Music Emotion Recognition on the PMEmo Dataset using Neural Networks

Davide D'Ascenzo

Abstract—Music emotion recognition is a complex task that aims to associate affective labels to music and sounds. In this work, audio features, EDA features and lyrics features are combined to predict affective labels in the form of valence-arousal values. In particular, a regression task is carried out using an ensemble of neural networks. The data comes from the PMEmo dataset, also used for benchmarking purposes.

1 Introduction

Music has always been a tool to convey emotions. In fact, since ancient times, music has been used during ceremonies, rituals or events considered sacred and important. Nowadays, almost everyone listens to music, and the main reason always lies in the emotional sphere: a sad song can accompany moments of loneliness, while a cheerful one can improve mood.

*M*usic *E*motion *R*ecognition (MER) is a subfield of *M*usic *I*nformation *R*etrieval (MIR) that aims to analyze music data (or related data) and determine the affective content of music applying machine learning and signal processing techniques. MER systems have many application areas such as music suggestion systems (e.g. Spotify), automatic playlist generation, music therapy, and so forth. [1]

In this work, the PMEmo dataset [2] is used to train a series of neural networks in order to map the feelings of each song to the well-known Valence-Arousal Russell's model [3]. The results will then be compared with the results of the authors of the PMEmo dataset.

2 State of the art

Many algorithms have been developed to address the problem of music emotion recognition, and emotion recognition in general. Some noteworthy works includes:

- [4], that used pLDA[1] for classification of musical emotions in the four quadrants of the valence-arousal plane using physiological signals;
- [5], which used cvxEDA [6] and k-NN to cluster and classify affective sounds using EDA[2] signals;
- [7], that reviewed many important algorithms and feature extraction methods on audio and EDA signals, using the PMEmo dataset as benchmark;
- and many others.

3 Theoretical Model

3.1 Deep neural network

A deep neural network (DNN) is an artificial neural network (ANN) with multiple layers between the input and output layers [8] [9]. There are different types of neural networks but they always consist of the same components: neurons, synapses, weights, biases, and functions [10].

3.2 Feedforward neural networks

A feedforward neural network is an artificial neural network wherein connections between the nodes do not form a cycle [11]. It was the first and simplest type of artificial neural network devised [12]. In this network, the information moves in only in the forward direction (and hence the name): from the input nodes, through the hidden nodes (if any) and to the output nodes. There are no cycles or loops in the network. [11]

3.3 Multilayer perceptron

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses, usually, a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training [13] [14]. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable [15].

The MLP architecture used in this work is parametrized by a series of hyperparameters, and it is composed of the following layers:

- Input[3] layer: this layer represent the entry point of a neural network. Its shape should be equal to

D. D'Ascenzo, Natural Interaction & Affective Computing, A/A 2020-2021, University of Milan, via Celoria 18, Milan, Italy
E-mail: davide.dascenzo@studenti.unimi.it

1. pLDA = *p*seudoinverse *L*inear *D*iscriminant *A*nalysis
2. EDA = *E*lectro*D*ermal *A*ctivity

3. https://www.tensorflow.org/api_docs/python/tf/keras/layers/InputLayer

the shape of the inputs. In our case the shape will be determined by the sum of audio, EDA and text features.

- GaussianNoise[4] layer: this layer augment the input data applying additive zero-centered Gaussian noise. This is useful to mitigate overfitting (you could see it as a form of random data augmentation). As it is a regularization layer, it is only active at training time.
- Dense[5] layer: this is a standard fully connected layer.
- Activation[6] layer: this layer applies a certain activation function. It can be instantiated as a layer or passed as an argument to other layers (e.g. to the Dense layer). Activation functions used in this work are 'PReLU'[7], 'ReLU'[8] and 'Sigmoid'[9]
- BatchNormalization[10] layer: this layer applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1 using last batch output as reference. This operation should reduce the training time and stabilize the network [16].
- Dropout[11] layer: this layer randomly sets input units to 0 with a certain (user-defined) frequency at each step during training time, which helps prevent overfitting. The Dropout layer only applies during training such that no values are dropped during inference.

All MLP structures implemented in this work follow the same architecture, which is, in order:

- Input layer;
- GaussianNoise layer;
- Hidden blocks, each composed of:
  - Dense layer;
  - Activation layer;
  - Dropout layer;
  - BatchNormalization layer;
- Dense layer;
- Activation layer.

Figure 1 shows an example of such architecture.

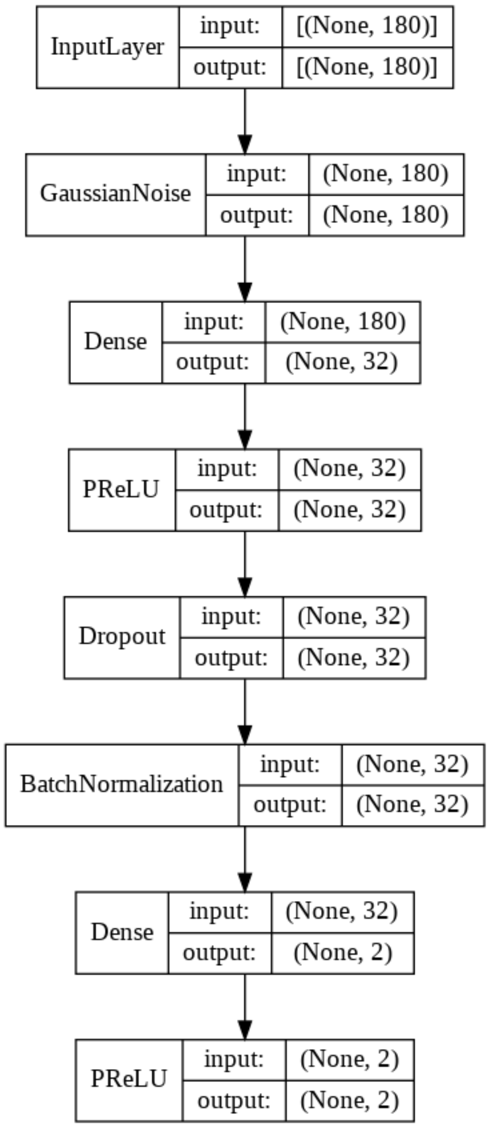


Fig. 1: An example of MLP architecture

3.4 Hyperparameters

An hyperparameter is a parameter which is not learned during training. This kind of parameters are tuned by the user (or another algorithm) to control the learning process in some way. We can identify two types of hyperparameters:

- Model hyperparameters: they are used to control the model hypothesis space. In our case, model hyperparameters are all those parameters that control the MLP architecture.
- Algorithm hyperparameters: they are used to control the speed and the quality of the learning process.

Model hyperparameters used in this work are:

- **Gaussian Noise standard deviation** ('**gaussian_stddev**'): this hyperparameter sets the standard deviation of the GaussianNoise Layer.
- **Number of hidden blocks** ('**hidden_layers**'): this hyperparameter controls the number of hidden blocks used in the MLP.
- **Number of hidden neurons** ('**hidden_neurons**'): this hyperparameter sets the number of neurons of

4. https://www.tensorflow.org/api_docs/python/tf/keras/layers/GaussianNoise
5. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense
6. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Activation
7. https://www.tensorflow.org/api_docs/python/tf/keras/layers/PReLU
8. https://www.tensorflow.org/api_docs/python/tf/keras/layers/ReLU
9. https://www.tensorflow.org/api_docs/python/tf/keras/activations/sigmoid
10. https://www.tensorflow.org/api_docs/python/tf/keras/layers/BatchNormalization
11. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dropout

all the Dense layers in the hidden blocks.

- **Hidden activation function** (‘**hidden_activation**’): this hyperparameter determines the activation function used in the activation layer of all hidden blocks.
- **L1 loss kernel regularizer**[12] (‘**l1_k**’): this hyperparameter applies a weight on the L1 loss on the layer’s kernel (e.g., in the Dense layer, the kernel is represented by the weight matrix). The L1 regularization penalty is computed as:

$$loss = \mathbf{l1} * reduce\_sum(|\mathbf{x}|) \tag{1}$$

where $\mathbf{l1} = \mathbf{l1\_k}$ and $\mathbf{x}$ represents the layer’s kernel.

- **L2 loss kernel regularizer**[13] (‘**l2_k**’): this hyperparameter applies a weight on the L2 loss on the layer’s kernel. The L2 regularization penalty is computed as:

$$loss = \mathbf{l2} * reduce\_sum(\mathbf{x}^2) \tag{2}$$

where $\mathbf{l2} = \mathbf{l2\_k}$ and $\mathbf{x}$ represents the layer’s kernel.

- **L2 loss bias regularizer** (‘**l2_b**’): this hyperparameter applies a weight on the L2 loss on the layer’s bias. The L2 regularization penalty is computed as (2) where $\mathbf{l2} = \mathbf{l2\_b}$ and $\mathbf{x}$ represents the layer’s bias.
- **L2 loss activity regularizer** (‘**l2_a**’): this hyperparameter applies a weight on the L2 loss on the layer’s output. The L2 regularization penalty is computed as (2) where $\mathbf{l2} = \mathbf{l2\_a}$ and $\mathbf{x}$ represents the layer’s output.
- **Batch Normalization** (‘**batch_norm**’): this hyperparameter enables the presence of a BatchNormalization layer at the end of each hidden block.
- **Dropout rate** (‘**dropout**’): this hyperparameter controls the number of neurons disabled by the Dropout layer.
- **Final activation function** (‘**final_activation**’): this hyperparameter determines the activation function used in the last activation layer.

Algorithm hyperparameters used in this work are:

- **PCA output features** (‘**audio_pca_features_n**’, ‘**eda_pca_features_n**’): this hyperparameter determines the number of features outputted by the PCA[14] procedure respectively on audio and EDA data.
- **Batch size** (‘**batch_size**’): this hyperparameter determines the number of examples in each training batch.
- **Training epochs** (‘**epochs_1**’, ‘**epochs_2**’): this hyperparameter sets the maximum number of epochs to train each MLP model.
- **Early stopping patience** (‘**patience_1**’, ‘**patience_2**’): this hyperparameter defines the number of epochs after which training is prematurely interrupted since the model has stopped improving.
- **Trials** (‘**trials**’): this hyperparameter sets the number of trials for the hyperparameter optimization algorithm.
- **Selected models** (‘**selected**’): this hyperparameter defines the number of best models to select for further training and manipulation during the Two-step Search[15] procedure.
- **Reiterations** (‘**reiteration**’): this hyperparameter sets the number of reiterations for the second step of the Two-step Search.

4 Experiment

A static annotation regression task was performed on the Colab[16] platform using the PMEmo dataset.

4.1 Dataset

The PMEmo dataset is a *P*opular *M*usic dataset with *Emo*tional annotations. It consists of 794 music clips annotated by 457 subjects and provides:

- song metadata (song title, artists, beginning and ending timestamps of chorus section),
- song lyrics (LRC),
- song comments from online music websites (Chinese and English texts),
- manually selected music chorus clips (MP3),
- pre-computed audio features for use in MER tasks,
- manually annotated emotion labels: static labels for the whole clips (i.e., overall labels), and dynamic labels for each 0.5-second segment (i.e., continuous labels over time),
- corresponding EDA physiological signals.

[2]

For the purpose of this work only song lyrics, audio features, EDA signals and static labels were used. Of the initial 794 music clips, only 482 have all the features just mentioned, hence the data points were restricted to only those clips.

4.1.1 Song lyrics

Song lyrics are organized in LRC files, each one providing some metadata about the song, and a sequence of lines with a timestamp and the song lyrics. The LRC files have been processed to extract only the lyrics text. Emotional features were then extracted from the lyrics text. The whole procedure is fully explained in Appendix A.

4.1.2 Audio features

Audio features are pre-computed features (by the authors of the PMEmo dataset) which consist of a 6373-dimensional feature space for each clip.

12. https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/L1
13. https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/L2
14. PCA is discussed in Section 4.3.1
15. Two-step Search is discussed in Section 4.2
16. https://colab.research.google.com/

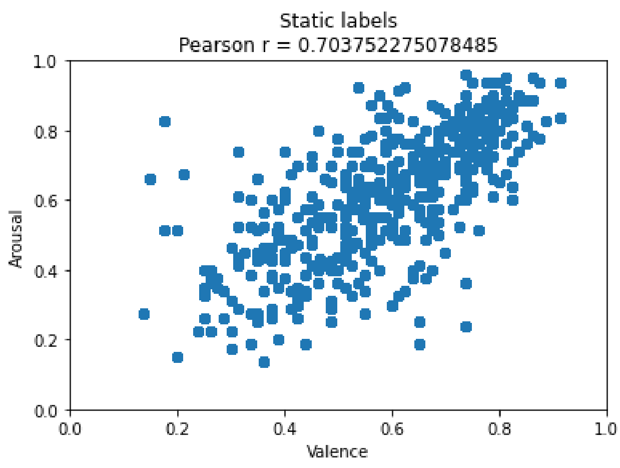


Fig. 2: Static labels distribution

4.1.3 EDA signals

EDA signals are physiological signals collected from the human dermis. Static features have been extracted from the EDA signals following the procedure suggested by the authors of the PMEmo dataset. The dimensionality of the feature space extracted is 128.

4.1.4 Static labels

Static labels are emotion labels, in the form of valence and arousal values, aggregated from the dynamic labels acquired from subjects during the construction of the PMEmo dataset. Both mean values and standard deviation values are available, calculated from the dynamic labels aggregation.

Figure 2 shows the distribution of such labels.

4.2 System architecture

Figure 3 shows the pipeline which takes audio features, EDA features, text features and static labels as input and train an ensemble of MLPs to predict valence-arousal values.

The first part of the pipeline is devoted to data preprocessing, where PCA, oversampling and ICA take place.

The second part is the actual training, where an hypermodel[17] is fed in the random search procedure to search and train models. Then, a subset of all trained models, namely the best ones (according to validation loss) are fed to a re-training procedure. At the end of such procedure the best ensemble is found aggregating the best models and selecting the combination that produces the best ensemble (according to validation loss).

The whole pipeline is repeated a certain number of times, each time randomly splitting the dataset in train, validation and test sets. This operation is called holdout.

The experiment was conducted on the Colab platform, whose computational resources limited the optimization procedure.

17. An hypermodel is a family of models defined by the search space of the model hyperparameters.

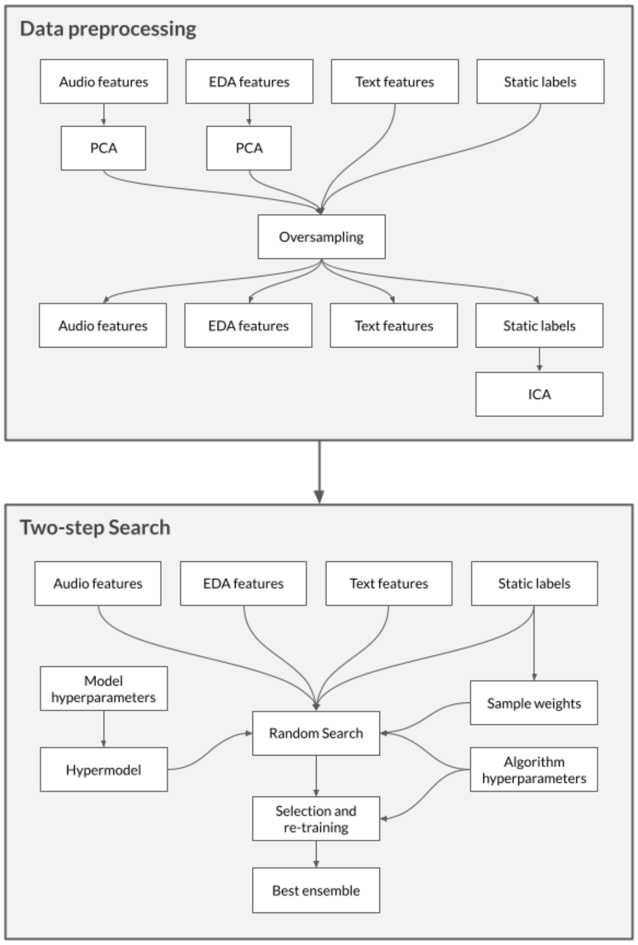


Fig. 3: Architecture pipeline

4.3 Implementation details

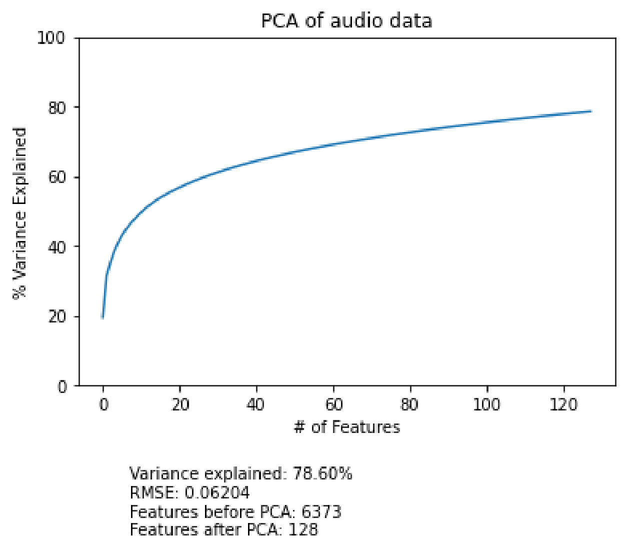
4.3.1 PCA

Principal Component Analysis, or PCA, is a linear dimensionality reduction technique that reduce the dimension of the input performing a change of basis from the input space to the space spanned by the first **n** principal components, where **n** is the desired output dimensionality.

It performs an orthogonal linear transformation that transforms the data to a new coordinate system such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on [17].

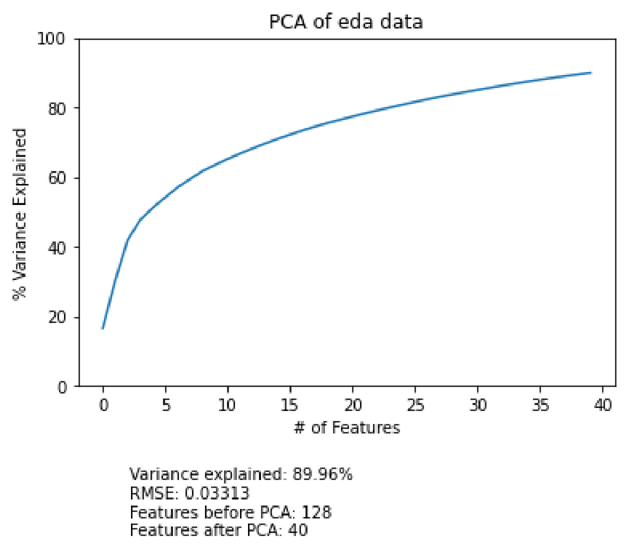
In this work, PCA was performed using the *sklearn* python package[18] and was used to reduce the huge dimensionality of the feature space (compared to the number of data points available), as shown in Figures 4 and 5.

18. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html



```
Variance explained: 78.60%
RMSE: 0.06204
Features before PCA: 6373
Features after PCA: 128
```

Fig. 4: Variance explained from PCA of audio features



```
Variance explained: 89.96%
RMSE: 0.03313
Features before PCA: 128
Features after PCA: 40
```

Fig. 5: Variance explained from PCA of EDA features

4.3.2 *Oversampling*

Oversampling is a data analysis techniques used to adjust the class distribution of a data set by taking more samples from the minority classes. The oversampling technique used in this work is the ADASYN [18] algorithm, which is a modified version of the SMOTE [19] algorithm.

To perform oversampling, classes were assigned to each data point based on their valence and arousal values. In particular, the valence-arousal plane was divided into four quadrants and a class was assigned to each quadrant.

In this work, oversampling was performed using the *imbalanced-learn* python package[19] and was used to augment the dataset with data points from underrepresented valence-arousal quadrants, as shown in Figure 6.

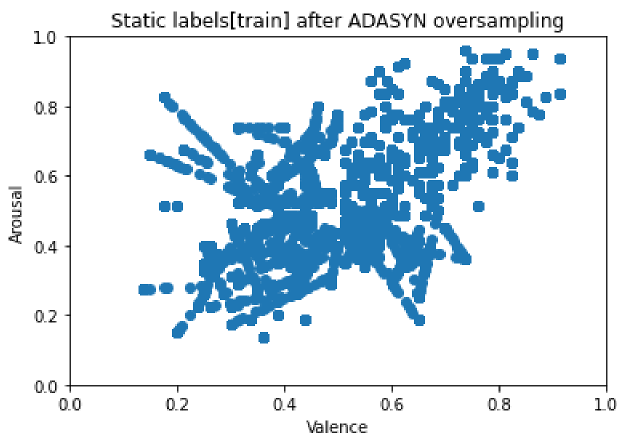


Fig. 6: Train static labels after oversampling

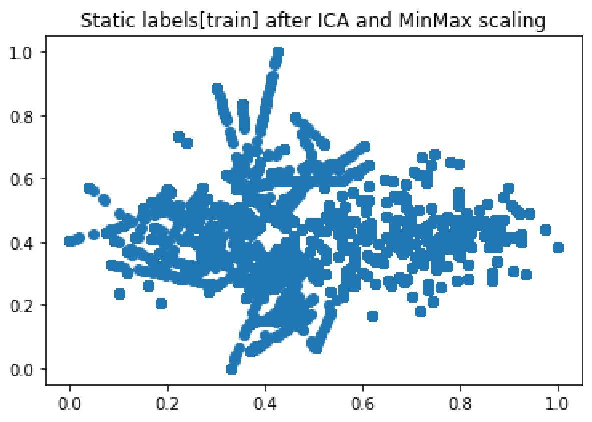


Fig. 7: Train static labels after ICA

4.3.3 *ICA*

Independent component analysis, or ICA, is a technique that attempts to find an affine transformation of the input data so that in the new coordinate system, the different dimensions are statistically independent. This is a stronger constraint compared with principal component analysis, which only requires that the different dimensions be uncorrelated. In other words, ICA seeks a factorizing transformation so that the joint probability density function becomes a product of unidimensional densities, by minimizing the mutual information between the different dimensions. [20]

In this work, ICA was performed using the *sklearn* python package[20] and was used to decouple the strong correlation (≈ 0.7) between valence and arousal values in static labels, as show in Figure 7. This transformation should facilitate the learning process of neural networks.

4.3.4 *Sample weights*

Sample weights are positive values associated to each data point, used in the training process to weight the

19. https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.ADASYN.html

20. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html

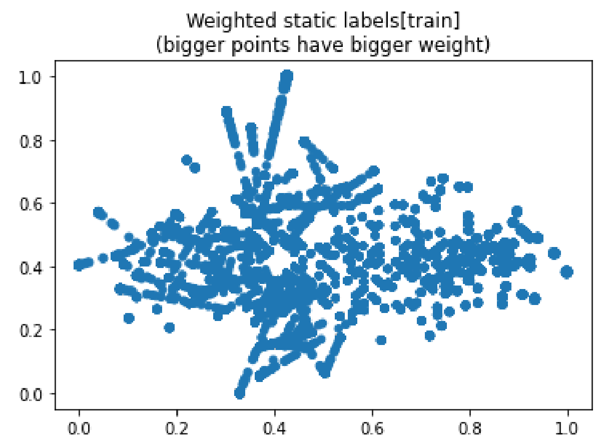


Fig. 8: Weighted train static labels

loss related to such point. When carefully tuned, sample weights can improve learning by changing how each data point should affect the learning process.

In this work, sample weights are calculated using the *normalized inverse standard deviation* times the *KDE*[21] *weight* associated to each data point. In particular, the *normalized inverse standard deviation* is obtained by the following formula:

$$norm\_inv\_std = \frac{1}{1 + valence\_std + arousal\_std}; \quad (3)$$

while the *KDE weight* is obtained by performing a Gaussian KDE (using the *sklearn* python package[22]) on the train and validation data, and exponentiating the resulting normalized log-likehoods. Figure 8 shows the result of the weighting procedure.

The rationale behind such procedure comes from the two aspects of the formula:

- The *normalized inverse standard deviation* weight the data point based on the confidence of the given valence and arousal values. If the standard deviation associated to these values is high, this means that we are less sure about the reliability of such values. On the other hard, if the standard deviation is small, we are more prone to trust the values. This inverse dependence between standard deviation and reliability is incapsulated in the first part of the sample weight formula;
- The *KDE weight* is used to address the data distribution problem of the static labels of the PMEmo dataset. As Figure 6 shows, even after the oversampling the labels are still quite unevenly distributed over the whole valence-arousal plane. In order to deal with this problem, data points are weighted based on how many neighbours around them, resulting is higher weights associated to more lonely points and lower weights associated to points in crowded areas.

21. KDE = *K*ernel *D*ensity *E*stimation
22. https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KernelDensity.html

4.3.5 Random Search

Random search is an optimization technique based on random sampling of the parameter space. In hyperparameter optimization, the random search algorithm simply select random values in the domain of each hyperparameter and perform the training of the generated model. This algorithm can be really powerful when only a small subset of the hyperparameters significantly affect the performance of the model. Since the algorithm perform a random sampling in the parameter space, aggregating and analyzing the results for each hyperparameter can lead to good insights of which hyperparameter we should focus on. In fact, a preliminary random search was performed to find an approximately good hyperparameter search space for the analyzes reported in the following section.

In this work, random search was performed using the *KerasTuner API*[23].

4.3.6 Hyperparameter search space

Table 1 shows the hyperparameter search space spanned by the model hyperparameters of the hypermodel.

4.3.7 Ensemble

An ensemble model, or simply ensemble, is a predictor that uses the outputs of multiple models to generate its predictions. There are various ways to aggregate the outputs of the models fed to an ensemble, which leads to different types of ensemble. Two of the most common ensemble models are the ‘majority voting ensemble’ (for classification tasks) and the ‘average ensemble model’ (for regression tasks).

In this work, the average ensembles were built using *Keras API*[24].

4.3.8 Holdouts

Holdout is the operation of randomly splitting the whole dataset in train, validation and test sets. In this work, due to the limited resources of the environment used for the experiment, only 5 holdouts were performed. Of course more holdouts would lead to more reliable results.

23. https://keras.io/api/keras_tuner/tuners/random/
24. https://keras.io/api/

TABLE 1: Model hyperparameters search space

Hyperparameter	**Start value**	**End value**	**Sampling**
gaussian_stddev	0	0.25	linear
hidden_layers	0	3	linear
hidden_neurons	4	64	linear
hidden_activation	Sigmoid	ReLU, PReLU	-
l1_k	1e-10	1e-4	logarithmic
l2_k	1e-10	1e-4	logarithmic
l2_b	1e-10	1e-4	logarithmic
l2_a	1e-10	1e-4	logarithmic
batch_norm	True	True	-
dropout	0	0.2	linear
final_activation	Sigmoid	ReLU, PReLU	-

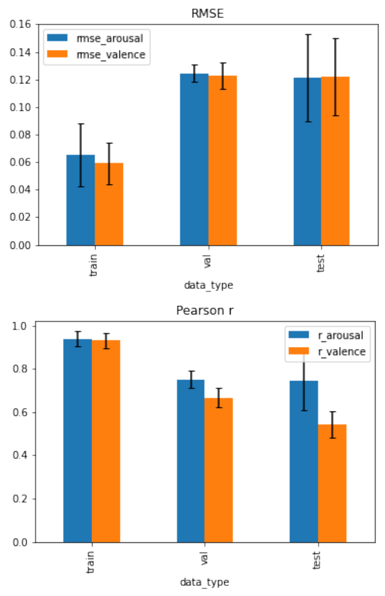


Fig. 9: RMSE and Pearson r calculated over 5 holdouts (confidence intervals are ±*stddev*)

In each holdout the whole dataset is subdivided in a training set (85% of the data) and a temporary set (15% of the data) using a stratified (based on the four quadrants) random split. Then the temporary set is subdivided in a validation set (85% of the remaining data) and in a test set (15% of the remaining data) using again a stratified random split.

Hence, the final proportion of data in each set is:

- 85% of the data in the train set,
- 12,75% of the data in the validation set,
- 2,25% of the data in the test set.

5 Results

Figure 9 and Table 2 shows the results obtained running 5 holdouts. To make a comparison, Tables 3 and 4 show the results obtained by the authors of the PMEmo dataset. We can observe how, even using only $\approx 60\%$ of the original clips, we achieve comparable results with the authors of the PMEmo dataset.

Furthermore, it must be remembered that the limited computational capabilities of the Colab platform have severely limited the training and optimization procedure: training and optimizing for longer time could lead to much better results.

TABLE 2: Evaluation results on static labels

Dimension	RMSE	r	Set
Valence	0.059	0.930	Train
Arousal	0.065	0.938	Train
Valence	0.123	0.666	Val
Arousal	0.124	0.751	Val
Valence	0.122	0.543	Test
Arousal	0.121	0.746	Test

Lastly, Figure 10 shows train, validation and test losses over the model hyperparameters search space.

6 Conclusions

Music emotion recognition is a valuable and complex task which tries to bring out the emotional content of music and sounds. Multi-modal approaches are increasingly common in trying to address complex problems such as those associated with affective or context-aware computing. In this work, we tried to merge audio data with physiological signals and lyrics. The use of these three different data sources is certainly a novel approach, even if the different data have been treated in the same way. Future works could try to develop specific methods and algorithms to treat each type of data differently, creating a common representation space (e.g. a meta-emotional hyperplane) from which to merge the various contributions into the final predictions. In this way, more data could be used to train the algorithms associated with the individual data types, and transfer learning methods can be used to build and train the specific multi-modal model. Indeed, one of the biggest problems in this work was the limited amount of data for a neural network approach. Training algorithms on single data types could solve this problem, since it is possible to use multiple datasets that do not require all three sources of information together.

TABLE 3: Evaluation results (of authors of the PMEmo dataset) on static labels using audio data as input and MLR[25] as regressor

Dimension	RMSE	r	Set
Valence	0.136	0.546	Test
Arousal	0.124	0.638	Test

TABLE 4: Evaluation results (of authors of the PMEmo dataset) on static labels using audio data as input and SVR[26] as regressor

Dimension	RMSE	r	Set
Valence	0.124	0.638	Test
Arousal	0.102	0.764	Test

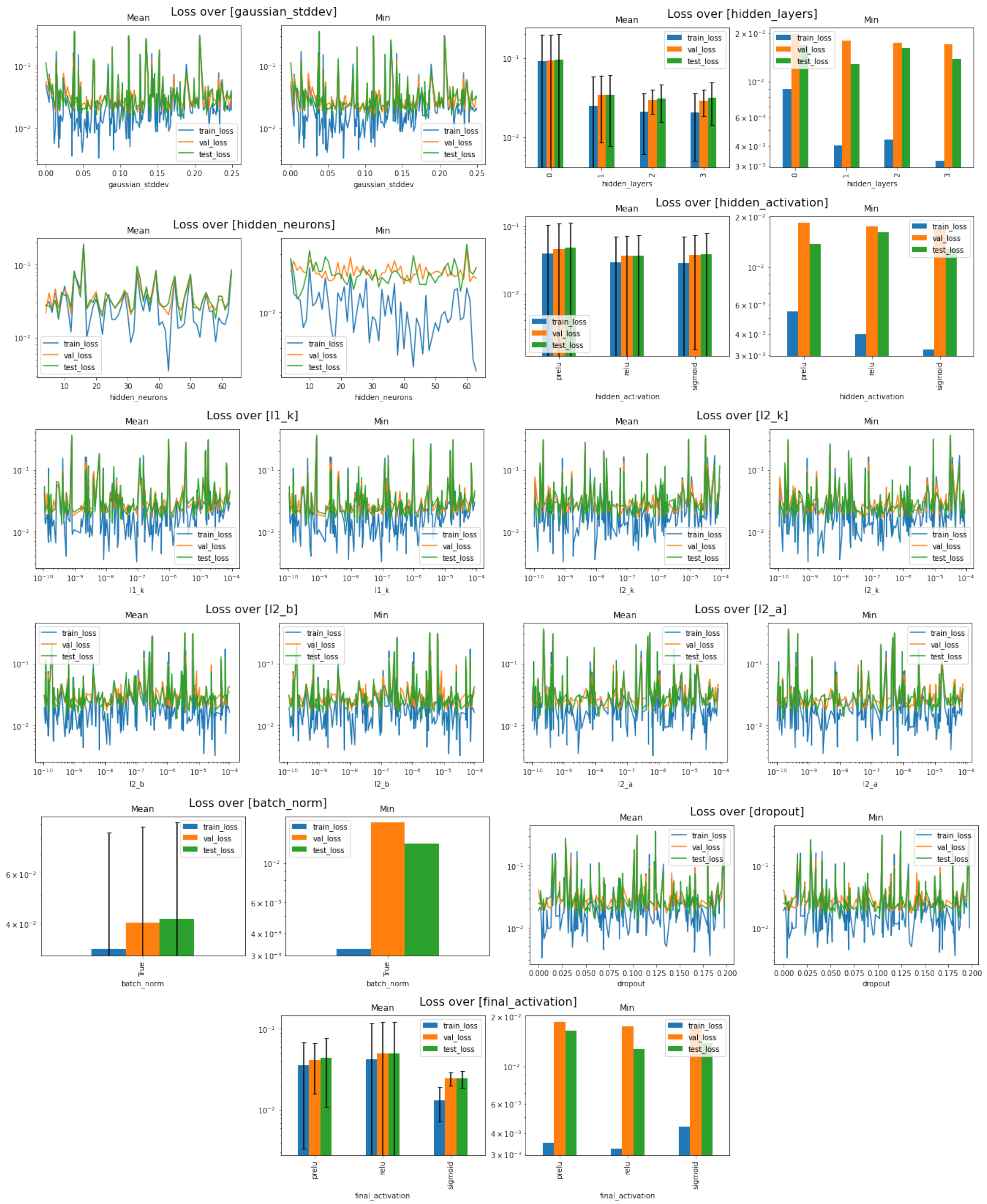


Fig. 10: Losses over the model hyperparameters search space (confidence intervals are ±*stddev*)

Appendix A
Text emotional feature extraction

Text emotional features were extracted using different sentiment analysis tools and classifiers:

- *NLTK*[27] Vader [21], which detects polarity (negative, neutral or positive opinions) within the text;
- *TextBlob*[28], which extracts polarity and subjectivity of a phrase;
- *Flair*[29] TextClassifier, which predicts the polarity of a text;
- *Text2emotion*[30], which output a probability distribution over the following emotion categories:
  - Angry,
  - Fear,
  - Happy,
  - Sad,
  - Surprise.

All those features were concatenated and used as text features associated with each audio clip, producing a 12-dimensional feature vector per clip.

References

[1] S. Hizlisoy, S. Yildirim, and Z. Tufekci, "Music emotion recognition using convolutional long short term memory deep neural networks," *Engineering Science and Technology, an International Journal*, vol. 24, no. 3, pp. 760–767, Jun. 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2215098620342385

[2] K. Zhang, H. Zhang, S. Li, C. Yang, and L. Sun, "The pmemo dataset for music emotion recognition," in *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, ser. ICMR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 135–142. [Online]. Available: https://doi.org/10.1145/3206025.3206037

[3] J. A. Russell, "A circumplex model of affect." *Journal of personality and social psychology*, vol. 39, no. 6, p. 1161, 1980.

[4] J. Kim and E. André, "Emotion recognition based on physiological changes in music listening," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 12, pp. 2067–2083, Dec. 2008.

[5] A. Greco, G. Valenza, L. Citi, and E. P. Scilingo, "Arousal and Valence Recognition of Affective Sounds Based on Electrodermal Activity," *IEEE Sensors Journal*, vol. 17, no. 3, pp. 716–725, Feb. 2017.

[6] A. Greco, G. Valenza, A. Lanata, E. P. Scilingo, and L. Citi, "cvxEDA: A Convex Optimization Approach to Electrodermal Activity Processing," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 4, pp. 797–804, Apr. 2016.

[7] G. Pozzi, "Music emotion detection. a framework based on electrodermal activities," 2020.

[8] Y. Bengio, "Learning Deep Architectures for AI," *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, Nov. 2009. [Online]. Available: https://www.nowpublishers.com/article/Details/MAL-006

[9] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608014002135

[10] Y. Gavrilova, "A Guide to Deep Learning and Neural Networks," https://serokell.io/blog/deep-learning-and-neural-network-guide, accessed: 2021-09-03.

[11] A. Zell, N. Mache, R. Hübner, G. Mamier, M. Vogt, M. Schmalzl, and K.-U. Herrmann, "SNNS (Stuttgart Neural Network Simulator)," in *Neural Network Simulation Environments*, ser. The Kluwer International Series in Engineering and Computer Science, J. Skrzypek, Ed. Boston, MA: Springer US, 1994, pp. 165–186. [Online]. Available: https://doi.org/10.1007/978-1-4615-2736-7_9

[12] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, Jan. 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0893608014002135

[13] D. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," 1986.

[14] F. Rosenblatt, "Principles of neurodynamics: Perceptrons and the theory of brain mechanisms," 1963.

[15] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, Dec. 1989. [Online]. Available: https://doi.org/10.1007/BF02551274

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. ICML'15. JMLR.org, 2015, p. 448–456.

[17] *Principal Component Analysis*, ser. Springer Series in Statistics. New York: Springer-Verlag, 2002. [Online]. Available: http://link.springer.com/10.1007/b98835

[18] H. He, Y. Bai, E. A. Garcia, and S. Li, "ADASYN: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, Jun. 2008, pp. 1322–1328, iSSN: 2161-4407.

[19] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, Jun. 2002, arXiv: 1106.1813. [Online]. Available: http://arxiv.org/abs/1106.1813

[20] S. Edelman and N. Intrator, "Learning as Extraction of Low-Dimensional Representations," in *Psychology of Learning and Motivation*. Elsevier, 1997, vol. 36, pp. 353–380. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0079742108602881

[21] C. Hutto and E. Gilbert, "VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text," *Proceedings of the International AAAI Conference on Web and Social Media*, vol. 8, no. 1, pp. 216–225, May 2014. [Online]. Available: https://ojs.aaai.org/index.php/ICWSM/article/view/14550

27. https://www.nltk.org/
28. https://textblob.readthedocs.io/en/dev/
29. https://github.com/flairNLP/flair
30. https://github.com/aman2656/text2emotion-library