Deep Learning for Audio and Music - Coursework Written Report

Yiannis Emmanouilidis

April 2025

1 Introduction and Aim

This project is an investigation around musical genre identification and tempo estimation, two fundamental tasks in Deep Learning for Audio and Music. We use the GTZAN dataset of 1 000 thirty-second audio excerpts spanning ten musical genres and explore three questions:

Q1. How accurately can a convolutional neural network learn to classify genres from mel-spectrograms of the GTZAN dataset alone?

Q2. Can a tempo estimation model based on the publicly available PANNs Cnn14 architecture and fine-tuned on the same GTZAN splits produce reliable tempo (BPM) estimates on GTZAN clips?

Q3. Can fusing the two models together by injecting the tempo estimates from the second model into the genre classifier further boost the latter's accuracy?

This report describes the approaches taken for data preparation and feature extraction, details the architectures and training procedures for the two individual models, presents their evaluation results, and subsequently outlines the design of the fused genre-plus- tempo system and its evaluation. Finally, we discuss the findings and propose next steps.

2 Data and Pre-processing

2.1 Dataset

The GTZAN dataset consists of 1 000 audio clips (30 s each) equally distributed across ten genres. We also have an accompanying CSV which we read to get the ground truth values for genre and tempo. Note that one corrupted file ('jazz.00054.wav') was omitted from this project altogether, after confirming its header did not begin with the standard 'RIFF' marker.

2.2 Feature Extraction

Each audio file is resampled (if necessary) to a fixed sample rate of 22,050 Hz. We compute mel-spectrograms with a 50 ms window (n fft=1,102), a 25 ms hop, and 64 mel bands. The resulting spectrograms have shape [1, 64, T], where T is the number of frames (about 1300 for 30 seconds). We convert magnitude to decibels (dB) via a log-scale transform with the help of the torchaudio library. dB conversion is known to tend to improve neural-network convergence. Because our convolutional architecture requires fixed-size inputs, we either pad spectrograms that are shorter than our target frame length or truncate longer ones so that every example ends up as a 64×400 matrix (roughly 10 seconds of audio). We hope that the shortening of the time length cuts down on subsequent computational costs without significantly reducing genre classification capabilities. Finally, we stack all spectrograms into a tensor of shape [N, 1, 64, 400], where "1" is for the channel dimension expected by PyTorch's Conv2d and normalize each mel-band channel using the mean and standard deviation computed on the training split only (so as to keep unseen data truly unseen). We also assign each genre an integer label to facilitate model output with ground truth comparison.

2.3 Data splitting

To ensure each genre remains proportionally represented, we used stratified splitting when creating training, validation and test sets. 20 % of the clips were used for the test set. From the remaining 80 %, 25 % (i.e. 20 % of total) were reserved for the validation set in a subsequent split. Thus, the rest (60 % of total) was used for training.

3 Genre Classification

3.1 Model

Our genre classifier is a 2D-CNN. The architecture comprises of three convolutional blocks:

- **Conv Block 1**: 64 filters of size 5×5, BatchNorm, ReLU, MaxPool (4×4).
- **Conv Block 2**: 128 filters of size 5×5, BatchNorm, ReLU, MaxPool (4×4).
- **Conv Block 3**: 128 filters of size 3×3, BatchNorm, ReLU, MaxPool (2×1).

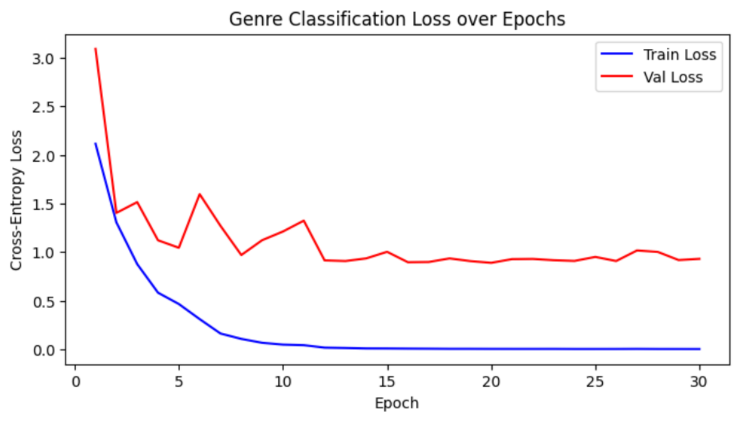
A final flattening layer feeds into two fully–connected layers. The first is a hidden FC with 128 units (ReLU) and the second an output FC with 10 units (one per genre). All convolutional and linear weights are initialised with Kaiming uniform, and biases (where used) are initialised to zero.

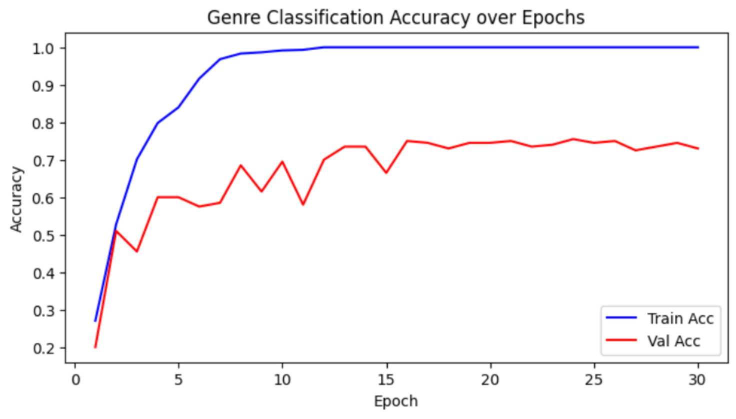
3.2 Training Procedure

We train using cross-entropy loss and the Adam optimiser, with a learning rate choice of 1×10^{-3}. To help tackle overfitting we use early stopping on the validation loss so that if it fails to improve for 10 consecutive epochs, training stops and we keep the weights at the epoch with the best validation loss. Batches of 32 spectrograms are drawn with shuffling on the train split; validation and test batches are drawn without shuffling to ensure re-runs produce the same outcome. Training runs for up to 30 epochs.

3.3 Evaluation

On the validation set, the CNN achieves a peak accuracy of approximately 65–75%, with cross-entropy loss decreasing from around 2.0 to below 0.3. On the test set, the model attains an overall accuracy of 70% and loss value of 1.2148. The Loss vs. Epochs Plot and Accuracy vs. Epochs Plot show some overfitting.





4 Tempo Estimation Model

4.1 Model

For tempo estimation we adopt the PANNs Cnn14 architecture (appropriate github repository was cloned - see the notebook). We then fine-tune it on our GTZAN splits. The raw waveform (30 s at 22,050 Hz) is chunked or padded to a fixed length (661,500 samples). The Cnn14 backbone processes raw audio into a single BPM prediction (which is a scalar), after scaling ground-truth BPM into the $[0, 1]$ range (a minimum of 40 BPM and maximum of 250 BPM is applied). The backbone was wrapped in a small Lightning `System` class that applies MSE loss between predicted and and the scaled target BPM, using Adam and early stopping on validation loss.

4.2 Training Procedure

We train with a batch size of 16, learning rate 1×10^{-4}, and early stopping (after 10 epochs) on the validation MSE.

4.3 Evaluation

On the test split, our fine-tuned tempo estimator yields:

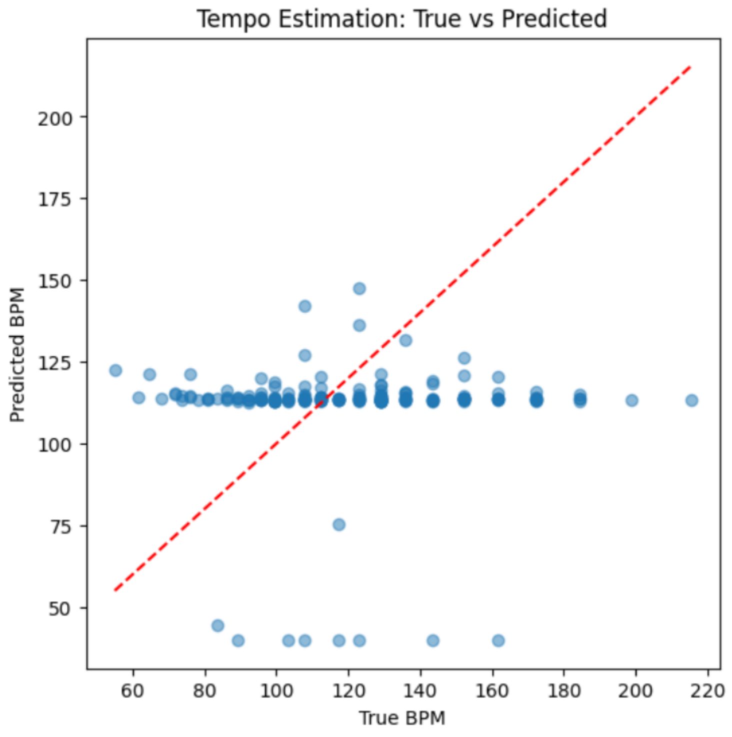
$$MAE \approx 25.2 \text{ BPM}, \quad MedAE \approx 18.0 \text{ BPM}.$$

The true-vs-predicted scatterplot shows a positive correlation but significant variance, especially at high and low BPM extremes.

5 Genre Classification assisted by Tempo Estimation

5.1 Model

The fused model builds upon the two separate systems for genre classification and tempo estimation by injecting the estimated tempo as an additional input feature midway through the genre classifier's fully-connected pipeline. We begin with the pretrained convolutional backbone from our genre-only CNN and freeze convolutions and batch-norm layers to make training quicker. This model takes a single-channel Mel-spectrogram of shape $1 \times 64 \times 400$ and produces a flattened embedding of dimension 128. In parallel, We also have the tempo estimator (a PANNs Cnn14 backbone wrapped in a Lightning `System`) yields a scalar $\widehat{\text{BPM}}$. To fuse the two, we first normalize the predicted BPM by scaling into the same range used in training (min/max BPM), then pass this scalar through a small two-layer MLP ($32 \to 16 \to 1$, ReLU activations). We then add the tempo to turn the 128-dimensional vector into a 129-dimensional vector, which is fed into two new fully-connected layers to produce the final genre logits. All


Tempo Estimation: True vs Predicted
Predicted BPM
True BPM
200
175
150
125
100
75
50
60
80
100
120
140
160
180
200
220

convolutional weights remain frozen, so during training only the two FC layers and the tempo embedding MLP are updated. Early stopping on validation loss (patience = 10) and a ModelCheckpoint callback ensure we save the best fusion weights.

5.2 Evaluation

An accuracy of 68% indicates slightly the worse performance of the fused model compared to the first model.

6 Discussion and Conclusions

In this project, we set out to classify music genres from audio and to estimate track tempo, then to investigate whether tempo information can enhance genre discrimination. Our genre classifier—a four-block CNN trained on 64×400 Mel-spectrograms—achieved 67.1% test accuracy on the GTZAN dataset, and our PANNs-based tempo estimator reached a mean absolute error of 25.2 BPM (median 17.9 BPM). When fusing the estimated BPM into the genre classifier, we were not able to get an improvement.

Perhaps it is the case that spectral features alone capture much of the rhythmic and timbral cues relevant to genre. However the model's limitations are more likely to explain the lack of imporved performance. Key limitations include: the small size and noisiness of GTZAN; the freezing of low-level features which may have restricted the network's flexibility to integrate tempo; the short 11 second clips used for tempo estimation and the absence of further experimenting with hyperparameters and tweaking of model architectures.