Gesichtserstellung mittels GANs

Tobias Rohrer
Hochschule Darmstadt
Data Science (Master)
Email: sttorohr@stud.h-da.de

In der vorliegenden Arbeit wurde ein Deep Convolutional Generative Adversarial Network (DCGAN) zur Erstellung von Gesichtern erstellt. Das DCGAN wurde mit Hilfe des CelebA-Datensatzes trainiert. Für die Implementierung wurden die state-of-the-art TensorFlow 2 und TensorFlow Dataset APIs verwendet.

1 Einleitung

Generative Adversarial Networks (GANs) [1] sind laut einer der Gründerväter der Convolutional Neural Networks, Yann LeCun:

> ”the most interesting idea in the last 10 years in Machine Learning”

GANs sind generative Modelle, die es ermöglichen aus zufälligen Eingabewerten neue Daten zu erstellen, die den Trainingsdaten ähneln. DCGANs sind eine Unterkategorie von GANs, welche sich an Convolutional Neural Networks (CNNs) orientieren und eine tiefe Netzwerkarchitektur, sowie Faltungen implementieren. Heute werden GANs und DCGANs unter anderem verwendet, um die Auflösung von Bildern zu erhöhen, Bilder zu restaurieren oder auch um neue Trainingsdaten zum trainieren von tiefen neuronalen Netzen zu erstellen. Das Ziel der vorliegenden Arbeit ist die Implementierung eines DCGANs zur Erstellung von Bildern mit menschlichen Gesichtern. Als Vorbild für das Vorhaben diente die Website `www.thispersonoesnotexist.com`. Das in dieser Arbeit entworfene und implementierte DCGAN orientiert sich an den Architektur-Richtlinien aus [2].

2 Generative Adversarial Networks

Die Idee der GANs wurde 2014 in [1] präsentiert. Ein GAN besteht aus einem generativen Model G und einem diskriminierendem Model D. Hierbei ist D ein Klassifizierer mit der Aufgabe, echte Trainingsdaten X_{train} von den durch G erstellten Daten X_{gen} zu unterscheiden. Das Model G hat das Ziel, neue Daten (X_{gen}) zu erstellen, die nicht von D von echten Trainingsdaten unterschieden werden können. Das Training von D und G erfolgt simultan, wobei die Modelle sich gegenseitig versuchen auszuspielen.

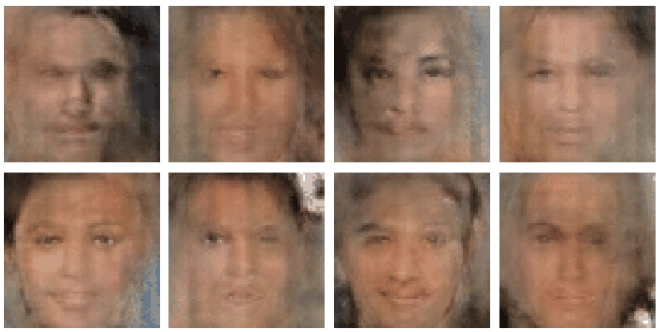

Fig. 1: Gesichter, die durch den Generator nach dem Training erstellt wurden.

3 Konzeption

Zum Trainieren des Diskriminators D wurde der Large-scale CelebFaces Attributes (CelebA) Datensatz verwendet [3]. CelebA enthält insgesamt 202.599 Bilder von 10.177 Identitäten. Um den Rechenaufwand gering zu halten, wurde eine zufällige Teilmenge von 50.000 Bildern verwendet. Die Daten durchlaufen vor dem Training mehrere Vorbereitungsschritte. Zum einen wird aus dem Zentrum des Bildes das Gesicht herausgeschnitten um unnötige Hintergrunddetails zu entfernen. Zum anderen werden alle Bilder auf eine einheitliche Größe von 64x64x3 Pixeln runterskaliert. Das Ergebnis eines Vorbereitungsschrittes wird in Abbildung 2 veranschaulicht. Die Architektur des GAN, bestehend aus G und D, ist in den Abbildungen 3 und 4 skizziert.

Fig. 2: Bild als Original (links) und nach der Vorverarbeitung (rechts)

3.1 Diskriminator

Der Diskriminator D besteht aus 2 Faltungsschichten, sowie einer voll vernetzen Schicht zur Klassifizierung. Die Faltungsschichten arbeiten mit 5x5 großen Faltungskernen und einer Schrittweite von 2. Wie in [2] als Teil der Richtlinie für GAN-Architekturen vorgeschlagen, werden somit die *Pooling-Schichten* durch die Schrittweite der Faltungskerne ersetzt. Als Eingabe erwartet der D einen Datensatz mit den Dimensionen 64x64x3. Die Ausgabe des Diskriminators ist ein Skalar $o \in [0;1]$, wobei $o \leq 0.5$ einer Klassifikation als unecht, also vom GAN erzeugt, und $o > 0.5$ einer Klassifikation als echt, also vom Trainingsdatensatz, entspricht.

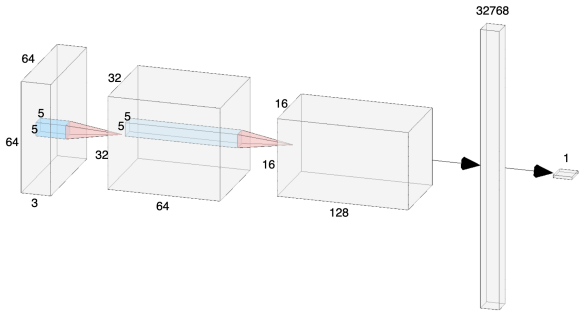

Fig. 3: Architektur des Diskriminators D

3.2 Generator

Die Eingabeschicht des Generator G mit der Dimension 100x1 nimmt einen Vektor X_{random} mit zufälligen Werten, die aus einer Standardnormalverteilung gezogen wurden, entgegen. Über eine voll vernetzte Schicht folgen drei Schichten, die eine transponierte Faltung implementieren. Analog zu D werden Faltungskerne der Größe 5x5 eingesetzt. Die Ausgabe von G ist ein Bild mit den Dimensionen 64x64x3.

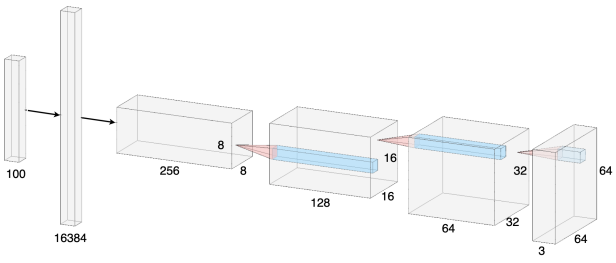

Fig. 4: Architektur des Generators G

4 Implementierungsdetails

Das entworfene Konzept wurde mit der *TensorFlow*-API [4] in Python implementiert. Unterstützend wurde die *TensorFlow Dataset*-API eingesetzt um die Trainingsdaten einzulesen und um die in Abschnitt 3 beschriebenen Vorbereitungsschritte umzusetzen. Das entworfene Netzwerk wurde in einer Online-Entwicklungsumgebung auf kaggle[1] implementiert. Kaggle bietet die Möglichkeit, die Berechnungen mit der Hilfe von GPUs und Tensor Processing Units (TPUs) zu beschleunigen. Das GAN wurde innerhalb von 150 Epochen mit einer Batch Size von 128 trainiert. Als *Optimizer* wurde das Adam-Verfahren [5] gewählt.

5 Ergebnisse

Die Ergebnisse nach 150 Epochen können in Abbildung 1 gesehen werden. Der Generator hat offensichtlich während des Trainings gelernt, wie menschliche Gesichter aussehen und ist in der Lage, anhand eines Zufallsvektors X_{random}, neue Bilder von Gesichtern zu generieren.

6 Fazit

In der vorliegenden Arbeit konnte mit relativ geringem zeitlichen Aufwand ein GAN zur Erstellung von Bildern mit menschlichen Gesichtern implementiert werden. Eine Verbesserung des Ergebnisses durch eine ausgefeiltere Wahl der Hyperparameter ist noch offen.

References

[1] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2014. Generative adversarial networks.

[2] Radford, A., Metz, L., and Chintala, S., 2015. Unsupervised representation learning with deep convolutional generative adversarial networks.

[3] Liu, Z., Luo, P., Wang, X., and Tang, X., 2015. "Deep learning face attributes in the wild". In Proceedings of International Conference on Computer Vision (ICCV).

[4] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., and Others, 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[5] Kingma, D. P., and Ba, J., 2014. Adam: A method for stochastic optimization. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[1] www.kaggle.com