UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Artificial Intelligence and Data Engineering

Bird Sound Classification

Project Documentation

Martina Marino

Roberta Matrella

Academic Year: 2022/2023

Contents

Abstract

This project aims to use convolutional neural networks (CNNs) to tackle the problem of bird classification in the *BirdCLEF2023* dataset. Two main strategies will be explored: using existing pre-trained networks and training a network from scratch. In the first strategy, we will exploit pre-trained neural networks, such as ResNet50 or MobileNetV2, that have been trained on large datasets such as ImageNet. These networks will have already learnt a wide range of visual features and we can adapt them to our specific task through transfer learning. By fine-tuning the weights of the final network layers, we will train the pre-trained network on the *BirdCLEF2023* dataset, allowing the model to learn and discriminate the distinctive features of the birds in the dataset. In the second strategy, we will train a CNN network from scratch. We will start with a basic architecture, such as a sequence of convolution, normalisation, activation and pooling layers. Next, we will train the model on data from the *BirdCLEF2023* dataset, initialising the weights randomly and using the back-propagation algorithm to optimise the model. This strategy will allow us to learn the specific characteristics of the birds in the dataset without depending on pre-existing information from other datasets. Both strategies will be evaluated in terms of classification accuracy and overall performance. Computational requirements and the time needed to train and test the models will also be considered.

1 Introduction

In recent years, there has been a growing public awareness of the importance of environmental protection and sustainable development. People are increasingly aware of the impact of human activities on the planet and are keen to take action to conserve natural resources and biodiversity. Among the various indicators used to assess the state of the environment, birds are of particular importance. Bird populations around the world are facing numerous challenges, mainly due to climate change and habitat destruction. These factors have contributed to a significant loss of biodiversity and bird populations are declining at an alarming rate. As birds play a crucial role in ecosystems, their decline can have far-reaching consequences, disrupting food chains, pollination and seed dispersal. It is therefore essential to gather accurate and up-to-date information on changes in bird communities in order to target conservation efforts effectively.

Figure 1: Beautiful Sunbird

Birds are particularly useful as biodiversity indicators because of their high mobility and diverse habitat requirements. Their presence or absence in an area can provide valuable insights into the overall health and functioning of ecosystems. However, detecting and identifying birds based solely on visual observations can be challenging, especially in dense vegetation or remote locations. Fortunately, birds communicate through vocalisations, making sound analysis a powerful tool for their detection and identification.

Traditionally, the identification of birds by their sounds has relied on the expertise of human experts. These experts listen to recordings and analyse the unique patterns and characteristics of birds' vocalisations to determine species. However, this manual identification process is time-consuming, laborious and not easily scalable. To overcome these limitations, there is a need for a reliable and efficient automatic identification system that can handle large volumes of bird sound data.

1.1 Goals

Convolutional Neural Networks (CNNs) have become one of the most effective and widely used methods for automated classification tasks, including audio analysis. CNNs excel at learning hierarchical representations from input data, making them well suited to tasks such as image and audio classification. Leveraging their ability to extract meaningful features from spectrograms, CNNs have shown promising results in identifying bird species based on their vocalisations.

Therefore, the main objective of this project is to develop a CNN for automatic classification of bird sounds. The project explores two approaches: building a CNN from scratch and using pre-trained CNN models. Both approaches involve the processing of spectrograms generated from audio recordings of bird sounds. Spectrograms provide a visual representation of sound frequencies over time, allowing the CNN models to extract relevant features and accurately classify bird species.

By developing a reliable and efficient automatic classification system, this project aims to contribute to the field of bird monitoring and conservation. The CNN models will enable researchers, conservationists and scientists to more effectively analyse large amounts of bird sound data, providing valuable insights into changes in bird populations and facilitating targeted conservation actions.

2 State of the art

The field of automatic bird sound classification has made significant progress in recent years. Researchers have explored various approaches and techniques to improve the accuracy and efficiency of bird species identification. This section provides an overview of the state of the art in this field, highlighting some notable studies and developments.

One of the main methods used in bird sound classification is the use of machine learning algorithms, in particular convolutional neural networks (CNNs). CNNs have shown impressive performance in image and audio classification tasks, including bird sound identification. Researchers have developed CNN architectures specifically tailored for bird sound analysis, incorporating techniques such as spectrogram processing and feature extraction.

Several studies have focused on the development of large-scale bird sound datasets to train and evaluate CNN models. These datasets contain a wide range of bird species and cover different geographical regions, ensuring robustness and generalisability of the models. Examples of such datasets include the BirdCLEF and Xeno-Canto datasets, which have facilitated the advancement of bird sound classification research.

Transfer learning has emerged as a powerful technique in bird sound classification. By using pre-trained CNN models, initially trained on large image datasets, researchers can use the learned visual features and transfer them to the bird sound classification task. This approach has shown promising results, achieving high accuracy and reducing the need for large amounts of labelled data.

To further improve the accuracy of bird sound classification, researchers have explored ensemble learning techniques. Ensemble models combine the predictions of several individual models to produce a more robust and accurate classification result. Ensemble learning has been successfully applied to bird sound classification, achieving higher accuracy and mitigating the limitations of individual models.

In addition to CNN-based approaches, researchers have also explored other machine learning algorithms, such as support vector machines (SVMs) and random forests, for bird sound classification. These algorithms, when combined with appropriate feature extraction techniques, have shown competitive performance in species identification tasks.

The integration of citizen science initiatives and crowd-sourced data has also contributed to advances in bird sound classification. Platforms such as eBird and Xeno-Canto allow birders and enthusiasts to contribute audio recordings and associated species labels. This data, when combined with automated classification systems, can significantly increase the scale and coverage of bird sound monitoring efforts.

Finally, advances in mobile technology have led to the development of smartphone applications for bird sound identification. These applications use the processing power and built-in microphones of smartphones to identify bird species in real time. They often use machine learning algorithms and spectrogram analysis techniques, allowing users to identify birds on the move.

Overall, the state of the art in automatic bird sound classification demonstrates the

effectiveness of CNNs and other machine learning algorithms in accurately identifying bird species based on their vocalisations. Ongoing research and advances in large-scale datasets, transfer learning, ensemble methods and citizen science initiatives continue to improve the accuracy, scalability and accessibility of bird sound classification systems. These developments have significant potential to improve bird monitoring, conservation efforts and our understanding of avian ecosystems.

Both in [6] and in [4] the audio recordings have been split into chunks of equal length. In [6] they have been transformed into mel-spectrograms while in [4] into STFT. In [6] have been recognized 659 species from 50,000 audio recordings with ResNet and Inception. With Inception, in particular, a c-mAP of 0.16 has been reached getting the fifth place in the competition. In [4] the spectrograms, both gray-scale both RBG, were given as input to Inception, MobileNet, ResNet and VGG. The number of classes evaluated was 2, 10 and 50. With 50 classes they have achived an accuracy of about 40%. In [7] 72 species over 146 have been considered. Also in [7] the audios were divided into chunks and converted to mel-spectrograms. These have been given in input to a from scratch CNN with four convolutional layers and a classification head of two fully connected layers. The model has been trained for 10 epochs with a learning rate of 0.0001 and a batch size of 64. With few hundred samples for each specie they have achieved about 90% of accuracy. In [10] the audio chunks have been given in input to an oject detection system in the spectral domain that performs diarization over 50 bird species, achiving an F1score of about 85%. In [5] different data augmentation methods have been applied over the 50000 samples of the 100 different species of the BirdCLEF 2017. Vertical Roll, Gaussian Noise, Noise Samples and Batch Augmentation have been applied at runtime. The better results have been reached with the Ensemble of ResNet e DenseNet with 0.62% of c-mAP.

3 Dataset

The dataset used for the development of the project is *BirdCLEF2023* [1], provided by Kaggle, containing several bird records made in Kenya. *BirdCLEF2023* contains a folder within which are present audio recordings of bird songs in ogg format divided into folders representing the different species. The name of the folders containing the audios of the different species has a specific code for each of them whose correspondence can be identified in the *eBird_Taxonomy_v2021.csv* file. There are also three csv files containing information about the bird species and the relationships between them. The about 16k audios collected in this dataset cover 264 species of East Africa uploaded by the users of *xenocanto.org* [2]. Each species has different audios in number and duration, with the overall length per species ranging from less than a minute to several hours. Species with few audios are extinct and therefore of little relevance to this project.

3.1 Preprocessing

3.1.1 Data Exploration

Once the source dataset was downloaded, a phase of data exploration followed to better understand the characteristics of the data. The 264 folders, each corresponding to a bird species, contain a series of audio recordings. The number of audio recordings per species ranges from a minimum of 1 to a maximum of 500. On the other hand, the total duration of the recordings for each species ranges from a minimum of 7 seconds to a maximum of 17 hours. Thus, the dataset is very unbalanced and species with a very low total number of minutes of recordings do not have enough data from which to extract useful information for classification. Therefore, we decided to reduce the number of species to 100 as explained in the following paragraph.

3.1.2 Audio splitting

The *BirdCLEF* dataset is a widely used dataset for bird sound classification tasks. To preprocess the audio data from this dataset, it is recommended to first split the audio recordings into smaller segments. Each segment should have a fixed duration, typically ranging from a few seconds to tens of seconds, depending on the duration of the bird vocalizations of interest and the temporal resolution needed for classification. This segmentation process allows us to extract spectrograms for each segment individually, capturing the temporal dynamics and patterns specific to different bird sounds. Moreover, it ensures a consistent input size for the model and facilitates event detection within each segment. For the dataset under consideration, the audio recordings have been splitted into chucks of 10 seconds each, discarding the ones not reaching this threshold.

At this point, the 264 species have been sorted by decreasing number of splitted audio recordings and the first 100 were selected to proceed. In fact, the discarded classes have

too few data to let the models learn enough information to correctly do the classification. Choosing to proceed in this way rather than selecting the 100 species with the most total minutes of recordings. made it possible to exclude classes with a greater total number of minutes but broken up into many small audios that would have been discarded in preference to those with longer audios from which more 10-second splits could be obtained.

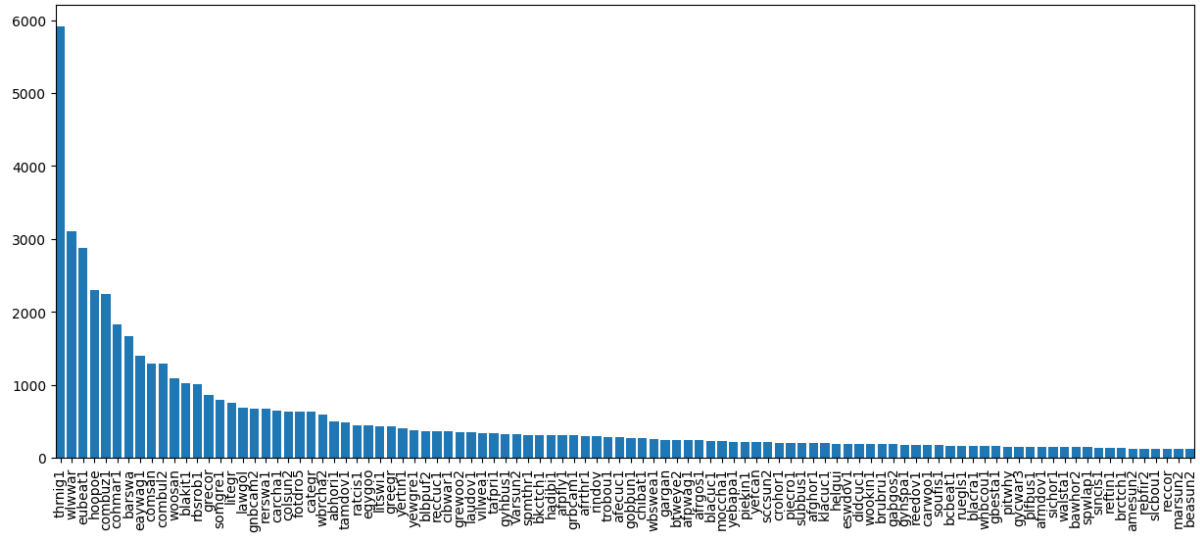


Figure 2: Number of audio recordings per specie

After segmenting the audio, spectrograms can be generated for each segment using suitable signal processing techniques. This process enables the extraction of frequency-based features that can be used as inputs to train a sound classification model on the dataset.

3.1.3 Spectrograms generation

When generating spectrograms for sound classification tasks, both Short-Time Fourier Transform (STFT) and Mel Spectrograms (MEL) are commonly used techniques. Each has its own advantages and considerations:

1. **STFT**: The STFT is a time-frequency representation that divides the audio signal into short overlapping segments and computes the Fourier transform on each segment. The resulting spectrogram represents the magnitude or power of the signal at different frequencies over time. STFT spectrograms provide detailed frequency resolution and can capture fine-grained spectral information. They are suitable when the exact frequency content of the sound is important for the classification task.

2. **Mel spectrograms**: Mel spectrograms are derived from STFT spectrograms by applying a Mel filterbank that mimics the human auditory system's perception of sound. The Mel scale is a perceptual scale that maps the linear frequency scale to

a logarithmic scale, emphasising lower frequencies and smoothing higher frequencies. Mel spectrograms are particularly useful for tasks where sound perception is important, such as speech recognition or music analysis. They can provide more compact representations with improved discrimination for human listeners.

The choice between STFT and Mel spectrograms depends on the specific requirements of the sound classification task and the characteristics of the audio data. If the fine-grained spectral details are crucial for the classification, STFT spectrograms may be preferred. On the other hand, if the perceptual aspects of sound are more important, Mel spectrograms can be a suitable choice. It's worth noting that Mel spectrograms are commonly used in bird sound classification tasks, as they align well with the perception of bird vocalizations by human listeners.

For these reasons, the second approach was followed, thanks to the *librosa* library, a popular Python library designed specifically for audio and music signal processing tasks. It provides a wide range of functions and tools for working with audio data, including the generation of Mel spectrograms. In 3 there is an example of the Mel spectrogram obtained from an audio recording.

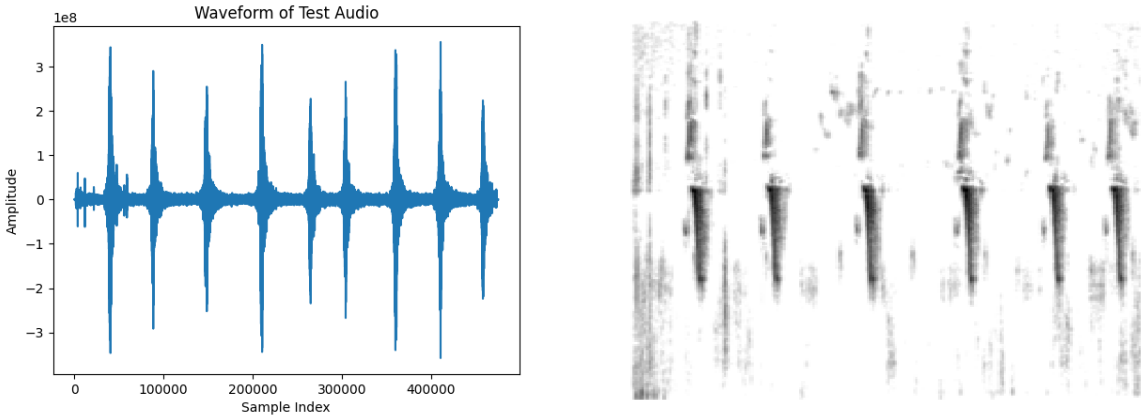


Figure 3: Waveform and relative Mel spectrogram

The *librosa* library simplifies the process of generating Mel spectrograms by providing easy-to-use functions that handle the necessary signal processing steps. It allows users to load audio files, perform resampling if necessary, and extract various audio features, including Mel spectrograms, with just a few lines of code.

One of the key features of *librosa* is its ability to compute Mel spectrograms directly from audio signals. It uses the STFT implementation in the background to convert the audio waveform into a time-frequency representation. By default, *librosa* applies a Mel filterbank to the resulting STFT spectrogram, effectively transforming it into a Mel spectrogram. This transformation follows the principles of the Mel scale, emphasising perceptually relevant frequencies and providing a more meaningful representation for sound classification tasks.

3.2 Sets generation

After that the train, validation and test sets have been created using two times the *train_test_split* function provided by *sklearn* with the following percentage:

- training set: 75%
- validation set: 15%
- testing set: 10%

The split has been done in *stratified* mode in order to mantain the proportion between the different classes.

3.3 Up-sampling and down-sampling

The pre-processed dataset, as mentioned earlier, is highly imbalanced and for this reason it is necessary to consider some techniques to rebalance the training set. It should be noted, in fact, that the training set rebalancing is applied to train the model so that it can learn to correctly classify all classes present equally but, subsequently, performance will be evaluated on the unbalanced validation and test sets to reflect the real case of application where some species are more populous and others rarer.

In the context of data processing, upsampling and downsampling are two techniques used to address the problem of unbalanced datasets, where the classes of interest are represented by significantly fewer samples than the other classes. Upsampling and downsampling are two opposite approaches that aim to balance the distribution of classes within the dataset. Let us see how they work:

- **Upsampling**: or oversampling, is a technique that increases the number of samples of minority classes in the data set. This is done by replicating or creating new samples from the minority class. For example, existing samples can be randomly replicated or data synthesis techniques, such as the Synthetic Minority Over-sampling Technique (SMOTE), can be used to generate new synthetic samples that retain the characteristics of the minority class. This increases the number of samples for minority classes, making the data set more balanced.
- **Downsampling**: or undersampling, is a technique that reduces the number of majority class samples in the data set. This is done by randomly selecting a subset of samples from the majority class so that the number of minority and majority class samples is approximately balanced. This reduces the dominance of the majority classes and makes the data set more balanced.

There are advantages and disadvantages to both approaches. Upsampling can lead to a risk of overfitting, as replicated or synthetically generated samples can introduce

noise or redundant information into the data set. Downsampling can lead to a loss of information by removing samples from the dataset that could be useful for model learning.

In order to obtain a balanced training set, ensuring that the model learns from a fair representation of the different classes in the database, both approaches were applied, as shown in 4.

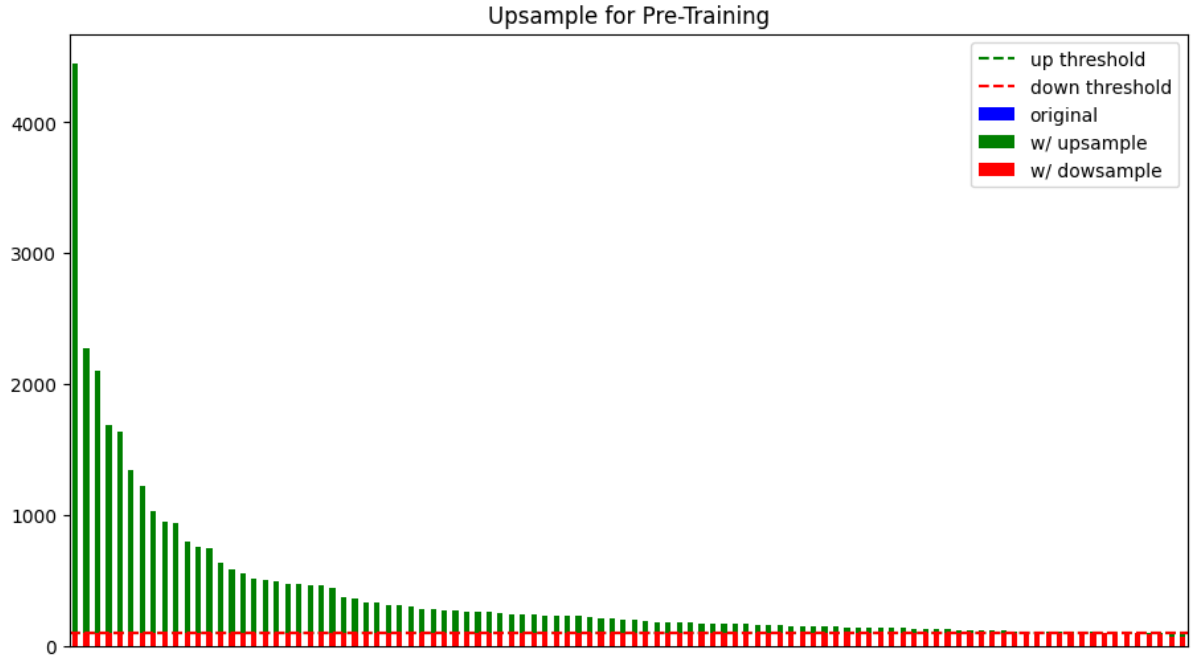


Figure 4: Upsampling and downsampling on training set

The red line represents the cut made for each class. First, downsampling to 100 was performed to reduce the majority classes; then oversampling to 100 of the minority classes was performed by exploiting data augmentation. The classes not reaching the threshold of 100 were:

- slcbou1
- rebfir2
- amesun2
- reccor
- beasun2
- marsun2

Validation set and test set have been left unbalanced in order to evaluate the model's generalization ability and real word performance where class distributions may be skewed.

3.4 Data augmentation

Data augmentation is a technique used in machine learning and image processing to increase the amount and variety of training data. It consists of applying artificial transformations to existing data, such as images, to create new examples that are similar but not identical to the originals. This helps to improve the performance and generalisation ability of machine learning models.

The goal of data augmentation is to make the model more robust and resilient to variations in test data or real-world situations. For example, when training a car image classification model, you can apply transformations such as rotation, blurring, zooming, cropping or horizontal flipping to existing training images. These artificial transformations create new images that have variations similar to those that might be present in the test data, such as different angles, sizes, blurs or orientations.

3.4.1 *SpecAugment* technique

SpecAugment is a data augmentation technique primarily used in the context of automatic speech recognition (ASR). It was introduced by Google AI in 2019 as an effective strategy for improving the performance of ASR models. SpecAugment works directly on the spectrogram, which is a graphical representation of the audio signal in terms of time and frequency. It applies three types of transformations:

1. **Time masking**: this transformation randomly selects a continuous region along the time axis of the spectrogram and replaces it with a constant value. This time masking prevents the model from relying on precise temporal information and encourages it to make predictions based on broader contexts.

2. **Frequency masking**: in this case, a continuous region along the frequency axis of the spectrogram is randomly selected and replaced with a constant value. Similar to time masking, frequency masking prevents the model from relying on precise frequency information and encourages greater generalisation.

3. **Random deletion**: this transformation randomly selects a region of the spectrogram along both the time and frequency axes and replaces it with randomly sampled values from a uniform distribution. This type of transformation introduces random variation into the spectrogram, helping to make the model more robust to perturbations.

The goal of SpecAugment is to reduce the overfitting of ASR models and improve their generalisation ability by creating a greater variety of training data. By applying masking and random erasure transformations to a spectrogram, new examples can be generated that retain the essential information, but exhibit variations similar to those found in the real world.

SpecAugment can also be applied to spectrograms in domains other than automatic speech recognition (ASR). Spectrograms are commonly used representations of audio signals in various tasks such as music classification, sound event detection and speech emotion recognition. When SpecAugment is applied to spectrograms in these domains, the basic principles remain the same. The aim is to introduce variations in the spectrograms during training to improve the generalisation and robustness of the model. The specific transformations used can vary depending on the task and the characteristics of the data. For example, in music classification tasks, SpecAugment can be applied by masking certain time or frequency regions of the spectrogram. This helps the model to focus on more representative features across different genres of music and reduces its reliance on specific time or frequency patterns. By applying SpecAugment, the model becomes more capable of generalising well to unseen music samples.

Similarly, in sound event detection or speech emotion recognition tasks, SpecAugment can be used to introduce temporal or spectral variations into the spectrograms. This can involve masking or erasing specific segments or regions in the spectrogram, thereby encouraging the model to extract more robust and discriminative features from the audio data. The specific choice of transformations and their parameters within SpecAugment can be customised based on the characteristics of the task and the dataset. The aim is to strike a balance between introducing useful variations that improve the generalisation of the model, and avoiding excessive distortions that may hinder its performance.

There are four basic policies of SpecAug[8] for which different values of time and frequency parameters are set.

Policy	**W**	**F**	m_F	**T**	**p**	m_T
LB	80	27	1	100	1.0	1
LD	80	27	2	100	1.0	2
SM	40	15	2	70	0.2	2
SS	40	27	2	70	0.2	2

Table 1: SpecAug policies

The LB and SM policies were chosen because they provide the best performance.

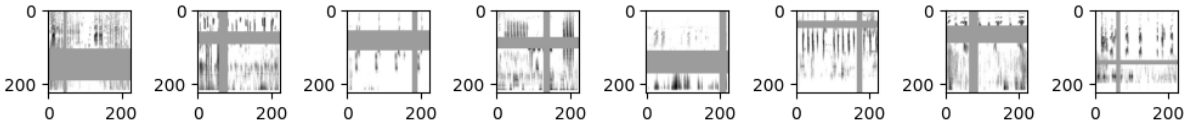


Figure 5: SpecAug with LB and SM

3.4.2 Image Augmentation

Traditional data augmentation techniques for images, such as rotation or vertical/horizontal flip are not appropriate for spectrograms images because they alter the wawe plot introducing. In fact, spectrograms represent the distribution of frequencies in the time domain, and are often used to analyse audio signals. Unlike traditional images, spectrograms have a three-dimensional structure: one axis represents time, another represents frequency and the third represents signal strength (power spectrum). Consequently only some of the usual techniques can be exploited:

- **Horizontal roll**: the image is moved horizontally in a random manner. This can be used to simulate slight temporal variations in the spectrogram image. For example, the image can be shifted left or right by a certain number of pixels.
- **Brightness**: adjusts the brightness of the image by a factor of 0.15. This can allow a slight variation in the overall intensity of the image without compromising the basic information.
- **Blur**: application of a Gaussian blur with a kernel size of 3x3. This can slightly reduce the sharpness of the image, but still preserves the basic characteristics of the spectral information.
- **Gaussian noise**: is a technique for adding random noise with a Gaussian distribution to spectrogram images. This can help make the model more robust to variations and imperfections in the data.

3.4.3 Noise injection

For better results, data augmentation has been also performed directly on the audio chunks, before spectrogram generation. In the context of audio data augmentation, noise injection refers to the technique of adding artificial noise to audio signals in order to augment the training data. By introducing random noise to the audio samples, the model learns to be more robust to different types of background noise and improves its generalization ability when exposed to real-world audio recordings. The specific implementation of noise injection as data augmentation on audio involves adding various types of noise to the original audio samples. The noise can be obtained from recordings of real-world environments, pre-defined noise databases, or synthetic noise generation methods. By incorporating noise injection as a data augmentation technique, audio models can learn to handle various acoustic conditions and background noise, resulting in more robust and accurate performance in practical applications.
Noise injection was exploited to perform data augmentation on minority classes. Specifically, new audio was generated from those in the training set of these classes by adding two different types of background noise, such as the noise of a train and the noise of a waterfall thanks to the audiomentation python library. The background noise have been taken from ESC-50 dataset that offers sounds of different kind.

4 CNN from scratch

In the following paragraph is described our from-scratch architecture. Basing on both the literature and the trial-and-error approach, some experiments were performed to build it. Some additional experiments have been performed to improve the performance.

4.1 Design of the network

For the image size have been choosen 224x224 as proposed in literature and a batch size of 32.
After the input layer, through a rescaling layer, normalization was performed, so that each pixel was in the range [0,1]. Exploiting the zero-padding technique, in order to give the same importance to each pixel, 5 convolutional layers have been used. For the activation function have been use the ReLU. For the local receptive fields the default size value of 3x3 have been used. Max-pooling has been used to summarize small regions into a single value. The final levels changes in the different experiments while the output layer is always characterized by a dense layer of 100 neurons. The softmax activation function has been used for the fully connected layer to perform the classification task. Have also been used Dropout in some experiments. Some callbacks have been used, such as ModelCheckpoint to store intermediate models in case of Google Colab crashes abd EarlyStopping with a patience of 5 to check the validation loss. Adam has been used as optimizer and the learning rate chosen is 0.0001. During the analysis of the experiments, both from scratch both pretrained, the metric taken into consideration have been the F1-score because even if the train set have been balanced, validation and test set are still strongly unbalanced.

4.2 First experiment

In the first experiment a Flatten layer have been added.

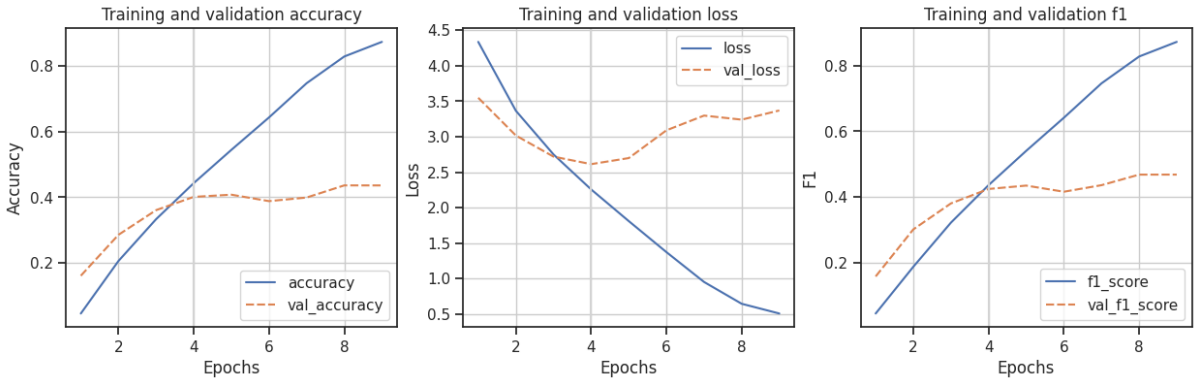


Figure 6: Accuracy, loss e F1-score of train and validation set

	Loss	Accuracy	F1 score
Training	0.5089	0.8727	0.8725
Validation	3.3705	0.4362	0.4682
Test	2.6472	0.3922	0.4174

Table 2: Report experiment 1

The results are disappointing, it is affected by overfitting and the performance is poor.

4.3 Second experiment

In the second experiment GlobalAveragePooling layer have been added.

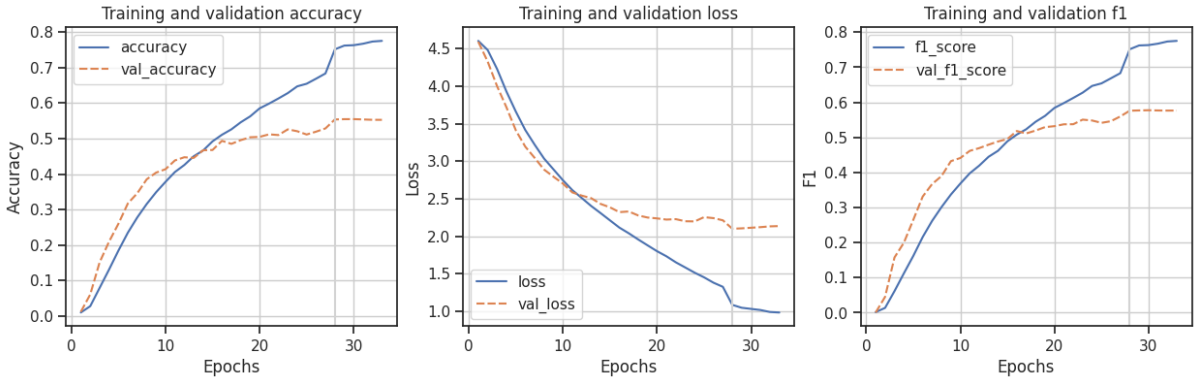


Figure 7: Accuracy, loss e F1-score of train and validation set

	Loss	Accuracy	F1 score
Training	0.9849	0.7753	0.7753
Validation	2.1362	0.5528	0.5766
Test	2.1542	0.5505	0.5725

Table 3: Report experiment 2

As it is possible to observer, the performance have been increased and the overfitting has decreased.

4.4 Third experiment

A dropout layer 0.2 was added to further reduce overfitting.

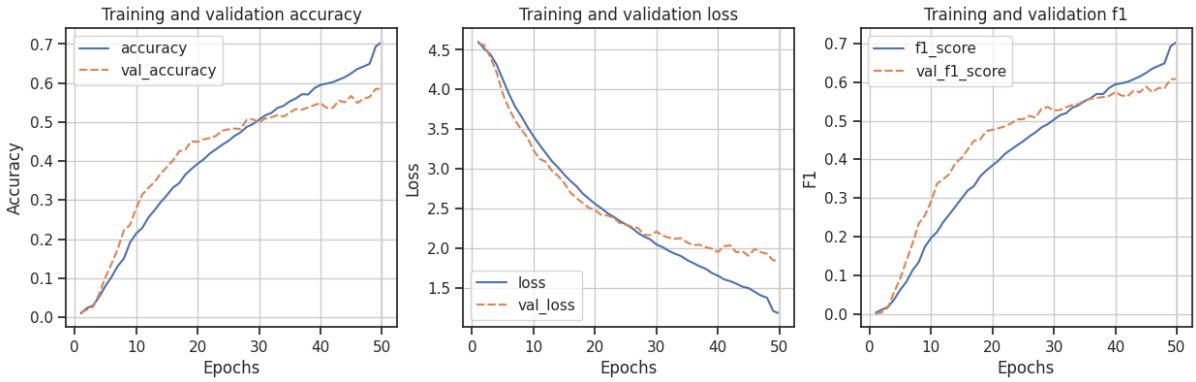


Figure 8: Accuracy, loss e F1-score of train and validation set

	Loss	**Accuracy**	**F1 score**
Training	1.1776	0.7054	0.7045
Validation	1.8458	0.5860	0.6084
Test	1.9132	0.5712	0.5929

Table 4: Report experiment 3

As shown above, the performance has increased and the overfitting has decreased.

5 Pre-Trained Models

This section describes the results obtained using pre-trained architectures and strategies[1]. The tested pre-trained networks are the following:

- VGG16
- MobileNetV2
- ResNet50

5.1 VGG16

VGG16 is a convolutional neural network (CNN) architecture proposed in 2014 by Karen Simonyan and Andrew Zisserman of the Visual Geometry Group Lab at the University of Oxford [11]. It was developed to compete in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual computer vision competition.

The goal of VGG16 in the ILSVRC competition was to perform two main tasks. The first was the localisation of objects in the images, which required objects to be located among 200 different classes. The second was image classification, where each image had to be assigned to one of 1000 available categories.

VGG16 demonstrated excellent performance in both tasks, winning first and second place respectively in image classification and object localisation in the ILSVRC 2014 challenge.

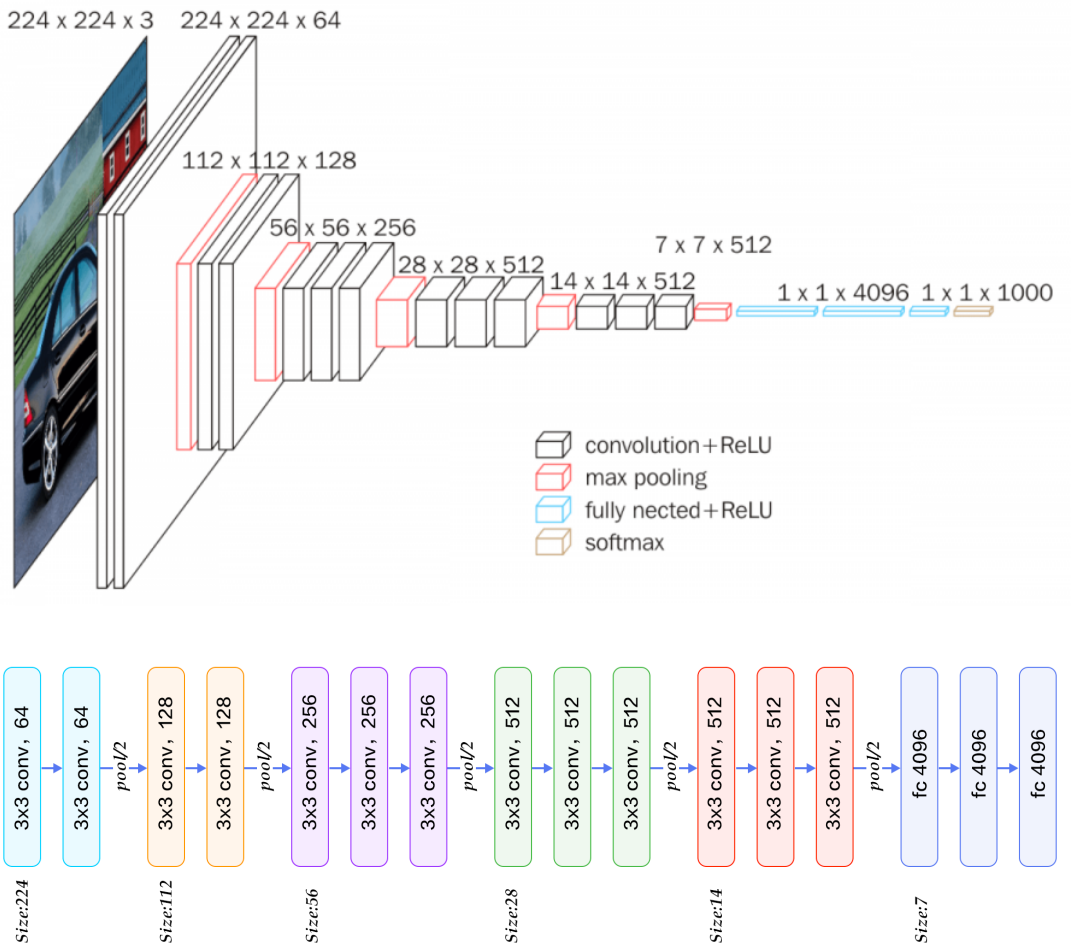
The architecture of VGG16 is characterised by considerable depth and a high number of parameters. It consists of 16 layers, of which 13 are convolutional layers and the last 3 are fully connected layers. During the classification process, VGG16 accepts as input images with a fixed size of 224x224 pixels and 3 RGB channels. Through its complex set of convolutional and pooling layers, the architecture processes the initial image and produces an output vector of size 1000 representing the classification probabilities for the different categories.

To obtain these classification probabilities, VGG16 uses the softmax function, which normalises the model outputs so that the sum of all probabilities is equal to 1. This allows the output vector to be interpreted as a probability distribution over the different classes.

It is important to note that VGG16 requires high computational and memory resources, especially during the training process. The architecture has been shown to achieve high accuracy results in image classification, but its complexity may make it slower to train than other newer architectures such as ResNet.

Despite its limitations in terms of computational efficiency, VGG16 remains a reference model in computer vision and can be used as a basis for transfer learning or as a pre-trained feature extractor in various artificial intelligence tasks.

[1]Data augmentation strategy is always used since we have very little data



(a) VGG16 Architecture

The unbalanced train set have been balanced using different techniques explained in the previous sections. The training sets thus obtained have been compared in order to choose the one that gives the best results. The comparison has been performed on the base models and the best one have been chosen in the following experiments.

5.1.1 Base Model

In this model only a Flatten plus Dense layer with 256 neurons have been added.

Balanced with SpecAugment

The first experiment on the base model uses a training set balanced with SpecAugment. In all the experiment a learning rate of 0.0001 and Adam as optimizer [12].

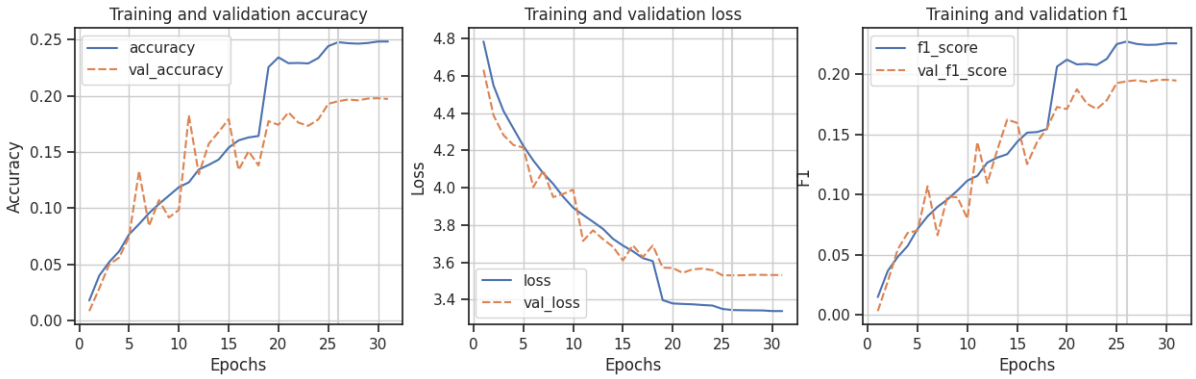


Figure 10: Accuracy, loss e F1-score of train and validation set

	Loss	**Accuracy**	**F1 score**
Training	3.3300	0.2483	0.2258
Validation	3.5311	0.1972	0.1948
Test		0.2024	0.2073

Table 5: Report SpecAugment augmentation

Balanced with spectrograms augmentation

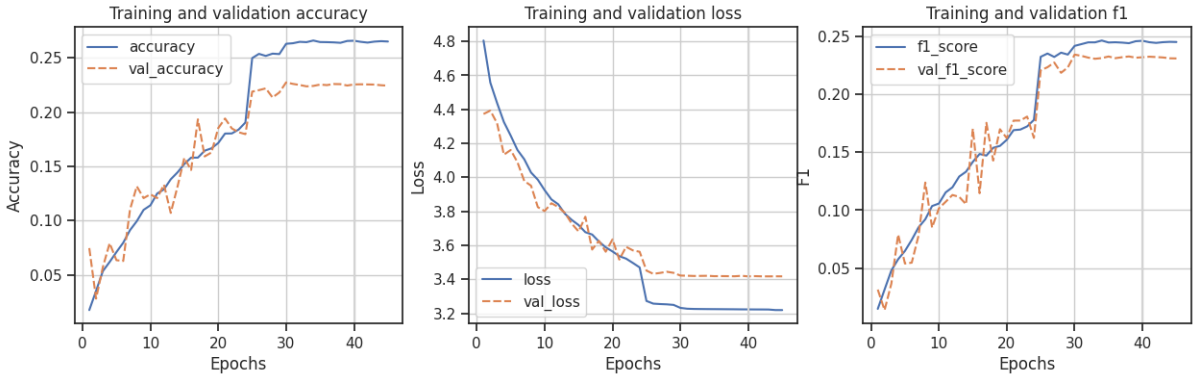


Figure 11: Accuracy, loss e F1-score of train and validation set

	Loss	Accuracy	F1 score
Training	3.2132	0.2685	0.2478
Validation	3.4161	0.2259	0.2324
Test		0.2286	0.2381

Table 6: Report Model spectrograms augmentation

Balanced with audio augmentation

The dataset has been balanced using noise injection for the recordings of the minority classes.

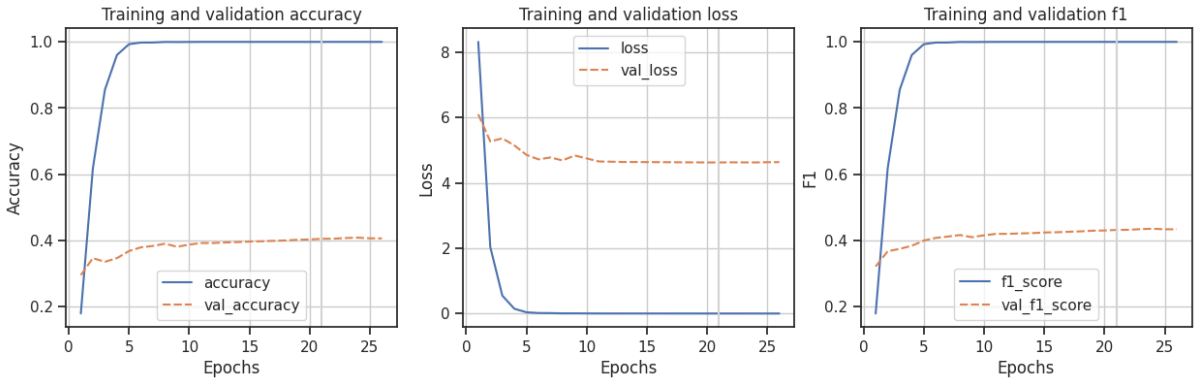


Figure 12: Accuracy, loss e F1-score of train and validation set

	Loss	Accuracy	F1 score
Training	$2.3406e^{-04}$	1.0000	1.0000
Validation	4.6332	0.4061	0.4334
Test		0.3939	0.4265

Table 7: Report audio augmentation

Even if there is a huge overfitting, the performance is much better than the case of the previous cases.

5.1.2 Reducing overfitting

Different experiment have been performed in order to reduce overfitting.

Adding Dropout layer The audio augmentation balanced training set have been chosen because offers the best performance. It is hugely affected by overfitting. A dropout layer have been added in order to reduce the overfitting.

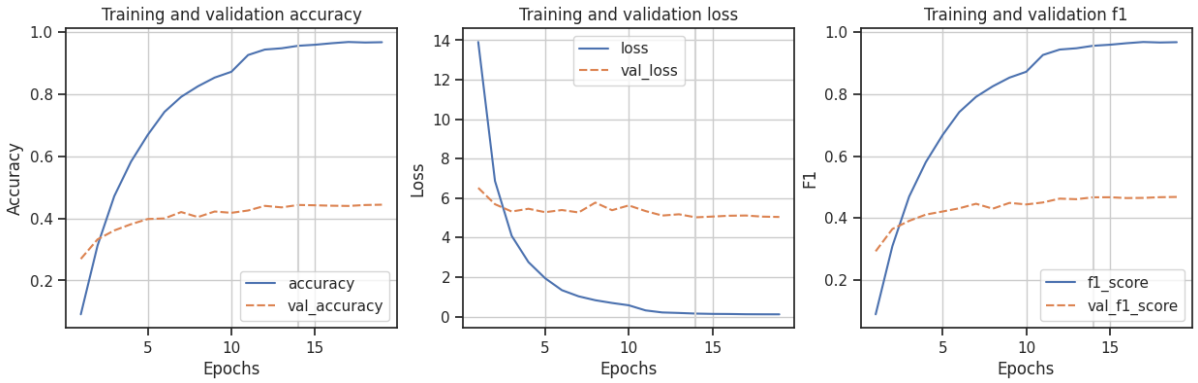


Figure 13: Accuracy, loss e F1-score of train and validation set

	Loss	**Accuracy**	**F1 score**
Training	0.1123	0.9676	0.9676
Validation	5.0450	0.4441	0.4683
Test		0.4352	0.4620

Table 8: Report Flatten + Dense + Dropout

The results are not the expected ones, the overfitting is still present.

5.1.3 Global Average Pooling layer

Some experiment with the different balanced train set have been performed with global average pooling.

Balanced with SpecAugment

In this case the dataset balanced with SpecAugment have been used with global average pooling and a dense layer. The performance are better than the previous ones.

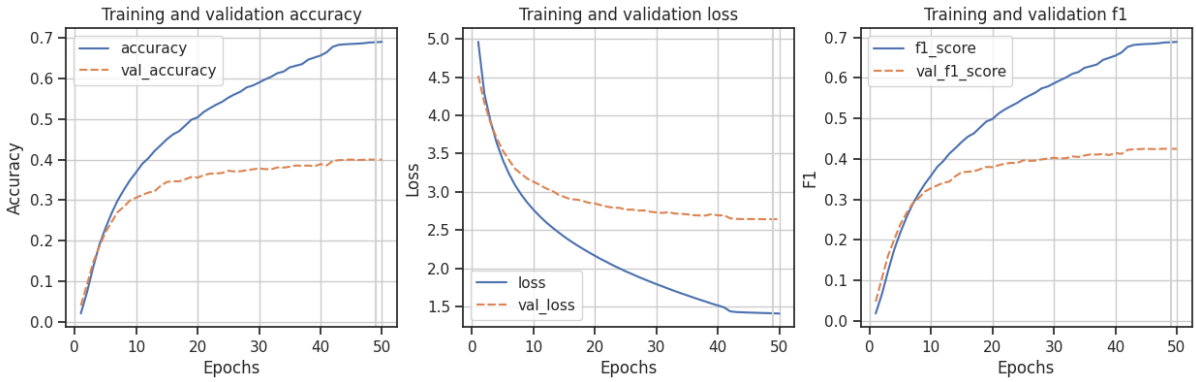


Figure 14: Accuracy, loss e F1-score of train and validation set

	Loss	**Accuracy**	**F1 score**
Training	1.4108	0.6905	0.6890
Validation	2.6450	0.4001	0.4249
Test		0.3884	0.4162

Table 9: Report Global Average Pooling + Dense

The model is affected by overfitting but less than the previous one.

Balanced with spectrograms augmentation

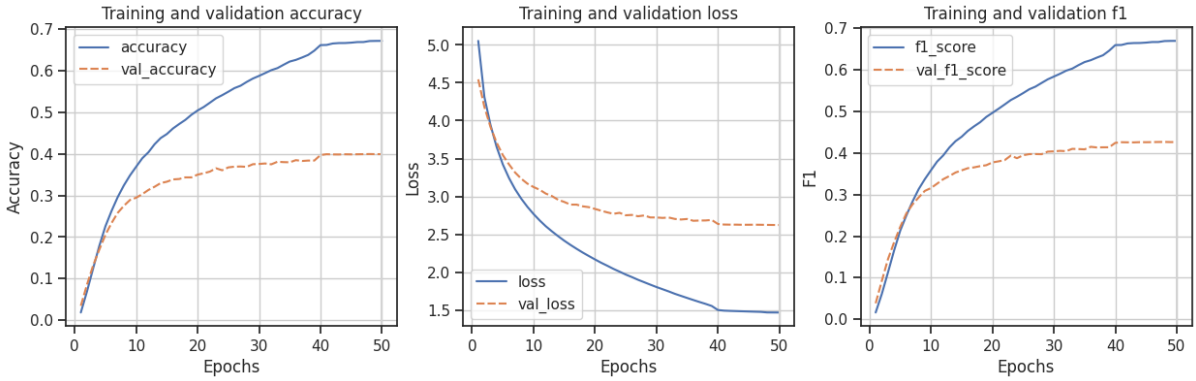


Figure 15: Accuracy, loss e F1-score of train and validation set

	Loss	Accuracy	F1 score
Training	1.4654	0.6716	0.6700
Validation	2.6214	0.3992	0.4265
Test		0.3901	0.4179

Table 10: Report Global Average Pooling + Dense

Balanced with audio augmentation

Using GlobalAveragePooling have been achieved a lower lever of overfitting.

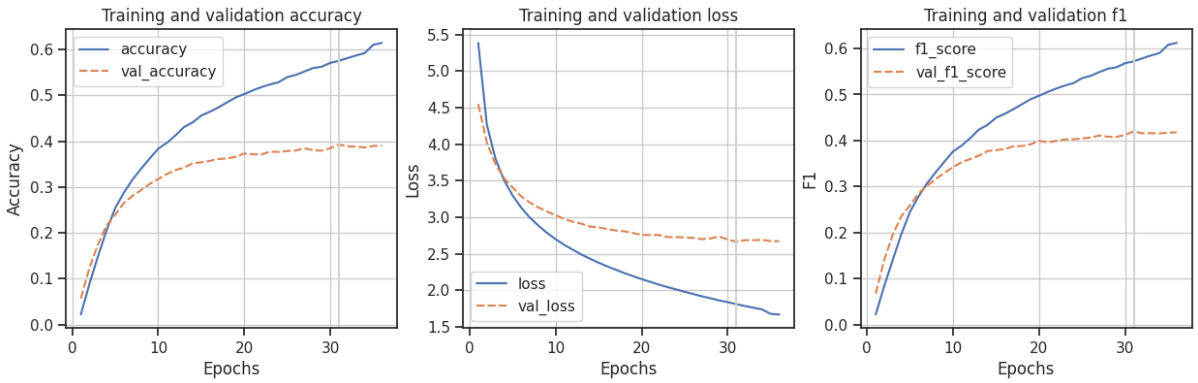


Figure 16: Accuracy, loss e F1-score of train and validation set

	Loss	Accuracy	F1 score
Training	1.2963	0.7172	0.7162
Validation	2.6172	0.4037	0.4309
Test		0.3963	0.4234

Table 11: Report GlobalAveragePooling + Dense

As it is possible to observe, the overfitting is much less than the one with the Flatten layer. Moreover, it is in overfitting as the model with the SpecAugment balanced data but it has better performance. For this reason, the dataset balanced on the audio have been chosen.

5.1.4 Reducing overfitting

In the previous experiment the validation results were considerably lower than the ones of the train set, so the model is not able to generalize well. In order to reduce the overfitting, different techniques and experiments have been tried.

Adding Dropout layer

In order to reduce the overfitting that is possible to observe from the curve of the F1-score of the previous experiment, a dropout layer have been added. The overfitting hasn't decrease as expected.

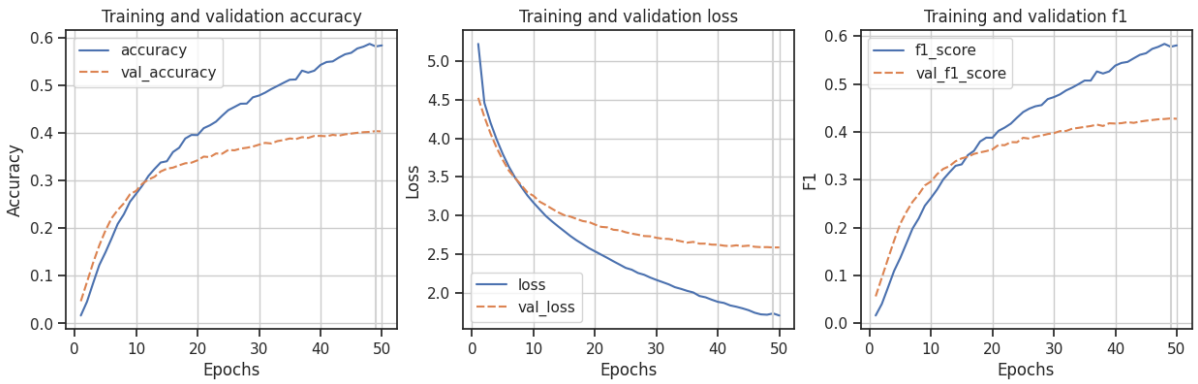


Figure 17: Accuracy, loss e F1-score of train and validation set

	Loss	**Accuracy**	**F1 score**
Training	1.7024	0.5840	0.5807
Validation	2.5852	0.4032	0.4272
Test		0.3939	0.4201

Table 12: Report GlobalAveragePooling + Dropout

Adding Batch Normalization layer

The BatchNormalization layer has been added to the previous experiment in order to reduce the overfitting but the results are similar to the ones of the previous experiment.

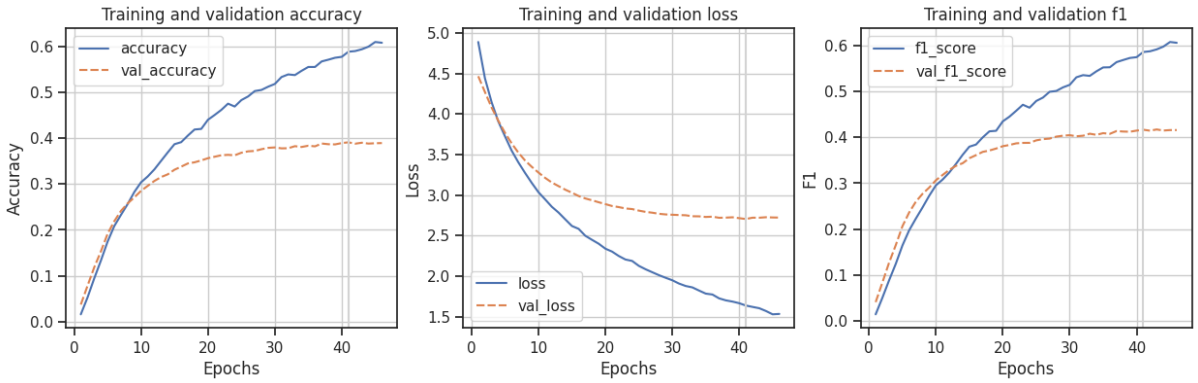


Figure 18: Accuracy, loss e F1-score of train and validation set

	Loss	Accuracy	F1 score
Training	0.6081	0.6081	0.6059
Validation	2.7271	0.3893	0.4157
Test		0.3881	0.4146

Table 13: Report GlobalAveragePooling + Dense + Dropout + BatchNormalization

Adding Dense layer

Adding another Dense layer the overfitting has been reduced a lot even if the performance has been decreased.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem (  (None, 224, 224, 3)       0
 SlicingOpLambda)

 tf.nn.bias_add (TFOpLambda  (None, 224, 224, 3)       0
 )

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d (  (None, 512)               0
 GlobalAveragePooling2D)

 batch_normalization (Batch  (None, 512)               2048
 Normalization)

 dense (Dense)               (None, 256)               131328

 dense_1 (Dense)             (None, 128)               32896

 dropout (Dropout)           (None, 128)               0

 prediction (Dense)          (None, 100)               12900

=================================================================
Total params: 14893860 (56.82 MB)
Trainable params: 178148 (695.89 KB)
Non-trainable params: 14715712 (56.14 MB)
_________________________________________________________________
```

Figure 19: Summary

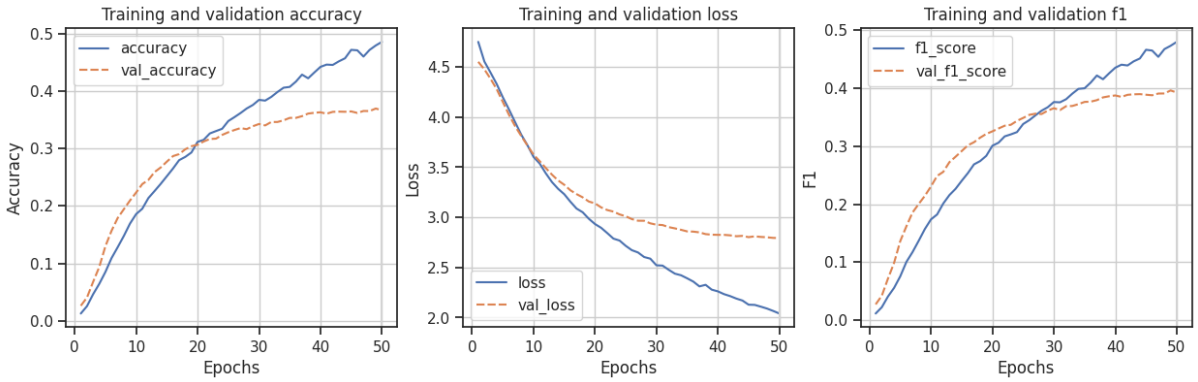


Figure 20: Accuracy, loss e F1-score of train and validation set

	Loss	**Accuracy**	**F1 score**
Training	2.0909	0.4726	0.4655
Validation	2.8289	0.3656	0.3908
Test		0.3636	0.3899

Table 14: Report Model 1

The F1-score has higher values than the accuracy; indeed the test shows that classes that have a very high number of data are recognized correctly while those that have an average number of data the performance are rather lower.

5.1.5 Fine tuning

Fine tuning have been performed in order to increment the performance of the model. However, the results have not been as expected.

Last layer unfrozen

Starting from the previous experiment, the last layer has been unfrozen. The F1-score has increased but the overfitting has increased too. Indeed, the validation loss is much higher than the training loss.

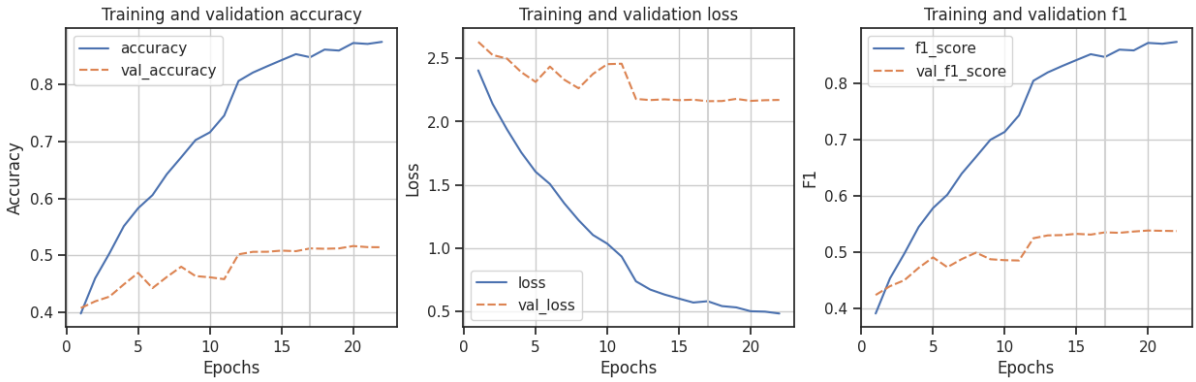


Figure 21: Accuracy, loss e F1-score of train and validation set

	Loss	**Accuracy**	**F1 score**
Training	0.4838	0.8741	0.8734
Validation	2.1716	0.5143	0.5373
Test		0.5132	0.5402

Table 15: Report fine tuning last layer

Last block unfrozen

The results of this experiment are not satisfying since it is affected by overfitting and the performance is lower than previously. For this reason, no further experiments have been performed.

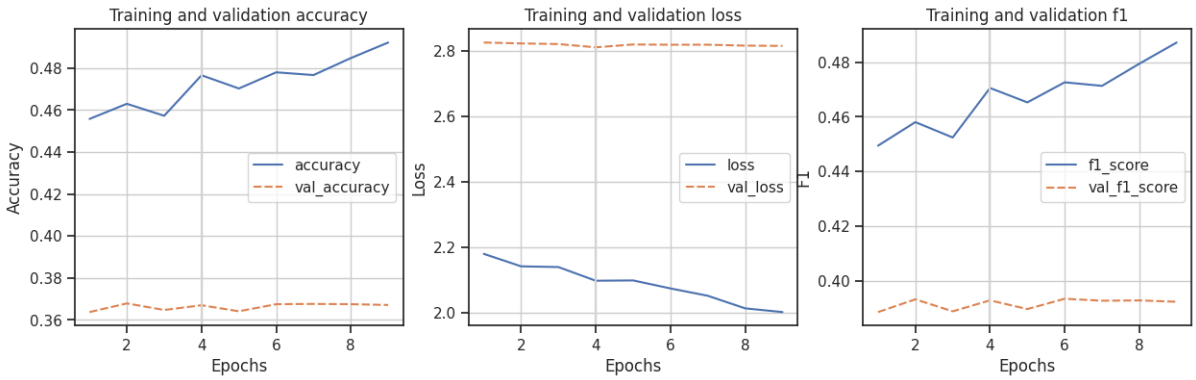


Figure 22: Accuracy, loss e F1-score of train and validation set

	Loss	Accuracy	F1 score
Training	2.0032	0.4922	0.4872
Validation	2.8166	0.3672	0.3925
Test		0.3685	0.3952

Table 16: Report fine tuning last block

5.1.6 Removing last block

Since the data of which is composed this dataset is really different from the images of Imagenet and the model suffers from overfitting, the removal of a block has been tried. After removing the last block (block_5), different experiments for feature extraction have been performed.

Feature extraction

In the first experiment shown in 23 has been added, after the global average pooling layer, batch normalization, two dense layers and dropout. In the second experiment in 24, after the global average pooling layer, batch normalization, dense layer and dropout have been inserted before the output layer while in the last experiment in 25 only a global average pooling and a dense layer.

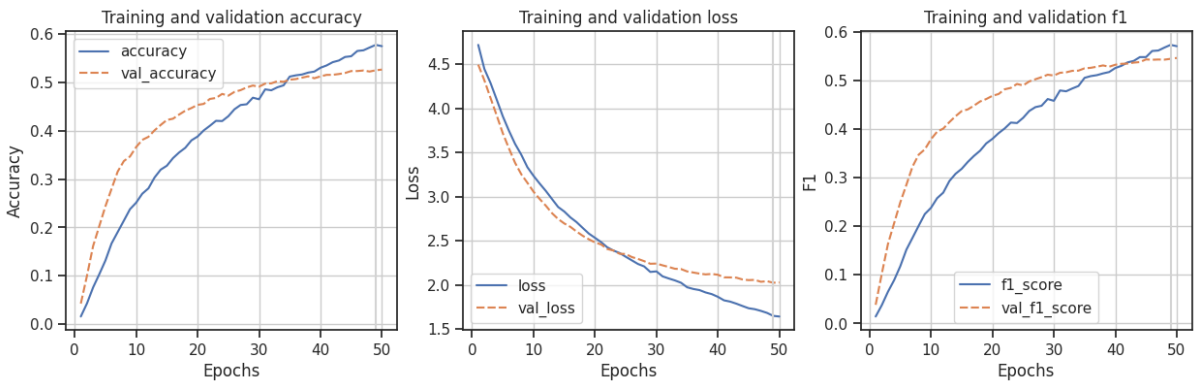


Figure 23: Accuracy, loss e F1-score of train and validation set

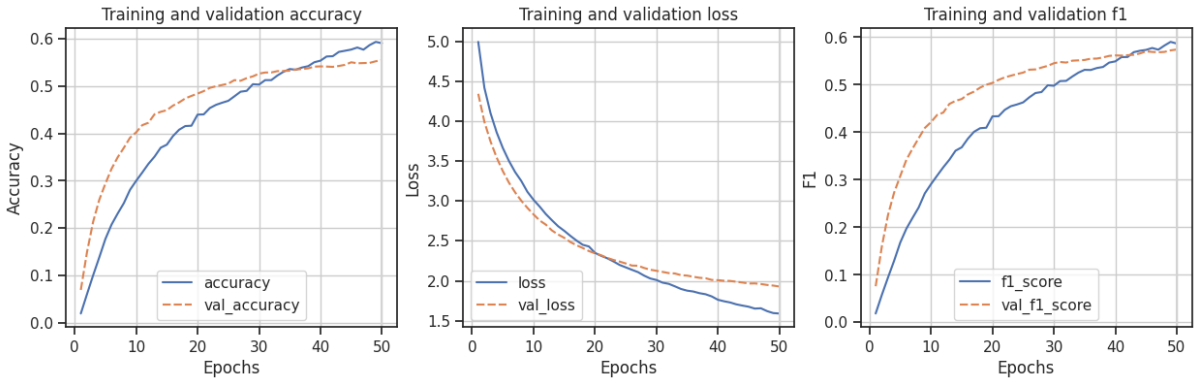


Figure 24: Accuracy, loss e F1-score of train and validation set

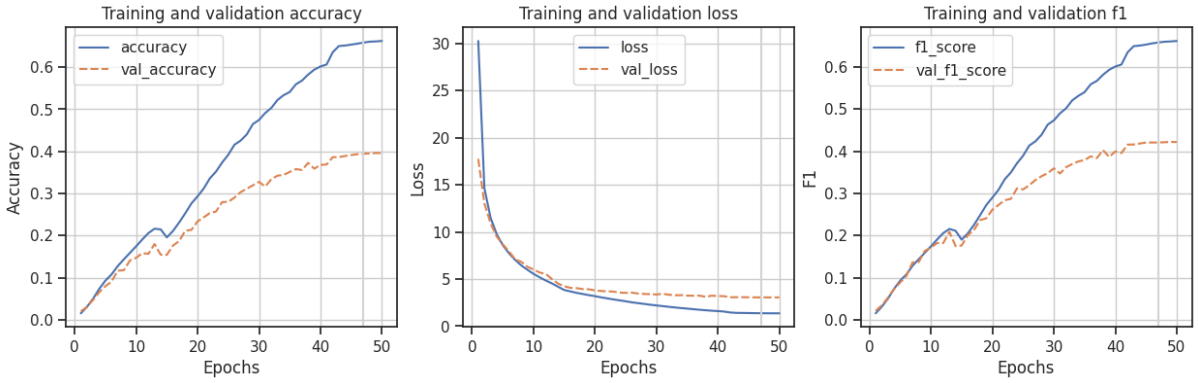


Figure 25: Accuracy, loss e F1-score of train and validation set

	Loss	**Accuracy**	**F1 score**
First experiment	1.6403	0.5745	0.5708
Second experiment	1.5894	0.5899	0.5867
Third experiment	1.3640	0.6622	0.6613

Table 17: Report Train

	Loss	**Accuracy**	**F1 score**
First experiment	2.0269	0.5261	0.5466
Second experiment	1.9282	0.5544	0.5742
Third experiment	3.0530	0.3960	0.4217

Table 18: Report Validation

	Loss	Accuracy	F1 score
First experiment	2.0777	0.5215	0.5441
Second experiment	1.9798	0.5492	0.5700
Third experiment	3.1526	0.3771	0.4075

Table 19: Report Test

The first two results are promising; most of the time they are in underfitting, as it is possible to observe in the F1-score plot, but they have achieved good results on both validation and train set. The third one, instead, suffers from overfitting; indeed, the train loss is 1.3640 while the validation loss is 3.0530.

Fine tuning

Since the highest layers of the network have most likely learned to recognize features that are more specific to the original dataset, we though that performing fine-tuning might be beneficial to the performance of the model. However, unfortunately, we were unable to achieve satisfactory results. Starting from the second experiment, fine tuning have been performed. First of all, the last block have been unfrozen on the second previous experiment.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 tf.__operators__.getitem_1  (None, 224, 224, 3)       0
  (SlicingOpLambda)

 tf.nn.bias_add_1 (TFOpLamb  (None, 224, 224, 3)       0
 da)

 base (Functional)           (None, 14, 14, 512)       7635264

 global_average_pooling2d_1  (None, 512)               0
  (GlobalAveragePooling2D)

 batch_normalization_1 (Bat  (None, 512)               2048
 chNormalization)

 dense_2 (Dense)             (None, 256)               131328

 dropout_1 (Dropout)         (None, 256)               0

 prediction (Dense)          (None, 100)               25700

=================================================================
Total params: 7794340 (29.73 MB)
Trainable params: 7533156 (28.74 MB)
Non-trainable params: 261184 (1020.25 KB)
_________________________________________________________________
```

Figure 26: Summary fine tuning on second experiment

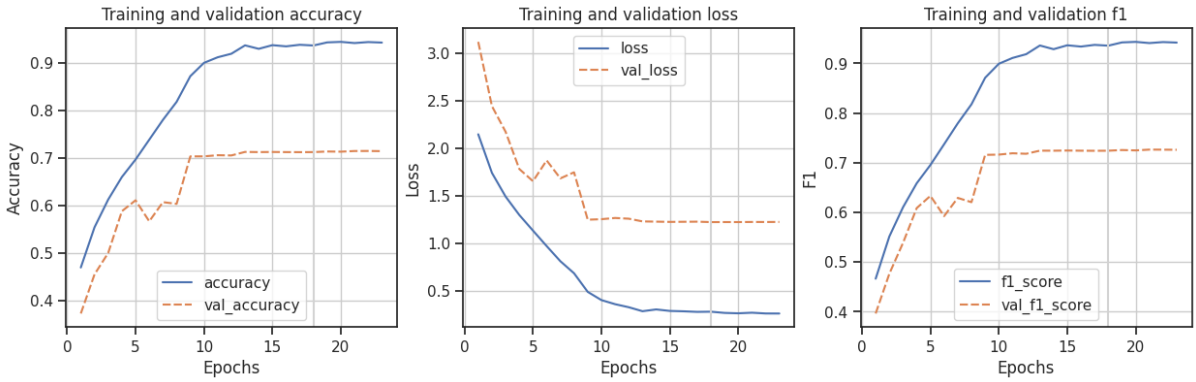


Figure 27: Accuracy, loss e F1-score of train and validation set

	Loss	**Accuracy**	**F1 score**
Training	0.2626	0.9420	0.9419
Validation	1.2265	0.7143	0.7261
Test		0.7111	0.7254

Table 20: Report fine-tuning last block

The F1-score has improved a lot but the overfitting has increased. The results are not already satisfying. For this reason, another experiment have been performed with the last two blocks unfrozen.

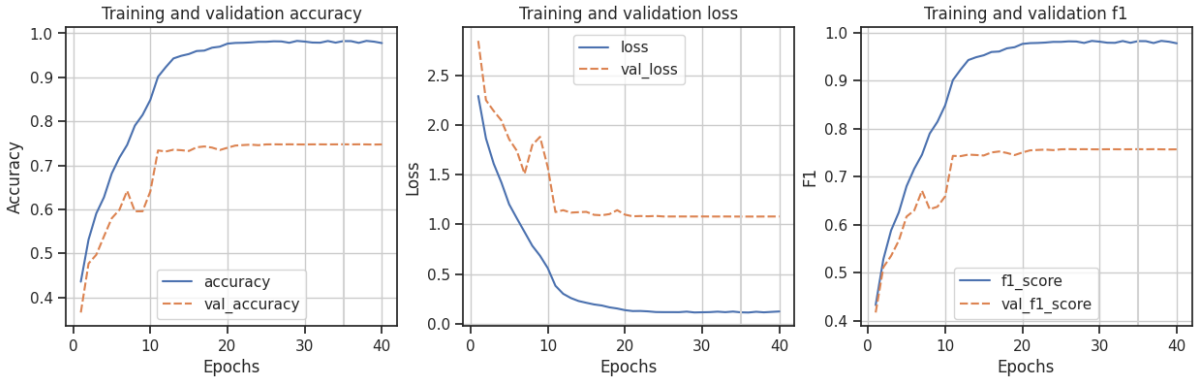


Figure 28: Accuracy, loss e F1-score of train and validation set

	Loss	Accuracy	F1 score
Training	0.1233	0.9779	0.9779
Validation	1.0793	0.7476	0.7570
Test		0.7401	0.7524

Table 21: Report fine-tuning last block

The overfitting is still present but the performance has increased. Moreover, the validation loss has decreased. Unfortunately with our experiments, despite the different combinations of techniques and trials, we were unable to obtain better results. Moreover, as expected, the majority classes give satisfying results on the test set, while the minority and some of the mid classes not. Indeed, thrnig1, wlwwar and eubeat1, that are the classes with the highest number of spectrograms, have achieved an F1-score of 0.89 while the minority class amesun2 has achieved 0.21.

5.2 MobileNetV2

MobileNetV2 is a remarkably efficient convolutional neural network (CNN) model designed for image processing on mobile devices and limited computational resources. It was presented in 2018 in the paper 'MobileNetV2: Inverted Residuals and Linear Bottlenecks' by Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov and Liang-Chieh Chen [9].

The increasing use of mobile devices and the explosion of visual content consumption have created a need for deep learning models that can offer high performance while being lightweight and resource efficient. MobileNetV2 is an answer to this need, providing an effective solution for image processing on computationally constrained devices.

The main feature of MobileNetV2 is the use of a network architecture called 'inverted residuals' and 'linear bottlenecks'. This architecture achieves greater efficiency by reducing the number of parameters and the computational burden required during model execution. In particular, the use of inverted residual blocks reduces computational complexity compared to traditional approaches, allowing similar or higher performance to be achieved with fewer parameters.

MobileNetV2 consists of several deep convolution layers, followed by normalisation, activation and pooling. The model architecture also features a 'linear bottleneck' layer, which reduces the dimensionality of the data and enables further computational savings. This combination of deep layers, inverted residual blocks and linear bottlenecks allows MobileNetV2 to achieve efficient and accurate image representation.

One of the main goals of MobileNetV2 is to balance performance and efficiency. Despite being a lightweight model, it is able to achieve good classification performance on various datasets, including ImageNet. MobileNetV2 has been trained on a wide variety of images to learn a wide range of features, making it suitable for a variety of machine vision tasks.

In addition, MobileNetV2 is designed for knowledge transfer learning, allowing knowledge learned from pre-trained models to be used to adapt to specific tasks with less training data. This feature is particularly useful when you have limited resources or want to quickly develop a model for a specific application.

5.2.1 Balancing dataset

All the following experiments with different techniques of audio augmentation are done on the simple model with a *Flatten* layer followed by a *Dense* layer witth dimensionality 256.

The *SoftMax* activation function has been used after the fully connected layer to perform the 100-class classification. The *ModelCheckpoint* callback has been used to store intermediate models in case of Google Colab crashes and also the *EarlyStopping* for monitoring the validation loss with a patience of 5 epochs. The optimizer used is Adam with a learning rate of 0.0001.

Unbalanced classes

This experiment represents the base case of the unbalanced training set. In this case the model runs with an additional argument *class_weights* in order to give more importance to the minority classes.

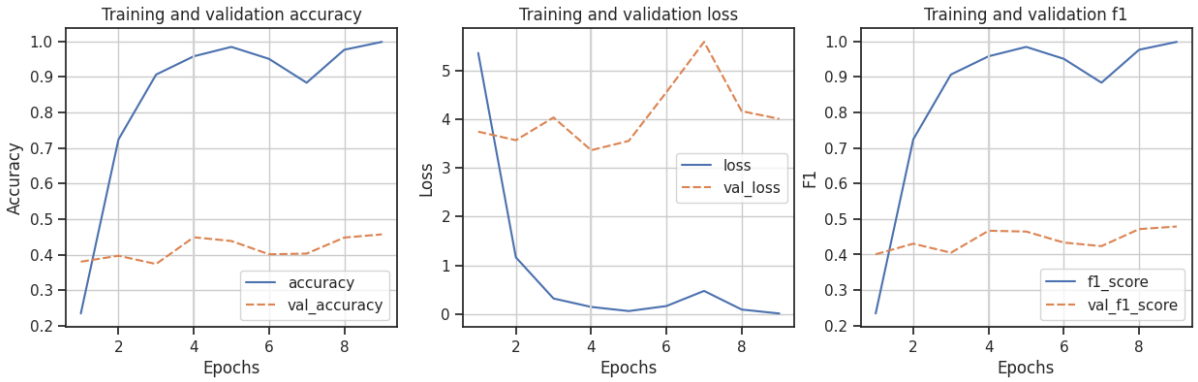


Figure 29: Training and validation results

	Loss	Accuracy	F1 score
Training	0.0110	0.9985	0.9985
Validation	4.0030	0.4572	0.4790
Test		0.4450	0.4634

Table 22: Performance summary

SpecAugment technique

Offline balancing in which the minority classes has been augmented before the training with the *SpecAugment* technique seen in 3.4.1.

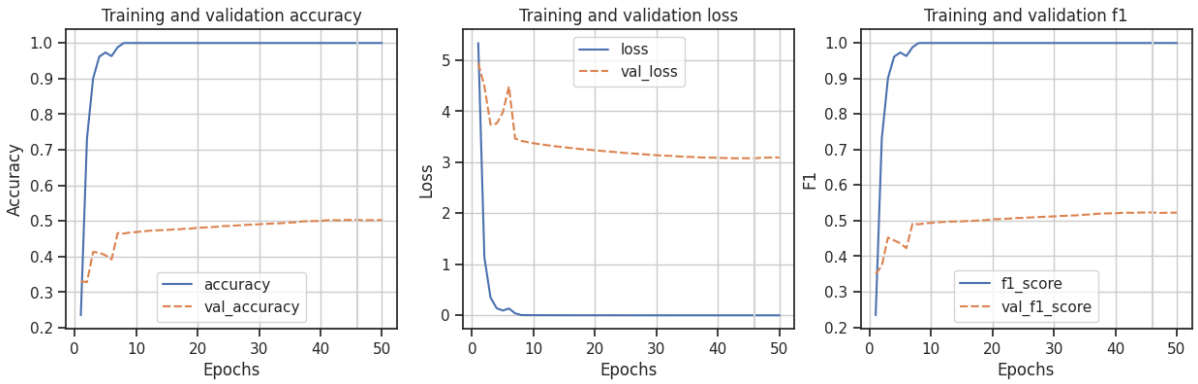


Figure 30: Training and validation results

	Loss	Accuracy	F1 score
Training	1.9476e-04	1.0000	1.0000
Validation	3.0954	0.4986	0.5194
Test		0.4930	0.5141

Table 23: Performance summary

Classical image augmentation

Offline balancing in which the minority classes has been augmented before the training with the 4 techniques described in 3.4.2.

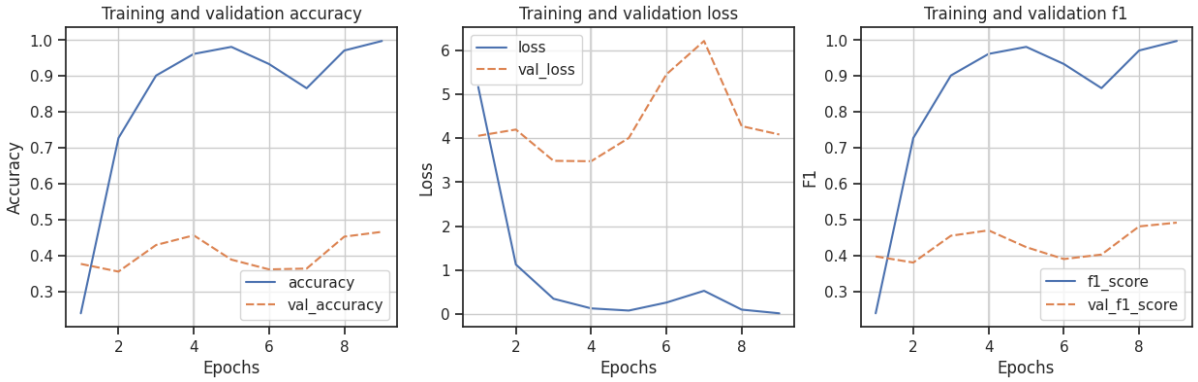


Figure 31: Training and validation results

	Loss	Accuracy	F1 score
Training	0.0127	0.9978	0.9978
Validation	4.0883	0.4671	0.4919
Test		0.4382	0.4546

Table 24: Performance summary

Audio augmentation

Offline balancing in which the minority classes has been augmented before the training with the augmentation on audio recordings explained in 3.4.3.

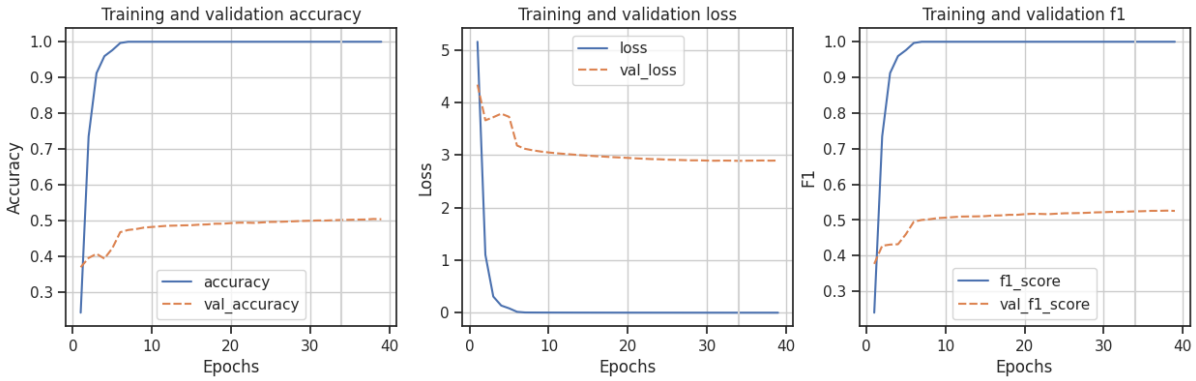


Figure 32: Training and validation results

	Loss	**Accuracy**	**F1 score**
Training	1.9928e-04	1.0000	1.0000
Validation	2.8943	0.5031	0.5256
Test		0.4885	0.5120

Table 25: Performance summary

The techniques of augmentation proposed for balancing the training set are very different. The best results are those given by *SpecAugment* and audio augmentation. The last one has been preferred because of the slightly better values in performance. From now on, in all the experiments that follows the training set used is the one balanced with audio augmentation.

5.2.2 Reducing overfitting

The proposed model suffers from overfitting. This can be seen in the large difference between the training and validation curves, which is a symptom of the network's inability to correctly do the classification. Below are reported various experiments that have been carried out to try to reduce this problem as much as possible.

GlobalAveragePooling + *Dense* (256)

First of all the *Flatten* layer has been replaced with a *GlobalAveragePooling* layer to see if it achieves better results.

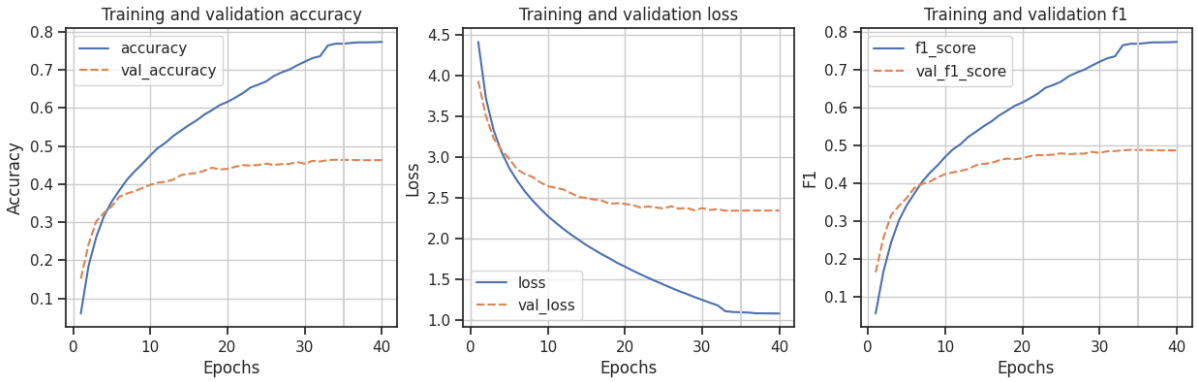


Figure 33: Training and validation results

	Loss	Accuracy	F1 score
Training	1.0836	0.7730	0.7727
Validation	2.2441	0.4627	0.4868
Test		0.4452	0.4711

Table 26: Performance summary

GlobalAveragePooling + Dense (256) + *Dropout*

A Dense layer with dimensionality 256 has been added right after the pooling layer for regularization purpose.

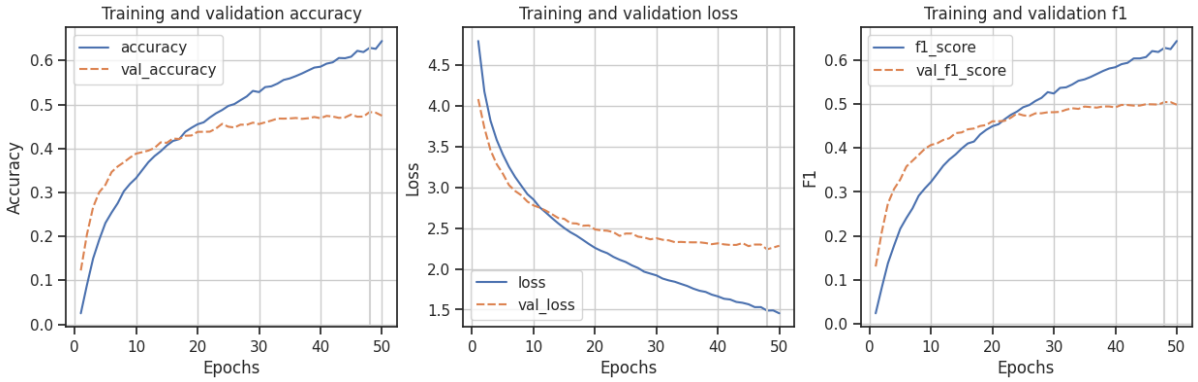


Figure 34: Training and validation results

	Loss	Accuracy	F1 score
Training	1.4543	0.6445	0.6429
Validation	2.2837	0.4746	0.4982
Test		0.4710	0.4938

Table 27: Performance summary

GlobalAveragePooling + *Dense* (256, ReLu)

The ReLu activation function has been added to the Dense layer with the dimensionality as before.

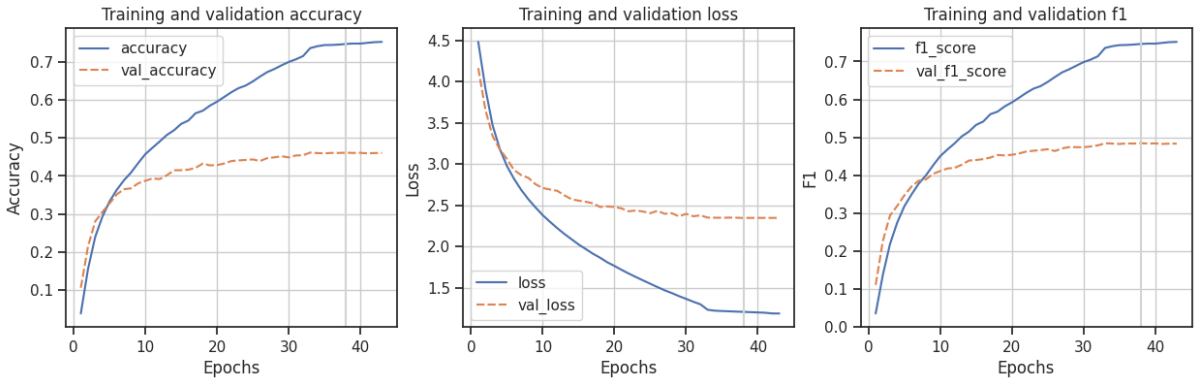


Figure 35: Training and validation results

	Loss	Accuracy	F1 score
Training	1.1902	0.7520	0.7517
Validation	2.3483	0.4602	0.4835
Test		0.3117	0.3256

Table 28: Performance summary

GlobalAveragePooling + *Dense* (256, ReLu) + *Dropout*

This experiments tests the results of applying all the precedent solutions together.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 224, 224, 3)]     0

tf.math.truediv_2 (TFOpLam   (None, 224, 224, 3)       0
bda)

tf.math.subtract_2 (TFOpLa   (None, 224, 224, 3)       0
mbda)

mobilenetv2_1.00_224 (Func   (None, 7, 7, 1280)        2257984
tional)

global_average_pooling2d_2   (None, 1280)              0
 (GlobalAveragePooling2D)

dense_4 (Dense)              (None, 256)               327936

dropout_2 (Dropout)          (None, 256)               0

dense_5 (Dense)              (None, 100)               25700

=================================================================
Total params: 2611620 (9.96 MB)
Trainable params: 353636 (1.35 MB)
Non-trainable params: 2257984 (8.61 MB)
_________________________________________________________________
```

Figure 36: Model summary

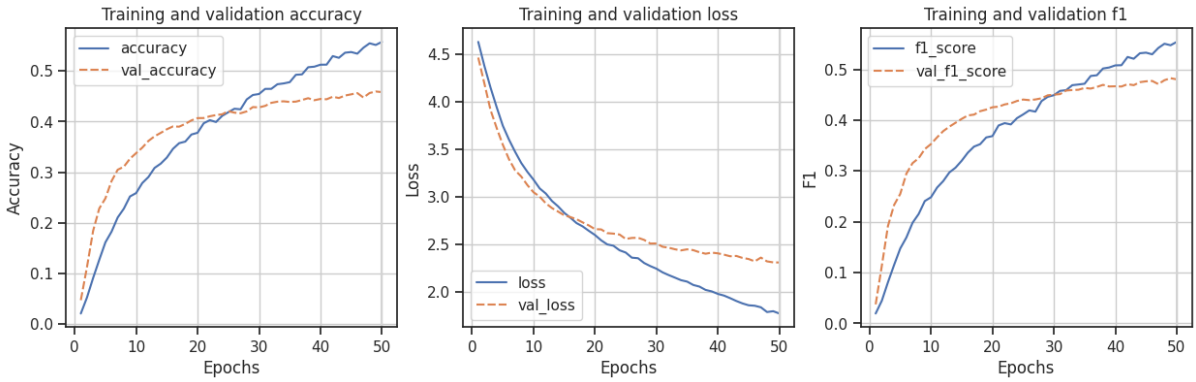


Figure 37: Training and validation results

	Loss	Accuracy	F1 score
Training	1.7768	0.5576	0.5543
Validation	2.3127	0.4583	0.4804
Test		0.4591	0.4822

Table 29: Performance summary

GlobalAveragPooling + 2 *Dense* (512, 256, ReLu) + *Dropout*

In this configuration another *Dense* layer with dimensionality 512 has been added before the one with 256.

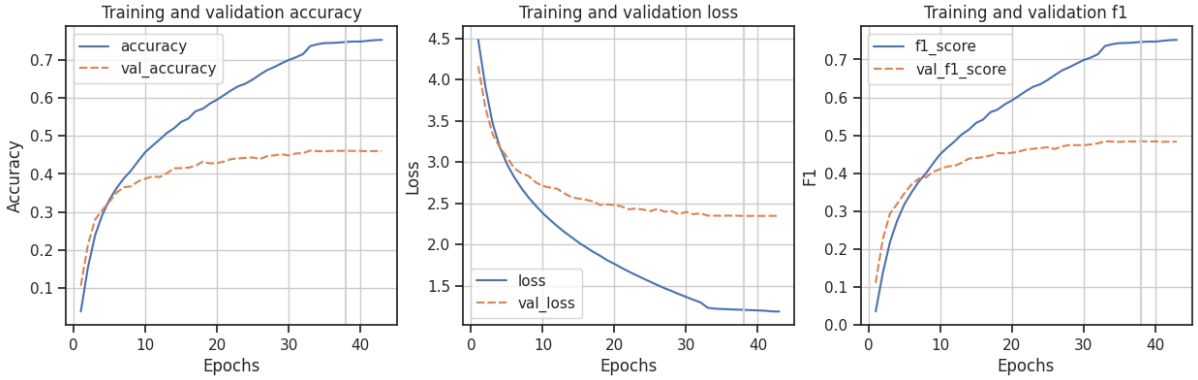


Figure 38: Training and validation results

	Loss	Accuracy	F1 score
Training	1.4076	0.6465	0.6438
Validation	2.2710	0.4675	0.4902
Test		0.4523	0.4783

Table 30: Performance summary

GlobalAveragePooling + *BatchNorm* + *Dense* (256, ReLu) + *Dropout*

In this experiment the *Batch Normalization* has been applied to try to regularize the curve.

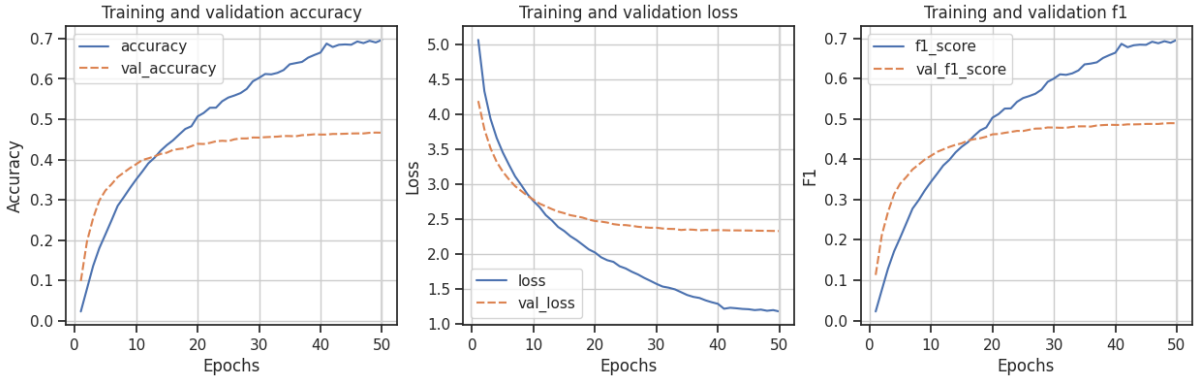


Figure 39: Training and validation results

	Loss	Accuracy	F1 score
Training	1.1762	0.6967	0.6962
Validation	2.3268	0.4666	0.4894
Test		0.4557	0.4817

Table 31: Performance summary

GlobalAveragePooling + BatchNorm + 2 Dense (512, 256, ReLu) *+ Dropout*

Now the *Batch Normalization* is applied to the model with 2 *Dense* layers.

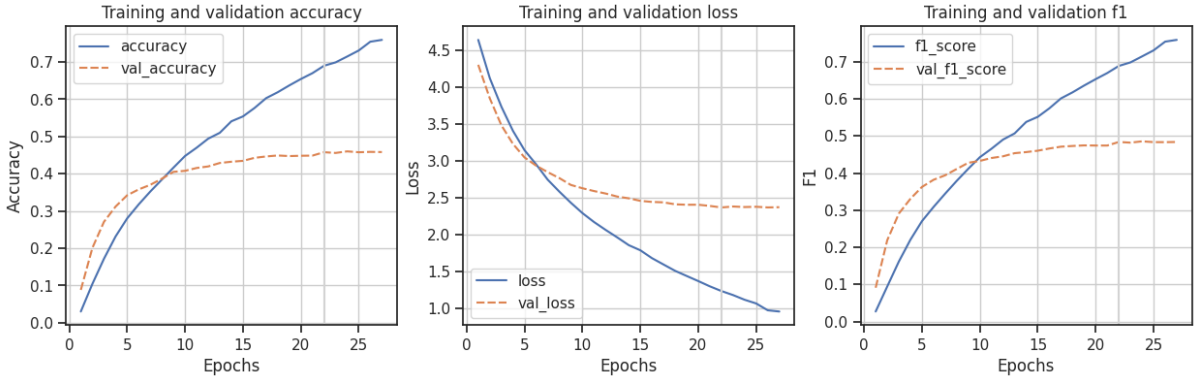


Figure 40: Training and validation results

	Loss	Accuracy	F1 score
Training	0.9540	0.7594	0.7590
Validation	2.3682	0.4582	0.4838
Test		0.4495	0.4753

Table 32: Performance summary

5.2.3 Fine tuning

Between all the experiments shown, the best in terms of reducing overfitting is the *GlobalAveragePooling + Dense* with ReLu + *Dropout* configuration 5.2.2. In fact, the difference between training and validation values is the lowest.

For this reason, the fine tuning technique has been tested on this model with an increasing number of unfrozen blocks. In fact, this approach is a common strategy used in transfer learning to adapt a pre-trained model to to a specific task or dataset and it

also helps speeding up the training phase and fitting better the small datasets as the one exploited for this project. The first way to perform fine tuning is to freeze the weights of the initial layers and then gradually unfreezing from the last blocks allowing the model to learn task-specific features.

1 Block

The following experiment is carried out with only the last block unfrozen.

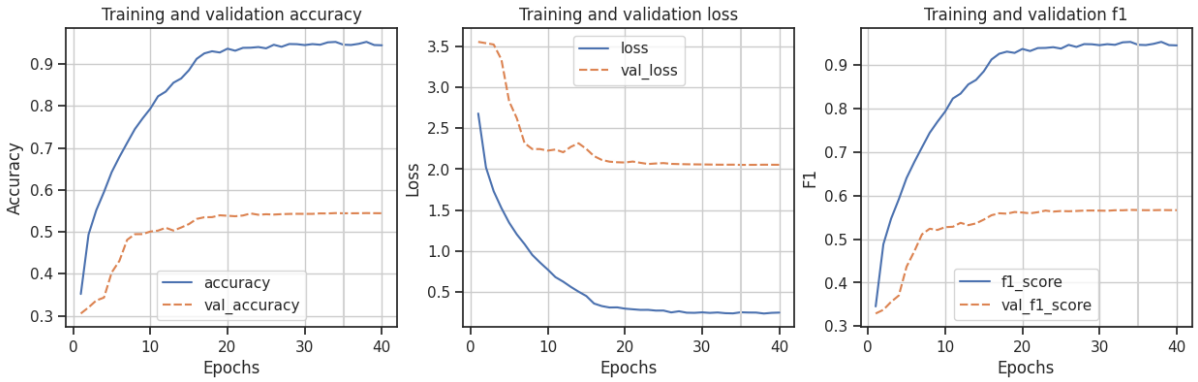


Figure 41: Training and validation results

	Loss	Accuracy	F1 score
Training	0.2453	0.9443	0.9443
Validation	2.0536	0.5443	0.5663
Test		0.5324	0.5551

Table 33: Performance summary

The last model suffers a lot from overfitting, but the f1-score is higher than model 5.2.2, so another experiment that can be done to try to reduce it consists in removing the last blocks of the network and successively doing again the feature extraction and the fine tuning. This approach effectively reduces the model complexity and the number of parameters it needs to learn.

5.2.4 Removing 1 Block

Removing the last block (block_16) the results are the following.

Feature Extraction

Perform feature extraction using the modified model. In this step, freeze the weights of the remaining layers and extract features from the dataset using these freezed layers.

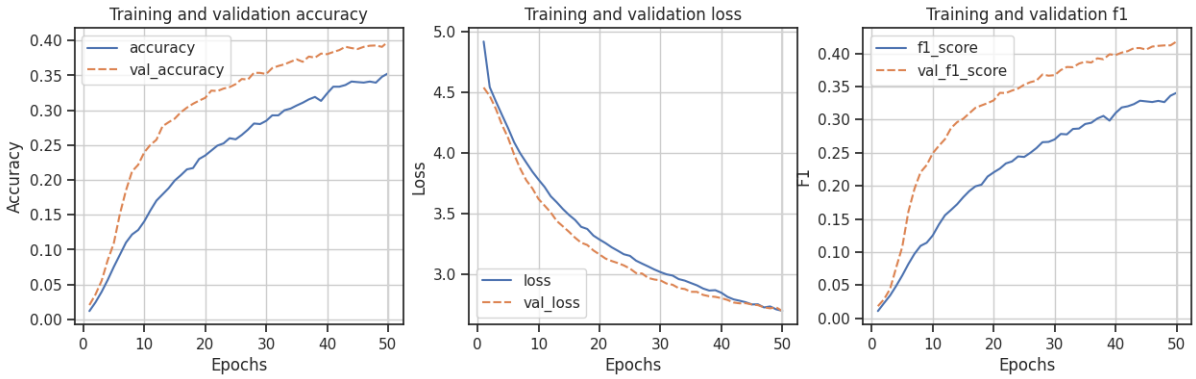


Figure 42: Training and validation results

	Loss	Accuracy	F1 score
Training	2.6944	0.3530	0.3405
Validation	2.6926	0.3986	0.4179
Test		0.3833	0.4040

Table 34: Performance summary

Fine tuning

The performance of the last feature extraction are not so satisfactory but the training phase does not stop because of the earling stopping and probably it could have been achieved good results with a higher number of epochs. This is the reason why a fine-tuning experiment has been tested.

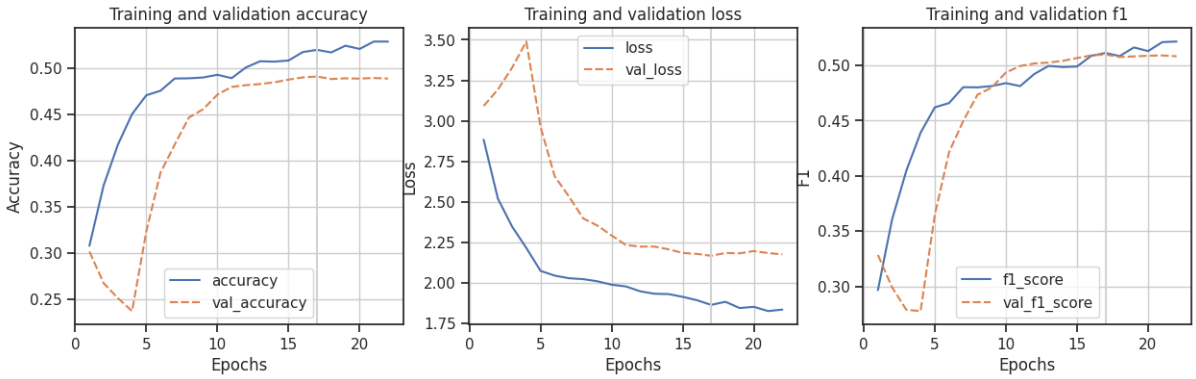


Figure 43: Training and validation results

	Loss	Accuracy	F1 score
Training	1.8359	0.5283	0.5216
Validation	2.1768	0.5083	0.5083
Test		0.4842	0.5041

Table 35: Performance summary

The model achieves 10% more in validation and test so another block can be removed to see if there can be further improvements.

5.2.5 Removing 2 Blocks

The last experiment consists in removing the last two blocks (block_15 and block_16) and unfrozen the last remained (block_14).

Feature Extraction

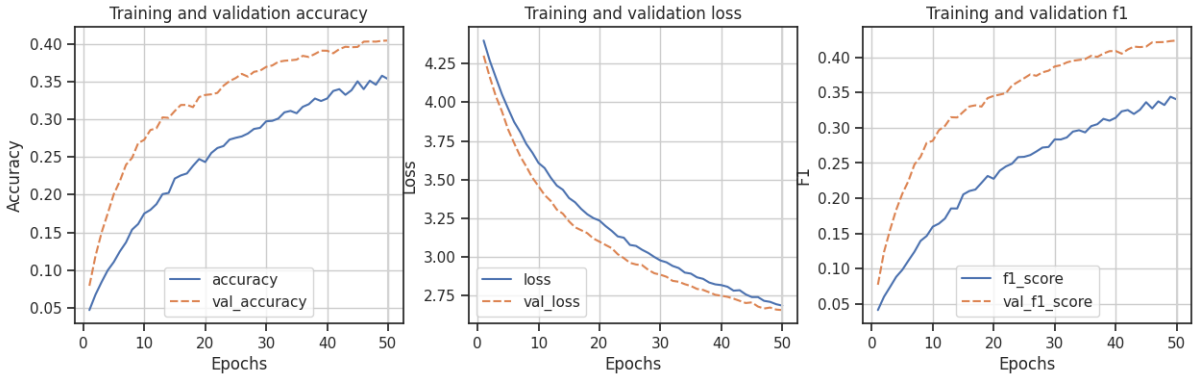


Figure 44: Training and validation results

	Loss	Accuracy	F1 score
Training	2.6857	0.3532	0.3404
Validation	2.6568	0.4044	0.4240
Test		0.3946	0.4140

Table 36: Performance summary

The results is a little improvements in validation and test performances so the fine tune phase has been carried out.

Fine tuning

```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0

tf.math.truediv (TFOpLambd   (None, 224, 224, 3)       0
a)

tf.math.subtract (TFOpLamb   (None, 224, 224, 3)       0
da)

base (Functional)            (None, 7, 7, 160)         1040704

global_average_pooling2d (   (None, 160)               0
GlobalAveragePooling2D)

dense (Dense)                (None, 256)               41216

dropout (Dropout)            (None, 256)               0

dense_1 (Dense)              (None, 100)               25700

=================================================================
Total params: 1107620 (4.23 MB)
Trainable params: 386916 (1.48 MB)
Non-trainable params: 720704 (2.75 MB)
_________________________________________________________________
```

Figure 45: Model summary

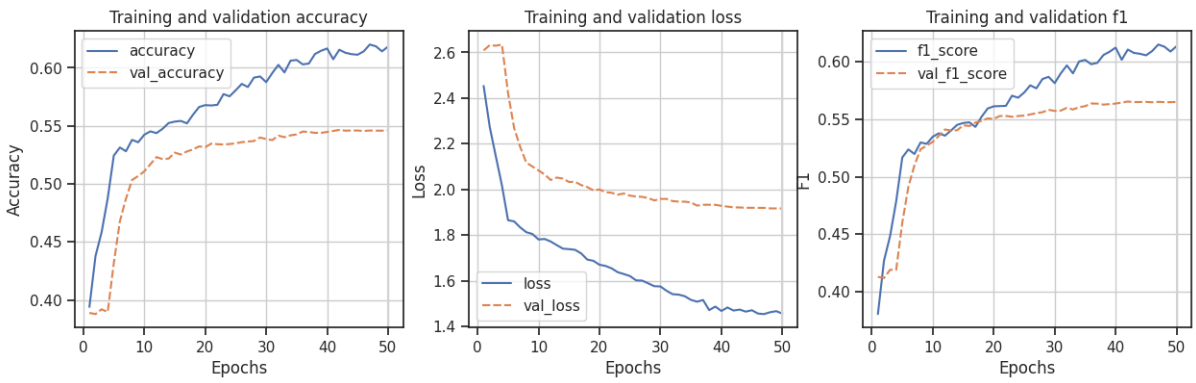


Figure 46: Training and validation results

	Loss	Accuracy	F1 score
Training	1.4566	0.6184	0.6135
Validation	1.9168	0.5460	0.5650
Test		0.5345	0.5544

Table 37: Performance summary

This is the best model obtained after all the process of optimization and it will be considered as the final model of the MobileNetV2 pre-trained network in the following chapters. Considering the results obtained in literature and the limits of a small-device oriented model and the characteristics of the datased used, the f1 score obtained can be considered satisfactory as the loss value and the low overfitting ($\tilde{5}\%$).

5.3 ResNet50

ResNet50, an abbreviation for 'Residual Network 50', is a powerful deep convolutional neural network (CNN) model that has revolutionised the field of computer vision. Introduced in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun [3], this architecture was presented in the paper 'Deep Residual Learning for Image Recognition'. The paper aroused great interest in the scientific community due to its innovations and outstanding results in image processing.

The need to address the problem of the depth of neural networks emerged with the advancement of CNNs. Because of their depth, these networks can suffer from problems such as accuracy degradation during training, due to the phenomenon known as 'gradient vanishing'. This problem makes it difficult to train deep networks that achieve results comparable to or better than those of shallower networks.

ResNet50 has distinguished itself precisely by its ability to overcome this problem through the introduction of residual blocks. Instead of trying to directly learn the mapping of input to output, residual blocks allow the network to learn the residual errors, i.e. the difference between the desired output and the actual output, and propagate them backwards through the network layers. This innovative 'residual learning' technique has been shown to mitigate the problem of gradient vanishing, allowing even deeper neural networks to be trained.

ResNet50 consists of 50 neural layers, including convolution, normalisation, activation and pooling layers ([3]). Its depth allows it to learn a hierarchical representation of images, starting from low-level features such as edges and textures, up to high-level features such as complex objects and structures. This feature has made ResNet50 one of the most effective models in the field of image classification.

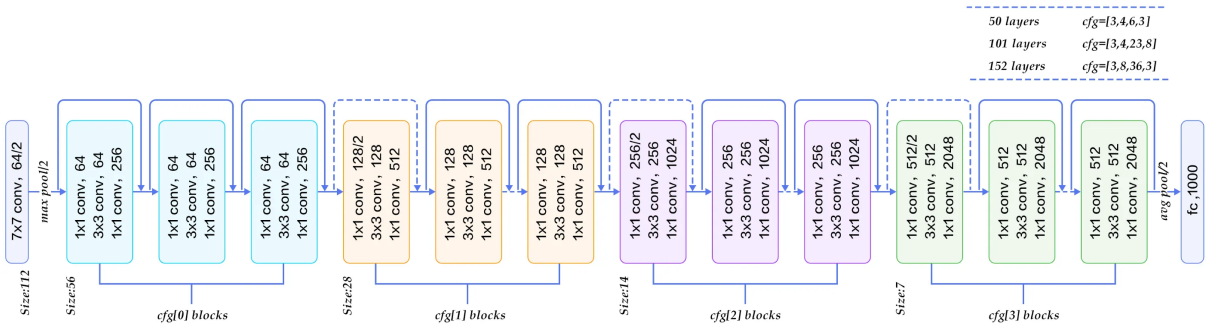


Figure 47: Architecture of ResNet50

ResNet50 was pre-trained on a huge image dataset called ImageNet, which contains millions of images belonging to different categories. Thanks to this pre-training, the model has achieved a state-of-the-art in classification performance, surpassing many of the previous models.

Furthermore, the versatility of ResNet50 goes beyond its excellent image classification capability. Thanks to its architecture and pre-training on ImageNet, the model can be

used as a starting point for knowledge transfer learning. This means that developers and researchers can adapt ResNet50 to specific computer vision tasks, greatly reducing the time and resources required to train a model from scratch.

5.3.1 Base model

The first experiment is the *Flatten*

The proposed model suffers from overfitting. This can be seen in the large difference between the training and validation curves, which is a symptom of the network's inability to correctly do the classification. Below are reported various experiments that have been carried out to try to reduce this problem as much as possible.

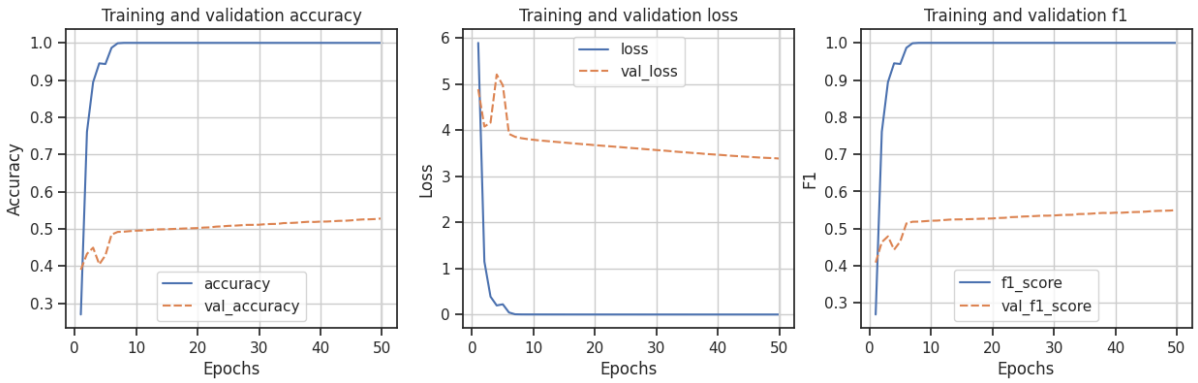


Figure 48: Training and validation results

	Loss	Accuracy	F1 score
Training	1.4174e-05	1.0000	1.0000
Validation	3.3834	0.5281	0.5502
Test		0.5136	0.5377

Table 38: Performance summary

5.3.2 Reducing overfitting

GlobalAveragePooling + Dense(256)

First of all the *Flatten* layer has been replaced with a *GlobalAveragePooling* layer to see if it achieves better results.

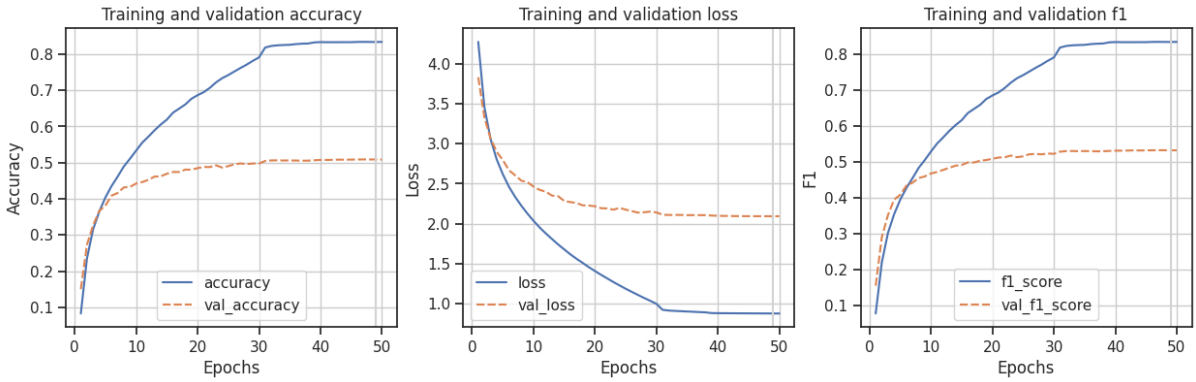


Figure 49: Training and validation results

	Loss	**Accuracy**	**F1 score**
Training	0.8765	0.8338	0.8339
Validation	2.0934	0.5085	0.5318
Test		0.5006	0.5271

Table 39: Performance summary

GlobalAveragePooling + *Dense* (256, ReLu)

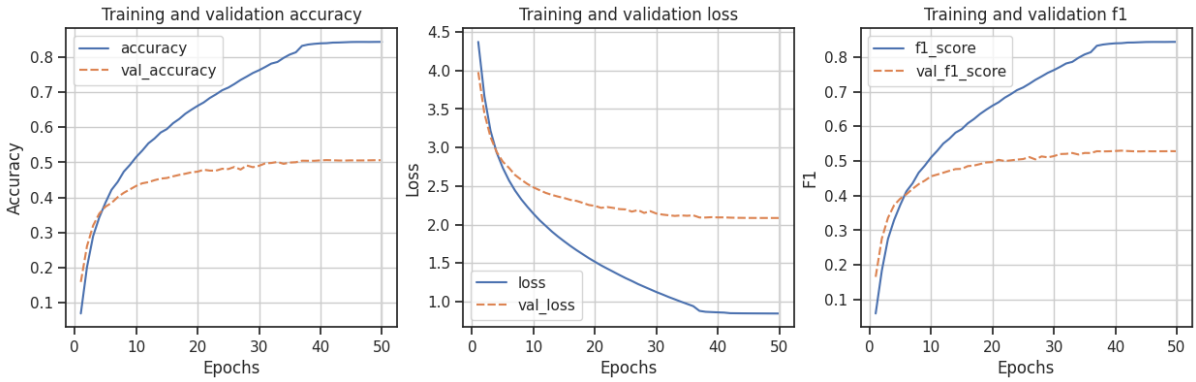


Figure 50: Training and validation results

	Loss	Accuracy	F1 score
Training	0.8452	0.8436	0.8436
Validation	2.0852	0.5061	0.5281
Test		0.4968	0.5237

Table 40: Performance summary

GlobalAveragePooling + *Dense* (256, ReLu) + *Dropout*

The previous models suffer by overfitting. In order to reduce it, some new experiments have been performed.

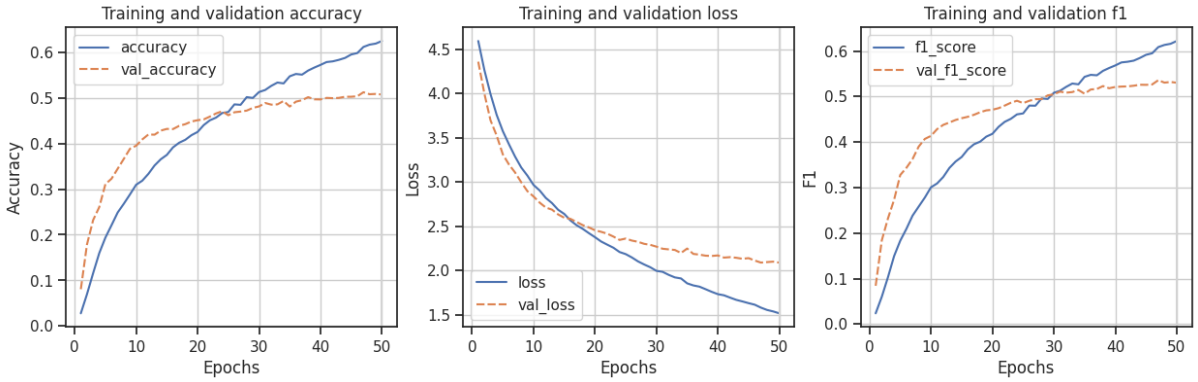


Figure 51: Training and validation results

	Loss	Accuracy	F1 score
Training	1.5187	0.6243	0.6221
Validation	2.0909	0.5071	0.5304
Test		0.4968	0.5245

Table 41: Performance summary

GlobalAveragePooling + Dense (512, ReLu)

In order to achieve better results, the number of neurons of the dense layer have been increased from 256 to 512.

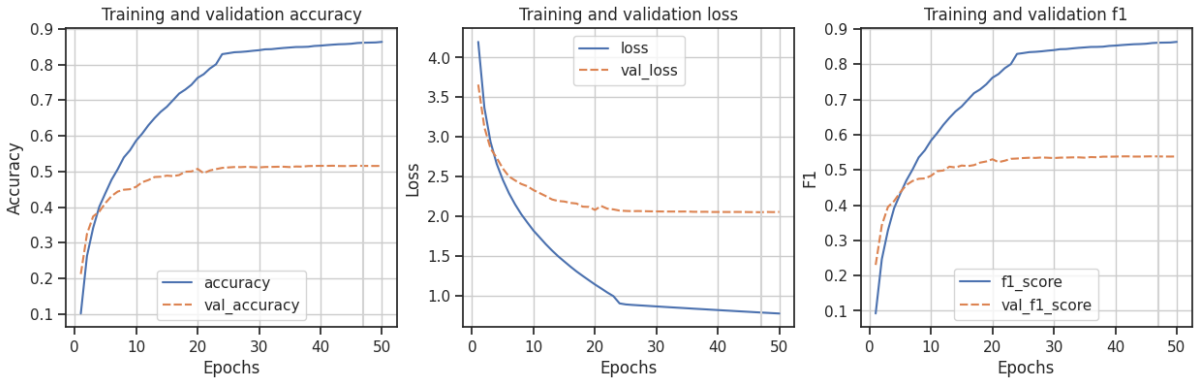


Figure 52: Training and validation results

	Loss	**Accuracy**	**F1 score**
Training	0.7759	0.8636	0.8638
Validation	2.0558	0.5157	0.5384
Test	2.1262	0.5030	0.5313

Table 42: Performance summary

GlobalAveragePooling + Dense (512, ReLu) + Dropout

A dropout layer have been added for reduce the overfitting that affects the previous model.

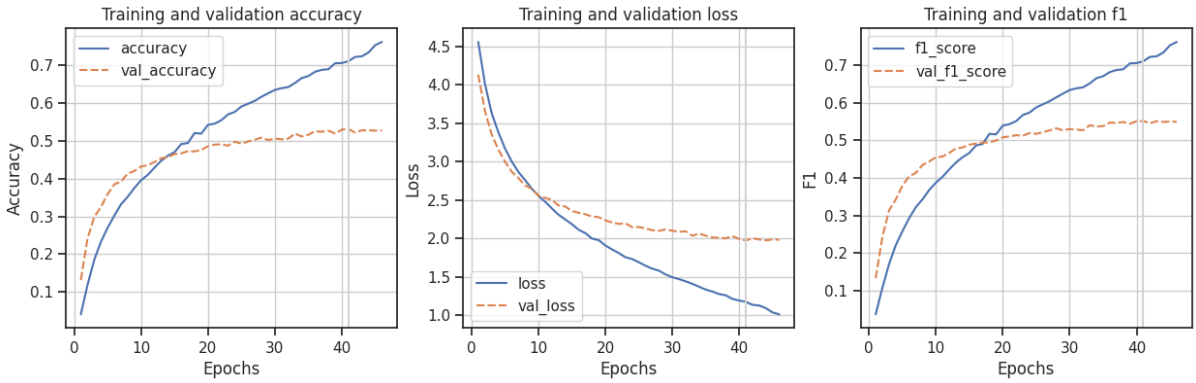


Figure 53: Training and validation results

	Loss	Accuracy	F1 score
Training	1.0136	0.7616	0.7608
Validation	1.9831	0.5275	0.5490
Test		0.4815	0.5096

Table 43: Performance summary

Adding the dropout layer has helped reducing overfitting but still the validation loss is considerably higher than the train loss

GlobalAveragePooling + Dense(512,ReLu) + Dense (256, ReLu) + Dropout

The results shown in 57 highlight that F1-score and loss of the validation are almost the same of the previous experiment but the overfitting have been reduced.

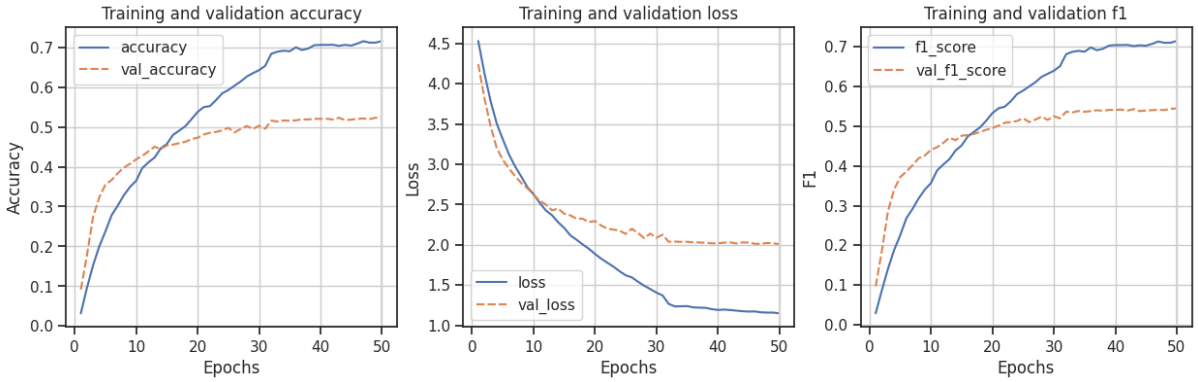


Figure 54: Training and validation results

	Loss	Accuracy	F1 score
Training	1.1513	0.7155	0.7147
Validation	2.0119	0.5241	0.5449
Test		0.5089	0.5336

Table 44: Performance summary

GlobalAvgPooling + BatchNorm + Dense(512) + Dense(256) + Dropout

```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem (   (None, 224, 224, 3)       0
SlicingOpLambda)

tf.nn.bias_add (TFOpLambda   (None, 224, 224, 3)       0
)

resnet50 (Functional)        (None, 7, 7, 2048)        23587712

global_average_pooling2d (   (None, 2048)              0
GlobalAveragePooling2D)

batch_normalization (Batch   (None, 2048)              8192
Normalization)

dense (Dense)                (None, 512)               1049088

dense_1 (Dense)              (None, 256)               131328

dropout (Dropout)            (None, 256)               0

predictions (Dense)          (None, 100)               25700

=================================================================
Total params: 24802020 (94.61 MB)
Trainable params: 1210212 (4.62 MB)
Non-trainable params: 23591808 (90.00 MB)
_________________________________________________________________
```

Figure 55: Summary

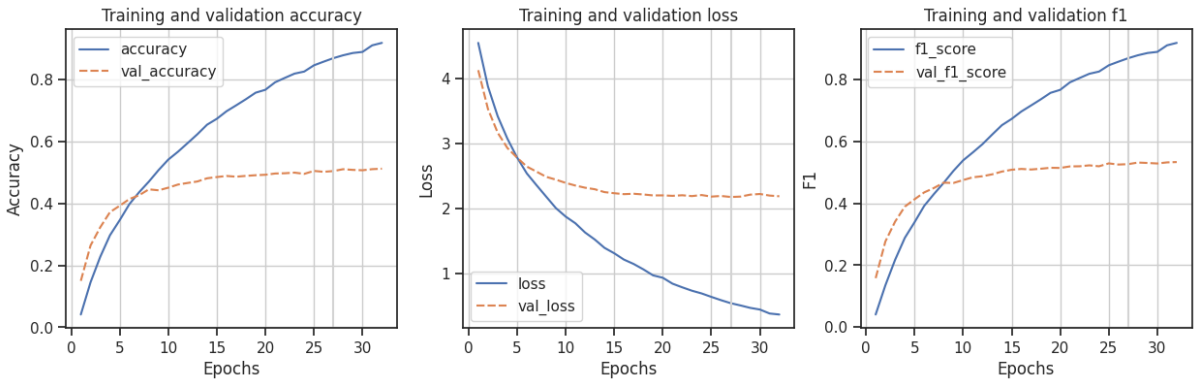


Figure 56: Training and validation results

	Loss	Accuracy	F1 score
Training	0.3646	0.9172	0.9171
Validation	2.1834	0.5115	0.5328
Test		0.4953	0.5228

Table 45: Performance summary

5.3.3 Fine tuning - Last block

Since the highest layers of the network have most likely learned to recognize features that are more specific to the original dataset, we thought that performing fine-tuning might be beneficial to the performance of the model. For this reason, the last block has been unfrozen. However, unfortunately, we were unable to achieve satisfactory results. Fine-tuning has been performed on the in 53. As expected the F1-score on the validation set has increased but the overfitting has increased too.

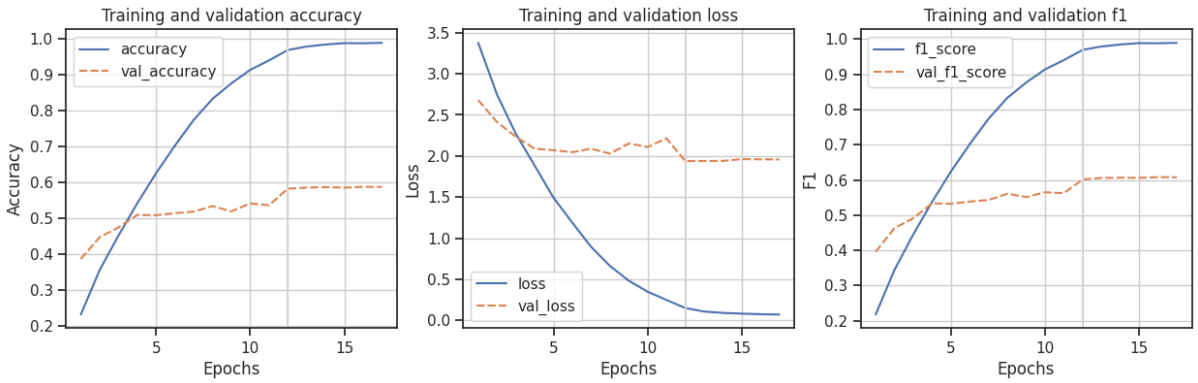


Figure 57: Training and validation results

	Loss	Accuracy	F1 score
Training	0.0700	0.9893	0.9893
Validation	1.9575	0.5871	0.6074
Test		0.5688	0.5908

Table 46: Performance summary

5.3.4 Remove last blocks

ResNet50 has been trained on ImageNet that it's really different from our dataset. For this reason, we have tried to remove some of the last blocks in order to achieve better results by removing the blocks that are more specialized on the ImageNet dataset. Again, we performed several experiments in both feature extraction and fine-tuning. Below, we report the ones that gave us the best results.

Remove blocks - Feature Extraction

After removing the relative last blocks, feature extraction have been performed adding global average pooling, two dense layers and a dropout of 0.5.

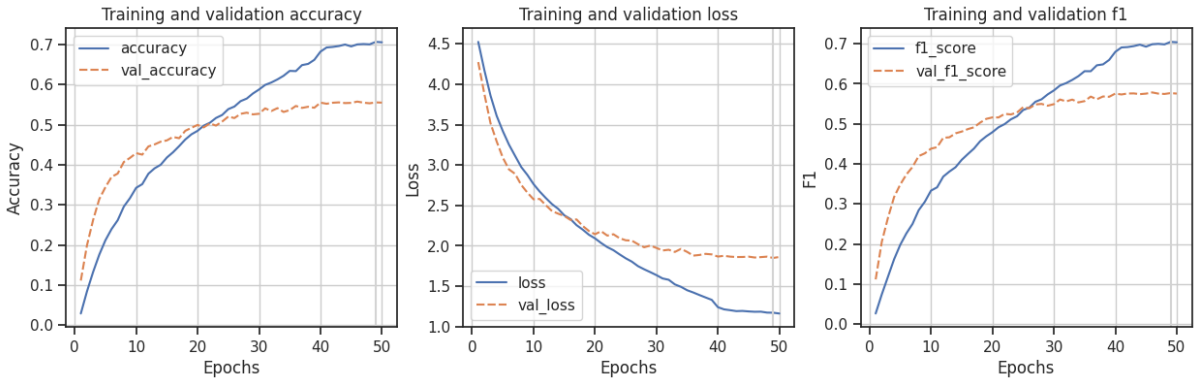


Figure 58: Training and validation results after removing last block (conv5_block3)

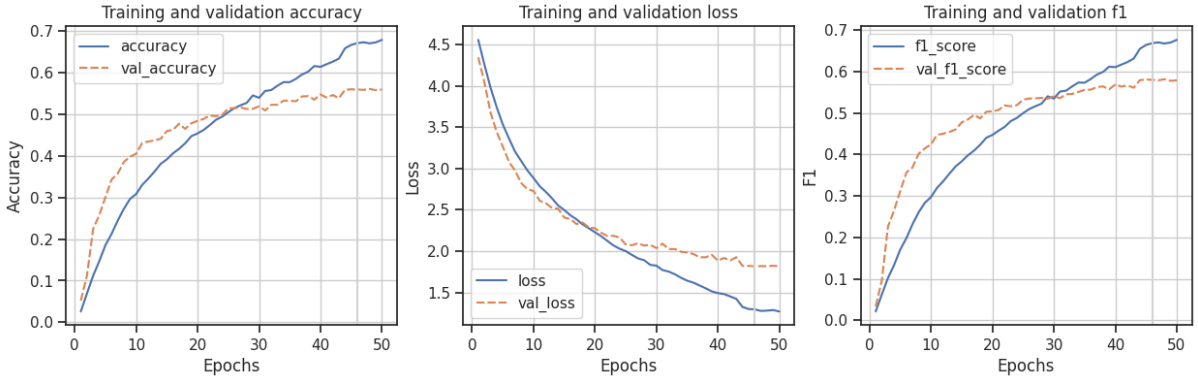


Figure 59: Training and validation results after removing last two blocks (conv5_block2)

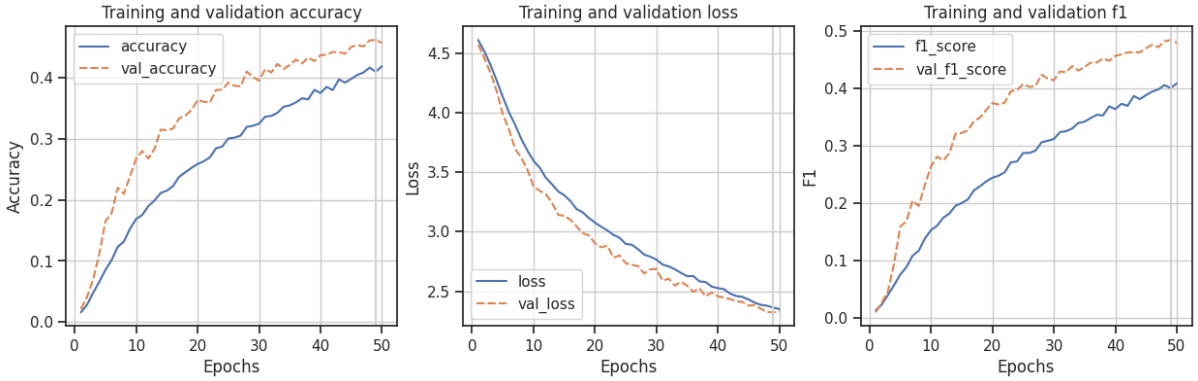


Figure 60: Training and validation results after removing last three blocks (conv5_block1)

	Loss	Accuracy	F1 score
Training	1.1618	0.7054	0.7036
Validation	1.8614	0.5549	0.5755
Test		0.5441	0.5679

Table 47: Performance summary removing last block (conv5_block3)

	Loss	Accuracy	F1 score
Training	1.2686	0.6788	0.6763
Validation	1.8161	0.5594	0.5788
Test		0.5554	0.5796

Table 48: Performance summary removing last two blocks (conv5_block2)

	Loss	Accuracy	F1 score
Training	2.3509	0.4194	0.4081
Validation	2.3305	0.4577	0.4788
Test		0.4529	0.4758

Table 49: Performance summary removing last three blocks (conv5_block1)

Remove blocks - Fine tuning

The fine-tuning of the base model hasn't achieved the expected results. For this reason, starting from the feature extraction previously performed, for each feature extraction experiment has been unfrozen the layer before the one removed.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_3 (InputLayer)         [(None, 224, 224, 3)]     0

tf.__operators__.getitem_1   (None, 224, 224, 3)       0
 (SlicingOpLambda)

tf.nn.bias_add_1 (TFOpLamb   (None, 224, 224, 3)       0
da)

base (Functional)            (None, 14, 14, 512)       7635264

global_average_pooling2d_1   (None, 512)               0
 (GlobalAveragePooling2D)

batch_normalization_1 (Bat   (None, 512)               2048
chNormalization)

dense_2 (Dense)              (None, 256)               131328

dropout_1 (Dropout)          (None, 256)               0

prediction (Dense)           (None, 100)               25700

=================================================================
Total params: 7794340 (29.73 MB)
Trainable params: 7533156 (28.74 MB)
Non-trainable params: 261184 (1020.25 KB)
_________________________________________________________________
```

Figure 61: Summary fine tuning on second experiment

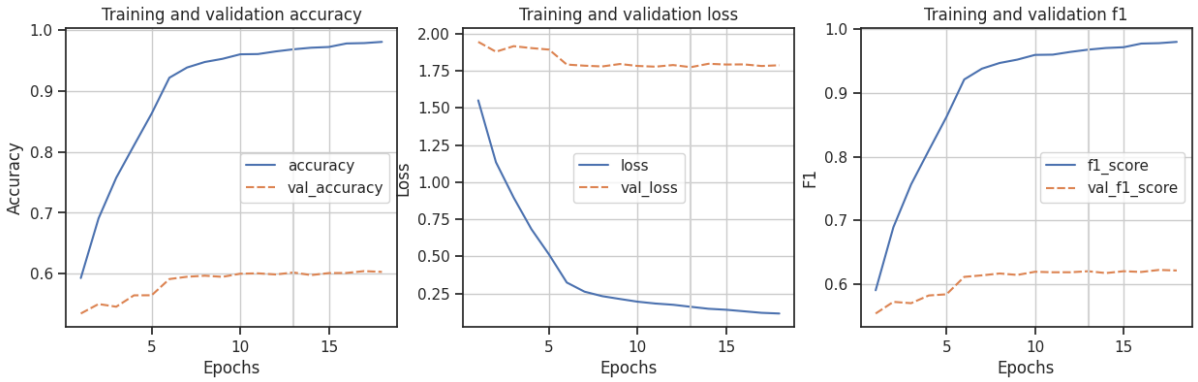


Figure 62: Training and validation results

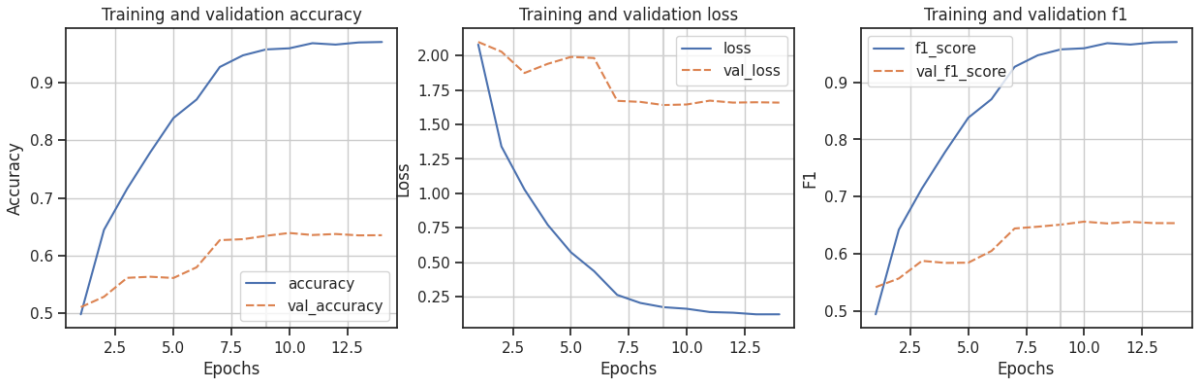


Figure 63: Training and validation results

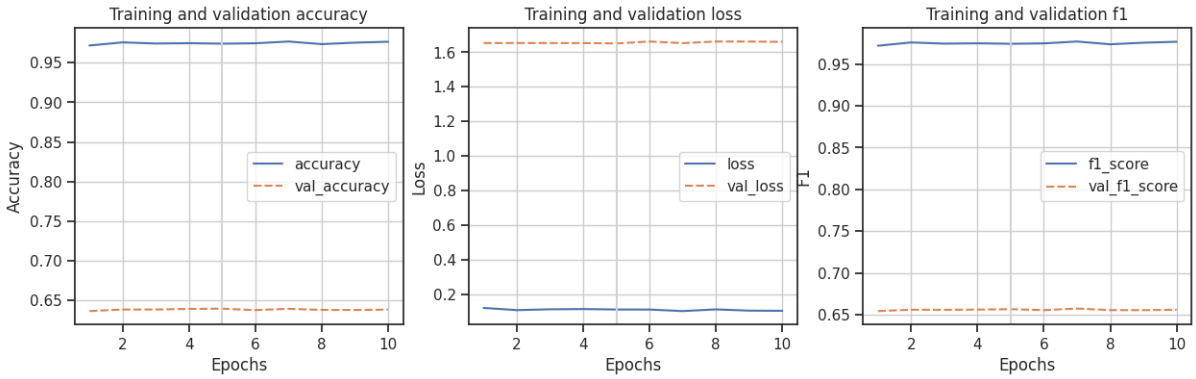


Figure 64: Training and validation results

	Loss	Accuracy	F1 score
Training	0.1147	0.9803	0.9803
Validation	1.7870	0.6032	0.6212
Test		0.5920	0.6135

Table 50: Performance summary removing last block

	Loss	Accuracy	F1 score
Training	0.1235	0.9697	0.9697
Validation	1.6607	0.6350	0.6537
Test		0.6295	0.6493

Table 51: Performance summary removing last two blocks

	Loss	Accuracy	F1 score
Training	0.1072	0.9766	0.9766
Validation	1.6602	0.6382	0.6562
Test		0.6370	0.6566

Table 52: Performance summary removing last three blocks

The best results have been obtaining with the unfrozen of one block after removing two blocks but the three model are all in huge overfitting, since the performance of the train set rapidely increases, at least in the first two experiments, while the validation after few epochs doesn't increase.

The majority classes, as thrnig1, wlwwar and eubeat1 achieve high F1-score value in the test set, all higher than 0.8. The F1-score decreases with the descrease of the images number of the classes; some mid classes achieve 0.6 of F1-score, while other as chibat1 and afpfly1 less than 0.3. This huge changes of performance between the classes are similar to the ones highlighted in VGG16. Indeed, even in with ResNet50, the minority classes, as amesun2 and sclbou1 don't reach good results.

6 Ensemble

6.1 Average voting

Since all the models have been trained individually and it is possible to retrieve their predictions on the test set, it's easy to merge their predictions is to average them at inference time. Taking four different models, the result is:

$$prediction = 0.25 * (prediction_1 + prediction_2 + prediction_3 + prediction_4)$$

Thanks to this simple operation, the resulting ensemble model outperforms the invidual models. Indeed, F1-score has achieved 0.7633.

	Precision	**Recall**	**F1 score**
Scratch_4ex	0.6500	0.5700	0.5929
ResNet50_finetuning_2blocks	0.7000	0.6300	0.6537
VGG16_2ft_lastblock	0.7600	0.7100	0.7254
VGG16_2ft_twoblocks	0.7800	0.7400	0.7524
Ensemble avg	**0.7925**	**0.7527**	**0.7633**

Table 53: Models performance comparison

6.2 Weighted average voting

Since the optimum weights can minimize the generalization error of ensemble, it's very important to find optimum weights for individual component networks. However, it's nearly impossible to find the optimum weights for the neural networks directly. Weighted esemble was performed by associating each model with a different weight based on the f1-score evaluated on the validation set. The same models used for average voting were used here.

	Precision	Recall	F1 score
Scratch_4ex	0.6500	0.5700	0.5929
ResNet50_finetuning_2blocks	0.7000	0.6300	0.6537
VGG16_2ft_lastblock	0.7600	0.7100	0.7254
VGG16_2ft_twoblocks	0.7800	0.7400	0.7524
Ensemble avg	0.7925	0.7527	0.7633
Ensemble weighted	**0.7927**	**0.7531**	**0.7637**

Table 54: Models performance comparison

The results obtained through this ensemble are slightly better than those obtained in the previous ensemble as expected.

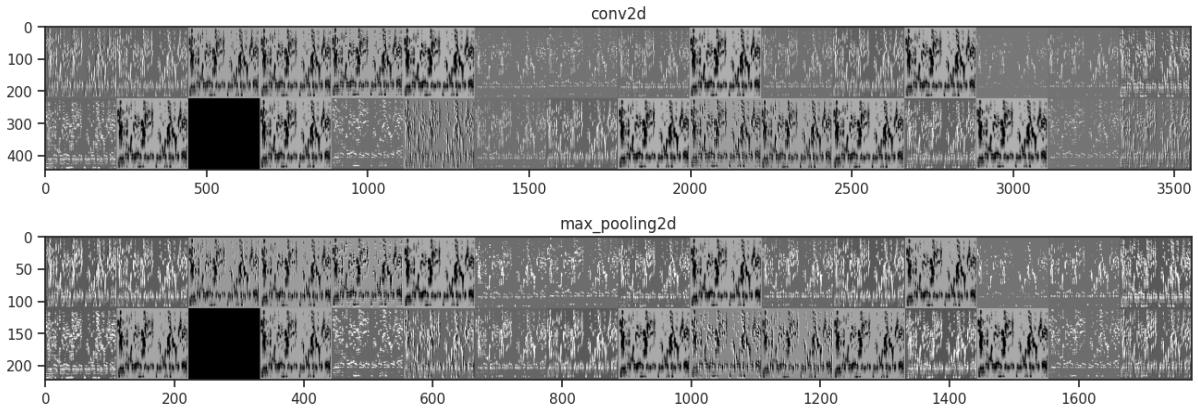
7 Explainability

The topic of explainability is of great relevance in the context of artificial intelligence. CNNs are powerful deep learning models widely used in computer vision applications, but their complex architecture and high-dimensionality representations can make it difficult to interpret why and how the model makes certain decisions. In this section, the aim is to examine how the networks classify the images using two distinct approaches: Intermediate Activations and Class Activations Heatmaps.

7.1 Intermediate Activations

The visualisation of intermediate activations involves displaying the feature maps produced by the convolution and pooling layers within the network, based on a given input. It is important to note that in order to compare the different networks, the experiments were performed with identical images.

7.1.1 CNN from scratch



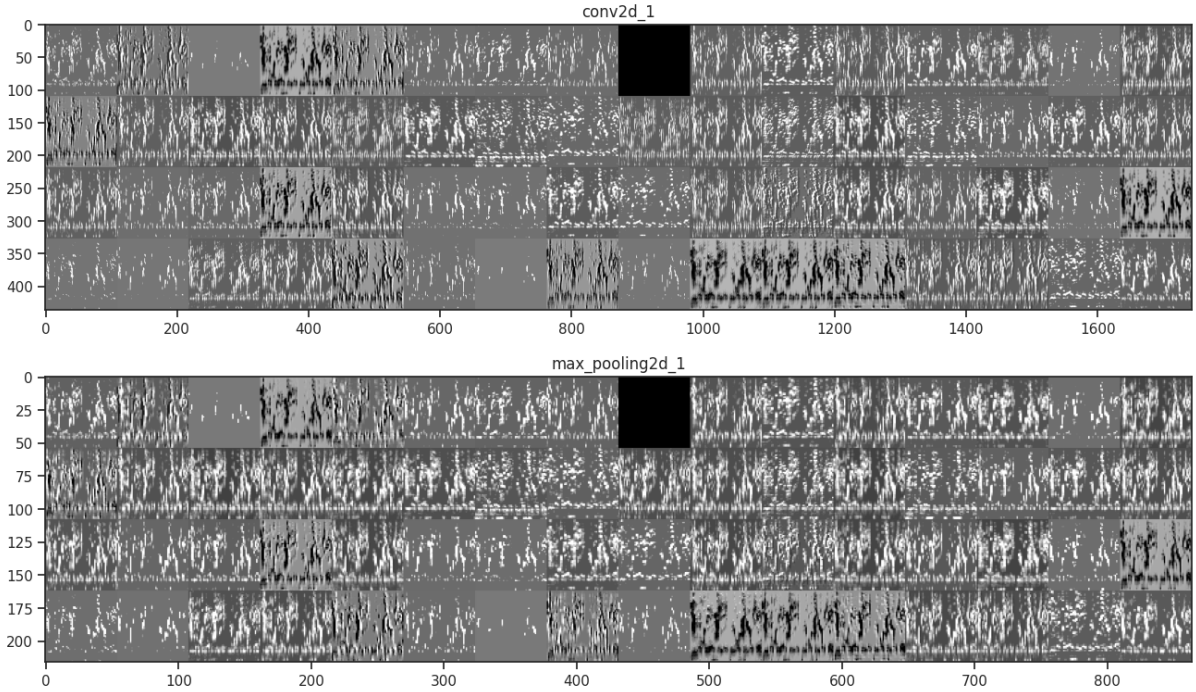


Figure 65: First intermediate activations of the CNN from scratch

The feature maps show that the model focuses on the signal wave first and then, in the pooling layer, on the borders of the edges dividing the wave from the background. In the next conv2d, the signal is sparser because only the most robust signals are mantained in the images, while the pooling layer tries again to look for some useful edges. The fact that some images are completely black is due to the ReLu trigger function, which sets the output to 0 for negative values. The same considerations can be done for the following models.

7.1.2 VGG16

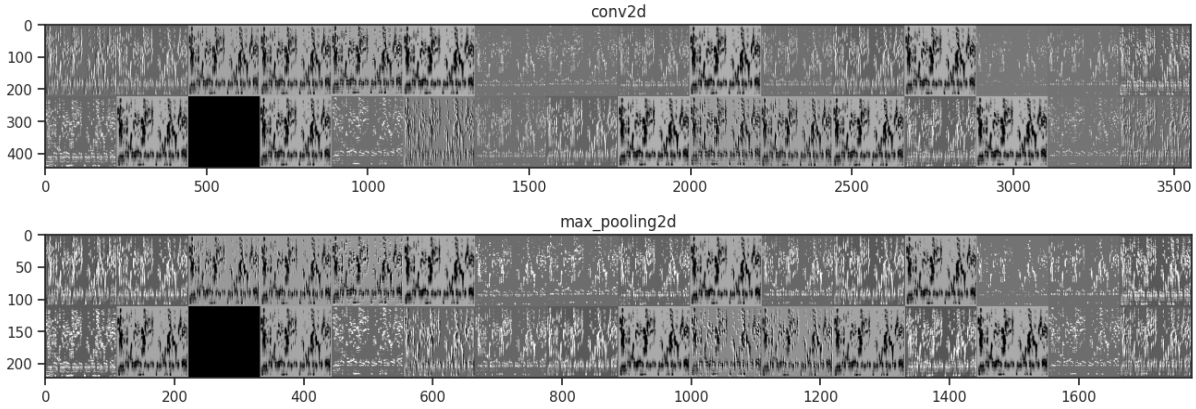


Figure 66: First intermediate activations of VGG16

7.1.3 MobileNetV2

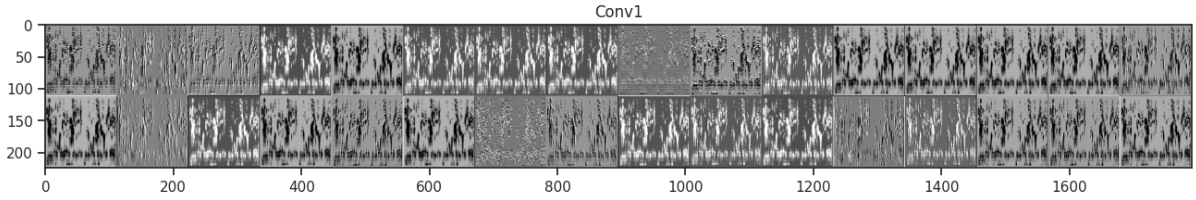


Figure 67: First intermediate activations of MobileNetV2

7.1.4 ResNet50

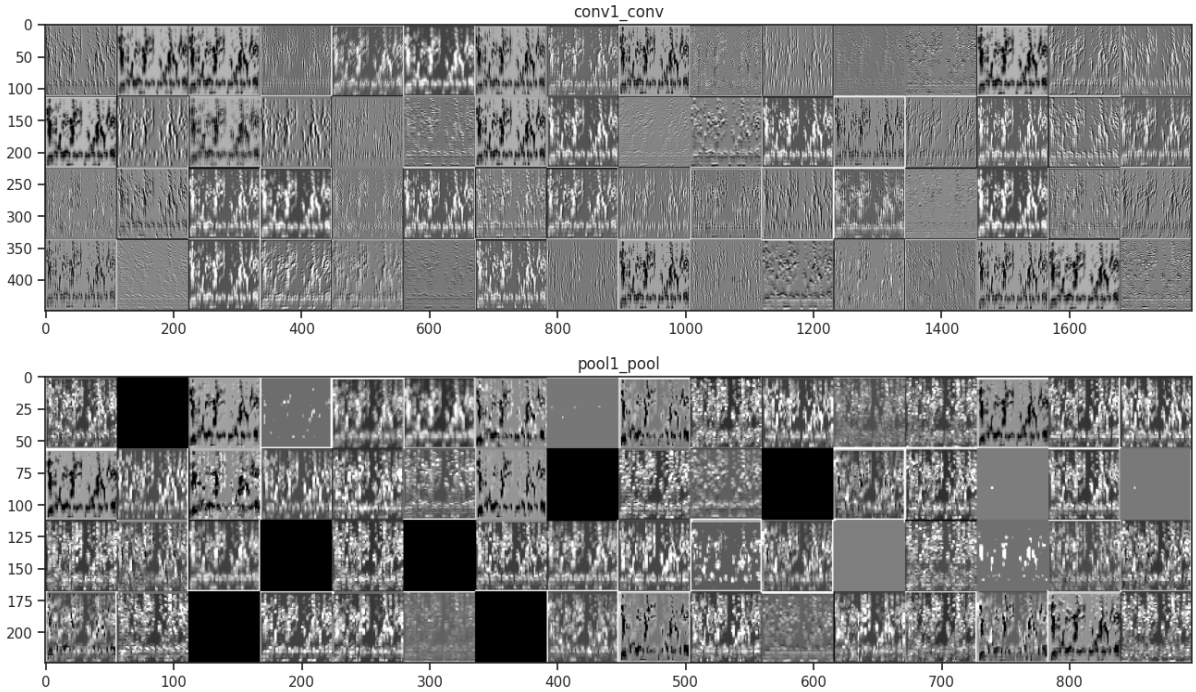


Figure 68: First intermediate activations of ResNet50

7.2 Heatmaps of Class Activation

This section focuses on understanding which parts of the images are responsible for the final classification. The behaviour of different networks is analysed when given an image of different classes as input. To visualise the heatmaps of the networks, a technique called Grad-CAM (Gradient Class Activation Map) is used. The concept is to calculate the gradient of a given class with respect to the final convolutional layer and then weigh it against the output of that layer. The result is heatmaps that indicate the regions of the input image that are more influential in determining the predicted label.

7.2.1 thrnig1

This is the majority class in terms of quantity of images in the validation and test sets.

Figure 69: Thrush Nightingale

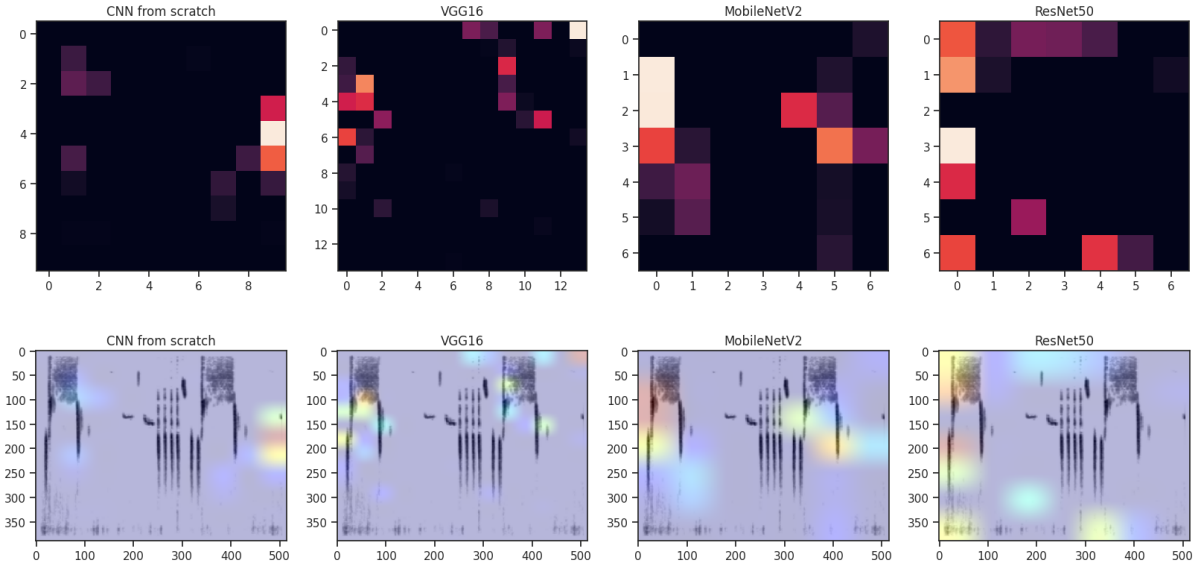


Figure 70: Thrush Nightingale heatmaps and grad maps

This one of the best recognised classes. All the models except from the scratch one focus on the left part of the image where the signal is more defined. The from-scratch one focuses on the right part in which there is not useful information and MobileNetV2 also recognises the repetition of the signal in the right part.

7.2.2 barswa

This class is the second in terms of data quantity and, as before, all the models find some area of interest in the test image given.

Figure 71: Barn Swallow

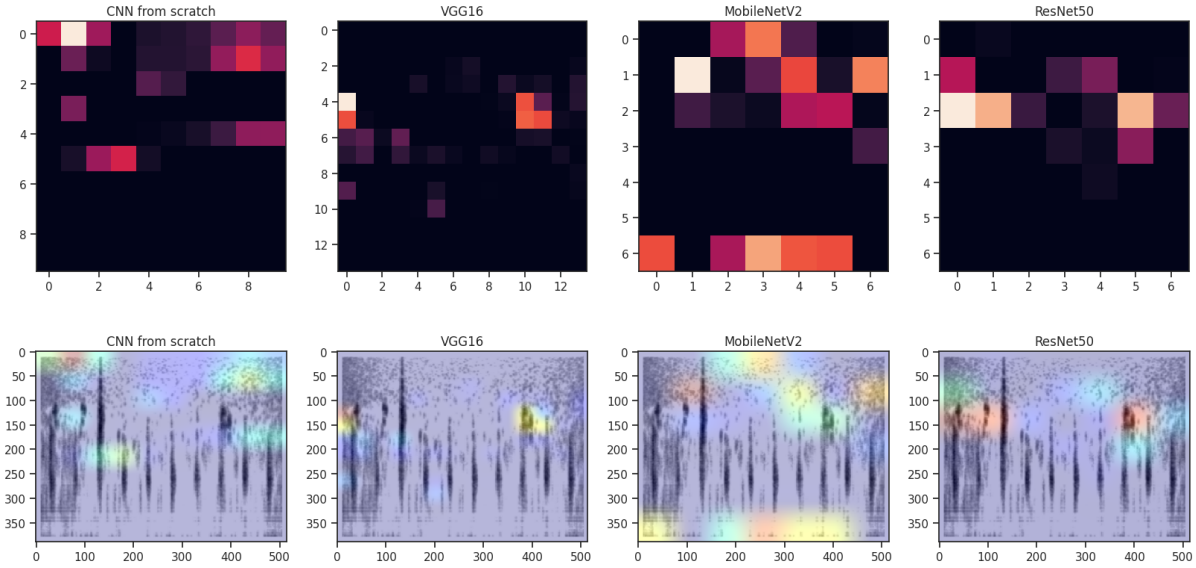


Figure 72: Barn Swallow heatmaps and grad maps

Here all the models performs pretty well and the grad. In particular, MobileNetV2 also focuses on the bottom part of the spectrogram and on the upper central zone.

7.2.3 amesun2

This is one of the minority classes that have been augmented at the beginning and that is not well classified in almost all the experiments.

Figure 73: Amethyst Sunbird

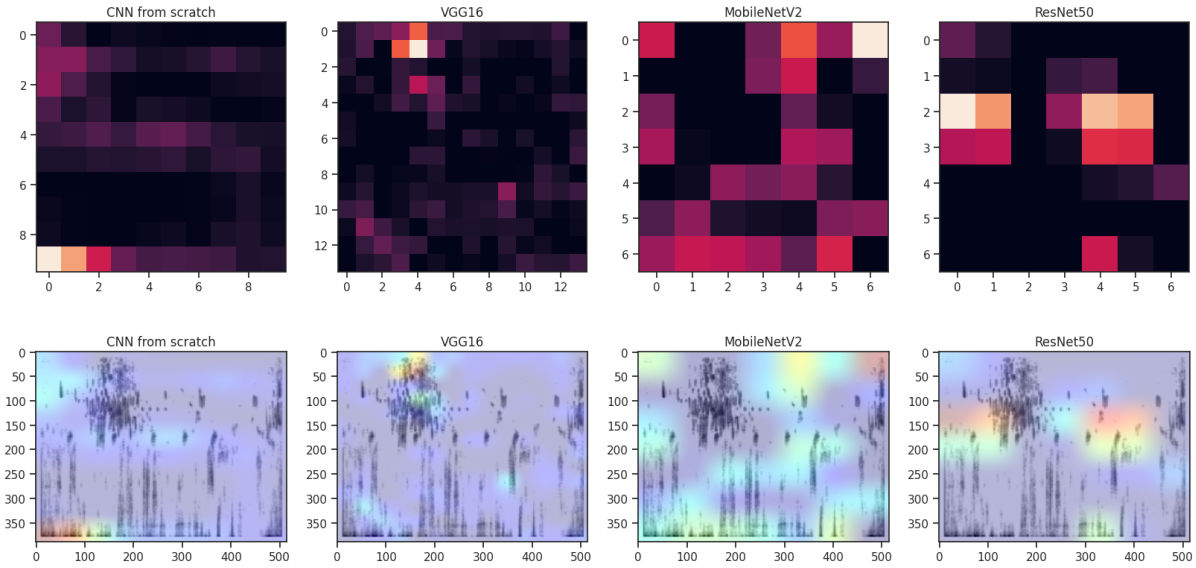


Figure 74: Barn Swallow heatmaps and grad maps

All the images are quite noisy and the main signal is not so evident like the spectrograms of other species. In fact, all the models focuses on different areas of the spectrogram. Only VGG16 highligths more than the others the dark zone in the upper left.

7.2.4 grbcam1

grbcam1's spectrograms are classified quite well in the average of experiments.

Figure 75: Green-backed Camaroptera

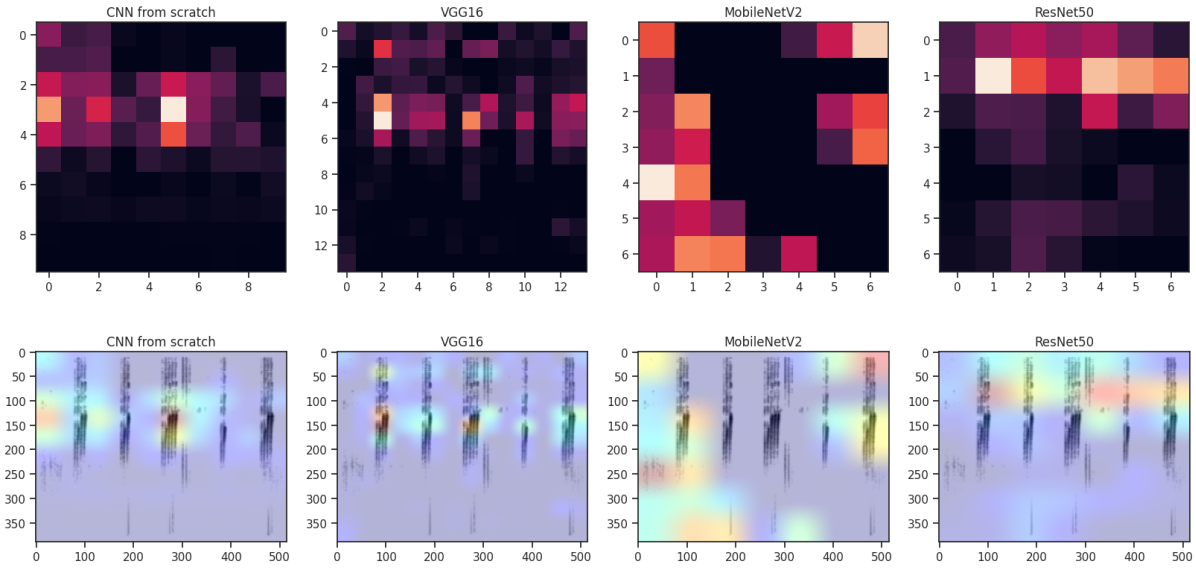


Figure 76: Green-backed Camaroptera heatmaps and grad maps

The grad maps clearly show that all networks are successful in recognizing the repeating signals at the top of the image. The main reason is that the images of this class are characterized with a high signal-noise ratio that, due to the noise reduction of the preprocessing, leads to an image characterized by a clean signal.

8 Conclusion

The results obtained by us are satisfactory because in line with those in the literature. As expected, the best results were obtained by removing blocks from the pre-trained networks since the dataset on which they were trained is very different from the one used in the project described here. Certainly one of the biggest problems is the strong bias of the data that characterizes the dataset and the strong similarity between the sounds, and therefore the images, representative of each species. Despite all the issues we ran into—most notably the fact that we lacked strong computational capabilities and could only use a portion of the dataset—as well as the limitations on Google Colab Free accounts that frequently caused us to become disconnected—we experimented with various configurations and made use of various pre-trained neural networks. We are confident that future network optimization utilizing hyperparameter optimization will help our analysis. In addition, better results could be obtained by using more epochs. In fact, at least 200 epochs have been used in the literature but, due to limited resources, only 50 epochs have been used while still obtaining satisfactory results.

References

[1] BirdCLEF 2023.

[2] xeno-canto.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Ágnes Incze, Henrietta-Bernadett Jancsó, Zoltán Szilágyi, Attila Farkas, and Csaba Sulyok. Bird sound recognition using a convolutional neural network. pages 295–300, 2018.

[5] Stefan Kahl, Thomas Wilhelm-Stein, Hussein Hussein, Holger Klinck, Danny Kowerko, Marc Ritter, and Maximilian Eibl. 2017.

[6] Chih-Yuan Koh, Jaw-Yuan Chang, Chiang-Lin Tai, Da-Yo Huang, Han-Hsing Hsieh, and Yi-Wen Liu. 2019.

[7] Patrik Lauha, Panu Somervuo, Petteri Lehikoinen, Lisa Geres, Tobias Richter, Sebastian Seibold, and Otso Ovaskainen. 2022.

[8] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. 2019.

[9] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[10] Roman Shrestha, Cornelius Glackin, Julie Wall, and Nigel Cannings. Bird audio diarization with faster r-cnn. pages 415–426, 2021.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[12] Eleni Tsalera, Andreas Papadakis, and Maria Samarakou. Comparison of pre-trained cnns for audio classification using transfer learning. *Journal of Sensor and Actuator Networks*, 10(4), 2021.