Recognizing Musical Notation Using Artificial Neural Networks

A thesis submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science

Pascal Attwenger

June 30, 2015
University of Vienna
Faculty of Computer Science

Artificial neural networks are a machine learning method that can be used to solve a multitude of different tasks. In this thesis we strive to discuss some of the theoretical basics behind this technology and apply it to a practical problem in the field of digital image classification: Recognizing musical notation from scanned sheet music.

We cover the entire process from finding, labelling and processing the data, over building the networks themselves, to optimizing their performance with different training techniques.

The Python-based implementation uses frameworks such as PyBrain and OpenCV to facilitate this process, and the output is presented in the format of MusicXML, a leading standard for exchanging musical notation.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

There is hardly another topic that gets computer scientists as excited as Artificial Intelligence and Machine Learning. Research has made great progress in the last few years, and when one watches a YouTube video of a robot controlled by cells from an actual rat's brain [18], one cannot help but think that the so-called future is—finally!—here.

However, it is not that simple for a lowly Bachelor student to dive into Machine Learning and immediately start building one's very own rat robot. An easier introduction to the subject matter had to be found, and what could be easier than OCR?

Coupled with a keen interest in music, the topic of choice was found at last: OCR applied to printed sheet music.

Alas, the typical workflow of processing a scanned page of music did not seem that exciting: Staff line removal was a big point of interest, and algorithms with fun-sounding names like GDA and LAG.

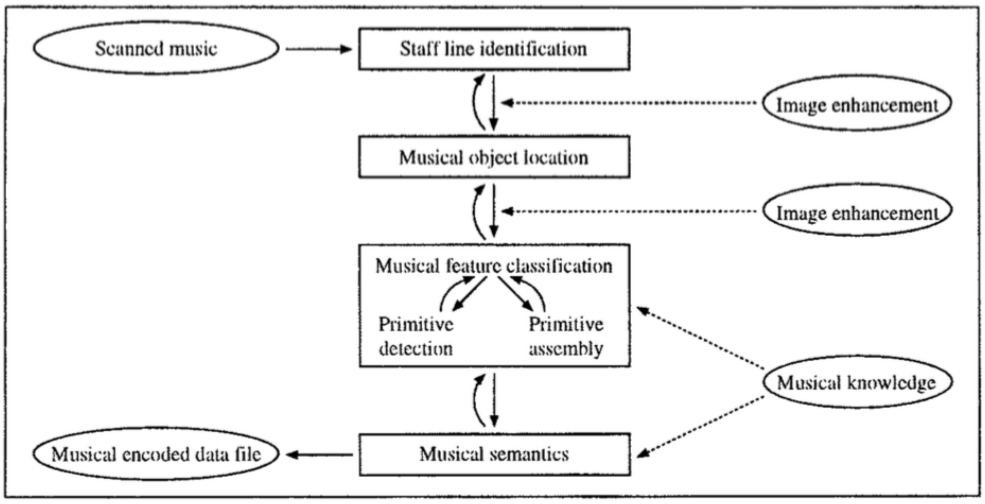

Figure 1: Typical workflow of music OCR

There had to be a more elegant approach, one where the computer itself learns how to cope with all those lines and dots and bars that we humans love so much. Only what?

In the end, the answer to this question was an easy one: If it takes a brain to appreciate music, let's build one and give it to the computer: An Artificial Neural Network.

1.2 Thesis summary

The first part of this thesis will deal with the theoretical foundations behind the topics presented here. Section 2 lists past research in the field of digital music recognition while Section 3 gives a short introduction to the basics of machine learning, neural networks, and musical notation; and Section 4 shortly describes the main software components used to realize the practical part of the thesis.

In Section 5 we explain the techniques used to find and label the data that was needed for the work described in Section 6, the actual implementation. Section 7 tries to analyze why and how well that worked.

Finally, Section 8 gives a few ideas how to continue this type of work in the future, and Section 8 concludes the thesis.

2 Related Work

Some of the earliest work concerning optical character recognition (OCR) for printed sheet music was done in the late 1960s by Dennis Howard Pruslin at MIT and later continued by David S. Prerau. Of course, machine learning techniques were more of a theoretical domain back then, as the computers of that time did not yet have the processing power needed for solving such complex tasks with artificial neural networks or similarly advanced solutions. Instead, they used algorithms that isolated contour traces for the identification of note values or assembled multiple black-region fragments to components roughly corresponding to different musical symbols. The results of those early work left contemporary collegues optimistic that practically usable solutions *"can be seen to be no further than another couple of M. I. T. dissertations away."* [9]

The authors of [2] give a thorough explanation of the challenges of optical music recognition that sets it apart from related tasks like character or digit recognition. They go on to listing and comparing recent trends in music OCR; those include several different technological approaches such as Line Adjacency Graphs, k-nearest neighbour classification, Artificial Intelligence models with intrinsic knowledge of musical notation, grammar-based systems and a specifically designed programming language with an object oriented interface for musical semantics.

Finally, [8] describes a recent sheet music analyzation experiment carried out at the University of Utah that has a similar scope as the one described in this thesis. Using Gaussian Discriminant Analysis, they achieve successful classification on their test set where note values are concerned (at 85% accuracy), but not particularly good ones on their pitch classifier (at 33% accuracy).

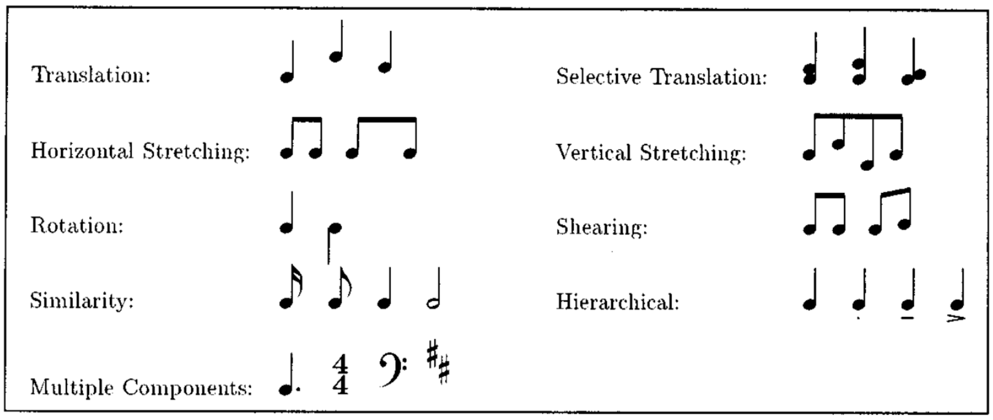

Figure 2: Properties of music not found in written text (from Bainbridge et al.)

3 Theoretical foundations

3.1 Machine learning

When shown a picture of a letter or a digit, most people will be able to identify the character effortlessly; it is something they have *learned* over the course of their lives, probably in school.

However, not many people will be able to explain how exactly their recognition process works. This is a problem when trying to write a computer program that should be able to perform said recognition: How can one come up with an algorithm in a programming language when there is no equivalent algorithm happening in the human brain of the programmer?

There are many such problems were people cannot *teach* the computers exactly how to solve them; instead, they build systems that allow the computers to *learn* how to solve those problems themselves. This is called *machine learning.*

One of the most often cited quotes defining what machine learning does goes as follows:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." [12, p. 2]

Put simply, if a program gets better at its job the more training it get, this can be considered machine learning.

Applied to OCR, the "task" is the identification of a character. The "experience" increases with more data to train on or with a higher number of repetitions (epochs). "Performance" can be measured by the error rate that a program has on a set of test data that it was not previously trained on.

Machine learning can be divided into three broad groups: Supervised learning, unsupervised learning, and reinforcement learning.

3.1.1 Unsupervised learning

Unsupervised learning is used to find hidden or at least previously unknown structures in a set of data. One example of this is clustering: Here the program tries to divide the data into several clusters of related samples without prior knowledge of which factors are most useful to discriminate between them. Those factors are only found during the learning process.

3.1.2 Reinforcement learning

Reinforcement learning provides the program with a task and a score of how well it is doing, but not many specifics on the task itself. This is the type of machine learning algorithm that allows computers to learn how to play games like *Tetris* without giving them instructions on how the game works or even which buttons to press; the computer only gets to see the graphical output of the game and the score it has reached, and it has to figure out all the game mechanics and strategies by itself. [6]

3.1.3 Supervised learning

Finally, there is supervised learning, which this thesis will concentrate on. Here, the training data is labelled with values and it is the program's task to produce correct labels for unknown data. Supervised learning itself can again be divided into two applications: Regression and classification.

Highly simplified, regression will output a continuous number, for example the predicted selling price of a house with a given amount of input parameters (features) like size, age, location etc.

Classification on the other hand will output a discrete number representing the category the input data was assigned; this is what we will try to achieve for musical notation.

3.2 Artificial neural networks

Artificial neural networks are an approach where software is used to imitate biological neural networks like the ones found in the brains of animals or humans. They are capable of modelling very complex functions if trained correctly, which makes them an excellent fit for many machine learning applications.

A neural network consists of multiple processing units called "neurons" that are arranged in several layers. Every neural network has one input layer, one output layer and a variable amount of hidden layers. Figure 3 shows a neural network with four input units, one hidden layer with five units and one output unit.

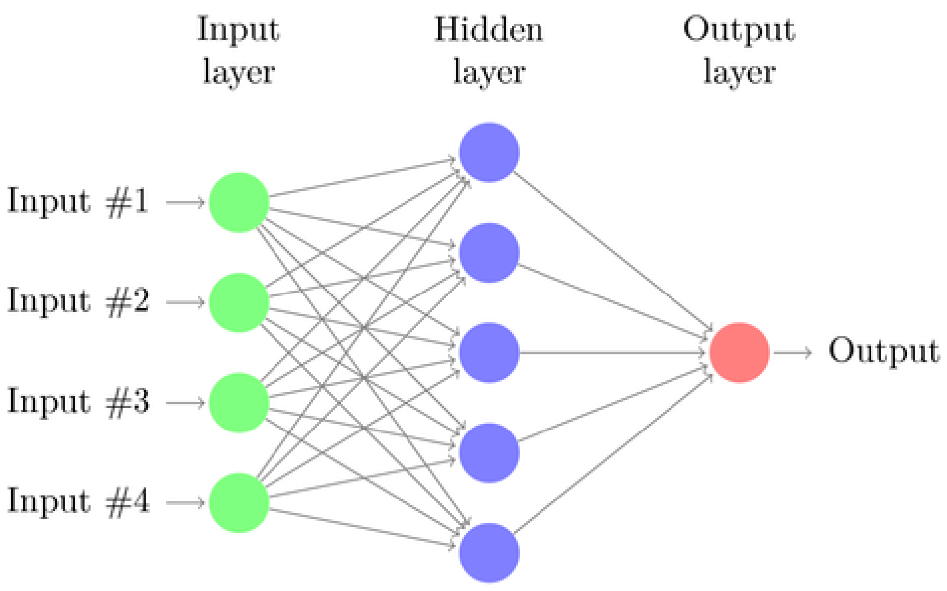

Figure 3: An example neural network

Between two adjacent layers exist connections from the units in the first layer to the units in the second one. Those connections represent nerve paths and can be assigned different weights. Mathematically, those weights are expressed as matrices, and the "thinking process" is modelled as a matrix multiplication.

In the above example, there would be two weight matrices: a 4×5 matrix that models the transitions between the input and the hidden layer, and a 5×1 matrix for the ones between the hidden and the output layer. When the input units get activated with some values, the two multiplications are performed and the result is sent to the output unit.

In practice, the input and hidden layers would often also have one additional so-called bias unit that is always initiated with the value 1 and only connected to the layer on its "right". Those units allow the resulting hypothesis function to be shifted by a constant value, which is not possible without them. The introduction of bias would change the weight matrices in the above example to 5×5 and 6×1, respectively.

A network with the above architecture (full connections between adjacent layers and at least one hidden layer) is also called a Multilayer Perceptron (MLP).

Networks that are used for classification have as many output units as there are classes in the problem space. The classification performance is indicated by the output values: The unit corresponding to the correct class should show a value close to 1, and all the other units a value close to 0.

3.2.1 Back-propagation

However, a neural network has to be trained before it can be used for classification. For this purpose, an algorithm called back-propagation is used.

First, the errors that occurred in the output units are determined. Those errors are then passed backwards, first to the hidden layer and then to the input layer. This way, the amount of “blame” that every connection holds for the error, can be determined. In the last step, a gradient descent search is performed to find better weights that minimize the error.

When this has happened for every set of data in the training set, one training epoch has passed, and the performance can be evaluated again.

3.3 Musical notation basics

Common Western musical notation is a complex system that is capable of describing countless aspects of any given piece of music; understanding it in its entirety is a field of study in its own right, and any attempt to describe it somewhat exhaustively would go beyond the scope of this thesis. Therefore, only its aspects directly related to the later classification process will be discussed shortly.

The five horizontal lines that are the basis of the notation are called the *staff*. *Notes* represent a tone while *rests* represent the absence thereof.

The frequency of a tone, i.e. how high or low it sounds, is its *pitch*. Pitch is indicated by a note’s vertical placement on the staff: Spatially higher notes sound higher than spatially lower notes.

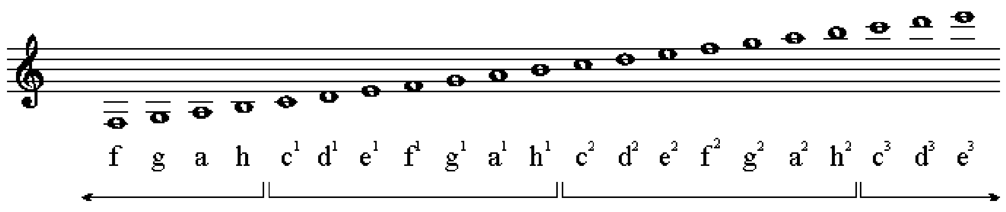

Figure 4: Notes with different pitches

There are different systems of naming the tones. In this thesis, we used the one most common in German literature where the *Middle C* (261.6 Hz) is designated by the name c^1. This is also known as the *Helmholtz designation*. However, it should be noted that in English scientific usage another designation is more common, by which the same tone would be called C_4. This is also the designation used in MusicXML.

The central time unit on the staff is a *bar*. The duration of a note is expressed as a ratio in regard to a bar: *Whole* notes sound for an entire bar, *half* notes sound half as long and so on. Symbolically, the duration of a note is indicated by its appearance as shown in Figure 5.

Finally, the different rest durations are shown in Figure 6.

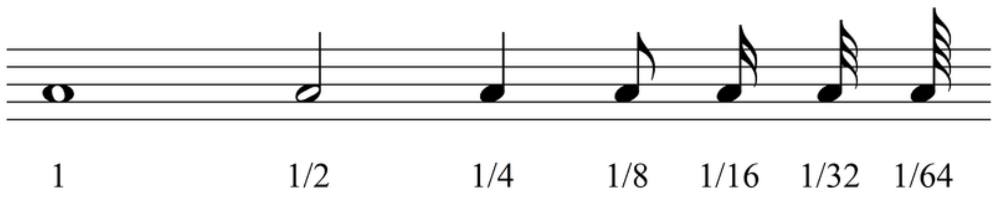

Figure 5: Note durations expressed as fractions

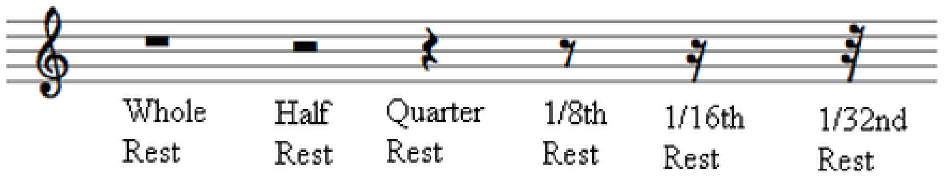

Figure 6: Rest durations

4 Used technologies

4.1 PyBrain

PyBrain is a modular machine learning library for Python. The name stands for ***Py**thon-**B**ased **R**einforcement Learning, **A**rtificial **I**ntelligence and **N**eural Network Library.* The original research that led to its creation can be found in [15].

According to their Web site, *"PyBrain aims to be a very easy-to-use modular library that can be used by entry-level students but still offers the flexibility and algorithms for state-of-the-art research."* [16]

It was chosen due to its good integration with the SciPy/NumPy libraries and its convenient functions that promise the creation of powerful neural networks with little configuration.

4.2 OpenCV

OpenCV *(Open Source Computer Vision)* is a widely used graphics library written in C/C++. [14]

In this project its Python interface was used for the image processing tasks.

4.3 MusicXML

MusicXML is an XML-based standard for exchanging written sheet music that this project uses as its output format for the classified notes. It is a proprietary format owned by *MakeMusic, Inc.*, but it is an open standard under Public Licence that can be freely used.

According to their Web site, more than 200 applications support MusicXML, including *Finale* and *Sibelius*, two of the most widely-used professional score-writing programs. [11]

5 Training data

5.1 Available databases

There are many databases of image classification data openly available online that one can use to train and test one's algorithms. One of the most frequently used ones is the *MNIST database of handwritten digits* [10], described in detail in [4]. It comprises of a training set of 60.000 examples and a test set of 10.000 examples, each showing a handwritten digit (from 0 to 9) as a 28×28-pixel image, and sets a benchmark for many studies concerning image processing and recognition.

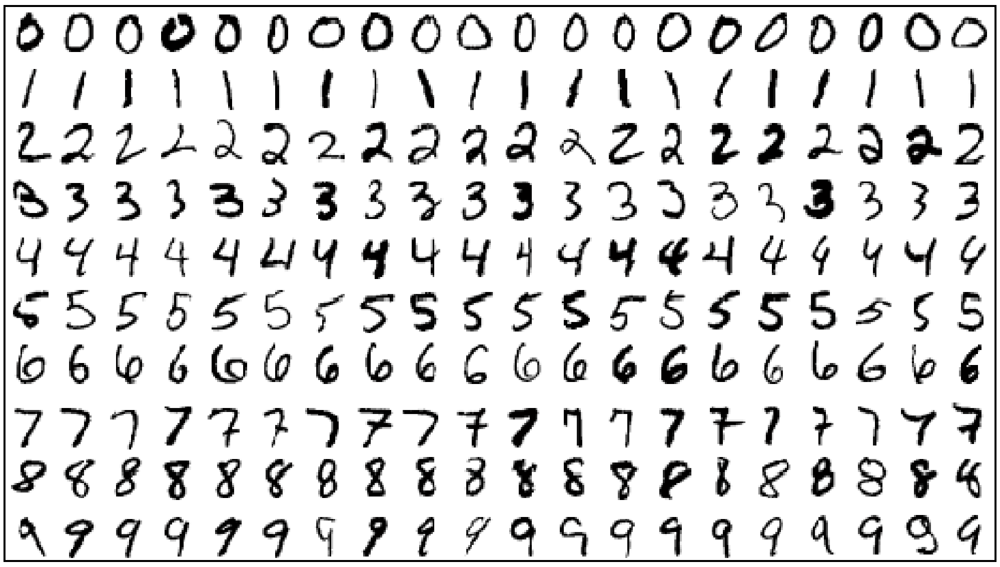

Figure 7: Examples of digits in the MNIST database

However, while there is much data available for tasks such as digit or letter recognition, examples of musical notation, particularly labelled samples of single notes and rests, are much harder to come by. The *CVC-MUSICMA database* [5] for example contains 1.000 sheets of handwritten music scores, but its applications are rather narrow: The data concentrates on the tasks of writer recognition and staff removal; while those are definitely important endeavours in the field of digital sheet music processing, this database – and several more like it – was not very well suited for the task at hand. As a result, the decision was made to collect the needed training data without the use of existing databases.

5.2 Data acquisition

Firstly, a large collection of scanned music sheets had to be found. Luckily, an established Vienna youth choir provided access to their online archive [13]. Vocal sheet music was especially apt for the task as it often consists of one voice line per staff (with one staff each for the four voices Soprano, Alto, Tenor, and Bass) and has generous spacing between the notes in order to fit the lyrics below.

For the next step, the individual notes had to be extracted; this was achieved manually with a screenshot snipping tool. Special care was applied to ensure that all five lines of the staff as well as the entire beam of the note were visible in the resulting picture. An image ratio of approximately 3:5 proved appropriate for this effort.

The individual files differed strongly in regard to their quality: Some had better resolution than others, most probably due to different DPI settings during their scanning processes; also, some files showed strong distortions near the edges of the page or slight rotations throughout. All those "errors" were kept as they were and not rejected from the process, in the hopes of achieving a more robust recognition in the end because of that.

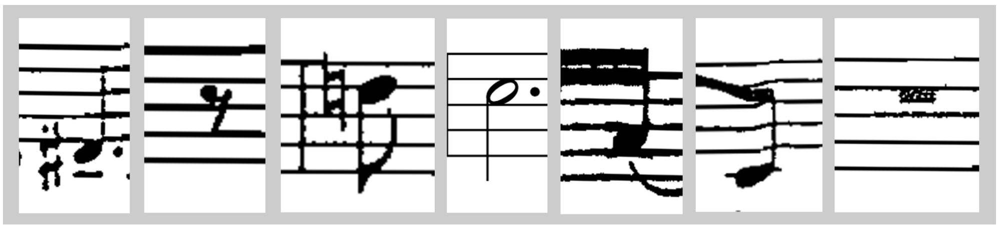

Figure 8: Examples of raw pictures taken from different files

Also, all notes were used in the exact same proportion they appeared in the source files, i.e. no efforts were made to reach a more uniform distribution of symbols with different pitch or duration. As a result, the symbols that would occur most often in a typical sheet of music would also occur most often in the newly built database.

The resulting data set consisted of just under 5.000 pictures. Labelling all those pictures alone would have cost a lot of time while at the same time being quite error-prone. Therefore, a solution had to be found that would "crowd-source" the labelling process, thereby saving time in the long run as well as increasing accuracy.

5.3 Data labelling

The labelling platform resulting from those considerations was build as a Web application [1], using PHP as its backend programming language and a MySQL database for persistent storage.

Users were shown a picture of a note and asked to identify its attributes:

- Is it a note or a rest?
- What is its duration?
(Ranging from *whole* to *sixteenth* with the option to mark it dotted)
- What is its pitch?
(Not applicable for rests; ranging from a to c^3 with the option to mark it with one of the accidentals "natural", "sharp", or "flat")

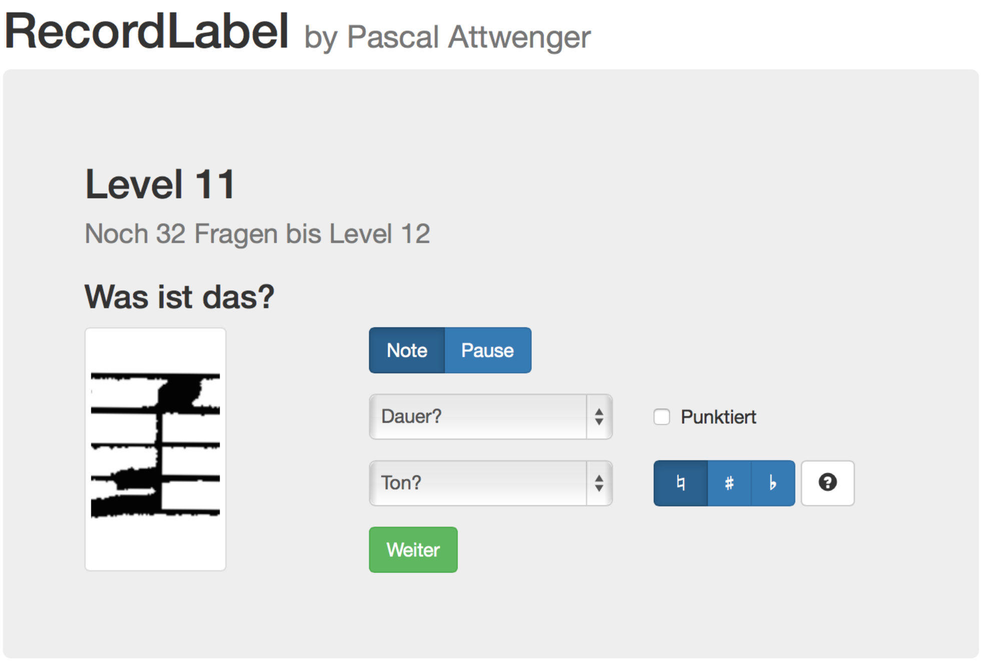

Figure 9: Screenshot of the labelling platform 'RecordLabel'

A "help" button as well as a link to a "frequently asked questions" page (not shown in the screenshot) were introduced in order to clarify the labelling process.

To keep the users engaged in the labelling process, some principles of "gamification" were introduced to the Web site to make the activity more rewarding. Users were enabled to "level up", with a counter showing them the amount of submissions needed for the next level. Upon reaching a new level, users were shown an image of a "prize" (among other things: cookies, coffee and kittens) accompanied by a short textual explanation. One user got as far as level 19, thereby submitting over 500 labels; consequently, this small experiment may be noted down as quite successful.

The labelling platform was distributed online using social media with a special effort to reach musically inclined people. In the first week, over 10.000 submissions from over 300 unique IPs were registered.

5.4 Data export

However, not every one of those submission could be directly translated into a label: The possibility of human error was still quite high, and the amount of mislabelled pictures had to be kept to a minimum. To solve this problem, a simple algorithm was included to determine which submissions were to be exported:

Pictures with only one submission were not considered, as the truth value of the submission could not be determined; pictures with two submissions were exported only if both submissions agreed about every attribute; and pictures with three or more submissions were exported if more than 60% of the submissions agreed about each attribute.

The pseudo-random generator that determined which picture to show to the user was configured to choose pictures with fewer submissions more often, so that most pictures would reach the level of two or more submissions rather quickly.

In the end, 3.782 pictures were eligible to be exported in this manner. The information about the attributes was encoded in the filename, and the file was saved locally for further processing. Information about "dotted-ness" and accidentals was not included here, as it would not be of use in the following steps.

Some examples of the resulting filenames are as follows: (The number in the end is a unique ID and not needed for the classification process.)

```
note-quarter-e1-776.png
note-sixteenth-g2-943.png
rest-whole-4390.png
```

6 Implementation

6.1 Image preprocessing

The one obvious step of preprocessing concerns the image size: The raw images that the labelling platform returns come in many different sizes and ratios. A Python script using OpenCV was run to normalize them. The size of 30×50 pixels was chosen based to some extent on the 28×28 images in *MNIST*, as an image size in the same order of magnitude but slightly bigger to allow for more details.

All images were shrunk down to the appropriate width or height (using OpenCV's INTER_AREA interpolation to keep resizing artefacts to a minimum) and padded with white pixels where needed. As a result, now all images consisted of exactly 1.500 pixels and could be fed into a neural network with this number of input units.

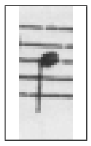

Figure 10: Resized image (30×50 pixels) with visible padding

6.1.1 Distortions

When dealing with image recognition tasks, one way of achieving more robust recognition algorithms is to apply distortions to the images before training the classifier with them. However, those distortions should be similar to the ones that also occur naturally in the data. [17] [3]

As we were dealing with scanned sheet music, three types of distortions were considered:

1. Rotation: When scanning a page, small rotations of the paper are common, and not every scanning software takes care of rectifying this. As a result, notes cut out from such files also show the same degree of rotation.

2. Skewing: Skewed scans can occur for a number of reasons, most notably when the lid of the scanner was not closed properly, for example if the page was scanned from a book. Particularly the edge regions of the page—and consequently, the notes there—are affected by this.

3. Noise: The printed and scanned version of a document will almost always have more noise than its original. In our example, this will mostly be visible as black spots in spaces that should be white or vice versa. Also, the contrast of the scanner could be wrongly configured, resulting in grey areas that should really be white.

All three of those distortions were applied to the images in the training set: Rotation was implemented naively through affine transformation. Skewing was approximated through shearing, also an affine transformation. Finally, noise was added in the form of “salt and pepper” noise with varying degrees of white and black.

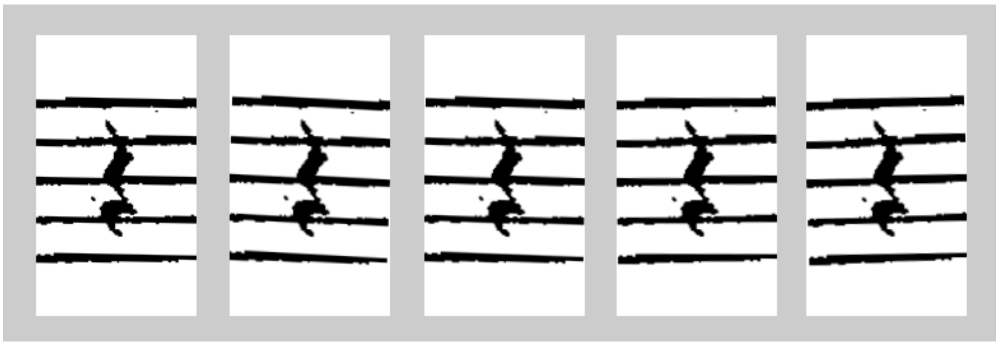

Figure 11: Image transformations: rotation
(1) original
(2) rotated 2 degrees clockwise
(3) rotated 1 degree clockwise
(4) rotated 1 degree counter-clockwise
(5) rotated 2 degrees counter-clockwise

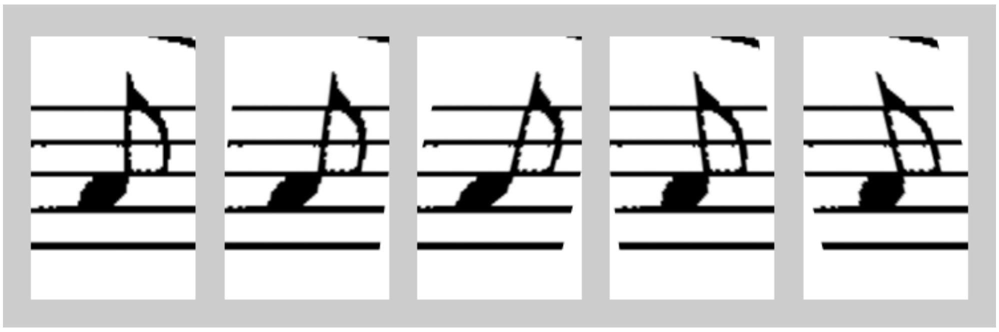

Figure 12: Image transformations: shearing
(1) original
(2) sheared left, 10%
(3) sheared left, 20%
(4) sheared right, 10%
(5) sheared right, 20%

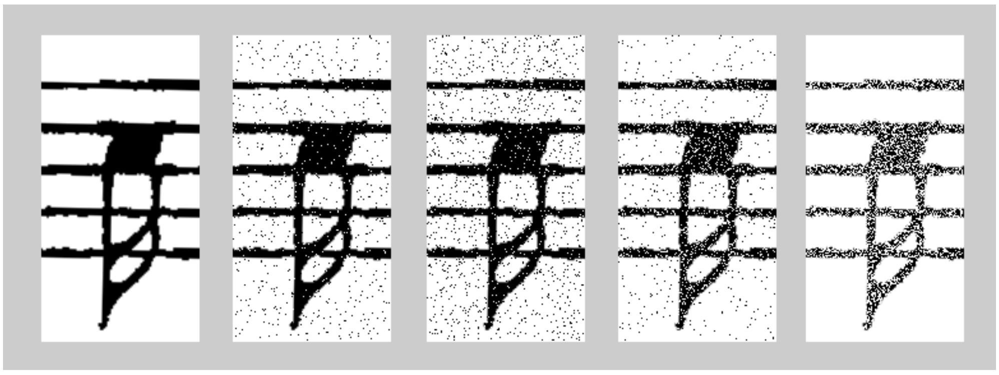

Figure 13: Image transformations: salt and pepper noise
(1) original
(2) 5% noise, 50% salt
(3) 10% noise, 50% salt
(4) 20% noise, 90% salt
(5) 50% noise, 100% salt

No distortions were applied to the images in the test set as this would falsify the resulting test error rates and class accuracies.

The positive results of this preprocessing step can be seen when comparing the classification results in Tables 5 and 6: Both networks were trained under the same conditions, with the exception of the latter not using any distorted images during training; the differences in recognition accuracy are highly significant.

6.2 Building the datasets

The images were imported into Python using OpenCV. The resulting objects were NumPy 50×30 arrays that could easily be flattened to 1500×1 arrays and used to build PyBrain's `ClassificationDataSets`.

70% of the images were used to build the training set, the remaining 30% for the test set. However, the distorted images were added to the training set afterwards, increasing its size thirteenfold.

6.3 Building the networks

The networks were created with PyBrain's convenient `buildNetwork` function, which has default settings that were mostly appropriate. It creates a `FeedForwardNetwork` with three layers: A `LinearLayer` as the input layer and one `SigmoidLayer` for the hidden layer. Only the output layer had be manually set to be a `SoftmaxLayer`, as this is better suited for classification tasks than the default `LinearLayer`. The layers are then connected with a `FullConnection`, which also automatically adds bias units.

6.4 Training the networks

Both the data set and the network itself were then used to set up a `BackpropTrainer`. In order to find the minimum cost function, this trainer uses an algorithm called gradient descent, which calculates the first derivative of the function: a vector that symbolizes the path of steepest descent; following this vector leads to a (local) minimum of the function.

After training the network for as many epochs as specified, it can then be saved persistently as an XML file using PyBrain's `NetworkWriter` in order to classify images as needed or continue training at a later time.

7 Performance

7.1 Counteracting under- and overfitting

The most important parameter for fine-tuning a network's performance is the number of units in its hidden layer: If there are too few hidden units, the network will not be able to adapt well to its training data – it will *underfit* the data. If there are too many hidden units, however, the network might virtually "memorize" its training data, which could lead to a quite low training error; unfortunately, in this scenario it will not be able to recognize data well if it has not been trained on it: it will have a high test set error as it has *overfit* the data.

To find the good "middle-ground", one possible approach would be to model finding the best number of hidden units as an optimization problem: The program could simply try many different possible numbers in order to find the one which leads to the lowest error on the test set. However, this brings with it two new difficulties: Firstly, the process of training a neural network is quite CPU-intensive; trying all different configurations even in a small range of possibilities would take an unreasonable amount of time on the used setup.

Secondly, one is at risk of over-optimizing the problem in regard to the particular test set. This is usually counteracted with the introduction of a third set of pre-labelled data, the cross-validation set. With this approach, the network would be trained on the training set as usual; then, the number of hidden units is determined by the performance over the test set, and only if the performance over the previously unseen cross-validation set is also acceptable can the optimization be considered successful.

For our problem, however, this approach was not feasible: The amount of data was already so small that splitting it one more time would have reduced the quality of the classification too much. Therefore, a fitting number of hidden units was carefully selected for each classification task by eliminating possible numbers from both ends of the under-/overfitting spectrum.

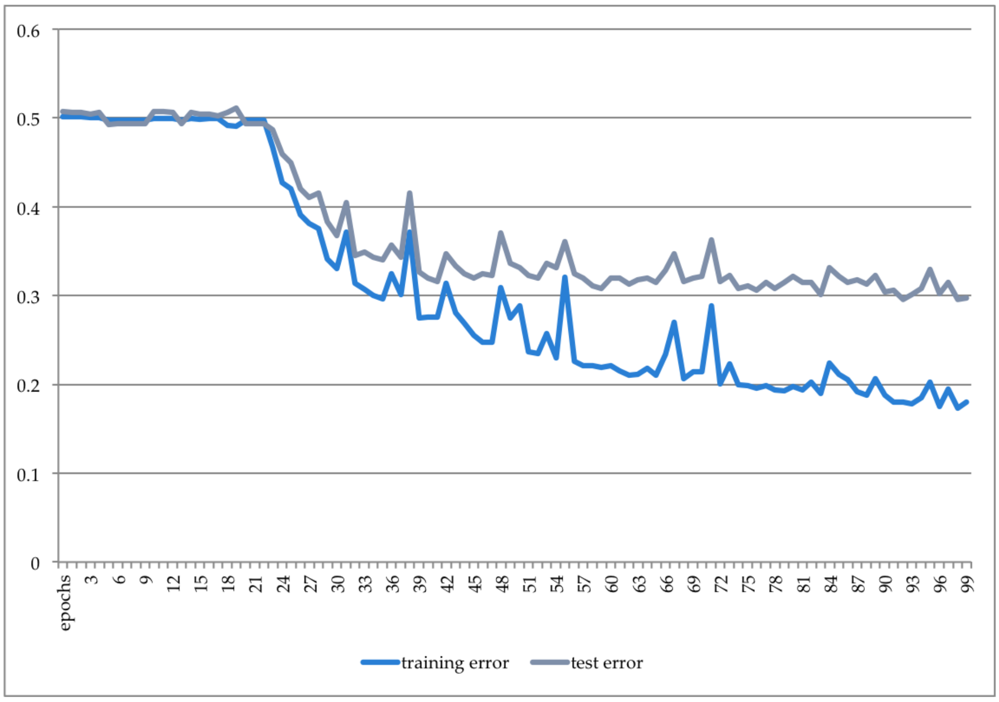

Figure 14: Training curves of an underfit network
(Classifying "quarter or eighth note" with 3 hidden units)

Figures 14, 15 and 16 illustrate this process with training curves showing the progress of error rates over time. The networks were trained over 100 epochs with the goal to differentiate between quarter and eighth notes. (This classification task was chosen specifically for illustration purposes, as both classes have many examples in the training data in a rather balanced ratio. As a result, the test error can be used as a direct indicator of performance without having to take into account sensitivity and specificity. The accuracy for each class is given in Table 5.)

Figure 14 shows an example of underfitting: The network with only three hidden units just was not able to model the specifics of its data sufficiently. Both error rates are consistently very high and do not decrease significantly over time. During the first 25 epochs, the network was not able to find any determining factors at all, with error rates of around 50% basically indicating random guessing. After more training epochs, the training error rate only just managed to get below 20% while the test error rate fluctuated around 30%.

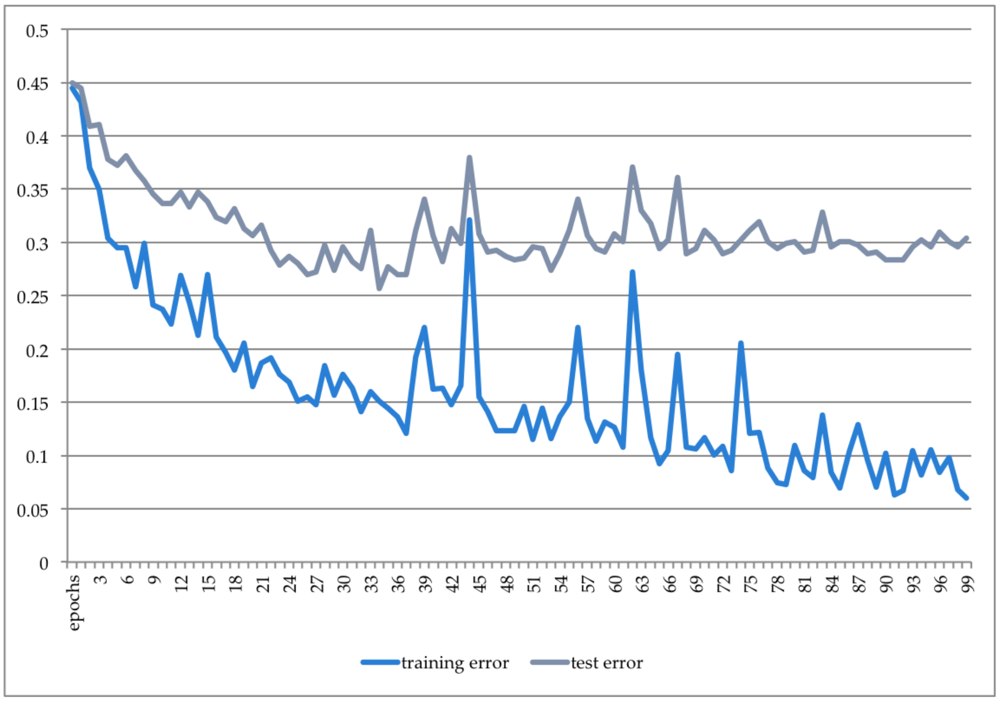

Figure 15: Training curves of an overfit network
(Classifying "quarter or eighth note" with 25 hidden units)

Figure 15 shows a network with 25 hidden units that overfits its data: While the training error rate continues to diminish over time, the test error stays at a high level of about 30%, indicating the aforementioned "memorization" instead of recognition. The big fluctuations are also an indicator of a less-than-optimal model.

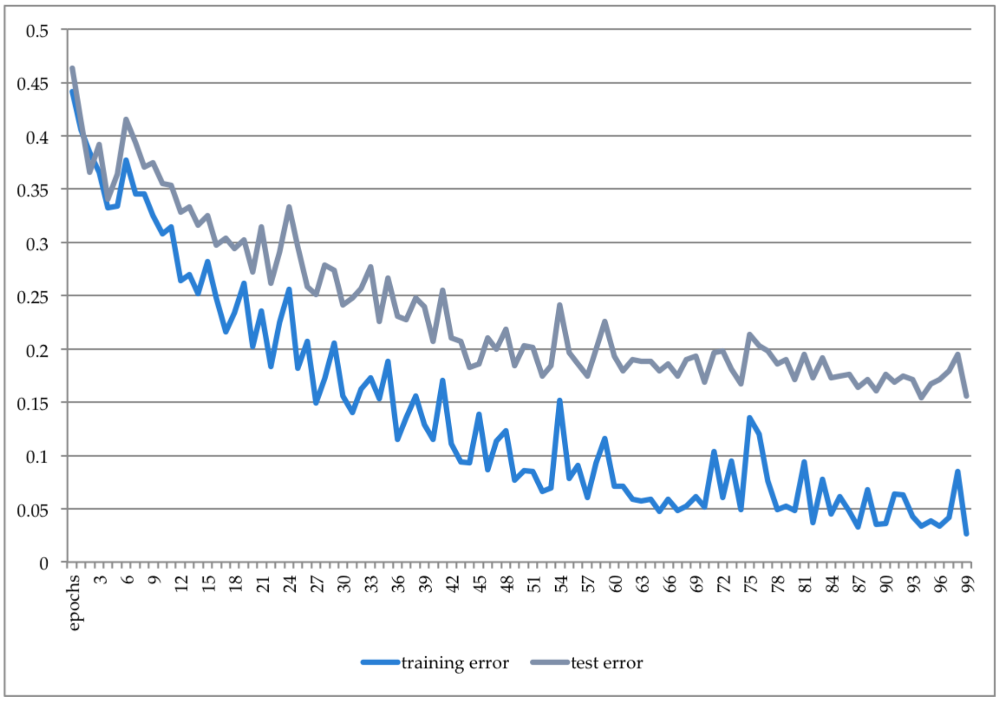

Figure 16: Training curves of a well-performing network
(Classifying "quarter or eighth note" with 10 hidden units)

Finally, Figure 16 shows a network that matches its data well. Both error rates decrease consistently, with a training error rate below 5% and a test error rate below 20%. It stands to reason that significantly better results could not be reached on this network architecture with the given amount of data.

7.2 Classification results

The following tables show the best results that could be reached on the test sets for each classification task.

Class	# Items	Accuracy
note	974	97.02%
rest	160	86.13%
# epochs trained:		120
# hidden units:		10

Table 1: Classification performance: “note or rest”

The fist task was to differentiate between notes and rests so that the appropriate further classifications could then be chosen. This classifier worked fairly well; it had the entire set of data at its disposal and only two different classes to identify. The only problem was that there were significantly more notes than rests in the data, so naturally the classification worked better on notes than it did on rests.

Still, this was a good first step to base further classification on.

Class	# Items	Accuracy
whole	9	77.77%
half	9	44.44%
quarter	71	92.96%
eighth	73	87.67%
sixteenth	0	0.00%
# epochs trained:		600
# hidden units:		12

Table 2: Classification performance: “rest duration”

With the classification of rest duration we see the first problems of the unbalanced data: While the results on quarter and eighth notes were quite satisfactory, the other classes had really little data to train on. The differentiation between whole and half rests was very hit-or-miss; those two symbols look quite similar, and with the small amount of training data, the classifier could not find enough differences between them and basically guessed “whole” most of the time.

The set of sixteenth rests was so small that is basically led to non-performance in this class. (In the above example there was not even a single sixteenth rest in the test data, so performance could not be determined at all.)

It should be noted, however, that out of all the classification tasks, this one had the smallest amount of training data, which put its results into perspective.

Class	**# Items**	**Accuracy**
whole	14	7.14%
half	128	19.53%
quarter	344	72.09%
eighth	403	72.46%
sixteenth	79	24.05%
# epochs trained:		120
# hidden units:		75

Table 3: Classification performance: "note duration"

With the classification of note duration we again see the results of unbalanced training data. There were proportionately more halves and sixteenths than in the rest set, so the results are a bit more balanced here; but the results in the smallest class, wholes, was rather bad nonetheless.

Class	**# Items**	**Accuracy**
a	8	87.50%
h	21	42.86%
c1	44	79.55%
d1	84	70.24%
e1	115	68.70%
f1	114	75.44%
g1	107	75.70%
a1	143	90.21%
h1	103	75.73%
c2	82	78.05%
d2	70	52.86%
e2	43	62.79%
f2	21	33.33%
g2	8	12.50%
a2	4	0.00%
h2	3	0.00%
c3	1	0.00%
# epochs trained:		150
# hidden units:		50

Table 4: Classification performance: "pitch"

Determining pitch was arguably the hardest task of all, as there were 17 distinct classes to identify. With that in mind, the classification worked surprisingly well, especially when considering the “worst case” strategy of random guessing, which would not even lead to 6% accuracy.

Still, we notice once again that the classification works better on the big classes and badly to not at all on the small ones.

Class	**# Items**	**Accuracy**
quarter	284	85.21%
eighth	301	83.72%
# epochs trained:		100
# hidden units:		10

Table 5: Classification performance: “quarter or eighth note”

The test results for the “quarter vs. eighth” classification are shown for the sake of completeness, as they were not used for practical classification but only for illustrating learning curves and the benefits of image distortions.

Class	**# Items**	**Accuracy**
quarter	284	70.77%
eighth	301	59.14%
# epochs trained:		100
# hidden units:		10

Table 6: Classification performance: “quarter or eighth note” (no image distortions)

8 Future work

Probably the most important aspect to focus on in order to achieve better results in the future is data acquisition: According to Peter Norvig, a leading mind in the field of machine learning, there exists a “data threshold” that makes many algorithms perform poorly when applied to thousands of training examples but quite well when applied to millions [7].

While we do not believe that a training set in the millions would be necessary for good results on the note recognition task described in this paper, a—say—tenfold increase of data, coupled with a more uniform distribution of recognition classes could lead to significant improvements.

One could then build more complex networks with more hidden units, multiple hidden layers, or other network architectures such as convoluted NNs. The use of entirely different machine learning techniques like support vector machines would also have to be considered.

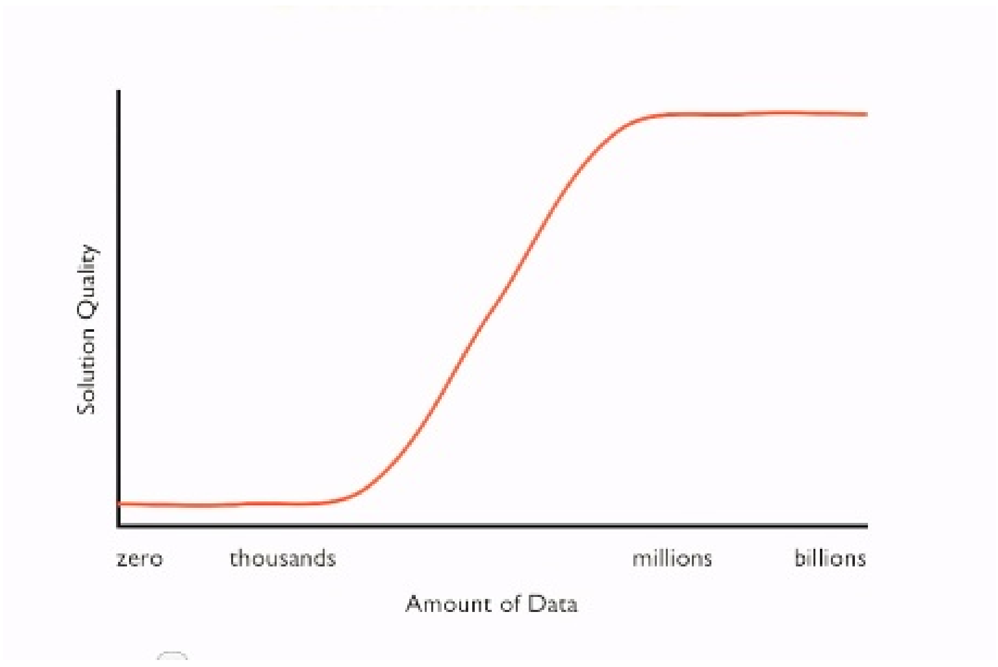

Figure 17: Data threshold as postulated by P. Norvig

Once the error rates have been brought down to an acceptable level, an obvious next step would be to build a more sophisticated OCR pipeline with sliding windows, which could not only classify notes but also find the position of the staffs on a sheet of music, identify the boundaries between single elements and only then classify those symbols with the techniques we have used here. Such a system could implement more intelligent algorithms, for example to ensure that all the bars amount to the same length or to find the clef and key signature at the beginning of each line and use this knowledge to identify pitch more correctly.

Of course, we have only considered a small subset of musical notation so far. For a system to be used in the real world, many more types of notation would have to be considered: accentuation marks like *staccato* and *tenuto*; dynamics both in textual *(forte/piano)* and symbolic *(crescendo/diminuendo)* form; slurs and ties, etc. The list goes on and on, and a complete understanding of musical notation becomes a really difficult endeavour at the latest when one considers old or non-Western forms of notation.

Last but not least, it would be a worthwhile effort to not only try to read the notes but also the lyrics that accompany them, bridging the gap to text recognition.

9 Conclusion

At the end of this thesis we could find that neural networks are definitely an approach worth pursuing when dealing with image classification tasks like recognizing sheet music: The classifiers lead to acceptable results in most recognition tasks, while using only basic image preprocessing and very simple network architectures.

We found that data collection with the use of a self-made online labelling platform is quite possible, and that this is an area definitely worth spending some more time and energy on in order to collect even more. Additionally, we have come to understand that building a large data set where the interesting classes are rather evenly distributed will improve the later classification results significantly.

On the technical side, we found Python to be a very adequate environment for such machine learning problems as it is quite simple to get working solutions in a short amount of time. PyBrain, the library used for the creation of the neural networks, proved to be easy to learn and use for basic tasks; however, there were quite a few unfixed bugs that had to be worked around, and the ongoing work on the project seems to have come to a halt. That, coupled with an online documentation that is incomplete where advanced functions are concerned and sometimes even a bit out of date, lead us to the opinion that the library cannot be fully recommended for future use at this point in time.

One last remark concerns the training process of the neural networks: This turned out to be more time-consuming and CPU-intensive than expected; if one were to follow the path of machine learning further, it would definitely be a worthwhile effort to find more efficient software solutions. A C/C++ based implementation would strip away a lot of overhead that comes with a programming language like Python and would lead to better results in a shorter amount of time.

References

[1] Attwenger, P.: RecordLabel (accessed June 27, 2015), http://homepage.univie.ac.at/a1200595/recordlabel/

[2] Bainbridge, D., Bell, T.: The challenge of optical music recognition. Computers and the Humanities 35(2), 95–121 (2001), http://dx.doi.org/10.1023/A:1002485918032

[3] Dalitz, C., Droettboom, M., Pranzas, B., Fujinaga, I.: A comparative study of staff removal algorithms. IEEE Transactions on Pattern Analysis and Machine Intelligence 30(5), 753–766 (May 2008), http://dx.doi.org/10.1109/TPAMI.2007.70749

[4] Fornés, A., Dutta, A., Gordo, A., Lladós, J.: CVC-MUSCIMA: a ground truth of handwritten music score images for writer identification and staff removal. International Journal on Document Analysis and Recognition (IJDAR) 15(3), 243–251 (Jun 2011), http://dx.doi.org/10.1007/s10032-011-0168-2

[5] Fornés, A., Dutta, A., Gordo, A., Lladós, J.: The CVC-MUSCIMA Database (accessed June 27, 2015), http://www.cvc.uab.es/cvcmuscima/index_database.html

[6] Galway, L., Charles, D., Black, M.: Machine learning in digital games: A survey. Artif. Intell. Rev. 29(2), 123–161 (Apr 2008), http://dx.doi.org/10.1007/s10462-009-9112-y

[7] Halevy, A., Norvig, P., Pereira, F.: The unreasonable effectiveness of data. IEEE Intelligent Systems 24(2), 8–12 (Mar 2009), http://dx.doi.org/10.1109/MIS.2009.36

[8] Hurst, E., Ray, K., Wilson, C., Yehle, T.: Analyzing Sheet Music (Jan 2014; accessed June 29, 2015), http://www.eng.utah.edu/~kray/code_sample/sheet_music/analyzing_sheet_music.pdf

[9] Kassler, M., Pruslin, D.H., Prerau, D.S.: Optical character-recognition of printed music: A review of two dissertations. Perspectives of New Music 11(1), 250 (1972), http://dx.doi.org/10.2307/832471

[10] LeCun, Y., Cortes, C., Burges, C.J.: The MNIST database of handwritten digits (accessed June 27, 2015), http://yann.lecun.com/exdb/mnist/

[11] MakeMusic: MusicXML (accessed June 29, 2015), http://www.musicxml.com

[12] Mitchell, T.M.: Machine Learning. McGraw-Hill Education (1997)

[13] Neue Wiener Stimmen: Der Jugendchor in Wien – eine Initiative der Jeunesse (accessed June 27, 2015), http://www.neuewienerstimmen.at

[14] OpenCV: Open Source Computer Vision (accessed June 29, 2015), `http://opencv.org`

[15] PyBrain: Homepage (accessed June 29, 2015), `http://pybrain.org/pages/home`

[16] Schaul, T., Bayer, J., Wierstra, D., Sun, Y., Felder, M., Sehnke, F., Rückstieß, T., Schmidhuber, J.: PyBrain. Journal of Machine Learning Research 11, 743–746 (2010)

[17] Simard, P., Steinkraus, D., Platt, J.: Best practices for convolutional neural networks applied to visual document analysis. In: Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings. IEEE Comput. Soc (2003), `http://dx.doi.org/10.1109/ICDAR.2003.1227801`

[18] YouTube: Rat Brain Robot (accessed June 30, 2015), `http://www.youtube.com/watch?v=1QPiF4-iu6g`