Image Classification of Rocket League Cars using Transfer Learning and Data Augmentation

Introduction

This project revolves around classifying a set of different cars from the video game "Rocket League" using transfer learning with the ResNet18 network architecture. The idea was to create a model which would be able to detect the type of Rocket League car in a given screenshot or image. The reason why this is harder than it may appear is due to the customisation aspect of the cars, although they all have a different and specific shape, they may be customised with different antennas, toppers, boosts, wheels, and decals/colours which all may interfere with the model.

Figure 1: Octane car with custom antenna, topper, decal, wheels, and boost

Related Work

Although there are not any specific Artificial Intelligence image classifiers for Rocket League cars, I have found research done into image classification for real life car models all of which use a range of networks from pretrained models such as ResNet34 (Seng, 2019), MobileNet, VGG-16 and EfficientNet (Krasniqi, 2021) or creating their own deeply trained convolutional neural network with the use of VGG-16 (Shah, 2021).

Seng (2019) used the Stanford car dataset through the use of the Kaggle, which is an online community platform that allows for users to find and publish datasets. Through the use of that dataset, they had then used the pre-trained model ResNet34 using a learning rate of 0.01 and 10 epochs which resulted in a 77.72% validation accuracy.

Krasniqi (2021) had also incorporated the Stanford car dataset into their project. However, they had used three different deep learning models and compared the differences between them, using different parameters that they had found most optimal such as a learning rate of 0.0001 for MobileNetV2 and 0.01 for EfficientNetB1. Krasniqi (2021) concluded that the EfficientNetB1 was the best model for vehicle recognition with a 71% accuracy on the validation set.

Shah (2021) trained their model using Keras, a neural network API for Python that is integrated with TensorFlow, which is another piece of software which is used to build machine learning models. Shah had decided to also use the Stanford University car dataset due to the number of images and

also the angles in which the pictures were taken being similar. An SSD algorithm was used for this paper's classification model training, the SSD algorithm is designed for object detection in real-time by using a region proposal network in order to create boundary boxes around objects and then utilizing them in order to classify the objects. In this case, using the VGG16 model in order to train the network.

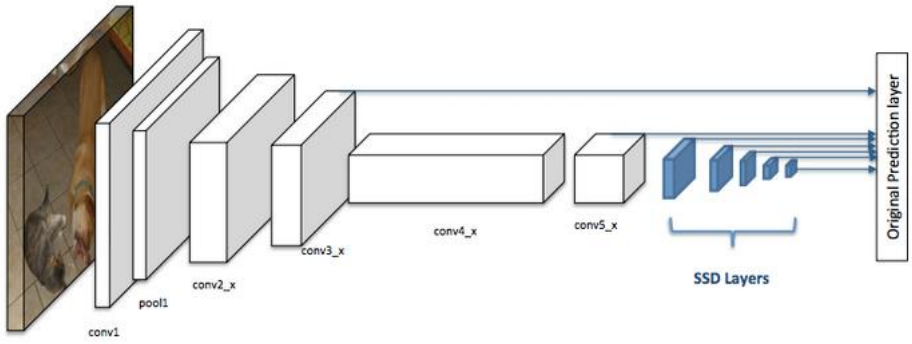

Figure 2: Architecture of a convolutional neural network with an SSD detector (ArcGIS, n.d.)

Figure 3: Example of an SSD algorithm being used to classify a taxi and jeep

Methods Used

Dataset Creation

The images that I had gathered to place into my dataset were all downloaded via an image scraper chromium extension. This allowed me to download hundreds of images from Google, the number of images per class totalled to around 300 images.

However, the main issue with the images found on google is that they are not designed to fit well within an AI image classification model, therefore, I had to come up with certain criteria to ensure that I was able to use them.

Firstly, the images could not be smaller than 224x224 as otherwise it would not be optimal for the ResNet18 model to train with otherwise.

Secondly, it was important that the images did not contain watermarks that could distress and confuse the model during training. On a similar note, images that were not taken from the game were also not included to ensure that only cars that could be seen in the game were used to train the model. For example, 2D or pixelated drawings.

Figure 4: Example of an image that does not fit the image collation criteria

Third, I ensured that no duplicates of the exact same image were found in the dataset as this would increase the time taken to train and validate the model unnecessarily.

Lastly, having the car being centred within the image was preferable as this would allow for easier data transformation and augmentation.

Figure 5: Example of an image that fits the image collation criteria

Data Augmentation

As the cars in Rocket League have the ability to rotate, flip, jump and fly, it is very important to ensure that the model is trained on a variety of images. However, due to the fact that the images from Google are not heavily varied in terms of angles, I have used the PyTorch torchvision transforms module in order to help with this. These are the following functions that I used:

```
RandomRotation(5)

CenterCrop(224)

RandomHorizontalFlip()

Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
```

Artificial Neural Networks

An Artificial Neural Network (ANN) is a computation model that is based on a collection of connected nodes which loosely models the neurons in a biological brain, each connected node is able to communicate with each other by transmitting signals.

A common ANN structure would include an input layer, hidden layers, and an output layer. With this approach, once we have loaded the input to the input layer, these values will be distributed to the hidden layer nodes which will then make a random change to the value and consider how these changes affects the final output and whether or not that adjustment was positive or negative towards the goal. Deep learning would typically refer to the stacking of many hidden layers.

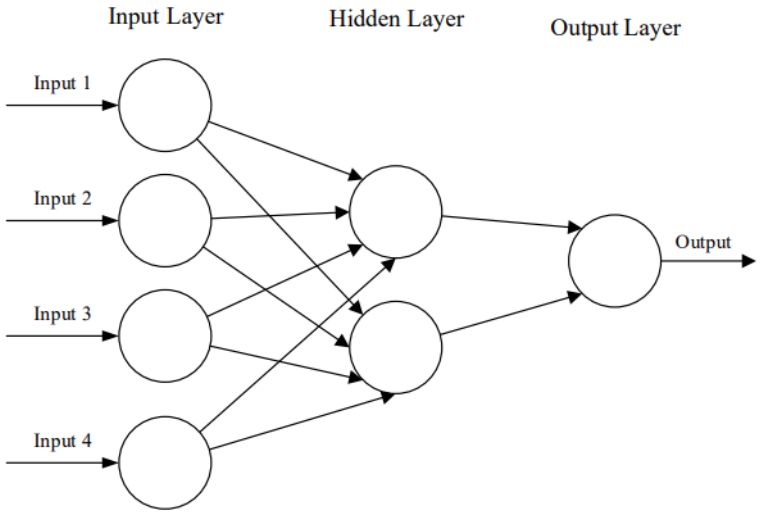

Figure 6: Basic structure of an ANN (Keiron O'Shea et al.)

Convolutional Neural Network

A Convolutional Neural Network (CNN) is very similar to a traditional ANN as they also include nodes that learn how to optimise themselves through learning. However, the main difference is that CNNs are mainly used in pattern recognition within images and therefore have an architecture that reflects this.

Three types of layers are included in a CNN. Convolutional layers, pooling layers, and fully connected layers.

The convolutional layer, as the name suggests, convolves two sets of information together. For the CNN, this would mean that the input data is filtered and produces a feature map.

In the pooling layer, down-sampling occurs in order to reduce complexity for the further layers, this means that the feature map's height and width are reduced which allows for faster computational speeds and ensures translation equivariance.

The fully connected layer will carry out the same task found in ANNs and will generate classification scores based on the activations for use in classification. Keiron O'Shea (2015) also suggests the use of a rectified linear unity (ReLu) in order to improve performance.

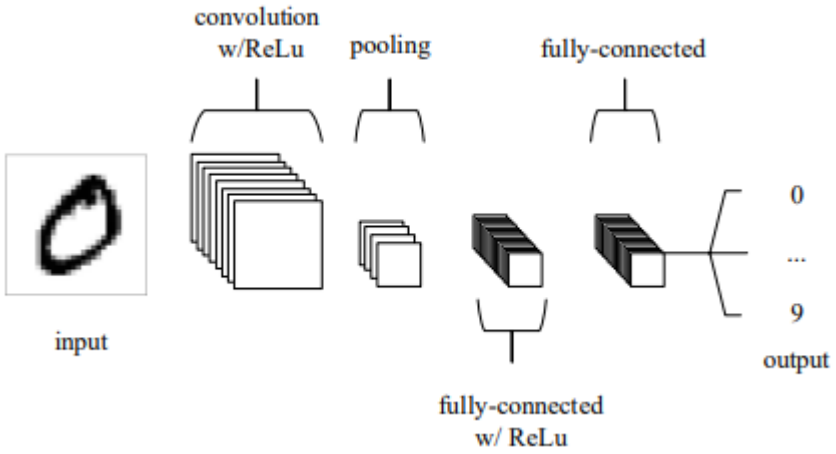

Figure 7: Example CNN architecture (Keiron O'Shea et al.)

ResNet18

Transfer learning can be defined as a machine learning method where a model that has been trained and developed for a task, serves as the basis for a model on a second task. A simple example could be having a classifier trained on recognising different foods and using its knowledge in order to recognise drinks.

Using transfer learning greatly decreases the processing and training time that would be needed to train a new CNN, in addition to typically resulting in higher accuracy.

For this project, I have used ResNet18. It is a CNN that is 18 layers deep and as it has been trained on ImageNet, an image database with over 14 million images, it is able to classify images into 1000 object categories.

Technical Implementation

Setup

In order to run and write the code used for this classification model, I have used Google Colab with Python and the Pytorch library, which is a machine learning framework based on the Torch library. The reasoning behind using Google Colab is due to the many features that it provides such as free access to a GPU that is able to be ran for up to 12 hours, cloud storage and automatic version control.

From my testing, I have found that training my model for 50 epochs with a learning rate of 0.0001 and a batch size of 8 had provided the best results. The reasoning for the low batch size is due to the small dataset that I am using.

Code

The beginning section of the notebook sets up all of the libraries, checks if a GPU is available, and if it is, ensures that the model is trained on that in order to increase speed of training. Some miscellaneous set up such as creating a function to display images and mount the google drive to allow access to the dataset and necessary files.

Afterwards, the training dataset was Randomly rotated and flipped horizontally in order to introduce more variety as explained earlier in this paper. The validation and testing dataset were both cropped and normalized in order to ensure it matched the requirements for the ResNet18 network.

Then, the pretrained model was loaded and modified to ensure that the final layer had 3 classes to output as this is the number of different cars I am classifying. The cross-entropy loss is then specified as the loss function with a learning rate of 0.0001 as this provided the best stability and accuracy in addition to having the number of epochs being 50, allowing for the model to learn long enough to reach a high accuracy.

Once everything is set, the model can start the training and validation process. Every epoch, the model is trained and then consequently validated using torch and Pytorch functions. Every time that the validation loss decreases, the model is saved at its current state in a google drive folder as this means that the accuracy of classification is higher.

```
Epoch: 1        Training Loss: 0.147814         Validation Loss: 0.111080
Validation loss decreased (inf --> 0.111080).  Saving model ...
Epoch: 2        Training Loss: 0.112730         Validation Loss: 0.067614
Validation loss decreased (0.111080 --> 0.067614).  Saving model ...
Epoch: 3        Training Loss: 0.110607         Validation Loss: 0.034386
Validation loss decreased (0.067614 --> 0.034386).  Saving model ...
Epoch: 4        Training Loss: 0.089502         Validation Loss: 0.020816
Validation loss decreased (0.034386 --> 0.020816).  Saving model ...
Epoch: 5        Training Loss: 0.055740         Validation Loss: 0.022114
Epoch: 6        Training Loss: 0.040971         Validation Loss: 0.035117
Epoch: 7        Training Loss: 0.060863         Validation Loss: 0.033370
Epoch: 8        Training Loss: 0.058187         Validation Loss: 0.019367
Validation loss decreased (0.020816 --> 0.019367).  Saving model ...
Epoch: 9        Training Loss: 0.023274         Validation Loss: 0.053699
Epoch: 10       Training Loss: 0.064338         Validation Loss: 0.029913
Epoch: 11       Training Loss: 0.065793         Validation Loss: 0.033646
Epoch: 12       Training Loss: 0.079945         Validation Loss: 0.122555
Epoch: 13       Training Loss: 0.078867         Validation Loss: 0.029786
Epoch: 14       Training Loss: 0.042761         Validation Loss: 0.030786
Epoch: 15       Training Loss: 0.028992         Validation Loss: 0.021254
Epoch: 16       Training Loss: 0.036656         Validation Loss: 0.045974
Epoch: 17       Training Loss: 0.036042         Validation Loss: 0.008807
Validation loss decreased (0.019367 --> 0.008807).  Saving model ...
Epoch: 18       Training Loss: 0.014273         Validation Loss: 0.005528
Validation loss decreased (0.008807 --> 0.005528).  Saving model ...
Epoch: 19       Training Loss: 0.020065         Validation Loss: 0.009783
Epoch: 20       Training Loss: 0.019629         Validation Loss: 0.008952
```

Figure 8: Visualisation of the training and validation loss changes per epoch

Results and Discussion

Visualisation

Through the use of t-distributed stochastic neighbour embedding (T-SNE) and GradCAM we are able to understand how the model identifies the different images and whether or not it had difficulty grouping certain classes.

T-SNE is a statistical technique that places each datapoint on, in this case, a two-dimensional map in order to visualise high-dimensional data.

GradCAM on the other hand, allows us to gain a better understanding of the model through the use of a heatmap with the warmer areas being the parts in which produced a higher classification score.

T-SNE

As shown in Figure 9, the base ResNet18 model, did not perform well on my Rocket League cars dataset which is to be expected due to the fact that it was trained on cars found in real life, although the graph does not appear to be completely random, as some classes are grouped together loosely.

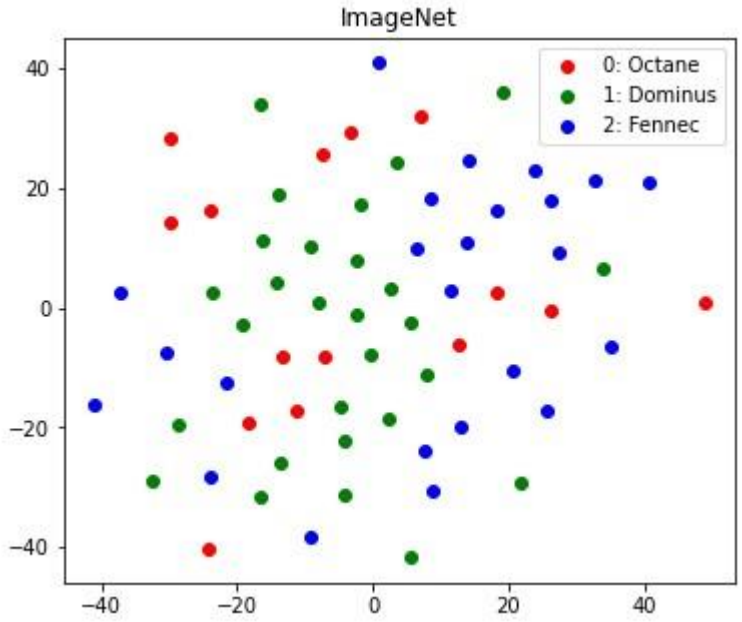

Figure 9: T-SNE map of the ResNet18 model only trained on ImageNet trying to identify Rocket League cars

However, with my transfer learning model that is based off the ResNet18 and trained on my Rocket League car dataset, you can see a clear difference and each class is grouped into their own section. With the only possible confusion being between the Fennec and Octane classes, yet not big enough to cause any issues.

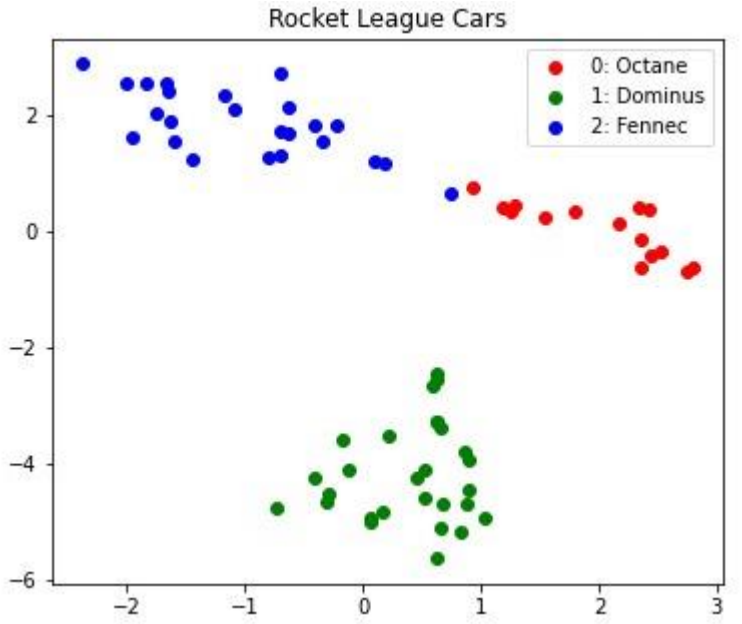

Figure 10: T-SNE map of the ResNet18 model trained through transfer learning on the Rocket League cars dataset

GradCAM

As seen in the figures 11, 12, 13 and 14, it appears that the model generally views the rear of the car as an important section for classification. However, this is not the case as much for the Fennec class as it appears that the model is unsure where exactly the best classification score is and is not as consistent.

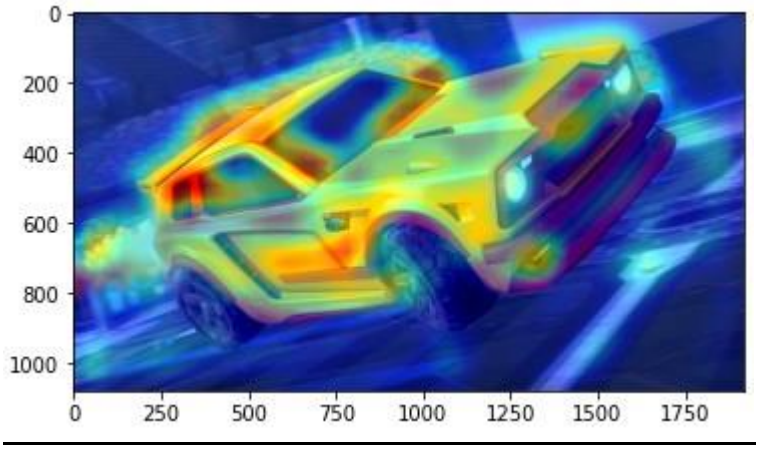

Figure 11: GradCAM heatmap of the Fennec car

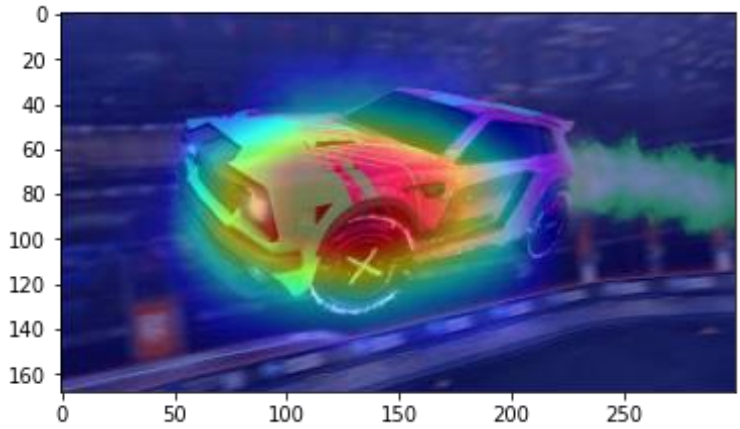

Figure 12: GradCAM heatmap of the fennec car with the warmer area being at the front of the car

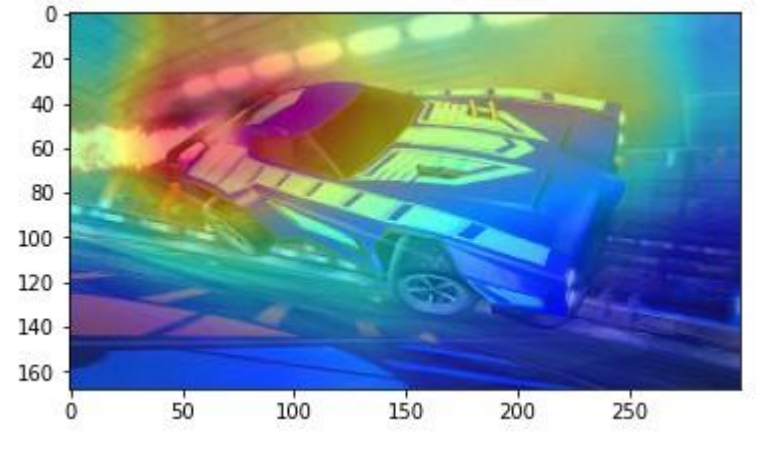

Figure 13: GradCAM heatmap of the Dominus car

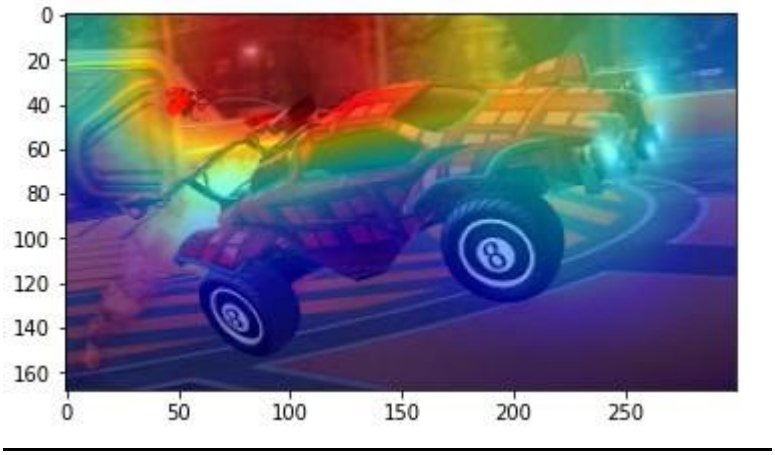

Figure 14: GradCAM heatmap of the Octane car

Model Accuracy

I have conducted various tests in regard to the training of the model in order to achieve the highest accuracy possible.

	Parameters			
Test No.	Batch Size	Epoch count	Learning Rate	Test Accuracy (Overall)
1	8	20	0.001	97%
2	8	20	0.0001	98%
3	8	20	0.00001	98%
4	8	50	0.0001	100%
5	8	100	0.0001	100%

From the beginning, the model performed very well even with parameters that were not modified with greatly, resulting in a 97% total accuracy. The reason for the missing 3% is due to a mis-classification of one Dominus and Octane class image.

As you can see in Figure 13, the validation loss loosely follows the training loss line albeit a few spikes.

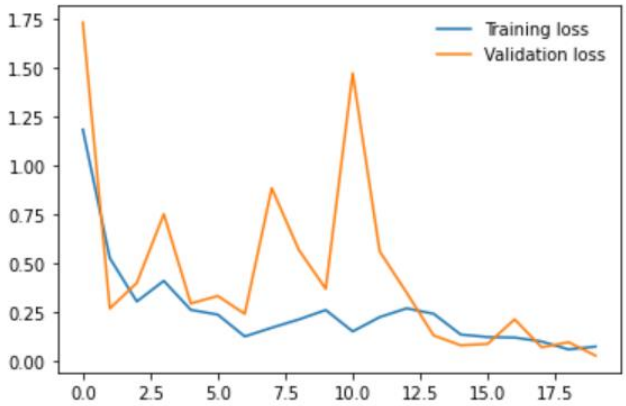

Figure 13: Test 1 training and validation loss graph

With the best parameters being from test number 4, the graph appears to be a lot smoother in terms of the validation and training loss lines following closely. As the loss was very low to begin with, the spikes from very small changes are more evident.

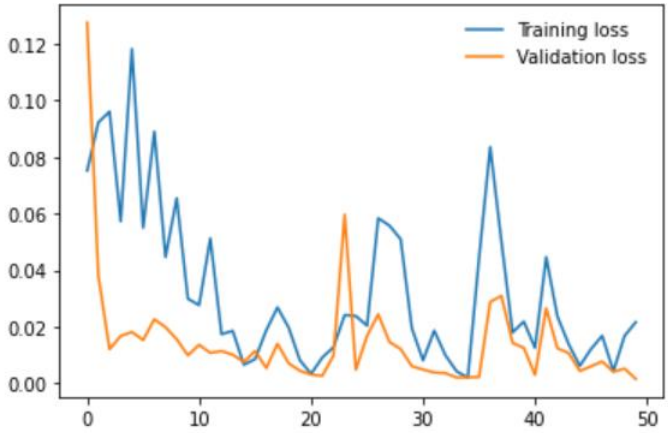

Figure 14: Test 4 training and validation loss graph

Application

For an in-game application, as the cars are not all available to the player in the beginning of the game and are only unlocked as they play more games or buy them, there is no way to find out what cars the other players are using when playing a match. With this model, they would be able to take a screenshot of the game with the car in view and find out what the car is.

Conclusions and suggestions for future work

In conclusion, it is clear that through the use of Transfer Learning and the ResNet18 pretrained model, Rocket League cars are able to be classified with certainty even when the car is customised, and the background is different from image to image.

Although this is a specific use case, due to the size of gaming as a whole, I believe that models like these can aid in promotional/advertising material. For example, as the size of the Rocket League esports scene is very large with over 368,721 all-time peak viewers in the RLCS 2021-22 World Championship (Charts, 2022), another application that may be applicable for this model is to allow people to draw and submit fan art or a doodle of a car and have the application classify that picture, similar to “Quick, Draw!” from Google.

Due to time limitations, I was unable to fully explore the drawn picture route. However, I had drawn a picture of an octane using Paint and once I had fed this image (Figure 15) to the model, it had no issues with identifying the image. From this, I believe with further development, it would be possible to create a web app that would allow users to draw and submit it to the application which would then use the model in order to classify it.

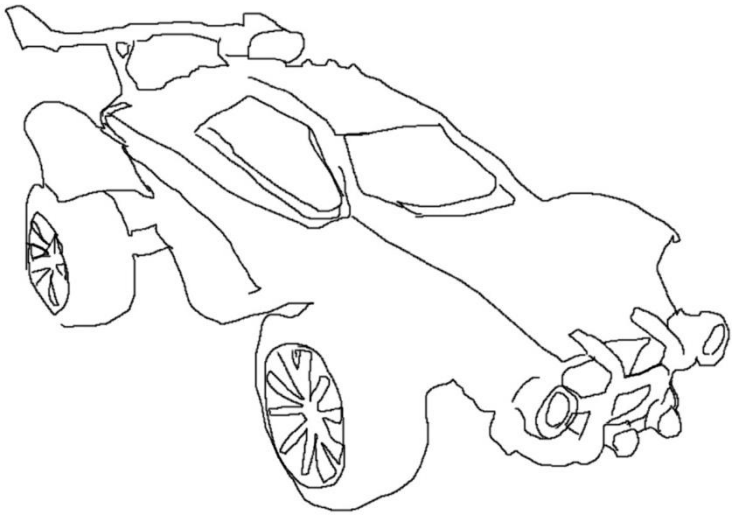

Figure 15: Self-made drawing of an Octane car

References

Albawi, S., Mohammed, T. A. & Al-Zawi, S., 2017. Understanding of a Convolutional Neural Network. *International Conference on Engineering and Technology,* pp. 1-6.

ArcGIS, n.d. *How single-shot detector (SSD) works?.* [Online]
Available at: https://developers.arcgis.com/python/guide/how-ssd-works/
[Accessed December 2022].

Charts, E., 2022. *Esports statistics Rocket League.* [Online]
Available at: https://escharts.com/games/rl
[Accessed December 2022].

Krasniqi, A., 2021. *Vehicle Classification using Machine Learning.* [Online]
Available at: https://medium.com/@albionkrasniqi22_80133/vehicle-classification-742403117f43
[Accessed December 2022].

O'Shea, K. & Nash, R., 2015. *An Introduction to Convolutional Neural Networks,* Aberystwyth: arXiv.

Seng, W., 2019. *Car Model Classification.* [Online]
Available at: https://towardsdatascience.com/car-model-classification-e1ff09573f4f
[Accessed December 2022].

Shah, D., 2021. Car Image Classification and Recognition. *International Journal for Research in Applied Science and Engineering Technology,* 9(9), pp. 2096-2101.