The automatic detection of structures in music.

Marco M. Grond

1 Introduction

Analysis of musical compositions, especially those composed before the current era of contemporary music, have traditionally taken place by analysing and interpreting the score of a composition. However, the current movement in contemporary music is to take all aspects of a musical performance into account, including improvisations, physical properties of the instruments used during the performance and ambient noises that occur within the concert area. Because of this, much information about the piece is not captured in the score, but rather has to be experienced. This causes many difficulties when attempting to analyse a piece of music, and makes it almost impossible for persons who have not experienced a performance to analyse it.

Analysing musical compositions allows for a better understanding of the piece itself. It is often hard to capture exactly what a piece is meant to convey without being physically present for a performance, but through sound recording a lot of information concerning the production can be captured. However, analysing this data can be quite tedious and time consuming, especially if multiple simultaneous recordings are made in an attempt to capture more aspects of the performance.

In this project, I attempted to automate a piece of the analytical process by identifying structures within a musical piece. By automatically identifying these structures, a musicologist could focus their attention on other aspects of the performance or simply use the extracted information in their analysis. This project focussed on extracting cells and lines from the spectrogram generated from a recording of *Zellen-Linien* by Hans Tutschku [1]. This piece was ideally suited for this task, since the composer intentionally added multiple instances of cells and lines throughout his work.

A motivation of this project follows, along with a brief overview of previous work done into object detection and recognition. Next the data processing and implementation details are explored followed by the results obtained during the course of this project. Finally, a discussion of the results and what was gained from them is presented, followed by the conclusions which were drawn and the current plans for future work.

2 Motivation

In most cases, analysing musical pieces requires an expert to identify certain structures within the piece. Furthermore, many pieces of contemporary music contain components not present on the musical score, requiring the expert to carefully consider recordings of the performance before a detailed analysis can take place. By applying algorithms to automate this process we will be able to gain significant speed increases for the entire process and allow for much larger datasets to be analysed. Automation could also open the door to research and analysis in this field by persons who would have previously been unable to do so, due to limitations on access to an expert who could help identify relevant and interesting structures.

By developing a method to automate this process, the door could also be opened to similar methods which might draw from this research and, with slight alterations, be able to detect and identify most types of structures in different musical compositions. This would be especially useful in ambisonics which could utilize these methods to automatically detect similar structures across multiple audio streams, rather than manually identifying these structures in each individual stream. Using information extracted by the methods proposed in this report and combining that with other computer vision related techniques, one might be able to identify the origin of different structures present in the recordings, to determine the location of sound sources. Although the focus of this project is closely related to the field of musicology, this research is also relevant to immersive technologies, such as audio reproduction in cinemas and virtual reality, both of which have grown significantly over the past couple of years and form a large part of the current entertainment industry.

Another important contribution of this research, is to validate how well object detection and recognition algorithms, which were initially conceived to be applied to mainly image data, generalizes to different data such as sound. These algorithms have been extensively tested on visual data such as images and videos, but relatively little research has gone into their efficacy when applied to different media sources such as sound. The results presented at the end of this report show that these algorithms are indeed viable for different data types and could even be expanded to completely new areas if properly adapted.

3 Related work

Since the infancy of the computer vision field, a lot of research has gone into object detection and recognition. This is an interesting problem in the field, since it relates to one of the most basic human aspects of vision. Solutions to these problems also have many interesting applications, both in

the academic and commercial worlds, including face detection, image classification and context extraction. Although this project does not focus on traditional object recognition, we believe that many of the algorithms developed in pursuit of this goal are still viable for our problem.

One research area in object recognition that has seen significant interest over the past couple of years is neural networks. Over multiple training iterations, using large annotated datasets, these networks learn how to classify input data into a number of possible categories by adjusting the weights of filters and classifiers in different layers of the network. These networks do not require engineered features, but rather attempt to identify features that best describe the data by training on large amounts of raw data and tuning the weights of the feature descriptors in such a way that the final classifications categorize the training data into the correct classes.

One of the first well known implementations of these networks was by LeCun *et. al.* [2], but they only became popular after Krizhevsky *et. al.* showed what they were capable of by placing first in the ImageNet 2012 object recognition competition [3]. These networks were initially implemented as a solution to categorization or recognition problems, but could be further extended to other fields such as image segmentation, which could in turn be extended to the problem of object detection. One implementation of this is by Girshick *et. al.*, who combined the selective search algorithm with neural networks to improve upon the regions found by selective search, and to categorize each region into one of the predetermined classes. This allowed them to successfully segment images, and by extension they were able to find and classify objects within these images [4].

Another algorithm that has shown promise with regards to object detection and recognition is cascading AdaBoost [5], an extension of boosting which was first described by Freund and Schapire [6]. This algorithm, as opposed to neural networks which pass the full image to the network and expects the network to determine relevant features, focusses on using predetermined features in its classifiers. Given a large amount of features, the algorithm attempts to find which of these features best describe the data and combines the weaker classifiers which make use of these features to end up with systems which performs well on classification tasks. Using predetermined features rather than attempting to implement algorithms which learn their own features is more suited when limited annotated data is available, since these features can be engineered to suit a specific problem.

The principle idea behind the cascading AdaBoost algorithm is to combine multiple weak classifiers, trained using boosting, into a cascade of classifiers, where samples are only passed to and categorized by subsequent classifiers in the cascade if all previous classifiers had determined that it belonged to the class being searched for. One of the main benefits of this algorithm, is that it operates by first passing features to classifiers with a low computation time, relatively low accuracy but good false negative rates. A significant speed increase is obtained through this method, since samples that would have previously been categorized by all classifiers are only seen by the more computationally expensive classifiers further down the cascade if they have passed all previous classifiers. The system built by Viola and Jones further computed an "Integral Image", which significantly reduced the computational time for computing features by exploiting the type of features which were chosen, and computing an image from which the features for a specific area could be easily extracted [5].

Another implementation of the cascading AdaBoost algorithm was developed by Chen *et. al.* who implemented classifiers which employed significantly more complex features in the lower levels of their cascade, in an attempt to detect text in images [7]. They adapted the AdaBoost algorithm to only classify data with these more expensive classifiers if all previous weak classifiers had classified the data example as the positive class. This reduced the number of times these stronger classifiers were executed and sped up the entire classification process.

Since a relatively small amount of data is available for this problem, the neural network solutions aren't viable. Instead, the AdaBoost algorithm which has produced impressive results for smaller datasets will be employed to solve this problem.

4 Implementation details and methodology

This section covers the implementation details and methodology of this project. The first part of this section discusses the data processing, datasets which were used in the project as well as the segmentation of data into smaller sets which were used during training and testing. The second part of this section dives into the details of how the classifier was trained to automatically detect structures within the musical piece.

4.1 Data processing and segmentation

A sound recording of *Zellen-Linien*, downloaded from Hans Tutschku's website [1], was used to generate training and testing data for this project. The data downloaded from the website consisted of the changes in air pressure recorded by the microphone during a performance of the piece. The data was sampled at 44.1kHz and contained two audio channels for stereo sound. However, upon further inspection it was discovered that a single audio channel was simply repeated to create the illusion of stereo sound. Due to this fact, only the first audio channel was used in this project.

Since the raw sound data consists only of differences in air pressure measured at certain time intervals, little information is available without preprocessing the data. Spectrograms [8], which show the power spectra of different frequencies plotted over time, were computed for the data. A least square linear predictor with an order of 300 was employed to generate these spectrograms. The raw sound data was carved into windows consisting of 882 data points, while consecutive windows had 50% overlap. This resulted in a spectrogram which had 100 measurements of the power spectra for the selected frequencies per second. The frequencies were also limited to be between 146.8Hz and 3520.0Hz, which correspond to MIDI note 50 and 105. The frequency increases within this range were also not linear, but rather followed the MIDI note scale of the frequencies produced by the different keys on a piano. Although the chosen range of

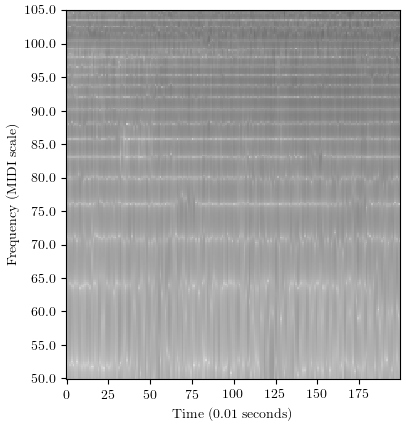


Fig. 1: A spectrogram of a line.

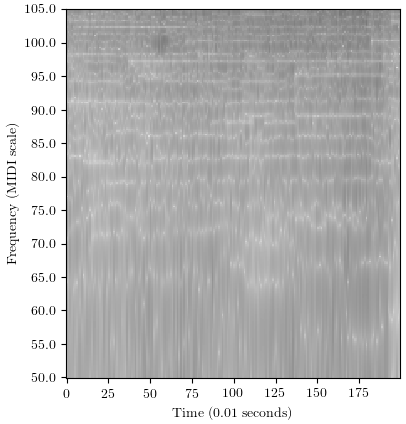


Fig. 2: A spectrogram of a cell.

frequencies spanned over 55 MIDI notes, the power spectra were computed for 220 frequencies, since the MIDI notes were further subdivided into quarters before the frequencies were computed. A spectrogram of a line is presented in Figure 1 and a spectrogram of a cell is shown in Figure 2. The spectrograms of all of the occurences of lines and cells that appear in *Zellen-Linien* are shown in Figures 3 and 4.

Even though the entire *Zellen-Linien* piece is around 20 minutes and 30 seconds, this is still a relatively small amount of data. Furthermore, there are only 20 occurrences of lines and 20 occurrences of cells spread throughout the entire piece. Due to this fact, the decision was made to employ cross-validation to estimate the effectiveness of the final classifiers. In order to do this, the data would have to be divided into folds which would alternate in being used as test or training data. Usually when data is divided into folds, the positive and negative samples for each fold are taken randomly. I did not follow this approach. The reasoning behind this decision is that taking examples from different occurrences of cells or lines throughout the piece would weaken the confidence in the final results. This is because the spectra that are visible for a single line or cell remain somewhat constant over the entire time interval that the line or cell appears in, which would mean that the testing data would be too similar to the training data and reduce the validity of the final results. Instead, all of the positive examples in a single fold would be taken from a continuous line or cell. The negative examples for a fold however, were randomly chosen from all of the available negative examples. The number of negative examples added to a fold was also equal to the number of positive examples in a fold, to maintain a data balance and avoid bias during training.

Since each vertical line of pixels in the spectrogram represents a hundredth of a second, the decision was made to combine ten consecutive pixel lines into a single data example used during the training and testing processes. The power spectra over a line or cell remain somewhat similar, so by combining consecutive power spectra a data example should be more representative of the cell or line that it was taken from. In addition to the dataset that was generated in this manner, another dataset which is simply centred around the origin as well as a dataset which took the average over consecutive time steps rather than combining them, were created. All three of these datasets were used during the training process to compute different classifiers.

4.2 Implementation

Boosting is an ensemble classification technique that combines many weak classifiers into a single stronger classifier. In this project the AdaBoost algorithm [6] was used to train a strong classifier by combining weaker classifiers in a manner that would best minimize the total error on the training data. Any classifier with an accuracy above 50% could be used, however classifiers which require less computational time and are easy to execute are preferred, since the final classifier, which consists of multiple of these weaker classifiers, can become slow if they are too complex. The classifiers are used to categorize the data into two classes, either positive or negative. This means that one set of classifiers are used to identify lines, while a different set of classifiers are used to identify cells, rather than having a single classifier which does both. This was done for simplicity and to reduce the run time of the classifiers.

For this project, two types of classifiers were implemented, both following the two class perceptron classification model [9]. The first type of classifier uses vectors of Haar-wavelets, shown in Figure 5, as the perceptron weights, while the second type of classifier simply uses normalized vectors who's entries are sampled from a normal distribution as the perceptron weights. The idea behind using these types of classifiers is that the resulting weight vectors would be orthogonal to one another. One of the properties of Haar-wavelets is that they are orthogonal, while it has also been shown that vectors acquired through sampling a normal distribution are orthogonal within some tolerance, which is sufficient for this project. Orthogonal weight vectors were chosen since enough of them would span the dimensional space in which the data exists. This would allow the final classifier to be able to distinguish between two separable classes within this space if the correct weak classifiers were chosen.

In theory, the AdaBoost algorithm should be able to choose the classifiers to add to its ensemble from an infinite

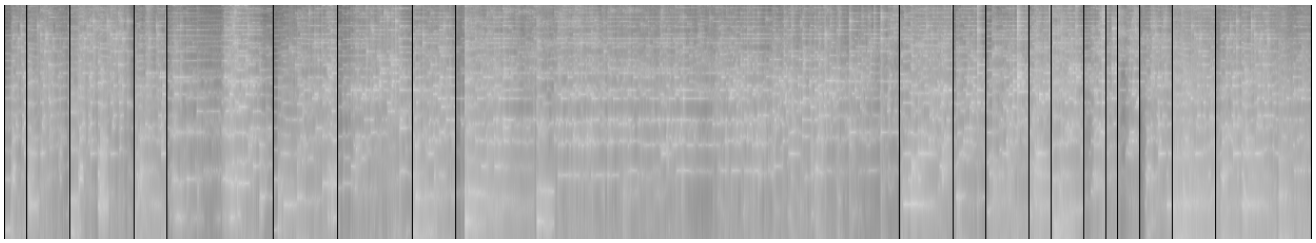


Fig. 3: Spectrogram of all of the different cells in *Zellen-Linien*, separated by black lines.

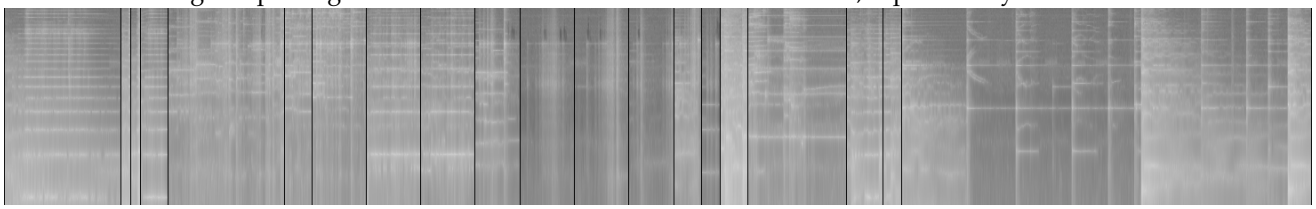


Fig. 4: Spectrogram of all of the different lines in *Zellen-Linien*, separated by black lines.

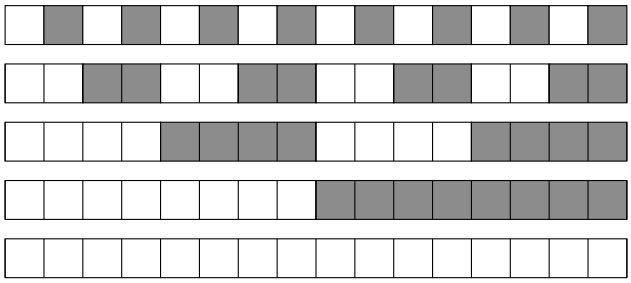


Fig. 5: The weight vectors of Haar-wavelets for training data with length 16. The white blocks have a value of 1 while the grey blocks have a value of -1.

set of classifiers. The larger the number of classifiers from which it can choose however, the slower the training process becomes. To end up with an effective training process that still resulted in an accurate final classifier, the number of weak classifiers was limited by the size of the training vectors. The Haar-wavelets are naturally limited by this factor, while the number of random vectors that were generated was limited to be one quarter of the dimension of the training data. This decision was made after experimentation to determine a good balance between efficiency and accuracy. Since perceptrons were used as classifiers, a threshold needed to be determined. Multiple classifiers could be generated from a single weight vector by assigning each a different threshold. To determine thresholds for a single weight vector, the minimum and maximum values for the entire dataset when passed through that perceptron were computed, after which the minimum-maximum interval would be divided into ten smaller intervals to be used as the thresholds for the perceptron. Additionally, ten more classifiers are computed by inverting the return values for perceptrons with the same weight vectors, resulting in twenty classifiers per weight vector.

The AdaBoost algorithm [6] operates by selecting multiple weak classifiers to combine into a single strong classifier. The weak classifiers are selected based on the previously selected weak classifiers, by selecting the next classifier to be the one that best improves upon the mistakes made by the previous classifiers. Each weak classifier that is added to the ensemble is also assigned a weight to determine its impact on the classification made by the final classifier. This final strong classifier operates by computing the weighted sum of all of the weaker classifiers and applying a sinc-function to the result.

The algorithm iteratively selects classifiers to add to its ensemble. Initially, all of the training examples are assigned equal weights. The algorithm then selects a classifier to add to the ensemble by classifying every example from the training set using each classifier, and choosing the classifier with the lowest error rate with respect to the data weights. The chosen classifier is assigned a weight based on the weights for the different data examples and whether it correctly classified each example. The data weights are also updated depending on whether their corresponding data examples were correctly classified by the selected classifier. If the data example was misclassified, the weight is increased while the weight is decreased if it was classified correctly. This means that the next classifier which is chosen will receive a better error rate if it correctly classifies the data examples which were misclassified by previous classifiers. This entire process is repeated until the predetermined number of iterations has been reached or until no classifier has an error rate better than 0.5. This algorithm is also presented as pseudo code in Algorithm 1 [10].

5 Results

As previously mentioned, multiple datasets were created for both line and cell data. These datasets were divided into folds, with a single fold consisting of an equal number of positive and negative examples. The positive examples in a fold are also taken from a single occurrence of either a line or a cell, while the negative examples are taken randomly from anywhere in the piece of music. The different folds are shown in Figures 6 and 7. Altogether there are 20 folds for both the cell and line data. Cross-validation was employed to determine the accuracy of the system. Rather than training on 19 of these folds and testing on one, the folds were combined into sets of four consecutive folds. This

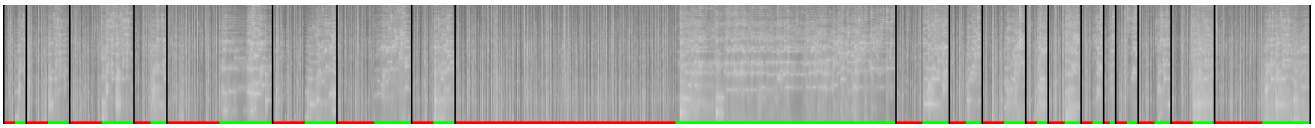


Fig. 6: The training folds for cell data, separated by black lines. The green line denotes examples of cells, while the red line denotes examples of not-cells.

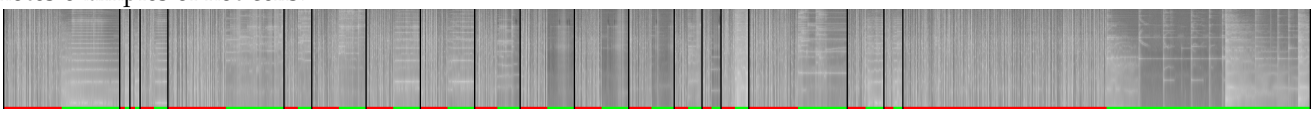


Fig. 7: The training folds for line data, separated by black lines. The green line denotes examples of lines, while the red line denotes examples of not-lines.

Algorithm 1 AdaBoost

Given data $x_1, \ldots, x_m; x_i \in \chi$
With labels $y1, \ldots, y_m; y_i \in \{-1, 1\}$
Initialize weight $D_1(i) = \frac{1}{m}; i = 1, \ldots, m$
while $t <$ Number of iterations T **do**
  Choose a classifier from the collection $h_t \in \mathcal{H}$:

$$h_t = \min_{h_j \in \mathcal{H}} \epsilon_j = \sum_{i=1}^{m} D_t(i) \left[y_i \neq h_j(x_i) \right]$$

  if $\epsilon_t < \frac{1}{2}$ **then**
    Stop execution
  Choose weight for classifier h_t: $\alpha_t = \frac{1}{2} \log \left(\frac{1+r_t}{1-r_t} \right)$
    where $r_t = \sum_{i=1}^{m} D_t(i) h_t(x_i) y_i$
  Update weights: $D_{t+1} = D_t e^{-\alpha_t y_i h_t(x_i)}$
  Normalize the new weights D_{t+1}
Return the strong classifier:

$$H(x) = sign \left(\sum_{t=1}^{T} \alpha_t h_t(x) \right)$$

was done as many of the folds were quite small, since some occurrences of lines or cells lasted for as little as a second. Four out of five of these sets would then be used for training, while the fifth set was used for testing. The testing set would be alternated, so that all five different combinations of training and testing sets were used. 50 classifiers were combined into an ensemble for each dataset, selected out of a pool of around 11 000 weak classifiers. On average it took around 100 seconds to select each one of the classifiers.

The accuracies obtained by running the AdaBoost algorithm with cross-validation on each of the datasets previously discussed are shown in Figures 8, 9 and 10 for the cell data, while the same measures for the line data are shown in Figures 11, 12 and 13. These figures show the accuracies obtained on both the test and training sets for each fold used during cross-validation as well as the average false positive and false negative rates on the testing sets. Multiple lines are drawn for each graph. The coloured, dotted lines in the first two graphs of each figure track the accuracy obtained after each iteration of the AdaBoost algorithm on the different folds during cross-validation, while the solid, black line in these graphs tracks the average across all folds. The last graph in each figure shows the average false positive and false negative rates obtained on the test set across all folds. The measures were plotted across all iterations of the AdaBoost algorithm to show the changes in accuracy of the strong classifier as more weak classifiers are added to it.

TABLE 1: Accuracies as well as false positive and false negative rates obtained on the training and testing sets of different datasets. The first dataset for both lines and cells is the raw dataset, the second is the dataset acquired by averaging the power spectra for all frequencies over ten consecutive time steps and the third is the dataset which is centred around the origin.

Dataset	Train Acc.	Test Acc.	Train False Pos	Train False Neg	Test False Pos	Test False Neg
Cell dataset 1	0.879	0.798	0.130	0.110	0.167	0.235
Cell dataset 2	0.896	0.832	0.105	0.100	0.144	0.189
Cell dataset 3	0.885	0.796	0.126	0.102	0.169	0.238
Line dataset 1	0.824	0.593	0.176	0.173	0.205	0.608
Line dataset 2	0.847	0.580	0.152	0.153	0.165	0.674
Line dataset 3	0.820	0.556	0.190	0.169	0.243	0.643

The graphs show how the accuracies increase and decrease as more classifiers are added to the ensemble. Another observation is that the averaged data seemed to result in the best accuracy for the cell data, rather than the raw dataset or dataset centred around the origin. The differences in these accuracies are somewhat negligible and could even be attributed to the differences in the random classifiers used during the training process. The accuracies obtained on the down sampled data test sets for both line and cell data are higher than for the two other datasets, which could be due to the lower dimensional data being classified more easily than the higher dimensional data of the other two datasets. The final average accuracies as well as the final average false positive and false negative rates for all datasets are presented in Table 1. It is apparent that the system was able to detect cells with an acceptable accuracy, while the lines obtained accuracies not significantly better than random guessing.

6 Discussion

The graphs in Figures 8 through 13 as well as the final accuracies displayed in Table 1 indicate that the system is able to learn how to detect cells effectively, while the line data seems to pose more of a problem. This is somewhat surprising, since the spectrograms of the line data seems to consist of very well defined structures while the cell data seems to be made up of more complex structures. When inspecting any line, there are clear lines visible on the

spectrograms with a high contrast between the frequencies that form part of the line and those that do not. The cells on the other hand seem to contain structures similar to the lines but spread over a wider frequency range and seem to vary quite a bit within the range. Initially, it was believed that the line data would perform well while the cell data would be more difficult to classify due to these observations, however the opposite seems to have been true.

Something that stands out amongst the graphs of the false positive and false negative rates (Figures 8c, 9c, 10c, 11c, 12c and 13c) is that both the false positive and false negative rates of the cell data seem to converge to an optimum, while for the line data the false positive rate improves, but the false negative rate remains somewhat constant or even worsens in some cases. This indicates that the classifiers selected from training are able to detect non-line data, but do not generalize to line data. This is somewhat of a problem, since the aim of the project was to identify the structures, which is exactly what the classifier seems to have problems with when applied to the line data.

Upon further inspection of the line data in Figure 4, it appears as if though these prominent structures differ over the different lines. Although these structures seem more visible and there seems to be higher contrast between the frequencies that form part of the structure and those that do not than for the cell data, the positioning of these structures in the spectrogram seems to be different for most line occurrences. The structures for the cells however, seem weaker when a single cell is inspected, but when all of the cells are compared some similarities become apparent between the cells, especially in the higher frequencies.

The choice of classifiers probably also had an effect on the accuracies obtained on the line data. The perceptron classifier was chosen for the weak classifiers due to its efficiency when classifying data, but does not allow for similar structures to be moved around in the data space. This might have prevented the classifiers from being able to detect the lines, since different occurrences of lines seem to contain similar structures at different frequencies. This seems to indicate that a different choice of classifiers would be better suited to detect lines.

The classifiers obtained reasonably high accuracies when attempting to identify cells. Furthermore, the false positive and false negative rates seem quite close to one another, indicating that the classifiers have learned to separate the two classes rather than just learning how to identify a single class. This seems to indicate that the AdaBoost algorithm is appropriate for this type of data and that with the correct classifier choices acceptable accuracies should be obtainable for line classification as well. Although the initial idea was to train two separate classifiers for lines and cells and then combine them into a single classifier to detect both, that is not feasible unless both the classifiers for lines and cells have acceptable accuracies.

7 Future work and conclusion

This project shows that it is possible to automatically identify structures within a musical piece using a small amount of training data. Although the accuracies obtained on the line data were disappointing, the accuracies obtained on the cell data showed that classifiers could be trained to effectively identify structures within music. The project further showed that the AdaBoost algorithm could be implemented to accomplish this task, but that the classifiers which are used during the AdaBoost training process should be carefully selected depending on the structure that they should identify.

The next steps for this project will be attempting to improve upon the line detection by using different weak classifiers or even by branching out into different classification techniques. Classifiers that will be explored for line classification could take the stability of lines over time into account, or even try to apply a threshold to the frequency images and use the resulting binary images for classification. After improving the line classification, this work will be expanded to other structures in different musical pieces to see how well this type of classification generalizes for this type of data. Finally, this work will shift focus to ambisonics in order to detect and identify structures present in multiple simultaneous audio streams to help with the analysis of spatial music.

References

[1] H. Tutschku, "Zellen linien." http://www.tutschku.com/zellen-linien/, 2007.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition*, June 2014.

[5] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Computer Vision and Pattern Recognition, 2001.*, vol. 1, IEEE, 2001.

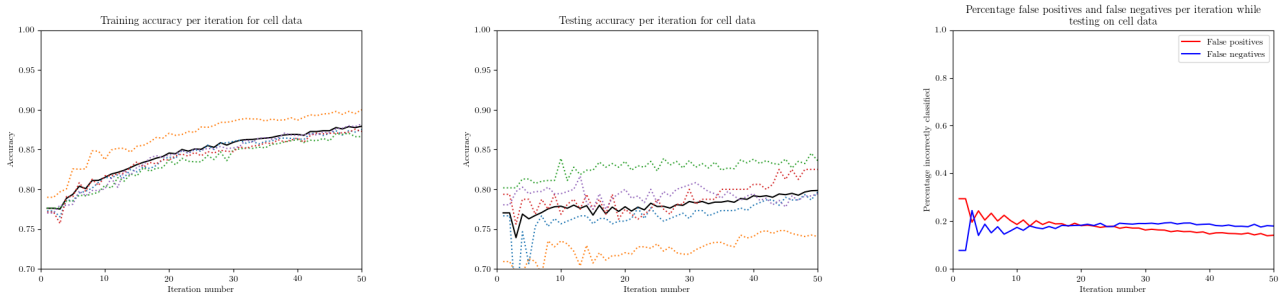
[6] Y. Freund and R. E. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *European Conference on Computational Learning Theory*, pp. 23–37, Springer, 1995.

[7] X. Chen and A. L. Yuille, "Detecting and reading text in natural scenes," in *Computer Vision and Pattern Recognition, 2004.*, vol. 2, IEEE, 2004.

[8] I. Barrodale and R. Erickson, "Algorithms for least-squares linear prediction and maximum entropy spectral analysis - Part I: Theory," *Geophysics*, vol. 45, no. 3, pp. 420–432, 1980.

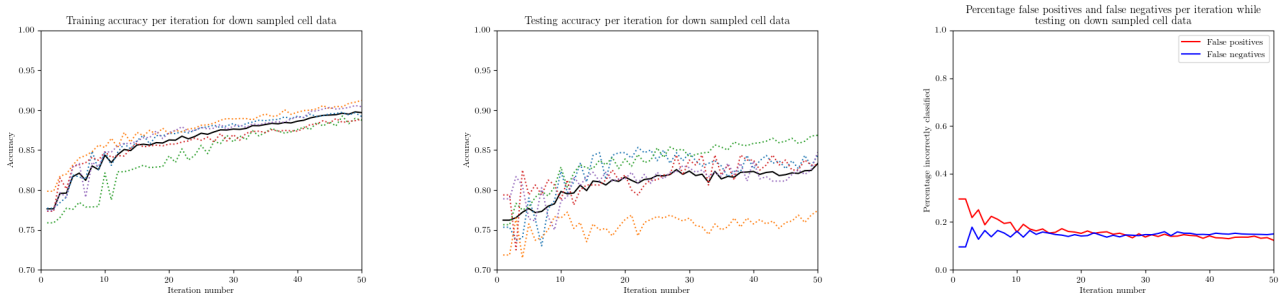
[9] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[10] R. E. Schapire, "Explaining adaboost," in *Empirical inference*, pp. 37–52, Springer, 2013.



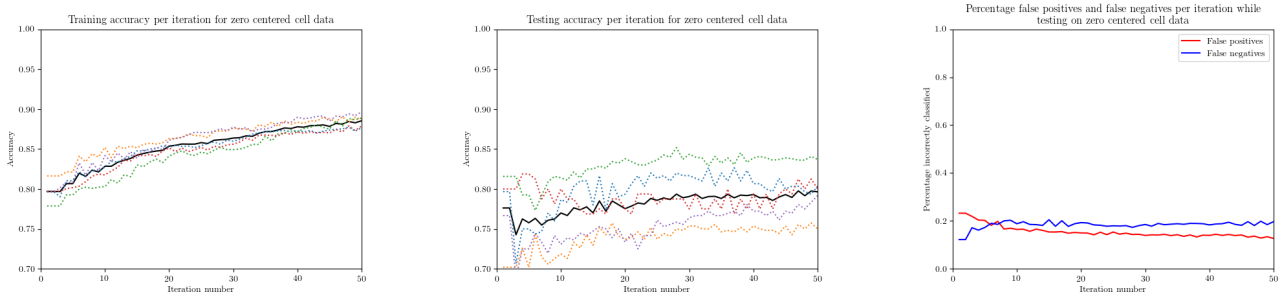
(a) (b) (c)

Fig. 8: The accuracies obtained as well as the false positive and false negative rates after each new classifier was added to the ensemble when training and testing on the cell dataset. The dotted lines in Figures 8a and 8b denote the accuracies obtained by training on different folds while the black line is the average of those accuracies.



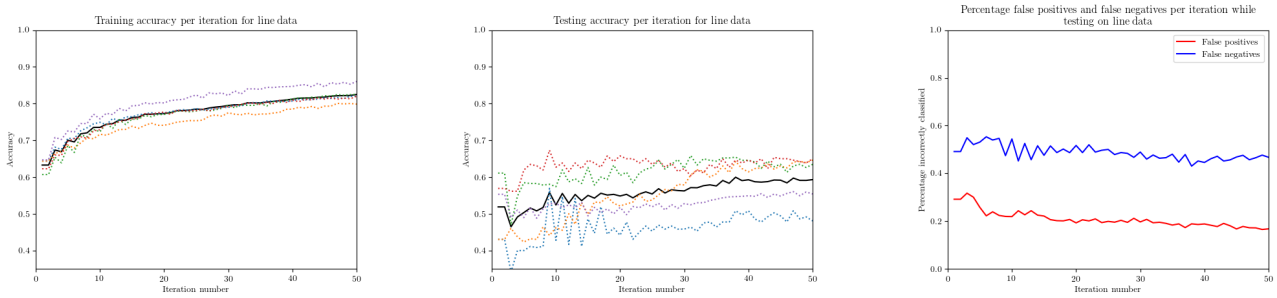
(a) (b) (c)

Fig. 9: The accuracies obtained as well as the false positive and false negative rates after each new classifier was added to the ensemble when training and testing on the cell dataset where the average over ten consecutive time intervals were taken. The dotted lines in Figures 9a and 9b denote the accuracies obtained by training on different folds while the black line is the average of those accuracies.



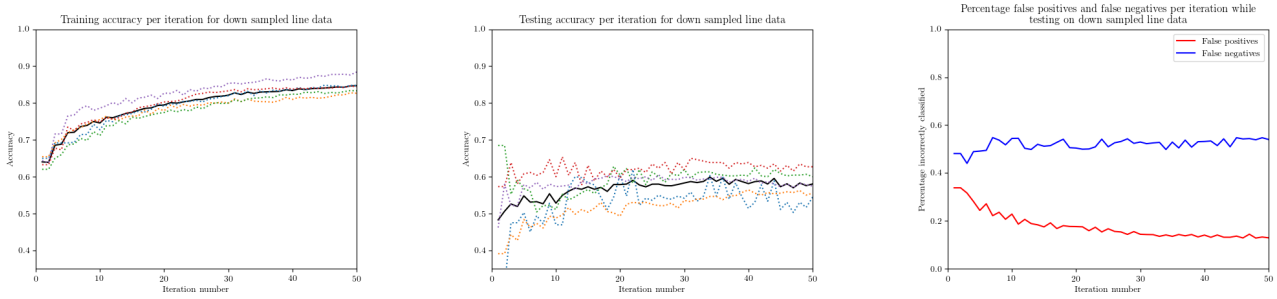
(a) (b) (c)

Fig. 10: The accuracies obtained as well as the false positive and false negative rates after each new classifier was added to the ensemble when training and testing on the cell dataset which was centred around the origin. The dotted lines in Figures 10a and 10b denote the accuracies obtained by training on different folds while the black line is the average of those accuracies.



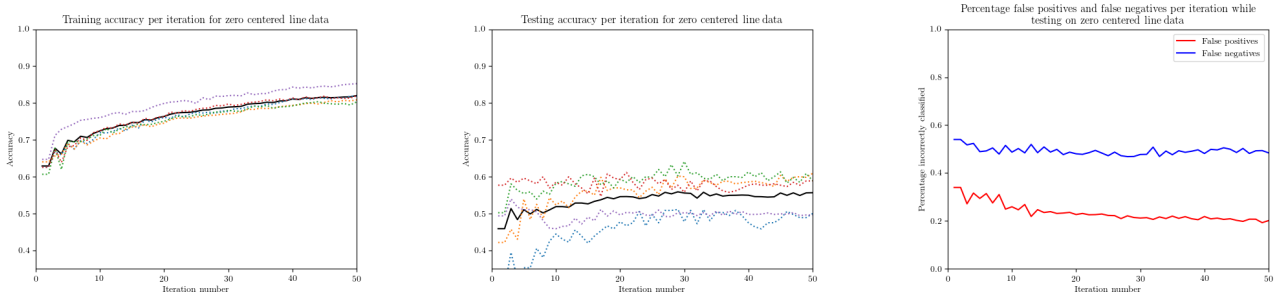
(a) (b) (c)

Fig. 11: The accuracies obtained as well as the false positive and false negative rates after each new classifier was added to the ensemble when training and testing on the line dataset. The dotted lines in Figures 11a and 11b denote the accuracies obtained by training on different folds while the black line is the average of those accuracies.



(a) (b) (c)

Fig. 12: The accuracies obtained as well as the false positive and false negative rates after each new classifier was added to the ensemble when training and testing on the line dataset where the average over ten consecutive time intervals were taken. The dotted lines in Figures 12a and 12b denote the accuracies obtained by training on different folds while the black line is the average of those accuracies.



(a) (b) (c)

Fig. 13: The accuracies obtained as well as the false positive and false negative rates after each new classifier was added to the ensemble when training and testing on the line dataset which was centred around the origin. The dotted lines in Figures 13a and 13b denote the accuracies obtained by training on different folds while the black line is the average of those accuracies.