Project documentation

PODATKOVNI ZNANSTVENICI S PMF-A

ZAGREB, MAY 2023

Contents

1 Introduction

1.1 Motivation

Music has been an integral part of human culture for thousands of years. With the relatively recent advent of digital technologies, there has been a surge in both the production and the consumption of music. With an unprecedented increase in the number of musical pieces and even genres, it has become increasingly challenging to identify and classify different musical instruments accurately. Automatic instrument recognition systems can aid in solving this problem and provide valuable insights into music analysis and classification.

1.2 Objectives

In this project, we aim to develop an automatic instrument recognition system for the IRMAS dataset using convolutional neural networks (CNNs). Specifically, we have the following objectives:

- Develop three separate CNN models that use mel-spectrograms, modgdgrams, and CQT-chromagrams as input data to classify which musical instruments appear and which don't appear in a given audio file. The models will be trained and evaluated separately to compare their performance.
- Combine the three CNN models using a fusion method to create a single model that achieves better accuracy and robustness in instrument recognition.
- Apply data augmentation techniques to improve the performance of the models, specifically using the WaveGan model to generate synthetic audio samples that can be used to augment the training data.

Our main objective is to create a REST API endpoint that provides a JSON object containing predictions for the instruments present in an audio file. To ensure that the endpoint is easily accessible, we aim to develop a user-friendly web application that enables users to obtain the predictions quickly and conveniently.

1.3 Dataset

The Instrument Recognition using Machine learning for Analysis and Synthesis (IRMAS) dataset is a popular dataset used for evaluating instrument recognition systems. The dataset contains over 6700 audio recordings of various musical instruments, covering 11 instrument categories, including acoustic guitar, piano, and trumpet, among others. The dataset has been annotated by expert musicians, making it a reliable source for training and evaluating machine learning models. The dataset contains certain imbalances in the number of data per instruments, especially in the validation dataset. Imbalances in the datasets are demonstrated in Table 1.

Instrument	Train Count	Validation Count
cello	388	111
clarinet	505	62
flute	451	163
accoustic guitar	637	535
electric guitar	760	942
organ	682	361
piano	721	995
saxophone	626	326
trumpet	577	167
violin	580	211
voice	778	1044

Table 1: Instrument distribution in the training and validation datasets.

2 Research

In recent years, several researchers have proposed methods for instrument recognition using deep learning techniques. Two of the state-of-the-art models that we will be building upon in this project are Han's model and Rajeev Rajan's model.

2.1 Han's model

In 2016, Han *et al.*[3] proposed a deep learning model for instrument recognition using mel-spectrograms as input data. The model achieved state-of-the-art performance on the IRMAS dataset.

The model architecture consists of multiple convolutional layers that capture both local and global patterns in the spectrograms, enabling it to learn discriminative features for each instrument class. Detailed ConvNet architecture is shown in Table 2.

Input size	Description
$1 \times 43 \times 128$	mel-spectrogram
$32 \times 45 \times 130$	3×3 convolution, 32 filters
$32 \times 47 \times 132$	3×3 convolution, 32 filters
$32 \times 15 \times 44$	3×3 max-pooling
$32 \times 15 \times 44$	dropout (0.25)
$64 \times 17 \times 46$	3×3 convolution, 64 filters
$64 \times 19 \times 48$	3×3 convolution, 64 filters
$64 \times 6 \times 16$	3×3 max-pooling
$64 \times 6 \times 16$	dropout (0.25)
$128 \times 8 \times 18$	3×3 convolution, 128 filters
$128 \times 10 \times 20$	3×3 convolution, 128 filters
$128 \times 3 \times 6$	3×3 max-pooling
$128 \times 3 \times 6$	dropout (0.25)
$256 \times 5 \times 8$	3×3 convolution, 256 filters
$256 \times 7 \times 10$	3×3 convolution, 256 filters
$256 \times 1 \times 1$	global max-pooling
1024	flattened and fully connected
1024	dropout (0.50)
11	sigmoid

Table 2: Han's model architecture

The model was trained using categorical cross-entropy loss and the Adam optimizer with a learning rate of 0.001 and a mini-batch size of 128. Dropout regularization with rates of 0.25 and 0.5 was applied after max-pooling and fully connected layers, respectively, to prevent overfitting. The training data was divided into 3.0, 1.5, 1.0, and 0.5 second chunks for analysis, with 1.0 second window proving to be the best. 15% of the data was used for validation. Training was stopped when validation loss did not decrease for more than two epochs. The authors found that LReLU with a very leaky ReLU ($\alpha = 0.33$) performed the best.

One of the unique features of Han's model is its S2 aggregation method, which involves summing and normalizing the feature maps across different time steps to capture the temporal dynamics of the music. This allows the model to learn long-term patterns in the spectrograms and improve its performance on the task of instrument recognition.

The performance of the system was evaluated using precision, recall, and F1 measures. Micro and macro averages were computed for each metric due to the unequal number of annotations for each class. The experiments were repeated three times, and the mean and standard deviation of the output were calculated.

The paper proposes using Convolutional Neural Networks (ConvNet) to identify the predominant instrument in real-world music without any source separation in the preprocessing. The proposed ConvNet architecture outperforms previous state-of-the-art approaches in the task. The study shows that advances in neural networks in image processing are transferable to audio processing.

2.2 Rajeev Rajan's model

In their 2021 paper, Rajeev Rajan *et al.*[1] proposed a deep learning model that aims to address the limitations of existing approaches for multiple predominant instrument recognition in polyphonic music. The model takes advantage of the complementary information captured by two types of time-frequency representations of audio signals, mel-spectrograms and modgd-grams, and integrates them using a score-level fusion.

Specifically, the model first extracts the mel-spectrogram and modgd-gram representations from the input audio signals using a short-time Fourier transform (STFT) and a modified group delay (MGD) method, respectively. The mel-spectrogram and modgd-gram representations are then fed into two separate CNN branches, each consisting of multiple convolutional and pooling layers. The model architecture is the same as for the Han's model. The individual scores of the two CNN branches are fused to make a decision.

For data augmenation, they used WaveGAN v2 to generate polyphonic files with the leading instrument required for training. The WaveGAN is trained for 2000 epochs using audio files that are three seconds long for each instrument class. The purpose is to generate audio files that are similar to the input files. As a result, a total of 6585 audio files are generated, including cello (625), clarinet (482), flute (433), acoustic guitar (594), electric guitar (732), organ (657), piano (698), saxophone (597), trumpet (521), violin (526), and voice (720).

The model showed an 8% improvement in micro F1 score and a 9% improvement in macro F1 score over Han's model on the same dataset. The Fusion-CNN with data augmentation achieved a micro and macro F1 score of 0.65 and 0.60 respectively, which is 1.56% and 5.26% higher than the Mel-spectrogram-CNN with data augmentation. This indicates that modgdgram added complementary information to the spectrogram approach, highlighting the importance of the fusion framework for the proposed task. The proposed Mel-spectrogram-CNN approach showed similar performance to Han's model.

2.3 Incorporating CQT-chromagrams

While researching related work, we found several papers that focused on using pitchgrams for tasks such as chord recognition and harmonic analysis. Given that pitch information is an essential component of music and can help distinguish between similar-sounding instruments, we decided to incorporate pitchgrams into our models. We aim to improve Han's and Rajeev Rajan's models by incorporating pitchgrams as an additional input modality, which we believe will lead to improved performance and accuracy in instrument recognition.

A pitchgram represents the distribution of the spectral energy in different pitch classes. It is usually obtained by summing up the energy in frequency bins that belong to the same pitch class. Pitchgrams are commonly used in music information retrieval tasks such as melody extraction and key detection.Specifically, as inputs for our model, we used CQT-chromagrams, which is strongly related to pitchgram. CQT-chromagram will be explained in more detail in subsubsection 3.1.3. All three spectrograms are shown in Figure 1.

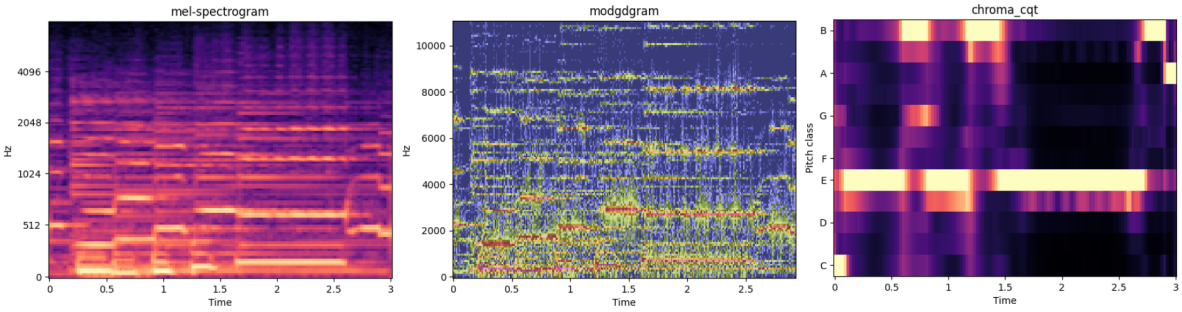

Figure 1: Visualization of the spectrograms

3 Methodology

3.1 Data preprocessing

Before moving to the details about the model, we need to explain how the raw data from the IRMAS dataset was preprocessed. We preprocess the audio data to extract relevant features that can be used for training the instrument recognition model. Specifically, we extract three types of features: mel-spectrogram, modgdgram, and CQT chromagram. These features are widely used in the audio processing domain and have shown promising results. In the following sections, we will describe each of these feature extraction methods in detail and explain how they are useful for instrument recognition.

3.1.1 Mel-spectrogram

In the project, we followed the approach outlined in Han's paper [3] for data preprocessing. The first step involves converting the stereo input audio to mono by taking the mean of the left and right channels. Then, the audio is downsampled from the original 44,100 Hz sampling frequency to 22,050 Hz. Additionally, all audios are normalized by dividing the time-domain signal with its maximum value.

After this initial processing step, the time-domain waveform is converted to a time-frequency representation using the short-time Fourier transform (STFT) with a window size of 1024 samples (approximately 46 ms) and a hop size of 512 samples (approximately 23 ms). The resulting linear frequency scale-obtained spectrogram is then converted to a mel-scale with 128 mel-frequency bins. This setting preserves the harmonic characteristics of the music while significantly reducing the dimensionality of the input data. The magnitude of the mel-frequency spectrogram is compressed using a natural logarithm. Finally, mel-spectrogram corresponding to an audio file is divided into three smaller mel-spectrograms corresponding to 1 second clip from an audio file. This way, one audio file from the training dataset corresponds to three pieces of input data for the model. Input mel-spectrograms are of the shape $(1, 128, 100)$.

```
import matplotlib.pyplot as plt
import librosa
import librosa.display
import numpy as np

AUDIO_FILE =
"../data/raw/IRMAS_Training_Data/flu/018__[flu][nod][pop_roc]0421__3.wav"
y, sr = librosa.load(AUDIO_FILE, sr = 22050, mono = True)

S = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=128, fmax=8000)
S_dB = librosa.power_to_db(S, ref=np.max)
librosa.display.specshow(S_dB, x_axis='time', y_axis='mel', sr=sr, fmax=8000)
plt.title('mel-spectrogram')
plt.show()
```

This code is a short demonstration of mel-spectrogram extraction from a file. Mel-spectrogram produced is displayed in Figure 2.

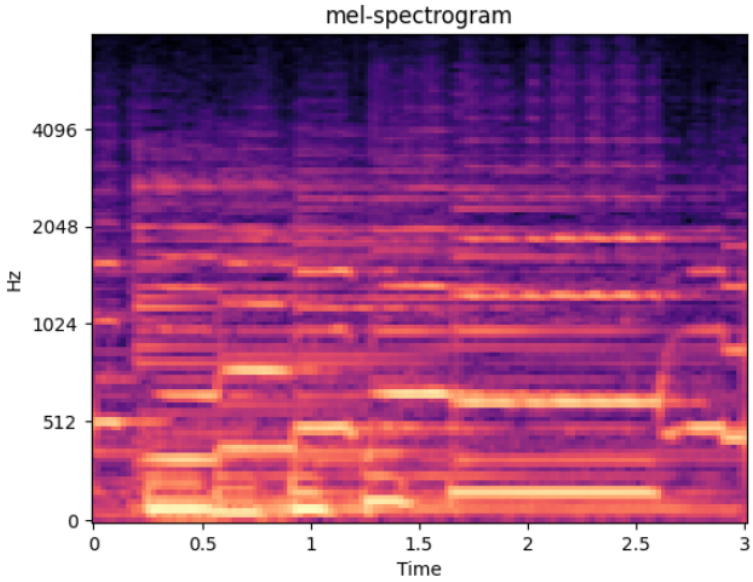

Figure 2: mel-spectrogram

The provided IRMAS train dataset was divided into training and validation data. Specifically, the training data was allocated 85% of the original train dataset, while the remaining 15% was designated as validation data. Number of files in the IRMAS train dataset is $N = 6716$, corresponding to 20148 mel-spectrograms. Shapes of the tensors used for training and validation steps are shown in Table 3.

Tensor	**Shape**
X_{train}	$(17126, 1, 128, 100)$
y_{train}	$(17126, 11)$
X_{val}	$(3022, 1, 128, 100)$
y_{val}	$(3022, 11)$

Table 3: Shape of the tensors used for training and validation.

The test dataset provided consists of 2874 audio files of various lengths, but most of them are 20 seconds long. To extract the features from the audio files, we divide the mel-spectrogram of each file into smaller mel-spectrograms, with each mel-spectrogram corresponding to a 1-second chunk of the audio file. Each audio file is assigned a label list based on the musical instruments present in the file. To organize the data, we use OrderedDict structures, where the key corresponds to the ordinal number of the file, and the value corresponds to a list of mel-spectrograms obtained from the file. Similarly, the label list for each file is associated with the corresponding key.

For instance, consider a 20-second long file containing the sounds of a cello and a clarinet. We extract 20 mel-spectrograms, each corresponding to a 1-second chunk of the audio file, and place them in a list. The key in the input data structure will be the ordinal number of the file, and the value will be the list of mel-spectrograms. The output data structure will also use the ordinal number of the file as the key, and the corresponding value will be a label list indicating the presence or absence of various musical instruments in the file, for example, $[1, 1, 0, ...]$.

3.1.2 Modgdgram

Group delay function is a signal processing technique used to estimate the power spectrum of a signal by taking the negative derivative of the continuous phase spectrum. The group delay function is given with

$$\tau_x(e^{j\omega}) = \frac{X_R(e^{j\omega})Y_R(e^{j\omega}) + Y_I(e^{j\omega})X_I(e^{j\omega})}{|X(e^{j\omega})|^2}, \tag{1}$$

where $X(e^{j\omega})$ and $Y(e^{j\omega})$ are Fourier transforms of signal $x[n]$ and $nx[n]$ respectively. Subscripts R and I stand for real and imaginary parts. However, group delay function is not useful for many practical signals, particularly speech signals, where zeros are located close to the unit circle. The presence of these zeros results in peaks that lead to a poor estimation of the power spectrum. To overcome this problem, Yegnanarayana and Murthy [7] proposed cepstral smoothing which eliminates the effects of zeros introduced by the excitation component of the speech signal. The modified group delay function is a group delay function computed through cepstral smoothing.

$$\tau_c(e^{j\omega}) = \frac{X_R(e^{j\omega})Y_R(e^{j\omega}) + Y_I(e^{j\omega})X_I(e^{j\omega})}{|S(e^{j\omega})|^2} \tag{2}$$

Murthy and Gadde [5] further modified the group delay function, introducing 2 new parameters, α and γ, used to adjust the bandwidth and sharpness of the peaks as well as compression. The new modified group delay function is given with

$$\tau_m(e^{j\omega}) = sign\left|\frac{X_R(e^{j\omega})Y_R(e^{j\omega}) + Y_I(e^{j\omega})X_I(e^{j\omega})}{|S(e^{j\omega})|^{2\gamma}}\right|^{\alpha}, \tag{3}$$

where *sign* is the sign of the original modified group delay function from (1). For the computation of modified group delay function of a signal, we followed the algorithm by Murthy and Gadde [5] which is explained in Table 4. We start by dividing the original audio signal into frames using a frame size of

Step	Description
1.	Let $x(n)$ be the given sequence.
2.	Compute the discrete Fourier transform (DFT) of $x(n)$ and of $nx(n)$. Let these be $X(k)$ and $Y(k)$, respectively.
3.	Compute the cepstrally smoothed spectra of $\|X(k)\|$. Let this be $S(k)$.
4.	Compute the modified group delay functions as in (3).
5.	Tune the parameters α and γ appropriately for a given environment.

Table 4: Algorithm for the computation of modified group delay function

50ms and a hop size of 10ms. We follow the parameters $\alpha = 0.9$ and $\gamma = 0.5$ suggested by Rajan [1]. For each of the frames, we compute the modified group delay function (3), resulting in a 2D modgdgram. To have modgdgrams with the same shape as mel-spectrograms, we reduce the frequency dimension of the modgdgram to only the first 128 bins. Since the shapes of the modgdgrams are the same as those of mel-spectrograms, the datasets for the model are prepared following the same process described in subsubsection 3.1.1. In Figure 3, we can see modgdgram.

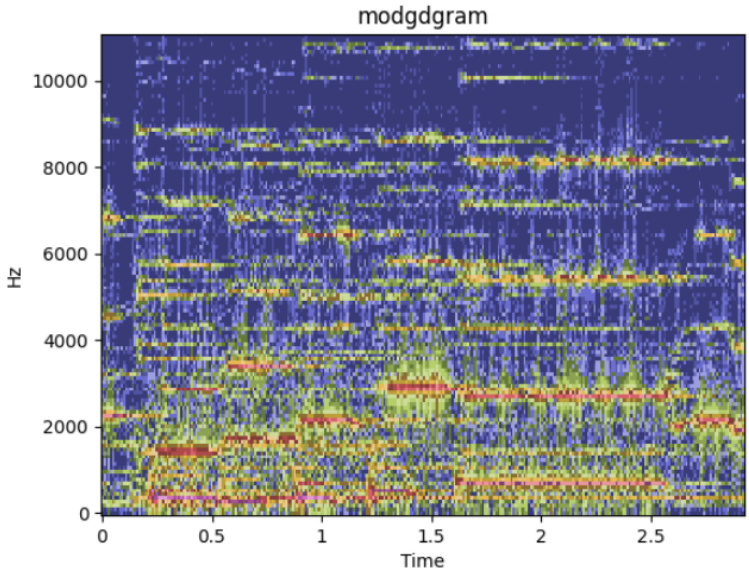

Figure 3: Modgdgram

3.1.3 CQT Chromagram

A CQT-chromagram is a variant of the chromagram that is derived from the Constant-Q Transform (CQT). The CQT is a time-frequency representation of an audio signal that is similar to the spectrogram, but with a logarithmically spaced frequency axis. A CQT chromagram is obtained by computing the energy of the audio signal in different pitch classes after performing the CQT. The resulting chromagram has a higher resolution in the low-frequency range and is well-suited for tasks such as chord recognition and harmonic analysis. Using the built-in function provided by Librosa, it is very easy to extract CQT-chromagram from a file.

```
import matplotlib.pyplot as plt
import librosa
import librosa.display
import numpy as np

AUDIO_FILE =
"../data/raw/IRMAS_Training_Data/flu/018__[flu][nod][pop_roc]0421__3.wav"
y, sr = librosa.load(AUDIO_FILE, sr = 22050, mono = True)

chroma_cq = librosa.feature.chroma_cqt(y=y, sr=sr)
librosa.display.specshow(chroma_cq, y_axis='chroma', x_axis='time')
plt.title('chroma_cqt')
plt.show()
```

The inclusion of CQT-chromagram in our models led to a significant improvement, suggesting that it provides additional complementary information to that obtained from mel-spectrograms and modgdgrams. CQT-chromagram is shown in Figure 4. The preparation of datasets for the model is identical to the one

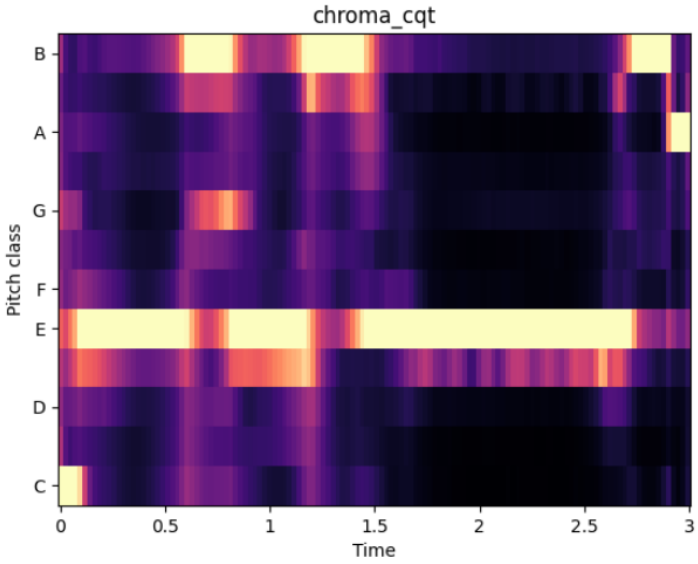

Figure 4: CQT-chromagram

described in subsubsection 3.1.1. The only difference is the shape of the CQT-chromagrams $(1, 36, 100)$, which is different than mel-spectrogram shape $(1, 128, 100)$

3.2 Model architecture

In this section, we will describe the convolutional neural network (CNN) models used for our instrument recognition task. All of our models are based on the CNN architecture, which has been shown to be effective for various audio classification tasks. Specifically, we use the same CNN architecture for both the mel-spectrogram and modgdgram input representations, while a separate CNN architecture is used for the CQT-chromagram input. In both of our models, every convolutional layer was followed by leaky ReLU(LReLU) activation function. The ReLU activation function is defined as

$$y_i = max(0, z_i) \tag{4}$$

However, LReLU was shown to perform better than ReLU for instrument recognition tasks, so we decided to use LReLU, which is defined as

$$y_i = \begin{cases} z_i, & \text{if } z_i \geq 0 \\ \alpha z_i, & \text{otherwise} \end{cases} \tag{5}$$

In the following subsections, we will describe the details of each model's architecture, including the number and size of the convolutional layers, the pooling layers, the fully connected layers and the activation functions. Our ConvNets share similarities with those developed by Han [3] and Rajan [1], although with some modifications.

3.2.1 CNN for mel-spectrogram and modgdgram

This model is a convolutional neural network with 8 convolutional layers, each followed by a leaky ReLU activation function with negative slope 0.33. Outputs of 4 of these convolutional layers are passed

through max pooling and dropout. The network takes in a tensor of shape $1 \times 128 \times 100$ as input, representing a mel-spectrogram or modgdgram. The first layer has a kernel size of 3 and 8 output channels. The output of this layer is then passed through a leaky ReLU activation function, followed by a second convolutional layer with a kernel size of 3 and 16 output channels. The output is then passed through another leaky ReLU activation function, max pooling, and dropout. This pattern is repeated with additional layers, with increasing numbers of output channels. The detailed model architecture, along with the input size in each layer is illustrated in Table 5.

Input size $\rightarrow$ output size	Description
$1 \times 128 \times 100 \rightarrow 8 \times 128 \times 100$	3×3 convolution, 8 filters
$8 \times 128 \times 100 \rightarrow 16 \times 128 \times 100$	3×3 convolution, 16 filters
$16 \times 128 \times 100 \rightarrow 16 \times 42 \times 33$	3×3 max-pooling, 3×3 stride
	dropout (0.25)
$16 \times 42 \times 33 \rightarrow 24 \times 42 \times 33$	3×3 convolution, 24 filters
$24 \times 42 \times 33 \rightarrow 32 \times 42 \times 33$	3×3 convolution, 32 filters
$32 \times 42 \times 33 \rightarrow 32 \times 14 \times 11$	3×3 max-pooling, 3×3 stride
	dropout (0.25)
$32 \times 14 \times 11 \rightarrow 64 \times 14 \times 11$	3×3 convolution, 64 filters
$64 \times 14 \times 11 \rightarrow 128 \times 14 \times 11$	3×3 convolution, 128 filters
$128 \times 14 \times 11 \rightarrow 128 \times 4 \times 3$	3×3 max-pooling, 3×3 stride
	dropout (0.25)
$128 \times 4 \times 3 \rightarrow 256 \times 4 \times 3$	3×3 convolution, 256 filters
$256 \times 4 \times 3 \rightarrow 512 \times 4 \times 3$	3×3 convolution, 512 filters
$512 \times 4 \times 3 \rightarrow 512 \times 1 \times 1$	3×3 max-pooling, 3×3 stride
	dropout (0.25)
	global max-pooling
$512 \times 1 \times 1 \rightarrow 1024$	flattened and fully connected
	dropout (0.50)
$1024 \rightarrow 11$	fully connected
	softmax

Table 5: Mel-spectrogram model architecture. Leaky ReLu activation function with negative slope 0.33 is applied after every convolutional layer, but is omitted for simplicity. First column is empty if the layer does not change the size of the tensor.

After the last convolutional layer, the output is passed through a global max pooling layer and then flattened. The flattened output is then passed through two fully connected layers, each followed by a dropout layer, and finally to the output layer. The output layer has 11 nodes, corresponding to the 11 possible labels in the dataset. Softmax activation function is applied to the output layer.

3.2.2 CNN for CQT-chromagram

After experimenting with Han and Rajan's model architectures, we developed our own model for processing CQT-chromagrams. Our model is composed of six convolutional layers with a leaky ReLU activation function applied afterwards, three of which are followed by both max-pooling and dropout layers. The detailed architecture of our model can be found in Table 6.

Similar to the previous models using mel-spectrograms and modgdgrams, the output from the last convolutional layer is fed into a global max-pooling layer and flattened. The flattened output is then passed through two fully connected layers, first of which is followed by a dropout layer, before finally

Input size → output size	Description
$1 \times 36 \times 100 \rightarrow 24 \times 36 \times 100$	3×3 convolution, 24 filters
$24 \times 36 \times 100 \rightarrow 32 \times 36 \times 100$	3×3 convolution, 32 filters
$32 \times 36 \times 100 \rightarrow 32 \times 12 \times 33$	3×3 max-pooling, 3×3 stride
	dropout (0.25)
$32 \times 12 \times 33 \rightarrow 64 \times 12 \times 33$	3×3 convolution, 64 filters
$64 \times 12 \times 33 \rightarrow 128 \times 12 \times 33$	3×3 convolution, 128 filters
$128 \times 12 \times 33 \rightarrow 128 \times 4 \times 11$	3×3 max-pooling, 3×3 stride
	dropout (0.25)
$128 \times 4 \times 11 \rightarrow 256 \times 4 \times 11$	3×3 convolution, 256 filters
$256 \times 4 \times 3 \rightarrow 512 \times 4 \times 11$	3×3 convolution, 512 filters
$512 \times 4 \times 11 \rightarrow 512 \times 1 \times 3$	3×3 max-pooling, 3×3 stride
	dropout (0.25)
$512 \times 1 \times 3 \rightarrow 512 \times 1 \times 1$	global max-pooling
$512 \times 1 \times 1 \rightarrow 1024$	flattened and fully connected
	dropout (0.50)
$1024 \rightarrow 11$	fully connected
	softmax

Table 6: CQT-chromagram model architecture. Leaky ReLu activation function with negative slope 0.33 is applied after every convolutional layer, but is omitted for simplicity. First column is empty if the layer does not change the size of the tensor.

reaching the output layer with 11 nodes representing the 11 possible labels in the dataset. The output layer uses a softmax activation function.

3.3 Model training

During the training process, we aimed to minimize the categorical cross-entropy loss between the predicted and target values. We used Adam optimizer [4] with a learning rate 0.001. Batch size was set to 128. We trained the model on Nvidia GPU, with 4GB of memory. During the training phase, we incorporated dropout regularization to avoid overfitting of the neural network to the training data. Dropout randomly drops some units from the neural network, making it perform better on unseen data. This technique is especially effective for preventing overfitting in fully connected layers, which are more prone to overfitting than convolutional layers. During the training, we monitored validation loss and saved the best model for each of the input types(mel-spectrogram, modgdgram and CQT-chromagram). That way, we produced 3 models which are later fused. For the sake of simplicity, training procedure is the same for each of the three models described in subsection 3.2. Usually, validation loss was minimal around the 20th epoch. Behaviour of validation loss during the training process is illustrated in Figure 5.

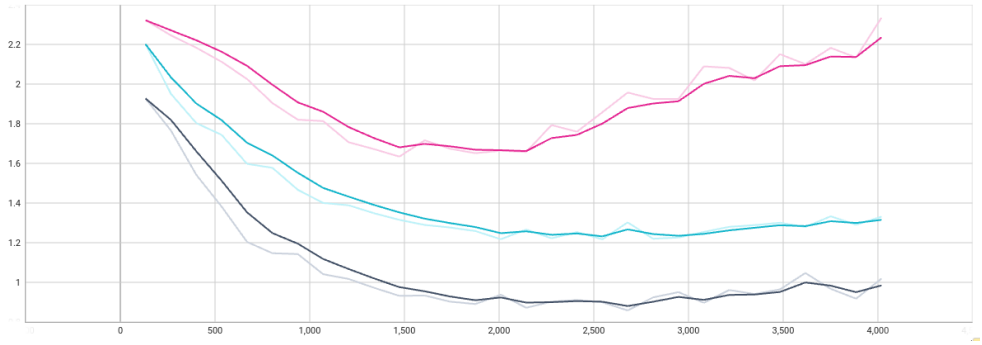

Figure 5: Validation loss during the training. Black curve represents the mel-based, blue the modgd-based, and pink the CQT-based CNN

3.4 Fusion

First, we start by explaining what is threshold in the context of our model. The threshold is a decision boundary that separates the positive and negative predictions of a model. For example, if the threshold for a model is set to 0.5, any prediction greater than or equal to 0.5 will be classified as positive, and any prediction less than 0.5 will be classified as negative. After training the model, we evaluated its performance on a test set. We found that the model's predictions varied widely across different instruments. Therefore, we decided to set a separate threshold for each instrument.

To determine the best threshold for each model, we used a grid search approach. We varied the threshold values from 0 to 1 in intervals of 0.01 and evaluated the performance of the model on a validation set using each of these thresholds. We measured the F1 score of the model at each threshold value and selected the threshold that yielded the best F1 score. We considered optimizing thresholds with respect to accuracy, but concluded F1 score is the more appropriate metric for the task. The grid search approach was a systematic way of searching for the optimal hyperparameters of the models. By varying the threshold in small intervals, we were able to explore a range of threshold values and find the one that yields the best performance. In addition to setting a separate threshold for each instrument based on individual models, we also employed a fusion method to improve the overall performance of our model. To achieve this, we calculated three thresholds for each instrument - one for each of the three models used in our fusion approach. This resulted in a total of 33 thresholds (3 models x 11 instruments) for our fusion model. To assess the impact of each individual model on the performance of the fusion model, we generated three graphs, with each graph showing the F1 score of the fusion model for the flute as a function of the threshold for a specific model, while keeping the other two thresholds fixed. For example, Figure 6 displays the F1 score of the fusion model for the flute as a function of the mel, modgd and CQT thresholds while keeping the other two thresholds fixed. The figure clearly shows the maximum point on each of the graphs.

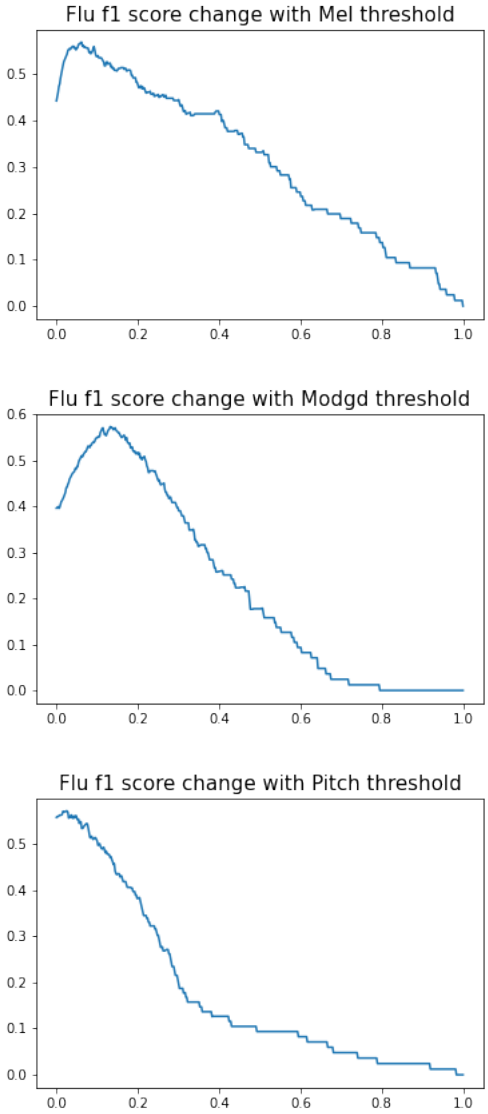

Figure 6: F1 grid search for flu

Once we found the best threshold for each model, we fuse the models using our algorithm which is described in Table 7.

Step	Description
1.	Let t_1, t_2 and t_3 be thresholds for mel-based, modgd-based and cqt-based CNN respectively for some instrument.
2.	Let p_1, p_2 and p_3 be predictions of mel-based, modgd-based and cqt-based CNN respectively for some instrument.
3.	If $p_1 > t_1$, $p_2 > t_2$ and $p_3 > t_3$, predict 1
4.	Otherwise, predict 0.

Table 7: Fusion algorithm

Our fusion method made a 0.9% improvement in accuracy upon the Rajan's fusion method [1], which

is a significant improvement in accuracy. Comparison is shown in Figure 7.

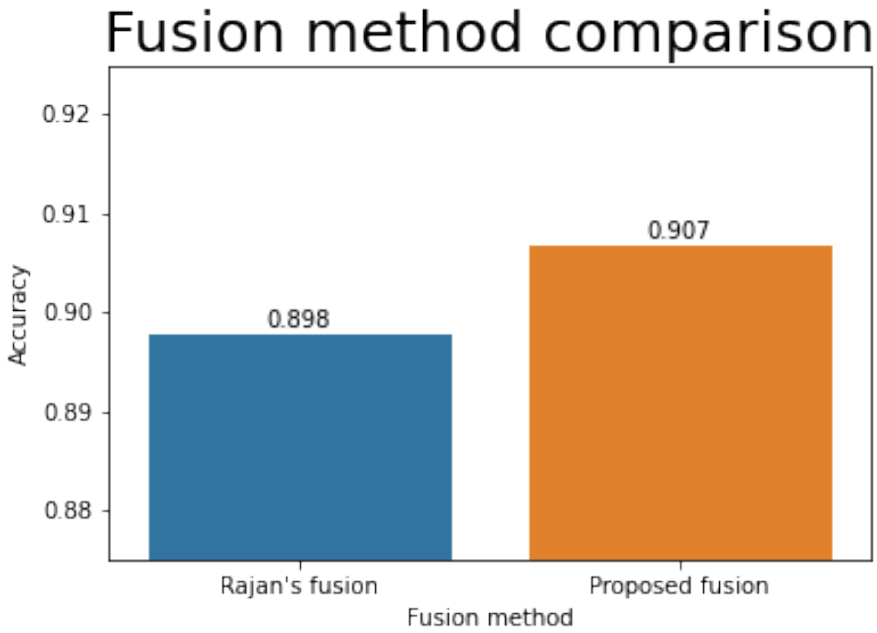

Figure 7: Comparison of proposed fusion method and Rajan's fusion method.

The threshold fusion method allowed us to combine the predictions of the three models and make a more accurate prediction about which instruments were present in an audio file.

3.5 Model evaluation

The main metric we used for evaluation of our model is F1 score. F1 score is a commonly used metric for evaluating the performance of classification models. It takes into account both precision and recall, which are two important metrics in binary classification tasks. Precision measures the percentage of correctly classified positive instances out of all predicted positive instances, while recall measures the percentage of correctly classified positive instances out of all actual positive instances. F1 score is the harmonic mean of precision and recall and is defined as

$$F1 = 2 \cdot \frac{P \cdot R}{P + R} = \frac{2TP}{2TP + FP + FN}, \tag{6}$$

where P is precision, R is recall, and TP, FP and FN stand for true positive, false positive and false negative respectively. F1 score ranges from 0 to 1, where a score of 1 indicates perfect precision and recall, and a score of 0 indicates that either precision or recall is 0. F1 score is often used in imbalanced datasets where the number of instances in each class is different, as it balances both precision and recall for each class. There are two ways to compute F1 score, micro-averaging and macro-averaging. In micro-averaging, we compute F1 score globally by considering all the true positives, false positives, and false negatives across all the classes. In other words, we aggregate the contributions of all the classes into a single F1 score. This is particularly useful when we have imbalanced datasets as it gives equal importance

to all the classes. Mathematically, micro-averaged F1 score is given by

$$F1_{micro} = \frac{2TP_{micro}}{2TP_{micro} + FP_{micro} + FN_{micro}}, \tag{7}$$

where TP_{micro} is the total number of true positives across all the classes, FP_{micro} is the total number of false positives across all the classes, and FN_{micro} is the total number of false negatives across all the classes. In macro-averaging, we compute F1 score for each class separately and then take the average across all the classes. In other words, we compute the F1 score for each class as if it were the positive class and then take the unweighted average of these scores. This is useful when we want to evaluate the performance of the classifier on each class separately. Macro-averaged F1 score is given by

$$F1_{macro} = \frac{1}{n}\sum_{i=1}^{n} F1_i, \tag{8}$$

where n is the total number of classes and $F1_i$ is the F1 score for the i^{th} class. In the context of instrument recognition, micro-averaged F1 score is useful when we want to evaluate the overall performance of the classifier across all the classes, while macro-averaged F1 score is useful when we want to evaluate the performance of the classifier on each class separately.

In the Lumen Data Science 2023 competition, we utilized accuracy as the primary evaluation metric for our classification models. Accuracy quantifies the fraction of correctly classified examples among the total number of instances in the test set. It is a widely used metric in classification tasks due to its interpretability. Specifically, in our case, for each element in the test set, we obtained eleven predictions corresponding to each of the eleven instruments represented in the IRMAS dataset. Hence, the total number of predictions was equal to $11 \cdot N$, where N denotes the number of elements in the test set. To calculate the accuracy, we determined the proportion of correctly classified predictions out of the total number of predictions.

It is worth noting that accuracy can be misleading if the dataset is imbalanced, meaning that one class is more prevalent than the others. In such cases, a high accuracy may be achieved simply by always predicting the majority class.

In this case, the IRMAS validation dataset is somewhat imbalanced, with roughly 85% of test examples having 0 as the prediction, meaning that the instrument does not appear in an audio file. The imbalance can be seen from the Figure 8 since most audio files have only 1 or 2 instruments appearing. That is why it is worth noting that F1 score is often a better metric for evaluating classification models, as it takes into account both precision and recall. F1 score provides a balanced measure of the model's performance across all classes, making it particularly useful when the dataset is imbalanced.

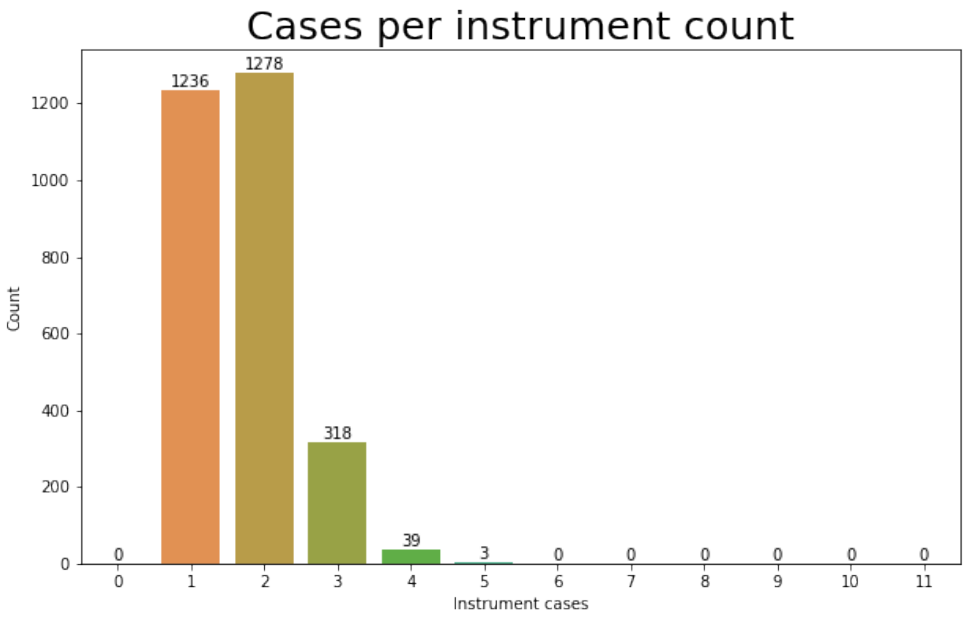

Figure 8: Number of audio files containg certain number of instruments

While F1 score may be a better metric for evaluating this type of model, we chose to use accuracy for the purpose of the competition, since it's the official metric used to test our models. It is a straightforward metric that is easy to calculate and interpret, and it allowed us to quickly assess the model's overall performance on the test set. However, we recommend that future evaluations of this model consider using F1 score as an additional metric to provide a more comprehensive evaluation of its performance.

4 Model performance and results

In this section, we evaluate the performance of our deep learning models on the test dataset.

4.1 Mel-spectrogram evaluation

Using only feature extraction from mel-spectrograms, the deep learning model achieved an accuracy of 90% on the test dataset as shown in Figure 9, indicating that it was able to accurately recognize which instruments appear on the polyphonic audio files in the IRMAS dataset. However, it is worth noting that the model's performance varied across different instruments, with some being predicted more accurately than others. In particular, the model was able to predict human voice more accurately than other instruments. One possible reason for this could be that human voice has unique acoustic properties that make it easier to identify in a polyphonic audio file. Another possibility is that the model was biased towards human voice due to the higher prevalence of this instrument in the dataset. On the other hand, the model had more difficulty predicting certain instruments such as clarinet, cello or organ, which may have more complex and variable acoustic properties, but also have less data in both the training and testing datasets as shown in Table 1. The micro and macro F1 scores of 0.625 and 0.537 respectively can be seen in Figure 10.

4.2 Fused mel-spectrogram and modgdgram evaluation

For the reasons explained in subsubsection 3.1.2 modgdgram is introduced as an additional ConvNet input feature improving the accuracy to 90.4% as shown in Figure 9. Modgdgram improved both micro and macro F1 scores to 0.641 and 0.550 respectively, as shown in Figure 10. Similar to the performance of the model that utilized only mel-spectrograms, the performance of the current model also varied depending on the instrument.

4.3 Fused mel-spectrogram, modgdgram and CQT-chromagram evaluation

Unlike mel-spectrogram or modgdgram, the CQT-chromagram represents the chroma content, which is a more musically relevant measure of the pitch of a sound, as was described in detail in subsubsection 3.1.3. Suprisingly, the CQT-chromagram alone does not have a good accuracy score, with accuracy just over 50%, but again it turned out that it added some complementary features to the mel-spectrogram and modgd-gram which allowed the further improvement upon the model, reaching the accuracy of 90.8% as can be seen in Figure 9. The model achieved micro and macro F1 scores of 0.648 and 0.561 respectively as shown in Figure 10. We demonstrate the variation in metrics by comparing three different feature extraction options in building the model, namely using only mel-spectrograms, using both mel-spectrograms and modgd-grams, and using all three feature extraction options. The comparison is presented in Figure 9 and Figure 10.

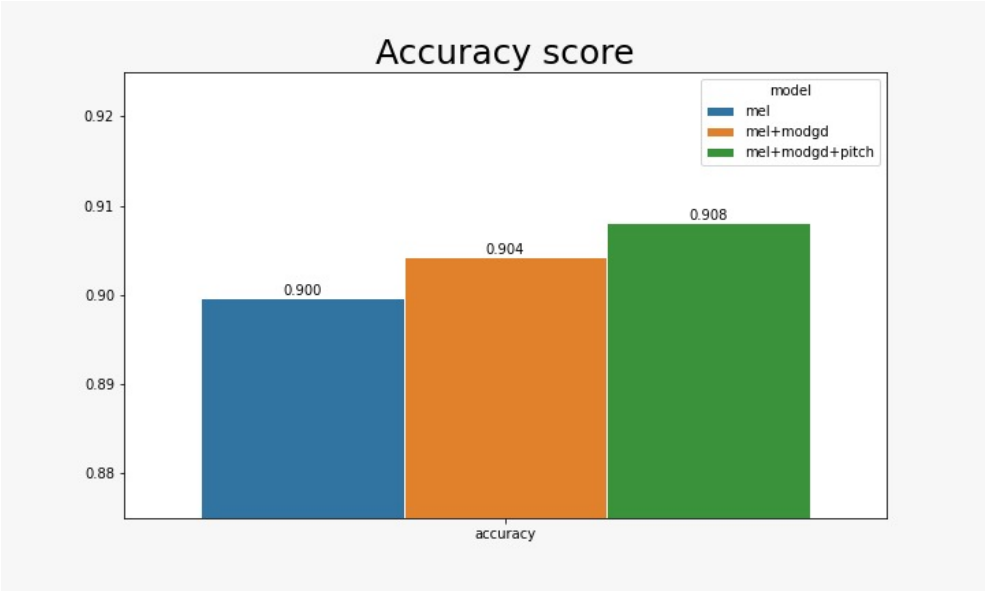

Figure 9: Accuracy comparison of three aproaches

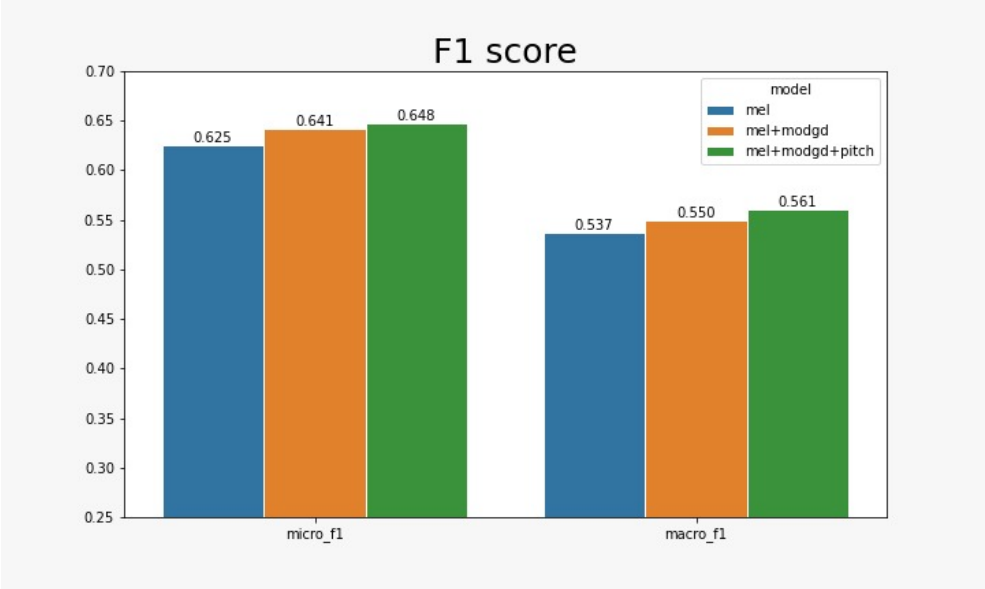

Figure 10: F1 score comparison of three aproaches

4.4 Comparison to state-of-the-art models

In order to present the performance of our model in more detail, we report several metrics to asses the model's performance such as precision, recall, micro and macro F1 score as shown in Table 8.

Metric	Score
Accuracy	0.908
Precision	0.830
Recall	0.420
Macro F1 score	0.648
Micro F1 score	0.561

Table 8: Metrics for the model

Our model achieved an accuracy of 0.908, correctly classifying 90.8% of all the test samples. The precision score of 0.83 indicates that among the predicted positive samples, 83% were actually positive, while the recall score of 0.42 suggests that our model correctly identified 42% of the positive samples.

We used micro and macro F1 score to evaluate the overall performance of our classifier. The micro F1 score calculates the F1 score globally, treating all the classes as a single class, while the macro F1 score calculates the F1 score for each class independently and then averages them. Our score of 0.648 for micro F1 and 0.561 for macro F1 suggest that there is a significant difference in classification score between the two classes.

We compared our model's performance to two state-of-the-art models mentioned in section 2. Our model outperformed Han's model [3] in terms of precision, F1-micro score, and F1-macro score, but performed worse in recall. Our model also performed better than the model proposed by Rajeev Rajan [1] before they used data augmentation with WaveGAN, in terms of F1-micro score and F1-macro score.

In Table 9 and Table 10, we can see how the evaluation changed when fusing two or three models into one prediction. The column percentage difference represents the difference in percentage between the accuracy of fused mel-spectrogram and modgd-gram as opposed to all three models fused together.

Instrument	**mel**	**mel + modgd**	**mel + modgd + CQT chroma**	**percentage difference**
cello	0.963	0.964	0.966	0.181
clarinet	0.978	0.978	0.979	0.036
flute	0.955	0.558	0.957	0.146
accoustic guitar	0.866	0.870	0.870	0.482
electric guitar	0.806	0.805	0.805	-0.043
organ	0.890	0.892	0.892	0.039
piano	0.754	0.756	0.764	1.058
saxophone	0.915	0.930	0.938	0.898
trumpet	0.958	0.959	0.960	0.036
violin	0.945	0.951	0.952	0.110
voice	0.910	0.915	0.915	0.038

Table 9: Accuracy per instrument

Instrument	**mel**	**mel + modgd**	**mel + modgd + CQT chroma**	**percentage difference**
cello	0.297	0.325	0.354	9.187
clarinet	0.152	0.213	0.256	19.597
flute	0.468571	0.558	0.569	1.981
accoustic guitar	0.507	0.518	0.535	3.273
electric guitar	0.646	0.685	0.686	0.114
organ	0.380	0.414	0.415	0.089
piano	0.603	0.611	0.615	0.678
saxophone	0.633	0.659	0.708	7.383
trumpet	0.565	0.596	0.605	1.480
violin	0.575	0.611	0.612	0.205
voice	0.871	0.875	0.878	0.328

Table 10: F1 score per instrument

5 Conclusion

In this project, we have presented a deep learning model for instrument recognition in audio files on the widely used IRMAS dataset. Our model is based on a combination of three sub-models: a mel-based model, a modgd-based model, and a CQT-based model. In order to achieve an accurate and robust instrument recognition model, we fused the three models utilizing fusion algorithm described in Table 7 from the subsection 3.4.

Through our experimentation, we found that our approach was highly effective for instrument recognition, achieving an overall accuracy of 90.8% on the test set. When observing the F1 score, we can see our model performed well across a range of instruments as can be seen in Table 10, including electric guitar, saxophone and especially human voice, having an F1 score for those instruments around 0.7 demonstrating its robustness and versatility. On the other hand, the model did not perform so well with instruments such as clarinet, cello and organ due to less data available for those instruments in both the training and validation datasets and the fact that they may have more complex and variable acoustic properties.

Contrary to our expectations, we found that standard data augmentation techniques such as random cropping, time stretching, and frequency masking were not successful in improving the model, even though it increased the size of our training set almost threefold. We believe that was due to the fact that our models were already highly accurate in recognizing the important features from the audio signals that such augmentation techniques did not provide any new input to the models. However, modern technologies and deep learning models such as WaveGAN [2], allow us to create synthetic audio data similar to the data we already have, and we believe that with that the models would improve significantly because they would have different features from the audio signals and would be able to recognize instrument features more precisely.

Overall, the model proposed here represents a slight improvement to the works in field of instrument recognition in audio files that are mentioned in the section 2 and are considered to be state-of-the-art. By leveraging the power of deep learning and a combination of different feature extraction techniques, we were able to develop a model that can accurately recognize a wide range of instruments in different types of audio files.

Moving forward, there are several areas for future work that could improve upon our results. For example, we could explore the use of different types of neural network architectures to further improve the performance of our model. Additionally, we could investigate the use of additional features, such as spectral contrast or spectral roll-off, to further enhance the accuracy and robustness of our model, alongside the above mentioned deep learning model WaveGAN that can be trained to reproduce synthetic audio files.

References

[1] Lekshmi CR and Rajeev Rajan. Predominant instrument recognition in polyphonic music using convolutional recurrent neural networks. *CMMR*, page 185.

[2] Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. *arXiv preprint arXiv:1802.04208*, 2018.

[3] Yoonchang Han, Jaehun Kim, and Kyogu Lee. Deep convolutional neural networks for predominant instrument recognition in polyphonic music. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(1):208–221, jan 2017.

[4] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[5] Hema A Murthy and Venkata Gadde. The modified group delay function and its application to phoneme recognition. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, volume 1, pages I–68. IEEE, 2003.

[6] Hema A Murthy and Bayya Yegnanarayana. Group delay functions and its applications in speech technology. *Sadhana*, 36:745–782, 2011.

[7] Bayya Yegnanarayana and Hema A Murthy. Significance of group delay functions in spectrum estimation. *IEEE Transactions on signal processing*, 40(9):2281–2289, 1992.