SAPIENZA
UNIVERSITÀ DI ROMA

Machine Learning

Homework II 2024/2025

Daniele Sabatini

matricola:

1890300

Contents

1 Introduction

The main goal of the project is to solve the image classification problem using a convolutional neural network (CNN). The task involves mapping visual input (images) to driving actions, such as steering, accelerating, and braking. The project is based on the Car Racing v2 environment provided by Gymnasium.

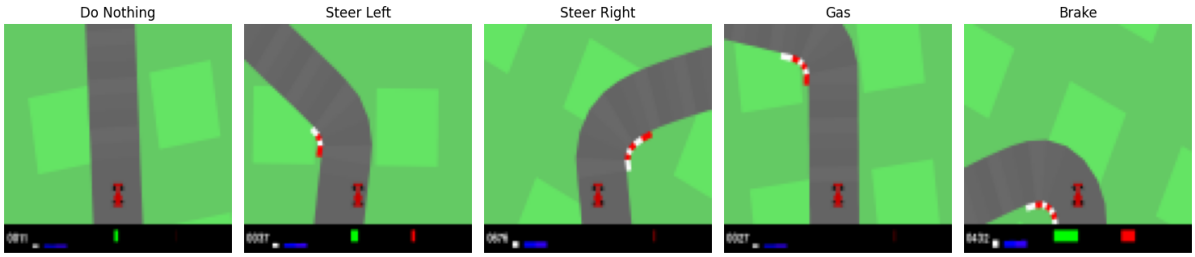

Figure 1: Plot of some images from the dataset

To address this task, I utilized two distinct convolutional neural network (CNN) models. For both models, I conducted a thorough exploration of various hyperparameters. The models were trained using the same dataset, but in two different scenarios: first, without applying any transformations to the dataset, and second, with the application of transformations. This approach was aimed at analyzing how the use of transformations impacts the performance of the models.

2 Dataset

Going into more detail, the dataset used in this project was provided by the professor [link] and consists of 9118 RGB images with dimensions of 96x96 pixels. The dataset is divided into 5733 images for training, 636 images for validation and 2749 images for testing.

By examining the dataset images in the various folders, I noticed that in folder 2, corresponding to the action `Steer Right`, all the images depicted curves where the car was required to turn left. Similarly, in the folder corresponding to the action `Steer Left`, the images depicted curves where the car was required to turn right. For this reason, I decided to invert the images in these folders.

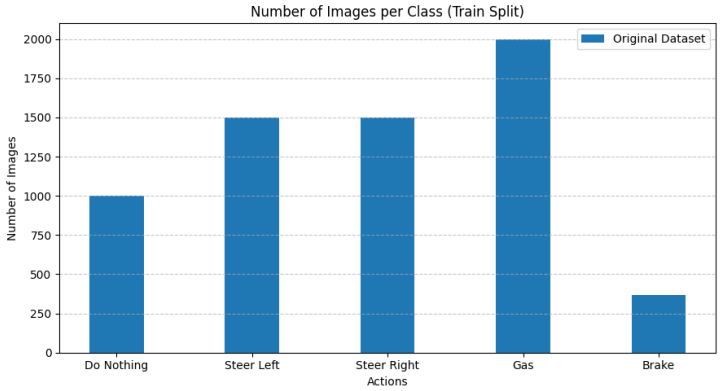

Figure 2: Class distribution of Training dataset

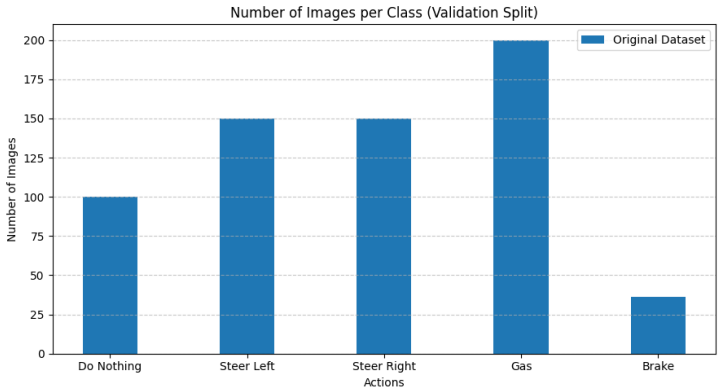

Figure 3: Class distribution of Validation dataset

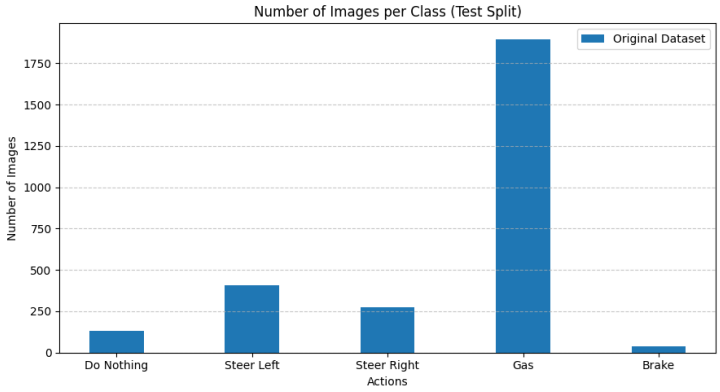

Figure 4: Class distribution of Test dataset

To explore the dataset I also plot the images [2, 4] and as we can see, the dataset used for both training validation and testing is highly imbalanced. In particular, the ”brake” class contains very few images, which will make it more challenging for the models to generalize effectively. Furthermore, the test dataset also exhibits significant imbalance, especially for the ”GAS” action. This implies that, to evaluate the models’ performance, it will not be sufficient to rely solely on accuracy; instead, it will be necessary to employ

multiple evaluation metrics.

Another issue is that the overall number of training images is not sufficiently large for the complexity of the task. As I will explain later, this leads to overfitting, causing the models to struggle with generalization.

3 Models

The models used to complete this task are two (although the code includes three models, as I will explain in the conclusion, the third is a specific case of the first).

3.1 1 CNN

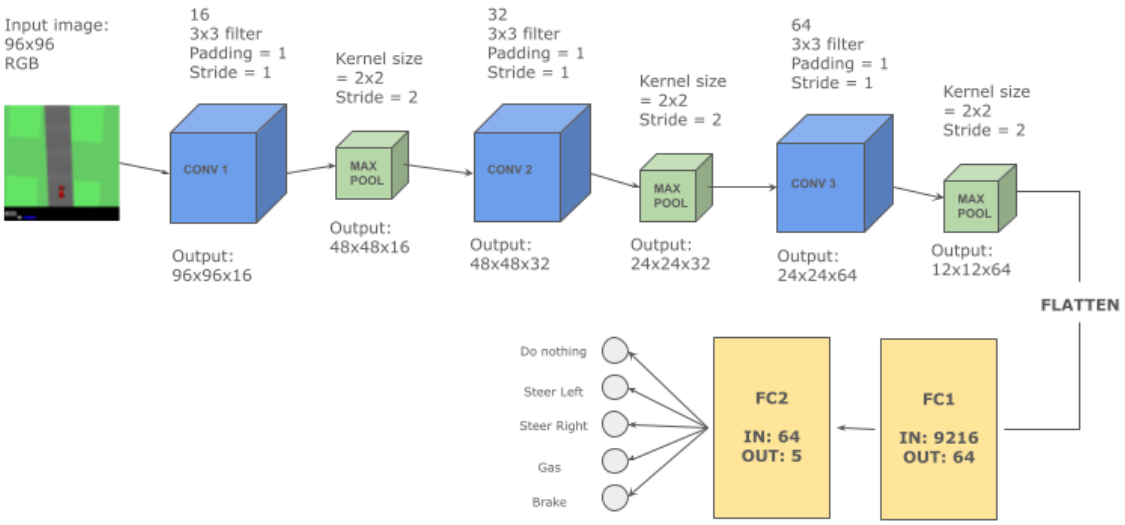

Figure 5: First Model

For the first model, I chose to use a small and simple neural network architecture. As shown in the corresponding image [5] The network consists of three convolutional layers with ReLU activation, followed by max-pooling layers that progressively reduce the spatial dimensions. The first convolutional layer has 16 filters, the second has 32, and the third has 64, all with a kernel size of 3×3 3×3, padding of 1, and a pooling kernel

size of 2x2 with a stride of 2. After the convolutional blocks, the output is flattened into a vector of size 9216 and passed through two fully connected layers: the first with 64 neurons and the second with 5 neurons corresponding to the output classes.

As I said before The architecture of the first model is simpler in fact consists of a total of 613,797 trainable parameters, with no non-trainable parameters. Most of the parameters are concentrated in the fully connected layers, particularly the first dense layer, which alone contributes 589,888 parameters. The convolutional layers, on the other hand, account for a smaller proportion of the parameters, with 448 parameters in the first layer, 4,640 in the second, and 18,496 in the third.
Despite its small size, as we will observe from the results, this model demonstrated the best ability to generalize with the provided dataset and achieved the highest performance during training.

3.2 2 CNN

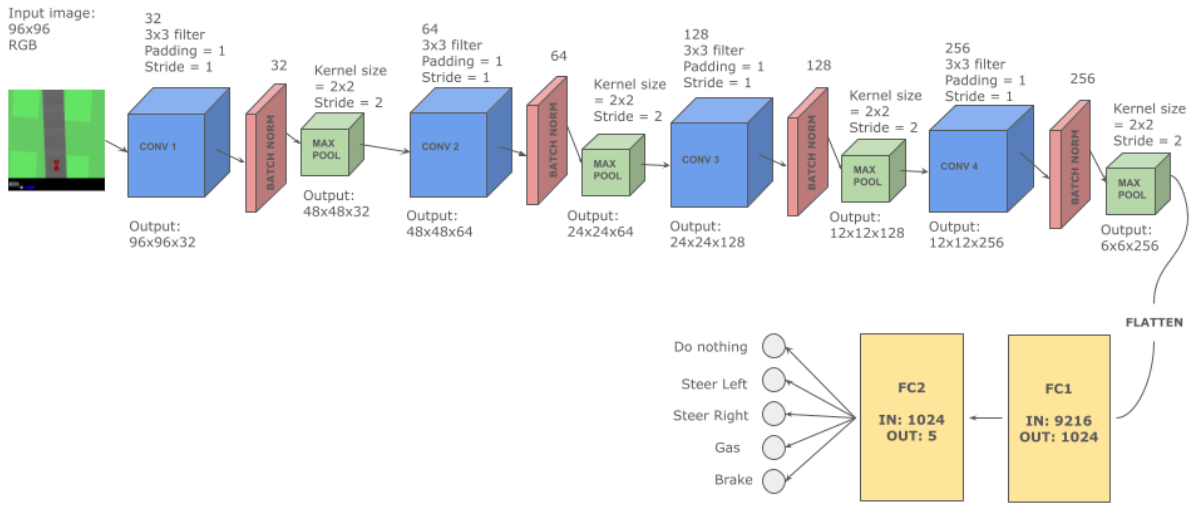

Figure 6: Second Model

For the second model, I decided to use a deeper CNN, and going into further detail as we can see from the image [6] the network consists of four convolutional layers with ReLU activation, each followed by batch normalization to stabilize learning and improve

convergence. Each convolutional layer is followed by a max-pooling layer with a kernel size of 2x2 and a stride of 2, which progressively reduces the spatial dimensions of the feature maps.
The final feature map output is flattened into a vector of size 9216, which is then passed through two fully connected layers. The first fully connected layer has 1024 neurons with ReLU activation, while the second outputs logits for the five output classes.
The deeper structure also results in a significantly higher number of parameters, with a total of 9,832,709 trainable parameters compared to the simpler model. This includes 896, 18,496, 73,856, and 295,168 parameters in the convolutional layers, along with additional parameters in the batch normalization layers. The fully connected layers contribute the majority of the parameters, with 9,438,208 in the first dense layer and 5,125 in the output layer. In conclusion, I would like to add that for the train of all models, I used `nn.CrossEntropyLoss()` and additionally I also included `torch.nn.utils.clip_grad_norm_()`, which prevents gradients from becoming excessively large during backpropagation.

4 First Strategy

As the first approach to addressing this task, I decided to train the model without applying any data augmentation to the images. Instead, I performed a grid search to identify the optimal hyperparameters, optimizer, and batch size. The specific values considered in the grid search were as follows:

- `learning_rate`: 0.01, 0.001, 0.0001
- `batch_size`: 32, 64
- `optimizer`: Adam, SGD

I would also like to specify that each combination of hyperparameters from the grid search was used to train the model for 20 epochs. At the end of each training session, the model was evaluated using the validation dataset to identify the best-performing model. Once the best model was selected, it was tested on the test dataset to compute the classification report and confusion matrix, and to evaluate its performance on unseen data.

4.1 Results

Due to the imbalanced nature of the dataset, relying solely on accuracy as the evaluation metric is not sufficient. For this reason, I utilized multiple metrics to assess the performance of the models, including Accuracy, Precision, Recall, F1-score, and the confusion matrix.

Batch Size	Learning Rate	Optimizer	Accuracy	Precision	Recall	F1 Score
32	0.01	Adam	0.6289	0.6199	0.6289	0.6187
32	0.01	SGD	0.5519	0.5238	0.5519	0.5188
32	0.001	Adam	0.6792	0.6648	0.6792	0.6687
32	0.001	SGD	0.3145	0.0989	0.3145	0.1505
32	0.0001	Adam	0.6179	0.6132	0.6179	0.5991
32	0.0001	SGD	0.3145	0.0989	0.3145	0.1505
64	0.01	Adam	0.6368	0.6243	0.6368	0.6288
64	0.01	SGD	0.5739	0.5358	0.5739	0.5236
64	0.001	Adam	0.6824	0.6867	0.6824	0.6702
64	0.001	SGD	0.3145	0.0989	0.3145	0.1505
64	0.0001	Adam	0.6289	0.6291	0.6289	0.6056
64	0.0001	SGD	0.2358	0.0556	0.2358	0.0900

Table 1: Results of different configurations for 1 CNN. The model with the best performance, based on Precision and Accuracy on Validation set, is highlighted in green.

Additionally, I clarify that the values reported in the tables for Precision, Recall, and F1-score represent the averages of these metrics computed across all classes. Meanwhile, the graphs showing the training loss, training accuracy, and the confusion matrix correspond to the model that achieved the best overall performance.

As we can see from the results [7 8], the model (related to 1 CNN) that achieved the best performance is the one using `learning_rate = 0.001`, `batch_size = 64`, and `optimizer = Adam`. The model struggled to generalize effectively, as observed in the confusion matrix [7]. This is likely due to the model overfitting during training, which limited its ability to accurately classify images, particularly for the BRAKE and DO NOTHING labels. These results are further confirmed by the classification report [9].

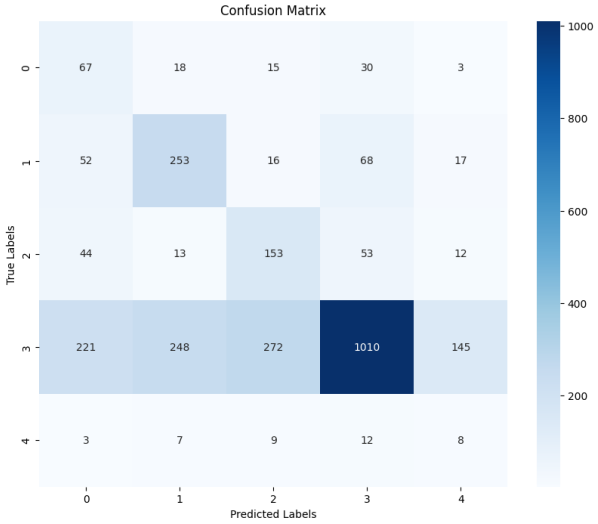

Figure 7: Confusion matrix of best model 1 CNN

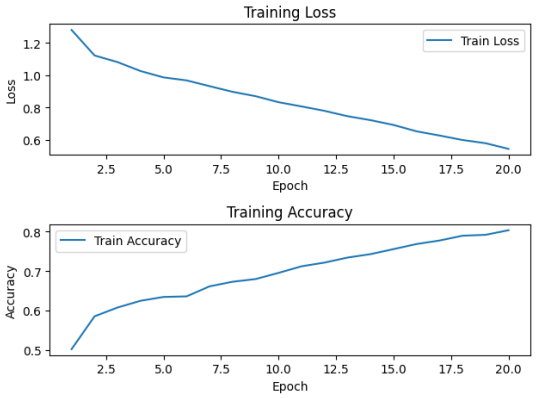

Figure 8: Plot of train loss and train accuracy of best model 1 CNN

```
Test Accuracy: 0.5424
Classification Report:
              precision    recall  f1-score   support

           0       0.17      0.50      0.26       133
           1       0.47      0.62      0.54       406
           2       0.33      0.56      0.41       275
           3       0.86      0.53      0.66      1896
           4       0.04      0.21      0.07        39

    accuracy                           0.54      2749
   macro avg       0.38      0.48      0.39      2749
weighted avg       0.71      0.54      0.59      2749
```

Figure 9: Classification report of best model 1 CNN

Batch Size	**Learning Rate**	**Optimizer**	**Accuracy**	**Precision**	**Recall**	**F1 Score**
32	0.01	Adam	0.6588	0.6520	0.6588	0.6410
32	0.01	SGD	0.6588	0.6537	0.6588	0.6423
32	0.001	Adam	0.6541	0.6531	0.6541	0.6481
32	0.001	SGD	0.5975	0.5607	0.5975	0.5613
32	0.0001	Adam	0.6824	0.6773	0.6824	0.6695
32	0.0001	SGD	0.5283	0.4312	0.5283	0.4578
64	0.01	Adam	0.6525	0.6462	0.6525	0.6425
64	0.01	SGD	0.6447	0.6448	0.6447	0.6326
64	0.001	Adam	0.6698	0.6573	0.6698	0.6581
64	0.001	SGD	0.5928	0.5536	0.5928	0.5485
64	0.0001	Adam	0.6541	0.6812	0.6541	0.6510
64	0.0001	SGD	0.4984	0.4171	0.4984	0.4288

Table 2: Results of different configurations for 2 CNN. The model with the best performance, based on Precision and Accuracy on the Validation set, is highlighted in green.

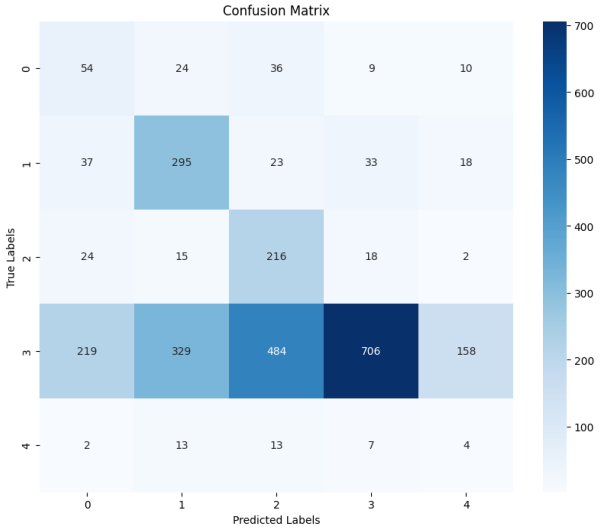

Figure 10: Confusion matrix of best model 2 CNN

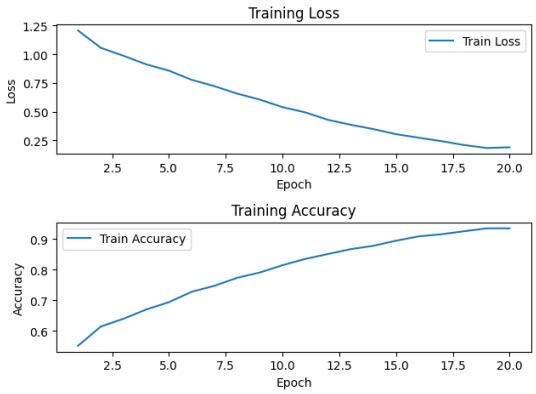

Figure 11: Plot of train loss and train accuracy of best model 2 CNN

```
Test Accuracy: 0.4638
Classification Report:
              precision    recall  f1-score   support

           0       0.16      0.41      0.23       133
           1       0.44      0.73      0.55       406
           2       0.28      0.79      0.41       275
           3       0.91      0.37      0.53      1896
           4       0.02      0.10      0.03        39

    accuracy                           0.46      2749
   macro avg       0.36      0.48      0.35      2749
weighted avg       0.73      0.46      0.50      2749
```

Figure 12: Classification report of best model 2 CNN

As shown in the results [10, 11], the best-performing model (related to 2 CNN) is the one with the hyperparameters `learning_rate = 0.0001`, `batch_size = 32`, and `optimizer = Adam`. Despite being a deeper model with a larger number of parameters, it performed worse than the previous model. This can be attributed to the limited number of training images, which caused the larger model to overfit. This overfitting is evident in the confusion matrix [10], where the model frequently misclassifies images corresponding to the GAS action, confusing them with other actions. This highlights the model's inability to generalize effectively.

5 Second Strategy

As a second approach to solving this task, and based on the results obtained previously, I decided to keep the Grid Search configuration unchanged. This means that, once again, I performed a search to determine the best values for `learning_rate`, `optimizer`, and `batch_size`, training each model for 20 epochs. The main difference in this approach, however, is the use of `transforms` from the `torchvision` library, which allows for the application of transformations to the tensors used during training in an attempt to reduce overfitting.
Specifically, the transformations I chose to apply are as follows:

- `transforms.RandomRotation(degrees=(-15, -15), interpolation=Image.BILINEAR)`
- `transforms.RandomRotation(degrees=(15, 15), interpolation=Image.BILINEAR)`
- `transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))`

I intentionally avoided transformations such as Horizontal Flip or large rotation values because the dataset contains classes like STEER LEFT and STEER RIGHT. For example, flipping an image corresponding to the STEER LEFT action would result in an image representing the STEER RIGHT action but in the wrong folder, thereby introducing significant noise into the dataset. The same issue applies to large rotation values. Lastly, I decided against using transformations like Color Jitter or Gaussian Blur because, even when the model is deployed to control the car, the input images it will receive are unlikely to have issues with lighting, blurring, or similar distortions. Also in this case in order to evaluate the performance of model metrics like: Accuracy, Precision, Recall and F1-score

5.1 Results

As shown in the results [13, 14, 15], the use of transformations improved the overall performance of the model. However, similar to the first case, due to the imbalance in the training dataset, the model predominantly predicts images associated with the GAS action. Nevertheless, as seen in the confusion matrix [13], there has been a general improvement in the prediction of all classes.

Batch Size	Learning Rate	Optimizer	Accuracy	Precision	Recall	F1 Score
32	0.01	Adam	0.2170	0.1274	0.2170	0.1580
32	0.01	SGD	0.5173	0.5893	0.5173	0.4932
32	0.001	Adam	0.3160	0.3271	0.3160	0.1786
32	0.001	SGD	0.5487	0.4349	0.5487	0.4838
32	0.0001	Adam	0.2390	0.1124	0.2390	0.0962
32	0.0001	SGD	0.2123	0.0911	0.2123	0.0977
64	0.01	Adam	0.0629	0.0790	0.0629	0.0408
64	0.01	SGD	0.5613	0.5518	0.5613	0.5225
64	0.001	Adam	0.0881	0.3685	0.0881	0.0743
64	0.001	SGD	0.3145	0.0989	0.3145	0.1505
64	0.0001	Adam	0.3145	0.0989	0.3145	0.1505
64	0.0001	SGD	0.3145	0.0989	0.3145	0.1505

Table 3: Results of different configurations for 1 CNN. The model with the best performance, based on Precision and Accuracy on the Validation set, is highlighted in green.

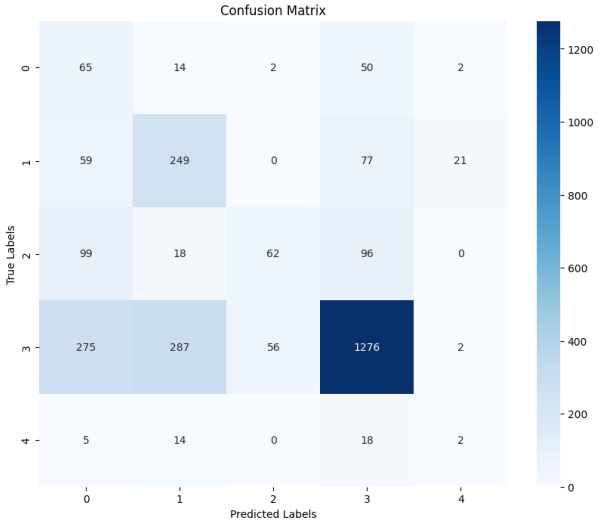

Figure 13: Confusion matrix of best model 1 CNN

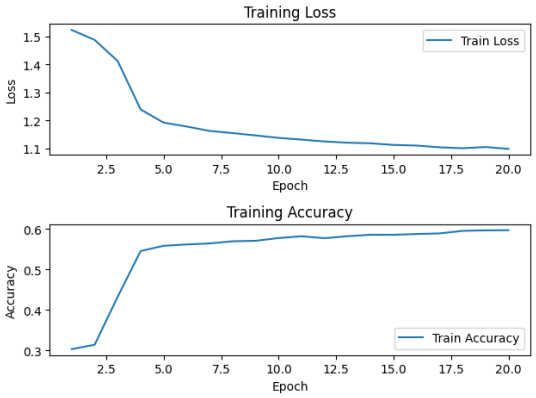

Figure 14: Plot of train loss and train accuracy of best model 1 CNN

```
Test Accuracy: 0.6017
Classification Report:
              precision    recall  f1-score   support

           0       0.13      0.49      0.20       133
           1       0.43      0.61      0.50       406
           2       0.52      0.23      0.31       275
           3       0.84      0.67      0.75      1896
           4       0.07      0.05      0.06        39

    accuracy                           0.60      2749
   macro avg       0.40      0.41      0.37      2749
weighted avg       0.70      0.60      0.63      2749
```

Figure 15: Classification report of best model 1 CNN

From the classification report [15], we observe that the model predicts very few images related to the BRAKE action. This indicates that, despite the improvements, the model still struggles to fully accomplish the task and generalize certain actions effectively.

Batch Size	Learning Rate	Optimizer	Accuracy	Precision	Recall	F1 Score
32	0.01	Adam	0.2358	0.0556	0.2358	0.0900
32	0.01	SGD	0.4135	0.4039	0.4135	0.2900
32	0.001	Adam	0.2563	0.4221	0.2563	0.1302
32	0.001	SGD	0.5173	0.5057	0.5173	0.5029
32	0.0001	Adam	0.2704	0.2504	0.2704	0.1520
32	0.0001	SGD	0.4890	0.4432	0.4890	0.4176
64	0.01	Adam	0.2846	0.3628	0.2846	0.1910
64	0.01	SGD	0.4088	0.2587	0.4088	0.2952
64	0.001	Adam	0.2579	0.3895	0.2579	0.1523
64	0.001	SGD	0.2925	0.2453	0.2925	0.1837
64	0.0001	Adam	0.3443	0.2755	0.3443	0.2062
64	0.0001	SGD	0.2469	0.5119	0.2469	0.1174

Table 4: Results of different configurations for 2 CNN. The model with the best performance, based on Precision and Accuracy on the Validation set, is highlighted in green.

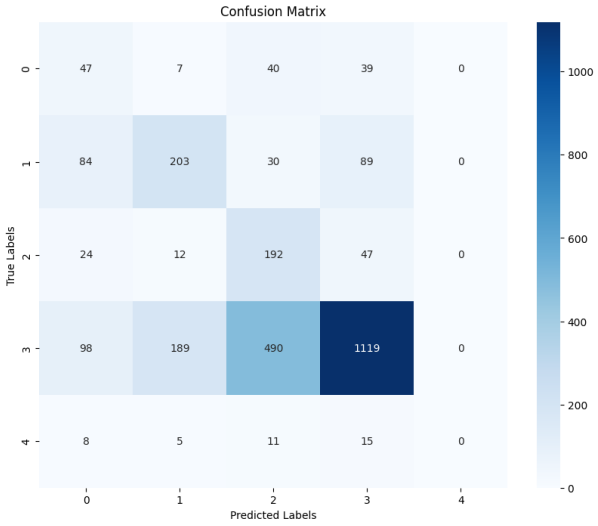

Figure 16: Confusion matrix of best model 2 CNN

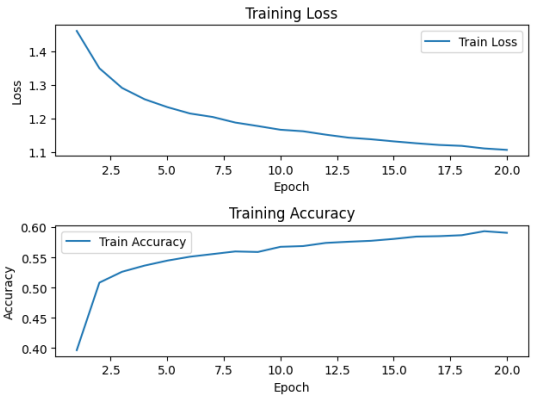

Figure 17: Plot of train loss and train accuracy of best model 2 CNN

Even in this case, the larger model achieved better performance [16, 17, 18] compared to the first strategy, thanks to the transformations applied to the training dataset. However, its performance remains inferior to the simpler model, indicating that the larger model still tends to overfit during training. As seen in the classification report [18], the

Test Accuracy: 0.5678
Classification Report:

	precision	recall	f1-score	support
0	0.18	0.35	0.24	133
1	0.49	0.50	0.49	406
2	0.25	0.70	0.37	275
3	0.85	0.59	0.70	1896
4	0.00	0.00	0.00	39
accuracy			0.57	2749
macro avg	0.35	0.43	0.36	2749
weighted avg	0.70	0.57	0.60	2749

Figure 18: Classification report of best model 2 CNN

larger model is also unable to predict the BRAKE class. This limitation is once again attributed to the imbalance present in the training dataset.

6 Conclusion

From the analysis conducted on the various trained models, it emerged that, contrary to expectations, the model with the fewest parameters achieved the best performance for this task. Additionally, applying transformations to the training dataset proved to be particularly beneficial. This behavior can be explained by the fact that models with a large number of parameters, while capable of better generalization, require datasets with a significantly larger number of images to converge effectively. Furthermore, the results indicated that the best-performing optimizer was SGD.

A key point to highlight is that the models, with this dataset, were not entirely able to solve the task. As shown in the confusion matrices, some images were misclassified. One way to improve performance is to use a balanced dataset. In particular, the BRAKE action was the least accurately predicted class. A potential future development could involve adding more images to the dataset to balance all classes. Another study that could be conducted involves assigning higher weights to the underrepresented classes to observe how this affects the model's performance. Additionally, exploring a broader range of hyperparameters could further improve the models' ability to address this task effectively.

As mentioned in the introduction, within the folder `daniele_sabatini_1890300.zip`, which contains the code used for dataset exploration and model training/testing, there

is a file named `HW2_WEIGHT_INITIALIZATION.ipynb`. This file includes an additional training process for a model with the same architecture as the previously presented `1_CNN` model. However, in this case, I investigated whether changing the weight initialization method would improve the model's performance. Specifically, I added the `_initialize_weights` function to the `CNNModel()` class. This function uses the line of code `nn.init.orthogonal_(layer.weight)`, which initializes the weights of the matrices such that their rows and columns are mutually orthogonal. The goal of this approach was to maintain the input scale during training, avoiding either amplification or reduction.

As seen in the results [19 20 21], this modification led to improved performance.

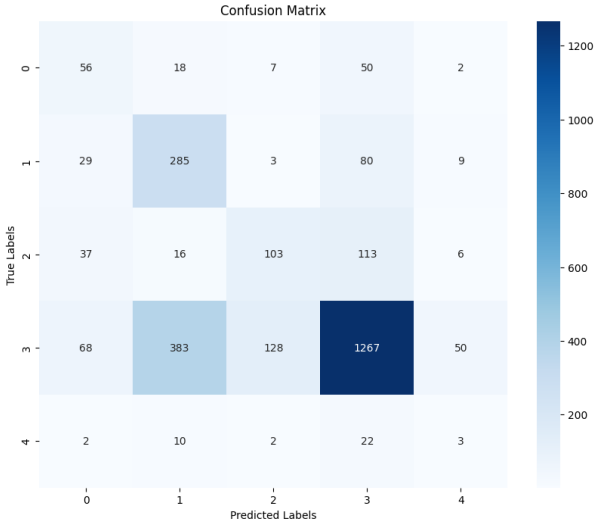

Figure 19: Confusion matrix of model with weights initialization

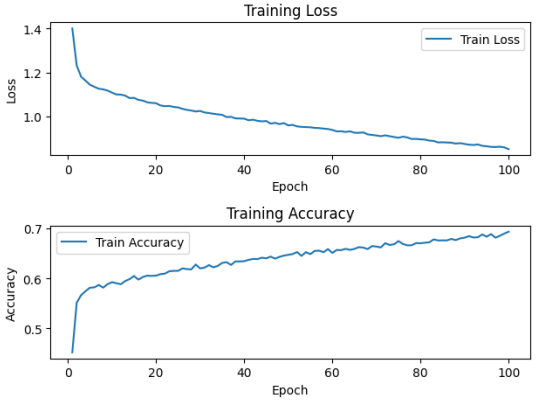

Figure 20: Plot of train loss and train accuracy of model with weights initialization

```
Test Accuracy: 0.6235
Classification Report:
              precision    recall  f1-score   support

           0       0.29      0.42      0.34       133
           1       0.40      0.70      0.51       406
           2       0.42      0.37      0.40       275
           3       0.83      0.67      0.74      1896
           4       0.04      0.08      0.06        39

    accuracy                           0.62      2749
   macro avg       0.40      0.45      0.41      2749
weighted avg       0.69      0.62      0.64      2749
```

Figure 21: Classification report of model with weights initialization

Finally, to develop a model that performed as well as possible during testing with the code provided by the professor for visualizing the car in the environment, I chose this last

model with the best hyperparameter values found in the previous grid search (is the same grid search described in the first and second strategies) and trained it for a longer period (100 epochs). At the end, I would like to add that all model training and testing were executed locally on my computer using the GPU. A video of the car being controlled by this model has been uploaded to Classroom, as specified in the homework instructions.

7 Remarks on Code Modifications

To create the video of the car being controlled by the model, as explained earlier, I used the code provided by the professor. However, to make it functional, I had to introduce some modifications. Specifically, since the model class did not include a `predict()` function, I added the following lines of code: `logits = model(obs_tensor)` and `action = torch.argmax(logits, dim=1).item()`.

Additionally, because the initial models were unable to complete a lap of the circuit, the program encountered issues when closing the rendering window. To address this, I set a maximum number of timestamps after which the window would close automatically (a problem that was resolved with the final model, which successfully completed the lap).

Finally, during initial testing, I observed that the car was reversing left and right directions. I suspect this was due to the inversion of the STEER LEFT and STEER RIGHT classes during preprocessing, as explained earlier, where the number of images for the STEER LEFT action were originally placed in the STEER RIGHT folder. To correct this, I added lines of code ensuring that when the model predicts STEER LEFT, the actual action taken is STEER RIGHT, and vice versa.