Deep Learning - Aufgabenblatt 5

André Glatzl, Beat Brändli, Leandro Gregorini, Raphael Brunold

Abgabetermin: 16.06.2023

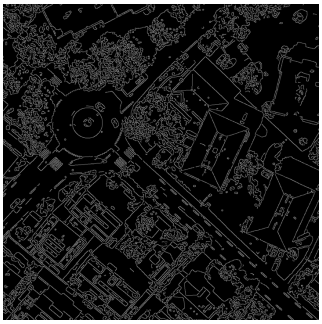
Problemstellung

Wir wollen mit unserem Deep Learning Modell Kreisel aus der Vogelperspektive erkennen.
Das könnte für den Bund interessant sein, um beispielsweise die Verkehrsplanung zu verbessern. Mit Hilfe der Verkehrs-/Staudaten und der Information, ob sich ein Kreisel in der Nähe diese Staus befindet, könnte man Rückschlüsse über die Effektivität von diesem Kreisel ziehen.

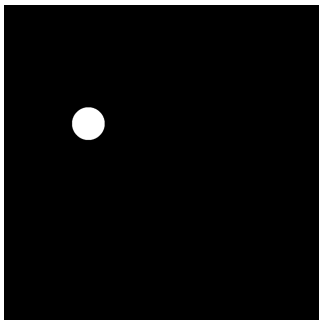
Datensatz

Im Rahmen dieses Projekts war es erforderlich, erste Erfahrungen im Bereich der Datenannotation zu sammeln. Datenannotation bezieht sich darauf, eigene Daten mit Labels zu versehen, um spezifische Fragestellungen beantworten zu können. Alexander van Schie hat ein Tool entwickelt, dass die SwissImage 10cm-Bilder in einer gewünschten Grösse zuschneidet und anschliessend die Annotation ermöglicht. In unserer Arbeit haben wir gemeinsam in etwa zehn Schweizer Städten nach Kreiseln gesucht und diese annotiert. Am Ende hat der SwissImage Annotator die Bilder in zwei Ordner getrennt, "y" und "n", um später eine binäre Klassifikation durchführen zu können. Der rohe Datensatz ist wie folgt aufgeteilt: 310 Bilder enthalten Kreisel und 3474 Bilder enthalten keine Kreisel. Der Datensatz weist eine deutliche Unbalanzierung auf. Um diese entgegenzuwirken, haben wir jeweils 300 Bilder aus jeder Klasse ausgewählt und diese in Trainingsdaten (200 Bilder), Validierungsdaten (50 Bilder) und Testdaten (50 Bilder) aufgeteilt. Für die Datenaugmentierung haben wir die OpenCV-Bibliothek verwendet, um zu versuchen, die Kreisel anhand von Edge Detection und Hough Circle Transformation zu verdeutlichen und diese zusätzlichen Informationen in das Modell zu integrieren.

(a) Natürlich



(b) Edge Detection



(c) *HoughCircles*

Abbildung 1: Drei Arten des Datensatzes

Modell-Architektur

Nach der Annotation und Vorbereitung des Datensatzes, wurden mehrere Deep Learning Modelle trainiert. In folgenden Abschnitten werden zwei der Modelle genauer beschrieben. Die Aufgabe der Modelle ist die binäre Klassifikation der Satelliten-Bilder (Abbildung 1a), wobei vorhergesagt werden soll, ob sich ein Kreisel (Output=1) im Bild befindet oder nicht (Output=0). Ergebnisse der Modelle sind im Absatz **Resultate** erläutert. Beim Testen von verschiedenen Architekturen wurde schnell klar, dass Data Augmentation, aufgrund des kleinen Datensatzes, in irgend einer Form eingesetzt werden muss.

Architektur 1 Eines der besten Modelle war ein etwas komplexeres Convolutional Neural Network mit Data Augmentation. Das Modell besteht aus 4 Convolution Layer (32 Nodes, 32 Nodes, 64 Nodes und 64 Nodes) und einem Dense Layer mit 512 Nodes. Nach den Convolutional Layer wird in diesem Modell, anstelle der weitverbreiteten flatten Methode, GlobalAveragePooling2D[1] eingesetzt. Dies wendet das Average-Pooling auf die räumlichen Dimensionen an, bis jede räumliche Dimension eins ist, und lässt die anderen Dimensionen unverändert. Beim Global Pooling werden alle Feature Maps in einer einzigen Feature Map zusammengefasst und alle relevanten Informationen gebündelt, die von einem einzigen Dense Layer leicht verstanden werden können[2]. Das Modell wurde anhand des originalen Datensatzes bestehend aus den Satelliten-Bildern (Abbildung 1a) trainiert. Um Overfitting zu minimieren wurde Dropout an verschiedenen Stellen eingesetzt. Totale Anzahl trainierbare Parameter beträgt bei diesem Modell: 99361. Das Training von 50 Epochen dauerte insgesamt 25 Minuten mit der A100 Nvidia GPU (Server: mercury).

[1] `https://keras.io/api/layers/pooling_layers/global_average_pooling2d/`
[2] `https://stackabuse.com/dont-use-flatten-global-pooling-for-cnns-with-tensorflow-and-keras/`

Architektur 2 Da der Datensatz anhand von OpenCV und Hough Circle Transformation erweitert wurde (ersichtlich in Abbildung 1), musste eine geeignete Architektur aufgebaut werden, welche mit den drei Input-Datensätzen umgehen kann, da die Daten nicht miteinander vermischt werden dürfen. Wir haben uns dafür entschieden, ein **Multi-Input Modell** aufzubauen. Dieses Modell besteht aus drei einzelnen Convolutional Neural Networks, welche jeweils anhand einer der drei Datensätzen (a, b oder c) trainiert wird. Die Outputs der einzelnen Modelle werden anhand eines Concatenate Layer[3] zusammengefügt und einem herkömmlichen Feed Forward Neural Network übergeben. Im Concatenate Layer wird eine Liste von Eingaben verkettet. Dieser Layer nimmt als Eingabe eine Liste von Tensoren, die bis auf die Verkettungsachse alle die gleiche Form haben und gibt einen einzigen Tensor zurück, der die Verkettung aller Eingaben ist. Die Dimension des Tensors wird anhand von GlobalAveragePooling2D und ResizingLayer verkürzt. Danach findet die schlussendliche binäre Prediction statt. Um Overfitting zu minimieren wurde hier ebenfalls Dropout an verschiedenen Stellen eingesetzt. Da es sich um ein grosses Modell handelt, wurden die Bilder etwas verkleinert um das Training zu ermöglichen. Eine Visualisierung des Modellaufbaus ist im Anhang zu finden (Abbildung 2).

Pretrained Model Zusätzlich zu den beschriebenen Architekturen wurden ebenfalls Modelle auf Basis von Pretrained Models gebaut und getestet. Unter anderem wurde mit den Modellen Resnet50[4] und VGG16[5] auf Basis des ImageNet Datensatzes experimentiert.

Resultate

Tabelle 1 zeigt einige erstellte Modelle, sowie deren Accuracy und Loss. Für die Loss-Metrik wurde Binary Crossentropy ausgewählt. Als **Baseline** wurde ein einfaches CNN erstellt, um einen Ausgangspunkt zur Genauigkeit der Modelle zu ermitteln. Bei den einfachen Modellen zeigte sich schnell, dass der verhältnismässig kleine Datensatz anfällig auf Overfitting ist (100% Accuracy auf den Trainings-Bildern und nur 63% auf den Test-Bildern).

Model	Train Acc.	Train Loss	Val. Acc	Val. Loss	Test Acc.	Test Loss
Basic CNN	100%	0	58%	1.256	63%	1.169
ResNet 50	67.80%	0.604	74%	0.585	67%	0.624
ResNet 50 FT	65.80%	0.620	74%	0.567	71%	0.608
Architektur 1	78.50%	0.440	72%	0.593	78%	0.483
Architektur 2	74.75%	0.621	65%	0.650	81%	0.599

Tabelle 1: Vergleich der fünf Modelle, FT steht für Fine Tuning

Mit der Verwendung einer pretrained Convolutional Base konnten genauere Modelle trainiert werden und mit Fine Tuning konnten weitere Verbesserungen erzielt werden. Auch das starke Overfitting der einfachen Modelle konnte gemindert werden. Mit der **Archtektur 1** (erweitertes CNN) konnten nochmals bessere Resultate erzielt werden. Dies hat uns gezeigt, dass eine geeignete Modellarchitektur einen grossen Unterschied machen kann. Auch die **Architektur 2**, (das Multi-Input Modell) hat interessante Resultate hervorgebracht. Auch wenn es eine leicht höhere Test Accuracy als die Architektur 1 aufweist, sind der Loss auf den Testdaten, sowie Accuracy und Loss auf den Trainings- und Validierungsdaten schlechter. Aus diesem Grund haben wir **Architekur 1** als **bestes Modell** ausgewählt.

Im Anhang sind einige Visualisierungen zur Performance der Architektur 1 aufgeführt. Es wurden Accuracy und Loss während des Trainings dargestellt und eine Konfusionsmatrix erstellt (Abbildung 3, 4, 5). Die Konfusionsmatrix zeigt, dass es recht wenige False Negatives gibt, also Bilder von Kreiseln, die aber nicht als Kreisel erkannt werden.

Persönliche Erkenntnisse

Insgesamt hat uns das Projekt Spass gemacht und wir konnten einige neue Inhalte der Vorlesung einsetzen. Uns ist aufgefallen, dass vor allem die Datenannotation ziemlich aufwendig ist und die Datenqualität stark von diesem Prozess abhängt. Eine Challenge bei der Annotation war es, mit Ausschnitten umzugehen, bei welchen nur ein Teil des Kreisels zu sehen war. Wir haben uns dafür entschieden auch diese Bilder als Positiv zu markieren. Dies bietet jedoch eine zusätzliche Schwierigkeit für die Modelle.

Zu Beginn unseres Tests haben wir den Flatten Layer in das Modell integriert, wodurch insgesamt etwa 2 Milliarden Parameter erzeugt wurden. Durch die Verwendung von GlobalAveragePooling2D konnten wir diese Anzahl auf 3500 Parameter reduzieren. Generell ermöglicht GlobalAveragePooling eine signifikante Verringerung der Anzahl zu trainierender Parameter.

Es war sehr interessant, zum ersten Mal ein Multi-Input Modell umzusetzten. Die grösste Herausforderung hier war der Concatenate Layer, welcher die Resultate der einzelnen Modelle zusammenführt.

Trotz wenig Daten haben wir ein einigermassen gutes Ergebnis erzielt. Es ist jedoch wichtig zu erwähnen, dass unser aktuelles Modell zu schlecht ist um es in der Praxis einzusetzen. Etwa jede vierte Vorhersage ist falsch. Wir gehen davon aus, dass mit mehr Daten noch bessere Resultate erzielt werden können.

[3] `https://keras.io/api/layers/merging_layers/concatenate/`
[4] `https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/ResNet50`
[5] `https://www.tensorflow.org/api_docs/python/tf/keras/applications/vgg16/VGG16`

Anhang

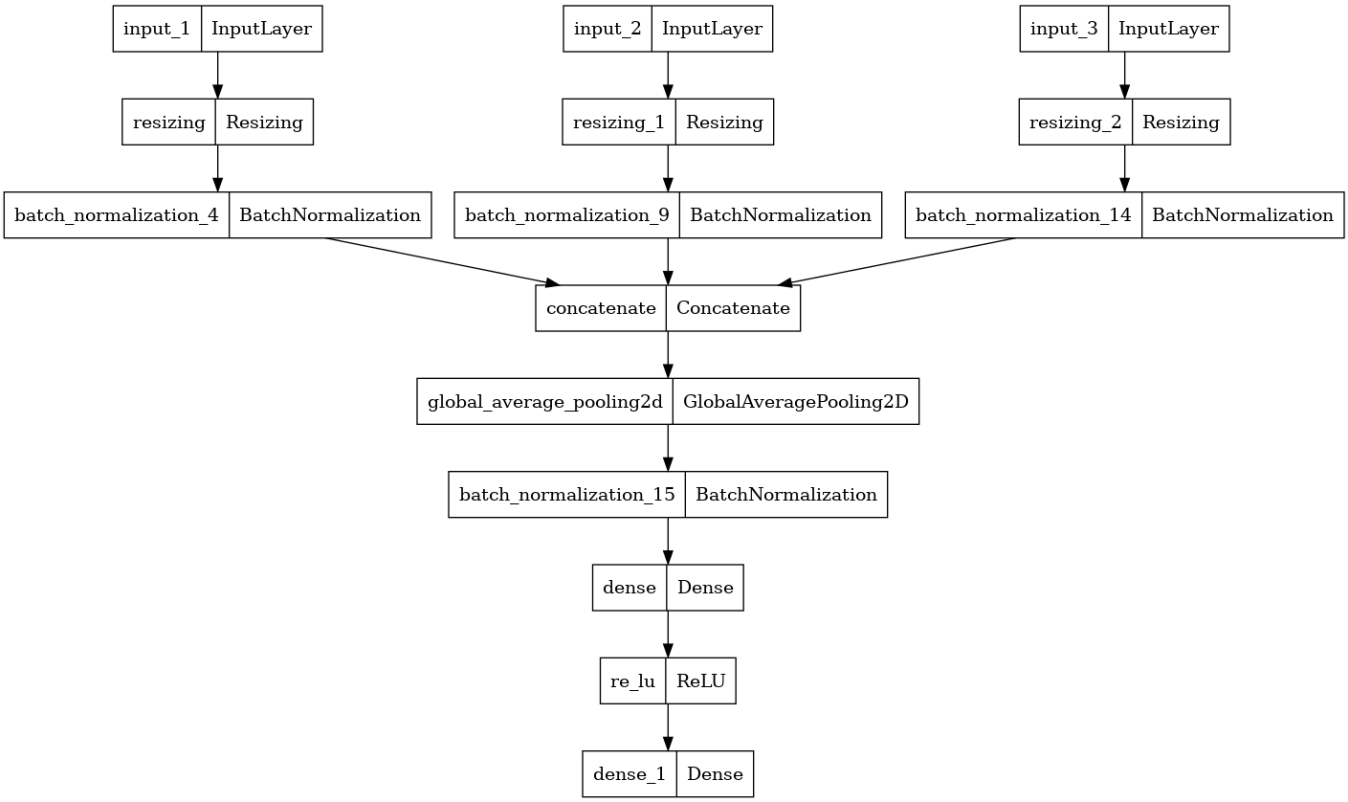


Abbildung 2: Modellübersicht - Architektur 2

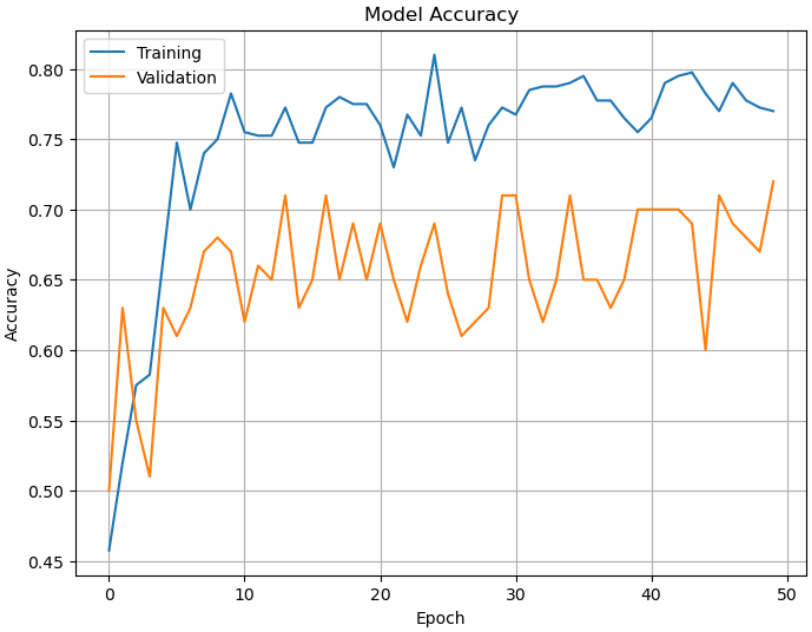


Abbildung 3: Accuracy während des Trainings - Architektur 1

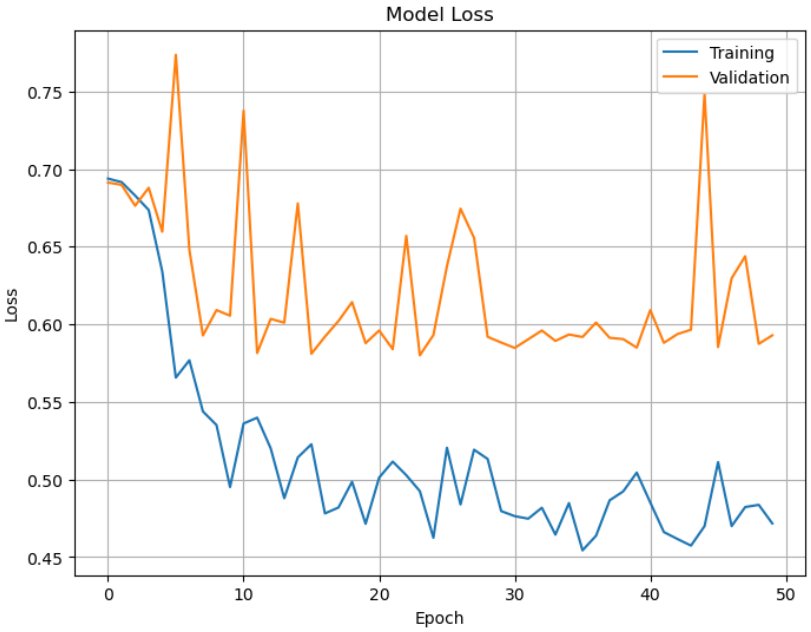


Abbildung 4: Loss während des Trainings - Architektur 1

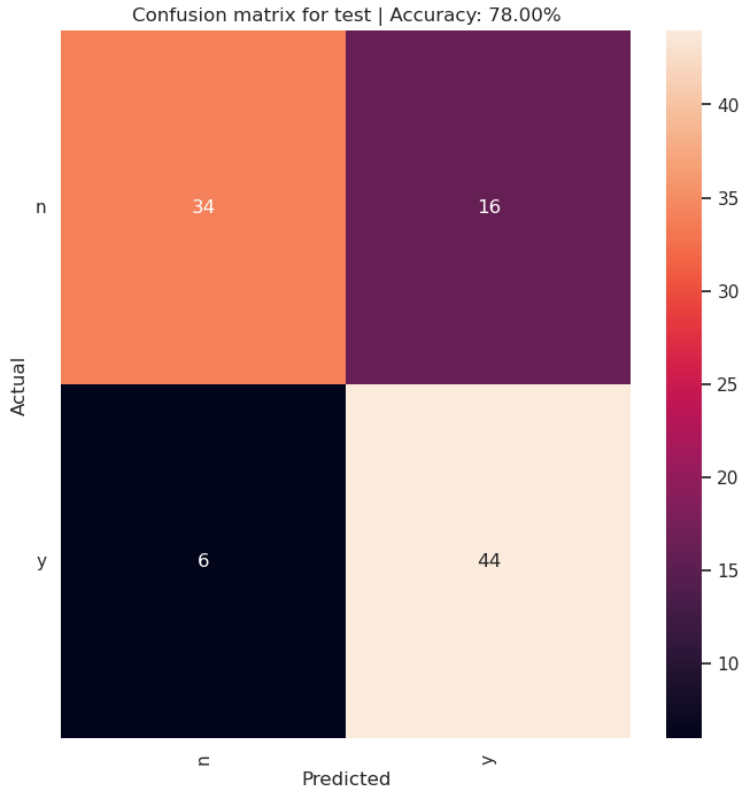


Abbildung 5: Konfusionsmatrix - Architektur 1