My New Favourite Song: Using Contractive Autoencoding for Spotify Songs Recommendation

Tim Hehmann
Hasso Plattner Institute

Yannik Schröder
Hasso Plattner Institute

Daniel Lewkowicz
Hasso Plattner Institute

Abstract

In this work, we introduce the usage of a deep neural network autoencoder system for recommending personalized suggestions towards favourable songs based on the online streaming platform Spotify. For this, we adopted a publicly available dataset from Kaggle, called the Spotify Dataset, with 160k+ Tracks released in between 1921 and 2020. Our recommender system considers the favourite song from a user and outputs a tailored feedback recommendation for exploring new favourite songs. Therefore, we build a contractive autoencoder model, which recommends similar songs based on a given input song. We also demonstrate our system by showcasing recommendations for various famous titles.

1 Introduction

Recommender systems, at their basis form, consider the behavior of a user by tracking the interaction of the user with the system, and consequently recommend specific items or actions based on the user's preferences.[1] Nowadays, machine-learning based recommender systems are broadly implemented in different domains and provide personalized recommendations for different kind of users.[1] Moreover, these systems consider personal preferences through various forms of feedback, i.e. implicit and explicit feedback, and are thus capable to provide highly relevant items in order to improve decision making and optimize user experience. For instance, Amazon book recommendation[2], Netflix movie recommendation[3], or Google Play Store mobile application recommendation[4], are well known research and industry applications which are gaining more and more momentum. In this research work, we explore another domain which has already received a lot of attention in the music industry: Songs recommendation for the online streaming platform Spotify. For this, we apply a deep neural network autoencoder recommender system to provide tailored songs recommendation for Spotify users that are based on their current favorite song.

2 Related work

In addition to our work, there has been several different approaches in scientific literature to explore the dimension of personalized songs recommendation for Spotify. Predominantly, research focused on recommender system for automatic music playlist continuation. In the context of the ACM RecSys challenge 2018,[5] several different approaches are explored in order to present new favorable songs to the user's playlist. For the RecSys challenge, the task was to recommend up to 500 tracks that fit the target characteristics of the original playlist, given a playlist of arbitrary length and metadata.[5]. Out of 113 Teams, the best system achieved an R-precision of 0.2241, an NDCG of 0.3946, and an average number of recommended songs clicks of 1.784. For these results, besides using only matrix factorization, as a dominant approach in collaborative filtering (CF), the winning teams took advantages of a multi-stage architecture.[6] Considering the predominantly used two-stage architecture, the first stage model retrieves only a small subset of tracks, while the second stage focuses on re-scoring or re-ranking the output of the first stage model with the goal of accuracy improvement.[5] Two-stage cascaded architecture encompassed matrix factorization, neural networks, and learning to rank, respectively.[6].

Furthermore, Pichl et al.[5] analysed user-curated playlists on Spotify in order to get a deeper understanding of how users organize music. The authors used different clustering methods, i.e. primarily PCA, and subsequently found groups of similar playlists which can be then again integrated into the recommender system using pre- or post-filtering techniques. In another study, Pichl and colleagues present a new dataset combining Spotify and Twitter data in which users posted their songs on Twitter. The authors use a pure collaborative filtering based recommender system and thereby introduce a new approach for more sophisticated and possibly beneficial recommendations.

Moreover, an emerging field in research deals with the ”black-box” problem which depicts that providing individual recommendations still remains unexplained to the users.[6] This ultimately leads to the question, how is it possible to increase the transparency in recommender systems in order to ultimately affect the trust of users in a positive way? Different approaches were already explored to address this issue, e.g. with the help of textual explanations or interactive visualisation.[6] Essentially, increasing knowledge about the provenance of recommendations led to higher user satisfaction and also to higher precision. However, Millecamp et al.[7] investigated that personal characteristics, such as domain knowledge or individual trust propensity, are also crucial factors in this view. In a qualitative and quantitative evaluation study, Millecamp and colleagues found that personal characteristics of participants have significant influence as well as different impacts on the benefits and perception of a recommender system for Spotify songs recommendations.[7]

3 Dataset

For this work, we adopted a publicly available dataset on Kaggle[1], collected from Spotify Web API and called the ”Spotify Dataset 1921-2020, 160k+ Tracks”. Basic statistic data of the Spotify dataset is summarized in Table 1.

Table 1: Basic statistic of the Spotify dataset

Feature	**Ranges**	**Mean**	**Std. Deviation**
Acousticness	0 to 1	0.49	0.38
Danceability	0 to 1	0.54	0.18
Energy	0 to 1	0.49	0.27
Duration in ms	200k to 300k	232k	121k
Instrumentalness	0 to 1	0.16	0.31
Valence	0 to 1	0.53	0.26
Popularity	0 to 100	31.6	21.6
Tempo	50 to 150	117	30.7
Liveness	0 to 1	0.21	0.18
Loudness	-60 to 0	-11.4	5.67
Speechiness	0 to 1	0.09	0.15
Year	1921 to 2020	1980	26

3.1 Data characteristics

The Spotify dataset consists of 160.000 tracks in total that were released between 1921 and 2020. It consists of (1) Primary feature, the track ID generated by Spotify (2) Numerical features, regarding acousticness, danceability, energy, duration in ms, instrumentalness, valence, popularity, tempo, liveness, loudness, speechiness, and year (3) Dummy features, regarding the mode, either minor or major, and explicit context (4) Categorical features, regarding the song key, artist, release date, and the name of the song. Since the dataset doesn’t contain any user-interaction with the songs we decided to take a content-based approach, which means that for recommending similar songs based on a given song we rely exclusively on features that describe the songs.

[1]https://www.kaggle.com/yamaerenay/spotify-dataset-19212020-160k-tracks

3.2 Data Cleaning and Aggregation

Since some of the features provided by the dataset do not provide any meaningful information to encode similarities between songs, we decided to discard the following attributes from further processing: id, name of the song and date of release. Obviously, similar release dates between songs can be an important indicator for similarity, however the year attribute is sufficient for that purpose. Numerical features that are interval scaled but not yet in the range of 0 to 1 are scaled to fit in that range using min-max-normalization (duration, loudness, tempo, popularity, year). The nominal feature key can be encoded into a numerical representation using one-hot-encoding, which creates 11 more features, one for each possible key present in the song. Furthermore, we provide cutoff for the popularity so results with varying degree of popularity are possible (sometimes one might prefer unknown music over popular songs).

The artists feature requires special consideration. The two most prominent techniques to represent non-numerical values are integer encoding and one-hot encoding, but both fall short when it comes to encoding categorical features with high cardinality. Integer encoding would introduce an artifical ordering that doesn't exist in the data. One-hot-encoding on the other hand creates a new feature for each categorical value. Because the feature contains approx. 30000 unique artists, this would create 30000 extra columns in the data, which makes training far too costly on standard hardware and puts too much emphasis on the artist feature.

A possible solution to deal with the problem that has emerged in the last years is to learn an *embedding* for the categorical variable. That way, we can represent each discrete value (each artist) as a low-dimensional vector. If the embedding captures the underlying structure sufficiently, closely related values are geometrically located close to each other within the given vector space. Such an embedding can be learned separately or as a part of the model training.

If the embedding can be constructed from a set of interactions between multiple categorical features, *multiple correspondence analysis* (MCA) [13] can be considered suitable, however in our case each sample after preprocessing consists of numerical features. A prominent technique to derive a low-dimensional embedding from numerical data is the *principal component analysis* (PCA).[14] In short, PCA reduces the dimensionality of the data up to a specified point by transforming the data into a new vector space that captures a maximum of the variance of the original data while using fewer dimensions.

We can apply a PCA before training in the following pattern:

1. Select the first artist of each song
2. Group the data by the artists name
3. Compute the mean for all songs grouped by artist (each song of a certain artist should have the same components later!)
4. Apply the PCA
5. Join the principal components back to the original data

The influence of applying the PCA has not yet been elaborated.

4 Architecture and Training

Our recommender system is based on a contractive autoencoder model which consists of an input layer, a hidden layer, and a reconstruction (output) layer. In general, the autoencoder is a neural network that learns to reconstruct its input in the output in order to encode the features into a hidden (and usually low-dimensional) representation.[11]

4.1 Architecture

In this project we chose to work with an contractive autoencoder instead of a regular autoencoder, since it ensures that similar feature inputs also lead to similar encodings in the hidden layer. This is useful in the context of recommending similar songs, since two songs that have small differences in features like loudness, tempo or danceability are still highly similar and should not differ significantly in their hidden layer representation.

In order to convert an autoencoder into a contractive one, the model also needs to require small derivatives of the hidden layer activations. This is done by creating a loss term that gives penalties for large activation derivatives. To achieve such a loss term, we compose it as the summation of the Mean Squared Error and the weighted L2 norm of the Jacobian matrix of the hidden units with respect to the inputs. Therefore higher values in the Jacobian matrix also lead to a higher loss.

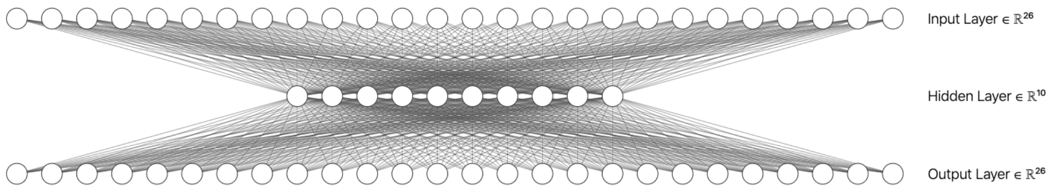
Input Layer ∈ $\mathbb{R}^{26}$
Hidden Layer ∈ $\mathbb{R}^{10}$
Output Layer ∈ $\mathbb{R}^{26}$

Figure 1: Autoencoder architecture

Our final neural network can be seen in Figure 1. Like already mentioned above our contractive autoencoder consists of only a single hidden layer. We encode our input features into the hidden layer by reducing the dimension from 26 to 10. This encoding is done with the ReLU activation function, since this is the standard for deep neural networks and shows a better convergance performance than Sigmoid.[12] The decoding from 10 features back to 26 is done with the Sigmoid activation function, because in our data loading process we normalize our inputs and the Sigmoid function produces an output in the range of [0,1]. This way we can compute the reconstruction error correctly.

4.2 Training

In order to train our model parameters, we use the Adam optimizer from PyTorch[2]. We use it instead of regular Stochastic Gradient Descent since it uses Momentum and Adaptive Learning Rates to converge faster [3]. We initialize this optimizer with a learning rate of 0.001.

Furthermore we chose to train the model in 10 epochs. For each epoch, we iterate through our complete data set in batches of size 4. At the start of each iteration we will first zero the gradients of all parameters so that we update the model parameters correctly when the optimizer steps. After that the loss gets calculated and then the backward pass is made. Finally the optimizer moves one step. We will sum up the losses of each batch and then compute the average loss for the whole epoch.

4.3 Hyperparameter selection

To explain why we chose these specific hyperparamaters in the previous sections, we will present different selections of hyperparameters and showcase how these impact the loss.

The initial training settings we chose were the following: a batch size of 10, a learning rate of 0.0001, a single hidden layer and hidden layer dimension of 10. The loss after 10 epochs was 0.1380%. The following experiments will compare their loss to the loss generated by these initial settings.

4.3.1 Batch size

The first hyperparameter we tried to adjust was the number of samples for each batch. Table 2 shows the results of that experiment.

As we can see, increasing the batch size increases the loss as well. On the other hand decreasing it from 10 to 4 decreases the loss further but also requires more computation.

[2]https://pytorch.org/docs/stable/optim.html#torch.optim.Adam
[3]https://mlfromscratch.com/optimizers-explained/#/

Table 2: Adjust batch size

Batch size	Loss after 10 epochs (in %)
4	0.118
10 (initial)	0.1380
20	0.1704

4.3.2 Learning rate

The learning rate is an important optimizer parameter that defines the step size at each iteration. In the following experiments we tried to find the best setting for our optimizer. Table 3 shows the results of the experiments.

Table 3: Adjust learning rate

Learning rate	Loss after 10 epochs (in %)
0.00001	1.0002
0.0001 (initial)	0.1380
0.001	0.0962
0.01	0.0890
0.1	3.8302

These results show that lowering the initial learning rate increases the loss after 10 epochs. But also increasing the learning rate to 0.1 increases the loss heavily. We can see that we get an optimal result for our loss if we choose a learning rate of 0.01.

4.3.3 Number of hidden layers

Initially our contractive auto encoder consisted of a single hidden layer. In this experiment we implement different number of hidden layers and see how these impact the loss. First we increase the number of hidden layers from 1 to 3 by including 3 layers with dimensions 16, 10 and 16 (Figure 2).

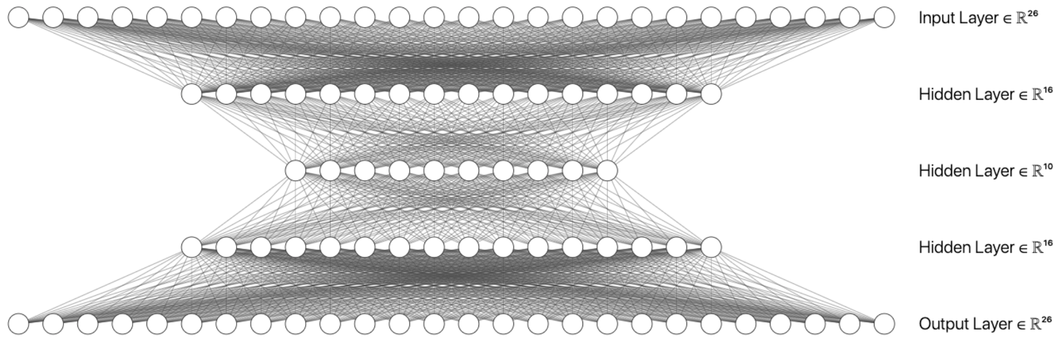
Input Layer ∈ $\mathbb{R}^{26}$
Hidden Layer ∈ $\mathbb{R}^{16}$
Hidden Layer ∈ $\mathbb{R}^{10}$
Hidden Layer ∈ $\mathbb{R}^{16}$
Output Layer ∈ $\mathbb{R}^{26}$

Figure 2: Autoencoder architecture with three hidden layers

The second network we created consists of 5 hidden layers with dimensions 20, 14, 10, 14, 20 (Figure 3). The results are presented in Table 4.

We can conclude from these experiments that creating a more complex contractive autoencoder with multiple hidden layers only decreaeses the rate of convergence. Therefore we decided to construct our autoencoder with a single hidden layer.

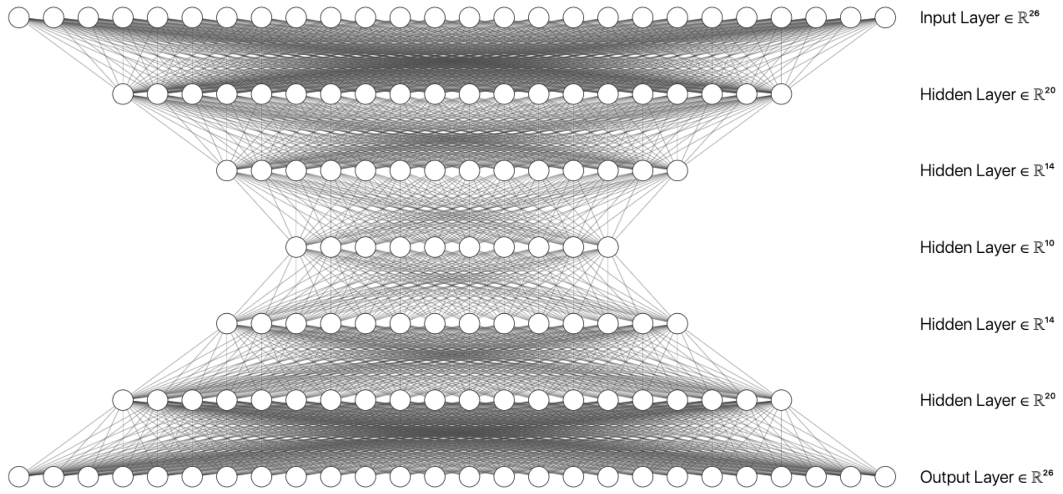

Figure 3: Autoencoder architecture with five hidden layers

Table 4: Adjust number of hidden layers

Number of hidden layers	**Loss after 10 epochs (in %)**
1 (initial)	0.1380
3	0.2991
5	0.6816

4.3.4 Hidden layer dimension

Our initial autoencoder (Figure 1) reduces the input dimension of 26 to the hidden layer dimension of 10. Changing the hidden layer dimension can influence how well our autoencoder can reconstruct the input. Table 5 demonstrates the results of changing this hyperparameter.

Table 5: Adjust hidden layer dimension

Hidden layer dimension	**Loss after 10 epochs (in %)**
4	0.7060
7	0.5770
10 (initial)	0.1380
15	0.0521
18	0.0270
21	0.0169

Although we can see that increasing the hidden layer dimension has a positive outcome on the loss, we need to consider what impact changing to a higher dimension has on the usefulness of the autoencoder in the context of recommending similar songs. If we would adjust the dimension to be 26 like the input layer, the autoencoder would just memorize the inputs and pass them through. In the end, we would have a perfect loss, since all our features are perfectly reconstructed, but our autoencoder wouldn't learn the deeper connection between these songs. This is why a dimension reduction is needed to create these connections and find out similar songs. Generally we want to reduce those dimensions as far as possible to get the best possible connection between similar songs. As we're still achieving a good loss at only a dimension of 10, we decided to keep those initial settings here.

4.3.5 Final hyperparameter selection

By adjusting the learning rate of 0.0001 to the optimal value 0.01 and changing the batch size from 10 to 4 at the same time, we ran into issues with our loss. These settings combined lead to a loss that was decreasing in the first 5 epochs but then increased again in the last 5 epochs. Here our optimizer might step to far and miss the optimum. Therefore we eventually chose a learning rate of 0.001 and a batch size of 4. Furthermore we also kept the number of hidden layers at 1 and the dimension of that hidden layer at 10.

5 Results and Evaluation

In this section we will evaluate how well our model performs. Evaluating the performance of our recommendation system is not entirely possible through objective performance values because similarity between songs depends also on the subjective feeling of similarity. This is why in this section we will evaluate the performance objectively but also subjectively.

5.1 Objective Evaluation

As we're performing an unsupervised training in this project (our data is not labeled) we cannot calculate any metrics that are used for classification tasks like accuracy, recall or precision, since there is no classification to be made. The only performance we can observe is how well our autoencoder reconstructs the input. A measure for that is the average loss in each epoch. By adjusting the hyperparameters we eventually achieved a loss of 0.0567% in 10 epochs.

5.2 Subjective Evaluation

In the following, we provide a test example of our recommender system using one favourite song, and produce an outcome of ten recommendations:

```
Favourite Song: Johnny Cash - Ring Of Fire

======Results======
1. Song name: These Eyes
Song artists: ['The Guess Who']
Spotify URI: spotify:track:5lEyMg4GNKnMbaarvT4sRd
==================
2. Song name: When the Morning Comes
Song artists: ['Daryl Hall  John Oates']
Spotify URI: spotify:track:75XQk1ewCddGnDuFMYg2iL
==================
3. Song name: Cupid
Song artists: ['Sam Cooke']
Spotify URI: spotify:track:6VqVieqjDEwS3mByMq4OzB
==================
4. Song name: You Don't Have to Cry - 2005 Remaster
Song artists: ['Crosby, Stills  Nash']
Spotify URI: spotify:track:0hroZQJfRxMVB5W7eOsJNj
==================
5. Song name: Tequila Sunrise - 2013 Remaster
Song artists: ['Eagles']
Spotify URI: spotify:track:1WYokrkFOb9TFVuTrmxTTt
==================
6. Song name: A Town with an Ocean View
Song artists: ['Joe Hisaishi']
Spotify URI: spotify:track:2ZJ28Rm7OXMYJshLtp5uff
==================
7. Song name: Spanish Pipedream
Song artists: ['John Prine']
```

```
Spotify URI: spotify:track:7zrxGPR1UVK2iSK793vLPl
==================
8. Song name: My Head Hurts, My Feet Stink And I Don’t Love Jesus
Song artists: [’Jimmy Buffett’]
Spotify URI: spotify:track:4RCxgNGNzgIeyDGajCPC5F
==================
9. Song name: The Stroke - Remastered
Song artists: [’Billy Squier’]
Spotify URI: spotify:track:52KvuGmgcgRdrLMXOtda0E
==================
10. Song name: One Of Us
Song artists: [’ABBA’]
Spotify URI: spotify:track:40IHflbrHcOuC8ZcYxUSAC
==================
```

Overall, no song really stands out as a song that is very different from the proposed song. Nearly all of them are from roughly the same time period (1960s to 1980s) with some of them even being other country songs that sound very similar (songs 7 and 8). For the most part, the songs are located in the blues/rock soundspace that is closely related to the given song. For other test songs, we have found similarly good results to be proposed by the recommender system.

6 Discussion

The results that we achieved through our model might be biased. As people tend to rate new songs based on their popularity, other features, such as tempo, duration, or liveness, which are equally weighted in our recommender system, are not actively elaborated by the user when hearing the new song recommendation. Thus, the user might be tempted to like or dislike tracks more or less, depending on the user’s knowledge with regard to the author’s or song’s name. Therefore, in future work, our recommender system will address people with certain personal characteristics, i.e. those who strongly prefer only popular or familiar songs, should weight these features accordingly.

A promising strategy could be to extend the model by incorporating user interactions with the songs. That way a more subjective decision based on the specific users tastes can be reached. Furthermore, some additional feature engineering could lead to better results, for example by creating a genre-artist interaction matrix that could supply more sophisticated information about artists and genres that occur together often, thus improving the recommendation quality.

In conclusion, we present a recommender system based on a contractive autoencoder that is entirely content-based and achieved promising results on a number of different songs, resulting in very similar songs compared to the input. In future work a hybrid model that uses more user-specific features like interactions with the songs could further improve the quality of the recommendations.

Acknowledgments

We want to thank our supervisor, Benjamin Bergner, for his support and thoughtful feedback to our work in weekly discussions during the project phase of this course.

References

[1] Shuai Zhang, Lina Yao, Aixin Sun, & Yi Tay. (2019) Deep Learning Based Recommender System: A Survey and New Perspectives. *ACM Comput. Surv.* 52, 1(5):1-38. DOI:https://doi.org/10.1145/3285029

[2] Adomavicius, G., & Tuzhilin, A. (2005) Toward the next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*.

[3] Carlos A. Gomez-Uribe and Neil Hunt. (2016) The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Trans. Manage. Inf. Syst.* 6(4): 1-19.

[4] Enrique Costa-Montenegro, Ana Belén Barragáns-Martínez & Marta Rey-López. (2012) Which App? A recommender system of applications in markets: Implementation of the service for monitoring users’ interaction. *Expert Systems with Applications* 39(10): 9367-9375.

[5] Hamed Zamani, Markus Schedl, Paul Lamere, and Ching-Wei Chen. (2019) An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation. *ACM Trans. Intell. Syst. Technol.* 19(5): 57

[6] Vasiliy Rubtsov, Mikhail Kamenshchikov, Ilya Valyaev, Vasiliy Leksin, & Dmitry I. Ignatov. (2018) A hybrid two-stage recommender system for automatic playlist continuation. *In Proceedings of the ACM Recommender Systems Challenge 2018. Association for Computing Machinery* 16:1-4

[7] Martin Pichl, Eva Zangerle, & Günther Specht. (2017) Understanding User-Curated Playlists on Spotify: A Machine Learning Approach. *Int. J. Multimed. Data Eng. Manag.* 8(4): 44–59.

[8] Martin Pichl, Eva Zangerle, & Günther Specht.(2015) Combining Spotify and Twitter Data for Generating a Recent and Public Dataset for Music Recommendation *Proceedings of the 26thGI-Workshop on Foundations of Databases (Grundlagen von Datenbanken)*

[9] Martijn Millecamp, Nyi Nyi Htun, Cristina Conati, & Katrien Verbert. (2019) To explain or not to explain: the effects of personal characteristics when explaining music recommendations. *In Proceedings of the 24th International Conference on Intelligent User Interfaces (IUI '19). Association for Computing Machinery* p:397–407.

[10] Martijn Millecamp, Nyi Nyi Htun, Yucheng Jin, & Katrien Verbert. (2018) Controlling Spotify Recommendations: Effects of Personal Characteristics on Music Recommender User Interfaces. *In Proceedings of the 26th Conference on User Modeling, Adaptation and Personalization (UMAP '18). Association for Computing Machinery* p:101–109.

[11] Aston Zhang, Zachary C. Lipton, Mu Li, & Alexander J. Smola. (2020) Dive into Deep Learning: An interactive deep learning book with code, math, and discussions. `https://d2l.ai`

[12] Alex Krizhevsky, Ilya Sutskever, & Geoffrey E. Hinton. (2017) ImageNet classification with deep convolutional neural networks. *Commun. ACM* 60(6):84–90.

[13] Abdi, H., & Valentin, D. (2007). Multiple correspondence analysis. Encyclopedia of measurement and statistics, 2(4), 651-657.

[14] Wold, S., Esbensen, K., & Geladi, P. (1987). Principal component analysis. Chemometrics and intelligent laboratory systems, 2(1-3), 37-52.