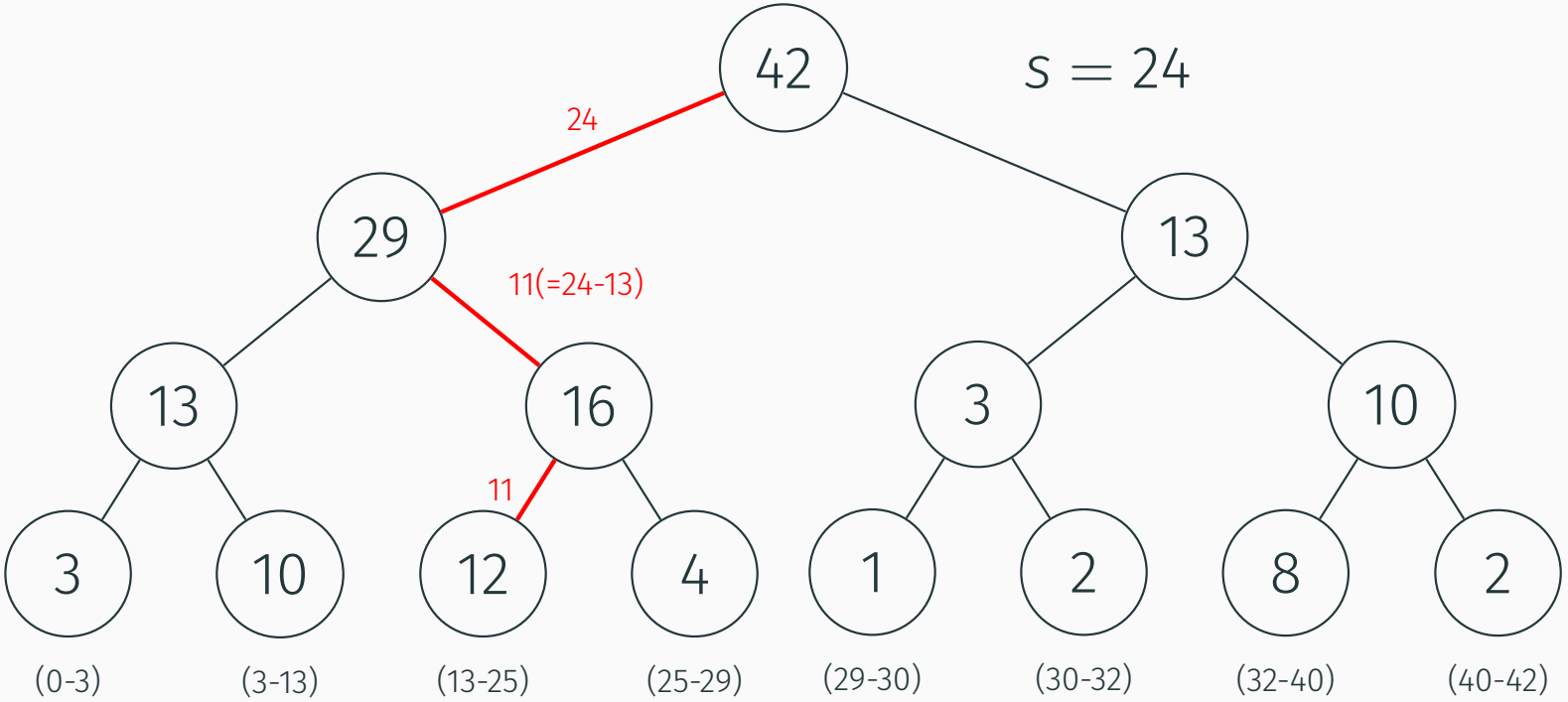
- ziehe Zufallszahl s, $0 \leq s \leq \sum_k p_k$
- durchlaufe Memory von Links nach Rechts
- summiere Prioritäten
- bis s erreicht



Laufzeit: $O(n)$

Sum Tree

- Unsorted Sum Tree = Binäre Baumstruktur
- Elternwert ist die Summe seiner Kinder
- Samples in Blattknoten



Laufzeit: $O(\log n)$

Advantage

ORGANIC COMPUTING

- Q-Value: Qualität von Aktion a in State s

$$Q(s, a) = V(s) + A(s, a)$$

- $V(s)$: Wie gut ist es in State s zu sein
- $A(s, a)$: Wie viel besser ist es Aktion a zu wählen im Vergleich zu allen anderen (Advantage)

Dueling DQN

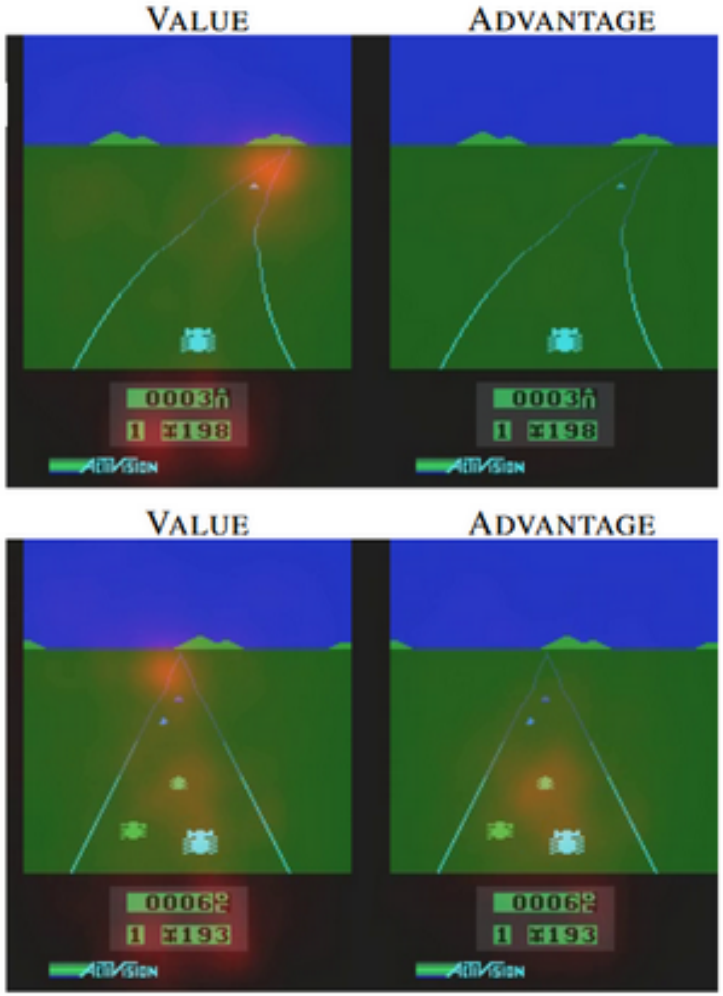
ORGANIC COMPUTING

- Netzwerk berechnet $V(s)$ und $A(s, a)$ separat und kombiniert sie zu $Q(s, a)$
- Agent interessiert sich nicht unbedingt für beide Werte
- Warum die Werte für alle Aktionen berechnen, wenn alle in den Tod führen?
- $\Rightarrow$ robustere Schätzung von $V(s)$

Beispiel

ORGANIC COMPUTING

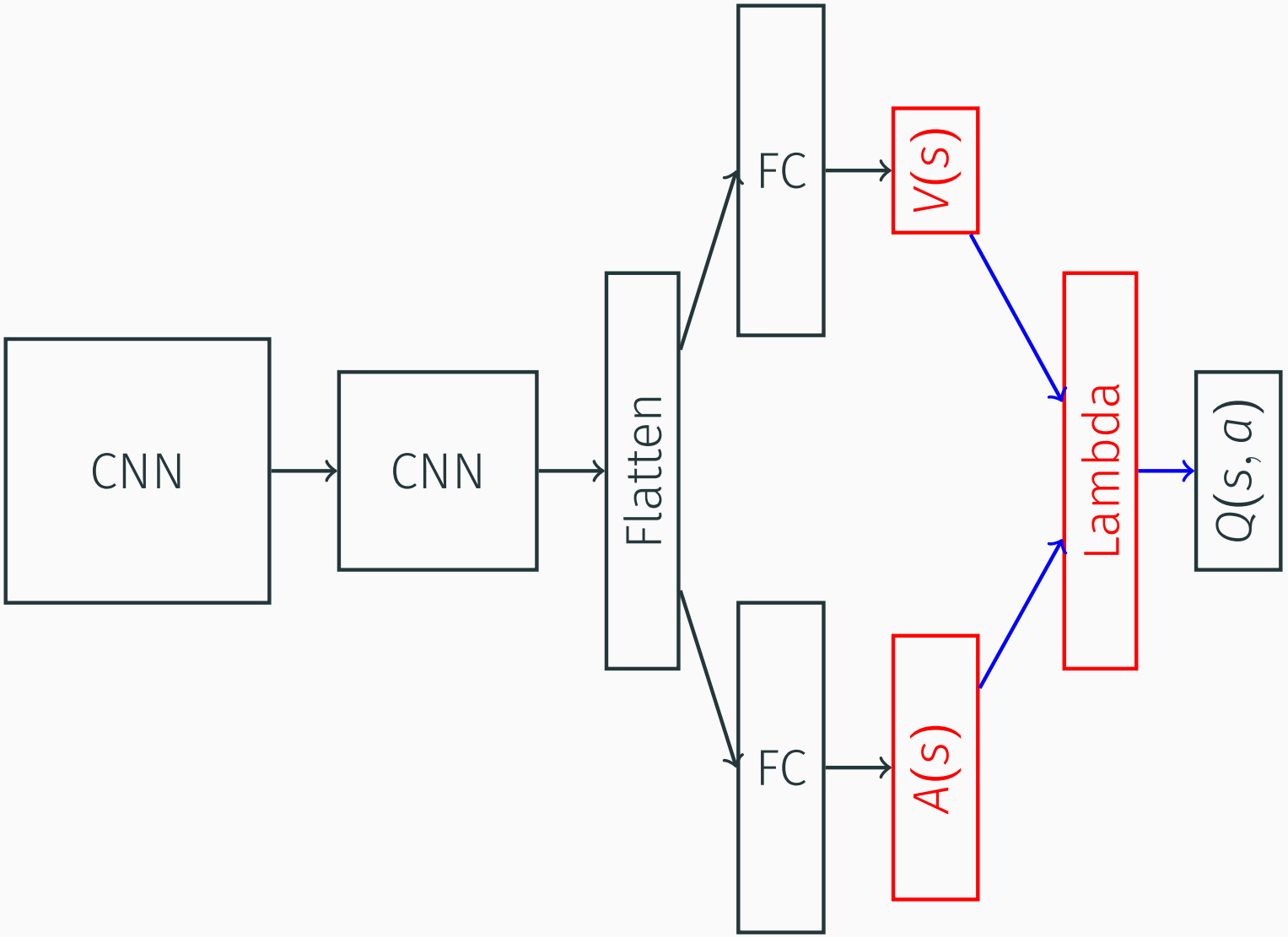
Focus on 2 things:
- The horizon where new cars appear
- On the score



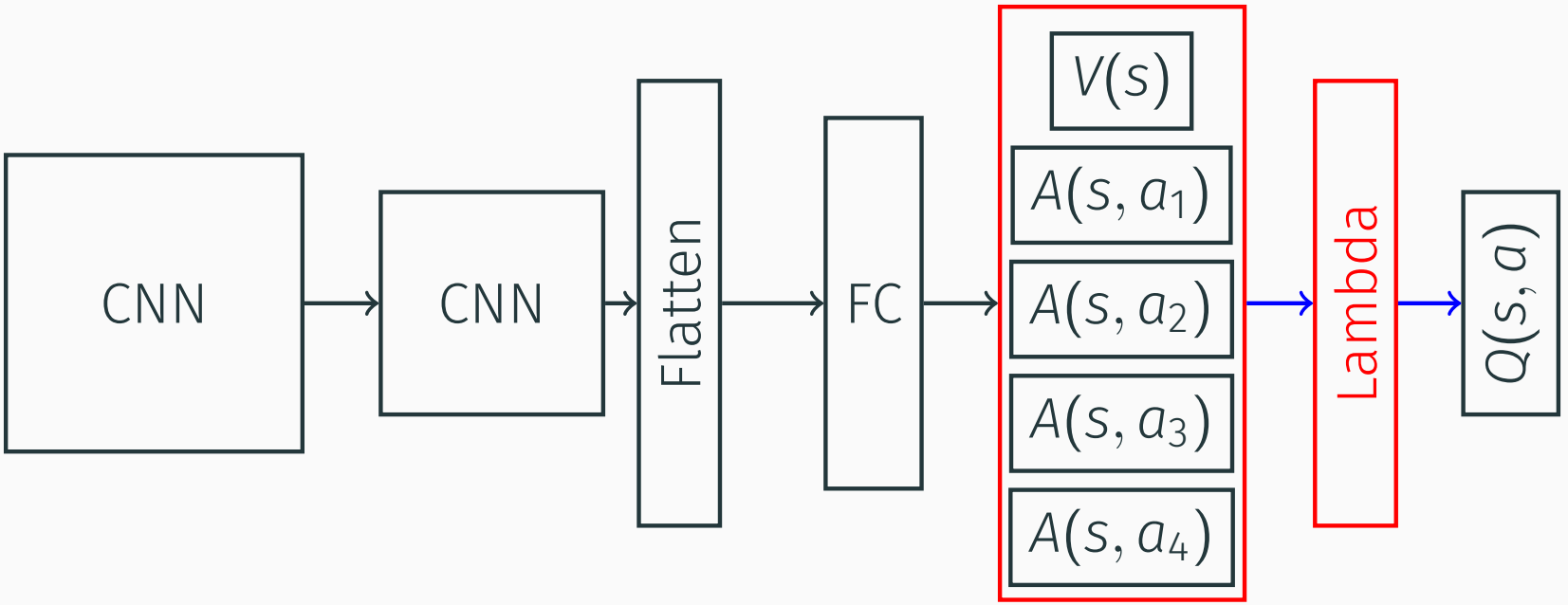
No car in front, **does not pay much attention because action choice making is not relevant**

Pays attention to the front car, in this case **choice making is crucial to survive**

Architektur



Architektur



Aggregation

ORGANIC COMPUTING

Naiver Ansatz:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; \theta, \alpha)$$

ABER: Problem der Identifizierbarkeit

$\Rightarrow$ Gegeben $Q(s, a)$ ist es nicht möglich $A(s, a)$ und $V(s)$ zu ermitteln

$\Rightarrow$ Problem für Backpropagation

Aggregation

ORGANIC COMPUTING

Lösung: zwingen der Advantage-Function auf 0 für die gewählte Aktion

- Durch Subtrahieren der durchschnittlichen Advantage aller Aktionen

$$Q(s, a; \alpha, \beta) = V(s, \theta, \beta) + \left(A(s, a; \theta, \alpha) - \frac{1}{A} \sum_{a'} A(s, a'; \theta, \alpha)\right)$$

- α: Parameter des Advantage-Streams
- β: Parameter des Value-Streams

Lamba-Layer

ORGANIC COMPUTING

- Keras-Layer
- zum Definieren von Custom-Layern

Beispiel:

```
1  from keras.layers import Lambda
2  from keras.models import Sequential
3  from keras import backend as K
4
5  model = Sequential()
6
7  model.add(Dense(16, activation='relu', input_dim=2))
8
9  # Berechnet das Quadrat
10 model.add(Lambda(lambda x: x ** 2))
11
12 # Ermittelt den Mittelwert
13 model.add(Lambda(lambda sq: K.mean(sq))
14
15 model.add(Dense(10))
16
17 # erste Stelle im Tensor minus die Summe aller nachfolgenden Stellen (9)
18 model.add(Lambda (lambda r: r[:, 0] – K.sum(r[:, 0:], axis=1), output_shape=(10, )))
```

Lambda-Layer Debuggen

ORGANIC COMPUTING

```
1  from keras.layers import Input, Dense
2  from keras.models import Model
3  from keras.optimizers import RMSProp
4  import keras.backend as K
5  import tensorflow as tf
6
7  # lambda Funktion
8  def lmd(prm):
9      # splittet den input in zwei tensoren mit shape=(None, 4)
10     x, a = tf.split(prm, [4,4], axis=1)
11     s = K.sum(x)
12     result = a - s
13
14     return [x, a, s, result]
15
16 # Netz erstellen
17 inL = Input(self.state_size)
18 h1 = Dense(8, activation='relu')(inL)
19 x, a, s, result = Lambda(lmd)(h1)
20
21 #Model erstellen
22 model = Model(input=inL, output=[x, a, s, result])
23 model.compile(loss='mse', optimizer=RMSProp(lr=0.0025))
24
25 # mit dem Netz predicten
26 x, a, s, result = model.predict(state)
```

Quellen

Bilder

ORGANIC COMPUTING

- https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
- https://view.stern.de/de/rubriken/tiere/hund-hunde-ohren-basset-hound-dumbo-basset-original-2012281.html
- https://www.quora.com/What-do-channels-refer-to-in-a-convolutional-neural-network
- https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050
- https://jaromiru.com/2016/11/07/lets-make-a-dqn-double-learning-and-prioritized-experience-replay/
- https://medium.freecodecamp.org/improvements-in-deep-q-learning-dueling-double-dqn-prioritized-experience-replay-and-fixed-58b130cc5682