SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER, CONTROL AND MANAGEMENT ENGINEERING

COURSE IN ARTIFICIAL INTELLIGENCE AND ROBOTICS

Homework 2 Report

MACHINE LEARNING

Professor:
Luca Iocchi

Student:
Alessio Borgi (1952442)

Academic Year 2023/2024

Contents

1 INTRODUCTION

The purpose of this report is to present the findings of an extensive analysis conducted as part of **Homework 2: Image Classification** assigned by Professor Luca Iocchi in the Master's Degree in Artificial Intelligence and Robotics. This homework revolves around a dataset containing images of the **CarRacing(v2)** racing car performing different actions, with five classes representing distinct maneuvers.

The investigation follows a structured pipeline, employing a **Convolutional Neural Network (CNN)** to solve the Image Classification problem. The tools selected for this endeavor include popular deep learning frameworks, in particular **PyTorch Lightning**, and the experiments were executed using **Kaggle**, which provided the necessary infrastructure. This report is presented together with the code. The CNN solution is presented in different variants, where different architectures, settings, and hyper-parameters are tried.

Despite considering various metrics such as Accuracy, F1 Score, Precision, and Recall in the evaluation, the ultimate decision has been made concerning the **Reward** obtained in the **CarRacingv2** Simulator.

Enjoy this exciting journey into image classification for racing car actions! :)

For any questions, please feel free to contact me at the institutional email provided below:
borgi.1952442@studenti.uniroma1.it

2 DATASET INSPECTION

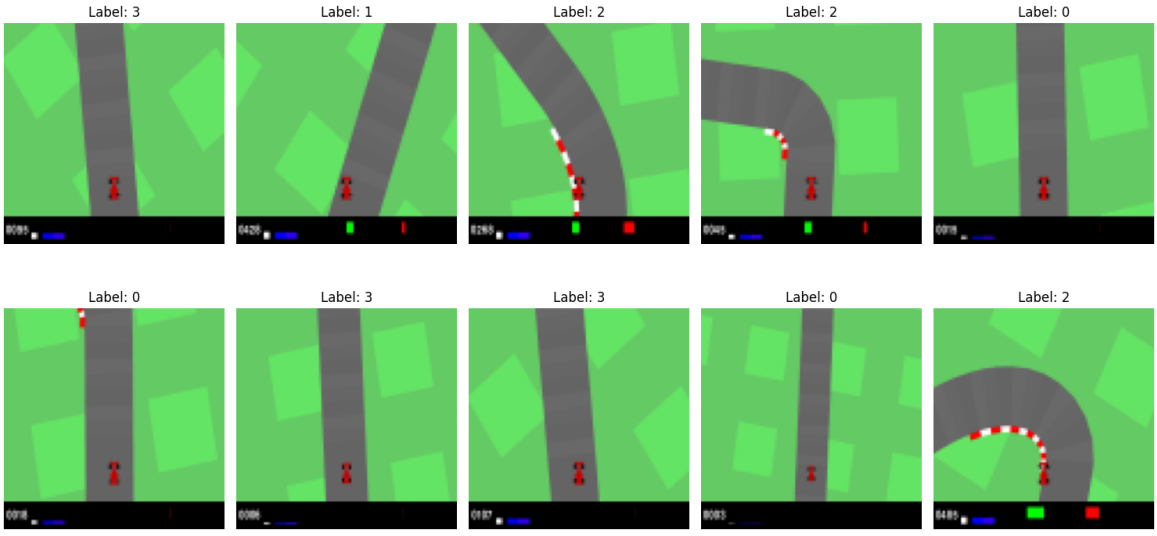
In this section, I perform a comprehensive inspection of the dataset, exploring its key characteristics, distributions, and potential challenges. Through this analysis, I aim to gain valuable insights into the dataset's composition and identify any issues that may impact the training and evaluation of my model.

2.1 Data Distribution Analysis

The initial phase of the experimentation process is dedicated to **Data Distribution Discovery**. In this study, the focus is on understanding the distribution of the image dataset. Unlike a typical pre-processing phase, where various transformations are applied, the emphasis here is solely on gaining insights into the inherent characteristics of the data, allowing for insights into the imbalance, diversity, and key features present in the images. This preliminary analysis lays the foundation for subsequent steps in the experimentation process.

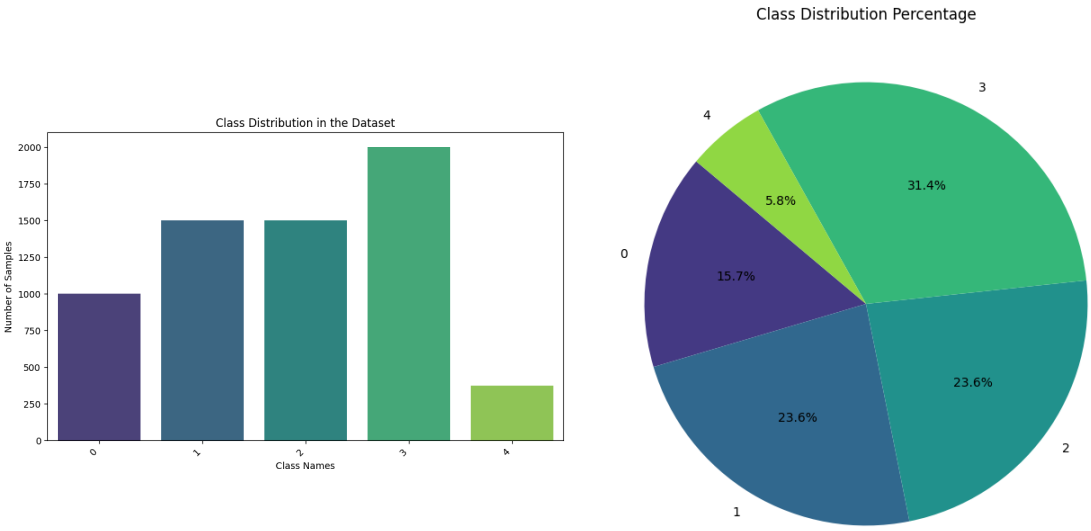
Before doing any pre-processing, is it usually a good habit to look at the Dataset.

Figure 1. Dataset Inspection: Let's have, first and foremost, an idea of the dataset we are going to deal with. We observe the samples and their labels.



Here I report the images of the Data Distribution, that will be used in the next steps to tackle the Data Imbalance present.

Figure 2. Data Distribution: The data distribution analysis revealed an imbalanced dataset, with varying class frequencies. The corresponding class weights, used to address the imbalance during training, are as follows: $class\_weights = [0.157, 0.236, 0.236, 0.314, 0.058]$. This consideration ensures that the model accounts for the differing importance of each class during the training process.



After extracting the distribution of the dataset, it is observed that the **Dataset** is **Imbalanced** concerning the 5 classes. The class distribution is as follows:

$$\text{class weights} = [0.157, 0.236, 0.236, 0.314, 0.058]$$

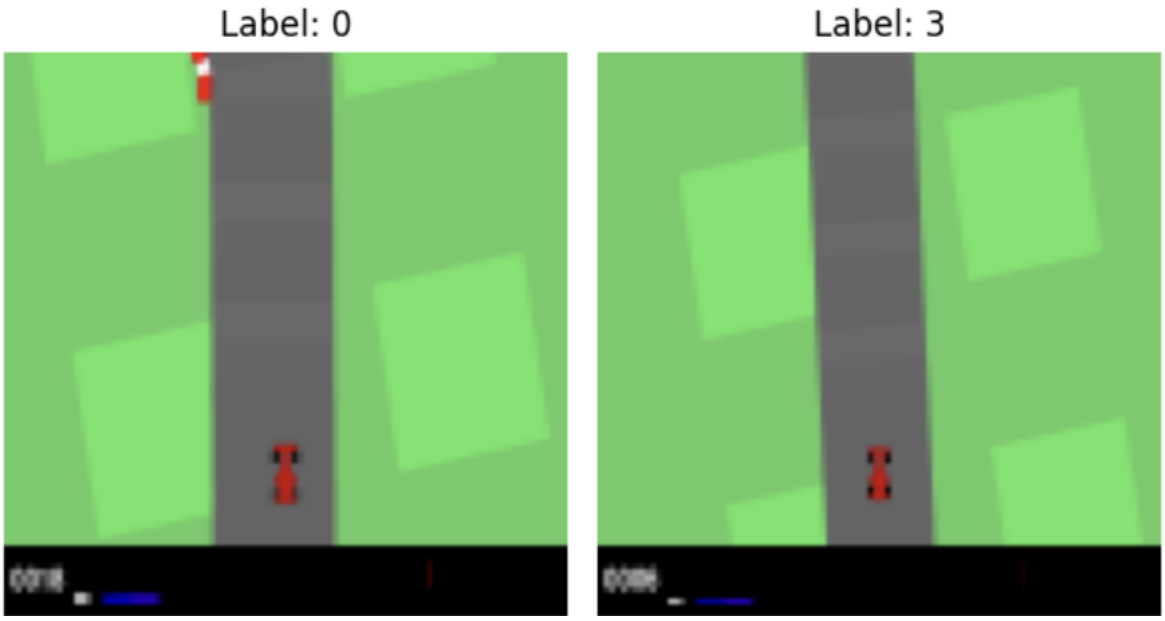
Upon recognizing this imbalance, a series of solutions has been taken. Specifically, a **Weighted Cross Entropy** loss is employed during training. This weighted loss assigns different weights to each class, allowing the model to better handle the imbalanced nature of the dataset.

Additionally, the model utilizes various evaluation metrics such as accuracy, F1 score, precision, and recall to provide a comprehensive assessment of its performance.

2.2 Noisy Labels

During the Dataset Inspection, a new challenge has come to light. Upon closer examination, it becomes evident that certain sample **Labels** exhibit **Noise**, as illustrated in the accompanying figure 3. The presence of noisy and inconsistent labels introduces complexity to our dataset, potentially impacting the model's training and performance.

Figure 3. Noisy Labels: As it can be observed from this two figure below, we have that the car in the Simulator environment presents almost the same class (go straight), but the Labels in the Training Set does not coincide. This is a red flag for me, letting me think that the Dataset could contain lots of other Noisy Labels.

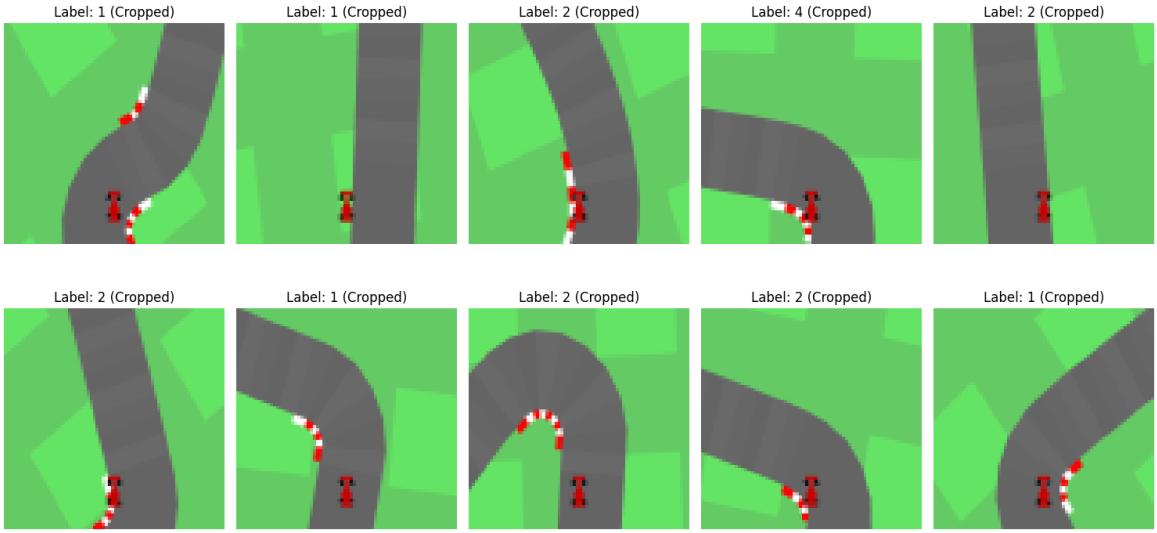


Addressing this issue is crucial for ensuring the Robustness and Reliability of DL models. To handle and mitigate the impact of these noisy labels, the solution I propose is to use **Label Smoothing**, a regularization technique that addresses the problem of noisy labels and enhances the generalization capability of a model by introducing a controlled amount of uncertainty during the training process. It addresses this issue by redistributing a fraction of the confidence from the correct class to other classes, introducing a more nuanced and robust learning signal.

2.3 Images Cropping

In the case of the CarRacingv2 Simulator, a specific preprocessing step was introduced to address the challenges posed by the environment. This involved performing a **Cropping** of 12 Pixels from the bottom of each image. The purpose of this cropping was to eliminate a distracting bottom bar present in the images, which was observed to harm the model's performance. Following this cropping step, the **ToTensor** transformation was applied, converting the cropped images into the PyTorch tensor format.

Figure 4. Images Cropping: As it can be observed from the figures below, we have that the images have to be cropped from the bottom bar present in the observation images.



2.4 Train-Validation-Test Split

As a basic step, in the pre-processing step in the pipeline, it can be found the **Training-Validation-Test Split**. In my case, the Dataset is already divided into two chunks: Training and Test. This means that it is needed only one split, in the original Training Set, dividing it into two chunks, which would become the Modified Training Set and Validation Set, which will be used for Hyper-Parameters Fine-Tuning. Here, I have opted for, to ensure representative samples for both training and evaluation, the **Random Split** method.

In alignment with best practices, I've chosen an **80-20 Split Ratio**.

3 DL MODEL BUILDING

In this section, I detail the process of building the **Best-Deep Learning Model** tailored for the specific task at hand. The model architecture, named `HW2_Model`, is constructed by inheriting from a base **Convolutional Neural Network (CNN)** model, `CNN_Model`, and extending it as a PyTorch Lightning module. The architecture is designed to handle input data with three channels and produce predictions across a specified number of classes, i.e., 5 in my case.

3.1 Weights Initialization

The `_initialize_weights` method in the model is responsible for initializing the weights of the convolutional and linear layers using the **Xavier-Uniform Initialization**. This initialization technique helps in stabilizing the training process by ensuring that the weights are set to suitable initial values.

The Xavier-Uniform Initialization is applied to each module in the neural network using the following procedure:

- For each module `m` in the neural network:
- Check if `m` is an instance of either `nn.Conv2d` or `nn.Linear`.
- If it is, apply Xavier-Uniform Initialization to the weight parameters (`m.weight`).
- Additionally, if the module has bias parameters (`m.bias`), initialize them to zeros using `nn.init.zeros_`.

This weight initialization strategy is crucial for **preventing** issues like **vanishing** or **exploding gradients** during the training of deep neural networks. It contributes to the overall stability and effectiveness of the model during the learning process.

3.2 CNN Model Architecture

My Convolutional Neural Network (CNN) model, denoted as `CNN_Model`, is tailored for image multi-class classification tasks. The architecture is carefully crafted to capture hierarchical features from input images and make accurate predictions across multiple classes.

The key components of the `CNN_Model` architecture are as follows:

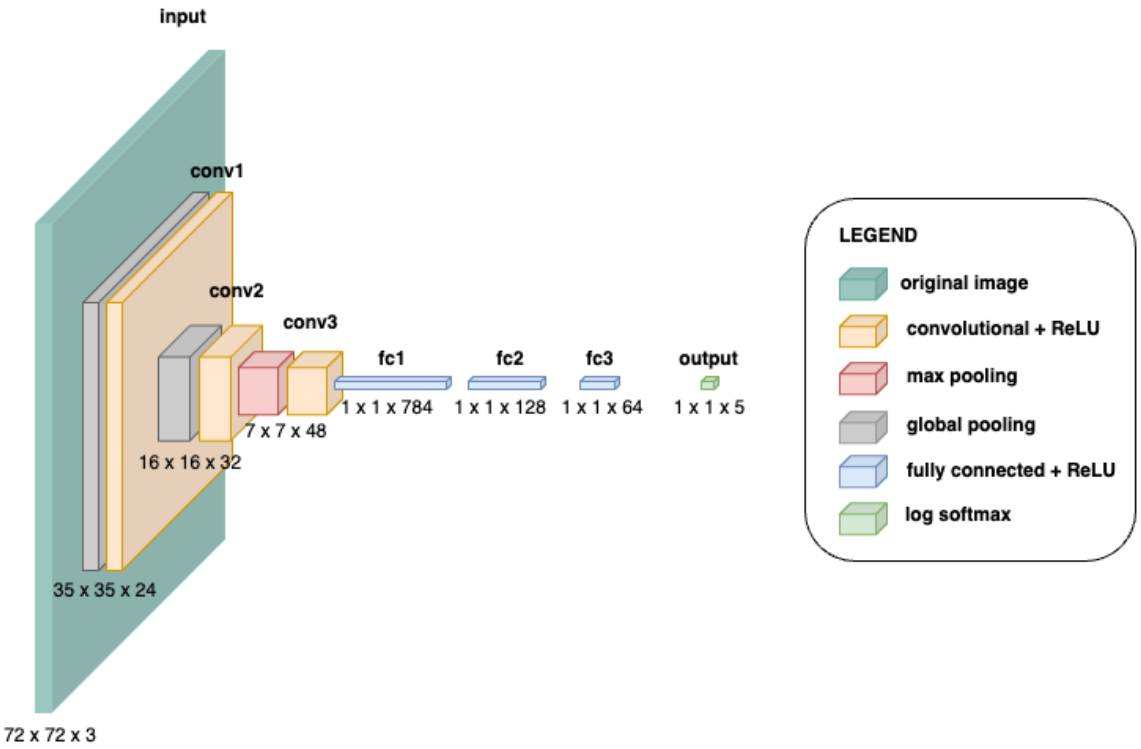
- **Convolutional Layers:** The model incorporates Convolutional Layers, leveraging **Rectified Linear Unit (ReLU)** activations to introduce non-linearity. Some of these layers are followed by **Average Pooling**, while the other two employ **Max Pooling**, contributing to feature extraction and dimensionality reduction.

- **Fully Connected Layers:** Following the convolutional layers, the network includes fully connected layers.
- **Log Softmax Activation:** The final layer employs a Log Softmax activation function, ensuring that the model outputs a probability distribution over the classes. This choice is motivated by several considerations that align with the characteristics of the dataset and the challenges posed by imbalanced classes and noisy labels. The Log Softmax function provides **improved numerical stability during training** and **inference** when dealing with **imbalanced classes** or **noisy labels**, promoting stability, robustness, and meaningful probabilistic outputs.
- **Weight Initialization:** The model's weights are initialized using the Xavier-Uniform Initialization strategy, as explained in the previous step.

The training process involves configuring the `Adam` Optimizer. The model undergoes training, validation, and testing phases, with monitoring and logging facilitated by PyTorch Lightning callbacks. These callbacks include a `Learning Rate Monitor`, `Early Stopping`, a `Progress Bar`, and `TensorBoard` logging.

For better understanding, here it is depicted the CNN Best Model Architecture:

Figure 5. CNN_Model: The model is defined as follows: The input to the network is a tensor with dimensions `[input_channels, height, width]`, where `input_channels` represents the number of input channels, and `height` and `width` represent the spatial dimensions of the input. The model consists of four convolutional layers, each followed by a rectified linear unit (ReLU) activation function. The first two convolutional layers are followed by average pooling, while the last two layers utilize max pooling. This combination of convolution and pooling layers helps extract and downsample features from the input data. The convolutional layers (`conv1`, `conv2`, `conv3`) aim to learn hierarchical representations of the input. Three fully connected layers (`fc1`, `fc2` and `fc3`) follow the convolutional and pooling layers. The `LogSoftmax` activation function is applied to the output to convert the raw scores into a probability distribution over the classes.



Adam Optimizer	Value
lr	1e-3
betas	(0.9, 0.999)
eps	1e-08
weight_decay	0
amsgrad	False

Early Stopping	Value
monitor	val_loss
patience	5
mode	min
min_delta	0.001

Table 1: **CNN-Model Parameters Setting:** On the left table, I list the setting used in the Best Model selected for the Adam Optimizer. On the right table, instead, I list the Early Stopping setting.

3.2.1 Metrics Performance: Models Comparison

In the **Metrics Performance Evaluation** of my models, **Accuracy** emerged as one of the primary metrics for my **Model Comparison**. While Accuracy provides a comprehensive overview of the model's performance, I also recognize the importance of delving deeper into the nuances of classification through additional metrics. The employment of **Precision**, **Recall**, and **F1-Score**, allows me to gain valuable insights into the models' strengths and weaknesses, metrics conventionally applied in scenarios involving imbalanced datasets as mine experiences.

In addition to these four metrics concerning the test set, I also consider the **Test Loss** and the **Reward** obtained from the **Car Simulator (v2)** as crucial factors in the **Model Selection**. The **Reward** from the Car Simulator v2 is treated as the **main criterion for model selection**. This holistic approach ensures that the selected model not only performs well in terms of traditional classification metrics but also excels in the real-world scenario simulated by the Car Simulator v2.

Note that the following table has intentionally omitted some of the tries, reporting not all the tries, for the best model architecture, but only some of the tries, representing, selectively, the results obtained. Again, the Metric on which I focus is the Reward Value in the Car Simulator (v2).

Models	Acc	F1	Prec	Rec	Loss	Train t	Rew
6MP(C)+4FC	0.718	0.412	0.499	0.489	0.830	5670.6	-∞
6AP(C)+4FC	0.715	0.409	0.487	0.476	0.845	4789.9	-∞
5MP(C)+3FC	0.701	0.397	0.479	0.459	0.864	3665.6	-∞
4MP(C)+3FC	0.700	0.389	0.455	0.4341	0.888	3239.9	124
4AP(C)+2FC	0.700	0.377	0.475	0.457	0.91	1900.3	97
3MP(C)+2FC	0.691	0.389	0.443	0.441	0.9788	784.3	367
4MP(C)+2FC	0.681	0.369	0.425	0.4341	0.948	1340.6	217
2AP(C)+2MP(C)+1FC	0.689	0.384	0.444	0.354	0.932	769.9	451
2AP(C)+1MP(C)+2FC	0.657	0.353	0.415	0.323	1.007	877.7	543
2AP(C)+1MP(C)+3FC	**0.564**	**0.241**	**0.327**	**0.265**	**1.266**	**1080**	$+\infty$

Table 2: **Dataset 1: Metrics Performances Comparison**: In the following, I will report the final results obtained over the application of different architectures over the Training Dataset. The one enhanced on Bold Green is the one that is represented in the image, i.e. the Best Model Architecture 5. Notice that in the table, for brevity, I use some acronyms. We have that xMP(C)=[MaxPool2D+ReLU(ConvLayer)] block repeated x times, xAP(C)=[AvgPool2D+ReLU(ConvLayer)] block repeated x times, xFC=[Fully Connected Layer] repeated x times. Moreover, here, the "x" is the number of those layers I employ in the architecture. Moreover, for what concerns the Reward, I have that I introduce the $-/+\infty$, to indicate, respectively, the fact that the Car in the Car Simulator (v2) is not moving(or doing negative reward), w.r.t. the former that completes the entire circuit successfully and continues to go autonomously-drive in the circuit.

3.3 Model Comparison and Sensitivity Analysis

Having identified the best-performing Convolutional Neural Network (CNN) model through rigorous evaluation and fine-tuning, we proceed to a broader exploration of model variations and different architectures. This phase involves comparing the chosen CNN architecture with alternative architectures, as well as experimenting with different optimization strategies, loss functions, and hyperparameters.

The objective is to conduct a **Sensitivity Analysis** to understand how variations in model components impact performance. This exploration will contribute valuable insights into the robustness and generalization capabilities of the selected model. The comparative study involves assessing metrics such as accuracy, precision, recall, F1-score, test loss, and reward, considering both the training and test datasets.

Optimizers In this section, I present the different results that the actual Best-Model Architecture obtains, with a different **Optimizer**.

Models	Acc	F1	Prec	Rec	Loss	Train t	Rew
Adam	**0.56413**	**0.24156**	**0.32772**	**0.26575**	**1.26621**	**1080**	$+\infty$
SGD	0.69727	0.42456	0.49308	0.38955	0.88631	988.7	412
Adagrad	0.60012	0.27991	0.35599	0.24391	1.15882	1583	401
RMSprop	0.55826	0.26482	0.34995	0.22982	1.32451	1207.7	710
Adadelta	0.69494	0.39752	0.45743	0.36559	0.87338	1780.9	439
Adamax	0.52450	0.20937	0.29683	0.17367	1.31749	1378.5	362

Table 3: **Optimizers Sensitivity Analysis:** In this table, I present how the variation over the Optimizers in the Architecture affects the Metrics I am taking into account. In particular, you can observe that with the current architecture, changing the Optimizer causes the car to not finish the Circuit.

Loss In this section, I present the different results that the actual Best-Model Architecture obtains, with a different **Loss Function**.

Models	Acc	F1	Prec	Rec	Loss	Train t	Rew
WCE+Smoth	**0.56413**	**0.24156**	**0.32772**	**0.26575**	**1.26621**	**1080**	$+\infty$
NLLL	0.63386	0.32698	0.39297	0.2955	1.10354	782	602
FocalLoss	0.63511	0.30603	0.37369	0.27367	0.58776	1708	420

Table 4: **Loss Sensitivity Analysis:** In this table, I present how the variation over the Loss in the Architecture affects the Metrics I am taking into account. In particular, you can observe that with the current architecture, changing the Loss causes the car to not finish the Circuit. However, another thing to notice is that, by using the Focal Loss, especially focusing for Imbalanced Dataset situations, the Loss is more than halved w.r.t. the Weighted Cross Entropy + Label Smoothing current solutions. This, however, obtains a smaller reward.

4 Conclusions

In this study, I undertook a comprehensive exploration of convolutional neural network (CNN) architectures to identify a model that excels in a simulated environment, specifically the CarRacing-v2 simulator. Through a systematic exploration of various CNN architectures, loss functions, and optimizers, I arrived at several noteworthy findings.

4.1 Best Model Architecture

The final selected model exhibits a unique architecture that proved to be particularly effective for the given task. Comprising two consecutive average pooling layers followed by a ReLU activation and a convolutional layer, and augmented by one max pooling layer with a subsequent ReLU-activated convolutional layer, this combination demonstrated exceptional performance. The architecture is further complemented by three fully connected layers, providing the network with the capacity to learn intricate features from the input images.

4.2 Loss Function and Optimizer Selection

The choice of loss function and optimizer played a pivotal role in model training. The weighted cross-entropy loss, enriched with label smoothing, emerged as a suitable candidate for handling imbalanced class distributions. Additionally, the Adam optimizer was employed to efficiently navigate the model parameters during the training process.

4.3 Depth vs. Performance Trade-off

An intriguing observation was the trade-off between model depth and performance. While deeper architectures exhibited higher accuracy on traditional image classification metrics, the reward-driven selection favored a more streamlined architecture. This finding underscores the importance of aligning model evaluation metrics with the specific requirements of the intended application.