Urban Sound Classification using Neural Networks

Giuseppe Scaffidi Caruso, Andrew Thomas Costa

May 2022

Sommario

In the last few years, one of the most prevalent topics concerning machine learning application is Environmental Sound Classification (ESC). The fields of application for ESC are in abundance, with notable examples like "Shazam" or "Alexa". The interest around all that concerns "smart speaking" is inviting the attention of market leaders such as "Google, Amazon and Facebook" and with this attention, focused research and investment. An important feature of this type of learning is the upgrading of their performance based on the live, constant learning from repetitive human habits.

The aim of this project is to exploit learning algorithm such as Neural Networks architecture, showing the most interesting result from 2 type of algorithm as Feed Forward Neural Network and Convolutional Neural Networks

Indice

Elenco delle figure

Elenco delle tabelle

1 Goal of the analysis

1.1 Benchmarks Model

This paper will explore two types of algorithms – a Multi-layer Perceptron and a Convolutional Neural Network – in order to make predictions on the Urban Sound 8K dataset. The aforementioned models will be compared to other types of learning algorithms, such as **SVM**, **Random Forest**, **ibk-5**[1], **j48**[2] and **ZeroR**.The models proposed are given by the paper "A Dataset and Taxonomy for Urban Sound Research" (Salamon, 2014)".

Algorithm	**Classification Accuracy**
SVM$_r bf$	68%
RandomForest500	66%
IBk5	55%
j48	48%
ZeroR	10%

Tabella 1: Algorithm Comparison

2 Analysis

2.1 Dataset Description

The UrbanSound8K dataset contains 8732 labelled slices of audio clips with varying sample rate and length. The audio clips, stored in wav. format, are roughly 4 seconds long with 10 different classes, those classes being:

Type	N°
dog bark	1000
children playing	1000
air conditioner	1000
street music	1000
engine idling	1000
jackhammer	1000
drilling	1000
siren	929
car horn	429
gun shot	374

Tabella 2: Data set Features

As we can see from the table, the majority of the classes are composed of 1000 clips, however there are outliers; to be precise the car horn with 37% and the gun shot with 43% less samples respectively.

[1]The IBk algorithm uses a distance measure to locate k "close" instances in the training data for each test instance and uses those selected instances to make a prediction

[2]J48 is a machine learning decision tree classification algorithm based on Iterative Dichotomiser 3

Air conditioner

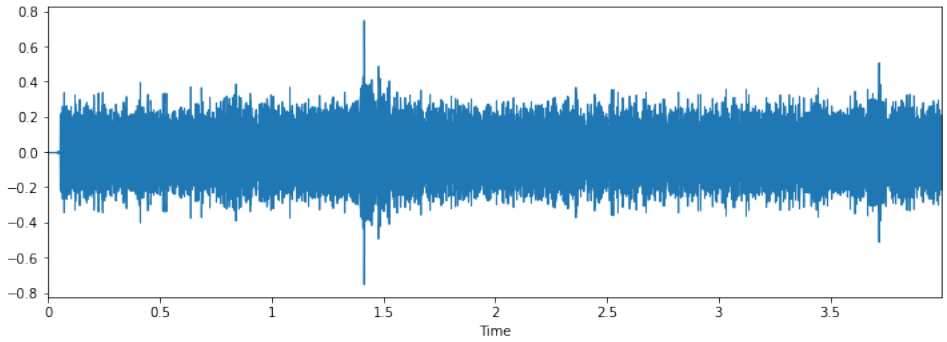

Figura 1: Air conditioner Wave plot

Car Horn

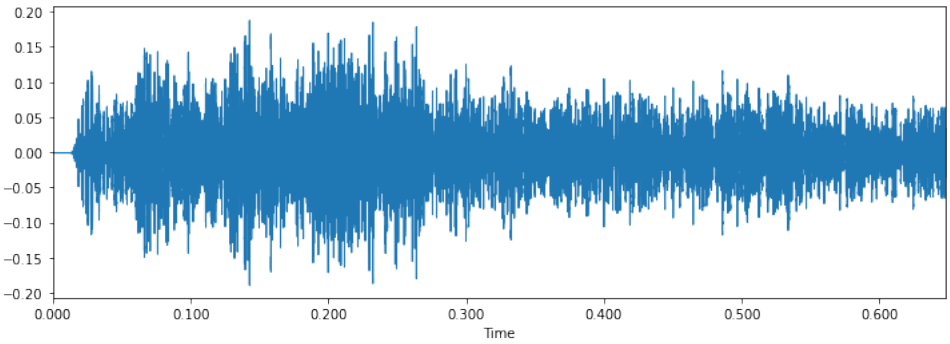

Figura 2: Car Horn Wave plot

Children Playing

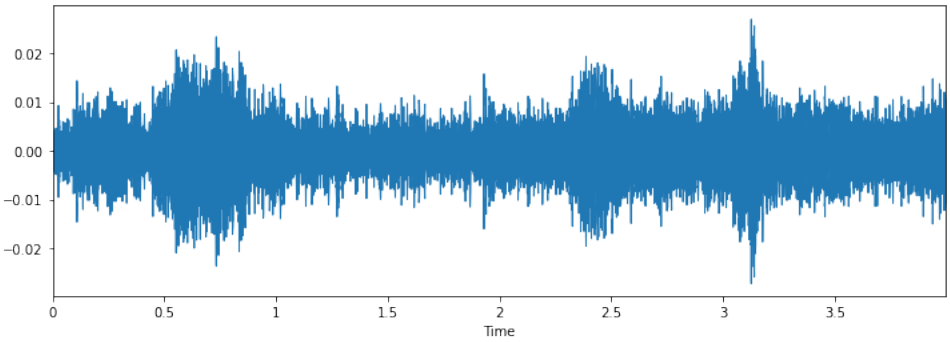

Figura 3: Children Playing Wave plot

Dog Bark

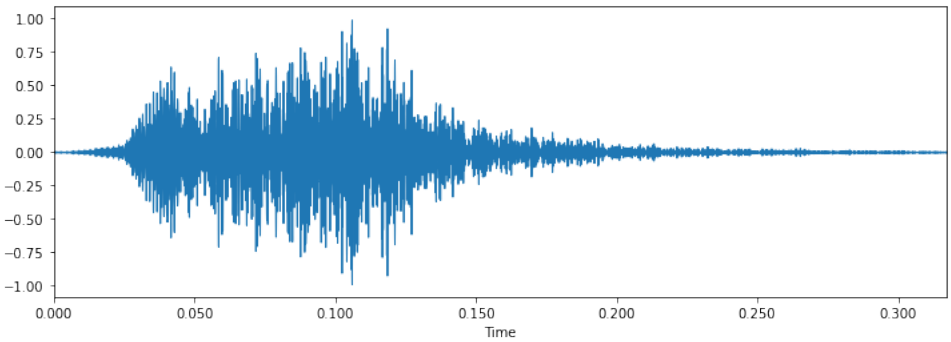

Figura 4: Dog Bark Wave plot

Drilling

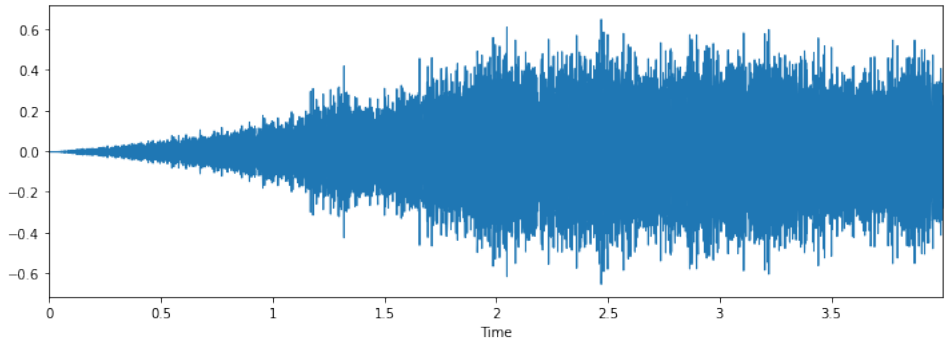

Figura 5: Drilling Wave plot

Engine Idling

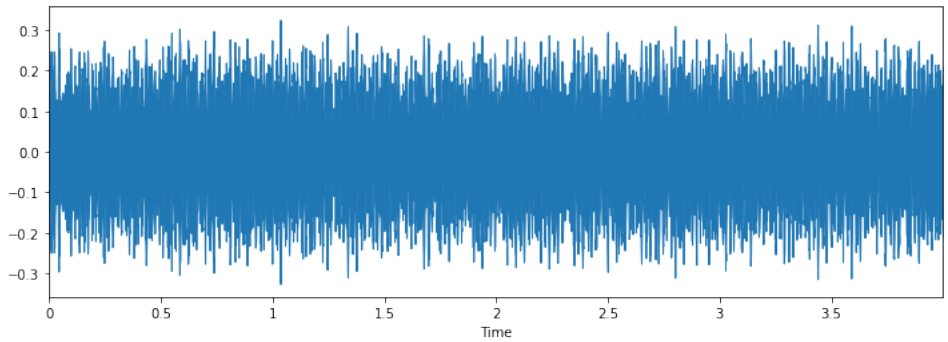

Figura 6: Engine Idling Wave plot

2.2 Audio Channels

Audio that comes from the environment must be translated in digital form so that it can be analysed and executed by the machine learning model. The computer, exploiting the natural audio wave, is able to recognize some patterns, like frequency and amplitude, so that the learning algorithm can receive the data in an appropriate format. This digital transformation is called sampling due to the action of taking sample snapshots from a wave process at a particular time from the wave audio and recording their amplitude. Each snapshots is made as a single number, with two parameters determining the quality of them:

- Sample rate
- Bit Depth

2.3 Sample Rate

After the data exploration, the next step will be to define the features of the plotted graph. The mechanism permitting us to translate this wave into data that can be manipulated is the sampling-rate process. The sample rate is the number of samples (or snapshots) from the original signal the computer receives in a single second (discussed in Kilohertz). For clarity, the standard unit of measurement is 44.1 KHz, with this number referring to the number of snapshots (or samples) the computer must expect in every second of audio. The sample size can be determined in the following manner: Considering a simple audio wave, whose shape is a sine wave, the system has to be able to detect how long it takes to complete one cycle. One can define a cycle of any complete wave that contains a positive and a negative stage. The cycle starts from 0 db and has both positive and negative momentum until the wave converges, once more, to 0 db. For a digital system to accurately capture and play back the pitch, one needs to measure the wave at least twice per cycle.
Additionally, it is generally accepted that human hearing range goes up to around 20 Kilohertz, so it stands that it would need to capture the audio until at least 40 kilohertz. This procedure allows the model to work with audio that goes all the way up to the highest frequency that humans can perceive. Just to be clear, if a signal is above that which can be handled by the sample rate i.e. greater than 40 kilohertz, it will not be interpreted properly in the audio system and will be mirrored back onto the existing signal, introducing new frequency information in a phenomenon called aliasing. To prevent aliasing, the software implemented to convert and store digital audio will use a low pass filter to eliminate frequency above 40 kilohertz so that the system can handle the sample rate properly. This process will prevent unwanted high frequency in the model and the reason why the 44.1 KHz is used is because it allows the low pass filter to have a gentler slope on the graph.

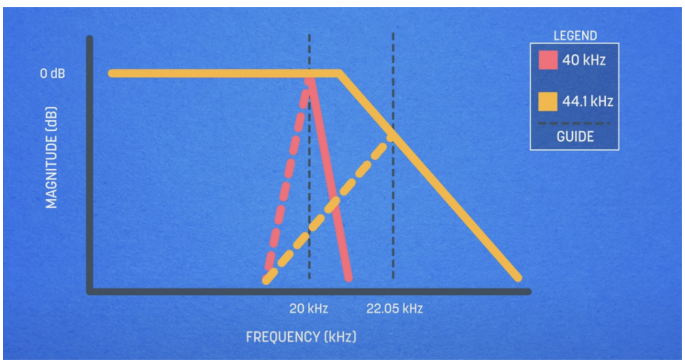

Figura 7: Difference between 44.1 KHz and 40 KHz

2.4 Bit Depth

The bit depth determines the amount of possible amplitude values one can record for each sample (N.B. amplitude is often referred to as level).

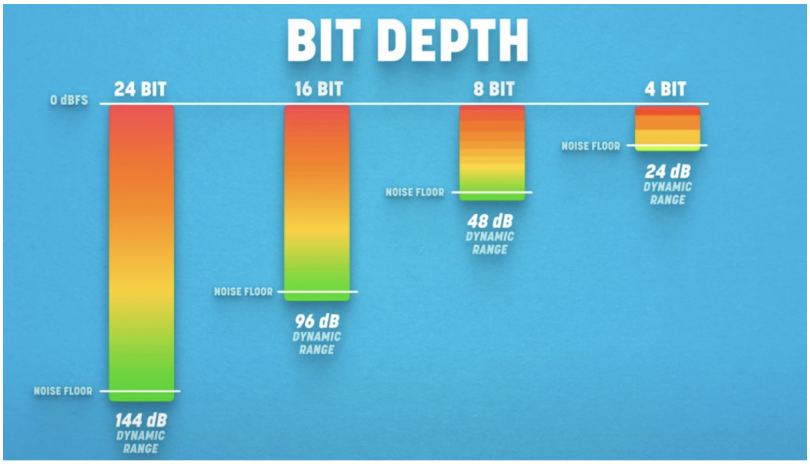

Figura 8: Difference of bit captured per db

In practice, if the algorithm is set to record at 24 bits, then it can take the snapshots of the incoming audio with more accuracy than if it were set to 16 bits, the way in which it captures parameters increases in accuracy. The difference between different sizes of bits can be about a million of values. The concept can be detailed as: the higher the bit depth, the greater the detail in the audio. Using the image storing the way in which bit depth acts can be more clear. The assumption made is: the higher the bit depth of an image, the more colours it can store. A simple image composed of 1 bit of information (1 maps bit image) can only show 2 colours, black and white. This happens because the 1 bit can only store one of the two values either 0, black or 1, white. Whilst increasing the number of bits one can have an 8 bit image that can store 256 colours or a 24 bit image that can display 16 million colours. As the bit depth increases, the file size of the image also increases because more colour information has to be stored for each pixel in the image.

2.5 Mel Frequency Cepstral Coefficients (MFCCs)

A Mel Frequency Cepstrum captures the short-term spectrum of sound. The coefficients are derived from a cepstral representation of an audio clip, here, one of the many urban sounds found within our dataset. A distinction between Mel-frequency cepstrum (MFC) and a cepstrum is needed to better understand the MFC's primary function, which is as follows: the MFC has frequency bands that are equally spaced on the mel scale, which is a way to approximate the range of human hearing more accurately than the linearly-spaced frequency bands used in a normal spectrum. By doing so, sound can be better represented for tasks like audio compression and audio classification.

The steps to derive the MFCCs are a follows:

- Perform a fourier transform on the window of a signal. This is a signal in the time domain:

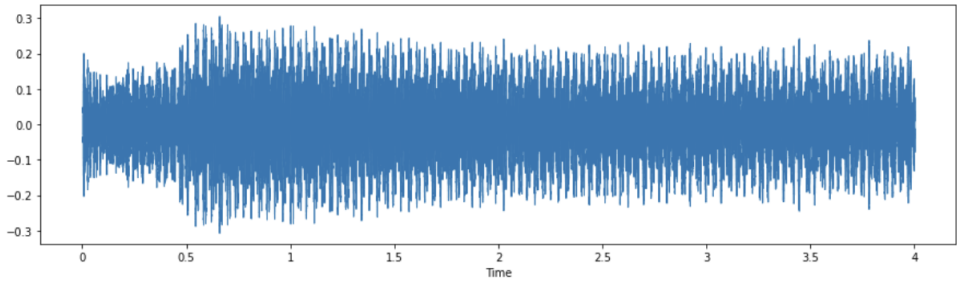

Figura 9: Graph represented the signal in **Time Domain**

After performing a Fourier transform on the signal above, we move from the time-domain to the frequency domain. The graph below shows the magnitude of the frequencies present in the signal:

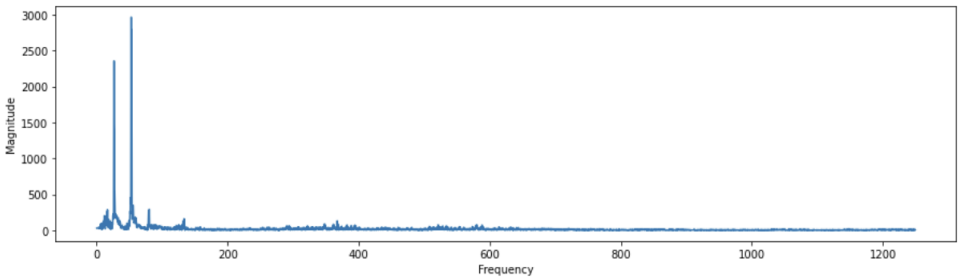

Figura 10: Graph represented the signal in **Frequency Domain**

- Link the powers of the spectrum obtained in the Fourier transform onto a *mel scale*

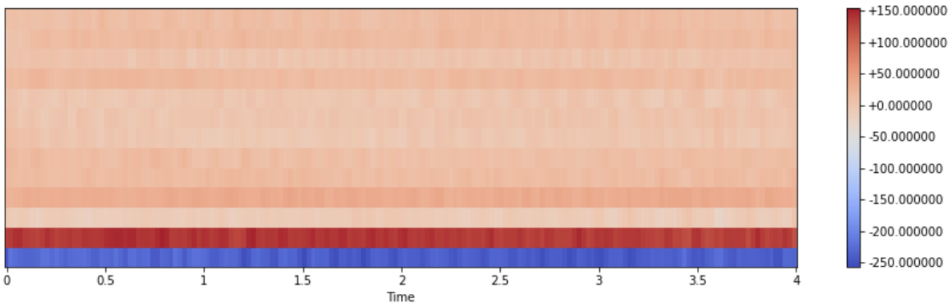

Figura 11: Audio Feature plotted on *mel scale*

- Log-transform the powers at each *mel frequency*
- Take the discrete cosine transform of the list of mel log powers
- the outputs are amplitudes, which are the MFCCs

The idea of MFCCs is to convert time-domain signals into frequency-domain signals, using Mel filters to mimic cochlea that has more filters at low frequency and fewer filters at high frequency. Thus, the features and characteristics of the coefficients obtained are focused on the audibility of the human hearing system, accommodating the dynamic nature of true-life sounds, in a vectorized format.

3 MLP algorithm Theory

The first model that will be used is called **Multi Layer Perceptron**. The introduction of this model comes after many experiments on neural networks. The new concept implemented on the **neural networks** was developed by Rosenblatt, who developed on the work of McCulloch's and Pitts' neural network.
The MLP based on **Perceptron** which is widely recognized as an algorithm, its name derives from perception which is the ability of the human being to recognize many patterns in a specific field of application through hearing or seeing. The theoretical parts implemented by Rosenblatt relied on the **Neuron structure** even if the greatest difference was the addition of the weights. These weights were combined with the inputs in a weighted sum and if the sum exceeds a predefined threshold, the neuron fires and produces output. From this theoretical introduction it can be said how the **M**ultilayer**L**ayer**P**erceptron is is a special case of a feed forward neural network where every layer is a fully connected layer, and in some definitions the number of nodes in each layer is the same

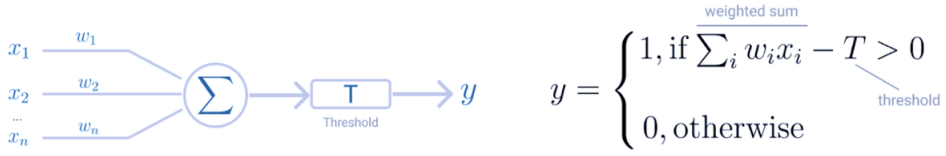

Figura 12: MLP Neural Network layout

In this case the threshold **" T "** represents the **activation function**. As it can be seen from the picture, the **weighted sum** establishes if the neuron outputs **" y "** will be 1 or 0.

3.1 Activation Function

The activation function chosen in the model is the **ReLU**. This type of activation function is a piece-wise linear function that will output the input directly if it is positive, otherwise, it will output zero. The rectified linear activation function is a simple calculation that returns the value provided as input directly, or the value 0.0 if the input is 0.0 or less.
This type of **activation function** is linear when for values greater than zero, this allows the function to exploit properties of the linear activation function when we are training the algorithm via back propagation. Yet, it is a nonlinear function as negative values are always output as zero.

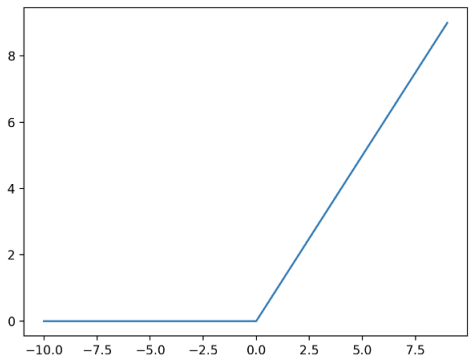

Figura 13: Plot of ReLU Activation Function

The **M**ultilayer**L**ayer**P**erceptron model is given by a neuron that receives inputs and picks a set of random weights initially. These weights are combined in weighted sum and then ReLU activation function determines the value of the output.
After we set the random weights, the model will need to re-balance these weights through learning process. The **MLP** uses stochastic gradient descent to learn.
The *stochastic gradient descent* is an optimization procedure in which on one side there is the *"gradient"* part which refers to the calculation of the slope error while on the other side there is the *"descent"* part whose aim is to move down along slope error to minimise it. The *compilation model* parameter in the model are:

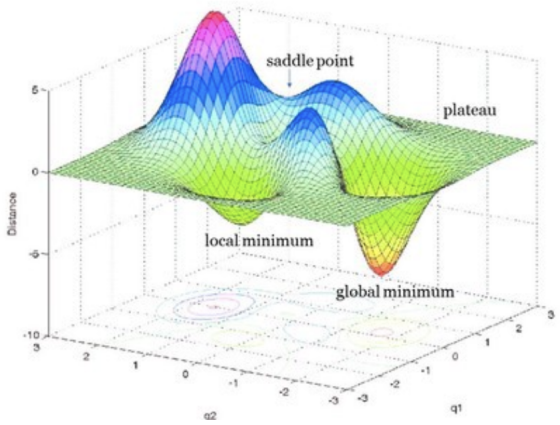

Figura 14: How the process of minimization works

– **Metrics**
Enables the algorithm to evaluate the performance of the model. **Accura-**

cy and the **AUC** (Area under the curve) of the ROC (Receiver operating characteristic. The latter one is also called **precision recall**

- **Optimizer**
  Optimization is an important process that helps the algorithm to optimise the input weights. It also allows algorithms to compare the prediction and the loss function. Adam is the optimizer used for the purpose of the analysis.
- **Loss**
  It enables the algorithm to find errors or deviations in the learning process. In the project we use **categorical cross entropy**.

3.2 Hyper Parameter Tuning on MLP

In order to choose a certain number of parameters to put inside the models the approach is driven by the tuning of parameters through *keras tuner*. As we know, to choose the Hyper Parameter itself we have to run a specific learning algorithm for it.
The parameter given to the learning algorithm of the Hyper Parameter are:

Min n° nodes = 50
Max n° nodes = 500
Min n° of layers= 2
Max n° of layers= 6

The parameters evaluation is given by the package that computes the best layout on specific metrics. In this case the metric is the accuracy

- **N° Nodes**
  Among a number of nodes between 50 and 500 the best The best number of nodes chosen by the learning algorithm is 260 nodes each layer.

```
Search: Running Trial #2

Value             |Best Value So Far |Hyperparameter
260               |260               |input_units
0.0001            |0.01              |learning_rate
```

Figura 15: Output table nodes

From the Hyper Parameter result we're gonna train and test the *1st* MLP with exactly this number of nodes except the output node that must include 10 nodes (it has been chosen the number of outputs the algorithm must give us[3]).

- **N° Layers**
  In the tuning of hyperparameters one has to run an optimization process aiming to discover the best number of layers and nodes to put inside the **MLP** . The iteration, run through a for-loop, scans the range of nodes (min= 50 , max=500), the number of layers (min= 2 , max=6) and the type best type of activation function (*"relu"*, *"sigmoid"* , *"silu"*) one could have in the layout.

[3]Disclaimer: The number of nodes generated by the algorithm could vary every notebook's run-time, the number chosen indicate the average value of nodes with which the performance is acceptable

3.3 1st MLP: One Hidden layer

Initially the number of layers chosen for the algorithm was 3, which is the minimum number of layers the model could have, otherwise if the model had only 2 layers it would be a linear function. The number of hyperparameters (such as n° nodes and n° layers) in the 1st algorithm are chosen by tuning the hyperparameters nodes. This first trial corresponded exactly to the number of nodes equal to 260 gained from the hyperparameter tuning estimate
The number of layers of this neural networks are 3, each of them with these features:

- **Layer 1** : this one represents the input layer, the columns are represented by 40 MFCCs. The nodes inside the networks are 256 which work with a *"relu"* activation function. The regularisation method applied is the dropout.
- **Layer 2** : this one represents the hidden layer where the numbers of nodes are 256 and the activation function used is the *"relu"* . The regularisation method applied is the dropout.
- **Layer 3** : this one represents the output layer where the number of elements inside will be equal to the number of sounds class, 10 (number of labels).

By computing the neural networks with this specific pre-compiled model, we get poor results in terms of its performance. The main problem from this model application came from the power deployed, which is too much compared to the size of the training set. This phenomenon has led to over fitting on the training set and poor performance, in terms of power prediction, on the test set. From now on the model will experience such this problem of having greater capacity than needed.

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_14 (Dense)             (None, 256)               10496

activation_14 (Activation)   (None, 256)               0

dropout_12 (Dropout)         (None, 256)               0

dense_15 (Dense)             (None, 256)               65792

activation_15 (Activation)   (None, 256)               0

dropout_13 (Dropout)         (None, 256)               0

dense_16 (Dense)             (None, 10)                2570

activation_16 (Activation)   (None, 10)                0

=================================================================
Total params: 78,858
Trainable params: 78,858
Non-trainable params: 0
_________________________________________________________________
```

Figura 16: Output Model 1

The *Pre-Training Accuracy* scored from the model is 11.803%. The fitted model on the validation data with n° *epochs* = 100 and n° *batch-size* = 32 has a *training accuracy* = 0.81% and a *test accuracy* = 0.62% . These values of accuracy have been obtained with 100 n° *epochs* and 32 n° *batch-size*.

Taking a look to the **Loss per Epochs** and the **Accuracy per Epochs**

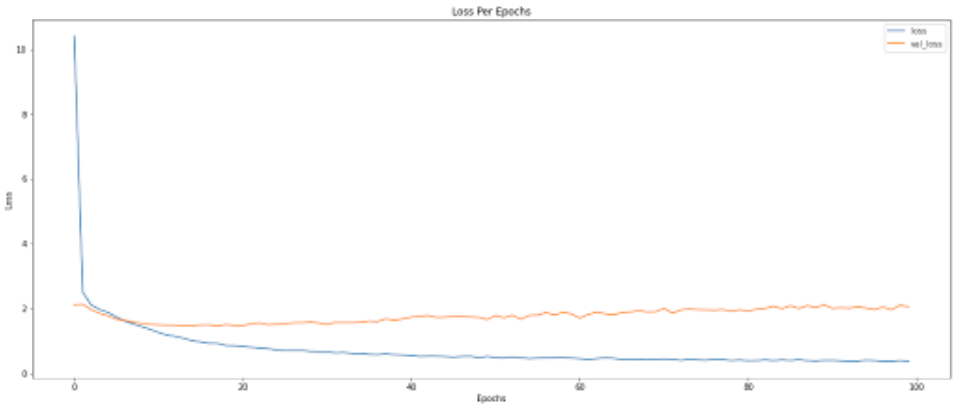

Figura 17: Loss per Epochs Model 1

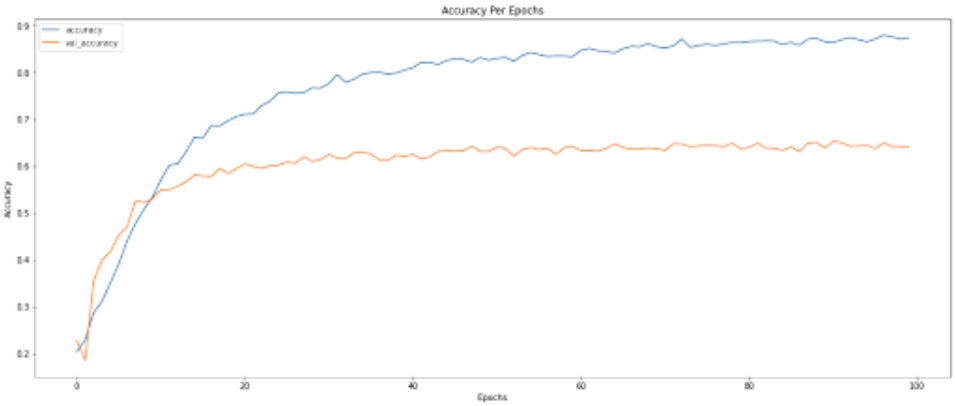

Figura 18: Accuracy per Epochs Model 1

From the *1st graph* it's clear as the *n° of epochs* goes by the loss of the model decreases in both training and test[4]. From the *2nd graph* it can be noticed how the test accuracy fluctuates on the same value value after 10 *n° of epochs*. Inside the model other methods of **regularisation** (such as early stopping or l1/l2) have been applied in order to control over fitting but no one of this worked well.

[4]*for the sake of notation remember how the val accuracy indicates the Test Accuracy

3.3.1 Confusion Matrix

Taking a look to the confusion matrix and some other metrics (precision-recall-f1 score), it can be shown how the model is able to predict well. Observing the main diagonal it can be seen how all the predicted and real values are correctly classified while outside from the main diagonal there are the values incorrectly classified.

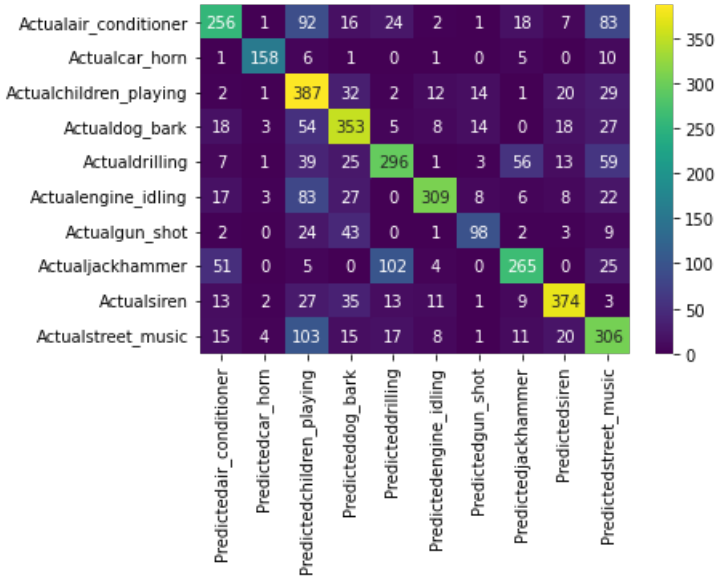

Figura 19: Confusion Matrix Model 1

The performance of the algorithm in terms of accuracy with the values suggested by the tuned Hyper Parameter , highlights a poor performance of the algorithm on the accuracy test but a relatively good performance on the confusion matrix.
One more step could be implemented inside the model is to change and custom the metrics. From the point of view of the metrics: Accuracy there is no possibilities to increase the number accuracy test per epochs and the prediction from confusion matrix.

- Macro Average **accuracy** = 0.70
- Macro Average **Recall** = 0.66
- Macro Average **f1-score** = 0.67

3.4 2nd MLP: Two hidden layers AUC metrics

In this second model implemented, the analysis will be focused on one other type of *metrics* = AUC.
The best performance gained on neural with **AUC** is given by this layout: The number of layers in this neural networks are 4 , each of them with these features:

- **Layer 1** : this one represents the **input layer**, the columns are represented by 40 MFCCs. The nodes inside the networks are 70 which work with a "Relu" activation function. The regularisation method applied is the dropout.
- **Layer 2** : this one represents the **hidden layer** where the numbers of nodes are 70 and the activation function used is the "Relu" . The regularisation method applied is the dropout.
- **Layer 3** : Layer 3 : this one represents the **hidden layer** where the numbers of nodes are 70 and the activation function used is the *"Relu"*. The regularisation method applied the dropout, in this particular case with value 0.3 to have low penalization.
- **Layer 4** : this one represents the **output layer** where the number of elements inside will be equal to the number of sounds class, 10 (number of labels).

Layer (type)	Output Shape	Param #
dense_15 (Dense)	(None, 70)	2870
activation_9 (Activation)	(None, 70)	0
dropout_6 (Dropout)	(None, 70)	0
dense_16 (Dense)	(None, 70)	4970
activation_10 (Activation)	(None, 70)	0
dropout_7 (Dropout)	(None, 70)	0
dense_17 (Dense)	(None, 70)	4970
activation_11 (Activation)	(None, 70)	0
dropout_8 (Dropout)	(None, 70)	0
dense_18 (Dense)	(None, 10)	710
activation_12 (Activation)	(None, 10)	0

Total params: 13,520
Trainable params: 13,520
Non-trainable params: 0

Figura 20: Output model 2

The pre-training accuracy equal to 52% scored through AUC metrics is the best score in all the trials of MLP. The *n°of epochs* are 10 and the *n° of batch size* are 32. After 10 epochs the algorithm tends to over fit on the training set. From model evaluation we can score the performances of both test and train AUC are 78% and 75% respectively.

Taking a look to the **Loss per Epochs** and the **AUC per Epochs**

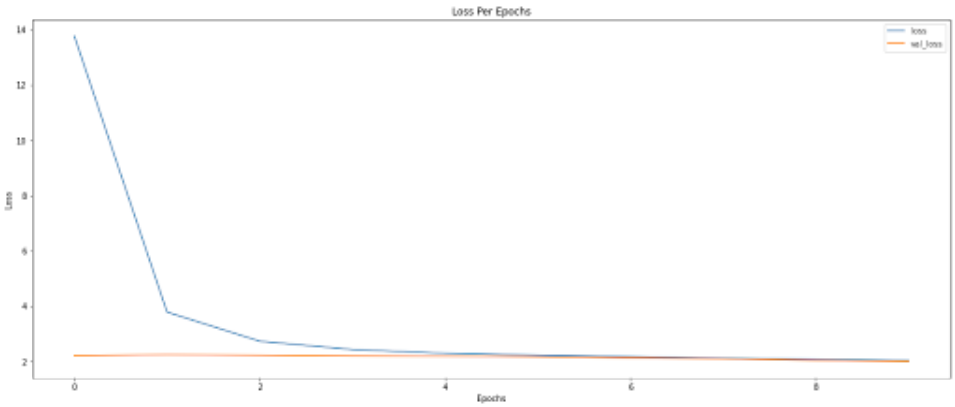

Figura 21: Loss per Epochs Model 2

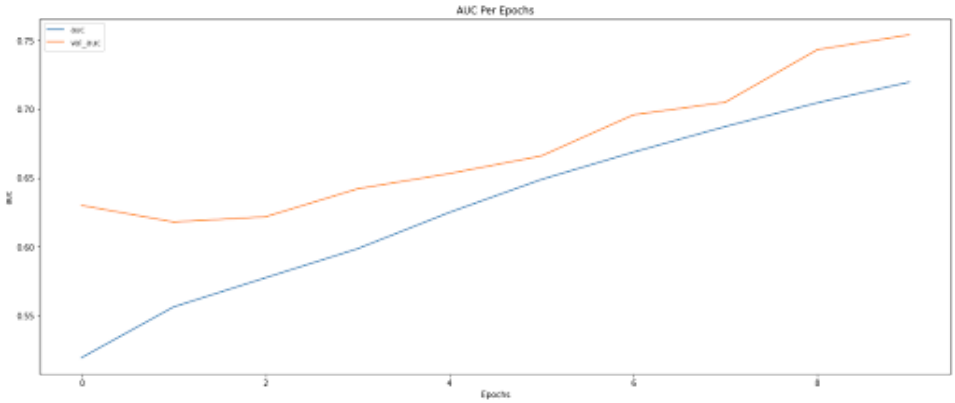

Figura 22: AUC per Epochs Model 2

The loss in the *1st graph* is at the same level and tends to be stable also when the number of epochs sharply increases. The loss in the *1st graph* decrease sharply in the first quadrant, remaining stable in the other quadrants. The plotted line tends to be stable also when the number of epochs sharply increases.
The increasing AUC value in the *2nd graph* indicates how the algorithm is able to increase its performance as the epochs increase. From this first analysis of the **AUC** value per Epochs the result can be interpreted as good result.

3.4.1 Confusion Matrix

Even if this metric ensures good numerical performance in terms of percentage (Training AUC and Test AUC) the main problem is the huge misclassification in terms of prediction.

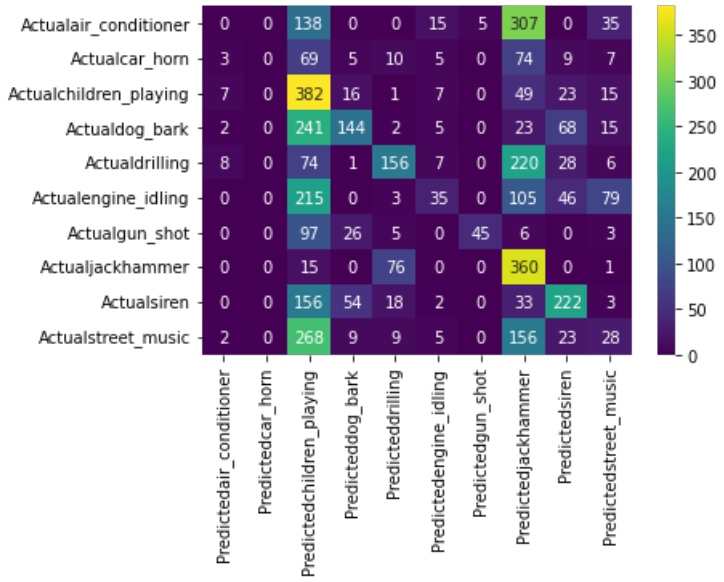

Figura 23: Confusion Matrix Model 2

The algorithm on average is able to output the most common audio class but isn't able to ensure the same performance in terms of output quality as the metric **Accuracy do**. Furthermore, the performance metrics such as precision, recall , f1-score are very low compared to the first approach.

- Macro Average **accuracy** = 0.36
- Macro Average **Recall** = 0.30
- Macro Average **f1-score** = 0.26

After we have discussed the main issue related to the AUC metrics in this specific problem the analysis will focus the attention on the *n° of nodes* and *n° of layers* suggested by the process of tuning the hyperparameter in the model.

3.5 3rd MLP : Four hidden layers

The layout in this 3rd model changes slightly from some differences such as the implementation of the sigmoid activation function. This change is also given by the customization of hyper-parameters; in fact, in the code, the algorithm is free to choose among 2 types of activation.

- **Layer 1** : this one represents the **input layer**, the columns are represented by 40 MFCCs. The nodes inside the networks are 402 which work with a *"sigmoid"* activation function. The regularisation method applied is the dropout.
- **Layer 2** : this one represents the **hidden layer** where the numbers of nodes are 338 and the activation function used is the "Relu" . The regularisation method applied is the dropout.
- **Layer 3** : Layer 3 : this one represents the **hidden layer** where the numbers of nodes are 50 and the activation function used is the *"Relu"*. The regularisation method applied the dropout, in this particular case with value 0.3 to have low penalization.
- **Layer 4** : this one represents the **output layer** where the number of elements inside will be equal to the number of sounds class, 10 (number of labels).

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_24 (Dense)            (None, 402)               16482

 activation_20 (Activation)  (None, 402)               0

 dropout_13 (Dropout)        (None, 402)               0

 dense_25 (Dense)            (None, 338)               136214

 activation_21 (Activation)  (None, 338)               0

 dropout_14 (Dropout)        (None, 338)               0

 dense_26 (Dense)            (None, 50)                16950

 activation_22 (Activation)  (None, 50)                0

 dropout_15 (Dropout)        (None, 50)                0

 dense_27 (Dense)            (None, 10)                510

 activation_23 (Activation)  (None, 10)                0

=================================================================
Total params: 170,156
Trainable params: 170,156
Non-trainable params: 0
_________________________________________________________________
```

Figura 24: Output Model 3

As pre-training accuracy we score 11.24%. Also in this model the parameters related to *n° of epochs* and *n° of batch size* are given by a customization process in which the early stopping of the process allows the model to avoid over fitting. The parameters related to *training accuracy* and *test accuracy* are respectively 0.87% and 0.61%. As we can see also from these scores above the accuracy on the test seems to be poor to the detriment that values on confusion matrix guarantees good prediction.

Taking a look to the **Loss per Epochs** and the **Accuracy per Epochs**

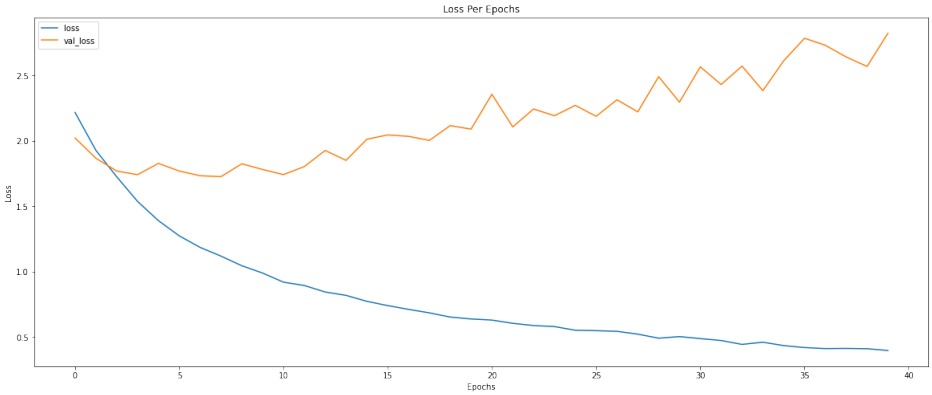

Figura 25: Loss per Epochs Model 3

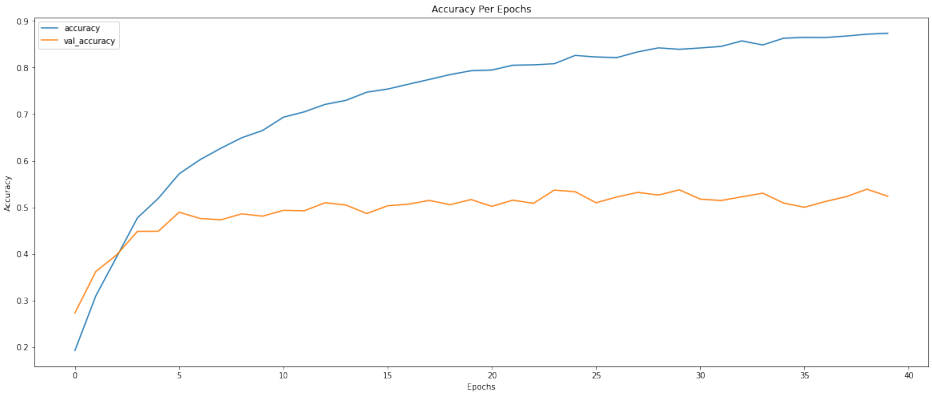

Figura 26: Accuracy per Epochs Model 3

The significant result in these graphs come from the first *loss per epochs* where the loss does not decrease over 20 epochs but instead increase. As we can see from the second graph the *accuracy per epochs* converges to its steady state after 5 epochs and remains constant over time.

3.5.1 Confusion Matrix

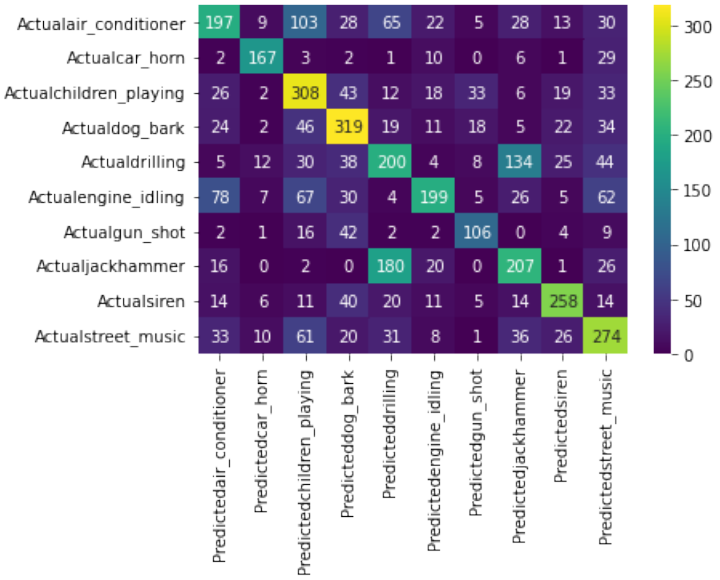

Figura 27: Confusion Matrix Model 3

The performance in terms of confusion matrix seems to be equal to the 1st MLP even if the score in this one is slightly worse than the previous 1st one.

- Macro Average **accuracy** = 0.65
- Macro Average **Recall** = 0.62
- Macro Average **f1-score** = 0.63

3.6 4th MLP : Five hiddens layers

The first-last layout result comes by the addition of one more layer suggested by the hyper parameter optimization. Remember that, the more the number of layers the more the complexity of the model.
The printed layout looks like:

- **Layer 1** : this one represents the **input layer**, the columns are represented by 40 MFCCs. The nodes inside the networks are 146 which work with a *"sigmoid"* activation function. The regularisation method applied is the dropout.
- **Layer 2** : this one represents the **hidden layer** where the numbers of nodes are 82 and the activation function used is the *"Relu"* . The regularisation method applied is the dropout.
- **Layer 3** : this one represents the **hidden layer** where the numbers of nodes are 210 and the activation function used is the *"Relu"*. The regularisation method applied the dropout, in this particular case with value 0.3 to have low penalization.
- **Layer 4** : this one represents the **hidden layer** where the numbers of nodes are 338 and the activation function used is the *"Sigmoid"*. The regularisation method applied the dropout, in this particular case with value 0.3 to have low penalization.
- **Layer 5** : this one represents the **output layer** where the number of elements inside will be equal to the number of sounds class, 10 (number of labels).

```
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_28 (Dense)            (None, 146)               5986

 activation_24 (Activation)  (None, 146)               0

 dropout_16 (Dropout)        (None, 146)               0

 dense_29 (Dense)            (None, 82)                12054

 activation_25 (Activation)  (None, 82)                0

 dropout_17 (Dropout)        (None, 82)                0

 dense_30 (Dense)            (None, 210)               17430

 activation_26 (Activation)  (None, 210)               0

 dropout_18 (Dropout)        (None, 210)               0

 dense_31 (Dense)            (None, 338)               71318

 activation_27 (Activation)  (None, 338)               0

 dropout_19 (Dropout)        (None, 338)               0

 dense_32 (Dense)            (None, 10)                3390

 activation_28 (Activation)  (None, 10)                0

=================================================================
Total params: 110,178
Trainable params: 110,178
Non-trainable params: 0
_________________________________________________________________
```

Figura 28: Output model 3

The pre training accuracy score is 10%. The values of training accuracy and test accuracy are respectively 0.85% and 0.59%.

Taking a look to the **Loss per Epochs** and the **Accuracy per Epochs**

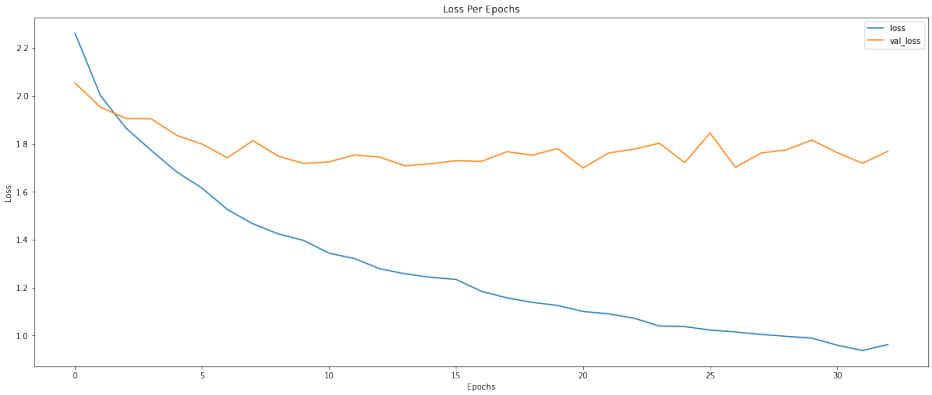

Figura 29: Loss per Epochs Model 4

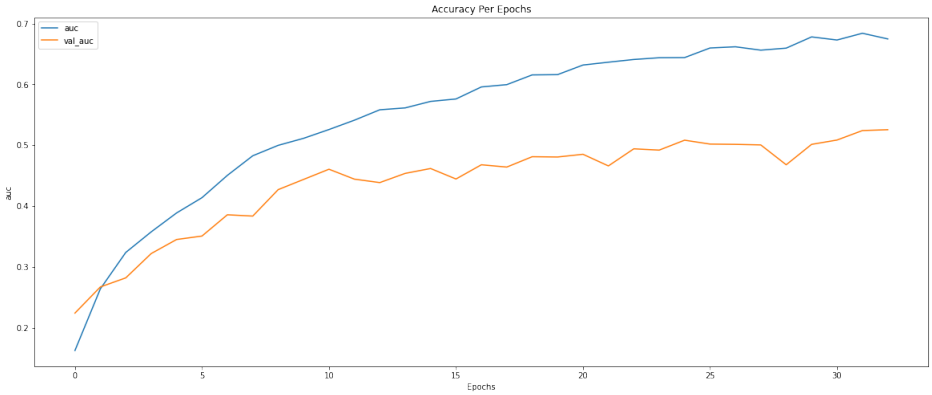

Figura 30: Accuracy per Epochs Model 4

The lines inside both graphs are more or less the same, the convergence rate of **test accuracy** is reached at the same number of epochs of the models plotted above. This means that the algorithm after a certain threshold is not able to maximise its own performance even if the input parameter change.

3.6.1 Confusion Matrix

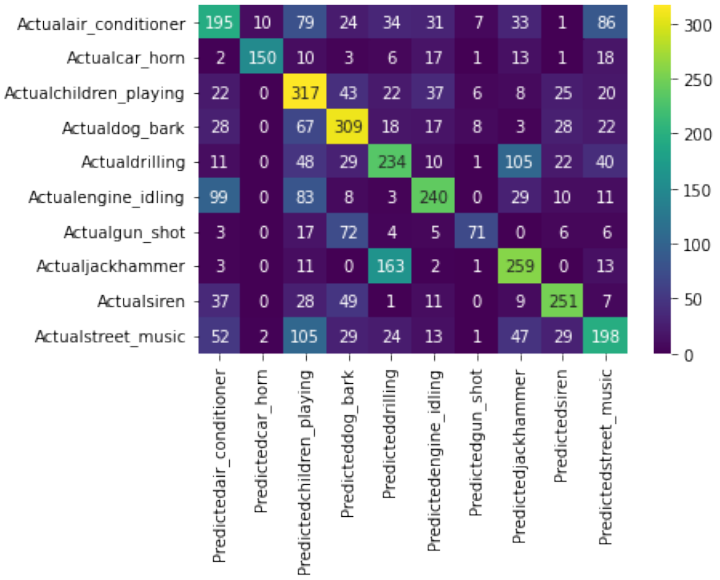

Figura 31: Confusion Matrix Model 4

The values from the confusion matrix are almost the same as the 1st and 3rd model.

- Macro Average **accuracy** = 0.66
- Macro Average **Recall** = 0.59
- Macro Average **f1-score** = 0.61

3.7 5th MLP : Six hiddens layers

For this last model suggested by the optimization algorithm provided by keras tuner we're going to evaluate the same number of layers but this time, with different configuration of nodes. Recalling that the percentage accuracy of this model provided by the optimization table is the one of the lowest.
The printed layout looks like:

- **Layer 1** : this one represents the **input layer**, the columns are represented by 40 MFCCs. The nodes inside the networks are 114 which work with a *"relu"* activation function. The regularisation method applied is the dropout.
- **Layer 2** : this one represents the **hidden layer** where the numbers of nodes are 146 and the activation function used is the *"sigmoid"* . The regularisation method applied is the dropout.
- **Layer 3** : this one represents the **hidden layer** where the numbers of nodes are 242 and the activation function used is the *"sigmoid"*. The regularisation method applied the dropout, in this particular case with value 0.3 to have low penalization.
- **Layer 4** : this one represents the **hidden layer** where the numbers of nodes are 210 and the activation function used is the *"relu"*. The regularisation method applied the dropout, in this particular case with value 0.3 to have low penalization.
- **Layer 5** : this one represents the **output layer** where the number of elements inside will be equal to the number of sounds class, 10 (number of labels).

```
Model: "sequential_10"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_33 (Dense)            (None, 146)               5986

 activation_29 (Activation)  (None, 146)               0

 dropout_20 (Dropout)        (None, 146)               0

 dense_34 (Dense)            (None, 434)               63798

 activation_30 (Activation)  (None, 434)               0

 dropout_21 (Dropout)        (None, 434)               0

 dense_35 (Dense)            (None, 178)               77430

 activation_31 (Activation)  (None, 178)               0

 dropout_22 (Dropout)        (None, 178)               0

 dense_36 (Dense)            (None, 402)               71958

 activation_32 (Activation)  (None, 402)               0

 dropout_23 (Dropout)        (None, 402)               0

 dense_37 (Dense)            (None, 10)                4030

 activation_33 (Activation)  (None, 10)                0

=================================================================
Total params: 223,202
Trainable params: 223,202
Non-trainable params: 0
_________________________________________________________________
```

Figura 32: Output model 5

The pre training accuracy for this model is 11The *training accuracy* score, with 70 n° of epochs and 32 n° of batch size, is 0.84% while the *test accuracy* is 0.58%.

Taking a look to the **Loss per Epochs** and the **Accuracy per Epochs**

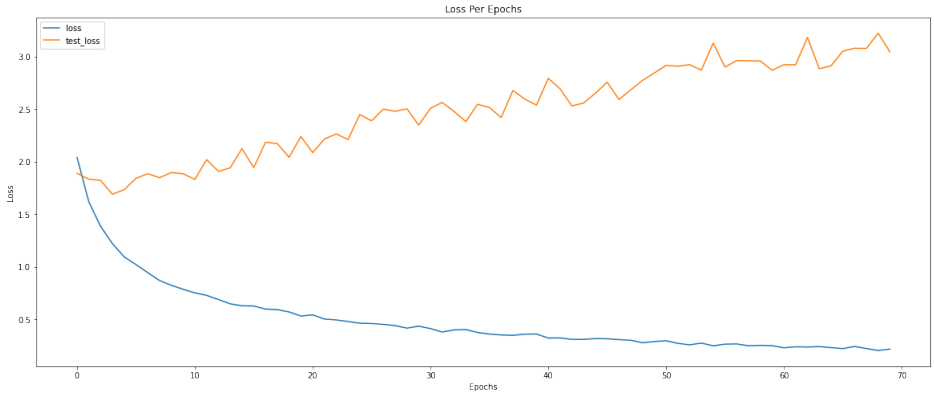

Figura 33: Loss per Epochs Model 5

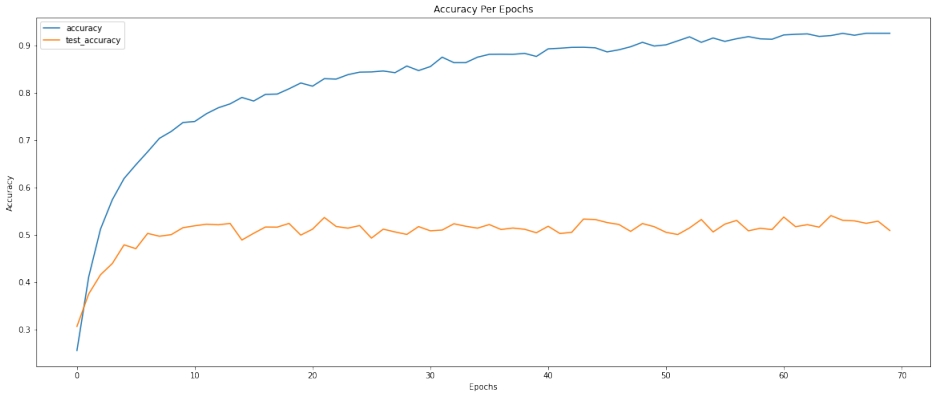

Figura 34: Accuracy per Epochs Model 5

By looking at the data plotted line inside this configuration, it can be seen how the *loss per epochs* sharply increase, getting worse. As it has already shown in the other layout the accuracy reach its steady state immediately after 5 *n° of epochs*

3.7.1 Confusion Matrix

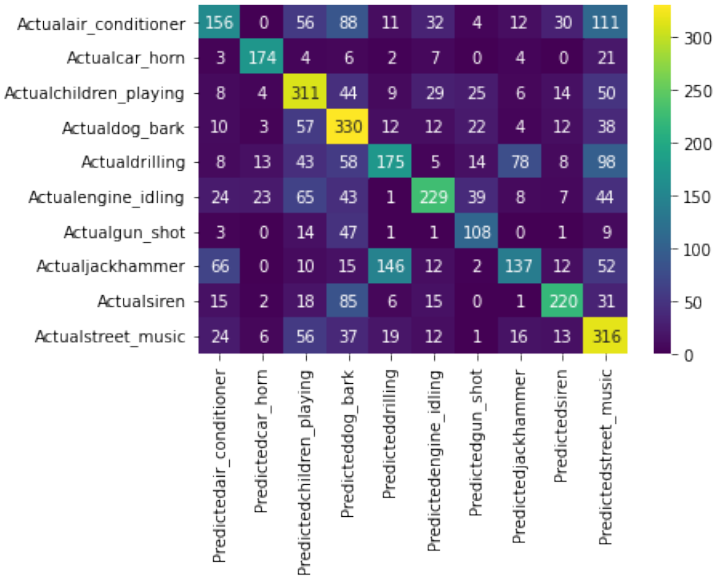

Figura 35: Confusion Matrix Model 5

- Macro Average **accuracy** = 0.54
- Macro Average **Recall** = 0.53
- Macro Average **f1-score** = 0.52

3.8 Prediction

The result of the whole process computed in the MLP model highlights how the scored prediction from *1st* to *5th* are reliable. Notice how the prediction model from some specific layouts tend to predict wrong, especially in certain class where the confusion matrix highlights a misclassification problem.

```
def predict5(path):
    audio = np.array([features_extractor(path)])
    classid = np.argmax(model5.predict(audio)[0])
    print('Class predicted :',classes[classid][0],'\n\n')
    return ipd.Audio(path)

# Class: Engine Idling
#The audio is referred to Engine Idling

predict5('/content/drive/MyDrive/ML Project/UrbanSound8K/audio/fold10/102857-5-0-0.wav')
```

4 Convolutional Neural Network

4.1 What is a CNN?

A Convolutional neural network stands out for its superior performance for certain tasks such as image and audio recognition as it is able to handle non-linear data, whilst also possessing the ability to understand data features with little human oversight.
The convolutional layer is the first layer of a convolutional network. While convolutional layers can be followed by additional convolutional layers or pooling layers, the fully-connected layer is the final layer.
The convolution between two functions,f ,g:

$$R^d \to R \tag{1}$$

is defined as:

$$(f \times g)(x) = \int f(z)g(x-z)\ dx$$

This process measures the overlap between two f and g when one function is "flipped" and shifted by x.
The aim of a CNN and the reason behind its success with image data falls down to its ability to systematize this idea of spatial invariance. Spatial invariance is the concept that a network can detect features/objects even if they are not accurately represented in the training set. So lets say we want to learn the characteristics of a spectrogram through the processing of its image structure, a CNN will learn this representation with few parameters.

4.2 Architecture

4.2.1 Activation Function

The activation function selected for this process was the 'ReLU'. At each layer in the model, an affine transformation is performed by applying the ReLU function, eliminating any negative values. After performing the ReLU operator a number of times, we are able to linearly separate the data

Figura 36: Affine Transformation of a ReLU operator showing how after each transformation, the values become non-negative.

The limitations of this ReLU operator are that if a bias, in one of the 5 layers, places a data point not in the top-right quadrant (i.e. not non-negative), then the ReLU will be under performing and everything will collapse to zero.
Other functions like the sigmoid cannot be used in a model with many layers due to the vanishing gradient problem.

4.2.2 Max pooling

The function of this layer is to reduce the complexity of the model whilst extracting local features. More specifically, it acquires the local features generated by convolving a filter over an image, here the spectrogram. Max Pooling does this by finding the maximum values for each 2x2 'pool' and progressively reduces the spatial size of the representation, which as aforementioned, ties into reducing the overall complexity of the model.

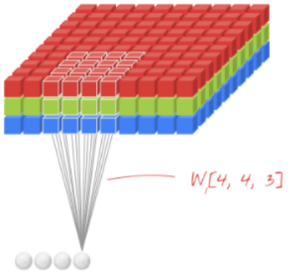

Figura 37: How maxpooling process works

4.2.3 Convolutional Layer

A convolution uses a filter that performs a convolution operation as it parses the input with respect to its dimensions.During this parsing phase, the operation will detect the desired features from the image and extract them for the prediction using a kernel.
For two dimensional inputs, such as images, we use a two dimensional version of a convolution:

$$y_{i,j} = \sum_{kl} \boldsymbol{w}_{kl} \boldsymbol{x}_i + k, j + l$$

$\boldsymbol{w}$ is the convolution kernel, which can be generalised to higher-dimensions

4.2.4 Pooling layer

The purpose of the pooling layer is to 'combine' in a way that reduces the size of the data There are a number of different pooling layers that could have been used, like Average Pooling. This research chose to use the Maximum Pooling layer instead, whereby a window scans an input image and selects the maximum value within each window. This step is imperative as it lowers the computational time of the algorithm by reducing the complexity of the data – in turn, reducing over-fitting of the model.

4.2.5 Flatten Layer

Flattening compresses the data from the convolutional layers into a one-dimensional feature vector in preparation for the classification task conducted within the dense layers.

4.2.6 Fully-Connected Dense Layer

This block is connected with its preceding layer. It therefore receives output from each neuron from the preceding layer, performing a matrix-vector multiplication.

4.2.7 Optimizer

Softmax assigns the probabilities to each of the ten classes, with the sum of the probabilities equals 1.0.

$$\sigma(\vec{z})i = \frac{e^{zi}}{\sum_{j=1}^{K} e^{zj}}$$

$\boldsymbol{Zi}$ values equate to the elements of the input vector. $\boldsymbol{e^{zi}}$ is an exponential function applied to each element of the input vector. It will provide a value above zero, which would be very small if the value was negative and a large value if the input was large. Nevertheless, the values can still fall outside of the 0-1 range required for probabilities.

The denominator is the normalisation term that ensures the values of the function sum to one as required.

4.2.8 Loss Function

The loss function used in this model was categorical cross entropy. Categorical cross entropy is a loss method that measures distance in the probability score between the predicted value and the true value. It is an effective function for multi-class problems because.

$$Loss = -\sum_{i=1}^{outputsize} y_i * log\hat{y}$$

During the training process, the model weights are iteratively adjusted in order to minimise the equation above.

5 CNN First Model

5.1 Architecture

```
Model: "sequential_85"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_93 (Conv2D)           (None, 16, 8, 300)        3000
_________________________________________________________________
max_pooling2d_49 (MaxPooling (None, 8, 4, 300)         0
_________________________________________________________________
flatten_76 (Flatten)         (None, 9600)              0
_________________________________________________________________
dense_155 (Dense)            (None, 300)               2880300
_________________________________________________________________
dense_156 (Dense)            (None, 10)                3010
=================================================================
Total params: 2,886,310
Trainable params: 2,886,310
Non-trainable params: 0
_________________________________________________________________
```

Figura 38: First Module Architecture

5.1.1 Methodology

The training data set was split into a training set and a validation set using sklearn's train-test-split function, with the validation set being 33.3% of the original train set. The batch size was set to 32 and the number of epochs ran was 50.

	Train	Validation	Test
Accuracy	0.9487	0.7660	0.4979

Observing the **Loss per Epochs** and the **Accuracy per Epochs**, the validation loss diverges from the training loss after only 10 epochs. This issue arises due to the model over fitting the data, learning the structure of the data too well and then failing to generalise out of sample. Additionally, this divergence can be explained as the model being too complex, having too much spare capacity than is required for the problem.
Furthermore, the roughness of the loss plots is indicative that the validation set is unrepresentative of the train data set.

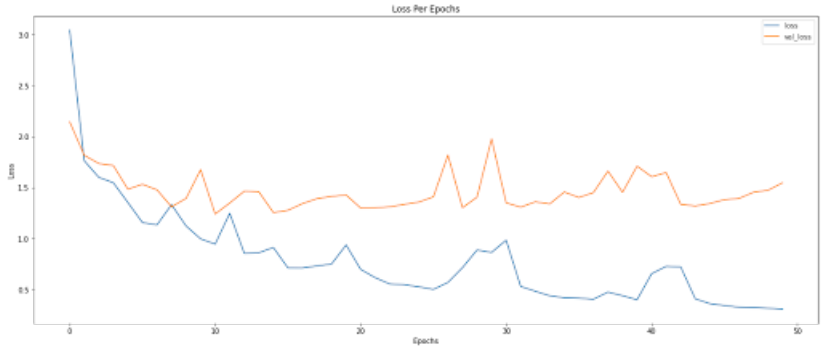

Figura 39: Loss per Epochs CNN model 1

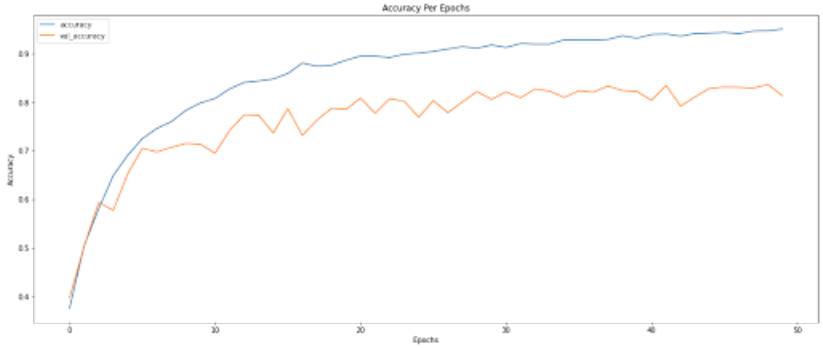

Figura 40: Accuracy per Epochs CNN model 1

The accuracies begin to diverge at 7 epochs, before they reach their maximum after around 20 epochs, after which they begin to plateau. This is further exemplary of over fitting.

5.2 Confusion Matrix

The results presented in the confusion matrix are promising for the validation data. The objective is to have all the values equi-distributed across the diagonal, relative to their count within the validation data set; this signifies that the predicted values equal the true values. The off-diagonal elements show where the model has misclassified. In this confusion matrix, the model has classified well, with a low number of misclassifications and high numbers across the diagonal. Taking a look to other metrics such as:

- Macro Average **accuracy** = 0.78
- Macro Average **Recall** = 0.76
- Macro Average **f1-score** = 0.77

The Classification Report and the Confusion Matrix for the test data are less convincing and are visible signs of over fitting:
From the confusion matrix on test data it can be seen how, for example, large number of misclassifications between jackhammer and the drilling, whilst dog barking is accurately classified 331 times.

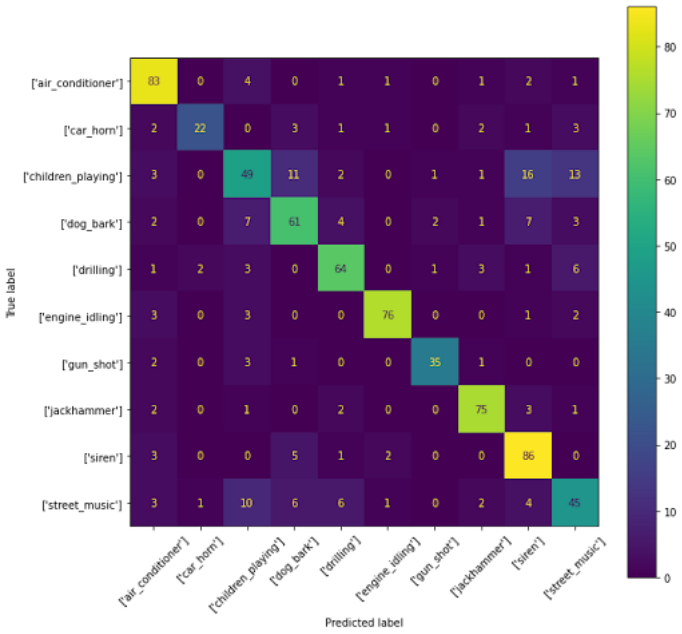

Figura 41: Confusion Matrix for the Validation Data

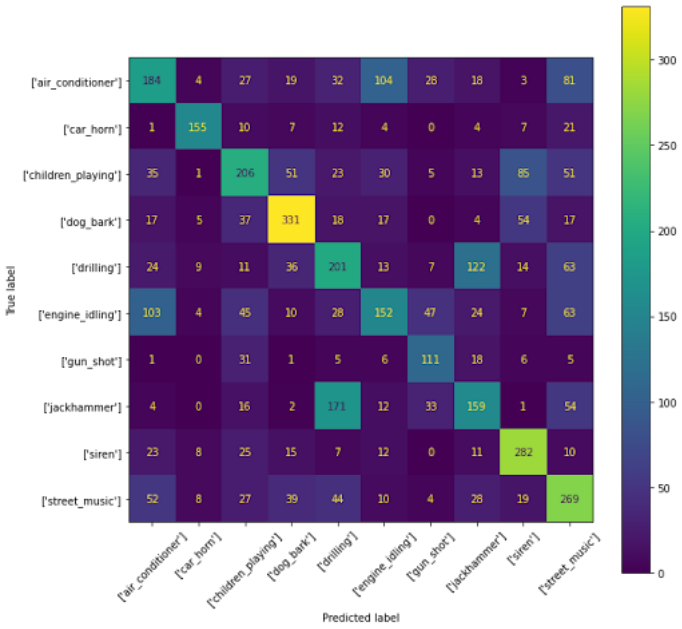

Figura 42: Confusion Matrix for the Test Data

6 CNN Second Model: Simpler Model Architecture

6.1 Methodology

With the first model, there was strong over-fitting to the data, with one hypothesis being that the model was too complex. In search of falsifying this hypothesis, this section will look to simplify the model and in turn have the model's capacity match the size of the data.

```
Model: "sequential_93"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_101 (Conv2D)          (None, 14, 6, 64)         640
_________________________________________________________________
flatten_84 (Flatten)         (None, 5376)              0
_________________________________________________________________
dense_171 (Dense)            (None, 64)                344128
_________________________________________________________________
dense_172 (Dense)            (None, 10)                650
=================================================================
Total params: 345,418
Trainable params: 345,418
Non-trainable params: 0
_________________________________________________________________
```

Figura 43: Second Module Architecture

6.2 Architecture

In comparison with the first model, this simpler model has a mere 345,418 parameters and only one convolutional block.

6.3 Results

Showing the result of the simplest model applied.

	Train	Validation	Test
Accuracy	0.9235	0.7455	0.4429

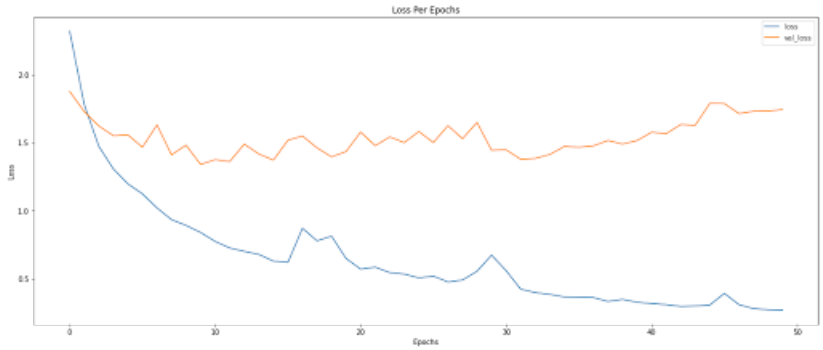

Figura 44: Loss per Epochs CNN model 2

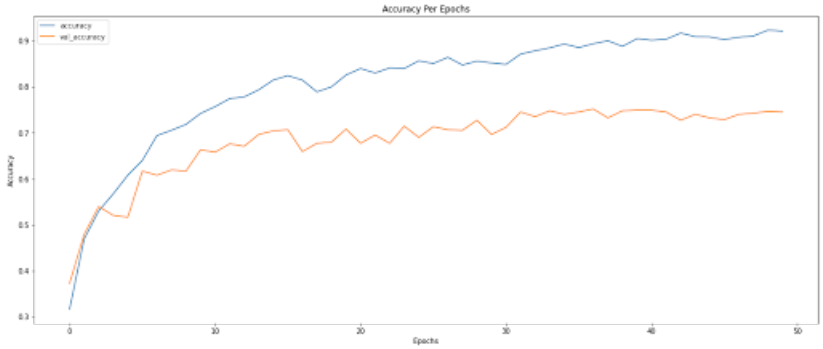

Figura 45: Accuracy per Epochs CNN model 2

The model loss and accuracy display signs of over-fitting as the validation set diverges quickly from the train set.

7 CNN Third Model: Cross-Validation

Instead of splitting the dataset using slearn's train-test-split function, this paper explores the stratified cross-validation approach instead, which in theory should help in the reduction of over-fitting within the data and hence improving the overall accuracy of the model. Cross-validation subdivides the data into k subsets and then rotates the training and validation sets among them. The image above offers a more transparent idea behind

Figura 46: Example of Cross-Validation

the approach. The training dataset is subdivided into 10 folds and within each fold, a validation set is left out; this allows one to estimate the model performance unseen data not used whilst training.

7.1 Methodology

The approach adopted in this project is Stratified Cross-Validation on the simpler model. This method splits the data whilst ensuring that each fold contains the same proportion of observations with a given categorical value, in this instance, one of the ten possible audio classes.

7.2 Results

The data was first subdivided into 10-fold, producing a mean accuracy of 50.44% across the different folds. The data set was also split into 5 folds, yet the mean accuracy remained the same.

8 Hyperparameter Tuning

There are a number of methods to improve the model's test accuracy and reduce over-fitting, which come in the form of the following:

- Early Stopping
- Reducing Learning Rate on Plateau
- Convolution Sub Sampling Pairs
- Number of Feature Maps
- Drop-out Rate

8.1 Early Stopping and Reduced Learning Rate (LR)

8.1.1 Methodology

The Early Stopping rate will halt the model after n-number of epochs when it observes no increase in accuracy/decrease in loss.

Reducing the Learning Rate on the plateau effectively reduces the learning rate when a metric ceases to improve.

8.1.2 Results

The early stopping and reduced learning rate do not improve the model performance on the train and validation set; in fact, they reduce the performance. Stopping too abruptly means the model might not make use of all the data and therefore under-perform/under-fit (hence the poorer accuracy). In this model, the learning rate used was 0.1. Setting the learning rate to this level, convergence to the minimum will be smooth but too slow.

Accuracy	Early Stopping	Reduced LR (LR)
Train	0.77933	0.74900
Validation	0.69	0.664442

8.2 Convolution Sub-Sampling Pairs

8.2.1 Methodology

The performance of the model will be tested using a different number of convolutional layers – 1,2,3 – using our standard training set and validation set. The number of convolutional blocks will increase the number of features extracted from the data, however the downside to the number of blocks is the increase in complexity, potential over-fitting and computational time.

8.2.2 Results

The best performing model configuration was the one that included three convolutional layers, with a training accuracy of 0.98 and a validation accuracy of 0.82. Adding three layers means greater complexity but also demonstrates our ability to extract more features from the input and induce better learning.

Accuracy	1 Layer	2 Layer	3 Layer
Train	0.87106	0.96702	0.98601
Validation	0.7417	0.7968	0.8218

Tabella 3: Train and Validation Accuracy for the Number of Layers

8.3 Number of Feature Maps

Showing the picture concerning the number of Feature Maps

Accuracy	8 Maps	16 Maps	24 Maps	32 Maps	48 Maps	64 Maps
Train	0.8225	0.9165	0.9580	0.9670	0.9750	0.9820
Validation	0.6926	0.7497	7787	0.7817	0.8058	0.7897

Tabella 4: Train and Validation Accuracy for the Number of Feature Maps

8.4 Drop-out Rate

Accuracy	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7
Train	0.8225	0.8066	0.7871	0.77461	0.7631	0.7101	0.6581	0.6346
Validation	0.7071	0.7057	0.6896	0.6866	0.6786	0.6666	0.6656	0.6293

Tabella 5: Train and Validation Accuracy for the Drop-out Rate

As drop-out rate increased, the training accuracy and validation accuracy decreased significantly, implying that the drop-out rate, given the model architecture, doesn't improve the model.

9 Conclusion

9.1 Multi-Layer Perceptron

Drawing conclusions about the outputs gained from all the experiments it can be noticed how the suggested model provided by the hyper parameters optimization gives on average the same performance. In particular the performance related to the **first algorithm** is the best compared to the other one shown. The main characteristics of that one is inside the right choice of the n° of nodes provided by the tuning. This result allows the algorithm to score a good match between predicted and real features. The **second algorithm** where the metrics (AUC) has been changed does not improve the algorithm in terms of prediction performance, the mismatch gained from the confusion matrix is high with a lot of classes mistakenly classified.
The implementation of the other 3 models from **3rd** up to **5th**algorithms show how the model tuning parameters (nodes and layers) suggested by the iteration through the model, don't improve the model performance, scoring on average the same result on the 10th classes available. The reason why the evaluation performance is not the best of results could be obtained is to research into the splitting process. The train and test set are splitted in a way in which the power deployed by the algorithms is more than the quantity of fresh data available to train, this condition leads to slightly low performance on the training set (*over fit*) and test set (*accuracy*) respectively.
Even if the parameters such as *n° of batch size* have been implemented in a gentle way the result is that the bigger batch size still wouldn't give a better solution in practice as compared to smaller batch size.
Also all the other parameters tested inside the model such as:

- *custom learning rate*
- *custom activation function*
- *custom dropout size*

Give results not in line with the one provided by the 1st model.
The proof of what explained in the line before is given by the experiment ran through the random split of entire sound data set (by **train_test_split(X, Y,training_size=0.8, testsize=0.2, random_state = 42**) in which the random splitting is given by the function ensuring large portion of training set in which the power deployed by the algorithm can be employed to find all the possible patterns and testing they on the remaining parts *(test set)*. Implementing this method it can be shown, using the same *metrics* how the accuracy improve sharply(around 83%[5]).
The accuracy score obtained from these **MLP** configurations executed by the following folder split in train and test respectively give a result which is 0.68. The accuracy reached by the best model, plotted in table 1 (*SVM*), is equal to 68%. It's clear how the performance of the neural networks tested are very close to the SVM's one.

[5]Disclaimer: The model with the random split configuration has been executed on python just as comparison and to proof what stated in the final conclusion

9.2 Convolutional Neural Network

The CNN is a model that is designed to handle the non-linearities present in image data. The exploration of this model type in the paper also revealed where the model type can fall short.

The first model produced a train and validation accuracy of 0.94 and 0.76 respectively, however it fell short when evaluated on the test set, with a mere 0.49 accuracy. The conclusion was that the model was too complex and it had a greater capacity than what the data needed. The simpler model had one convolutional layer with only 64 filters and two dense layers, with one of those layers having 64 filters, too. The accuracy for this was less than the first model, with 0.44. Overall, the CNN under-performed compared to the benchmark scores and the MLP – different model complexities were explored, the hyperparameters were adjusted and cross-validation was used.

Whilst various methods fell short in this study, a point for future consideration would be to split the dataset not in the manner the authour intended (with the split between train and test being almost equally). One could assign more training data to the training split, giving the model a greater number of data points to learn from and then generalise better out of sample, yet still assigning a suitable amount for the test set so as not to over-fit.

We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.