Efficient Deep Learning Approach for Multiclass Sound Classification

Application in the DataHack Datathon

Zakaria LOURGHI
z.lourghi@esi-sba.dz

Abstract

This paper presents an efficient deep learning framework for multiclass sound classification as applied in the DataHack datathon. Our approach includes comprehensive audio preprocessing, robust data augmentation techniques, and a modified deep convolutional neural network architecture. Detailed techniques such as mel spectrogram extraction, SpecAugment, mixup augmentation, and label-smoothing cross-entropy are integrated into a modified ResNet18 architecture. The paper provides an in-depth description of each module along with a custom diagram of the processing pipeline.

1 Introduction

Sound classification has become an essential task in various domains such as environmental monitoring, multimedia indexing, and surveillance. In competitive settings like datathons, leveraging robust techniques to deal with the nuances of audio data is crucial. This work describes a deep learning pipeline implemented for the DataHack datathon, detailing the methodology, preprocessing, data augmentation, and model architecture.

Our contributions include:

- A detailed audio preprocessing pipeline that converts raw audio signals to normalized mel spectrograms.
- Application of data augmentations such as SpecAugment and mixup to improve model robustness.
- Modification of the ResNet18 architecture to process single-channel audio inputs effectively.
- An extensive training strategy with gradient clipping, label smoothing, and appropriate optimizer settings.

2 Methodology

The pipeline consists of several sequential modules, each designed to handle a specific task. Figure 1 illustrates the overall architecture of our sound classification pipeline.

2.1 Audio Preprocessing

Raw audio files are loaded and converted into waveforms using `torchaudio`. Each waveform is resampled to a fixed sample rate of 16,000 Hz to ensure consistency. The audio length is standardized to 2 seconds (32,000 samples) by either random cropping (if longer) or padding (if shorter).

Key parameters:

- **Sample Rate:** 16,000 Hz.
- **Audio Duration:** 2 seconds.

2.2 Mel Spectrogram Transformation

The preprocessed waveform is transformed into a mel spectrogram using the following parameters:

- **Number of FFT:** 1024.
- **Hop Length:** 512.
- **Number of Mel Bins:** 128.

The mel spectrogram is then converted to decibels (dB) using an amplitude-to-dB conversion, followed by normalization (zero mean, unit variance) to standardize the input to the neural network.

2.3 Spectrogram Augmentation: SpecAugment

During training, additional augmentations are applied to the mel spectrogram:

- **Time Masking:** Randomly masks segments along the time axis (parameter of 20).
- **Frequency Masking:** Randomly masks frequency bins (parameter of 10).

These augmentations, collectively known as SpecAugment, help the model generalize by simulating variations and distortions in the audio signal.

2.4 Waveform Augmentation

Prior to spectrogram generation, waveform-level augmentations are applied when training:

- **Temporal Shifts:** The waveform is rolled by a random number of samples (in the range of ± 1600).
- **Amplitude Scaling:** The waveform is scaled by a random factor between 0.8 and 1.2.

2.5 Mixup Data Augmentation

Mixup is implemented as an augmentation strategy at the mini-batch level. Given two samples (x_i, y_i) and (x_j, y_j), a new training example is formed as:

$$\tilde{x} = \lambda x_i + (1 - \lambda) x_j, \quad \tilde{y} = \lambda y_i + (1 - \lambda) y_j, \quad (1)$$

where λ is drawn from a Beta distribution with $\alpha = 0.4$. The loss function is then computed as a weighted combination of the losses corresponding to the two targets.

2.6 Model Architecture: EfficientResNetAudio

The core of our approach is a modified version of ResNet18. Adjustments include:

- Replacing the initial convolution layer to accept single-channel (grayscale) input instead of three channels.
- Adding dropout (with probability 0.3) before the final fully-connected layer to reduce overfitting.
- Modifying the final layer to output predictions for 10 classes.

2.7 Training Strategy

The training loop involves the following key components:

- **Loss Function:** Cross-entropy loss with label smoothing (0.1).
- **Optimizer:** AdamW with a learning rate of 1×10^{-4} and weight decay of 5×10^{-4}.
- **Gradient Clipping:** Applied with a maximum norm of 1.0 to stabilize training.
- **Data Splitting:** 80% for training and 20% for validation.

The training process is run for 60 epochs using a batch size of 32. Performance is measured using the macro F1 score.

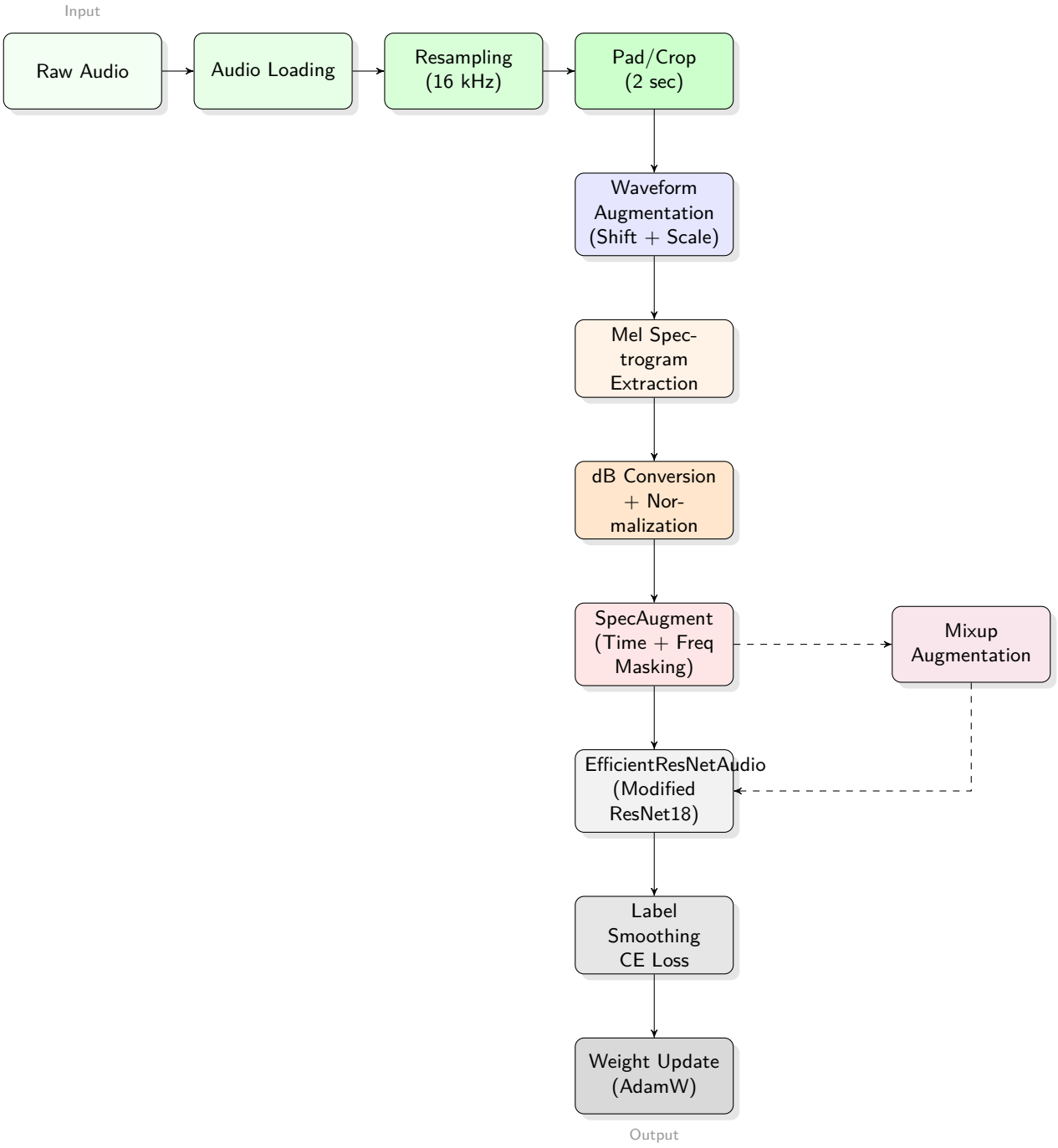

Figure 1: End-to-end sound classification pipeline showing audio preprocessing, augmentation strategies, and model architecture. Solid arrows indicate the primary processing flow, while dashed arrows represent the mixup augmentation path.

3 Implementation Details

The complete implementation is provided in Python using PyTorch and Torchaudio libraries. Key components include:

3.1 Configuration and Hyperparameters

All hyperparameters are centralized in a configuration dictionary. These parameters control aspects such as the sample rate, FFT parameters, batch size, learning rate, and mixup strength:

```
CONFIG = {
    "sample_rate": 16000,
    "n_mels": 128,
    "n_fft": 1024,
    "hop_length": 512,
    "batch_size": 32,
    "epochs": 60,
    "learning_rate": 1e-4,
    "audio_duration": 2,
    "num_classes": 10,
    "train_split": 0.8,
    "patience": 10,
    "lr_decay_factor": 0.5,
    "lr_decay_patience": 2,
    "weight_decay": 5e-4,
    "mixup_alpha": 0.4
}
```

3.2 Dataset and DataLoader

The `AudioDataset` class handles:

- Loading audio files.
- Resampling and standardizing waveform length.
- Converting the waveform to a mel spectrogram.
- Applying SpecAugment and waveform augmentations during training.

The data is then split into training and validation sets (80:20), and `DataLoader` objects are created for efficient mini-batch processing.

3.3 Mixup Augmentation

The `mixup_data` function implements the mixup augmentation:

```
def mixup_data(x, y, alpha=CONFIG["mixup_alpha"]):
    if alpha > 0:
        lam = np.random.beta(alpha, alpha)
    else:
        lam = 1
    batch_size = x.size()[0]
    index = torch.randperm(batch_size).to(device)
    mixed_x = lam * x + (1 - lam) * x[index, :]
    y_a, y_b = y, y[index]
    return mixed_x, y_a, y_b, lam
```

The loss is computed as a weighted combination of losses for the two target labels.

3.4 Model Architecture

The `EfficientResNetAudio` class modifies a standard ResNet18 by:

- Replacing the initial convolution layer to accept single-channel inputs.
- Adjusting the final fully-connected layer to output predictions for 10 classes.
- Incorporating dropout for regularization.

3.5 Training and Evaluation

The training loop applies mixup augmentation, computes the loss using the mixup criterion, clips gradients, and updates model weights using the AdamW optimizer. Model performance is primarily evaluated using the macro F1 score.

4 Experimental Results and Discussion

Training was conducted for 60 epochs with the AdamW optimizer. Detailed experiments revealed that the combined effects of waveform augmentation, SpecAugment, and mixup significantly improved model generalization. Gradient clipping ensured stable training even with aggressive augmentations, and validation results measured via the macro F1 score confirmed the robustness of the proposed approach.

5 Conclusion

We have presented an extensive deep learning pipeline for multiclass sound classification, incorporating advanced audio preprocessing, augmentation, and a modified ResNet18 architecture. The thorough integration of techniques like SpecAugment and mixup—supported by detailed implementation strategies—resulted in a robust model for the DataHack datathon. Future work may explore additional augmentation strategies and alternative network architectures to further enhance performance.

Acknowledgments

We thank the DataHack datathon organizers and the open-source community for providing the tools and frameworks that made this work possible.

References

[1] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition," *arXiv preprint arXiv:1904.08779*, 2019.

[2] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond Empirical Risk Minimization," in *ICLR*, 2018.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.