

POLITECNICO DI TORINO

Master's Degree in Electronic Engineering



**Politecnico
di Torino**

Master's Degree Thesis

Impact of Noise on Different Neural Network Architectures for Environmental Sound Classification

Supervisors

Prof. LAZARESCU MIHAI

Prof. LAVAGNO LUCIANO

Candidate

STABELLINI VALENTINA

July 2023

Acknowledgements

To my partner in life Tiberius,
To my family,
Thank you all to always have supported me.
Thanks to my supervisors Mihai and Luciano
for giving me the precious opportunity to "have fun" with Deep Learning.

Table of Contents

List of Tables	VI
List of Figures	VII
Acronyms	XII
1 Introduction	1
2 Deep Learning Overview	3
2.1 Machine Learning Basics	3
2.1.1 Linear Regression	3
2.1.2 Overfitting and Underfitting	4
2.2 Deep Feedforward Neural Networks	6
2.2.1 Deep Feedforward Networks working	6
2.2.2 Optimization Algorithms with Adaptive Learning Rate	8
2.2.3 Backward Propagation	9
2.3 Convolutional Neural Networks	10
2.3.1 Functioning of CNN	10
2.4 Transformer Architecture	16
2.5 Feature Extraction	20
2.5.1 Mel Spectrogram	20
2.6 Data Augmentation	22
2.6.1 Spectrogram Masking/Spec-Augmentation	22
3 Models Architecture and Dataset	28
3.1 Dataset	28
3.1.1 ESC50	28
3.1.2 AudioSet	30
3.2 State-of-art Architecture	31
3.2.1 CNN10	32
3.2.2 TFNet	33

3.2.3	mn40_as	35
3.2.4	HTS_AT	42
4	Metodology	45
4.1	Training	45
4.2	Inference	50
4.2.1	Dataset Selection for Inference	51
4.2.2	Noises Selection	51
4.2.3	Noise Insertion	53
5	Experimental Results	58
5.1	Model Complexity	58
5.2	Inference Results	59
6	Conclusion	73
A	Python Codes	75
	Bibliography	77

List of Tables

2.1	Mathematical definition of ReLU, Softmax and Sigmoid activation functions. Source: [6]	8
4.1	How the five folders are split for training and test: when a training is performed on fold 1 it means folds 2,3,4,5 are used as training set and fold 1 is used as test set and so on.	47
4.2	Features extraction details about reproduction sound, sampling frequency, window size, hop size and number of mel bins for each model	48
4.3	Data augmentation techniques used in pre-training and training . .	48
4.4	Reached accuracies during training for each fold and model	51
4.5	A comparison between testing accuracy and accuracy reached during inference without noise for each model. Also accuracy loss is listed .	52
4.6	Power spectral density and slope for colored noises	53
4.7	Nominal noises power calculated in model sensitivity bandwidth . .	56
4.8	The frequency band that the model is able to detect and classify . .	57
5.1	Model complexity in terms of number of parameters for each model	58

List of Figures

2.1	Overfitting, appropriate capacity and overfitting.	5
2.2	Typical capacity and error relationship.	6
2.3	A simple structure of a neural network.	7
2.4	ReLU activation function graph. Source: [6]	8
2.5	Sigmoid activation function graph. Source: [6]	8
2.6	Softmax activation function graph. Source: [6]	9
2.7	RGB additive model.	11
2.8	A $5 \times 5 \times 3$ matrix RGB image example. Each channel defines the pixel intensity across the image.	12
2.9	A convolution operation with 2×2 kernel and stride length 1 in a matrix dimension of 4×4 . The resulting matrix have dimension of 3×3	13
2.10	A convolution operation with 3×3 kernel and stride length 2. In this case, padding is required.	13
2.11	An Average Pooling operation in a 4×4 input matrix with 2×2 kernel dimension is performed. The resulting feature map has dimension 2×2	14
2.12	An Max Pooling operation in a 4×4 input matrix with 2×2 kernel dimension is performed. The resulting feature map has dimension 2×2	14
2.13	Visual representation of max and avg pooling on an image.	15
2.14	Dropout in fully connected neural nets	15
2.15	Fashion-MNIST dataset consists of 10 classes of clothing, such as shirts, pants, clothes, shoes, bags, etc. A matrix of pixel intensity values is used to represent each grey-scale image, which has a dimension of $28 \times 28 \times 1$ pixels. Source: [24]	16
2.16	A sample CNN.	17
2.17	Transformer Architecture proposed in [25]. Source: [25]	18
2.18	Multi-Head Attention of Transformer in [25]. Source: [25]	19

2.19	Illustration of how short-time Fast Fourier Transform works. The signal is first multiplied by overlapping window producing consecutive windowed segments. Then FFT is applied to get spectrogram. Source: [29]	24
2.20	Plot of Hann Window with 50 samples length. Source: [33]	25
2.21	Conversion from Hertz scale to Mel scale. When frequency is 1,000 Hz it corresponds to 1,000 mel.	25
2.22	Mel Filter Banks.	26
2.23	Time Warp Data Augmentation.	27
2.24	Frequency Masking Data Augmentation.	27
2.25	Time Masking Data Augmentation.	27
3.1	Sound classes of ESC-50 dataset. Source: [44].	29
3.2	The 19 categories selected from ESC-50 for the experiment.	30
3.3	The top two layers of Audioset Ontology. For example "Human voice" covers the classes Speech, Shout, Screaming, Whispering, Laughter, Crying, Wail, Sigh, Humming, Groan, Grunt and Yawn. Source: [48]	32
3.4	Paperswithcode screenshot in which filters by method, task and dataset are applied to find moodels. Source: [53]	33
3.5	Audio Classification Accuracy trend on ESC-50 of SOTA models over the years. Source: [53]	33
3.6	Audio Classification on ESC-50 rank of SOTA models. Source: [53]	34
3.7	Temporal attention allows the network to identify the most relevant frames in order to recognize where the sound occurs. Source: [49].	35
3.8	Spectral attention attention is employed to give different level of attention to different frequency bands which are significant for the task. Source: [49].	35
3.9	Parallel temporal-spectral attention attention mechanism applied to each convolutional block. Source: [49].	36
3.10	TFNet overall architecture. A zoom of what is inside a convolutional block. Source: [54]	36
3.11	ViT.	37
3.12	AST.	37
3.13	PaSST.	38
3.14	MobileNetV1 Depthwise Separable Convolution.	39
3.15	MobileNet V2 layer.	40
3.16	MobileNet V3 block.	41
3.17	mAP results obtained during student training on AudioSet using different values of λ and τ . Source: [55]	42
3.18	MobileNet performance on AudioSet with and without KD and pre-training on ImageNet. Source: [55]	42

3.19	HTS_AT.	44
4.1	By default, all accessible models undergo pre-training on ImageNet (otherwise indicated as 'no_im_pre'), which is followed by training on AudioSet. Source: [66]	46
4.2	Plot contrasts each task's results for the PANNs (CNN14), PaSST, and mn40_as models. The score is normalised using a model's peak performance. Source: [66]	47
4.3	On the left loss functions versus iteration and on the right accuracy trend over iterations for the CNN10 model	49
4.4	On the left loss functions versus iteration and on the right accuracy trend over iterations for the TFNet model	49
4.5	On the left loss functions versus epoch and on the right accuracy trend over iterations for the mn40_as model	50
4.6	On the left loss functions versus epoch and on the right accuracy trend over iterations for the HTS_AT model	50
4.7	Power spectral density of the white noise	53
4.8	Power spectral density of the pink noise	54
4.9	Power spectral density of the brownian (red) noise	54
4.10	Power spectral density of the blue noise	55
4.11	Power spectral density of the violet noise	55
5.1	Models Accuracy when white noise is applied.	60
5.2	Class 31 audio without (up) and with white noise (down) applied at 10 dB SNR mel spectrogram.	60
5.3	Models Accuracy when pink noise is applied.	61
5.4	Class 31 audio without (up) and with pink noise (down) applied at 10 dB SNR mel spectrogram.	61
5.5	Models Accuracy when brownian (red) noise is applied.	62
5.6	Class 31 audio without (up) and with brownian (red) noise (down) applied at 10 dB SNR mel spectrogram.	62
5.7	Models Accuracy when blue noise is applied.	63
5.8	Class 31 audio without (up) and with blue noise (down) applied at 10 dB SNR mel spectrogram.	63
5.9	Models Accuracy when violet noise is applied.	64
5.10	Class 31 audio without (up) and with violet noise (down) applied at 10 dB SNR mel spectrogram.	64
5.11	Models Accuracy when TV noise is applied.	65
5.12	Class 31 audio without (up) and with TV noise (down) applied at 10 dB SNR mel spectrogram.	65
5.13	Models Accuracy when door open and close noise is applied.	66

5.14	Class 31 audio without (up) and with door open and close noise (down) applied at 10 dB SNR mel spectrogram.	66
5.15	Models Accuracy when dog noise is applied.	67
5.16	Class 31 audio without (up) and with dog noise (down) applied at 10 dB SNR mel spectrogram.	67
5.17	Models Accuracy when car engine pass noise is applied.	68
5.18	Class 31 audio without (up) and with car engine pass noise (down) applied at 10 dB SNR mel spectrogram.	68
5.19	Models Accuracy when rain drops on window noise is applied. . . .	69
5.20	Class 31 audio without (up) and with rain drops on window noise (down) applied at 10 dB SNR mel spectrogram.	69
5.21	Confusion matrix obtained for CNN10 model, white noise at SNR of -25 dB.	70
5.22	Confusion matrix obtained for TFNet model, white noise at SNR of -25 dB.	70
5.23	Confusion matrix obtained for mn40_as model, white noise at SNR of -25 dB.	71
5.24	Confusion matrix obtained for HTS_AT model, white noise at SNR of -25 dB.	71
5.25	Confusion matrix obtained for mn40_as model, white noise at SNR of -10 dB.	72

Acronyms

MSE

Mean Square Error

ReLU

Rectified Linear Unit

RGB

Red Green Blue

CNN

Convolutional Neural Network

FC

Fully Connected

RNN

Recurrent Neural Network

ESC

Environmental Sound Classification

SOTA

State-of-the-art

TF

Temporal-Frequency

SE

Squeeze-Excitation

KD

Knowledge Distillation

GA

Global Attention

WA

Window Attention

FFT

Fast Fourier Transform

STFT

Short-Time Fourier Transform

SNR

Signal to Noise Ratio

Chapter 1

Introduction

The constantly increasing need to classify environmental sounds such as the breaking of a glass, the honking of a car, the crying of a child can lead to improvements in safety and health of society. The detection of human activities can bring advantages in contexts such as home security for the identification of intrusions or fall detection, and in the energy efficiency of buildings to regulate lighting, heating or air conditioning. These are just a few of the possible applications that the classification of environmental sounds can have, which is why more and more researchers decide to deepen the study and develop advanced algorithms such as Deep Learning. In general despite the classification of ambient sounds is important, it is a challenge that is still complicated because this type of sounds do not have a well defined structure and can vary in terms of intensity, duration and spectral content. They can also be influenced by environmental factors such as the presence of background noise that can lead to incorrect classification and degradation of performance. From this arises the importance of seeking increasingly sophisticated techniques for deep learning in the classification of environmental sounds that can exceed this limit. The aim of this research is to assess the impact of noise on different neural network architectures of Deep Learning for the classification of environmental sounds when subjected to background noise of different natures, with the goal of identifying which techniques could make a model more robust and provide input into the development of future networks, with the hypothesis that the use of advanced techniques, such as pre-training and attention mechanisms, improves the classification capacity of the model in the presence of background noise.

In the following chapters follow a brief introduction to Deep Learning and a description of the most common basic architectures such as convolutional neural networks (CNN) and Transformer also focusing on the main technique of extracting audio characteristics surrounded by the most common data augmentation techniques. The reference dataset (ESC-50) and the related models on which the experiment is built will be described below. In addition, underlining the starting hypothesis, a brief description of AudioSet, a dataset of ambient sounds wider than the ESC-50, which some models use in the pre-addextraction phase. The methodology is illustrated in the next chapter to highlight the methods and procedures used to conduct the experiment. This section provides details on training, noise selection and noise creation and their inclusion in the test dataset. Finally, experimental results and conclusions, close the structure of the chapters of this thesis in which the results obtained are presented and discussed accompanied by final conclusions that will include a summary of the thesis and the results.

Chapter 2

Deep Learning Overview

2.1 Machine Learning Basics

A machine learning algorithm is programmed to have a final purpose, such as recognizing sounds or images, which is made possible through a training phase. In this thesis we will address a classification problem that, starting from an input described by a vector \mathbf{x} , the algorithm must be able to determine which of the k categories it belongs to. The output y represents an entire number from 0 to $k - 1$ that refers to the category calculated by the algorithm. The above category, however, must be compared with the actual label category to determine whether the output has been calculated correctly or not. This is useful because during the training and validation phase of the algorithm, it allows you to have a performance reference metrics through accuracy. [1]

2.1.1 Linear Regression

Linear regression is a machine learning algorithm that predicts a y output from an input vector \mathbf{x} using a linear-type function. Defining y the output value predicted by the algorithm and \hat{y} the value it should assume, we can describe linear regression as (2.1)

$$\hat{y} = \mathbf{w}^T \mathbf{x} \tag{2.1}$$

where \mathbf{w} identifies a vector of parameters. Parameters can be seen as weights that influence the importance of features on prediction, in particular the higher the weight in amplitude, the more the feature matters and, on the contrary, if it is close to zero, it means that x_i will not affect.

The Mean Square Error is also defined as algorithm evaluation metric on the

test set (2.2)

$$MSE_{\text{test}} = \frac{1}{m} \sum_i (\hat{\mathbf{y}}^{\text{test}} - \mathbf{y}^{\text{test}})_i^2 = \frac{1}{m} (\|\hat{\mathbf{y}}^{\text{test}} - \mathbf{y}^{\text{test}}\|_2)^2. \quad (2.2)$$

To have good performance it is needed to get a difference between predicted output and small real output, implementable by a good choice of \mathbf{w} weights during the training phase.

So the algorithm looks for weights (2.5) that minimize (2.4) MSE_{train} (2.3)

$$\nabla_{\mathbf{w}} MSE_{\text{train}} = 0 \quad (2.3)$$

$$\nabla_{\mathbf{w}} \frac{1}{m} (\|\hat{\mathbf{y}}^{(\text{train})} - \mathbf{y}^{(\text{train})}\|_2)^2 = 0 \quad (2.4)$$

$$\mathbf{w} = (\mathbf{X}^{(\text{train})T} \mathbf{X}^{(\text{train})})^{-1} \mathbf{X}^{(\text{train})T} \mathbf{y}^{(\text{train})}. \quad (2.5)$$

This just described is an example of a machine learning algorithm although simple, it is still significant. [2]

2.1.2 Overfitting and Underfitting

So far we have described an optimization problem that deals with finding the weights that minimize the error in training, but what really impacts performance, is the error you get in the test phase.

But how can the training error affect the test error?

Are the training sets and test sets generated with the same probability distribution $p_{\text{data}}(\mathbf{x}, \mathbf{y})$ (data generating process) and identically distributed (are they the same thing?) And if each example, belonging to the same data set, is independent of each other, for fixed values of \mathbf{w} the expected errors on the training set and the test set are exactly the same because both are generated by the same process. However, what differentiates the two situations is that the parameters are not fixed at the same time and for this reason it is expected that the error on the test is greater or equal to that expected for the training.

The main condition for having a good machine learning algorithm is, first, to get the error on the low training and secondly to a difference between training error and test error as small as possible. These goals are important because if not achieved they can lead to problems such as overfitting, it means that the algorithm learned well during the training, but failed to good accuracy in the test, and the underfitting, which, on the contrary, the system did not learn enough in the training and therefore was not able to get a good accuracy.

The concept of capacity describes the ability to control and adapt the model by modifying the capability of an algorithm, for example by expanding the set of functions.

More practically increasing the capacity would mean allowing a linear regression algorithm (2.6) to work even with square functions (2.7) and are not linear passing from

$$\hat{y} = b + wx \tag{2.6}$$

$$\hat{y} = b + w_1x + w_2x^2. \tag{2.7}$$

Despite the square function on the x input, the output is still linear compared to the w parameters.

The **Figure 2.1** intuitively displays the underfitting due to a poor algorithm capacity so it is not able to solve too complicated tasks that in the example described above translates into having a set of functions limited to only linear function. On the contrary, we also observe the overfitting in which the algorithm has a too high capacity compared to that required for the resolution of tasks. This results in having extended the space of polynomial functions to the ninth degree. The optimum situation, on the other hand, is achieved when the capacity is sufficient for the algorithm requests and, similar to the example, is obtained with function space up to the second degree.

In real-world situations that do not only include linear regression algorithms, the training error decreases asymmetrically to the lowest value, while the test error, when the model capacity turns out to be too high, returns to be significant and far from the training target as well documented in **Figure 2.2** [3].

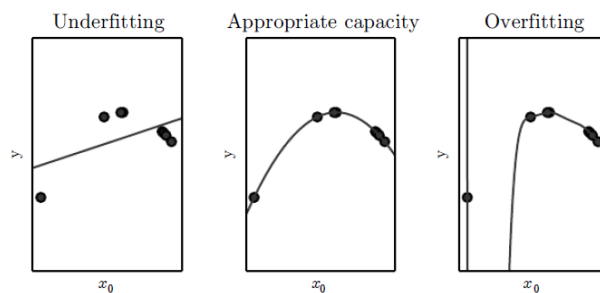


Figure 2.1: On the left a linear function fit to the data on the left figure exhibits underfitting because it is unable to account for the data’s curvature. In the middle, a quadratic function that fits the data well generalises to points that are not visible. It doesn’t have a lot of overfitting or underfitting problems. On the right, a polynomial of degree 9 that was fitted to the data exhibits overfitting. Source: [3]

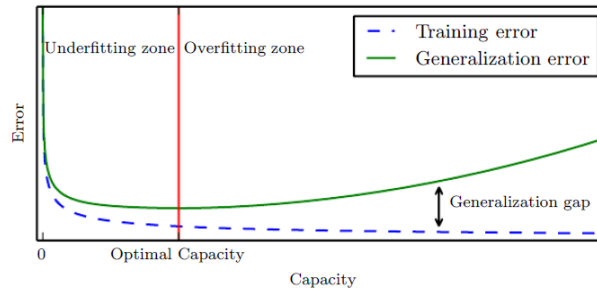


Figure 2.2: Typical capacity and error relationship. Error in training and testing reacts dissimilarly. Both the training error and the generalisation error are large near the left end of the graph and this indicates the underfitting regime. Training error reduces as capacity rises, but the difference between training and generalisation error widens, this means that the overfitting regime is reached, where capacity is too high and above the ideal capacity. Source: [3]

2.2 Deep Feedforward Neural Networks

2.2.1 Deep Feedforward Networks working

Deep Feedforward Neural Networks (a visual representation in **Figure 2.3**) is a type of deep learning model that aims to approximate a \hat{f} function.

In a classification problem, the network defines a new function $\mathbf{y} = f(\mathbf{x}; \boldsymbol{\theta})$ in which it determines which parameters are $\boldsymbol{\theta}$ that best approximate the function $\mathbf{y} = \hat{f}(\mathbf{x})$.

"Feedforward" is the term that describes the path of information flowing through the network starting from the input \mathbf{x} , intermediate calculations to calculate the function and finally to the output \mathbf{y} .

"Networks", however, identifies the composition of different functions. Assuming you have a network consisting of the first layer that has a function $f^{(1)}$, a hidden layer with the function $F^{(2)}$ and a output layer which has $F^{(3)}$, the final function will be the composition of the three functions $f(\mathbf{x}) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x})))$. The concept of "Deep" lies in the fact that the neural network might be so complex that it would have to be described with an enormous amount of layers, that is, the chain of composition of functions would not be limited to three, as in the example, but also several hundred or thousands.

The neural network training phase requires that each \mathbf{x} input is mapped into the output with the corresponding labels, i.e. the actual output, to allow driving \mathbf{x} in such a way that the approximate function produces a \mathbf{y} output as close as possible to $\hat{f}(\mathbf{x})$.

The generic function f is called activation function which aims to introduce nonlinearities into the neural network. These functions enable nonlinear mapping

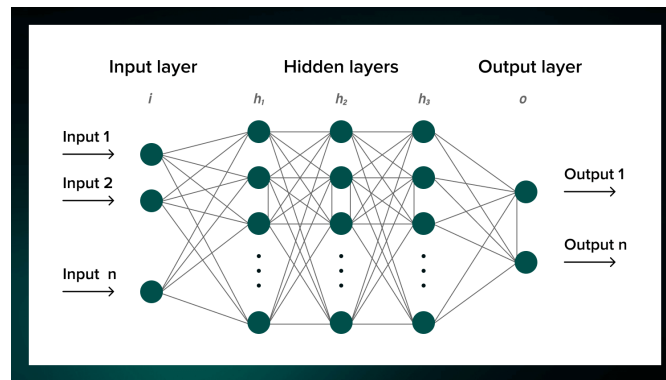


Figure 2.3: A simple structure of a neural network consists of an input layer, three hidden layers, and one output layer. In binary classification, the output layer possesses a single neuron, on the contrary, in non-binary classification, there would be one neuron for each class. The output is generally passed through an activation function to transform it into probability. Source: [4]

between input and output and, being differentiable, can be easily integrated during the Backpropagation algorithm [5].

The most used activation functions are listed below :

- ReLU (Rectified Linear Unit): is the most widely used function as it accelerates the convergence of the descent of the gradient to the global minimum and results in faster and more efficient learning. However, in **Figure 2.4** it is easily noticed that for negative x values, the function is value zero and this implies that when a neuron enters this condition, they do not respond to input or error variations [6].
- Sigmoid: is a function that receives real numbers as input and returns a real number between 0 and 1 as output and this makes it mainly used in binary classification problems. But the output, not being centered at zero, makes it more difficult to optimize and update the gradient (**Figure 2.5**) [6].
- Softmax: the combination of sigmoid functions gives rise to the softmax activation function. It is generally used in multi-category classification and in the last layers of a neural network to normalize output in a probability distribution of each individual category **Figure 2.6** [7, 8].

Table 2.1: Mathematical definition of ReLU, Softmax and Sigmoid activation functions. Source: [6]

ReLU	$f(x) = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x > 0 \end{cases}$
Sigmoid	$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^J e^{x_j}}, i = 1, \dots, J$

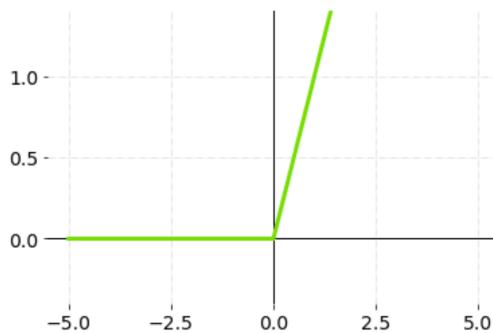


Figure 2.4: ReLU activation function graph. Source: [6]

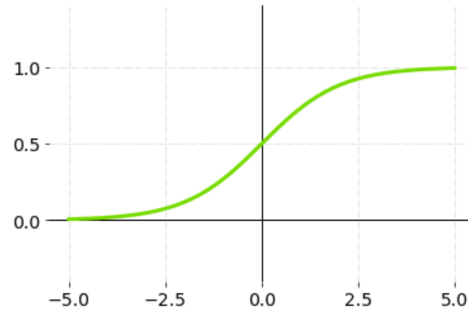


Figure 2.5: Sigmoid activation function graph. Source: [6]

2.2.2 Optimization Algorithms with Adaptive Learning Rate

The concept of optimisation describes a situation in which minimising (or maximising) a function is needed. When a minimisation of a function to find a local minimum is required, the basic idea is to find the direction in which the function decreases faster by proceeding in the opposite direction to the gradient of the function at that point. This procedure is called "gradient descent" or "method of steepest descent".

The next steepest point is defined as:

$$\mathbf{x}' = \mathbf{x} - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x}) \tag{2.8}$$

where ϵ is the step size or learning rate. The negative sign before the gradient indicates that the direction of the local minimum is in the opposite direction from the gradient.

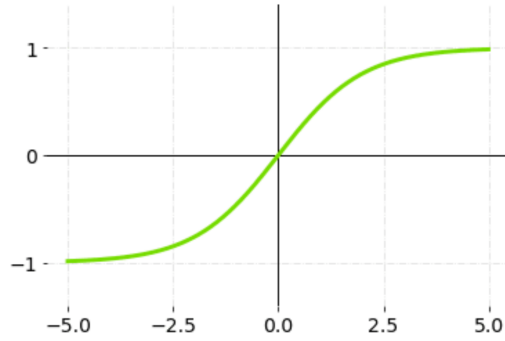


Figure 2.6: Sigmoid activation function graph. Source: [6]

More generally, iterating the algorithm, the point should converge towards the local minimum [9]

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \epsilon \nabla_{\mathbf{x}} f(\mathbf{x}_n), n \geq 0. \quad (2.9)$$

The basic idea is to use gradient descent to minimise the loss function during algorithm training [10].

2.2.3 Backward Propagation

The backpropagation algorithm uses the gradient propagation rule to calculate errors in network weights. The process can be divided into two phases: the forward pass phase and the reverse propagation phase.

In the forward propagation phase, the x input is passed through the network to generate the y output. The activation function is used to generate the output. The activation function can be a sigmoid function, ReLU function, or others.

In the reverse propagation phase, the output error is calculated against the desired output \hat{y} . The error is given by the error function $L(y, \hat{y})$. The error is then spread back through the network to calculate the weight errors.

The rule for updating weights is given by

$$\Delta w(i, j) = -\epsilon \frac{\partial L}{\partial w(i, j)} \quad (2.10)$$

where $w(i, j)$ is the weight that connects the i -first neuron of the input layer with the j -third neuron in the output layer, ϵ (learning rate) is a learning constant, and $\frac{\partial L}{\partial w(i, j)}$ is a partial derivative of the error function relative to the weight $w(i, j)$.

The backpropagation algorithm then uses gradient descent to optimise network weights. This means that weights are updated in the negative gradient direction, i.e., in the direction that reduces the value of the error function [11][12][13][14].

Loss Functions

The loss function gradually gains the ability to lower prediction error with the aid of some optimisation functions. Depending on learning task, loss functions may be divided into two main categories, such as regression losses and classification losses.

Let's focus our attention on classification loss only, since this thesis will analyse models for audio classification. The most typical function for classification task is Cross Entropy Loss defined in (2.11). This formulation suggests that loss function rises when predicted output \hat{y}_i diverges from actual output y_i [15]

$$CrossEntropyLoss = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)). \quad (2.11)$$

2.3 Convolutional Neural Networks

2.3.1 Functioning of CNN

A convolutional neural network is a type of deep learning algorithm that is based on mathematical convolutional operations. Considering two functions $f(t)$ and $g(t)$ defined on \mathbb{R} , the convolutions between the two functions are defined as (2.12) [16]

$$s(t) = (x * w)(t) := \int_{-\infty}^{+\infty} x(\tau)w(t - \tau)d\tau. \quad (2.12)$$

In practise, one can imagine wanting to track the position of a spacecraft with a laser sensor. The sensor produces a $x(t)$ output corresponding to the x position at the moment t . During measurement, the value is subject to uncertainty caused by noise and to obtain a more accurate estimate, it would be necessary to collect various measurements and mediate them. Since, in the meantime, the position may have changed and the spacecraft continues its movement, we want to give more importance to the latest measurements. Similarly, if a weighed $w(\tau)$ function of duration τ for the entire measurement is applied, the result will be a new function representing the ship's position in the most filtered time.

Convolutional neural networks are widely successful in image recognition applications thanks to their structure. In convolutional neural networks, x represents the input, the weighed w function is called kernel and the output generated by s refers to bfeature map. In particular, if the input is a digital image, the convolutional operation will be discretized into a two-dimensional as shown in (4.10) [16]

$$s(i, j) = (X * K)(i, j) = \sum_m \sum_n X(m, n)K(i - m, j - n) \quad (2.13)$$

where X represents the colour matrix of the image and K the kernel matrix [16].

Digital images have the size $(Height) \times (Breadth) \times (Numberofchannels)$ where the number of canals identifies the range of colours used for the representation.

If the last dimension is 1, it means digital image has a grayscale image, but if the image is coloured, it has dimension of 3 (RGB). RGB (Red Green Blue) is a colour additive model (**Figure 2.7**) widely used in digital displays for image representation that is based on colour representation as a combination of intensities of three main colours, such as red, green and blue.

The choice of this model is essentially inspired by the physiological functioning of the human eye, which is composed of photoreceptive cells, that is, light-sensitive cells, which respond best to red light (long wavelength), green light, and blue light (short wavelength). The brain is able to distinguish a wide range of colours due to the difference between these three colours.

An example of a $5 \times 5 \times 3$ digital image that uses the RGB colour model is presented in **Figure 2.8**. There are three matrices, one per colour, and each element in the matrix indicates the intensity of the reference colour. Therefore, we will have a matrix indicating the intensity of the green colour in the image for each pixel, one matrix for red and one for blue. The composition of the three matrices returns the colour range in the image [17].

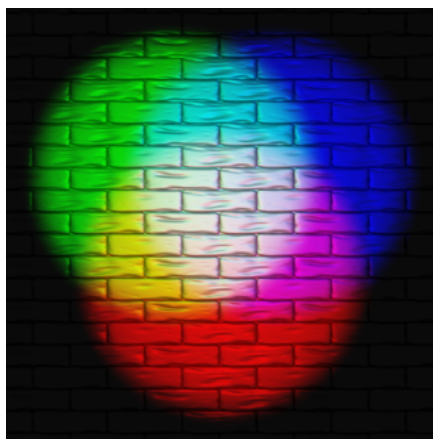


Figure 2.7: When primary colour lights are projected on a surface, additive colour mixing occurs when the three primary colours are combined in equal amounts to form white. The composition of them in different intensity, creates a wide range of colours. Source: [17]

Typical layers which compose a CNN are:

- **Convolution layer:**

The convolution layer consists of a kernel matrix that acts as a filter to extract high-level features such as the contours of an image, colors and gradient orientation.

In **Figure 2.9** is represented a typical convolution operation, for simplicity of

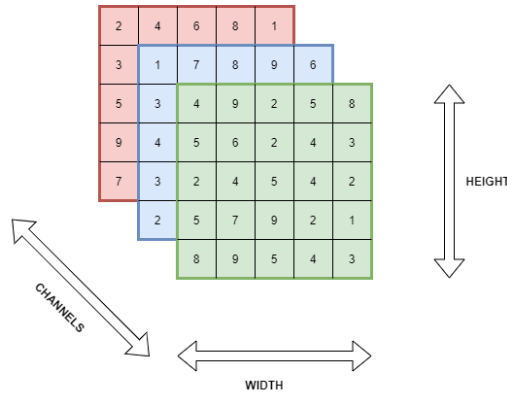


Figure 2.8: A $5 \times 5 \times 3$ matrix RGB image example. Each channel defines the pixel intensity across the image.

representation, the input is a grey-scale image with a single colour channel, with a 2×2 kernel size and stride length unit.

This operation can also be replicated when the number of channels is more than one (three in the case of RGB image). The resulting matrix is called **activation map** and has the final dimension $W_{out} \times H_{out} \times C_{out}$ obtained as follows:

$$\begin{aligned}
 - W_{out} &= \frac{W_{in} + 2P}{S} + 1 \\
 - H_{out} &= \frac{H_{in} + 2P}{S} + 1 \\
 - C_{out} &= K
 \end{aligned}$$

Where W_{in} and H_{in} are the sizes of the input image, S represents the stride length, i.e., the number of pixels the kernel moves from one scan to the next in the input matrix. When $S = 1$ the kernel moves one pixel at a time, while when $S = 2$ it moves two pixels at a time. The P value defines the amount of padding added. In **Figure 2.10** is shown an example of when the padding is applied to the original input is shown. The input has a $5 \times 5 \times 1$ size and a 3×3 kernel, with stride length equal to 2, the number of pixels in the image would not be enough to perform a convolution, so H and W are increased by adding zeros around the matrix [18].

- **Pooling layer:**

Suppose using an 8K image with a size of 7680×4320 as an input to a convolutional neural network. Convolutional layers do not significantly reduce the size of the matrix and would require a high number of layers as well as large computing power for image processing. For this problem, a layer of

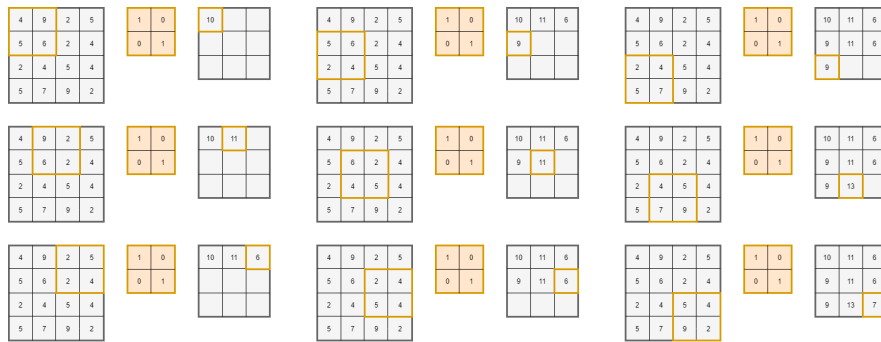


Figure 2.9: A convolution operation with 2×2 kernel and stride length 1 in a matrix dimension of 4×4 . The resulting matrix have dimension of 3×3 .

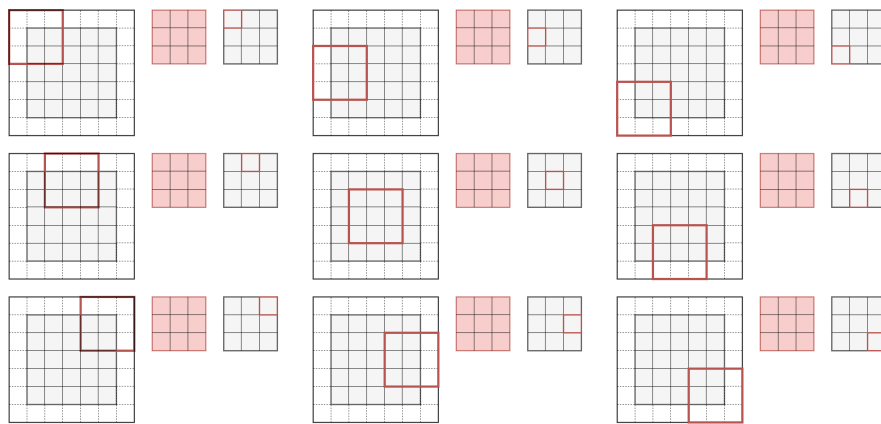


Figure 2.10: A convolution operation with 3×3 kernel and stride length 2. In this case, padding is required.

pooling is used that aims precisely to significantly reduce the size of feature maps in the hidden layers, as shown by **Figure 2.11** and **Figures 2.12**.

The final dimensions of the feature map are:

- $W_{out} = \frac{W_{in}}{S} + 1$
- $H_{out} = \frac{H_{in}}{S} + 1$
- $C_{out} = C_{in}$

Pooling is also used to extract and send the most important features to the next layer.

There are two types of pooling (**Figure 2.13**): Average Pooling Layer which returns the average of the values covered by the kernel, and Max Pooling layer which returns the maximum value of the portion of the image covered by the

nucleus. The choice of type of pooling is dictated by the ultimate purpose, Avg Pooling makes the image more regular, while Max Pooling identifies the brighter pixels and turns out to be especially useful when the background of the image is purely dark.

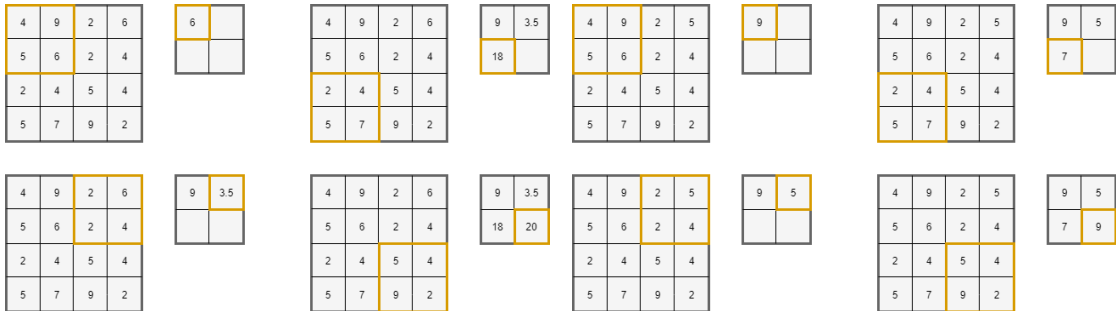


Figure 2.11: An Average Pooling operation in a 4×4 input matrix with 2×2 kernel dimension is performed. The resulting feature map has dimension 2×2 . **Figure 2.12:** An Max Pooling operation in a 4×4 input matrix with 2×2 kernel dimension is performed. The resulting feature map has dimension 2×2 .

- **Batch Normalization:**

[20] introduces a brand-new method for significantly quickening deep network training. It is based on the idea that covariate shift (having different sources of data inside the same range), which is known to make training machine learning systems more difficult, also applies to sub-networks and layers and that eliminating it from internal activations of the network can help with training. This technique consist of normalising each mini-batch and backpropagate the gradients through the normalisation parameters to enable stochastic optimisation techniques frequently employed in deep network training. Batch normalisation maintains the network’s capacity for representation by only adding two more parameters per activation.

- **Dropout:** When a model during the training starts to learn statistical noise means there is overfitting. This leads to complex co-adaptations, which fails to generalize to the unseen dataset. In [21] is introduced a technique called Dropout: when a few units (nodes) are dropped at random (**Figure 2.14**), layers are forced to assume varying amounts of input responsibility, assuring generalisation and minimising the overfitting issue [22].

- **Fully Connected layer (FC) and Global Average Pooling:**

A group of interdependent non-linear functions make up neural networks. An independent neuron (or perceptron) performs each function. The neuron in

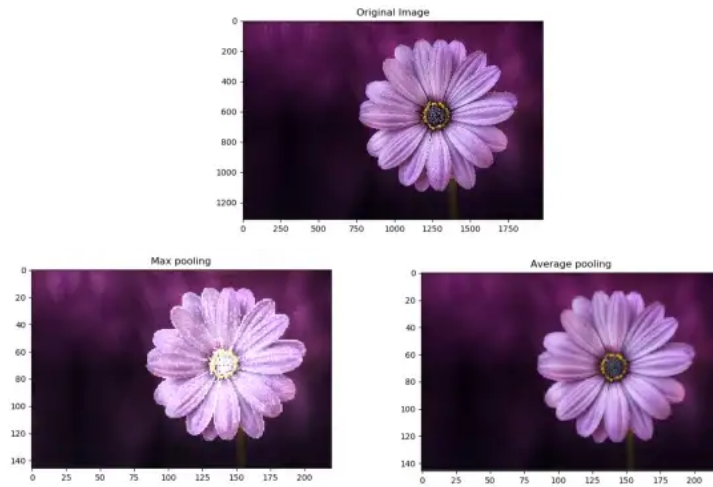


Figure 2.13: Original image on the top, Max Pooling on left below and Average Pooling on the right below. Sharp features may not be visible when using the average pooling approach since it smooths down the image. The brighter pixels in the picture are chosen via max pooling. It is helpful when we just care about the image's lighter pixels and the image's dark backdrop. Source: [19]

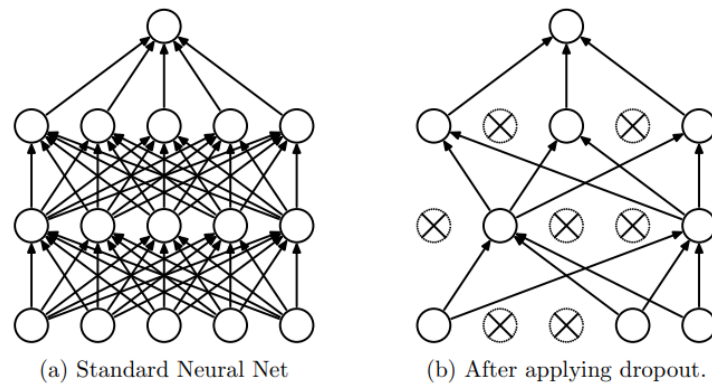


Figure 2.14: (a) standard neural net with 2 hidden layer. (b) An illustration of a thinned net created by applying dropout to the left network. Source: [21]

fully connected layers transforms the input vector linearly using a weights matrix. The result is then subjected to a non-linear transformation using a non-linear activation function f . The representation between the input and the output is mapped using the FC layer.

Let's take an example of a typical CNN architecture taken from [23] which uses a dataset of Zalando's article [24] images consisting of a training set of 60,000 examples and a test set of 10,000 examples. Each example is a 28×28 grayscale image, associated with a label from 10 classes.

The draft architecture proposed by [23] is composed as shown in **Figure 2.16**. The convolution layer is intended to extract features through the application of a filter; batch normalisation allows to initialise the weights of the matrices to a Gaussian distribution during training; and max pooling makes it so that the sizes are significantly reduced between one layer and the next.


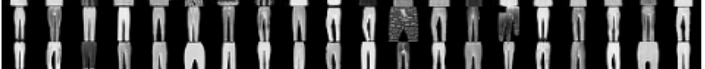








Label	Description	Examples
0	T-Shirt/Top	
1	Trouser	
2	Pullover	
3	Dress	
4	Coat	
5	Sandals	
6	Shirt	
7	Sneaker	
8	Bag	
9	Ankle boots	

Figure 2.15: Fashion-MNIST dataset consists of 10 classes of clothing, such as shirts, pants, clothes, shoes, bags, etc. A matrix of pixel intensity values is used to represent each grey-scale image, which has a dimension of $28 \times 28 \times 1$ pixels. Source: [24]

2.4 Transformer Architecture

Transformers and a sequence-to-sequence architecture, described in the paper [25], are a neural networks called series-to-sequence (also known as Seq2Seq) translates

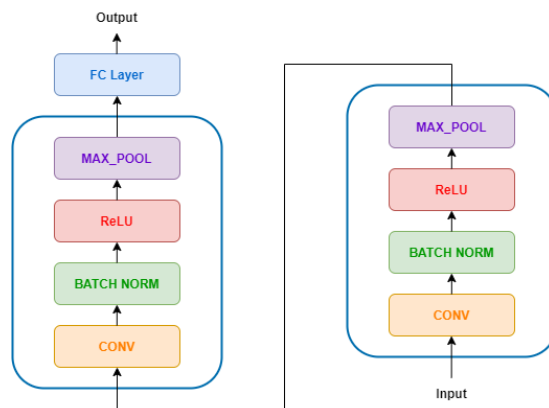


Figure 2.16: A sample CNN proposed by [23] to show how layers are connected each other in a simple CNN network. Two consecutive convolutional block each containing convolution operation followed by batch normalization, ReLU activation function and max pooling layer. At the end fully connected layer is employed to extract the predictions.

one series of components, such as the words in a phrase, into another sequence. Long-Short-Term-Memory (LSTM)-based models are a common option for this kind of model. The Encoder and Decoder in Seq2Seq models take the input sequence and map it into an n-dimensional vector in a higher level space. It is converted into an output sequence by the decoder, which may be in another language, symbols, a duplicate of the input, etc.

The Encoder translates a German sentence into another language it is familiar with, notably the imagined language, in order to translate German into French. The model is trained on a large number of instances in order to learn the made-up language. A single LSTM for each of the encoder and decoder functions in the Seq2Seq model is a straightforward option. The attention-mechanism is a technical feature that analyzes an input sequence and determines whether other sequence elements are critical at each stage. A human encoder and decoder is an example of this, where the encoder records keywords that are crucial to the meaning of the phrase and provides them to the decoder along with the standard translation. The attention mechanism considers many inputs simultaneously for each input that the LSTM (Encoder) reads and determines which inputs are significant by assigning various weights. The attention mechanism considers many inputs simultaneously for each input that the LSTM (Encoder) reads and determines which inputs are significant by assigning various weights to those inputs. The encoded message and the weights produced by the attention mechanism will subsequently be entered into the decoder.

Transformer is a revolutionary architecture that is introduced in the same paper

[25]. It employs the attention-mechanism we already observed, as the title suggests. Transformer is an architecture for transforming one sequence into another with the aid of two components (Encoder and Decoder), similar to LSTM, but different from the previously described/existing sequence-to-sequence models since it excludes any recurrent networks (GRU, LSTM, etc.).

Up until recently, recurrent networks were one of the strongest tools for capturing the temporal connections in sequences. However, the team which presented the article demonstrated that an architecture using simply attention processes and no RNN (Recurrent Neural Networks) may enhance performance in tasks like translation.

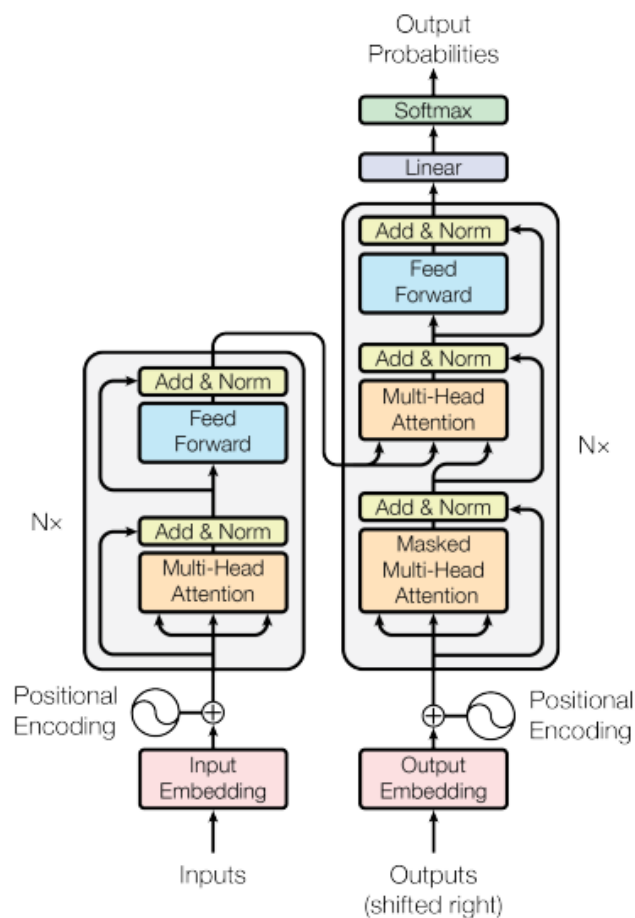


Figure 2.17: Transformer Architecture proposed in [25]. Source: [25]

In **Figure 2.17** is shown on the left is the encoder, and on the right is the decoder. According to $N \times$ in the image, Encoder and Decoder are both made up

of modules that may be stacked on top of one another several times. We can observe that Multi-Head Attention and Feed Forward layers make up the majority of the modules. Since we can't utilize strings directly, the inputs and outputs (target phrases) are first embedded into an n-dimensional space. The positional encoding of the various words is a small but significant component of the model. Since a sequence depends on the order of its constituents, we need to somehow assign every word or component in our sequence a relative position since we do not have recurrent networks that can recall how sequences are fed into a model. The embedded representation (n-dimensional vector) of each word is expanded to include these locations.

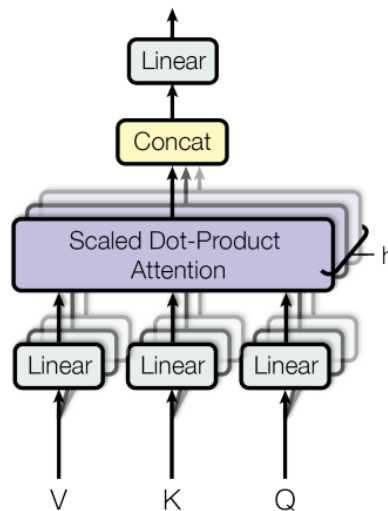


Figure 2.18: Multi-Head Attention of Transformer in [25]. Source: [25]

The attention is the key element of Transformer architecture, taking a look on Multi-Head Attention in **Figure 2.18**, let's define Q is a matrix which includes the query (vector representation of one word in the sequence), K are all the keys (vector representations of all the words in the sequence) and V are the values, which are similarly the vector representations of all the words in the sequence. For the encoder and decoder, multi-head attention modules, V comprises of the same word sequence than Q . However, for the attention module that has taken into account the encoder and the decoder sequences, V is distinct from the sequence represented by Q [26]. There are several parallelizations of this attention mechanism that may be utilized simultaneously. The linear projections of Q , K , and V are used to repeatedly repeat the attention process. As a result, the system can benefit from learning from various Q , K , and V representations. By multiplying Q , K , and V by weight matrices W that are acquired during training, these linear representations are created. Depending on whether the attention modules are in the encoder, decoder,

or anywhere in between the encoder and decoder, the matrices Q , K , and V are different for each position of the attention modules in the structure. We wish to focus on either the entire encoder input sequence or a specific section of the decoder input sequence, which is the main driver. The encoder and decoder are connected by a multi-head attention module, which makes sure that the input sequences from both are considered up to a certain location.

2.5 Feature Extraction

Feature extraction is an important step in which input is represented by a more meaningful representation for the network. In audio classification, features are extracted by means of several techniques. In [27] is investigated multiple time-frequency representations for environmental sound classification to determine which forms are able to better improve the classification performance of a CNN and the authors demonstrate that Mel-Spectrogram usage as feature extraction gives good performance. Since all the models under evaluation employ this method, it will be discussed later.

2.5.1 Mel Spectrogram

A change in a certain quantity over time is referred to as a signal. Air pressure is the variable quantity for audio. The air pressure may be measured over time with samples. We sample the data at various rates, but most frequently at a rate of 44.1 Hz for audio signals, or 44,100 samples per second. Numerous sound waves with a single frequency compose an audio signal. We merely record the resultant amplitudes while sampling the signal over time. A mathematical procedure called the Fourier transform enables us to break down a signal into its constituent frequencies and their amplitudes. It transforms the signal from the time domain to the frequency domain, whose outcome is known as a spectrum. This is conceivable because, as Fourier's theorem asserts, any signal may be decomposed into a series of sine and cosine waves that sum up to the original signal, and the Fast Fourier Transform (FFT) is a technique that can efficiently perform the Fourier transform. In case a non-periodic signal is elaborated, the short-time Fourier Transform (STFT) allows to compute FFT on overlapping windowed segments of the signal (see **Figure 2.19**), getting a spectrogram [28].

Since digital audio is a discrete signal, let's apply discrete-time STFT, where data is split into overlapped chunks or frames. For each frame, a Fourier transform is applied, and the complex result is added to a matrix that records magnitude and phase for each point in time and frequency as described in (2.14), where $x[n]$ denotes the signal, $\omega[n]$ the window, H the hop-size, $m \cdot H$ the starting sample of

the current frame and m the signal windowed and m the windowed segments

$$S(m, k) = \sum_{n=0}^{N-1} x(n + m \cdot H) \omega(n) e^{-\frac{i2\pi nk}{N}}. \quad (2.14)$$

The STFT results a spectral matrix composed by complex coefficients with dimension $(\#frequency_bins, \#frames)$ where $\#frequency_bins = \frac{frame_size}{2} + 1$ and $\#frames = \frac{samples - frame_size}{hop_size} + 1$ [30].

Regarding the window function, the most used one due to its main advantage of controlling the leakage [31], is Hann Window defined in (2.15) [32] and shown in **Figure 2.20**

$$\omega(k) = 0.5 \left(1 - \cos \frac{2\pi n}{N-1} \right). \quad (2.15)$$

After getting the matrix of windowed FFT, the spectrogram is outlined as a square module of the STFT matrix. To get Mel-Spectrogram, after getting the short-time Fourier Transform for each window, frequencies are converted to the mel scale. The need for building a Mel-spectrogram arises from the fact that humans perceive frequency logarithmically. A perceptual scale of pitches that humans perceive to be equally spaced from one another is known as the mel scale. By giving a tone at 1,000 Hz, 40 dB over the listener's threshold, with a perceived pitch of 1,000 mels, the starting point between this scale and conventional frequency measurement is established. Listeners perceive increasingly longer pauses to create equivalent pitch increments over 500 Hz [34].

Douglas O'Shaughnessy's [35] formulation, which represent an approximate average of common perception, converts from hertz scale to mel scale is defined in (2.16) and respective plot in **Figure 2.21**

$$m = 2595 \log \left(1 + \frac{f}{700} \right). \quad (2.16)$$

So, in order to get a mel-spectrogram, STFT is performed first, then amplitude is converted to dB scale and frequencies to mel scale [36].

The conversion of frequencies into mel-scale is carried out by first selecting the number of frequency bands. Next, the construction of the mel filter banks occurs by converting the highest and lowest frequency with the formula into (2.16), creating a number of bands equally spaced between the extremes of the converted frequencies. Each band of the mel frequency is then re-converted to hertz rounding to the nearest bin frequencies thus eventually creating a series of triangular filters as in **Figure 2.22** [37].

It is also widely used the log mel spectrogram a variation of what previously explained. while a mel spectrogram represents the frequencies in the mel scale, a log mel spectrogram is obtained by taking the logarithm of the mel spectrogram values. The logarithm transformation is commonly used to compress the dynamic range of the spectrogram and emphasize lower intensity sounds.

2.6 Data Augmentation

Due to the improper quantity of data used to train the network, adequate results are not obtained even after applying an appropriate model. A huge dataset is necessary for the deep learning model to function well. However, a sufficient volume and variety of data could not be reached to train the model, for this reason, data augmentation was introduced.

Data augmentation is a method for faking fresh training data out of old training data. To do this, domain-specific approaches are applied to instances from the training data to produce brand-new and distinctive training examples.

For image classification, for example, the data augmentation consists of making changes to them by rotating, scaling, cropping, and flipping [39].

The data augmentation methods employed in the test models, such as Mixup and Spectrogram Masking/Spec-Augmentation, will be discussed as this work has applications in audio classifications.

2.6.1 Spectrogram Masking/Spec-Augmentation

[40] suggests an augmentation technique that works on the input audio's log mel spectrogram rather than raw audio itself. This approach directly affects the log mel spectrogram of the input audio, making it easy to use and computationally inexpensive. Time warping and time and frequency masking are two of the three types of deformations of the log mel spectrogram that make up this phenomenon. This method is surprisingly efficient in automatic speech recognition and enables the authors to train end-to-end ASR networks, called Listen Attend and Spell (LAS) [41] to outperform and achieve state-of-the-art results without more complex system usage.

The authors seek to develop an augmentation strategy that works on the log mel spectrogram directly, which helps the network acquire relevant features. Motivated by the idea that these traits should be resilient to deformations in the time direction, partial loss of frequency information and partial loss of tiny portions of speech, they have selected the following deformations.

The first deformation is time warping (see **Figure 2.23**) implemented by means of the native tensorflow function `sparse_image_warp`. Given a log mel spectrogram with τ time steps, it is interpreted as an image where the time axis is horizontal and the frequency axis is vertical. A random point along the horizontal line crossing through the center of the image within the time steps $(W, \tau - W)$ is to be warped either to the left or right by a distance w selected from a uniform distribution from 0 to the time warp parameter W along that line. In other words, W represents the maximum distance by which a point can be warped. The uniform distribution is used to randomly select a distance value w within the range of 0 to W end this

ensures that all possible distances within the range have an equal chance of being selected.

The second deformation proposed by researchers is frequency masking (**Figure 2.24**) which aims to mask f consecutive mel frequencies in the range $[f_0, f_0 + f)$. f is a parameter selected from a uniform distribution from 0 to frequency mask parameter F and f_0 is picked within the range $[0, \nu - f)$ where ν denotes the number of mel frequency channels.

The last deformation provided by writes is time masking (**Figure 2.25**), quite similar to frequency one in which masking is employed to t consecutive time steps in the range $[t_0, t_0 + t)$ likewise t is a parameter picked from an uniform distribution from 0 to time mask parameter T and t_0 is chosen from a range $[0, \tau - t)$.

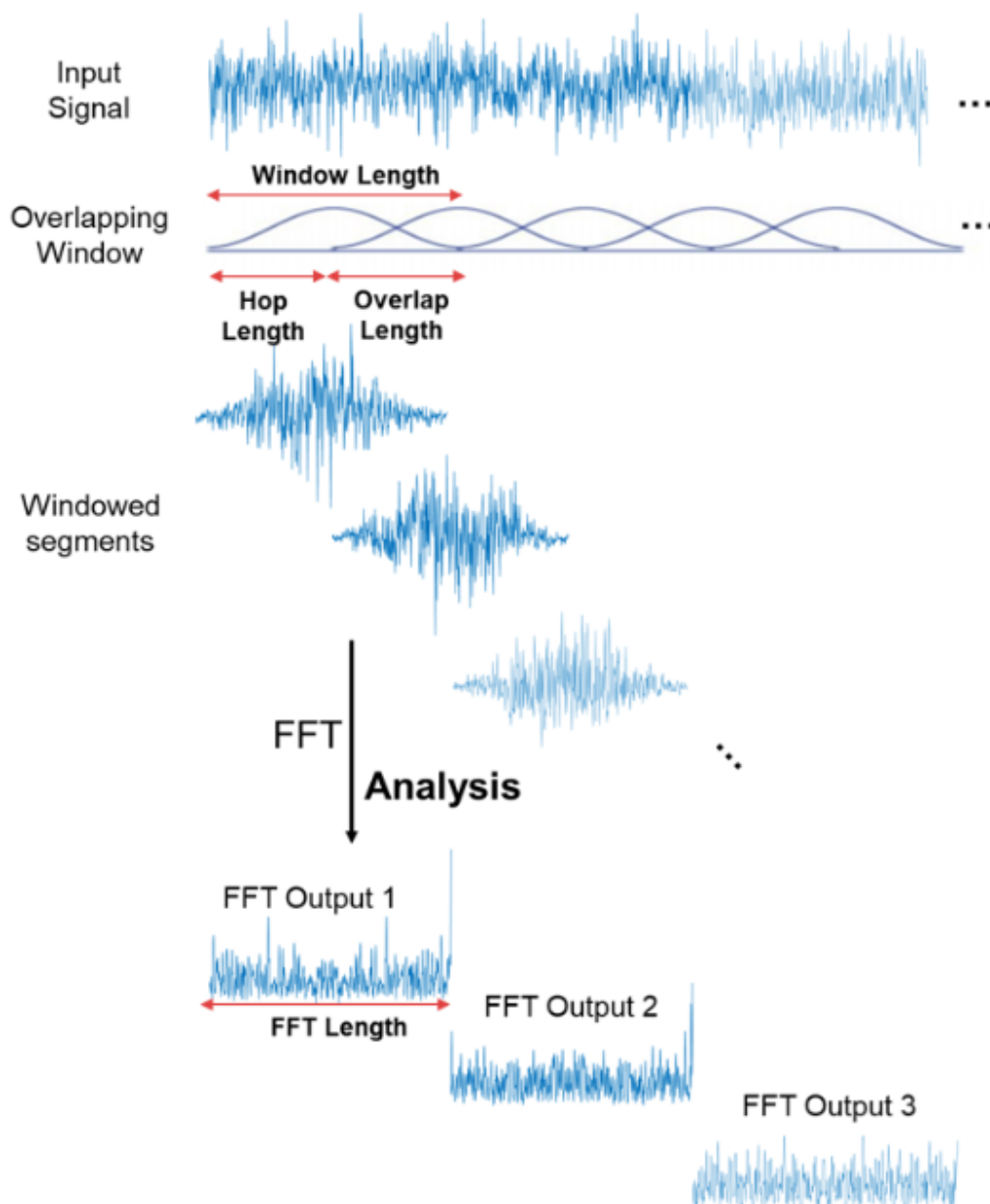


Figure 2.19: Illustration of how short-time Fast Fourier Transform works. The signal is first multiplied by overlapping window producing consecutive windowed segments. Then FFT is applied to get spectrogram. Source: [29]

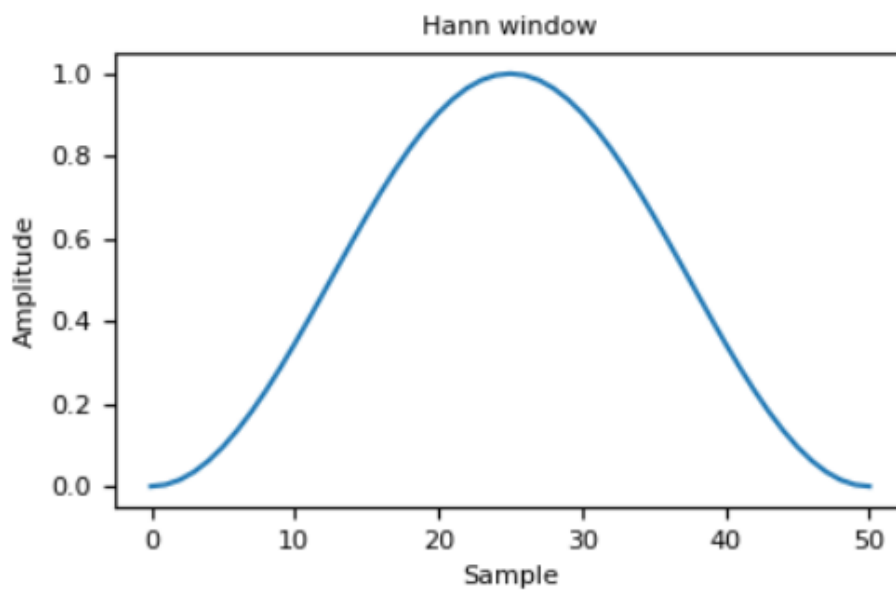


Figure 2.20: Plot of Hann Window with 50 samples length. Source: [33]

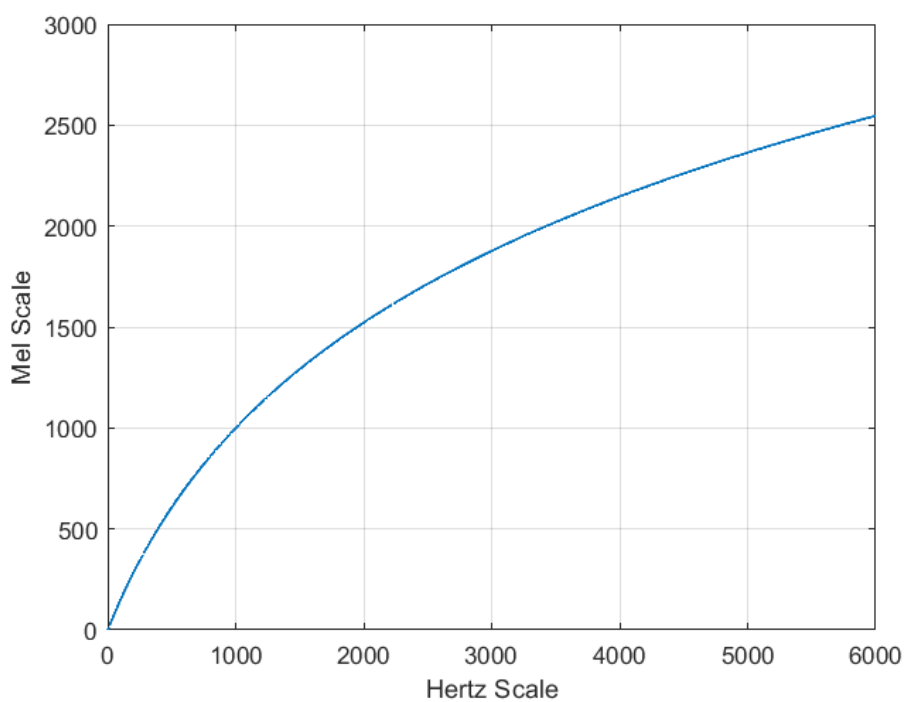


Figure 2.21: Conversion from Hertz scale to Mel scale. When frequency is 1,000 Hz it corresponds to 1,000 mel.

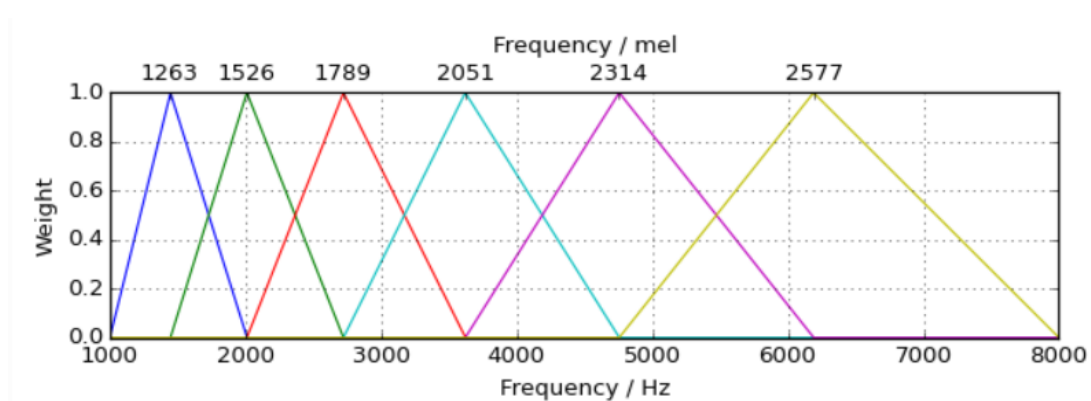


Figure 2.22: Mel Filter Banks is a filter bank with triangular shaped bands arranged on the mel frequency scale. This is used to convert a spectrogram from Hertz scale to Mel scale getting a Mel-Spectrogram. Source: [38]

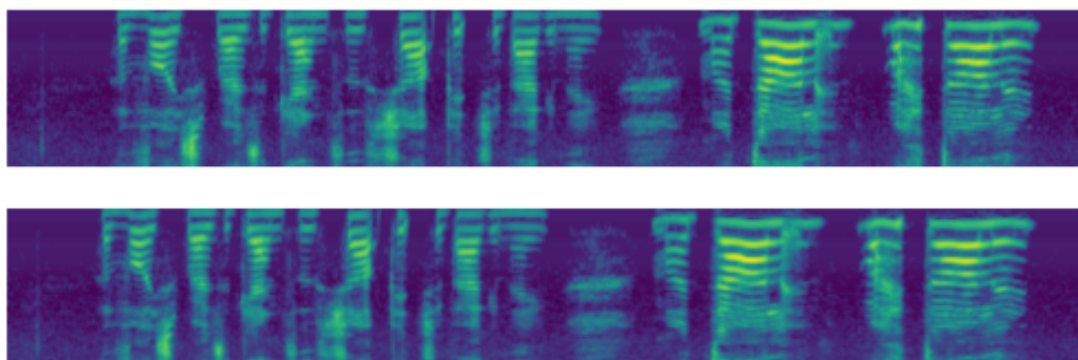


Figure 2.23: The image above is the original Mel-Spectrogram, the one below instead is the Mel-Spectrogram after Time Warp. Source: [40]

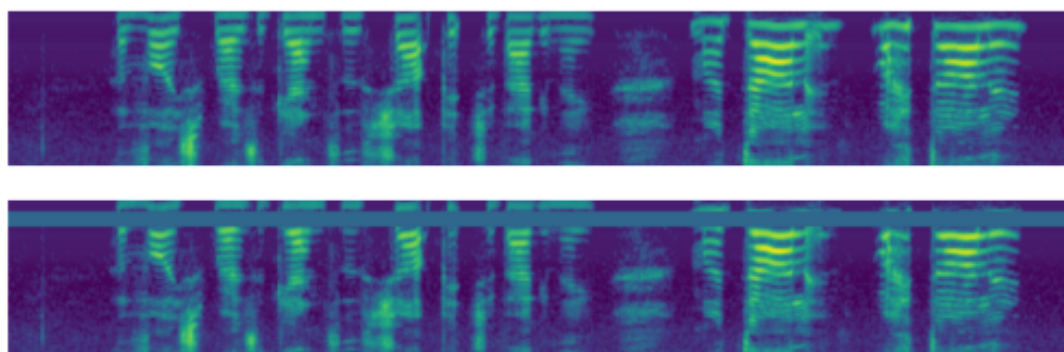


Figure 2.24: The image above is the original Mel-Spectrogram, the one below instead is the Mel-Spectrogram after Frequency Masking. Source: [40]

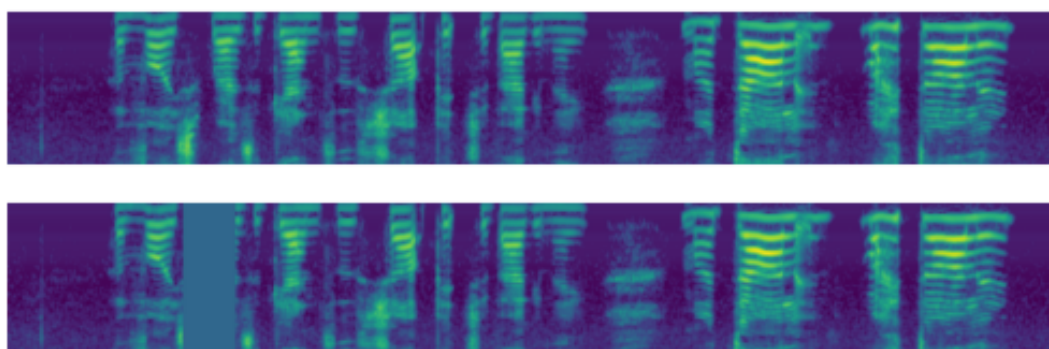


Figure 2.25: The image above is the original Mel-Spectrogram, the one below instead is the Mel-Spectrogram after Time Masking. Source: [40]

Chapter 3

Models Architecture and Dataset

3.1 Dataset

The thesis aims to emphasize the recognition of human activities in domestic environments and based on this, the search for an existing dataset was carried out on the Internet. The site [42] proposes several open-source audio datasets, but it was chosen as the reference one the ESC-50: Environmental Sound Classification because it contains categories that can group human and domestic sounds.

3.1.1 ESC50

The ESC-50 [43] dataset is a compilation of 50 classes of different environmental sounds. Sound snippets created from recordings made publicly accessible through the Freesound ¹ initiative make up each dataset. The classes in the labeled portion of the dataset were chosen at random with the intention of preserving balance between the main categories of sound events. The created classes were searched for in the Freesound field recording database, and then 5-second recordings of audio events were extracted from annotated fragments containing events from that class. The collected samples were then re-formatted to a single channel, 44.1 kHz format with 192 kbit/s Ogg Vorbis compression. Therefore, the labeled datasets were organized into 5 cross-validation folds of uniformly sized, guaranteeing that clips coming from the same original source file are always contained in a single fold. The ESC-50 dataset [44] includes 2,000 environmental recordings that have been

¹<https://freesound.org/>

categorized and are evenly distributed across 50 classes (40 clips per class). They are divided into 10 groups in each of 5 ill-defined primary categories for convenience: Natural soundscapes and water sounds, animal sounds, human (non-speech) sounds, interior/domestic sounds, and exterior/urban noises are all examples of audible stimuli. When feasible, the extraction procedure aimed to retain exposed sound events in the foreground with little background noise. Field recordings aren't sterile, though, so some movies could still have audible blending in the background. The dataset exposes users to a range of sound sources, some of which are quite frequent (laughing, cat meowing, dog barking), others of which are fairly unique (glass breaking, brushing teeth), and yet others of which have more subtle variations (helicopter and aircraft sounds). The few clips that are accessible for each class is one potential flaw in this collection. This is because human annotation and extraction are expensive and because tight class balance was maintained despite the scarcity of recordings for more unusual forms of sound occurrences.

Animals	Natural soundscapes & water sounds	Human, non-speech sounds	Interior/domestic sounds	Exterior/urban noises
Dog	Rain	Crying baby	Door knock	Helicopter
Rooster	Sea waves	Sneezing	Mouse click	Chainsaw
Pig	Crackling fire	Clapping	Keyboard typing	Siren
Cow	Crickets	Breathing	Door, wood creaks	Car horn
Frog	Chirping birds	Coughing	Can opening	Engine
Cat	Water drops	Footsteps	Washing machine	Train
Hen	Wind	Laughing	Vacuum cleaner	Church bells
Insects (flying)	Pouring water	Brushing teeth	Clock alarm	Airplane
Sheep	Toilet flush	Snoring	Clock tick	Fireworks
Crow	Thunderstorm	Drinking, sipping	Glass breaking	Hand saw

Figure 3.1: Sound classes of ESC-50 dataset. Source: [44].

Only 19 of these categories have been chosen for the thesis due to human action detection. **Figure 3.1** lists the ESC-50 classes. In order to choose just noises produced by humans or as a result of their activities, categories are minimized as **Figure 3.2** reports. It should be emphasized that both the categories of noises produced by the person, crying, sneezing, coughing, breathing, etc. and the categories of sounds produced by his activity typing, operating a washing machine, mouse, or using a vacuum cleaner have been chosen. Despite coming from various contexts, they are all about having people in a space.

Animals	Natural soundscapes & water sounds	Human, non-speech sounds	Interior/domestic sounds	Exterior/urban noises
Dog	Rain	Crying baby	Door knock	Helicopter
Rooster	Sea waves	Sneezing	Mouse click	Chainsaw
Pig	Crackling fire	Clapping	Keyboard typing	Siren
Cow	Crickets	Breathing	Door, wood creaks	Car horn
Frog	Chirping birds	Coughing	Can opening	Engine
Cat	Water drops	Footsteps	Washing machine	Train
Hen	Wind	Laughing	Vacuum cleaner	Church bells
Insects (flying)	Pouring water	Brushing teeth	Clock alarm	Airplane
Sheep	Toilet flush	Snoring	Clock tick	Fireworks
Crow	Thunderstorm	Drinking, sipping	Glass breaking	Hand saw

Figure 3.2: The 19 categories selected from ESC-50 for the experiment.

3.1.2 AudioSet

[45] describes how AudioSet was developed as a dataset and ontology of audio events to give thorough coverage of actual sound. Their goal is to develop automatic auditory event identification systems that are on par with the cutting edge when it comes to identifying things in real-world photos in an ImageNet-like [46] fashion. The goal of the research is to create more effective learning approaches by taking into account all sound occurrences as opposed to a small area.

In order to build a system that can anticipate human labelling from audio, the study provides a collection of classes to gather human-labeled data.

When the classifier finds ambiguity among numerous subcategories during recognition, hierarchical relations allow backing off to more generic descriptions (for instance, a sound that is ambiguously identified as a "growl," "bark," or "howl" might revert to a category of "dog sounds").

For the purpose of describing audio events found in authentic recordings, the Audio Set Ontology is an organised collection of audio event types. It adheres to the tenets of offering a thorough set that can be used to describe the audio events encountered in real-world recordings, correlating to the thought or comprehension that pops into a listener's head as soon as they hear the sound, and being distinguishable by a "usual" listener. The hierarchical structure enables annotators to quickly choose the best, most precise categories for given audio occurrences. Individual categories should be distinguishable based on their sound alone.

[45] used a modified version of the "Hearst patterns" [47] to find hyponyms of "sound" in order to seed the audio event lexicon and prevent the category from being biased by the orientation of a certain researcher. When these principles are applied to text at the size of a website, a very large number of words are produced,

which are then ranked according to how effectively they reflect sounds.

The hierarchy was manually put by researchers together from the top of the sorted list, and it was made more precise by comparing it to other taxonomies or lists of audio events. Some categories appear more than once, and the original structure is not strictly hierarchical because nodes might exist many times. By comparing the structure to existing audio event lists or taxonomies, the structure was gradually improved to the point where practically every class from other sets was covered. Because they were too specialised or otherwise did not match our requirements for being easily recognisable, certain classes from other sets were excluded.

Although "Car radio" is listed as a source of recorded music in the urban sounds taxonomy of [12], it is too specialised or context-dependent for the set. There are 165 verified-distinct sound clips used in [21], with examples like "Trumpet jazz solo" and "Walking on leaves" providing further information. The category set has now undergone additional changes in response to input from the larger group of participants, and it currently has 632 audio event categories that are organised in a hierarchy with a maximum depth of 6 levels. "Sounds of things" "Vehicle" "Motor vehicle" "Emergency vehicle" "Siren" "Ambulance (siren)" are a few examples from the eight level-6 nodes. This dataset will not be used to fine-tune the models; it has been explained since two of them are pre-trained with it.

3.2 State-of-art Architecture

The choice of models was made according the need to consider different architectures between them. The first model found is TFNet [49], chosen because it has publications and codes made public on GitHub and is therefore easily evaluable and testable. The second model is CNN10 [50], selected because it is cited and taken as a model of comparison in TFNet. It was also made possible by the development of the network in the source codes of the authors of TFNet. The last two models [51][52] were decided, however, by consulting the site PapersWithCode [53]. This site a community-driven platform for learning about state-of-the-art research papers on machine learning. It contains articles and codes for a wide range of tasks and datasets. On the site, the templates were selected using filters by mode (audio), by task (audio classification), and by dataset (ESC-50). After returning the search result, the View filter is set to Accuracy (5-folds), i.e., the average accuracy achieved on 5 folders of the ESC-50 dataset. Of the three best models, the second and the third were chosen. The first one has been excluded due to the lack of complete official codes published by the researchers.

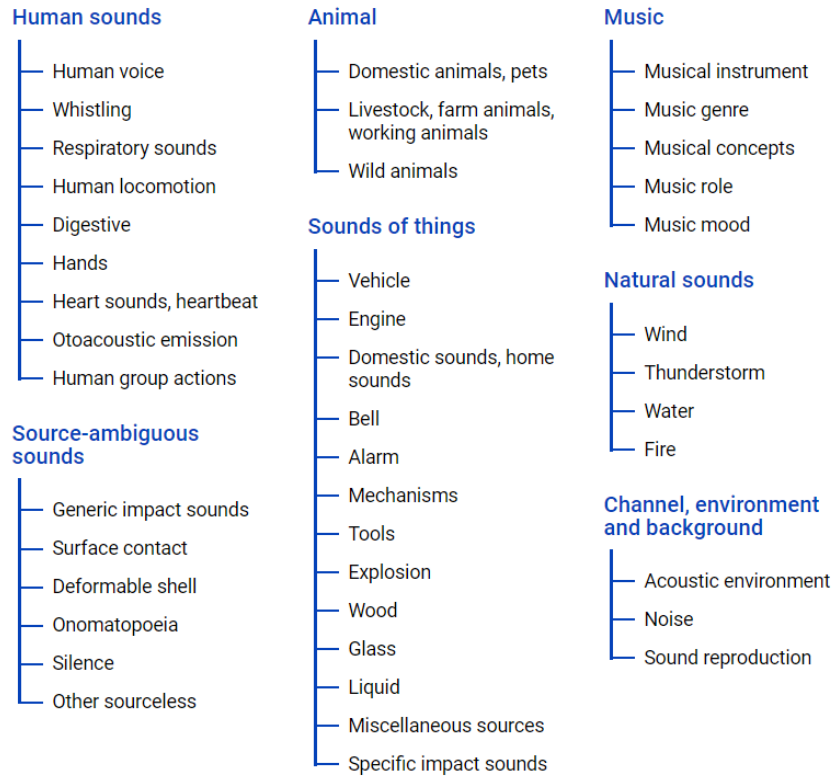


Figure 3.3: The top two layers of Audioset Ontology. For example "Human voice" covers the classes Speech, Shout, Screaming, Whispering, Laughter, Crying, Wail, Sigh, Humming, Groan, Grunt and Yawn. Source: [48]

3.2.1 CNN10

The model network proposed by PANNs [50] is composed of four convolutional blocks, each made up of two convolutional layers, batch normalization between them, ReLU nonlinearity to make the training faster, and at the end for down-sampling, average pooling is inserted into each convolutional block. Next to the last convolutional layer, global pooling is used to sum up the feature maps into a vector, and a fully connected layer is added just after for the purpose of extracting embedding features and improving the representation ability. A linear classifier with a softmax nonlinearity function is included for classification. The original CNN10 model in PANNs has been trained with AudioSet [45] with a binary cross-entropy loss function. The CNN10 network of the experiment has been implemented in the official Github folder of TFNet authors [49][54] which has maintained the same structure of the model proposed by PANNs, but no pre-training on AudioSet is performed.

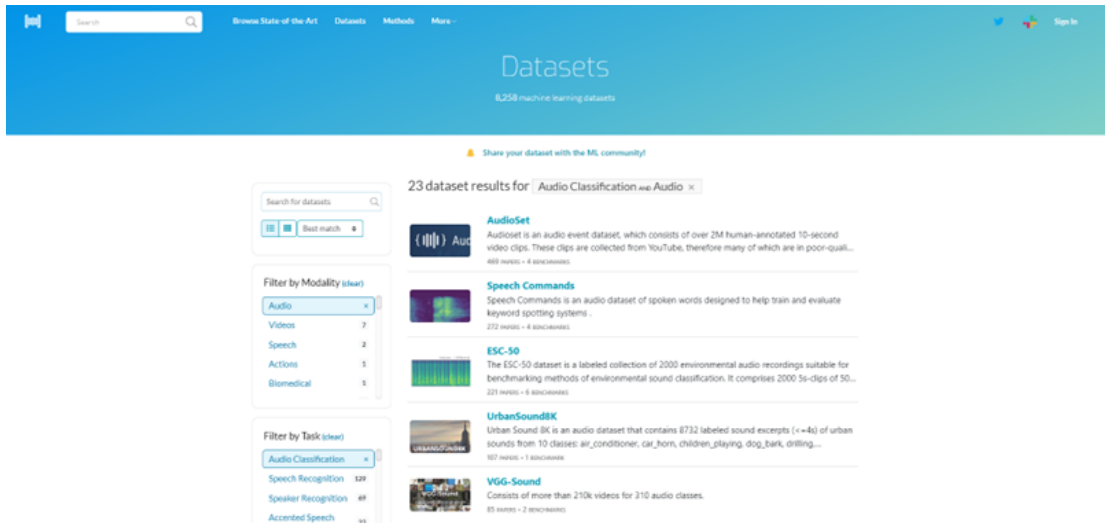


Figure 3.4: Paperswithcode screenshot in which filters by method, task and dataset are applied to find moodels. Source: [53]

Audio Classification on ESC-50

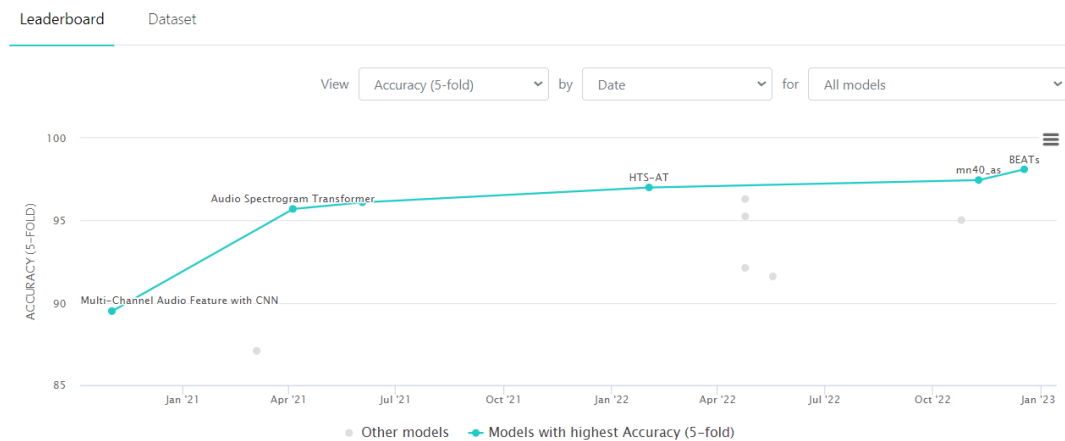


Figure 3.5: Audio Classification Accuracy trend on ESC-50 of SOTA models over the years. Source: [53]

3.2.2 TFNet

For CNN to learn discriminative sound representation, TFNet model [49] suggests a parallel temporal-spectral attention mechanism. This mechanism strengthens the temporal and spectral characteristics by recognizing the significance of various time frames and frequency bands.

Rank	Model	Top-1 Accuracy	PRE-TRAINING DATASET	Accuracy (5-fold)	Extra Training Data	Paper	Code	Result	Year	Tags
1	BEATs	98.1	AudioSet	98.1	✓	BEATs: Audio Pre-Training with Acoustic Tokenizers			2022	Self-Supervised Learning
2	mn40_as	97.45	AudioSet	97.45	✓	Efficient Large-scale Audio Tagging via Transformer-to-CNN Knowledge Distillation			2022	CNN knowledge distillation
3	HTS-AT	97.0	AudioSet	97.0	✓	HTS-AT: A Hierarchical Token-Semantic Audio Transformer for Sound Classification and Detection			2022	

Figure 3.6: Audio Classification on ESC-50 rank of SOTA models. Source: [53]

The key-part of this model is the usage of both temporal and spectral attention mechanisms pictured in **Figures 3.7, 3.8** which improve the representation of features focusing on relevant time frames and frequency bands by giving different weights to them. Attention mechanism consists of convolutional layers used for achieving channel-wide global feature maps followed by filters to reduce the number of channels to one to ensure the network can acquire channel-wise global feature map \mathbf{V}_T and \mathbf{V}_F from local feature map \mathbf{U} . After that, Global Average Pooling, applied to time axis and frequency axis according to temporal or spectral attention, allows to yielding activations \mathbf{v}_T and \mathbf{v}_F by compressing global feature maps. Finally, the news timewise and frequency-wise feature map \mathbf{U}_T and \mathbf{U}_F respectively, are reached by multiply starting feature map and involved activations vector.

Since temporal and spectral features would be concurrently captured, a summation among timewise feature map, frequency-wise feature map and starting feature map is performed with different weights in such a way the network can gives separate attention to each as reported in **Figure 3.9**. So, the final time-frequency feature map of the TF-Block is achieved

$$\mathbf{U}' = \alpha\mathbf{U}_T + \beta\mathbf{U}_F + \gamma\mathbf{U}, \alpha + \beta + \gamma = 1 \tag{3.1}$$

where α, β, γ are the learnable weights of each addition branch which initially start at the same value and during the training, the network learns to give appropriate weight to time feature-map rather than frequency one.

In **Figure 3.10** is shown the overall architecture of TFNet composed by four Temporal-Frequency block which takes inspiration from original baseline model CNN10 [50] where four convolutional block has been replaced by TF-Block. Each TF-Block employs the temporal-frequency attention mechanism that has been explained above, illustrated in the bottom part of the **Figure 3.10**.

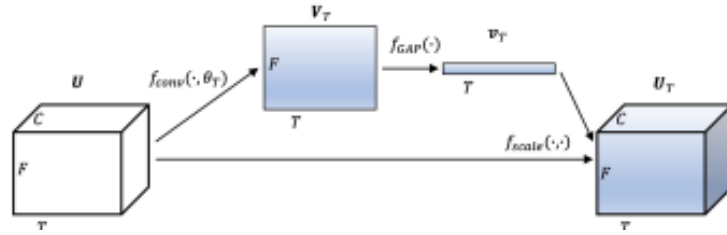


Figure 3.7: Temporal attention allows the network to identify the most relevant frames in order to recognize where the sound occurs. Source: [49].

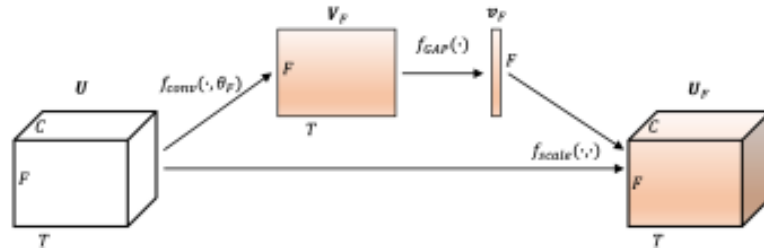


Figure 3.8: Spectral attention attention is employed to give different level of attention to different frequency bands which are significant for the task. Source: [49].

3.2.3 mn40_as

The model [55] is a composition of CNN and Transformer. Each architecture has its advantages: CNN can learn complex tasks with limited data, while Transformers outperforms with extremely large datasets, even if this increases computational complexity due to a higher number of parameters. For the CNN component, an efficient network like MobileNets was chosen, while the Transformer component utilizes the Audio Spectrogram Transformer [56] (based on ViT [57]). The paper proposes a training procedure for the CNN component based on offline knowledge distillation (i.e., teacher model predictions are pre-generated and saved before training the student model). Architecture definition: The student model (CNN MobileNetsV3 [58]) is pretrained on ImageNet [46] while the teacher model, namely PaSST Transformer [59], is trained on AudioSet [45], and checkpoints are saved offline.

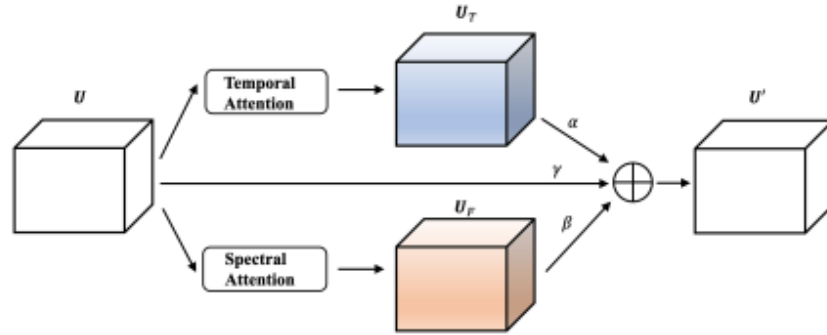


Figure 3.9: Parallel temporal-spectral attention mechanism applied to each convolutional block. Source: [49].

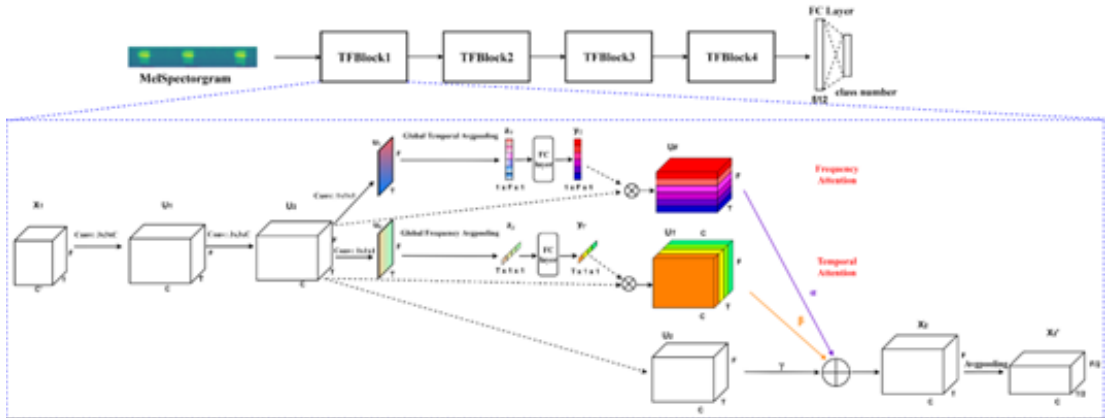


Figure 3.10: TFNet overall architecture. A zoom of what is inside a convolutional block. Source: [54]

Audio Spectrogram Transformer Introduction

AST [56] is based on transformer architecture, a type of model initially born for natural language processing. It is very similar to network already existing such as ViT [57] transformer mainly specialized to image recognition. **Figure 3.12** shows the architecture of Audio Spectrogram Transformer and the audio elaboration flow starting with Mel-spectrogram (128 bank filters calculated with Hamming window of 25 ms every 10 ms) divided in N patches of dimension 16×16 with overlap of 6 on both dimensions. Each patch of the spectrogram is flattened by means of linear projection first, then trainable positional embeddings are inserted in such a way that spatial order of patch is given. Next patches go as input to transformer-encoder and a linear layer with sigmoid as activation function is applied to map the predicted label. ViT and AST are quite similar each other, but the main difference among

them is the fixed dimension of data elaboration for the first architecture against a more variable dimension of input spectrogram of the second one.

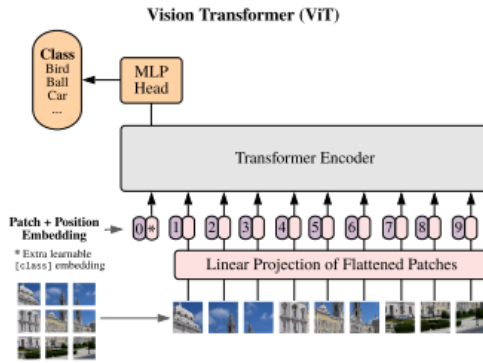


Figure 3.11: Vision Transformer (ViT) model overview: an image is split into fixed-size patches each of which is linearly embedded, position embeddings are added, and the resulting sequence of vectors is sent to a typical Transformer encoder. Then the standard approach of adding an extra learnable “classification token” to the sequence is employed. Source: [57]

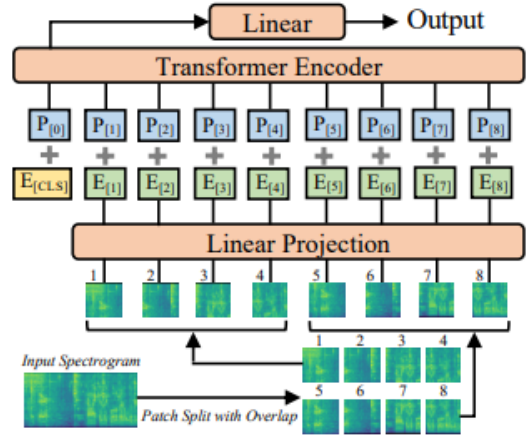


Figure 3.12: Audio Spectrogram Transformer (AST) architecture: the 1-D patch embeddings are created by linearly projecting the 2D audio spectrogram into a series of 16×16 patches with overlap. A learnable positional embedding is added to each patch embedding. The sequence has a further categorization token prepended to it. A Transformer receives the output embedding as input, and uses the classification token as its output to classify data with a linear layer. Source: [56]

Teacher Model: PaSST

Transformer used in mn40_as model is PaSST [59] very similar to ViT/AST architectures but with some modifications to reduce input sequence length.

Figure 3.13 shows the idea behind The Patchout faSt Spectrogram Transformer (PaSST). The starting architecture is based on ViT in which from input spectrogram, patches are extracted and added trainable positional encodings, then a modification using the technique of overlapping the patches introduced by [56] to improves the performance during the training. The main drawback of overlapping is the increasing of patches sequence and as results of memory and compute requirements

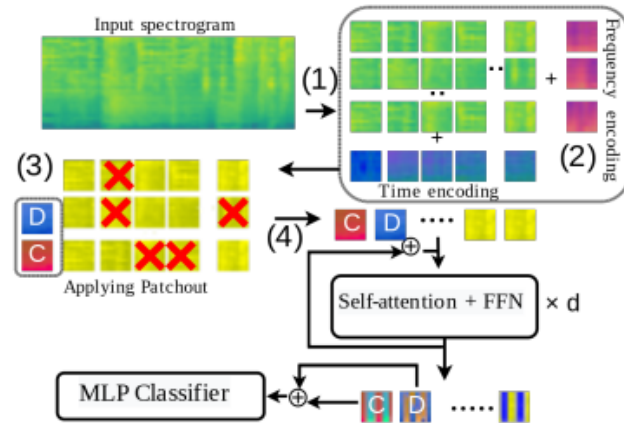


Figure 3.13: Patchout Transformer (PaSST) Architecture. Input spectrogram is divided into patches and then randomly discarded before applying self-attention. Source: [59]

rising encouraged to propose a method called Patchout to overcome these problems. Patches extraction from input spectrogram and linear projection as in ViT is performed in step (1), frequency and time positional encoding are inserted in step (2) to make simpler the inference and fine-tuning of pre-trained models with shorter audio length. In Step (3) patchout and classification token addition are applied. Patchout consists of randomly discarding part of patches sequence during the training to encouraging the transformer make classification even if an incomplete sequence is used reducing the length and so computation complexity. During the inference no patchout is performed instead. PaSST-U identifies unstructured patchout where patches selection is done in a random way without considering position. PaSST-S is based on SpecAugment [40] technique for data augmentation, where a frequency bin or time frame is chosen randomly and its whole row/column is removed. Step (4) flatten the sequence by passing it in d layers of self-attention blocks in which d detects the depth of the transformer. Then, as last step, the classifier to predict the input label. Since computational complexity on attention computing rises quadratically $\mathcal{O}(n^2)$ with sequence length n , it goes without saying that reducing the length allows to significantly reduce the complexity and memory requirements.

Student Model: MobileNets

As CNN student model, MobileNetV3-Large [58] is deployed (mn stands for MobileNet abbreviation) typically used for mobile phone CPUs application. MobileNetV3-Large is a variation of previous network version which can be more accurate on

ImageNet classification and faster when producing an output which is very important for real-time application, with respect to earlier version MobileNetV2 [60]. The main motivation which has driven the realization of MobileNetV3 is to adapt high accuracy with power consumption of mobile device. MobileNetV3 is a result to make more efficient the blocks of previous versions. MobileNetV1 [61] introduces the Depth-Wise Separable Convolutions, a more efficient convolution layer able to decompose the traditional convolution in two separated steps, the spatial filtering and feature generation. By looking at the **Figure 3.14** it is easy to notice how this structure allows to reduce the number of parameters of the convolution operation.

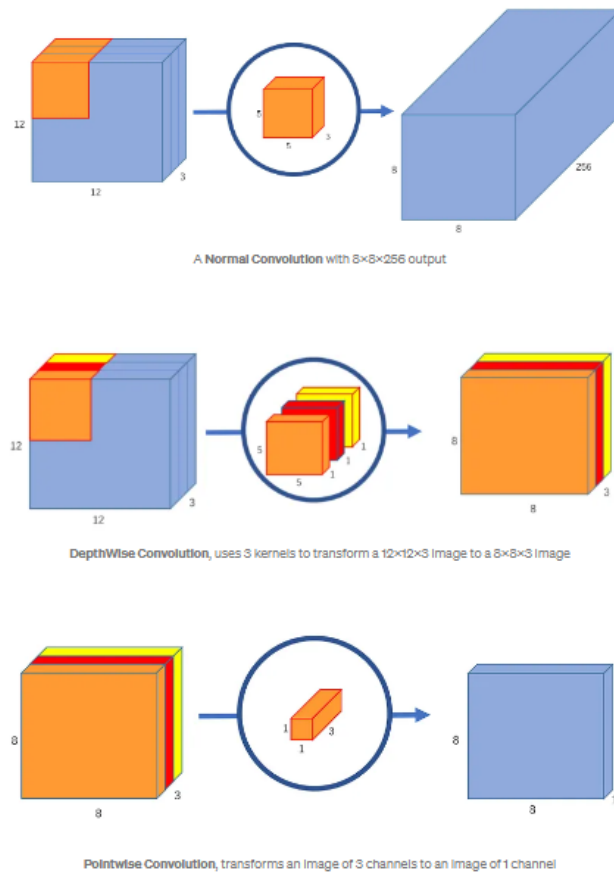


Figure 3.14: MobileNetV1 Depthwise Separable Convolution: is a form of decomposed convolutions that factorize a standard convolution into a depthwise convolution, which applies a single filter to each input channel, and a 1×1 convolution called a pointwise convolution that employs a 1×1 convolution to combine the outputs of the depthwise convolution. Source: [62]

MobileNetV2 uses linear bottleneck and inverted residual block to keep a narrow

representation while internal expansion to higher dimension improving features representation is performed. 1×1 pointwise convolution is applied to reduce the input channels and representing feature maps into a lower dimension. Depthwise separable convolution is performed by depthwise convolution to capture spatial information from channel independently as first then pointwise convolution is deployed to create a new feature map with higher representation. Residual means that original input is added to output of bottleneck to don't lose input information and encouraging the model to capture and propagate main information in the network.

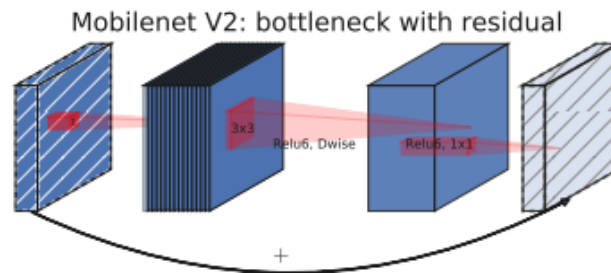


Figure 3.15: MobileNet V2 layer: Inverted Residual and Linear Bottleneck to keep narrow representation while internal expansion to improve features representation is performed. Pointwise convolution to reduce input channels, deptwise convolution to capture spatial information, pointwise convolution to create new feature map and residual in which original input added to output of bottleneck to propagate main information along the network. Source: [58]

MnasNet [51] implements a modified bottleneck block based on squeeze and excitation. Squeeze operation is done just after depthwise separable convolution, which involves global average pooling to reduce dimension of feature maps to a single value per channel and capturing channel information. Then excitation operation consists of passing squeeze output into two fully connected layers with a non-linear activation function for the first one to reduce feature dimension and second layer to expand the features to original channel dimension. After this block, a multiplication between original feature maps and SE (Squeeze-Excitation) output gives model the ability to concentrate only on relevant features.

MobileNetV3 uses the blocks described before to create a more efficient model it combines the modified bottleneck block of MnasNet (**Figure 3.16**) applying squeeze and excite in the residual layer, replacing ReLU6 activation function in bottleneck block in MobileNetV2 with a new one Hard-Swish a linear approximation of Swish non-linear activation function which improve accuracy and reduces the

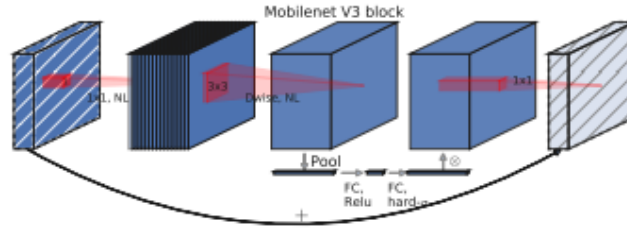


Figure 3.16: MobileNet V3 block: MobileNetV2 + Squeeze-and-Excite. Squeeze operation after depthwise convolution implemented by global average pooling to reduce dimension of feature maps to a single value per channel. This allows to capture channel information. Excitation operation consist of passing the squeeze output into two FC layers with non-linear activation functions. First one allows to reduce feature dimension. Second one is used to expand the features to original channel dimension. Source: [58]

number of memory accesses defined as follows [63]

$$h - swish[x] = x \frac{ReLU6(x + 3)}{6}. \quad (3.2)$$

Knowledge Distillation

Knowledge Distillation [64] is a technique used to transfer the knowledge from a higher complexity model to a lower complex one. In this case the more complex model, also called teacher, is the PaSST (a Transformer) and lower one denoted as student, is the MobileNetV3 (a CNN). The student will try to reduce standard classification loss, but also a distillation loss based on teacher’s predictions. Following equation shows the total loss the student will reduce is a weighted sum of classification loss L_1 given target label y and student prediction $\delta(z_s)$ and distillation loss L_{kd} between teacher prediction $\delta(z_t/\tau)$ and student prediction

$$Loss = \lambda L_1(\delta(z_s), y) + (1 - \lambda) L_{kd}(\delta(z_s), \delta(z_t/\tau)). \quad (3.3)$$

The KD is performed by researchers to transfer the knowledge during student training on AudioSet. As loss function, Binary-Cross-Entropy is used for both L_1 and L_{kd} furthermore Sigmoid activation function for δ has been chosen. According to mn40_as results in **Figure 3.17**, for KD the best performance is reached when $\lambda = 0.1$ and $\tau = 1$.

The powerful of knowledge distillation applied to MobileNetV3 is demonstrated by the following **Figure 3.18**: using KD where a PaSST transformer teach to a pre-trained MN improves the performance with respect to a single PaSST model for the same complexity in terms of number of parameters.

	$\lambda = 0.5$	$\lambda = 0.3$	$\lambda = 0.1$	$\lambda = 0.0$
$\tau = 1$.454	.460	.464	.454
$\tau = 3$.455	.459	.457	.454

Figure 3.17: mAP results obtained during student training on AudioSet using different values of λ and τ . Source: [55]

	# PARAMs	mAP
MN-Baseline	4.88M	.401
MN-Baseline pre-trained	4.88M	.417
MN-KD	4.88M	.458
MN-KD pre-trained	4.88M	.470

Figure 3.18: MobileNet performance on AudioSet with and without KD and pre-training on ImageNet. Source: [55]

3.2.4 HTS_AT

The Hierarchical Token-Semantic Audio Transformer [52], also known as HTS_AT, is a network specially designed to reduce the model size and training time thank to hierarchical structure usage.

Swin Transformer

Swin Transformer Block is based on Swin Transformer [65], a novel hierarchical vision Transformer architecture for computer vision tasks which handles the weakness of conventional Transformers in processing high-resolution picture inputs. It employs a patch-based self-attention method and separates the picture into smaller, non-overlapping patches. However, the Swin Transformer adopts a method know as Path-Merge to detect dependencies across patches rather than processing these patches directly. Patch merging and patch splitting are the two operations of the patch-merge procedure in the Swin Transformer. The patches are collected and pooled inside a local window during the patch merging process. The model could acquire contextual data inside each window according to this method. The aggregated features are next separated into non-overlapping patches during the patch splitting stage, however with a reduced spatial resolution. With less computational expense, this approach enables the model to analyse high-resolution pictures effectively. The patch-merge approach makes it possible to build a hierarchical design. Another key-approach employed in Swin Transformer are the shifted windows attention between consecutive self-attention layers which lets to capture long-range relationships and propagate information along neighbouring windows.

HTS_AT Architecture

The initial phase of the network is Mel-Spectrogram encoding in which spectrogram is first divided into windows and each window is split in patches with the order defined by indicators in such a way different frequencies bins at the same time frames are close to each other. Then patches are converted to patch tokens by means of Patch-Embed CNN with a kernel size of $(P \times P)$. The second step of the structure consists of putting patch tokens into several clusters of transformer-encoder blocks. Each one ends with a Patch-Merge layer, which minimizes the sequence size inside the group. The sequence is first reshaped to its original 2D map for this merging procedure $(\frac{T}{P} \times \frac{F}{P}, D)$, where D denotes the dimension of the latent state.

Next it combines neighbouring patches as $(\frac{T}{2P} \times \frac{F}{2P}, 4D)$ and a linear layer is applied to decrease the latent dimension to $(\frac{T}{2P} \times \frac{F}{2P}, 2D)$. It can be noticed that after four blocks the patch tokens' dimension is reduced by eight times with a starting dimension $(\frac{T}{P} \times \frac{F}{P}, D)$ till $(\frac{T}{8P} \times \frac{F}{8P}, 8D)$, which causes the GPU memory usage to decrease exponentially. A window attention technique is used for each transformer block inside the group to lessen the calculation. In Figure x, the patch tokens (in 2D format) is first divided into nonoverlapping $(M \times M)$ attention windows aw_1, aw_2, \dots, aw_k , as indicated by various colored boxes in the middle right corner.

The attention matrix is thus only calculated for each $(M \times M)$ attention window. As a result, rather of a complete Global Attention (GA) matrix, we have k Window Attention (WA) matrices which allows to significantly reduce the complexity as confirmed as follows

$$GA : \mathcal{O}(ftD^2 + (ft)^2D)$$

$$WA : \mathcal{O}ftD^2 + M^2ftD$$

Where a transformer block with $f \times t$ audio patch tokens with starting latent dimension D is considered. Window Attention enables to shorten the complexity of $\frac{ft}{M^2}$.

So, to implement the window attention, a Swin-transformer block with a shifted window attention is involved a more beneficial window attention mechanism which also allow to use the pretrained Swin Transformer.

Each token in the final layer output provides details about the time frames and frequency bins to which it belongs. In addition, this model allows to process dataset with strong labels by computing the loss in a given time window or weakly labelled because of to its robust relational data capture capabilities. Last stage of structure includes a Token Semantic CNN layer following the last transformer block to combine all frequency bins and map the channel size 8D into event classes C by means of a kernel size $(3, F \times 8P)$ and padding size $(1,0)$. The output $(\frac{T}{8P}, C)$ is seen as map of event present and performing an Average Pooling the outcome vector $(1, C)$ denotes the predicted label for consecutive time frames.

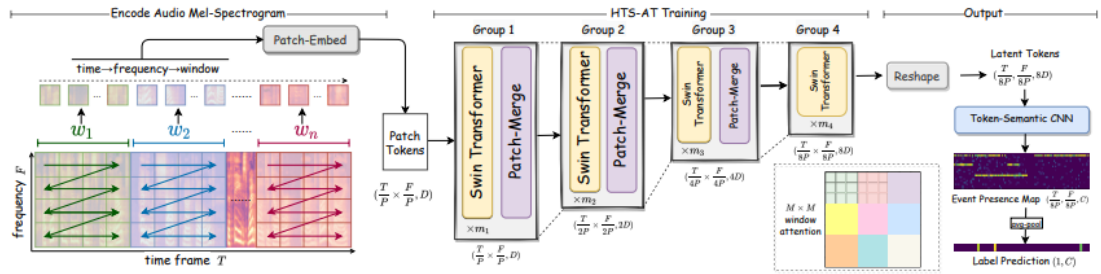


Figure 3.19: HTS-AT architecture: Encode Audio Mel-Spectrogram in which the spectrogram is divided into patch windows and then each window is split into patches. Each patch is a token. Then the Transformer-encoders capture long-term relationships and dependencies between sounds in the spectrogram. They use attention mechanism to assign weight to tokens. At the end, Token Semantic CNN extracts relevant semantic features from audio tokens. Source: [52]

Chapter 4

Methodology

4.1 Training

Models Review

The `mn40_as` model has different components which require a pre-training: student model (MobileNetV3), initially pre-trained on ImageNet, is trained by researchers on AudioSet by means of KD from teacher model (PaSST). In this experiment, however, pre-trained student model on AudioSet has been selected and executing only fine-tuning on ESC-50 reduced to 19 categories. Official Github repository [66] proposes different models based on width mult. All of them are MobileNet architecture pre-trained on ImageNet and then trained on AudioSet and are listed in **Figure 4.1** where Performance indicates the mean Average Performance reached with Audioset. **Figure 4.2** suggests that best model for ESC-50 is `mn40_as_ext` (**Figure 4.1**) so a MobileNetV3-Large with width multiplier equals to 4.0 trained on AudioSet with extended training of 300 epochs, will be deployed for fine-tuning the model.

HTS_AT also requires pre-training: the researches select pre-trained Swin-Transformer on ImageNet and train the entire structure of HTS_AT on AudioSet. Also in this case, in the experiment, the network has been selected already pre-trained on AudioSet and only fine-tuning on ESC-50 reduced to 19 categories has been performed. `mn40_as` and HTS_AT pre-trained on AudioSet are available on official GitHub folders of researches [66] [67]. TFNet and CNN10, on the other hand, do not require any pre-training step, so they have been trained directly on ESC-50 reduced to 19 categories. TFNet implementation codes have been taken from official GitHub folder of researches [54]. TFNet researchers, in their paper [49], take CNN10 as comparison model to highlight how their temporal-frequency attention mechanism insertion at the end of each convolutional block, helps the network to reach an higher accuracy on ESC-50 with respect to CNN10 one. Therefore, in

Model Name	Config	Params (Millions)	MACs (Billions)	Performance (mAP)
mn04_as	width_mult=0.4	0.983	0.11	.432
mn05_as	width_mult=0.5	1.43	0.16	.443
mn10_as	width_mult=1.0	4.88	0.54	.471
mn20_as	width_mult=2.0	17.91	2.06	.478
mn30_as	width_mult=3.0	39.09	4.55	.482
mn40_as	width_mult=4.0	68.43	8.03	.484
mn40_as_ext	width_mult=4.0, extended training (300 epochs)	68.43	8.03	.487
mn40_as_no_im_pre	width_mult=4.0, no ImageNet pre-training	68.43	8.03	.483
mn10_as_hop_15	width_mult=1.0	4.88	0.36	.463
mn10_as_hop_20	width_mult=1.0	4.88	0.27	.456
mn10_as_hop_25	width_mult=1.0	4.88	0.22	.447
mn10_as_mels_40	width_mult=1.0	4.88	0.21	.453
mn10_as_mels_64	width_mult=1.0	4.88	0.27	.461
mn10_as_mels_256	width_mult=1.0	4.88	1.08	.474
Ensemble	width_mult=4.0, 9 Models	615.87	72.27	.498

Figure 4.1: By default, all accessible models undergo pre-training on ImageNet (otherwise indicated as 'no_im_pre'), which is followed by training on AudioSet. Source: [66]

TFNet github folder the researchers have implemented CNN10 model as well and this experiment consequently uses CNN10 codes provided by [54].

Folds Split

ESC-50 dataset expects a total audio samples number of 2,000 that is 40 samples per class. It implements k-fold cross validation technique where k is 5. This suggests that all 2,000 audio are split into 5 folds where each fold contains 400 samples (8 per class). Most models which use this dataset, including the ones selected in this experiment, train the network 5 times by selecting in rotation a fold as test set and missing 4 folds as training set: referring to **Table 4.1**, when a model is trained on fold 1 means test set is fold 1 and training set are folds 2, 3, 4, 5. Then the second training on fold 2 selects fold 2 as test set and folds 1, 2, 4, 5 as training

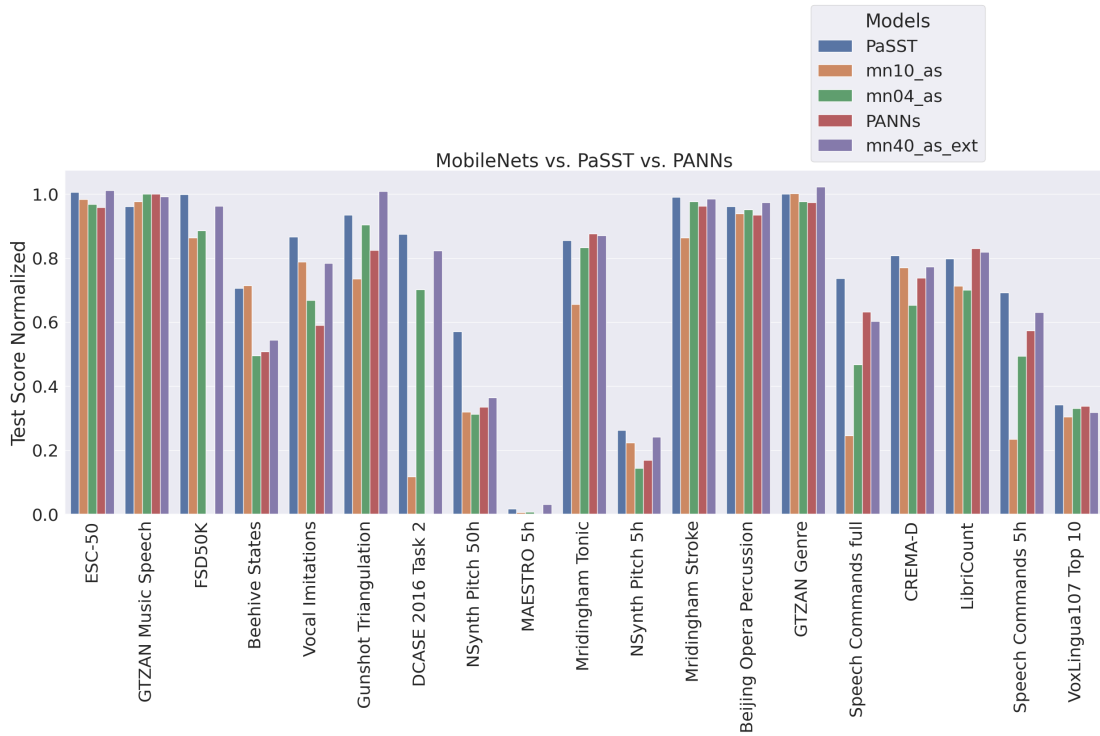


Figure 4.2: Plot contrasts each task’s results for the PANNs (CNN14), PaSST, and mn40_as models. The score is normalised using a model’s peak performance. Source: [66]

set and so on. In this experiment, where ESC-50 has been reduced to 19 categories instead of maintaining original 50 ones, folds split remains the same as explained before (**Table 4.1**), but now the total number of samples contained in each fold is decreased from 400 to 152. Also in this case the networks are trained 5 times by selecting in rotation a fold as test set and missing 4 folds as training set.

Table 4.1: How the five folders are split for training and test: when a training is performed on fold 1 it means folds 2,3,4,5 are used as training set and fold 1 is used as test set and so on.

Fold	Train	Test
1	2, 3, 4, 5	1
2	1, 3, 4, 5	2
3	1, 1, 4, 5	3
4	1, 2, 3, 5	4
5	1, 2, 3, 4	5

Feature Extraction

The input representation mode common to all models is the mel-spectrogram and main information regarding the pre-processing are collected in **Table 4.2**.

Table 4.2: Features extraction details about reproduction sound, sampling frequency, window size, hop size and number of mel bins for each model

	Model			
	mn40_as	HTS_AT	TFNet	CNN10
Mono/Stereo	Mono	Mono	Mono	Mono
Sampling Frequency (Samples/s)	32000	32000	44100	44100
Window Size	1024	1024	1764	1764
Hop Size	320	320	882	882
Number of mel bins	128	64	40	40

Data Augmentation

CNN10 and TFNet employ Time-Frequency Masking techniques during the training. For the models mn40_as and HTS_AT, on the other hand, the data augmentation has been applied only during the pre-training not during the fine-tuning because the application of data augmentation can increase the complexity of the model and increase the risk of overfitting especially if the target dataset is relatively small and therefore could make the models not able to generate. The related techniques used are collected in **Table 4.3**.

Table 4.3: Data augmentation techniques used in pre-training and training

Model	Data Augmentation	
	Pre-training	Training
CNN10	Pre-training unexpected	Time-Frequency Masking
TFNet	Pre-training unexpected	Time-Frequency Masking
mn40_as	Mixup (On AudioSet)	None
HTS_AT	Mixup, Time-Frequency Masking (On AudioSet)	None

General Training Details

- Evaluation Metric: Accuracy
- Batch Size: 64

- Training Duration: 4,500 iterations for CNN10 and TFNet, 50 epochs for mn40_as and 100 epochs for HTS_AT
- Loss Function: Cross entropy loss

Training Results

The **Figures 4.3, 4.4, 4.5, 4.6** indicate loss function and accuracy trend versus iteration/epoch. They show the average loss and accuracy among the five folds. No model exhibits overfitting or underfitting suggesting good generalization and performance on data.

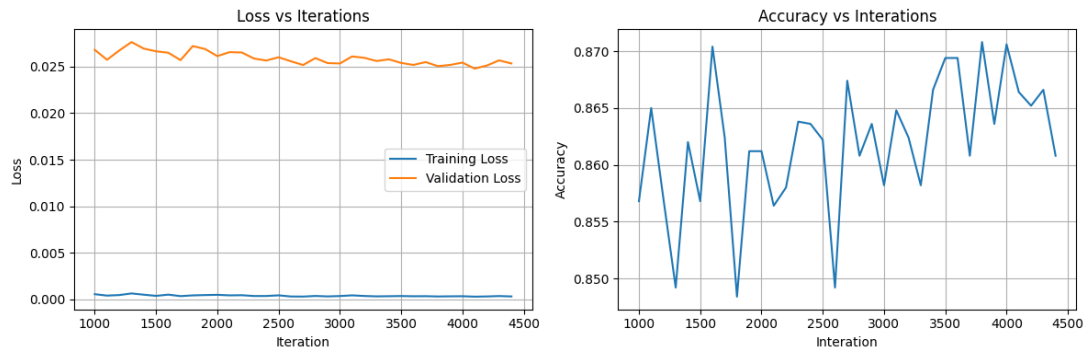


Figure 4.3: On the left loss functions versus iteration and on the right accuracy trend over iterations for the CNN10 model

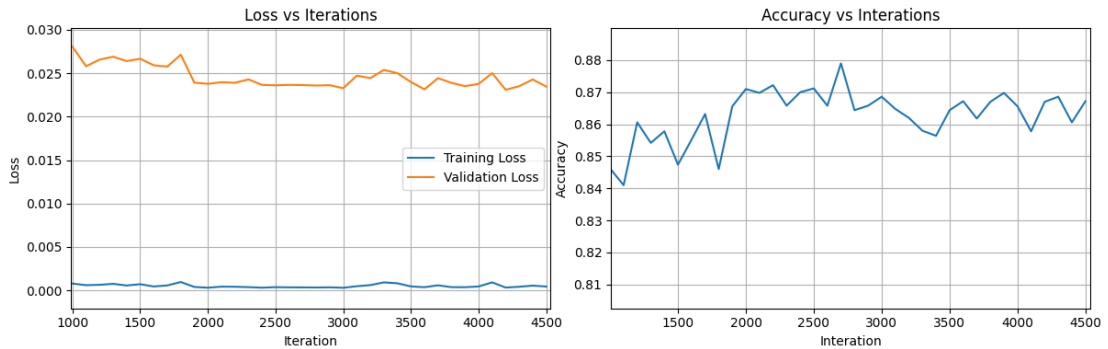


Figure 4.4: On the left loss functions versus iteration and on the right accuracy trend over iterations for the TFNet model

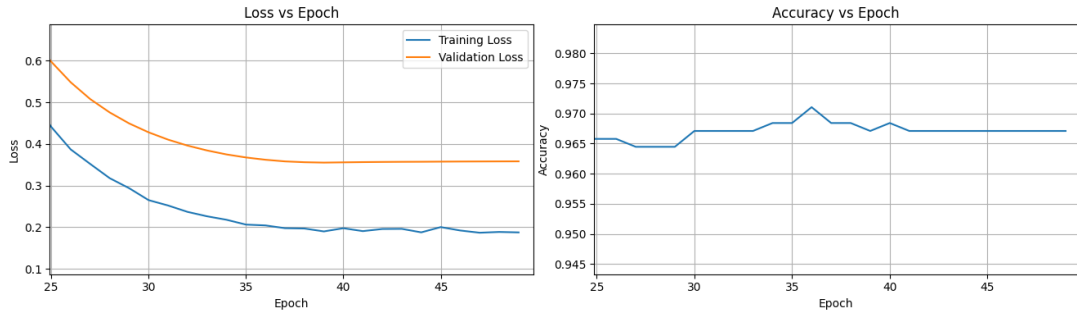


Figure 4.5: On the left loss functions versus epoch and on the right accuracy trend over iterations for the mn40_as model

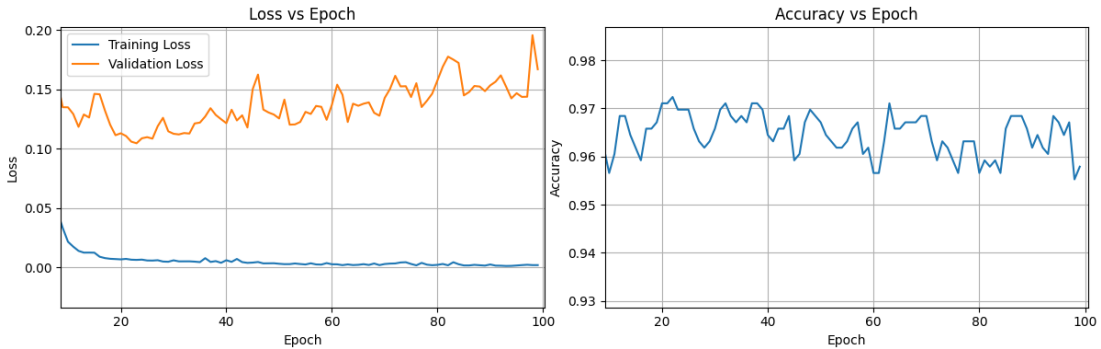


Figure 4.6: On the left loss functions versus epoch and on the right accuracy trend over iterations for the HTS_AT model

4.2 Inference

To evaluate the resilience of the models when the background noise is present, the accuracy is assessed. To perform this, it was necessary to save the checkpoints at the last training iteration/epoch of each model. Then we proceed with the generation of two categories of noise such as colored and mix. Once the nominal noise is generated, the power of the noise introduced into the sensitivity bandwidth of the model is adjusted by setting SNR values. The resulting accuracy reached in the last iteration/epoch for each model and folds are available in **Table 4.4**. In this experiment the model which reaches the best performance over the folds is selected to perform inference. Looking the **Table 4.4**, CNN10 and TFNet achieve best validation accuracy during the training on fold 4, while for mn40_as and HTS_AT the best one is achieved on fold 2.

Table 4.4: Reached accuracies during training for each fold and model

	ACCURACY (%)			
Fold	CNN10	TFNet	mn40__as	HTS__AT
1	85.4	88.2	94.74	94.74
2	88.2	86.2	99.34	99.34
3	86.8	82.2	96.71	96.71
4	91.0	92.1	96.05	94.08
5	79.0	84.9	96.71	94.08

4.2.1 Dataset Selection for Inference

As previous mentioned, the five folds are totally used for training and testing. However, to provide models with never-before-seen data, a new dataset for inference was created by collecting audio from the webpage FreeSound ¹. Since ESC-50 has collected data from FreeSound as well and the publication of dataset dates on 2015, the searching of data for inference has been limited to audio inserted as recent as possible to avoid the risk to choose unintentionally the same sample adopted during the training and test. 8 audio per class are picked for a total of 152 samples. In addition, to satisfy the 5 seconds long constraints, part of them have been shortened or stretched by using the Python library `librosa`. For those samples which audio length results to be longer than 5 seconds and paying attention to temporal range where event occurs, they have been shortened by setting offset and duration parameters of `librosa.load()` function. For audio shorter than 5 seconds, instead, zero-padding is carried out by inserting zero-value samples at the end. Once the new dataset is created, inference without noise is done to evaluate the percentage loss accuracy between accuracy reached during the training and the accuracy achieved in the initial inference step. The results are listed in **Table 4.5**. As expected, all models record a accuracy loss when they are put in a situation to predict data never seen. The model which loses less is HTS_AT with a value loss of 6.62%, followed by mn40__as loss of 8.6%. Despite CNN10 starts with a reached accuracy during the training lower than TFNet, it still records a minor loss in the inference of 9.62% against the 11.32% of TFNet.

4.2.2 Noises Selection

In this experiment, due typologies of noise are considered: colored noise and mix noise. Colored noise, composed by white, pink, brownian (red), blue and violet

¹<https://freesound.org/>

Table 4.5: A comparison between testing accuracy and accuracy reached during inference without noise for each model. Also accuracy loss is listed

Model	Testing Accuracy (%)	Inference Accuracy (%)	Accuracy Loss (%)
CNN10	91.0	82.24	9.62
TFNet	92.0	81.58	11.32
mn40_as	99.34	90.79	8.60
HTS_AT	99.34	92.76	6.62

is a type of noise which has a power spectral density that varies according to the frequency as indicated in **Table 4.6**. White noise has power density equally distributed across the frequencies, pink and brownian (red) focus the noise at lower frequencies with a decreasing spectrum by $\frac{1}{f}$ and $\frac{1}{f^2}$ respectively. Then blue and violet concentrate the noise at higher frequencies with a trend of f and f^2 respectively. Pink, brownian (red), blue and violet noises have been generated by dedicated functions provided by Matlab ² with a sampling frequency of 44.1 kHz. However, for white noise the same blue noise function has been used by modifying PSD slope α to 0. The power spectral density of each noise are pictured in **Figures 4.7, 4.8, 4.9, 4.10, 4.11**. Mix noise offers sounds that can make the environmental more realistic such as TV, door open and close, barking of a dog, car engine pass and rain drops on window. Those noises are picked from website FreeSound ³ properly sampled at 44.1 kHz, transformed if necessary in mono reproduction and shorted to 5 seconds by means of Python library `librosa`.

The noises applied to dataset are:

- white
- pink
- brownian (red)
- blue
- violet
- TV
- door open and close
- dog barking
- car engine pass
- rain drops on window

²<https://it.mathworks.com/matlabcentral/fileexchange/42919-pink-red-blue-and-violet-noise-generation-with-matlab>

³<https://freesound.org/>

Table 4.6: Power spectral density and slope for colored noises

Color Noise	Power Spectral Density	Slope
White	1	0
Pink	$1/f$	-10 dB/dec
Brownian (Red)	$1/f^2$	-20 dB/dec
Blue	f	10 dB/dec
Violet	f^2	20 dB/dec

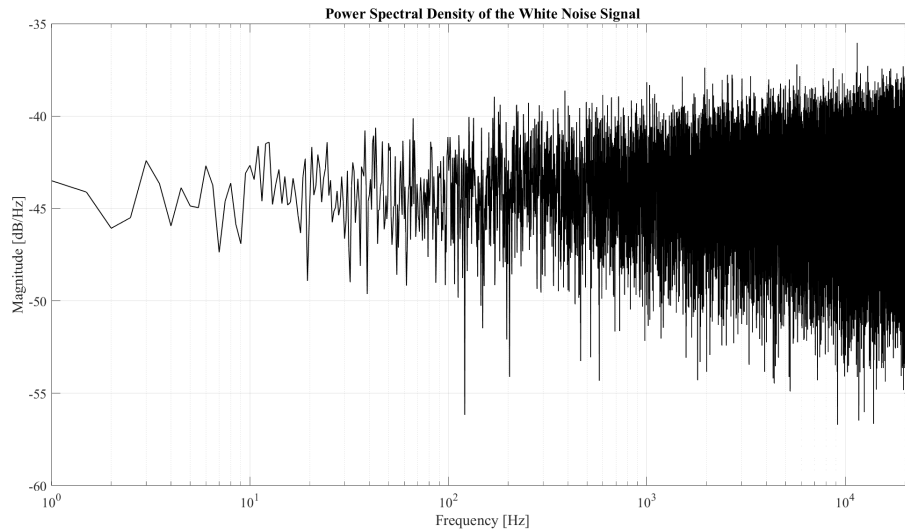


Figure 4.7: Power spectral density of the white noise

4.2.3 Noise Insertion

The generation of the noise is executed once at a nominal power (**Table 4.7**) to make the measurements correlated each other. The variable that controls the noise intensity is SNR, by keeping unchanged audio power, only noise power varies to reach desired SNR. This variation is implemented multiplying temporal samples by a factor α . The amount of noise power that varies with respect to nominal one is demonstrated below.

Starting from PSD definition [68] of a time-discrete variable x_n defined in (4.1), where x_n is the samples sequence of noise, T is the total measurement period defined as $T = (2N + 1)\Delta t$ and t is the duration during which a single sample of the signal is acquired. This formulation is usually called Periodogram and this

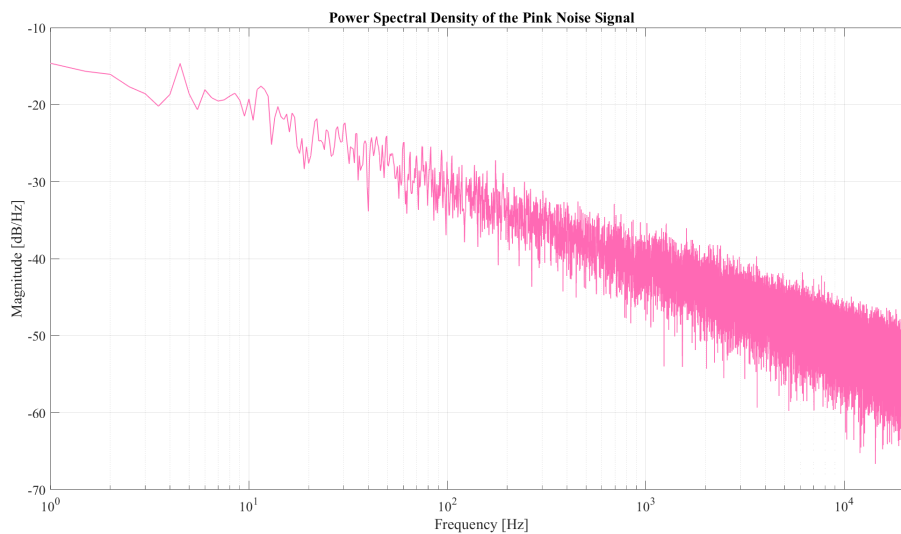


Figure 4.8: Power spectral density of the pink noise

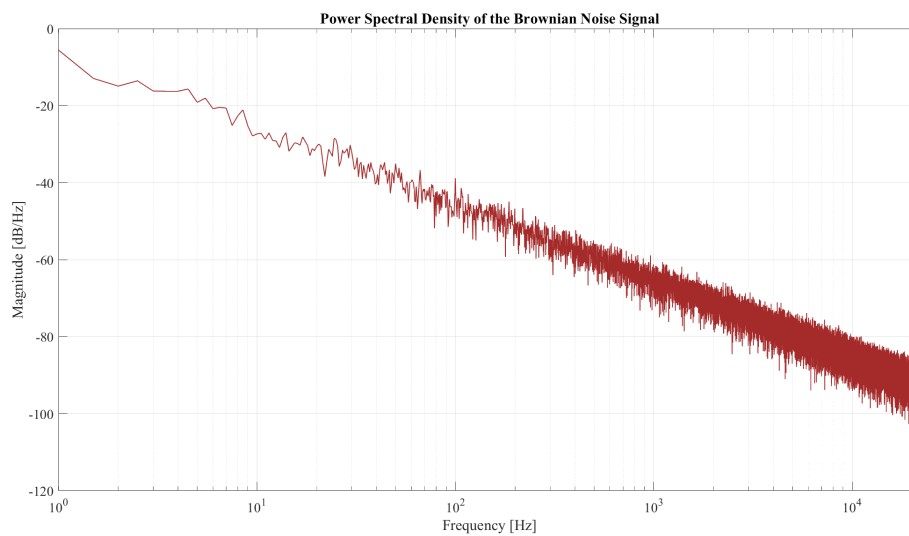


Figure 4.9: Power spectral density of the brownian (red) noise

converges to actual PSD as the averaging time interval T approach to infinity [69]

$$S_{xx}(f) := \lim_{n \rightarrow \infty} \frac{(\Delta t)^2}{T} \left| \sum_{n=-N}^N x_n e^{-i2\pi f n \Delta t} \right|^2. \quad (4.1)$$

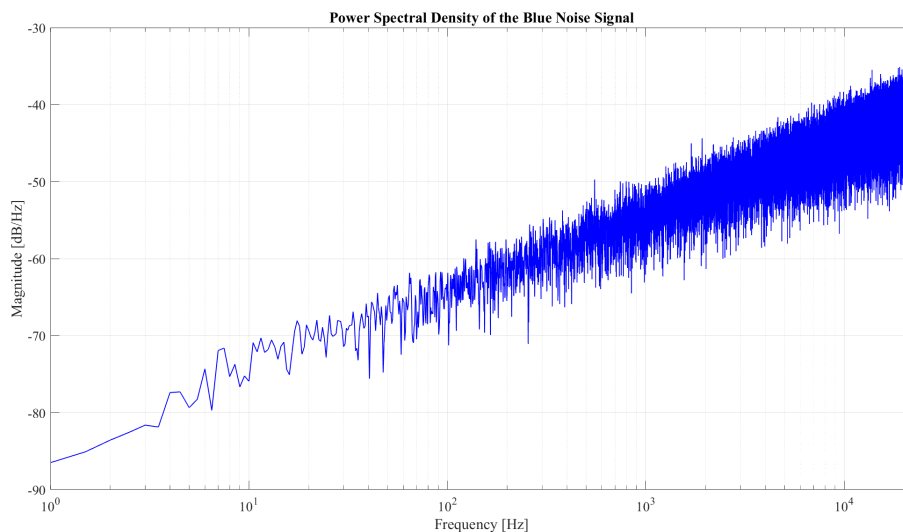


Figure 4.10: Power spectral density of the blue noise

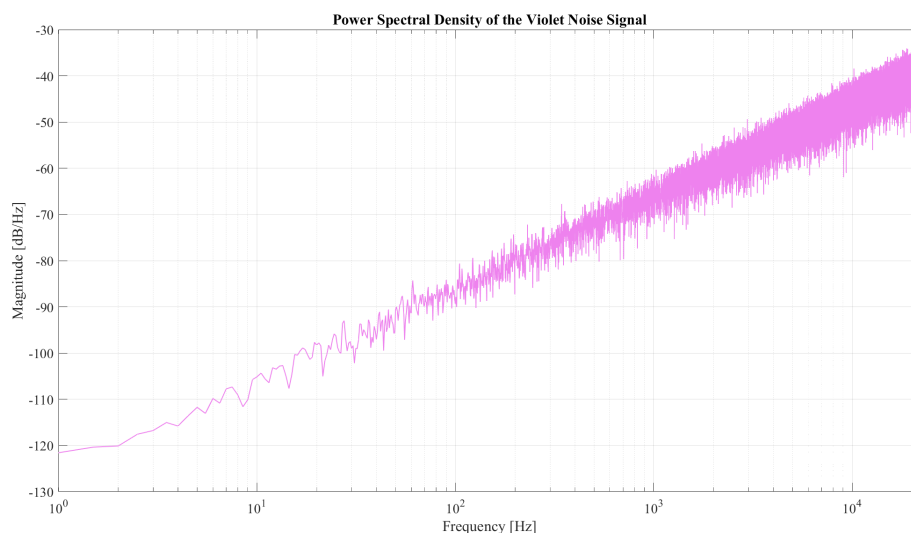


Figure 4.11: Power spectral density of the violet noise

By multiplying the samples by a factor α , the new PSD is described in (4.2)

$$S_{xx}(f) = \lim_{n \rightarrow \infty} \frac{(\Delta t)^2}{T} \left| \sum_{n=-N}^N \alpha \cdot x_n e^{-i2\pi f n \Delta t} \right|^2. \quad (4.2)$$

Since α is a constant, it can be taken out by exploiting the distributive property

Table 4.7: Nominal noises power calculated in model sensitivity bandwidth

Nominal Noise Power	Model			
	CNN10	TFNet	mn40_as	HTS_AT
White (mW)	497.77	497.77	681.88	633.66
Pink (mW)	491.61	491.61	965.08	513.14
Brownian (Red) (mW)	13.74	13.74	999.98	13.76
Blue (mW)	251.34	251.34	464.12	404.32
Violet (mW)	124.14	124.14	313.05	254.10
TV (μ W)	4.32	4.32	6.12	4.32
Door Open and Close (μ W)	107.12	107.12	151.10	107.20
Dog Barking (mW)	3.0863	3.0863	3.0892	3.0864
Car Engine Pass (mW)	11.481	11.481	11.485	11.495
Rain Drops on Window (μ W)	6.74	6.74	6.74	6.74

of summation (4.3)

$$S_{xx}(f) = \lim_{n \rightarrow \infty} \frac{(\Delta t)^2}{T} |\alpha|^2 \left| \sum_{n=-N}^N x_n e^{-i2\pi f n \Delta t} \right|^2 \quad (4.3)$$

and the properties of homogeneity of the limits (4.4)

$$S_{xx}(f) = |\alpha|^2 \lim_{n \rightarrow \infty} \frac{(\Delta t)^2}{T} \left| \sum_{n=-N}^N x_n e^{-i2\pi f n \Delta t} \right|^2. \quad (4.4)$$

The new PSD as function of α factor and nominal $S_{xx0}(f)$ is obtained in (4.5)

$$S_{xx}(f) = |\alpha|^2 S_{xx0}(f). \quad (4.5)$$

Then performing the integration of PSD into sensitivity bandwidth of the model (**Table 4.8**), the noise power P_n as function of nominal one P_{n0} and multiplication factor α is achieved in (4.6)

$$P_n = \int_{f_{min}}^{f_{max}} S_{xx}(f) df = \int_{f_{min}}^{f_{max}} |\alpha|^2 S_{xx0}(f) df = |\alpha|^2 P_{n0}. \quad (4.6)$$

Substituting (4.7)

$$P_n = |\alpha|^2 P_{n0} \quad (4.7)$$

into SNR definition (4.8),

$$SNR = 10 \log\left(\frac{P_x}{P_n}\right) \quad (4.8)$$

Table 4.8: The frequency band that the model is able to detect and classify

Model	f_{min} (Hz)	f_{max} (Hz)
mn40_as	0	15000
HTS_AT	50	14000
TFNet	50	11025
CNN10	50	11025

(4.9) is got

$$SNR = 10 \log \left(\frac{P_x}{\alpha^2 P_{n0}} \right). \quad (4.9)$$

During the inference, SNR values are chosen earlier, so α values should be extracted to reach those SNRs. By executing simple algebraic steps in (4.10)

$$10^{\frac{SNR}{10}} = \frac{P_x}{\alpha^2 P_{n0}} \quad (4.10)$$

and (4.11),

$$\alpha^2 = \frac{P_x}{P_{n0}} 10^{-\frac{SNR}{10}} \quad (4.11)$$

(4.12) will be used to derive, given a SNR, the factor α by which noise samples are multiplied with

$$\alpha = \sqrt{\frac{P_x}{P_{n0}} 10^{-\frac{SNR}{10}}}. \quad (4.12)$$

Each audio belonging to dataset of inference has different level of power, so if the same factor α would be used to multiply the noise, a different SNR values are obtained over all dataset. Considering that a unique SNR value is preferred over all dataset, each audio has its corresponding factor α (extracted using (4.12)) to adjust noise and to reach desired SNR.

Refer to Appendix A for more details about noise insertion on dataset.

Chapter 5

Experimental Results

5.1 Model Complexity

The complexity of the models is reported in terms of the number of parameters even if it is not the only evaluation metric and additional factors should be considered to obtain a more accurate estimate. Time complexity and memory complexity have not been analyzed because they are values that strongly depend on the available resources and that could influence the results.

As can be seen from **Table 5.1**, HTS_AT is the most complex model while the least complex one is mn40_as. The good result obtained by mn40_as is due to the use of the Depthwise Separable Convolution of the student model (MobileNetV3), a convolution operation that replaces the traditional one and specifically designed to reduce the number of parameters. The high complexity of HTS_AT, on the other hand, is mainly due to the use of the Transformer and the self-attention mechanism it employs, but for which it is distinguished by its ability to capture long-range relationships and learn complex representations.

Table 5.1: Model complexity in terms of number of parameters for each model

Model	Parameters
CNN10	4.95M
TFNet	4.96M
mn40_as	4.88M
HTS_AT	31M

5.2 Inference Results

Experimental results are collected in **Figures 5.1, 5.3, 5.5, 5.7, 5.9, 5.11, 5.13, 5.15, 5.17, 5.19** and noise mel spectrogram applied to an class 31 audio of the dataset with 10 dB SNR pictured in **Figures 5.2, 5.4, 5.6, 5.8, 5.10, 5.12, 5.14, 5.16, 5.18, 5.20**. The first observation is the decreasing of the accuracies when SNR gets lower, as expected. HTS_AT and mn40_as are the models which better resist to the noise and for some of them slowly degrade and this confirms the initial hypothesis that pre-training helps to improve the classification even in noisy environment. TFNet and CNN10 are quite similar in trend for white, pink, brownian (red), blue, violet, TV, dog and rain noises. However for door and car, CNN10 seems to get worse slower than TFNet. HTS_AT has been more robust or similar to mn40_as in almost all noises except for brownian (red) noise. The results obtained with mix noise (TV, door open and close, dog barking, car engine pass and rain drops on the window) do not ensure a comprehensive evaluation of model robustness. This is because mix noises do not have defined spectral characteristics, unlike colored ones. For example, a barking dog audio should contain one or more events, and even if they are realistic for practical applications, the model may expose training biases. Since in this experiment an individual audio mix has been added to the dataset and so the evaluation of accuracy is limited to it, the validity of the results obtained for TV, door, dog, car engine and rain should be further tested in more depth by selecting more audio tracks per noise category.

In addition, during the experiment has been observed that the curves tends to an accuracy value of almost 5%. This should not happen because if a model always make mistakes, it should get an accuracy of zero instead. The reason is easily understandable by looking **Figures 5.21, 5.22, 5.23, 5.24** where show confusion matrices of models when white noise is applied at SNR of -25 dB: the networks classify the category 36 for CNN10, TFNet and HTS_AT or 35 for mn40_as regardless of input audio it receives. Therefore this to say that even for low values (for example in **Figure 5.25** where white noise is applied at -10 dB) the correct classification of category 35 (indentified by the network as the default class when it no longer recognizes the input) is questioned precisely because, having learned to recognize it in case of doubt, it is not clear from the results if it did right because he really recognized the sound or because it is undecided.

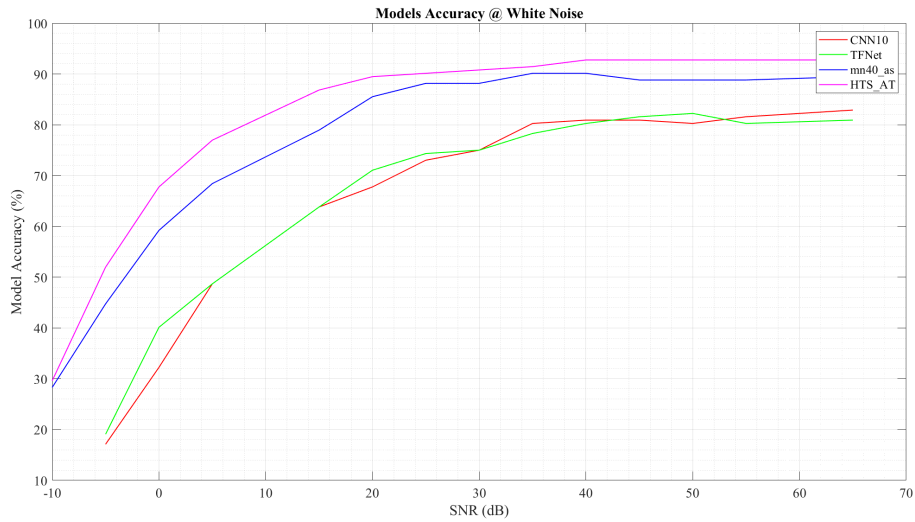


Figure 5.1: Models Accuracy when white noise is applied.

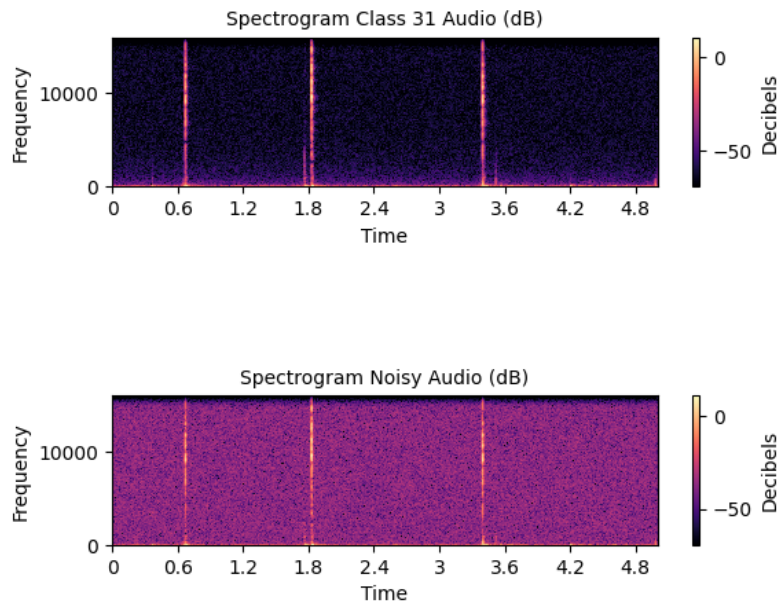


Figure 5.2: Class 31 audio without (up) and with white noise (down) applied at 10 dB SNR mel spectrogram.

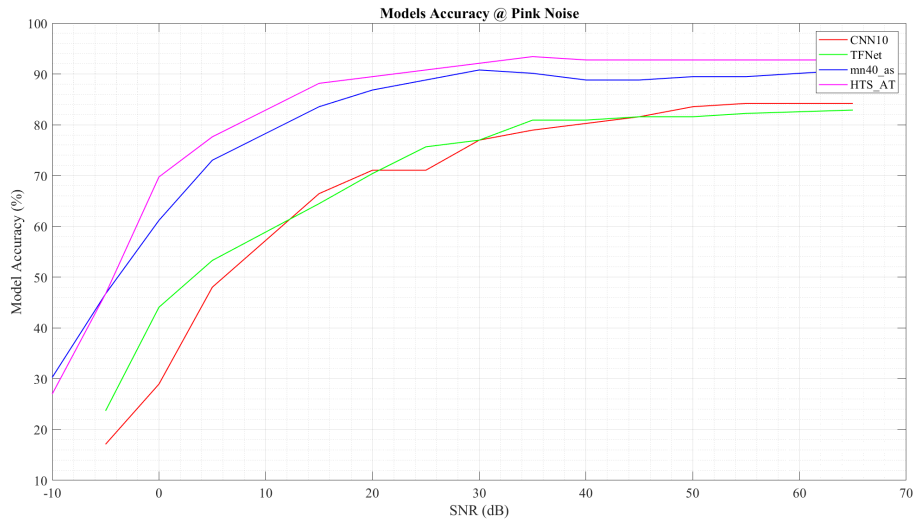


Figure 5.3: Models Accuracy when pink noise is applied.

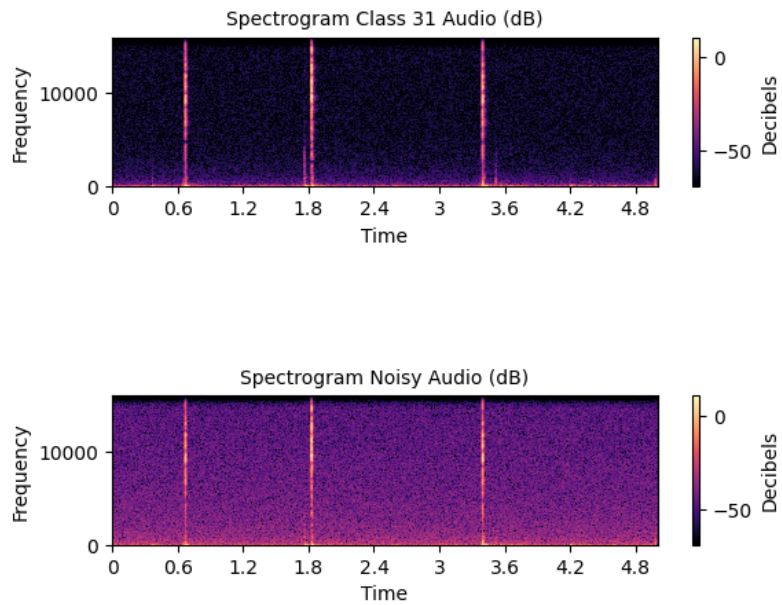


Figure 5.4: Class 31 audio without (up) and with pink noise (down) applied at 10 dB SNR mel spectrogram.

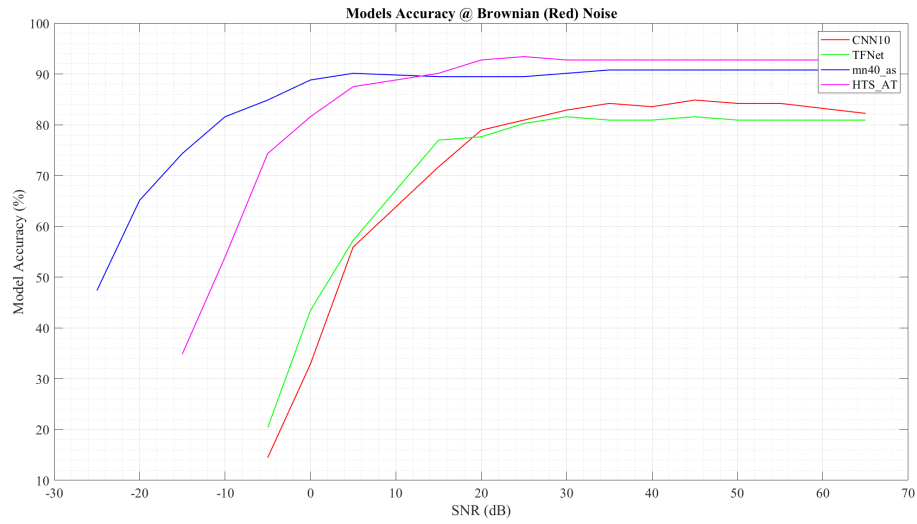


Figure 5.5: Models Accuracy when brownian (red) noise is applied.

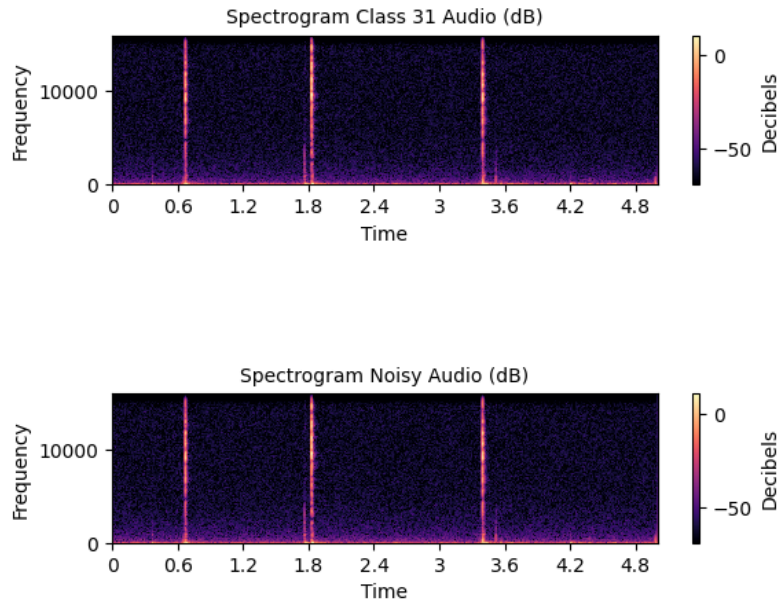


Figure 5.6: Class 31 audio without (up) and with brownian (red) noise (down) applied at 10 dB SNR mel spectrogram.

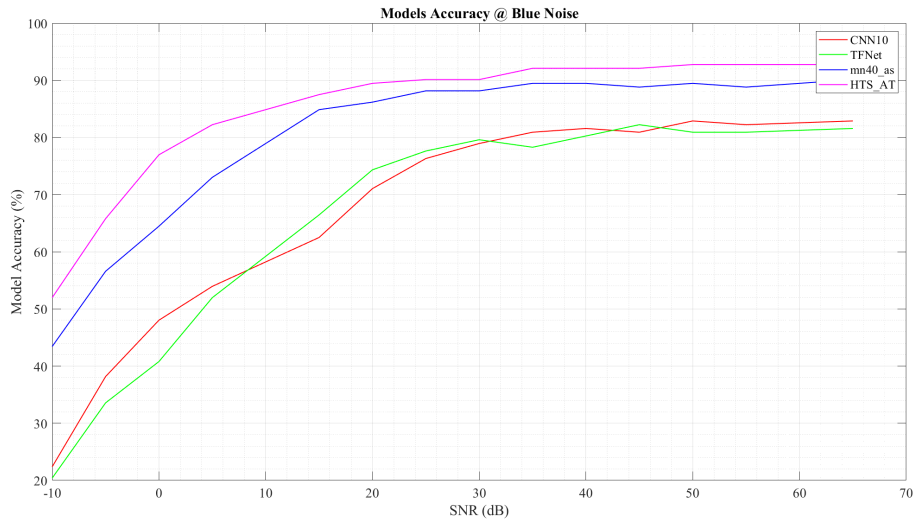


Figure 5.7: Models Accuracy when blue noise is applied.

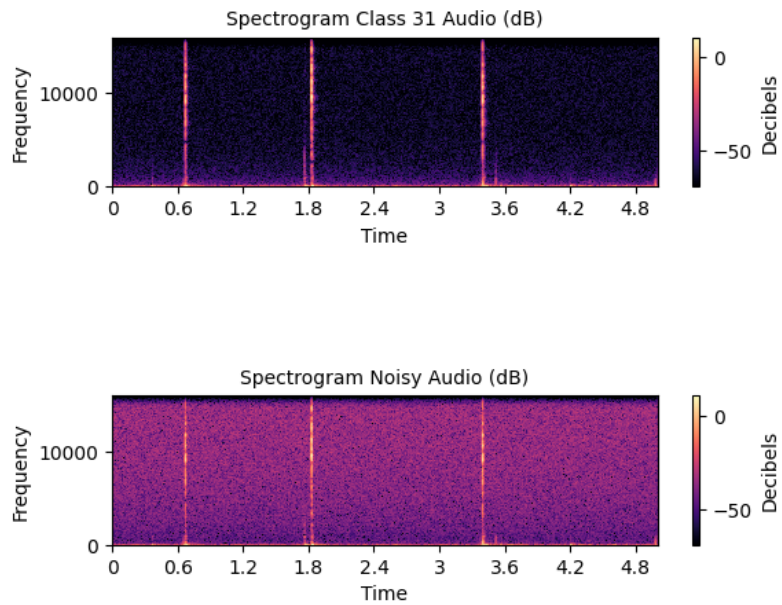


Figure 5.8: Class 31 audio without (up) and with blue noise (down) applied at 10 dB SNR mel spectrogram.

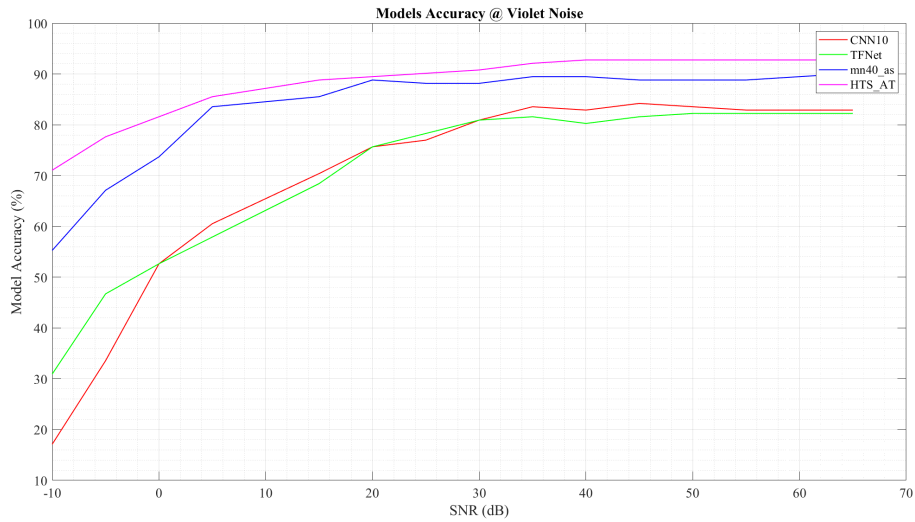


Figure 5.9: Models Accuracy when violet noise is applied.

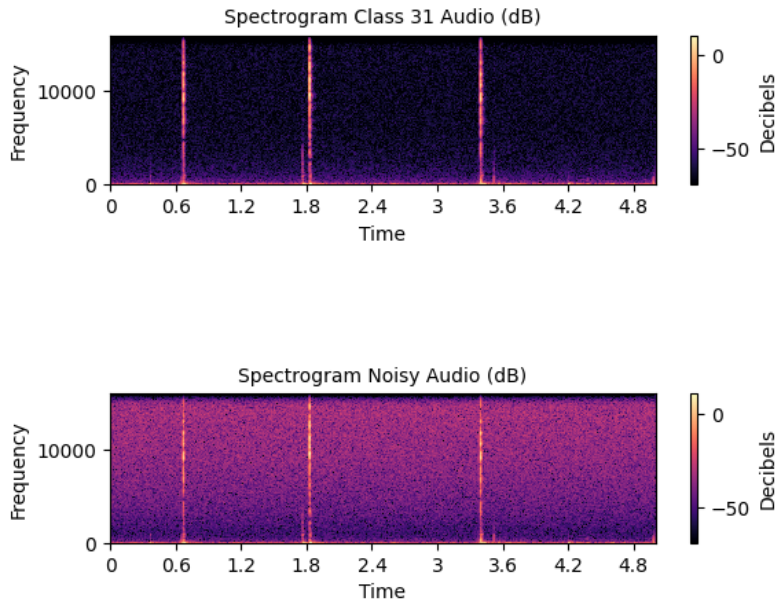


Figure 5.10: Class 31 audio without (up) and with violet noise (down) applied at 10 dB SNR mel spectrogram.

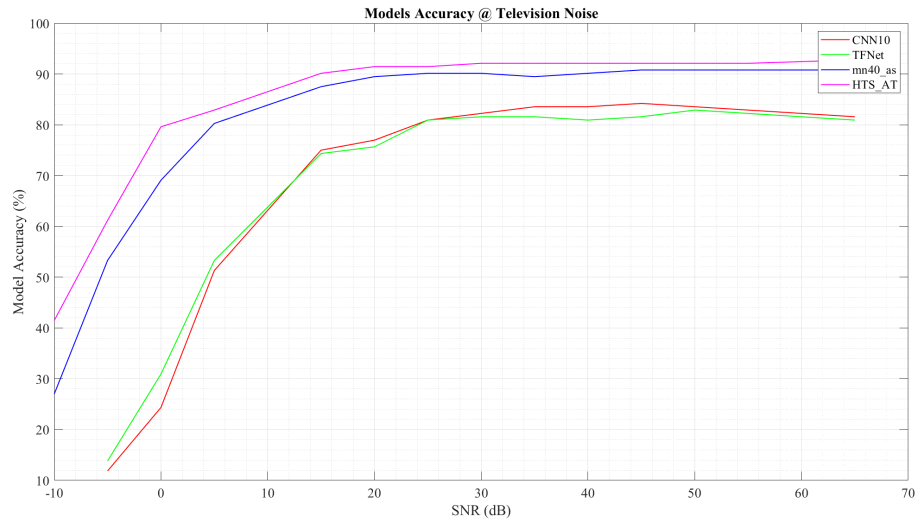


Figure 5.11: Models Accuracy when TV noise is applied.

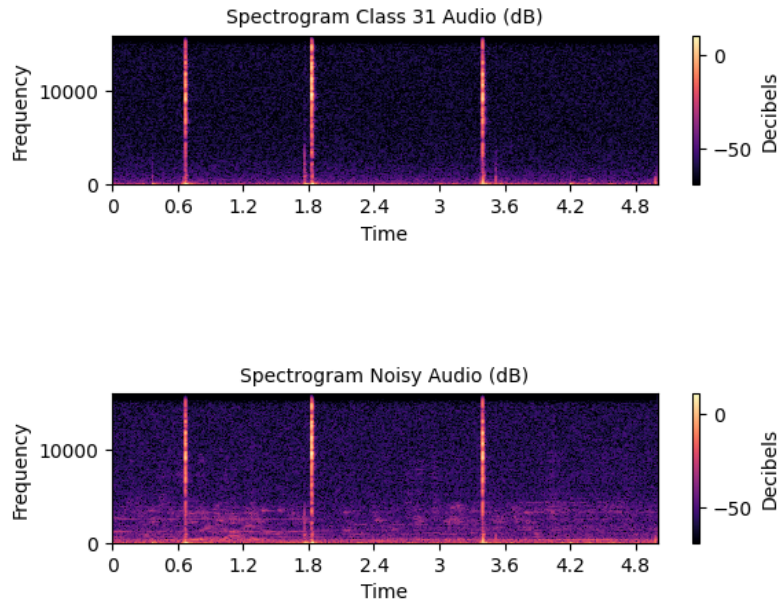


Figure 5.12: Class 31 audio without (up) and with TV noise (down) applied at 10 dB SNR mel spectrogram.

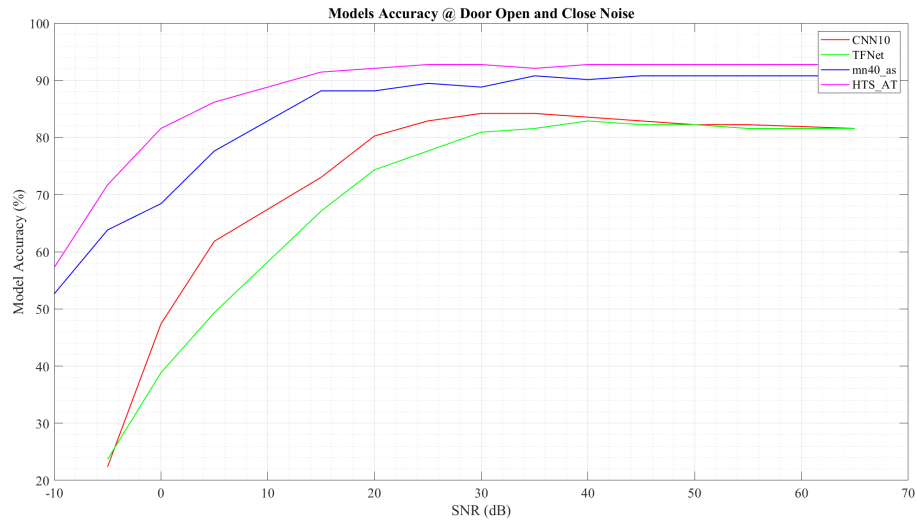


Figure 5.13: Models Accuracy when door open and close noise is applied.

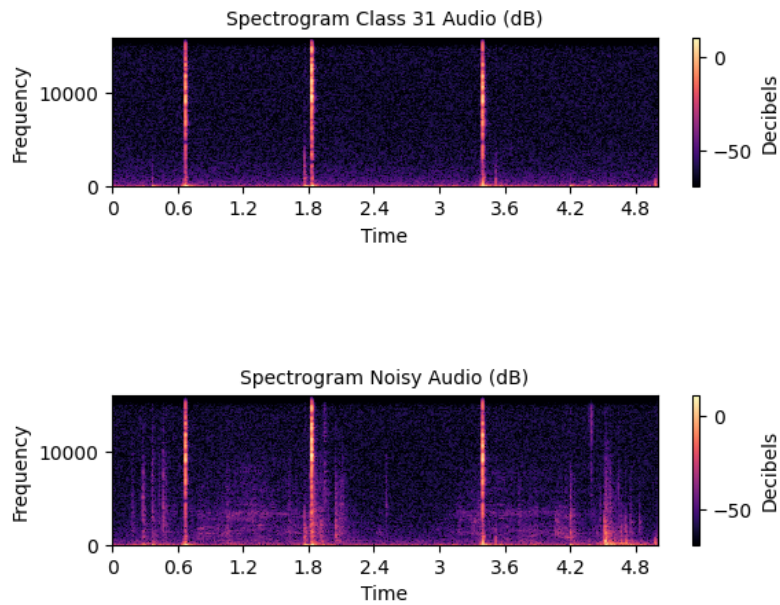


Figure 5.14: Class 31 audio without (up) and with door open and close noise (down) applied at 10 dB SNR mel spectrogram.

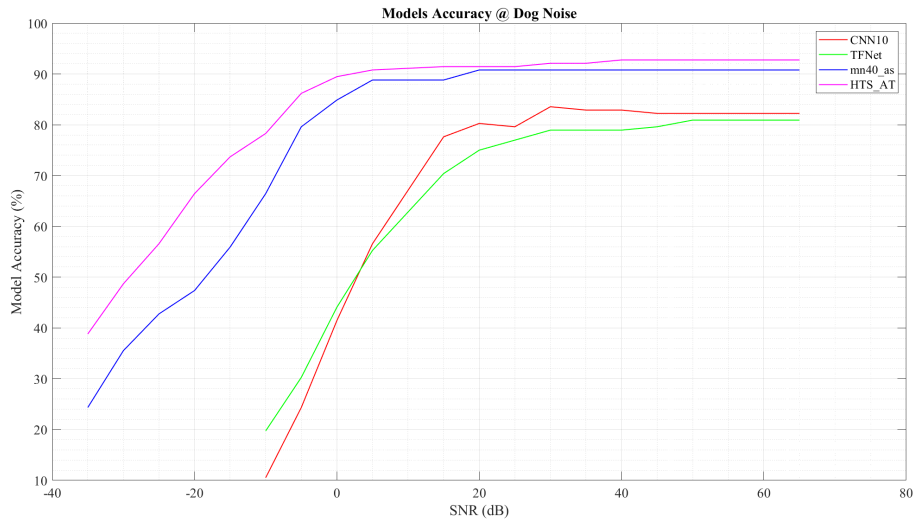


Figure 5.15: Models Accuracy when dog noise is applied.

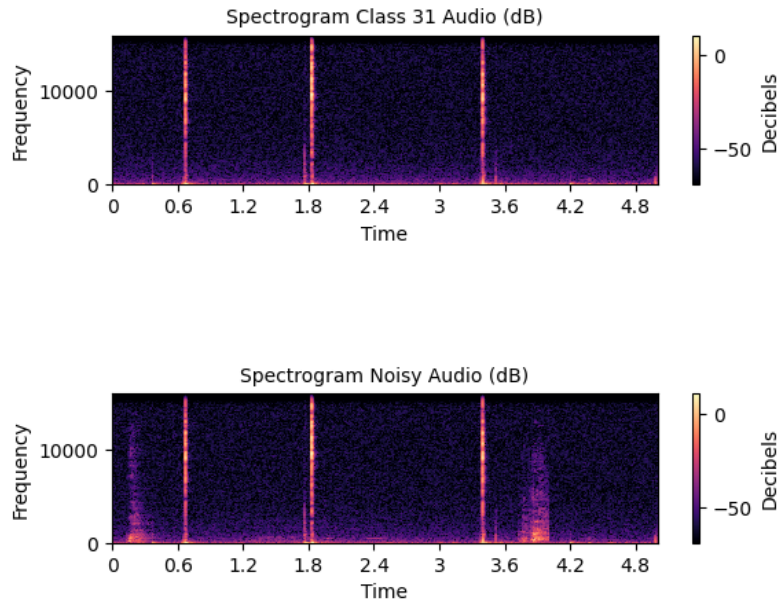


Figure 5.16: Class 31 audio without (up) and with dog noise (down) applied at 10 dB SNR mel spectrogram.

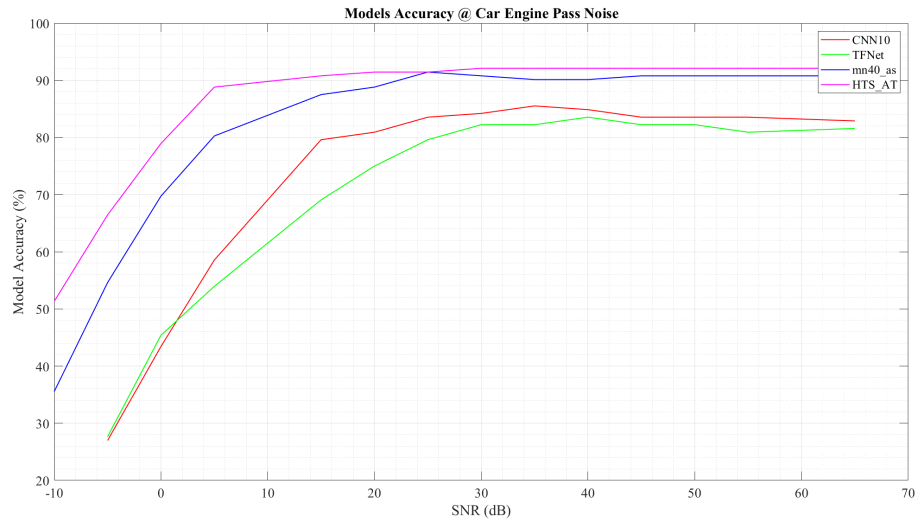


Figure 5.17: Models Accuracy when car engine pass noise is applied.

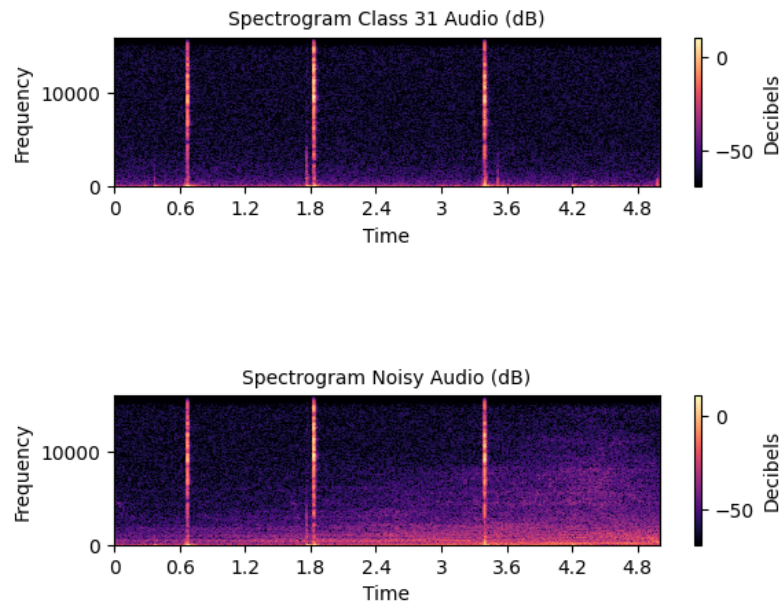


Figure 5.18: Class 31 audio without (up) and with car engine pass noise (down) applied at 10 dB SNR mel spectrogram.

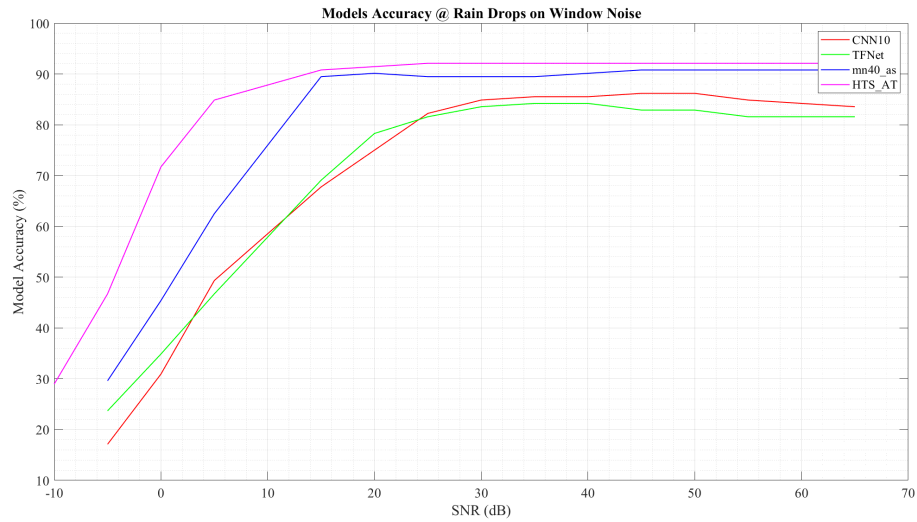


Figure 5.19: Models Accuracy when rain drops on window noise is applied.

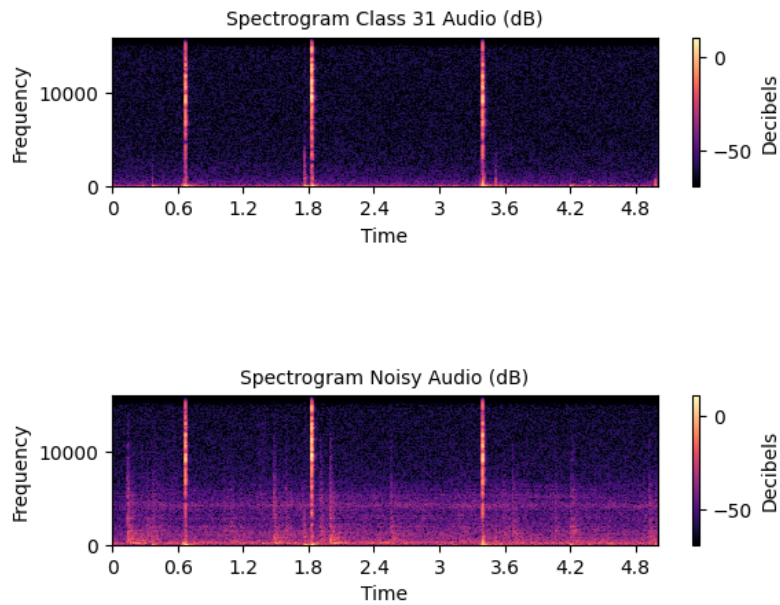


Figure 5.20: Class 31 audio without (up) and with rain drops on window noise (down) applied at 10 dB SNR mel spectrogram.

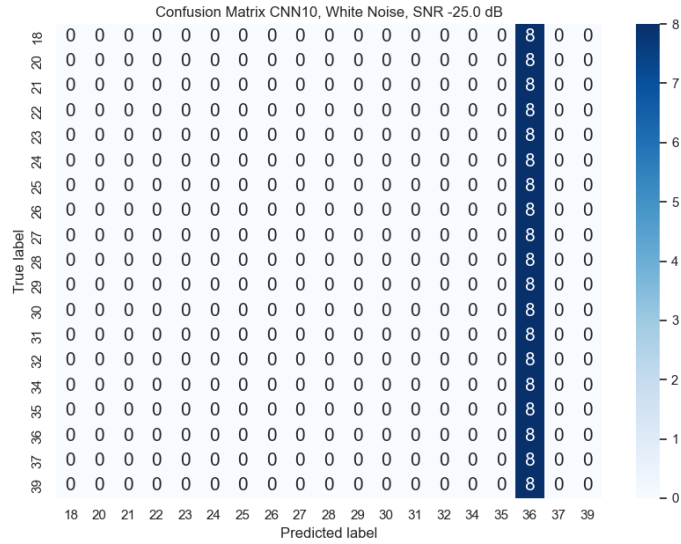


Figure 5.21: Confusion matrix obtained for CNN10 model, white noise at SNR of -25 dB.

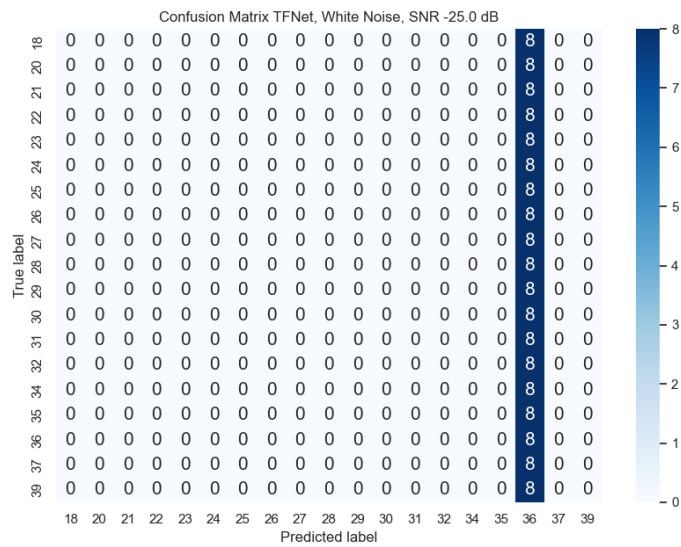


Figure 5.22: Confusion matrix obtained for TFNet model, white noise at SNR of -25 dB.

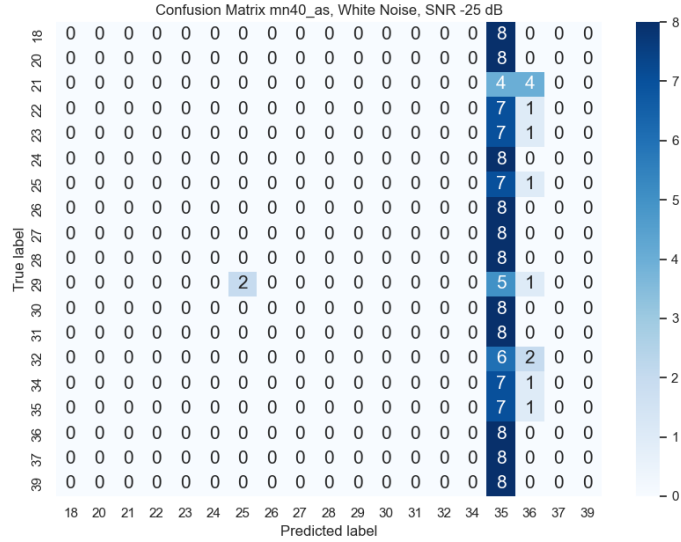


Figure 5.23: Confusion matrix obtained for mn40_as model, white noise at SNR of -25 dB.

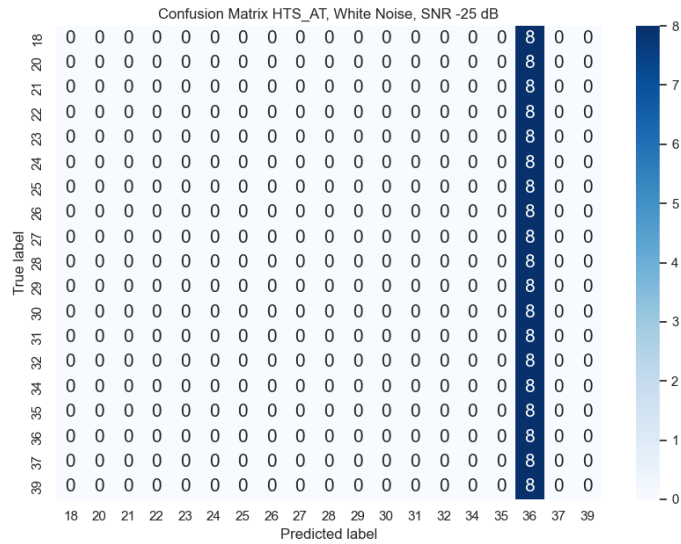


Figure 5.24: Confusion matrix obtained for HTS_AT model, white noise at SNR of -25 dB.

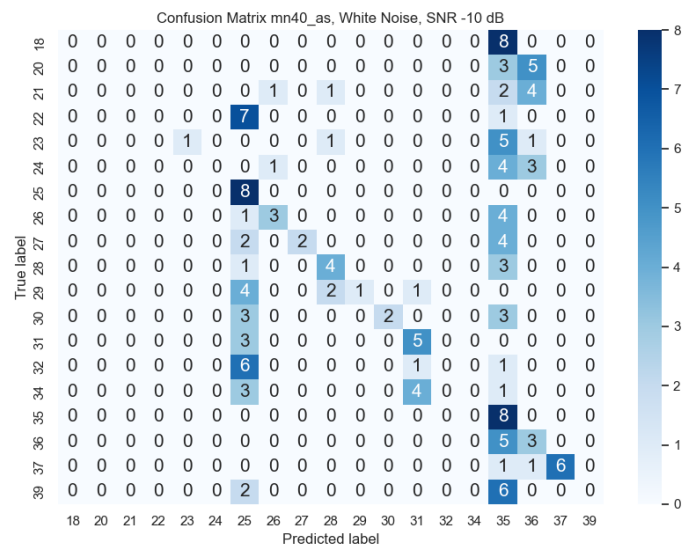


Figure 5.25: Confusion matrix obtained for mn40_as model, white noise at SNR of -10 dB.

Chapter 6

Conclusion

This experiment has been conducted to identify what techniques could make a model more robust to noise and to give some inspiration in the realization of future neural networks. We noticed how the tested models showed a worse performance when exposed to noise of different nature and power and this shows the importance of considering noise as a critical variable when developing classification algorithms environmental. In addition, the results show how the interchangeability of components originally designed for different tasks (image classification for MobileNetV3 and sequence transduction or neural machine translation for Transformers) find positive application also for the classification of environmental sounds. Pre-trained models with AudioSet showed greater generalization capability and greater sensitivity than models trained exclusively with ESC-50 reduced to 19 classes. HTS_AT and mn40_as have had access to a previous knowledge of the characteristics of the sounds by discriminating more accurately and allowing them to get the best performance. The time-frequency attention mechanisms introduced by TFNet to improve CNN10 do not seem to improve much the classification in the presence of noise, however, the self-attention mechanisms in the HTS_AT Swin-Transformer and transformer-encoder of the model teacher of mn40_as (passt), have given good results in terms of robustness.

By making a trade-off between robustness and model complexity, mn40_as can be a model which taking inspiration from. It is able to exploit the advantages of Transformer and CNN: the first one is able to capture complex relationships from a large amount of data, however CNN is able to learn complex tasks from a limited one.

Possible future works could be:

- including in the comparison the BEATs model [70], which for lack of complete codes of the researchers was not considered;
- including in the comparison also a model which uses a RNN;
- pre-training CNN10 with AudioSet.

Appendix A

Python Codes

```
1 def bandpower(x, fs, fmin, fmax):  
2     f, Pxx = scipy.signal.periodogram(x, fs=fs)  
3     ind_min = scipy.argmax(f>fmin)-1  
4     ind_max = scipy.argmax(f>fmax)-1  
5     return scipy.trapz(Pxx[ind_min:ind_max], f[ind_min:ind_max])
```

```
1 def noise_insertion(NOISE, fmin, fmax, SNR):
2     noise_audio_path = '../Noise/{}_noise.wav'.format(NOISE)
3     noise, sr = librosa.load(noise_audio_path, sr=None)
4     audio_path = '../esc19/audio'
5     noisy_audio_path = '../esc19_noise/audio'
6
7     if not os.path.exists(noisy_audio_path):
8         os.makedirs(noisy_audio_path)
9         os.makedirs('../esc19_noise/meta')
10
11     shutil.copyfile('../esc19/meta/esc19.csv', '../esc19_noise/meta/
12     esc19.csv')
13
14     for audio_file in os.listdir(audio_path):
15         if audio_file.endswith('.wav'):
16             audio, sr = librosa.load(os.path.join(audio_path,
17             audio_file), sr=None)
18             P_x = bandpower(audio, sr, fmin, fmax)
19             P_n0 = bandpower(noise, sr, fmin, fmax)
20             alpha = np.sqrt((P_x*10**(-SNR/10))/(P_n0))
21             noisy_audio = audio + noise * alpha
22             write(os.path.join(noisy_audio_path, audio_file), sr,
23             noisy_audio.astype(np.float32))
```

Bibliography

- [1] Y. Bengio I. Goodfellow and A. Courville. In: *Deep Learning*. 2016. Chap. 5, pp. 100–107 (cit. on p. 3).
- [2] Y. Bengio I. Goodfellow and A. Courville. In: *Deep Learning*. 2016. Chap. 5, pp. 100–107 (cit. on p. 4).
- [3] Y. Bengio I. Goodfellow and A. Courville. In: *Deep Learning*. 2016. Chap. 5, pp. 113–115 (cit. on pp. 5, 6).
- [4] Yuan Feng, Di Wu, Mark G. Stewart, and Wei Gao. «Past, current and future trends and challenges in non-deterministic fracture mechanics: A review». In: *Computer Methods in Applied Mechanics and Engineering* 412 (2023). DOI: <https://doi.org/10.1016/j.cma.2023.116102>. URL: <https://www.sciencedirect.com/science/article/pii/S0045782523002268> (cit. on p. 7).
- [5] Y. Bengio I. Goodfellow and A. Courville. In: *Deep Learning*. 2016. Chap. 5, pp. 168–171 (cit. on p. 7).
- [6] I.A. Italia. *Funzioni Di Attivazione Nel Deep Learning La Guida Completa / Intelligenza Artificiale Italia Blog*. 2022. URL: <https://www.intelligenzaartificialeitalia.net/post/funzioni-di-attivazione-nel-deep-learning-la-guida-completa> (cit. on pp. 7–9).
- [7] K. E. Koech. *softmax Activation Function – How It Actually Works*. *Medium*. 2021. URL: <https://towardsdatascience.com/softmax-activation-function-how-it-actually-works-d292d335bd78> (cit. on p. 7).
- [8] *Softmax function - Wikipedia*. URL: https://en.wikipedia.org/wiki/Softmax_function (cit. on p. 7).
- [9] Y. Bengio I. Goodfellow and A. Courville. In: *Deep Learning*. 2016. Chap. 4, pp. 82–86 (cit. on p. 9).
- [10] *Gradient descent - Wikipedia*. 2014. URL: https://en.wikipedia.org/wiki/Gradient_descent (cit. on p. 9).
- [11] *Backpropagation - Wikipedia*. URL: <https://en.wikipedia.org/wiki/Backpropagation> (cit. on p. 9).

- [12] K. E. Koech. *How Does Back-Propagation Work in Neural Networks?* Medium. 2022. URL: <https://towardsdatascience.com/how-does-back-propagation-work-in-neural-networks-with-worked-example-bc59dfb97f48> (cit. on p. 9).
- [13] Y. Bengio I. Goodfellow and A. Courville. In: *Deep Learning*. 2016. Chap. 6, pp. 204–224 (cit. on p. 9).
- [14] Y. Bengio I. Goodfellow and A. Courville. In: *Deep Learning*. 2016. Chap. 9, pp. 364–371 (cit. on p. 9).
- [15] R. Parmar. *Common Loss functions in machine learning*. Medium. 2018. URL: <https://towardsdatascience.com/common-loss-functions-in-machine-learning-46af0ffc4d23> (cit. on p. 10).
- [16] Y. Bengio I. Goodfellow and A. Courville. In: *Deep Learning*. 2016. Chap. 9, pp. 330–333 (cit. on p. 10).
- [17] *RGB color model - Wikipedia*. 2022. URL: https://en.wikipedia.org/wiki/RGB_color_model#cite_note-RWGHunt-7 (cit. on p. 11).
- [18] *Understanding of Convolutional Neural Network (CNN) – Deep Learning*. Medium. 2019. URL: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148> (cit. on p. 12).
- [19] M. Basavarajaiah. *Which pooling method is better? Maxpooling vs minpooling vs average pooling*. Medium. 2019. URL: <https://medium.com/@bdhuma/which-pooling-method-is-better-maxpooling-vs-minpooling-vs-average-pooling-95fb03f45a9> (cit. on p. 15).
- [20] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: 1502.03167 [cs.LG] (cit. on p. 14).
- [21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. «Dropout: A Simple Way to Prevent Neural Networks from Overfitting». In: *Journal of Machine Learning Research* 15 (June 2014), pp. 1929–1958 (cit. on pp. 14, 15).
- [22] H. Yadav. *Dropout in Neural Networks*. Medium. 2023. URL: <https://towardsdatascience.com/dropout-in-neural-networks-47a162d621d9> (cit. on p. 14).
- [23] M. Mishra. *Convolutional Neural Networks, Explained*. Medium. 2020. URL: <https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939> (cit. on pp. 16, 17).

- [24] Han Xiao, Kashif Rasul, and Roland Vollgraf. «Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms». In: (Aug. 2017) (cit. on p. 16).
- [25] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. arXiv: 1706.03762 [cs.CL] (cit. on pp. 16, 18, 19).
- [26] Maxime. *What is a Transformer?* Medium. 2019. URL: <https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04> (cit. on p. 19).
- [27] M. Huzaifah. *Comparison of Time-Frequency Representations for Environmental Sound Classification using Convolutional Neural Networks*. 2017. arXiv: 1706.07156 [cs.CV] (cit. on p. 20).
- [28] L. Roberts. *Understanding the Mel Spectrogram*. Medium. 2020. URL: <https://medium.com/analytics-vidhya/understanding-the-mel-spectrogram-fca2afa2ce53> (cit. on p. 20).
- [29] *Short-time FFT - MATLAB- MathWorks Italia*. 2020. URL: <https://it.mathworks.com/help/dsp/ref/dsp.stft.html> (cit. on p. 24).
- [30] musikalkemist. *Short-Time Fourier Transform Explained Easily*. 2020. URL: <https://github.com/musikalkemist/AudioSignalProcessingForML/blob/master/15%20-%20Short-Time%20Fourier%20Transform%20explained%20easily/Short-Time%20Fourier%20Transform%20Explained%20Easily.pdf> (cit. on p. 21).
- [31] S. Braun. «WINDOWS». In: *Encyclopedia of Vibration*. Ed. by S. Braun. Oxford: Elsevier, 2001, pp. 1587–1595. ISBN: 978-0-12-227085-7. DOI: <https://doi.org/10.1006/rwvb.2001.0052>. URL: <https://www.sciencedirect.com/science/article/pii/B0122270851000527> (cit. on p. 21).
- [32] *Hann (Hanning) Window - MATLAB Hann- MathWorks Italia*. URL: <https://it.mathworks.com/help/signal/ref/hann.html> (cit. on p. 21).
- [33] *Numpy.Hanning - NumPy v1.24*. URL: <https://numpy.org/doc/stable/reference/generated/numpy.hanning.html> (cit. on p. 25).
- [34] *Mel Scale - Wikipedia*. URL: https://en.wikipedia.org/wiki/Mel_scale (cit. on p. 21).
- [35] D. O’Shaughnessy. *Speech Communication: Human and Machine*. Addison-Wesley series in electrical engineering. Addison-Wesley Publishing Company, 1987. ISBN: 9780201165203. URL: <https://books.google.it/books?id=mHFQAAAAMAAJ> (cit. on p. 21).

-
- [36] D Gartzman. *etting to Know the Mel Spectrogram*. Medium. 2019. URL: <https://towardsdatascience.com/getting-to-know-the-mel-spectrogram-31bca3e2d9d0> (cit. on p. 21).
- [37] musikalkemist. *AudioSignalProcessingForML/17 - Mel Spectrogram Explained Easily at master*. 2020. URL: <https://github.com/musikalkemist/AudioSignalProcessingForML/tree/master/17%20-%20Mel%20Spectrogram%20Explained%20Easily> (cit. on p. 21).
- [38] *Mel Filter Bank — PyFilterbank devN documentation*. 2020. URL: <https://sigfigue.github.io/pyfilterbank/melbank.html> (cit. on p. 26).
- [39] Gondhalekar. *Data Augmentation – Is it really necessary?* Medium. 2020. URL: <https://medium.com/@RaghavPrabhu/understanding-of-convolutional-1-neural-network-cnn-deep-learning-99760835f148> (cit. on p. 22).
- [40] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. «SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition». In: *Interspeech 2019*. ISCA, Sept. 2019. DOI: 10.21437/interspeech.2019-2680. URL: <https://doi.org/10.21437%2Finterspeech.2019-2680> (cit. on pp. 22, 27, 38).
- [41] William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. *Listen, Attend and Spell*. 2015. arXiv: 1508.01211 [cs.CL] (cit. on p. 22).
- [42] N. Barazida. *40 Open-Source Audio Datasets for ML*. Medium. 2021. URL: <https://towardsdatascience.com/40-open-source-audio-datasets-for-ml-59dc39d48f06> (cit. on p. 28).
- [43] Karol J. Piczak. «ESC: Dataset for Environmental Sound Classification». In: *Proceedings of the 23rd Annual ACM Conference on Multimedia*. Brisbane, Australia: ACM Press, Oct. 13, 2015, pp. 1015–1018. ISBN: 978-1-4503-3459-4. DOI: 10.1145/2733373.2806390. URL: <http://dl.acm.org/citation.cfm?doid=2733373.2806390> (cit. on p. 28).
- [44] karolpiczak. *ESC-50*. URL: <https://github.com/karolpiczak/ESC-50> (cit. on pp. 28, 29).
- [45] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter. «Audio Set: An ontology and human-labeled dataset for audio events». In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017, pp. 776–780. DOI: 10.1109/ICASSP.2017.7952261 (cit. on pp. 30, 32, 35).

- [46] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. «ImageNet: A large-scale hierarchical image database». In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on pp. 30, 35).
- [47] Marti A. Hearst. «Automatic Acquisition of Hyponyms from Large Text Corpora». In: *COLING 1992 Volume 2: The 14th International Conference on Computational Linguistics*. 1992. URL: <https://aclanthology.org/C92-2082> (cit. on p. 30).
- [48] URL: <http://research.google.com/audioset/index.html> (cit. on p. 32).
- [49] Helin Wang, Yuexian Zou, Dading Chong, and Wenwu Wang. *Environmental Sound Classification with Parallel Temporal-spectral Attention*. 2020. arXiv: 1912.06808 [cs.SD] (cit. on pp. 31–33, 35, 36, 45).
- [50] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, and Mark D. Plumbley. *PANNs: Large-Scale Pretrained Audio Neural Networks for Audio Pattern Recognition*. 2020. arXiv: 1912.10211 [cs.SD] (cit. on pp. 31, 32, 34).
- [51] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. *MnasNet: Platform-Aware Neural Architecture Search for Mobile*. 2019. arXiv: 1807.11626 [cs.CV] (cit. on pp. 31, 40).
- [52] Ke Chen, Xingjian Du, Bilei Zhu, Zejun Ma, Taylor Berg-Kirkpatrick, and Shlomo Dubnov. *HTS-AT: A Hierarchical Token-Semantic Audio Transformer for Sound Classification and Detection*. 2022. arXiv: 2202.00874 [cs.SD] (cit. on pp. 31, 42, 44).
- [53] *Papers with Code - The latest in Machine Learning*. URL: <https://paperswithcode.com/> (cit. on pp. 31, 33, 34).
- [54] hadryan. *TFNet-for-Environmental-Sound-Classification*. 2019. URL: <https://github.com/Hadryan/TFNet-for-Environmental-Sound-Classification/tree/db5008a48e66e7272263434244c07d3daa253794> (cit. on pp. 32, 36, 45, 46).
- [55] Florian Schmid, Khaled Koutini, and Gerhard Widmer. *Efficient Large-scale Audio Tagging via Transformer-to-CNN Knowledge Distillation*. 2023. arXiv: 2211.04772 [cs.SD] (cit. on pp. 35, 42).
- [56] Yuan Gong, Yu-An Chung, and James Glass. *AST: Audio Spectrogram Transformer*. 2021. arXiv: 2104.01778 [cs.SD] (cit. on pp. 35–37).
- [57] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. 2021. arXiv: 2010.11929 [cs.CV] (cit. on pp. 35–37).

- [58] Andrew Howard et al. *Searching for MobileNetV3*. 2019. arXiv: 1905.02244 [cs.CV] (cit. on pp. 35, 38, 40, 41).
- [59] Khaled Koutini, Jan Schlüter, Hamid Eghbal-zadeh, and Gerhard Widmer. «Efficient Training of Audio Transformers with Patchout». In: *Interspeech 2022*. ISCA, Sept. 2022. DOI: 10.21437/interspeech.2022-227. URL: <https://doi.org/10.21437%2Finterspeech.2022-227> (cit. on pp. 35, 37, 38).
- [60] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV] (cit. on p. 39).
- [61] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861 [cs.CV] (cit. on p. 39).
- [62] R. Haque. *MobileNet Model Information*. *Medium*. 2021. URL: <https://rafat97.medium.com/mobilenet-model-information-6701ca0cdc17> (cit. on p. 39).
- [63] R. Avenash and Prashanth Viswanath. «Semantic Segmentation of Satellite Images using a Modified CNN with Hard-Swish Activation Function». In: *VISIGRAPP*. 2019 (cit. on p. 41).
- [64] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531 [stat.ML] (cit. on p. 41).
- [65] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. *Swin Transformer: Hierarchical Vision Transformer using Shifted Windows*. 2021. arXiv: 2103.14030 [cs.CV] (cit. on p. 42).
- [66] fschmid56. *EfficientAT*. URL: <https://github.com/fschmid56/efficientat> (cit. on pp. 45–47).
- [67] RetroCirce. *HTS-Audio-Transformer*. URL: <https://github.com/retrocirce/hts-audio-transformer> (cit. on p. 45).
- [68] *Spectral density - Wikipedia*. 2022. URL: https://en.wikipedia.org/wiki/Spectral_density (cit. on p. 53).
- [69] R.G. Brown and P.Y.C. Hwang. *Introduction to Random Signals and Applied Kalman Filtering with Matlab Exercises and Solutions*. Introduction to Random Signals and Applied Kalman Filtering: With MATLAB Exercises and Solutions. Wiley, 1997. ISBN: 9780471128397. URL: <https://books.google.it/books?id=De9SAAAAMAAJ> (cit. on p. 54).

- [70] Sanyuan Chen, Yu Wu, Chengyi Wang, Shujie Liu, Daniel Tompkins, Zhuo Chen, and Furu Wei. *BEATs: Audio Pre-Training with Acoustic Tokenizers*. 2022. arXiv: 2212.09058 [eess.AS] (cit. on p. 74).