1 Dataset description

As it is going to be seen in the section 2, classification is performed thanks to a neural network. Such classifiers are known to have better performance with large dataset (it is actually the case for a lot of classifiers). In this way, the need of increasing the dataset's size is a major point since the original dataset is poor in term of quantity of data. Then, the data received and computed on the Virtual Machine is under the form of melspectrograms which are similar to images. One idea would be to compute the melspectrogram of the original ESC-50 dataset and performing data augmentation. However, this method suffers from the fact that there are a lot of differences between the data computed thanks to the ESC-50 dataset and the data that is received from the whole chain in the real conditions. Indeed, in real conditions, the sounds are displayed thanks to an audio device and are recorded through the microphone of the board. These two parameters are the bottleneck of the classification since they are varying a lot with the orientation of the board, the place where the sounds are displayed, the intensity used or even the frequency response of the audio device. Those variations make the melspectrograms changing a lot, and the pattern that a neural network could identify into a certain dataset, can not be recovered during another acquisition. That's why the need of a large and with a lot of variety, dataset is needed.[1] To take into account the melspectrograms' variation and the need of increasing the dataset's size, a good solution is to record the sound of the ESC-50 dataset with the microphone of the board. However, the dataset is still very dependent of the place of where it is recorded, the intensity received at the microphone and the orientation. It is not resolvable, the most qualitative dataset that can be recorded is the one that would be done in the contest's room.

2 Convolutional Neural Network

Convolutional Neural Network (CNN) are often used for image classification. Basically, the purpose of CNN is to extract local information by applying convolution of filter with the image (also called kernels). Then, CNNs assign importance to aspects/shapes/objects in the image and are able to differentiate one from the other. Images, depending on the resolution, are high-dimensional inputs, such that, this is quite hard (impractical) to use the fully-connected neural network by linking all the neurons of a specific layers to all the neurons of the following one. In this way, convolutions solve this by using filters that will connect neurons to only a local region of the image thanks to the filter size.

[1]See Annex for explanation on what would be the solution that we think is the best but quite impractical.

2.1 Basic architecture

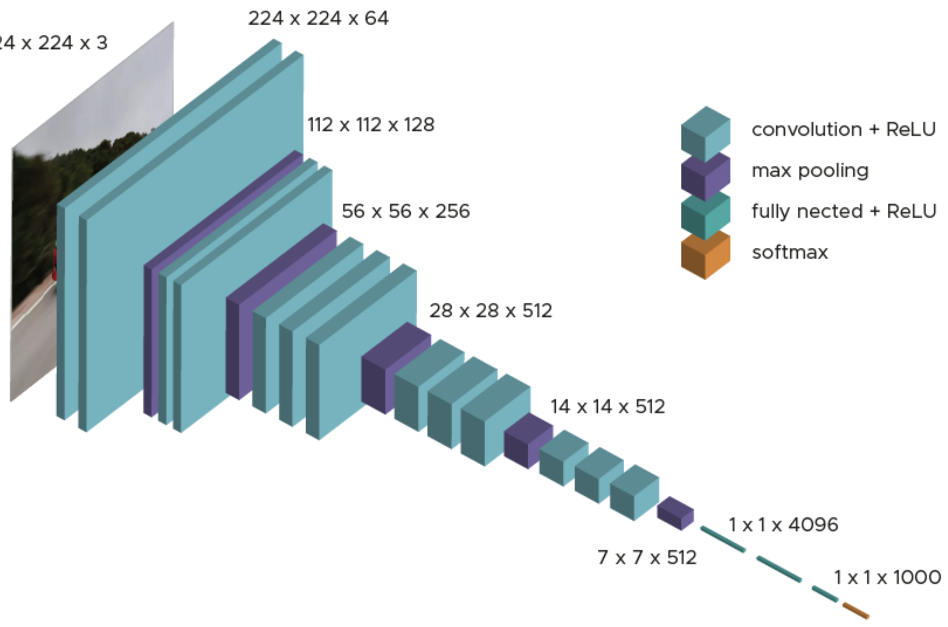


Figure 1: VGG16 Architecture

VGG16 Architecture proved its high efficiency with RGB image classification for many years. This architecture is composed of different layers based on convolutions. The basis is to identify patterns into the image by applying convolutions in 2D. [2]

VGG16 architecture is composed of many convolution layers followed by ReLU activation which is a popular use in CNNs since it has the property to be equal to zero for negative input. It involves that even if a pattern is not recognised into a new input, its contribution is not taken into account (it would create a negative output if a sigmoid was used, for example). Since every pattern is assumed indepedent from the others, there is no reason that a specific pattern influence the decision if it is not present. After the convolution has been done, max pooling is usually performed to reduce the input size. For the dataset used, (see section 1) the input size is 20x10, which is a really poor resolution. In this way, performing max pooling is not recommended. Then, the VGG16 architecture has to be modified to match with the dataset specifications. Our implementation follows the pattern :

[2]See Annex for explanation on convolutions in 2-dimensions.

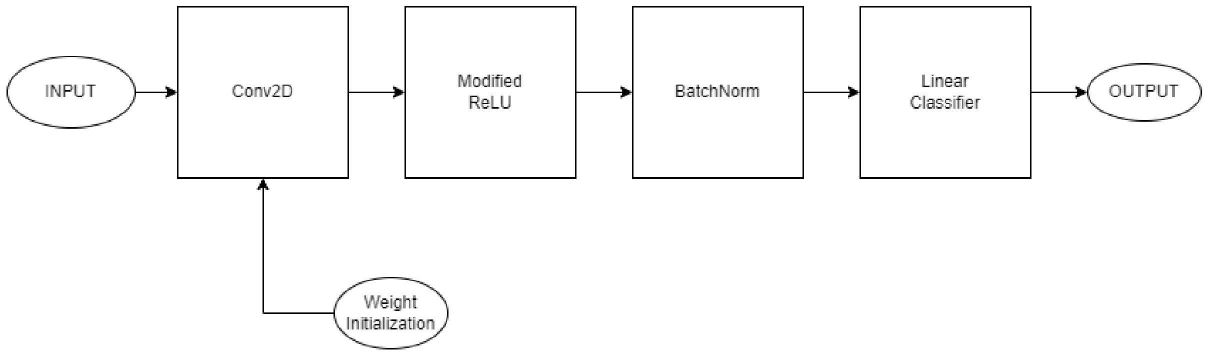


Figure 2: CNN Architecture

The architecture has been modified to match with the dataset. The number of dimensions of the feature vectors are so low that it is not recommended to perform max pooling. Instead, convolution with ReLU activation is applied, followed by a normalization of the batch. The three first blocks are called "Convolution Block". This block is detailed in the section 2.2. Our final architecture is the following one : four convolution blocks are placed after each others and a final block is placed after those, called "linear classifier". This is a neural network that flatten the output of the CNN through an average pooling and a linear transformation, to obtain, at the output, five probabilities representing each classes to classify.

2.2 Convolution Block

2.2.1 Weight Initialization

The output of the convolution layer (torch.nn.conv2D) can be described as :

$$out(N_i, C_{out_j}) = biais(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) * input(N_i, k)$$

where * is the 2D cross-correlation operator, N is the batch size, C denotes a number of channel, H is the height of the input and W the width of the input.

Then, biais and weights can both be initialized to render better performance of the classification. The paper "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification" [3] describes a weight initialization nammed "Kaiming initialization". This initialization turns the biais tensor to zero. Weights are initialized as a tensor with values sampled from a normal distribution. Kaiming initialization shows better stability results than others initialization since it provides a standard deviation close to 1 (and a mean increment quite low) during the feedforward phase and then, during the backpropagation phase, the problem of gradient exploding

[3]Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification is from "Kaiming He", "Xiangyu Zhang", "Shaoqing Ren" and "Jian Sun", that describes how they proposed a different definition of the well-know ReLU function and an innovative way of doing weight's activation. They have the first result to perform the human-level performance on the visual recognition challenge ILSVRC 2014.

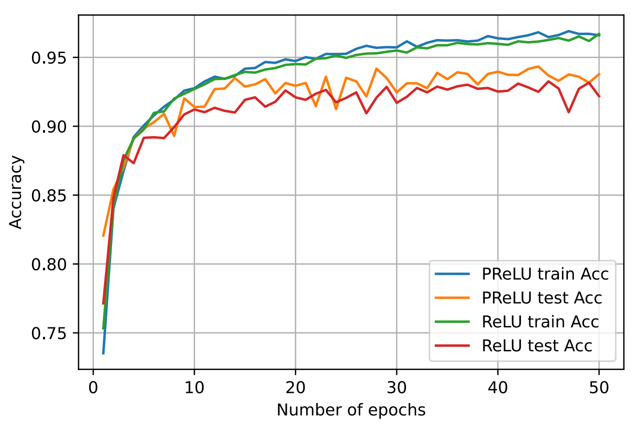
or vanishing is avoided. Even for deep neural networks, the gradient descent is still applicable thanks to the stability of this initialization.

2.2.2 PReLU activation

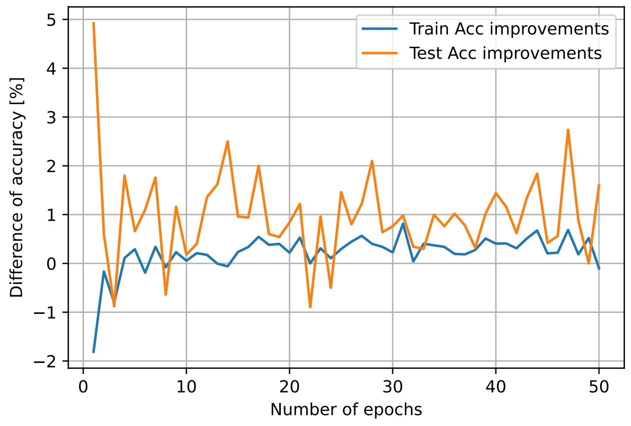
Empirically, a lot of papers have shown that training a deep network with ReLU activation tended to converge much more quickly and reliably than sigmoid function. ReLU is easy to compute and its gradient is equal to one for positive values. However, for negative values, the gradient is zero and then the gradient descent is no more used. Then, the paper "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification" describes a new activation function that they named "PReLU" (also known as "leaky ReLU").

$$PReLU(x) = \begin{cases} x & \text{if } x \geq 0, \\ ax & \text{otherwise} \end{cases}$$

This function gives a non-zero gradient, even for negative inputs. The paper of "Kaiming He" demonstrated that PReLU activation allows to go into deeper networks (neural networks with many more layers). According the dataset used for the project (see section 1), the improvements in term of classification's accuracy between ReLU and PReLU activation is given in the figure **??** :



(a) ReLU vs PReLU train/test accuracy with regards to the number of epochs



(b) PReLU improvements with regards to the classification's accuracy

Figure 3b shows the improvements in term of train/test accuracy with regards to the number of epochs. This improvement is derived as the difference of accuracy between the whole network trained with ReLU and PReLU activation. Actually, this process needs to be repeated with shuffling on train set, test set to avoid as possible, the randomness. It turns out that the improvement is about of 1.1%. For such high accuracy, this is not negligible.

2.2.3 Batch Normalization

3 Annex

3.1 Dataset Solution

As seen in the section 1, the dataset is a key point to achieve a good classification. In this way, the undesired effects of the whole chain (way of displaying sounds, microphone, ...) have to be compensated. The best solution found would be to compute the frequency response of the whole chain. In a first step, single tone sounds would be displayed in the place where the response has to be estimated. These sounds would be acquired by the microphone. The basic purpose of a frequency response is the ratio between the input fourier transform and the output one in term of amplitude. In this way, displaying sounds at many "important" single tone frequency (important in the way, corresponding to the frequencies used in the project) and computing the fourier transform at the input and the output of the whole chain, could, hypothetically, give the frequency response of a specific place and the whole chain. In practice, this is quite hard to realise. First, this is not realistic since the embedded system do not display sounds (and we have no idea of the requirements to do so). Then, this process may need to be done so often, for maybe poor results, that this is more a constrain than a help.

3.2 2-dimensions convolutions

Here, 2D-convolution is not the well-known convolution with the property of commutativity. Indeed, in the case of CNNs, the 2D-convolution is what we could call a cross-correlation, where :

$$(I * K)(i,j) \neq (K * I)(i,j)$$

Then, the cross-correlation is defined in this way :

$$(K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n)$$

which suffers from the lack of the commutativity property but these is not important in the machine learning context. This is essentially the generalization of the 1D-convolution for the 2D space.

Formula linking input and output volume of a convolution in 2 dimensions :

$$(W - F + 2P)/S + 1$$

where W is the input volume size, F the receptive field size of the convolution layer (essentially corresponds to the filter size), P the amount of zero-padding used and S the stride.

For example, the features vectors are melspectrograms which are given to the CNN as input. The shape of each melspectrogram is (1, 20, 10) corresponding to (channel, height, width). Indeed, melspectrograms are only in a unique channel since these are not RGB images. For reminder, the melspectrograms are composed of 10 components, which are themselves composed of 20 components.

Input has size of 20x10, if a filter of size 5x5 is applied with an amount of zero-padding of 1 and a stride of 2, then the output is size of 9x4.

References

[1] https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

[2] Wikipedia : Convolutional Neural Network

[3] Convolution formula linking input and output dimensions

[4] https://arxiv.org/pdf/1502.01852.pdf

[5] https://towardsdatascience.com/weight-initialization-in-neural-networks-a-journey-from-

[6] ReLU activation advantages : spread use

[7] https://pytorch.org/docs/stable/generated/torch.nn.PReLU.html

[8] https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

[9] https://pytorch.org/docs/stable/nn.init.html

[10] https://www.baeldung.com/cs/ml-relu-dropout-layers

[11] https://pytorch.org/docs/stable/generated/torch.nn.AdaptiveAvgPool2d.html

[12] https://pytorch.org/docs/stable/generated/torch.nn.Linear.html

[13] https://arxiv.org/abs/1207.0580

[14] https://pytorch.org/docs/stable/generated/torch.nn.Dropout.html

[15] https://arxiv.org/abs/1411.4280