Bewegungserkennung auf mobilen Geräten mit Verwendung von GANs für eine automatische Datensatzgenerierung

Master-Thesis

Florian Hansen
Hochschule Flensburg

6. Juni 2021

Inhaltsverzeichnis

1 Einleitung

2 Grundlagen

2.1 Notationen

In dieser Arbeit werden verschiedene Notationen aus der Statistik und dem Machine-Learning-Umfeld verwendet und sollen hier aufgrund der Les- und Verständlichkeit aufgelistet werden.

Der Term $\mathbb{E}_{x\sim P}[f(x)]$ stellt den Erwartungswert einer Verteilung P dar und liest sich als *erwarteter Wert von $f(x)$ unter x verteilt als P.*

2.2 Lipschitz-Funktionen

2.3 Kullback-Leibler-Divergenz

Die Kullback-Leibler-Divergenz (KL-Divergenz) misst, wie sehr sich zwei Verteilungen voneinander unterscheiden und hat seinen Ursprung in der Informationstheorie.

Definition 1 (Kullback-Leibler-Divergenz [1]) *Seien P und Q zwei Wahrscheinlichkeitsfunktionen über den gleichen Wahrscheinlichkeitsraum X. Dann ist der Abstand bzw. die Divergenz der beiden Verteilungen definiert als*

$$D_{KL}(P||Q) = \sum_{x\in X} P(x) \log \frac{P(x)}{Q(x)}.$$

Dabei gibt $P||Q$ eine Divergenz von der Ausgangsverteilung P zur Zielverteilung Q an. Das Messen der Divergenz zwischen zwei Wahrscheinlichkeitsverteilungen findet insbesondere im Machine-Learning statt, um künstliche neuronale Netze und ihre Gewichte zu trainieren. Deshalb kann die KL-Divergenz auch als Loss-Funktion verwendet werden. Bemerkenswert ist hierbei, dass die KL-Divergenz asymmetrisch ist, also

$D_{KL}(P||Q) \neq D_{KL}(Q||P)$. Die Distanz zwischen zwei Verteilungen unterscheidet sich demnach je nach Ausgangsverteilung.

2.4 Jensen-Shannon-Divergenz

Definition 2 (Jensen-Shannon-Divergenz [1]) *Seien P und Q zwei Wahrscheinlichkeitsfunktionen über den gleichen Wahrscheinlichkeitsraum X. Dann ist die Jensen-Shannon-Divergenz der beiden Verteilungen definiert als*

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M) \qquad mit \;\; M = \frac{1}{2}(P+Q)$$

Die Jensen-Shannon-Divergenz kann als Erweiterung der Kullback-Leibler-Divergenz angesehen werden. Im Gegensatz zur Kullback-Leibler-Divergenz ist die Jensen-Shannon-Divergenz (JS-Divergenz) symmetrisch. Das bedeutet, dass der Abstand zwischen zwei Wahrscheinlichkeitsverteilungen gleich groß ist, egal von welchen er beiden Distributionen aus betrachtet wird.

2.5 Wasserstein-Abstand

Eine weitere Methode zum Messen des Abstands zwischen zwei Wahrscheinlichkeitsverteilungen ist die Berechnung des Wasserstein-Abstands.

Definition 3 (Wasserstein-Abstand [1]) *Seien P_r und P_g zwei Wahrscheinlichkeitsverteilungen, dann ist der Wasserstein-Abstand definiert als*

$$W(P_r, P_g) = \inf_{\gamma \in \Pi(P_r, P_g)} \mathbb{E}_{(x,y)\sim\gamma} \left[\|x - y\| \right],$$

wobei $\Pi(P_r, P_g)$ die Menge aller gemeinsamen Verteilungen $\gamma(x, y)$ darstellt, dessen Grenzen P_r und P_g sind.

Der Term $\gamma(x, y)$ stellt dabei die *Masse* dar, die von x nach y transportiert wird, um schließlich die Verteilung P_r in die Verteilung P_g umzuformen. Aus diesem Grund ist der Wasserstein-Abstand auch als *Earth-Mover-Abstand* (EM-Abstand) bekannt.

3 Generative Adversarial Networks

In Machine-Learning existieren viele verschiedene Modelle, die vorhandene Datensätze analysieren und anhand der Daten lernen, Strukturen in den Datensätzen zu erkennen. Besitzt man beispielsweise einen Datensatz bestehend aus Fotoaufnahmen von Tieren, so kann ein Klassifizierer trainiert werden, um einem Bild eine Tierklasse zuzuweisen. Aus diesem Grund fässt man diese Modelle unter dem Begriff *Bildklassifizierung* zusammen.

Wesentlich interessanter ist das Erkennen von vielen Objekten innerhalb eines Bildes, anstatt das gesamte Bild nur einer einzigen Klasse zuzuweisen. In der *Objekterkennung* entwickelt man Modelle, welche mehr als nur eine Klasse erkennen können. Sie liefern zusätzlich zu den erkannten Klassen ihre Position und Größe innerhalb des Bildes. Diese Modelle treffen also keine Aussage über das Gesamtbild, sondern treffen Aussagen über einzelne Objekte innerhalb des Bildes.

Neben Modellen, die zu einem bestimmten Sachverhalt eine Aussage treffen können, existieren auch Modelle, welche in der Lage sind, neue Sachverhalte zu erzeugen. Diese fallen unter dem Begriff *Generative Adversarial Networks* (GANs) und bilden das Hauptthema dieses Abschnitts. Das interessante an diesen generativen Modellen ist, dass sie nicht nur die Strukturen eines Datensatzes lernen, sondern darüber hinaus neue Elemente der Ausgangsdistribution erzeugen können. Trainiert man also ein generatives Modell auf einen Datensatz, welcher Bilder von verschiedenen Tieren enthält, können neue Bilder der gleichen Art erzeugt werden.

Aber nicht nur zum Erzeugen von Bildern kann diese Art von Modellen verwendet werden. Auch bei Aufgaben, bei denen eine Voraussagung getroffen werden soll, werden generative Modelle eingesetzt. Beispielsweise wurde in [2] gezeigt, wie zu bereits getätigten menschlichen Bewegungen unterschiedliche, darauf folgende Bewegungssequenzen aussehen können. Hier hat man also versucht, eine Vorhersage zur Entwicklung von menschlichen Bewegung zu tätigen.

Die Funktionsweise von GANs ist im Prinzip ziemlich simpel. Während beim klassischen supervised-learning in der Regel nur ein Modell beim Training involviert ist, verhält

sich das bei generativen Modellen etwas anders. Zum Einen wird ein Generator definiert, welcher, wie sein Name andeutet, Ausgaben selbst erzeugt. Zum Anderen wird ein Diskriminator in das Training eingebaut, welcher zwischen künstlich erzeugten und reellen Daten unterscheidet. Diese beiden Modelle werden dann gleichermaßen trainiert. Während der Generator versucht, Fälschungen immer genauer zu erzeugen, versucht der Diskriminator immer besser zwischen Fälschung und Realität zu unterscheiden. Die Ausgabe des Diskriminators ist dementsprechend entweder 0 für Fälschung und 1 für Realität. Mit anderen Worten, die beiden Komponenten spielen Spiel, in welchem die eine Partei versucht, die andere zu täuschen [3].

$$\min_G \max_D V(G, D) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

Im Verlauf des Trainings entwickelt sich damit ein Generator, welcher im Idealfall so gute Fälschungen erzeugt, sodass sich diese nicht mehr von Daten der Ausgangsdistribution unterscheiden lassen. Der Diskriminator kann hier bestenfalls nur raten, kann also eine Genauigkeit von höchstens 50% erreichen. Ist dies nicht der Fall, d.h. der Diskriminator kann Fälschungen mit einer höheren Wahrscheinlichkeit von realen Daten unterscheiden, so entsteht ein Ungleichgewicht. Aus diesem Grund sollten die Lernparameter sorgfältig ausgewählt und untersucht werden, damit ein stabil laufendes GAN trainiert wird.

3.1 Das Mode-Collapse-Problem

Ein großes Problem beim Trainieren von generativen neuronalen Netzen ist, dass sich der Generator sehr häufig auf bestimmte Merkmale der Ausgangsdistribution des Datensatzes fixiert. Das Ergebnis sind signifikant erhöht wiederkehrende Ergebnisse, die sich kaum bis gar nicht von anderen Ausgaben unterscheiden. Man erwartet jedoch, dass das jeweilge GAN eine vielseitige Variation aus allen Elementen des Datensatzes erzeugt. Mit anderen Worten, bei einer zufälligen Eingabe in das Netz, soll immer eine unterschiedliche Ausgabe erzeugt werden. Bei einem Mode-Collapse ist dies nicht der Fall. Es kann beispielsweise passieren, dass wenn das Netz auf das Erzeugen von neuen Gesichtern trainiert wird, dass dieses ausschließlich weibliche Gesichter erzeugt, weil das Netz herausgefunden hat, dass es einfacher ist, weibliche Gesichtszüge zu generieren, als männliche [6]. Dies lässt sich damit erklären, dass der Generator beim Trainingsvorgang mehr Erfolg beim Generieren von weiblichen Gesichtern hatte und der Diskriminator es schwerer hatte, Fälschung von Realität zu unterscheiden. Um das Problem zu beseitigen

wurden einige Erweiterungen an dem Standardmodell des GAN von [3] hinzugefügt.

3.2 Deep Convolution GAN

Das *Deep Convolution GAN* (DCGAN) ist ein Versuch, *Convolutional Neural Networks* (CNNs) mit GANs zu verküpfen. Nach vielen Fehlschlägen in der Entwicklung von GANs mit CNNs ist die Version von [5] stabil und auf viele unterschiedliche Datensätze anwendbar. Dafür wurden viele verschiedene Kombinationen von Schichten untersucht und es wurde dabei eine Architektur ausgearbeitet, die in ein stabiles Training über verschiedenste Datensätze resultierte. Zusätzlich können mithilfe dieser Architektur höherere Auflösungen und tiefere Netze erreicht werden.

Zusätzlich zur eigentlichen Architektur von DCGAN werden moderne Techniken verwendet, um CNN-Architekturen zu vereinfachen. Damit der Generator über mehrere Schichten hinweg die räumliche Darstellung von Objekten lernen kann, werden Convolutional-Layer verwendet. Anstatt, dass sogenannte Max-Pooling-Layer zum Einsatz kommen, können nach [7] einfach Convolutional-Layer mit erhöhtem Stride verwendet werden, ohne dass die Genauigkeit sinkt. In Bezug zu DCGANs von [5] werden solche Schichten verwendet, um dem Generator das Erlernen vom räumlichen Upsampling zu ermöglichen. Auch der Diskriminator wird mit solchen CNN-Layer ausgestattet, um räumliches Downsampling zu erlernen.

Neben dem Auslassen von Max-Pooling-Layer folgt DCGAN auch dem Trend, Fully-Connected-Layer vor jedem Convolutional-Feature zu vermeiden. Dabei wurde festgestellt, dass die Verknüpfung von Fully-Connected-Layer und der Eingabe des Generators bzw. mit der Ausgabe des Diskriminators am besten funktionieren. Die erste Schicht des Generators ist also ein Fully-Connected-Layer (1-dimensional), jedoch wird die Ausgabe der Schicht in einen 4-dimensionalen Tensor umgewandelt. Im Falle des Diskriminators wird die Ausgabe des letzen Convolutional-Layers (4-dimensional) abgeflacht und in eine 1-dimensionale Schicht mit einer Sigmoid-Aktivierungsfuntion gefüttert [5].

Um Mode-Collapse zu vermeiden, verwendet [5] Batch-Normalization-Layer. Dadurch wird das Training stabilisiert und Probleme wie *Internal-Covariate-Shifting* angegangen [4]. Vor allem wird dadurch aber auch verhindert, dass der Generator immer die gleichen Ausgaben erzeugt. Das Anwenden der Batch-Normalisierung in allen Schichten des Netzwerks führt jedoch zur Stichprobenoszillation und Instabilität des Modells. Aus diesem Grund wird auf Batch-Normalization in der Ausgabesschicht des Generators und

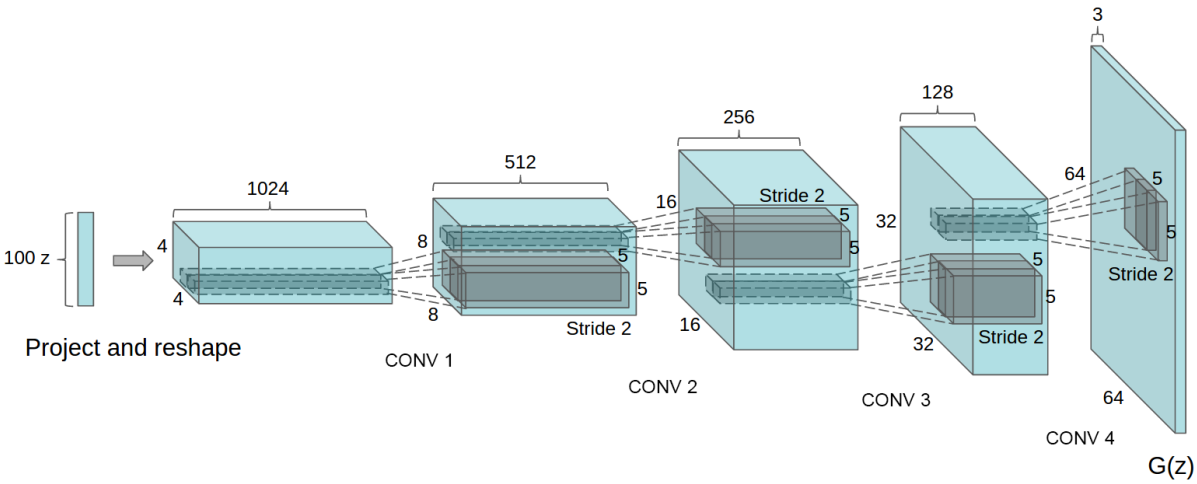

Abbildung 3.1: DCGAN-Architektur des Generators von [5]. Als Eingabe dient ein 100-dimensionaler Vektor, dessen Elemente zufällig gewählt werden. Dieser wird dann in den ersten Schichten umgeformt und durch vier Convolutional-Layer auf die Form 3×64×64 gebracht. Die Strides geben dabei den Vergrößerungsfaktor pro Convolution-Schicht an, während die Anzahl der Filter den Farbkanälen entsprechen.

in der Eingabeschicht des Diskriminators verzichtet.

Als letzte Beobachtung stellt [5] fest, dass das Hinzufügen von ReLU-Aktivierungsfunktionen in allen Schichten des Generators zu schnellerem Lernen und Abdeckung der Farbräume der Trainingsdistribution führt. In der Ausgabeschicht wird jedoch anstatt von ReLU-Aktivierung eine Tanh-Aktivierung verwendet. Innerhalb des Diskriminators werden schließlich Leaky-ReLU-Aktivierungen angewandt.

3.3 Wasserstein GAN

Anders als andere GAN-Varianten verwendet das Wasserstein-GAN (WGAN) die Wasserstein-Distanz anstelle der JS- oder KL-Divergenz, um die Gewichte von generativen neuronalen Netzen zu optimieren. Da sich die Berechnung über ein Infimum etwas schwierig gestaltet, formt [1] die Definition unter Berücksichtigung der Kontorovich-Rubinstein-Dualität um, sodass

$$W(P_r, P_\theta) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r}[f(x)] - \mathbb{E}_{x \sim P_\theta}[f(x)]$$

gilt, wobei das Supremum über alle 1-Lipschitz-Funktionen $f : X \to \mathbb{R}$ ist.

$$W(P_r, P_\theta) = \max_{\|f\|_L \leq 1} \mathbb{E}_{x \sim P_r} [f(x)] - \mathbb{E}_{x \sim P_\theta} [f(x)]$$

3.4 Wasserstein GAN mit Gradient Penality

4 Erstellen eines Datensatzes

4.1 Rahmenbedingungen

4.2 Verwendung von GANs

4.3 Durchführung von Experimenten mit unterschiedlichen GANs

4.4 Analyse der Ergebnisse aus den Experimenten

5 Bewegungserkennung

5.1 Ground-Truth

5.2 Background-Substraction

5.3 Erkennung von Geschwindigkeiten

5.4 Erkennung von Anomalien

5.5 Erkennung von Bewegungsarten

5.6 Vorhersage von Bewegungen

5.7 Architektur einer mobilen Anwendung

6 Fazit und Ausblick

Literatur

[1] Martin Arjovsky, Soumith Chintala und Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[2] Emad Barsoum, John Kender und Zicheng Liu. *HP-GAN: Probabilistic 3D human motion prediction via GAN*. 2017. arXiv: 1711.09561 [cs.CV].

[3] Ian J. Goodfellow u. a. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[4] Sergey Ioffe und Christian Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *Proceedings of the 32nd International Conference on Machine Learning*. Hrsg. von Francis Bach und David Blei. Bd. 37. Proceedings of Machine Learning Research. Lille, France: PMLR, Juli 2015, S. 448–456. URL: http://proceedings.mlr.press/v37/ioffe15.html.

[5] Alec Radford, Luke Metz und Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[6] Eitan Richardson und Yair Weiss. *On GANs and GMMs*. 2018. arXiv: 1805.12462 [cs.CV].

[7] Jost Tobias Springenberg u. a. *Striving for Simplicity: The All Convolutional Net*. 2015. arXiv: 1412.6806 [cs.LG].