shown the models that were either partially trained or implemented and not trained. In orange are shown CLMR and CALM, which as discussed previously show great promise in terms of reduction of storage space and latent representation learning, which are scheduled to be implemented before the end of the internship[1].

Note : CNSSA was implemented and was to be trained on all datasets. However, due to the different input length and shape, switching between datasets was time-consuming and preprocessing the whole dataset multiple times was not desirable for the obtention of results before the end of the internship. So, CNSSA was only trained on genres.

10. Model training and evaluation

10.1. training

This section focuses on the training of all the selected models. MusiCNN, Short-Chunk and ShortRes are trained on all tag types. CNSSA is trained only on genres. A first round of training on small subsets of the datasets was run to dial in our training hyperparameters as well as eventual dropout values to prevent overfitting. Naturally, this training on smaller subsets highly overfits but shows that the models are capable of learning the features for our custom task. The global training process, from first run to last, took about 3 months. In between each training run, back-and-forth considerations were had on preprocessing, input size, hyperparameters, etc, which explains the retraining of some models. The following figure (10.1) shows the amount of training runs undertaken for all models, as well as annotations of the different phases and models trained The recorded metric here is ROC-AUC on the train and validation sets.

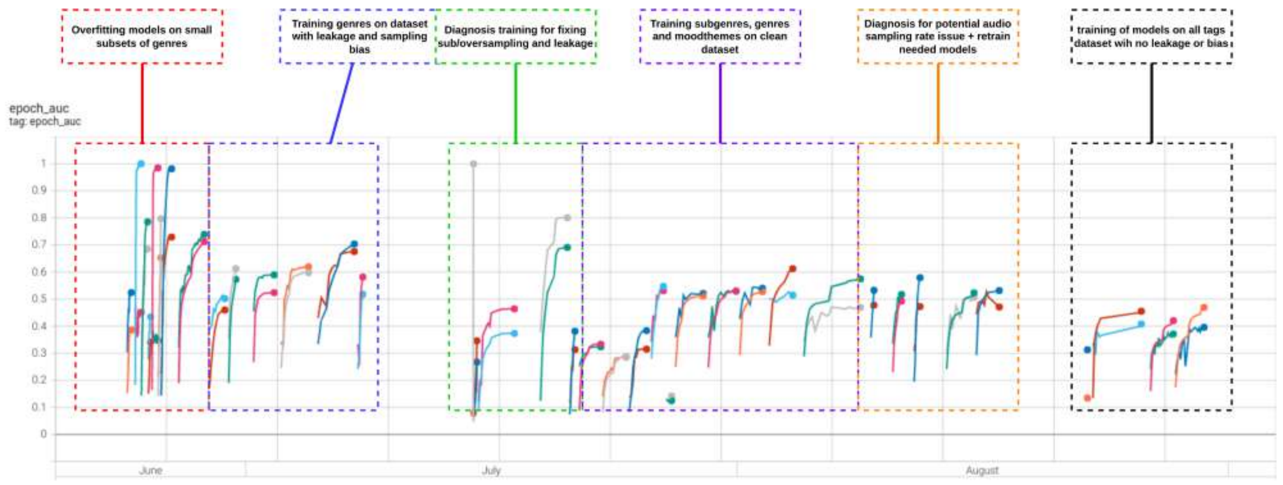


Figure 10.1.: All training runs with various training phases

The first phase allows us to dial in a convolutional dropout value of 0.1 for all convolutional front ends of the models, and per literature a 0.5 dropout layer is added between the dense layers at the end of each network. Phase 1 also allows us to dial in our training hyperparameters, which are the following :

```
training_config = {
"split": {"train_size": 0.85, "val_size": 0.1, "test_size": 0.05, "random_seed": 1},
"training": {"batch_size": 16, "epochs": 100, "steps_per_epoch": None,
"validation_steps": None,"optimizer": Adam(learning_rate=1e-4),"loss": BinaryCrossentropy(),
    "metrics": [AUC(curve="PR"), AUC(curve="ROC")],"max_queue_size": 2,
    "early_stop_patience": 4,"learning_rate_patience": 2,
}}
```

[1]All models which were implemented were implemented in Tensorflow either from scratch or using the pytorch formulations as a template to our own structure. These models can be found at this github repository

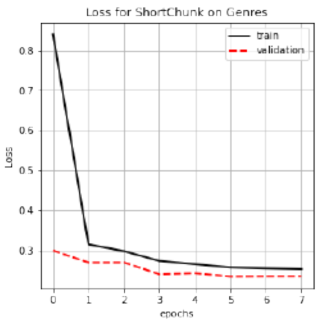
To prevent saddle point sticking, we use an adam optimizer with an original learning rate of 10^{-4}, and a learning rate scheduler which reduces the learning rate by a factor 10 every 2 epochs the validation loss does not decrease. This allows to not overshoot local minima due to gradient momentum. Per state of the art, our problem is a multilabel classification task, which requires a sigmoid activation function on the last layer of each model with and use of the binary cross-entropy loss function:

$$\mathcal{L}(y, gt) = \frac{-1}{N} \sum_{i=1}^{N} gt_i \log(y_i) + (1 - gt_i) \log(1 - y_i) \tag{10.1}$$

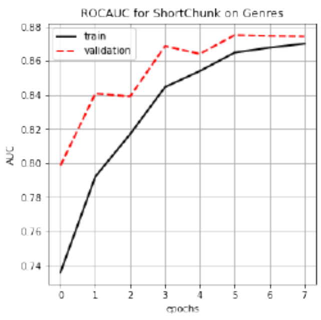
where gt_i is the ground truth label for each sample, and y_i is the model prediction for the given label. This frames our optimization problem. All models are trained for max 100 epochs with an early stopping callback that stops training when validation loss does not go down for 4 epochs. We use a batch size of 16, which is the max value allowed to not overblow the memory of the training GPU (which is a **RTX 2070 QMAX** on a personal laptop). Each model takes about 2 days to train, which explains why it is not possible to iterate endlessly on all these models. However, we manage to train all models in time except for moodthemes on ShortChunk. Here are shown training graphs (monitored on ROCAUC, PRAUC and Loss) for the training of all models on genres. Notice that no overfitting is shown here except for in one case - CNSSA.

ShortChunk

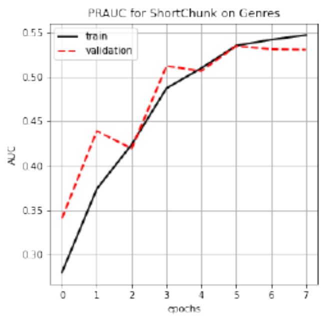
Training for ShortChunk on genres in shown in the following figures 10.2:



(a) Training loss for ShortChunk - genres



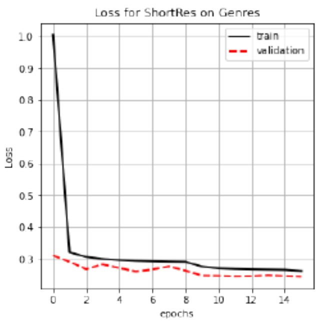
(b) Training ROC-AUC for Shortchunk - genres



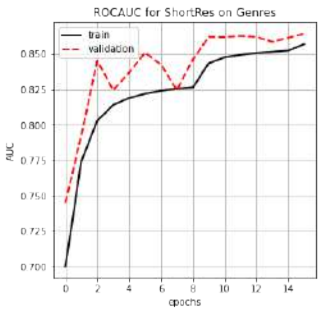
(c) Training PR-AUC for ShortChunk - genres

Figure 10.2.: Training - genres - ShortChunk

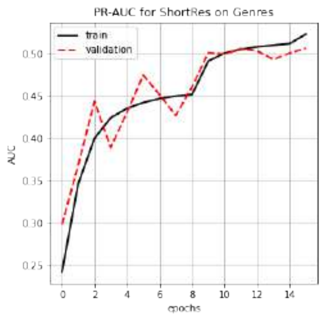
ShortRes



(a) Training loss for Shortres - genres



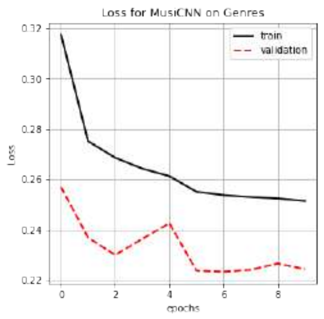
(b) Training ROC-AUC for Shortres - genres



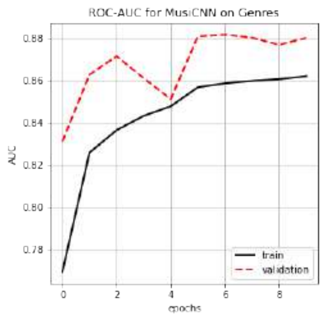
(c) Training PR-AUC for Shortres - genres

Figure 10.3.: Training - genres - Shortres

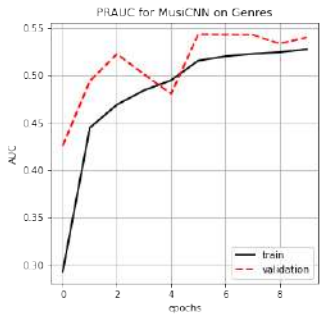
MusicNN



(a) Training loss for MusiCNN - genres



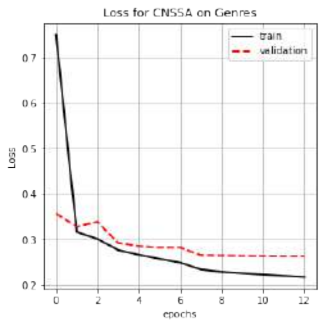
(b) Training ROC-AUC for MusiCNN - genres



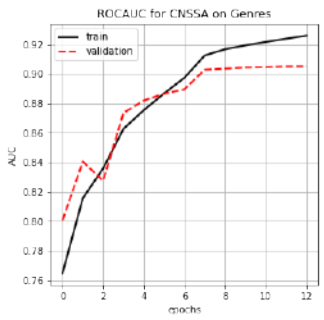
(c) Training PR-AUC for MusiCNN - genres

Figure 10.4.: Training - genres - MusiCNN

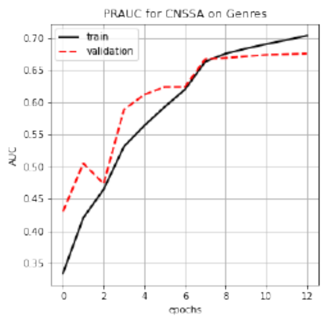
CNSSA



(a) Training loss for CNSSA - genres



(b) Training ROC-AUC for CNSSA - genres



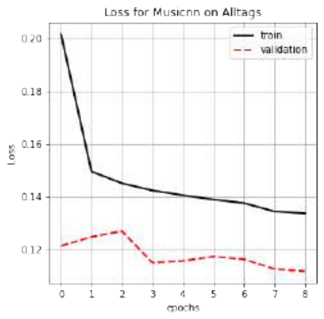
(c) Training PR-AUC for CNSSA - genres

Figure 10.5.: Training - genres - CNSSA

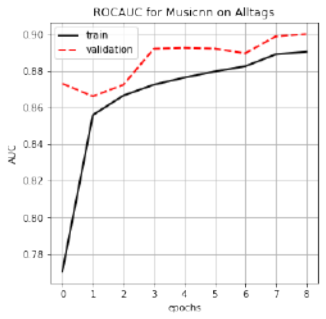
Note that the previous figures (10.5) show a high overfitting on the training set for CNSSA, made obvious by the crossing of the train and validation loss curves even though dropout and regularization is implemented. This as well as higher performance overall is due to the fact that CNSSA was trained with data leakage and sampling bias.

10.1.1. All tags

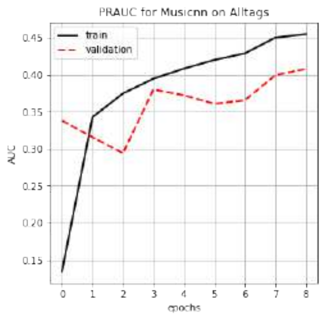
For the sake of brevity, training graphs for other models on the rest of the tag types are not shown here. We show here the training graphs on alltags as this is our final classification objective (Figures 10.6, 10.7, 10.8).



(a) Training loss for MusicNN - Alltags

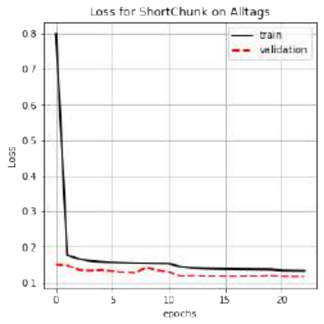


(b) Training ROC-AUC for MusicNN - Alltags

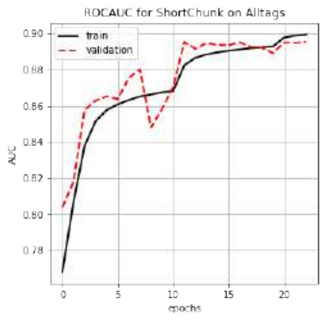


(c) Training PR-AUC for MusicNN - Alltags

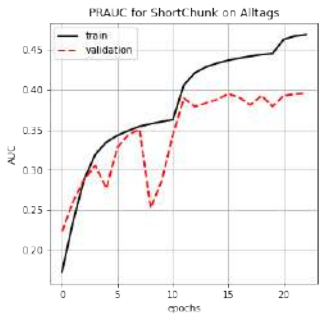
Figure 10.6.: Training - Alltags - MusicNN



(a) Training loss for Shortchunk - Alltags

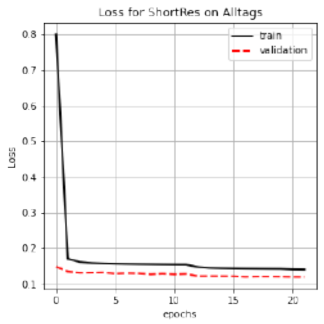


(b) Training ROC-AUC for Shortchunk - Alltags

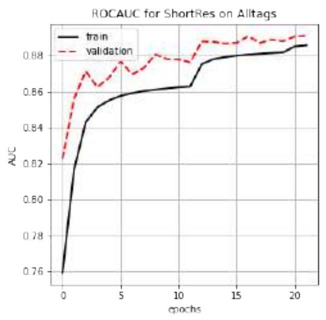


(c) Training PR-AUC for Shortchunk - Alltags

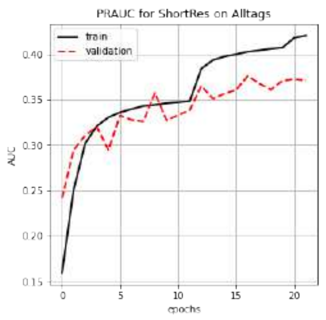
Figure 10.7.: Training - Alltags - Shortchunk



(a) Training loss for Shortres - Alltags



(b) Training ROC-AUC for Shortres - Alltags



(c) Training PR-AUC for Shortres - Alltags

Figure 10.8.: Training - Alltags - Shortres

In all these trainings, though no overfitting can be observed via loss monitoring, all models perform much better on the training set in terms of PR-AUC than on the validation set, while performance based on ROC-AUC remains relatively the same. This is also the case for other tag types with more labels : the higher the label count, the lower the performance of the models based on PR-AUC on the validation set. This hints at a need to relabel and clean the tags that are performing poorly on PR-AUC basis, which we will see in the next section.

10.2. Evaluation and visualization of model results

10.2.1. Key metrics and evaluation levels

In section 2.2.2 We described our macro metrics for music tagging evaluation, which are ROC-AUC, PR-AUC and average cosine similarity. These provide metrics on a global level, but do not distinguish performance between classes, which is important to determine which labels could be better labeled to be cleaner and/or oversampled for better representativity. To this end, we use per-class metrics to evaluate the performance of each model on each dataset:

- **No-average ROC-AUC and PR-AUC**. The global PRAUC and ROCAUC are computed by performing a micro average or macro average of the individual ROCAUC and PRAUC for each class. To visualize the performance of each class on the dataset, we also consider the non-averaged metrics (per-class ROCAUC and PRAUC)
- **Confusion matrices** are used to evaluate multiclass classification problems by comparing the number of true predictions vs for each class versus the actual prediction (see figure 10.9)

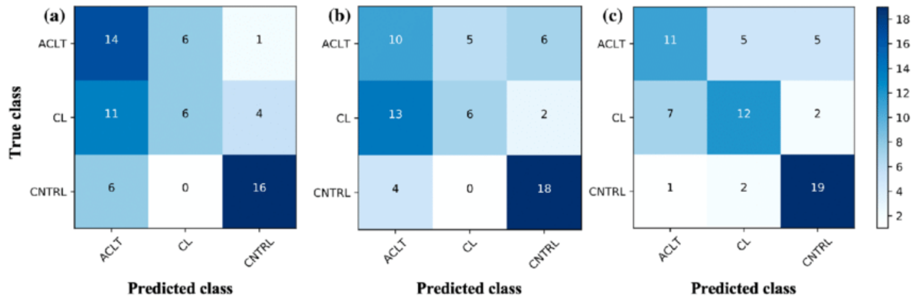


Figure 10.9.: Example of a traditional confusion matrix

This allows to evaluate which classes are missing the mark the most and more importantly which class they are missing the mark on. However, traditinal confusion matrices are best used on single-label multiclass classification problems due to the fact that each prediction can be singled down to a single label which is either correctly predicted or wrongly predicted. In our case, models output a score from 0 to 1 which is not aimed at isolating a single label but rather at ranking them. To solve this problem, we implement a custom confusion matrix in which each cell's value is the sum of predicted scores for that class which were meant for another class:

$$M_{i,j} = \sum_{k \in gt_{k,j}=1} y_{k,i} \tag{10.2}$$

We also normalize each column by the total score of the column, i.e the total score attributed to that class in predictions. So, the custom matrix can be read by column : "for the class *rock*, 25% of the total predicted scores in all the test dataset were attributed to rock, 5% to indie, etc.)

- Finally, we also include **mean Top-K accuracy (MTA_K) and mean soft Top-k accuracy in our test metrics** ($MSTA_K$), which are another custom metric. Top-K accuracy is considered a hit (1) if all ground truth labels are in the top K labels for the given prediction. It is a more harsh metric than soft top-K accuracy, which we consider a hit if at least one of the ground truth top tags is present in the top predictions.

$$MTA_K = \frac{1}{N} |y_i \in \{y_i | \forall k, gt_{i,k} = 1 \rightarrow y_{i,k} \in y_{topk}\}| \tag{10.3}$$

$$MSTA_K = \frac{1}{N} |y_i \in \{y_i | \exists k, gt_{i,k} = 1 \rightarrow y_{i,k} \in ytopk_i\}| \tag{10.4}$$

Where $ytopk_i$ is the k-truncated sorted vector of predictions for the ith sample, y_i is the prediction vector for the ith sample, gt_i is the ground truth label vector for the ith sample.

Multi Instance Learning and song-level evaluation

Previously, he have only discussed generating predictions on individual melgrams. As a matter of fact, all our models generate a prediction on one instance of a melgram, most of which generate predictions on 3s audio. However, depending on the tag, relevant characteristics are not necessarily predominant in the entire music recording. For example, when a song has a tag female vocal, it does not imply that the female vocal appears in every time segment of the song. In essence, this is a multiple instance problem [25]. A song can have many acoustic characteristics (instances) that describe it, but often only specific characteristics (instances) are responsible for the associated music tag. In most cases, we do not have time precise instance-level annotations because the precise labeling can be laborsome, therefore a music tag associated with a song is simply applied to all music excerpts (instances) of the song during training. There are two approaches to handle the multiple instance problem in music tagging. One is to train the model on full songs and produce song-level predictions from a songlevel input. From a given song-level input, the model has to predict relevant tags. Another one is to train the model on short audio chunks (instances) and generate chunk-level predictions, which can be later aggregated (e.g., majority vote, average) in the evaluation phase. We have already decided to conduct chunk-level training on our models, and it is now interesting to evaluate these models on whole songs.

To do this, we generate what is called a **taggram**, which represents the prediction distribution generated by the model on all the instances of the song. For taggram generation, a whole song is split into 3s segments with a given overlap which are fed into the model and the output matrix is a time-series of model predictions (this is shown in figure 10.10 for the track *The nights - Avicii.*

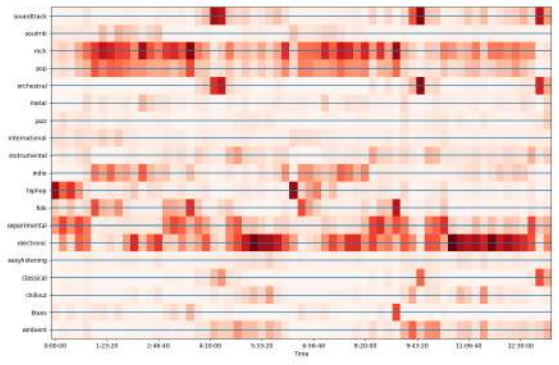


Figure 10.10.: Genre-taggram generated using ShortRes on the song *The nights - Avicii*

Though taggram provide the advantage of being interpretable - they show the evolution of the tags throughout the song : for instance here, the dance choruses are correctly labeled as electornic while the folk-rock intro and verses are labeled as rock, indie, folk - they do not perform song-level evaluations.

The solution is to pool the values of the taggram timewise, by using a pooling strategy (mean, softmax, majority voting...). As this has not been explored in literature yet in terms of the inlfuence of pooling strategy or hop length on music tagging tasks - studies mostly average the predictions into a global prediction - we evaluate all our models on the test dataset at the song level for all tagging types and report results depending on pooling strategy and instance overlap.

10.2.2. Chunk-level metrics

In this section, we focus on results for the melgram-level split for all models. As described in the previous metric section, these models were all evaluated based on all metrics. As their global performance is similar (as it was in the state of the art [51]), and for the sake of brevity, we do not show all the per-class metrics for each model for each tagging type but rather one comprehensive model report per tagging type, to exhibit the model behavior and potential next steps towards improving performance, as all models behave rather similarly and exhibit the same problems within the data.

Genres

Below is shown the performance report for MusicNN on the genres dataset (figure 10.11)

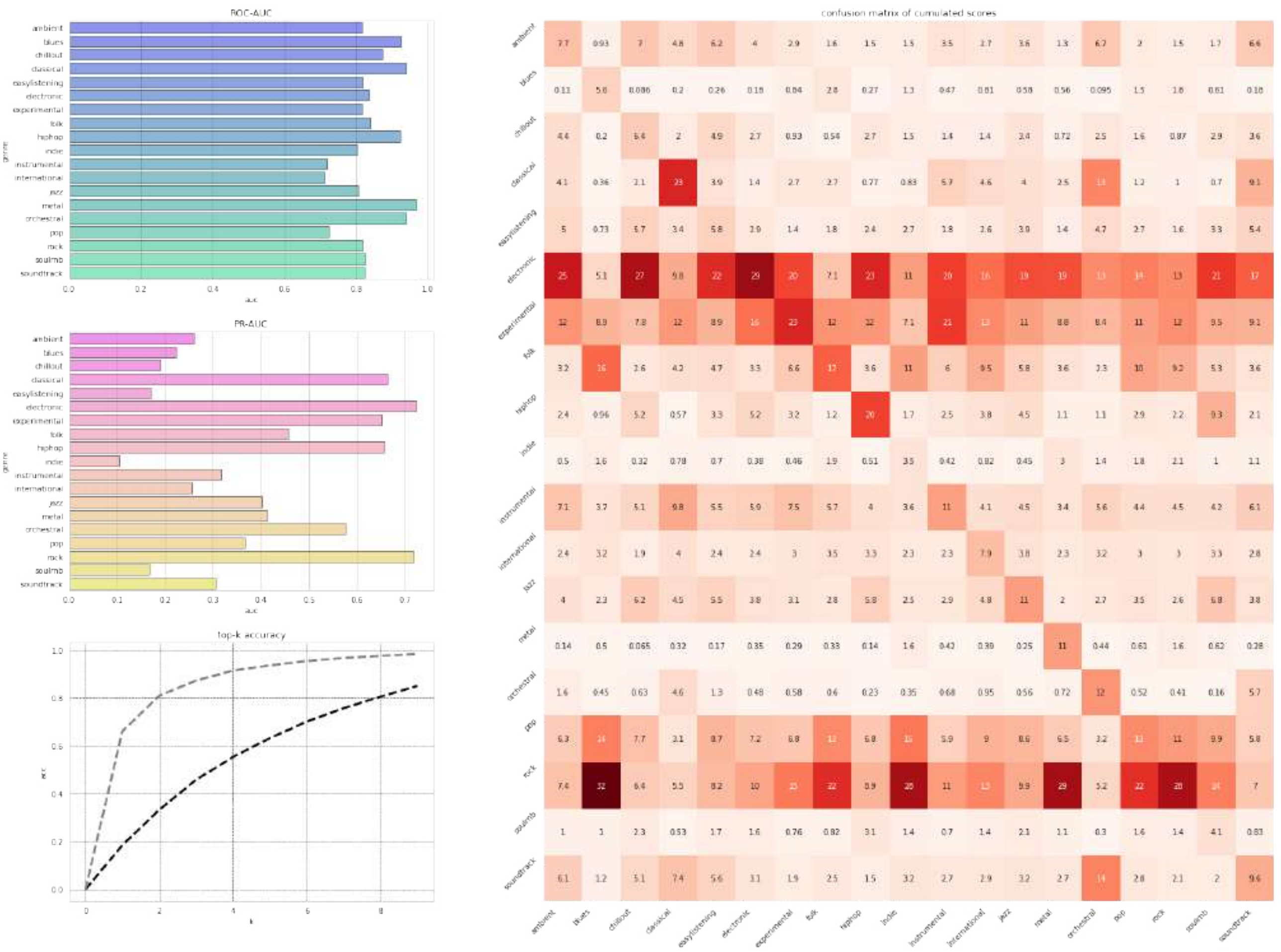


Figure 10.11.: Performance of MusicNN on the genres dataset - Chunk level

As can be seen, ROC-AUC performance is rather consistent across all genres. PR-AUC, however, which represents the hit rate of the model (how many relevant results are retrieved) varies wildly depending on the genre (tag). More specifically, by looking at the confusion matrix we see two horizontal bands that show that the model is skewed towards rock, electronic and pop. This corresponds to the genre distributions shown in the previous sections, and shows the need to consider per-genre resampling. It is reassuring to see, however, that there still is a diagonal of of correct predictions in the confusion matrix, showing that the model does learn relevant features. Furthermore, even though the model is skewed towards overrepresented tags, it makes human-coherent mistakes, such as often confusing *rock* with *indie* or *orchestral* with *sountrack*.

Looking at results on unseen data, we test out the model on the song *Test drive*, an orchestral soundtrack from the pixar animation film *how to train your dragon*. The results in the form of a taggram are shown in figure 10.12:

The model captures the top tags for the song perfectly, and even shows a bridge transition towards the middle of the song with an experimental feel to it. This human validation is corroborated by accuracy at K and soft

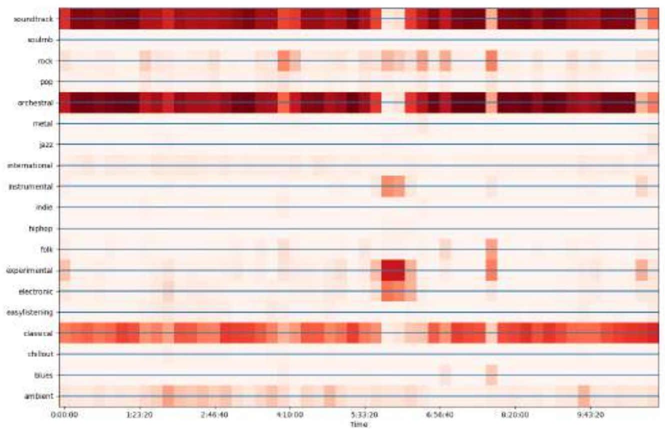


Figure 10.12.: taggram results of shortRes on *Test drive - John Powell*

accuracy at K that show that as early as 5 tags in, at least one of the relevant tags will be present in the results with 95% certainty.

Melgram-level results on the other tag types

In the following figures are shown the results obtained using the best musiCNN model on other tagging types. Though not as clear as in the genres section, the classification metrics and confusion matrix consistently show the model is skewed towards popular tags, and exhibit the same behavioral biases, thus also exhibit a clear road towards performance bettering. Fixing this goes through either data augmentation, attributing lighter wieght to these tags during training, or undersampling these tags even more, specifically on single-label samples. Figure 10.13 shown the performance report for musicnn on subgenres, 10.14 on all tags and 10.15 on moodtheme:

Global results on the melgram split

Results for all models on a standardized split are shown in the table below (10.1):

tags	genres			subgenres			moods/themes			Alltags		
model	**PR**	**ROC**	**Cosim**	**PR**	**ROC**	**Cosim**	**PR**	**ROC**	**Cosim**	**PR**	**ROC**	**Cosim**
Short	0.389	0.829	0.3	0.16	0.79	0.13	0.291	0.810	0.22	0.258	0.799	0.22
ShortRes	0.393	0.831	0.299	0.15	0.79	0.15	**0.31**	**0.819**	**0.26**	0.245	0.801	0.23
MusiCNN	**0.401**	**0.838**	**0.307**	**0.32**	**0.87**	**0.1**	0.26	0.799	0.25	**0.31**	**0.819**	**0.26**

Table 10.1.: Global results for melgram-level evaluation

As intuited in [51], musicnn seems to be a better performer across the board except in the case of moodtheme (we suspect musicnn was trained with an erroneous sample rate when compared to the evaluation sample rate, which could explain the performance delta. On the whole, as shown by the previous figures of class-level behaviour, there is a need for rebalancing classes and potentially exploring single-label data augmentation (or at least minimizing the oversampling of majoritary classes such as *rock* or *electronic*. Overall, results are close to what could be expected based on the state of the art. As our training was on a new dataset, which was potentially noisier and less comprehensive in terms of labels and data quality, and since we did not have time to fine-tune these models as much as possible, it is to be expected that our performance be slightly worse. Not to mention that our tagging tasks were different from those in the literature, and even the alltags dataset had 80 tags in the place of 50 in all papers.

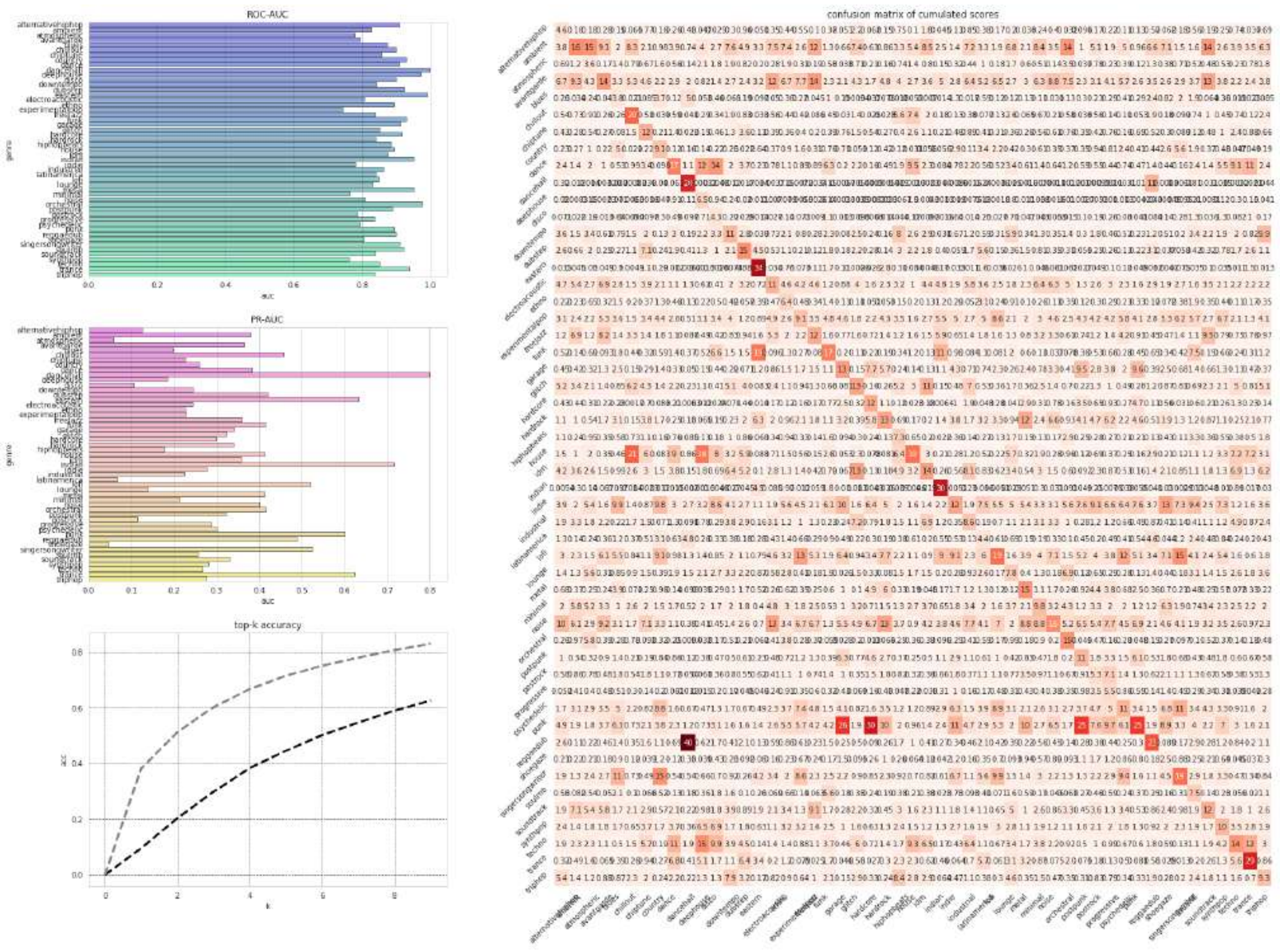


Figure 10.13.: Performance of MusicNN on the subgenres dataset - Chunk level

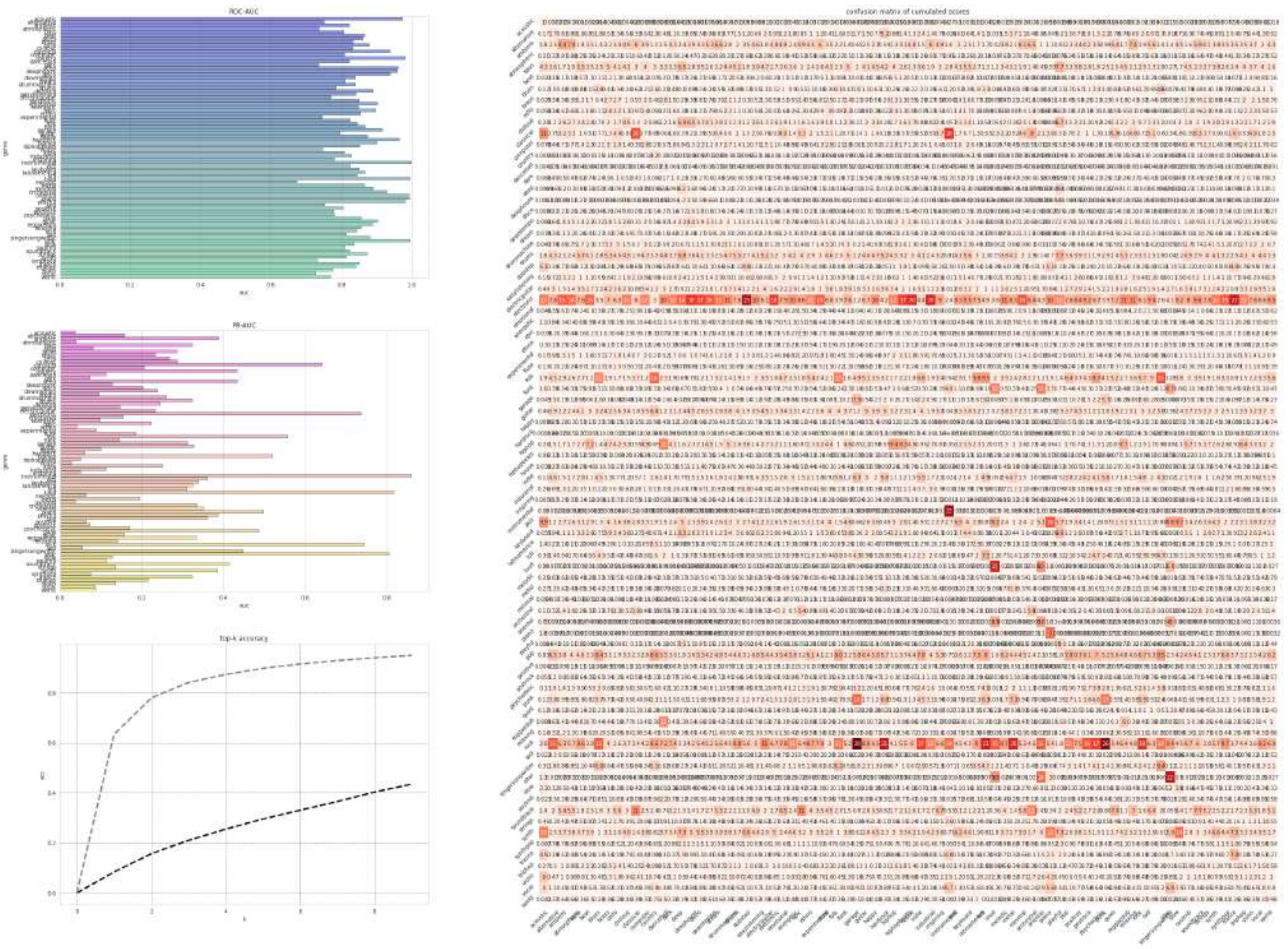


Figure 10.14.: Performance of MusicNN on the alltags dataset - Chunk level

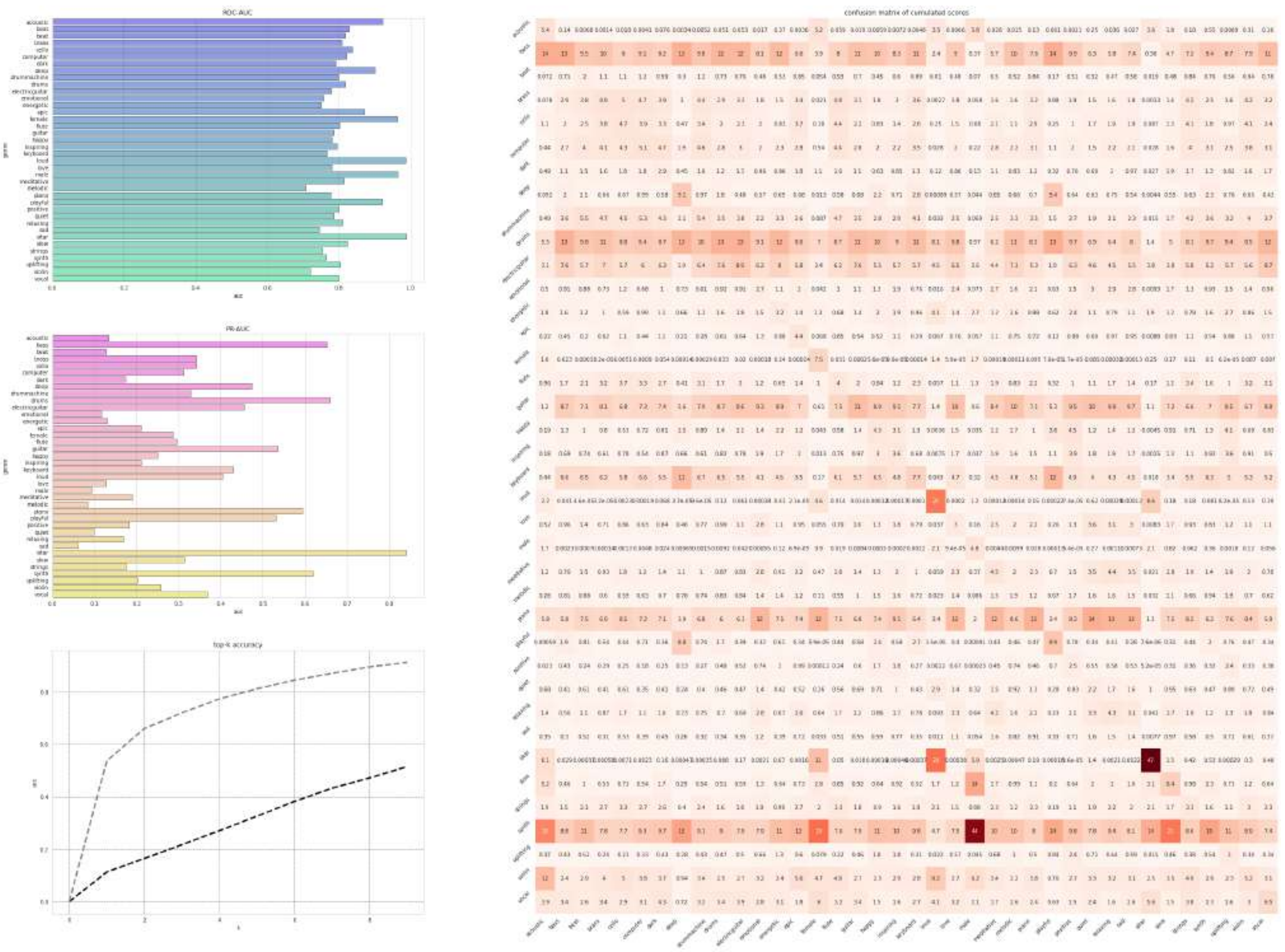


Figure 10.15.: Performance of MusicNN on the moodtheme dataset - Chunk level

Song-level evaluation As described in section 10.2.1, We also evluate the trained models on song-level samples, which allows us to compare CNSSA and other models with various input representation lengths. Taggrams are generated with no overlap between instances (e.g a 21s song is split into 7 3s chunks to be fed to the models and generate a melgram for), and are pooled into a final classification vector using **Timewise average, time-wise majority voting and timewise softmax**. The formula for the softmax activation of a vector is shown below:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{10.5}$$

Which converts vector z into a probability distribution with high discrimination towards low values (low values are rendered very low and high values are close to 1). This type of activation layer is often used in single label classification tasks. Results are reported in table 10.2 for all tag types and pooling methods

genres

Method	Mean			Majority vote			Softmax		
	PRAUC	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**
Short	0.399	0.837	0.6	0.35	0.67	0.33	0.281	0.558	0.403
ShortRes	0.441	0.861	0.618	0.39	0.66	0.35	0.316	0.594	0.499
MusiCNN	**0.448**	**0.876**	**0.634**	0.41	0.72	0.46	0.311	0.608	0.553
CNSSA	0.46	0.89	0.7	0.38	0.73	0.41	0.339	0.621	0.538

subgenres

Method	Mean			Majority vote			Softmax		
	PRAUC	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**
Short	0.233	0.827	0.358	0.238	0.712	0.319	0.19	0.637	0.29
ShortRes	0.266	0.831	0.381	0.249	0.751	0.336	0.19	0.641	0.289
MusiCNN	**0.28**	**0.849**	**0.388**	**0.251**	0.77	0.35	0.21	0.679	0.33

Moodtheme

Method	Mean			Majority vote			Softmax		
	PRAUC	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**
Short	-	-	-	-	-	-	-	-	-
ShortRes	**0.34**	**0.821**	**0.28**	**0.26**	0.731	0.212	0.251	0.632	0.233
MusiCNN	0.31	0.797	0.26	0.253	0.715	0.211	0.236	0.611	0.

Alltag

Method	Mean			Majority vote			Softmax		
	PRAUC	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**	**PRAUC**	**ROCAUC**	**Cosim**
Short	0.26	0.809	0.23	0.233	0.71	0.18	0.245	0.66	0.16
ShortRes	0.291	0.811	0.25	0.258	0. 721	0.2	0.278	0.692	0.2
MusiCNN	0.312	0.823	0.26	0.261	0.729	0.21	0.285	0.691	0.2

Table 10.2.: Song level evaluation

Again, it is shown that MusiCNN substantially outperforms other models on out small dataset, except for CNSSA (which, recall, was trained on the leaky dataset and thus is overestimated in terms of performance). It is also shown that mean pooling is by far the best method for song-level estimation and no other method comes close in terms of performance. Thus, We keep mean pooling as a future reference for implementation at scale. This is also good for artists with multiple tracks as it allows to create an average tag representation over all tracks for the artist without being incoherent with our song level predictions.