# Project on the UrbanSound8K Dataset

Carlos Pedrosa

**Abstract**

In this project I compare the performance of a basic Multi-Layer Perceptron (MLP) and a more complex Convolutional Neural Network (CNN) with residual connections. While the MLP struggles with generalization, the CNN, despite superior results, exhibits confusion between specific classes and lower robustness at test time.

# 1 Introduction

In this project, I tackle the problem of classifying the 10 types of sounds from the UrbanSound8K dataset, using two neural network architectures. This dataset is split into ten folds, therefore I do cross-validation in order to benchmark my models. Therefore, for each fold $i$ used to test the model, in the dataset, we use $(i + 1) \mod 10$ for validation and the remaining for training[1]. From this process we retrieve the parameters: test loss, validation loss, test accuracy, robustness (DeepFool),validation accuracy and confusion matrix. For this problem, I considered two approaches for model implementation: 2D CNN and MLP. Depending on the preprocessing. of course, it may be expected that the CNN model will fare better then the latter approach.

[1]I employed a practice that induces "data leakage" from validation to test results since I built my models and regularization based on validation fold = 1 and test set = others. This project aims to solidify intuition and understanding of these models, which this approach helped with, but it must be known that best practices encourage having an unseen fold during model tweaking . The proper practice was taken only with early stopping where I select the model that has the minimum validation loss for testing.

# 2 Implementation

## 2.1 Preprocessing

The audio files from the dataset have different sampling rates, different channel numbers and different durations. Therefore, as a first preprocessing step, I resampled the audio files to 22.05kHz, and averaged all the channels (i.e. an audio file with two channels becomes a file with one channel whose entries are the average of the two entries of the original file). In the final step, the audio files are extended by replicating each sound many times, resulting in a longer duration. Subsequently, all sounds are truncated to a fixed length of 4 seconds to ensure uniformity across the dataset. Both Neural Networks will receive the MelSpectrogram of the audio files as features. The Mel Scale is selected for its enhanced accuracy in representing human sound perception. All spectrograms are normalized to have a mean of 0.4 and a standard deviation of 0.3. This normalization aligns with the PyTorch implementation of ResNet models, which operates optimally with means and standard deviations in proximity to these values. During training, the MelSpectrograms are augmented with the SpecAugment technique: a combination of Time Masking (window of size 30, in this case) and Frequency Masking (window of size 10, in this case). In addition, I use Image Augmentation techniques: applying a random rotation of up to 5 degrees, random vertical flip and random horizontal flip. This appeared to yield less generalization error during experiments. I use the torchaudio implementation of the MelSpectrogram with the following parameters: 16384 fft components; normalized; 400ms window length; 31.5 ms;128 mels, such that the final output has a shape of (128,128).

## 2.2 Model Design and Learning

All models used output a vector $\hat{\boldsymbol{y}} \in \mathbb{R}^{10}$ of which is applied a softmax function $\hat{\boldsymbol{p}}$, so we can interpret the output as a probability distribution we are trying to approach. With this distribution, the loss is taken as the cross entropy between this distribution and a target distribution that is regularized using label smoothing of 0.005, to account for a minority of misclassifications that might have occured in the dataset. The learning optimizer that I chose to implement is AdamW (see the Pytorch page for the algorithm difference between Adam). This algorithm decouples the weight regularization from the explicit loss function. The chosen learning rate for both models is $1 \times 10^{-5}$ and the weight decay parameter is 0.1. The number of epochs chosen for training is **12** and the best model is chosen to be the one with the lowest validation loss. Also, the *amsgrad* argument is set to true in order to avoid ambitious steps in the final parts of the training process. In my experiments, this made the validation loss follow the training loss tendency for a larger number of epochs.

### 2.2.1 Different types of CNN

Many CNN architectures were toyed with, in the experiments. The CNNs I implemented were mainly of two types:

1. Sequences of convolutional layers (ReLU activations), followed by Max-Pooling layers. The final layers had the structure of an MLP preceded by an Average Pooling layer, in order to have a flattened out feature vector.

2. A ResNet.

Although, I will not present results regarding the first kind of models since they had significant disadvantages, compared with the ResNets, I believe it is worth describing some traits that I grasped from my experiments. These models were regularized with Dropout2d layers (they ignore some parts of feature map at random, in order to make it learn the "bigger picture" and prevent feature maps from "conspiring with each other" to overfit) and in the later designs, it was also tried to regularize this model with Batch Normalization: this regularization technique allows for a better gradient flow and therefore is good at preventing "dead kernels". This type of model needed more training epochs in order for the validation loss to reach a minimum and many times the validation loss wouldn't change consistently with the training loss. In addition, they needed significantly more time (this architecture is deeper than the ResNet described in the next section) to go through the same number of epochs, comparing with the other option.

### 2.2.2 The ResNet

I will now describe the architecture of the ResNet I implemented. Take $C_{I,O,K,s}$ to be the 2d convolution with a $K$ square kernel (with padding $\frac{K-1}{2}$) that takes as an input a feature map with $I$ channels and outputs $O$ channels with stride $s$. Additionaly, take $BN$ to be the Batch Normalization 2d function. Then, the output of the building block $BB_{I,O,K,s}$ of the resnet

is as follows for an input $\boldsymbol{x}$:

$$O_1 = ReLU(BN(C_{I,O,K,s}(\boldsymbol{x}))) \tag{1}$$

$$O_2 = BN(C_{O,O,K,s}(O_1)) \tag{2}$$

Now, if $s = 1$ and $I = O$ we just have:

$$BB_{I,O,K,s}(\boldsymbol{x}) = ReLU(O_2 + x)$$

Otherwise, we need to turn $\boldsymbol{x}$ into the "right shape":

$$I = BN(C_{I,O,1,s}(\boldsymbol{x})) \tag{3}$$

$$BB_{I,O,K,s}(\boldsymbol{x}) = ReLU(O_2 + I) \tag{4}$$

We notice that $BB_{I,O,K,s}$ has many layers but also encourages the gradient to not vanish (because of the last step where it uses the the feature map that it started with). This basic block is also used in the architectures of the ResNet18 and ResNet34, in their paper. The architecture $NN$ that I employed was similar to ResNet18, as I will show. Take $MP_{K,s}$ to be a MaxPooling layer with K and s being defined as above (the same with

padding). What this neural network does is the following:

$$O_1 = ReLU(BN(C_{1,64,7,2})) \tag{5}$$

$$O_2 = MP_{3,2}(O_1) \tag{6}$$

$$O_3 = BB^{(5)}_{64,64,3,1}(O_2) \tag{7}$$

$$O_4 = BB_{64,128,3,2}(O_3) \tag{8}$$

$$O_5 = BB^{(9)}_{128,128,3,2}(O_4) \tag{9}$$

$$O_6 = AdaptativeAvgPool(1,1)(O_5) \tag{10}$$

$$O_7 = Linear_{128,10}(O_6) \tag{11}$$

$$\tag{12}$$

Where $f^{(n)}$ means composing the function $f$ $n$ times and $Linear_{I,O}$ is a normal affine $\mathbb{R}^I \to \mathbb{R}^O$ transformation present in an MLP. This model showed superior results in validation accuracy (although with a bigger training loss) when comparing with the true ResNet18 or ResNet34 models. If I compare with ResNet18, I decreased the number of re-samplings of $\boldsymbol{x}$ through equation 3 and instead used more stride 1 layers and less layers which output a larger number of feature maps. This suggests that the information in the Mel Spectrograms are condensed in small areas in the "image" and in a smaller number of features then what ResNet18 and ResNet34 aim to capture (in other words, these models have too much capacity for this purpose), which seems reasonable, given that I'm dealing with frequency information.

## 2.3 MLP

This model is supposed to make a simpler treatment of the data provided, in order to show that the dense connections implied by the MLP architecture are more prone to learn the "noise" of the data or, in other words, overfit. The MLP will take as an input flattened versions of the Mel Spectrograms, in order to make the comparison with the CNNs more clear. The architecture I sticked with was as follows:

$$O_1 = ReLU(BN(Linear_{128^2,512}(\boldsymbol{x}))) \tag{13}$$

$$O_2 = ReLU(BN(Linear_{512,256}(O_1))) \tag{14}$$

$$O_3 = ReLU(BN(Linear_{256,128}(O_2))) \tag{15}$$

$$O_4 = Tanh(BN(Linear_{128,128}(O_3))) \tag{16}$$

$$O_5 = ReLU(BN(Linear_{128,256}(O_4))) \tag{17}$$

$$O_6 = ReLU(BN(Linear_{256,128}(O_5))) \tag{18}$$

$$Output = ReLU(Linear_{128,10}(O_6)) \tag{19}$$

$$\tag{20}$$

The *tanh* activation function was initially employed in the neural network to observe its behavior. However, there is a possibility that replacing it with *ReLU* could yield similar results. Before settling on this architecture, various experiments were conducted, including trying Dropout Regularization, adjusting the number of Batch Normalization ($BN$) layers, and modifying the number of layers and channels.

In terms of Dropout regularization, it was observed that its application

led to a slower training curve without any discernible improvement in validation performance. This suggests that $BN$ was effectively serving as a regularization technique.

Regarding $BN$, it was noticed, as mentioned earlier, that it contributes to a better training curve by facilitating improved gradient flow. Additionally, it imparts a beneficial regularization effect as all mini-batches, during training, become connected through the combined mechanisms of gradient descent and normalization imposed by $BN$.

# 3 Results and Discussion

In this section, I briefly present the results of both final architectures of the Neural Networks and leave some observations regarding them. In particular, I will present the validation loss curves as a function of training epochs for each of the test fold options. In addiction, I will discuss the final test loss, test accuracy (and its standard deviation), robustness [2] (using DeepFool type perturbations), test confusion matrix. All training was performed with the use of pytorch as the chosen framework, where I took advange of its CUDA capabilities during training. The GPU used was an Nvidia RTX 3060.

The validation curves for the ResNet architecture can be seen in figure 1 and that of the MLP is presented in figure 2 [3]. It is worth noting that the validation loss for the MLP model appears to have room for improvement with the use of more epochs. However, I chose to not do this because of time

[2]The robustness is calculated as in the article suggested in the project statement.

[3]In order to facilitate readability I chose to not include in this document the training curves. However, the training runs (as well as the validation data presented here) can be checked with the help of tensorboard in the proper directory of the zip file provided.

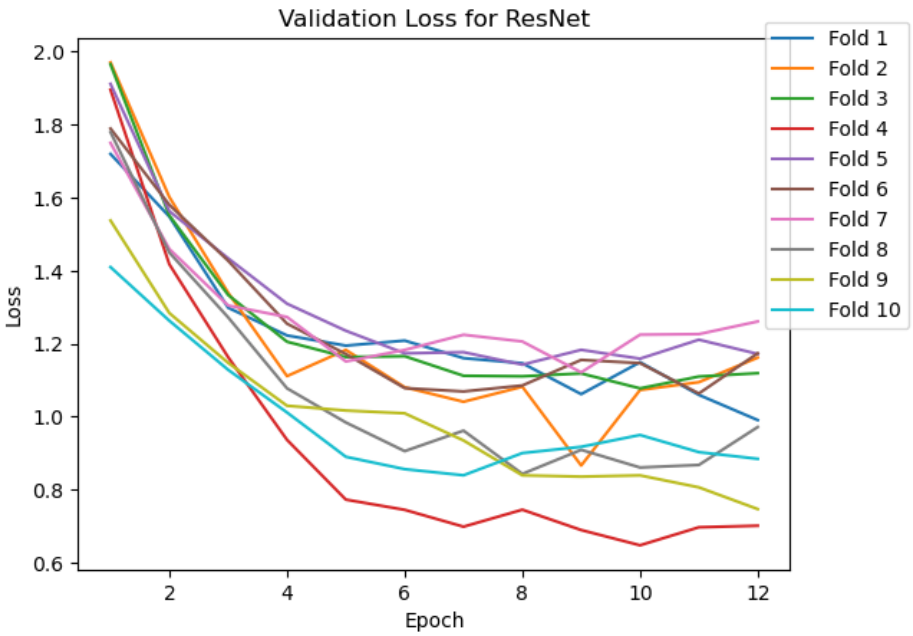


Figure 1: Validation loss for 12 epochs of training of the final ResNet architecture.

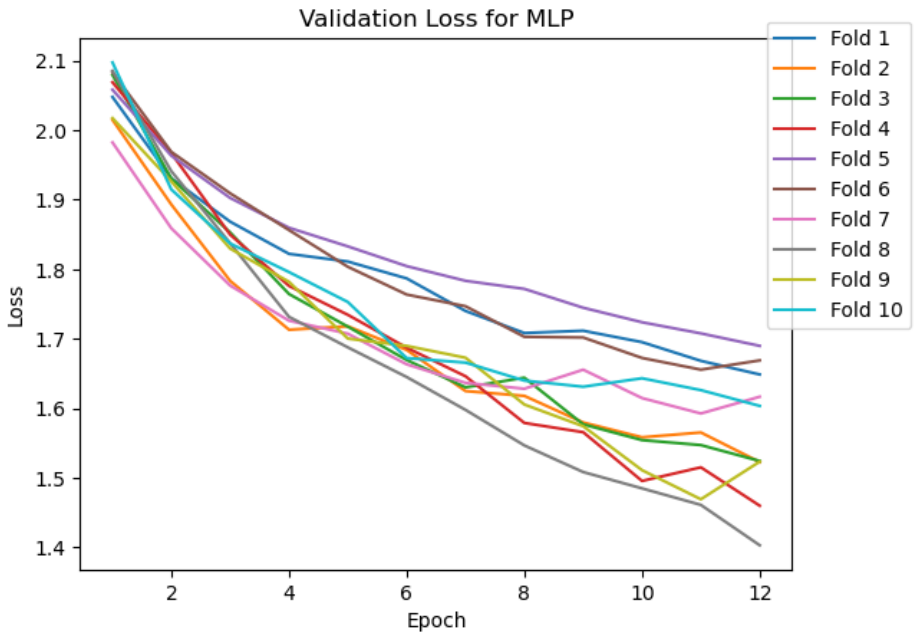


Figure 2: Validation loss for 12 epochs of training of the final MLP architecture.

constraints and by noticing in experiments that using more epochs didn't improve accuracy consistently. Therefore, this can be seen as a stronger form of early stopping in order to ensure greater robustness for the model. Examining the confusion matrices presented in figures 2 and 3, it is evident that, in both cases, the diagonal entries have higher values than other elements in their respective rows/columns, particularly in the ResNet architecture. Notably, in the fifth row, the ResNet model exhibits a tendency to confuse samples from class 0 and class 5, corresponding to air conditioner and engine idling, respectively. In contrast, the MLP model appears to be less susceptible to this confusion.

This observation may be linked to the varying robustness of these models whose values are found in table 1. Despite the ResNet outperforming the MLP in terms of accuracy during test time, the latter demonstrates greater robustness to perturbations in adversarial example generation. Intuitively, the similarity between mel spectrograms of air conditioner noises and engine idling noises could explain this phenomenon. Consequently, the MLP, being more resilient to noise or small similarities, experiences this issue to a lesser extent. In 1, it is evident that the validation losses and accuracy, both assessed at the iteration when the model is chosen for testing, exhibit similar values to the test losses. This observation implies that the model neither overfit nor underfit the data, indicating a balanced performance on both the validation and test sets.

| | ResNet | MLP |
|---|---|---|
| Test Accuracy | 69% | 53% |
| Test Accuracy standard deviation | 4% | 5% |
| Test Loss | 1.03 | 1.57 |
| Robustness | 0.004 | 0.014 |
| Validation Loss | 0.93 | 1.56 |
| Validation Accuracy | 70% | 52% |

Table 1: Performance Metrics for both models.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 49.9 | 1.5 | 4.3 | 7.3 | 6. | 12.9 | 0.1 | 12.1 | 2.6 | 3.3 |
| 1.4 | 28.9 | 0.3 | 0.3 | 2.9 | 0.1 | 0. | 1.1 | 1.2 | 6.7 |
| 1.3 | 0. | 77.9 | 8.3 | 1.3 | 2.3 | 1. | 1.1 | 2.7 | 4.1 |
| 3.2 | 0.2 | 8.5 | 76.8 | 2. | 1.4 | 1.8 | 0.2 | 4.5 | 1.4 |
| 3.1 | 0.5 | 1.7 | 1.6 | 69.4 | 2.5 | 1.3 | 15.5 | 3.5 | 0.9 |
| 20.6 | 0.3 | 3.6 | 0.6 | 6.5 | 55.6 | 0.1 | 9.8 | 2. | 0.9 |
| 0.3 | 0. | 0.4 | 1.4 | 0.7 | 0.3 | 33.5 | 0.5 | 0.1 | 0.2 |
| 9.3 | 0. | 1. | 0. | 14.4 | 11.3 | 0.1 | 63.5 | 0.3 | 0.1 |
| 1.9 | 3.3 | 8.2 | 3.1 | 3.9 | 1.6 | 0. | 0.4 | 65.1 | 5.4 |
| 4.6 | 4.3 | 7.2 | 1.3 | 2.2 | 0.4 | 0.1 | 1.1 | 0.9 | 77.9 |

Table 2: Confusion Matrix for ResNet

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 43.6 | 0.5 | 4.0 | 3.9 | 6.9 | 13.7 | 1.1 | 8.7 | 6.5 | 11.1 |
| 1.9 | 26.4 | 2.9 | 1.3 | 1.1 | 1.2 | 0.1 | 4.1 | 1.2 | 2.7 |
| 9.0 | 1.0 | 45.6 | 10.9 | 2.5 | 7.3 | 1.8 | 3.2 | 5.0 | 13.7 |
| 4.5 | 1.2 | 11.0 | 61.4 | 1.6 | 2.6 | 2.1 | 0.7 | 7.2 | 7.7 |
| 4.7 | 2.3 | 9.1 | 3.2 | 43.5 | 5.0 | 0.2 | 21.2 | 5.6 | 5.2 |
| 11.2 | 0.3 | 9.1 | 0.5 | 2.4 | 54.7 | 0.1 | 7.1 | 8.5 | 6.1 |
| 0.8 | 0.0 | 2.9 | 2.3 | 0.7 | 0.9 | 25.9 | 1.2 | 0.5 | 2.2 |
| 10.3 | 0.3 | 3.5 | 0.3 | 17.6 | 2.2 | 0.0 | 45.0 | 9.5 | 11.3 |
| 4.7 | 0.2 | 7.6 | 7.0 | 0.5 | 2.9 | 0.0 | 2.6 | 65.7 | 1.7 |
| 10.0 | 0.8 | 14.3 | 5.7 | 3.9 | 4.5 | 0.1 | 5.6 | 4.9 | 50.2 |

Table 3: Confusion Matrix for MLP

# 4 Conclusion

In this project, I delved into the implementations of two distinct Deep Neural Network architectures: a basic Multi-Layer Perceptron (MLP) and a more intricate Convolutional Neural Network (CNN) equipped with residual connections. The MLP exhibited challenges in generalization, whereas the CNN, despite achieving superior results, displayed a tendency for confusion between specific classes. Additionally, when assessing robustness through various performance metrics, it was observed that the CNN demonstrated less resilience at test time.

These findings underscore the trade-offs between model simplicity and complexity, with the MLP struggling in generalization but potentially offering better robustness. On the other hand, the CNN, while excelling in performance, encountered challenges in class differentiation and exhibited reduced robustness. These nuances highlight the importance of considering both accuracy and robustness metrics when selecting a neural network architecture based on the specific demands of the task at hand.