Machine Learning - Homework #2

(2044902) Antonio Rapuano

January 2024

Contents

1 Introduction

The goal of this homework is to put into practice the concepts acquired on convolutional neural networks during the lectures by solving an image classification problem with the ultimate (optional) goal of learning a function that is capable of driving a race car in the ”Car Racing” Gym environment. The dataset (provided already split into training and validation sets) contains color images of size 96×96, each labeled with one of five actions available for race car control.

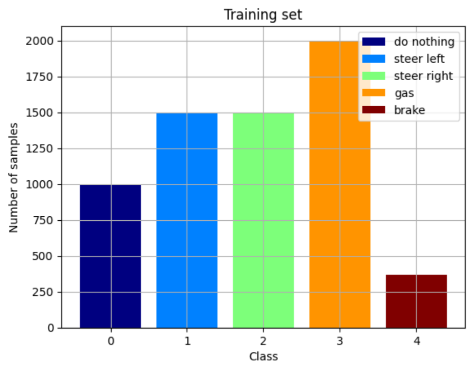
It is required to define two different approaches, train them with different hyperparameters, compare them, and visualize and discuss the results. For this specific report, the Python language (and the Keras library) is used via the Jupyter application.

2 Workflow

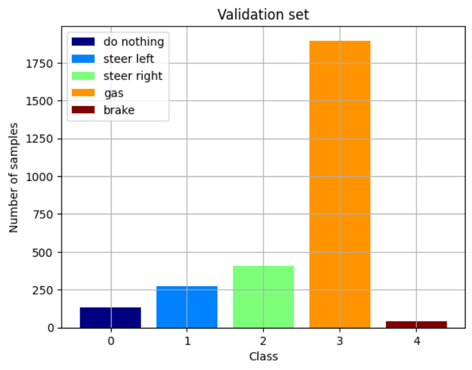
2.1 Data import and preprocessing

The color images that make up the dataset (called *observations*), once imported into the work environment, can be represented as tensors, each $96 \times 96 \times 3$ in size (96 pixels high, as many wide, and 3 RGB color channels). They are originally grouped in folders that are labeled with the class id to which they belong, corresponding to the action of the car in the racing video game: a random sample is shown in Fig. 2.2a.

In total, 9,118 samples are available, of which 6,369 belong to the training set and the remaining 2,749 to the validation set. However, the classes are not balanced in terms of the number of elements, and nor does this imbalance occur in the same way in the two subsets (see Fig. 2.1a and 2.1b).
Since it would not be fair to modify the way the subsets are split, this suggests that accuracy problems may arise in the solution. In fact, in general, the division between the training and validation sets should occur randomly, and consequently, they should be distributed in the same way even if class-imbalanced.
Furthermore, another question arises: how well does the training set represent the ”correct” behavioral model of the player in the Car Racing environment? Based on the answer to this question, we will obtain a model capable of navigating the race track or one that will crash the car at the first turn.



(a) Training set.



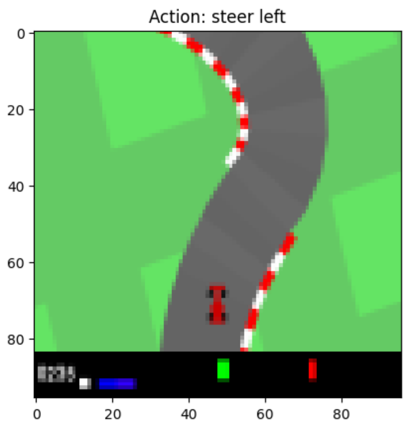
(b) Validation set.

Figure 2.1: Composition of the dataset.

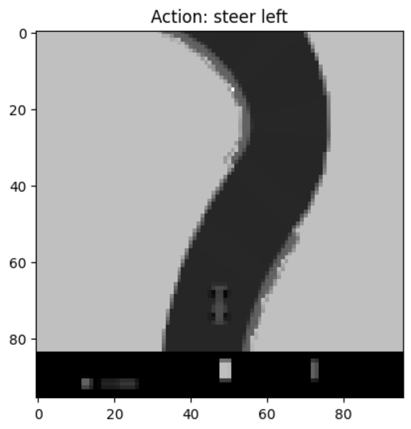
Another aspect to consider is the information contained in a single observation. In fact, it is common practice to preprocess images before feeding them into a classification model (both instances belonging to the dataset and not), mainly to reduce computational load and to improve training performance. For this particular problem, the following manipulations are applied:

1. the texture of the grass is made uniform;
2. the saturation is enhanced;
3. the colors are converted to grayscale;
4. the reward counter in the lower left corner is concealed;
5. the elements of the tensor are normalized to 1.

This way, we expect the model to converge more easily to an acceptable solution and to have fewer issues in generalizing what it will learn from the data.
In Fig. 2.2b the preprocessing of the previous random sample is carried out as an example.



(a) A random sample of the dataset.



(b) The same sample after preprocessing.

Figure 2.2: Effect of preprocessing on the observations.

2.2 CNN tuning and fitting

As requested, two convolutional neural networks have been implemented and trained.

The idea behind model 0 is to implement large filters that are able to capture the general structure of the input image (i.e., a tensor of shape $96 \times 96 \times 1$), which shrink in size as the network goes deeper. The fully-connected section is composed of three dense layers of decreasing number of units.
On the other hand, model 1 implements smaller filters that are able to capture the details of the observations. Its dense part is made of two layers with a greater number of units than model 0.

For both structures, depicted in Fig. 2.3, the following choices have been made:

- the activation function is ReLU, except for the output layer, where softmax is used to obtain a probability distribution over the 5 classes;

- the convolutional layers are followed by a max pooling layer with strides always equal to the pool size in order to reduce the dimensionality of the feature maps;
- overfitting is prevented by using dropout layers after the dense layers, and by applying weight decay (with l2 regularization penalty always equal to 0.001) to the convolutional and dense layers.

Model 0 has 84,901 trainable parameters, while model 1 has 80,213.

About the training phase, take note the following:

- in both cases, the maximum epoch number is 50, the batch size is 64, and the loss function is sparse categorical crossentropy (since the labels are integers);
- once again to prevent overfitting, early stopping is activated from the 20th epoch onwards with a patience of 6 epochs by monitoring loss over the validation set;
- model 0 is trained with the Adam optimizer, while model 1 is trained with Nadam (a variant of Adam that incorporates Nesterov momentum), both with a learning rate of 0.001;

2.3 Model evaluation

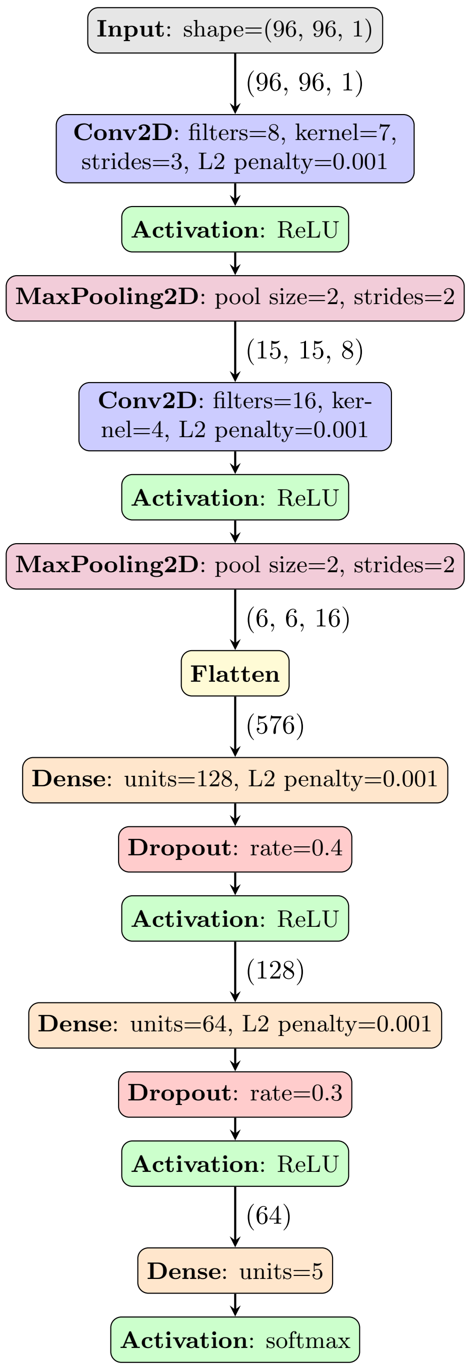
The performance of the two models can be evaluated by looking at some specific metrics. In particular, the following ones are considered:

1. **Loss and accuracy vs. epochs graphs**: observing the trend of these metric during the fitting phase is useful to understand whether the model is overfitting or not;
2. **Classification report**: having a look at the main classification metrics both globally and class-wise is particularly useful when the dataset is unbalanced, as in this case;
3. **Confusion matrix**: this completes the classification report by showing the number of correct and incorrect predictions for each class.

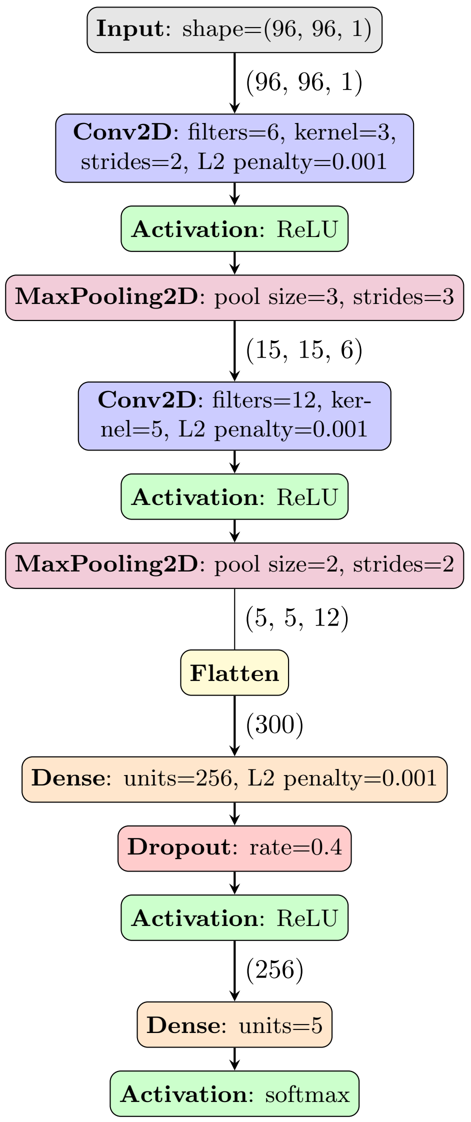
Obviously, the models could also be evaluated on the basis of how well they manage to drive the car in the Car Racing environment, but since we do not know the ground truth for the actions to be taken in each state or the relation with the validation set, this is not a reliable metric.

2.4 Hyperparameter fine-tuning

After the the training and evaluation phases are over, we may consider how satisfactory the results are and whether we can improve them by tuning the hyperparameters. There are lots of hyperparameters that can be fine-tuned, and as a matter of fact the ones in the models described in Sec. 2.2 and shown in Fig. 2.3 are not chosen randomly, but they are the result of a preliminary tuning phase based on trial and error that is not reported here for the sake of brevity.
Instead, we are only going to focus on the analysis of the hyperparameters whose effect is more evident (see Sec. 3.3), namely the weights of the classes used during training, by relating them to the aforementioned performance metrics.


Input: shape=(96, 96, 1)
(96, 96, 1)
Conv2D: filters=8, kernel=7, strides=3, L2 penalty=0.001
Activation: ReLU
MaxPooling2D: pool size=2, strides=2
(15, 15, 8)
Conv2D: filters=16, kernel=4, L2 penalty=0.001
Activation: ReLU
MaxPooling2D: pool size=2, strides=2
(6, 6, 16)
Flatten
(576)
Dense: units=128, L2 penalty=0.001
Dropout: rate=0.4
Activation: ReLU
(128)
Dense: units=64, L2 penalty=0.001
Dropout: rate=0.3
Activation: ReLU
(64)
Dense: units=5
Activation: softmax

(a) Structure of model 0.


Input: shape=(96, 96, 1)
(96, 96, 1)
Conv2D: filters=6, kernel=3, strides=2, L2 penalty=0.001
Activation: ReLU
MaxPooling2D: pool size=3, strides=3
(15, 15, 6)
Conv2D: filters=12, kernel=5, L2 penalty=0.001
Activation: ReLU
MaxPooling2D: pool size=2, strides=2
(5, 5, 12)
Flatten
(300)
Dense: units=256, L2 penalty=0.001
Dropout: rate=0.4
Activation: ReLU
(256)
Dense: units=5
Activation: softmax

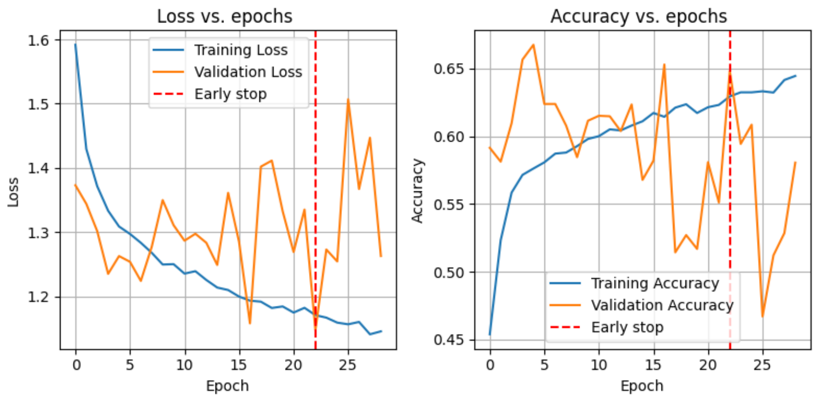
(b) Structure of model 1.

Figure 2.3: Comparison of CNN structures. The padding is always *valid*, and where not specified the strides are set to 1.

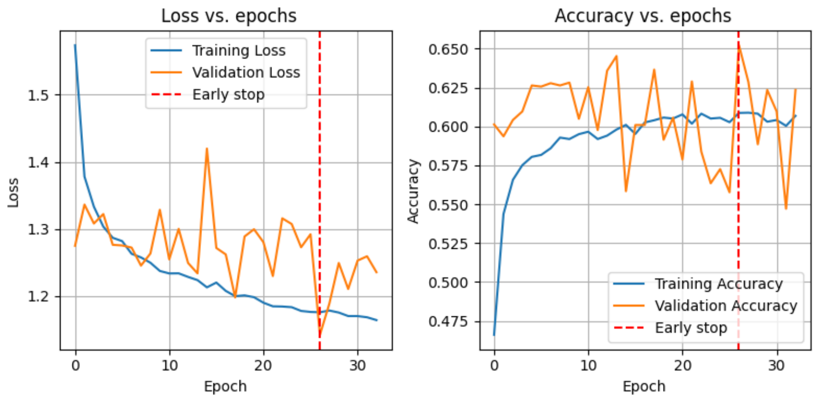
3 Results

3.1 Loss and accuracy vs. epochs

As anticipated, Fig. 3.1 allows to visualize the trends of training and validation loss and accuracy of the two models during the training phase relatively to the epoch number.
Observe that, while the metrics relative to the training set smoothly decrease and increase respectively, the ones relative to the validation set show a more erratic behavior. It is probably due to the composition of the dataset, which is possibly noisy and unbalanced, and to the fact that the validation set is not very large: this is supported by the fact that this phenomenon occurrs for both models, even though model 1 is slightly less affected by it, probably because of its structure and of the Nesterov momentum in the Nadam optimizer.
Anyway, the early stopping callback does its job of preventing overfitting: the best weights are achieved at the 22nd epoch for model 0 and at the 26th for model 1, and those are the ones that are used for the evaluation phase.



(a) Model 0.



(b) Model 1.

Figure 3.1: Graph of loss and accuracy of the two models during the training phase, highlighting the point of early stopping.

3.2 Classification scores

Even though we have taken many precautions with the purpose of maximizing the performance of the models, the actual results turned out to be quite disappointing. As a matter of fact, the classification reports in Tab. 3.1 show that the models are not able to correctly classify a great portion of the samples in the validation set.
The important thing to notice is that the lowest scores come from the classification of class 4, i.e. *brake*. The reason for this is probably due to the fact that it is the least represented in the dataset, and therefore the models have not been able to learn the features that characterize them. This is confirmed by the fact that the results become proportionally better the higher the number of samples in the specific class, even though they are still far from being good.

Class	**Precision**	**Recall**	**F1-score**
0	0.24	0.47	0.31
1	0.38	0.58	0.46
2	0.52	0.68	0.59
3	0.87	0.68	0.76
4	0.05	0.08	0.06
Accuracy			0.65
Macro avg	0.41	0.50	0.44
Weighted avg	0.72	0.65	0.67

(a) Model 0.

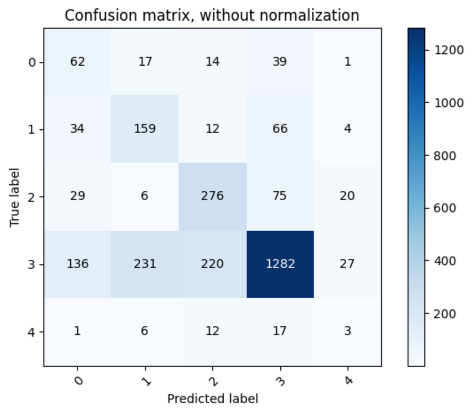
Class	**Precision**	**Recall**	**F1-score**
0	0.28	0.38	0.32
1	0.36	0.54	0.43
2	0.49	0.65	0.56
3	0.84	0.70	0.77
4	0.06	0.05	0.05
Accuracy			0.65
Macro Avg	0.40	0.46	0.43
Weighted Avg	0.70	0.65	0.67

(b) Model 1.

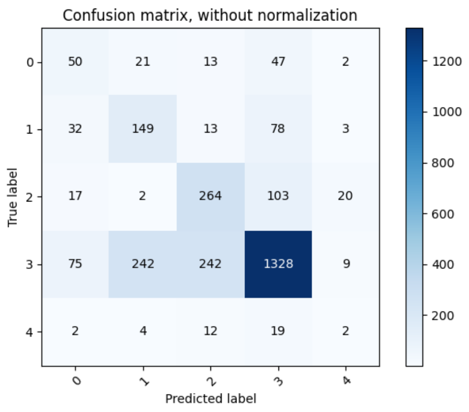
Table 3.1: Classification reports of the two models, showing class-wise precision, recall and f1-score, and global accuracy over the validation set.

The increased level of detail provided by confusion matrices (see Fig. 3.2) does not tell a different story. However, it is way more useful when it comes to adjusting the models in order to make them drive in the Car Racing environment: for instance, steering left instead of braking on a straight is not as big of a deal as accelerating instead of braking when a curve is coming up.

After having tried with minimum improvement and considerable effort to obtain better performance by tuning the structures and the hyperparameters of the models (see Sec. 3.3), the logical conclusion is that the dataset needs to be improved and possibly its split into training and validation sets needs to be changed and/or randomized.



(a) Model 0.



(b) Model 1.

Figure 3.2: Confusion matrices of the two models over the validation set.

3.3 Effect of class weights

In the previous section we have speculated that the poor performance of the CNN on the test set is due to the unbalanced nature of the dataset. Among the most common techniques to deal with unbalanced datasets, using class weights is one of the simplest and most effective (unlike oversampling, it does not require additional computing power, nor does it cause loss of information in the dataset like undersampling). In this section we will evaluate the effect of different class weights on the validation performance of the CNNs.
The experiments were carried out with the configurations transcribed in Tab. 3.2. In particular, observe that:

- **W1** corresponds to the default configuration, i.e. no class weights;
- **W2** is computed automatically to balance the training set;
- **W3** is composed of the ratios between the number of samples in each class in the test set and in the training set;
- the remainder is chosen arbitrarily.

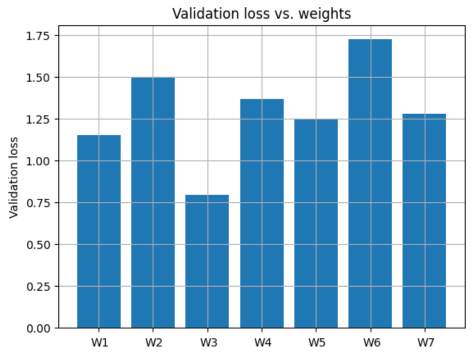
Weights	Class 0	Class 1	Class 2	Class 3	Class 4
W1	1.000	1.000	1.000	1.000	1.000
W2	1.274	0.849	0.849	0.637	3.452
W3	0.133	0.183	0.271	0.948	0.106
W4	1.200	1.000	1.000	0.700	2.200
W5	0.900	1.000	1.000	0.800	1.800
W6	1.000	1.000	1.000	0.200	2.000
W7	0.600	0.700	0.700	0.400	2.000

Table 3.2: Class weights used for the experiments.

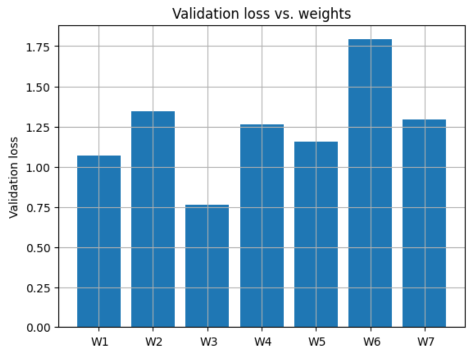
As seen in Sec. 3.2, evaluating the class-wise performance is as crucial as evaluating the overall one. For the latter we can use validation loss (do not forget that this is the metric monitored by the early stopping callback, hence we will necessarily look at the best one after epoch 20), while for the former we may use f1-score, which summarizes the precision and recall scores for each class.

The plots in Fig. 3.3 and 3.4 show the outcomes of the experiments. Observe that:

- using no class weights (W1) is definitely better than using the ones computed automatically (W2), since the validation set unbalance is even more severe than the training set one;
- W3 is the best configuration in terms of validation loss since it is taylored on the test set, but it scores 0 in terms of f1-score for class 4 for both models, thus it would not be a good choice for driving the race car;
- W6 penalizes excessively the majority class 3 and (as a consequence) the validation loss skyrockets, thus it is the worst configuration;
- finally, the two models score significantly different f1-scores with the same weights - especially using W3 and W6 - but the validation losses are very similar.

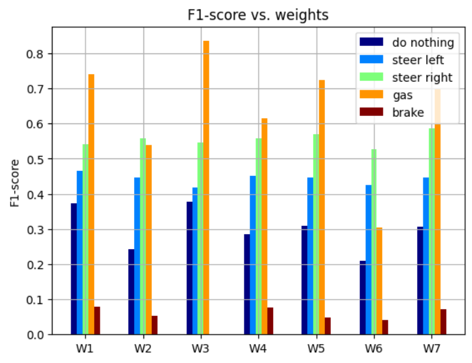


(a) Model 0.

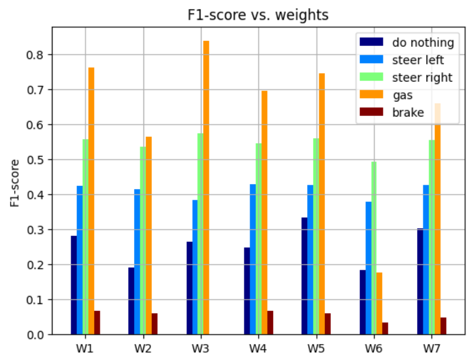


(b) Model 1.

Figure 3.3: Effect of different class weights on the validation loss.



(a) Model 0.



(b) Model 1.

Figure 3.4: Effect of different class weights on the class-wise f1-score.

The final choice of weights is W4, which is the one that provides the best trade-off between validation loss, f1-scores and driving capabilities (see App. A). As a matter of fact, the results reported in the previous sections were obtained with this configuration.

4 Conclusions

Let us sum up the main results of this report. Two different CNNs (with different structures and hyperparameters) have been built and used to solve the image classification problem that was proposed. The dataset has been preprocessed in order to delete unnecessary and/or misleading information. With the objective of improving performance, adjustments have been made, precautions have been taken and experiments have been carried out. The threat of overfitting has been kept away with regularization strategies and early stopping. Moreover, the analysis of the performance in realtion to the class weights has shown that the choice of the weights is crucial for the success of the models.

Despite these efforts, though, the final results are not outstanding: the main problem could be that the dataset is not sufficient to train a CNN that can generalize well, especially on the given validation set, which is quite different from the training set. Supporting this there is the fact that the models score quite similarly to each other in the tests that have been conducted, so the problem would not seem to reside in the network tuning phase.

A Driving a race car

In order to allow the two models to drive a race car in the Car Racing environment, they should be included in a feedback loop, as shown in Figure A.1. Based on an initial observation, the agent (the CNN) takes an action, which is passed to the environment. The environment then computes a response and returns a new observation, which is passed to the agent. The agent then takes another action, and so on, until the game ends (because of the car either crashing or completing a full lap, or because of the expiration of the maximum time of 30 seconds).

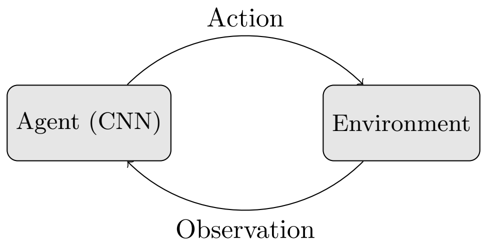


Figure A.1: Closed loop between the CNN and the Car Racing environment.

Mainly due to the class weights choice of Sec. 3.3, both the models are capable of playing the racing game with a good degree of success on a great number of randomly-generated tracks, as shown in Fig. A.2 and in the video linked in the footnote[1]. However, there is still a lot of room for improvement, as the models sometimes act in a way that is not optimal and that can possibly lead to "crashes", as shown in Fig. A.3. With this in mind, it would probably be best to combine a CNN model with one of reinforcement learning that also exploits the reward system of the Gym environment.

[1]YouTube video showing the two models driving on randomly-generated tracks: `https://youtu.be/EUGlU5wEMvg`

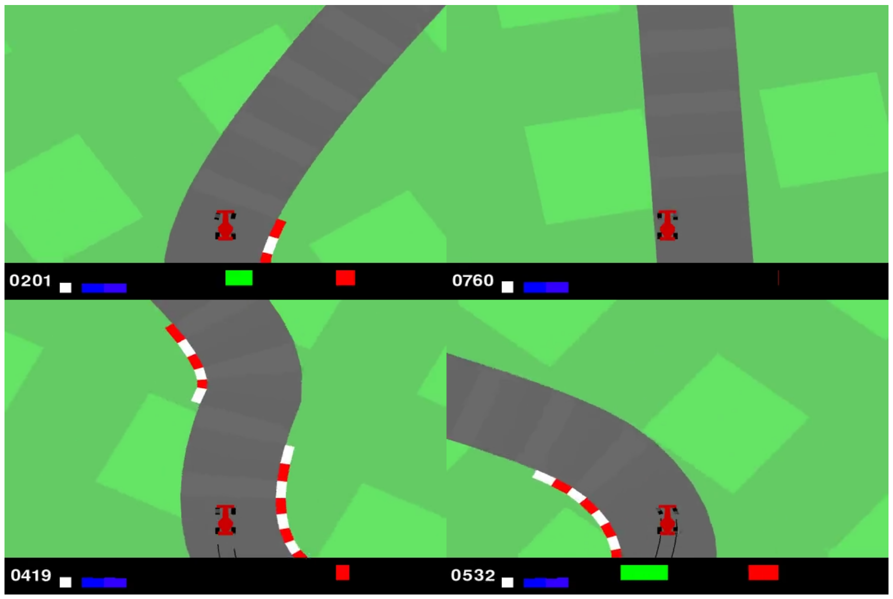


Figure A.2: The models driving in the Car Racing environment.

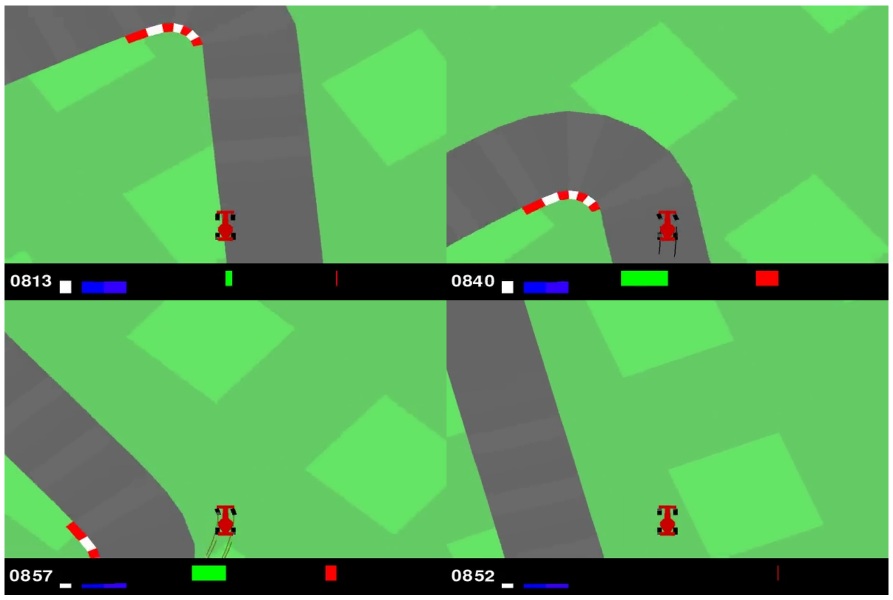


Figure A.3: Model 0 causing the race car to skid off the road.