Hochschule für angewandte Wissenschaften

Würzburg - Schweinfurt

Indoor Lokalisierung

mithilfe von Deep Learning

Verfasser: Dennis Grünwald

Matrikelnummer: 5015070

Abgabetermin: 11.07.2018

Inhalt

1 Einleitung

In komplexen Gebäuden, wie z. B. Flughäfen oder Krankenhäusern, kann es schwer werden sich ohne Hilfe zu orientieren. Besonders blinde Menschen Leiden aufgrund Ihrer Behinderung oft an Desorientierung, wodurch sie auf eine Navigation angewiesen sind. Ein Indoor Navigation System wäre hier die Lösung, jedoch weisen die heutigen Technologien Schwächen auf (ASSETS'17 2017).

Als Global Positioning System (GPS) wird ein System zur Positionsbestimmung und Navigation mit Hilfe von Satelliten bezeichnet. Diese umkreisen die Erde zweimal täglich in einer präzisen Umlaufbahn. Jeder Satellit überträgt ein Signal, das es GPS-Geräten in der Nähe ermöglicht, den genauen Standort der Sender zu entschlüsseln und diesen zu berechnen. GPS-Empfänger verwenden diese Informationen, um den genauen Standort eines Benutzers zu berechnen. Problematisch wird dieses Verfahren jedoch innerhalb Gebäuden. Obwohl es heutzutage sehr empfindliche GPS-Chips gibt, ist der resultierende Standort in der Regel nicht genau genug. Die Signale werden durch Dächer, Wände und anderen Objekten gedämpft. Auch kann die Fehlerreichweite vieler Chips größer sein als der Innenraum selbst.

Zurzeit bietet die Beacon-Technologie eine relativ gute Möglichkeit für Indoor Navigation. Diese wird vor allem von Apple genutzt. Mit Beacon wird ein Sender oder Empfänger bezeichnet, der auf der „Bluetooth Low Energy Technologie“ basiert. Diese Beacons werden an verschiedenen Stellen im Gebäude platziert und senden, über Bluetooth, Signale an mobile Endgeräte. Befinden sich mindestens drei Beacons in Reichweite des Endgeräts, lässt sich die Position des Empfängers im zweidimensionalen Raum errechnen. Bei vier empfangenen Beaconsignalen kann sogar das Stockwerk ermittelt werden. (Azmitia et al. 2016). Obwohl Beacons relativ günstig und einfach anzubringen sind, haben diese auch ihre Schwächen. Die ausgeschriebenen Reichweiten entsprechen nicht immer der Realität, aufgrund von beispielweise Betonwänden, Türen oder auch Glasscheiben kann es hier zu erheblichen Schwankungen kommen. Hinzu kommen Überschneidungen der Wlan- und Bluetooth-Signale. Ein weiteres Problem sind die Akkulaufzeiten der Beacons, welche zwar bis zu einem Jahr halten können, jedoch variieren, und damit auch ein größerer Aufwand mit dem Austausch verbunden ist (Azmitia et al. 2016).

Aufgrund des immer weiter steigenden Erfolges der künstlichen Intelligenz, welche sich von einer beeindruckenden Gesichtserkennung bis hin zu einem fast komplett autonomen

Fahrsystem erstrecken, werde ich mich innerhalb meiner Seminararbeit mit dem innovativen Thema „Deep Learning“, einem Teilgebiet der künstlichen Intelligenz, auseinandersetzen. Basierend auf den Einschränkungen von GPS und der Beacon-Technologie, verwende ich Deep Learning um das Indoor Navigation Problem anzugehen. Hierfür entwickle ich ein neuronales Netz, welches mithilfe von Bildern Räume klassifizieren kann, um auf diese Weise das Indoor navigaiton Problem zu lösen. Die Arbeit ist dabei wie folgt gegliedert. Zu Beginn erläutere ich die Mechanismen von Deep Learning. Im weiteren Verlauf werde ich das Neuronale Netzwerk erläutern, welches den Schwerpunkt meiner Arbeit bildet. Zum Schluss erörtere ich verschiedene Geschäftsmöglichkeiten, die sich durch das neuronale Netz ermöglichen.

1.1 Einführung Deep Learning

1.1.1 Maschine Learning

Beim Deep Learning handelt es sich um ein Teilgebiet von Maschine Learning. Der große Unterschied zwischen Maschine Learning und dem klassischen Programmieren liegt darin, dass nicht nach Lösungen gesucht wird, sondern dem Maschine Learning System, Daten und Lösungen übergeben werden und nach bestimmten Regeln gesucht wird (siehe Abbildung 1) (Chollet 2017).

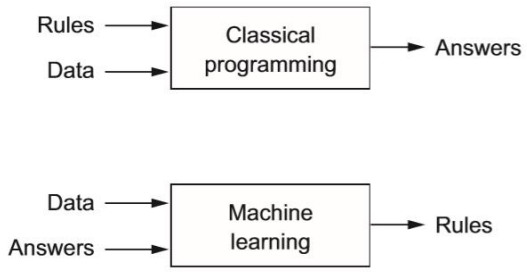


Abbildung 1 Unterschied Maschine Learning, klassisches Programmieren

Beim Maschine Learning werden, „Input Daten“ wie z.B. Bilder, zu aussagekräftigen „Output Daten“ wie z.B. verschiedene Klassen (Hund, Katze) transformiert. Das Ganze funktioniert indem „Representations“ erlernt werden. Bei einer Representation handelt es sich dabei lediglich um eine Änderung des Blickwinkels auf die Daten. Letztendlich macht ein Maschine Learning Algorithmus also nichts anderes als nach nutzvollen Representations mithilfe eines Feedback Signales zu suchen, um sich so gut wie möglich an die mitgegebenen Output Daten anzunähern (Chollet 2017).

1.1.2 Deep Learning

Beim Deep Learning wird mit aufeinanderfolgenden Schichten(Layers) von representations gearbeitet. Die Anzahl dieser Layers wird „depth“ genannt, daher auch der Name „Deep Learning“. Francois Chollet vergleicht diese Layers mit Legosteinen. Es werden mehrere Schichten aufeinander gebaut, die als „Data-Transformation Pipelines“ dienen. Jeder dieser Bausteine akzeptiert nur „Input Tensors“ einer bestimmten Form und gibt auch „Output Tensors“ einer bestimmten Form zurück. Bei einem Tensor oder Numpy Array handelt es sich um einen Behälter von Zahlen, welcher Daten repräsentiert. Die genaue Transformation wird in dem Gewicht(Weights) der Layer gespeichert. Diese Weights werden zu Beginn zufällig initialisiert. Die Schwierigkeit liegt darin, die richtigen Werte der Weights zu finden, da es Millionen von Möglichkeiten gibt und die Änderung eines Gewichtes alle anderen beeinflusst. Um diese Aufgabe zu lösen wird mit einer Loss Function gearbeitet, welche eine Voraussage mit den Gewichten und den wahren Output Daten vergleicht. Ein Optimizer versucht anschließend diesen Unterschied zu minimieren indem die Gewichte geändert werden (Siehe Abbildung 2) (Chollet 2017).

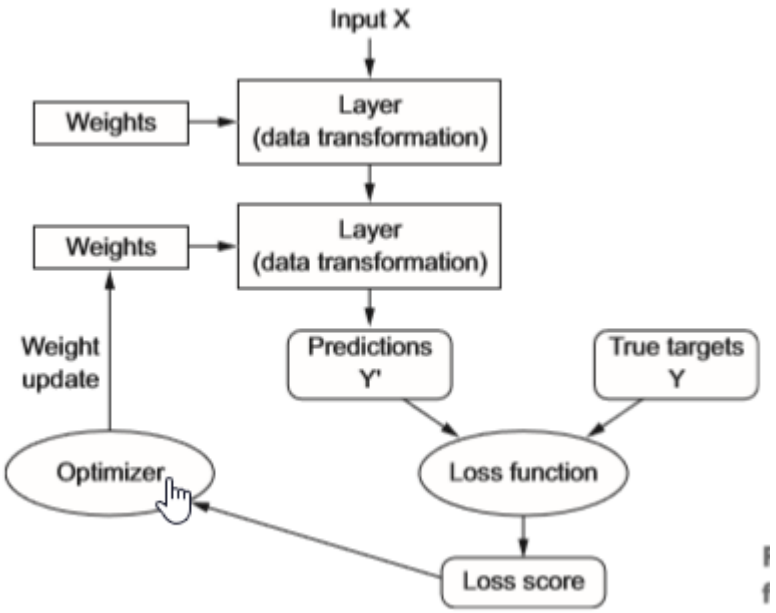


Abbildung 2 Backpropagation Algorithmus

1.2 Navigation mit Deep Learning

Besonders spannend wurde das Thema Deep Learning in den letzten Jahren aufgrund von technischen Fortschritten. Zum einen wurde die Rechenleistung aufgrund von besserer Hardware und der Nutzung von GPUs und TPUs um ein Tausendfaches verbessert. Die weiteren Gründe beziehen sich auf verbesserste Algorithmen und insbesondere die vereinfachte Datenbeschaffung, die sich dank des Internets ermöglicht haben (Chollet 2017).
Zwei Artikel betonen den technischen Fortschritt im Jahr 2018 auf dem Gebiet der Navigation. Bei dem ersten Artikel wurde ein neuronales Netz mit einer Feedback Loop darauf trainiert, sich in einem Labyrinth zu orientieren (Navigating with grid-like representations in artificial agents | DeepMind). Das Netz wurde mit Beispielen gefüttert, in denen Mäuse ein echtes Labyrinth durchquerten. Das spannende ist, dass das Netz unerwartet etwas Ähnliches wie Gitterzellen in einem biologischen Gehirn entwickelt hat (siehe Abbildung 3). Diese Gitterzellen ermöglichen es vielen Lebewesen ihre Position im Raum zu verstehen.

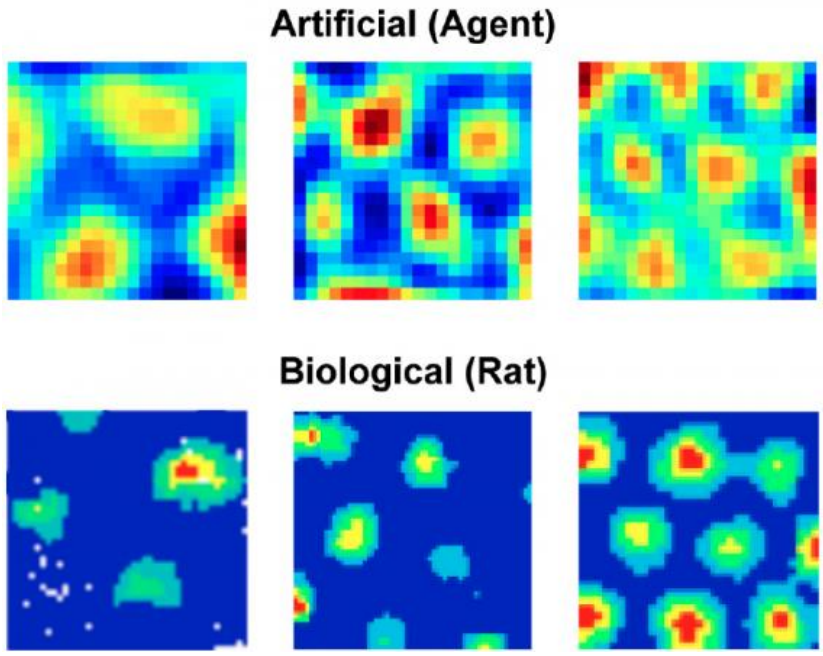


Abbildung 1 Gitterzellen

Das Netz wurde mit Reinforcement-Learning, einem Belohnungssystem, erweitert und konnte sich aufgrund der Gitterzellen-ähnlichen Struktur weitaus effektiver orientieren als jedes KI System zuvor. Bis zu diesem Punkt waren neuronale Netze nämlich nicht geeignet für diese Art von Aufgabe. Auch ist bemerkenswert, dass sich das Netz aufgrund der Gitterzellen angefangen hat mehr wie ein Tier zu orientieren, was zukünftig möglicherweise zur Roboternavigation beitragen könnte (Navigating with grid-like representations in artificial agents | DeepMind).

Im zweiten Projekt von DeepMind wird ein neuronales Netz auf die Navigation in Städten ohne Karte trainiert (Mirowski et al. 2018). Der Ansatz ist an die Navigation des Menschen angelehnt, welcher mit visuellen Beobachtungen ganze Städte durchqueren kann, und das ohne Karten, GPS-Lokalisierung oder andere Hilfsmittel. Das Netz soll sich nur durch Bildern aus Google Streetview und den Navigationspfeilen zum nächsten Ort orientieren (siehe Abbildung 4).

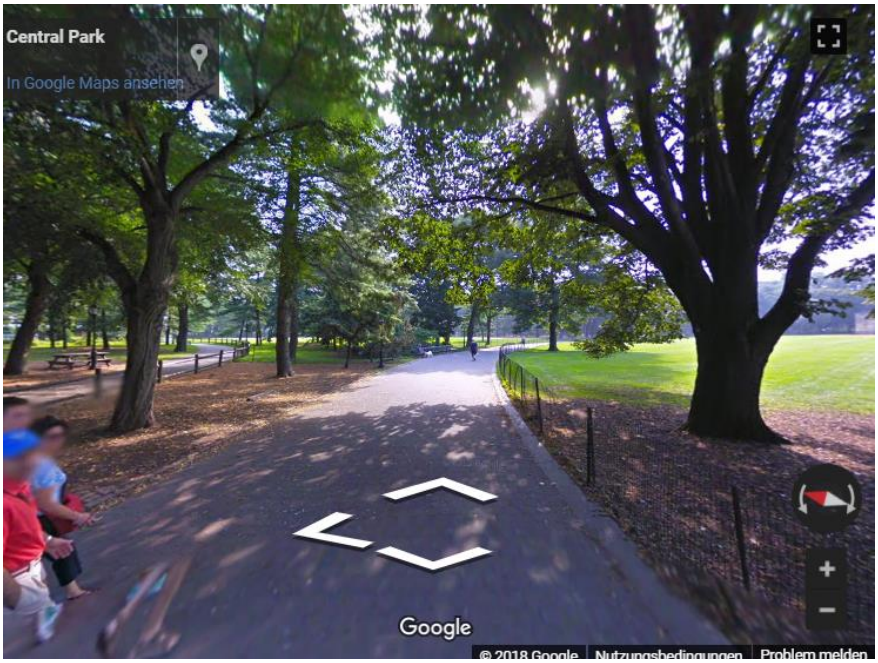


Abbildung 4 Google Streetview

Es wird Curriculum Learning genutzt, welches die Komplexität der Aufgabe stufenweise steigert. Es werden Ziele in einer Reichweite von 500m vergeben und Entfernung schrittweise auf bis zu mehreren Kilometern erhöht (Mirowski et al. 2018).

Das DeepMind Projekt besteht grundlegend aus drei Teilen. Im ersten Teil wird ein convolutional Network gebaut, welches Bilder bearbeiten und visuelle Merkmale extrahieren kann. Diese werden dann im zweiten Teil zu Orten rekonstruiert, womit ermöglicht wird, den jetzigen Standort, sowie den Zielort zu identifizieren. Im dritten Teil geht es dann um die Navigation. Wie bewege ich mich vom Standort zum Ziel(Mirowski et al. 2018)?

2 Umsetzung der Indoor Lokalisierung

2.1 Beschaffung der Daten

Um ein neuronales Netz zu trainieren, werden zunächst Input Daten benötigt. Die Idee des Projektes ist es mithilfe weniger Panoramabilder ausreichend Daten zu generieren, um eine Lokalisierung zu bauen ähnlich wie im zweiten Artikel (Mirowski et al. 2018). Zu diesem Zweck wurden zu Beginn fünf Panoramabilder von jeweils drei „Räumen“ gemacht. Die Anzahl der Bilder wurde im weiteren Verlauf auf 9 Bilder erhöht, um die Performance zu steigern. Um möglichst verschiedene Bilder zu generieren, wurden verschiede Objekte umgestellt, wie beispielsweise Tische oder Stühle. Zudem wurde der Raum aus verschiedenen Blickwinkeln und Ecken fotografiert.

Bei dem ersten Raum handelt es sich um den Seminarraum „H 1.5“ der FHWS (siehe Abbildung 5).

Abbildung 2 Room 1 H1.5

Bei dem zweiten Raum handelte es sich um den Vorlesungsraum „I3.24“ (siehe Abbildung 5). Hier ist zu erkennen, dass einige Ähnlichkeiten auftreten, wie z. B. die Fenster oder die Wand- und Deckenstruktur.

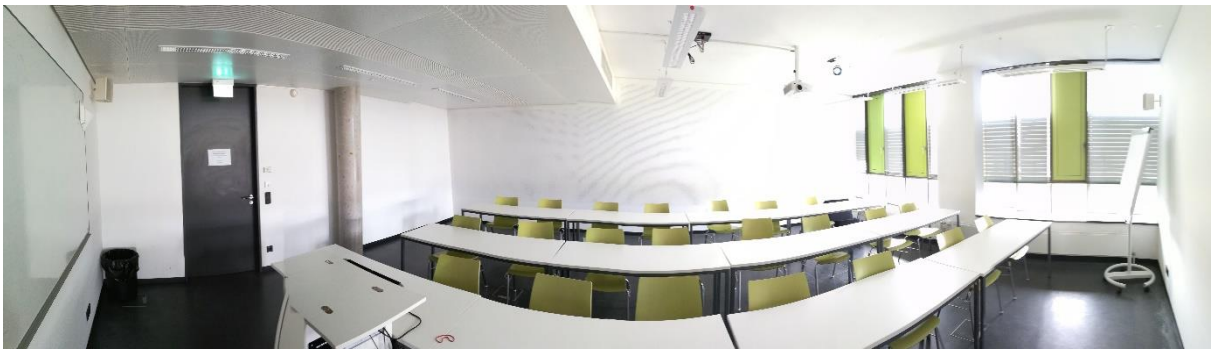


Abbildung 3 Room 2 I 3.24

Bei dem dritten Raum handelt es sich um die Klasse „other“. Es wurden Bilder von unterschiedlichen Orten genutzt, um dem neuronalen Netz zu vermitteln, dass nicht nur die zwei vorherigen Räumlichkeiten existieren. Damit soll sichergestellt werden, dass falls bei der Nutzung der Indoor Navigation ein Ort übergeben wird, welcher noch nicht erlernt wurde, das Netz diesen Vorfall versteht und eine entsprechende Antwort geben kann. Hierfür wurden beispielsweise Aufnahmen in einem Lebensmittelgeschäft (siehe Abbildung 7), an einer Straßenbahnhaltestelle oder auch in einem Treppengelände (siehe Abbildung 8) durchgeführt.

Abbildung 4 class other: Lebensmittelgeschäft

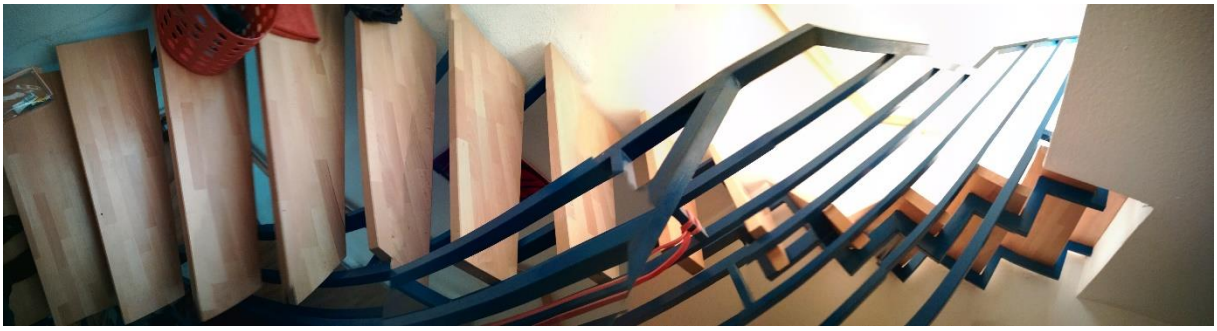


Abbildung 5 class other „Treppen“

2.2 Aufbereitung der Daten

Beim Deep Learning werden große Datenmengen benötigt, um eine hohe Genauigkeit zu generieren. Da jedoch mit Panoramabildern gearbeitet wird, welche sich durch die Abdeckung eines großen Betrachtungswinkels auszeichnen, entsteht die Möglichkeit das Bild in viele kleinere Bilderausschnitte zu teilen und damit eine große Datenmenge zu generieren. Für dieses

Vorgehen wurde ein Python Skript geschrieben. Der erste Teil besteht daraus, eine Ordnerstruktur aufzustellen. Im Grunde werden hier die Ordner „train“, „validate“ und „test“ erstellt mit den Klassen als Unterordnern(siehe Abbildung 9).

```
In [246]:
1  import os
2
3
4  primes = ["test", "validate", "train"]
5  for prime in primes:
6      os.makedirs("testse/"+prime+"/Raum1")
7      os.makedirs("testse/"+prime+"/Raum2")
8      os.makedirs("testse/"+prime+"/other")
```

Abbildung 6 Erstellung der Ordnerstruktur

Im zweiten Teil des Skriptes werden aus allen Bildern des Ordners „Original Dataset“ zufällige quadratische Teile ausgeschnitten und in die neue Ordnerstruktur kopiert (siehe Abbildung 11). Besonders wichtig war es hierbei bestimmte Einstellungen vornehmen zu können. Dies dient dem späteren Zweck das neuronale Netz auf verschiedene Konstellationen zu testen um ein optimales Ergebnis zu generieren.

Es gibt insgesamt drei Einstellungen, die im Skript geändert werden können. Zum ersten kann in den Zeilen 7-9 die Aufteilungen der Bilder in die Kategorien „Train“, „Test“ und „Validate“ bestimmt werden. Bei der zweiten Einstellung in Zeile 11 kann entschieden werden, wie viele Teile aus jedem Bild ausgeschnitten werden. Die letzte Option findet in Zeile 45 statt, hier kann entschieden werden wie groß die jeweiligen Teile sein sollen. Im unteren Beispiel werden zufällige Teile zwischen der Größe 1000 und 1600 Pixel ausgeschnitten (siehe Abbildung 10). Auch stellt das Skript sicher, dass jede Klasse dieselbe Anzahl an Bildern erhält um eine gleichmäßige Verteilung zu gewähren.

In [22]:

```python
from PIL import Image
from random import randint
import glob
import random
import sys

numberTrain = 6
numberValidate = 1
numberTest = 1

numberTiles = 100

picSum = numberTrain + numberValidate + numberTest

rooms = ["Raum1", "Raum2", "other"]
primes = ["train", "validate", "test"]

for room in rooms:

    path= ("Original Dataset/"+ room)
    picPaths = glob.glob(path+"/*")
    random.shuffle(picPaths)

    for prime in primes:


        for NumberPictures in range(picSum):
            print(str(NumberPictures))
            print(picPaths[NumberPictures])
            if NumberPictures<numberTrain and prime == "train" or NumberPictures>=numberTrain and NumberPictures<(numberTrain


                if len(picPaths) < picSum:
                   sys.exit("Not Enough Pictures!")


                for NumberTile in range(numberTiles):


                    img = Image.open(picPaths[NumberPictures])
                    width = img.size[0]
                    height = img.size[1]


                    randomSize = (randint(1000, 1600))
                    randomWidth =(randint(0, width-randomSize))
                    randomHeight = (randint(0, height-randomSize))



                    img3 = img.crop(
                        (
                        randomWidth ,
                        randomHeight,
                        randomWidth+ randomSize,
                        randomHeight+ randomSize,
                    )
                    )
                    img3.save("testse/"+prime+"/"+room+"/"+str(NumberTile+1+(NumberPictures*(numberTiles+1)))+".jpg")
```

Abbildung 7 Skript cut Picture

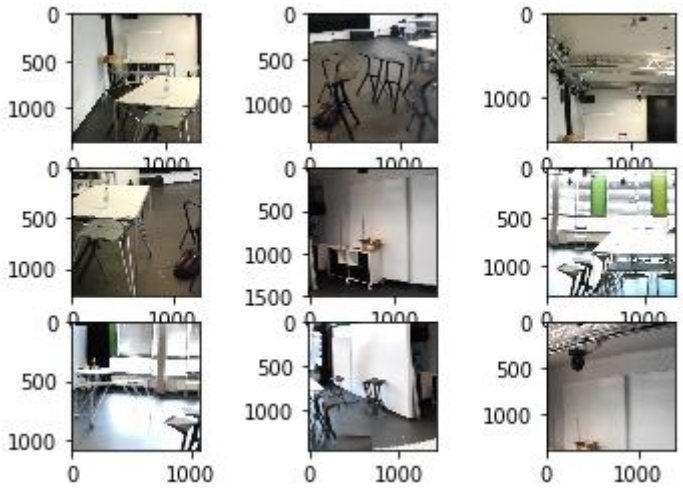


Abbildung 8 Bildausschnitte Room1

2.3 Aufbau des neuronalen Netzes

2.3.1 Keras

Im Projekt wird mit dem Keras Deep Learning Framework gearbeitet. Der Fokus von Keras liegt darin, schnelle Experimente zu ermöglichen und besonders nutzerfreundlich zu sein. Vorteilhaft sind die konsistenten und einfachen APIs, welche die Benutzeraktionen minimiert. Auch handhabt Keras keine Low-Level-Operationen wie Tensor-Manipulation und -Differenzierung. Stattdessen stützt es sich auf eine spezialisierte, gut optimierte Tensor-Bibliothek, die als Backend-Engine von Keras dient. In unserem Fall nutzt Keras TensorFlow als Backend, weil es die am weitesten verbreitete, skalierbare und produktionsreife Lösung ist (Chollet 2017).

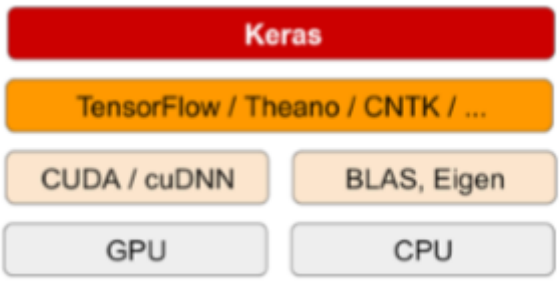


Abbildung 9 Aufbau

2.3.2 Pretrained Convent

Bei der Entwicklung der Lokalisierung wurde aufgrund der relativ geringen Datenmenge, Nutzen von vortrainierten „convolutional neural Networks“ (Convnets), gemacht. Dabei handelt es sich um ein neuronales Netz, das zuvor an einem großen Datensatz trainiert wurde. Grundsätzlich bestehen Convnets aus zwei Teilen: (1) die „Convolutional base“, deren Layers generisch sind und damit leicht wiederverwendbar sind, und (2) dem Classifier, deren Layers sehr spezifisch und abhängig an den davor gelernten Klassen sind. Es wird der ImageNet Datensatz benutzt, bestehend aus 1,4 Millionen Bildern, welche in 1000 verschiedene Klassen zugeordnet sind. Als Architektur wird „VGG16“ aufgrund der einfachen Verständlichkeit genutzt. Es gibt zwei verschiedene Ansätze das Netz zu benutzen, es wird jedoch ausschließlich auf Feature Extraction eingegangen, da mit dieser Methode gearbeitet wurde (Chollet 2017).

2.3.3 Feature extraction

Bei der Methode „Feature extraction“ gibt es zwei Möglichkeiten; bei beiden wird die „convolutional base“ wiederverwendet und ein neuer Classifier verwendet. Bei der ersten Option, welche auch in dem Projekt umgesetzt wurde, laufen die Daten durch die „convolutional base“ und die Ausgabe wird in Tensoren gespeichert. Anschließend werden diese Daten als Input für das Training des neu gebauten Classifiers verwendet. Diese Lösung ist sehr schnell umzusetzen, da alle Daten insgesamt nur einmal durch die „convolutional base“ laufen. Bei der zweiten Möglichkeit, welche im Projekt auch angewandt wurde, jedoch aufgrund von Kapazitätsmangel gescheitert ist, wird die „convolutinal base“ zunächst „eingefroren“ und anschließend um einem neu gebauten Classifier erweitert(siehe Abbildung 13). Die Daten laufen komplett „end-to-end“ durch das gesamte Netz. Diese Lösung benötigt eine hohe Rechenleistung, ermöglicht aber „data augmentation“, wodurch künstlich neue Bilder generiert werden und somit eine Maßnahme gegen Overfitting geschaffen wird (Chollet 2017).

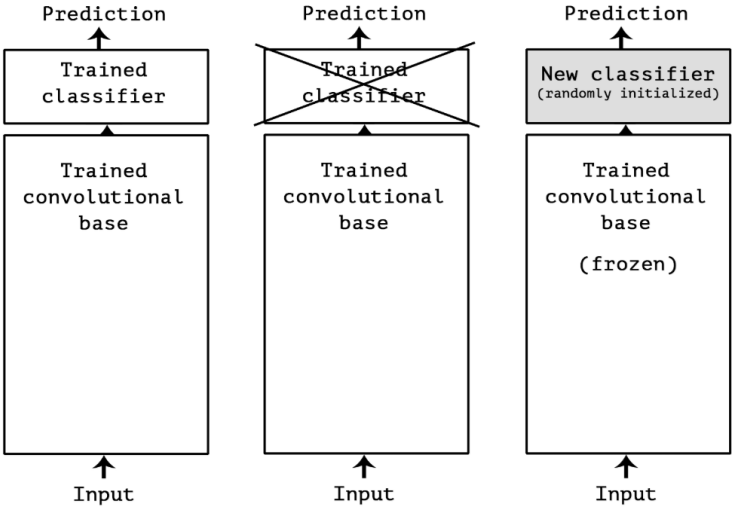
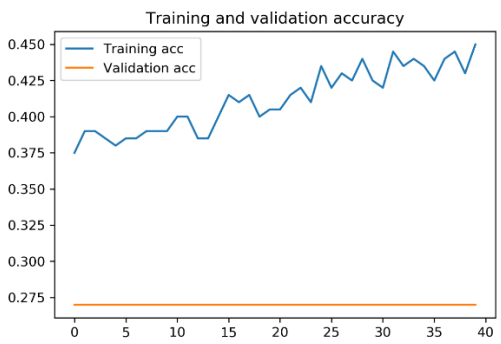


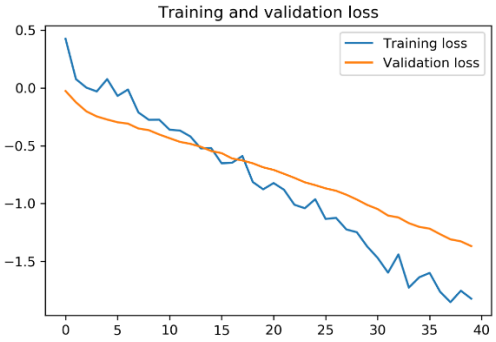
Abbildung 10 Feature extraction

2.3.4 Lösungsverlauf

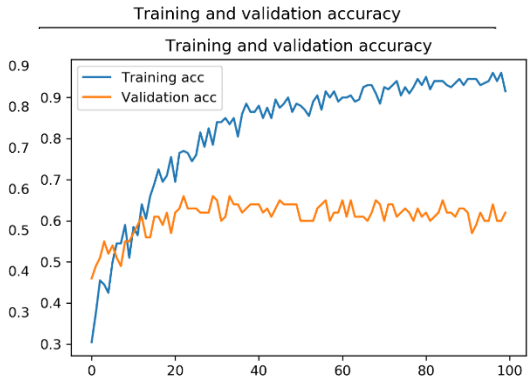
Die Training Accuracy gibt an, wie gut das Netz Bilder erkennt, mit denen es trainiert wurde, während die Validation Accuracy die Wahrscheinlichkeit angibt, komplett neue Bilder zu klassifizieren. Overfitting tritt auf sobald die Validation Accuracy deutlich niedriger als die Training Accuracy ausfällt. Grund hierfür ist eine zu hohe Spezifizierung des Models. Anders ausgedrückt, merkt sich das Model zu viele Kleinigkeiten und arbeitet zu spezifisch.

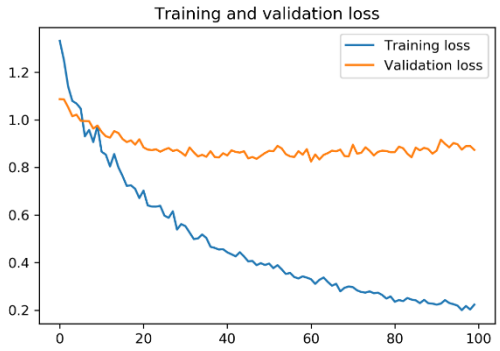
Die ersten Ergebnisse des neuronalen Netzes erreichen eine Validation Accuracy von unter 30%. Das bedeutet, dass würfeln eine höhere Chance hätte das Bildteil richtig zuzuordnen.



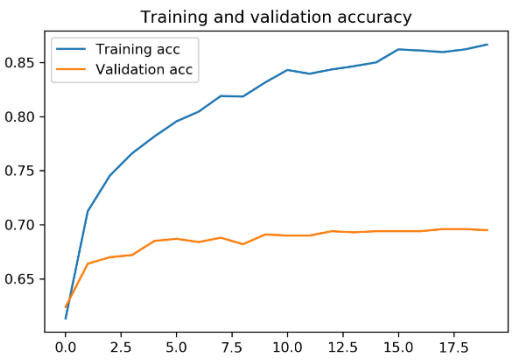


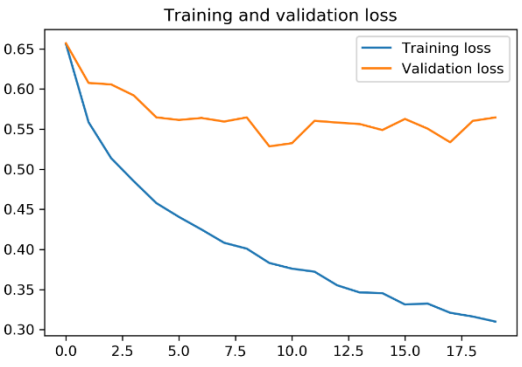
Zunächst wurde die Klasse „Other“ entfernt und eine Validation Accuracy von 60% konnte erreicht werden. Ein starkes Overfitting ist zu sehen, da die Training Accuracy bei über 90% liegt.



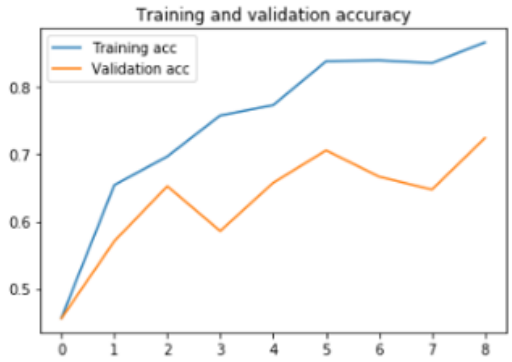


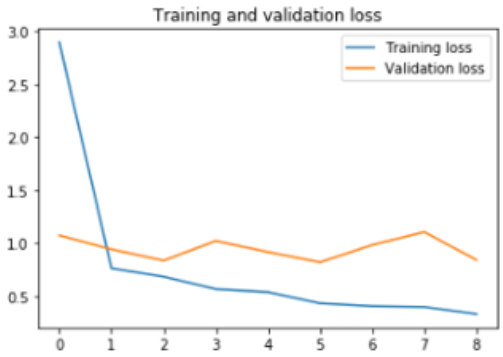
Hier wurden nun drei Maßnahmen ergriffen. Es wurde mit mehr Bildern und Bilderteilen gearbeitet. Die Größe des Netzwerkes wurde verkleinert, sodass das Netz gezwungen ist mehr zu generalisieren. Als letzte Maßnahme wurde ein Dropout eingebaut. Beim Dropout werden zufällige Output-features einer Schicht auf 0 gesetzt mit dem Ziel, zufällige Muster aufzulösen. Mit diesen Maßnahmen wurde das Overfitting minimiert und es konnte eine Validation Accuracy von circa 70% erreicht werden.





Als letzten Schritt wurde die Klasse „Other“ wieder hinzugefügt und die Parameter angepasst. Als endgültiges Ergebnis konnte eine Validation Accuracy von 72% erreicht werden.





2.3.5 Praktische Umsetzung

2.3.5.1 Netz trainieren

Im folgenden Kapitel werde ich meine Lösung mit dem Python Code erläutern. Im ersten Abschnitt wird Keras importiert, welches aufgrund der Installation automatisch Tensorflow als Backend einrichtet. Wir deklarieren „conv_base“ was in dem Fall unser vortrainiertes Netz darstellt. Wir benutzen hierfür die VGG16 Architektur mit den „imagenet“ Daten als Gewichtung. Auch wurde die Variable „include_top“ auf False gesetzt, womit der Classifier des vortrainieren Netzes nicht mehr benutzt wird, da ein neuer Classifier gebaut und genutzt werden soll.

```
1  import keras
2  from keras.applications import VGG16
3
4  conv_base = VGG16(weights='imagenet',
5                    include_top=False,
6                    input_shape=(150, 150, 3))
7
```

Im nächsten Abschnitt werden die ausgeschnittenen Bildteile durch das vortrainierte Netz vorausgesagt (predicted). Um dieses jedoch zu ermöglichen, müssen die Daten zu Tensors umgewandelt werden. Mit Keras kann dieser Schritt automatisiert werden, indem die Klasse „ImageDataGenerator“ verwendet wird. In der Methode wird im Parameter (rescale=1./ 255)

mitgegeben, was aussagt das wir 255 Farben für jeden Pixel haben und diese Werte zwischen 0 und 1 umgewandelt wiedergeben. Ebenfalls ist die Batch_size zu sehen. Sie gibt an, wie viele Bildteile auf einmal bearbeitet werden. Die Variable class_mode mit dem Wert „categorical“ wird verwendet, da wir mit mehr als zwei Klassen arbeiten und jedes Bild genau einer Klasse zugeordnet ist. Dieser Vorfall wird „single-label categorcial classification“ genannt. In den letzten Zeilen ist zu sehen, wie viele Bildteile durch das vortrainierte Netz laufen. Als Ausgabe erhalten wir die Features und die Labels (Klassen) als Tensoren zurück

```
import os
import numpy as np
import random
import glob
#from keras.utils import to_categorical
from keras.preprocessing.image import ImageDataGenerator
random.shuffle(glob.glob("c:/Users/Dennis1/ki/testse/train/*"))
random.shuffle(glob.glob("c:/Users/Dennis1/ki/testse/validate/*"))
random.shuffle(glob.glob("c:/Users/Dennis1/ki/testse/test/*"))
base_dir = ("c:/Users/Dennis1/ki/testse")
train_dir = ("c:/Users/Dennis1/ki/testse/train")
validation_dir = ("c:/Users/Dennis1/ki/testse/validate")
test_dir = ("c:/Users/Dennis1/ki/testse/test")
datagen = ImageDataGenerator(rescale=1./255)
batch_size = 10
def extract_features(directory, sample_count):
    features = np.zeros(shape=(sample_count, 4, 4, 512))
    labels = np.zeros(shape=(sample_count, 3))
    generator = datagen.flow_from_directory(
        directory,
        target_size=(150, 150),
        batch_size=batch_size,
        class_mode='categorical')
    i = 0
    for inputs_batch, labels_batch in generator:
        features_batch = conv_base.predict(inputs_batch)
        features[i * batch_size : (i + 1) * batch_size] = features_batch
        labels[i * batch_size : (i + 1) * batch_size] = labels_batch
        i += 1
        if i * batch_size >= sample_count:
            break
    return features, labels
train_features, train_labels = extract_features(train_dir, 4000)
validation_features, validation_labels = extract_features(validation_dir, 1200)
```

```
Found 4200 images belonging to 3 classes.
Found 1200 images belonging to 3 classes.
```

Die extrahierten feature Tensoren werden nun von den Werten (Bilderanzahl, 4, 4, 512) zu den Werten (Bilderanzahl, 8192) umgewandelt, sodass diese durch unseren neuen Classifier laufen können.

```

train_features = np.reshape(train_features, (4000, 4 * 4 * 512))
validation_features = np.reshape(validation_features, (1200, 4 * 4 * 512))


```

Im folgenden Abschnitt wird nun der Classifier gebaut. Die letzte Schicht hat einen Speicher von drei und benutzt als activiation softmax. Das bedeutet, dass das Netzwerk eine Wahrscheinlichkeitsverteilung über die drei verschiedenen Klassen ausgibt. Das Netz benutzt nun die formatierten Features, welche wir aus dem vortrainierten Netz erhalten haben, und lässt sie durch den neu erstellten Classifier laufen. Die Epochen geben an, dass die kompletten Daten 9-mal durch den Classifier laufen.

In [60]:

```
from keras import models
from keras import layers
from keras import optimizers


model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=4 * 4 * 512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(3, activation='softmax'))

model.compile(optimizer="rmsprop",
              #binary_crossentropy 2 Klassen
              loss='categorical_crossentropy',
              metrics=['acc'])

history = model.fit(train_features, train_labels,
                    epochs=9,
                    batch_size=200,
                    validation_data=(validation_features, validation_labels))
```

```
Train on 4000 samples, validate on 1200 samples
Epoch 1/9
4000/4000 [==============================] - 3s 783us/step - loss: 2.8965 - acc: 0.4565 - val_loss: 1.0728 - val_acc: 0.4558
Epoch 2/9
4000/4000 [==============================] - 2s 487us/step - loss: 0.7621 - acc: 0.6542 - val_loss: 0.9419 - val_acc: 0.5708
Epoch 3/9
4000/4000 [==============================] - 2s 490us/step - loss: 0.6834 - acc: 0.6968 - val_loss: 0.8351 - val_acc: 0.6525
Epoch 4/9
4000/4000 [==============================] - 2s 498us/step - loss: 0.5663 - acc: 0.7578 - val_loss: 1.0213 - val_acc: 0.5858
Epoch 5/9
4000/4000 [==============================] - 2s 504us/step - loss: 0.5373 - acc: 0.7730 - val_loss: 0.9164 - val_acc: 0.6575
Epoch 6/9
4000/4000 [==============================] - 2s 535us/step - loss: 0.4314 - acc: 0.8382 - val_loss: 0.8190 - val_acc: 0.7058
Epoch 7/9
4000/4000 [==============================] - 2s 533us/step - loss: 0.4047 - acc: 0.8395 - val_loss: 0.9830 - val_acc: 0.6667
Epoch 8/9
4000/4000 [==============================] - 2s 523us/step - loss: 0.3967 - acc: 0.8355 - val_loss: 1.1048 - val_acc: 0.6475
Epoch 9/9
4000/4000 [==============================] - 2s 495us/step - loss: 0.3305 - acc: 0.8665 - val_loss: 0.8411 - val_acc: 0.7242
```

Im nächsten Teil wird die Accuracy und Loss Funktion graphisch dargestellt und gespeichert. Auch wurde hier ein timestamp hinzugefügt, sodass die Bilder sich nicht überschreiben.

```
%matplotlib inline
import matplotlib.pyplot as plt
import time

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc,  label='Training acc')
plt.plot(epochs, val_acc, label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.savefig("Training Acc-"+str((int(time.time())))+".png", dpi=300, bbox_inches='tight')
plt.figure()

plt.plot(epochs, loss,  label='Training loss')
plt.plot(epochs, val_loss,  label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.savefig("Training Loss-"+str((int(time.time())))+".png", dpi=300, bbox_inches='tight')
plt.show()
```

Als letztes wird das Netz lokal abgespeichert, sodass dieses wiederverwendet werden kann.

```
In [4]:
from keras.models import save_model

model.save(".h5")
```

2.3.5.2 Netz testen

Das Ziel in diesem Kapitel ist es das trainierte Netz nun zu verwenden um komplett neue Bilder zu klassifizieren. Zunächst wird das neue Bild genau wie in Kapitel 4 in verschiedene Bildteile

geschnitten. Hier laufen alle Bildteile durch das vortrainierte Netz und geben wieder Numpy Arrays aus. Jetzt wird der gespeicherter Classifier geladen und verwendet. Nachdem nun die Accuracy für jedes einzelne Bild generiert wurde, berechnet das Programm die Summe dieser, um eine Wahrscheinlichkeit für das Gesamtbild zu generieren. Hier sollen nur Bilder mit einer Accuracy von über 60% beachtet werden. Nicht identifizierbar Bildteile, wie z. B. eine weiße Wand, sollen damit ausgefiltert werden. Zum Schluss wird als Text zurückgegeben, um welchen Raum es sich handelt.

3 Mögliche Anwendungsfälle des neuronalen Netzes

Bei meiner ersten Idee handelt es sich um eine mobile Navigation App. Die App benötigt Panoramabilder von verschiedenen Räumen und eine Offline-Map. Der Endnutzer soll dann in der Lage sein, an diesen Orten ein Foto zu schießen und sich mit der offline Karte, auf welcher der Standort nun angezeigt wird, zu orientieren. Eine komplette Navigation zu entwickeln wäre theoretisch möglich, jedoch ziemlich aufwendig und erst bei genügend Bedarf umzusetzen. Die App würde sich vor allem in Flughafen, Museen, Universitäten oder sogar großen Kaufhäusern, in welchen man Werbung anbringen könnte, einsetzen lassen.
Bei der zweiten Idee handelt es sich um eine Blindenbrille. Diese soll es blinden Menschen ermöglichen sich leichter innerhalb blindenungerechten Gebäuden zu orientieren. Die Brille soll in der Lage sein, Bilder zu schießen und auf Basis dieser dem Blinden per Kopfhörer den Standort, sowie eine ideale Wegbeschreibung zu einem gewünschten Ort angeben.
Bei meiner letzten Idee handelt es sich um eine Hilfe der Roboternavigation. Bei der Selbstlokalisierung, welcher ein großer Teil der Roboternavigation ist, wird zurzeit mit verschiedenen Sensoren gearbeitet. Dabei werden Radencoder, Kompässe, Infrarot, Ultraschall, Laser und noch weiteren Mitteln genutzt. Zwar sehe ich die Lokalisierung, durch mein Netz in seinem derzeitigen Studium nicht als eine Alternative, jedoch als eine günstige Ergänzung. In einer statischen Umgebung kann der Roboter bei Verlust der Orientierung mit einer einfachen Kamera den Raum bestimmen.

4 Fazit

Auf die Frage, ob mit der Hilfe eines neuronalen Netzes, welches auf Bilder trainiert wurde, eine Raumnavigation möglich ist, kann mit einem Ja geantwortet werden.
Es war möglich in einer relativ kurzen Zeit ein lauffähiges neuronales Netz zu bauen, welches in der Lage ist Bilder drei verschiedenen Orten zuzuorten. Problematisch könnte das Ganze werden, sobald Räume zu identisch aussehen, wie z.B. die Vorlesungsräume der FHWS. Obwohl die Beacon Technologie aufgrund von einer höheren Genauigkeit und einem geringeren Aufwand die Nase noch immer vorne hat, ist klar zu erkennen, dass der Deep Learning Ansatz sich im Bereich der Navigation durchsetzen wird.

5 Literaturverzeichnis

ASSETS'17. Proceedings of the19th International ACM SIGACCESS Conference on Computers and Accessibility : Oct. 29-Nov. 1, 2017, Baltimore, MD, USA (2017). New York, New York: Association for Computing Machinery.

Azmitia, Alfredo; Mohnke, Janett; Wiechers, Henning (2016): Wo bin ich? iBeacons im Einsatz in der Bibliothek der TH Wildau. In: *Bibliothek Forschung und Praxis* 40 (3). DOI: 10.1515/bfp-2016-0050.

Chollet, François (2017): Deep learning with Python. Shelter Island, NY: Manning (Safari Tech Books Online). Online verfügbar unter http://proquest.safaribooksonline.com/9781617294433.

Mirowski, Piotr; Grimes, Matthew Koichi; Malinowski, Mateusz; Hermann, Karl Moritz; Anderson, Keith; Teplyashin, Denis et al. (2018): Learning to Navigate in Cities Without a Map. Online verfügbar unter http://arxiv.org/pdf/1804.00168v2.

Navigating with grid-like representations in artificial agents | DeepMind. Online verfügbar unter https://deepmind.com/blog/grid-cells/, zuletzt geprüft am 10.07.2018.