DL4AED - PROJECT 1

Joel Tokple, Wassim Boubaker, Nico Jahn

{j.tokple, wassim.boubaker, nico.jahn}@campus.tu-berlin.de

ABSTRACT

Neural networks displace ordinary algorithms in many applications. Classifying short song snippets into genres with generalized embeddings is a novel approach. We trained an autoencoder to fingerprint ~ 3 second long audio sequences. Those are used to classify them with a k-nearest neighbors algorithm into already trained as well as new genre classes. We extend the neural network to semi-supervised learning, creating task specific embeddings. We improved the quality from the simple autoencoder slightly and reached 85% top-2 genre classification as well as 93% top-2 song-ID re-identification. Showing also the weaknesses of classes like ”Pop” while still improving the accuracy by more than 10% compared to an autoencoder-only model.

Index Terms— autoencoder, audio event classification, k-nearest neighbor

1. INTRODUCTION

Products like Shazam[1] are daily used more than 20 million times to re-identify songs[2]. Competitive companies like Soundhound or Musixmatch provide song recognition as well. As some of those exist nearly 20 years, we decided to solve the same problems differently. Song recognition can be divided into two aspects. The first one beeing a mechanism which asserts an unique fingerprint for each input sample. The second one comparing the fingerprint of a requested input sample to an existing database of fingerprints for similarities. The task is, to predict whether the song is in the database and if, returning the song-ID. We trained a neural network to create fingerprints of ~ 3 second long song snippets. A K-Nearest Neighbors approach is used to re-identify similar objects in a high dimensional space. Instead of the supposedly harder problem of song recognition, we decided to it with genres classification. Music is grouped into genres, where all tracks within the same group share stylistic properties and conventions. We assume that each song snippet holds at least some of these specialties and can therefore be classified into the right genre.

2. DATA PREPARATION

2.1. Datasets

To train our network(see Sec. 3) we used the GTZAN dataset [3]. The dataset consists of 1000 songs, each labeled with 1 out of 10 genres. As the dataset is balanced across genres, each genre contains 100 samples, 30 seconds in duration.

To see how our model performes on data coming from a completely different data source than the training data, we used the FMA dataset [4] in its small version. The dataset consists of 8000 songs, each labeled with 1 out of 8 genres. Samples are 30 seconds in duration as well, and the dataset is balanced too (each genre contains 1000 samples).

Another dataset that we used for developing our autoencoder is the FSDKaggle2018 dataset [5]. It is a dataset consisting of various audio events such as sighing, key jangling or coughing. In total the dataset contains 11073 samples, annoted with 1 out of 41 labels. Sample durations range from 0.3 to 30 seconds. We made use of this dataset, in order to introduce natural noise to our training data. In a real world scenario, our classifier should ideally perform well with various other noises interfering with the pure audiosignal of the song. We manually selected a handful of categories for our augmentation.

2.2. Data Preprocessing

To preprocess our data (all 3 datasets), we normalized the raw audio signals into a range of [0,1]. Following this step, we transformed the data into log-mel spectograms, as this is the input format a vast amount of current deep learning research is effectively using for audio data. As a subsequent step, we again normalized the log-mel spectograms into a range of [0,1]. Normalizing the input to our model can result in the autoencoder being able to converge more rapidly [6]. Our resulting log-mel spectograms were shortened on the time axis to a size of 64x640. We split these into sets of 10 non-overlapping images, each being of size 64x64.

2.3. Data Augmentations

Data Augmentations were only performed on the training data we used for the autoencoder. We augmented the data by performing spectogram rolling, frequency- and time masking,

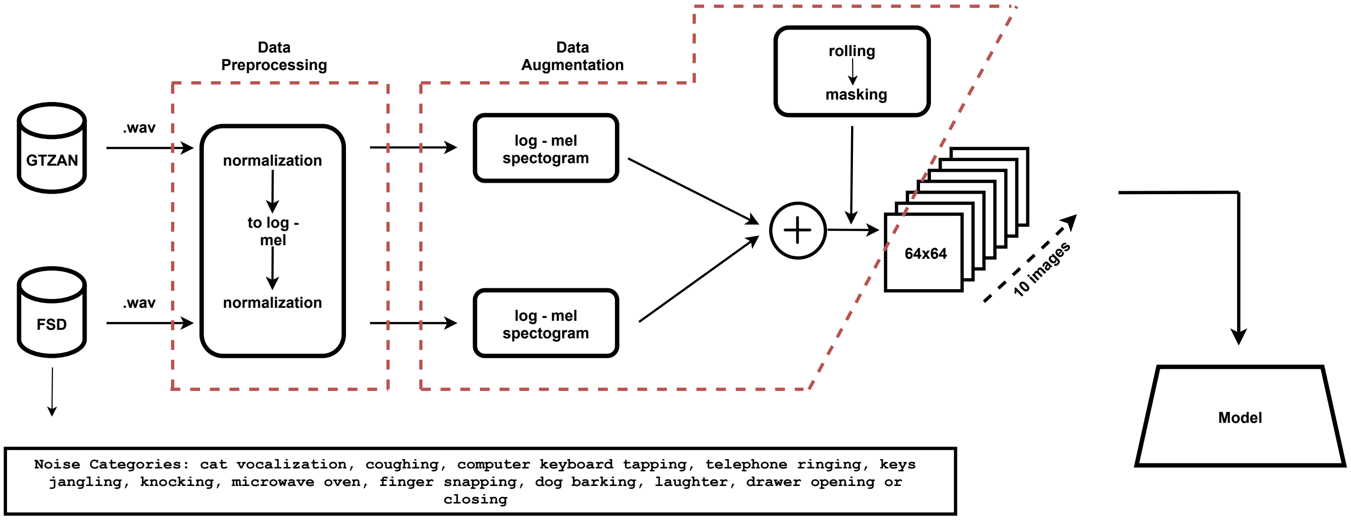
Data
Preprocessing
Data
Augmentation
GTZAN
.wav
FSD
.wav
normalization
to log - mel
normalization
log - mel
spectogram
log - mel
spectogram
rolling
masking
64x64
10 images
Model
Noise Categories: cat vocalization, coughing, computer keyboard tapping, telephone ringing, keys jangling, knocking, microwave oven, finger snapping, dog barking, laughter, drawer opening or closing

Fig. 1: Data Preparation Pipeline used for our model. Manually chosen noise categories can be seen on the bottom left corner.

and mixing noise to the spectograms. Augmentations were performed on log-mel spectograms before the spectograms were split into sets of 64x64 images.

Spectogram rolling describes the process of simply shifting the spectogram randomly either left or right. Since augmentations were performed *online*, as batches were processed during training, rolling the spectograms will cause training samples to be split into slighlty different sets of 64x64 images as they are being processed by the autoencoder multiple times.

Frequency- and time masking describe the process of either randomly setting a range of frequencies throughout the whole spectogram to 0, or randomly setting the whole spectogram in a certain time range to 0. This will prevent the autoencoder from focusing entirely on a set of dominant frequencies or events on samples when trying to learn meaningful features during training.

To introduce noise to our log-mel spectograms, we added one log-mel spectogram, generated from the FSDKaggle2018 dataset, to each of our training samples. For this matter, we randomy chose one sample from the set of categories we manually selected from the FSDKaggle2018 dataset. The theory behind our data preparation approach, as well as noise categories we used, is visualized again in Fig. 1.

3. NETWORK

3.1. Autoencoder

Similar to the previous work of [7] we encode an audio signal into a fixed-size vector representation. Instead of using recurrent neural networks like long short-term memory(LSTM)[8] our network encodes fixed-size input segments. The data is preprocessed as described in Sec. 2. The main network is an autoencoder[9] which commonly used to create feature embeddings. The encoder is based on ResNet[10] a deep residual network. The number of layers is kept at 18 and is therefore known as ResNet18. To decode the 1024-dimensional feature embedding a repeated combination of upsampling and convolutions was applied. The output has the same shape as the input images of $(64 \times 64 \times 1)$. The autoencoder minimizes the reconstruction loss.

3.2. Classifier

Training the above autoencoder without further constraints leads to an embedding which groups items according to their similarity. This might not be appropriate for the classification process mentioned in 4. To form the embedding according to a targeted similarity measure, we introduced an auxiliary loss. Adding a single or multi layered classifier to predict the output class of each embedding. The output classes are the 10 genres mentioned in GTZAN. Weighting the importance of the classifier during the training is a new hyperparameter, the classifier loss scale. The combined networks are plotted int Fig. 2.

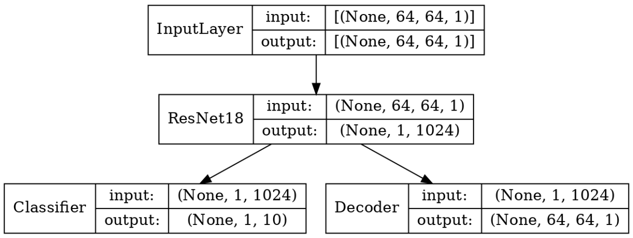

Fig. 2: The neural network with input and output shape of each separate stage. The shapes are specified batch-wise. The same output of ResNet18 is passed to the classifier and the decoder.

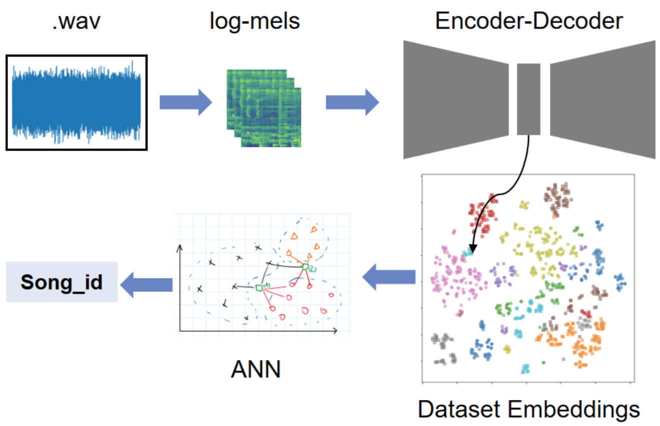

Fig. 3: Full inference pipeline: from song input to both Genre assignment and track recognition.

4. INFERENCE PIPELINE

Once the model is trained, we run the tracks we would like to either classify or recognize through the same preprocessing. The output is fed to the trained ResNet18 encoder part. It returns a 1024-dimensional embedding vector, which is heuristically compared to the known songs (that are embedded in the same space) using a variant of the K-Nearest Neighbors (which is discussed later in 5.1).

A majority vote decides which label or song-ID to assign to each sub-track of the original input. The pipeline returns the final genre or song-ID that are closest to the input sample. Fig. 3 shows the complete inference pipeline for both genre classification and music track recognition.

5. EXPERIMENTS

We trained the neural network for 2500 epochs where the decoder reached a mean-absolute reconstruction error below 0.05. Optimizing 27 million randomly initialized parameter on the GTZAN dataset took 10 hours on a GPU. The classifier loss was tuned with the previously mentioned loss scale. The training was monitored with Weights&Biases[11] which enabled collaborative insights into the training process. We trained different settings on the classifier loss scale, the classifier depth and the decoder loss scale. Our final model was trained with a learning rate of 0.001. The scale of the decoder loss was 1, but the classifier loss was scaled with a value of 0.005. As the 3-layer classifier is smaller than the decoder, it converged faster. We trained decoder-only and classifier-only models to identify the impact of the combined approach.

5.1. Nearst Neighbor Search

Using simple K-Nearest Neighbors (KNN) algorithm and performing an extensive search in the embedded dataset would not scale with a growing dataset. Therefore, we chose to use the Approximal Nearest Neighbors (ANN). Spotify provides a python implementation: ANNOY [12]. Table 5.1 shows how much runtime this spares us compared to the sklearn implementation of the KNN algorithm [13].

	sklearn KNN	ANNOY ANN
Fitting model	51.5s	13.2s
Retrieving 30NNs	206×10^-3s	1.95×10^-3s
Adjusting 1000NNs	58.8	0s (not needed)
Retrieving 1000NNs	172×10^-3s	8×10^-3s

Table 1: Runtime comparison: Fitting on 80000 1024-dimensional embedding vectors and retrieving 30 and 1000 neares neighbors of a sample.

We have tried majority voting of the (approximal) nearest 10, 30 and 100 neighbors to perform Genre classification. $K = 10$ yielded the most stable and reproducible classification performance.

6. EVALUATION

Our pipeline was designed to solve three main problems: Genre Classification, Meaningful Track Embedding and Music Fingerprinting. We evaluate our models on the FMA dataset. The dataset was not used for training our models and 5 out of the 8 genres contained in it were unseen by the Encoder-Decoder network.

6.1. Genre Classification

Using the Approximal Nearest Neighbors algorithm, we were able to achieve a fair performance. Looking at the confusion matrix on Fig.4 (left), one can see that with the exception of the Pop-class, the ANN model does a good job classifying the embedding vectors into their corresponding genres. The model reaches an average accuracy of 63%, with some weaker spots (e.g. recall value of Pop ~ 0.3). If we consider a top-2-guess genre assignment for every sub-track, then the model accuracy is around 85%.

On the other hand, the model performs much better on familiar genres from the GTZAN data: Hip-Hop, Pop and Rock (see confusion matrix on Fig.4 (right)). The model reaches an average classification accuracy of 77% and a top-2-guess for every sub-track accuracy of 97%.

6.2. Embedding Quality

Instinctively, the song embedding should encode the genre information; songs of the same genre should be mapped to closer points in our embedding space. In order to inspect this, we decided to visualize the embeddings. For this end, we used t-SNE [14] to reduce embedding dimensionality from 1024 down to 2. Fig. 5 shows both embedding results on previously unseen data (left) and on the test split of the GTZAN dataset

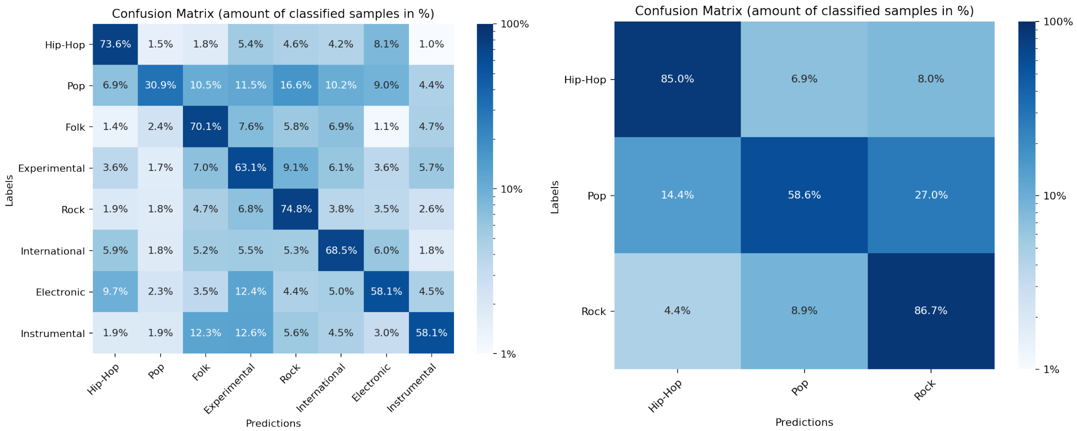

Fig. 4: Confusion Matrices of the ANN model on all FMA genres (left) and only on familiar genres (right).

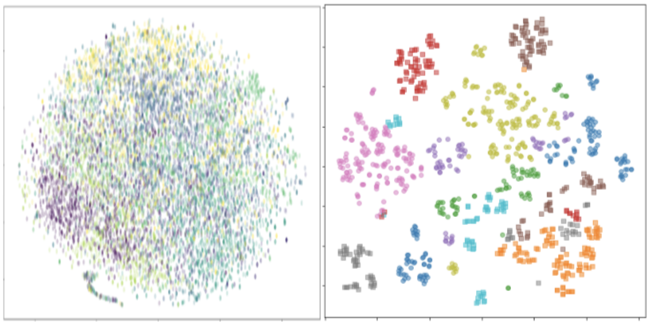

Fig. 5: 2-D t-SNE projection of the 79935 FMA snippets (left) and the 1000 GTZAN testset snippets (right) embeddings. Colors correspond to genres.

(right). One can see a much stronger cluster-structure on the GTZAN data, whereas it is less present on the FMA data (clustering is still present; e.g. purple cluster on the bottom-left, green on the right half). This shows that we should work more on the generalization capability of our Encoder-Decoder model or maybe include a re-training routine, once new genres come into play.

6.3. Music Fingerprinting

Running a song through our pipeline results in splitting the song to subparts, mapping these into our embedding space and performing a majority vote (10 nearest neighbors). This returns the predicted song. Our pipeline was able to correctly identify 76% of the FMA dataset songs and its top-2-guess accuracy is around 93%. We reused the same network as in Sec. 6.1.

7. CONCLUSION

We propose a novel methodology to form a generalized embedding, based on semi-supervised learning. Our inference classification process is scalable as well as extensible. Adapting from genre classification to song recognition is possible with the same network.

Under the assumption that each song snippet has some genre properties we did improve the results to a base autoencoder slightly. While this assumption can be feasible, we expect a better performance with a longer time window for each input. Future research should investigate the neural networks predictions to produce interpretable results, on how decoder and classifier are coupled. It is supposedly simple to improve the overall results with model tuning and the use of a larger training dataset with the developed pipeline. The discrimination of audio events is a relevant research area beyond song or genre classification.

8. REFERENCES

[1] Avery Wang et al., "An industrial strength audio search algorithm.," in *Ismir*. Citeseer, 2003, vol. 2003, pp. 7–13.

[2] Apple, "Apple acquires shazam, offering more ways to discover and enjoy music," 9 2018.

[3] George Tzanetakis, Georg Essl, and Perry Cook, "Automatic musical genre classification of audio signals," 2001.

[4] Michaël Defferrard, Kirell Benzi, Pierre Vandergheynst, and Xavier Bresson, "FMA: A dataset for music analysis," in *18th International Society for Music Information Retrieval Conference (ISMIR)*, 2017.

[5] Eduardo Fonseca, Manoj Plakal, Frederic Font, Daniel PW Ellis, Xavier Favory, Jordi Pons, and Xavier Serra, "General-purpose tagging of freesound audio with audioset labels: Task description, dataset, and baseline," *arXiv preprint arXiv:1807.09902*, 2018.

[6] Andrew Senior, Georg Heigold, Marc'aurelio Ranzato, and Ke Yang, "An empirical study of learning rates in deep neural networks for speech recognition," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 6724–6728.

[7] Yu-An Chung, Chao-Chung Wu, Chia-Hao Shen, Hung-Yi Lee, and Lin-Shan Lee, "Audio word2vec: Unsupervised learning of audio segment representations using sequence-to-sequence autoencoder," *arXiv preprint arXiv:1603.00982*, 2016.

[8] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[9] Geoffrey E Hinton and Ruslan R Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[11] Lukas Biewald, "Experiment tracking with weights and biases," 2020, Software available from wandb.com.

[12] Erik Bernhardsson, *Annoy: Approximate Nearest Neighbors in C++/Python*, 2018, Python package version 1.13.0.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[14] Laurens van der Maaten and Geoffrey Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.