Vehicle Classification with Convolutional Neural Networks and Image Enhancements

Daniilidou Viktoria Paraskevi

Abstract

Vehicle classification is an important task for managing traffic, enabling self-driving cars, and improving transportation systems. The development of research in image processing, pattern recognition, and deep learning has encouraged more research interest in categorizing vehicles using deep learning technology. In everyday life, vehicle classification can be found in vehicle search, real-time traffic monitoring, automatic toll gates, and others. In this project, a Convolutional Neural Network (CNN) model is used in order to classify vehicles based on labeled pictures. The dataset is a collection of vehicle images sorted into five different groups. The model is trained on grayscale and colored images, both with and without data augmentation enhancements. After training, the model showed good accuracy, precision, recall, and F1 scores on both validation and test sets, proving that a CNN can effectively classify vehicles.

1. Introduction and Motivation

Nowadays, with the evolution of technology and the development of self-driving systems, it's very important to identify in a more accurate way different types of vehicles. We can find vehicle classification in many different tasks like managing traffic and collecting tolls. Traditional methods of computer vision don't work well in different lighting, when parts of the vehicle are hidden or unclear. Deep learning, especially using Convolutional Neural Networks (CNNs), has done better in classifying images. CNN models usually need a lot of data and special techniques to work well in real-world situations. In this project I tried to check how much the efficiency of the model increases, by using data augmentation that adds more variety to the data.

2. Methodology

In this project a Convolutional Neural Network model has been trained and tested with grayscale, colored and image optimization techniques in order to test how the model's learning can be improved. The model sorts vehicle images into different classes based on the images it receives. The dataset includes vehicle images that are labeled, with each label representing a different type of vehicle.

2.1 Dataset

The data set is taken from Kaggle (https://www.kaggle.com/datasets/mrtontrnok/5-vehichles-for-multicategory-classification). The images were obtained from the COCO dataset and have been edited to remove the background and resize them to 192x192 pixels. It is divided into three parts: training, validation, and testing. So, the initial size of the images is 192 x 192 but during the data preparation the images are resized to 64x64 pixels. In addition to the resize of the images, due to the fact that the dataset is not balanced there are more pictures of some categories than other, meaning that upsampling was necessary in order to get the same number of images in all classes. For that reason, upsampling from scikit learn was used (sklearn.utils.resample) in order to resample the sparse samples in a consistent way. However, the model was trained both in balanced and unbalanced dataset with the aim of checking if upsampling have a significant impact to the evaluation results. So, the initial training dataset size is 5418 images and the upsampled dataset is 6592 images.

Process of Data:
In order to improve the efficiency of the model, the training was also tested after using data augmentation techniques from Pytorch which included Random Horizontal Flip, Random rotation, Random Resized Crop and Color Jitter. As a result, there was a 50% chance that the image would be flipped sideway, and the image could be turned up to 15 degrees in any direction. Moreover, the image was randomly cut and then resized to the target size, with the size of the cut being between 80% and 100% of the original. Finally, the brightness, contrast, saturation, and color of the image were randomly change and the image tensor was normalized with mean and standard deviation.

2.2 Model Structure

First of all, the label of each image was extracted from the path of the file. In total, there are five unique labels: car, truck, motorcycle, train and bus.
The CNN model, which is called in the code TransportClass , is developed with both convolutional and fully connected layers. The kernel size is 3x3 with stride=1 and padding=1. A filter 3x3 might capture a small amount of patterns, but when it's used in multiple layers it can capture larger and more complex features and at the same time is more computationally efficient. There are four convolutional layers and each layer is followed by Batch Normalization to reduce overfitting and improve the generalization ability of the model and ReLU activation functions to keep the model sparse and efficient. After each convolutional layer, there is a Max Pooling layer which by choosing the largest values, reduces the size of the input image while it keeps the important information. The decision to use four layers was made after experimenting with different number of layers which yielded suboptimal results. So, when the dataset is converted to grayscale the input channel is 1 and the size of the image is 64x64. Following the Max Pooling layer, the image size is reduced to 32x32. In the second layer =;./there are 128 output channels and the output image is 16x16 and in the third layer there are 256 output channels and the output image is further reduced to 8x8. Finally in the fourth layer there are 512 output channels and the output size is 4x4. The same applies also to colored images, with the only difference that in the first layer the input channel is 3 in order to accommodate the RGB channels. This structured approach of decreasing the spatial dimensions while increasing the depth of the architecture enables the model to learn more and more complex features from each layer. Reducing the image size progressively allows the network to look at more intricate details while the increase of channels allows for capturing more complex and abstract representations. This hierarchical feature extraction is important in the success of the model as it ensures that the model understands both general patterns and subtle details within the dataset, regardless of whether the images are grayscale or colored.

The output from the convolutional layers is flattened and then passed through fully connected layers before the final classification. The fully connected layers are defined using the nn.Sequential module. In the beginning there is a nn.Linear layer which flattens the output from the convolutional layers. So, the input to this layer has a size of 512x4x4 and is transformed to a vector of size 1024. After the nn.Lineal layer, ReLu activation function is used to introduce non linearity and a Dropout layer (p=0.5) to avoid overfitting. The output of the dropout layer is then fed into another nn.Linear layer which reduces the dimension from 1024 to the number of classes existed in the dataset and a LogSoftmax layer transforms the output into log probabilities. Finally, the multidimensional array is flattened into a one dimensional vector using the view function, is passed through the fully connected layers self.fc_layers and the result is returned as the model's output, which is the class prediction for each image.

2.3 Training

The aim of the project was to test the efficiency of the model trained with grayscale and colored image, but also with balanced and unbalanced dataset. Also, data augmentation was applied both in grayscale and colored images in order to compare the results. So, the model was trained for 50 epochs when the dataset is grayscale and colored images and for 120 epochs when the dataset is with data augmentation. As a Loss Function, Negative Log Likelihood Loss (NLLLoss) is used which is a common choice in classification tasks, which measures how closely the model predictions align with the ground truth labels. Moreover, Adaptive Moment Estimation (Adam) optimizer with a learning rate of 0.001 is used to minimize the loss function during the training of the model. It is a common choice for a CNN due to its efficiency. The device in which the training was conducted is GPU, in order to accelerate the training procedure.

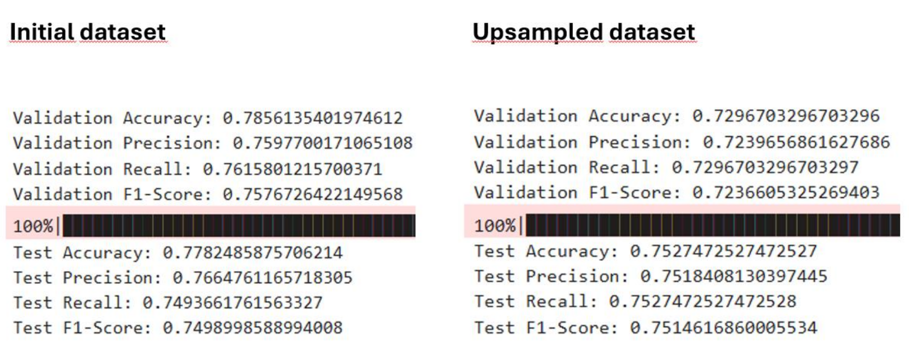
3. Evaluation

After the training, the evaluate_model function is responsible for evaluating how well the trained model performs with a different dataset each time. The evaluation is completed with the calculation of the evaluation metrics from sklearn to analyze the model's performance. First of all, model.eval() is used in order to deactivate the dropout layer and any updates related to batch normalization. This ensures that the model is not affected by any random factors incorporated during training, but it is based only on the learned parameters. Again, in the evaluation torch.device("cuda:1") is used in order to run the model to GPU and speed up the processing. Moreover, two lists are created inside the function one for the predicted labels and one for the true labels. The function runs for the dataloader until there are no more batches left and the outputs are computed for the given inputs and labels based in which output has the highest value (outputs.argmax(dim=1)). Finally, after each epoch the evaluation of the model's performance is calculated with the metrics of accuracy, precision, recall and F1-score, providing an overall understanding of how well or bad the model performs in this specific classification task.

3.1 Results

As it is described above the model´s performance is calculated with the sklearn metrics of accuracy, precision, recall and F1-score both in validation and test dataset. After eight different trainings the evaluation proves that color matters and the data augmentation plays a significant role for enhancing the efficiency of the model. The images below present the results after each training session.

Evaluation(Grayscale)



Initial dataset

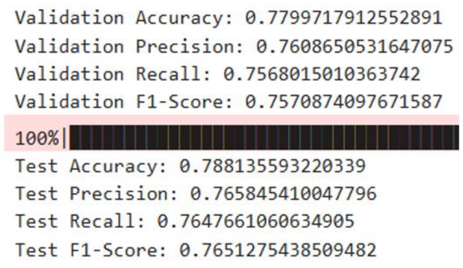
```
Validation Accuracy: 0.7856135401974612
Validation Precision: 0.7597700171065108
Validation Recall: 0.7615801215700371
Validation F1-Score: 0.7576726422149568
100%|██████████████████████████████
Test Accuracy: 0.7782485875706214
Test Precision: 0.7664761165718305
Test Recall: 0.7493661761563327
Test F1-Score: 0.7498998588994008
```

Upsampled dataset

```
Validation Accuracy: 0.7296703296703296
Validation Precision: 0.7239656861627686
Validation Recall: 0.7296703296703297
Validation F1-Score: 0.7236605325269403
100%|██████████████████████████████
Test Accuracy: 0.7527472527472527
Test Precision: 0.7518408130397445
Test Recall: 0.7527472527472528
Test F1-Score: 0.7514616860005534
```

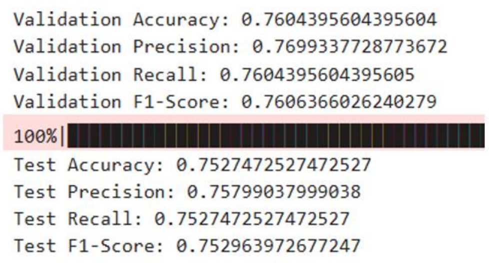
Evaluation(RGB)

Initial dataset



```
Validation Accuracy: 0.7799717912552891
Validation Precision: 0.7608650531647075
Validation Recall: 0.7568015010363742
Validation F1-Score: 0.7570874097671587
100%|████████████████████████████████████
Test Accuracy: 0.788135593220339
Test Precision: 0.765845410047796
Test Recall: 0.7647661060634905
Test F1-Score: 0.7651275438509482
```

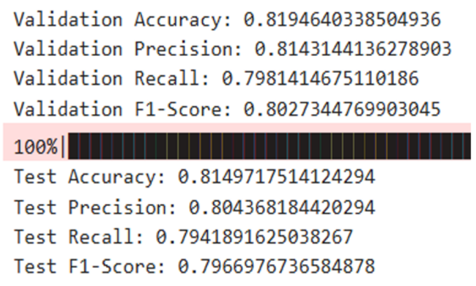
Upsampled dataset



```
Validation Accuracy: 0.7604395604395604
Validation Precision: 0.7699337728773672
Validation Recall: 0.7604395604395605
Validation F1-Score: 0.7606366026240279
100%|████████████████████████████████████
Test Accuracy: 0.7527472527472527
Test Precision: 0.75799037999038
Test Recall: 0.7527472527472527
Test F1-Score: 0.752963972677247
```

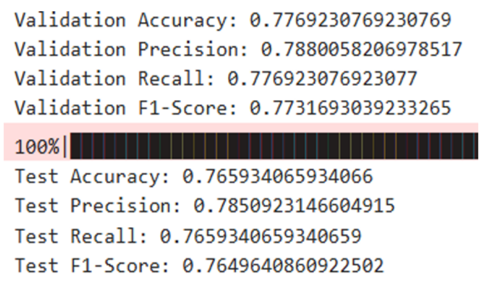
Evaluation(Grayscale-Data Aug.)

Initial dataset



```
Validation Accuracy: 0.8194640338504936
Validation Precision: 0.8143144136278903
Validation Recall: 0.7981414675110186
Validation F1-Score: 0.8027344769903045
100%|████████████████████████████████████
Test Accuracy: 0.8149717514124294
Test Precision: 0.804368184420294
Test Recall: 0.7941891625038267
Test F1-Score: 0.7966976736584878
```

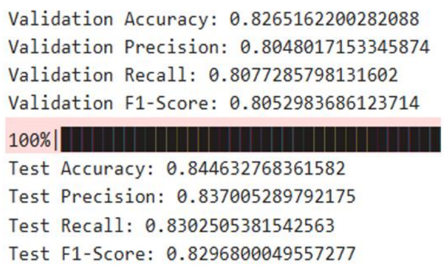
Upsampled dataset



```
Validation Accuracy: 0.7769230769230769
Validation Precision: 0.7880058206978517
Validation Recall: 0.776923076923077
Validation F1-Score: 0.7731693039233265
100%|████████████████████████████████████
Test Accuracy: 0.765934065934066
Test Precision: 0.7850923146604915
Test Recall: 0.7659340659340659
Test F1-Score: 0.7649640860922502
```

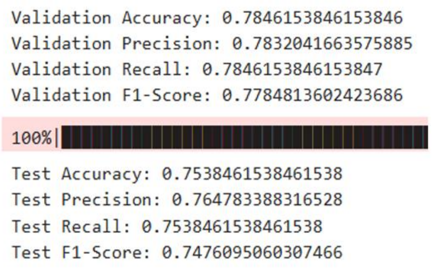
Evaluation(RGB-Data Aug.)

Initial dataset



```
Validation Accuracy: 0.8265162200282088
Validation Precision: 0.8048017153345874
Validation Recall: 0.8077285798131602
Validation F1-Score: 0.8052983686123714
100%|████████████████████████████████████
Test Accuracy: 0.844632768361582
Test Precision: 0.837005289792175
Test Recall: 0.8302505381542563
Test F1-Score: 0.8296800049557277
```

Upsampled dataset

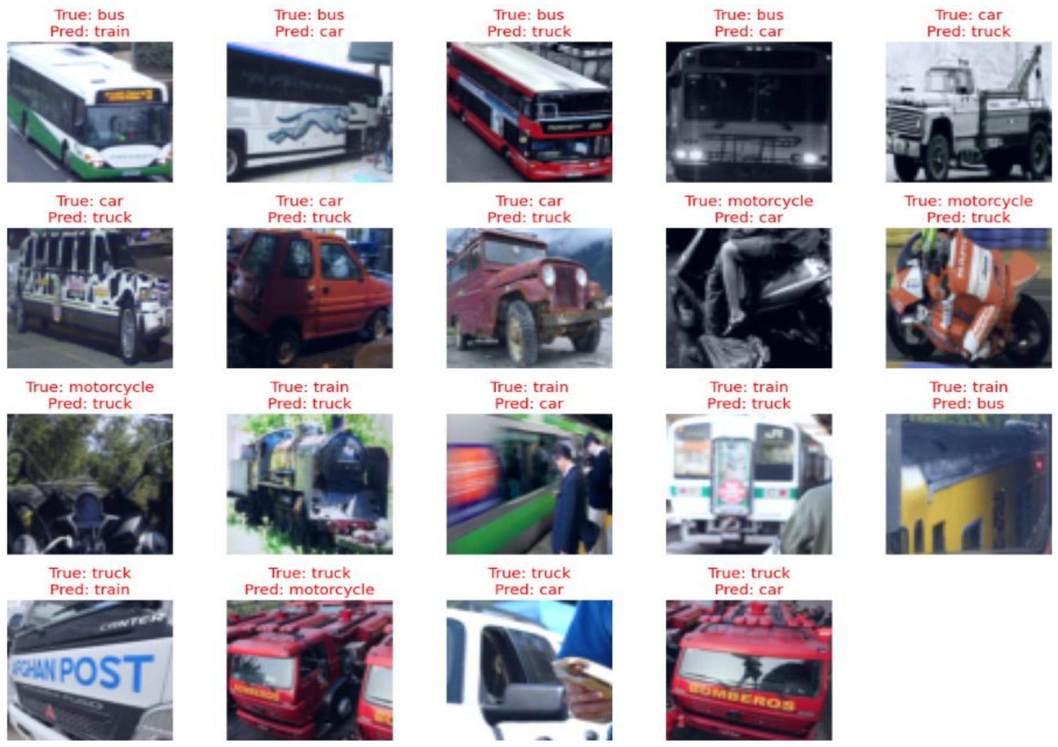


```
Validation Accuracy: 0.7846153846153846
Validation Precision: 0.7832041663575885
Validation Recall: 0.7846153846153847
Validation F1-Score: 0.7784813602423686
100%|████████████████████████████████████
Test Accuracy: 0.7538461538461538
Test Precision: 0.764783388316528
Test Recall: 0.7538461538461538
Test F1-Score: 0.7476095060307466
```

Starting from the grayscale images, the loss after 50 epochs using the initial dataset is approximately 0.05 and with the upsampled dataset is around 0.02. The accuracy around 78% in the initial dataset is satisfactory, especially if we take into consideration that some image details can be harder to be captured without color. However, there is potential for enhancement in the evaluation metrics. Going through the evaluation of the balanced grayscale dataset there is a slightly higher consistency between precision and recall especially in the test set but the validation accuracy was decreased from 78.56% to 72.97% and the test accuracy from 77.82% to 75.27%. As a result, while the upsampled dataset demonstrates more balanced metrics the drop in accuracy might indicate that the learning is restricted to some specific patterns. Naive oversampling might mitigate class imbalance effects by simply duplicating minority class examples. Due to this strategy, they bear the risk of overfitting the model to the training data, resulting in poor generalization in the inference phase. A similar pattern is observed when evaluating the training performance on colored images, where the initial dataset achieves an accuracy of around 78%,while the upsampled dataset has a slightly lower accuracy of 76%. However, better results can be found after applying data augmentation, both in grayscale and colored images. In the case of grayscale images, data augmentation improves the test and validation accuracy is increased to 81%, though the upsampled dataset still has lower evaluation metrics around 77%. The highest improvements are seen with the data augmentation on the colored images dataset, where the accuracy of the initial dataset increases to 84%. Despite this improvement, the accuracy in the upsampled dataset remains below the initial, around 78%.

3.2 Error Analysis

As presented above, the evaluation metrics have been improved when the training was performed with data augmentation on the colored images. However, the accuracy still did not exceed the 84%. Using pyplot from Matplotlib to analyze the model's performance and identify what was difficult for the network to recognize, apart from the incorrect predictions that the model makes, there are also some instances of misclassifications where the model´s predictions were actually correct but the dataset's true label is wrong. For example, in the last image of the first row below, the model predicted a truck which is correct, but the true label given by the dataset is car. This suggests that instances where the model correctly identifies an image but the dataset's label is inaccurate may be limiting further improvements to the evaluation metrics.



4. Next Steps and Future Considerations.

In future work, there are several ways the model's performance could be improved. The first step would be to compare the CNN with Vision Transformers (ViT). This could show us if their better architecture and feature extraction using self attention techniques can make the model more accurate. Moreover, observing that the upsampled datasets demonstrated lower evaluation metrics, changing the upsampling technique could affect positively the model's accuracy. SMOTE for nominal/ categorical data, which generates new records by interpolating between existing minority-class samples, leading to higher diversity can reduce overfitting risks and might be a more benificial upsampling technique. Finally, trying different setups for the layers and different sizes of images could help the model catch more details, which would make it more accurate and better.

5. References

Funt, B., & Zhu, X. (2018). *Does Colour Matter?* CIC26 Conference Proceedings https://www2.cs.sfu.ca/~funt/Funt_Zhu_DoesColourMatter_CIC26_2018.pdf

Singh, A., Bay, A., & Mirabile, A. (2020). Assessing the importance of colours for CNNs in object recognition https://arxiv.org/pdf/2012.06917

IBM Cloud Education. *What is Upsampling?* https://www.ibm.com/topics/upsampling

Mostly AI. (n.d.). *Upsampling methods for improving prediction accuracy.*. https://mostly.ai/blog/upsampling-methods-for-improving-prediction-accuracy

Choudhury, A. Here's what I've learnt about sklearn resample. Towards Data Science. https://towardsdatascience.com/heres-what-i-ve-learnt-about-sklearn-resample-ab735ae1abc4

Som, A. Image Augmentation for Creating Datasets Using PyTorch for Dummies by a Dummy. https://anushsom.medium.com/image-augmentation-for-creating-datasets-using-pytorch-for-dummies-by-a-dummy-a7c2b08c5bcb