Wrocław University of Science and Technology
Faculty of Electronics, Photonics and Microsystems

Field of study: **Electronic and Computer Engineering**

ENGINEERING THESIS

Application of machine learning methods in music classification

Krzysztof Hoszowski

Supervisor
Maciej Walczyński, PhD

Consultant
Maurycy Kin, PhD

Keywords: Music Information Retrieval, Supervised Machine Learning, Convolutional Neural Network, Tonality

WROCŁAW 2023

Politechnika Wrocławska
Wydział Elektroniki, Fotoniki i Mikrosystemów

Kierunek: **Inżynieria Elektroniczna i Komputerowa (EAC)**

PRACA DYPLOMOWA
INŻYNIERSKA

Wykorzystanie metod uczenia maszynowego w klasyfikacji utworów muzycznych

Krzysztof Hoszowski

Opiekun pracy
dr Maciej Walczyński

Konsultant
dr inż. Maurycy Kin

Słowa kluczowe: Pozyskiwanie Informacji Muzycznych, Uczenie Maszynowe Nadzorowane, Konwolucyjna Sieć Neuronowa, Tonalność

WROCŁAW 2023

Abstract

Quick and simple filtering of digital content is highly desireable nowadays. The thesis shows how this can be automated with the help of machine learning methods. To do this, an artificial neural network was built that predicts the tonic (or keynote) of a classical music track with 50% accuracy.

Streszczenie

Szybkie i proste filtrowanie mediów cyfrowych jest współcześnie bardzo pożądane. W pracy dyplomowej zademonstrowano, jak ten proces można zautomatyzować przy pomocy metod uczenia maszynowego. W tym celu zbudowano sztuczną sieć neuronową, która przewiduje akord toniczny utworu muzyki klasycznej z dokładnością 50%.

Contents

1. Introduction

With the advent of videos and music on-demand, there is an increasing need for quick ways of finding recordings that are desireable to each person. A systematic solution to this problem involves attaching metadata, or virtual tags, to these recordings. A user can then select which tags should or should not be included, narrowing down the available digital media to a satisfying subset. Traditionally, these tags were added manually by the media's authors, or by individuals. However, this presents a number of problems. There is little standardization for creating metadata. Oftentimes, even if a digital media is professionally tagged, it lacks specific metadata a user is interested in. That encourages the pursuit of efficient ways to classify music that are readily available to individuals. The issue is that this task normally requires human intelligence. A contemporary solution is artificial intelligence. With this technology, it is possible for computers to perform even subjective classification.

1.1. Theory

In music, tonality is the principle of organizing musical compositions around a central note, the tonic[21]. As the composition diverges from the tonic (enters dissonance), it can arouse emotions, such as awe, agitation or wonder. Returning to it brings back a sense of stability and releases tension (consonance)[20]. The sequence of entering consonance from dissonance is called resolution. The tonal system became dominant during the Baroque period and remained that way until around 1900[21], when other approaches were being explored. Tonality can be used as one of the elements of classification of a music track, by indicating the tonic. Those aspiring to become musicians and enthusiasts are likely to benefit from this information.

Tonality exists in various scales. In this paper, the chromatic scale was used. It assumes all notes are in a ring, where each two neighboring notes are exactly a semitone apart.

1.2. Problem Statement

The broad question is how to quickly discriminate between large collections of digital media. This work focuses on classification of music tracks using metadata tags. Specifically, an attempt was made at quantifying the concept of tonality.

1.3. Thesis Objectives

The thesis can be viewed as having two objectives:

1. Demonstrate that machine learning can automate classification of digital media
2. Deliver software that attaches a metadata tag to music files within a certain accuracy.

1.4. Thesis Outline

The work is divided into 5 chapters. This Introduction is followed by Related Work overview. Chapter 3 discusses how I prepared data for training the neural network. Chapter 4 shows how the model is constructed and how it performs. Lastly, the Conclusions and Future prospects.

2. Related Work

In the last three decades, many attempts were made at quantifying acoustic phenomena using computers [1, 14]. Most notably, artificial intelligence has been employed toward this purpose[19, 25]. Methods have progressively become more accurate at this task. Surprisingly though, while a committee of experts always achieves a consensus when assessing a quality of a music track, acoustic characteristics continue to elude formalization. While mostly a matter of art and academic deliberation, there exist useful applications of this process.

Musical features that have been the subject of quantization include timbre [2, 11], key [3], and genre [4, 15]. Instrument identification can be helpful for people without musical training[14]. Also prominently studied quality is valence, or emotional response to listening to music, closely related to the concept of brightness [10]. A number of both open-source and proprietary solutions are available online [7, 22], including in Spotify [24]. Particularly noteworthy is Essentia, a C++ library and tools for audio and music analysis, description and synthesis, released by Dmitry Bogdanov et al. [6]. Examples of Essentia implementations can be found on their website [23].

3. Preparing Input

This chapter gives an overview of how I gained and processed samples for training the artificial intelligence.

3.1. Music Files

After performing a lot of research, I was unable to find any truly standardized and robust databases with classical music in the Internet. While there exist databases with metadata about classical music, they don't contain any recordings themselves, nor do they state the tonality (example). The best resource I could find is Wikimedia Commons, the free media repository. Very conveniently, it contains the Compositions by key category, which allowed me to immediately know the tonality of every song I downloaded. It is also fairly standardized, almost all the songs using Ogg, a multimedia container file format.

At first, I struggled to automate the process of downloading all the files. However, realizing that downloading manually in the browser would take far too long, I continued to search for a solution. I discovered the Download tools page. Unfortunately, none of the solutions allowed to download the Ogg files recursively (ie. all at once from each category). I tried using the Imker tool, since it's capabilities weren't fully documented, and decided to continue with it. With it, I was able to download all files from each Wikimedia Commons subcategory, which made the process drastically faster.

The results weren't perfect: a few songs appeared in two tonalities at the same time. I discarded them completely. Also, the songs are not all equal in volume, length, and recording conditions. Some, for example, contain applause. However, these disadvantages are easily eliminated, or may even contribute to the system's robustness. In the end, I had 1157 songs at my disposal, which amounts to almost 5 days of music.

Another issue was how to sample these songs. From each of them, a sample of the same duration needed to be used, due to the nature of neural networks. The question was, how to choose which part of each song to use? After discussing the issue with my consultant and supervisor, I decided to use the beginning 20 seconds of each song. Traditionally, the tonic appears close to the beginning of a composition. This rule usually holds for the range of Baroque-Romanticism period works being used. In the future, different approaches to solving this problem could be investigated.

3.2. Machine Learning Framework

First, I had to decide what framework to use. I decided to use Anaconda, a Python distribution. It includes libraries for machine learning and Jupyter Notebook, which I used as my integrated development environment. I primarily used the TensorFlow machine learning module[8], with Keras as its high-level application programming interface (API).

This and similar kinds of setups are the *de facto* standard for prototyping and delivering artificial intelligence projects.

Another possibility was to use Google Colaboratory, which is like Jupyter Notebook in the cloud. However, I decided to host the project on my machine. Therefore, I installed Anaconda and necessary machine learning modules. Then I started to familiarize myself with the environment and to look for manuals to help me develop my program. An excellent manual with abundant examples is provided by the TensorFlow team itself. At first, I tested the whole setup by following the Basic classification: Classify images of clothing tutorial. Having confirmed its viability, I focused on the Load and preprocess images guide. This formed the basis of my understanding and strategy.

3.3. Integration of Audio and Python

In order to transform the music files into a format useful to TensorFlow, my supervisor recommended me the *librosa* Python package[9]. I used the latest available version, 0.10.1[18]. The standard approach in computerized music information retrieval (MIR) is to use a song's spectrogram. However, since this neural network was designed specifically to look for tonality, my supervisor recommended using chroma features, or chromagrams. To put it simply, they plot the percentage of each tone from B to C over time. Listing 1 shows an example program to obtain such a chromagram from an audio file.

```python
# Chromagram extraction example
import matplotlib.pyplot as plt
import librosa

# Load the example clip (Brahms - Hungarian Dance #5)
y, sr = librosa.load(librosa.ex('brahms'))

# Separate harmonics and percussives into two waveforms
# Working with pure harmonic signal leads to less noise
y_harmonic, y_percussive = librosa.effects.hpss(y)

# Compute chroma features from the harmonic signal
chromagram = librosa.feature.chroma_cqt(y=y_harmonic, sr=sr)

# Showing first 15 seconds
idx = tuple([slice(None), slice(*list(
    librosa.time_to_frames([0, 15])))])

# Export the chromagram as PNG file
fig = plt.figure()
img = librosa.display.specshow(chromagram[idx],
                    y_axis='chroma', x_axis='time')
plt.savefig('brahms.png', bbox_inches='tight')
```

Listing 1: Chromagram extraction example.

Figure 3.1 shows the output of listing 1.

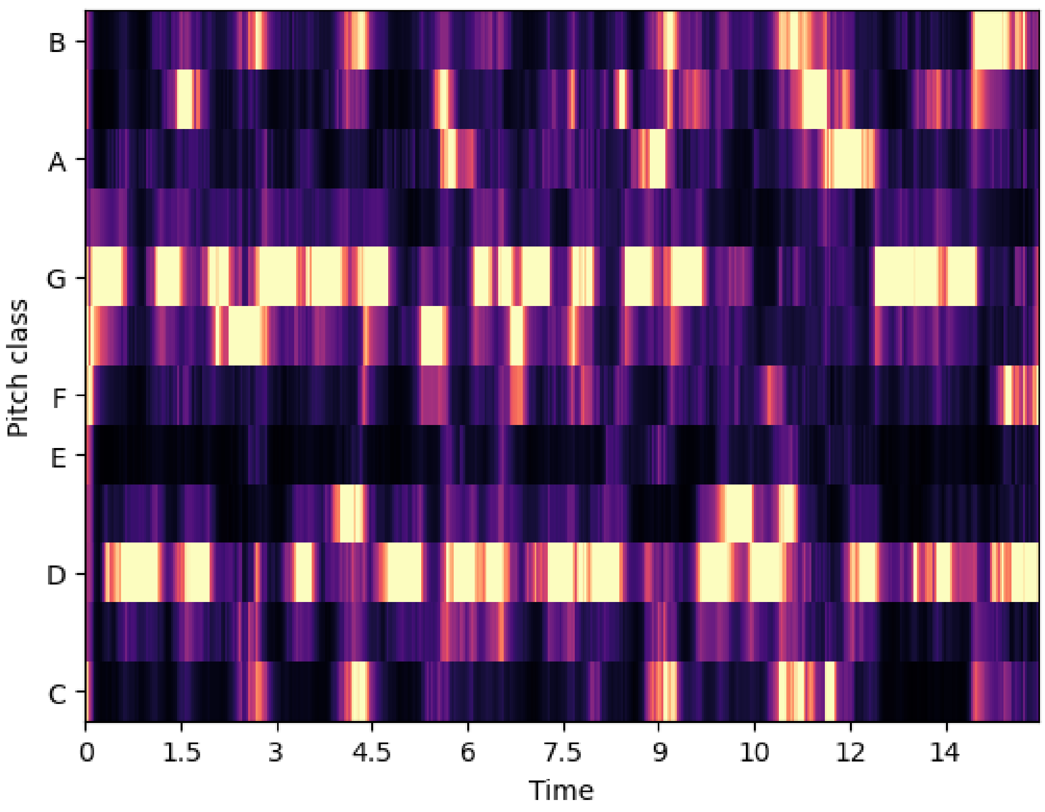

Figure 3.1: Example chroma feature. The unlisted pitch classes are one semitone higher than the previous one, that is C#, D# etc.

The first 20 seconds of each Ogg file were extracted into separate files using the FFmpeg multimedia framework. An exemplary Bash script used is shown in listing 2. Finally, the algorithm used to generate chromagrams from the samples is presented in listing 3.

```
for file in ./major-key/A/*.ogg
    do
    ffmpeg -y -t 20 -i $file -acodec copy
        "${output_directory}/A/$(basename "${file%.*}").ogg"
    done
```

Listing 2: Sampling the music tracks. Example for A major scale. The Bash *basename* variable is used to extract only the filename from the path.

```python
# Generating chromagrams
import matplotlib.pyplot as plt
import librosa

# Removing noise from chromagrams
import numpy as np
import scipy

# Create chromagram database
# Retrieve all data file names
from glob import glob
for i in glob("./data_dir/*"):
    for j in glob(''.join([i, "/*.ogg"])):

        # Load the clip
        y, sr = librosa.load(j)

        # Separate harmonic signal from percussive signal
        y_harmonic = librosa.effects.harmonic(y=y, margin=8)

        # Compute chroma feature from the harmonic signal
        chromagram = librosa.feature.chroma_cqt(y=y_harmonic, sr=sr)

        # Apply extra filtering
        chromagram = np.minimum(chromagram,
                                librosa.decompose.nn_filter(chromagram,
                                                            aggregate=np.median,
                                                            metric='cosine'))
        chromagram = scipy.ndimage.median_filter(chromagram, size=(1, 9))

        # Export the chromagram as PNG file
        fig = plt.figure()
        img = librosa.display.specshow(chromagram)
        plt.savefig(''.join([j, ".png"]), bbox_inches='tight')

        # Free memory
        plt.close(fig)
```

Listing 3: Generating input images.

4. Machine Learning Model

This chapter presents the structure and performance of the neural network.

4.1. Model Summary

Figure 4.1 is a visual representation of the machine learning model. It was produced by Visualkeras[12].

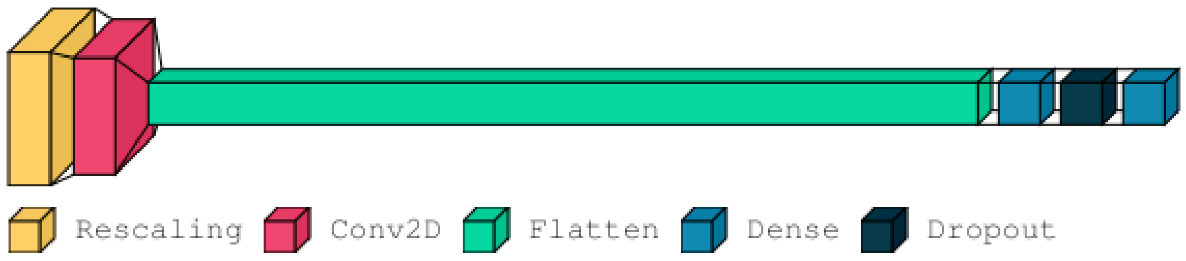

Figure 4.1: Neural network visualization by layers.

Table 4.1 is a textual summary generated by Keras. It provides more details, including numbers of neurons in each layer.

Table 4.1: Neural network summary.

Model: test

Layer (type)	Output Shape	Param #
rescaling (Rescaling)	(None, 16, 16, 3)	0
conv2d (Conv2D)	(None, 14, 14, 32)	896
flatten (Flatten)	(None, 6272)	0
dense (Dense)	(None, 32)	200736
dropout (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 12)	396
======================	==================	=======
Total params: 202028 (789.17 KB)		
Trainable params: 202028 (789.17 KB)		
Non-trainable params: 0 (0.00 Byte)		

The structure of the network was chosen empirically. First, the input images were resized into 16x16 squares. The *Rescaling* layer simply changed these RGB images into grayscale ones, as they are easier to process. Then a convolutional layer is applied to help reveal patterns. Its output is flattened from 2x2 matrix to vector and fed into the *Dense* layer. This layer has a great number of trainable parameters, and it is responsible for the pattern recognition. Then two mechanisms are used to help reduce overfitting. First, a weight regularizer is applied to this layer: `keras.regularizers.l2(0.001)`. Next, a layer of *Dropout* with value 0.5 . Both of these are different ways to prevent the model from pure memorization of the training dataset, instead of learning patterns. Finally, since there are 12 classes (tones) to be recognized, the last layer contains 12 neurons.

4.2. Model Performance

Once the input is ready, the model is generated within seconds. A validation split of 25% is employed. That means that 75% of input images are used for training. The remaining 25% are used at the end of each training cycle to test the performance of the model. The results are presented in figure 4.2.

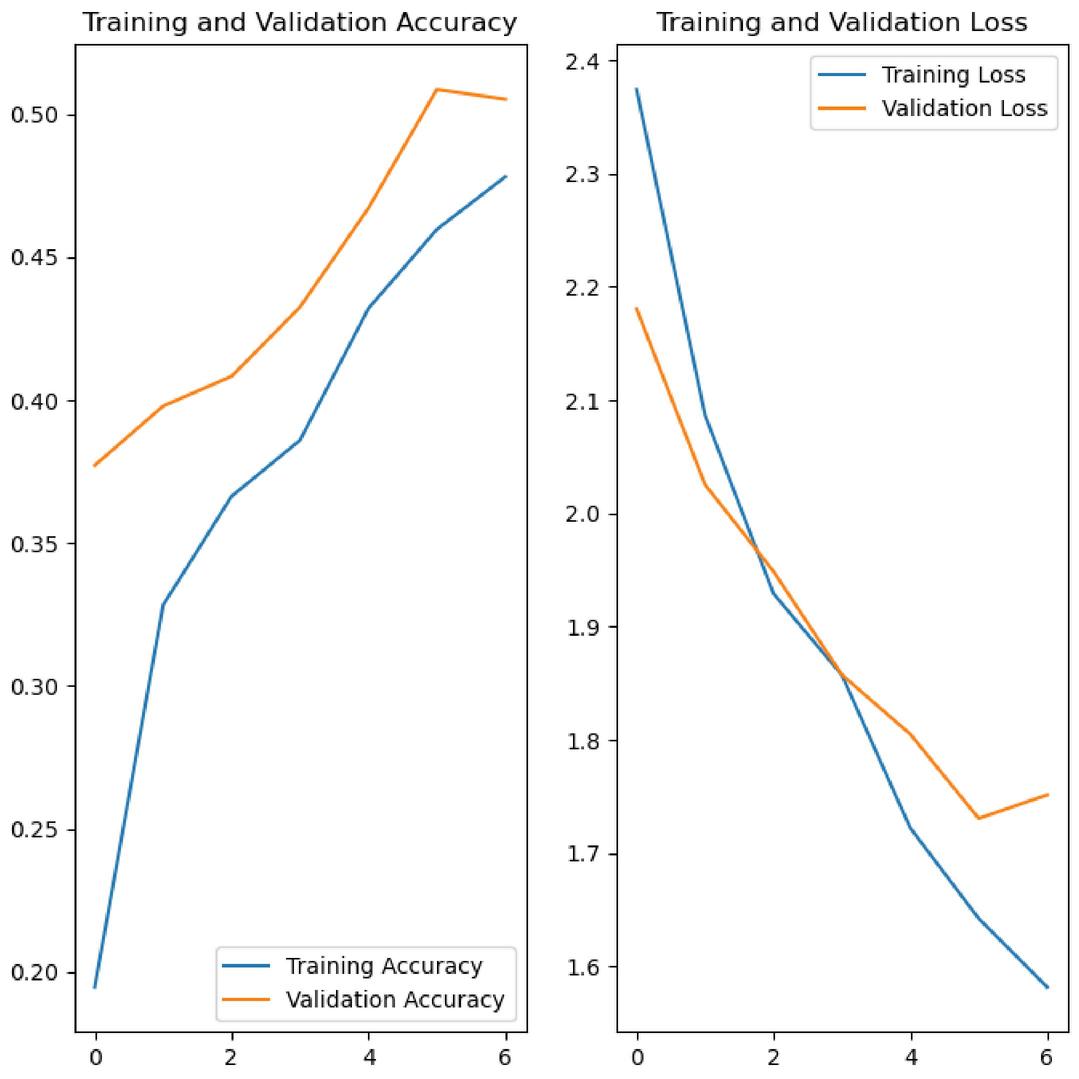

Figure 4.2: Performance of the machine learning model. The x-axis are training epochs (in other words, training iterations). The y-axes are percentage and unity, respectively, for accuracy and loss function.

Validation is important, as it shows more or less the ideal value. Training accuracy shouldn't be higher than it to avoid memorization of training set. If it's smaller, that means more training is possible. Accuracy, in turn, is derived from the loss function used in the training algorithm. As visible, the accuracy saturates at 50%. This is 6 times better than the random guess of $\frac{1}{12}$, or 8.(3)%.

Using scikit-learn, a free software machine learning library[5], I also generated two more metrics. The first is a classification report, which uses a standard format to describe specific qualties of the model. Detailed information is available in the scikit-learn User Guide. The report is presented as table 4.2.

Table 4.2: Model classification report.

Class	Precision	Recall	F-score	Support
A	0.45	0.64	0.53	28
A#	0.44	0.7	0.54	27
B	0.0	0.0	0.0	9
C	0.73	0.54	0.62	56
C#	0.73	0.5	0.59	16
D	0.51	0.7	0.59	47
D#	0.46	0.48	0.47	25
E	0.31	0.4	0.35	10
F	0.52	0.42	0.46	31
F#	0.0	0.0	0.0	6
G	0.38	0.33	0.35	27
G#	0.0	0.0	0.0	7
accuracy			0.51	289
macro avg	0.38	0.39	0.38	289
avg	0.49	0.51	0.49	289

The second metric is the confusion matrix. It's a square matrix in which columns stand for the amount of predictions for each class. Rows indicate what the real class is. It is visualized in figure 4.3.

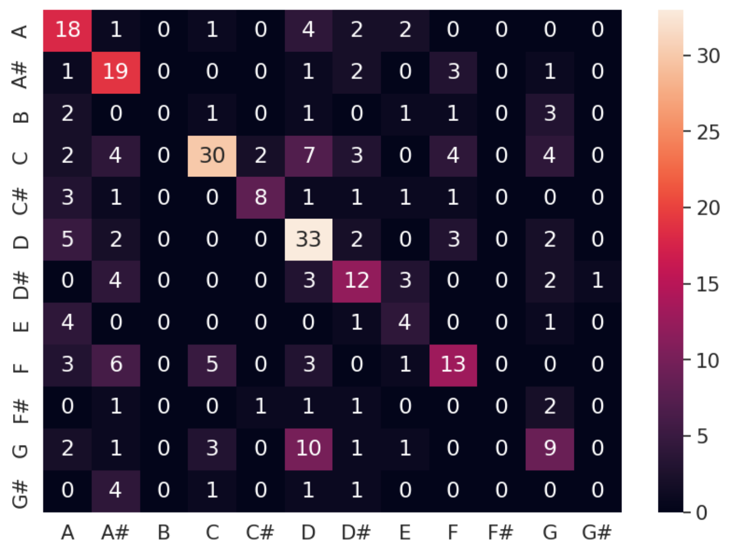

Figure 4.3: Confusion matrix. The main diagonal contains amounts of correct predictions.

As visible in figure 4.3, three tones were not assigned any predictions at all: B, F# and G#. The suspected reason is class imbalance. This problem occurs when "the total number of labels of each class differs significantly"[17]. The following table 4.3 shows how many samples were used for each key.

Table 4.3: Amount of samples for validation by class.

Class	A	A#	B	C	C#	D	D#	E	F	F#	G	G#
Amount	28	27	9	56	16	47	25	10	31	6	27	7

The set consisting of numbers in the *Amount* row has arithmetic mean $x \approx 24.1$ and median $\tilde{x} = 26$. There is clear correlation between the number of samples, and the amount of predictions for each class. Unfortunately, the inequality in the amounts of compositions in each key is a limitation of the dataset being used. Possible solutions include downsampling and upweighting[16]. Downsampling means reducing the amount of samples from other classes. Upweighting means increasing the impact, or weight, that this class will have on neurons, thus substituting for the lack in numbers.

The other false positives are harder to explain. In most instances there are a few false positives a semitone higher or lower than the expected label, which is similar to human mistake. Most of them however are distributed fairly evenly.

4.3. Numerical Experiments

Experiments were performed to assess what is the optimal number of layers and their parameters. As a result, various models were derived, with accuracy in the range of 30-50%.

4.3.1. Layers

Combinations of the following changes were put to the test (hyphen indicates range):

- Removing/Keeping the rescaling layer
- Using 1-3 convolutional layers
- Using *MaxPooling2D* layer after each convolutional layer
- Using 1-3 deep layers
- Trying different amounts of neurons: 16-32, 64, 128, 256.
- Changing the reguralizer penalty within an order of magnitude
- Changing the dropout rate by 0.1.

The max pooling operation cab be used to decrease the size of data, but it didn't improve performance. Not all possible combinations of the above items were tested, but also the superposition principle wasn't assumed to be true. Instead, a few sets of different configurations were tested, and gradually the less performing ones were discarded. This process was repeated a few times based on the insight gained from the previous iteration.

4.3.2. Hyperparameters

Hyperparameters were kept constant throughout the process. They are visible in the following snippet of code (listing 4).

```
model.compile(
  optimizer='adam',
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
```

Listing 4: Model hyperparameters. The optimizer used, Adam, is an extended version of stochastic gradient descent[13].

Furthermore, the amount of epochs was dynamic. This was achieved with early stopping, which was passed as a callback argument to the model when fitting:

```
tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience = 1)
```

This means that the training was automatically stopped once the validation loss increased by a small amount compared to the previous epoch.

4.3.3. Input

Different sizes of input images were tested: 12-32, 64, 256, including rectangular images. However, a square image within the first range proved the best, in the range of 14-18 pixels. The original chromagrams were 515 pixels wide and 389 pixels high, as output by librosa.

5. Conclusions and Future

It was demonstrated that machine learning methods, in particular artificial neural networks, have the potential to automate classification of digital media. Modern computers and programming resources allow rapid development and deployment of such networks. This encourages their widespread use to tasks that normally required human intelligence to perform.

The main problem lowering the accuracy achieved probably lies in the training dataset. Even significant changes to the neural network structure and image size led to only 20 percentage point improvement. I suspect that balancing and expanding it would allow much better results. Alternatively, tuning the hyperparameters could improve the results. These include the learning rate, optimizer, and activation functions. The last determine how each neuron in a layer assigns values based on its input. Common approaches to hyperparameter tuning include grid search (parameter sweep), random search, Bayesian optimization and gradient descent.

A future work could also explore different ways to sample music tracks used for training the model. In addition to chromagrams, other forms of input could be investigated. Also, deeper understanding and experience in the field of artificial intelligence would allow engineering a better neural network structure.

Bibliography

[1] Hendrik Purwins, Benjamin Blankertz, and Klaus Obermayer. "A new method for tracking modulations in tonal music in audio data format". In: *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium* 6 (2000), 270–275 vol.6. URL: https://api.semanticscholar.org/CorpusID:14522098.

[2] Don Lewis and M. J. Larsen. "The Measurement of Timbre". In: *The Journal of the Acoustical Society of America* 8.3_Supplement (June 2005), pp. 207–207. ISSN: 0001-4966. DOI: 10.1121/1.1901995. URL: https://pubs.aip.org/asa/jasa/article-pdf/8/3_Supplement/207/11848619/207_2_online.pdf (visited on 12/09/2023).

[3] Geoffroy Peeters. "Chroma-based estimation of musical key from audio-signal analysis". In: *International Society for Music Information Retrieval Conference*. 2006. URL: https://api.semanticscholar.org/CorpusID:5000729.

[4] Antti J. Eronen. "Signal Processing Methods for Audio Classification and Music Content Analysis". In: 2009. URL: https://api.semanticscholar.org/CorpusID:60298607.

[5] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[6] Dmitry Bogdanov et al. "Open-source library and tools for audio and music analysis, description and synthesis". In: ed. by Britto A., Gouyon F., and Dixon S. Essentia: an audio analysis library for music information retrieval. 14th Conference of the International Society for Music Information Retrieval (ISMIR); 2013 Nov 4-8; Curitiba, Brazil. [place unknown]: ISMIR; 2013. p. 493-8. 2013. URL: https://essentia.upf.edu/ (visited on 12/08/2023).

[7] The Echo Nest. *Plotting Music's Emotional Valence, 1950-2013*. Nov. 2013. URL: https://web.archive.org/web/20170422195736/http://blog.echonest.com/post/66097438564/plotting-musics-emotional-valence-1950-2013 (visited on 12/08/2023).

[8] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[9] Brian McFee et al. "librosa: Audio and Music Signal Analysis in Python". In: *Proceedings of the 14th Python in Science Conference*. Ed. by Kathryn Huff and James Bergstra. 2015, pp. 18–24. DOI: 10.25080/Majora-7b98e3ed-003.

[10] Joydeep Bhattacharya and Job P. Lindsen. "Music for a brighter world: Brightness judgment bias by musical emotion". en. In: *PLoS One* 11.2 (Feb. 2016), e0148959. URL: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0148959 (visited on 12/09/2023).

[11] Andy Pearce, Timothy Brookes, and Russell Mason. "Timbral attributes for sound effect library searching". In: Audio Engineering Society, 2017, pp. 2–2. URL: https://www.audiocommons.org/2018/09/05/timbre-sound.html (visited on 12/09/2023).

[12] Paul Gavrikov. *visualkeras*. https://github.com/paulgavrikov/visualkeras. 2020.

[13] Akash Ajagekar. *Adam*. Dec. 16, 2021. URL: https://optimization.cbe.cornell.edu/index.php?title=Adam (visited on 01/02/2024).

[14] Minz Won, Janne Spijkervet, and Keunwoo Choi. *What is Music Classification?* 2021. URL: https://music-classification.github.io/tutorial/part1_intro/what-is-music-classification.html (visited on 12/09/2023).

[15] Nandkishor Narkhede, Sumit Mathur, and Anand Bhaskar. "Automatic Classification of Music Genre Using SVM". In: *Computer Networks and Inventive Communication Technologies*. Ed. by S. Smys et al. Singapore: Springer Singapore, 2022, pp. 439–449. ISBN: 978-981-16-3728-5. URL: https://link.springer.com/chapter/10.1007/978-981-16-3728-5_33 (visited on 12/09/2023).

[16] Google for Developers. *Machine Learning / Imbalanced Data*. June 9, 2023. URL: https://developers.google.com/machine-learning/data-prep/construct/sampling-splitting/imbalanced-data (visited on 01/02/2024).

[17] Google for Developers. *Machine Learning Glossary / class-imbalanced dataset*. Nov. 14, 2023. URL: https://developers.google.com/machine-learning/glossary#class-imbalanced-dataset (visited on 01/02/2024).

[18] Brian McFee et al. *librosa/librosa: 0.10.1*. Version 0.10.1. Aug. 2023. DOI: 10.5281/zenodo.8252662. URL: https://doi.org/10.5281/zenodo.8252662.

[19] Param Raval. *Solved Music Genre Classification Project using Deep Learning*. Oct. 12, 2023. URL: https://www.projectpro.io/article/music-genre-classification-project-python-code/566 (visited on 12/09/2023).

[20] The Editors of Encyclopaedia Britannica. *consonance and dissonance*. In: *Encyclopedia Britannica*. URL: https://www.britannica.com/art/consonance-music (visited on 01/02/2024).

[21] The Editors of Encyclopaedia Britannica. *tonality*. In: *Encyclopedia Britannica*. URL: https://www.britannica.com/art/tonality (visited on 01/02/2024).

[22] Chosic. *Music Genre Finder | Song Analyzer*. URL: https://www.chosic.com/music-genre-finder/ (visited on 12/09/2023).

[23] Essentia contributors. *Interactive demos/Essentia TensorFlow models*. URL: https://essentia.upf.edu/demos.html (visited on 12/09/2023).

[24] Spotify Community contributors. *Home/Help/Spotify for Developers/Valence as a measure of happiness*. URL: https://community.spotify.com/t5/Spotify-for-Developers/Valence-as-a-measure-of-happiness/td-p/4385221 (visited on 12/08/2023).

[25] Parul Pandey. *Music Genre Classification with Python*. URL: https://towardsdatascience.com/music-genre-classification-with-python-c714d032f0d8 (visited on 12/08/2023).

List of Figures

List of Tables

List of Listings