Embedded ML: GAN - Styletransformation von Bildern zu van Gogh-Kunstwerken

Adrian Kaßmann
HAW Hamburg

February 28, 2021

Abstract

Mit der Hilfe eines CycleGAN werden Bilder einer Webcam in Van Gogh Gemälde transformiert. Dafür werden 3 verschiedene Generator Architekturen (UNet, ResNet und ResUNet) miteinander verglichen und der Aufbau der Modelle genauer erläutert. Die Modelle mit UNet Architektur stellen sich dabei zielführend heraus, da die Ergebnisse schärfer sind und eine kürzere Interferenz-Zeit haben. Die Resultate der UNet- und ResUNet-Architekturen sind ähnlich gut und unterschieden sich nicht großartig voneinander. Für stärkere Abstraktionen von der Wahl eines CycleGANs abzuraten, da ein CycleGAN nur so weit verändert, wie es auch wieder zurück wandeln kann.

Einführung

Mit Machine Learning können nicht nur technische Probleme gelöst werden, sondern Machine Learning kann auch dazu eingesetzt werden, um kreative Probleme zu lösen. Generative Adversarial Networks (GAN) sind spezielle Netzwerke, die es ermögliche Bilder zu erzeugen oder zu verändern. Anwendungsgebiete können zum Beispiel Erstellung von Trainingsdaten sein, das Einfärben von Zeichnungen, Umwandlung von Zeichnungen in realistische Bilder [1], sommerliche Landschaften in winterliche Landschaften oder auch Bilder in Gemälde von spezifischen Künstlern [2].

Die Umwandlung von Bildern zu Gemälden wird auch Styletransfer genannt, denn der Stil von Künstlern wie Vincent van Gogh wird auf andere Bilder übertragen.

GANs bestehen aus zwei Modellen, einem Generator und einem Diskriminator. Der Generator erstellt bzw. transformiert die Bilder und der Diskriminator unterscheidet, ob das Bild real ist oder vom Generator erstellt wurde. Diese beiden Modelle arbeiten gegeneinander, denn Ziel des Generators ist es Bilder zu erzeugen, die der Diskriminator nicht von realen Bildern unterscheiden kann [3].

CycleGAN Eine spezielle Form von GANs sind CycleGANs. CycleGANs bieten eine effektivere Möglichkeit Quell-Bilder zu Ziel-Bildern transformieren.

Für Jede Kategorie (Reale Bilder/ VanGogh-Bilder) gibt es einen Diskriminator, der entscheidet ob das Bild zu dieser Kategorie gehört. Außerdem gibt es 2 Generatoren, die Bilder einer Kategorie in die andere Kategorie transformiert. Also ein Generator für Photo →VanGogh und ein Generator für Van Gogh →Photo.

Der Vorteil liegt darin, dass ein Bild doppelt transformiert werden kann. Also von Photo zu Van Gogh und zurück zu einem Photo. Das Original und das Fake-Photo können verglichen werden und es kann ein zusätzlicher Loss ermittelt werden, abhängig davon wie stark sich die Bilder unterscheiden. Das Ziel des Losses ist es, die Struktur des Bildes besser zu beibehalten [2]. Siehe Figure 1.

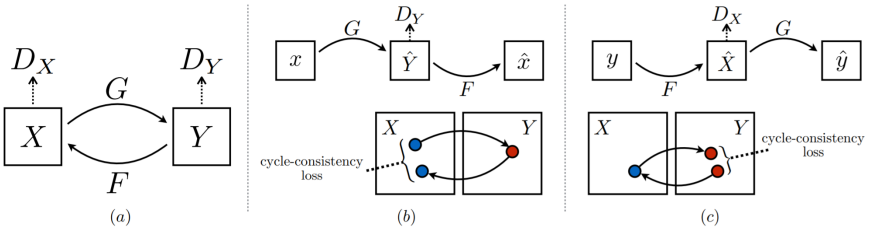


Figure 1: CycleGAN-Modell: Doppel-Transformation. Grafik aus dem CycleGan Paper [2].

PatchGAN Das Ergebnis eines normalen Diskriminators kann mit einem simplen Wert zwischen 0 und 1 dargestellt werden, welcher die Zugehörigkeit zur Kategorie bewertet. Dieser Wert zählt für das gesamte Bild.

In einem PatchGAN ist das Ergebnis des Diskriminators eine 2d-Matrix. Das Eingangs-Bild wird unterteilt und jedes Teil (Patch) wird einzeln bewertet. Das Ergebnis ist des Patches ist ein Wert in der Matrix. Durch die spezifischere Bewertung ergibt sich eine verbesserte Optimierung der Generator-Parameter. Diese Diskriminatoren wurden auch in [1] und in [2] verwendet.

Residual Block Residual Blöcke werden oft in tiefen Netzen verwendet. Mit Hilfe eines Bypass können Informationen einige Layer überspringen, siehe Figure 2. Dadurch ist es möglich, dass das Netz selbst entscheiden kann, welche Informationen von welchem Layer abändert werden. Außerdem wird das "Vanishing gradient"-Problem in tieferen Netzen reduziert. Je tiefer ein Netzwerk ist, desto kleiner ist der Loss, der durch die Backpropagation in die Netzwerk-Ebenen zurückgerechnet wird. Bei einem kleinen Loss sind die Gewichtsänderungen der Layer kleiner und das Netzwerk lernt weniger bzw. nichts mehr. [4]

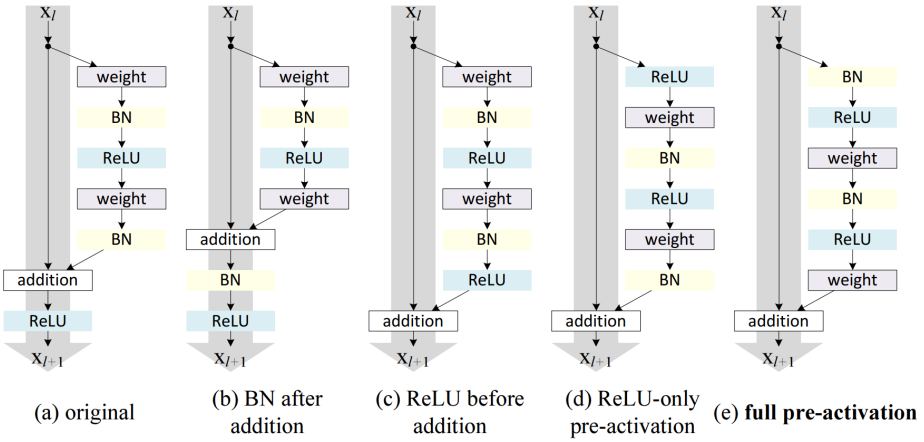


Figure 2: Unterschiedliche Anordnung der Layer in Residual Blöcken. Die Variante full-pre-activation bietet die beste Performance im vergleich zu den anderen Möglichkeiten [5].

Instance Normalization Bei dem Batch-Normalization-Layer wird der Input, abhängig von selbstgewählten Gewichten, standardisiert und vom Mittelwert befreit. Dabei wird der Mittelwert und die Varianz vom gesamten Batch genommen.

Bei Instance Normalization wird das gleiche durchgeführt, mit einer kleinen Abweichung, die Berechnungen finden nicht auf den gesamten Batch statt, sondern nur auf den einzelnen Bildern.

Bei einem Input von 32x32x256 wird bei der Instance Normalization jedes 32x32 Bild unabhängig von den anderen normalisiert, bei Batch Normalization werden alle 256 Bilder zusammen normalisiert. Für Styletransfer-Netzwerke wird die Instance Normalization empfohlen. [6]

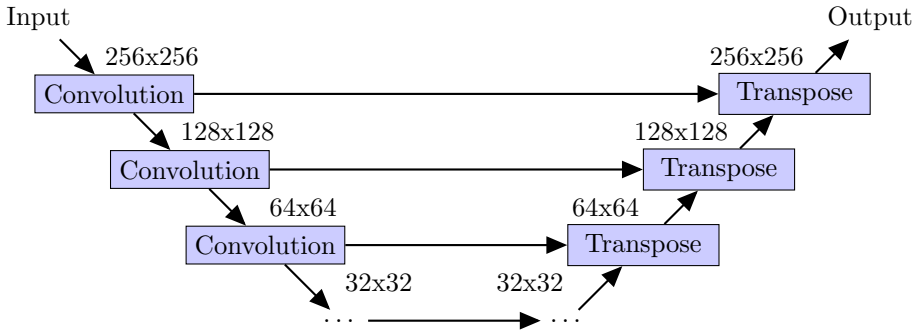


Figure 3: UNet Architektur. Auf der linken Seite werden die Bilder runter skaliert, rechts hoch skaliert. Zusätzlich werden auf jeder Ebene die Bilder mit "Concatinate" zusammengefügt.

Modell-Architektur

Diskriminator

Das Modell des Diskriminators ist aus dem Tensorflow Beispiel für das Pix2Pix-Modell [7] übernommen. Der Pix2Pix-Diskriminator besteht hauptsächlich aus fünf Convolutional-Layern und umfasst insgesamt 2.765.569 Parameter. Es ist ein Patch-Diskriminator und der Output ist eine 30x30x1 Matrix.

Generator

Für die Architektur des Generators wurden drei verschiedene Modelle miteinander verglichen.

U-Net-Modell Das erste Modell stammt auch aus dem Tensorflow Beispiel Pix2Pix [7]. Der Aufbau des Modells entspricht einer U-Net Architektur, welches in Figure 3 dargestellt ist und in [8] vorgestellt wurde.

Statt MaxPooling zu verwenden, um die Größe der Bilder zu verringern, wird ein Convolutional-Layer mit einem Stride von 2 eingesetzt. Hinter jedem Convolution- und Transpose-Layern wird mit Instance-Norm normalisiert und leakyReLU als Aktivierungsfunktion genutzt.

Das U-Net Modell folgt einem Encoder- Decoder Modell. Die Struktur des Bildes bleibt gut erhalten, da es mit den Concatinate-Layern Abkürzungen gibt und damit nur wenige Layer durchlaufen. Durch das Reduzieren der Größe werden Features extrahiert. Die Features werden mit Concatinate wieder zusammen zu einem Bild gefügt.

ResNet Das zweite Modell basiert auf einer Sequenz von acht Residual Blöcken, siehe Figure 4. Das Modell ist in Figure 4 dargestellt und besitzt 46.699.779

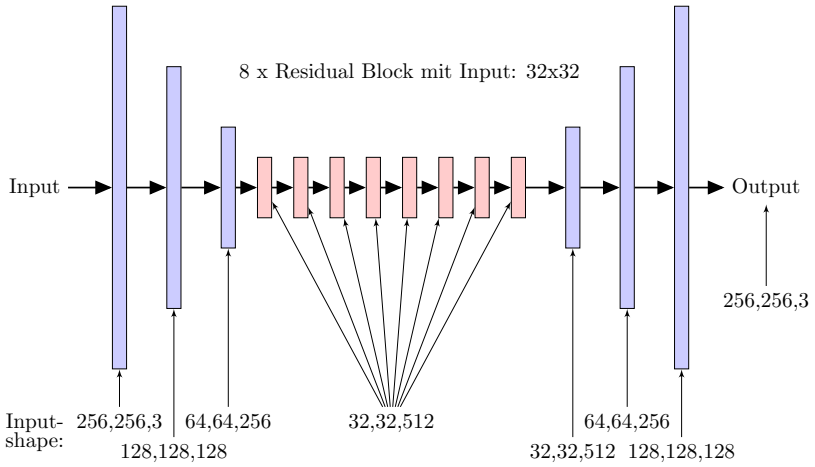


Figure 4: ResNet Architektur. Reduktion der Größe mit Convolutional-Layern und Stride=2, acht Residual-Blöcke als Hauptnetz und am Ende mit Transpose-Layern auf Originalgröße skalieren.

Parameter. Jeder dieser Residual Blöcke ist nach dem "full pre-activation"-Modell [5] aufgebaut (Siehe Figure 2 (e)) und enthält Instance Normalization, LeakyReLU und Convolution-Layer. Durch full pre-activation werden die Input-Daten des Blockes nur mit dem Add-Layer verändert. Es wird, anders wie beim Original, keine Aktivierungsfunktion nach dem Add genutzt.

Zum Runter- und Hochskalieren der Bilder werden die gleichen Layer genutzt, wie in der UNet-Architektur. Also Convolution- und Transpose-Layer, gefolgt von Instance-Normalization und LeakyReLU.

Res-U-Net Der dritte Ansatz besteht aus einer Kombination aus den vorherigen Modellen zu einem Res-U-Net. Zwischen dem Runter- und Hochskalieren der Layer befinden sich Residual Blöcke. In Figure 5 ist die Architektur schematisch dargestellt.

Das Modell hat 54.034.435 trainierbare Parameter und hat damit eine vergleichbare Größe an trainierbaren Parameter wie die anderen beiden Netze.

Durch die Kombination sollen die Vorteile beider Architekturen dargestellt werden. In folgenden Paper [9] wird diese Kombination genauer vorgestellt.

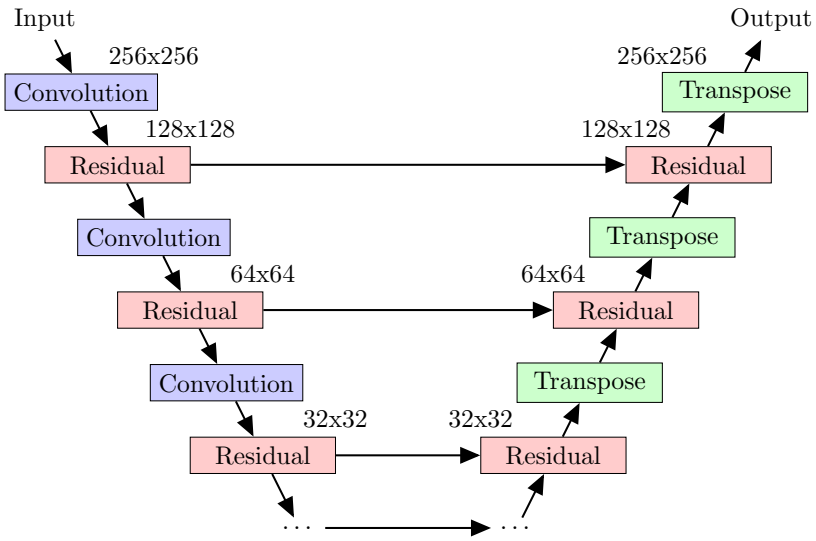


Figure 5: ResUNet Architektur. Ähnlich zu der Architektur aus Figure 3. Zusätzliche Einbindung von Residual Blöcken

(a) (b) (c) (d)

Figure 6: 4 Beispiel Bilder aus dem Datensatz, 2 reale Bilder (a, b) und 2 Gemälde von Van Gogh (c, d).

Training des Netzes

Daten

Als Vorlage zum Trainieren der Modelle wurde ein Tutorial von Google Colab [10] genommen. In dem Tutorial sind schon die Berechnung des Losses und der Gradienten für CycleGANs vordefiniert.

Der Datensatz für zum Trainieren stammt von Tensorflow und heißt "Vangogh2photo" [11]. Die Bilder sind schon in Test- und Trainigsdaten unterteilt, in Table 1 ist die Aufteilung der Bilder dargestellt. Jedes Bild hat eine feste Größe: 256x256x3. In Figure 6 sind Beispiel-Bilder aus dem Datensatz abgebildet.

	Van Gogh	Reale Bilder
Training	400	6.287
Test	400	751

Table 1: Anzahl der Bilder im Datensatz "vangogh2photo".

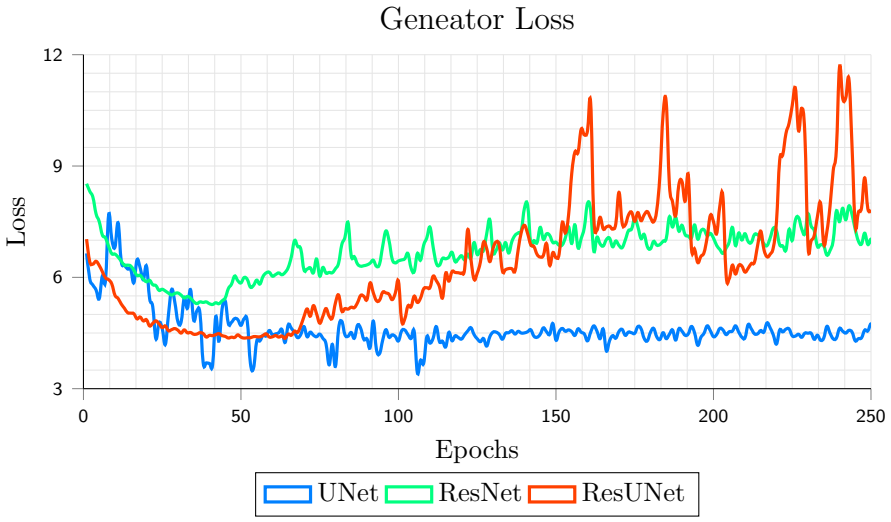
Für ein besseres Resultat werden die Daten noch vor verarbeitet. Dazu werden die Bilder auf einen Bereich von -1.0 bis 1.0 normalisiert, um dem Vanishing-Gradients-Problem entgegen zu wirken. Um die Diversität des Datensatzes zu erhöhen, wird das Bild zufällig zugeschnitten und zufällig gespiegelt.

Trainingsverlauf

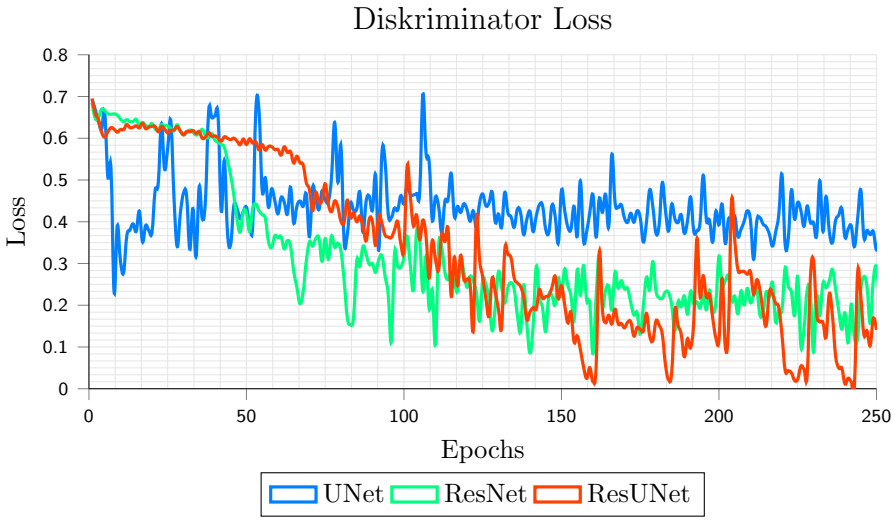
In den beiden Diagrammen in Figure 7 sieht man den Trainingsverlauf der Generatoren und Diskriminatoren für die Transformation: Photo →Van-Gogh.

Was zunächst an den Diagrammen auffällig ist, dass der Loss der Generatoren zunimmt. Dabei ist es kein Overfitting, sondern ein typischer Verlauf für GANs, denn Generator und Diskriminator arbeiten gegeneinander.

Wenn der Diskriminator viele Bilder falsch klassifiziert, steigt der Loss der Diskriminators und es sinkt der Loss des Generators. Gleiches gilt auch umgekehrt, klassifiziert der Diskriminator gut, sinkt der Loss des Diskriminators und es steigt der Loss des Generators. Das gegeneinander Arbeiten ist vor allem ab



(a)



(b)

Figure 7: (a) zeigt den Loss der Generatoren, für die Weg: Photos →Van-Gogh. In (b) ist der Loss der Diskriminatoren abgebildet, die zwischen echten und falschen VanGogh Gemälden unterscheiden.

Epoche 150 beim ResUNet zu erkennen. Dort reißt der Loss des Generators öfters nach oben aus und zeitgleich sinkt der Loss des Diskriminators stark ab.

Als Beispiel für die starken Schwankungen: Findet der Diskriminator ein besonderes Merkmal, an dem er die Bilder gut unterscheiden kann, schießt der Loss des Generators hoch, bis dieser seine Bilder soweit abändert, dass der Diskriminator nicht mehr sein Merkmal zur Unterscheidung nutzen kann.

In den Verläufen ist auch ein Unterschied zwischen den drei Modellen zu erkennen. Da bei allen Modellen der gleiche Diskriminator verwendet wurde, liegen die Unterschiede bei den Generatoren. Beim UNet-Generator gibt es nur noch wenig Änderungen ab Epoche 100. Der Loss pendelt um 4,5 herum. Der Loss des Diskriminators ändert sich auch nicht mehr stark und bleibt um ein Wert von 0.4.

Bei den anderen beiden Modellen steigt der Loss des Generators ab Epoche 70 wieder an. Zuvor ist der Loss der Generatoren und Diskriminatoren auffällig stabil. Erst bei Anstieg des Losses fängt es an stärker zu schwanken.

Ergebnis

In Figure 8 und in Figure 9 sind Ausgabebilder der Modelle nebeneinander dargestellt. Der Output des ResNet-Modells ist im Vergleich zu den anderen beiden Modellen unschärfer.

Zwischen den Resultaten des UNet-Modells und des ResUNet-Modells gibt es keine großen Unterschiede. Die Farbpalette der Modelle unterscheidet sich etwas, aber das sollte Unabhängig vom Modell sein, denn Würde man das gleiche Modell erneut trainieren, wird voraussichtlich auch ein etwas anderes Ergebnis resultieren. Im Output des ResUNets ist oben links ein schwarzes Artefakt, welches in fast jedem Ergebnis zusehen ist. Es könnte eine Art von Mode Collapse sein, bei dem der Gradient des Patch-Diskriminator nicht aus dem lokalen Minimum heraus findet und der Generator einen extrem Wert (hier Schwarz) an nimmt.

Die unteren Zeilen in Figure 8 und Figure 9 zeigen das Ergebnis der CycleGANs nach einem vollständigen Cycle. Also einer Transformation von Real →VanGogh und dann VanGogh →Real. Die Ergebnisse sind gut mit dem Originalbild vergleichbar, da ein reales Bild des CycleGAN angestrebt wird. Der Output des ResNets ist unscharf, aber ähnelt dem Original. UNet und ResUNet sind auch hier relativ ähnlich. Die Details vor allem im Feld vor dem Zug sind bei dem ResUNet deutlicher. Dafür hat sich die Farbe des Zugs beim ResUNet von rot in orange/gelb geändert.

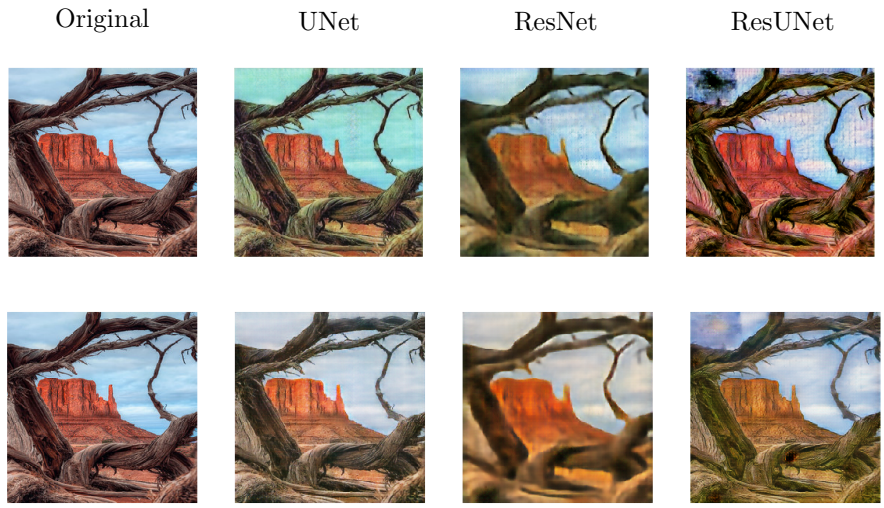


Figure 8: Oben Bild in Gemälde transformiert (Bild →VanGogh). Unten zurück in Original (VanGogh →Bild) transformiert.

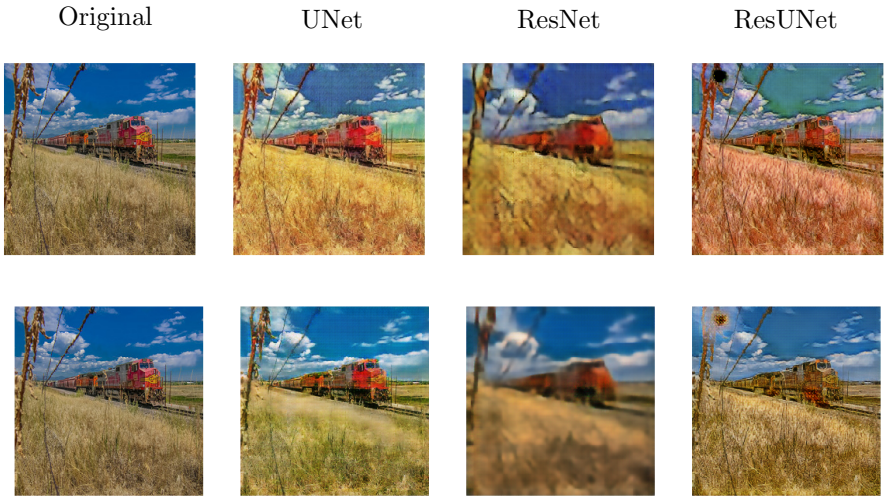


Figure 9: Oben Bild in Gemälde transformiert (Bild →VanGogh). Unten zurück in Original (VanGogh →Bild) transformiert.

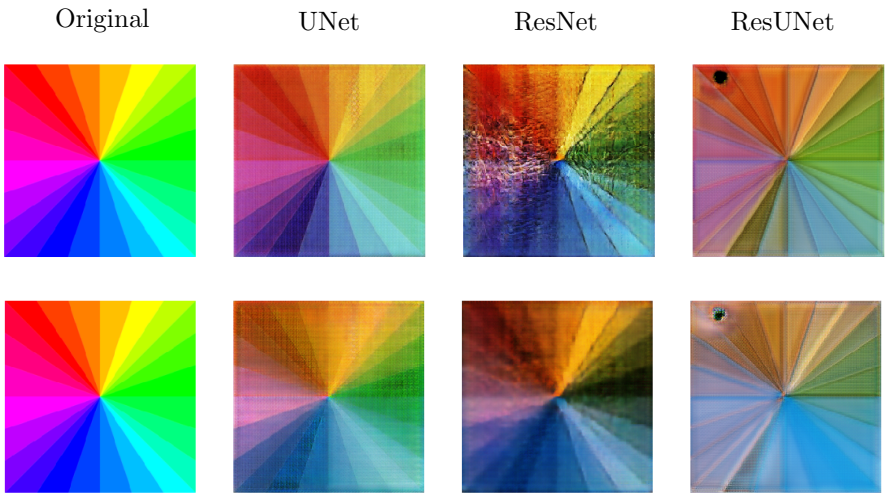


Figure 10: Oben Farbtest in Gemälde transformiert (Bild →VanGogh). Unten zurück in Original (VanGogh →Bild) transformiert.

In Figure 10 ist die Farbänderung der Transformation zu erkennen. Die Farben aus der oberen Reihe müssten typische Farben für Gemälde von Van Gogh sein. In der unteren Zeile ist zu erkennen, dass nicht alle Farben wiederhergestellt werden können und Informationen verloren gehen. Die Farben aus dem UNet-Modell sind nah an den Farben, die Van Gogh genutzt hatte. In Figure 11 ist zu sehen, dass das UNet-Netz die Farben minimal ändert. Im Vergleich zu dem ResUNet sind starke Unterschiede zu erkennen.

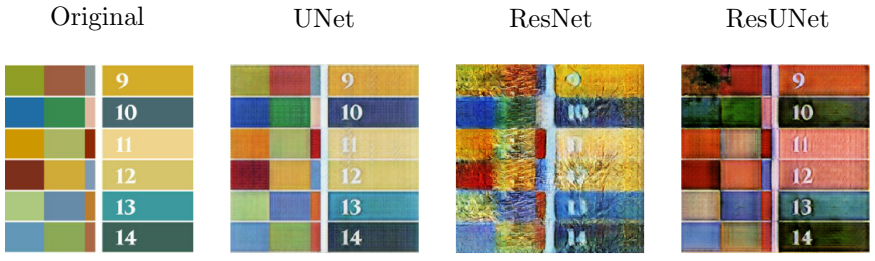


Figure 11: Das Original zeigt typische Farben von Van Gogh [12].

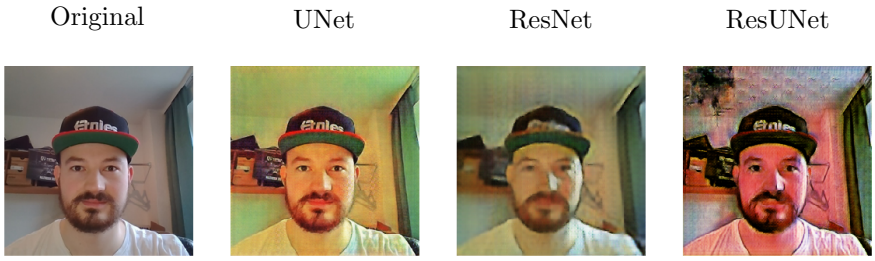


Figure 12: Input Bild von der Webcam nach VanGogh transformiert.

Webcam

Die Bilder der Webcam werden über OpenCV aufgenommen und vor verarbeitet. Das Modell berechnet die Transformation des 256x256 Bildes und es wird über OpenCV dargestellt. In Figure 12 ist ein Beispiel Bild dargestellt.

Die Zeit für die Verarbeitung und Transformation des Bildes wird in Table 2 gezeigt. Das Skript läuft nur auf einer CPU. Das ResNet unterscheidet sich mit ca. einer Sekunde stark von den anderen beiden Modellen.

Der Unterschied lässt sich dadurch begründen, dass im Vergleich viele Berechnungen mit Layern der Größe von 32x32 stattfinden. Die Runter- und Hochskalierung ist ähnlich in allen Netzen. In den beiden Modellen mit UNet-Architektur werden die Layer noch wesentlich kleiner als 32x32 skaliert und dadurch ist eine schnellere Berechnung bei einer größeren Parameteranzahl möglich.

Modell	Durchschnittliche Zeit [sec]	Frames per Second
UNet	0.245	4.08
ResNet	1.285	0.78
ResUNet	0.317	3.15

Table 2: Zeit, zwischen Bildaufnahme und Wiedergabe.

Fazit

Im Vergleich zwischen den Modellen hat sich die UNet Architektur als vorteilhaft herausgestellt. Die Ergebnisse des ResNet-Modells unscharf und die Interferenz Zeit ist im Vergleich relativ hoch. Acht residual Blöcke sind für ein ResNet noch recht gering, das Modell könnte man ggf. verbessern, indem die Größe der einzelnen Blöcke reduziert wird und dafür die Tiefe des Netzes erhöht wird.

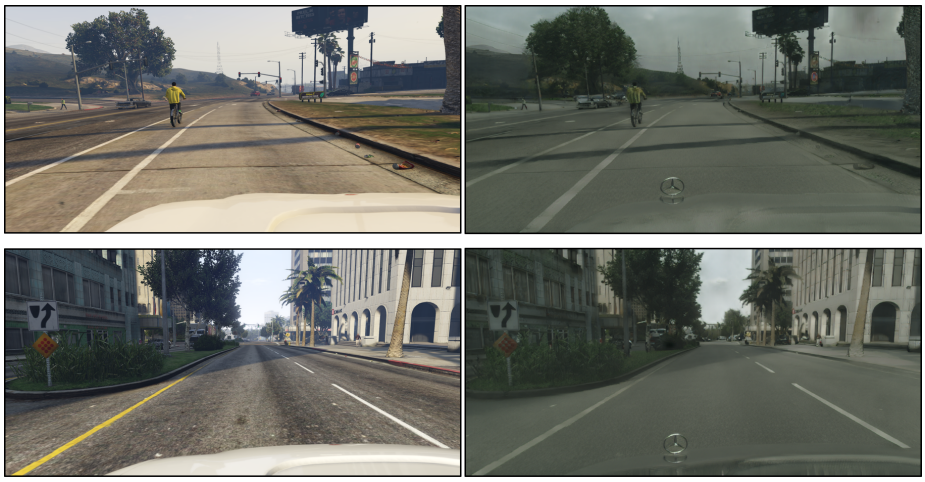
Zwischen dem UNet-Modell und dem ResUNet-Modell gibt es im Resultat nur geringe Unterschiede. Das ResUNet-Modell scheint minimal detaillierter zu sein, dafür kann das UNet-Modell die Farben besser darstellen. Beides kann auch vom Training abhängen, da es zwischen den Epochen starke Schwankungen in der Qualität der Transformation gab. Nach 40 trainierten Epochen sind auch schon annehmbare Resultate zu erkennen, mit weiterem Training hat die Qualität aber nicht abgenommen, daher wurde bis 250 Epochen trainiert.

Was vermutlich auch noch stark das Ergebnis des Trainings verbessern kann, wäre ein größerer Datensatz, da im hier verwendeten Datensatz nur 400 Trainings- und 400 Test-Bilder von Van Gogh enthalten waren.

Für die Anwendung auf Live-Bildern mit einer Webcam, würde eine GPU vermutlich die Performance deutlich verbessern und ein flüssigeres Ergebnis liefern, sodass die Performance mehr als 4 fps möglich sein sollte.

Es ist noch einfach die Ergebnisse von echten Van Gogh Gemälden zu unterscheiden, da die Van Gogh-Gemälde eine starke Abstraktion der Realität abbilden. Das CycleGAN abstrahiert die Bilder nur soweit, wie es diese wieder zurück wandeln kann und schränkt sich dadurch selbst ein. Für stärkere und künstlerische Abstraktionen empfehle ich "Exploring the structure of a real-time, arbitrary neural artistic stylization network" [13]. Bei diesem Modell werden 2 Input-Bilder kombiniert, ein Bild für den Inhalt / Struktur und ein Bild für den Style.

Es besteht aber viel Potential für Erstellung von Trainingsdaten in GANs. In Figure 13 wird die Umwandlung aus einer Spielwelt in die reale Welt gezeigt. Mit einem CycleGAN als Input, könnte autonomes Fahren in einer Spielwelt trainiert und das Modell ohne CycleGAN in die reale Welt getestet werden. Eine Voraussetzung dafür, wäre aber, dass die Verzögerung durch das GAN reduziert werden müsste.



GTA → Cityscapes

Figure 13: Beispiel für die Umwandlung von Bildern aus einem Computerspiel (Grand Theft Auto 5) in eine reale Umgebungen [14].

References

[1] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. *CoRR*, abs/1611.07004, 2016.

[2] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.

[3] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27, pages 2672–2680. Curran Associates, Inc., 2014.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[6] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization, 2017.

[7] Tensorflow examples: Pix2pix. `https://github.com/tensorflow/examples/blob/master/tensorflow_examples/models/pix2pix/pix2pix.py`.

[8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[9] Foivos I. Diakogiannis, François Waldner, Peter Caccetta, and Chen Wu. Resunet-a: a deep learning framework for semantic segmentation of remotely sensed data. *CoRR*, abs/1904.00592, 2019.

[10] Google colab tutorial: Cyclegan. `https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/generative/cyclegan.ipynb`.

[11] Tensorflow datasets: vangogh2photo. `https://www.tensorflow.org/datasets/catalog/cycle_gan#cycle_ganvangogh2photo`.

[12] Vincent van gogh – color theory. `https://ipoxstudios.com/vincent-van-gogh-color-theory/`.

[13] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. Exploring the structure of a real-time, arbitrary neural artistic stylization network. *CoRR*, abs/1705.06830, 2017.

[14] Cyclegan project page. `https://junyanz.github.io/CycleGAN/`.