



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Alexander Lange

Die automatische Erstellung von
Bildbeschreibungen, sowie deren Auswertung,
mittels selbst trainiertem neuronalem Netzwerk

Alexander Lange

Die automatische Erstellung von Bildbeschreibungen, sowie deren Auswertung, mittels selbst trainiertem neuronalem Netzwerk.

Bachelorarbeit eingereicht im Rahmen des Studiums

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Michael Neitzke
Zweitgutachter: Prof. Dr.-Ing. Andreas Meisel

Abgegeben am

Alexander Lange

Thema der Arbeit

Die automatische Erstellung von Bildbeschreibungen, sowie deren Auswertung, mittels selbst trainiertem neuronalem Netzwerk.

Stichworte

künstliche Intelligenz, Machine-Learning, Bildbeschreibungen, CNN, RNN, Transformer, Attention, BLEU

Kurzzusammenfassung

Die Arbeit behandelt das Erstellen von künstlichen neuronalen Netzen zum Erstellen von Bildbeschreibungen. Dabei wird auf das Konzept, die Realisierung sowie die Auswertung der Modelle eingegangen. Für die Auswertung wurden unter der Verwendung von drei Datensätzen verschiedener Größen verschiedene Modelle erstellt. Diese Modelle dienten der Findung geeigneter Parameter für finale Modelle. Jede realisierte Architektur wird durch mindestens ein Modell widerspiegelt. Die Qualität der finalen Modelle wird anhand von Beispielbildern ausgewertet. Bei den Beispielbildern handelt es sich um Bilder, die nicht für das Training der Modelle genutzt wurden.

Alexander Lange

Title of the paper

The automatic creation of image descriptions, as well as their evaluation, using a self-trained neural network.

Keywords

artificial Intelligence, machine-learning, image captioning, cnn, rnn, tranformer, attention, BLEU

Abstract

This thesis deals with the creation of artificial neural networks for the task of image captioning. It deals with the concept, the realisation, and the evaluation of the models. For the evaluation, different models were created using three datasets of different sizes. These models were used to find suitable parameters for final models. Each realised architecture is reflected by at least one model. The quality of the final models is evaluated using example images. The sample images were not used for training.

Inhalt

1	Einleitung	9
1.1	Themenbeschreibung	9
1.2	Wissenschaftlicher Stand.....	9
1.3	Ziel der Arbeit	10
1.4	Struktur der Arbeit.....	11
2	Analyse.....	12
2.1	Ziele/Anforderungen der Arbeit	12
2.2	Umsetzbarkeit der Anforderungen.....	14
3	Grundlagen.....	17
3.1	Neuronale Netze	17
3.2	Backpropagation	19
3.3	Convolutional Neural Network (CNN)	22
3.3.1	Convolutional Layer / Faltungsschicht	23
3.3.2	Pooling Layer	26
3.3.3	Linear Layer	26
3.3.4	Dropout	27
3.3.5	Softmax	28
3.3.6	Inception Layer	29
3.4	Recurrent Neural Network (RNN)	30
3.5	Embedding	32
3.6	Long Short-Term Memory (LSTM)	32
3.7	Gated Recurrent Unit (GRU)	34
3.8	Transformer	35
3.8.1	Attention	36
3.8.2	Encoder	37
3.8.3	Decoder	37
4	Konzept	38
4.1	Architektur: CNN-RNN	38
4.2	Architektur: CNN-RNN-Attention	41
4.3	Architektur: Transformer	42
4.4	Framework	43
4.5	Auswertung.....	43

5 Realisierung.....	44
5.1 Datengrundlage	44
5.2 CNN	45
5.3 RNN	46
5.4 CNN-RNN-Attention	46
5.5 Transformer	47
5.6 Training	48
6 Schluss.....	49
6.1 Auswertung/Fazit.....	49
6.1.1 Vorüberlegungen.....	49
6.1.2 Auswertung	52
6.2 Ausblick	65
6.3 Abschluss.....	68
Links	69
<i>Datensets</i>	69
<i>Code</i>	69
Literaturverzeichnis	70

Abbildungsverzeichnis

[Albawi u.a. 2017] ALBAWI, Saad ; MOHAMMED, Tareq Abed ; AL-ZAWI, Saad: Understanding of a Convolutional Neural Network. In: *2017 International Conference on Engineering and Technology (ICET)* ; 21-23 Aug 2017, Antalya, Türkei. Phuket, Thailand: ICET 2017 – Elektronische ISBN 978-1-5386-1949-0 – S.1-S.6 – URL <https://ieeexplore.ieee.org/document/8308186>

[Degeratu u. a. 2001] DEGERATU, Vasile ; SCHIOPU, Paul ; DEGERATUR, Stefania: *McCulloch-Pitts neuron with directional couplers*, In: *2001 International Semiconductor Conference. CAS 2001 Proceedings ; 9-13 Okt. 2001, Sinaia, Rumänien*. Piscataway, NJ: IEEE, 2001 – ISBN 0-7803-6666-2 – S. 209-212 – Online unter: <https://ieeexplore.ieee.org/document/967448> - Abruf: 2020-04-13

[He u.a. 20120] HE, Sen ; LIAO, Wentong ; TAVAKOLI, Hamed R. ; YANG, Michael ; ROSENHAHN, Bodo ; PUGEAULT, Nicolas: *Image Captioning through Image Transformer*. Stand: 2020-04-29. Online unter: <https://arxiv.org/abs/2004.14231> Abruf: 2020-10-13

[Hayat u. a. 2018] HAYAT, Shaukat ; KUN, She ; TENGTAO, Zuo ; Yu, Yue ; TU, Tianyi ; DU, Yantong: *A Deep Learning Framework Using Convolutional Neural Network for Multi-class Object Recognition*. In: *2018 3rd IEEE International Conference on Image, Vision and Computing* ; 27-29 Jun 2018, Chongqing, China. Piscataway, NJ: IEEE, 2018 – ISBN 978-1-5386-4992-3 – S.194-198 – Online unter: <https://ieeexplore.ieee.org/document/8492777> – Abruf: 2020-04-16

[Karpathy u. a. 2017] KARPATY, Andrej ; FEI-FEI, Li: *Deep Visual-Semantic Alignments for Generating Image Descriptions*. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), April, Nr. 4, S. 664-676 – URL: <https://ieeexplore.ieee.org/document/7534740> - Zugriffsdatum: 2020-04-12 – Elektronische ISSN 1939-3539

[Kruse u. a. 2011] KRUSE, Rudolf ; BORGELT, Christian ; KLAWONN, Frank ; MOEWES, Christian ; RUß, Georg ; STEINBRECHER, Matthias: *Künstliche neuronale Netzwerke*. In: BIBEL, Wolfgang (Hrsg.) ; KRUSE, Rudolf (Hrsg.) ; NEBEL, Bernhard (Hrsg.): *Computational Intelligence : Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. Vieweg+Teubner Verlag, 2011, Wiesbaden, Deutschland. - ISBN 978-3-8348-1275-9 – S. 5-149 – Online unter: https://link.springer.com/chapter/10.1007/978-3-8348-8299-8_2#citeas

[Li u.a. 2017] LI, Fei-Fei ; JOHNSON, Justin ; YEUNG, Serena: Lecture 4 | Introduction to Neural Networks: Vortrag / Stanford University 2017-04-13, Stanford, Kalifornien - <https://www.youtube.com/watch?v=d14TUNcbn1k&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=4> - Abruf: 2020-06-12

[Li u.a. 2017 2] LI, Fei-Fei ; JOHNSON, Justin ; YEUNG, Serena: Lecture 10 | Recurrent Neural Networks: Vortrag / Stanford University 2017-08-11, Stanford, Kalifornien - <https://www.youtube.com/watch?v=d14TUNcbn1k&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=4> - Abruf: 2020-08-10

[Masuda u. a. 2018] MASUDA, Shuichi ; MANABE, Yusuke ; KAERI, Yuki ; SUGAWARA, Kenji: *Scene Recognition Method by Bag of Objects Based on Object Detector*. In: *2018 Joint 10th International Conference of Soft Computing and Intelligent Systems and 19th International Symposium on Advanced Intelligent Systems (ISIS)* ; 5-8 Dez. 2018, Toyama, Japan. Piscataway, NJ: IEEE, 2018 – ISBN 978-1-5386-2634-4 – S. 321-324 – Online unter:

<https://ieeexplore.ieee.org/document/8716173> - Abruf: 2020-04-15

[Phi 2018] PHI, Michael: *Illustrated Guide to LSTM's and GRU's: A step by step explanation*. Stand 2018-09-24. Online unter: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21> Abruf: 2020-08-11

[Rammer u.a. 2019] RAMMER, Werner ; SEIDL, Rupert: *Harnessing Deep Learning in Ecology: An Example Predicting Bark Beetle Outbreaks*. In: *Frontiers in Plant Science* 10 (2019), Okt. – Online unter: <https://doi.org/10.3389/fpls.2019.01327> Abruf: 2020-06-11

[Redmon u. a. 2018] REDMON, Joseph ; FARHADI, Ali: *YOLOv3: An Incremental Improvement*. Stand 2018-04-08. Online unter: <https://arxiv.org/abs/1804.02767> Abruf: 2020-06-12

[Redmon u. a. 2017] REDMON, Joseph ; FARHADI, Ali: *YOLO9000: Better, Faster, Stronger*. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 21-26 Juli 2017, Honolulu, HI. Piscataway, NJ: IEEE 2017 – ISBN 978-1-5386-0485-8 – S.6517-S.6525 – URL <https://ieeexplore.ieee.org/document/8100173> - Zugriffsdatum: 2020-04-17

[Saengthongpattana u.a. 2019] SAENGTHONGPATTANA, Kanchana ; KRIENGKET, Kanyanut ; PORKAEW, Peerachet ; SUPNITHI, Thepchai: *Thai-English and English-Thai Translation Performance of Transformer Machine Translation*. In: *2019 14th International Joint Symposium on Artificial Intelligence and Natural Language Processing (ISAI-NLP)* ; 30 Okt. – 1 Aug. 2019, Chiang Mai, Thailand. Piscataway, NJ: IEEE, 2020 – ISBN 978-1-7281-5631-6 – S.1-S.5 – Online unter: <https://ieeexplore.ieee.org/document/9045174> Abruf: 2020.08.13

[Saha 2018] SAHA, Sumit: *A Comprehensive Guide to Convolutional Neural Networks – the ELI5 way*. Stand: 2018-12-15. Online unter: <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53> Abruf: 2020-06-12

[Srivastava u. a. 2014] SRIVASTAVA, Nitish ; HINTON, Geoffrey ; KRIZHEVSKY, Alex ; SUTSKEVER, Ruslan: *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. In: *The Journal of Machine Learning Research* 15 (2014), 1, S.1929-1958. – Online unter: <https://dl.acm.org/doi/10.5555/2627435.2670313> Abruf: 2020-06-16

[Szegedy u.a. 2014] SZEGEDY, Christian ; LIU, Wei ; JIA, Yangqing ; SERMANET, Pierre ; REED, Scott ; ANGUELOV, Dragomir ; ERHAN, Dumitru ; VANHOUCKE, Vincent ; RABINOVICH, Andrew: *Going Deeper with Convolutions*. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 7-12 Jun. 2015, Boston, USA. Piscataway, NJ: IEEE, 2015 – ISBN 978-1-4673-6964-0 – S.1-S.9 – Online unter: <https://arxiv.org/abs/1409.4842> Abruf: 2020-08-11

[Tompson u.a. 2015] TOMPSON, Jonathan ; GOROSHIN, Ross ; JAIN, Arjun ; LECUN, Yann ; BREGLER, Christopher: *Efficient Object Localization Using Convolutional Networks*. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 7-12 Jun. 2015, Boston, USA. Piscataway, NJ: IEEE, 2015. – ISBN 978-1-4673-6964-0. – S.648-656 – Online unter: <https://ieeexplore.ieee.org/document/7298664/figures#figures> Abruf: 2020-06-16

[Vinodababu 2018] VINODABABU, Sagar: a-PyTorch-Tutorial-to-Image-Captioning, Online unter: <https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Image-Captioning/blob/master/img/model.png> Abruf: 2020-12-29

[Vinyals u. a. 2015] VINYALS, Oriol ; TOSHEV, Alexander ; BENGIO, Samy ; ERHAN, Dumitru: *Show and Tell: A Neural Image Caption Generator*. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 7-12 Juni 2015, Boston, USA. Piscataway, NJ: IEEE 2015 – URL <https://ieeexplore.ieee.org/document/7298935> - Zugriffsdatum: 2020-04-12 – Electronic ISBN 978-1-4673-6964-0

[Zhang u.a. 2019] ZHANG, Wei ; NIE, Wenbo ; LI, Xinle ; YU, Yao: Image Caption Generation With Adaptive Transformers. In: *2019 34th Youth Academic Annual Conference of Chinese Association of Automation (YAC)* ; 6-8 Jun. 2019, Jinzhou, China. Piscataway, NJ: IEEE, 2019. – ISBN 978-1-7281-3936-4 – S.521-526 – Online unter: <https://ieeexplore.ieee.org/document/8787715> Abruf: 2020-04-24

1 Einleitung

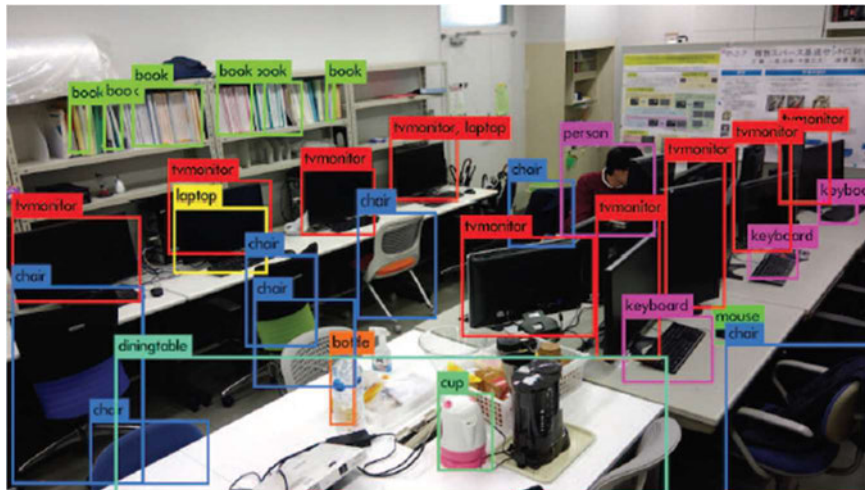
1.1 Themenbeschreibung

"Being able to automatically describe the content of an image using properly formed English sentences is a very challenging task, but it could have great impact, for instance by helping visually impaired people better understand the content of images on the web" (Vinyals u. a., 2015, S. 3156). Man stelle sich nun eine Welt vor, in der eine blinde Person nicht mehr mit einem Stock herum gehen muss, sondern sich wie jeder Mensch einfach bewegen kann. Unser Gehirn interpretiert was wir sehen innerhalb von dem Bruchteil einer Sekunde und gibt uns intuitiv eine Beschreibung von dem, was wir sehen. Wenn man nun einer blinden Person eine Kamera geben würde, die das Bild was sie sieht, analysiert und es der Person beschreibt, dann könnten sich blinde Personen größtenteils ohne einen Blindenstock bewegen. Dies wäre ein Szenario, welches in Zukunft denkbar wäre, in dem man ein Video/Bild mittels neuronalen Netzen analysiert und beschreibt. Wie das Zitat schon erwähnte, ist das Thema eine schwere Aufgabe, die jedoch in Zukunft einen großen Nutzen haben könnte. Es heißt nicht umsonst ein Bild sagt mehr als tausend Worte so könnten in Zukunft auch Emotionen erkannt werden, um so den Umgang von Personen im Bild zu beschreiben.

In dieser Bachelorarbeit geht es daher um den einfachsten Schritt in so einem Szenario. Es geht darum, ein neuronales Netzwerk auszuwerten welches dahingehend trainiert wurde Beschreibungen für Bilder zu generieren. Dabei ist wichtig, welche Objekte erkannt wurden, welche Beschreibungen verwendet wurden, um die erkannten Objekte zu beschreiben, sowie die Lesbarkeit und Verstehbarkeit der gesamten Beschreibung.

1.2 Wissenschaftlicher Stand

Der aktuelle Stand ist äußerst fortgeschritten. Moderne Modelle können eine Vielzahl von Objekten in Bildern zu einer hohen Korrektheit erkennen, unterscheiden und lokalisieren.



(Masuda u. a. 2018, S.323) - Objekterkennung durch YOLOv2

Natürlich hängt die Anzahl der verschiedenen Objekte mit der Datengrundlage, mit der das Netzwerk trainiert wurde, zusammen. Datensätze wie "Pascal VOC 2012" bestehen aus 11.530 Bildern, die jeweils in 20 Klassen unterteilt wurden, in denen 27450 Objekte markiert sind. Andere wie MSCOCO bestehen aus ca. 80.000 Bildern. Mit "Generative Adversarial Networks" lassen sich ganze Bilder und Texte durch ein Modell erstellen, welche nicht unbedingt als von einem Computer erstellt zu erkennen sind. Die Kombination von Modellen ermöglicht dann auch, Bilder in einfachen oder ausführlicheren Sätzen zu beschreiben.



(Vinyals u. a. 2015, S.8) – Personen beim Frisbee spielen

1.3 Ziel der Arbeit

Ein kurzer Blick auf ein Bild reicht für einen Menschen aus, um eine ungeheure Menge an Details hervorzuheben und detailliert zu beschreiben (vgl. Karpathy u. a. 2017). In den meisten Beschreibungen findet sich nur das wichtigste bzw. das für das Bild relevante. Schaut man auf das Bild von eben "Objekterkennung durch YOLOv2", dann würde eine Person eher eine Beschreibung wählen wie "Es handelt sich um einen Computerraum". Ein Netzwerk könnte jedoch nun alle Objekte in die Beschreibung einbauen und so etwas machen wie "Ein

Raum mit Computern auf Tischen, vor einem Regal mit Büchern". Es können jedoch nur die Objekte erkannt werden, die durch die Datengrundlage erlernt wurden. Im Bild ist z. B. zu erkennen, dass kein Whiteboard erkannt wird, obwohl eines hinten im Raum steht.

Das Ziel dieser Arbeit ist die Auswertung erstellter sowie trainierter Modelle, welche Beschreibungen für Bilder erstellen, die ähnlich die einer echten Person wären.

1.4 Struktur der Arbeit

Der Abschnitt Analyse beschäftigt sich mit den allgemeinen Anforderungen an das Ziel der Arbeit, sowie deren Umsetzbarkeit. Es wird analysiert welche Anforderungen sich für das Modell ergeben, für die technischen Aspekte sowie für die Validierung es Modells.

Die Grundlagen dienen zum Verständnis der verwendeten Fachbegriffe sowie Modelle/Architekturen. Sie sollen einen guten Überblick über die Grundlagen geben. Es werden Fachbegriffe erklärt, die für das Modell eine Rolle spielen. Einige Fachbegriffe können für das Verständnis nützlich sein. Sind diese Begriffe nicht entscheidend für das Modell, dann werden diese in ihrem Satz erklärt. Sofern für das Verständnis notwendig unterstützen Bilder die Erklärung.

Der Abschnitt Konzept beschreibt den theoretischen Aufbau des Modells und seiner Komponenten. Es wird darauf eingegangen, wie die Komponenten aufgebaut sein sollten. Die Datengrundlage zum Trainieren des Modells wird hier definiert. Darunter fällt welche Daten genommen werden und wie sie für das Modell geformt sein sollten. Das Grundkonzept für die verwendete Auswertung wird hier beschrieben. Es werden alle Entscheidungen begründet.

Die Realisierung beschäftigt sich mit der Umsetzung. Es wird beschrieben, wie das Konzept umgesetzt wird. Dabei konzentriert sich die Beschreibung auf eine generelle Implementierung. Es wird nicht erklärt, wie es genau im verwendeten Framework zu implementieren ist. Vielmehr werden die Schritte beschrieben, die für die Implementierung nötig sind. Der Schluss bildet eine Auswertung der Modelle und der Ergebnisse.

2 Analyse

2.1 Ziele/Anforderungen der Arbeit

Am Ende der Arbeit soll ein Modell eines trainierten neuronalen Netzwerkes stehen, welches Beschreibungen für Bilder ähnlich die einer Person erstellt. Dabei lassen sich die Anforderungen in die unten aufgeführten Punkte unterteilen und besser separat betrachten.

Objekterkennung

Wenn eine Person ein Bild beschreibt, dann kennt er in der Regel alle wichtigen Objekte, um den Kern bzw. die Aussage des Bildes zu beschreiben. Da das Modell am Ende Beschreibungen ähnlich die einer Person erstellen soll, muss auch das Modell möglichst alle Objekte richtig erkennen können. In einer Beschreibung finden sich normalerweise Relationen von Objekten untereinander, wie etwa "Die Katze liegt auf dem Tisch". Damit das Modell so etwas wie "Objekt A ist auf Objekt B" erkennt, müssen die Objekte im Bild lokalisiert werden, da sich dann die Relationen ableiten lassen. In den meisten Texten führen Adjektive dazu, dass sich der Text natürlicher anfühlt. Schaut man sich Sätze an wie "Der peitschende Wind, der die Regentropfen des kalten Regens gegen mein Fenster stoßen lies, weckte langsam eine ungewohnte, erdrückende Kälte in mir." und "Der Wind, der die Regentropfen des Regens gegen mein Fenster stoßen lies, weckte langsam eine ungewohnte Kälte in mir.", dann fühlt sich der erste Satz natürlicher an. Für das Modell wäre es daher ein Vorteil, wenn es passende beschreibende Adjektive für die Objekte erkennt. "Ein roter Vogel" beschreibt ein Objekt besser als "Ein Vogel" bleibt dabei aber nicht zu detailliert.

Textgenerierung

Das Erstellen eines natürlich klingenden Textes zeichnet die wichtigste Anforderung an die Textgenerierung aus. Weiter geht aus den bereits erwähnten Anforderungen her, dass die Beschreibung nicht zu detailliert sein soll. Fehler darf das Modell nicht im Hinblick auf die Grammatik machen, da sonst leichter zu erkennen wäre, dass die Beschreibung nicht von einer Person ist. Aus der Anforderung an ein natürlich klingenden Text geht hervor, dass nicht jedes Objekt einen Satz bekommen sollte.



(Vinyals u. a. 2015, S.8) – Hunde auf einer Wiese

Schaut man auf dieses Bild, dann sollte das Modell nicht so etwas erstellen wie "Ein hellbrauner Hund auf einer Wiese. Ein brauner Hund auf einer Wiese. Ein grauer Hund auf einer Wiese", sondern eher "Drei Hunde auf einer Wiese"

Validierung

Eine zufriedenstellende Validierung setzt eine Gruppe von Bildern voraus, die nicht zum Trainieren für das Modell bereits benutzt wurden. Diese Gruppe von Validierungsbildern müssen mit zwei Beschreibungstexten ausgestattet werden. Eine Beschreibung ist die einer Person, wobei nicht der Schreibstil der Person erkannt werden sollte. Die andere ist die des Modells. Nun kann eine Validierung der Beschreibungen stattfinden, hinsichtlich welche davon von einer Person erstellt wurde. Die Validierung sollte sich dabei auf die vorher gestellten Anforderungen konzentrieren, zum Beispiel dahin gehend welche Beschreibung klingt natürlicher oder beschreibt das Bild besser.

2.2 Umsetzbarkeit der Anforderungen

Objekterkennung

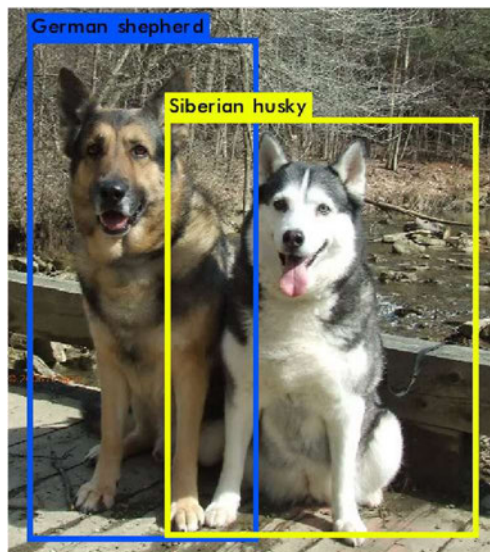
Schaut man sich verschiedene Internationale Veröffentlichungen zur Objektklassifizierung an, so bildet bei allen ein "Convolutional Neural Network" (CNN) die Grundlage für ihre Modelle.

Methods	Testing Accuracy
BOW-Color with Max-Pooling	84.00
BOW-Color with Sum-Pooling	82.40
BOW-Gray with Max-Pooling	82.00
BOW-Gray with Sum-Pooling	81.40
HOG-BOW-Gray with Sum-Pooling	82.60
HOG-BOW-Gray with Max-Pooling	78.40
HOG-BOW-Color with Sum-Pooling	73.20
HOG-BOW-Color with Max-Pooling	63.60
CNNs (proposed)	90.12

(Hayat u. a. 2018, S.197) – Vergleich von Methoden

Wie in der Tabelle zu sehen können CNNs eine Genauigkeit von >90 % aufweise, bei der Klassifizierung von Bildern. Je nach Datensatz oder Objekte kann diese Genauigkeit auch höher sein. In den meisten Modellen findet sich jedoch kein einfaches CNN.

Je nach Ziel der Veröffentlichung variiert das CNN. Für die Objekterkennung finden meist sogenannten "region-based CNN" (RCNN) Anwendung, wie "Fast RCNN" oder "Faster RCNN". Weitere Ansätze versprechen "Region-based Fully Convolutional Networks" (RFCN). Veröffentlichungen wie "Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors" (Huang u. a. 2017) zeigen es gibt nicht das beste Modell, sondern es ist eher eine Frage der Ressourcen. Entscheidende Faktoren sind daher das Anwendungsgebiet des Modells (z. B. Echtzeit). Hat das Netzwerk viel oder wenig Festplattenspeicher oder RAM zur Verfügung. Aktuell beschränken die Datensätze die Anzahl der zu erkennenden Objekte und weiterer Unterklassen. Modelle wie YOLO9000 (vgl. Redmon u. a. 2017) versprechen Objekte aus über 9000 Kategorien erkennen zu können und sollen selbst für Echtzeiterkennung zu verwenden sein. Darunter sollen auch Unterklassen sein wie folgend gezeigt.



(Redmon u. a.: Slides 2017, Folie 41) – Zwei Hunde

Textgenerierung

Recurrent neural networks (RNN), long-short-term memory (LSTM) und gated recurrent neural networks (GRU) im speziellen haben sich als Stand der Technik, bei der Modellierung von Sequenzen und Übertragungsprobleme, wie Sprachmodellierung und maschinelle Übersetzung etabliert (vgl. Vaswani u. a. 2017, S.5998-5999). Dieser Stand geht auch aus aktuellen wissenschaftlichen Veröffentlichungen hervor wie "*Deep Visual-Semantic Alignments for Generating Image Descriptions*" (Karpathy u. a. 2017), die gute Ergebnisse generieren. Weiter Veröffentlichungen nutzen auch CNNs, dies jedoch eher für die Textanalyse als für die Textgenerierung.

Validierung

Eine einfache, aber akzeptable Validierung, lässt sich durch eine Umfrage erhalten. Es soll validiert werden, inwieweit die durch das Modell erstellen Beschreibungen für eine Person besser sind, als welche die durch eine andere Person erstellt worden ist. Um einen Vergleich anzustellen benötigt es eine Gruppe von Bildern die sowohl vom Modell als auch von einer Person beschreiben werden. Eine Umfrage kann nun abfragen, welche Beschreibung zutreffender ist, welche nicht von einer Person ist oder was die eine Beschreibung explizit besser macht als die andere. Die Umfrage sollte alle Anforderungen an das Modell zumindest implizit abfragen bzw. bedenken, welches sich durch verschiedene Fragestellungen zu den Bildern ergeben würde.

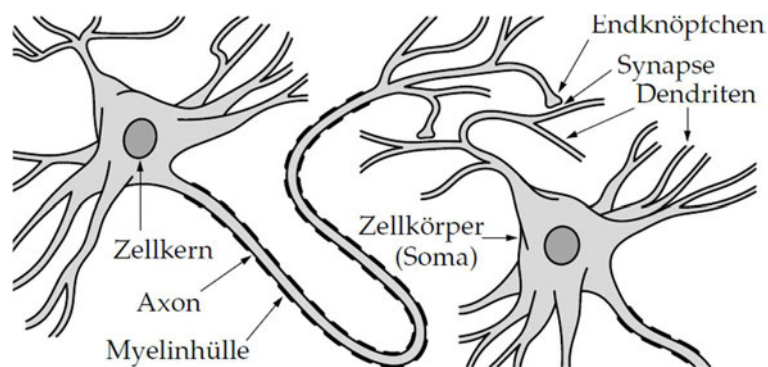
Generelle Umsetzbarkeit

Wenn die Datengrundlage gut genug ist, dann lässt sich das Ziel soweit Umsetzen, das eine Validierung stattfinden kann. Es gibt verschiedene Ansätze bei der Objekterkennung/-lokalisierung, aber wie bei der Textgenerierung gibt es auch dort einen Standard. Generell gibt es für die Objekterkennung sowie der Textgenerierung auch bereits trainierte Modelle, die benutzt werden könnten. Diese Entscheidung hängt davon ab wie aufwendig die Beschaffung von ausreichend guten Datensätzen, für das Training der Modelle ist.

3 Grundlagen

3.1 Neuronale Netze

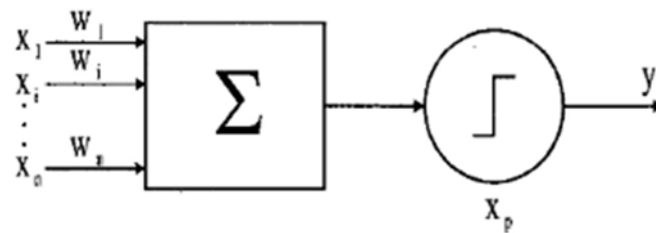
Unter neuronalen Netzen versteht man ein Modell, welches unserem Nervensystem bzw. unserem Gehirn nachempfunden ist. In unserem Nervensystem ist die kleinste kommunizierende Einheit ein Neuron, auch Nervenzelle genannt. Die Neuronen in unserem Körper gehen zwischen 1.000 und 10.000 Verbindungen mit anderen Neuronen ein (vgl. Kruse u. a. 2011, S.10).



(Kruse u. a. 2011, S.10) - Prototypischer Aufbau biologischer Neuronen

Ein Neuron ist wie in der obigen Abbildung zu sehen aufgebaut. Vom Zellkörper geht genau ein Axon ab und eine Vielzahl an Dendriten. Vereinfacht gesagt ist das Axon ein Sender und die Dendriten sind Empfänger. Am Ende des Axons gibt es eine Vielzahl von Verzweigungen, deren Enden man Endknöpfchen nennt. Diese Endknöpfchen sind so angelegt, dass sie die Dendriten von anderen Neuronen fast berühren, wobei diese kleine Lücke Synapse genannt wird. Neurotransmitter, die durch die Endknöpfchen an die Membran der Dendriten gesendet werden, führen zur Änderung des elektrischen Potenzials des Neurons (vgl. Kruse u. a. 2011, S.10). Die Änderung des elektrischen Potenzials muss einen gewissen Schwellenwert übersteigen, damit das Neuron ein Signal an andere Neuronen weitersendet (vgl. Kruse u. a. 2011, S.11).

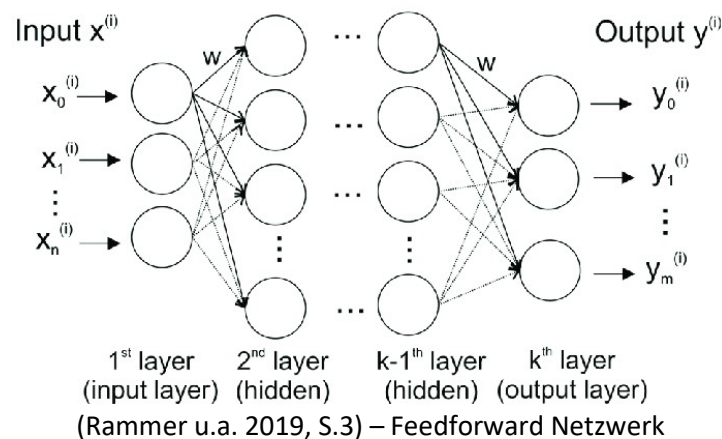
Ausgehend von dem groben Aufbau und der Funktion der echten Neuronen, kamen schon 1943 Warren McCulloch und Walter Pitts mit einem einfachen Modell an, dem sogenannten McCulloch-Pitts-Neuron.



(Degeratu u. a. 2001, S.210) – Aufbau eines McCulloch-Pitts Neurons

Das künstliche Neuron berechnet eine Funktion, in diesem Fall eine Summe aus gewichteten Eingabewerten und sobald diese Summe einen Schwellenwert überschreitet, dann ist die Ausgabe der logische Zustand 1 andernfalls 0 (vgl. Degeratu u.a. 2001, S.211). Dieses Modell repräsentiert so gut ein echtes Neuron, das es seit gut 77 Jahren immer noch die Grundlage für aktuelle neuronale Netzwerke bildet. Das Einzige, das sich in verschiedenen Netzwerken ändert, ist die Form der Eingabewerte (z. B. 0, 1, Matrizen etc.), die Funktion für den Schwellenwert und die Topologie des Netzes selbst.

Wie das biologische Vorbild so werden auch bei neuronalen Netzen viele künstliche Neuronen miteinander vernetzt. Die Daten von den ersten Neuronen (Input Layer) wandern durch das Netzwerk bis zu den letzten Neuronen (Output Layer), die dann zu interpretieren sind je nach Aufgabe des Netzwerkes.



Hat man ein Netzwerk wie das obige und möchte Bilder klassifizieren, ist Folgendes ein veranschaulichendes Beispiel. Dem Netzwerk werden die Pixel als Eingabe des zu klassifizierenden Bildes gegeben. Die letzten Neuronen (Output Layer) machen durch ihre Funktion für den Schwellenwert eine Ausgabe. Enthält die Ausgabe nun fünf Neuronen mit den Werten $[1, 0, 0, 0, 0]$, dann kann das als Katze interpretiert werden, sofern das Netzwerk dahingehend trainiert wurde. Das heißt, beim Training wurde dem Netzwerk mitgeteilt, dass ein Bild mit einer Katze der Ausgabe entspricht.

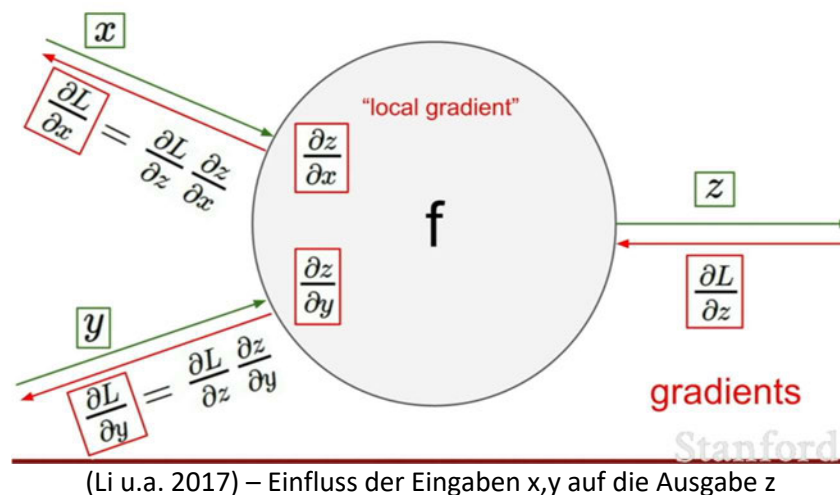
Wie im obigen Bild zu sehen ist, lässt sich ein Netzwerk ab einer gewissen Größe bzw. Neuronen Anzahl nicht mehr verständlich darstellen. Neuronale Netzwerke werden daher meist in ihren Schichten (engl. Layer) dargestellt, wobei der Output einer Schicht der Input der nächsten Schicht im Netzwerk ist. Wandern die Daten in einer Richtung durch das Netzwerk, dann spricht man von sogenannten "Feedforward Netzwerken". Im Kontrast dazu stehen sogenannte rekurrente Netzwerke, in dem die Daten auch zurückgehen können.

3.2 Backpropagation

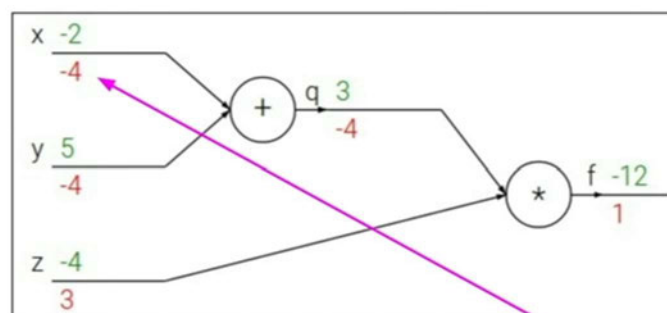
Der Mensch ist ein Produkt seiner Erlebnisse. Menschen und Tiere können Unbekanntes nicht klassifizieren. Ein Hase lässt sich als solcher klassifizieren, wenn das Erlebnis verarbeitet wurde. Die Verarbeitung kann in dem Fall das Lernen von einem neuen Tier sein. Danach lassen sich begrenzt Aussagen über unbekannte Hasenarten treffen, da nur das eine Erlebnis existiert.

Das Ziel des Backpropagationalgorithmus ist es, die Gewichte so zu modifizieren, dass das Netz trainiert wird, um beliebige Eingaben korrekt auf die Ausgaben abzubilden (vgl. Amrutha 2018, S. 2). Um bei dem Beispiel mit dem Hasen zu bleiben, sorgt Backpropagation dafür, dass ein neuronales Netzwerk bereits gesehene Hasen besser erkennt. Daraus folgt das andere Tiere, wie z. B. Katzen weniger gut erkannt werden. Daraus lässt sich indirekt ableiten, was das Netzwerk gelernt hat. Nämlich zwischen Hasen und nicht Hasen zu unterscheiden.

Damit ein Netz überhaupt lernen kann, muss es einen Wert haben, der eine Fehlerquote darstellt. Diese Fehlerquote wird durch die Verlustfunktion (engl. "loss function") berechnet. Backpropagation versucht nun mittels Modifizierung der Gewichte der Eingaben, diesen Wert der Verlustfunktion gegen null zu bringen. Null bedeutet es gab keinen Fehler. Um die Gewichte nun korrekt zu modifizieren, ist es notwendig, den Einfluss der einzelnen Gewichte auf die Ausgabe zu berechnen. Da nicht jedes Neuron direkt mit der Ausgabe in Verbindung steht, muss der Einfluss der Neuronen von hinten nach vorne berechnet werden, daher auch der Name Backpropagation.



Der Einfluss eines Neurons und der Einfluss seiner Eingaben, lässt sich wie im oberen Bild zusehen, anhand der Ableitungen berechnen. Der sogenannte lokale Gradient (engl. "local gradient") bezeichnet den Einfluss der Ausgabe eines Neurons. Also er ist ein Richtwert, wie sehr die Ausgabe z das nächste Ergebnis verändern würde, sollte sich z verändern. Hat man ein Ergebnis L , so lässt sich der Einfluss der Ausgabe z berechnen, indem man die Funktion die L berechnet nach z ableitet ($\partial L / \partial z$). Angenommen L wird durch $f(L) = 2 * z + 1$ berechnet, dann ist $\partial L / \partial z = 2$. Der lokale Gradient gibt durch die zwei an, dass wenn sich z um einen Wert ändert, dann ändert sich L um zwei. Durch das Ableiten nach den jeweiligen Eingaben ($\partial z / \partial x$, $\partial z / \partial y$) der Funktion die z berechnet, lässt sich genauso deren Einfluss bestimmen. Der Einfluss einer Eingabe für z auf das Ergebnis L ($\partial L / \partial x$, $\partial L / \partial y$) lässt sich durch die sogenannte Kettenregel berechnen. Die Kettenregel für x ist $\partial L / \partial x = \partial L / \partial z * \partial z / \partial x$. Dieser lokale Gradient gibt an, um wie viel sich der Wert L ändert, wenn sich x ändert.



(Li u.a. 2017) – Backpropagation an einem Graph

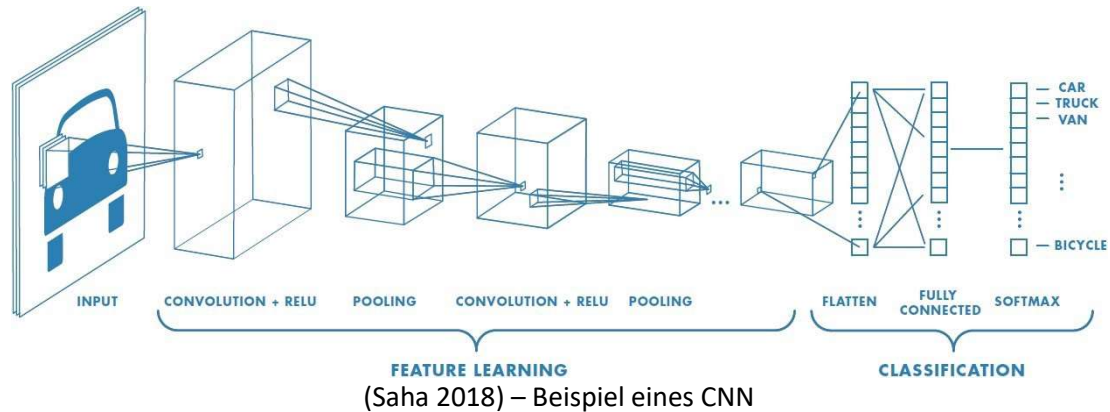
Das obige Bild zeigt ein einfaches Beispiel für Backpropagation. Der gezeigte Graph ist ein sehr einfaches neuronales Netz. Das erste der beiden Neuronen enthält eine Addition als

Funktion, das zweite eine Multiplikation. Dabei sind x , y und z die Eingaben für die jeweiligen Neuronen und f die Ausgabe des neuronalen Netzes. Der lokale Gradient (rot) der Eingabe y gibt eine Auskunft darüber, wie stark y die Ausgabe f verändern wird. Dabei ist der Einfluss -4 auf das Ergebnis f . Erhöht man die Eingabe um 1 dann ändert sich das Ergebnis um -4 zu -16 oder reduziert man es um 1 dann ändert es sich zu -8.

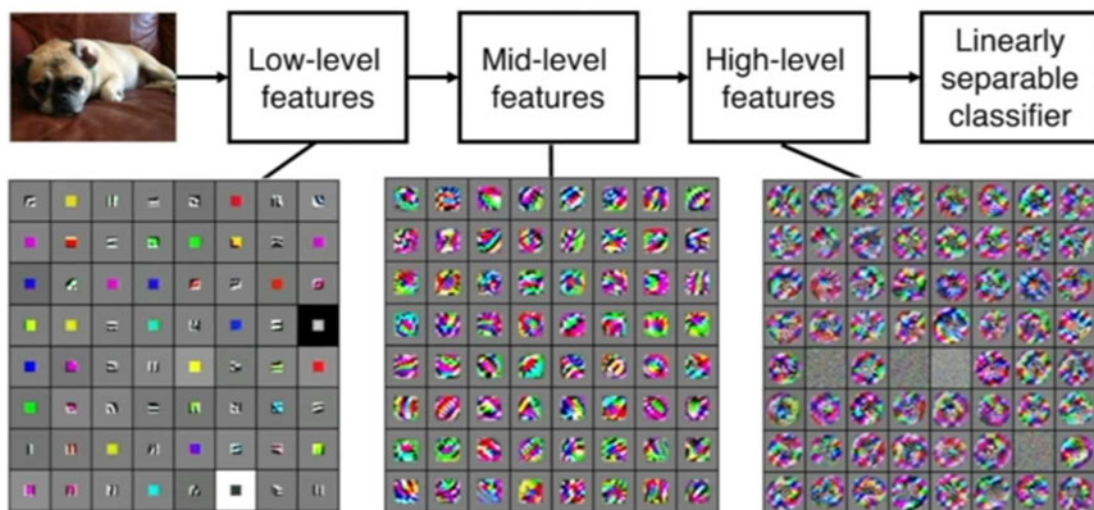
Die Gewichte, die durch Backpropagation geändert werden, sind eigene Werte, die mit der jeweiligen Eingabe multipliziert werden. Hat ein Neuron die Funktion $f(x, y) = x * y + 1$ ohne Gewichte, dann ist die Funktion mit Gewichten $f(x, y) = (x * w_x) * (y * w_y) + 1$. Die Gewichte sind dabei Konstanten, die durch Backpropagation geändert werden. Backpropagation kann nun durch ändern dieser Gewichte den Einfluss der Eingaben verändern, damit die Eingaben korrekt auf die Ausgaben abgebildet werden. Die Gewichte werden bei jedem Durchlauf so geändert das die Verlustfunktion gegen null geht, da dies ein Maß für die Korrektheit der Abbildung ist.

3.3 Convolutional Neural Network (CNN)

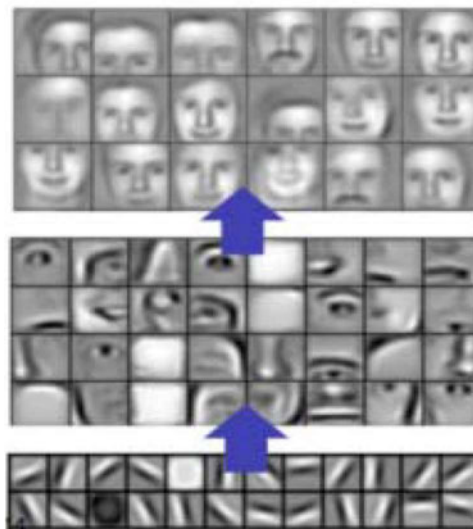
Sogenannte CNNs, zu Deutsch etwa "Faltungsnetze", bilden in den meisten Fällen den Grundstein für die Klassifizierung von Bildern oder der Objekterkennung. Sie eignen sich aufgrund ihrer Funktionsweise hervorragend für die Erkennung von Mustern (engl. Features).



Ein CNN besteht aus einigen verschiedenen Schichten wobei die namensgebenden Faltungsschichten (engl. "convolutional layer") die Muster in einem Bild erkennen. Das CNN hat dadurch den Vorteil das es die Position von Objekten innerhalb des Bildes nicht bestimmen braucht, um ein Objekt zu erkennen. "Mit anderen Worten, brauchen wir bei einer Gesichtserkennung nicht darauf zu achten, wo sich die Gesichter in dem Bild befinden. Die einzige Sorge ist sie unabhängig von ihrer Position im Bild zu erkennen." (vgl. Albawi u.a. 2017, S. 1). Mit diesem Ansatz der CNN, können jedoch allgemeine Klassifizierungen gemacht werden, wie in etwa das Bild enthält eine Katze und einen Hund.



Wandern die Daten nun durch das Netzwerk, dann werden die erkannten Muster für den Menschen immer abstrakter. Die Schichten werden mit zunehmender Position im CNN kleiner, dadurch betrachten spätere Schichten mehrere Muster der vorherigen Schicht. Die Schichten lassen sich daher in einem CNN in "low-level features" (Low), "mid-level features" (Mid) und "high-level features" (High) einteilen. Die Low-Schichten können z. B. Kanten oder Farbunterschiede erkennen. Regionen wie Auge, Mund oder Nase würden durch die Mid-Schichten erkannt, da mehrere Muster von den Low-Schichten betrachtet werden. Die High-Schichten könnten dann aus den Mustern der Mid-Schichten ganze Gesichter erkennen.



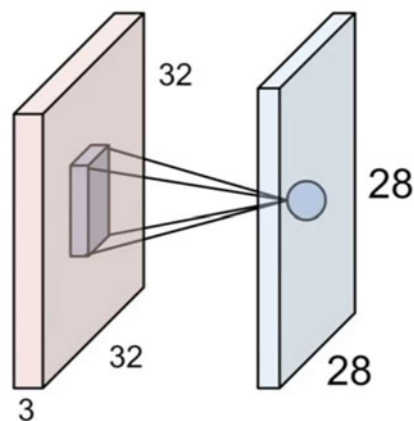
(Albawi u.a. 2017, S. 1) – Filtersicht der Schichten

Am Ende hat ein CNN noch einen ein dimensional Feature-Vektor, der eine Aussage über die Klassifizierung der Bilder trifft. Dies ist möglich durch die vorherigen High-Schichten.

3.3.1 Convolutional Layer / Faltungsschicht

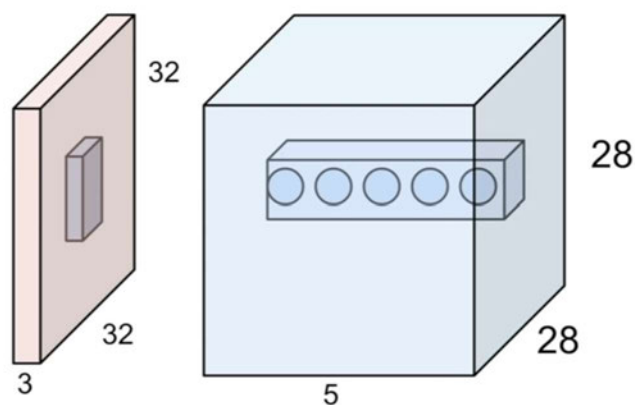
Filter

Die Erkennung von Mustern erfolgt innerhalb einer Faltungsschicht. Durch sogenannte Filter, wobei der Filter das Bild Stück für Stück abtastet. Jedes Abtasten erschafft ein Neuron, welches die Information des Filters für diesen Abschnitt des Bildes beinhaltet. Die Ausgabe aller Neuronen des Filters nennt sich im englischen "activation map" und stellt dar, was der Filter erkannt hat.



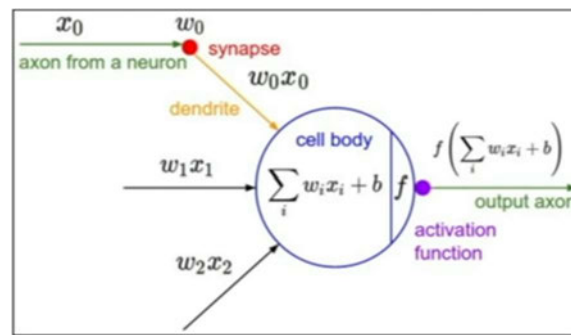
(Li u.a. 2017) – "activation map"

Das obige Bild zeigt, wie das Anwenden eines solchen Filters aussieht. Das äußere linke Rechteck ist das Bild, welches klassifiziert werden soll und das blaue Rechteck ist die "activation map". Der Kreis ist ein Neuron, welches durch den Filter entstand, an der Stelle, wo der Filter gerade im Bild ist. Der Filter hat eine Größe von $3 \times 5 \times 5$ und das Neuron gibt genau einen Wert für die "activation map" wieder. Das führt dazu, dass die "activation map" selbst $1 \times 28 \times 28$ groß ist.



(Li u.a. 2017) – Komplette "activation map"

Da ein Filter ein Muster erkennt, würden wichtige Merkmale verloren gehen, wenn eine Faltungsschicht nur ein Filter nutzt. Die genaue Anzahl an Filtern pro Schicht ist ein Parameter, der optimiert werden kann. Eine höhere Anzahl an Filter resultiert auch in einer höheren Anzahl an Neuronen. Eine höhere Anzahl an Neuronen resultiert dann in höheren Rechenzeiten. Die Faltungsschicht im Bild hat z. B. 3920 Neuronen, da jeder Punkt in der "activation map" durch ein Neuron gegeben wurde.



(Li u.a. 2017) – Neuron in einer Faltungsschicht

Obiges Bild zeigt eine detailliertere Ansicht der Neuronen. Die x Werte sind die Pixel des Bildes, die der Filter gerade betrachtet. Die w Werte sind die Gewichte des Filters, welche zufällig bei der Initialisierung erstellt wurden. Wie bei einem echten Neuron werden alle Signale (Werte $x_i \cdot w_i$) summiert und ausgewertet. Im Fall eines künstlichen Neurons mittels einer sogenannten Aktivierungsfunktion (engl. "activation function"). Daher auch der Name "activation map", da sie die Aktivierung aller Neuronen auf Basis des Filters anzeigt. Für diese Funktion gibt es einige Ansätze, wobei einer der meist genutzten die sogenannte ReLU-Funktion ist. Die ReLU-Funktion lässt sich durch $\text{ReLU}(x) = \max(0, x)$ beschreiben. Ausschließlich positive Werte sind zugelassen, alle anderen werden zu einer Null geändert.

Padding

Das kleiner werden der Höhe und Breite von Schicht zu Schicht liegt daran dass der Filter z. B. 9 Felder auf eines reduziert, das dazu führt dass Höhe und Breite um jeweils zwei Werte reduziert werden. Jedoch können Muster verloren gehen, die sich am Rand eines Bildes befinden. Um dies zu verhindern gibt es für die Faltungsschichten das sogenannte Padding. Das Padding sorgt dafür, dass die Höhe und Breite um eine Größe erhöht wird, welche mit null ausgefüllt wird. Einfach ausgedrückt, das Bild bekommt einen Rahmen mit Nullen, damit diese nicht die anderen Werte beeinflussen. Durch den Rahmen ist nun die Höhe und Breite im Input um jeweils zwei erhöht, sodass die Ausgabe die gleichen Höhe und Breite hat wie der Input vor dem Padding.

Stride

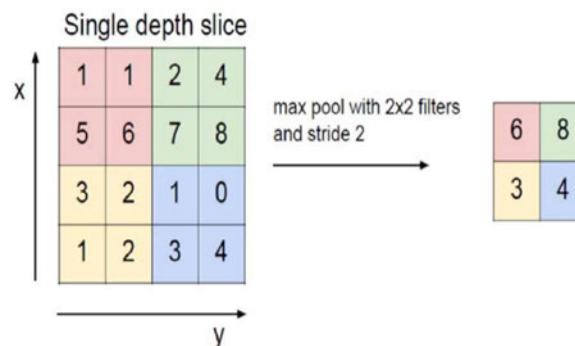
Der sogenannte Stride gibt Auskunft darüber, in welcher Schrittzahl das Abtasten durchgeführt wird. Meist wird ein Stride von Eins genutzt, so dass der Filter von i nach $i+1$ wandert. Ein höherer Stride würde bedeuten, dass es weniger Überschneidungen in den Neuronen gibt da jeder mehr von "seiner" Region hat. Die "activation map" würde dementsprechend als Resultat kleiner ausfallen. Die weniger Neuronen kommen der Rechenzeit zugute.

Zusammengefasst lässt sich sagen, Faltungsschichten lassen N -Filter mit zufällig initialisierten Werten über einen Input ($Y \times M \times M$) wandern und erstellen eine $N \times M \times M$ große "activation

map", die als Output an die nächste Schicht gegeben wird. Durch Backpropagation werden die Werte in den Filtern modifiziert, um die Muster besser zu erkennen und eine korrektere Ausgabe zu ermöglichen.

3.3.2 Pooling Layer

"Die Grundidee vom Pooling ist das down-sampling um die Komplexität für spätere Schichten zu reduzieren" (vgl. Albawi 2017, S. 5). Die reduzierte Komplexität spiegelt sich darin wider, dass das Netzwerk weniger Neuronen enthält. Der Input von $3 \times 32 \times 32$ würde auf $3 \times 16 \times 16$ reduziert werden, dass auch die Muster kleiner bzw. unkenntlicher macht. Beim Pooling gibt es auch verschiedene Ansätze, genau wie bei der Aktivierungsfunktion in den Faltungsschichten. Das Max-Pooling ist jedoch eines der häufigsten Pooling Methoden. (vgl. Albawi 2017, S. 5). Beim Max-Pooling wird der höchste Wert, aus dem betrachtenden Fenster genommen. Genau wie bei den Faltungsschichten mit den Filtern, wandert das Fenster hier auch über den Input. Da hier auch ein Fenster über den Input wandert, gibt es auch den Stride. Der Stride entspricht üblicherweise der Größe des Fensters. Das am häufigsten verwendete Max-Pooling nutzt einen 2×2 -Filter und einem Stride von zwei. In der Regel wird bei einer ungeraden Anzahl der Breite oder Höhe der resultierende Output aufgerundet. Wo bei den Faltungsschichten noch die Filter alle drei Dimensionen (Tiefe, Höhe, Breite) betrachtet haben, reduziert die Pooling-Schicht jede einzelne Schicht der Tiefe nacheinander, sodass die Tiefe nicht halbiert wird.



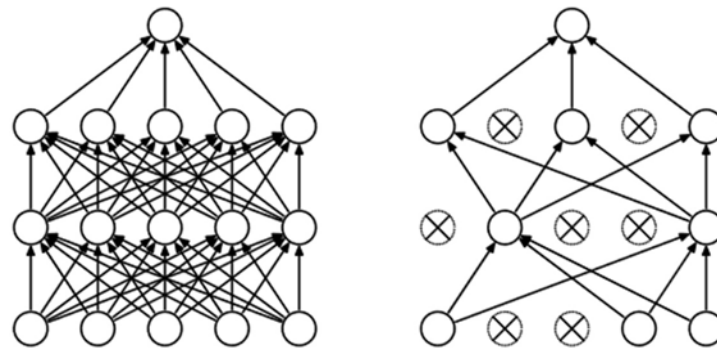
(Albawi 2017, S. 5) – Max-Pooling auf eine Tiefe

3.3.3 Linear Layer

Eine Linear-Schicht wird verwendet, um die Größe einer Dimension zu verändern. Diese Änderung wird durch eine Abbildung realisiert. Durch die Abbildung ist gegeben, dass bei selben Input auch derselbe Output herauskommt. Erzielen lässt sich diese Abbildung durch eine Matrizenmultiplikation. Ein Linear-Schicht enthält eine Gewichtsmatrix. Eine Gewichtsmatrix der Dimension 100×50 bildet nun 100 Werte auf 50 ab.

3.3.4 Dropout

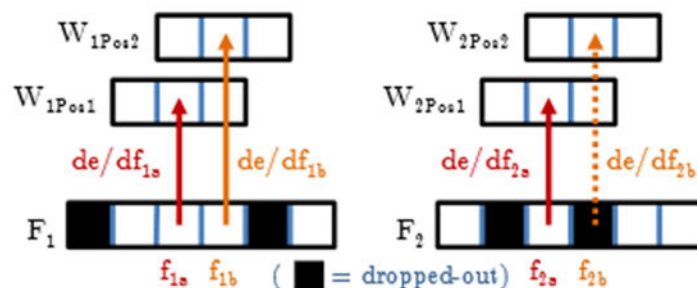
Die Kernidee ist es zufällig ausgewählte Einheiten (samt ihrer Verbindungen) während des Trainings nicht weiter zu betrachten (vgl. Srivastava u.a. 2014, S. 1929). Damit wird versucht eine Überanpassung (engl. "overfitting") zu verhindern.



(Srivastava u.a. 2014, S. 1930) – Dropout bei einem Feedforward Netz

Ein Netzwerk ist überangepasst, wenn es die Trainingsdaten erkennt sonst jedoch nichts. Das Netzwerk kann nicht mehr generelle Aussagen treffen, da es die Trainingsdaten auswendig gelernt hat. Welche Einheiten vom Dropout betroffen sind, wird zufällig bestimmt. Im einfachsten Fall wird jede Einheit mit einer festen Wahrscheinlichkeit p behalten, unabhängig von anderen, wobei p aus einem Validierungssatz stammen kann oder einfach als 0.5 gesetzt werden kann, das eher dem Optimum für viele Netze und Aufgaben entspricht (vgl. Srivastava u.a. 2014, S. 1930).

Bei CNNs gibt es folgendes Problem, welches in "Efficient Object Localization Using Convolutional Networks" von Tompson u. a. 2015 erklärt wurde. Die "activation maps" in einem CNN besitzen eine starke räumliche Korrelation, genau wie natürliche Bilder. Bei diesem Aufbau scheitert der "normale" Dropout.



(Tompson u.a. 2015, S. 651) – "Normaler Dropout" nach einem 1D-Convolutional Layer

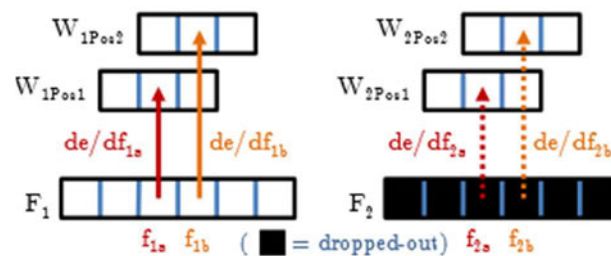
In dem zu sehendem Bild sind F_1 und F_2 jeweils eine "activation map" aus einer Faltungsschicht. Die zwei oberen Reihen (W_1 und W_2) sind die jeweiligen Pixel, die von dem

Filter für das Neuron an der Position f betrachtet wurden. Die vorher erwähnte starke räumliche Korrelation ist daran zusehen, dass zwei nebeneinanderliegende Neuronen auch zwei gleiche Pixel betrachten.

In der "activation map" F_1 gilt $f_{1a} \approx f_{1b}$ für deren lokale Gradienten, da sie sich zwei von drei Pixel teilen. Auch ist deren Differenz im Einfluss von dem unterschiedlichen Pixel abhängig. Daraus lässt sich folgern, dass auch in F_2 dasselbe gelten muss, da sie aus demselben Faltungsschicht stammen. Die starke Korrelation, die in F_1 noch herrscht, ist in F_2 nicht mehr vorhanden, wenn f_{2b} durch den Dropout rausfällt. Da f_{2b} nun keinen Einfluss mehr auf die Schicht danach hat, wird durch Backpropagation der lokale Gradient null sein. Der lokale Gradient vom stark korrelierenden f_{2a} ist aber nicht null.

Im Wesentlichen wird die effektive Lernrate skaliert mit der Wahrscheinlichkeit p vom Dropout, dabei ist eine Unabhängigkeit der beiden Werte nicht garantiert (vgl. Tompson u.a. 2015, S. 651). In ersten Experimenten fanden wir, dass die Anwendung eines normalen Dropouts vor einer 1×1 Faltungsschicht generell die Trainingszeit erhöht hat, aber das Überanpassen nicht verhindert hat (vgl. Tompson u. a. 2015, S. 651).

Tompson und sein Team kamen auch gleich mit einer Lösung an, nämlich dem "Spatial Dropout". Anstelle einzelner Neuronen bzw. deren Ausgabe in der "activation map" auf null zu setzen ("normaler" Dropout), setzten sie eine ganze "activation map" auf null.



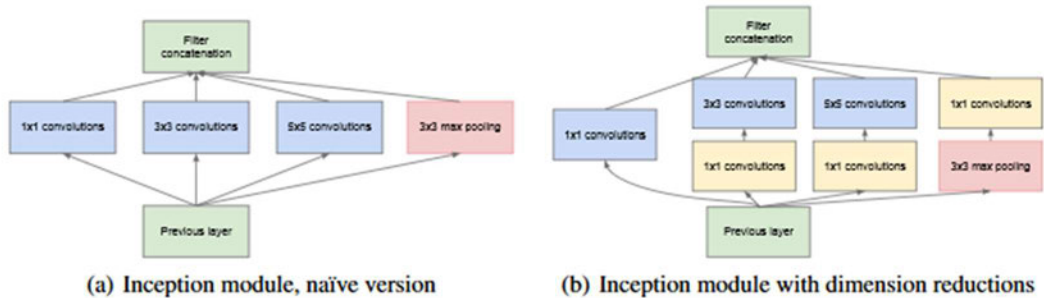
(Tompson u.a. 2015, S. 652) – "Spatial Dropout"

Tompson und sein Team haben herausgefunden, dass diese modifizierte Dropout Implementation die Performance verbessert, besonders bei kleineren Datensets wie dem FLIC Datenset (vgl. Tompson u.a. 2015, S. 651).

3.3.5 Softmax

Beim Softmax wird ein Vektor transformiert, sodass seine Werte nun im Bereich von null bis eins liegen. Kernmerkmal beim Softmax ist das die entstandenen Werte summiert eins ergeben. Softmax findet dadurch auch Anwendung in CNNs. Der Ausgabevektor, auch Klassifizierungsvektor genannt, benötigt den Softmax. Ohne Softmax kann der Klassifizierungsvektor keine Aussage über die Wahrscheinlichkeit für die Klassen tätigen.

3.3.6 Inception Layer

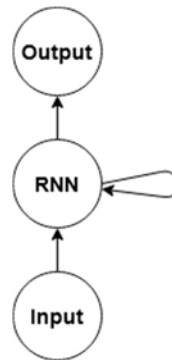


(Szegedy u.a. 2014, S.5) – GoogLeNet Inception Schicht

Das Prinzip von einer Inception-Schicht ist es verschiedene Schichten auf dieselbe Eingabe anzuwenden. Die Ausgaben der verschiedenen Schichten werden dann zusammengeführt und als Ausgabe der Inception-Schicht betrachtet. Wie im Film "Inception" schaut die Schicht mit verschiedenen Filtern auf die Eingabe. Umgangssprachlich findet ein Zoom statt. Die Schichten schauen mit verschiedenen Größen auf die Eingabe. Daher heißt das wissenschaftliche Paper aus denen diese Schicht stammt "Going deeper with convolutions".

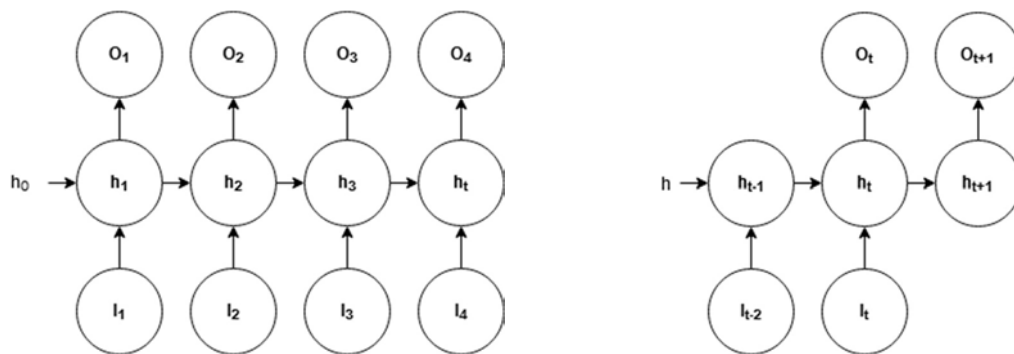
3.4 Recurrent Neural Network (RNN)

Im Gegensatz zu CNNs eignen sich RNNs besonders für sequenzielle Daten. Wenn es um Text geht finden sie aufgrund ihrer sequenziellen Verarbeitung häufig Anwendung. Anders als bei anderen neuronalen Netzen, können ihre Eingaben und Ausgaben unterschiedliche Größen haben.



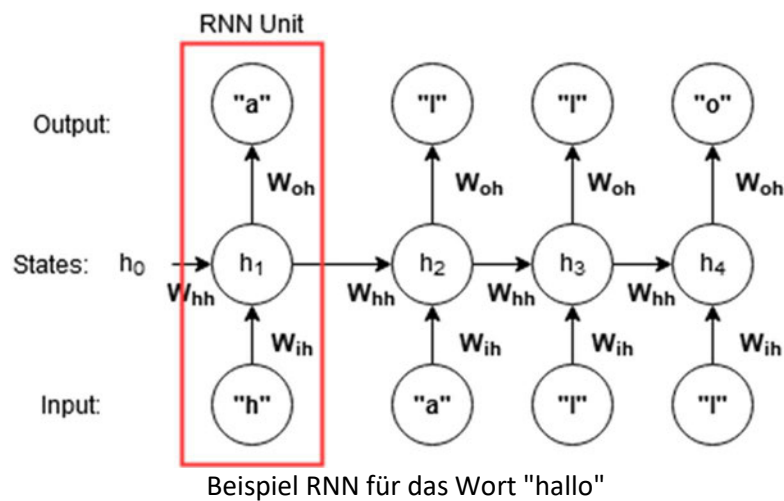
Einfaches Recurrent Neural Network

Der Name "Recurrent" (dt. Rekurrent/Wiederkehrend) kommt durch die Funktionsweise des Netzwerks. Ein RNN hat ein "Gedächtnis" für jeden Punkt in den Sequenzen in der Form von sogenannten "States". Diese States beeinflussen den nächsten Punkt in einer Sequenz. Abhängig vom Input/Output eines RNNs spricht man von 'one-to-one', 'one-to-many', 'many-to-one' und 'many-to-many' RNNs. Ein 'one-to-many' RNN nimmt einen Wert als Input und hat eine Sequenz als Ausgabe. Eine Übersetzung lässt sich durch ein 'many-to-many' RNN erzielen.

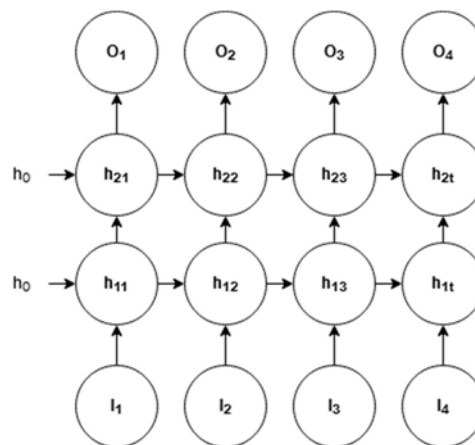


Zwei 'many-to-many' RNN's aufgerollt

Wie schon bei den CNNs ist das Training dafür zuständig bestimmte Gewichte so zu verändern, dass ein gewünschtes Ergebnis erzielt wird. Das gewünschte Ergebnis ist, dass jeder State den Input des nächsten States vorhersagt.



Im Bild ist ein RNN zu sehen, welches korrekt das Wort "hallo" gelernt hat. Soll das RNN mit Sätzen anstelle mit Wörtern umgehen, dann würde jede Unit das nächste Wort vorhersagen. Jede Unit nutzt für die Berechnung folgende allgemeine Formel: $h_t = f_W(h_{t-1}, I_t)$. Die Funktion kann variieren, jedoch wird meistens die tanh Funktion genutzt. Der zweite State h_2 würde durch $\tanh(W_{hh}h_1 + W_{ih}I_2 + b_2)$ berechnet werden. Jeder State bekommt dabei dieselben Gewichte W_{hh}, W_{ih} . Diese Gewichte sind die, die im RNN trainiert werden. Bei b handelt es sich um eine optionale Befangenheit (engl. Bias) für den State. Damit die Unit für State h_2 das "l" vorhersagen kann, wird der State mit dem lernenden Gewicht W_{oh} multipliziert. Genau wie bei einem CNN kann nun mittels Softmax eine Klassifikation vorgenommen werden.



Zwei Schichten RNN

Genau wie bei einem CNN können mehrere Schichten das Ergebnis verbessern. Bei einem N-Schichten RNN ist die Ausgabe der einen Schicht die Eingabe der nächsten Schicht. In anderen Kontexten haben tiefere Modelle gezeigt, dass diese dazu tendieren, besser zu performen, daher findet man für die meisten Probleme RNNs mit zwei oder drei Schichten. (vgl. Johnson, 2017, 51:05-51:20)

Einige RNNs können auch bidirektional sein. Durch diesen Schritt laufen die Daten das Model noch einmal durch. Bei dem zweiten Durchlauf wird die andere Richtung genommen. Daraus ergibt sich, dass der Vorgänger versucht wird zu bestimmen.

3.5 Embedding

Ein Embedding findet statt, wenn sich die Eingabewerte nicht verwenden lassen. Da mathematische Funktionen genutzt werden, muss es sich bei den Eingabewerten um Zahlen handeln.

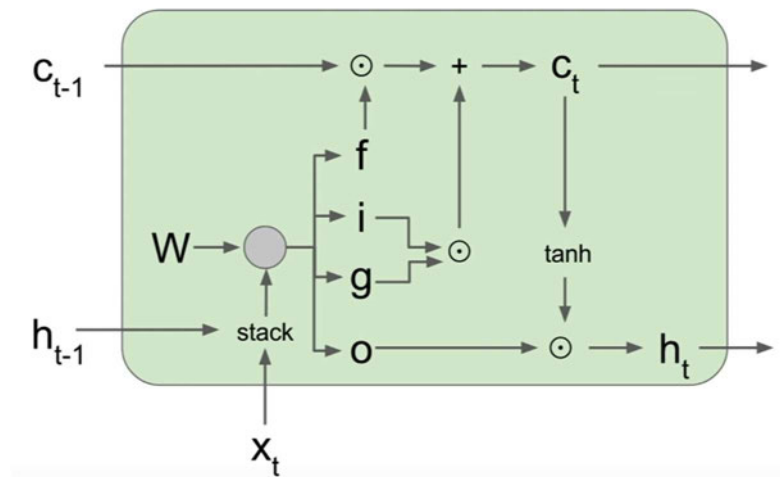
Beim Embedding werden die Eingabewerte in Vektoren von Zahlen umgewandelt, die als Repräsentationen für die jeweiligen Werte stehen. Beim Umgang mit Sprache hat man meist Sätze als Ausgangslage. Diese Sätze lassen sich in Tokens aufteilen, wobei bis auf das Leerzeichen jeder Bestandteil des Satzes zu einem Token wird. Damit die Umwandlung für dasselbe Token immer gleich ist werden die Repräsentationen gespeichert, sodass eine Art Wörterbuch entsteht, indem der Vektor zu jedem Token enthalten ist.

Der Hauptvorteil einer kompakten Darstellung ist die Verallgemeinerung: Wenn geglaubt wird, dass einige Merkmale ähnliche Anhaltspunkte liefern können, lohnt es sich, eine Darstellung anzubieten, die diese Ähnlichkeiten erfassen kann (vgl. Goldberg 2017, S. 92).

3.6 Long Short-Term Memory (LSTM)

Bei einem LSTM handelt es sich um ein komplexeres RNN. Die Architektur eines RNNs kann zum sogenannten "vanishing gradient problem" führen. Dabei erhöht die Länge der Sequenz auch die Wahrscheinlichkeit für das Problem. Der Gradient repräsentiert die Änderung für die Gewichte. Durch Backpropagation kann es dazu führen, dass der Gradient immer weiter gegen null geht. Dies passiert umso weiter Backpropagation zum Anfang des Modells wandert. Ein Gradient von null bedeutet, dass das Gewicht nicht geändert werden muss. Das Problem führt dazu, dass der Anfang eines Netzes nicht trainiert wird.

Ein LSTM umgeht dieses Problem durch eine komplexere Architektur.



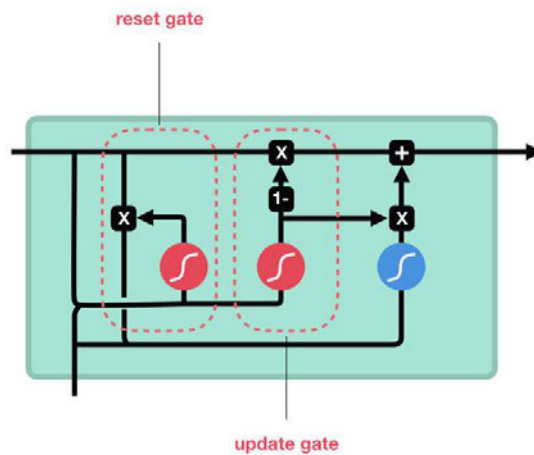
(Li u.a. 2017 2) – LSTM Unit Cell/State

Ein LSTM unterscheidet sich durch seine sogenannten Gates (f, i, g, o) und seinen Zellzustand. Der Zellzustand fungiert als Transportautobahn, die relative Informationen auf dem gesamten Weg der Sequenzkette überträgt (vgl. Phi, 2018). Der Zellzustand sorgt dafür, dass während des Backpropagationalgorithmus nicht jede Unit einen Einfluss hat. Vorher war das Problem, dass jeder State einen Einfluss hatte. Dieser Einfluss führte zu dem "vanishing gradient problem".

Die Gates haben die Aufgabe, Informationen hinzuzufügen oder zu entfernen. Die Gates regulieren somit den Einfluss. In Kombination sorgen der Zellzustand und die Gates dafür, dass es nicht zum "vanishing gradient problem" kommt. In einem LSTM gibt es vier Gates. Das "forget gate" (f) entscheidet welche Informationen vom vorherigen State, sowie vom Input vergessen werden sollen. Die Entscheidung welche vergessen werden sollen, übernimmt die Sigmoid-Funktion. Durch die Sigmoid-Funktion werden alle Werte zwischen null bis eins gebracht. Null signalisiert, dass diese Information vergessen werden soll, wobei eins signalisiert welche Information behalten werden soll. Erzielt wird das Vergessen durch die Multiplikation.

Das "input gate" (i) arbeitet mit dem "g gate" (g) zusammen und entscheidet, welche Informationen hinzugefügt werden. Wie auch das "forget gate" nutzt das "input gate" eine Sigmoid-Funktion. Das "g gate" hingegen nutzt wie ein normales RNN die $\tanh$ -Funktion. Die $\tanh$ -Funktion bringt die Werte in den Bereich minus eins bis eins. Durch die Multiplikation von Input-Gate und G-Gate wird entschieden, welche Information dem Zellzustand hinzugefügt wird. Das letzte Gate bildet das Output-Gate. Mittels Sigmoid-Funktion bestimmt das Output-Gate welche Informationen an die nächste Unit gegeben wird.

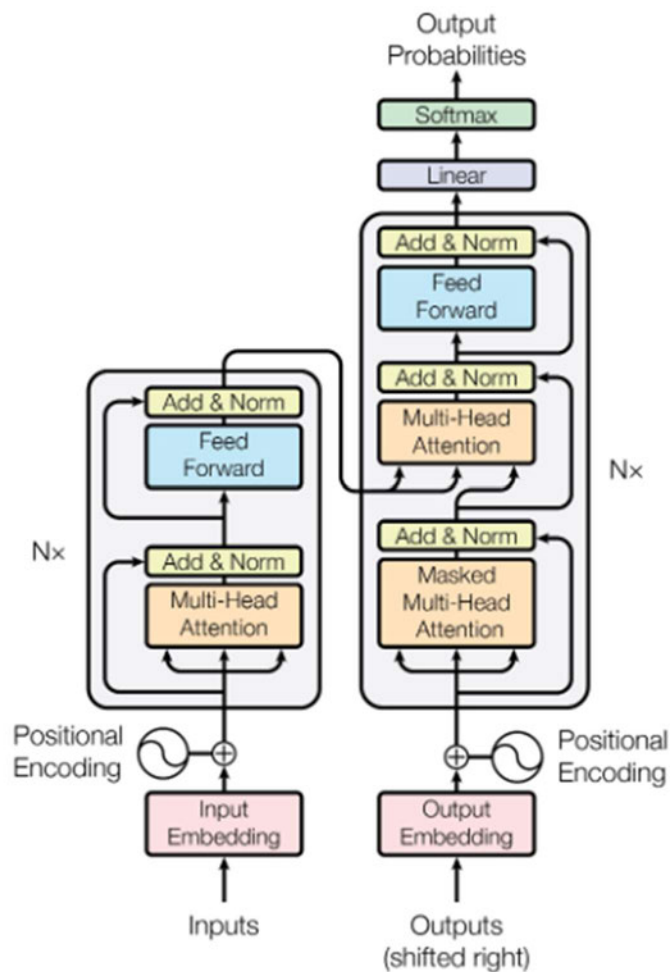
3.7 Gated Recurrent Unit (GRU)



(Phi 2018) – Gated Recurrent Unit

Das GRU-RNN ist die neuere Generation der rekursiven neuronalen Netzwerke und ähnelt einem LSTM (vgl. Phi 2018). Die GRU verzichtet auf den Zellzustand und nutzt den vorhandenen State für dieselbe Aufgabe. Im Gegensatz zu den vier Gates des LSTM, enthält das GRU zwei Gates. Das "reset gate" ist ähnlich dem "forget gate" beim LSTM. Es bestimmt welche Information vergessen werden soll mittels Sigmoid-Funktion. Das Update-Gate ist dem Input-Gate ähnlich. Der Unterschied ist, dass das Update-Gate noch ein "forget gate" hat. Dieses sorgt dafür, dass weitere Informationen vergessen werden können. Wie schon im "input gate" und "g gate" im LSTM, findet im Update-Gate ein Zusammenspiel aus Sigmoid und tanh-Funktion statt.

3.8 Transformer



(Vaswani u. a. 2017) – Transformer Architektur

Die Transformer-Netze wurden erstmalig 2017 im Wissenschaftlichen Paper "Attention Is All You Need" vorgestellt. "... eine Modellarchitektur, die auf Rekursion verzichtet und sich stattdessen ganz auf einen Attention-Mechanismus verlässt, um globale Abhängigkeiten zwischen Eingabe und Ausgabe zu erhalten." (vgl. Vaswani u. a. 2017, S.5999). Mit diesem Ansatz wird das Problem der RNN's umgangen. Aufgrund der Funktionsweise der RNN's betrachten die einzelnen Einheiten nur das vorherige Wort, in der Form der States. In den Sprachen haben die Wörter jedoch Beziehungen untereinander. Je nach Sprache kommen die Wörter an verschiedensten Stellen des Satzes vor. Der Transformer versucht nun dies mit einzubeziehen.

3.8.1 Attention

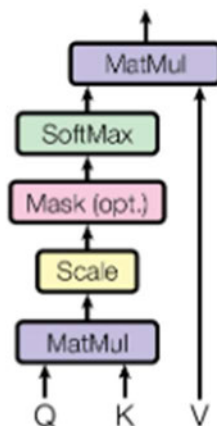
Die Idee der Attention ist genau die Bedeutung des Wortes. Es findet eine Berechnung statt die eine Aussage darüber trifft, in welcher Beziehung die Ausgangsdaten standen. Je nach Modell kann die genaue Ausführung/Berechnung der Attention variieren. In der Regel handelt es sich bei ihr um mathematische Operationen an deren Ende ein Abbildung heraus kommt. Da es sich um eine künstliche Aufmerksamkeit hat entspricht die Summer der einzelnen Werte stets eins. Die Abbildung kann wie folgt aussehen.

	Dies	Ist	Ein	Willkürliches	Beispiel
Dies	0.0	0.2	0.2	0.3	0.3
Ist	0.05	0.0	0.15	0.1	0.7
Ein	0.1	0.1	0.0	0.1	0.7
Willkürliches	0.0	0.0	0.2	0.0	0.8
Beispiel	0.1	0.1	0.4	0.4	0.0

Vereinfachte Veranschaulichung des Resultats des Attention-Mechanismus

In dieser Veranschaulichung ist abzulesen, dass der Eintrag "Dies" eine Attention von 0.3 für den Eintrag "Willkürliches" hat. Bei dem Eintrag "Willkürliches" gibt es bei "Dies" und "Ist" eine 0, daher zu diesen Einträgen besteht keine Beziehung.

Im Falle des Transformers findet sich folgender Attention-Mechanismus.



(Vaswani u. a. 2017) – Scaled Dot-Product Attention

3.8.2 Encoder

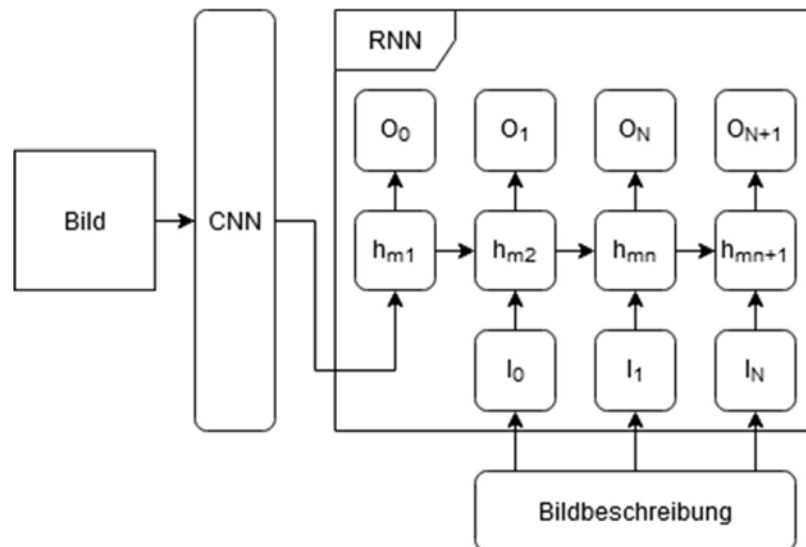
Die Aufgabe des Encoder (linker Teil der Architektur) ist es den Attention für den Input zu bestimmen. In der Praxis werden mehrere Encoder-Schichten, zu einem Encoder hintereinandergeschaltet, um bessere Ergebnisse zu liefern. Im Original Paper verwendet das Team sechs Stück, bevor das Ergebnis dem Decoder übergeben wird.

3.8.3 Decoder

Der Decoder (rechter Teil des Architektur) benötigt neben dem Zielsatz auch die Ausgabe des Encoders, weswegen die beiden nicht parallel durchlaufen werden können. Genau wie der Encoder besteht der Decoder aus genau so vielen Decoder-Schichten wie der Encoder. Der Decoder unterscheidet sich zum Encoder in einem Punkt. Nach der Berechnung der eigenen Attention wird diese genutzt, um mit der Ausgabe des Encoders eine neue Attention zu berechnen.

4 Konzept

4.1 Architektur: CNN-RNN



Kombiniertes Modell aus CNN und RNN

Dieses Modell wurde gewählt, da es eine einfache Kombination zweier guter Techniken ist. Beide Techniken sind in der Wissenschaft bereits etabliert. Die CNNs sind sehr gut darin Bilder zu analysieren und zu klassifizieren. RNNs haben im Bereich der Textverarbeitung gute Ergebnisse erzielt. Das kombinierte Modell ist ein einfaches modulares Modell. Die Modularität gibt dem Modell die Vorteile, einzelne Bestandteile gegebenenfalls auszutauschen. So kann mit verschiedenen CNNs und RNNs getestet werden, um das Beste zu nehmen. Die Modularität erlaubt es auch Erweiterungen gegebenenfalls einzubauen. Das CNN könnte durch weitere Schichten erweitert werden oder ganz durch ein komplexeres Modell ausgetauscht werden. Durch die festen Schnittpunkte zwischen den einzelnen Komponenten wird das Fehlerrisiko im Vorfeld minimiert. Ein RNN erkennt Objekte ausreichend gut, sofern die Datengrundlage gut genug ist. Um Zeit zu sparen wurde auf eine Objektlokalisierung verzichtet. Die geringere Komplexität macht das Modell verständlicher und Wartbarer.

Die Wahl für ein CNN hat mehrere Gründe. Erstens ist es einfacher zu verstehen als komplexere Netzwerke und einige komplexere Nutzen die Grundlage von CNNs. CNNs liefern eine gute Qualität bei der Klassifizierung von Bildern. Komplexere Netzwerke werden eher zur Erkennung von Objekten in Bildern (engl. "object detection") genutzt. Das Erkennen von Objekten gibt mehr Informationen als nur das Bild. Der Feature-Vektor eines CNNs beschreibt das ganze Bild. Diese Beschreibung ist ausreichend, daher kann auf ein komplexeres Netzwerk verzichtet werden. Der Feature-Vektor lässt sich als Satz oder Wort des Bildes betrachten. Für das Modell ist der Feature-Vektor ein weiteres Wort, welches die Informationen des Bildes enthält.

Das "perfekte" CNN für die Aufgabe lässt sich aus Zeitgründen nicht herausfinden. Es kostet viel Zeit, verschiedenste CNNs ordentlich zu evaluieren und die Parameter zu bestimmen. Die Qualität eines CNNs hängt stark von der Qualität, sowie der Quantität des verwendeten Datensatzes ab. Vergleiche von CNN Modellen zeigen daher ein Potenzial, welches Modell besser ist. Ein anderer Ansatz ist es, Modelle zu betrachten, die auf dem Markt vorhanden sind. Ein gutes Netzwerk, welches zur "real-time-object-detection" genutzt wird, ist das Modell "YOLOv3".

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	171
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	77.6	93.8	29.4	1090	37
Darknet-53	77.2	93.8	18.7	1457	78

(Redmon u.a. 2018, S. 3) – Grundlegende CNN Evaluation von YOLOv3

Die gezeigte Tabelle zeigt, dass das Darknet-53 eine sehr gute Performance bietet, was Operationen/s und Qualität entspricht. Ein sehr gutes CNN zu trainieren benötigt ausreichend Zeit, Daten und Rechenleistung, die nicht verfügbar sind. Die meisten Frameworks bieten bereits trainierte Netzwerke an. Im verwendeten Framework gibt es kein bereits trainiertes Darknet Netzwerk. Die Netzwerke, die das Framework anbietet, wurden mit der Bilddatenbank ImageNet trainiert. Die Wahl fiel auf das ResNet-101, sowie das GoogleNetv3 (Inceptionv3).

Genau wie beim CNN gibt es Alternativen für das RNN-Modell. Das grundlegende RNN hat einige Schwächen wie das "vanishing gradient problem", weswegen es sich nicht eignet. Komplexere Architekturen wie LSTM oder GRU eignen sich dagegen hervorragend. Es ist nicht klar, welches der beiden besser ist, da sie sich sehr ähneln. Wie es auch bei vielen anderen Modellen der Fall ist, kommt es auf die Daten und die richtigen Parameter an. Wie im Falle der CNNs lohnt sich auch hier beide Architekturen zu testen.

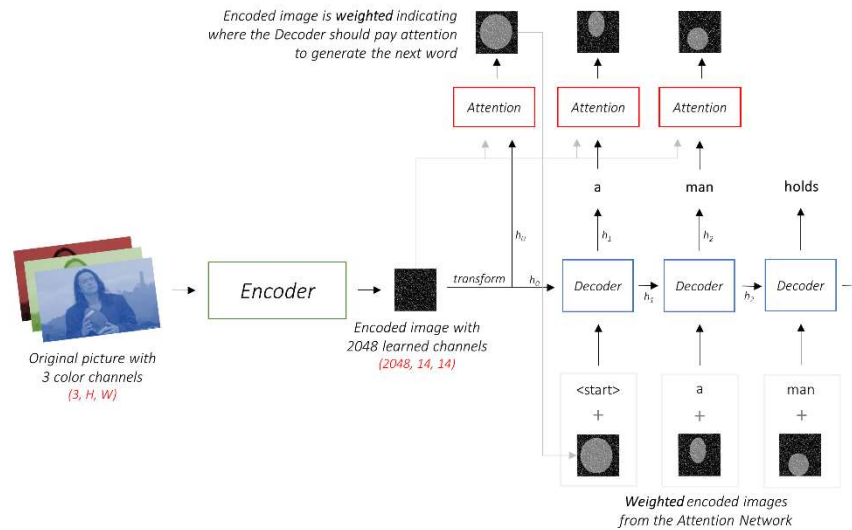
Für ein gutes Modell muss eine passende Datengrundlage existieren. Eine ausgewählte Datengrundlage muss folgende Punkt erfüllen. Zu jedem Bild muss ein Text existieren, der

das Bild beschreibt. Die Sprache der Texte muss einheitlich sein. Die Bilder müssen verschiedene Darstellungen enthalten. Ein Datensatz von ausschließlich Gesichtern wäre nicht gültig. Betrachtet man verschiedene Datensätze unter diesen Punkten, dann sind das immer noch einige. Um den Fokus des Aufwandes auf das Modell zu legen, wurde das Microsoft COCO Datensatz (MSCOCO) genommen. Der Vorteil vom MSCOCO liegt darin, dass für verschiedenen Aufgaben verschiedene Datensätze erstellt wurden. So gibt es Datensätze ausschließlich für Klassifikation, Objekt Erkennung oder die Beschreibung eines Bildes. Das MSCOCO hat 80 Objektkategorien, 330.000 Bilder (>200.000 mit einem Label), fünf Beschreibungen pro Bild und 1.5 Millionen Objekte. Die Objekte für eine Objekterkennung könnten nützlich sein, wenn ein komplexeres CNN, wie Faster-RCNN, von Nöten wird.

Die Datensätze Flickr8k und Flickr30k eignen sich ebenfalls für die Generierung von Bildbeschreibungen. Im Gegensatz zum MSCOCO, beschränken sich diese jedoch ausschließlich auf die Beschreibungen. Das Flickr8k enthält ca. 8000 Bilder. Genau wie beim MSCOCO gibt es fünf Beschreibungen pro Bild. Das Flickr30k enthält ca. 30.000 Bilder samt Beschreibungen.

Da es sich um internationale Datensätze handelt, sind die Beschreibungen in Englisch. Eine manuelle Übersetzung wäre zu aufwendig. Bei einer Maschinellen müsste sichergestellt werden, dass alle korrekt übersetzt sind, was ebenfalls zu aufwendig wäre. Die Ausgangssprache für das Modell bleibt daher bei Englisch. Die Wahl auf die englische Sprache hat für eine Auswertung keine kritische Rolle.

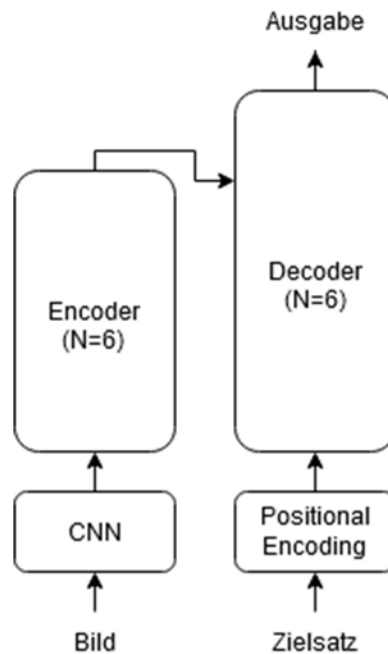
4.2 Architektur: CNN-RNN-Attention



(Vinodababu 2018) - Architektur aus "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention"

Die Architektur aus dem Paper "Show, Attend and Tell: ..." ist die grundlegende CNN-RNN-Architektur nur mit einem Attention-Mechanismus. Bei ihr handelt es sich nicht um eine "state of the art" Architektur. Trotzdem ist sie eine gute Wahl, da der Vergleich mit der Standardarchitektur gut möglich ist. Die Funktionsweise ist genau wie bei der Standardarchitektur. Der große Unterschied ist, dass die Eingaben nicht nur die Wörter selbst sind. Als Eingabe wird eine Kombination aus Wort und Attention im Bild benutzt. Die Attention gibt dann an, welche Pixel für das Wort relevant sind. Der größte Vorteil dieses Modells ist, dass die Bildinformationen für jedes Wort zur Verfügung stehen. Im Gegensatz zu der CNN-RNN-Architektur, wo mit dem Bild nur das erste Wort hervorgesagt wird. Ansonsten gilt dasselbe wie bei der CNN-RNN-Architektur.

4.3 Architektur: Transformer



Vereinfachte Darstellung des kombinierten Modell's aus CNN und Transformer

Für einen guten Vergleich sollte eine komplett andere Architektur gewählt werden. Die Transformer-Architektur gilt als eine der Vielversprechendsten. Im Erscheinungspaper "Attention Is All You Need" hat sie gezeigt, wie gut sie ist. Seitdem bauen immer mehr modernere Modelle auf dieser Architektur auf. Bei der Architektur aus "Image Caption Generation With Adaptive Transformer", von Zhang und seinem Team, handelt es sich um eine, die noch ziemlich nah am Original ist. So ist wie bei den anderen gewählten Architekturen auch ein CNN für die Bildinformationen zuständig. Die Bildinformationen werden daraufhin, wie ein normaler Satz einer Ausgangssprache behandelt. Danach funktioniert der Transformer genau wie die Original-Architektur von Vaswani und seinem Team. Die größte Änderung, die Zhang und sein Team gemacht haben, ist einen anderen Attention-Mechanismus zu verwenden. Für das CNN entschieden Sie sich für das ResNet-101 Modell. Die eigentliche Ausgabe des CNN's ist ein 7x7x2048-Vektor. Damit der Attention-Mechanismus korrekt funktioniert, muss die Ausgabe jedoch dieselbe Größe wie das Embedding haben. Damit dies gegeben ist, verwenden Zhang und sein Team noch eine Linear sowie ReLu-Schicht.

4.4 Framework

Was Deeplearning oder neuronale Netze angeht, sind Tensorflow und PyTorch die größten Frameworks auf dem Markt. Beide verwenden Python als Sprache und bieten das meiste Grundlegende an, wie Faltungsschichten. Trotz der gemeinsamen Sprache unterscheiden sie sich bei den Formulierungen. Der größte Unterschied der beiden Frameworks ist die Funktionsweise, wie die Netzwerke berechnet werden. Keines der beiden Frameworks ist entscheidend besser als das andere. Beide bieten höhere APIs für komplexere Aufgaben an. Die Entscheidung fiel auf PyTorch, da während des Testens Fehler in Tensorflow aufgetreten sind, die nicht behoben werden konnten. Mit der Festlegung auf PyTorch mussten auch voneinander abhängige Bibliotheken genommen werden. Somit sollte sichergestellt sein, dass keine Probleme auftauchen. Die PyTorch Version 1.4 benötigt das cudatoolkit 10.1 und die Version 3.6.9 von Python. Für das COCO Datensatz wird die Bibliothek pycocotools angeboten. Die Datensätze Flickr8k und Flickr30k benötigen keine Bibliothek. Zum Trainieren stellt PyTorch Objekte wie DataLoader bereit. Diese Objekte sorgen dafür dass die Bilder und ihre Beschreibungen immer korrekt sind. Somit wird nicht Bild A mit der Beschreibung von Bild B betrachtet.

4.5 Auswertung

Für eine metrische Auswertung des Modells lässt sich "Consensus-based Image Description Evaluation" (CIDEr) verwenden. Vedantam und sein Team schlagen ein neues Paradigma zur Bewertung von Bildbeschreibungen vor, dass den menschlichen Konsens nutzt (vgl. Vedantam u.a. 2015, S. 4566). Eine weitere metrische Auswertung ist die Methode "bilingual evaluation understudy" (BLEU). BLEU hat den Nachteil dass im Groben nur die Anzahl der Wörter eine Rolle spielt. BLEU fehlt der menschliche Konsens, der durch CIDEr erhofft wird. Trotzdem ist BLEU nützlich für erste Auswertungen.

Neben einem metrischen Ansatz kann eine einfache Umfrage genauso gut funktionieren. Die Umfrage ist so zuverlässig wie die Frage, die daraus beantwortet werden soll. Eine Umfrage über die Qualität beantwortet die Frage nicht korrekt, da die Personen subjektive Meinungen haben. Fragt man, welche Beschreibung von einem Netzwerk stammt, dann lässt sich die Frage beantworten, ob die erstellten Texte überzeugend genug sind.

Eine Umfrage hat den Vorteil, dass Unbeteiligte befragt werden. Es gibt ein Feedback, das zwar subjektiv ist, da es aber um Sprache geht, akzeptabel ist. Verschiedene Personen oder gar Generationen würden Bilder unterschiedlich beschreiben. Eine Kombination aus beiden Auswertungen sollte zufriedenstellend sein. Der wissenschaftliche Aspekt wird durch die metrische Auswertung gut abgedeckt. Die Umfrage deckt die Glaubwürdigkeit der Sätze ab.

5 Realisierung

Die Implementierung der Grundkonzepte aus den Grundlagen wurden alle durch PyTorch bereitgestellt. Die Codebasis für das Modell, sowie den meisten verwendeten Funktionen wurden von dem YouTuber/Coder Aladdin Persson geschrieben. Die Codebasis ist unter der MIT Lizenz nutzbar. Ein Link zur Codebasis findet sich im Abschnitt Links.

Der aktuelle verwendete Code enthält einige Änderungen sowie Anpassungen, die ihn für die Arbeit zugänglicher machen.

5.1 Datengrundlage

Es wurden die drei Datensätze Flickr8k, Flickr30k sowie MSCOCO genutzt. Für jedes der Datensätze wurde eine Textdatei im CSV-Format erstellt. Bei der Textdatei handelt es sich um die sogenannte "annotation file" oder "caption file". Die Überschriften für das CSV-Format bilden "image" und "caption". Die Dateneinträge bestehen dann aus dem Dateinamen des Bildes und seiner Beschreibung. Da zu jedem Bild genau fünf Beschreibungen existieren, wurden auch alle in die Textdatei aufgenommen.

```
image,caption
COCO_train2014_000000318556.jpg,A very clean and well decorated empty bathroom
COCO_train2014_000000318556.jpg,A blue and white bathroom with butterfly themed wall tiles.
COCO_train2014_000000318556.jpg,A bathroom with a border of butterflies and blue paint on the walls above it.
COCO_train2014_000000318556.jpg,An angled view of a beautifully decorated bathroom.
COCO_train2014_000000318556.jpg,A clock that blends in with the wall hangs in a bathroom.
COCO_train2014_000000116100.jpg,A panoramic view of a kitchen and all of its appliances.
```

Aufbau der MSCOCO Textdatei

Ein Datensatz Objekt liest dieses CSV-Format aus. Alle Daten, die in der Textdatei stehen sind in ihrer Reihenfolge im Objekt gespeichert. Ein Wörterbuch wird für spätere Verwendung zeitgleich erstellt, welches die wichtigsten Wörter enthält. Es wird ein Dictionary erstellt, welches die Daten enthält. Bei dem Schlüssel handelt es sich um die Position des Bildes in der Reihenfolge der Textdatei. Fünf Einträge werden zu einem zusammengefasst. In der Textdatei sind die ersten fünf Einträge dasselbe Bild. Diese fünf Beschreibungen, sowie das Bild sind über den Index null erreichbar. Bei einer Datensatzgröße von 8.000 ergibt sich eine Textdatei mit 40.000 Einträgen, aber ein Dictionary der Größe 8.000.

Wird ein Datensatz angesprochen, dann wird das zugehörige Bild geladen. Das Objekt wandelt dann auf dem Bild die richtigen Transformationen um.

Das Bild wird dabei in eine einheitliche Größe gebracht. Zur weiteren Transformation wird ein kleiner Ausschnitt des Bildes zufällig genommen. Dieser Schritt wird unternommen, da die Objekte in den Bildern nie gleich sind. Die Orientierung, Distanz zur Kamera oder ob das Objekt komplett ist, ändert sich pro Bild. Dieser Schritt hilft dem Modell eine generelle Repräsentation zu trainieren. Anschließend muss das Bild in sogenannte Tensor gebracht werden. Tensor sind n-Dimensionale Matrizen mit der PyTorch funktioniert. Der letzte Schritt ist eine Normalisierung des Bildes, damit es keine Extreme gibt.

Bei den Beschreibungen wird die erste der fünf genommen. Genau wie bei dem Bild werden auch bei der Beschreibung, die korrekten Transformationen angewendet. Mittels der Bibliothek `spacy` und dem Package `'en_core_web_sm'` werden die Sätze in Tokens aufgeteilt. Darunter fallen dann auch Satzzeichen. Die Tokens sind alle in der Kleinschreibung gehalten, da sonst "A" und "a" als verschiedene Wörter angesehen werden würden. Jeder Satz bekommt noch ein Anfangstoken (`<eos>`) und ein Endtoken (`<eos>`). Vor allem das Endtoken wird benötigt, da die Sätze verschiedene Längen haben und nicht alle auf demselben Wort enden. Durch das Endtoken wird nun dem Modell einheitlich das Ende des Satzes gezeigt.

Die anderen Sätze werden für die Berechnung des BLEU-Scores genutzt.

Die Daten wären durch den Datensatz schon zum Trainieren geeignet. Ein `DataLoader` Objekt gibt dem Modell die Daten in der korrekten Batchgröße. Die Batchgröße wurde auf 32 gesetzt, da dies den Speicher nicht überfüllt. Während des Ladens der Batches wird Padding an den Beschreibungen durchgeführt. Das Padding orientiert sich an der längsten Beschreibung des Batches. Mittels Parameter lässt sich bestimmen, dass jeder Durchlauf durch den Datensatz eine andere Reihenfolge hat.

Das Modell Training somit mit zwei Tensor. Einer für die Bilder mit den vier Dimensionen $32 \times 32 \times 299 \times 299$. Der andere mit $N \times 32$, wobei N der Länge der Sätze entspricht. Die umgedrehten Dimensionen entsprechen der Reihenfolge, wie die RNNs die Daten erwarten.

Das MSCOCO ist in drei Datensätzen von seinen Erstellern aufgeteilt worden. Das Erste der drei Datensätze wird für das Training verwendet (`train2014`). Der zweite der drei Datensätze wird für die Validierung verwendet (`val2014`). Der letzte Datensatz dient Testzwecken (`test2014`). Der Validierungs-Datensatz enthält als einziger keine Beschreibungen, da es für eine manuelle Auswertung gedacht ist.

5.2 CNN

Das CNN des Modells bekommt neben wichtigen Parametern auch mitgeteilt, welches CNN verwendet werden soll. Aus Zeitgründen und um die Fehler so gering wie möglich zu halten, wurde die Entscheidung für ein bereits trainiertes CNN beibehalten. Sowohl das Resnet-101 wie das Inception_v3 enthalten eine letzte Schicht für die Klassifizierung. Diese Schicht wird durch eine eigene Linear Schicht ausgetauscht. Diese Linear Schicht ist notwendig, um die entstehenden Daten in das richtige Format zu bringen. Dieser Linear Schicht muss als einzige bei einem bereits trainierten CNN mit trainiert werden. Die Ausgabe für jedes Bild

entspricht derselben Dimension wie auch bei dem RNN. Der CNN-Teil des Modells gibt einen Tensor mit den Dimensionen [BatchGröße]x[Embedding] heraus.

Die Verwendung des Inception_v3 setzt die Dimension der Bilder fest. So erwartet das Inception_v3 Bilder der Dimension 3x299x299. Bei dem Aufbereiten und Laden des Datensatzes muss die Größe schon feststehen. Gleichzeitig ermöglicht eine einheitliche Größe einen besseren Vergleich, da beiden Varianten dieselben Bildinformationen zur Verfügung stehen. Diese Limitierung setzt die Größe für das ResNet-101 ebenfalls fest.

5.3 RNN

Die grundlegende Realisierung des RNN-Teils lässt sich in drei Teile aufteilen. Zuerst wird ein Embedding an den Beschreibungen durchgeführt. Die Embedding-Schicht erwartet, dass die Beschreibungen bereits in Tokens umgewandelt sind. An dieser Stelle wird die Größe des Wörterbuchs von dem Datensatz benötigt. Die Embedding-Schicht wandelt die einzelnen Wörter (Token) in Tensor der Größe 512 um. Bei vorherigen Testzwecken wurde jedoch eine kleinere Größe gewählt. Nach dem Embedding sind die Daten bereit für das grundlegende RNN.

Genau wie bei dem CNN bestimmt ein Parameter bei der Initialisierung, welche Architektur genommen wird. Da die LSTM und GRU Architektur sehr ähnlich sind, bedarf es keine weitere Anpassung der Daten. Der einzige Unterschied wäre die Anzahl der States. Das LSTM nutzt neben dem Hiddenstate noch den Cellstate. Das GRU nur den Hiddenstate. Für die CNN-RNN-Architektur ist dies kein Problem, da im Framework die RNN's vorhanden sind.

Für die CNN-RNN-Architektur wird das Bild als Vorwort der Beschreibung behandelt. Die Ausgabe des CNNs enthält eine Dimension weniger, nämlich die der Wörter. Der Tensor der Ausgabe lässt sich einfach um die fehlende Dimension erweitern. Jetzt erst lässt sich der Beschreibungstensor an die Ausgabe des CNNs konkatenieren. Wenn der Beschreibungstensor die Dimension 22x32x512 (Wörter, Batch, Embedding) hatte, dann hat er jetzt 23x32x512.

Nachdem das RNN die Daten verarbeitet hat, kommt der letzte Teil. Wie bei so ziemlich jedem Neuronalen Netz kommt eine Linear-Schicht zum Einsatz. Diese transformiert die Ausgabe in den benötigten Klassifizierungsvektor. So das am Ende ein Vektor der Dimensionen 23x32xN entsteht. Das N steht dabei für die Anzahl an Wörtern, die dem Modell zur Verfügung stehen. Dies ist aus dem Datensatz zu entnehmen.

5.4 CNN-RNN-Attention

Um möglichst nah an der Implementation von Vinodababu zu bleiben, wurde diese als Grundlage genommen. Daraus folgten folgende Änderungen. Das CNN muss eine Schicht früher enden als in der CNN-RNN-Architektur. Dadurch fällt die Linear-Schicht für das Embedding weg. Anstelle ihrer kommt eine Pooling-Schicht (AdaptiveAvgPool2d) zum Einsatz. Die Ausgabe des CNNs ist nun ein 4-Dimensionaler Vektor der Größe [BatchGröße]x14x14x2048.

Für das RNN kommt nicht mehr das Framework eigene LSTM oder GRU infrage. Der Grund liegt darin, dass für das Wort an Position N erst mal der Attention-Vektor bestimmt werden muss. Zum Bestimmen des Attention-Vektors werden zwei Vektoren vorausgesetzt. Der eine ist das Bild. Bei dem anderen handelt es sich um den Hiddenstate, der genutzt wird, um das Wort vorherzusagen. Im Framework sind die RNNs geschlossene Systeme, die die einzelnen Zellen ablaufen und den fertigen Klassifizierungsvektor bereitstellen. Das komplette RNN muss jedoch nicht selbst implementiert werden, da das Framework die Zellen für LSTM und GRU bereitstellt. Es bedarf daher einer For-Schleife, die die Zellen nach und nach durchgeht und den Klassifizierungsvektor erstellt.

Der verwendete Attention-Mechanismus konnte nicht im Framework gefunden werden. Im ersten Schritt werden die letzten Dimensionen der Eingaben auf dieselbe Größe gebracht. Da die Eingaben nun dieselbe letzte Dimension haben, können sie konkateniert werden. Die Attention wird umgesetzt durch einen Softmax. Dieser Vektor ist der Faktor, wie viel Beachtung einem Wert gegeben wird. Vor dem Softmax muss der konkatenierte Vektor auf drei Dimensionen reduziert werden, wobei die letzte nur eins groß sein darf. Nur so kann der Softmax über die zweite Dimension gebildet werden. Dieser sogenannte Alpha-Wert wird dann mit der Ausgabe des CNNs multipliziert. Anschließend muss nur noch summiert werden, damit der entstehende Wert später mit den Worten korrekt konkateniert werden kann. Die RNN-Zelle benötigt nämlich einen zweidimensionalen Vektor. Eine der Dimensionen steht für die Batchgröße und der andere für die Größe des Hiddenstate. Die Größe des Hiddenstate wird nur erreicht durch das Konkatenieren der beiden Größen für das Embedding und der Attention.

5.5 Transformer

Die geplante Transformer-Architektur konnte nicht eins zu eins implementiert werden. Als erstes wurde die Größe der CNN Ausgabe limitiert. Die eigene letzte Linear-Schicht im CNN wurde auf 25 gesetzt. Dadurch ist jedes Bild für den Transformer ein Satz aus 25 Wörtern. Ein weiterer Vorteil ist das kein Padding notwendig ist wie bei den Zielsätzen. Wie auch bei Zhang und sein Team kommt eine Linear-Schicht und eine ReLu-Schicht zum Einsatz. Da es keinen Grund gibt diese von dem CNN zu trennen, findet dies am Ende des CNNs statt.

Es konnte kein Code gefunden werden, der den von Zhang und seinem Team genutzten Attention-Mechanismus implementiert. Aufgrund von Problemen mit der Zeit und einer eigenen Implementation musste der Standardtransformer genommen werden. Dieser wird vom Framework bereitgestellt. Der Transformer erwartet eine andere Reihenfolge für die Dimensionen. Die Ausgabe des CNN-Teils des Modells wird daher vor der Übergabe zum Transformer dem entsprechend angepasst. Daneben wird noch eine Padding-Maske erstellt, damit der Transformer weiß, wo das Padding ist. Dies ist am einfachsten, bevor das Embedding angewendet wird. Die Maske enthält an den Positionen, wo das Padding ist den Boolean "True".

Die Universität Harvard stellt eine performante Implementierung des "Positional Encodings" des Originalpapers zur Verfügung. So dass keine komplett eigene Implementation notwendig war.

5.6 Training

Um effektiv über Nacht die Modelle zu trainieren, werden die Datensets zu Beginn einmal initialisiert. Dieser Ansatz spart Zeit, da nicht bei jedem Modell alle Aufbereitungen wiederholt werden müssen. Gleichzeitig ist gegeben das jedes Modell mit dem gleichen Datensatz arbeitet. Die vom Framework bereitgestellte Klasse zum Laden der Daten (DataLoader) funktioniert nur einmal. Abhilfe schafft ein Multi-Epochen-Dataloader. Nachdem der Dataloader initialisiert wurde zu Beginn, können die Modelle nacheinander trainiert werden.

Als Optimizer für das Training wurde Kreuzentropie gewählt. Das Framework stellt dafür die Klasse CrossEntropyLoss bereit. Der Vorteil hierbei ist das die Klasse selbst das Softmax übernimmt. Damit müssen die Modelle ihre Klassifizierungsvektoren nicht selbst Softmaxen. Gleichzeitig ist es gut, falls es eine Imbalance im Trainingsset gibt. Da die Modelle jeweils anders Funktionieren, hat jedes eine Funktionalität, um die Eingaben korrekt anzupassen. Für den Optimizer ist es wichtig, dass er die Zielsätze so bekommt, dass ein Wort das nächste vorhersagt. Am Anfang wurde der Fehler gemacht. Das Modell lernt dann nur, wie jedes Wort sich selbst vorhersagt. Die gewählte CNN-RNN-Architektur bekommt durch das Bild ein Wort mehr, wodurch der Startsatz verschoben ist. Trotzdem ist dadurch gesichert, dass jedes Wort das nächste vorhersagt.

Nach jeder gelernten Epoche werden die Loss-Werte abgespeichert und es wird geprüft, ob ein früher Stopp erfolgen soll. Dieser frühe Stopp dient dazu, das Modell dann zu stoppen, wenn sich der Trainings-Loss nicht mehr wesentlich ändert. Um ein Overfitting zu bestimmen muss erstmal eine hohe Anzahl an Epochen trainiert werden. Mit dem frühen Stoppen ist es nicht notwendig das Overfitting manuell zu bestimmen. Selbst mit vorherigem bestimmen des Overfittings, kann das frühe Stoppen helfen. So kann sich sicher dem Overfitting angenähert werden. Sobald alle Epochen durch gelaufen sind wird das Modell abgespeichert, sofern der Parameter dafür gesetzt ist.

6 Schluss

6.1 Auswertung/Fazit

6.1.1 Vorüberlegungen

Trainiert wird auf dem eigenen Computer. Dieser enthält 32GB RAM, eine RTX 2070 sowie einen Intel i7-8700K mit 3.7GHz. Die RTX 2070 ist zwar eine der modernsten Grafikkarten, jedoch enthält sie nur 8GB Speicher. Dadurch dass nur eine Grafikkarte vorhanden ist, ist kein parallel Training von mehreren Modellen möglich. Funktionstests der Modelle zeigten bereits, dass 8GB gerade so ausreichend sind. Die meisten Modelle können mit einer Batchgröße von 32 trainiert werden. Die Ressourcen des eigenen Computers stehen nicht die ganze Zeit für das Training zur Verfügung. Dies ist jedoch kein Problem. Es können so Modelle trainiert, geprüft und ausgewertet werden. Erkannte Fehler können behoben werden. Es gibt so nicht viele unnötige Modelle die nur Ressourcen verbraucht haben.

Da der eigene Computer trotz allem völlig ausreicht, fiel die Wahl auf ihn. Bei Cloud-Computing-Anbietern kostet es meist Geld je nach Ressourcen. Ein externer Dienst heißt immer, es gibt keine Unabhängigkeit. So liegt die Erreichbarkeit der Server nicht in der eigenen Hand. Es bedeutet auch Programmieren in Remote bzw. die Dateien dann rüberschieben. Für den Aspekt der Programmierung der Modelle ist der eigene Computer am besten. Nur für die Auswertung einen externen Dienst in Anspruch zu nehmen bringt extra Aufwand mit sich. So müssen alle Programme installiert, Daten herüber geladen und die Ergebnisse zurück geladen werden. Die Datensätze allein machen 34 GB Speicher aus.

Erste kleine Tests bezüglich der Funktionalität zeigten, dass ein bidirektionales RNN nicht nützlich ist. Es kann sein, dass bei längerem Training das Ergebnis besser wird. Bei zehn Epochen bringt ein bidirektionales RNN keine korrekten Sätze hervor. Ein nicht bidirektionales RNN erzielt bei zehn Epochen schon grammatikalisch korrekte Sätze.

<pre><SOS> a dog is running through the water . <EOS> <SOS> a brown dog is running through the snow . <EOS> <SOS> a brown dog is running through the snow . <EOS></pre>	<pre><SOS> four four four four four four <SOS> cyclists cyclists cyclists cyc <SOS> fisherman fisherman fisherman</pre>
---	---

Ergebnisse von nicht-bidirektionalen und bidirektionalen Modellen

Durch den Verzicht auf das bidirektionale RNN reduziert sich die Anzahl der zu testenden Modelle.

Für das Training wurde folgende Strategie genutzt. Da es drei Datensätze in verschiedenen Größen gibt, sollte das ausgenutzt werden. Die drei Datensätze lassen sich anhand der Anzahl der Bilder in klein, mittel und groß aufteilen.

Der kleinste Datensatz (Flickr8k) zeigt sich durch Schnelligkeit aus, da jedes Modell hier am schnellsten eine Epoche abschließt. Eine Epoche benötigt im Flickr8k ca. 30 bis 70 Sekunden, je nach Architektur des Modells. Damit benötigen 50 Epochen 25 bis 60 Minuten. Durch die Schnelligkeit im Vergleich zu den anderen lassen sich hiermit die meisten Parameter vergleichen. Je nach Anzahl der Parameter und ihrer Kombinationen könnte das relativ lange dauern. Des Weiteren beeinflusst jeder Parameter nicht nur die Qualität des Modells, sondern auch die Anzahl der benötigten Epochen. Nimmt man die Lernrate des Modells, so bedeutet eine geringere Lernrate ein langsames Lernen. Eine Variante könnte doppelt so viele Epochen benötigen, um ein besseres Modell zu werden. Somit müsste für einen optimalen Vergleich jede Kombination der Parameter bis kurz vor dem Overfitting trainiert werden. Gleichzeitig geben Parameter wie die Lernrate, ein unendliches Spektrum an zu testenden Werten. Da die Optimierung der Parameter kann eine eigene Bachelorarbeit sein. Da das Thema der Arbeit aber nicht die Optimierung von Parametern ist, werden bestimmte Werte nur verglichen. Die Einschränkung auf nur wenige Werte für einen Parameter kommt der Zeit zugute.

Der mittlere Datensatz (Flickr30k) benötigt pro Epoche schon ca. drei bis sechs Minuten. Das heißt, der Mittlere kann pro Stunde 10- bis 20-mal verarbeitet werden. Als mittlerer Datensatz eignet er sich nicht für eine spezielle Aufgabe wie die anderen beiden. Der Datensatz lässt sich jedoch dafür nutzen, den Einfluss der Anzahl an Trainingsdaten auf die Qualität des Modells abzuschätzen. Sollten bei beiden Datensätzen (Flickr8k, Flickr30k) dieselben Modelle eine gleiche Qualität haben, dann lässt sich annehmen, dass für den großen Datensatz dasselbe gilt. Somit sieht man, welche Parameter mit dem großen Datensatz zu testen sind und welche nicht ausgewertet werden müssten. Dieser Ansatz lässt sich nutzen, da der große Datensatz über doppelt so groß ist, wie die anderen beiden kombiniert. Weiter lässt sich der mittlere Datensatz dazu nutzen, das Overfitting abzuschätzen. Jedes Modell kann in einer vernünftigen Zeit genug Epochen durchlaufen, damit sich das Overfitting abzeichnet.

Der MSCOCO Datensatz benötigt pro Epoche ca. siebeneinhalb Minuten. Der Datensatz kann somit 8-mal verarbeitet werden pro Stunde. Ein Modell, welches mit diesem Datensatz trainiert wird, benötigt länger als bei den anderen. Um Zeit und Ressourcen zu sparen, sollten mit diesem Datensatz wenige Modelle trainiert werden. Dieser Datensatz deckt die meisten Abbildungen ab. Dadurch eignet sich dieser am besten, um ein finales Modell zu bestimmen. Der Vorteil im besten Falle ist, dass das Modell möglichst viele Bilder korrekt beschreibt.

Um die Modelle optimal Vergleichen zu können, wird alles Zufällige abgestellt. Somit profitiert kein Modell vom Zufall. Die Modelle verzichten damit auf Verbesserungen wie dem Dropout. Dadurch lassen sich die Parameter am besten Vergleichen, da ihr Einfluss nicht begleitet wird, durch so etwas wie Dropout. Bei dem Training der finalen Modelle wird dann jeglicher Zufall wieder eingeschaltet. Damit ist die Findung der Parameter gut vergleichbar, da ihre Auswirkungen gut sichtbar sind.

Der MSCOCO Datensatz stellt einen separaten Datensatz für die Validierung zur Verfügung. Dieser Validierungs-Datensatz besteht aus ca. 40.000 Bildern mit jeweils fünf Beschreibungen. Aus Zeitgründen wurde folgender Validierungsansatz gewählt. Nachdem das Modell trainiert wurde, kommt der Validierungs-Datensatz zum Einsatz. Der erste von den fünf Sätzen, die jedes Bild hat, wird dem wie beim Training übergeben. Der daraus erstellte Satz wird mit den übrigen vier Sätzen mittels BLEU verglichen und bildet den BLEU-4 Wert. Aus allen BLEU-4 Werten wird ein Durchschnitt gebildet, dieser repräsentiert die Qualität des Modells. Der Ansatz, während des Trainings schon eine Validierung durchzuführen, würde zu viel Zeit kosten. Trotz der Schwäche des fehlenden Konsens, im gewählten Ansatz ist er genügend für eine erste Auswertung. In Kombination wird eine Handvoll Bilder manuell ausgewertet. So lässt sich die Qualität besser bewerten, da der Konsens über die manuelle Auswertung kommt.

Ein entscheidender Faktor des Modells bildet der erste Parameter. Bei ihm handelt es sich nämlich um das grundlegende CNN-Modell. Er liefert dem eigentlichen Modell die Informationen des Bildes, auf dem alles Weitere aufbaut. Neben der eigentlichen Frage, welche Architektur für das CNN genommen werden soll, gibt es noch die Frage des Trainings. Im Grunde bildet der Parameter die ersten Schichten des ganzen Modells. Die Funktionalität des Parameters ist eine andere als die des gesamten Modells. Der Parameter kann somit nicht mit denselben Trainingsdaten trainiert werden. Den Parameter vorher selbst zu trainieren erfordert mehr Zeit und andere Datensätze. Es ist nicht vorherzusehen, welchen Einfluss die Architektur spielt, für das gesamte Training. Daher empfiehlt sich ein vortrainiertes Modell. Dadurch wird enorm Zeit gespart. Je nach Größe des CNN's würden dessen Gewichte den Speicher überladen. Es müsste dann eine kleinere Batchgröße gewählt werden. Daher wird von Beginn an auf ein vortrainiertes CNN gesetzt.

Einen weiteren Faktor macht das Embedding aus. Beim Embedding sollten ähnliche Wörter auch ähnliche Zahlenwerte haben. Diese Ähnlichkeiten sollen die Beziehung der Wörter widerspiegeln. Nun gibt es zwei Ansätze. Ein bereits trainiertes wird genommen oder es wird selbst trainiert. Das bereits trainierte Embedding hat den Vorteil, dass weniger Gewichte zum Trainieren vorhanden sind. Es gibt dadurch ein zentrales Wörterbuch und nicht für jedes Modell ein eigenes. Dazu kommt die Ansicht, ob das Embedding zum Modell gehört oder nicht. Die Architekturen erwarten in der Regel den Vektor, der durch das Embedding entsteht. Das Embedding ist daher eher eine Datenvorbereitung. Da es nicht zum Modell explizit gehört, muss es auch nicht mit dem Modell trainiert werden. Die Wahl fällt daher auf ein bereits trainiertes Embedding. Der Vorteil dadurch ist, dass es eine mögliche Fehlerquelle weniger gibt. Spätere Fehler in den Ergebnissen oder der Qualität kann dann nicht an einem Training des Embedding liegen. Genutzt werden die GloVe-Embeddings der Universität Stanfords. Im Genauen die aus "glove.6B.300d.txt". Jedes Wort aus dieser Datei hat einen Zahlenvektor aus 300 Dimensionen. Für die Performance wurden nur die Wörter genommen, die für das jeweilige Datenset eine Rolle spielen. Diese sind in den jeweiligen Dateien "g6b300d.default.[datenset]" zu finden.

6.1.2 Auswertung

Die ersten zwei Parameter beziehen sich nur auf die CNN-RNN Architektur. Da die anderen Architekturen erst nach der ersten Auswertung dazu kamen.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	JSON	Architektur	Training Validation Batch	Pretraine	Earlybre	Epoc	Runtime		BLEU		Bsp, Caption	Datum	Runtime Differenz
2	configs	cnn-rnn (inception-1stm)	flickr8k_Coco_Val2	32	g6b300d.	False	10	397,721533	0,0142158	<eos> a man	2020_12_28		2,418836736
3	configs	cnn-rnn (resnet-1stm)	flickr8k_Coco_Val2	32	g6b300d.	False	10	962,023454	0,0143156	<eos> a man	2021_01_01		0,413421867
4													
5	configs	cnn-rnn (inception-1stm)	flickr8k_Coco_Val2	32	g6b300d.	False	10	406,33582	0,0162993	<eos> a man	2020_12_28		2,33561535
6	configs	cnn-rnn (resnet-1stm)	flickr8k_Coco_Val2	32	g6b300d.	False	10	949,044178	0,01607	<eos> a man	2021_01_01		0,428152692
7													
8	configs	cnn-rnn (inception-1stm)	flickr8k_Coco_Val2	32	g6b300d.	False	10	416,459083	0,0179329	<eos> a black	2020_12_28		2,452601839
9	configs	cnn-rnn (resnet-1stm)	flickr8k_Coco_Val2	32	g6b300d.	False	10	1021,40831	0,0173407	<eos> a boy i	2021_01_01		0,407730266

Ausschnitt aus dem Datenblatt "resnet_inception" der Datei
"auswertung_parameter_excel.xlsx"

Das obige Bild zeigt einen Vergleich für den ersten Parameter, nämlich das grundlegende CNN-Modell. Die gewählten Modelle Resnet-101 und Inception_v3 sind immer in Paaren in der Tabelle zu finden. Spalte M zeigt den Unterschied in der Trainingsgeschwindigkeit auf. Die Zeile zwei und drei bilden den ersten Vergleich. Beide enthalten bis auf das CNN dieselben Parameter. Zeile zwei Spalte M enthält den Faktor ~2.4, was bedeutet, dass dasselbe Modell mit Inception_v3 als CNN 2.4-mal so schnell ist. Dieser Wert spiegelt sich auch in der Spalte i – Runtime wider. Nimmt man die Werte der Spalte I für die Zeilen, dann gilt $962 \approx 2.4188 \cdot 397.7215$, dasselbe gilt für den anderen Faktor. Der Vergleich zeigt auf das Inception_v3 mindestens doppelt so schnell ist wie das Resnet-101. Aufgrund der Funktionalität des CNN kann das Resnet-101 theoretisch besser sein. Die Spalte J zeigt den BLEU-4 Wert. Die Spalte offenbart, dass keines von beiden besser zu sein scheint. Je nachdem wie die anderen Parameter sind, ist das eine mal besser als das andere. Bei den meisten Modellen ist der Unterschied im BLEU-4 marginal. Die Zeitersparnis überwiegt dem kleinen, nicht immer vorhandenen Unterschied im BLEU-4. Daher wird für weitere Modelle auf die Inception_v3 Variante gesetzt.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	JSON	Architektur	Trainingsset Validationse	Batch	Pretrained	E	Earlyb	Epoc	Runtime	BLEU	Bsp, Caption	Datum	Runtime Diff
2	configs/cnn-rnn	(inception-1stm)	flickr8k_defc	Coco_Val201	32	g6b300d.def	False	10	397,72153	0,0142158	<eos> a man	2020_12_28	-5,26795554
3	configs/cnn-rnn	(inception-gru)	flickr8k_defc	Coco_Val201	32	g6b300d.def	False	10	402,98949	0,0145412	<eos> a man	2021_01_01	5,26795554
4													
5	configs/cnn-rnn	(inception-1stm)	flickr8k_defc	Coco_Val201	32	g6b300d.def	False	10	406,33582	0,0162993	<eos> a man	2020_12_28	-1,82160497
6	configs/cnn-rnn	(inception-gru)	flickr8k_defc	Coco_Val201	32	g6b300d.def	False	10	408,15742	0,0171782	<eos> a black	2021_01_01	1,82160497
7													
8	configs/cnn-rnn	(inception-1stm)	flickr8k_defc	Coco_Val201	32	g6b300d.def	False	10	416,45908	0,0179329	<eos> a black	2020_12_28	3,94052219
9	configs/cnn-rnn	(inception-gru)	flickr8k_defc	Coco_Val201	32	g6b300d.def	False	10	412,51856	0,0091283	<eos> a black	2021_01_01	-3,94052219

Ausschnitt aus dem Datenblatt "rnn " der Datei "auswertung_parameter_excel.xlsx"

Der nächste Parameter wäre das grundlegende RNN-Modell. Wieder ist im obigen Bild in Spalte M die Differenz in der Trainingszeit zu sehen. Diesmal jedoch als absoluter Wert in Sekunden. Die meisten Modelle haben einen Unterschied von unter 10 Sekunden. Einige Ausnahmen gab es, die zwischen 30 bis 110 Sekunden lagen. Das erste Inception Modell

findet sich in Zeile zwei und drei wieder, dort gibt es einen Unterschied von 5 Sekunden. Das erste Resnet-101 Modell findet sich in Zeile 26 und 27 wieder, dort gibt es einen Unterschied von 51 Sekunden. Das letzte Inception Modell findet sich in Zeile 23 und 24, dort gibt es einen Unterschied von 36 Sekunden. Das letzte Resnet-101 mit denselben Parametern wie das letzte Inception_v3 ist in Zeile 47 und 48. Hier findet sich ein Unterschied von 3 Sekunden. Diese Ausnahmen scheinen nichts mit den Modellen an sich zu tun zu haben. Die Vermutung liegt nahe, dass der Computer etwas anderes nebenbei gemacht hat. Genau wie bei dem Vergleich zwischen Resnet-101 und Inception_v3, gibt es kein eindeutiges besseres RNN. In Spalte J für den BLEU-4 finden sich stets ähnliche Werte, bis auf einige Ausreißer. Die Tabelle zeigt des Weiteren, dass die Ausreißer nichts mit der Trainingsdauer zu tun haben. Es scheint eher eine Frage der anderen Parameter zu sein, welches Modell besser ist.

	A	B	C	D	E	F	G	H	I	J	K	L	M	
1	JSON	Architektur	Trainingss	Validation	Batch	Pretrained	E	Earlybr	Epo	Runtime	BLEU	Bsp, Caption	Datum	Lernrate
2	configs	cnn-rnn (inception-lstm)	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	397,721533	0,0142158	<sos> a man	2020_12_28	0,0001		
3	configs	cnn-rnn (inception-lstm)	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	406,33582	0,01629929	<sos> a man	2020_12_28	0,0002		
4	configs	cnn-rnn (inception-lstm)	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	416,459083	0,01793289	<sos> a black	2020_12_28	0,0003		
5	configs	cnn-rnn (inception-lstm)	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	408,769027	0,00671578	<sos> a black	2020_12_28	0,0004		
6														
7	configs	cnn-rnn (inception-lstm)	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	415,871786	0,01278613	<sos> a man	2020_12_28	0,0001		
8	configs	cnn-rnn (inception-lstm)	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	402,347797	0,01397585	<sos> a man	2020_12_28	0,0002		
9	configs	cnn-rnn (inception-lstm)	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	445,96492	0,01543441	<sos> a man	2020_12_28	0,0003		
10	configs	cnn-rnn (inception-lstm)	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	436,331262	0,00514567	<sos> a brow	2020_12_28	0,0004		

Ausschnitt aus dem Datenblatt "lernrate" der Datei "auswertung_parameter_excel.xlsx"

Der Parameter der Lernrate kann den meisten Unterschied in der Qualität hervorbringen. Im obigen Bild bilden nun vier Zeilen ein Modell mit denselben Parametern, wobei nur die Lernrate eine andere ist. Die hervorgehobene Zeile entspricht der mit dem höchsten BLEU-4 für das Modell. Die ersten beiden Modelle sind Inception-LSTM-Modelle, die jeweils eine bzw. zwei Schichten haben. Ihr jeweils bestes Ergebnis haben sie mit einer Lernrate von 0,0003.

Für die anderen Architekturen eignen sich scheinbar eher die folgenden Lernraten.

42	configs	attention	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	725,103282	0,01338265	<sos> a brow	2020_12_28	0,0001
43	configs	attention	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	744,46759	0,01690805	<sos> a brow	2020_12_28	0,0002
44	configs	attention	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	727,37676	0,02134142	<sos> a brow	2020_12_28	0,0003
45	configs	attention	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	726,445453	0,02245666	<sos> a brow	2020_12_28	0,0004
46												
47	configs	transformer	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	584,906739	0,00372306	<sos> a brow	2020_12_28	0,0001
48	configs	transformer	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	560,145552	0,00296093	<sos> a brow	2020_12_28	0,0002
49	configs	transformer	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	555,06774	0,00219389	<sos> a boy	2020_12_28	0,0003
50	configs	transformer	flickr8k_deCoco_Val2	32	g6b300d.def	False	10	570,29502	0	<sos> a , , , ,	2020_12_28	0,0004

Ausschnitt aus dem Datenblatt "lernrate" der Datei "auswertung_parameter_excel.xlsx"

Schaut man auf die übrigen Einträge für die anderen Varianten der CNN-RNN-Architektur, dann variiert die Lernrate hier auch.

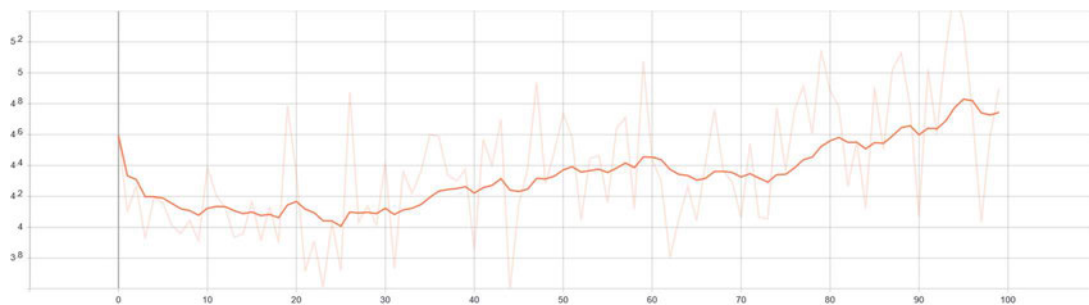
Der letzte Parameter bezieht sich wieder nur auf die CNN-RNN-Architektur, da er sich auf das RNN bezieht. Bei dem letzten Parameter handelt es sich um die Schichten des RNN.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	JSON	Architektur	Trainingsset	Validationset	Batch	Pretrained E	Earlyb	Epoch	Runtime	BLEU	Bsp. Caption	Datum	Layer
2	config	cnn-rnn (inception-1stm)	flickr8k_def	Coco_Val201	32	g6b300d.def	False	10	397,721533	0,0142158	<eos> a man	2020_12_28	1
3	config	cnn-rnn (inception-1stm)	flickr8k_def	Coco_Val201	32	g6b300d.def	False	10	415,871786	0,01278613	<eos> a man	2020_12_28	2
4													
5	config	cnn-rnn (inception-1stm)	flickr8k_def	Coco_Val201	32	g6b300d.def	False	10	406,33582	0,01629929	<eos> a man	2020_12_28	1
6	config	cnn-rnn (inception-1stm)	flickr8k_def	Coco_Val201	32	g6b300d.def	False	10	402,347797	0,01397585	<eos> a man	2020_12_28	2
7													
8	config	cnn-rnn (inception-1stm)	flickr8k_def	Coco_Val201	32	g6b300d.def	False	10	416,459083	0,01793289	<eos> a black	2020_12_28	1
9	config	cnn-rnn (inception-1stm)	flickr8k_def	Coco_Val201	32	g6b300d.def	False	10	445,96492	0,01543441	<eos> a man	2020_12_28	2
10													
11	config	cnn-rnn (inception-1stm)	flickr8k_def	Coco_Val201	32	g6b300d.def	False	10	408,769027	0,00671578	<eos> a black	2020_12_28	1
12	config	cnn-rnn (inception-1stm)	flickr8k_def	Coco_Val201	32	g6b300d.def	False	10	436,331262	0,00514567	<eos> a brow	2020_12_28	2

Ausschnitt aus dem Datenblatt "layers" der Datei "auswertung_parameter_excel.xlsx"

Wie schon in den anderen Tabellen bilden zwei Zeilen ein Modell. Bis auf drei Ausnahmen zeigen alle Modelle ein ähnliches Ergebnis wie die im Bild. Die Modellvariante mit nur einer Schicht scheint besser als zwei Schichten zu sein.

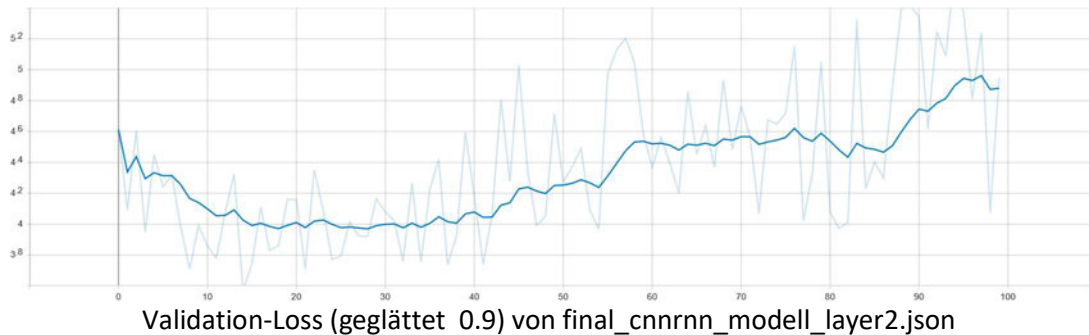
Sollte noch genug Zeit vorhanden sein, dann kann auch ein Modell mit zwei Schichten für einen besseren Vergleich erstellt werden. Für weiteres Training wird neben der einen Schicht des RNN's, die LSTM-Architektur für das RNN genutzt. Da es keinen klaren Sieger zwischen LSTM und GRU gibt, wird das weiterentwickelte LSTM genommen. Für die Lernrate zeigte sich sowohl für einen Layer als auch zwei, eine Lernrate von 0,0003. Für die Attention-Architektur ist laut der Auswertung die Lernrate von 0,0004 am besten. Für einen besseren Vergleich mit dem CNN-RNN-Modell, wird dieselbe Lernrate wie dort gewählt. Dadurch lässt sich gut der Einfluss der Attention am Ende bewerten. Für die Transformer-Architektur war das beste Ergebnis eine Lernrate mit 0,0001. Diese Architektur ist mit die langsamste und komplexeste. Eine Lernrate von 0,0002 ist etwas schwächer bei dem Ergebnis, benötigt tendenziell aber weniger Zeit zum Trainieren bzw. weniger Epochen. Trotzdem wird mit dem langsameren 0,0001 weiter gemacht.



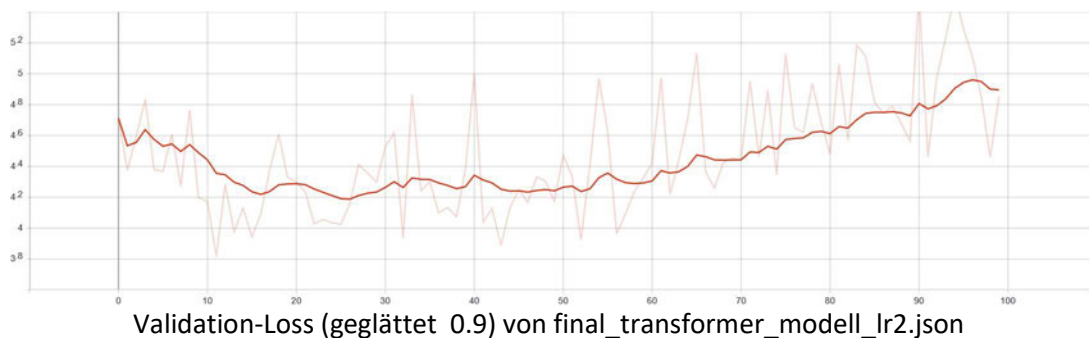
Validation-Loss (geglättet 0.9) von final_cnnrnn_modell_layer1.json

Mittels dem Validation-Loss während des Trainings lässt sich das Overfitting abschätzen. Bei dem Modell aus der JSON-Datei "final_cnnrnn_modell_layer1" handelt es sich um die CNN-RNN-Architektur mit einer Schicht. Dieses wurde mit der zugehörigen Lernrate von 0,0003 trainiert. Den kleinsten Validation-Loss hatte das Modell bei der 25ten Epoche von den insgesamt 100, die trainiert wurden. Im Graph ist zu sehen, dass nach der 30ten Epoche der geglättete Validation-Loss nur noch steigt. Daraus geht hervor, dass zwischen 25ter und 30ter

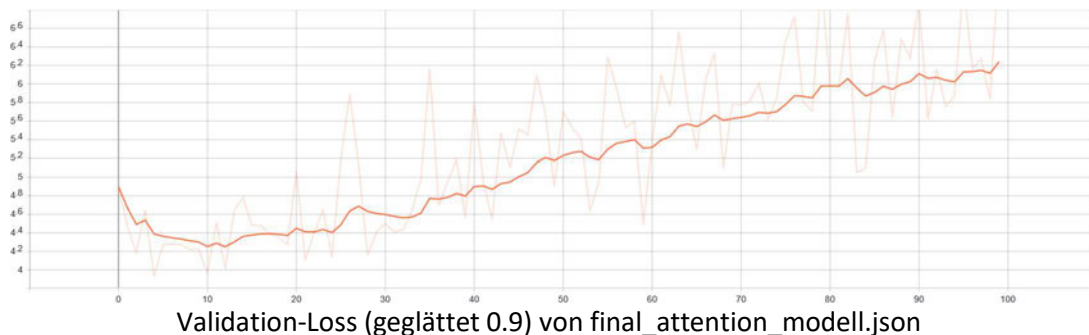
Epoche die Grenze liegt. Nach dieser Grenze ist das Modell bei egal wie vielen Epochen wahrscheinlich schlechter. Für dieses Modell lässt sich das Overfitting auf 30 Epochen festlegen. Für die finale Variante müssen es daher weniger Epochen sein. Es empfiehlt sich dann 25 Epochen zu nehmen, da dies auch der beste geglättete Validation-Loss ist. Der exakte beste Wert wäre schon bei 23 Epochen erreicht.



Bei dem nächsten Modell handelt es sich um die CNN-RNN-Architektur mit zwei Schichten. Hier zeigt sich das die Grenze für das Overfitting bei 35 festlegbar ist, da erst dann eine kontinuierliche Steigung vorhanden ist. Zwischen der 15ten und 35ten Epochen ist der geglättete Wert nahezu konstant. Es scheint das eine höhere Schichtanzahl im RNN für ein konstanteres Training sorgt. Neben der Tatsache, dass bei zwei Schichten das Overfitting einige Epochen später auftritt, hat das Modell bessere geglättete Validation-Loss Werte. Mathematisch gesehen sind mehr Wörter korrekt bei der Vorhersage. Aus diesen beiden Graphen lässt sich abschätzen, dass beide Modelle ähnlich gut Performen werden. Die Variante mit zwei Schichten könnte ein besser sein, wenn auch nur ein klein wenig. Auch bezogen auf die Trainingszeit ist der Unterschied relativ klein. In der Datei "auswertung.csv" sind die Trainingszeiten zu finden. Für das Modell mit einer Schicht sind es 15344 Sekunden. Das zweite Modell brauchte mit 15710 Sekunden schon länger. Im Vergleich zu den ~4.5 Stunden ist der Unterschied von 366 Sekunden sehr gering. Selbst bei dem MSCOCO Datensatz sollte der Unterschied etwa zwei Prozent der Trainingszeit ausmachen. Daher ist die zwei Schichten Variante zu bevorzugen.



Für die Transformer-Architektur sieht das Overfitting schon schwieriger aus. Die geglätteten Werte sind zwischen der 15ten und 50ten Epoche im Schnitt nahe zu konstant. Danach ist eine deutliche Steigung zu erkennen. Nach Epoche 50 lässt sich sagen, dass es auf alle Fälle ein Overfitting gibt. Der beste geglättete Wert liegt jedoch bei Epoche 25. Im Graphen ist danach eine kleine Steigung zu sehen. Bis Epoche 50 sind die geglätteten Werte wieder größer, jedoch nicht stark. Trotzdem könnte argumentiert werden, dass ab Epoche 25, weiteres Training nur überflüssig wäre. Der Validation-Loss deutet von seinen Werten auf keine signifikante Verbesserung hin. Zum Wohle der Zeit sollte das Modell auf dem MSCOCO Datensatz auch nur mit 25 Epochen trainiert werden. Sofern genug Zeit vorhanden ist, lohnt sich eine weitere Variante, die mit 50 Epochen trainiert wurde.



Bei der CNN-RNN-Attention Architektur scheint das Overfitting ab Epoche 12 stattzufinden. Aufgrund der Implementation ist derzeit nur eine Schicht in der Architektur möglich. Der Validation-Loss ist nur ein Indikator für das Overfitting. Der geringe Unterschied in den Werten von Epoche zehn bis 20, könnte noch akzeptabel sein. Das Overfitting könnte daher auch erst ab Epoche 20 beginnen. Da das Overfitting frühestens ab Epoche zehn beginnt, sollte dieser Wert für ein Modell genommen werden. Der Unterschied zu den 20 Epochen eines möglichen späteren Overfittings, darf nicht ausreichend gelassen werden. Immerhin ist der Unterschied die doppelte Trainingszeit. Der beste Weg ist, zwei Modelle zu trainieren und deren BLEU-4 Wert in Hinblick auf die Qualität zu vergleichen. Sollte das Modell, welches mit zehn Epochen trainiert wurde, besser sein, dann beginnt dort schon das Overfitting.

Mit dem MSCOCO-Datensatz wurden nun die Modelle, die aus dem vorherigen Schritt mit dem Flickr30k-Datensatz hervorgingen, neu trainiert. Die entstandenen Modelle wurden, wie in Punkt 6.1.1 beschrieben trainiert, mit den zuvor bestimmten Parametern. Im Folgenden werden folgende Kürzel genutzt. Modell-1 bezeichnet das Modell aus "final_cnnrnn_modell_layer1.json", welches die CNN-RNN-Architektur mit einer Schicht darstellt. Modell-2 bezeichnet das Modell "final_cnnrnn_modell_layer2.json". Modell-3 bezeichnet "final_transformer_modell_lr2.json". Zum Schluss bezeichnet Modell-4 "final_attention_modell.json", welches mit 10 Epochen trainiert wurde. Modell-5 das mit 20 Epochen. Folgende gekürzte BLEU-4 Werte haben die jeweiligen Modelle erzielt.

Modell	Architektur	BLEU-4 / als %	Epochen	Trainingszeit
Modell-1	CNN-RNN	0.1176 / 11.76	25	~2.03h
Modell-2	CNN-RNN	0.1087 / 10.87	35	~2.38h
Modell-3	Transformer	0.1027 / 10.27	25	~2.84h
Modell-4	CNN-RNN-Attention	0.1359 / 13.59	10	~1.55h
Modell-5	CNN-RNN-Attention	0.1349 / 13.49	20	~3.13h

Werte in der Datei "bild_auswertung.xlsx"

Da in einigen wissenschaftlichen Papers der BLEU-Wert auch als Prozent angegeben wird, findet sich beide Varianten in der Tabelle. Die beste Architektur nach den BLEU-Werten wäre die CNN-RNN-Attention Architektur. Hätte Modell-4 dieselbe Epochenanzahl wie Modell-1, wäre es bei ca. 3.87 Stunden und damit fast doppelt so langsam. Das zeigt gut, dass die implementierte Attention-Architektur nicht optimiert ist. Wäre sie optimiert, dann wären die Trainingszeiten für Modell-1 und Modell-4 bei gleicher Epochenanzahl näher beieinander. Der Attention-Mechanismus dürfte nicht ungefähr so viel Trainingszeit einnehmen wie das eigentliche Training. Nimmt man die ungefähre Zeit 3.87 Stunden für 25 Epochen, dann sind davon ca. 2 Stunden für die CNN-RNN-Funktionalität und 1.87 Stunden für den Attention-Mechanismus. Eine besser optimierte/implementierte Version der Architektur wäre erheblich besser als die Standardarchitektur. Selbst durch die Tatsache, dass für die Attention-Variante eine Batch-Größe von 24 genommen werden musste, sollte sie nicht so hoch sein. Der Transformer schneidet am schlechtesten ab, braucht aber die längste Zeit zum Trainieren. Schätzungsweise liegt der Grund entweder an dem Input oder an der Architektur bzw. derer Implementation. Ein Blick auf die Ergebnisse der kompletten Transformer Implementation aus "Image Captive Generation With Adaptive Transformer" zeigt andere Ergebnisse.

Method	Flickr30k					
	<i>B-1</i>	<i>B-2</i>	<i>B-3</i>	<i>B-4</i>	<i>METEOR</i>	<i>CIDER</i>
NIC [2]	0.663	0.423	0.277	0.183	-	-
DeepVS [3]	0.573	0.369	0.240	0.157	-	0.247
Soft-Attention [8]	0.667	0.434	0.288	0.191	0.185	-
Adaptive-LSTM [9]	0.667	0.494	0.354	0.251	0.204	0.531
Ours	0.670	0.496	0.355	0.252	0.204	0.530

(Zhang u.a. 2019, S.525) – Performanz bei Flickr30k

Die Ergebnisse beziehen sich zwar auf das Flickr30k-Set, jedoch sollte das keinen Unterschied machen. In den Ergebnissen von Zhang und seinem Team entspricht die Methode "Soft-Attention" derselben Architektur wie meine CNN-RNN-Attention. Im Gegensatz zur verwendeten Implementation benutzen Zhang und sein Team einen anderen Attention-Mechanismus. Die Ergebnisse meiner eigenen Modelle zeigen die Nützlichkeit eines Attention-Mechanismus. Beide Ergebnisse deuten daraufhin, dass für den Umgang mit

Bildern ein anderer Attention-Mechanismus besser geeignet ist, zumindest für die Transformer-Architektur.

Da der BLEU-Wert nicht perfekt für eine Validierung geeignet ist, werden im Nachfolgenden die Ergebnisse für fünf Bilder betrachtet. Sofern nicht hervorgehoben, hat jedes Modell die Sätze mit einem <eos> begonnen und einem <eos> abgeschlossen. Aus Gründen der Leserlichkeit würden die beiden Token weggelassen in dieser Arbeit. In der "bild_auswertung.xlsx" sind die noch enthalten. Durch diese Betrachtung lässt sich besser sagen, welche Architektur besser ist, zumindest für die jeweiligen Bilder.



Bild-1 - Coco Validation Set 2014 - COCO_val2014_000000000285.jpg

Modell-1:	a brown bear is walking through a grassy
Modell-2:	a brown bear is walking through a grassy area
Modell-3:	a brown bear in a basket in the
Modell-4:	a brown bear is standing in the grass
Modell-5:	a brown bear is laying on a grassy

Durch die Modelle erstellte Sätze für Bild-1

Bei Modell-1, Modell-3 und Modell-5 zeigt sich, dass beide es nicht hinbekommen, den Satz korrekt zu beenden. Modell-2 im Vergleich zu Modell-1 bekommt einen korrekten Satz hin. Das deutet daraufhin, dass die zweite Schicht im RNN nötig ist.

Alle Modelle schaffen es, den Braunbären zu erkennen. Aus dem Bild ist nicht komplett ersichtlich, ob der Bär nun läuft, sitzt oder was genau er macht. Die Bildinformationen, um ein Verb zu bestimmen, sind daher auch kaum vorhanden. Das führt dazu, dass der Vektor zum Bestimmen des Verbs sein Maximum jeweils wo anders hat. Für die Wörter braun und Bär sind die Bildinformationen eindeutig genug, dass jedes Modell dieselben Wörter bestimmt. Im Gegensatz zum Verb ist bei diesen Wörtern das Maximum der Vektoren an derselben Position gewesen. Daher ist es verständlich, dass die Modelle verschiedenen Verben nutzen. Eine Erklärung ist, dass bei den Bildern, wo das Verb nicht genau zu erkennen bzw.

herleiten, da sich das Verb dauernd geändert hat. Die Modelle haben noch nicht genau gelernt was genau in so einem Fall zu produzieren ist. Aufgrund der gelernten Satzstruktur versuchen sie trotzdem ein Verb zu bestimmen. In so einem Fall wäre es besser hätten die Modelle eine alternative Satzstruktur gelernt, was sie offensichtlich jedoch nicht haben.

Bis auf Modell-3 erkennen die Übrigen das Grass im Hintergrund. In der Datei "bild_auswertung.xlsx", im Tabellenblatt "ersteller des datensatzes", finden sich die fünf Beschreibungen zu den Bildern, die von den Erstellern des Datensatzes bereitgestellt wurden. Der ähnlichste Eintrag dort ist "A large bear that is sitting on grass". Abgesehen davon, dass keines der Modelle mit dem Verb richtig lag, sind Modell-2 und Modell-4 am besten. Modell-2 hat den großen Nachteil, dass sich für das Modell der Bär bewegt. Die Beine, die am Bildrand zu erkennen sind, zeigen das der Bär nicht steht. Trotzdem des falschen Verbes ist Modell-4 am besten bei diesem Bild.



Bild-2 - Coco Validation Set 2014 - COCO_val2014_000000000241.jpg

Modell-1:	a man and a woman are playing video games
Modell-2:	a man and a woman are playing with a wii
Modell-3:	a man who is sitting near some rocks and a chair in front of the legs and a seat <pad>
Modell-4:	a group of people playing a video game on a couch
Modell-5:	a man and a woman playing a video game

Durch die Modelle erstellte Sätze für Bild-2

Modell-3 springt sofort einem ins Auge. Die Transformer-Architektur hat es nicht hinbekommen, korrekt den <eos> Token für das Ende zu bestimmen. Stattdessen nimmt es ein <pad> Token, welches nur für das Lernen als Batch nötig ist. Das Token kommt immer

nach dem <eos>. Interessant ist das, dass Modell die höchstmögliche Anzahl an Wörtern genutzt hat. Beim Erstellen des Satzes wird aufgehört, sobald <eos> bestimmt wurde oder nach 20 Wörtern. Dies ist eine Sicherheitsmaßnahme, falls <eos> nie bestimmt werden würde. Auszuschließen ist das der Transformer gelernt hat <pad> abzuleiten, da diese ignoriert werden und somit kein Lernmaterial darstellen. Jedoch bestimmt der Attention-Mechanismus im Transformer die Attention zwischen allen Tokens. In Verbindung mit der Linear-Schicht, die den Featurevektor erstellt, lässt sich das <pad> begründen.

Hinzukommend zum falschen Satzende ist der Satz an sich nicht korrekt. Es wurden Sachen erkannt, die nicht auf dem Bild sind. Dort ist weder ein Stein noch ein Stuhl zu erkennen. Auch hier lässt sich das durch den Attention-Mechanismus begründen. Im Transformer ist die Beziehung zwischen den Wörtern größer. Die RNN-Komponente der anderen Modelle betrachtet nur indirekt frühere Wörter. Ein Wort im RNN hat nur den vorherigen State der ihn beeinflusst, zwar ist dieser auch von dessen vorherigen beeinflusst, jedoch ist es für den aktuellen, als gäbe es nur den einen davor. Eine Kombination aus Wörtern, die noch nie mit einem Wort in Verbindung standen, kann dazu führen, dass ein nicht gelerntes bzw. eines, was bisher keine Verbindung hatte gelernt wird. Dies würde auch die Länge des Satzes erklären. Modell-3 ist für dieses Bild durchgefallen.

Modell-2 erkennt als einziges Modell, das mit einer Wii Spielekonsole gespielt wird. Der Satz von Modell-2 ist dennoch falsch, da er nur zwei Personen erkennt. Im Bild sind jedoch drei Personen komplett zu sehen. Dasselbe gilt für Modell-1 und Modell-5, die einen ähnlichen Satz produziert haben. Das nur ein Modell die Gruppe erkannt hat lässt sich anhand der Trainingsdauer erklären. Modell-5 wurde mit der doppelten Trainingsdauer von Modell-4 trainiert. Diese längere Trainingszeit führte scheinbar dazu, dass das Modell übertrainiert wurde. Das Modell hat wahrscheinlich gelernt, Bilder mit Personen mit "a" zu starten und danach mit "man" weiter zu machen. Die anderen Wörter wie "woman" sind eine Konsequenz aus "man" und den Bildinformationen, die eine Gruppe darstellen. Nur Modell-4 hat noch gelernt, bei einer Gruppe "group" anstelle von "man" zu verwenden.

Modell-4 liefert das beste Ergebnis. Streng genommen spielt aktuell im Bild nur die Person, die steht, da scheinbar abwechselnd gespielt wird. Technisch gesehen ist gespielt daher keiner, der auf der Couch sitzt, so wie der Satz es sagt. Modell-4 liefert wieder das beste Ergebnis, da es korrekt eine Gruppe von Personen erkennt. Es lässt sich auch argumentieren, dass die Personen auf der Couch mitspielen. Nur weil jemand nicht an der Reihe ist, heißt das nicht er spielt nicht mit.



Bild-3- Coco Validation Set 2014 - COCO_val2014_000000000397.jpg

Modell-1:	a pizza sitting on top of a wooden cutting board
Modell-2:	a pizza with olives , mushrooms and cheese on a white
Modell-3:	a pizza with a slice on the board and is getting ready to cut
Modell-4:	a pizza sitting on a table with a large pizza on it
Modell-5:	a pizza sitting on top of a pan on a table

Durch die Modelle erstellte Sätze für Bild-3

Das Erste, was auffällt, ist das Modell-2 eine Aufzählung korrekt darstellt. Das Problem ist, dass das Modell Sachen erkennt, die nicht drauf zu sehen sind. So sind keine Oliven und keine Pilze zu erkennen, sondern eher Kräuter/Blätter. Bei Modell-2 ist der Unterschied zu Modell-1 die zweite Schicht im RNN. Aufgrund der Funktionsweise des RNN's ist anzunehmen, dass nach der ersten Schicht auch ein ähnliches Ergebnis wie bei Modell-1 feststand. Die zweite Schicht hat dann dieses weiter verändert. Offensichtlich führte dies dazu, dass Bilder mit verschiedenen Belag stärker trainiert wurden. Aufgrund dessen, dass die Pizza Kräuter/Blätter als Belag enthält, erklärt dies, wieso sie das einzige Modell ist, welches Belag auflistet. Meine Vermutung ist das die zweite Schicht dafür sorgt, dass das Modell Wichtiges schneller erkennt. Daraus folgt das Unwichtiges weggelassen wird. Für Modell-1 ist der wenig zu sehende Tisch wichtiger als der Belag. Bei Modell-2 ist es dann umgekehrt. In Modell-2 deutet "on a" daraufhin, dass das Modell begonnen hat, eine Ablage zu beschreiben. Modell-1 deutet daraufhin, dass sich der Tisch nicht so spät im Satz ableiten lässt, was für Modell-2 dann bedeutet hat, den Satz zu beenden. Zu sehen an dem nicht korrekten Satzende.

Modell-1 hat es diesmal hinbekommen, einen vollständigen korrekten Satz zu erstellen. An dem Satz ist das einzige Problem, dass der Tisch als Schneidebrett erkannt wird. Dies ist der Architektur zu verschulden. Die Bildinformationen sind so weit im Satz kaum mehr zu erkennen in dem State des Wortes. Der Tisch nimmt so wenig Platz im Bild ein, dass das Modell nur noch "wooden" bestimmen kann. Das Schneidebrett ergibt sich dann aus "wooden" und den noch weniger vorhanden Bildinformationen bezüglich des Tisches.

Modell-3 erkennt wieder Sachen, die nicht auf dem Bild zu erkennen sind. Daher lässt sich der Satz genauso begründen wie im vorherigen Bild. Der Satz von Modell-4 ist interessant, da er die Pizza zwei Mal erwähnt. Das Problem liegt wahrscheinlich daran, dass der Attention-Mechanismus sich nur auf das Bild bezieht. Es zeigt sich dazu das Problem der RNN's nämlich, dass ein Wort nur eine Beziehung zum vorherigen Wort hat. Hat das Modell dann noch dazu gelernt, das zu beschreiben, was auf einem Tisch liegt, dann erklärt sich der Satz. Das Modell kombiniert in diesem Fall scheinbar zwei Satzstrukturen, was dazu führt, dass zwei Mal Pizza erwähnt wird. Der Satz scheint eine Kombination aus "a pizza sitting on a table" und "a table with a large pizza on it" zu sein.

Der Ansatz der Transformer-Architektur ist da hilfreicher, da aus dessen Funktionsweise so was eigentlich nicht hervorgehen dürfte. Der Satz, den Modell-5 produziert, hat das einzige Problem, das eine Pfanne erkannt wird. Im Bild ist eindeutig zu erkennen, dass die Pizza nicht auf einer Pfanne serviert wird. Der Grund dafür ist wahrscheinlich wieder eine Kombination aus zu wenigen Bildinformationen und RNN. Der Satz von Modell-1 ist am besten, da der Tisch nur einen Bruchteil des Bildes ausmacht, im Gegensatz zur Pizza.



Bild-4 - Coco Validation Set 2014 - COCO_val2014_000000000730.jpg

Modell-1:	a red double decker bus parked in front of a building
Modell-2:	a red double decker bus driving down a street
Modell-3:	a red train station - in front of <unk> and gold <unk> and white at the
Modell-4:	a double decker bus is driving down the street
Modell-5:	a red double decker bus driving down a street

Durch die Modelle erstellte Sätze für Bild-4

Die Transformer-Architektur macht wie bisher noch die meisten Probleme. Das Modell möchte Wörter verwenden, die es nicht im Wörterbuch des Modelles gibt, daher die <unk> Token. Durch die Verwendung vom GloVe-Embedding kann es vorkommen, dass spezifische Wörter nicht im Embedding vorhanden sind. Beim Umwandeln in die Tokens wird das Wort nicht gefunden, daraufhin bekommt das Bild ein <unk> an der Position. So kann aus "Der

Koalabär schläft" der Satz "Der <unk> schläft" werden. Der Attention-Mechanismus des Transformers begründet dann wieder, wieso dieser Fall nur hier auftritt. Bei den anderen Modellen hat das direkte Vorwort einen stärkeren Einfluss. Zum wieder fehlenden Satzende kommt hinzu, das der rote Bus als rote Bahnstation erkannt wird. Der Grund wird wahrscheinlich wieder der Attention-Mechanismus sein. Das Modell stellt scheinbar eher eine Verbindung zwischen der Farbe und dem Objekt her. Dies ist dadurch möglich, da der Attention-Mechanismus an den Bildinformationen angewendet wird. Da der dargestellte Bus spezieller aussieht, wird das Ergebnis nach dem Attention-Mechanismus wahrscheinlich näher an dem eines roten Zuges sein. Wenn die Bildinformationen nach dem Attention-Mechanismus dem ähneln, was das Modell aus dem Training bei roten Zügen kennt, dann erklärt es den nicht erkannten Bus.

Modell-2 und Modell-5 haben den exakt selben Satz hervorgebracht. Modell-4 erkennt die Farbe nicht korrekt, hat stattdessen noch das Wort "is". Das Fehlen der Farbe ist diesmal der geringen Trainingszeit zu verdanken. Da Modell-5 das rot und den Doppeldeckerbus erkennt, hat Modell-4 noch nicht gelernt das die Farbe wichtig ist. Daher wird gleich mit Doppeldeckerbus begonnen bzw. als zweites Word verwendet. Bei dem "is" kann ich nicht genau sagen, ob der Satz so richtig ist oder ob für das "is" noch ein "that" davor muss. Anzunehmen ist jedoch, dass das "is" mit der fehlenden Farbe bzw. der Trainingsdauer zusammenhängt. Womöglich hat Modell-4 eine leicht andere Satzstruktur gelernt bzw. diese Satzstruktur einer Vorstufe der nächsten Entspricht, die dann die Farbe auch enthält.

Modell-1 erkennt nicht, dass der Bus zu fahren scheint. Die beiden Konstrukte neben ihm sind auch keine Gebäude. Bei dem hinter dem Bus, im Hintergrund, ist es schwer das festzulegen. Ähnlich wie bei dem Bild mit der Pizza scheinen die Konstrukte neben dem Bus für Modell-1 wichtig zu sein. Das Fahren und die Straße sind dann für Modell-2 wichtiger. Dies bestätigt die Vermutung mit der zweiten Schicht noch weiter.

Modell-2 und Modell-5 haben hier den besten Satz. Alle Modelle haben die Richtung nicht korrekt erkannt, in die der Bus fährt. Auf dem Bild ist zu erkennen, dass er die Straße hochfährt. Die Sätze wären erst korrekt, wenn "up" anstelle von "down" verwendet worden wäre. Der Fehler muss nicht zwangsläufig bei den Architekturen liegen. Es kann sein, das in den Datensätzen mehr Bilder sind, die einen Bus zeigen, der die Straße runterfährt. Es kann auch im Englischen üblich sein, das mit "down" einfach nur von einem Ende der Straße zu einem anderen gemeint ist.



Bild-5 - Coco Validation Set 2014 - COCO_val2014_000000190546.jpg

Modell-1:	a train is coming down the tracks near a
Modell-2:	a train is coming down the tracks near a building
Modell-3:	a train track at a station
Modell-4:	a train traveling down train tracks next to a train station
Modell-5:	a train is traveling down the tracks in the middle of the road

Durch die Modelle erstellte Sätze für Bild-5

Wie bei dem ersten Bild bekommt Modell-1 es nicht hin, den Satz korrekt zu beenden. Eine Erklärung wäre, dass der Zug samt Gebäude als Zug erkannt wird. Die Gebäude links im Bild wären dann nicht mehr ausreichend als solche zu erkennen. Wie schon Modell-2 das bei der Pizza hatte, beendet das Modell den Satz stattdessen. Modell-2 schafft es jedoch, den Satz zu beenden. Das bekräftigt die bisherige Vermutung, dass die zweite Schicht dem Modell sagt, was Wichtiges ist. Modell-1 hat den Satz daher beendet, da es im Bild nichts weiter Wichtiges gibt.

Modell-4 zeigt dasselbe Problem wie schon bei Bild-3. Hinzukommt das der Satz durch das zweite "train" grammatikalisch kein Sinn macht. Der Satz, den Modell-5 generiert, wäre perfekt, wenn es sich bei dem Zug um eine Straßenbahn handeln würde. Das Schienenbett/Gleisbett heißt im englischen auch "Track bed", daher wird in dem Satz sehr wahrscheinlich eine Straße gemeint sein. Vermutlich sorgte die längere Trainingszeit dafür das ein größerer Fokus auf die Schienen bzw. dem Untergrund gelegt wurde. Im Trainingsdatenset gibt es vermutlich kaum Bilder mit "Track bed" in der Beschreibung. Hingegen wird es wahrscheinlich viele Fahrzeuge geben, die sich auf einer Straße fortbewegen. Zumindest in denen der Untergrund wichtig ist und am Ende des Satzes steht. Das Modell wurde für diese Bild übertrainiert. Der Satz eignet sich daher nicht.

Die einzigen korrekten Sätze bringen Modell-2 sowie Modell-3 hervor. Modell-3 ist jedoch das einzige Modell, welches den gut zu erkennenden Zug nicht erkennt. Der Satz ist komplett richtig, da es eine Zugstrecke neben einer Bahnstation zeigt. Ich vermute das der Zug korrekt erkannt wurde, aber aufgrund des Attention-Mechanismus "track" als nächstes Wort kam. Dies passiert, wenn während des Trainings die Wörter einen hohen Attention-Wert hatten. Dies erklärt dann auch wieso nach "train" nicht "is" oder "traveling", wie bei den andere Modellen folgte. Der eigentlich bessere Ansatz des Transformer, hat ihn hier behindert.

Der Zug ist so prägnant im Bild, das dieser auf keinen Fall fehlen darf bei einer Beschreibung. Damit ist nur der Satz von Modell-2 verwendbar. Es wäre besser, wenn im Bild die Station erwähnt würde, aber die Erwähnung eines Gebäudes ist besser als nichts.

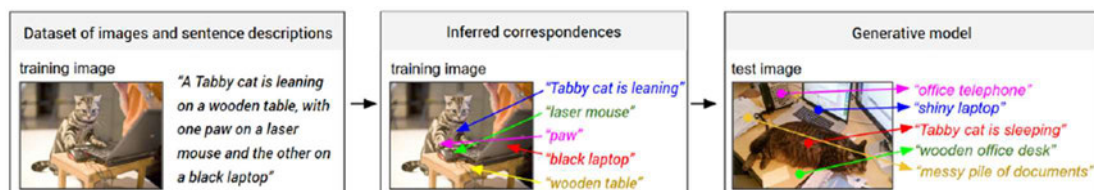
6.2 Ausblick

Die erstellten Modelle zeigen, dass die Einbindung eines Attention-Mechanismus notwendig ist. Gezeigt wurde auch das ein einfacher Attention-Mechanismus, der nur die Wörter mit Pixel in Verbindung bringt, nicht reicht. Es ist wichtig, dass die Wörter wissen, welche Wörter vor ihnen sind. Die wohl beste Zukunft für das generieren von Bildbeschreibungen hat die Transformer-Architektur.

Model	Thai-English	English-Thai
	BLEU	BLEU
SMT	30.71	31.96
RNNs	36.65	25.89
Transformer	44.22	46.48

(Saengthongpattana u.a. 2019, S.3) – Performance bei der Übersetzung

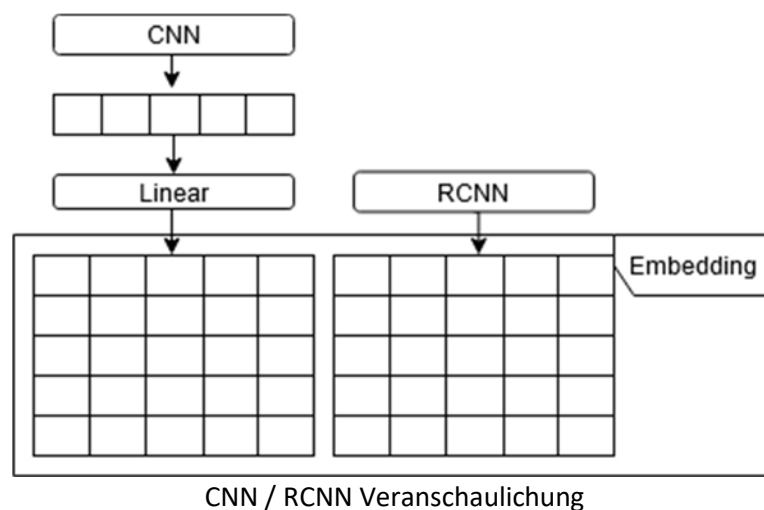
Im Feld der Textübersetzungen schlägt diese Architektur schon längst den früheren Standard. Auch die Ergebnisse von Zhang und seinem Team (S. 55) auf dem Flickr30k Datensatz, zeigen ein ähnliches Ergebnis für die Bildbeschreibungen. Das Bild mit dem Zug hat gezeigt, das so etwas wie BEAM-Search nötig sein kann, bei der Erstellung der Beschreibung. So etwas wie BEAM-Search nimmt dann nicht immer das Wort, welches der Transformer als Nächstes erachtet. So sind Sätze wie "a train on a track" wahrscheinlicher als "a train track", bei denen der Attention-Mechanismus zum selben Ergebnis wie bei in der Auswertung kommt. Eine weitere Verbesserung lässt sich durch andere Architekturen bei den Untermodellen (CNN/RNN) erzielen. Die Verwendung einer Objekterkennung oder eines Regionen-Bezogenen (engl. Region-Based) Ansatzes zeigen Verbesserungen. Diese CNN's nennen sich dann auch RCNN's und zeigten schon gute Verbesserungen bei Objekterkennung.



(Karpathy u. a. 2017, S.666) – Ein weiterer Ansatz an eine Beschreibung

Abhängig von dem Detailgrad des RCNN's bekommt das Modell so alle wichtigen Informationen. Bei der Auswertung zeigte sich in Bild-5, das der Transformer nicht den Zug erkannte. Mit einem RCNN wären die wichtigsten/prägnantesten Elemente erkannt worden. "Im Vergleich zu diesen Ansätzen legt unser Modell den Schwerpunkt auf Einfachheit und Geschwindigkeit zu geringen Kosten der Leistung." (vgl. Karpathy u.a. 2017, S.670). Mit den anderen Ansätzen ist unter anderem eine Architektur ohne RCNN gemeint.

Das von Zhang und seinem Team verwendete Transformer-Modell nutzt ein einfaches CNN. Das Problem der CNN's ist, dass Bildinformationen im Verlauf der Schichten verloren gehen können. Je kleiner ein Element im Bild ist, desto höher ist die Wahrscheinlichkeit, dass es nicht erkannt wird. In dem Paper von Zhang und seinem Team ist für den Transformer in Wort mit einem Feature gleichzusetzen. Jedes Wort wird als X-Dimensionaler Zahlenvektor dargestellt. Aufgrund der Funktionsweise muss es auch bei dem Resultat des CNN's gemacht werden. Ein RCNN stattdessen hätte als Ausgabe keinen herkömmlichen Featurevektor. Bei den RCNN's ist die Ausgabe ein zwei dimensionaler Vektor. Er enthält die einzelnen Regionen mit ihren Featurevektoren. Für den Transformer würde es bedeuten, jedes Wort entspricht einer Region im Bild. Jede Region kann auch als Objekt oder Element im Bild gesehen werden. Das RCNN benötigt dann auch kein Embedding mehr, sofern eingestellt ist, dass die Featurevektoren dieselbe Größe wie das Embedding haben.



Die Kombination der Transformer-Architektur mit einem RCNN haben He und sein Team genommen. Sowohl Zhang und sein Team als auch He und sein Team haben einen unterschiedlichen Attention-Mechanismus genutzt.

model	Bleu1	Bleu4	METEOR	ROUGE-L	CIDEr	SPICE
single-stage model						
Att2all[29]	-	34.2	26.7	55.7	114.0	-
two-stages model						
n-babytalk[24]	75.5	34.7	27.1	-	107.2	20.1
up-down[2]	79.8	36.3	27.7	56.9	120.1	21.4
scene graph based model						
GCN-LSTM*[37]	80.9	38.3	28.6	58.5	128.7	22.1
AUTO-ENC[36]	80.8	38.4	28.4	58.6	127.8	22.1
ALV*[9]	-	38.4	28.5	58.4	128.6	22.0
GCN-LSTM-HIP*[38]	-	39.1	28.9	59.2	130.6	22.3
transformer based model						
Entangle-T*[19]	81.5	39.9	28.9	59.0	127.6	22.6
AoA[14]	80.2	38.9	29.2	58.8	129.8	22.4
VORN[12]	80.5	38.6	28.7	58.4	128.3	22.6
Ours	80.8	39.5	29.1	59.0	130.8	22.8

(He u. a. 2020, S.10) – Vergleich auf dem MSCOCO Datensatz

Wir charakterisieren aktuelle Attention Bildbeschriftungsmodelle in single-stage Modelle, two-stages Modelle, auf scene graph basierende Modelle und transformer basierende Modelle. (vgl. He u. a. 2020, S.3). Bis auf einer Ausnahme erzielen die Modelle, die auf den Transformern basieren, die besten Ergebnisse. Beachten Sie, dass die meisten Modelle, die auf dem Scene-Graph Modell basieren, den semantischen und räumlichen Scene-Graph verschmelzen und Hilfsmodelle benötigten, um den Scene-Graph zuerst zu erstellen, unser Modell ist effizienter in den Berechnungen. (He u. a. 2020, S.10-11)

Noch ist das erfolgreichste Modell für die Bildbeschreibung nicht gefunden. Viele Paper zeigen, dass die Zukunft auf einer Weiterentwicklung der Transformer-Architektur liegt. Des Weiteren zeigen Sie, dass viele nur marginal besser sind. Der Einfluss eines RCNN's über eines CNN's fällt da auch drunter. Neben der Frage nach der Qualität zeigt sich, dass auch die der Geschwindigkeit oder Komplexität nicht zu verachten sind.

6.3 Abschluss

Die eigenen erstellten Modelle haben gut die Wichtigkeit eines Attention-Mechanismus gezeigt. Die Implementation sowie die Qualität der eigenen Modelle kann mit modernen Modellen aus wissenschaftlichen Papern nicht konkurrieren. Alle Probleme der Qualität lassen sich von den Parametern herleiten. So waren zwei Schichten besser als eine bei der CNN-RNN-Architektur. Bei der CNN-RNN-Attention-Architektur war eine Schicht ausreichend, jedoch eine geringere Trainingszeit besser.

Die eigenen Modelle liefert trotz ihrer Qualität für einige Bilder gute Ergebnisse. So hatte bei dem Bild mit dem Zug das Modell-2 den besten Satz, der das Bild korrekt beschreibt. Generell war die Qualität der Modelle von der Qualität der Bilder abhängig. Bei dem Bären war nicht ersichtlich, was er macht, was in den Modellen sichtbar war. Die Qualität der Ergebnisse ist auch subjektiv. Bei der Pizza hat das eine Modell den Belag erkannt, das andere ein Schneidebrett, da der Tisch wenig sichtbar ist. Angenommen, das zweite hätte den Tisch korrekt erkannt, dann wären beide Sätze "a pizza on a wooden table" und "a pizza with topping" korrekt. Die Frage der Qualität wäre dann eher, ob es wichtiger ist, alles von der Pizza zu erkennen oder alles aus dem Bild zu erkennen.

In den meisten Fällen sind die eigenen Modelle durchgefallen und nur für bestimmte Bilder gut genug gewesen. Die Sätze der Modelle wären bei einer Umfrage mit echten Personen größtenteils gefunden worden, da die Qualität für so eine Umfrage nicht gut genug ist. Seit der erstmaligen Erwähnung der Transformer-Architektur kamen schon viele neue verbesserte Architekturen. In Zukunft wird es Weitere geben, die sich immer ein Stück verbessern werden.

Links

Datensets

Flickr8k

<https://www.kaggle.com/dataset/e1cd22253a9b23b073794872bf565648ddbe4f17e7fa9e74766ad3707141adeb>

Flickr30k

<https://forms.illinois.edu/sec/229675>

MSCOCO

<https://cocodataset.org/#download>

Code

Codebasis von Aladdin Persson

https://github.com/AladdinPerzon/Machine-Learning-Collection/tree/master/ML/Pytorch/more_advanced/image_captioning

Literaturverzeichnis

[Albawi u.a. 2017] ALBAWI, Saad ; MOHAMMED, Tareq Abed ; AL-ZAWI, Saad: Understanding of a Convolutional Neural Network. In: *2017 International Conference on Engineering and Technology (ICET)* ; 21-23 Aug 2017, Antalya, Türkei. Phuket, Thailand: ICET 2017 – Elektronische ISBN 978-1-5386-1949-0 – S.1-6 – URL <https://ieeexplore.ieee.org/document/8308186>

[Amrutha u. a. 2018] AMRUTHA, J. ; REMYA, A. A. S.: *Performance analysis of Backpropagation Algorithm of Artificial Neural Networks in Verilog*. In: *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)* ; 18-19 Mai 2018, Bangalore, India. Piscataway, NJ: IEEE 2018 – ISBN 978-1-5386-2441-8 – S.1547-1550 – URL <https://ieeexplore.ieee.org/document/9012614> - Zugriffsdatum: 2020-04-12 – Elektronische ISSN 978-1-5386-2441-8

[Goldberg 2017] GOLDBERG, Yoav: *Neural Network Methods in Natural Language Processing*. Williston, VT: Morgan & Claypool, 2017. (Synthesis Lectures on Human Language Technologies 37). ISBN 9781627052955

[He u.a. 20120] HE, Sen ; LIAO, Wentong ; TAVAKOLI, Hamed R. ; YANG, Michael ; ROSENHAHN, Bodo ; PUGEAULT, Nicolas: *Image Captioning through Image Transformer*. Stand: 2020-04-29. Online unter: <https://arxiv.org/abs/2004.14231> Abruf: 2020-10-13

[Huang u. a. 2017] HUANG, Jonathan ; RATHOD, Vivek ; SUN, Chen ; ZHU, Menglong ; KORATTIKARA, Anoop ; FATHI, Alireza ; FISCHER, Ian ; WOJNA, Zbigniew ; SONG, Yang ; GUADARRAMA, Sergio ; MURPHY, Kevin: *Speed/Accuracy trade-offs for modern convolutional object detectors*. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 21-26- Juli 2017, Honolulu, HI. Piscataway, NJ: IEEE 2017– ISBN 978-1-5386-0458-8 – S.3296-3305 – URL <https://ieeexplore.ieee.org/document/8099834> - Zugriffsdatum: 2020-04-17

[Johnson 2017] JOHNSON, Justin: Lecture 10 | Recurrent Neural Networks: Vortrag / Stanford University 2017-08-11, Stanford, Kalifornien - <https://www.youtube.com/watch?v=d14TUNcbn1k&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=4> - Abruf: 2020-08-18

[Karpathy u. a. 2017] KARPATY, Andrej ; FEI-FEI, Li: *Deep Visual-Semantic Alignments for Generating Image Descriptions*. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2017), April, Nr. 4, S. 664-676 – URL:

<https://ieeexplore.ieee.org/document/7534740> - Zugriffsdatum: 2020-04-12 – Elektronische ISSN 1939-3539

[Kruse u. a. 2011] KRUSE, Rudolf ; BORGELT, Christian ; KLAWONN, Frank ; MOEWES, Christian ; RUß, Georg ; STEINBRECHER, Matthias: *Künstliche neuronale Netzwerke*. In: BIBEL, Wolfgang (Hrsg.) ; KRUSE, Rudolf (Hrsg.) ; NEBEL, Bernhard (Hrsg.): *Computational Intelligence : Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. Vieweg+Teubner Verlag, 2011, Wiesbaden, Deutschland. - ISBN 978-3-8348-1275-9 – S. 5-149 – Online unter: https://link.springer.com/chapter/10.1007/978-3-8348-8299-8_2#citeas

[Phi 2018] PHI, Michael: Illustrated Guide to LSTM's and GRU's: A step by step explanation. Stand 2018-09-24. Online unter: <https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21> Abruf: 2020-08-11

[Redmon u. a. 2017] REDMON, Joseph ; FARHADI, Ali: *YOLO9000: Better, Faster, Stronger*. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 21-26 Juli 2017, Honolulu, HI. Piscataway, NJ: IEEE 2017 – ISBN 978-1-5386-0485-8 – S.6517-S.6525 – URL <https://ieeexplore.ieee.org/document/8100173> - Zugriffsdatum: 2020-04-17

[Redmon u. a. Slides 2017] REDMON, Joseph ; FARHADI, Ali: *YOLO9000: Better, Faster, Stronger*. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 21-26 Juli 2017, Honolulu, HI. Piscataway, NJ: IEEE, 2017. – https://docs.google.com/presentation/d/14qBAiyhMOFI_wZW4dA1CkixgXwf0zKGBpw_0oHk8yEM/edit#slide=id.p – Zugriffsdatum: 2020-04-17

[Srivastava u. a. 2014] SRIVASTAVA, Nitish ; HINTON, Geoffrey ; KRIZHEVSKY, Alex ; SUTSKEVER, Ruslan: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. In: *The Journal of Machine Learning Research* 15 (2014), 1, S.1929-1958. – Online unter: <https://dl.acm.org/doi/10.5555/2627435.2670313> Abruf: 2020-06-16

[Tompson u.a. 2015] TOMPSON, Jonathan ; GOROSHIN, Ross ; JAIN, Arjun ; LECUN, Yann ; BREGLER, Christopher: Efficient Object Localization Using Convolutional Networks. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 7-12 Jun. 2015, Boston, USA. Piscataway, NJ: IEEE, 2015. – ISBN 978-1-4673-6964-0. – S.648-656 – Online unter: <https://ieeexplore.ieee.org/document/7298664/figures#figures> Abruf: 2020-16-06

[Vaswani u. a. 2017] VASWANI, Ashish ; SHAZEER, Noam ; PARMAR, Niki ; USZKOREIT, Jakob ; JONES, Llion ; GOMEZ, Aidan N. ; KAISER, Lukasz ; POLOSUKHIN, Illia: *Attention Is All You*

Need. In: GUYON, Isabelle (Hrsg.) ; VON LUXBURG, Ulrike (Hrsg.) ; BENGIO, Samy (Hrsg.): *Advances in Neural Information Processing Systems 30* ; 4-9 Dez. 2017, Long Beach, USA. Montreal, Québec: Curran Associates Inc., 2018. – ISBN 978-1-5108-6096-4 – S.5998-6008 – Online unter: <https://papers.nips.cc/paper/7181-attention-is-all-you-need> - Abruf 2020-04-18

[Vedantam u.a. 2015] VEDANTAM, Ramakrishna ; ZITNICK, C. Lawrence ; PARIKH, Devi: CIDEr: Consensus-based Image description evaluation. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 7-12 Jun. 2015, Boston, USA. Piscataway, NJ: IEEE, 2015 – ISBN 978-1-4673-6964-0 – S.4566-4575 – Online unter: <https://arxiv.org/abs/1411.5726> Abruf: 2020-06-23

[Vinyals u. a. 2015] VINYALS, Oriol ; TOSHEV, Alexander ; BENGIO, Samy ; ERHAN, Dumitru: *Show and Tell: A Neural Image Caption Generator*. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* ; 7-12 Juni 2015, Boston, USA. Piscataway, NJ: IEEE, 2015 – Electronic ISBN 978-1-4673-6964-0 – S.3156-3164 – URL <https://ieeexplore.ieee.org/document/7298935> - Abruf: 2020-04-12

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____

