Documentation of Project 'BaLo'

Building a Band Logo Generator with Generative Adversarial Networks

Scholar	Laura Hartgers
Matriculation Number	s0556238
Lecturer	Patrick Baumann
Course	Robotics and AI
Subjects	AI, Machine Learning, Neural Networks, Computer Vision, Image Generation
Subject Area	Data Science
Study Program	Applied Computer Science
Faculty	4
University	HTW Berlin

Contents

Chapter 1

Introduction

Artificial intelligence (AI) is a vast field, with numerous current and future use cases and applications. This projects focus lies on generating images artificially, more specifically on the generation of convincing metal band logos from various sub-genres.
Training a computer to generate images can be done through building generative models. This project focuses on Generative Adversarial Networks (GANs) specifically. GANs are generative models which use two adversarial models to enable the generation of new images after training, as discussed in Chapter 2. Looking at the bigger picture of AI, GANs and Generative models are a form of deep learning, which itself is a form of representation learning, that again is a form of machine learning. Machine learning in its turn is an essential part of AI nowadays. [GBC16]

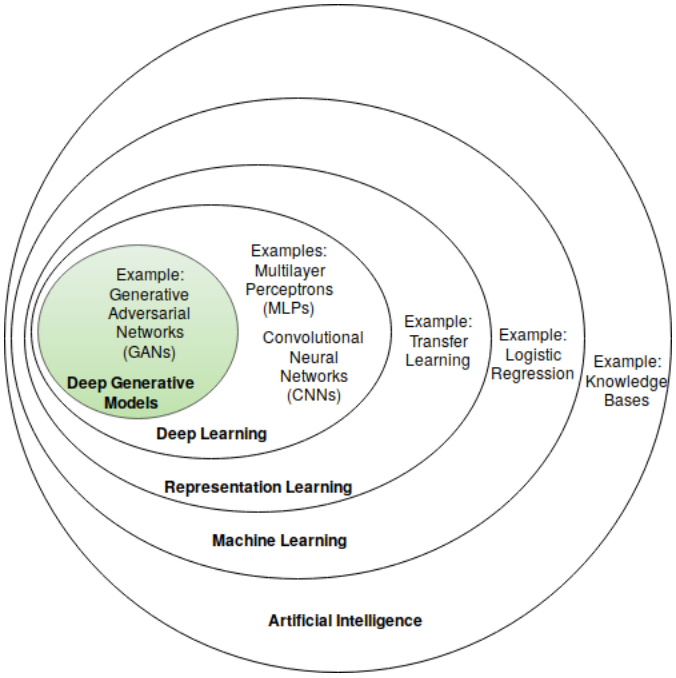


Figure 1.1: Venn diagram which contextualizes GANs in the world of AI, inspired by Goodfellow, Bengio, and Courville's book *Deep Learning* [GBC16]

Band logos are graphic symbols used to facilitate and promote public identification and recognition. They may be of an abstract, figurative design or include the text of the name they represent. In metal music, logos often indicate what sub-genre a bands music belongs to. For example, between Heavy Metal band logos and Black Metal in Figure 1.2, the difference is evident. Furthermore, relations between sub-genres become apparent when observing band logos as well. To demonstrate this, in Figure 1.3 are two logos of bands belonging to different genres. The tie between Pagan/Viking/Folk music and Black Metal is noticeable to the observer, as is the link between Power Metal and Heavy Metal.

(a) Heavy Metal logos.

(b) Black Metal logos.

Figure 1.2: Collections of two sub-genres. Images may be subject to copyright.

(a) Gloryhammer, Symphonic Power Metal with both old (above) and new logo.

(b) Falkenbach, Viking/Folk Metal.

Figure 1.3: An example for logos of other sub-genres. Images may be subject to copyright.

1.1 Motivation

During the winter semester 2018, I immersed myself in the field of machine learning and neural networks. After understanding basic principles and solutions for somewhat simpler problems like regression and classification, the desire to learn and challenge myself is still lasting. Therefore, this project is geared to gradually solving more complicated problems.

My fondness of metal music and the realization that band logos have a distinct look for various sub-genres sparked a personal interest in building a band logo generator. Humans with sufficient domain knowledge and/or experience are, when provided with the logo from a band unknown to them, in all probability capable of describing sound and genre of the music.

1.2 Exploratory work

Building a band logo generator is harder than for example classifying the MNIST images. The MNIST dataset consist of 60.000 training & 10.000 test images of handwritten digits 0-9. The digits have been size-normalized and centered in a fixed-size image. It is a good dataset for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. [LCB]

It is harder to build a band logo generator because the dataset needs to be build from scratch and new images - band logos - need to be generated. A neural network build and trained for classification of the MNIST images can classify them confidently, which is comparable to a multiple choice question, yet it has no ability to generate an image. This means that is essential to solve problems like building a dataset which meets all requirements for the task at hand, as is reading additional papers in order to find pointers on solving the problem of generating images etc.

Recent publications [[Goo+14] [Goo16], [RMC16], [MO14], [LN18]] haven proven GANs to be effective for solving this task. In chapter 2 and chapter 3 further exploration of the theoretical foundation will take place, followed by an evaluation of preliminary implementations of various GAN varieties on the MNIST dataset. Once the foundation has been laid, we will continue with the implementation of our Band Logo Generator (BaLo) in chapter 4.

1.3 Objective and key challenges

The main objective is to build a generative model which creates new band logos that people can recognize as such. Generated band logos should be of a chosen metal sub-genre. The available sub-genres will be determined by the training data.

Key challenges of this project include:

- **Understanding the theoretical foundation, Neural Networks, GANs**
  In order to build, manipulate and improve such a generative model, an adequate understanding about the principles of Neural Networks (NNs) and GANs specifically, as well as the theoretical foundation on which they are build is required.

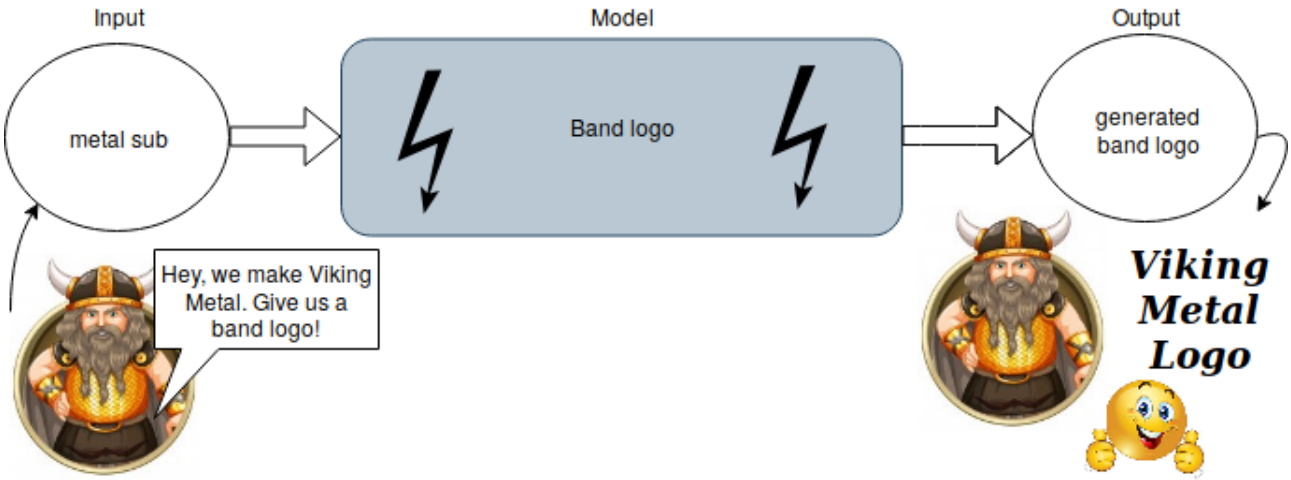


Figure 1.4: The main objective of this project is to build a model which generates convincing metal band logos for a given sub-genre.

- **Creating a dataset**
  The main challenge is the size of the data set: it must be large enough for the models to generalize well. Other challenges are the varying sizes of existing band logos: all training images need to have the same dimensions, differences in quality of the existing band logos, labeling, etc. In order to create a dataset we need data, which fortunately is found online at `https://www.metal-archives.com/`[Met].

- **Making design/implementation decisions**
  How can data of the dataset be optimally used by the model in order to achieve the projects objective?
  For example: Which sub-genres will be included for our model? Genres are described in natural language. Like many problems encountered in language, this is a Power Law distributed problem.[Ste16] Typical sub-genres represented by huge numbers of bands include for example Death Metal, Black Metal, Thrash Metal etc. Yet, there are virtually endless possibilities to describe a bands genre more precise, as shown in Table 1.1. In order enable a network to learn so much about a class that it can generalize well, the need for a significant amount of training data is inevitable.[GBC16]
  Other exemplary design/implementation decisions include the number and kind of layers and hyperparameters (e.g. learning rate, batch size), image size input and output, etc.

Genre description	*Occurrences*
'Black Metal'	14776
'Raw Black Metal'	821
'Thrash/Black Metal'	192
'Progressive Black/Death Metal'	51
'Raw Black/Death Metal with Industrial and Noise influences'	1
'Black Metal (early), Post-Metal/Shoegaze (later)'	1
'Black/Death Metal with Middle Eastern Folk Influences'	1

Table 1.1: Examples of genre descriptions in original data demonstrate this is a Power Law distributed problem

Chapter 2

Literature review and related work

An overview of the current state of research and the theoretical foundation.

2.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs) are a category of artificial intelligence algorithms that consist of a system of two NNs contesting with each other in a zero-sum game. Zero-sum games are a specific example of constant sum games where the sum of each outcome is always zero. [Hui18] If one wins, the other one loses, like when sharing a pizza.
GANs were introduced by Goodfellow et al. in 2014. This technique can generate images that look at least superficially authentic to human observers, having many realistic characteristics.

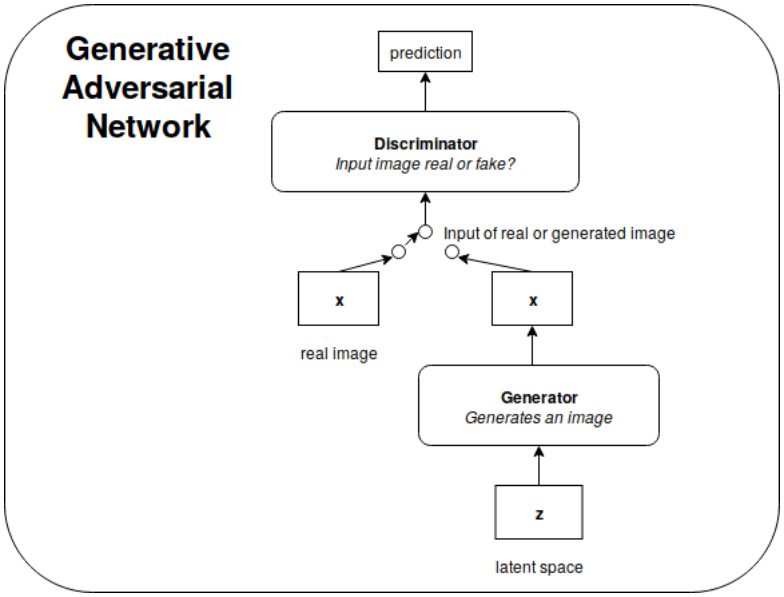


Figure 2.1: A GAN and its components. The generative model is challenging an adversary: a discriminative model that tries to determine whether a sample is from the generative model or the real dataset.

As an insightful analogy, one can think of generative model G as a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistinguishable from genuine money. [Goo+14]

Or more formally: A GAN consist of a generative model or generator G that captures the data distribution (fake currency), and a discriminative model or discriminator D that estimates the probability that a sample came from the training data (real currency) rather than from G (fake currency). The training procedure for G (the team of counterfeiters) is to maximize the probability of D (the police) making a mistake (e.g. not recognizing the fake currency). [Goo+14]

2.2 Deep Convolutional Generative Adversarial Networks

The DCGAN architecture is similar to the 'vanilla' GAN. However, the models D and G consist of convolutional layers instead of regular, fully connected neural networks. [RMC16]

Regular NNs don't scale well to full images. MNIST images are only of size 28x28x1 (28 wide, 28 high, 1 color channel) [LCB], so a single fully-connected neuron in a first hidden layer of a regular Neural Network would have 28*28*1 = 784 weights. This amount still seems manageable. Nevertheless, this fully-connected structure does not scale to larger images. For example, an image of more usual, every-day size that is still on the small side, e.g. 256x256x3, would lead to neurons that have 256*256*3 = 196.608 weights per neuron in the first layer only. Furthermore, we would almost certainly want to have at least a layer with several such neurons. As a result, the parameters would add up quickly. Clearly, this full connectivity is wasteful. [Sta]

Deep Convolutional NNs are specifically designed for computer vision tasks like visual recognition by successively looking at smaller parts of images (e.g. a central pixel and its neighbors) for recognizing patterns. The convolutional operation can loosely be compared to a human looking at an image top to bottom, left to right, searching for familiar patterns. In a NN, the convolutional operation utilizes a filter with multiple kernels to learn and recognize various familiar edges, shapes and patterns. The term filter and kernel are often used interchangeably. The convolution operation is best explained visually, as shown in Figure 2.3.

With Convolutional NNs the number of parameters can be dramatically reduced by making one reasonable assumption: That if one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2). The parameter reduction emerges by constraining the neurons in each depth slice to use the same weights and bias. With this parameter sharing scheme, the convolutional layer has only one unique set of weights for each depth slice (the number of kernels or filters of that layer). [Sta]

The Deep Convolutional GAN (DCGAN) architecture is similar to the 'vanilla' GAN. However, the models D and G consist of convolutional layers instead of regular, fully connected neural networks. [RMC16]

2.3 Conditional Generative Adversarial Networks

GANs originate in unsupervised machine learning. Unsupervised learning is a part of machine learning techniques, as shown in Figure 1.1, to find the patterns in data. The data provided to unsupervised algorithms are not labeled, meaning only the input variables (X) are given with no corresponding output variables. In unsupervised learning, the algorithms are left to themselves to discover interesting structures in the data. [Kur18]

Since GANs originate in unsupervised machine learning, the resulting generative model is unconditioned. Meaning there is no control on classes of the data being generated, as the models are not being provided with the label of a requested class. In these GANs, the generator G is inclined to generate an image that does seem to belong to the dataset it was trained on, which makes it impossible to request an image for a specific class.

By conditioning both G and discriminator D on additional information y it is possible to direct the data generation process. [Goo+14] Such conditioning could be based on class labels but also any kind of auxiliary information, such as data from other modalities. During training, D learns to recognize images of all specific classes and G learns to create images for specific classes. After training, G can produce images for a specified class y. [MO14]

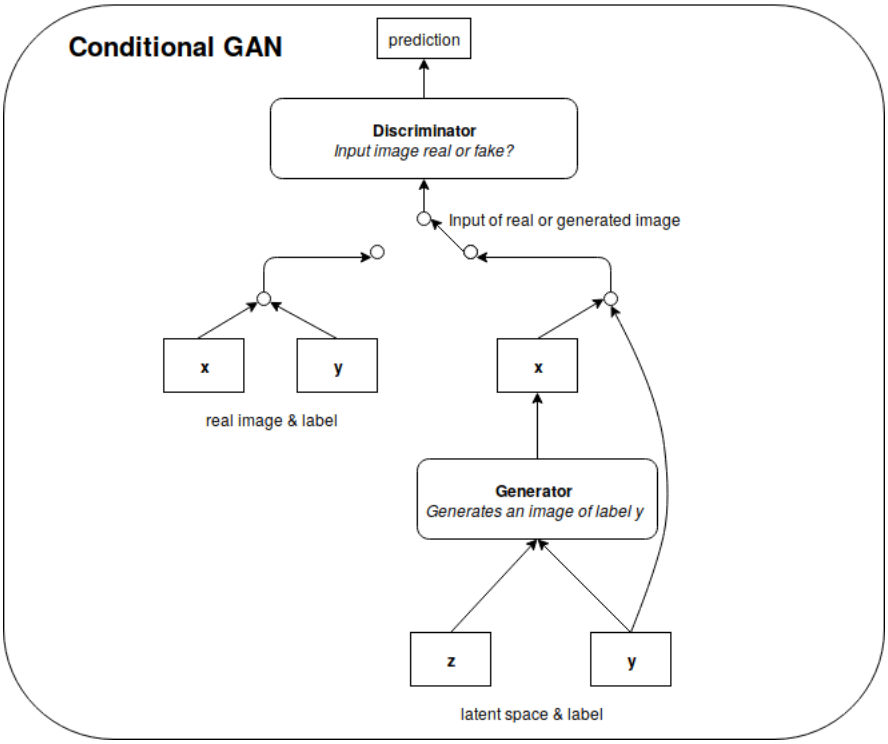
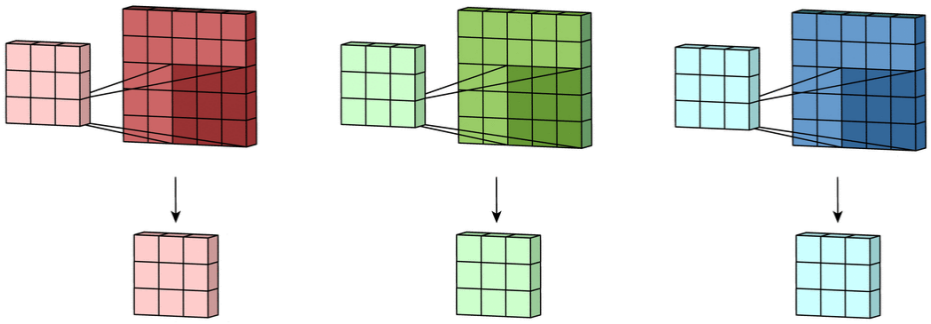
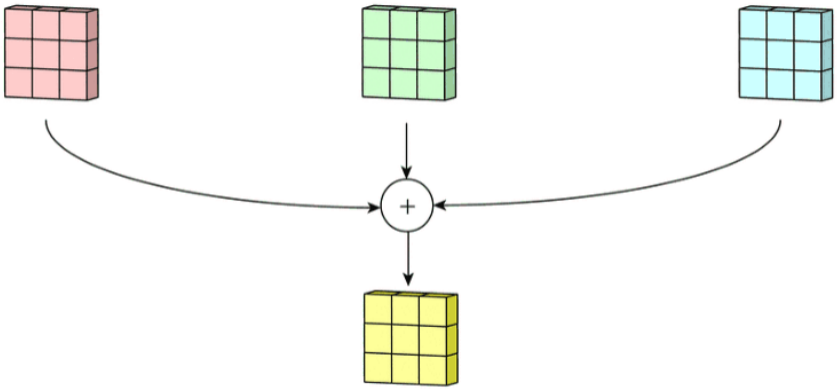


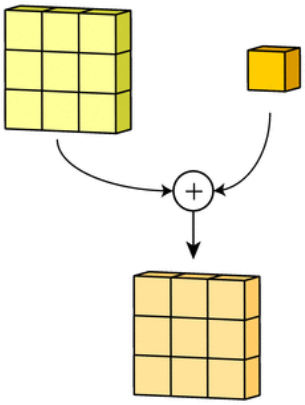
Figure 2.2: A Conditional Generative Adversarial Network or a *conditional generative model* $p(x \mid y)$ can be obtained by adding y as input to both G and D. [Goo+14]



(a) Each of the kernels "slides" over their respective input channels, producing a processed version of each. Each kernel has its own weights. A convolutional layer mostly contains several kernels.



(b) Each of the per-channel processed versions are then summed together to produces one overall output channel.



(c) The bias term is added. Each output filter has one bias term. The bias gets added to the summed output channel to produce the final output channel for the convolutional operation.

Figure 2.3: Convolution: multiple kernels or a filter produce one output channel. Images by Shafkat[Sha18]

Chapter 3

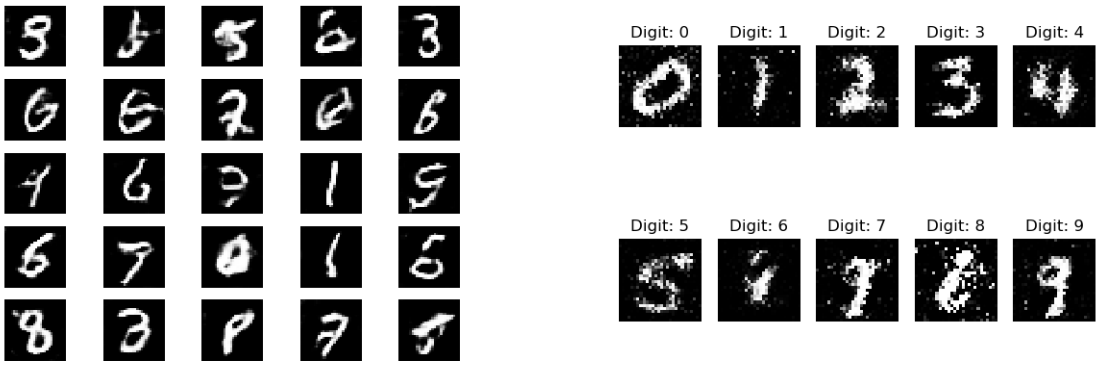
Evaluation of GAN architectures

After undergoing the theory, it is time for some practical implementations.

Starting with exemplary implementations by Linder-Norén concerning a Conditional GAN implementation as discussed in "Conditional Generative Adversarial Nets" by Mirza and Osindero. Followed by a Deep Convolutional GAN as discussed in Radford, Metz, and Chintala's paper "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". More specifically we will analyze this DCGAN this CGAN implementation by examining and deducing their results on the MNIST dataset.

Using a well known and well-prepared dataset a good place to start, since it allows us to fully focus on the GANs instead. Furthermore, using the same dataset with different models makes comparing the models manageable. Both implementations utilize Python 3 and the Keras API with TensorFlow as backend. Used hardware: NVIDIA GeForce GTX 1050 (2GB).

Figure 3.1, 3.4 and 3.3 show the image generating process of digits. The digits of both GANs evolve incrementally and become more convincing over time. However, the digits generated by the DCGAN and CGAN evolve in a different manner.



(a) DCGAN after 4.000 epochs.

(b) CGAN after just 1400 epochs.

Figure 3.1: Amount of epochs before the model generates digits that look recognizable.

3.1 Evaluation of results from Conditional GAN

In Figure 3.3 we can observe the Conditional GAN (CGAN) having a slower start at the beginning of training. It seems to take more epochs to form something that is recognizable as a number for humans. However, after 1400 epochs the generated digits start to look recognizable. Compared to the DCGAN, this is more than two times as fast. We can assume the cause for this to be its simpler net configuration (dense layers in the CGAN versus convolutional layers in the DCGAN). As training continues the CGAN consistently produces very convincing images for each individual digit, probably as a result of its knowledge about the label of the digit it needs to generate. This turns out to be an advantage that outweighs its simpler net configuration and provides better results overall. It also helps to solve a problem for the band logo generator (BaLo): to be able to dictate what kind of image should be produced. Digit numbers can be substituted by sub-genres with relative ease.

3.2 Evaluation of results from Deep Convolutional GAN

In comparison, the Deep Convolutional GAN (DCGAN) seems to be quicker at first, as shown in Figure 3.4 (b) and (c), compared to the respective images in Figure 3.3. However, it takes a whopping 4.000 epochs until it generates clearly recognizable digits, and even then its results are not consistent, as shown in Figure 3.4. Overall, the DCGAN produces less noisy images, but not all of them resemble human recognizable digits.

3.3 Conclusion CGAN & DCGAN

We conclude that the generated images by the DCGAN are sharper and their quality is superior, but the consistency is lacking. Especially when compared to the CGAN, where generated images look more noisy, yet consistently approximate digits well. Therefore, the next logical step is to develop a combination with the best features of both networks. This combination includes and the CGANs knowledge about the label of the image it should to generate and the network architectures with the convolutional layers from the DCGAN. Consequently, it will be a Conditional Deep Convolutional GAN (C-DCGAN).

3.4 Implementation of Conditional Deep Convolutional GAN

Experimentally we must conclude that the Conditional Deep Convolutional GAN (C-DCGAN) does not yield the results we initially expected. The generator with convolutional layers in combination with knowledge of the labels seems to take a lot longer to learn to draw digits, as demonstrated in Figure 3.5. Therefore, training has been elongated to 100.000 epochs. Resulting in examples shown in Figure 3.2.

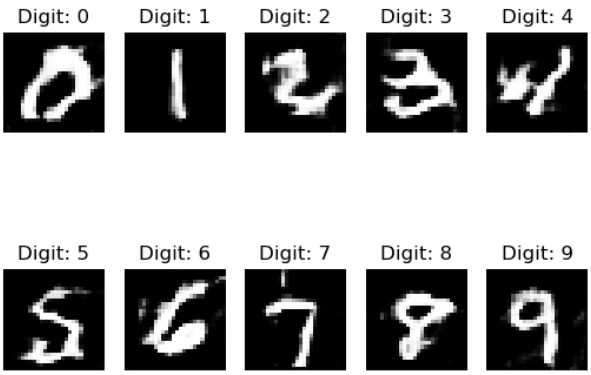
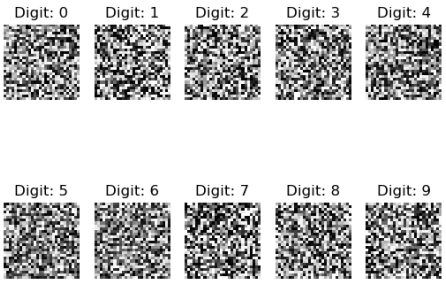
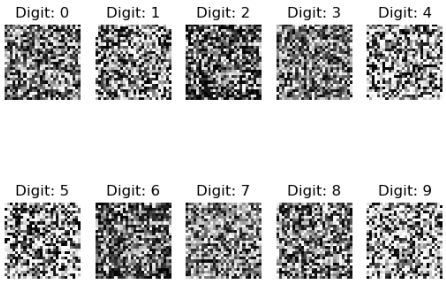


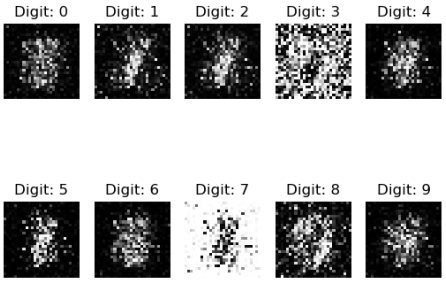
Figure 3.2: C-DCGAN generated digits after 100.000 epochs



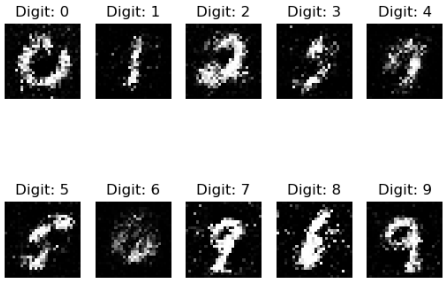
(a) Epoch 0



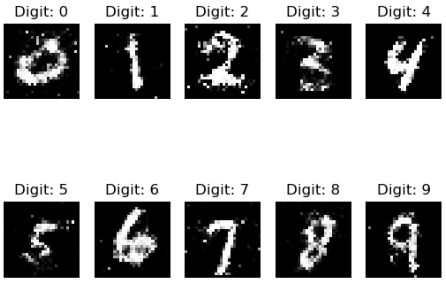
(b) Epoch 50



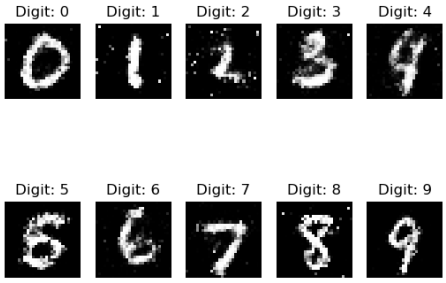
(c) Epoch 200



(d) Epoch 1.000

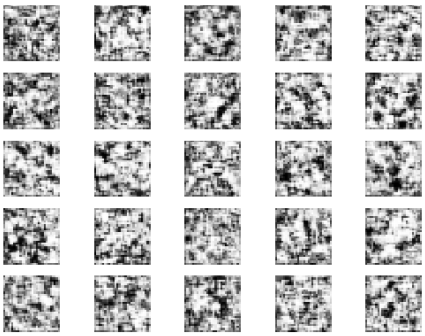


(e) Epoch 10.000

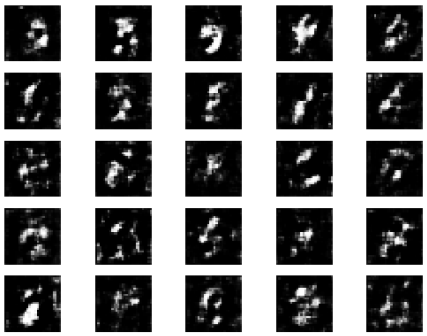


(f) Epoch 20.000

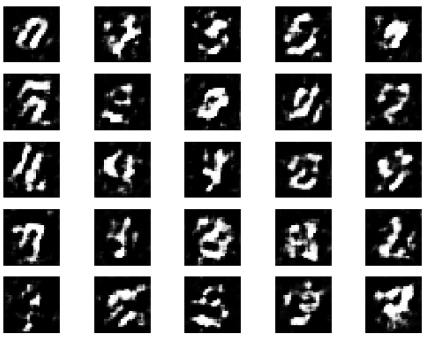
Figure 3.3: Generated images after n epochs of training by the **CGAN**. At the start, there is only noise as well. After 200 epochs we begin to see a separation of background and the pixels which should resemble a number somewhere in the center. The development of some digits is inverted at first, in terms of having a solid black border and a white cloud of pixels in the center. This situation resolves itself between 600 and 1000 epochs. As training continues the generated images start resembling numbers better. After 20.000 epochs, the results are quite pleasing.



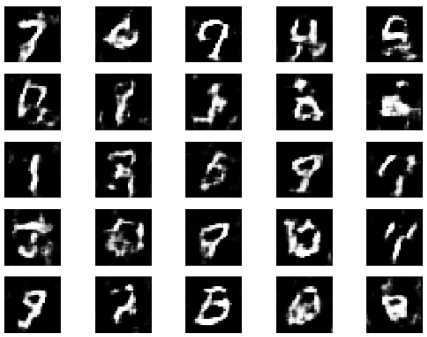
(a) Epoch 0



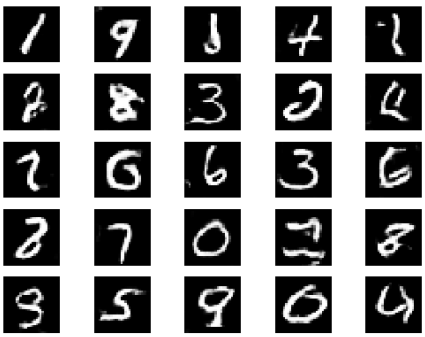
(b) Epoch 50



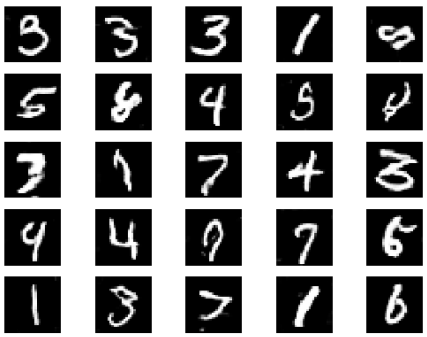
(c) Epoch 200



(d) Epoch 1.000

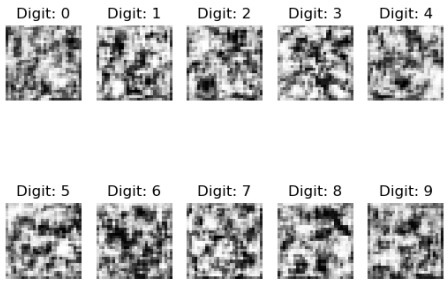


(e) Epoch 10.000

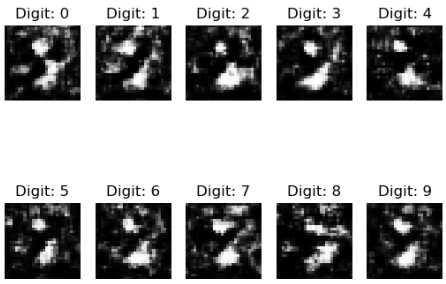


(f) Epoch 20.000

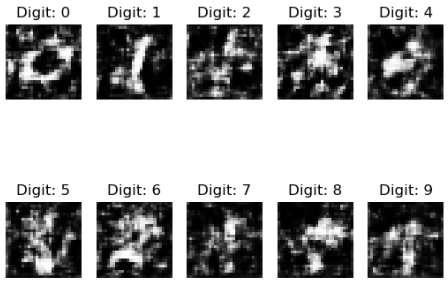
Figure 3.4: Generated images after n epochs of training by the **DCGAN**. With no training there is only noise. After just 50 epochs we begin to see a more distinct separation of the black background and the white future resemblance of a number somewhere in the center. As training continues the generated images start resembling numbers better. However, they are not always recognizable for humans.



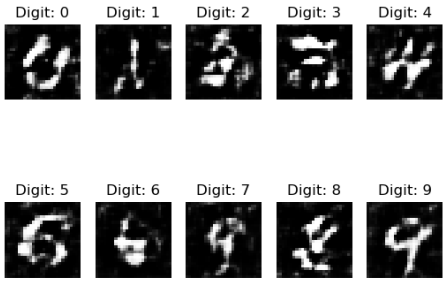
(a) Epoch 0



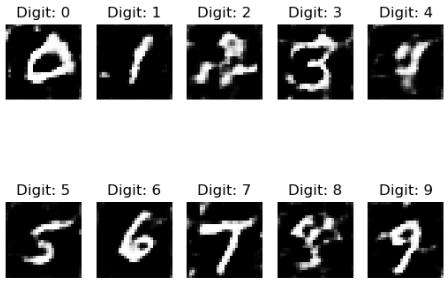
(b) Epoch 50



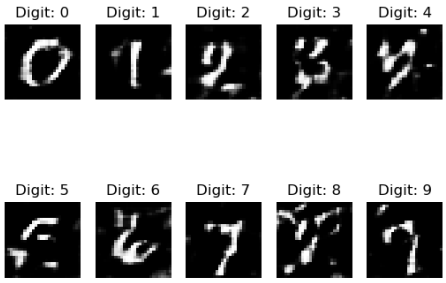
(c) Epoch 200



(d) Epoch 1.000



(e) Epoch 10.000



(f) Epoch 20.000

Figure 3.5: Generated images after n epochs of training by the **C-DCGAN**. Using the most prominent properties of both approaches - CGANs label input into the generator and DCGANs deep convolutional net structure - sounded promising. Unfortunately it does not reward us with better results in this amount of training.

Chapter 4

Implementation of BaLo

4.1 Creating the dataset

Band logos and additional information like band name and genre were crawled from Encyclopaedia Metallum: The Metal Archives [Met], a worldwide oriented website regarding metal bands. In total 43.636 band logos were downloaded.

4.2 Decisions concerning design and implementation

How can data of the dataset be optimally used by the model to achieve the project's objective of creating new band logos of a chosen sub-genre, that people can recognize as such?

4.2.1 Data preprocessing

Several steps need to be taken before the raw data can be processed to train our models.

Included sub-genres

As expected, since genres are described in natural language, the distribution of genres turns out to be a Power Law problem.[Ste16] Meaning there are a small number of frequently assigned genres, yet a considerably large amount of very specific genres that occur for only one or just a few bands. To be exact, 5887 distinct genres are so specific they emerge for one band only. Furthermore, 7654 genres are allocated to ten bands or less, 8084 genres for 100 bands or less and 8181 for 1000 bands or less. This leaves just 15 genres with 1000 or more allotted bands. Typical genres represented by larger numbers of bands include 'Black Metal', 'Death Metal', 'Thrash Metal' and 'Heavy Metal'. Based on domain knowledge similar sub-genres have been merged, ending up with 10 distinct classes to attribute bands to, as shown in Table 4.1. They represent the labels for the band logos BaLo is trained on.

Label	*Contains*	*n bands*
black	Black and Post-Black Metal, Post-Rock and Shoegaze	11.473
core	Metalcore, Deathcore, Hardcore, Grindcore etc. Goregrind and Nu-Metal	4.379
death	Death Metal	9.778
doom	Doom, Stoner, Sludge, Depressive and Dark Metal	2.827
gothic	Gothic and Symphonic Metal, Avant-Garde	1.291
heavy	Heavy Metal, NWOBHM	4.265
pagan	Pagan, Viking, Folk and Celtic Metal	981
power	Power and Speed Metal	1.174
progressive	Progressive Metal, Experimental and Shred	1.128
thrash	Trash and Groove Metal	6.214

Table 4.1: Labels, the sub-genres they contain and the number of bands we assigned to the various categories.

Image preprocessing

After downloading 43.636 band logos with all different modes, shapes and sizes, we prepare them to be suitable as input for BaLo.

Color 'RGB' is the original image mode in 41826 out of 43635 images. Calculating in 'RGB' mode is three times as expensive as black and white mode, due to the three channels in the former. Yet, color might to be of importance concerning genre recognition. Therefore the first run will be in 'RGB' mode with models dimensioned accordingly. For example, Heavy Metal logos do often contain red or other bold colors, while Black Metal logos are generally all white on a black background, as shown in Figure 1.2.

Size The task at hand is the challenge to find what size is large enough to leave sufficient details in the logos, so our models can learn and generate typical shapes which occur in sub-genres, yet is not too large to be calculated in a reasonable amount of time on the available hardware, a NVIDIA GeForce GTX 1050 (2GB). Mean size of our original data is 507 width x 256 height, suggesting most labels have a 2:1 (width:height) ratio. However, with a variance of 47227 and 21645 respectively the data are leaning towards a high disparity. Keeping the preprocessing steps as simple as possible in the first run and avoid fussing with padding, the initial resize will be 128 x 64. This honors the ratio and a visual check of resized logos confirms that it looks good in most cases. Nevertheless, this method disregards the variance, thus we apply this quite naive method knowing that there might be room for improvement in this area.

Data Augmentation Data Augmentation is a technique in which images are slightly altered with the goal of producing more training data. Alteration methods include mirroring, rotating, adding some noise to help combat against color space distortions such as lighting and moving slightly to left/right/top/bottom so the image is not so perfectly focused, making

sure the model learns features from the image and not it's location. [Sho18] In cases like ours, where the text is important, rotating needs to be limited to the point where text would still line up as it would do in real life. Mirroring is completely out of the question for this reason. Since there is no lightning on logos that we otherwise would be confronted with in images of the real world, adding noise does not seem to make sense for this project. Recapitulating, some aspects of data augmentation are possible to use for projects of this kind, but it would be of considerable expenditure. Benefits remain uncertain until it would be tested, therefore choosing to refrain from this technique because of the time constraints for this course.

Organizing and loading image and corresponding label into models

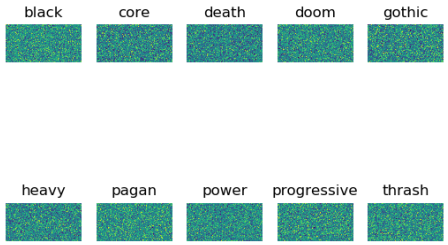
4.2.2 Design of BaLo

As discussed in chapter 3, the Conditional GAN (CGAN) performed best during the preliminary evaluation of different GANs with MNIST data. Therefore it is only logical to continue with this model.

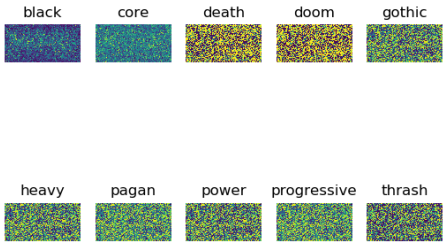
After deciding on the size (128 width x 64 height) and mode ('RGB') of the input image, the according parameters of the CGAN were adjusted to the more substantial input. Experimental adjustments of net configuration, such as adding an extra layer in the generator or increasing the size of a hidden layer, were based on intuition and did not result in a better outcome. However, such adjustments lead to higher computational costs. Therefore, this evaluation in done on the original CGANs network architectures, solely the inputs were adjusted.

4.3 Evaluation

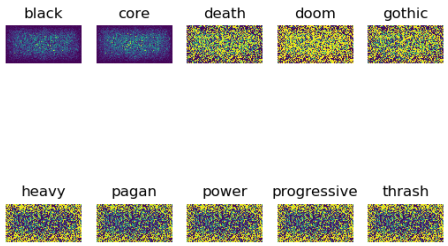
Results of training on this specific implementation are shown in Figure 4.1. Between 1.000 and 100.000 epochs result still look promising, yet more extensive training of 1.000.000 epochs, as shown in Figure 4.2, proves to be ineffective. It seems that BaLo found itself a happy local minimum, where it can fool the discriminator into believing that this noise resembles band logos of the given sub-genres. However, the human observer cannot be fooled by this. Therefore, improvements need to be made, which we discuss in chapter 5.



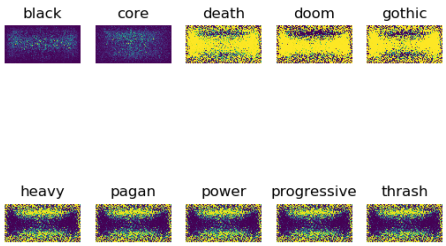
(a) Epoch 0



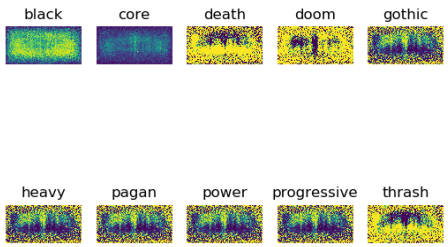
(b) Epoch 200



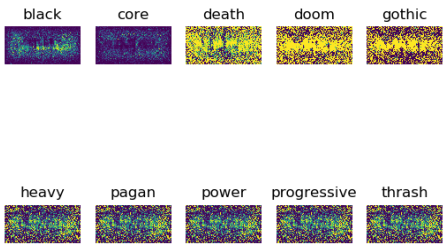
(c) Epoch 500



(d) Epoch 1.000

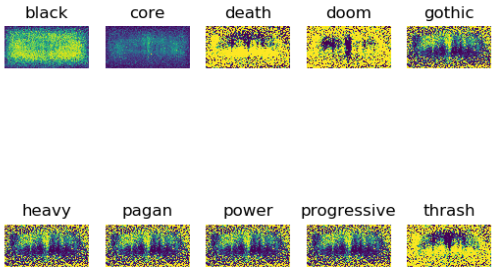


(e) Epoch 6.000

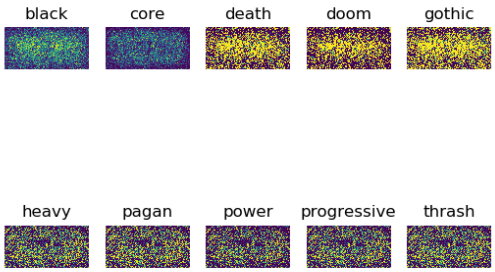


(f) Epoch 100.000

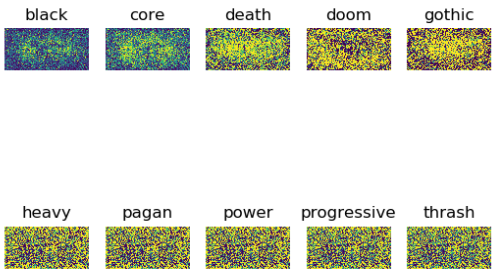
Figure 4.1: Generated images after n epochs of training by the **BaLo CGAN**. With no training there is only noise. After 1.000 epochs it seems like BaLo starts generating something that looks familiar to band logos. Epoch 6.000 and 10.000 yield better result, meaning BaLo is getting better with more training.



(a) Epoch 6.000



(b) Epoch 500.000



(c) Epoch 1.000.000

Figure 4.2: BaLo generated band logos after 6.000, 500.000 and after 1.000.000 epochs, proving that beyond a certain point, more training does not yield more satisfactory results. On the contrary.

Chapter 5

Discussion & Conclusion

Building a dataset, e.g. writing scripts for crawling data and preprocessing, takes a substantial amount of time and thought. The necessary time and effort has been underestimated, as well as the impact of practical difficulties like to load own data into our models. To gain a pleasing and ultimately achieving a higher quality result, there is room for improvement in both the dataset and the network architecture.
In this chapter, we will discuss these learned lessons in more detail.

5.1 Future work

For eventual future continuation, there are several goals. They might lead to the development of a product in the form of a website or app. Before turning BaLo into a product, it needs to be made sure that there are no copyright issues and that no unwanted shapes of forms, such as the swastika and other symbols of National Socialists will be presented as the output of BaLo. Such issues can be expected since unfortunately, the problem of fascistic ideas is distributed into the metal community as well, mainly in parts of the Black Metal and Pagan/Viking community. These ideas reflect as symbols which are probably also on some logos in the dataset we build to train BaLo. Additionally, there might be other similar issues that we are currently unaware of.

5.1.1 Achieving a higher quality output image

BaLo does currently not generate believable band logos. Several approaches are possible. After evaluation, we must conclude that it is advisable to improve in all areas.

Improve the dataset quantity and quality

We propose to expand and improve the dataset to have more and better data to train, validate and test on.

For training, it will be beneficial to have more data, since 'black', the category with the highest number of training data, proves the importance of numbers by yielding superior results, when compared to other categories with significantly less data, like 'gothic', 'pagan',

'power' and 'progressive'. More data can, for example, be found in other image folders the original data source, *Encyclopaedia Metallum: The Metal Archives*, since we only used `https://www.metal-archives.com/images/3/5/4/0/` for our convenience. Also, as discussed in section 4.2.1 'Image preprocessing', images can be used like a 'new' image multiple times by utilizing data augmentation. Furthermore, 'black and white' mode would reduce the computational cost to three times less compared to 'RGB', making it possible and more appealing to increase the number and/or size of hidden layers. Therefore, we propose to test 'black and white' mode as well. Additionally, the quality of our input images can be improved by implementing a more refined preprocessing procedure for resizing, e.g. utilizing padding in order not to distort logos.

A visual examination of band logos assigned to each category showed a less cohesive collection than anticipated. It is expected that this is very hard, maybe even impossible, for a neural network to generalize such differences in a way that looks good to humans. Therefore, the assignment of categories needs to be revised. This could be done by further examining the original dataset and writing a more sophisticated category assignment algorithm by hand. The major disadvantages are that this method is very time consuming and might not yield the needed amount of improvement, since we practically cannot write such an algorithm with regard for the forms, shapes, edges etc. of the original band logos. Therefore, we propose to let a neural network sort the band logos into categories by itself, performing unsupervised learning, with the goal being getting more insight into the kind of patterns it picks up. From there, based on domain knowledge it is plausible to end up with a category assignment both humans and computers can understand.

Improve the Networks

Having a bigger dataset available is not only beneficial for training. Performing a hyperparameter search using a validation dataset and optimizing the nets of generator and discriminator, will be beneficial to the outcome as well. For example, examining multiple net sizes (number of hidden layers, number of neurons per layer), net architectures such as convolutional NNs, learning- and dropout rates for BaLo.

Resolving issues with the C-DCGAN

To be able to utilize Convolutional NNs, issues with the C-DCGAN need to be resolved. It is expected that a Convolutional NN in the discriminator picks up on important features, e.g. shapes, patterns and edges from input logos. Therefore it will probably not accept the kind of noisy creations by the generator, as shown in Figure 4.2, leading to better results. Also, a generator with convolutional hidden layers produces sharper images, as shown in by the DCGAN on MNIST data in Figure 3.4. Our current assumption is that how the label is put into the generator causes it to have trouble recognizing its meaning. We suggest further investigating this issue and propose to use concatenation rather than multiplication of label and input noise, since this might clear up the matter because it is easier to interpret. Another idea is to make the input even more optimal for a CNN to understand: show it as an image. For example, the input of band names could be an image of the band name in a neutral font.

5.1.2 Band name as second input

A more advanced objective of this project is to extend the model input with a band name. This input option will also increase the chance to transform the project into a practical application. For example, one could make a website where users can let the model create a band logo which corresponds to both their genre and band name. Since a band name is an essential characteristic of a band, the possibility of having a logo made with consideration of both style and band name will most likely be the difference between just fun and our model providing additional value for our users, by generating good, usable logos.

This feature brings a considerable amount of new questions. For example, what is the best or most practical way to translate words into a label that our models can understand? One-hot-encoded works for our class input since the number of classes is very limited the one-hot-encoded vector remains overseeable. Nevertheless, band names are unlimited in number, and it is impossible to know what band names future users of BaLo might come up with.

Therefore, solutions are likely to be found in more compact vector representations of words, for example character-based one-hot-encoded with a fixed maximum length. A one-hot encoding is a representation of categorical variables as binary vectors. If the length is set to 15 characters and no special characters are being allowed, the character-based one-hot-encoded vector will have a dimension of [390,1], which is manageable because it is well within the size of inputs we can take into our models.

As an example, *'Amon Amarth'* could be represented by a character-based one-hot-encoded vector as follows:

1. Convert the band name to lower case, remove spaces, special characters and make sure to take only the 15 (for this example) first characters. Result: 'amonamarth'.

2. Generate a one-hot-encoded vector for each character

   - Map the categorical values (characters) to integer values: a = 0, b = 1 etc.
   - Each integer value is represented as a binary vector (size = 26) that is all zero values except the index of the integer, which is marked with a 1.
     Result: 'a' = [1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
   - Repeat until all characters in 'amonamarth' have been translated to a one-hot-encoded character vectors.

3. Append the one-hot-encoded character vectors in the same order the band name dictates.

4. Pad leftover space with 0's, in case the maximum length has not been reached (meaning the band name was shorter than 15 characters).

Other solutions might include word embeddings like Word2Vec [Mik+13] and FastText [Boj+16] [Jou+16].

A digression on word embeddings A word embedding is a popular framework to represent each word as a vector in a vector space of many (for example 300) dimensions, based on the semantic context in which the word is found. [CBN17] This technique has been used in many machine learning and natural language processing tasks. The term word embedding is being used to describe multiple word vectors in a data set as well as for a single word vector in such a word embedding.

Imagining such high-dimensional vectors and word embeddings is challenging. Therefore, the Embedding Projector [STN] has been developed for interactive visualization of high-dimensional data by rendering them in two or three dimensions. The example in Figure 5.1 is based on input data from Word2Vec All, which means this word embedding consists of roughly 70.000 vectors with 200 dimensions. Word2Vec [Mik+13] is a publicly available trained model and one from the collection of models in the Embedding Projector by Google. To reduce the dimensionality of this data set, the t-SNE (T-distributed stochastic neighbor embedding), a nonlinear nondeterministic algorithm has been chosen because it tries to preserve local neighborhoods in the data, meaning that vectors of words with similar semantics are being projected in close approximation. [Goo]

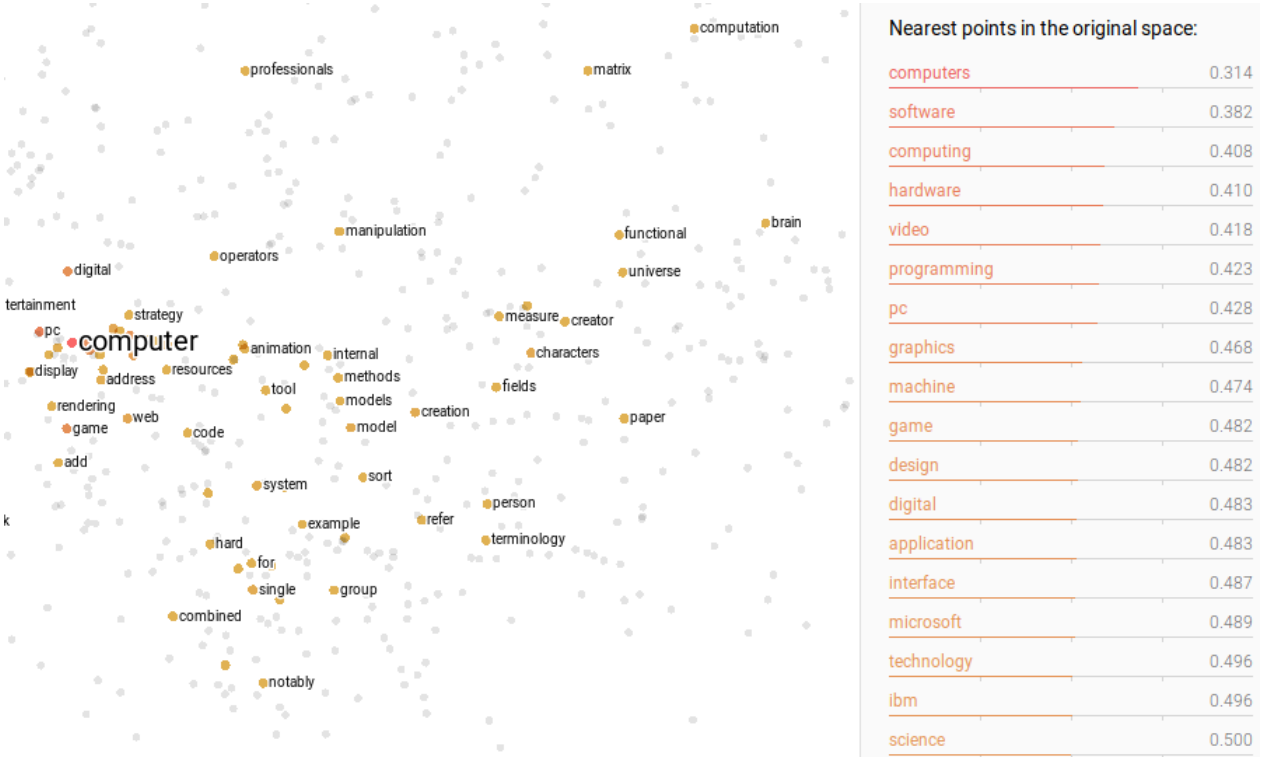


Figure 5.1: An example of a word embedding visualization after it has been rendered in two dimensions. Each dot represents a vector of 200 dimensions. Nearest neighbors of the *computer* vector in the original space are highlighted in the graph and listed on the right side. [STN]

5.2 Conclusion

The ultimate goal to build a model which creates new band logos of a chosen sub-genre has not been achieved, as BaLo does not yield satisfactory results yet. However, it was apparent that this goal was very ambitious from the beginning. Therefore, with the difficulties of such project in mind, e.g. building a dataset, literature research in order to solve emerging problems, understanding and training GANs, and the additional time constraints by this course, it is far more important that the underlying objectives, e.g. engaging in a personal challenge, working on a real-world problem and learning a lot, have been achieved.

Bibliography

[Boj+16] Piotr Bojanowski et al. "Enriching Word Vectors with Subword Information". In: *arXiv:1607.04606 [cs]* (July 15, 2016). arXiv: 1607.04606. URL: http://arxiv.org/abs/1607.04606 (visited on 02/05/2019) (cited on page 23).

[CBN17] Aylin Caliskan, Joanna J. Bryson, and Arvind Narayanan. "Semantics derived automatically from language corpora contain human-like biases". In: *Science*. Volume 356. Apr. 14, 2017, pages 183–186. DOI: 10.1126/science.aal4230. (Visited on 12/28/2017) (cited on page 24).

[Goo16] Ian Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *arXiv:1701.00160 [cs]* (Dec. 31, 2016). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160 (visited on 12/20/2018) (cited on page 3).

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL: http://www.deeplearningbook.org/ (visited on 12/18/2018) (cited on pages 1, 4).

[Goo+14] Ian J. Goodfellow et al. "Generative Adversarial Networks". In: *arXiv:1406.2661 [cs, stat]* (June 10, 2014). arXiv: 1406.2661. URL: http://arxiv.org/abs/1406.2661 (visited on 12/17/2018) (cited on pages 3, 6–8).

[Goo] Google. *Embeddings*. TensorFlow. URL: https://www.tensorflow.org/guide/embedding (visited on 02/02/2019) (cited on page 24).

[Hui18] Jonathan Hui. *GAN — Why it is so hard to train Generative Adversarial Networks!* Medium. June 21, 2018. URL: https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b (visited on 02/05/2019) (cited on page 6).

[Jou+16] Armand Joulin et al. "Bag of Tricks for Efficient Text Classification". In: *arXiv:1607.01759 [cs]* (July 6, 2016). arXiv: 1607.01759. URL: http://arxiv.org/abs/1607.01759 (visited on 02/05/2019) (cited on page 23).

[Kur18] Vihar Kurama. *Unsupervised Learning with Python*. Towards Data Science. May 11, 2018. URL: https://towardsdatascience.com/unsupervised-learning-with-python-173c51dc7f03 (visited on 02/05/2019) (cited on page 8).

[LCB] Yann LeCun, Corinna Cortes, and Chris Burges. *MNIST handwritten digit database*. URL: http://yann.lecun.com/exdb/mnist/ (visited on 02/04/2019) (cited on pages 3, 7).

[LN18] Erik Linder-Norén. *Keras implementations of Generative Adversarial Networks.: eriklindernoren/Keras-GAN*. original-date: 2017-07-11T16:24:53Z. Dec. 11, 2018. URL: https://github.com/eriklindernoren/Keras-GAN (visited on 12/11/2018) (cited on pages 3, 10).

[Met] Encyclopaedia Metallum. *Encyclopaedia Metallum: The Metal Archives*. URL: https://www.metal-archives.com/ (visited on 01/30/2019) (cited on pages 4, 16, 22).

[Mik+13] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *arXiv:1301.3781 [cs]* (Jan. 16, 2013). arXiv: 1301.3781. URL: http://arxiv.org/abs/1301.3781 (visited on 02/05/2018) (cited on pages 23, 24).

[MO14] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *arXiv:1411.1784 [cs, stat]* (Nov. 6, 2014). arXiv: 1411.1784. URL: http://arxiv.org/abs/1411.1784 (visited on 12/23/2018) (cited on pages 3, 8, 10).

[RMC16] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: (2016), page 16 (cited on pages 3, 7, 10).

[Sha18] Irhum Shafkat. *Intuitively Understanding Convolutions for Deep Learning*. Towards Data Science. June 1, 2018. URL: https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1 (visited on 02/02/2019) (cited on page 9).

[Sho18] Connor Shorten. *Data Augmentation and Images*. Towards Data Science. Aug. 31, 2018. URL: https://towardsdatascience.com/data-augmentation-and-images-7aca9bd0dbe8 (visited on 02/02/2019) (cited on page 18).

[STN] Daniel Smilkov, Nikhil Thorat, and Charles Nicholson. *Embedding projector - visualization of high-dimensional data*. URL: http://projector.tensorflow.org (visited on 02/02/2018) (cited on page 24).

[Sta] Stanford University. *CS231n Convolutional Neural Networks for Visual Recognition*. URL: http://cs231n.github.io/convolutional-networks/ (visited on 02/05/2019) (cited on page 7).

[Ste16] Stephanie. *Power Law and Power Law Distribution*. Statistics How To. July 15, 2016. URL: https://www.statisticshowto.datasciencecentral.com/power-law/ (visited on 01/03/2019) (cited on pages 4, 16).