SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence and Robotics

Homework 2 - Image Classification

Machine Learning

Student:
Giovanni Zara
1929181

Academic Year 2024/2025

Contents

1 Introduction

This project explores the application of machine learning techniques to solve a car racing control problem within a 2D simulation environment. The objective of this study is to develop a machine learning model capable of mapping images from the racing environment into specific car control actions, such as steering, accelerating, braking, or maintaining the current trajectory. The problem is framed as a supervised 5-class image classification task, where images represent states of the car's environment and are labeled with one of five possible actions.

The dataset employed in this study comprises 96x96 RGB images, organised into folders based on their respective action classes. The data were collected through a Reinforcement Learning-trained policy, which provides a mapping from the car's visual input to its control actions. While the primary focus of the project is on image-based classification, the task also allows for optional inclusion of car sensor data (e.g. speed, steering angle, and gyroscopic data) to enhance predictions.

To address this challenge, various convolutional neural network architectures have been implemented as models , given the proven effectiveness of CNN in image-based tasks. The project involves several key stages, including data preprocessing, model training, performance and loss evaluation, and some simulations. A systematic hyper-parameter tuning process was conducted to optimize model performance. Experimental comparisons were made to evaluate the impact of different configurations of hyperparameters and architecture structures on the classification efficiency.

This report delineates the methodology, experiments, and findings.

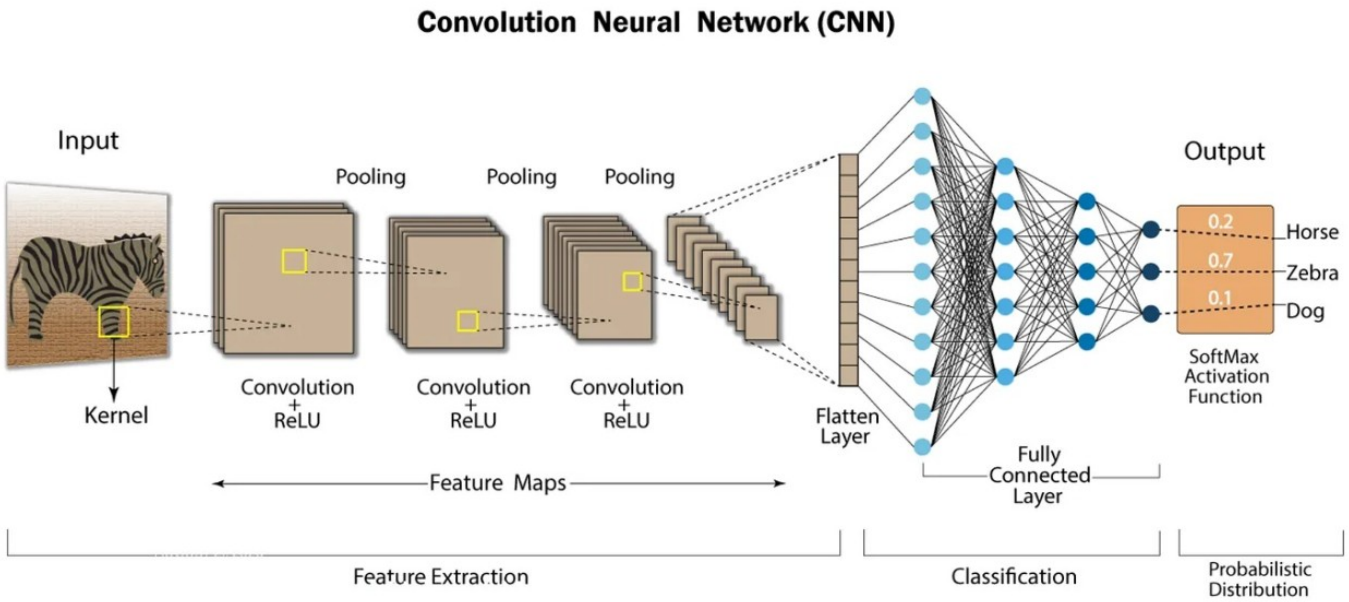

Figure 1: Typical CNN structure

1.1 Dataset

The dataset utilised for the analysis comprises a series of images extracted from a video of a car navigating a circuit, with each image representing a 96x96 RGB (three-channel) representation of the car's visual input. The dataset has been pre-divided into a training set and a test set, with the training set further sub-divided to create a validation set as follows:

- Training Dataset: Eighty percent of the data in the train directory was used for training the model.
- Validation Dataset: Twenty percent of the data in the train directory was used to validate the model's performance during training.
- Test Dataset: The images in the test directory were used for final model evaluation, ensuring unbiased performance metrics.

Through tensorflow, a tf.data.Dataset object was created directly from a directory of images, making it convenient to prepare images for the model. Each subdirectory was designated as a class, and the images contained therein were automatically assigned to the corresponding class. The class labels inferred from the folder structure were saved for reference in one-hot encoding, representing the five possible actions that the car could take: turn right, turn left, do nothing, gas and brake. The class names for the training, validation, and test sets were recorded separately to ensure consistency and to verify that the labels matched across all subsets.

To prepare the images for input into the neural network, a normalization step was applied. Pixel values in the raw images range from 0 to 255, has been known to hinder the model's ability to converge during training. To address this, a normalization layer was added to scale pixel values to the range [0, 1]. The normalization was applied to the training, validation, and test datasets using the map() function, ensuring consistent preprocessing across all subsets. In order to optimise data loading and improve training performance, the preprocessed datasets were configured to use TensorFlow's prefetch() operation. Prefetching facilitates the preparation of data for the subsequent training step during the execution of the current step, thereby minimising data loading bottlenecks that may be caused by the limited hardware power available.

1.1.1 Dataset imbalance

In the context of this project, the utilisation of data augmentation techniques was deliberately avoided, despite the dataset exhibiting a substantial imbalance across the five action classes.Data augmentation, such as flipping, rotating, or adding noise to the images, is a commonly employed technique to artificially augment the size of the dataset and address class imbalances. However, in this particular scenario, augmentation was deemed unnecessary for two primary reasons:

1. The necessity for precise labels: the car's performance in the racing simulation is dependent on the model's capacity to predict precise actions based on visual input, and the introduction of augmented

images could potentially alter the input distribution in ways that do not accurately represent the car's actual environment in the simulation, thereby leading to degraded performance.

2. Simulation-Specific Challenges: the primary objective considered here is to enable effective driving behaviour over improved metric values. While the incorporation of augmented data might enhance certain metrics, it is likely to result in erratic driving behaviour, thus undermining the model's intended purpose.

Similarly, no up-sampling or down-sampling techniques were employed to balance the dataset. While these techniques could potentially enhance classification metrics, they were considered counterproductive for the present application. The primary motivation behind this choice was that balancing the dataset might cause the model to over-prioritize certain underrepresented actions (e.g., "brake") beyond their actual frequency in the simulation, which could disrupt the natural distribution of actions and result in poor driving performance.

Instead of modifying the dataset, the focus remained on training the model on the raw, imbalanced data, an approach that prioritises the practical purpose of the model over achieving balanced metrics in a purely academic setting. As seen in the last section, the only balancing try was made through an alternative loss function, without alterating datas.

2 Metrics

2.1 F1 score

In order to evaluate the performance of the classification model, traditional metrics such as accuracy were initially considered. However, given the highly imbalanced nature of the dataset, reliance on accuracy alone would have resulted in incomplete and potentially misleading conclusions. Accuracy is known to favour the majority classes, as it reflects the overall proportion of correct predictions, regardless of how well the model performs on minority classes. Consequently, the analysis shifted its focus to the F1 score, a metric specifically designed to balance precision and recall, thereby providing a more robust evaluation in the context of imbalanced datasets.

The F1 score is calculated as the harmonic mean of precision and recall. Precision is defined as the proportion of true positive predictions among all predicted positives, whereas recall is the proportion of true positives among all actual positives. The F1 score's integration of both precision and recall, thereby, ensures that it is particularly well-suited for datasets where certain classes are underrepresented, as it captures both the accuracy of the positive predictions and the model's ability to identify all relevant instances.
In this project, two variations of the F1 score were analyzed:

- Global F1 Score (F1_score1): This version computes precision, recall, and the F1 score across all samples in the dataset collectively, treating the classification task as if it were binary. It calculates true positives, false positives, and false negatives globally, without breaking down performance by class. While this approach provides a single overall F1 score, it tends to overestimate performance in imbalanced datasets, as the majority classes dominate the metric.

- Macro-Averaged F1 Score (F1_score2): This variation calculates precision, recall, and F1 score separately for each class and then averages the results. Unlike the global approach, it treats each class equally, regardless of how many samples it contains, making it more robust for imbalanced datasets. However, it can produce lower values, as it penalizes poor performance on underrepresented classes more strongly.

In view of the characteristics outlined above, F1_score2 (the macro-averaged F1 score) was selected as the primary metric for this project. Despite the fact that the global F1 score appeared higher, it was considered to be less reliable in capturing the model's weaknesses on minority classes. By relying on F1_score2, the analysis ensured a balanced evaluation of the model's predictions, reflecting its ability to generalise well across all classes. This decision was crucial in achieving the project's objective of generating reliable predictions that would lead to smooth driving behaviour in the car simulation.

3 CNN models - first type

In this project, a range of CNN architectures were evaluated. However, following a final analysis, the models were categorised into two primary types based on the loss function employed. The following section will evaluate the architecture using categorical crossentropy loss.

3.1 Loss Function: Categorical Crossentropy

The **categorical crossentropy** is one of the most suitable choicess for multi-class classification problems. This loss function measures the dissimilarity between the true labels (ground truth) and the predicted probability distribution produced by the model.
Categorical crossentropy is defined mathematically as:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij}\log(\hat{y}_{ij})$$

Where:

- N is the total number of samples in the batch.
- C is the total number of classes.
- y_{ij} is the one-hot encoded ground truth label for sample i and class j (1 if the sample belongs to class j, 0 otherwise).
- $\hat{y}_{ij}$ is the predicted probability for sample i belonging to class j (output of the softmax layer).

The function works by summing the logarithmic difference between the predicted probabilities and the true labels for each class and averaging this over all the samples in the batch. The negative sign ensures that higher probabilities assigned to the correct class lead to a lower loss.

The efficacy of categorical crossentropy for multi-class classification tasks is attributable to its capacity to evaluate the discrepancy between the predicted probability distribution and the true label distribution, whilst concomitantly encouraging the model to output high confidence for the correct class by penalising incorrect predictions.

3.2 Models

In this section, four CNNs architectures were explored. The models ranged from relatively simple designs to more complex architectures with additional layers, advanced pooling strategies, and hyperparameter configurations. Below, each architecture is summarized, highlighting their design and key differences, while outlining the progression from the simplest to the most complex.

CNN 1a

The initial architecture, designated as CNN 1a, functioned as the baseline model. It comprised three convolutional layers, followed by max-pooling layers, a flattening layer, two fully connected dense layers, using Adam optimizer (as all the following models). The activation function employed in the convolutional and dense layers was ReLU, with a final softmax layer for the 5-class output. The model utilised a fixed stride of (2, 2) to accelerate computation and prioritised simplicity to establish an initial performance benchmark. Despite its simplicity, this model achieved a best validation F1 score of 0.576 and a test accuracy of 0.575, demonstrating its ability to reach satisfactory metrics.

CNN 1b

Building upon the baseline, CNN 1b introduced dynamic hyperparameter configurations and additional regularization techniques. The convolutional layers were dynamically added based on the num-layers parameter, with batch normalization included after each convolution to stabilise learning. Regularization was applied using L2 penalties, and dropout layers were added to mitigate overfitting. Global Average Pooling was here tried in place of the flattening operation. Despite these enhancements, the model attained a maximum validation F1 score of 0.500, accompanied by a test accuracy of 0.439 and a test F1 score of 0.308, indicating difficulties in generalization, despite its increased complexity relative to the initial model.

CNN 3

The third architecture, CNN 3, expanded CNN 1b by incorporating flexibility in both the convolutional and fully connected layers. In addition to the variation in the number of convolutional layers, the model permitted multiple dense layers, with the number of units in each dense layer designated as a hyperparameter. The impact of dropout was investigated, both with and without application. The dropout version demonstrated slightly diminished training metrics but analogous test metrics, suggesting reduced overfitting. The highest validation F1 score achieved with dropout was 0.501, accompanied by a test accuracy of 0.458 and a test F1 score of 0.306. In the absence of dropout, the validation F1 score exhibited a marginal increase to 0.510, while test performance remained largely unchanged.

CNN 4

The most advanced architecture, CNN 4, incorporated additional flexibility in pooling strategies and feature extraction methods. A comprehensive exploration of both max-pooling and average pooling was conducted, with Global Average Pooling being compared to a flattening operation. This model offered the most comprehensive hyperparameter grid, including variations in the number of convolutional and dense layers, stride size, dense layer units, and learning rate. Following extensive tuning and a very large amount of training time, the optimal model was determined to have the following hyperparameters: 'dense units': 128, 'epochs': 30, 'global average pooling': False (so flatten was used), 'learning rate': 0.0005, 'max pooling': True (so avg pooling was not

used), 'num dense layers': 3, 'num convolutional and pooling layers': 3, 'stride': (2, 2). Despite its complexity, the model achieved a best validation F1 score of 0.432 and a validation accuracy of 0.582. The lower F1 score suggests that the increased architectural complexity does not always translate into better handling of the data, especially in this specific case of highly imbalanced dataset. Nevertheless, this model achieved the highest test accuracies, close to 70%, indicating effective generalisation despite the modest training-validation metrics. Consequently, while simpler models surely turned out worthy of recognition, this particular model is the one considered for the testing on the simulation drive (simulation video attached as external file).

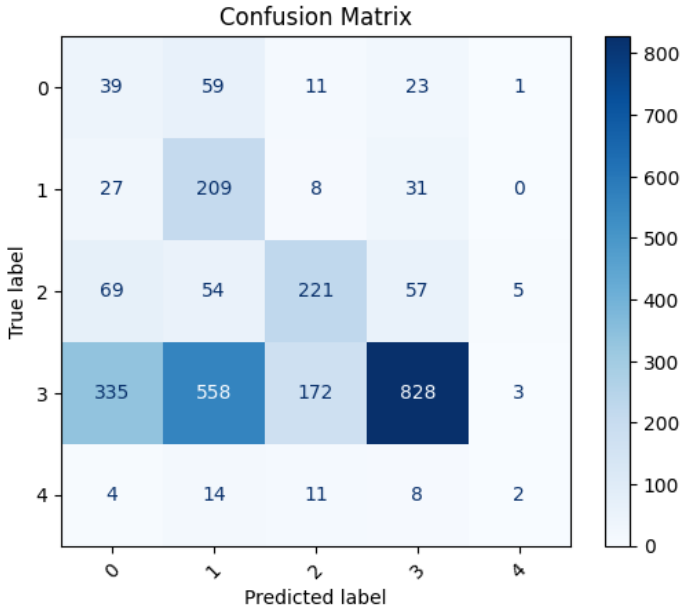

Figure 2: confusion matrix of CNN 4 model - the dominance of class 3 (most present one) is appreciable

4 CNN model - second type

In this section, a new model is examined, constructed from the CNN 4. The grid structure and the architecture remained relatively unaltered, with the sole variation being the loss function.

4.1 Focal loss function

After some researches the loss function that was tried as a possible alternative was focal loss. Suffering the dataset from class imbalance, a model trained on such imbalanced data tends to fail predictions not in the majority class (as clearly visible from the above confusion matrix). The model can end up ignoring underrepresented classes and fail to learn meaningful features from them. While the use of focal loss may improve certain performance metrics by addressing class imbalance, it is not guaranteed that these improvements will directly translate into better driving performance in the simulated environment (as anticipated in introduction section). In a driving simulation, class imbalance can occur intentionally to reflect the dynamics of driving behaviour, where certain actions are more frequent than others. For example, the "gas" action (the class number 3, that I know being dominant) may be much more common than other actions like "braking" or "turning," because, in many driving scenarios, the vehicle is often accelerating or cruising at a steady speed, while other actions are less frequent but still critical for controlling the car.

So,the idea is to down-weight easy examples and focus on hard, misclassified examples.
In this analysis two variants were used.

- One calculated as:

$$\text{loss} = -\alpha \cdot y_{\text{true}} \cdot (1 - y_{\text{pred}})^{\gamma} \cdot \log(y_{\text{pred}})$$

  where:

  - y_{true} is the true label,
  - y_{pred} is the predicted probability for the true class,
  - α is the class balancing factor,
  - γ is the focusing parameter.

  Then loss is summed over all classes (across axis=1), giving one value per sample.

- And the other including class-specific weighting using the parameter α, which now is a list containing the weight for each class (subsection focal loss model - focal_loss_w of the code).
  The weighted loss for a single sample is always given by:

$$\text{loss} = -\alpha_{\text{factor}} \cdot (1 - y_{\text{pred}})^{\gamma} \cdot \log(y_{\text{pred}})$$

but with α_{factor} being the class-specific weight factor.

Then, the mean loss across the batch is returned.

The calculation of α necessitated the computation of the inverse of the class frequencies, to assign higher weights to underrepresented classes. For each class, the number of samples in the training dataset was counted and then normalised so that they summed to 1, and the α vector was populated.

4.2 Model and conclusions

As previously stated, the model's architecture was based on CNN 4. The initial attempt incorporated the unweighted focal loss, which yielded suboptimal metrics. The subsequent attempt employed the weighted focal loss (essentially weighting less the class 3), resulting in satisfactory metrics performance. The model that demonstrated the optimal performance (always based on validation F1 score as before) was:'dense units': 64, 'epochs': 30, 'global average pooling': False(so used flatted), 'learning rate': 0.0001, 'max pooling': False(so used average), 'num dense layers': 3, 'num convolutional and pooling layers': 3, 'stride': (2, 2) with a test accuracy not far from the CNN 4 one. Despite these results, it is important to note that the new weighted focal loss did not lead to significant improvements in performance as initially expected. However, a closer inspection of the confusion matrix revealed noteworthy enhancements, suggesting that the model is actually producing a greater number of accurate right predictions. This observation is further supported by the simulation video, which depicts the vehicle executing a greater number of left-to-right movements during the drive. However, it is evident that the overall driving performance of this model remains inferior to that of CNN 4. This finding reinforces the notion that the enhancement in label prediction does not necessarily translate into an improvement in driving behaviour, emphasising the potential discordance between performance metric optimisation and enhanced driving capabilities in the simulation (simulation videos attached as external files).

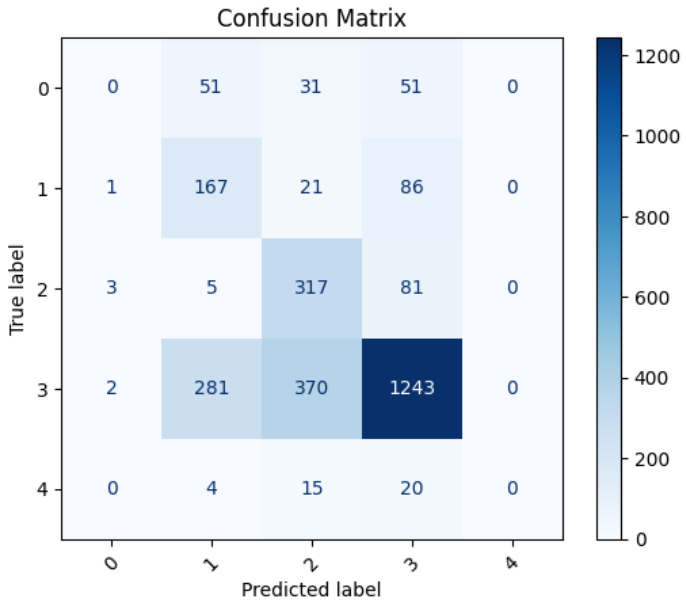

Figure 3: consufion matrix of the weighted focal loss model