Musical instrument detection using tensorflow convolutional neural networks

José Antonio Villar Caparroso

Abstract

This paper includes a study on the training of a convolutional neural network for the recognition of musical instruments and its use for recognizing the instruments used in a song

1 Introduction

Automatic musical instrument recognition is an important task in the field of music information retrieval (MIR), with applications ranging from music transcription and categorization to recommendation systems and digital music education. Identifying the instruments present in an audio recording poses unique challenges due to the complexity of sound mixtures, overlapping harmonics, and variations in playing styles. While traditional machine learning approaches have made significant strides in this domain, deep learning methods, particularly convolutional neural networks (CNNs), have emerged as powerful tools for extracting and learning features directly from raw or processed audio data.

This paper focuses on the training and evaluation of a CNN-based model for the task of musical instrument recognition using the Instrument Recognition in Musical Audio Signals (IRMAS) dataset. The IRMAS dataset is a widely used resource in the MIR community, designed to support research on instrument recognition in polyphonic audio. It provides a rich set of audio samples with detailed annotations, making it ideal for training supervised models.

2 Project objectives

1. Dataset processing and feature extraction
    - Develop an efficient pipeline for preprocessing raw audio files from the IRMAS dataset into suitable formats for CNN training (e.g., spectrograms, Mel-frequency cepstral coefficients (MFCCs), or log-mel spectrograms).
    - implement techniques to handle multi-label classification, ensuring that each audio clip's multiple instrument labels are correctly encoded and used in the training process.
    - Reimplement the preprocessing pipeline for use in the prediction of new audio files
2. Design and Implement a CNN Model
    - Design a deep convolutional neural network (CNN) architecture tailored for audio data, optimizing it for the task of musical instrument recognition.
    - incorporate relevant techniques, such as data augmentation, to improve model robustness to variations in audio characteristics, including volume, pitch, and noise.
3. Train the CNN Model
    - Train the CNN model on the processed IRMAS dataset, ensuring the model can accurately classify multiple instruments in polyphonic audio recordings.
    - Assess the trained model using appropriate evaluation metrics, such as F1-score, particularly focusing on the model's ability to predict multiple instruments in a single audio clip.
4. Research problems of multi-label classification

- Investigate challenges specific to multi-label classification in the context of polyphonic audio, such as label imbalance and overlapping instruments.
- Experiment with different loss functions (e.g., binary cross-entropy) and evaluation strategies to ensure the model is appropriately handling multiple simultaneous instrument labels.

3 State of the art

1. Convolutional Neural Networks (CNNs): CNNs have become a dominant approach for audio classification tasks due to their ability to automatically learn relevant features from spectrogram representations. In music information retrieval (MIR), CNNs are often used to identify patterns in time-frequency representations of audio.

2. Spectrograms: Traditional spectrograms (such as short-time Fourier transforms, STFT) are widely used, but more specialized representations like log-mel spectrograms or MFCCs are often preferred due to their ability to highlight musical features and reduce the dimensionality of the data. Log-mel spectrograms are especially useful for capturing the harmonic content of musical signals.

3. Multi-Label Classification: Since multiple instruments often play simultaneously in a track, handling multi-label classification is essential. Techniques such as binary cross-entropy for each instrument label and specialized architectures for multi-output classification have become standard.

4 Design and implementation

4.1 Data collection and preprocessing

For the training and validation of the AI model, the Instrument Recognition in Musical Audio Signals (IRMAS) dataset was used. This data set includes different labels for cello, clarinet, flute, acoustic guitar, electric guitar, organ, piano, saxophone, trumpet, violin and the human singing voice as well as whether the sound file includes drums or not and finally the genre of the song in the file. Only the first two were considered as the genres of the dataset seemed a bit limited and may be very inaccurate when testing for more modern or different music. The data set is divided into training and testing.

The training portion of the dataset includes several folders for each of the instrument labels and inside of these folders there where the sound files labeled with the instrument, presence of drums and genre of the song. e.g.: 01[cla][nod][cla]02.wav indicates that this soundbite is of a clarinet playing classical music with no drums (note how cla repeats itself but the one denoting instrument is always the first one and genre the last one).

For the testing portion it itself is divided into three parts, each containing extracts from real songs in wav format and a txt file with the labels for that song. e.g: 15more than a feeling - boston-1.wav is a 6 second file from the part 2 of the testing data and 15more than a feeling - boston-1.txt is a text file containing only the gac label for acoustic guitar.

All data were then pre-processed into spectograms of the wav files npy files for the labels. This was achieved using python with the help of libraries like librosa for converting wav to spectograms images that where converted to png images using pyplot.

4.2 Building the dataset with tensorflow

The dataset used for training the convolutional neural network (CNN) consists of spectrogram images and their associated multi-labels. These spectrograms are processed into a format suitable for input into the model.

The load-dataset function handles the loading of the dataset from the directory containing the spectrogram images. For each image in the directory, the function checks if the file ends with the .png extension (indicating that it is a spectrogram image). For each spectrogram image, a corresponding .npy label file is loaded, which contains the multi-label annotations for the instruments present in the

audio sample. These labels are stored as numpy arrays, which are later converted into TensorFlow tensors.

Then a data-generator function converts the spectrogram image paths and labels into a TensorFlow-compatible format for training. Each spectrogram image is loaded, decoded from its PNG format, resized to 128x128 pixels, and normalized to a range between 0 and 1. The corresponding labels are loaded and converted into TensorFlow tensors of type float32.

Using tf.data.Dataset.from-generator, the spectrogram paths and labels are transformed into TensorFlow datasets. The output-signature specifies the shape and data types for the images and labels. The images have a shape of (128, 128, 3) (indicating the image dimensions and the number of color channels), and the labels have a shape of (len(INSTRUMENTS) + 1,) to account for the multiple instruments and an additional label for drums.

The training dataset is batched, repeated, and prefetched to optimize performance during training. The validation dataset is similarly batched and prefetched.

4.3 Building the AI model with tensorflow

The model is built as a convolutional neural network (CNN), which is commonly used for image-related tasks due to its ability to learn spatial hierarchies of features.

- Input Layer: The model expects input images of size 128x128x3, corresponding to the resized spectrograms. The images are represented in three color channels (RGB), even though they are spectrograms, to leverage standard image processing techniques.
- Convolutional Layers: The CNN consists of three convolutional layers, each followed by a max-pooling layer:
  - The first convolutional layer has 32 filters of size (3, 3) and uses the ReLU activation function.
  - The second convolutional layer has 64 filters, followed by another ReLU activation and a max-pooling layer.
  - The third convolutional layer has 128 filters, again followed by ReLU and max-pooling.

  These convolutional layers extract features from the spectrograms, such as edges and textures, which are important for identifying different instruments.
- Flattening and Dense Layers: After the convolutional layers, the model flattens the output into a one-dimensional vector and passes it through a dense layer with 128 units and ReLU activation. This layer allows the network to learn higher-level representations.
- Output Layer: The final layer of the model is a dense layer with len(INSTRUMENTS) + 1 units (one for each instrument plus an additional unit for drums), and it uses a sigmoid activation function. The sigmoid activation allows the model to predict the probability of each instrument being present independently (suitable for multi-label classification).

4.4 Model compilation and metrics

The model is compiled using the Adam optimizer and the binary cross-entropy loss function, which is appropriate for multi-label classification tasks. Binary cross-entropy measures the error between the predicted probabilities and the true labels for each instrument independently.

A custom F1 score metric is implemented to evaluate the model's performance, particularly because it balances precision and recall. This is important in multi-label classification tasks where both false positives and false negatives need to be minimized.

This custom metric ensures that the model's ability to detect each instrument is measured appropriately, considering both the precision (correctly predicted instruments) and recall (ability to detect all true instruments).

5 Results

The model with some accuracy predicts the instruments from the initial dataset that are used in a provided soundfile. However, it needs to first be converted into a spectrogram with 128x128 dimensions.

6 Conclusion

In this project, we designed and implemented a convolutional neural network (CNN) for multi-label musical instrument recognition from spectrograms. Using the IRMAS dataset, we demonstrated that a well-structured CNN can successfully classify multiple instruments from polyphonic music, leveraging spectrogram images as input features. The model's performance was evaluated using F1 score, ensuring a balance between precision and recall in the multi-label context. The combination of data preprocessing, a robust CNN architecture, and custom metrics allows the model to identify instruments with high accuracy and efficiency.

7 Future work

While the current model provides solid performance for the classification of instruments present in the IRMAS dataset, there are several avenues for improvement and future exploration.

- One significant direction for future work is expanding the model to recognize a broader range of musical instruments, including rare or non-traditional instruments. This could be achieved by integrating larger and more diverse datasets or by applying data augmentation techniques to simulate different instruments or recording conditions.
- Enhancing the model with more advanced architectures, such as convolutional recurrent neural networks (CRNNs) or attention mechanisms, could further improve the model's ability to capture temporal dependencies in music and identify instruments more accurately. Real-time performance and application to live music processing would also be valuable, requiring optimizations for speed and computational efficiency.
- Finally, incorporating domain-specific features, such as timbral characteristics, could further refine the instrument classification process, making the model more robust in real-world scenarios.

References

- IRMAS: a dataset for instrument recognition in musical audio signals. (s. f.). Upf. Recuperado 11 de diciembre de 2024, de https://www.upf.edu/web/mtg/irmas
- tensorflow. (s. f.). GitHub. https://github.com/tensorflow