CNN-Based Sound Classification for Multi-Class Recognition | CS F425 Deep Learning

Bhuman Pandita
`2021A2PS2605P`

Ojasva Goyal
`2021A2PS2378P`

M N Vignesh Kumar
`2023H1120192P`

Abstract

This project explores the classification of 13 distinct sound classes utilizing Convolutional Neural Networks (CNNs). After testing for multiple architectures and for multple hyperparameters, Two primary architectures were employed: a 6-layer basic CNN and ResNet-18. Each sound clip underwent a transformation process to generate its corresponding spectrogram, capturing the temporal and frequency content of the audio signal. These spectrograms were then fed into the CNN architectures for further processing and classification.The report explains the experimentations with CNN models, investigating the impact of various parameters on performance.

1 Preprocessing

CNN are typically used for images. But if we want to use CNN for audio classification, we have to convert the audio signals into spectograms. Spectrograms represent a visual depiction of the frequency content of an audio signal over time, providing insight into the varying intensity and distribution of frequencies within the signal.

In the code we have used some functions to load audio files, preprocess them, and convert them into spectrograms.

1. The `load_and_preprocess_audio` function loads an audio file using librosa, ensuring it is of a specified duration and sampling rate.

2. It then generates a Mel spectrogram from the audio data using librosa's `melspectrogram` function and converts it to decibel units using `power_to_db`.

3. The `load_dataset` function iterates over the dataset folders, loads each audio file, and preprocesses it into a spectrogram.

4. Finally, the `prepare_data` function prepares the training and validation datasets by loading and preprocessing the audio data, reshaping it for CNN input, and encoding the labels.

2 Model description

2.1 6 Layer CNN

For the first model we used a basic 6 layer CNN model. We opted for the 6-layer CNN, adhering to the KISS (Keep It Simple, Stupid) principle in software engineering, prioritizing simplicity for efficient model development and training. It turns out that this model demonstrates satisfactory performance with decent training and validation accuracy. Coming to hyperparameters we used Learning rate of 0.0001, Stocastic Gradient descent , momentum of 0.62. We have used SGD because it caters to outliers/noise. It helps to mitigate the effect of outliers. We have experimented with Adam optimizer as well but it gave us greater loss as compared to SGD.We have used categorical cross entropy loss function. The parmeters of the model are :

- Total params: 28,386,541 (108.29 MB)
- Trainable params: 28,385,133 (108.28 MB)
- Non-trainable params: 1,408 (5.50 KB)

We have trained the model for 40 epochs with a batch size of 32. The model architecture is depicted below :

Layer (type)	**Output Shape**	**Param #**
conv2d_12 (Conv2D)	(None, 128, 108, 32)	320
batch_normalization_15 (BatchNormalization)	(None, 128, 108, 32)	128
conv2d_13 (Conv2D)	(None, 128, 108, 32)	9,248
batch_normalization_16 (BatchNormalization)	(None, 128, 108, 32)	128
max_pooling2d_6 (MaxPooling2D)	(None, 64, 54, 32)	0
dropout_9 (Dropout)	(None, 64, 54, 32)	0
conv2d_14 (Conv2D)	(None, 64, 54, 64)	18,496
batch_normalization_17 (BatchNormalization)	(None, 64, 54, 64)	256
conv2d_15 (Conv2D)	(None, 64, 54, 64)	36,928
batch_normalization_18 (BatchNormalization)	(None, 64, 54, 64)	256
max_pooling2d_7 (MaxPooling2D)	(None, 32, 27, 64)	0
dropout_10 (Dropout)	(None, 32, 27, 64)	0
flatten_3 (Flatten)	(None, 55296)	0
dense_6 (Dense)	(None, 512)	28,312,064
batch_normalization_19 (BatchNormalization)	(None, 512)	2,048
dropout_11 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 13)	6,669

2.1.1 Evaluation Metrices

While training this model for 30 epochs, we achieved the following performance measures:

- Training accuracy: 0.8976
- Training loss: 0.3195

- Validation accuracy: 0.8561
- Validation loss: 0.4584

For the performance metrices, We have also used Recall as an evaluating metrix. The overall Recall 0.83 was achieved. The Class wise recall have been tabulated below :

Label	0	1	2	3	4	5	6	7	8	9	10	11	12
Recall	0.91	0.89	0.89	0.71	0.90	0.90	0.81	0.93	0.62	0.89	0.86	0.93	0.63

Table 1: Class Wise Recall for 6 Layer CNN

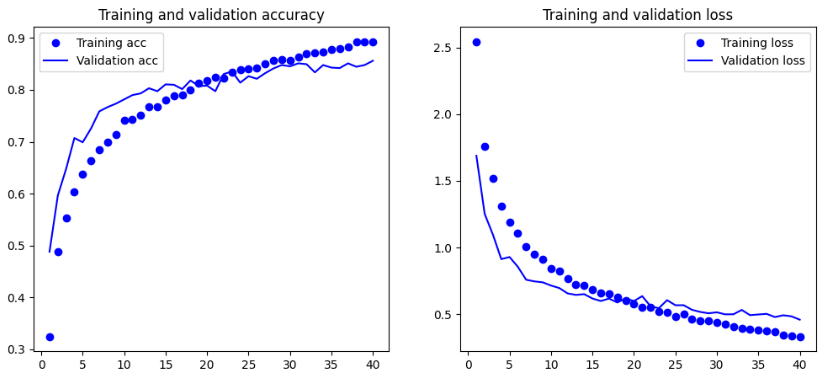

Figure 1: Plots for accuracy and loss on training and validation dataset

2.2 Resnet-18

For the second model we used resnet architecture. ResNet architecture introduces skip connections, allowing for deeper networks without suffering from the vanishing gradient problem. This enables more effective training of very deep neural networks by facilitating the flow of gradients during backpropagation. As a result, ResNet architectures, such as ResNet-18 (which we have considered as aour second model), achieves superior performance in various tasks, including image classification and feature extraction, compared to classical CNN architectures.

For this model we have set the learning rate to 0.001 and SGD optimizer. We have trained the to 30 epochs and batch size of 32 as well. We have used categorical cross entropy loss function. We have used L2 regularization in ResNet-18 mitigates overfitting by penalizing large weights, promoting simpler model patterns and better generalization, ultimately enhancing performance in tasks like image classification.

The parmeters of the model are :

- Total params: 11,191,309 (42.69 MB)
- Trainable params: 28,385,133 (42.65 MB)
- Non-trainable params: 1,408 (37.50 KB)

Classification Report:

	precision	recall	f1-score	support
car_horn	0.92	0.91	0.91	85
dog_barking	0.69	0.89	0.78	160
drilling	0.90	0.89	0.90	140
Fart	0.88	0.71	0.78	72
Guitar	0.98	0.90	0.94	136
Gunshot_and_gunfire	0.81	0.90	0.86	112
Hi-hat	0.92	0.81	0.86	42
Knock	0.93	0.93	0.93	41
Laughter	0.87	0.62	0.72	73
Shatter	0.85	0.89	0.87	53
siren	0.90	0.86	0.88	140
Snare_drum	0.94	0.93	0.93	112
Splash_and_splatter	0.64	0.63	0.64	43
accuracy			0.86	1209
macro avg	0.86	0.84	0.85	1209
weighted avg	0.86	0.86	0.86	1209

Figure 2: Classification Report

2.2.1 Evaluation Metrices

While training this model for 30 epochs, we achieved the following performance measures:

- Training accuracy: 0.9996
- Training loss: 0.4097
- Validation accuracy: 0.8610
- Validation loss: 0.5324
- Precision : 0.75
- Overall Recall : 0.73

For the performance metrices, We have also used Recall as an evaluating metrix. The overall Recall 0.83 was achieved. The Class wise recall have been tabulated below :

Label	0	1	2	3	4	5	6	7	8	9	10	11	12
Recall	0.94	0.84	0.87	0.76	0.94	0.84	0.76	0.80	0.59	0.92	0.90	0.89	0.60

Table 2: Class Wise Recall for Resnet 18

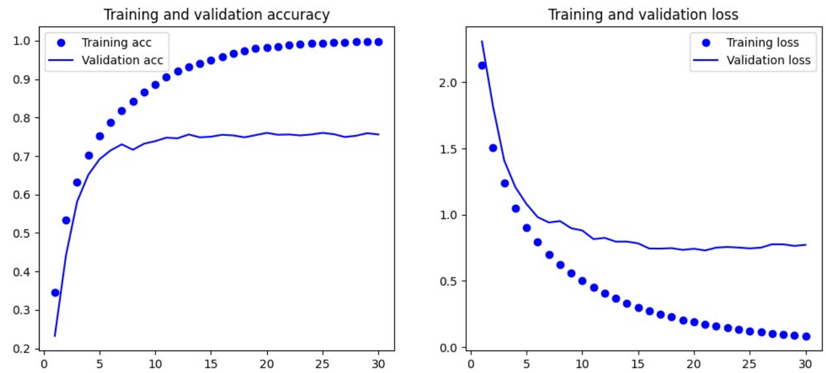

Figure 3: Plots for accuracy and loss on training and validation dataset

3 Other Experimentations

Before finalizing these two architectures, we have experimented on multiple other models and by tuning their parameters. Some of them have been mentioned below with their hyper parameters and training, validation accuracy er achieved:

3.1 DenseNet121

It is a densely connected convolutional neural network architecture where each layer receives feature maps from all preceding layers, promoting feature reuse and facilitating gradient flow, resulting in enhanced learning efficiency.

Train Acc.	Val Acc.	LR	Optimizer	Epochs	Batch Size	Regularization
0.9189	0.7262	0.0005	Adam	30	32	None

Table 3: DenseNet121 Model Training Details

3.2 MobileNetV2

It is a lightweight convolutional neural network designed for mobile and embedded vision applications, characterized by depthwise separable convolutions and inverted residuals to achieve efficiency without sacrificing performance.

Train Acc.	Val Acc.	LR	Optimizer	Epochs	Batch Size	Regularization
0.8837	0.1323	0.01	SGD	10	32	None

Table 4: Model Training Details

3.3 ResNet50

We have experimented with resnet architecture and tabulated the results in this image.

Overall Precision: 0.75
Overall Recall: 0.73

Classification Report:

	precision	recall	f1-score	support
1	0.98	0.55	0.71	85
2	0.78	0.72	0.75	160
3	0.98	0.73	0.84	140
4	0.43	0.67	0.52	72
5	0.93	0.89	0.91	136
6	0.82	0.78	0.80	112
7	0.96	0.64	0.77	42
8	0.48	0.78	0.59	41
9	0.64	0.59	0.61	73
10	0.68	0.75	0.71	53
11	0.80	0.87	0.84	140
12	0.86	0.91	0.89	112
13	0.39	0.63	0.48	43
accuracy			0.76	1209
macro avg	0.75	0.73	0.72	1209
weighted avg	0.80	0.76	0.76	1209

Figure 4: Classification Report for resnet 18

References

[1] Build a Deep Audio Classifier with Python and Tensorflow - Nicholas Renotte- https://www.youtube.com/watch?v=ZLIPkmmDJAc

[2] Deep Learning for Audio Classification - Seth Adams

Table

MODEL	Learn Rate, Mo...	Epoch	Training A...	Validation Acc...	L2 Reg	Batch
ResNet 18	0.01	10	1.0000	0.8933	0.0001	32
ResNet 18	None SGD	6	0.9982	0.8892	None	32
ResNet 18	0.001 SGD	30	0.9996	0.8610	0.0005	32
6-Layer CNN	0.01 SGD MOMENTUM=0....	30	0.8976	0.8564	None	
ResNet 18	0.0001 SGD Momentum = 0.9	30	0.9996	0.8495	None	32
ResNet18	0.00065 SGD	30	0.9959	0.8486	None	32
ResNet 18	0.001 SGD	30	1.0000	0.8453	None	32
ResNet 18	0.0005 SGD	40	0.9979	0.8387	None	32
ResNet 18	0.001 SGD	30	0.9996	0.8197	0.0001	32
ResNet 18	0.0003	30	0.9974	0.8089	None	16
Resnet 18	0.0001 RMSprop	10	0.9829	0.8089	None	32
ResNet 18	0.0001	70	0.9906	0.7957	None	16
ResNet 18	0.0001 SGD	150	0.9928	0.7792	None	32
EffecientNetB4	None	10	0.2035	0.1158	None	32
DenseNet121	0.0005	30	.9189	.7262	None	32

Figure 5: Architecture Experimentation

https://www.youtube.com/playlist?list=PLhA3b2k8R3t2Ng1WW_7MiXeh1pfQJQi_P

[3] Densely Connected Convolutional Networkshttps://arxiv.org/abs/1608.06993

[4] EfficientNet: Rethinking Model Scaling for Convolutional Neural Networkshttps://arxiv.org/abs/1905.11946

[5] MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applicationshttps://arxiv.org/abs/1704.04861

[6] Deep Residual Learning for Image Recognitionhttps://arxiv.org/abs/1512.03385