Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

SP
Signal Processing

Final Report

Project “Deep Learning for Audio Processing” by Jakob Kienegger and Martin Jälmby

Group Geschke — Louis Tank and Marius Niveri
19.02.2026

1 Baseline Model

In this report we want to give an insight into how we made our final deep learning model and examine some of the roadblocks we faced on our way. The overarching task was to classify audio clips. The data came from the DCASE 2017 [1] challenge.

To begin, we realized the multilayer perceptron architecture laid out in the provided git repository. The model was quite small at only 13.4 thousand parameters (2 hidden layers and 50 hidden features) but performed quite well already. It peaked at about 65% validation accuracy after 12 epochs.

2 First Steps

In order to better understand the Python program we extended and how the plethora of third party libraries were used, we rewrote all the functions from scratch in a separate git branch.

This was very helpful and set us up to tweak aspects of the program other than the model definition itself.

One example here is the PyTorch accelerator. Most of our training was happening on a remote x86 machine with an NVIDIA card. However, we also wanted to test small changes locally which happened to be a newish Apple computer – hence with an arm64 CPU. Luckily PyTorch brings native backends for both. This allowed us to work with the same Python code across CPU architectures:

```python
def determine_accelerator():
    if sys.platform == "darwin" and torch.backends.mps.is_available():
        return "mps"
    elif torch.cuda.is_available():
        return "gpu"
    else:
        return "cpu"

def get_trainer(logger, max_epochs, ckpt_dir):
    return pl.Trainer(
        accelerator=determine_accelerator(),
        # ...
    )
```

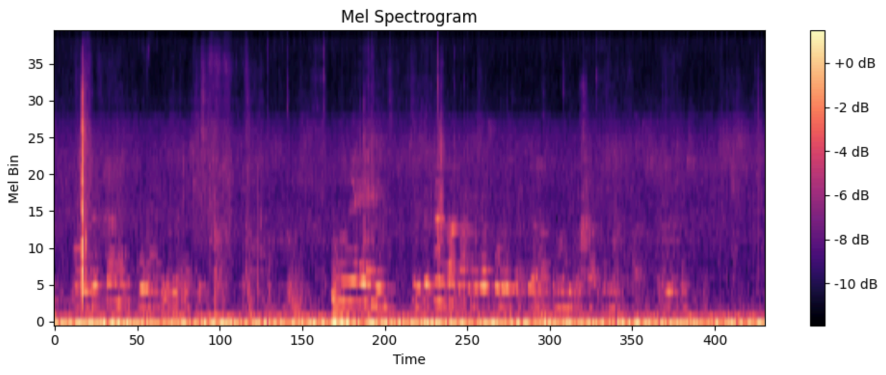
We applied the same logic to the strategy (`DDP` for NVIDIA and `auto` for all other cases). This allowed us to run the model both on the remote machine and local hardware.

2.1 Configuration

As with the template, we chose to configure most of our model's parameters in a separate config file. This also seemed like the perfect start to improve its accuracy. Most parameters were straight-forward and easy to grasp. However, the ones for the Mel spectrogram seemed a little random.

As we understood it, the spectrogram maps each audio clip to a 2D image which effectively turns our audio classification task into image classification.

So to immediately get feedback on changing the spectrogram's parameters, we visualized it using a small helper script:



Playing around with `F_MIN`, `F_MAX` and `N_MELS` helped us understand the variables a bit better and we decided to leave them at their default values.

However, we thought that the default model was extremely small so the first optimization we did was increasing the hidden features from 40 to 64 (also increased the spectrograms' bins to match) leaving everything else as is.

Even though this almost doubled the model's total parameters to 25.7K, the resulting validation accuracy improved a bit ($+6\%$ after 10 epochs) but dropped even harder than before indicating overfitting.

3 Convolutional Layers

Next, we replaced some of the linear layers with convolutional layers which promise better results for image classification.

For a fair comparison with the baseline MLP, we kept the model size the same. This checkpoint consisted of 3 convolutional layers and totaled 26.4K parameters.

However, even after extensive trial and error, we never exceeded the initial validation accuracy of the baseline MLP.

4 Problem: Overfitting

Every time we tried to increase the model size in general, we ran into heavy overfitting. With a convolutional architecture, there are multiple steps to reduce overfitting:

1. Increase Dropout
2. Artificially increase training dataset (Data augmentation)
3. Add identity mappings aka. residual blocks [2]

Unfortunately increasing the dropout did not do much for us so we moved on to the second solution.

4.1 Data Augmentation

After drastically increasing the model size to 1.6M parameters and more than doubling the Mel spectrogram bins to `128`, we expected heavy overfitting. As anticipated, the model overfit drastically (about 99% training accuracy and 10% validation accuracy).

Now we introduced random noise to the audio files as they were retrieved from the dataset:

```
def add_white_noise(self, audio_data: torch.Tensor) -> torch.Tensor:
    noise = torch.randn(audio_data.shape) * np.random.uniform(0, 0.005)
    return audio_data + noise
```

This did not improve our scores much so we moved on.

Looking into more sophisticated ways to slightly change each audio clip during training, we landed on the methods provided by `torchaudio` – a library we already imported. [3]

Specifically, we implemented four methods:

1. Noise – very slight noise is added to the file (as shown in the code snippet above)
2. Time shift – the recording is shifted forward or backward by a maximum of 0.5 seconds
3. Time stretch – the recording is slowed down or sped up by a maximum of 20%
4. Frequency shift – the recording is raised or lowered by a maximum of 2 semitones

4.1.1 Benchmark

Since training was quite fast for the baseline model, we ran a benchmark using different augmentation approaches:

Approach	Training Accuracy	Validation Accuracy
No augmentation	96.4%	66.7%
Time shift	96.1%	62%
Time stretch	95%	59.8%
Frequency shift	92.4%	63.3%
Noise	82.1%	65.2%
Combination of all four	78.5%	65%

We landed on a combination of frequency and time shifting but kept in mind that data augmentation can also decrease performance – especially with smaller models. Applied to the large model, we saw an increase in validation accuracy while training accuracy did not increase as quickly. Still, overfitting became an issue after a couple of epochs and validation accuracy did not surpass our baseline.

We tried decreasing the model size until we reached the size of the baseline but, again, never reached its performance.

4.1.2 Generative Augmentation

At this point we were also familiar with a couple of the papers from original participants of the DCASE 2017 challenge. All of the well-performing teams used data augmentation. Some even went as far as training a separate model to generate completely new training data. [4]

We started going down that route as well but never got to any useable audio files – they all sounded like a robot having a stroke. Figuring this out would have been nice but we didn't have the time required to get there.

5 Trying Other Architectures

After lots of trial and error with convolutional layers, we also tried multiple recurrent neural networks and even a huge transformer-based model. None of these exceeded the performance of our initial baseline model. So we went back to that first approach.

6 Final Model

We noticed that both the convolutional neural network and baseline MLP performed approximately the same. Why not combine them? Our final model is a dual-stream MLP and CNN. Dual-stream means that both models run in parallel and their outputs are then fused together. In contrast, the data would first pass through either the MLP or CNN and then through the other in a single-stream model.

6.1 Description

We use the same Mel spectrogram as we did in the baseline. The only adjustment is a slight increment of resolution by increasing the bins from 40 to 64.

6.1.1 Multilayer Perceptron Stream

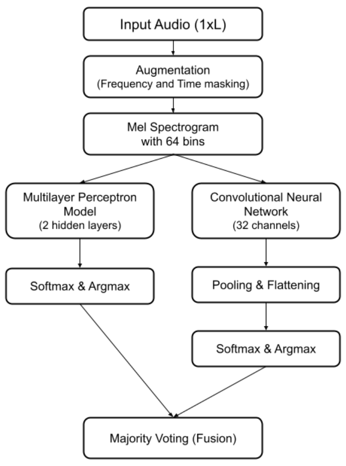
We've barely changed the MLP side of the model because we did not exceed the baseline's performance by a significant amount. This means that our MLP has $24.7\mathrm{K}$ parameters and consists of 2 hidden layers with 64 features each.

6.1.2 Convolutional Stream

The CNN is even smaller than the MLP. Reason being drastic overfitting very early on when parameters are increased even slightly. We tried using residual blocks but that did not help. We suspect that their advantage becomes apparent only in larger models.

6.1.3 Dual-Stream Architecture

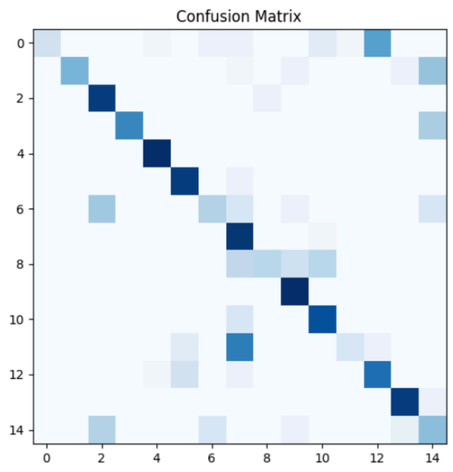
The following figure visualizes the parallel nature of the dual-stream architecture we landed on:


Input Audio (1xL)
Augmentation (Frequency and Time masking)
Mel Spectrogram with 64 bins
Multilayer Perceptron Model (2 hidden layers)
Convolutional Neural Network (32 channels)
Softmax & Argmax
Pooling & Flattening
Softmax & Argmax
Majority Voting (Fusion)

6.2 Performance Analysis

Our expectations of performance in regards to each individual label (tram, beach, car, etc.) are shaped by their distinctness. Some of the scenarios are very distinct (e.g. tram vs library). However, some others are quite similar (e.g. tram vs train). So we expect the model to do well on distinct classes that have no similar neighbor but underperform on the others.

Plotting the confusion matrix only partially confirms this:



We observe that some labels were well labeled (such as class 4/car and 9/metro-station) while others were confused more often (such as class 11/park with 8/library). This makes intuitive sense as both the car and metro station scene seem distinct and both park and library are quiet places without many distinct sounds.

The total validation accuracy peaked at 70% in epoch 39 whilst training accuracy stayed stable at 77% on average. We submit the checkpoint from epoch 39 to be tested.

7 Conclusion

The similar classes on top of the small total amount of training data made this quite a difficult challange for us. Our model only barely exceeded the performance set by the baseline MLP.

The model we used was precicely tweaked to work great on the DCASE 2017 challange. We expect it to perform similarly on other image classification tasks with a small to medium training dataset.

For larger datasets or with more extensive data augmentation, we would prefer a larger model such as the ones we experimented with earlier on.

[1] “DCASE 2017 Challange Homepage.” [Online]. Available: https://dcase.community/challenge2017/index

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, “Deep Residual Learning for Image Recognition.” [Online]. Available: https://arxiv.org/abs/1512.03385

[3] “Audio Data Augmentation.” [Online]. Available: https://docs.pytorch.org/audio/stable/tutorials/audio_data_augmentation_tutorial.html

[4] Seongkyu Mun, Sangwook Park, David Han, and Hanseok Ko, “Generative Adversarial Network Based Acoustic Scene Training Set Augmentation and Selection Using SVM Hyper-Plane.”