Hochschule Augsburg
University of Applied Sciences

Hausarbeit

Bildgenerierung mit neuronalen Netzen

Temkeng Thibaut

Datum der Abgabe: 28. Juni 2020

Betreuer: Prof. Dr. Thomas Rist

Fakultät für Künstliche Intelligenz

Inhaltsverzeichnis

1 Einleitung

Die neuronalen Netze (NN) haben vor paar Jahren mehrere Bereiche insbesondere den Bereich der Bildverarbeitung revolutioniert. Die Methoden mit NNs haben sogar viele andere Methoden einfach obsolet gemacht. Die NNs scheinen besser zu funktionieren, wenn es viele Daten zum Training gibt. Aber es ist nicht immer möglich ausreichend Daten zu sammeln. Um diesen Datenmangel zu beheben, werden synthetische Daten generiert.In dieser Arbeit werden zwei spezielle Arten künstlicher neuronaler Netze zur Bildgenerierung erläutert: Autoencoder 3 und Generativ Adversarial Network (GAN) 4.

2 Datensatz

Um unsere NNs zu trainieren und zu evaluieren, den MNIST-Datensatz1[1] und den Smile-Datensatz2 verwendet. MNIST-Datensatz besteht aus $(60.000, 28, 28)$ Bildern zum Training und $(10.000, 28, 28)$ Bildern zum Testen während Smile-Datensatz nur aus $(400, 28, 28)$ zum Training und $(17, 28, 28)$ zum Testen besteht.

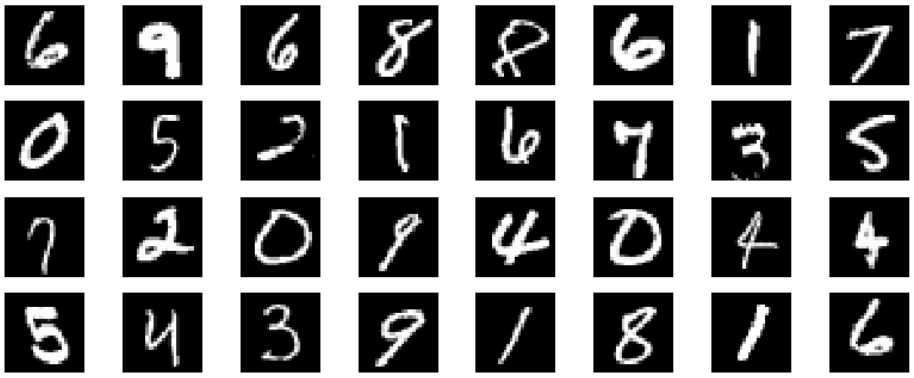


Abbildung 1: MNIST-Datensatz

3 Autoencoder

Ein *Autoencoder* ist eine Art künstliches neuronales Netzwerk, das dazu dient, effiziente Datencodierungen unbeaufsichtigt zu lernen[2]. Ein Autoencoder besteht hauptsächlich aus folgenden Teilen:

1. Der *Encoder*, der lernt, wie man die Eingabedimensionen reduzieren und die Eingabedaten in eine kodierte Darstellung komprimieren kann.

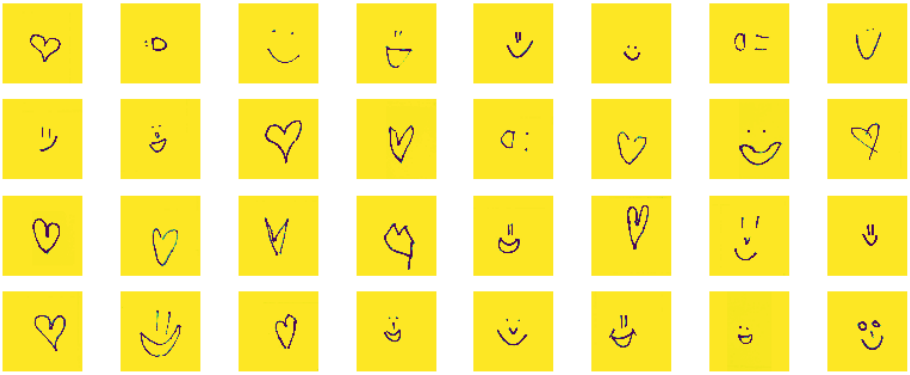


Abbildung 2: Smile Datensatz

2. Der *Decoder*,der Decoder lernt, wie er die Daten aus der kodierten Darstellung rekonstruieren kann, um der ursprünglichen Eingabe so nahe wie möglich zu kommen.

3. Die Verlustfunktion auch Rekonstruktionsverlust genannt, mit der es gemessen wird, wie gut der Decoder arbeitet und wie nahe der Ausgang am ursprünglichen Eingang liegt.

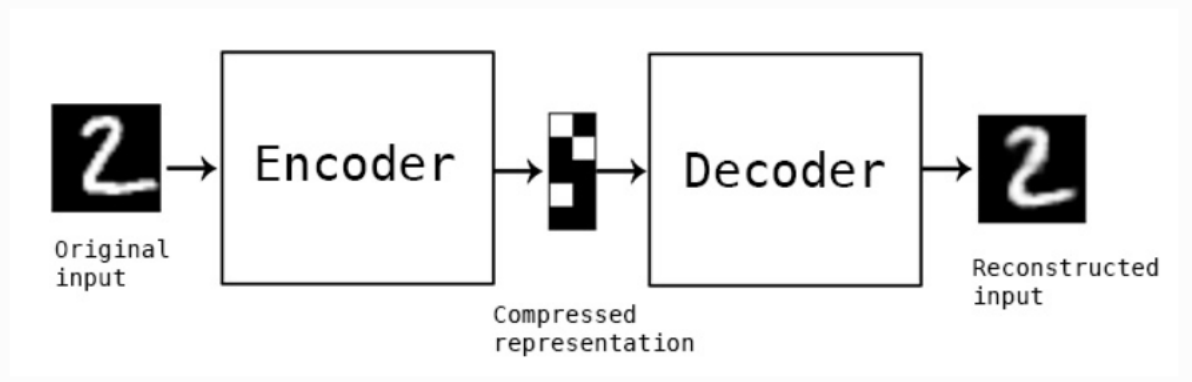


Abbildung 3: Autoencoder Architektur source

Der Autoencoder bekommt Daten als Eingabe und er versucht, dieselbe bzw. eine sehr ähnliche Eingabe zu rekonstruieren. siehe Abbildung 3

3.1 Einfache Autoencoder

Ein einfachste Autoencoder besteht nur aus drei Schichten: Eingabeschicht, versteckte Schicht und eine Ausgabeschicht. Siehe Abbildung 4 (Links). Nach 100 Epochen von

Training des Autoencoder 4a auf MNIST-Datensatz erhalten wir 4b (Rechts), wobei die Eingabebilder oben sind und die generierten Bilder unter sind. Die generierten Bilder entsprechen leider nicht den Originalen, aber sie sind schon erkennbar. Wenn man jedoch

```
Model: "Simple_Autoencoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input (InputLayer)           [(None, 784)]             0
_________________________________________________________________
encoder_layer (Dense)        (None, 32)                25120
_________________________________________________________________
decoder_layer (Dense)        (None, 784)               25872
=================================================================
Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0
```

(a) Einfaches Autoencoder-Modell

(b) Originale und generierte Bilder aus MNIST-Datensatz

Abbildung 4: Einfache Autoencoder

derselbe Autoencoder zum Trainieren von Smile-Datensatz verwendet, erhalten wir den linken Teil der Abbildung 5 nach 4.000 Epochen. Obwohl er zu viel und zu lange trainiert wurde, kann er den Input nicht rekonstruieren und dass sollte an folgenden liegen:

1. Der Smile-Datensatz ist zu klein.siehe 2
2. Jede Sample aus Smile-Datensatz enthält zu viel „unnötige“ Informationen, und zwar den ganzen weißen Bereich, der eigentlich nur aus Pixeln mit 255 als Werte bestehen. Diese Werte haben auch einen großen Einfluss, denn sie sind wegen ihrer großen Anzahl nicht einfach vorherzusagen.

Um diese Problem zu lösen, haben wir alle 255 Werte in 0 und alle andere Werte in 1 umgewandelt. Im rechten Teil der Abbildung 5 kann eine deutliche Verbesserung wahrgenommen werden, obwohl wir nur 700 Epochen durchgelaufen sind.

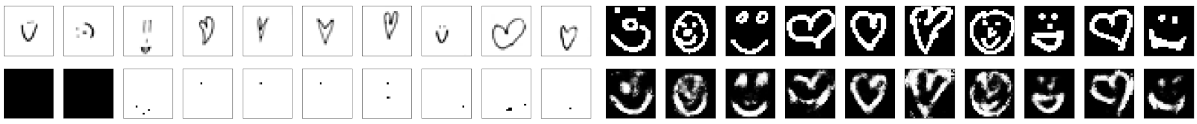


Abbildung 5: Smile mit einfache Autoencoder

3.2 Deep Autoencoder

Je tiefer sind die NNs, desto besser sind sie in der Lage, komplexere Daten zu verarbeiten. Eine Erweiterung des einfachen Autoencoders3.1 ist der tiefe (Deep) Autoencoder (DAE). Der einzige Unterschied zwischen Einfachen und tiefen Autoencoder besteht in der Anzahl der verborgenen Schichten. Die zusätzlichen versteckten Schichten ermöglichen es dem Autoencoder, mathematisch komplexere zugrundeliegende Muster in den Daten zu lernen. Die erste Schicht des DAEs kann Merkmale erster Ordnung in der

Rohdateneingabe lernen (wie z.B. Kanten in einem Bild). Die zweite Schicht kann Merkmale zweiter Ordnung lernen, die Mustern im Erscheinungsbild von Merkmalen erster Ordnung entsprechen (z.B. im Hinblick darauf, welche Kanten dazu neigen, zusammen aufzutreten - z.B. zur Bildung von Kontur- oder Eckendetektoren). Tiefere Schichten des DAEs neigen dazu, sogar Merkmale höherer Ordnung zu lernen.

Model: "Deep_autoencoder"

Layer (type)	Output Shape	Param #
input_15 (InputLayer)	[(None, 784)]	0
dense_14 (Dense)	(None, 128)	100480
dense_15 (Dense)	(None, 64)	8256
dense_16 (Dense)	(None, 32)	2080
dense_17 (Dense)	(None, 64)	2112
dense_18 (Dense)	(None, 128)	8320
dense_19 (Dense)	(None, 784)	101136

Total params: 222,384
Trainable params: 222,384
Non-trainable params: 0

(a) Deep autoencoder model

(b) Original und generierte Bilder (MNIST-Datensatz)

(c) Original und generierte Bilder (Smile-2-Datensatz).

Abbildung 6: Deep Autoencoder

Ein Beispiel für ein tiefes Autoencoder-Modell ist in der Abbildung 6a zu sehen. Die Anzahl von Parameter von Deep-Autoencoder im Vergleich zu einfachen Autoencoder ist enorm (50.992 gegen 222.384), was eine noch länger Trainingszeit verlangt. Wenn man aber die Abbildungen 6b, 6c und 4b, 5 vergleicht, stellt man schnell fest, die Deep Autoencoder-Modelle ihre Eingaben besser rekonstruieren können.

3.3 Convolution Autoencoder

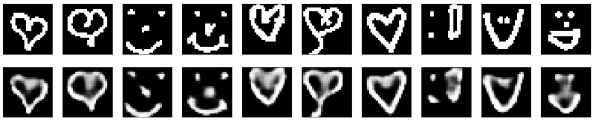
Die oben beschriebenen Autoencoder berücksichtigen nicht die Tatsache, dass ein Signal als Summe anderer Signale gesehen werden kann und versuchen sich auch die Position von Merkmalen zu merken, was zu eine sehr schlechte Leistung führen kann, wenn man dieselbe Trainingsdaten nimmt und leichter verschiebt. Um diesen Problemen entgegenzuwirken, wenn vollständigen verbundene Schichte (*fully connected layer*) durch Faltungsschichten (*Convolution layer*) ersetzen, die die Filterung eines Eingangssignals ermöglichen, um einen Teil seines Inhalts zu extrahieren und dadurch entstand eine andere Variante von Autoencoder, die Faltung-Autoencoder (*convolution autoencoder*). Obwohl diese Variante viele Schichte enthält, verwendet sie zu weniger Parameter (4.385). siehe Abbildung 7a.

Die gleichen Experimente wie oben wurden mit Faltung-Autoencoder-Modell durchgeführt und es ergibt sich die Abbildungen 7b und 7c. Aus Experimenten kommt auch heraus, dass das Training von Faltung-Autoencoder $30x$ länger als das von anderen Modellen dauern und weniger Epochen brauchen.

```
Model: "Convolutional_autoencoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 28, 28, 16)        160
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 8)         1160
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 8)           0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 8)           584
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 8)           0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 4, 4, 8)           584
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 8, 8, 8)           0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 8, 8, 8)           584
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 8)         0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 14, 14, 16)        1168
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 28, 28, 16)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 28, 28, 1)         145
=================================================================
Total params: 4,385
Trainable params: 4,385
Non-trainable params: 0
```

(a) Convolutional Autoencoder Modell

(b) Original und generierte Bilder (MNIST-Datensatz)



(c) Original und generierte Bilder (Smile-2-Datensatz).

4 Generative Adversarial Network

Generative Adversarial Networks (erzeugende gegnerische Netzwerke) (GANs) sind eine leistungsstarke Klasse von neuronalen Netzwerken, die für unüberwachtes Lernen eingesetzt werden. Sie wurde 2014 von Ian J. Goodfellow entwickelt und eingeführt. GANs bestehen im Wesentlichen aus einem System von zwei konkurrierenden neuronalen Netzmodellen (einen Generator und einen Diskriminator), die miteinander konkurrieren und in der Lage sind, die Variationen innerhalb einer Datenbank zu analysieren, zu erfassen und zu kopieren [4]. Der Generator bekommt zufällige uniforme Daten und generiert gefälschte Daten, die mit Samplen aus Trainingsdatensatz sehr ähnlich sind und versucht damit den Diskriminator zu täuschen. Der Diskriminator hingegen versucht, zwischen echten und gefälschten Proben zu unterscheiden. Ein allgemeine Architektur von GAN kann aus der Abbildung 8.

Es gibt heutzutage sehr viele Versionen von GAN und werden in folgenden mehr nur auf zwei davon eingehen

4.1 Vanilla GAN

Vanilla GAN ist der einfachste Typ von GAN. Hier sind der Generator und der Diskriminator einfache Multi-Layer-Perzeptrons, siehe Abbildung 9 .

Eine kurzer Überblick von Bildern, die Mit Vanilla GAN generiert wurden, ist in der Abbildung 10 zu finden. Man stellt auch fest, dass die generierten Bilder im Vergleich zu Bildern, die mit Autoencoder generiert wurden, nicht so ähnlich wie die Bilder aus

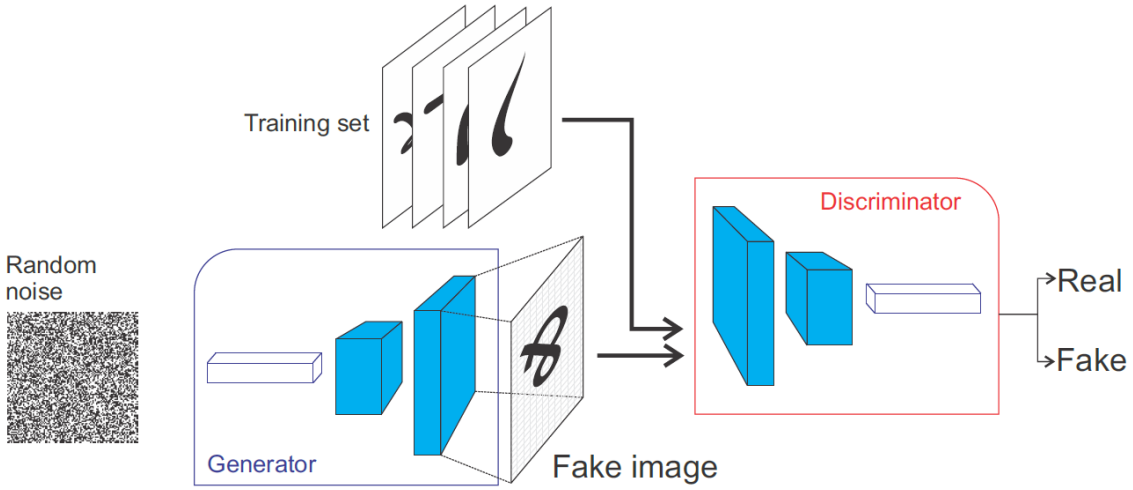


Abbildung 8: Generative Adversarial Networks Architektur

Model: "generaator"

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 784)]	0
dense_3 (Dense)	(None, 32)	25120
dense_4 (Dense)	(None, 784)	25872

Total params: 50,992
Trainable params: 50,992
Non-trainable params: 0

(a) Generator Modell

Model: "discriminator"

Layer (type)	Output Shape	Param #
input_5 (InputLayer)	[(None, 784)]	0
dense (Dense)	(None, 32)	25120
dense_1 (Dense)	(None, 16)	528
dense_2 (Dense)	(None, 1)	17

Total params: 25,665
Trainable params: 25,665
Non-trainable params: 0

(b) Discriminator Modell

Abbildung 9: Vanilla GAN Modell

dem originalen Datensatz aussehen. Unter Betrachtung der generierten Bildern lässt sich sagen, dass die GAN beim Problem vom Datenmangel besser helfen, denn die generierten Bilder von GAN Merkmale enthalten, die in Trainingsbilder auch vorhanden sind und die GANs können leichter verschiedene Versionen eines Bildes erzeugen.

4.2 Deep Convolutional GAN (DCGAN)

Deep Convolutional GAN (DCGAN) [6] ist die populärste Version von GAN. Es verwendet sowohl die vollständigen verbundenen Schichten als auch die Faltungsschite , um Daten bzw. Bilder zu generieren.Ein Beispiel von DCGAN-Modell kann in der Abbildung 11 entnehmen werden. Aus der Abbildung 12 stellt man fest, dass das NN sich in Laufe des Trainings immer bessern. Wenn man die Abbildungen 12 und 10 kommt heraus, dass die DCGAN Merkmale aus Datensatz besser als Vanilla GAN generalisieren können.

Die DCGAN-Modelle ergeben zwar gute Ergebnisse, aber benötigt auch zu viel Trai-

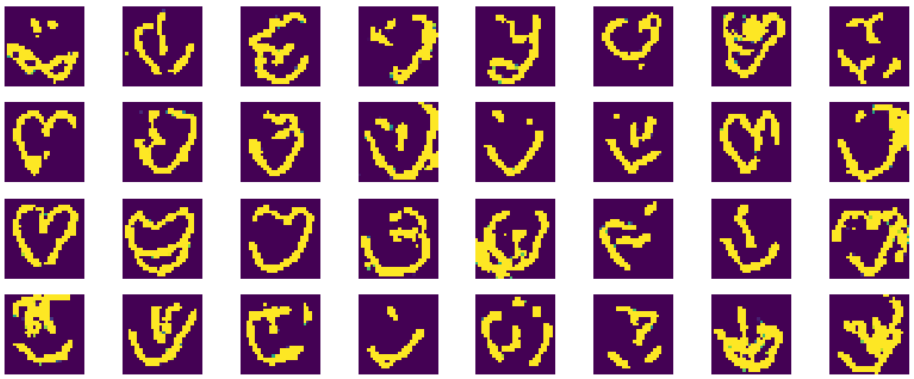


Abbildung 10: Generierte Bilder mit Vanilla GAN

Model: "generator"

Layer (type)	Output Shape	Param #
dense_9 (Dense)	(None, 6272)	633472
leaky_re_lu_5 (LeakyReLU)	(None, 6272)	0
reshape_2 (Reshape)	(None, 7, 7, 128)	0
conv2d_transpose_2 (Conv2DTr	(None, 14, 14, 128)	262272
leaky_re_lu_6 (LeakyReLU)	(None, 14, 14, 128)	0
conv2d_transpose_3 (Conv2DTr	(None, 28, 28, 128)	262272
leaky_re_lu_7 (LeakyReLU)	(None, 28, 28, 128)	0
conv2d_29 (Conv2D)	(None, 28, 28, 1)	6273

Total params: 1,164,289
Trainable params: 1,164,289
Non-trainable params: 0

(a) Deep Generator Modell

Model: "discriminator"

Layer (type)	Output Shape	Param #
conv2d_35 (Conv2D)	(None, 14, 14, 64)	640
leaky_re_lu_13 (LeakyReLU)	(None, 14, 14, 64)	0
dropout_4 (Dropout)	(None, 14, 14, 64)	0
conv2d_36 (Conv2D)	(None, 7, 7, 64)	36928
leaky_re_lu_14 (LeakyReLU)	(None, 7, 7, 64)	0
dropout_5 (Dropout)	(None, 7, 7, 64)	0
flatten_2 (Flatten)	(None, 3136)	0
dense_12 (Dense)	(None, 1)	3137

Total params: 40,705
Trainable params: 40,705
Non-trainable params: 0

(b) Deep Discriminator Modell

Abbildung 11: Deep Convolutional GAN-Modell

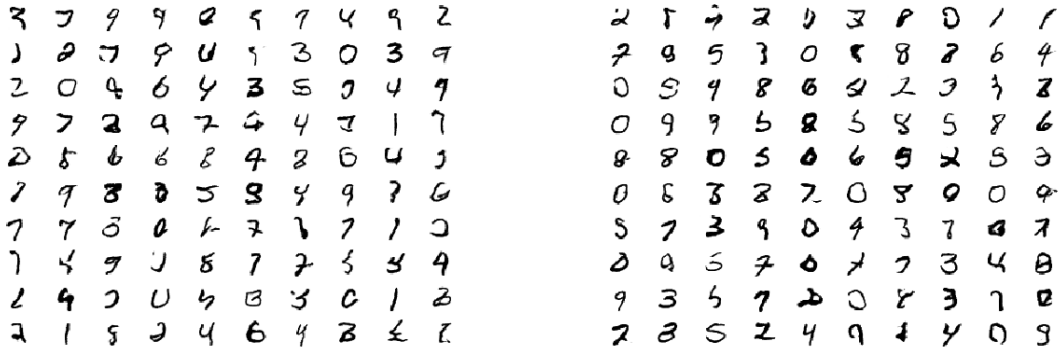
ningszeit und Ressourcen im Vergleich zu anderen Verfahren.

5 Zusammenfassung

Die Verwundung von neuronalen Netzen ist eine gute Möglichkeit, um synthetische Bilder zu generieren, die viele Ähnlichkeiten mit den Trainingsdaten haben. Man unterscheidet heutzutage zwei große Klasse von neuronalen Netzen zur Bildgenerierung: *Autoencoder* und *GAN*, die sich am meisten in der Architektur und die Art und Weise, wie sie trainiert sind, unterscheiden. Obwohl sie sehr gute Ergebnisse liefern, werden sie schnell problematisch, wenn es um Ressourcen und Training geht. Man kann eigentlich nie vorhersagen, wann man mit Training fertig ist, ohne selber die Qualität des Generators zu

(a) 1000 Iterationen (b) 20000 Iterationen

Abbildung 12: Generierte Bilder mit DCGAN (Smile-2)



(a) 500 Iterationen (b) 5000 Iterationen

Abbildung 13: Generierte Bilder mit DCGAN (MNIST)

testen und wenn die generierten Daten z.B auf ein Klassifikationsproblem angewandt, haben wir noch das Problem, dass es zuerst geprüft werden soll, ob die generierten Daten erstens akzeptabel und zweitens welcher Klasse sie zugeordnet werden sollen und leider muss diese Nacharbeit sehr oft per Hand gemacht werden.

Literatur

[1] Yann LeCun, et al. MNIST Database of Handwritten Digits

[2] Autoencoder

[3] convolutional autoencoder

[4] Generative Adversarial Networks

[5] Ian Goodfellow, et al. Generative Adversarial Networks. NIPS 2014

[6] Alec Radford and Luke Metz unsupervised representation learning with deep convolutional generative adversarial networks