Car Recognition System

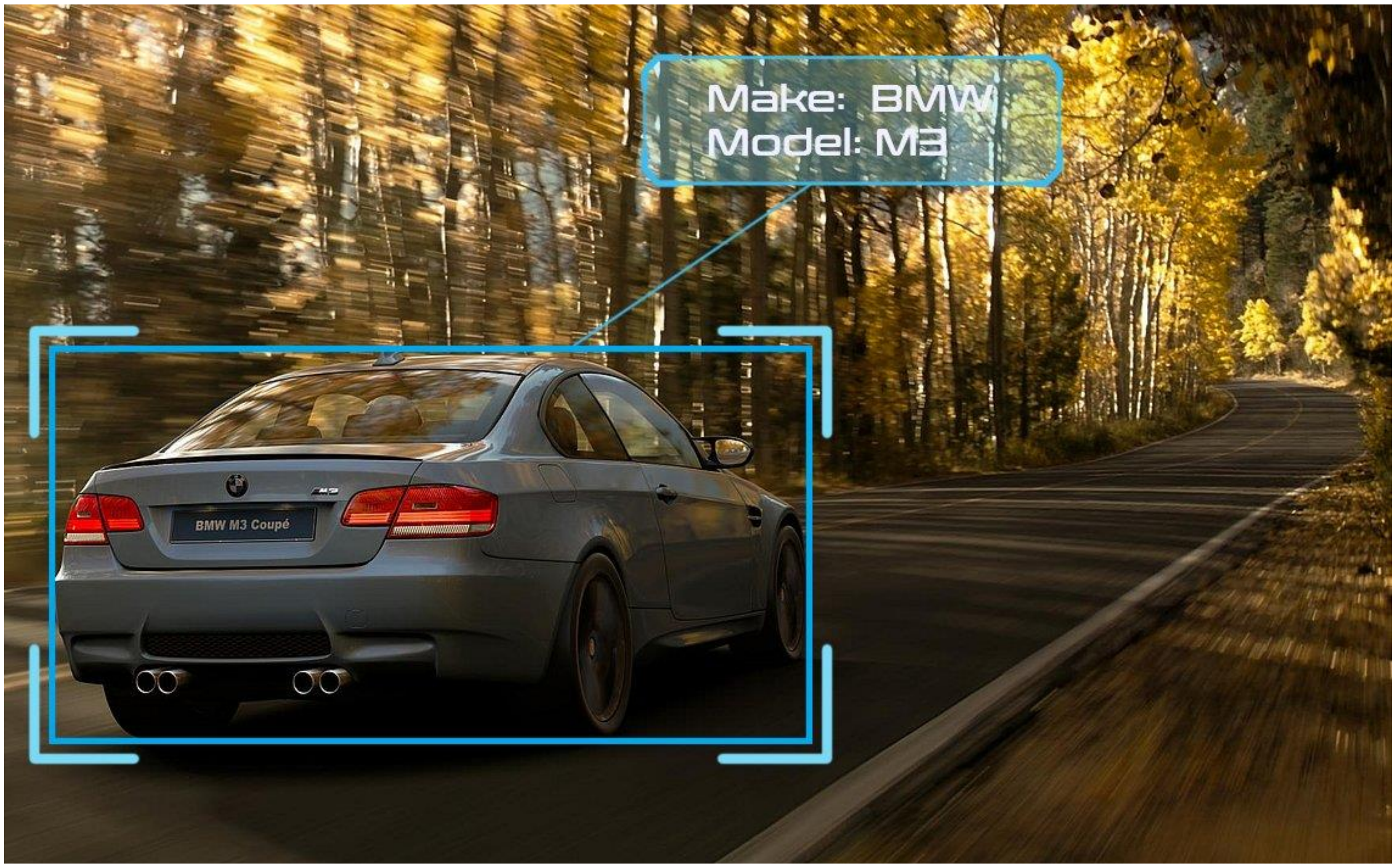


Image Source: Spectrico | Car Make and Model Recognition

Project Report

Version 1.0

Date: 09/07/2023

Place: Eindhoven

Semester 6: AI for Society
Author: Momchil Valkov
Type: Personal Challenge

ABSTRACT

The following report presents documentation of my Car Recognition System, a personal project I have been working on for the past 5 months at Fontys. The objective of this system is to identify and validate the make and model of vehicles based on images, which is particularly valuable for insurance companies in tasks such as policyholder verification, claim validation for accidents, and identification of stolen vehicles. Additionally, car spotters, enthusiasts who photograph exotic or rare cars, can benefit from this system to determine the make and model of such vehicles.

To achieve accurate vehicle identification, cutting-edge techniques, including Convolutional Neural Network training, have been implemented in this project. The system was trained and tested using the Stanford Cars Dataset. Python libraries such as VGG16, NumPy, Mathplotlib, Pandas, Keras, and TensorFlow were leveraged for the implementation. The evaluation results demonstrate a significant improvement in accuracy and efficiency compared to previous models, confirming the effectiveness of the approach.

The outcomes of this project not only contribute to the field of vehicle recognition but also hold potential applications in traffic management and autonomous vehicles. Further research and enhancements can be explored to optimize the system for specific environments, thereby expanding its applicability and potential societal impact.

Table of Contents

Introduction

The aim of this project is to develop a car recognition system that facilitates the identification of car models and makes. The system is designed to assist car enthusiasts, spotters, and professionals in the automotive industry by providing a digital tool for quick and accurate car identification. Additionally, the system has the potential to benefit insurance companies by streamlining claim processing tasks.

Car spotting is a popular activity among car enthusiasts, who often strive to identify and catalogue unique and rare car models. However, manual identification can be challenging and time-consuming, especially for less common or older models. The car recognition system aims to overcome these challenges by leveraging automatically identifying car models and makes from images.

In addition to its applications for car enthusiasts, the system also holds potential for use in the automotive industry and insurance companies. In the automotive industry, the system can be utilized for cataloguing purposes, assisting professionals in efficiently organizing and managing car inventories. This can be particularly beneficial for car dealerships and manufacturers. Furthermore, insurance companies can leverage the system to streamline claim processing tasks by automating the identification and verification of car models, making the process more efficient and reducing human error.

The primary goal of this project is to develop a robust and accurate car recognition system that can provide reliable results across a wide range of car models and makes. By employing state-of-the-art machine learning and computer vision techniques, we aim to create a system that can accurately identify cars from various angles, lighting conditions, and backgrounds.

This report will delve into the details of the system's development process, including the data preprocessing steps, model architecture, training methodology, and evaluation metrics. Additionally, we will explore the potential impact of the system on car enthusiasts, professionals in the automotive industry, and insurance companies.

Problem Description

The idea of developing a car recognition system came from the need to address the challenges faced by car enthusiasts and professionals in the automotive industry when it comes to accurately identifying car models and makes. Manual identification can often be subjective, time-consuming, and prone to errors, especially when dealing with rare or less common car models. This led to the realization that leveraging the power of artificial intelligence and computer vision could provide a potential solution to this problem.

By utilizing the capabilities of machine learning algorithms and deep neural networks, the car recognition system has the potential to accurately classify and identify car models based on their unique visual features. The system can analyze attributes, like distinctive features to make informed predictions about the make and model of a given car. This not only simplifies the identification process but also enhances the overall car spotting experience for enthusiasts.

Moreover, the idea of using the car recognition system for professional purposes, such as cataloguing in the automotive industry and streamlining claim processing in insurance companies, arose from the recognition of the system's broader applicability. Automating the identification process can significantly reduce manual effort, improve accuracy, and enhance operational efficiency in these domains. The advanced deep learning techniques, along with the availability of large-scale car image datasets, have made it feasible to develop a robust and accurate car recognition system. These advancements have paved the way for training models capable of recognizing a wide range of car models and makes, regardless of variations in lighting conditions, angles, or backgrounds.

Considering these factors, I believed that the car recognition system had the potential to provide an effective and efficient solution to the challenges faced by car enthusiasts and professionals in identifying car models and makes. By automating the recognition process and making use of complicated machine learning methods, the idea aimed to improve accuracy, save time, and enhance the overall experience for users, ultimately contributing to a more informed and seamless interaction with the world of cars.

Process

1. Finding the right data

Finding the right dataset for my project proved to be a challenging task that required extensive research and exploration. I invested a significant amount of time scouring various platforms, including Kaggle, in search of a suitable dataset that could meet the requirements of my car recognition project. The process involved sifting through numerous options, and evaluating the size, quality, and diversity of the available datasets.

After tirelessly searching, I stumbled upon the Stanford Car Dataset, and it immediately caught my attention. This dataset proved to be the perfect fit for my project due to its substantial size and comprehensive coverage of car images. With 16 000 high-quality images, split equally for testing and training, labelled, spanning a wide range of car models and makes, the Stanford Car Dataset offered the diversity and richness needed to train a robust car recognition system. It provided the foundation I needed to develop a powerful model capable of accurately identifying car models and makes. The extensive effort invested in finding the right dataset ultimately looked promising, but the next step was exploring the data.

2. Making the plan

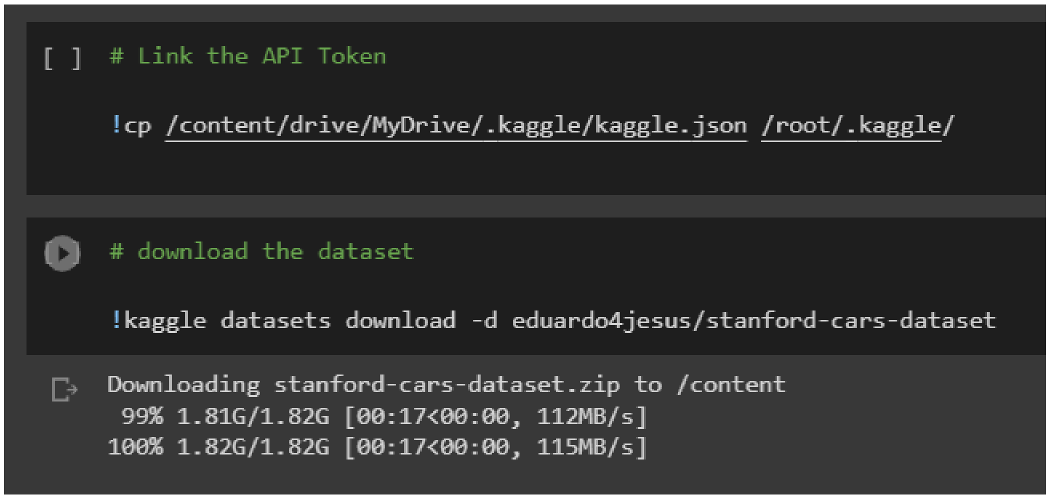
Before jumping into working on the project, I should have a good plan, of how I would approach it. Here came the moment, when I choose which machine learning technique I will be using. Since I was working on RGB images, I decide the best choice would be Convolutional Neural Networks, because they are highly effective for training images due to their ability to capture spatial relationships and local features. They employ small receptive fields that scan the input images, allowing them to learn hierarchical representations by combining local features. CNNs also utilize parameter sharing, enabling them to efficiently learn translation-invariant features and reducing the number of parameters needed. Additionally, pooling layers help achieve translation invariance and preserve important features, while convolutional filters extract relevant patterns from the images.

In addition, I will explore the data using various techniques such as data visualization and statistical analysis to gain insights into the dataset's properties. I will also preprocess the data by resizing the images to a uniform size, converting them to RGB, and normalizing the pixel values to facilitate the training process.

Moreover, I will split the dataset into training, validation, and testing sets and train the model on the training set. I will then fine-tune the model using transfer learning techniques to further improve its accuracy. Finally, I will evaluate the model's performance on the testing set and make necessary adjustments to improve its accuracy.

3. Getting the data

This might be sounding like a silly one, but I was wondering, what is the best method for getting the data from Kaggle. Of course, you can directly download it, but I was planning to use Google Collab and Drive, since my PC is not powerful at all, and my video card is not suitable for training, since I am on Windows and I have an AMD video card. So I figure out, that the best practice is to get it with an API command. I set up the API file on My Drive, and download it with a command.



```
# Link the API Token

!cp /content/drive/MyDrive/.kaggle/kaggle.json /root/.kaggle/
```

```
# download the dataset

!kaggle datasets download -d eduardo4jesus/stanford-cars-dataset
```

```
Downloading stanford-cars-dataset.zip to /content
 99% 1.81G/1.82G [00:17<00:00, 112MB/s]
100% 1.82G/1.82G [00:17<00:00, 115MB/s]
```

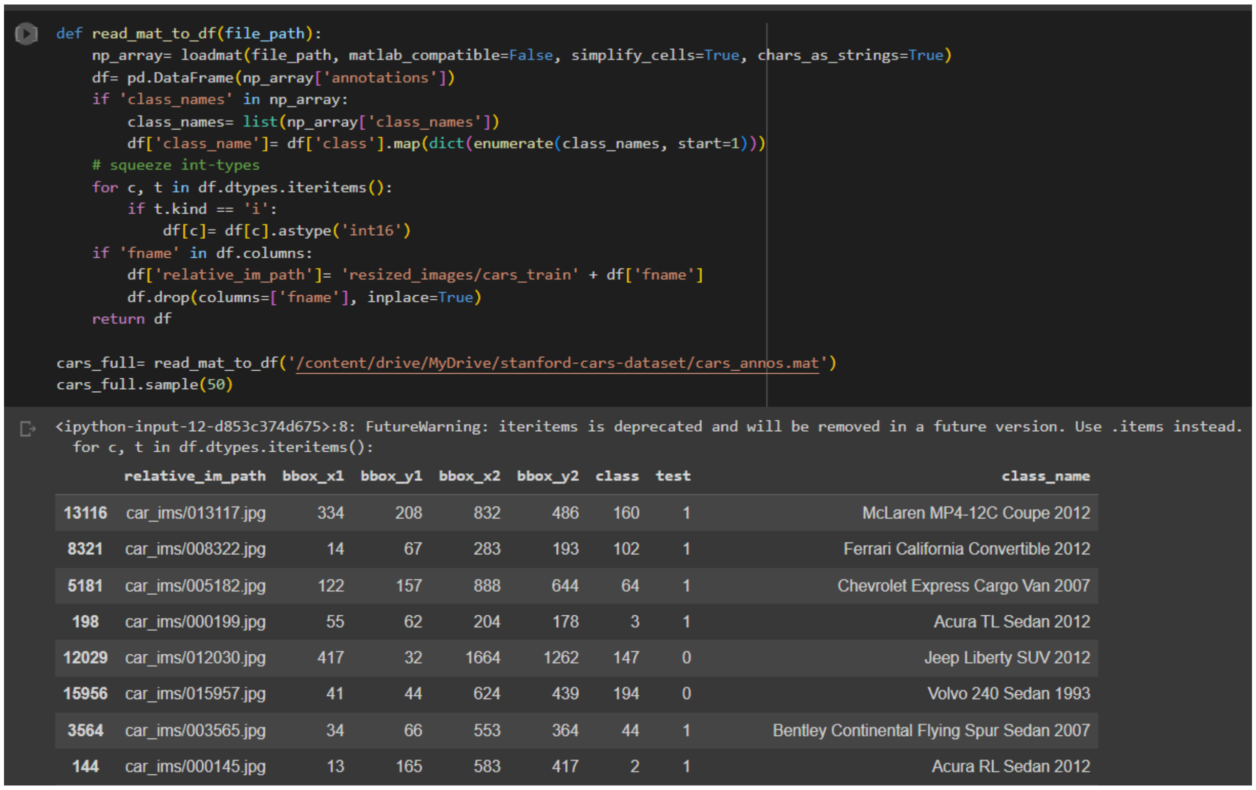
Figure 1: Linking Kaggle API and fetching dataset

4. Imports

I had put a block of code at the beginning, containing the project's imports. As you can imagine, it was quite empty in the beginning, but by getting deeper into the project, more imports showed up which were needed in order to build my model, so I end up with quite a big list by the end. The important part was to keep them all in the same place, so when I have to re-run the project, I just run the script in the beginning, and I can continue with my work, where I left it last time.

5. First version

Here is a good time to mention that I had a first and second (final) version of my project. The reason for it was the data. In the original Stanford Cars Dataset, there were a lot of issues. Working with complex data formats, such as .mat files, can often present challenges and issues during data processing. In my project, I encountered several difficulties while handling the .mat file format, leading to unexpected complications. The initial attempts to work with the .mat file format proved to be problematic, as it required specialized tools and libraries for proper extraction and interpretation. Additionally, when converting the data to a more manageable format like CSV, I discovered that the paths to the associated images were incorrect, requiring extensive cleaning and adjustment. This process involved identifying and rectifying errors, ensuring proper data alignment, and addressing inconsistencies. Although these challenges were time-consuming and frustrating, I manage to clean it. Here is before:



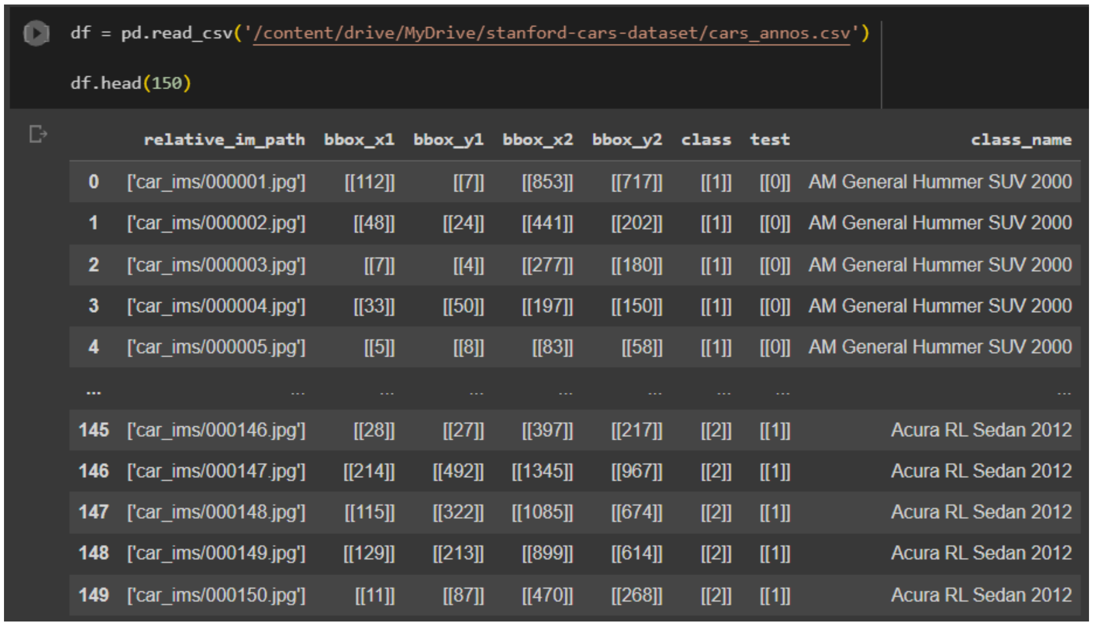
```python
def read_mat_to_df(file_path):
    np_array= loadmat(file_path, matlab_compatible=False, simplify_cells=True, chars_as_strings=True)
    df= pd.DataFrame(np_array['annotations'])
    if 'class_names' in np_array:
        class_names= list(np_array['class_names'])
        df['class_name']= df['class'].map(dict(enumerate(class_names, start=1)))
    # squeeze int-types
    for c, t in df.dtypes.iteritems():
        if t.kind == 'i':
            df[c]= df[c].astype('int16')
    if 'fname' in df.columns:
        df['relative_im_path']= 'resized_images/cars_train' + df['fname']
        df.drop(columns=['fname'], inplace=True)
    return df

cars_full= read_mat_to_df('/content/drive/MyDrive/stanford-cars-dataset/cars_annos.mat')
cars_full.sample(50)
```

```
<ipython-input-12-d853c374d675>:8: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  for c, t in df.dtypes.iteritems():
```

	relative_im_path	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	test	class_name
13116	car_ims/013117.jpg	334	208	832	486	160	1	McLaren MP4-12C Coupe 2012
8321	car_ims/008322.jpg	14	67	283	193	102	1	Ferrari California Convertible 2012
5181	car_ims/005182.jpg	122	157	888	644	64	1	Chevrolet Express Cargo Van 2007
198	car_ims/000199.jpg	55	62	204	178	3	1	Acura TL Sedan 2012
12029	car_ims/012030.jpg	417	32	1664	1262	147	0	Jeep Liberty SUV 2012
15956	car_ims/015957.jpg	41	44	624	439	194	0	Volvo 240 Sedan 1993
3564	car_ims/003565.jpg	34	66	553	364	44	1	Bentley Continental Flying Spur Sedan 2007
144	car_ims/000145.jpg	13	165	583	417	2	1	Acura RL Sedan 2012

Figure 2: Visualizing .mat file successfully

Throughout the process:

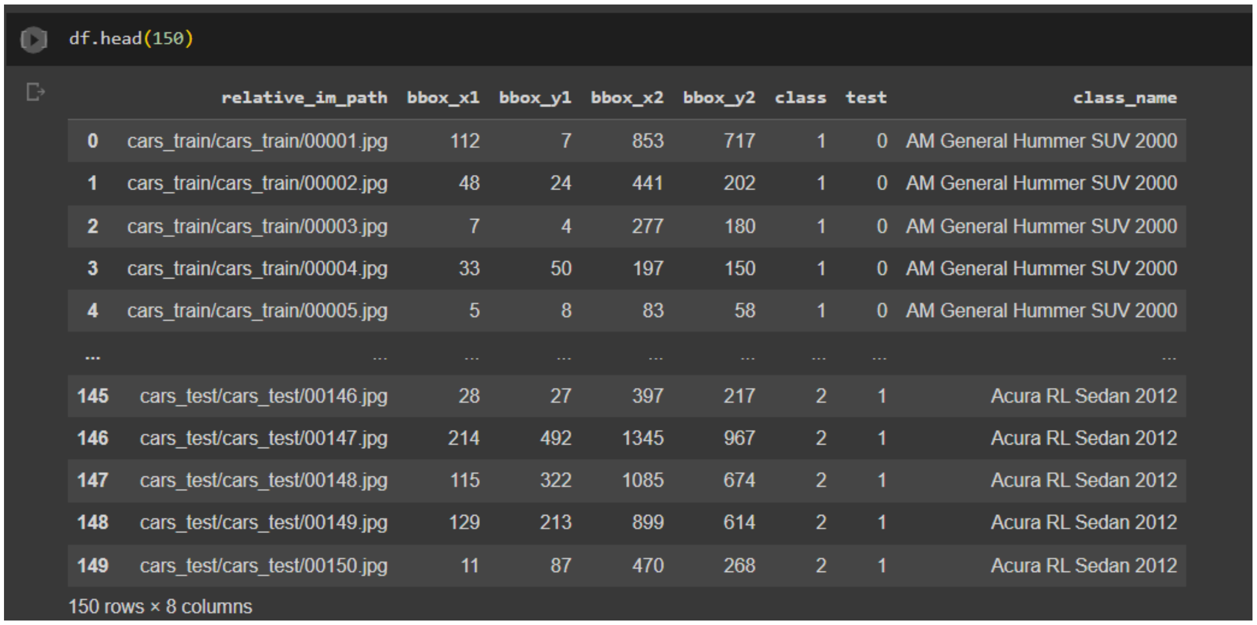


```python
df = pd.read_csv('/content/drive/MyDrive/stanford-cars-dataset/cars_annos.csv')

df.head(150)
```

	relative_im_path	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	test	class_name
0	['car_ims/000001.jpg']	[[112]]	[[7]]	[[853]]	[[717]]	[[1]]	[[0]]	AM General Hummer SUV 2000
1	['car_ims/000002.jpg']	[[48]]	[[24]]	[[441]]	[[202]]	[[1]]	[[0]]	AM General Hummer SUV 2000
2	['car_ims/000003.jpg']	[[7]]	[[4]]	[[277]]	[[180]]	[[1]]	[[0]]	AM General Hummer SUV 2000
3	['car_ims/000004.jpg']	[[33]]	[[50]]	[[197]]	[[150]]	[[1]]	[[0]]	AM General Hummer SUV 2000
4	['car_ims/000005.jpg']	[[5]]	[[8]]	[[83]]	[[58]]	[[1]]	[[0]]	AM General Hummer SUV 2000
...	...	...	...	...	...	...	...	...
145	['car_ims/000146.jpg']	[[28]]	[[27]]	[[397]]	[[217]]	[[2]]	[[1]]	Acura RL Sedan 2012
146	['car_ims/000147.jpg']	[[214]]	[[492]]	[[1345]]	[[967]]	[[2]]	[[1]]	Acura RL Sedan 2012
147	['car_ims/000148.jpg']	[[115]]	[[322]]	[[1085]]	[[674]]	[[2]]	[[1]]	Acura RL Sedan 2012
148	['car_ims/000149.jpg']	[[129]]	[[213]]	[[899]]	[[614]]	[[2]]	[[1]]	Acura RL Sedan 2012
149	['car_ims/000150.jpg']	[[11]]	[[87]]	[[470]]	[[268]]	[[2]]	[[1]]	Acura RL Sedan 2012

Figure 3: Converted .mat to .csv

and after results:



```
df.head(150)
```

	relative_im_path	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	test	class_name
0	cars_train/cars_train/00001.jpg	112	7	853	717	1	0	AM General Hummer SUV 2000
1	cars_train/cars_train/00002.jpg	48	24	441	202	1	0	AM General Hummer SUV 2000
2	cars_train/cars_train/00003.jpg	7	4	277	180	1	0	AM General Hummer SUV 2000
3	cars_train/cars_train/00004.jpg	33	50	197	150	1	0	AM General Hummer SUV 2000
4	cars_train/cars_train/00005.jpg	5	8	83	58	1	0	AM General Hummer SUV 2000
...	...	...	...	...	...	...	...	...
145	cars_test/cars_test/00146.jpg	28	27	397	217	2	1	Acura RL Sedan 2012
146	cars_test/cars_test/00147.jpg	214	492	1345	967	2	1	Acura RL Sedan 2012
147	cars_test/cars_test/00148.jpg	115	322	1085	674	2	1	Acura RL Sedan 2012
148	cars_test/cars_test/00149.jpg	129	213	899	614	2	1	Acura RL Sedan 2012
149	cars_test/cars_test/00150.jpg	11	87	470	268	2	1	Acura RL Sedan 2012

150 rows × 8 columns

Figure 4: Data Manipulation

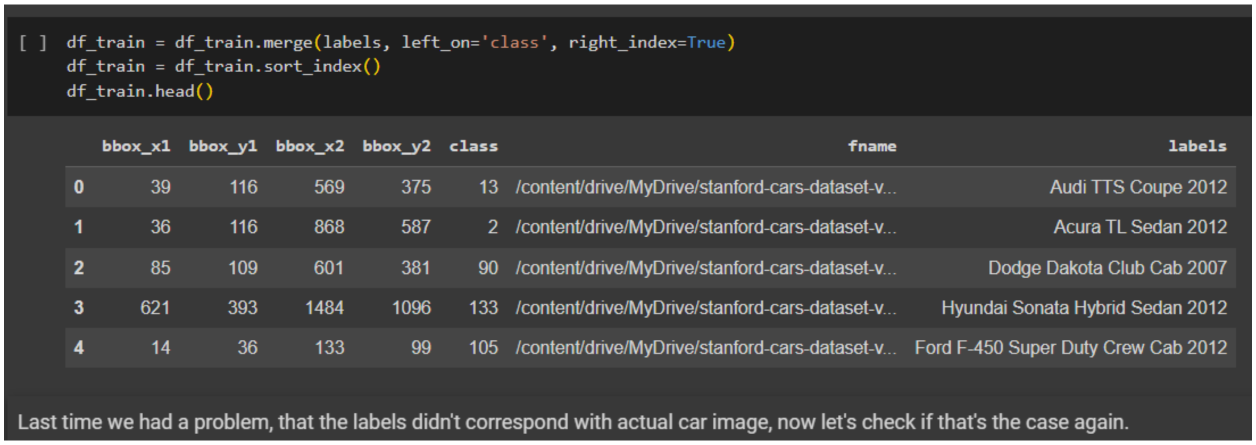
The main problem was, that the table provided, assumes all of the images are located in `car_ims` folder, while the truth was, depending if they are test or train images, they were located in respectively `cars_test/cars_test` and `cars_train/cars_train`. But with a bit of data manipulation, it turns out how it should look like.

However, during the pre-processing of the images and labels, I encountered so many problems, like missing images, wrong labels, wrong shapes of `.npy` files, which I was going to use for training, to the point that the whole training process was failing, going back and forth to training and pre-processing, to the point I just thought it's not worth it anymore. Obviously, the dataset was a mess, I already had spent so much time on it, so I just start it's better if I start my project from beginning, which is version two, where I found the reworked data.

6. Second version – Data Preparation

Even though the stress was high, starting over with my project, which I have been working on for a few months and now I have a few weeks left to finish it up, it wasn't that complicated, because I borrowed a lot from my first one, especially in the beginning.

In the reworked dataset, I had a `README.txt` file, which I made use of, because there was a good explanation about how to use the data, and again `.mat` files, but this time I had experience with them, so I manage to convert them to `.csv` and attach the labels of the cars.



```
[ ] df_train = df_train.merge(labels, left_on='class', right_index=True)
    df_train = df_train.sort_index()
    df_train.head()
```

	bbox_x1	bbox_y1	bbox_x2	bbox_y2	class	fname	labels
0	39	116	569	375	13	/content/drive/MyDrive/stanford-cars-dataset-v...	Audi TTS Coupe 2012
1	36	116	868	587	2	/content/drive/MyDrive/stanford-cars-dataset-v...	Acura TL Sedan 2012
2	85	109	601	381	90	/content/drive/MyDrive/stanford-cars-dataset-v...	Dodge Dakota Club Cab 2007
3	621	393	1484	1096	133	/content/drive/MyDrive/stanford-cars-dataset-v...	Hyundai Sonata Hybrid Sedan 2012
4	14	36	133	99	105	/content/drive/MyDrive/stanford-cars-dataset-v...	Ford F-450 Super Duty Crew Cab 2012

Last time we had a problem, that the labels didn't correspond with actual car image, now let's check if that's the case again.

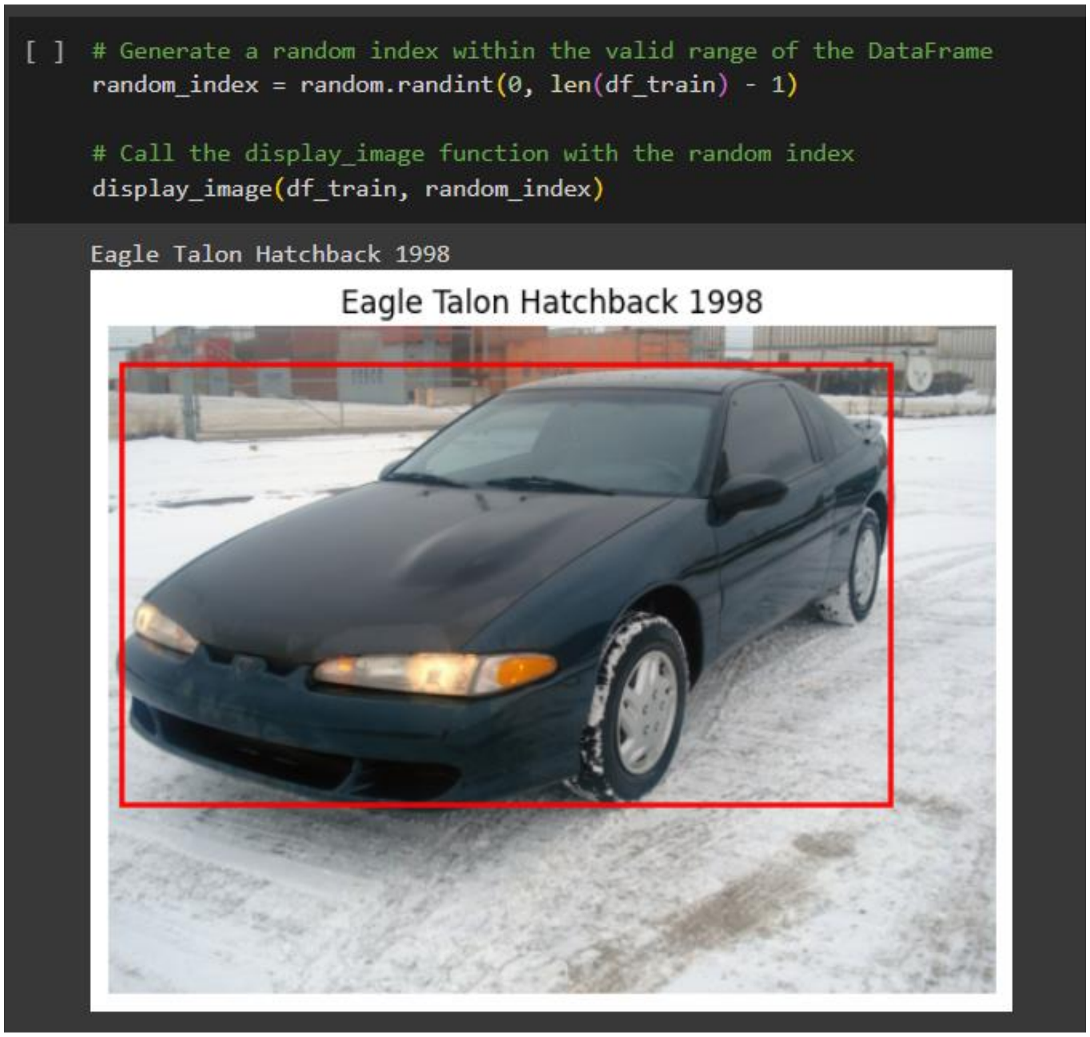
Figure 5: Final version of the .csv table

7. Data visualisation

Data visualization plays a crucial role in machine learning as it helps to gain insights, understand patterns, and communicate complex information effectively. By visualizing data, we can explore the relationships between variables, identify trends, and detect outliers or anomalies. Visualization techniques such as scatter plots, histograms, and heatmaps allow us to uncover patterns and distributions within the data, aiding in feature selection and engineering. Moreover, visualizations help in the evaluation and interpretation of machine learning models by providing a visual representation of performance metrics, such as accuracy, precision, and recall. Visualizing model predictions and decision boundaries can enhance our understanding of how the model is making classifications or predictions.

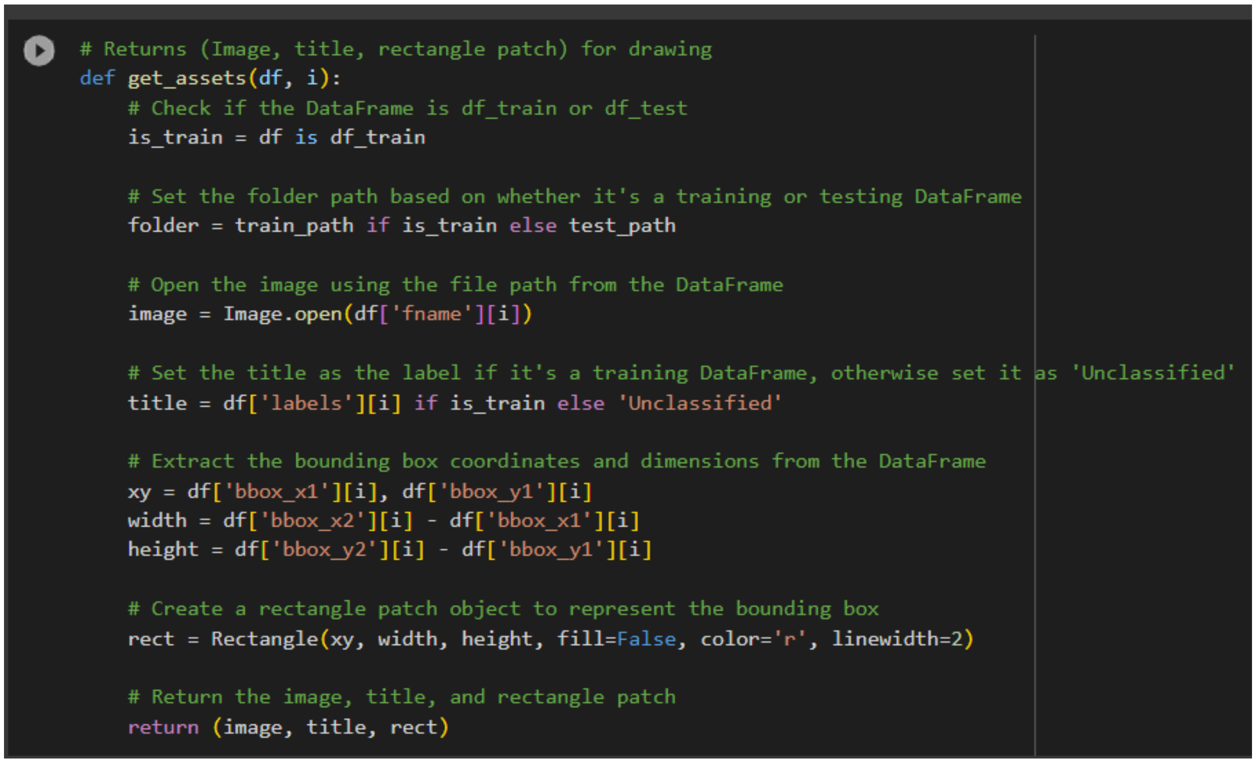
In machine learning, it is essential to ensure the accuracy of the labels and the quality of the images. To meet this requirement, one popular method is to match dataset labels with the respective images and conduct data visualization. Examining the images together with their labels visually allows us to confirm that the labels are an accurate representation of the image content. This verification process ensures correct alignment between the images and labels and assists in pinpointing potential mismatches or mistakes. Data visualization offers a useful means of visually authenticating the dataset, guaranteeing both image quality and appropriate label assignment. Moreover, data visualization can reveal any patterns, trends, or outliers hidden within the dataset, facilitating further study and analysis. In essence, this method bolsters confidence in the dataset's validity, thereby aiding the development of trustworthy and precise machine-learning models.

If you take a look at Figure 5, you will see the ‘bbox’ value. These values are more than just numbers in a table; they are powerful tools for focusing on specific areas within the images. By effectively using these bounding box coordinates, I can directly select the relevant areas where a car is situated, making the image analysis and data processing more efficient and targeted. This realization underscores the value of perseverance in navigating the complexities of machine learning. My initial plan to discard the 'bbox' would have remove a good resource, which will make it harder to process the data accurately. I am grateful for the article that led me to understand its importance.



```
[ ]  # Generate a random index within the valid range of the DataFrame
     random_index = random.randint(0, len(df_train) - 1)

     # Call the display_image function with the random index
     display_image(df_train, random_index)

     Eagle Talon Hatchback 1998
```

Figure 6: Visualising image, using 'bbox' values



```python
# Returns (Image, title, rectangle patch) for drawing
def get_assets(df, i):
    # Check if the DataFrame is df_train or df_test
    is_train = df is df_train

    # Set the folder path based on whether it's a training or testing DataFrame
    folder = train_path if is_train else test_path

    # Open the image using the file path from the DataFrame
    image = Image.open(df['fname'][i])

    # Set the title as the label if it's a training DataFrame, otherwise set it as 'Unclassified'
    title = df['labels'][i] if is_train else 'Unclassified'

    # Extract the bounding box coordinates and dimensions from the DataFrame
    xy = df['bbox_x1'][i], df['bbox_y1'][i]
    width = df['bbox_x2'][i] - df['bbox_x1'][i]
    height = df['bbox_y2'][i] - df['bbox_y1'][i]

    # Create a rectangle patch object to represent the bounding box
    rect = Rectangle(xy, width, height, fill=False, color='r', linewidth=2)

    # Return the image, title, and rectangle patch
    return (image, title, rect)
```

Figure 7: Function behind drawing the lines from 'bbox' values

Additionally, all kind of visualizations were made to the data, like graphs and tables, testing if all of the classes are corresponding to the labels, what is the division of the different cars, which we have most of etc. The idea of the data visualization was not only to check for patterns, but to check for errors also. The data turns out pretty clean at the end.

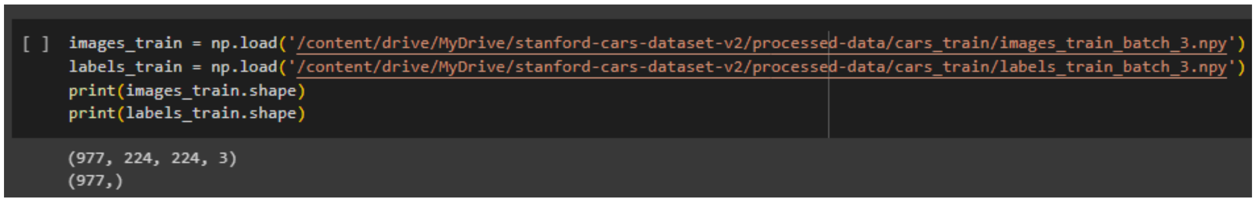
8. Data Preparation for training

After validating all the data, it was time to prepare it. Preparing images for a Convolutional Neural Network (CNN) involves several key steps of pre-processing:

- **Resize**: CNNs typically require all input images to be the same size, so images are often resized to meet this requirement.
- **Normalization**: Image pixel values, which are typically in the range of 0-255, are often normalized to fall within the range 0-1 or -1 to 1. This is done to help the model train more efficiently.
- **Batching**: For efficient training, images are usually fed into the CNN in batches. This allows the model to process multiple images simultaneously, which can lead to faster training times.

So this is exactly what I did. However, first a start with only 10 images, just to validate that my pre-process code is working properly. It got few iterations before it was final. Since it pass the test, I proceed with the whole data. The whole idea was to crop out the image of the car, based on the 'bbox' values, resized it to 224x224, because this is a common practise and it most efficient for most of the models, make sure it have 3 chanels (RGB) and put it in a `.npy` file, which to use later for training. Additionaly, labels for train were saved also.
However, first tries have failed, because some of the paths mentioned in the table, were not existing. This wasn't because they were wrong, but because simply those images were not existing in the dataset anymore, for some reason. So I have created a error handler, which to skip the missing images and count them, so I know by the end, how much is the lost.

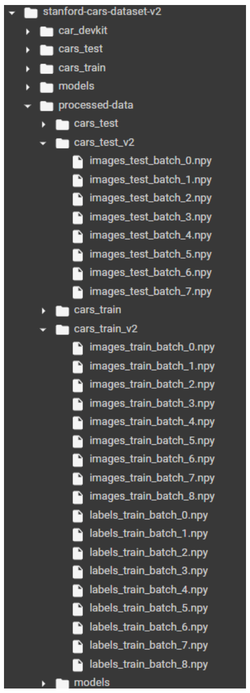
In the end I validate that I have same amount of images and labels, for the train data



```
[ ]  images_train = np.load('/content/drive/MyDrive/stanford-cars-dataset-v2/processed-data/cars_train/images_train_batch_3.npy')
     labels_train = np.load('/content/drive/MyDrive/stanford-cars-dataset-v2/processed-data/cars_train/labels_train_batch_3.npy')
     print(images_train.shape)
     print(labels_train.shape)

     (977, 224, 224, 3)
     (977,)
```

Figure 8: Validating amount of single batch labels and images. (977 images, in size 224x224, 3 channels RGB and 977 labels for them)

In the end I had **9 batches** of train images and labels and **8 batches** of test images. Each batch contains around 1000 images, the total images were **15 697. 7835** train and **7862** test images.



```
stanford-cars-dataset-v2
  car_devkit
  cars_test
  cars_train
  models
  processed-data
    cars_test
    cars_test_v2
      images_test_batch_0.npy
      images_test_batch_1.npy
      images_test_batch_2.npy
      images_test_batch_3.npy
      images_test_batch_4.npy
      images_test_batch_5.npy
      images_test_batch_6.npy
      images_test_batch_7.npy
    cars_train
    cars_train_v2
      images_train_batch_0.npy
      images_train_batch_1.npy
      images_train_batch_2.npy
      images_train_batch_3.npy
      images_train_batch_4.npy
      images_train_batch_5.npy
      images_train_batch_6.npy
      images_train_batch_7.npy
      images_train_batch_8.npy
      labels_train_batch_0.npy
      labels_train_batch_1.npy
      labels_train_batch_2.npy
      labels_train_batch_3.npy
      labels_train_batch_4.npy
      labels_train_batch_5.npy
      labels_train_batch_6.npy
      labels_train_batch_7.npy
      labels_train_batch_8.npy
    models
```

Figure 9: Processed train and test batches of images and labels

9. Building and training the Convolutional Neural Network

The first step in building the CNN involves loading the batch data. A function is defined to load the image and label data from the processed dataset. The data is stored in NumPy arrays, and each batch is loaded sequentially.

Then, a sequential CNN model is defined. This model comprises several layers including convolutional layers (`Conv2D`), pooling layers (`MaxPooling2D`), dropout layers (`Dropout`), and dense layers (`Dense`). Each type of layer plays a specific role in the model:

- The Conv2D layers are used for feature extraction. The parameters are the number of output filters, the size of filters, padding, and activation function.
- MaxPooling2D layers are used for downsampling the input along its spatial dimensions (height and width).
- Dropout layers help prevent overfitting by randomly setting a fraction of input units to 0 at each update during training time.
- The Flatten layer flattens the input and prepares it for input into the Dense layer.
- Dense layers are the fully connected layers. The last Dense layer uses `'softmax'` activation function which is typically used in multi-class classification problems - it returns the probabilities of each class, with all probabilities summing up to 1.

The model is then compiled using the Adam optimizer with a specified learning rate, and the loss function used is `'sparse categorical crossentropy'`, which is suitable for multi-class classification problems. The model is trained batch-wise, by loading each batch and then feeding the images and labels to the model.

Finally, the trained model is saved to Google Drive with the filename 'model.h5'. Saving the model allows us to reuse the model later without needing to retrain it, which can be very time-consuming, particularly for large datasets or complex models. Once the training is complete, the model is saved for future use.

All I had to do afterwards was to play with `learning_rate` and `epochs`, so I can find which are the perfect values for perfect results.

After spending a lot of time in waiting for the training process and even paying for Collab Pro, so I can use more resources, I finally found the train, which led me to the desired results. It was epochs of **400** and a learning rate of **0.0001**. However these are not magical numbers, but there is a pretty good explanation about it.

The number of epochs refers to the number of times the entire training dataset is passed through the network during training. Increasing the number of epochs allows the model to learn from the data for a longer duration, potentially improving performance. However, using too many epochs can lead to overfitting, where the model becomes too specialized to the training data and performs poorly on unseen data. Therefore, the choice of 400 epochs was based on observing the model's performance and validating on test data and monitoring for signs of overfitting or convergence.

The learning rate determines the step size at which the model's parameters are updated during training. A higher learning rate allows for larger updates, which can lead to faster convergence but may also risk overshooting the optimal solution. On the other hand, a lower learning rate provides smaller updates, which can help the model to fine-tune its performance gradually. The value of 0.0001 for the learning rate suggests a relatively small update step, which is perfect for dealing with complex and diverse datasets.

10. The experiment

Before the results, it's good point to mentioned that I had tried more complicated models, but they didn't work. I tried:

1. Increasing the Complexity of the Model: While the current model is a good starting point, it might not be complex enough to handle the intricacies of a large dataset like Stanford Cars, which has 196 classes. Consider using a deeper model or even a pre-trained model such as ResNet50, VGG16, or EfficientNet. These architectures are designed to handle complex image datasets and might provide better results.

2. Data Augmentation: Data augmentation is a technique that can increase the diversity of your training data without actually collecting new data. Augmentation techniques such as flipping, rotation, scaling, etc., can make your model more robust to changes in the input data.

3. Using Early Stopping and Model Checkpointing: Early stopping allows you to stop training when the model performance no longer improves on a held out validation dataset. Model checkpointing allows you to save the model that achieves the best performance on the validation dataset. These techniques can help you avoid overfitting.

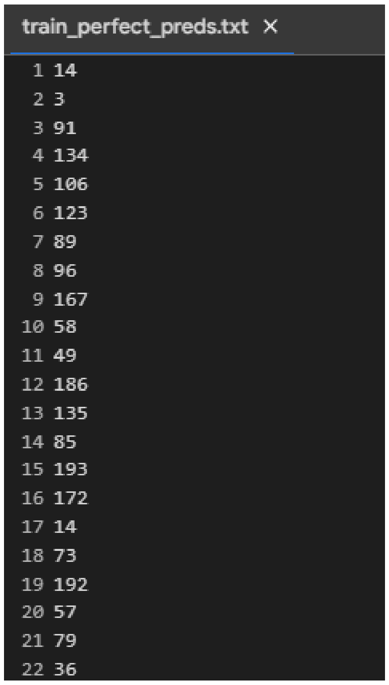
4. Using a Validation Set: Divide your training set into a training set and a validation set. The validation set is used during the training phase to get an unbiased estimate of the model skill on unseen data.

However, none of them gave an better output at the evaluation, but the important part is that I had try them.

Final Results

Since I was done with the training, it was time to do some evaluation on the model. I was planning to use the test images for this, but there was a problem – they didn't have any labels, so how could I possibly check the accuracy of my model?

If we take a look back at the README file of the dataset, we have a `train_perfect_preds.txt` file, which contains a column of number. Each row corresponds to the image to an image in the test dataset, for example 1st row number – the class number of the 1st image of the test dataset.

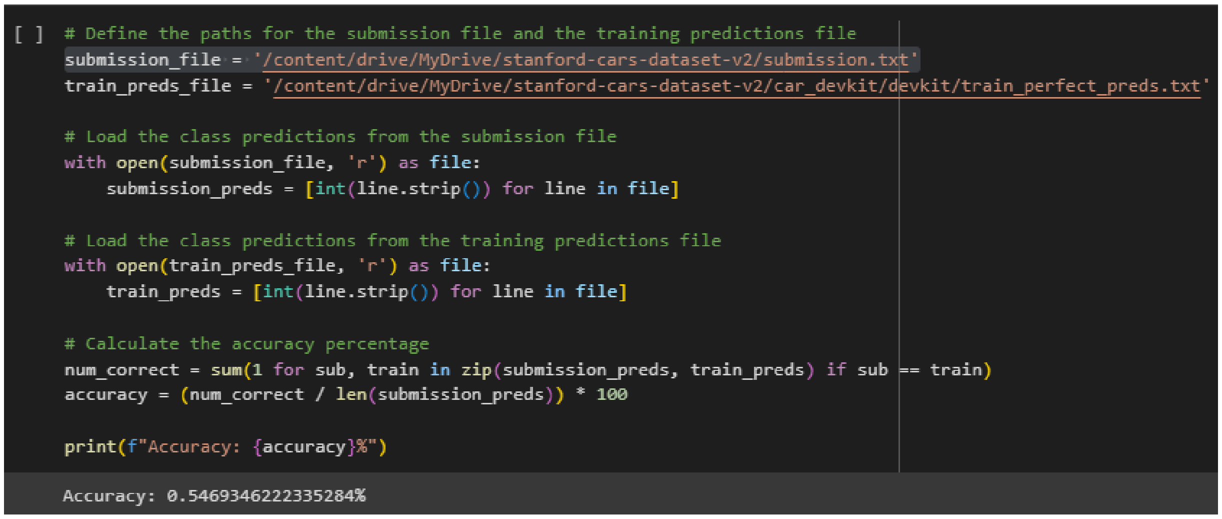

train_perfect_preds.txt

```
14
3
91
134
106
123
89
96
167
58
49
186
135
85
193
172
14
73
192
57
79
36
```

Figure 10: Snippet of train_perfect_peds.txt file

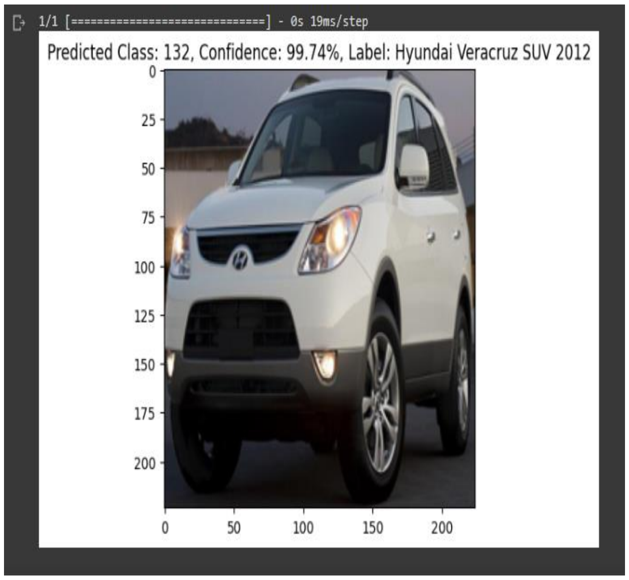
So, this suggests that I have to create the same file for my test images, and then from comparing them, check the accuracy. However, there was a problem. Earlier we mentioned that I had skipped images while we were processing them for training. The exact count was 134, and they were on random places, so I don't know are they sequential, which images are they exactly – etc. So even when I compare both files, I had accuracy around half percent, which was predictable.

Instead, I decide to generate a few times a random image from a random batch from test, and make the model make a prediction, on confidence level and the results turns out pretty good. Additionally, a quick google search confirmed those were the right model cars, so I has proved that my model is working.



```
[ ] # Define the paths for the submission file and the training predictions file
    submission_file = '/content/drive/MyDrive/stanford-cars-dataset-v2/submission.txt'
    train_preds_file = '/content/drive/MyDrive/stanford-cars-dataset-v2/car_devkit/devkit/train_perfect_preds.txt'

    # Load the class predictions from the submission file
    with open(submission_file, 'r') as file:
        submission_preds = [int(line.strip()) for line in file]

    # Load the class predictions from the training predictions file
    with open(train_preds_file, 'r') as file:
        train_preds = [int(line.strip()) for line in file]

    # Calculate the accuracy percentage
    num_correct = sum(1 for sub, train in zip(submission_preds, train_preds) if sub == train)
    accuracy = (num_correct / len(submission_preds)) * 100

    print(f"Accuracy: {accuracy}%")

    Accuracy: 0.5469346222335284%
```

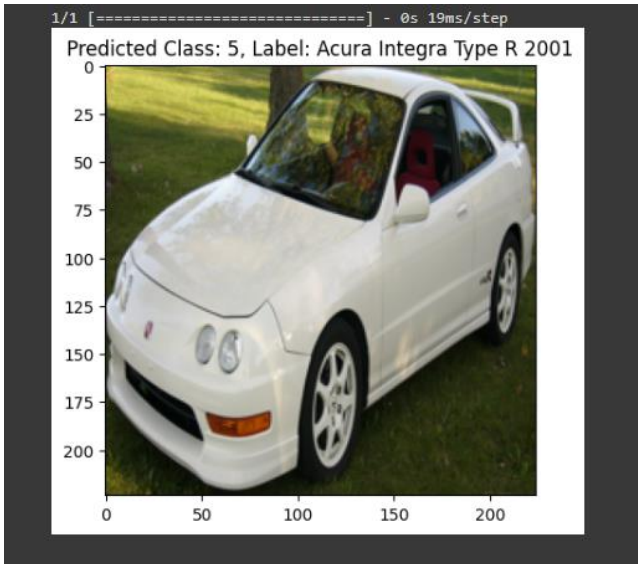
Figure 13: Accuracy of making the evaluation on file comparison



```
1/1 [==============================] - 0s 19ms/step
```

Predicted Class: 132, Confidence: 99.74%, Label: Hyundai Veracruz SUV 2012
0
25
50
75
100
125
150
175
200
0
50
100
150
200

Figure 12: Results 1



```
1/1 [==============================] - 0s 19ms/step
```

Figure 11: Results 2

Impact Assessment

Impact on Society

The primary purpose of our AI model is to serve as a digital tool for car enthusiasts and spotters, enhancing their experience by identifying unique and rare cars. However, its potential impact extends beyond hobbyist use. By automating the recognition of vehicle makes and models, this tool could serve professional purposes in the automotive industry for cataloguing, streamline claim processing tasks in insurance companies, and assist in law enforcement. This claim is based on the assertion from the International Journal of Computer Vision (2018) which highlighted the potential use of image recognition technology in such scenarios. However, we are mindful of the fact that the broader implications and practical implementation will need to be explored further.

Stakeholders

The immediate beneficiaries of this AI model would be car enthusiasts and spotters. Additionally, the automotive industry, car dealerships, and insurance companies could find this tool useful for various professional tasks. On a broader level, law enforcement agencies could potentially benefit from this technology in identifying vehicles related to criminal activities. While these outcomes hold promise, we are committed to ongoing engagement with all stakeholders to understand their needs and concerns better.

Privacy

The AI model aims to recognize cars, not individuals. However, the potential for misuse in tracking individuals through their vehicles does exist. I am cognizant of this risk and committed to establishing comprehensive usage guidelines that prioritize respect for privacy rights and strict adherence to legal norms.

Data

The AI model will be trained on the publicly accessible Stanford Car Dataset, comprising high-quality car images. I am committed to respecting privacy during data handling. The dataset will be reviewed to ensure it does not unintentionally contain personally identifiable information such as unique license plates or individual identities.

Transparency

I believe transparency is key to establishing trust. A comprehensive report outlining our data preprocessing, model training, transfer learning techniques, and model evaluation will be available for review. This document will provide stakeholders with insights into the model's development and operation.

Inclusivity

While inclusivity in our project refers to considering diverse types and models of cars, I also understand its importance in not excluding groups of people. Even though my project will be mostly used by car enthusiasts, my goal is to develop a model that functions effectively for all users regardless of their location, language, or other potentially excluding factors. I am committed to regular reviews and updates to the model to ensure it remains inclusive.

Sustainability

To ensure the project's sustainability, I will regularly update the model to recognize new car models and make the necessary upgrades. From an environmental perspective, I aim to minimize the model's energy footprint by optimizing computational efficiency and selecting sustainable infrastructure for model training and deployment. By focusing on these impact areas, I can balance the realization of benefits with risk mitigation, leading to a tool that contributes positively to society. The primary objective of this project is to provide valuable assistance to individuals, but it is crucial to acknowledge that in the wrong hands, any technology can potentially be misused. Therefore, I am fully committed to taking proactive measures to prevent such misuse and ensure that the tool is utilized exclusively for lawful and ethical purposes. By doing so, I aim to create a positive societal contribution that upholds the values of integrity and responsibility.

Recommendation

The completed car recognition system exhibits promising capabilities, with a room for future enhancements. One such area of improvement could be the integration of logo recognition functionality, which would assist in determining the make of a car before identifying the model. Machine learning models trained for logo detection and recognition could help achieve this, as they can locate and identify logos from car images.

Such an addition could enhance the system's accuracy and reliability in car identification tasks, especially in cases where the overall design or structure of a car is not sufficiently distinctive but the logo is clearly visible. However, the challenge lies in acquiring a sufficient amount of data to train the model for logo recognition. A diverse dataset encompassing a wide variety of logos, photographed under different conditions and angles, would be required to ensure robust performance.

In terms of practical applications, the car recognition system could be an instrumental tool in traffic management and law enforcement. By incorporating license plate recognition, traffic authorities could use the system to monitor and manage traffic, identify stolen vehicles, or enforce traffic laws more efficiently. Moreover, the integration of real-time video processing could extend the system's utility to live traffic surveillance and automatic ticketing for traffic violations.

For law enforcement agencies, such a system could prove invaluable for identifying vehicles involved in criminal activities. This could potentially aid in investigations, criminal identification, and overall public safety. However, it's crucial to ensure the ethical use of this technology, taking into account privacy concerns and legislative regulations.

Conclusion

In conclusion, this project was a roller-coaster ride, filled with many ups and downs. There were plenty of challenges, but each obstacle provided a learning opportunity, helping me grow in my understanding of machine learning, data handling and project management. One of the aspects which I had never thought about before was the societal impact of every project, but from this after this semester, I understood how important it is. My personal skills, such as leadership and reporting went to another level with my projects, so I can say they are getting professional. Because of my work, it was hard for to manage the time for university, work and free time, but I am glad that I managed to do it at the end! The experience of working with real-world data was invaluable, and each problem encountered refined my problem-solving skills.

Throughout the journey, I was able to practically apply the knowledge I gained from my studies, making the project incredibly fulfilling. The process was challenging, but the thrill of seeing the car recognition system accurately identifying car makes and models was worth the effort.

I am incredibly grateful for the guidance and support I received from AI consultants. Their advice and expertise played a vital role in the successful completion of this project. Although the project is now completed, it doesn't signal the end of my learning journey. Rather, it has fuelled my desire to continue exploring and innovating in the field of machine learning. I hope I see you at AI-Advanced!