Sapienza
Università di Roma

Master's degree in Artificial Intelligence & Robotics

Car Racing Image Classification

Machine Learning, Homework 2

Professor:
Luca Iocchi

Student:
Kristjan Tarantelli,
2009153

Academic Year 2024/2025

Contents

1 Introduction

1.1 Goal of the Assignment

The objective of this assignment is to classify images of a racing car in order to predict its next action based on the road conditions and car sensor data. The possible actions the model aims to determine are as follows:

0. Do Nothing
1. Steer Left
2. Steer Right
3. Gas
4. Brake

As a bonus task, the Gymnasium environment could be utilized to deploy the trained model and guide the car in a simulated setting, allowing for real-time testing and evaluation of its performance.

1.2 Data Preprocessing

The dataset consists of five classes, each containing a different number of images. The images are in RGB format with dimensions of 96x96 pixels. A distinctive feature of these images is a black bar at the bottom, which displays car sensor data, including acceleration and steering angle.

To preprocess the images, I experimented with two approaches: one where the black bar was removed and another where it was retained. Additionally, I resized the images to 32x32 pixels and converted them to grayscale for further testing. Interestingly, the different preprocessing techniques did not result in significant variations in performance.

To help the model focus more effectively on the road, I addressed the presence of green grass squares by replacing them with a uniform green color. This aimed to reduce distractions caused by irrelevant visual details.

Finally, I normalized the pixel values by dividing them by 255.0, ensuring all values were scaled to a range between 0 and 1.

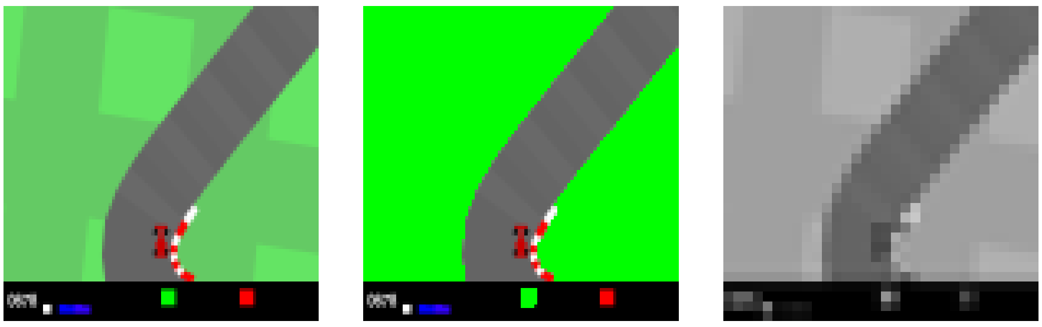
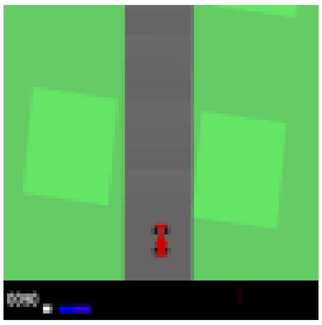
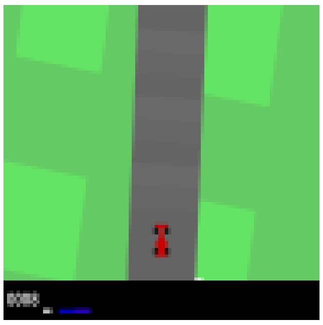

(a) Original Image (b) Preprocessed Image (c) Grayscale Image

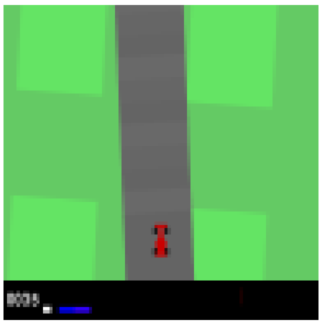
Figure 1: Different Preprocessing Methods

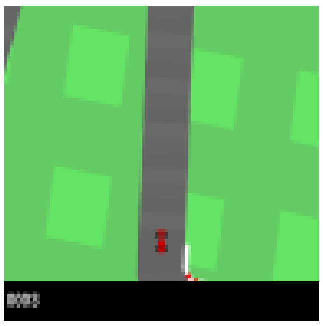
1.3 Noise and Ambiguity in the Dataset

One of the challenges encountered during the assignment was the noise and ambiguity in the dataset. For instance, there are lots of images where more than one action is possible, making it difficult to determine the correct label. Additionally, images with similar road conditions are often present in multiple classes, further complicating the classification task.


(a) Class 0: Do Nothing


(b) Class 1: Steer Left


(c) Class 2: Steer Right


(d) Class 3: Gas

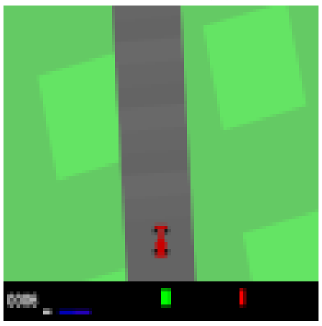

(e) Class 4: Brake

Figure 2: Labeled Images from the Training Dataset

1.4 Different Approaches

To classify the images, I experimented with a range of models, including Convolutional Neural Networks (CNNs), Support Vector Machines (SVMs), and models based on LeNet and VGG16 architectures.

The CNN models were designed from scratch to suit the specific requirements of the task. The LeNet model was adapted from its original architecture, as discussed in the lectures, and tailored for this dataset. The VGG16 model, pre-trained on the ImageNet dataset, was fine-tuned for this classification task by adding custom layers. Additionally, I used an SVM model on top of the features extracted from the VGG16 model to leverage its feature representation capabilities.

I conducted hyperparameter tuning for all models, except VGG16, to optimize performance. This included experimenting with different learning rates, optimizers, and numbers of epochs to identify the best configuration for each model. The VGG16 model was fine-tuned with its default pre-trained features, focusing primarily on adapting the added layers for this task.

2 Models

2.1 First CNN Model

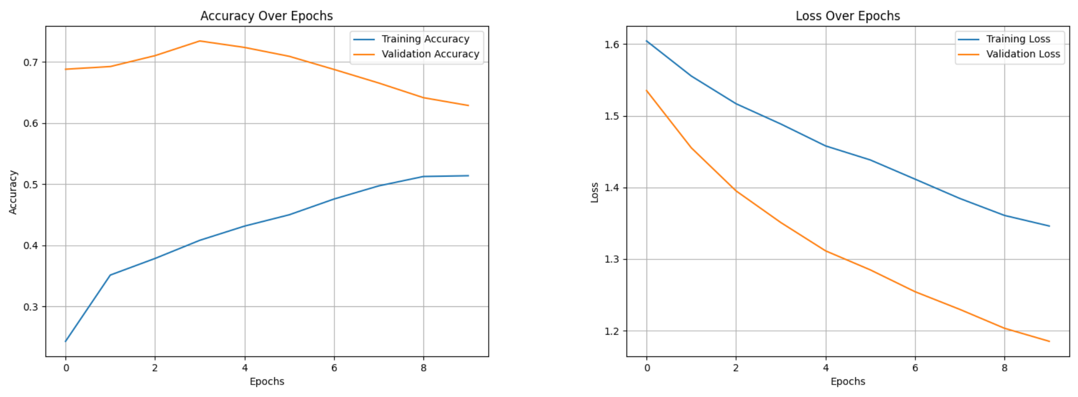
The first CNN model I designed consisted of three convolutional layers, each followed by a max-pooling layer. The output of the final pooling layer was flattened and passed through two fully connected layers, with a dropout layer in between to prevent overfitting. The model was compiled using the Adam optimizer and sparse categorical cross-entropy loss function, which does not require one-hot encoding of the labels.

As input, I used the green preprocessed images with the bottom cropped out, as shown in Table 1 in which the input is 82x96x3, instead of 96x96x3.

Layer Type	**Out Shape**	**Filters**	**Kernel Size**	**Activation**	**Param**
Conv2D	(82, 94, 32)	32	(3, 3)	ReLU	896
MaxPooling2D	(41, 47, 32)	-	(2, 2)	-	0
Conv2D	(39, 45, 64)	64	(3, 3)	ReLU	18,496
MaxPooling2D	(19, 22, 64)	-	(2, 2)	-	0
Conv2D	(17, 20, 128)	128	(3, 3)	ReLU	73,856
MaxPooling2D	(8, 10, 128)	-	(2, 2)	-	0
Flatten	(10240)	-	-	-	0
Dense	(128)	-	-	ReLU	1,310,848
Dropout	(128)	-	-	-	0
Dense	(5)	-	-	Softmax	645
Total Params	1,404,741 (5.36 MB)				

Table 1: First CNN Model Architecture

With this high number of parameters, I was not able to track the *F1 Score* during training, as it was too computationally expensive and the GPU went out of memory. However, I was able to track the accuracy and loss, which are shown in Figure 3.



(a) Accuracy (b) Loss

Figure 3: Training Metrics. Epochs: 10 / Batch Size: 32 / Learning Rate: 0.001

Class	Precision	Recall	F1-Score	#
Class 0	0.00	0.00	0.00	133
Class 1	0.34	0.50	0.40	275
Class 2	0.41	0.75	0.53	406
Class 3	0.81	0.68	0.74	1896
Class 4	0.00	0.00	0.00	39
Accuracy			0.63	2749
Macro Avg	0.31	0.38	0.33	2749
Wgt Avg	0.65	0.63	0.63	2749

Figure 4: Classification Metrics

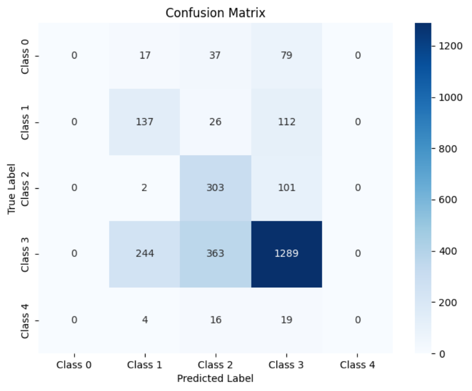


Figure 5: Confusion Matrix

As can be seen by the classification report in Table 11, the model performed well on classes 2 and 3, which are the most common classes in the dataset. However, it struggled with classes 0 and 4, which have fewer samples and are more challenging to classify. The model achieved an accuracy of 63%, which is a good starting point but can be improved further.

Also the confusion matrix in Figure 5 shows that the model is struggling to classify classes 0 and 4, which are never choose by the model. This is likely due to the imbalanced nature of the dataset, which contains fewer samples for these classes compared to classes 2 and 3.

I tryed some hyperparameter tuning on the learning rate (L) of the optimizer, on the number of epochs (E) and on the batch size (B), and a slight improvement was achieved. It can be seen that a learning rate of *1e-5* improves the model's performance, as shown in Figure 6, while *1e-7* is too low and the model does not learn, despite the other parameters changing.

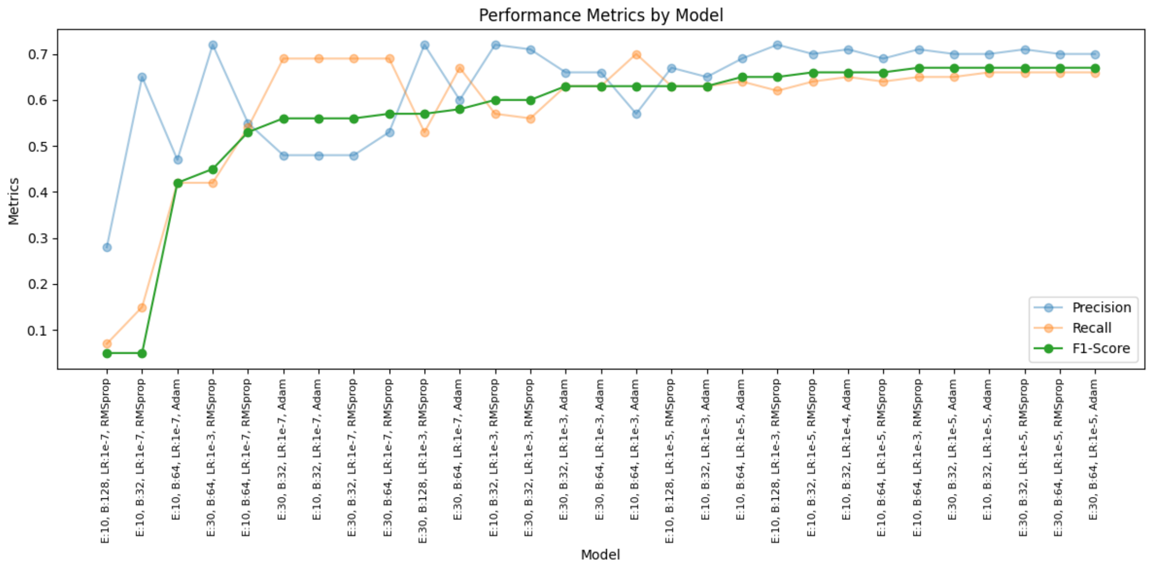


Figure 6: Hyperparameter Tuning

2.2 Second CNN Model

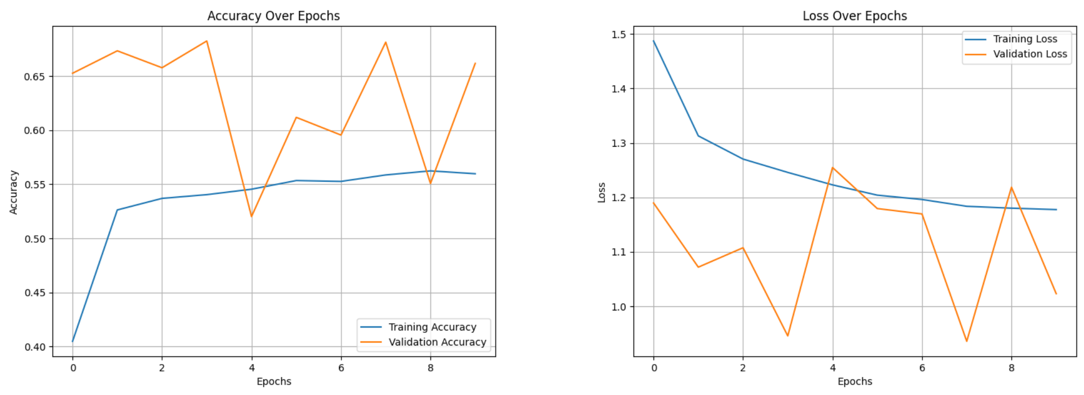
For the new model, I opted for a design with four small convolutional layers, each followed by a max pooling layer, and concluded with a global average pooling layer.

Additionally, I replaced the activation function with tanh and selected the Lion optimizer. As shown in the summary in Table 2, this architecture significantly reduces the number of parameters compared to the previous model. Note that this time, the input image is not cropped.

Layer Type	**Out Shape**	**Filters**	**Kernel Size**	**Activation**	**Param**
Conv2D	(94, 94, 16)	16	(3, 3)	Tanh	448
Conv2D	(92, 92, 16)	16	(3, 3)	Tanh	2320
MaxPooling2D	(46, 46, 16)	-	(2, 2)	-	0
Conv2D	(44, 44, 32)	32	(3, 3)	Tanh	4640
Conv2D	(42, 42, 32)	32	(3, 3)	Tanh	9248
GlobAvgPooling2D	(32)	-	-	-	0
Flatten	(32)	-	-	-	0
Dense	(64)	-	-	Tanh	2112
ActivityReg	(64)	-	-	-	0
Dense	(5)	-	-	Softmax	325
Total Params	19,093 (74.58 KB)				

Table 2: Second CNN Model Architecture

Even if the parameter are much less, the accuracy is still high. Although, in Figure 7 it can be seen that the model is not stable over the epochs on the validation set.



(a) Accuracy (b) Loss

Figure 7: Training Metrics. Epochs: 10 / Batch Size: 32 / Learning Rate: 0.001

Class	Precision	Recall	F1-Score	#
Class 0	0.14	0.02	0.03	133
Class 1	0.32	0.55	0.40	275
Class 2	0.47	0.73	0.57	406
Class 3	0.84	0.72	0.78	1896
Class 4	0.00	0.00	0.00	39
Accuracy			0.66	2749
Macro Avg	0.35	0.40	0.36	2749
Wgt Avg	0.69	0.66	0.66	2749

Figure 8: Classification Metrics

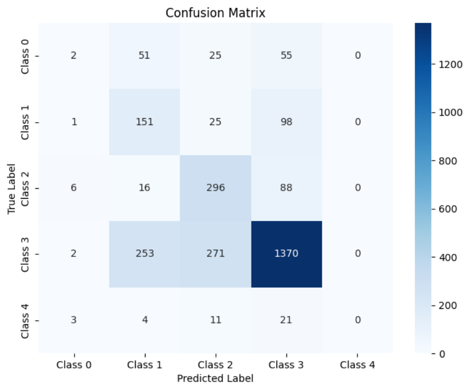


Figure 9: Confusion Matrix

As shown in the confusione matrix, in Figure 12, the model is still not able to classify classes 0 and 4. Differently than the other model, even after a hyperparameter search, the best model does not improve the performance about these classes.

The hyperparameter search shows that for this optimizer model, a learning rate of *1e-3* is preferred. As for the previous model, a learning rate of *1e-7* decreases the model's performance.

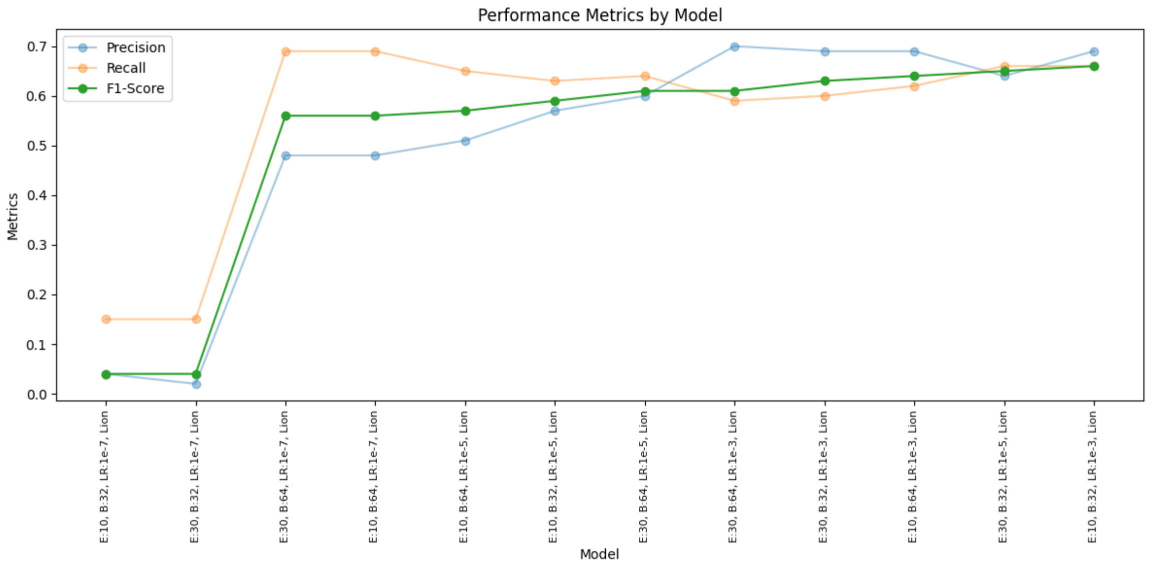


Figure 10: Hyperparameter Tuning

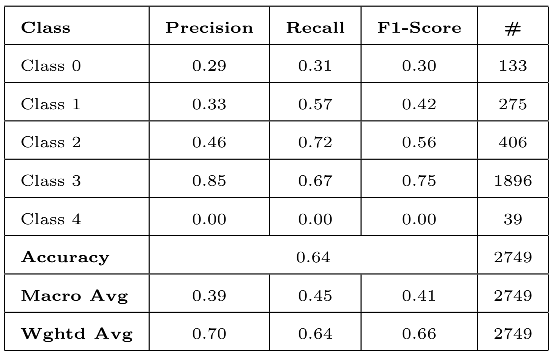
2.3 LeNet based Model

Interesting results were obtained by using the LeNet architecture, which was adapted to suit the requirements of the task. The model consists of three convolutional layers, with average pooling layer in between, and two fully connected layers. The model was compiled using the Adam optimizer and sparse categorical cross-entropy loss function, similar to the first CNN model.

The input image is resized to 32x32, greyscale and with the bottom cropped, like in Figure 1c. Despite the low resolution of the inputs and the low number of parameters, the model achieved high performance, similar to the first CNN model.

Layer Type	**Out Shape**	**Filters**	**Kernel Size**	**Activation**	**Param #**
Conv2D	(28, 32, 6)	6	(5, 5)	ReLU	156
AvgPooling2D	(14, 16, 6)	-	(2, 2)	-	0
Conv2D	(10, 12, 16)	16	(5, 5)	ReLU	2416
AvgPooling2D	(5, 6, 16)	-	(2, 2)	-	0
Conv2D	(1, 2, 120)	120	(5, 5)	ReLU	48120
Flatten	(240)	-	-	-	0
Dense	(84)	-	-	ReLU	20244
Dense	(5)	-	-	Softmax	425
Total Params	71,361 (278.75 KB)				

Table 3: LeNetModel Architecture



Class	**Precision**	**Recall**	**F1-Score**	**#**
Class 0	0.29	0.31	0.30	133
Class 1	0.33	0.57	0.42	275
Class 2	0.46	0.72	0.56	406
Class 3	0.85	0.67	0.75	1896
Class 4	0.00	0.00	0.00	39
Accuracy	0.64			2749
Macro Avg	0.39	0.45	0.41	2749
Wghtd Avg	0.70	0.64	0.66	2749

Figure 11: Classification Metrics

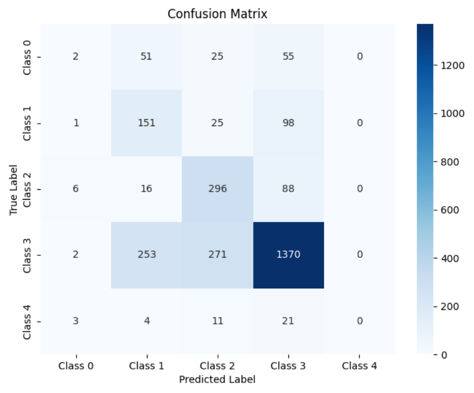


Figure 12: Confusion Matrix

2.4 VGG16 based Model

Just for fun, I tried some other approaches to see if the performance could be improved. I used the VGG16 model, pre-trained on the ImageNet dataset, and fine-tuned it for this classification task. The model was compiled using the Adam optimizer and sparse categorical cross-entropy loss function, similar to the first CNN model. The model architecture is shown in Table 4.

Despite the high number of parameters, the model did not outperform the first CNN model. The model was able to achieve an accuracy of 64% and an F1-score of 66%, similar to the other models. The model was able to classify classes 0 and 4.

Even if the trainable parameters were less than the first model, this one was a lot more time consuming to train, reaching an average of 8s 39ms/step with GPU. For this reason, I decided not to proceed with hyperparameter tuning for this model, even if it could have improved the performance a lot.

Layer Type	**Out Shape**	**Filters**	**Activation**	**Param**
VGG16 (Pretrained)	(3, 3, 512)	512	-	14,714,688
GlobalAvgPooling2D	(512)	-	-	0
Dense	(512)	512	ReLU	262,656
Dropout	(512)	-	-	0
Dense	(5)	5	Softmax	2,565
Total Params	14,979,909 (57.14 MB)			
Trainable Params	265,221 (1.01 MB)			
Non-trainable Params	14,714,688 (56.13 MB)			

Table 4: Model Architecture for ImageNet with VGG16

2.4.1 SVM based Model

To further explore classification performance, a Support Vector Machine (SVM) classifier was employed. Features were extracted from the dataset using a pretrained VGG16 model, truncated to remove the top classification layers.

The truncated model was used to extract high-level features through a global average pooling operation, significantly reducing the spatial dimensions while retaining key information. These features served as input to the SVM, which was trained to classify the images.

The training and testing datasets were preprocessed and fed through the VGG16 model to extract feature vectors. Various SVM kernels, including `rbf`, `sigmoid`, and `polynomial` (with varying degrees), were explored, alongside multiple values for the regularization parameter C.

Each combination of kernel, degree, and C was evaluated to determine its impact on classification accuracy. The hyperparameter search, in Figure 13 revealed that the polynomial kernel with degree 5 and C equal to 0.01 achieved the highest accuracy on the test set.

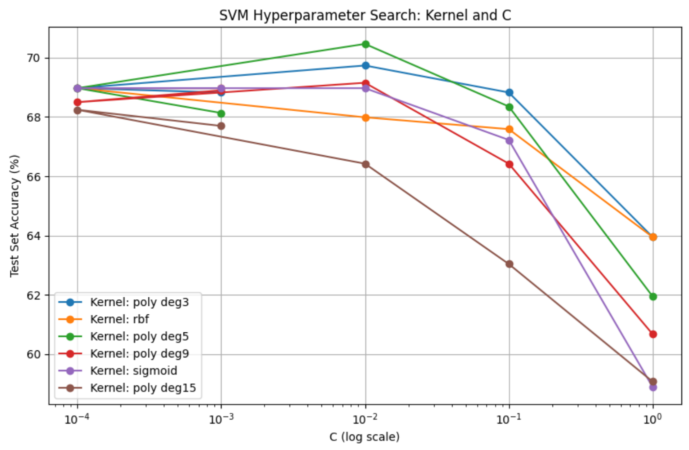


Figure 13: SVM Hyperparameter Search

3 Results and Discussion

3.1 Performance Comparison

3.1.1 CNN Models

The performance of the second model, with respect to the first one, was suboptimal due to several architectural and design choices. Firstly, the use of the `tanh` activation function in convolutional layers, instead of `ReLu`, led to vanishing gradients, limiting the model's ability to learn complex features effectively.

Additionally, the network had relatively few filters, which constrained its capacity to capture meaningful patterns, especially for complex data. With only 19k trainable parameters, with respect to 1MLN of the first one, the model lacked sufficient complexity to generalize well to the dataset.

The inclusion of a `Global Average Pooling` layer reduced the spatial dimensions, potentially discarding critical spatial information required for accurate classification. Probably, also the use of `Activity Regularization` for regularization was not as effective as more standard techniques such as `Dropout` or `Batch Normalization`.

Finally, the optimizer (`Lion`) used in this configuration may not have been well suited for the task without extensive tuning. In comparison, traditional optimizers like `Adam` or `RMSprop` may provide better stability and convergence.

3.1.2 LeNet based Model

The LeNet model, despite its simplicity and low number of parameters, performed surprisingly well on the dataset. The model achieved an accuracy of 64% and an F1-score of 66%, similar to the other models. The model was able to classify classes 0 and 4, which were challenging for the other models.

3.1.3 VGG16 and SVM based Model

The results, visualized in Figure 13, highlight the sensitivity of SVM performance to kernel type and regularization parameters. This experiment demonstrated that leveraging feature extraction from pretrained networks, combined with traditional classifiers like SVM, can yield competitive performance while avoiding the need for end-to-end training.

3.2 Drive Test

I evaluated the models by testing them in the simulated environment to observe how effectively they could drive the car and assess their real-world performance.

As mentioned earlier, one of the main challenges was that many models with high accuracy overfitted to the most frequent class in the dataset, Class 3 (Gas), by predominantly classifying all images under this label.

Conversely, some models failed to learn Class 4 (Brake), which is critical for navigating sharp curves effectively. This imbalance underscores the importance of evaluating models beyond accuracy to ensure balanced learning across all classes.

After testing all the models on the simulated track (excluding the SVM-based models), I observed the following:

- **First CNN Model:** This model demonstrated stability in maintaining a straight line along the road and had the highest accuracy. However, it struggled with sharp curves as it tended to go too fast, leading to poor handling in these sections. This behavior was consistent for both cropped and full images (including sensor data). Additionally, models using the RMSprop optimizer performed worse than those using the Adam optimizer. The best configuration for this model was **E:10, B:256, LR:1e-5, Adam**.

- **Second CNN Model:** Despite its lower accuracy, this was the only model that successfully completed the entire race, including sharp curves. Although it was less stable in maintaining a straight line and moved slower compared to the first model, this slower speed proved advantageous in handling curves. The best configuration for this model was **E:30, B:32, LR1e-3, Lion**.

- **LeNet Model:** Despite the lower resolution of its inputs (cropped images), this model achieved performance comparable to the first CNN model while requiring significantly less computational cost during training. Like the first CNN model, it struggled with sharp curves due to excessive speed. The best configuration for this model was **E:10, B:32, LR:1e-5**.

- **VGG16 Model:** The VGG16-based models were not extensively tested, but the model with the highest accuracy failed to complete the first curve, highlighting significant limitations in its ability to adapt to the track.