Machine Learning Homework 2

Filippo Tatafiore 1934988

2024-25

1 Introduction

The problem for this homework is image classification to drive a racing car (2D simulation).
The datasets therefore consist of images, each associated to one of the following labels, that correspond to the action that the car needs to take:

- 0: do nothing
- 1: steer left
- 2: steer right
- 3: gas
- 4: brake

Two datasets are used for this homework, a training dataset, used to train the model, and a separate testing dataset, only used to test the performance of the model.

As requested by the homework specifications, two different approaches were used. These differ in architectural characteristics, optimization strategies, regularization techniques and hyperparameters.

2 Approach 1

For the two approaches, the chosen models are both Convolutional Neural Networks (CNN's), which are composed of three main types of layers: Convolutional layer, Pooling layer and fully connected layers.

For this first approach, 2 Convolutional/Pooling layers are used (Convolutional layer - Pooling layer - Convolutional layer - Pooling layer). This number of layers determine the depth of the network. Deeper networks can capture more complex features but may also be harder to train and more prone to overfitting.

For the fully connected layers, a width of 128 neurons was chosen.

In the Convolutional layers, the kernel (filter) is used to perform the action of convolution.
It is a two-dimensional array of weights that moves across the image. When it is applied to an area of the image a dot product is calculated between the input pixels of the image and the filter, which is then fed into an output array. After that it shifts by a stride, repeating the process.
The size of the kernel chosen for this first approach is $3x3$.

In Convolutional layers another choice to make regards the zero-padding, this controls the spatial dimensions of the output feature maps by adding rows and columns of zeros around the input image before applying the convolution.
The type of padding used in this case is valid padding (no padding), where the last convolution is dropped if dimensions do not align.

After each Convolutional layer a Pooling layer is present. Similarly to the Convolutional layer, the Pooling layer sweeps a filter over the entire input, but this time the kernel applies an aggregation function populating the output array.

The type of pooling chosen is Max Pooling, where as the filter moves across the input, it selects the pixel with the maximum value to send to the output array.

Other than the previous architectural choices, other choiches were made regarding optimization strategies and regularization techniques.

Optimization strategies are techniques used to improve the performance of a model by efficiently finding the optimal set of parameters that minimize the loss function.
The one used for this model is Stochastic Gradient Descent (SGD), which updates weights based on the gradient of the loss function.

Regularization techniques are used to prevent overfitting. In this approach, the dropout technique is applied to the fully connected layers.
Dropout combats overfitting by introducing randomness into the training process, which forces the network to learn more robust features that are not reliant on specific neurons. During each training iteration, a random fraction of the neurons is deactivated.
In this case the dropout rate is 0.5, meaning 50% of the neurons are deactivated at each training step.

Both approaches perform Grid Search for the hyperparameter tuning, testing each possible combination of the given hyperparameter values.
The hyperparameters for the first approach are the number of epochs, the number of fully connected layers in the CNN and the learning rate for the SGD optimizer.

Each epoch is a full pass through the entire training dataset, the model makes predictions on the training data, computes the loss, and updates its weights using the optimization algorithm.
If the number of epochs is too small, the model does not have enough opportunities to learn the underlying patterns of the data, resulting in underfitting and poor performance. On the other hand, if the number of epochs is too large the model overfits on the training data, not learning the generalizable patterns and resulting once again in bad performance on new data.
This is why it important to tune the number of epochs.

The number of fully connected layers (depth) is also a value that needs to be tuned. Having too few fully connected layers may cause the model to struggle to capture complex relationships between features, leading to underfitting, and having too many of them may cause overfitting.

The learngin rate is important to tune since it determines the size of the steps taken during the SGD optimization. It is used to scale the gradient of the loss function with respect to the model's weights, controlling how much the weights are adjusted in each update.
A larger learning rate can lead to faster convergence, but risks overshooting the optimal solution and may lead to divergence or oscillations. A smaller learning rate can lead to more precise convergence, but may slow down considerably the training and get stuck in local minimum of the loss function.

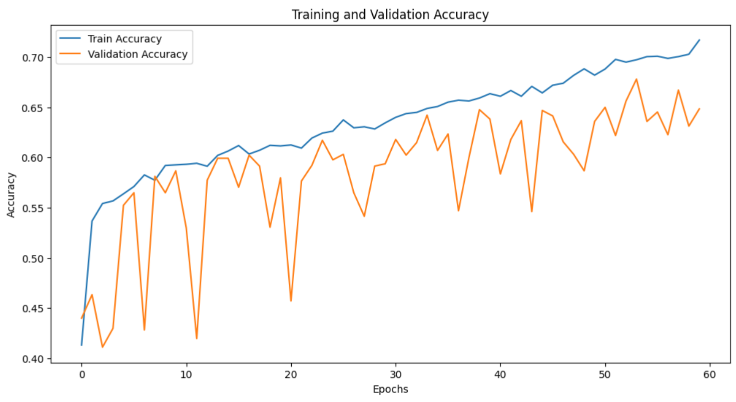
The code starts with collecting the two datasets for training and testing.
The training dataset is splitted into the training set, train_ds, and the validation set, val_ds, which is used to evaluate the model during the training.

The datasets are then normalized, this is important for faster convergence, stability and better generalization of the model. The normalization is done dividing by 255.0, since the pixel values are represented as integers in the range $[0, 255]$.

After this step follows the definition of the function used to build the model. The hyperparameter search is performed testing the accuracy of the models, built with each combination of hyperparameters, on the validation set.

The hyperparameter tuning showed a generally better performance with a shallow fully connected network and a learning rate of 0.01 instead of smaller ones. Finally, better results were obtained with a higher number of epochs, since it gives the model more chanches to learn the patterns in the data.

Having obtained the hyperparameter values that performed best, the final model is built and trained using these.
The graph here below shows the evolution of the accuracy, calculated on the training set and validation set during the training, gradually increasing.

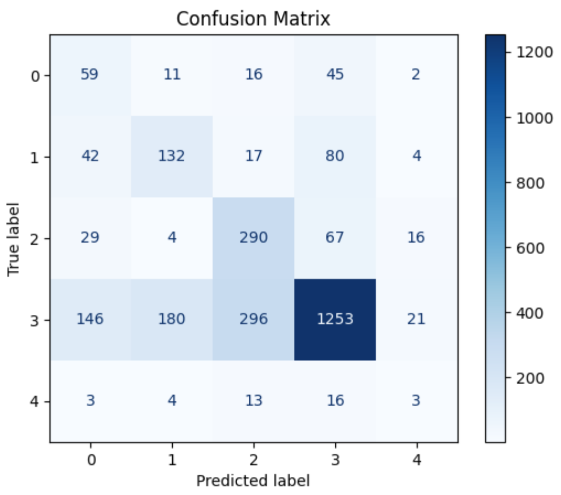


At the end of the training, the model can predict the label to assign to each image on the testing dataset.
The metrics used to evaluate the model for this problem are: Accuracy, F1-score, Precision and Recall.
Accuracy measures the proportion of correctly predicted instances out of all the predicted instances. It is defined as the ratio between the number of correct predictions and the total number of predictions.
The Precision is the ratio between the number of correctly predicted positive instances and the number of predicted positives. Since there are 5 classes in this case, the Precision is calculated for each class, treating that class as the positive class and the other classes as negative.
For this problem the metric represents the weighted average Precision across all classes.
The Recall performance metric measures the ability of a model to correctly identify all relevant instances of a class. It is defined as the number of instances correctly predicted as positive over the total number of positives.
The metric calculated is the weighted average Recall across all classes.
The F1-score is used in classification tasks to balance precision and recall. It is the harmonic mean of precision and recall, providing a single score that reflects both the model's ability to correctly identify positive instances and its ability to find all relevant instances.

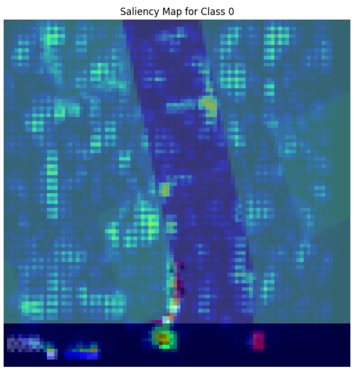
As for the other metrics, the weighted average across all classes is calculated.

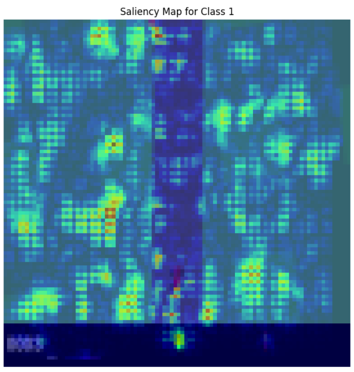
The evaluation of the model shows this values:
Accuracy: 0.6319
F1-score: 0.6558
Precision: 0.7103
Recall: 0.6319

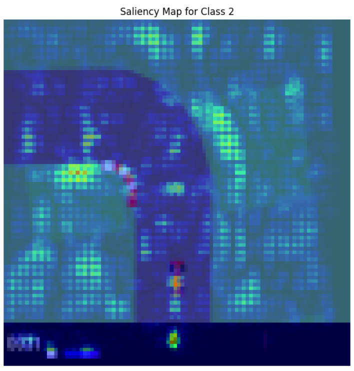
Up next the Confusion Matrix is shown. It provides a breakdown of the model's predictions compared to the actual labels. The elements on the diagonal of the Confusion Matrix represent the correct predictions for each class, while the other represent misclassification.
Ideally, the diagonal elements (true positives) should have the highest values in their respective rows and columns, indicating that the model is correctly predicting most samples for each class. In our case diagonal elements are generally higher, but not always. The training dataset is in fact unbalanced, having a lot more class 3 images than others, so we can see higher values corresponding to this class.

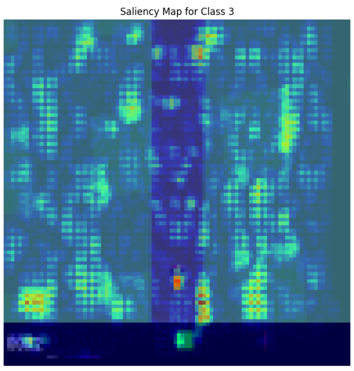


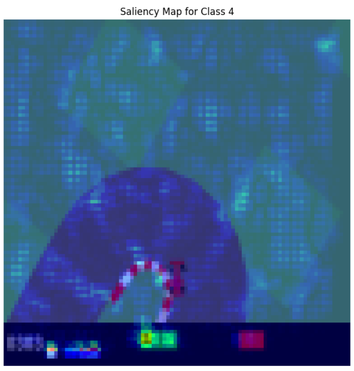
After the evaluation, I added some code to generate and visualize saliency maps for a set of test images.
Saliency maps are useful for highlighting the regions of the image that the model focuses on the most to recognise patterns and predict the output.
The following images show the original image with the saliency map overlaid on top. Bright regions indicate areas of the image that had the most influence on the model's decision.
I reported 5 different images, one for each class of the problem.











As we can expect, the model focuses predominantly on the edge of the track and on the green area off-road, as well as the current position of the car and on the car sensors at the bottom of the image.

In the file "model1_simulation" is present the video of the learned model driving the car.

3 Approach 2

For the second approach, different choices were made.
This time 4 Convolutional/Pooling layers were used. More layers allow the model to learn more

complex and abstract features, but result in a computationally more expensive model to train.

The width of the fully connected layers is 64 neurons. Being this narrower than the previous approach, it helps to speed up training reducing the number of parameters.

A larger filter was used, $5x5$, which captures broader spatial features in the input image rather than a $3x3$ filter. However, a bigger filter requires more parameters and computation.

The type of padding used for this approach is same padding, which ensures that the output layer has the same size as the input layer. This preserves spatial information maintaining resolution.

This time, the chosen type of pooling is Average Pooling, where, as the filter moves across the input, it calculates the average value within the receptive field to send to the output array (instead of selecting the pixel with the maximum value as done in Max Pooling).

Another different choice regards the optimizer. The second approach uses Adam, an adaptive optimizer whose key features include an adaptive learning rate, adjusted dynamically during the course of the training for each parameter individually.

The regularization techniques applied in this case were L2 regularization and data augmentation. L2 regularization works by adding a penalty term to the loss function based on the magnitude of the model's weights. This penalty discourages the model from learning large weights, which can lead to overfitting.
Data augmentation applies random transformations on the training data, like rotation, flipping and zooming the images. This exposes the model to more variations of the data, causing it to learn to generalize better and become less sensitive to specific patterns in the training set.

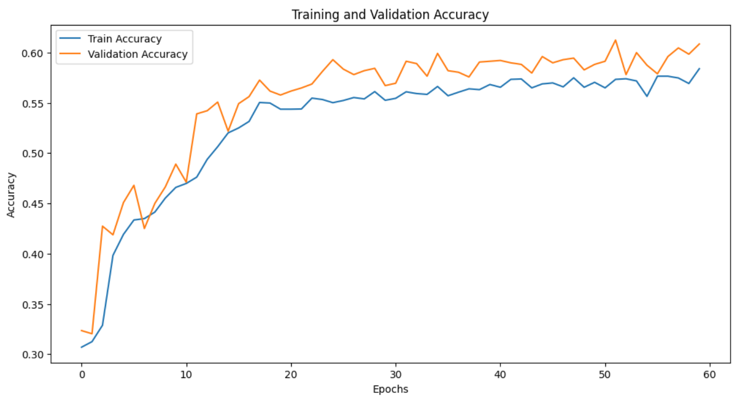
The Grid Search performs the tuning for the following hyperparameters: number of epochs, number of fully connected layers and regularization strength.
This time, since the Adam optimizer has an adaptive learning rate, this wasn't included in the hyperparameters. Instead of the learning rate, I added the regularization strength, which controls the impact of the L2 regularization term of on the model's loss function.

As before, after collecting the datasets and performing normalization, follows the function used to build the model and the hyperparameter search.

Better results were obtained for higher number of epochs and less fully connected layers, like before. Also a smaller regularization strength resulted in better performance.

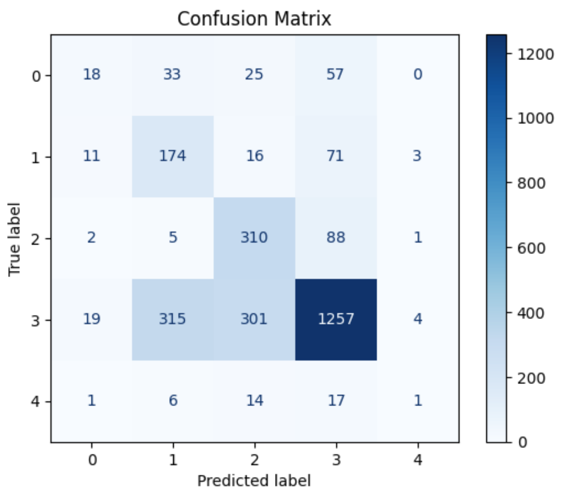
As mentioned previously, the dataset is unbalanced. For this reason, the best hyperparameters where selected giving more importance to the F1-score metric, instead of only accuracy. This ensures that the model performs well also on the classes that are less frequent.

The following image shows the accuracy calculated on the training set and validation set during the training, done with the selected best hyperparameters.



As before, to evaluate the model I used Accuracy, F1-score, Precision and Recall:
Accuracy: 0.6402
F1-score: 0.6506
Precision: 0.7019
Recall: 0.6402

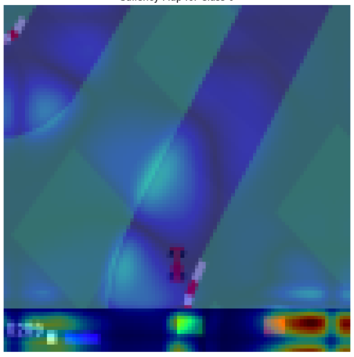
Below, the Confusion Matrix for this model:



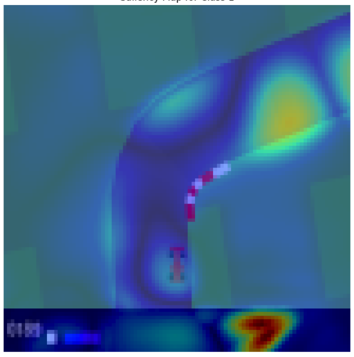
True label \ Predicted label	0	1	2	3	4
0	18	33	25	57	0
1	11	174	16	71	3
2	2	5	310	88	1
3	19	315	301	1257	4
4	1	6	14	17	1

Finally, I generated the saliency maps for 5 images of 5 different classes, overlaid on top of the original images, to show the areas that influence the model the most.

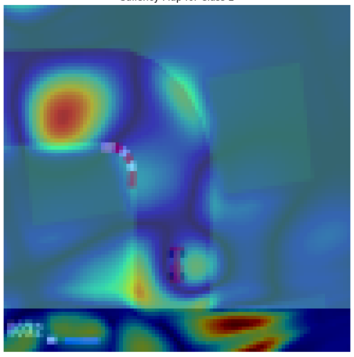
Saliency Map for Class 0



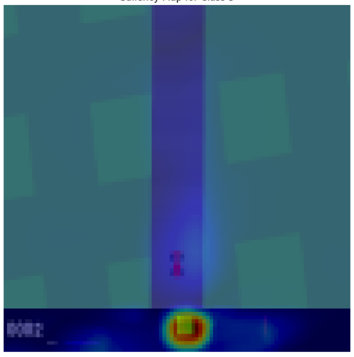
Saliency Map for Class 1



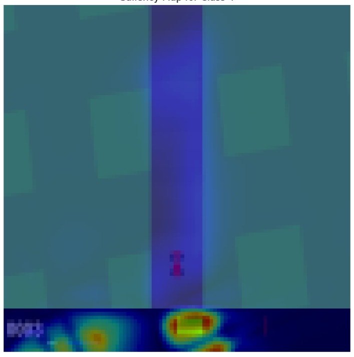
Saliency Map for Class 2



Saliency Map for Class 3



Saliency Map for Class 4



Like before, these areas include the car position, the track and the car sensors.

In the file "model2_simulation" is present the video of the learned model driving the car.