Instrument Activation Detection with CRNN

Ioannis Koutsoukis
MSc Student in Artificial Intelligence
University of Piraeus & NCSR Demokritos
GitHub · LinkedIn

Abstract

This project tackles the task of **frame-level instrument activation detection** in music recordings. Given the multi-label nature of the problem, where multiple instruments may be active at any time, we rely on the **AAM dataset**, which includes metadata `.arff` files and corresponding `.flac` audio tracks. We extract **mel spectrograms** and construct both a simple **MLP** baseline and a more advanced **CRNN** model to evaluate performance. We address challenges such as class imbalance with **oversampling** and **class-weighted training**, and perform detailed evaluation through both **metrics** and **visual inspection**. Our findings show promising results for low-resource baselines and indicate the need for larger datasets and more expressive models for improved generalization.

1. Dataset & Preprocessing

1.1 Dataset: AAM

- Used subset: `AAM (subset of 1-1000 songs)`
- Contents: `.arff` files (segment metadata) and `.flac` audio files
- Each `.arff` contains time-annotated instrument activations

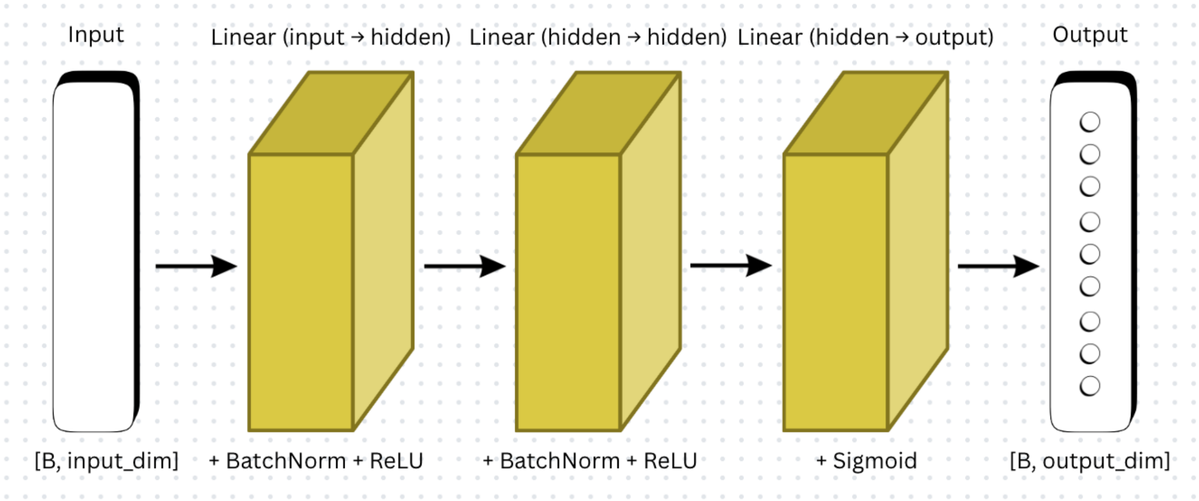
1.2 Feature Extraction: Mel Spectrogram

Extracted using `librosa` with the following parameters:

- `n_fft = 2048`: FFT window size (provides a good balance between time and frequency resolution, allowing us to capture harmonic content while maintaining reasonable temporal precision)
- `hop_length = 512`: hop size (This stride yields around 75% overlap between windows, which is standard practice to ensure smooth time evolution in the spectrogram)
- `n_mels = 128`: number of mel filters (ensures detailed resolution across the perceptual frequency scale, especially in the mid-to-high range where many instruments' timbral features are distinct)
- Converted to decibel scale using `librosa.power_to_db` (This logarithmic scaling better aligns with human hearing perception and helps stabilize training by compressing dynamic range)

2. Model Architectures

2.1 Baseline: MLP



- **Two Hidden Layers**:

Two fully connected layers allow the model to learn **non-linear combinations** of input features. This depth is sufficient for modeling moderate complexity patterns in the input without overfitting given the limited training data.

- **ReLU Activations**:

The **Rectified Linear Unit (ReLU)** is chosen for its simplicity and effectiveness in deep learning. It avoids the vanishing gradient problem and encourages sparsity, helping the model learn more robust features.

- **Batch Normalization**:

Added after each linear layer (before activation), **BatchNorm** stabilizes and accelerates training by normalizing the input distributions per batch. This can allow for higher learning rates and better generalization, especially in small batch setups.

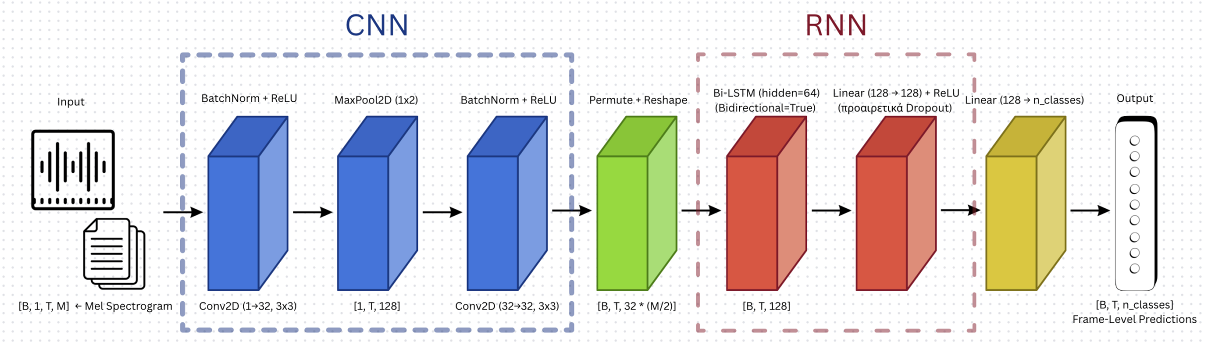
- **Sigmoid Output Layer**:

Since this is a **multi-label classification** task (multiple instruments can be active simultaneously), each output neuron represents the activation probability for one instrument. The **sigmoid activation** ensures independent probability outputs in [0,1][0, 1][0,1], suitable for `BCEWithLogitsLoss`.

```
class MLPInstrumentClassifier(nn.Module):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.bn1 = nn.BatchNorm1d(hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, hidden_dim)
        self.bn2 = nn.BatchNorm1d(hidden_dim)
        self.out = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        x = F.relu(self.bn1(self.fc1(x)))
        x = F.relu(self.bn2(self.fc2(x)))
        return torch.sigmoid(self.out(x))
```

2.2 Main Model: CRNN



- **Mel Spectrogram Input**:

The input is a 2D time–frequency representation that preserves both **spectral and temporal** characteristics, making it ideal for instrument recognition.

- **CNN Front-End (Feature Extraction)**:
    - Two convolutional layers (`Conv2D`) with **batch normalization** and **ReLU** allow the model to learn **local time–frequency patterns** (e.g., instrument timbre).
    - `MaxPool2D` reduces the **frequency resolution** (mel bins), helping generalization and reducing computational load.
    - Output reshaped into a time-major format for the RNN stage.

- **Bidirectional LSTM**:
    - A **Bi-LSTM** captures **temporal dependencies** in both past and future directions for each frame.
    - Hidden size of 64 (→ 128 when bidirectional) balances expressiveness with overfitting risk.

- **Fully Connected Decoder**:

  - A linear + ReLU + (dropout) layer enables richer transformations before the final prediction.
  - The last linear layer maps to the number of instrument classes.
  - No sigmoid here: the final logits are passed to `BCEWithLogitsLoss`, which combines sigmoid internally.

- **Output**:

Produces **frame-level, multi-label predictions**, where each output neuron represents the presence probability of an instrument at a given time frame.

```
class CRNNInstrumentClassifier(nn.Module):
    def __init__(self, n_mels, n_classes, conv_channels=32,
lstm_hidden=64):
        super().__init__()
        self.cnn = nn.Sequential(
            nn.Conv2d(1, conv_channels, kernel_size=(3, 3), padding=1),
            nn.BatchNorm2d(conv_channels),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(1, 2)),
            nn.Conv2d(conv_channels, conv_channels, kernel_size=(3, 3),
padding=1),
            nn.BatchNorm2d(conv_channels),
            nn.ReLU()
        )
        reduced_mels = n_mels // 2
        self.lstm = nn.LSTM(
            input_size=conv_channels * reduced_mels,
            hidden_size=lstm_hidden,
            batch_first=True,
            bidirectional=True
        )
        self.classifier = nn.Sequential(
            nn.Linear(2 * lstm_hidden, 128),
            nn.ReLU(),
            nn.Linear(128, n_classes)
        )

    def forward(self, x):
        x = self.cnn(x)
        B, C, T, M_half = x.shape
        x = x.permute(0, 2, 1, 3).reshape(B, T, C * M_half)
        x, _ = self.lstm(x)
        return self.classifier(x)
```

3. Training Strategy

3.1 Class Imbalance Handling

- Computed class frequency over training set
- Rare classes (<5 tracks) were oversampled ×4
- Also experimented with **class weighting** in loss

3.2 Training Details

- Batch size: 2
  - A small batch size was used due to:
    - **GPU memory limitations**, as frame-level multi-label targets can be large tensors
    - Empirical evidence suggesting **better generalization** with small batches in low-data regimes
      In music tagging literature (e.g., Choi et al., 2017), small batch sizes have been effective for CRNN architectures
- Loss function: `BCEWithLogitsLoss`
  - Suitable for multi-label classification because:
    - It applies **sigmoid activation** to each output logit
    - Computes **binary cross-entropy** independently for each label (instrument)
    - Supports **non-exclusive classes**, allowing multiple instruments to be active at each frame
- Optimizer: Adam
  - Chosen for its:
    - Adaptive learning rates per parameter
    - Robust convergence without the need for heavy tuning
    - Proven performance in training deep models in audio and sequence-based tasks
- Dataloaders handled frame-level masking and padding
  - To handle **variable-length audio clips** within batches:
    - Inputs and targets are **padded** to the longest sequence in the batch
    - A **binary mask tensor** is used during loss computation to ignore padded time steps
      This ensures the model learns only from valid frames, making training **stable and accurate**.

4. Evaluation

4.1 Metrics Used

- **F1 Score (macro / micro / weighted)**

The F1 Score balances **precision** and **recall**, especially important in imbalanced multi-label tasks:

- **Macro**: treats all classes equally
- **Micro**: aggregates all true positives and false negatives globally
- **Weighted**: accounts for class imbalance by weighting by support

- **Hamming Loss**

Measures the fraction of **incorrect labels** to total labels. It is particularly useful in multi-label problems where false positives and false negatives are equally important.

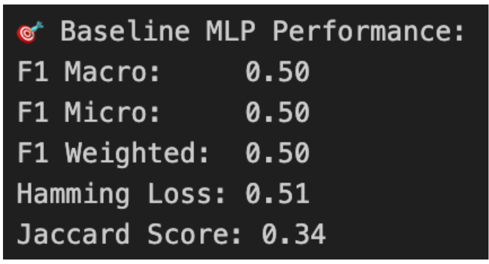
- **Jaccard Score (sample-based)**

Computes the **intersection-over-union** between predicted and true labels for each sample. This is a strict metric that penalizes both over- and under-prediction of labels.

- **Classification Report per Instrument**

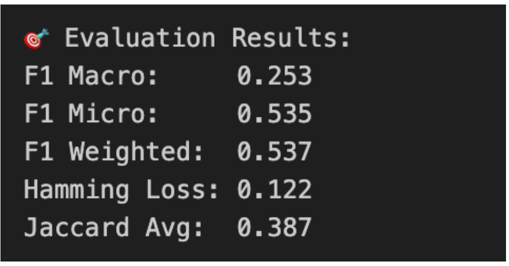
Provides detailed **precision**, **recall**, and **F1-score** per instrument class. This helps identify:

- Which instruments are consistently predicted well
- Which are frequently confused or underrepresented



```
🎯 Baseline MLP Performance:
F1 Macro:     0.50
F1 Micro:     0.50
F1 Weighted:  0.50
Hamming Loss: 0.51
Jaccard Score: 0.34
```

MLP



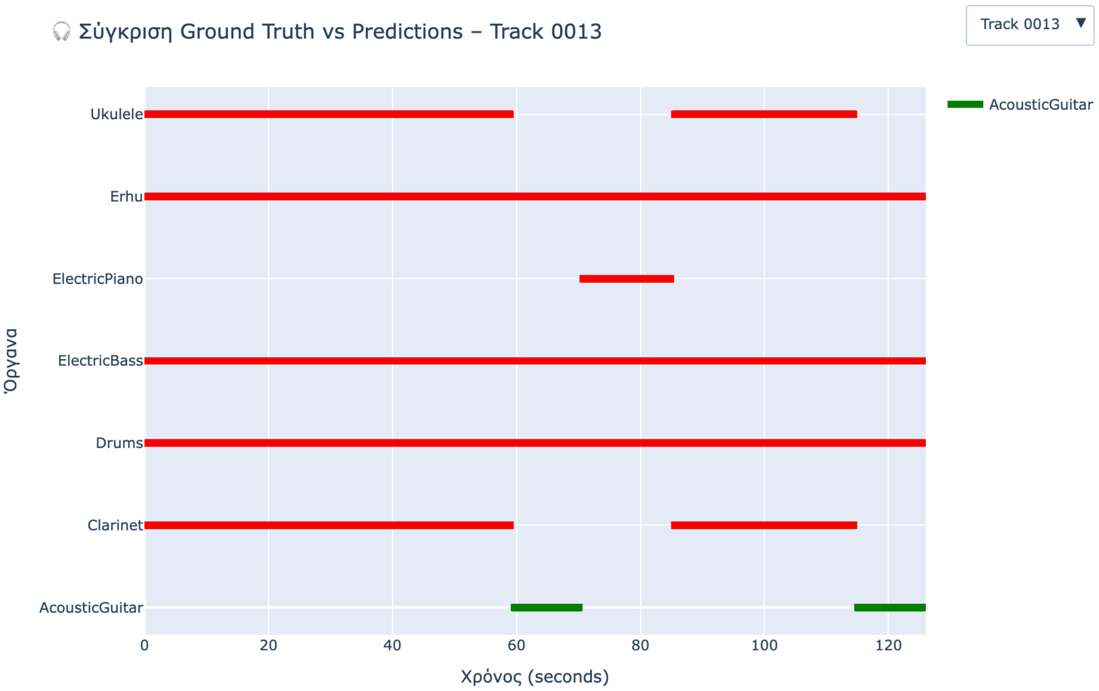
```
🎯 Evaluation Results:
F1 Macro:     0.253
F1 Micro:     0.535
F1 Weighted:  0.537
Hamming Loss: 0.122
Jaccard Avg:  0.387
```

CRNN

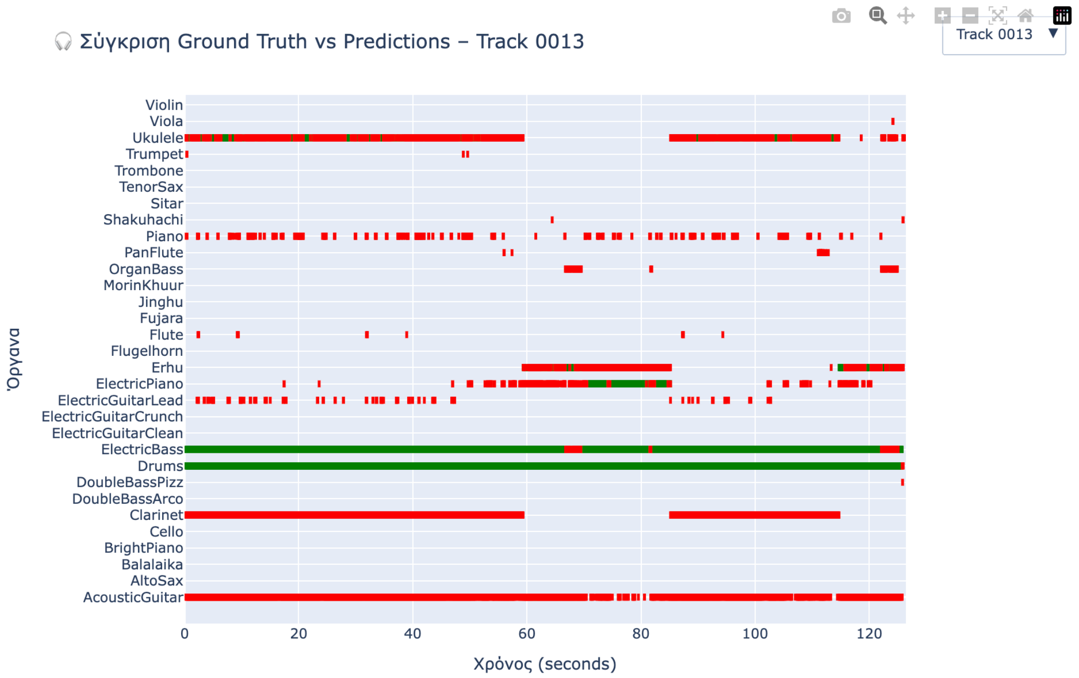
4.2 Visualizations

- **Per-track visual inspection** of predicted vs ground truth activations
- **Sorted classification reports** per instrument
- (To be added: screenshots/plots at this section)

Track 0013

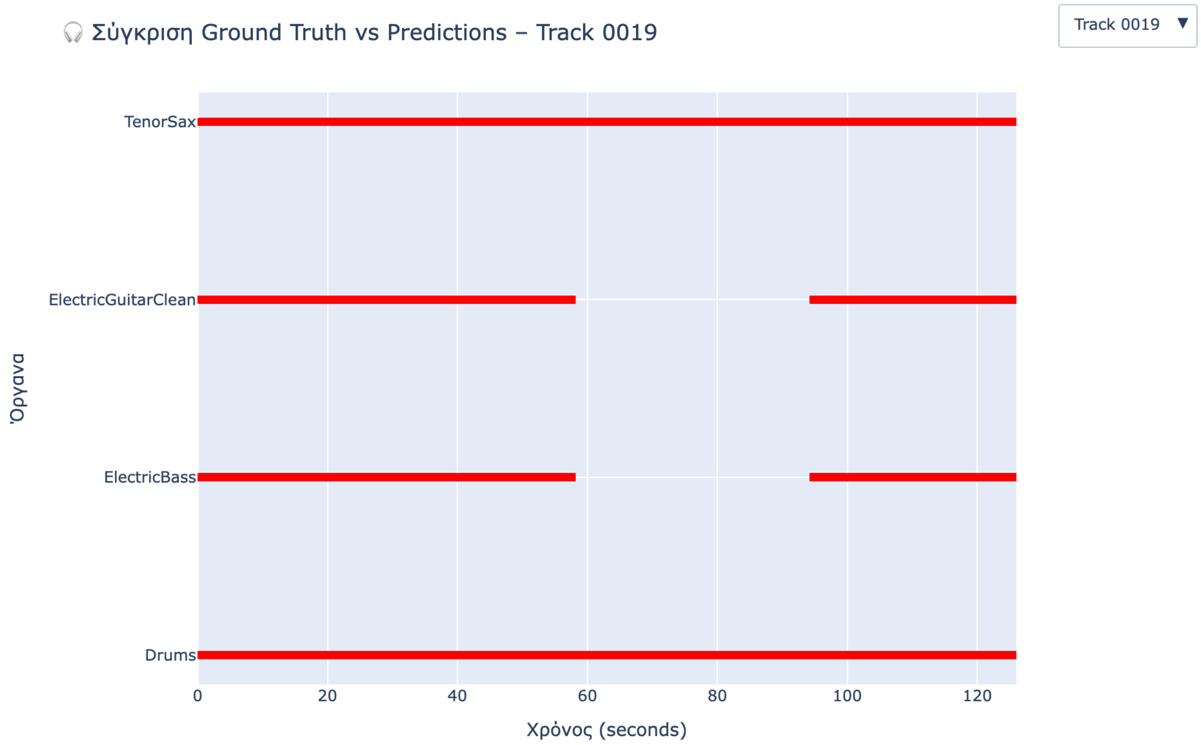


MLP - 150

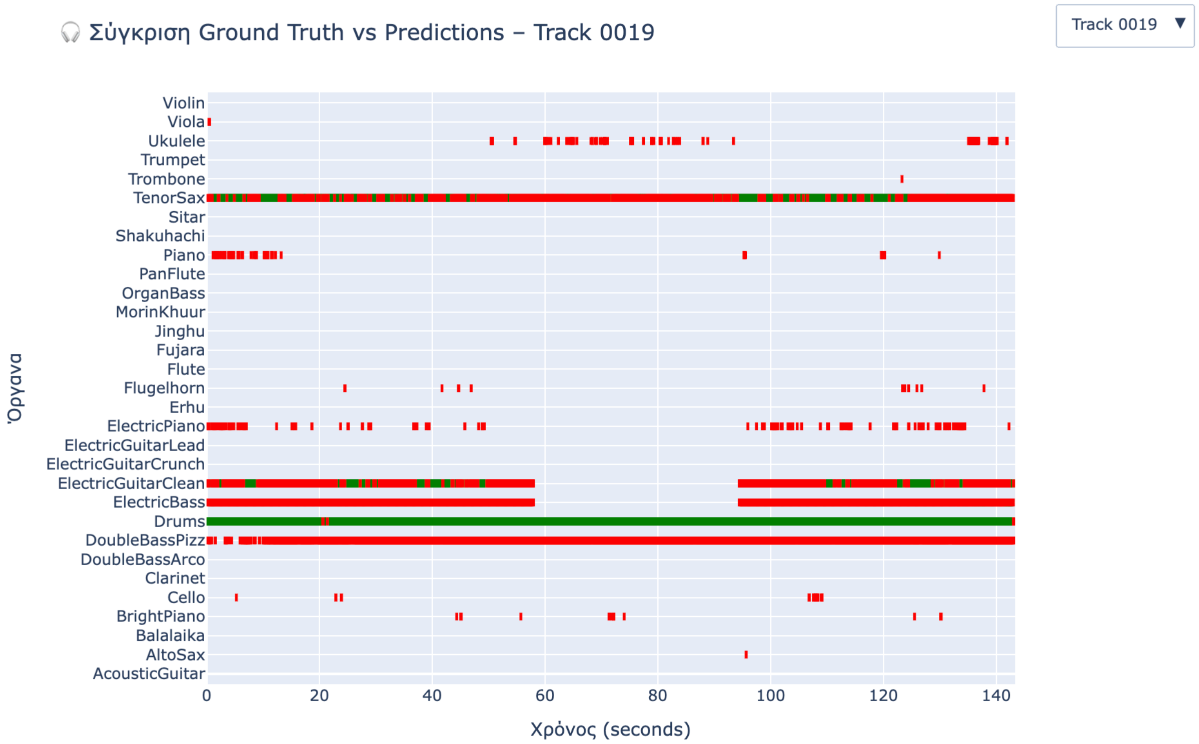


CRNN – 150

Track 0019

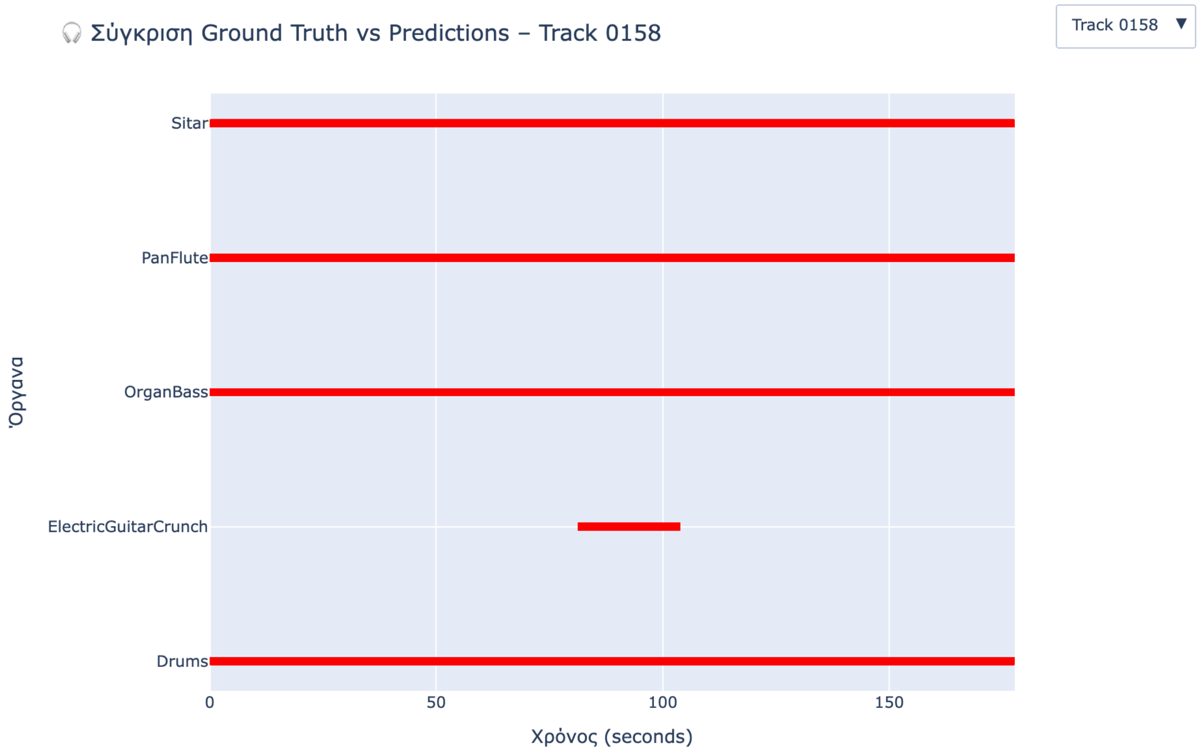


MLP - 150

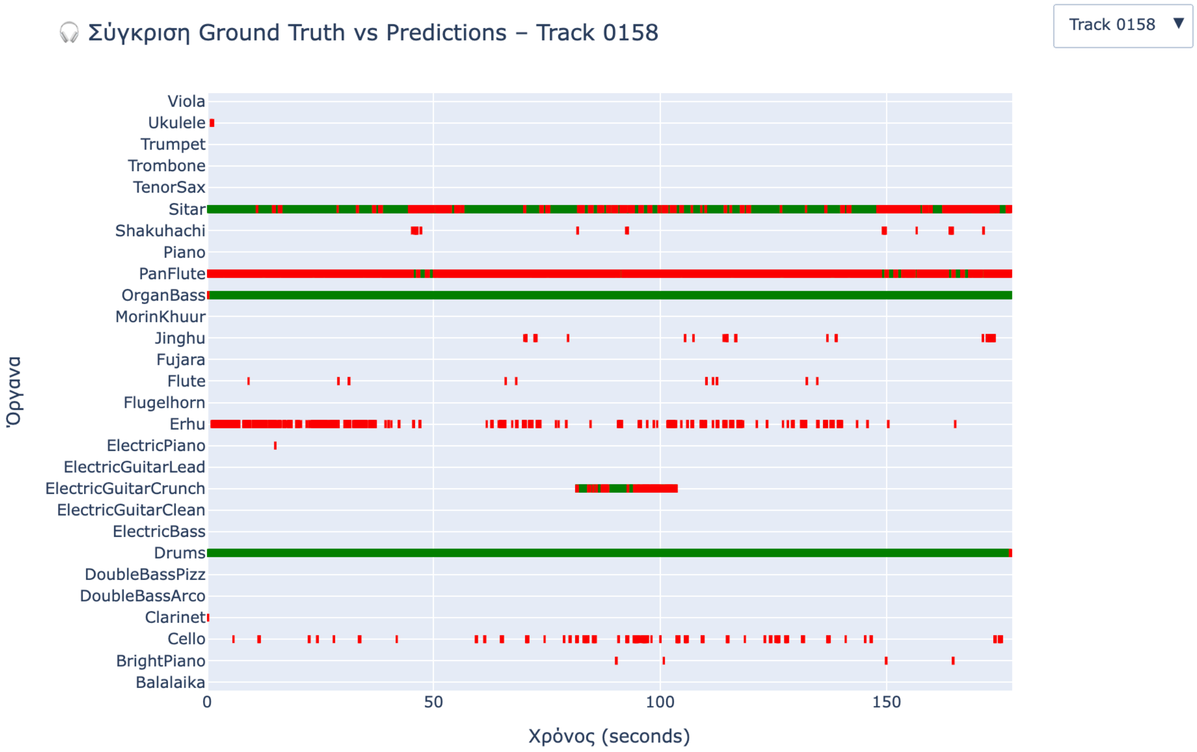


CRNN - 150

Track - 158



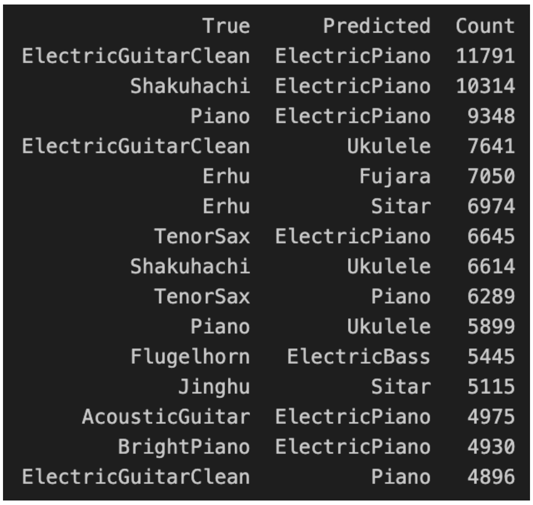
MLP – 300



CRNN - 300

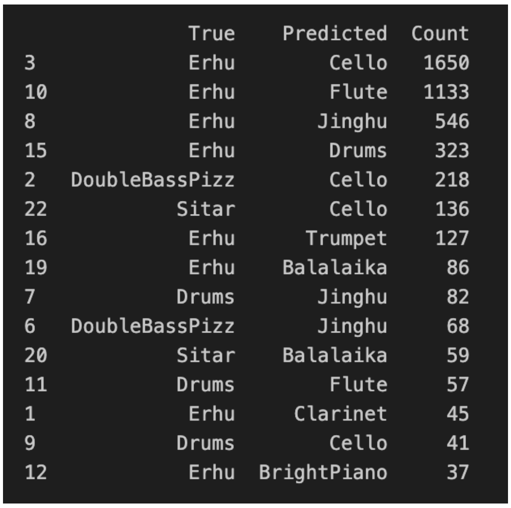
5. Confusion Analysis

- Built confusion matrix of most frequent instrument misclassifications
- Found confusion particularly in harmonically similar instruments
  - There seems to be improvement when increasing the training dataset.
- Visual analysis helps reveal overlap patterns due to co-occurrence



```
                 True      Predicted  Count
ElectricGuitarClean  ElectricPiano  11791
         Shakuhachi  ElectricPiano  10314
              Piano  ElectricPiano   9348
ElectricGuitarClean        Ukulele   7641
               Erhu         Fujara   7050
               Erhu          Sitar   6974
           TenorSax  ElectricPiano   6645
         Shakuhachi        Ukulele   6614
           TenorSax          Piano   6289
              Piano        Ukulele   5899
         Flugelhorn   ElectricBass   5445
             Jinghu          Sitar   5115
     AcousticGuitar  ElectricPiano   4975
        BrightPiano  ElectricPiano   4930
ElectricGuitarClean          Piano   4896
```

CRNN - 150 Songs



```
             True    Predicted  Count
3            Erhu        Cello   1650
10           Erhu        Flute   1133
8            Erhu       Jinghu    546
15           Erhu        Drums    323
2   DoubleBassPizz       Cello    218
22          Sitar        Cello    136
16           Erhu      Trumpet    127
19           Erhu    Balalaika     86
7           Drums       Jinghu     82
6   DoubleBassPizz      Jinghu     68
20          Sitar    Balalaika     59
11          Drums        Flute     57
1            Erhu     Clarinet     45
9           Drums        Cello     41
12           Erhu  BrightPiano     37
```

CRNN - 300 Songs

6. Observations

- MLP performs reasonably well in low-data scenarios
- CRNN performs worse at low data but scales better with more data
- Frequent confusions are observed in overlapping spectral instruments (e.g., flute vs clarinet)
- Weighted losses + oversampling help address class imbalance

7. Future Work

- Add more training data from the full AAM set
- Experiment with **pretrained CNNs** for better audio embeddings
- Evaluate **attention-based** and **Transformer-based** models
- Study **temporal consistency** with post-processing smoothing

8. References

- Choi, K., Fazekas, G., Sandler, M., & Cho, K. (2017). *Convolutional recurrent neural networks for music classification*. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). https://doi.org/10.1109/ICASSP.2017.7952585
- Huang, Y., Wang, J., & Liu, Y. (2022). *Audio Spectrogram Transformer: Learning on the Time-Frequency Representation of Audio*. arXiv preprint arXiv:2104.01778.
- Won, M., Ferraro, D., Han, Y., & Nam, J. (2020). *Evaluation of CNN-based automatic music tagging models*. arXiv preprint arXiv:2006.00751.