Fine Grained Vehicle Classification

A Final Thesis Report Presented in Partial Fulfillment of the Requirements for EGR 598 – Connected and Automated Vehicles

Manohar Akula

ARIZONA STATE UNIVERSITY

November 2022

ABSTRACT

A key task in the development of an intelligent traffic system has been the identification of a vehicle's category based only on footage from security cameras. There are obstacles to fine-grained automobile recognition partly because photographs of different classes of cars sometimes look quite like one another. In addition, unlike other types of object identification, automobile recognition has a huge number of car models, which makes fine-grained car recognition both crucial and difficult. Training a deep convolutional neural network (DCNN) on a huge dataset has been the focus of recent studies. These approaches often just consider automobiles, which might lead to subpar results when trying to train a deep convolutional neural network (DCNN) to integrate data from a variety of sources. To tackle this problem, we conduct in-depth evaluations of various automobile features, with special emphasis on the significance of vehicle fronts in fine-grained automobile recognition. This project aims to train and evaluate a multi-class classification model to classify distinct vehicle types and well-known vehicle models using images from Carvana, a Phoenix-based online used car store.

This will be performed by using a web scraping algorithm that will retrieve and label images of a car model from the Carvana website from various viewpoints. Carvana features 64 distinct view angles of a certain vehicle model on average. This allows for the development of a dataset divided into train, test, and validation segments, which may then be used to train a classifier. This dataset will be used to develop and train multiple CNN models, which will then be fine-tuned to improve precision. This model's accuracy will then be tested on validation images to determine which one performs the best. The completed model may then be used to precisely classify the vehicle model and type.

Keywords: VGG16, VGG11, InceptionV4, & Beautiful soap.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Fine-grained automobile recognition has garnered a lot of interest considering the advancements in computer vision and intelligent traffic. Determine the model information of an automobile using fine-grained car recognition [1]. Applications include parking management, forensics, and traffic security, among many other [2]. A precise automobile recognition system may be of utmost significance. However, due to several reasons, reliably identifying automobile model information is a challenging computer vision problem. Furthermore, fine-grained automobile detection is tough because to the abundance of (a few thousand) car models, which most other object recognition tasks lack [3]. Even though some automobiles have a similar look, they fall into distinct categories based on the manufacturer or year of production. On the other hand, because to external decorating, positions, and lighting variations, among other factors, vehicles within the same category may differ greatly from one another. All the problems might negatively affect accuracy.

Recently, a few techniques for handling fine-grained automobile identification have been created. Building reliable feature representations for automobiles using techniques like SIFT [4], Gabor filter [5], etc., is a typical strategy. These manually produced conventional descriptors frequently fall short of giving features enough discriminative data. Parts-based pooling techniques [6], which frequently involve two-step processes, are the subject of another area of research. These methods' initial goal is to extract features from various discriminative sections. To aggregate these characteristics into a single, robust representation, a few pooling algorithms are then developed [7], etc. Due to its impressive performance on several computer vision issues, Deep Convolution Neural Network (DCNN) [8] has recently attracted a lot of interest. Fine-grained automobile recognition has shown some degree of success with DCNN [9]. Large-scale automobile dataset and some baseline findings were published by Yang et al. [10]. Spatial weighted pooling, which Hu et al. [11] proposed, significantly improves the resilience and efficiency of feature representations for fine-grained automobile detection. A bilinear DCNN model with two feature extractors based on DCNNs was proposed by Lin et al. [12]. The outputs of these two DCNNs are combined into a single picture description by utilizing element-wise product to multiply their outputs. It had high performance for fine-grained recognition and translated-invariantly modeled local paired feature interactions. By learning a bank of convolutional filters that capture class-specific discriminative patches without the need for additional part or bounding box annotations, Wang et al. [12] demonstrated how mid-level representation learning may be improved inside the CNN framework for fine-grained recognition. To enhance the performance of fine-grained recognition, Zheng et al. [13] suggested a unique part learning strategy using a multi-attention convolutional neural network, where part creation and feature learning may reinforce one another. A progressive-attention model called PA-CNN [14] utilizes attention data to accurately identify pieces without the use of a bounding box. All the approaches looked at the issue of fine-grained recognition from various angles. Additionally,

other DCNN-based approaches have been presented [15] to address a variety of vehicle-related issues. These approaches have shown promising results and received a lot of attention.

The objective of this project was to support the team working on the Camera/Lidar fusion base algorithm in improving their vehicle model and detection performance. Additionally, this research seeks to train and evaluate a multi-class classification model to identify various vehicle types and well-known vehicle models using photos from Carvana, an online dealer of used vehicles with headquarters in Phoenix.

This was done by creating a web scraping algorithm to extract car images from the Carvana website from different view angles and label these images to support the metrics team for localization and traffic scene reconstruction framework using infrastructure cameras, dubbed as CAROM, [16] i.e., "CARs On the Map." For traffic scene reconstruction and replay, CAROM analyses traffic monitoring footage and transforms them into anonymous data structures of vehicle type, 3D shapes, location, and velocity.

LITERATURE REVIEW

Major Theories of the Field

- VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION https://arxiv.org/pdf/1409.1556.pdf
- Multi-Path Deep CNNs for Fine-Grained Car Recognition https://ieeexplore.ieee.org/document/9140403/
- Automated Brain Image Classification Based on VGG-16 and Transfer Learning https://link-springer-com.ezproxy1.lib.asu.edu/content/pdf/10.1007/s00138-020-01069-2.pdf

Important Theorists

- Karen Simonyan ∗ & Andrew Zisserman + Visual Geometry Group, Department of Engineering Science, University of Oxford {karen,az}@robots.ox.ac.uk
- Huibing Wang; Jinjia Peng; Yanzhu Zhao; Xianping Fu
- Taranjit Kaur1 · Tapan Kumar Gandhi1

Thoughts and Arguments

More modern approaches have been developed in the literature that take advantage of optimized machine learning techniques. These approaches include Particle Swarm Optimization (PSO) [17] , Biogeography-based Optimization (BBO) [17], Artificial Bee Colony (ABC), BAT [18] and their hybridizations [19], etc. These approaches combine machine learning algorithms with optimization techniques. In addition to these, several additional studies have suggested the categorization using wavelet-based feature extraction approaches [17]

The field of machine learning has recently seen a rise in the use of deep learning methods like convolutional neural networks (CNN). By parameterizing the convolutional ('conv') and pooling ('pool') layers, CNN or Deep CNN (DCNN) may automatically extract the image's features. Many applications [20] have used CNN models with success. The amount of training data utilized, and its quantity have a significant impact on CNN's categorization abilities. The CNN begins to overfit when the dataset size is small. Transfer learning has been a popular idea in this situation and has been shown to be successful for classification problems [21]. Only the last layers are trained with data from the more recent courses in transfer learning, using the 'conv' layer weights from previously learned models. In this research, the categorization of vehicle images using the well-known pre-trained CNN's model VGG-16 [[22] has been examined.

METHODOLOGY

Web scraping

Most of the information in the world is unstructured, with estimates ranging from 80 percent to 90 percent. Photos, papers, tapes, videos, and even text on the internet all qualify. Extracting the information inside and identifying patterns/drawing meaningful insights is necessary for making use of the data. But the question is how to transform the data from its current unorganized state. And here is when web scraping comes in handy. [23]

Web scraping, also known as web harvesting or web data extraction, is the automated process of extracting huge amounts of data from websites. The information related to specified sites, or just certain information, may be extracted by the user. Data may be organized for storage and later use.

Web scraping has come into its own in the modern day, and it may be used in many different contexts. To name only a few examples:

- An Evaluation of Public Opinion as Reflected in social media
- The Role of Lead Generation in the Marketing Industry
- Online price comparisons and market research in the field of electronic commerce

Train and test data collection for machine learning applications

- Web scraping entails the following procedures:
- Locate the webpage's URL that you want to scrape.
- Pick out the specific components by closely examining
- create the necessary code to get the information from the specified items.
- Save information in the correct format

Popular online scraping frameworks and technologies include: 1. Selenium, a framework for testing websites and web applications. Markup languages such as HTML and XML may have their data extracted with the help of a Python package called BeautifulSoup. Pandas is a data-analysis and manipulation package for the Python programming language [23].

To do this task, we utilized Beautifulsoap. Python's Beautiful Soup provides a module for extracting data from HTML and XML documents using the language. It works in conjunction with the parser of your choice to give idiomatic means of traversing, searching, and altering the parse tree. Programmers often report time savings of many hours or even days as a result.

ANN

Computational processing systems known as artificial neural networks (ANNs) take a significant amount of inspiration from the way organic nerve systems (such as the human brain) carry out their functions. ANNs are primarily made up of a large number of linked computational nodes, which are referred to as neurons. These neurons work together in a distributed manner to collectively learn from the input in order to optimize the ultimate output of the ANN [24].

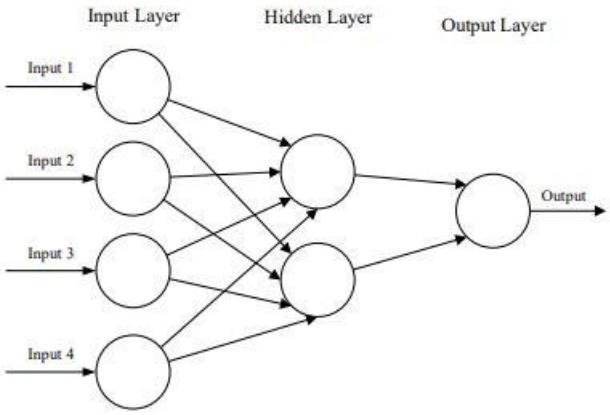

Figure 1. A basic feedforward neural network (FNN) has only three layers: input, hidden, and output.
Source: (https://arxiv.org/pdf/1511.08458.pdf)

Figure 1 presents one possible representation of the fundamental architecture of an ANN. To begin, we would load the input, which would typically take the shape of a multidimensional vector, into the input layer, which would then distribute it to the hidden layers. The process of learning is referred to as the hidden layers making judgments based on the information provided by the preceding layer and determining whether an internal stochastic change has a positive or negative impact on the output of the system. Deep learning refers to a kind of learning that involves numerous hidden layers piled one on top of the other.

Unsupervised learning

Unsupervised learning is differentiated from supervised learning in that the training set does not include any labels. In most cases, success is measured by whether the network is able to decrease or raise a cost function that is connected with it. On the other hand, it is essential to keep in mind that the vast majority of image-centered pattern-recognition tasks often rely on categorization in the form of supervised learning.

CNN

Like standard artificial neural networks (ANNs), convolutional neural networks (CNNs) are made up of neurons that may improve themselves via the process of learning. Each neuron will still take an input and carry out an operation (such as a scalar product followed by a non-linear function), which is the foundation of numerous artificial neural networks (ANNs). From the first raw picture vectors that are fed into the network all the way to the final class score that is output, the whole network will still express a single perceptual scoring function (the weight) [24] .

The final layer will include loss functions connected with the classes, and all the standard tips and techniques that have been created for conventional ANNs will still be applicable. The only significant distinction that can be made between CNNs and more conventional artificial neural networks (ANNs) is that CNNs are predominantly used around pattern detection inside pictures. Because of this, we can embed image-specific properties into the architecture, which makes the network more suited for image-focused tasks. At the same time, the number of parameters needed to build up the model is reduced even further. The tendency of older kinds of ANN to struggle with the computational complexity necessary to calculate picture data is one of the most significant drawbacks associated with these types of ANN.

Due to the comparatively low picture dimensionality of merely 28 by 28, standard machine learning benchmarking datasets, such as the MNIST database of handwritten digits, are appropriate for most kinds of artificial neural networks (ANN). With this dataset, a single neuron in the first hidden layer will hold 784 weights (28281) where 1 keep in mind that MNIST is normalized to simply black and white values), which is doable for most kinds of ANN.

When a more substantial colored picture input of 64 by 64 pixels is taken into consideration, the number of weights that may be attached to a single neuron in the first layer significantly rises to 12, 288. If you additionally consider the fact that the network will need to be a lot bigger to cope with this scale of input than the one that was used to identify color normalized MNIST digits, you will be able to comprehend the limitations of utilizing such models.

CNN Architecture

Three different kinds of layers make up CNNs. These consist of fully linked, pooling, and convolutional layers. A CNN architecture results from the stacking of these layers. Figure 2 depicts a streamlined CNN architecture for MNIST categorization [24].

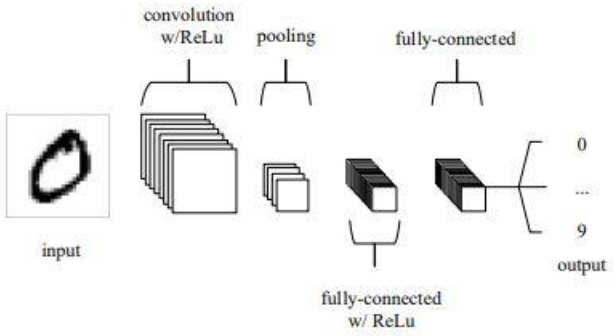

Figure 2. A simple CNN architecture comprised of just five layers.
Source: (https://arxiv.org/pdf/1511.08458.pdf)

There are four main components to the sample CNN's fundamental functioning.

1. Similar to other ANN types, the input layer will store the image's pixel values.

2. The convolutional layer will calculate the scalar product between the neurons' weights and the area linked to the input volume to identify the output of neurons whose local areas of the input are related to them. The rectified linear unit (often abbreviated to ReLu) tries to apply a "elementwise" activation function, such as sigmoid, to the output of the activation generated by the preceding layer.

3. After that, the pooling layer will simply downscale the input along the spatial dimensions, thus lowering the number of parameters in that activation.

4. After that, the fully linked layers will carry out the same tasks as ordinary ANNs and try to create class scores from the activations, which may be utilized for classification. ReLu is another option that is proposed for usage between these layers in order to boost performance.

Transfer learning with dense convolutional neural networks

Deep Neural Networks (DNNs), also known as Deep Learning (DL), utilize deep NN architectures to automatically learn hierarchy of features from raw input data without the requirement for feature engineering [16]. Deep learning methods are characterized by deep architectures with multiple hidden layers that allow them to learn many levels of abstraction, as opposed to shallow architectures with only one or two hidden layers [35]. These methods are loosely inspired by how the mammalian brain uses different areas of the cortex to abstract different levels of features when given an input percept.

A frequent illustration used to demonstrate the DL mechanism is how the monkey visual system perceives and learns an item via a series of transformations and representations: edge, basic

shape, and more complex characteristics. Hinton et al. [16] presented Deep Belief Networks (DBNs) using a novel unsupervised training approach known as layer-wise-greedy-training, which contributed to the surge in popularity of deep learning techniques. DL technologies have already produced breakthrough results in Natural Language Processing (NLP), speech recognition, computer vision, and image analysis, surpassing traditional machine learning and artificial intelligence approaches that rely on human-crafted features. This is due to their capacity for automatic high-level feature abstraction and exponential growth of data. Restricted Boltzman Machines (RBMs), Deep Belief Networks (DBNs), Autoencoder (AE), and Deep Convolutional Neural Networks (DCNNs or Deep ConvNets) are the four primary DL architectures [20].

DCCNs have proven to be highly effective at processing visual data, such as images and videos. DCCNs take raw input data at the lowest level and transform them by processing them through a sequence of basic computational units to obtain representations that have intrinsic classification values in the higher layers . Typically, a DCNN has three kinds of layers convolution layers, subsampling layers, and fully linked layers. Parameters of a convolutional layer include the number of channels, kernel size, stride factor, border mode, and connection table.

The convolution layer applies a convolution filter to the input picture to generate the output image or the filter response. Multiple convolutional layers are used to account for the spatial relationships between picture pixels. Subsampling is used to make the neural network more resilient and invariant. The max-pooling approach is often employed for the sub-sampling layer since it has been found to result in quicker convergence and improved generalization. Commonly, many fully connected layers are used after numerous rounds of convolution, and the structure of the last convolutional layer is flattened before to its connection to the next fully connected layer.

In addition to their higher classification and regression performance, DCNNs are gaining popularity because to their interpretability. Visualizing the first-layer weights of a trained DCNN has become a common approach for understanding what the network has learnt. The cleanliness of the features learnt by the initial DCNN layer is a crucial indicator of the network's training quality.

Activations of the network during the forward pass for specific pictures are plotted and seen using a standard visualization tool. Several popular open-source DL frameworks, such as Caffe (created by the Berkeley Vision and Learning Center), The Microsoft Cogni- tive Toolkit, Google's TensorFlow, Theano Framework (with a Python interface), Torch Framework, dmlc MXnet, Chainer, Keras, etc., are available for the implementation of DCNNs involving significant parallelism in computations.

For DCNNs to attain high predicted accuracy, huge, annotated picture datasets are often required. In many fields, acquiring such data is difficult and labeling them is expensive. Considering these challenges, it has been demonstrated that 'off-the-shelf' DCNN features of well-established DCNNs such as VGG-16, AlexNet, and GoogLe-Net that have been pre-trained on large-scale annotated natural image datasets (such as ImageNet) are very useful for solving cross domain image classification problems via the concepts of transfer learning and fine-tuning [25]. In a deep convolutional neural network, the representations learnt at successive network layers correspond to different degrees of abstraction contained in the input pictures. The early layers extract information about edges and color, while the subsequent layers' filters encode shape and texture. Using deep learning models trained on "big data" picture datasets (such as ImageNet) and "transferring" their learning abilities to new categorization scenarios is cheaper and more efficient than training a DCNN classifier from scratch [4] [25]. In some medical imaging applications, pre-trained DCNN has been proven to outperform even DCNN taught from scratch when subjected to proper adjustment.

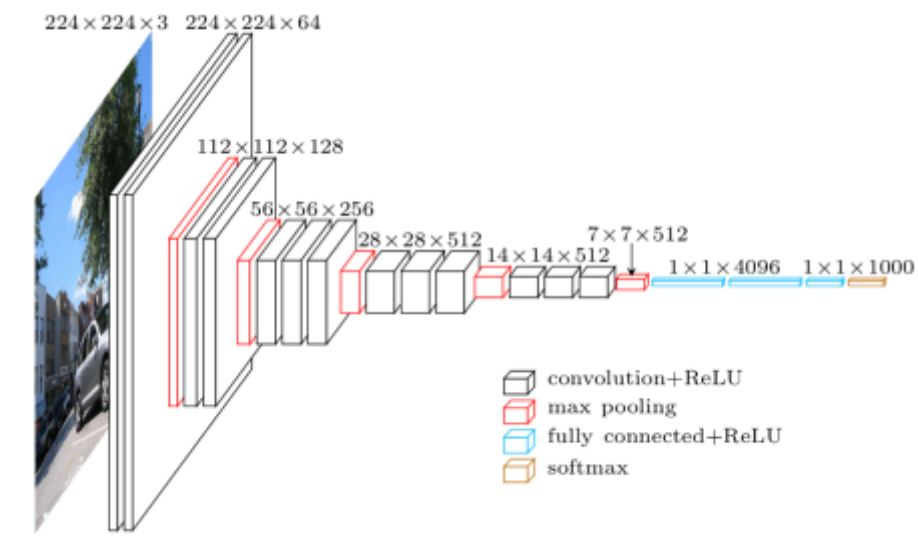

Figure 3. Depicts the VGG 16's architectural layout
Source: (https://ieeexplore.ieee.org/document/9031952/)

As shown in Figure 1, the conv1 layer is fed a picture with an input size of 224 x 224. The input picture is then sent through a set of convolutional layers with a 3 by 3 receptive field. It is assumed that the convolution stride is 1 pixel. For spatial pooling, five max-pooling layers with a stride of two are used (down sampling). A 22-pixel window is used for max-pooling layers, which succeed some of the convolutional layers. After the set of convolutional layers, there are three fully connected (fc) layers with channel sizes of 4096, 4096, and 1000, respectively. Each neuron in the fc layer receives information from the activations of the neuron in the layer below.

The number 1000 represents the number of categories in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). The last layer is called the soft-max layer. All the hidden layers are outfitted by the rectification (ReLU) non-linearity layer. VGG-16 architecture's main benefit is that it generalizes well to various datasets.

Softmax activation function

The softmax function transforms a vector of K real values into a vector of K real values whose total equals 1. Softmax turns the input values, which may be positive, negative, zero, or higher than one, into probabilities between 0 and 1. If one of the inputs is tiny or negative, the softmax transforms it into a small probability, and if one of the inputs is high, it transforms it into a large probability, but it always remains between 0 and 1 [26].

The softmax function is sometimes referred to as softargmax or multi-class logistic regression. This is since the softmax is an extension of logistic regression that can be used for multi-class classification, and its formula is extremely like that of the sigmoid function used in logistic regression. Only when the classes are mutually exclusive can the softmax function be employed in a classifier.

Numerous multilayer neural networks conclude with a penultimate layer that generates real-valued ratings that are not simply scaled and may be challenging to manipulate. Here, the softmax is quite beneficial since it transforms the scores to a normalized probability distribution that can be shown to the user or utilized as input for other systems. As a result, it is common practice to attach a softmax function to the last layer of a neural network.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

σ = softmax
$\vec{z}$ = input vector
e^{z_i} = standard exponential function for input vector
K = number of classes in the multi-class classifier
e^{z_j} = standard exponential function for output vector
e^{z_j} = standard exponential function for output vector

where zi values are the components of the input vector and may be anything in the real number range. The normalizing factor, located at the formula's end, guarantees that the total of the function's output values is 1, making the distribution a legitimate one.

Negative Log-Likelihood (NLL)

In real-world applications, the softmax function and the negative log-likelihood are combined (NLL). If we consider this loss function considering the behavior of softmax, we find it to be highly intriguing [26]. Let's start by outlining our loss function:

$$L(y)=-log(y)$$

This is totaled for all applicable courses. Recall that the objective of model training is to discover the minima of a loss function given a set of parameters (in a neural network, these are the weights and biases). We might interpret the loss as the network's "unhappiness" regarding its parameters. The more the loss, the greater the misery; we do not want this. We aim to make our models happy.

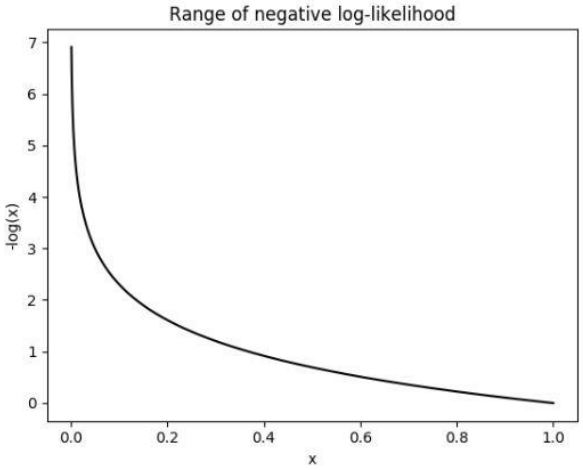

Figure 4. The loss function reaches infinity when input is 0 and reaches 0 when input is 1.
Source: (https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/

The log - likelihood gets dissatisfied at lower values, where it may approach infinite dissatisfaction, and it becomes less miserable at greater values. Because we are adding the loss function over all correct classes, when the network assigns high confidence to the right class, the dissatisfaction is low, but when it gives low confidence, it is high.

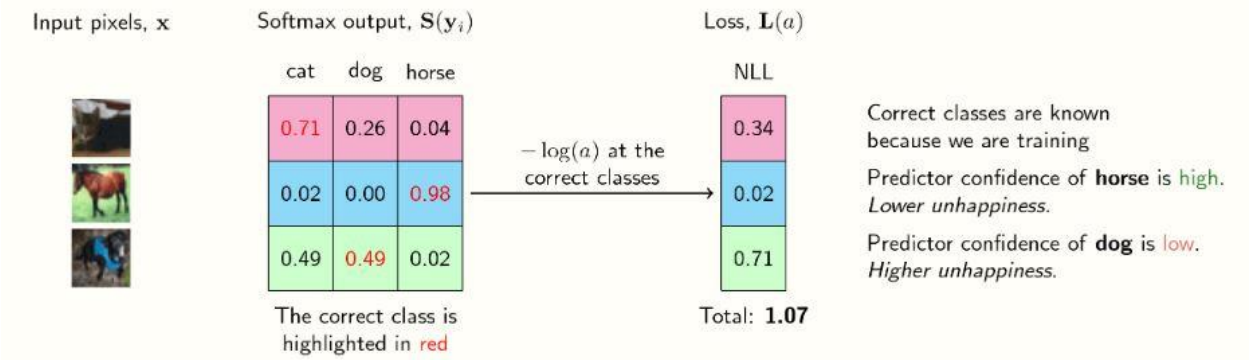

Figure 5. Computing the loss, with higher confidence at the correct class leads to lower loss and vice-versa
Source: (https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/)

Adam Optimizer

Adaptive Moment Estimation is a gradient-descent-based optimization approach. When dealing with a complex issue that has many data or parameters, the strategy proves to be quite effective. Effective and using little memory. At first glance, it seems to be a hybrid of the 'gradient descent with momentum' and 'recursive minimum spanning tree' (RMSP) algorithms [27].

Adam optimizer combines the following two kinds of gradient descent:

- Momentum: By factoring in an "exponentially weighted average" of the gradients, this approach is used to speed up the gradient descent process. When averages are used, the algorithm quickly approaches the minimum.

$$w_{t+1} = w_t - \alpha m_t \qquad m_t = \beta m_{t-1} + (1-\beta)\left[\frac{\delta L}{\delta w_t}\right]$$

where,

mt = the sum of slopes at time t [at the moment] (At first, mt equals 0)

mt-1 = the sum of the slopes at time *t-1*

Weights at time *t* = *Wt*

Weights at time t+1 equal *Wt*.

t = the rate of learning at time t

L is the loss function's derivative.

Wt = weights at time t's derivative

The moving average parameter is (const, 0.9)

- RMSP: Root Mean Square Propagation : An adaptive learning system called Root Mean Square Prop, or RMSprop, aims to enhance AdaGrad. It uses the "exponential moving average" as opposed to AdaGrad's method of computing the cumulative sum of squared gradients.

$$w_{t+1} = w_t - \frac{\alpha_t}{(v_t+\varepsilon)^{1/2}} * \left[\frac{\delta L}{\delta w_t}\right] \qquad v_t = \beta v_{t-1} + (1-\beta) * \left[\frac{\delta L}{\delta w_t}\right]^2$$

where,

Wt = weights at time *t*

Wt+1 = weights at time *t+1*

αt = learning rate at time *t*

∂L = derivative of Loss Function

∂Wt = derivative of weights at time *t*

Vt = sum of square of past gradients. [i.e sum*(∂L/∂Wt-1*)] (initially*, Vt = 0*)

β = Moving average parameter (const, 0.9)

ϵ = A small positive constant (10-8)

The two approaches have their own strengths, and Adam Optimizer relies on those strengths to provide a gradient descent that is better optimal.

Here, we regulate the gradient descent rate to achieve the global minimum with the least amount of oscillation possible while passing the local minima obstacles with sufficiently large steps (step-size). Thus, combining the advantages of the approaches will effectively obtain the global minimum.

Using the formulae from the two ways mentioned above, we get

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] \; v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2$$

used parameters

1. To prevent the "division by 0" mistake when (*vt -> 0*), = a tiny +ve constant. (10-8)

2. The decay rates of the average gradients in the first and second approaches are equal. (*β1* = 0.9 & *β2* = 0.999)

3. Learning rate and step size parameter (0.001)

VGG Configuration

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 6. ConvNet Configuration
Source: (https://iq.opengenus.org/vgg-11/)

Model-1: VGG11

VGG is short for Visual Geometry Group. Prior to discussing VGG-11 and its specifications, it is of the highest importance to examine AlexNet (just the basic part). AlexNet is mainly able to identify objects. It compensates for overfitting by using data augmentation and dropout. It substitutes the tanh activation function with ReLU by capturing its unique over-pooling characteristics. VGG entered the scene because it addressed the breadth of CNNs [25].

It is a model that has been trained on a dataset and includes the weights that reflect the characteristics of that dataset. By using a pre-trained model, one saves time. A considerable amount of time and computational resources have already been devoted to learning many characteristics, and the model will certainly reap the benefits.

The number after the keyword indicates the quantity of weighted layers included in the model. The input picture for VGG models should be 224 by 224 pixels and in RGB format. If the reader is curious as to why just 224 of the 0-255 RGB pixel range was considered to maintain a consistent

picture size, the answer is that this was done to maintain a constant image size. Now arriving and listing out the layers of VGG model that also form the foundation of VGG-11 as well.

Regarding VGG-11, it consists of 11 weighted layers. Weights describe the strength of connections between units in neighboring network layers and are used to link each neuron in one layer to each neuron in the next layer. Weights close to 0 indicate that modifying this input would not affect the output. Note that the Max-Pooling layer is not considered as a weighted layer since it is a map of the most prominent features. Maximum pooling, or max pooling, is a pooling procedure that estimates the maximum or greatest value in each feature map's patch. In the case of average pooling, the outputs are down sampled or pooled feature maps that emphasize the most prevalent feature in the patch, as opposed to the average occurrence of the feature. This has proven more effective than average pooling for computer vision applications like as image categorization.

Model-2: VGG16

Edge and color information has been shown to typically be present in the features of a pre-trained CNN's early layers. The properties in the subsequent levels, on the other hand, are more tailored to the specifics of the classes. Therefore, there is little to no need to adjust the settings of the prior layers [23]. This served as our inspiration for solely fine-tuning the final three layers of the VGG-16 in the current study.

The following gives the justification for changing the layers. More than a million photos are used to train the VGG-16. It can divide photos into a whopping 1000 classes [22]. The VGG-16's final three levels are set up for these 1000 classes. These require fine-tuning for a more recent categorization task [24, 25]. The strategy for fine-tuning is to remove all network layers other than the last three. Substitute a fc layer, a softmax layer, and a classification output layer for the last three layers to adapt the layers to the new classification goal. The fresh data's class count is used to determine the size of the fc layer [24, 25].

The size value for the current work is two, which corresponds to the number of classes—normal and abnormal—in the study. Algorithm 1 provides the whole process for automating the categorization of brain images utilizing the transfer learning mechanism via the VGG-16 network design.

Database

The trials were conducted using the Carvana website's database. This collection contains a total of 2,368 images from the Carvana website, which have been separated into 2134 train images and 234 test images. All input pictures had in-plane resolution of 256 × 256.

Performance Measures

To verify the efficiency of the pre-trained VGG-11 and VGG-16 models using transfer learning, the precision, recall, and F1 scores are selected as the relevant metrics. They are described theoretically in the following equations:

- Precision is the proportion of accurately detected positive instances out of all expected positive cases. Thus, it is beneficial when False Positive costs are large.

$$\text{Precision} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Positive})}$$

- Recall that it is the proportion of properly detected positive instances relative to the total number of real positive cases. When the cost of False Negatives is considerable, it is essential.

$$\text{Recall} = \frac{\text{True Positive}}{(\text{True Positive} + \text{False Negative})}$$

- Accuracy is one of the more straightforward measures; it is the number of instances that were successfully detected. It is most prevalent when all courses are of similar importance.

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{(\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative})}$$

- F1-score: This is the harmonic mean of Precision and Recall, and it provides a more accurate measurement of erroneously categorized instances than the Accuracy Metric.

$$\text{F1-score} = \left(\frac{\text{Recall}^{-1} + \text{Precision}^{-1}}{2}\right)^{-1} = 2 * \frac{(\text{Precision} * \text{Recall})}{(\text{Precision} + \text{Recall})}$$

- Accuracy is used when True Positives and True Negatives are more important than False Negatives and False Positives; F1-score is used when False Negatives and False Positives are more essential. Accuracy may be employed when the class distribution is comparable, however F1-score is a superior measure in cases when classes are unequally distributed, as in the example given. Why Uneven class distribution appears in most real-world classification tasks; hence, F1-score is a superior measure for evaluating our approach.

Algorithm

The following procedures are carried out in order to get the performance measurements for both the VGG11 and the VGG16 models.

Table 1. Steps to perform for running VGG11 and VGG16 Models

Step: 1	Using Google Cloud Vertex AI platform or Google Colab – Mounting the google drive
Step: 2	Unzipping the dataset extracted from the Carvana website and placing them in the /content/drive/MyDrive/carvana_cars.zip
Step: 3	Create a visual. Datastore for reading pictures Include all the directory's subfolders and give each image its own folder name label.
Step: 4	Resize the input images to match the size of the input layer for the VGG-16 algorithm and VGG11.
Step: 5	Divide the data into training and test sets, using 80% of the photos from each category for training and 20% for each fold's network testing.
Step: 6	Review Network Architecture (Replace/Modify the Pre-Trained Network's Final Layers) x Modify just the last three layers of the pretrained network for "vgg16" to classify the pictures into the normal class and the abnormal class (fc, softmax, and classification output).
Step: 7	Train the Network
Step: 8	Test the new classifier on the testing dataset
Step: 9	Report the performance metrics

Equipment Used

Laptop or computer with enough processing power is necessary. Because the picture dataset has to be scraped from the internet, a machine with a large amount of memory (preferably 8 GB or more) and a large amount of storage space (10 GB of storage) is strongly suggested. There is no need for any further specialized apparatus. It is possible to expedite the training of the model by using a specialized GPU. These models were trained with the help of Google Colab since it enables cloud computing and scalable resource allocation. In order to scrape the photos from Carvana, you required a consistent internet connection that was somewhat fast.

A large number of software and package installations are necessary in addition to the necessary hardware in order to scrape and train the model. The following examples of packages and software were used for this project:

a. Python 3.8 or higher
b. Git
c. BeautifulSoup
d. Pytorch
e. Numpy
f. Python libraries like math, os, Requests
g. PILLOW
h. Matplotlib
i. OpenCV
j. Sklearn
k. Google Cloud Vertex AI or Google Colab platform

DATA ANALYSIS AND RESULTS

Dataset Description

For this particular project, a total of nine different automobile models were chosen. Table 1 provides a breakdown of the various automobile types as well as the individual images. The Carvana website had a total of 2368 photos scraped, with only 2134 being utilized for training and the remaining 234 being placed aside as a validation set. Within the Carvana system, the multi-POV photos may be found presented as two different mosaics. A example mosaic that was taken from the Carvana website is shown in Fig. n. In order to get these photographs, a separate request must be made. The mosaic has four rows and eight columns of the automobile seen from a variety of perspectives. Due to the fact that there are two mosaics, each offers a somewhat unique perspective when looking at a picture.

Figure 7. Vehicle Mosaic from Carvana
Source: (https://www.carvana.com/)

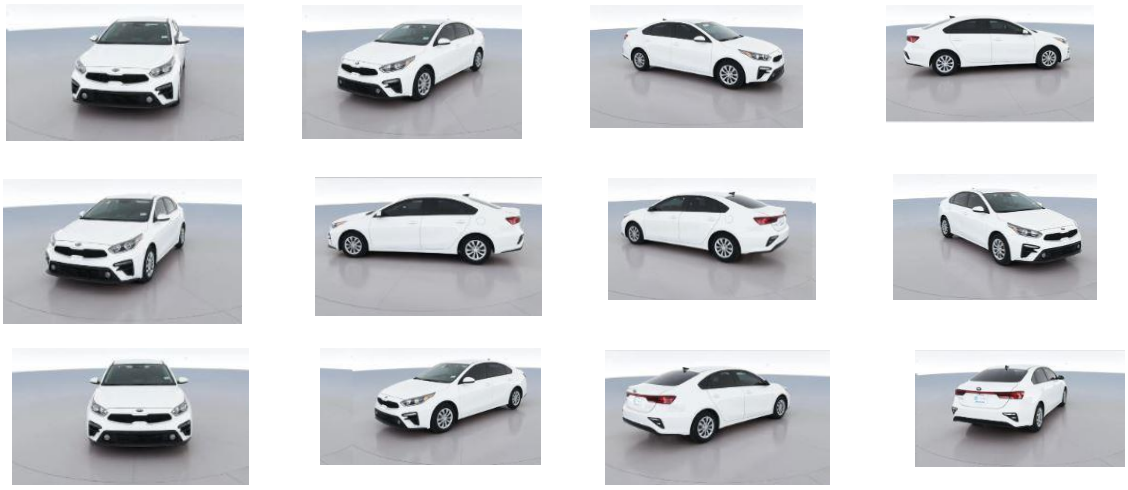

Figure 8. Image splitting

The image mosaic was cut up into thirty-two different images of the automobile, each with a different viewing angle. Each picture has a resolution that is 1920 pixels wide and 1080 pixels high. The validation set consists of 10% of images from each vehicle class that were saved for further use.

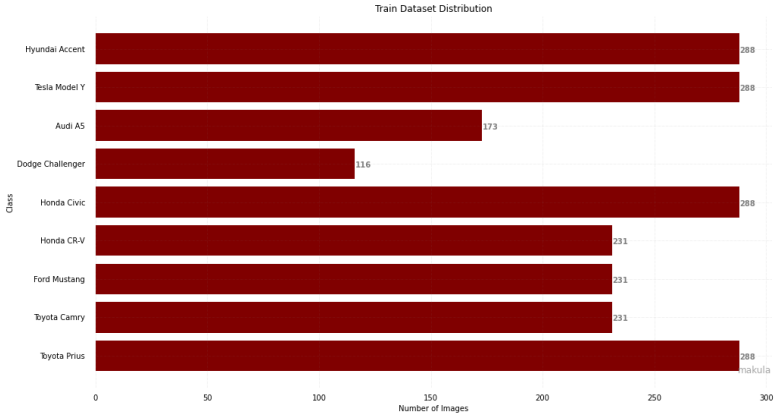

Figure 9. Train dataset Distribution

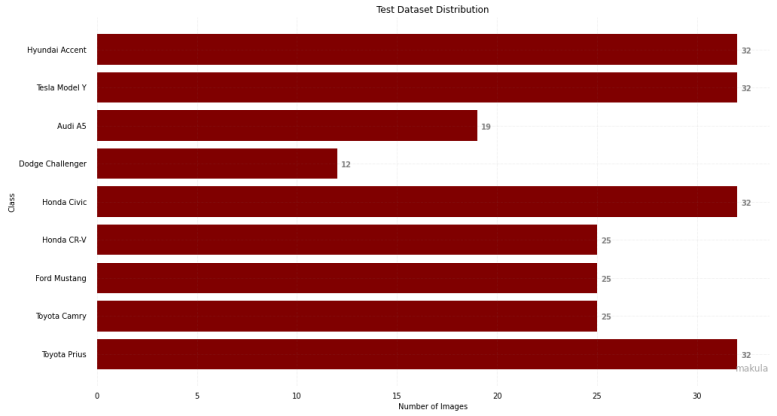

Figure 10. Test dataset Distribution

Training Results

Nvidia Tesla GPUs were used throughout the whole of the training and model creation process on Google Cloud Vertex AI. I decided to go with Google Cloud because it offers easily accessible GPU support, facilitates rapid environment setup, and makes it simple to upload files from Google Drive. After being stored in Google Drive, the dataset was subsequently made accessible via Google Cloud. In addition to this, a personal computer equipped with a GPU or Google Colab is an option for this.

Both the VGG11 and VGG16 models were trained using Transfer Learning. To ensure maximum precision, the model's hyperparameters were fine-tuned. Here are the final values used:

- Number of Epochs:05, **10,** 20
- Learning Rate: **0.0001**, 0.001, 0.1
- Optimizers: GD, SGD, **ADAM**.
- Activation function: Log **SoftMax** function
- Loss function: Negative Log Likelihood Loss(NLLLoss)
- Google Cloud Vertex AI platform used for training.

The maximum F1 score for both models was attained by the values in bold.

VGG11 Model Results

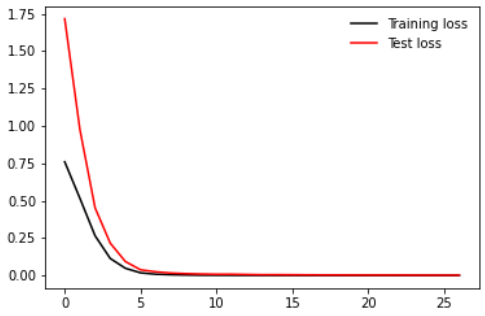

Figure 11. Train and Test Loss curve for VGG11 model

- Achieved 0.9934 F1- Score with VGG11 model on Validation dataset

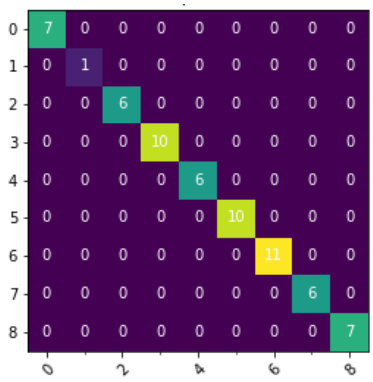

Figure 12. Confusion for VGG11 model

VGG16 Model Results

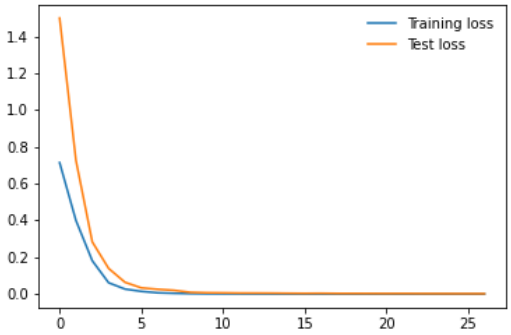

Figure 13. Train and Test Loss curve for VGG16 model

- Achieved 0.99821 F1- Score with VGG16 model on Validation dataset

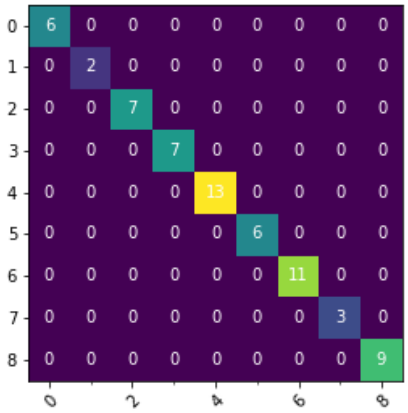

Figure 14. Confusion for VGG16 model

Predictions:

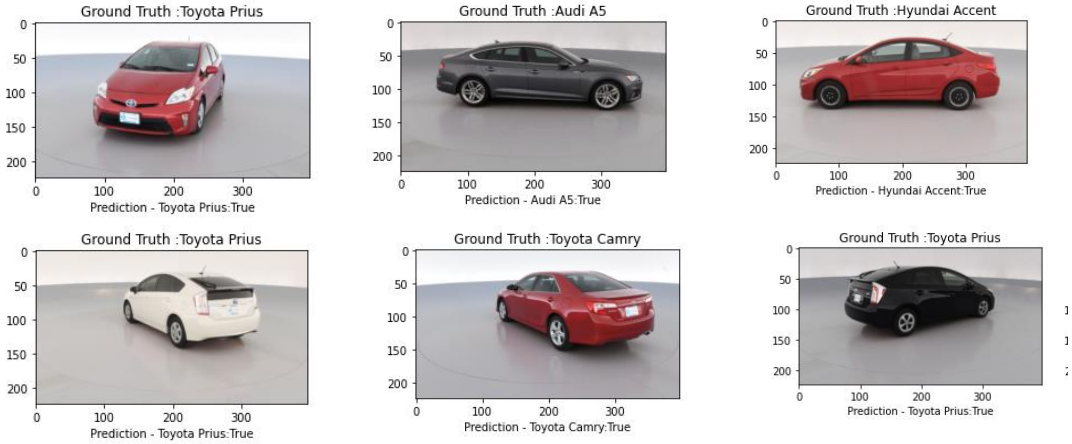

Figure 15. Predictions for both VGG11 and VGG16 models

CONCLUSIONS

Future Research

Despite the fact that the designed system should be capable of performing all essential duties, it is not ideal. It may be enhanced in the future to include additional features and increase its precision. The model is presently a multi-label single-class predictor, meaning it can only predict one vehicle class at a time, despite the presence of several cars. Due to the fact that the model was trained on photographs including a single car, it will not be able to make accurate predictions for images containing several vehicles. This indicates that the dataset is the limiting element in this case.

In addition, the dataset restricts the model in another manner. Since all of the photographs were obtained from the Carvana website, all of the images have a gray and white backdrop. The photographs have a great quality and are almost free of noise. While these photos are good for training purposes, we cannot anticipate perfect photographs in real-world settings. It is anticipated that the photos will have a poor resolution, include a great deal of background noise, and may cause the model predictions to be inaccurate.

To make the model more tolerant of actual photos and to address predictions for a particular class, the dataset must be changed. This may be accomplished by first modifying the output labels to display many classes rather than a single class. There is a need for additional photos, notably ones with lower quality, more noise, and landscapes in the backdrop. Also, in order to make the model genuinely multi-class, we will need to include photos of numerous cars

Obviously, the dataset must also be increased by adding other vehicle classifications. This is important since the existing dataset only contains a subset of the most popular automobiles on American roads. Since traffic scene reconstruction is not limited to certain cars, we must include as many automobile models as possible to make the model realistic.

The architecture of the model might potentially be enhanced to offer extra capabilities. The cars are identified using a MaskRCNN, as described in the CAROM study. MaskRCNN is undoubtedly a viable option for the needed functionality, however by training an object detection model, we may combine car detection and identification into a single model. YOLO and fastRCNN are ideal for this, with YOLO being among the quickest object detection models. This makes YOLO an excellent choice for replacing the current system, since it is speedier and more feature-rich overall.

REFERENCES

[1] J. P. Y. Z. a. X. F. H. Wang, "Multi-Path Deep CNNs for Fine-Grained Car Recognition," *IEEE Transactions on Vehicular Technology, vol. 69, no. 10, pp. 10484-10493,* p. doi: 10.1109/TVT.2020.3009162., 2020.

[2] Wang, Huibing and Peng, Jinjia and Zhao, Yanzhu and Fu and Xianping, "Multi-Path Deep CNNs for Fine-Grained Car Recognition," *IEEE Transactions on Vehicular Technology,* no. 10.1109/TVT.2020.3009162, 2020.

[3] L. Y. D. C. Z. M. a. J. C. X. Li, ""“Dual cross-entropy loss for small-sample fine-grained vehicle classification," IEEE Trans. Veh. Technol., vol. 68, no. 5, pp. 4204–4212, 2019..

[4] G. D. a. C. Schmid, "Selection of scale-invariant parts for object class recognition,”," in Proc. Int. Conf. Comput. Vis., 2003, vol. 1, pp. 634–640..

[5] T. R. Lim and A. T. Guntoro, ""“Car recognition using Gabor filter feature extraction,”," Proc. Circuits Syst. Asia-Pacific Conf, 2002.

[6] J. Krause, H. Jin, and J. Yang, and L. Fei-Fei,, "Fine-grained recognition without part annotations," Proc. IEEE Conf. Comput. Vis. Pattern Recognit2015, pp. 5546–5555..

[7] W. Wei and C. Guo,, "A text semantic topic discovery method based on the conditional co-occurrence degree," Neurocomputing, vol. 368, pp. 11–24, 2019.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, , ""“Imagenet classification with deep convolutional neural networks,," Proc. Adv. Neural Inf. Process. Syst., 2012, pp. 1097–1105..

[9] T.-Y. Lin, A. RoyChowdhury, and S. Maji, , "Bilinear cnn models for finegrained visual recognition,," IEEE Int. Conf. Comput. Vis., 2015, pp. 1449–1457..

[10] L. Yang, P. Luo, C. C. Loy, and X. Tang,, ""“A large-scale car dataset for fine-grained categorization and verification,," IEEE Conf. Comput. Vis. Pattern Recognit., 2015, pp. 3973–3981..

[11] Hu, H. Wang, T. Li, and C. Shen, "Deep CNNs with spatially weighted pooling for fine-grained car recognition,," IEEE Trans. Intell. Transp. Syst., vol. 18, no. 11, pp. 3147–3156, Nov. 2017..

[12] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear cnn models for finegrained visual recognition," IEEE Int. Conf. Comput. Vis., 2015, pp. 1449–1457..

[13] T.-Y. Lin, A. RoyChowdhury, and S. Maji,, ""“Bilinear cnn models for finegrained visual recognition," IEEE Int. Conf. Comput. Vis., 2015, pp. 1449–1457..

[14] H. Zheng, J. Fu, Z.-J. Zha, J. Luo, and T. Mei,, "Learning rich part hierarchies with progressive attention networks for fine-grained image recognition," IEEE Trans. Image Process., vol. 29, pp. 476–488, 2020..

[15] F. Tang et al, "On removing routing protocol from future wireless networks: A real-time deep learning approach for intelligent traffic control,," IEEE Wireless Commun., vol. 25, no. 1, pp. 154–160, Feb. 2018..

[16] Duo Lu1, Varun C Jammula1, Steven Como1 , Jeffrey Wishart2, Yan Chen1 and Yezhou Yang, "CAROM - Vehicle Localization and Traffic Scene Reconstruction from," *https://arxiv.org/,* vol. arXiv:2104.00893v1 [cs.RO], 2021.

[17] Zhang, Y.,Wang, S., Ji, G., Dong, Z., "An MR brain images classifier system via particle swarm optimization and kernel support vector machine.," Sci. World J. 2013, 1–9 (2013).

[18] Yang, G., Zhang, Y., Yang, J., Ji, G., Dong, Z., W, "Automated classification of brain images using wavelet-energy and biogeography-based optimization.," Multimed. Tools Appl. 75, 15601–15617 (2016).

[19] Wang, S., Zhang, Y., Dong, Z., Du, S., Ji, G., Yan, "Feed-forward neural network optimized by hybridization of PSO and ABC for abnormal brain detection.," Int. J. Imaging Syst. Technol. 25, 153–164 (2015).

[20] Bar, Y., Diamant, I.,Wolf, L., Lieberman, S., Kone, "Chest pathology detection using deep learning with nonmedical training.," ISBI. pp. 294–297 (2015).

[21] Zhou, M., Tian, C., Cao, R., Wang, B., Niu, Y., Hu, "Epileptic seizure detection based on EEG signals and CNN. Front.," Neuroinform. 12, 95 (2018).

[22] Zuo, H., Fan, H., Blasch, E., Ling, H, "Combining convolutional and recurrent neural networks for human skin detection.," IEEE Signal Process. Lett. 24, 289–293 (2017).

[23] geek4geeks, "Implementing Web Scraping in Python with BeautifulSoup," [Online]. Available: https://www.geeksforgeeks.org/implementing-web-scraping-python-beautiful-soup/.

[24] Keiron O'Shea and Ryan Nash, "An Introduction to Convolutional Neural Networks," in *https://arxiv.org/pdf/1511.08458*.

[25] Karen Simonyan, "VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION," in *arXiv:1409.1556*.

[26] L. Miranda, "Understanding softmax and the negative log-likelihood," [Online]. Available: https://ljvmiranda921.github.io/notebook/2017/08/13/softmax-and-the-negative-log-likelihood/.

[27] Geek4geeks, "Intuition of Adam Optimizer," [Online]. Available: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/.

[28] J. Wishart, S. Como, M. Elli, B. Russo, J. Weast, A. Niraj, E. James and Y. Chen, "Driving Safety Performance Assessment Metrics for ADS-Equipped Vehicles," *SAE Technical Paper 2020-01-1206,* 2020.

[29] H. Liu,, Y. Tian,, Y. Yang, L. and L. Pang, and T. Huang, "Fine-grained recognition without part annotations," *IEEE Conf. Comput. Vis. Pattern Recognit.,,* vol. pp. 5546–5555..

APPENDIX A – CODE FOR BOTH VGG11 & VGG16 MODELS

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = models.vgg11(pretrained=True)
print(model)

for param in model.parameters():
  param.requires_grad = False

model.classifier = nn.Sequential(nn.Linear(25088,4096),
                                 nn.ReLU(),
                                 nn.Linear(4096, 1024),
                                 nn.ReLU(),
                                 nn.Linear(1024, 512),
                                 nn.ReLU(),
                                 #nn.Dropout(0.2),
                                 nn.Linear(512, num_class),
                                 nn.LogSoftmax(dim=1))
criterion = nn.NLLLoss()
optimizer = optim.Adam(model.classifier.parameters(), lr=0.0001)
model.to(device)

[14]
# !cp /content/vgg16.pth /content/drive/MyDrive/VGG16/vgg16.pth
!cp /content/vgg11.pth /content/drive/MyDrive/VGG11/vgg11.pth

# Loss graph:

plt.plot(train_losses, label='Training loss', color = 'black')
plt.plot(test_losses, label='Test loss', color = 'red')
plt.legend(frameon=False)
plt.savefig("1.png")
plt.show()

# !cp /content/1.png /content/drive/MyDrive/VGG16/1.png
!cp /content/1.png /content/drive/MyDrive/VGG11/1.png
```

```
# device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# model=torch.load('/content/drive/MyDrive/VGG16/vgg16.pth')
# model.eval()

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model=torch.load('/content/drive/MyDrive/VGG11/vgg11.pth')
model.eval()

test_transforms =
transforms.Compose([transforms.Resize(224),transforms.ToTensor(),])
def get_images(path_val):
    data = datasets.ImageFolder(path_val, transform=test_transforms)
    classes = data.classes
    indices = list(range(len(data)))
    np.random.shuffle(indices)
    idx = indices
    from torch.utils.data.sampler import SubsetRandomSampler
    sampler = SubsetRandomSampler(idx)
    loader = torch.utils.data.DataLoader(data,
                   sampler=sampler, batch_size=64)
    dataiter = iter(loader)
    images, labels = dataiter.next()
    return images, labels,classes

def predict_image(image):
    image_tensor = test_transforms(image).float()
    image_tensor = image_tensor.unsqueeze_(0)
    input = Variable(image_tensor)
    input = input.to(device)
    output = model(input)
    index = output.data.cpu().numpy().argmax()
    return index

#F1 score and confusion Matrix:

path_val = '/content/dataset/val/'
```

```
to_pil = transforms.ToPILImage()
images, labels,classes = get_images(path_val)
total=len(images)
pred = []
y_true = []
for i in range(total):
    image = to_pil(images[i])
    index = predict_image(image)
    pred.append(index)
    y_true.append(int(labels[i].numpy()))

print(pred)
print(y_true)
f1 = f1_score(y_true, pred, average=None)
conf_mat = confusion_matrix(y_true, pred)
print(f1)
print(conf_mat)

#Heat map Generation:

fig, ax = plt.subplots()
im = ax.imshow(conf_mat)

# Show all ticks and label them with the respective list entries
ax.set_xticks(np.arange(len(classes)), classes)
ax.set_yticks(np.arange(len(classes)), classes)

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",
         rotation_mode="anchor")

# Loop over data dimensions and create text annotations.
for i in range(len(classes)):
    for j in range(len(classes)):
        text = ax.text(j, i, conf_mat[i, j],
                       ha="center", va="center", color="w")
```

```
ax.set_title("Heat map for VGG11")
fig.tight_layout()
plt.savefig("2.png")
plt.show()

# !cp /content/2.png /content/drive/MyDrive/VGG16/2.png
!cp /content/2.png /content/drive/MyDrive/VGG11/2.png

def get_random_images(val_path,num):
    data = datasets.ImageFolder(val_path, transform=test_transforms)
    classes = data.classes
    indices = list(range(len(data)))
    np.random.shuffle(indices)
    idx = indices[:num]
    from torch.utils.data.sampler import SubsetRandomSampler
    sampler = SubsetRandomSampler(idx)
    loader = torch.utils.data.DataLoader(data,
                   sampler=sampler, batch_size=num)
    dataiter = iter(loader)
    images, labels = dataiter.next()
    return images, labels

to_pil = transforms.ToPILImage()
images, labels = get_random_images(path_val,6)
fig=plt.figure(figsize=(30,200))
for ii in range(len(images)):
    image = to_pil(images[ii])
    index = predict_image(image)
    sub = fig.add_subplot(1, len(images), ii+1)
    res = int(labels[ii]) == index
    sub.set_title(str("Ground Truth :"+classes[labels[ii]]))
    a = str("Prediction - " + classes[index]) + ":" + str(res)
    # ax.set(xlabel=None)
    # ax.set_yticklabels()
    plt.xlabel(a)
    plt.imshow(image)
plt.savefig("3.png",bbox_inches='tight')
```

```
plt.show()

# !cp /content/3.png /content/drive/MyDrive/VGG16/3.png
!cp /content/3.png /content/drive/MyDrive/VGG11/3.png

# Training dataset Visualistion:


train="/content/dataset/train/"
val="/content/dataset/val/"
a=os.listdir(train)
print(a)
numImg=[]
for i in a:
  numImg.append(len(os.listdir(train+i)))
print(numImg)

# Figure Size
fig, ax = plt.subplots(figsize =(16, 9))

# Horizontal Bar Plot
ax.barh(a,numImg, color = "maroon")

# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    ax.spines[s].set_visible(False)

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey',
```

```
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.2)

# Show top values
ax.invert_yaxis()

# Add annotation to bars
for i in ax.patches:
    plt.text(i.get_width()+0.2, i.get_y()+0.5,
             str(round((i.get_width()), 2)),
             fontsize = 10, fontweight ='bold',
             color ='grey')

# Add Plot Title
ax.set_title('Train Dataset Distribution',
             loc ='center', )

# Add Text watermark
fig.text(0.9, 0.15, 'makula', fontsize = 12,
         color ='grey', ha ='right', va ='bottom',
         alpha = 0.7)
plt.xlabel("Number of Images")
plt.ylabel("Class")
# fig = plt.figure(figsize =(10, 7))
plt.savefig("4.png", bbox_inches='tight')
plt.show()

# !cp /content/4.png /content/drive/MyDrive/VGG16/4.png
!cp /content/4.png /content/drive/MyDrive/VGG11/4.png

# Testing dataset Visualistion:

val="/content/dataset/val/"
a=os.listdir(val)
print(a)
numImg=[]
for i in a:
```

```
  numImg.append(len(os.listdir(val+i)))
print(numImg)

# Figure Size
fig, ax = plt.subplots(figsize =(16, 9))

# Horizontal Bar Plot
ax.barh(a,numImg, color = "maroon")

# Remove axes splines
for s in ['top', 'bottom', 'left', 'right']:
    ax.spines[s].set_visible(False)

# Remove x, y Ticks
ax.xaxis.set_ticks_position('none')
ax.yaxis.set_ticks_position('none')

# Add padding between axes and labels
ax.xaxis.set_tick_params(pad = 5)
ax.yaxis.set_tick_params(pad = 10)

# Add x, y gridlines
ax.grid(b = True, color ='grey',
        linestyle ='-.', linewidth = 0.5,
        alpha = 0.2)

# Show top values
ax.invert_yaxis()

# Add annotation to bars
for i in ax.patches:
    plt.text(i.get_width()+0.2, i.get_y()+0.5,
             str(round((i.get_width()), 2)),
             fontsize = 10, fontweight ='bold',
             color ='grey')

# Add Plot Title
```

```
ax.set_title('Test Dataset Distribution',
             loc ='center', )

# Add Text watermark
fig.text(0.9, 0.15, 'makula', fontsize = 12,
         color ='grey', ha ='right', va ='bottom',
         alpha = 0.7)
plt.xlabel("Number of Images")
plt.ylabel("Class")
# fig = plt.figure(figsize =(10, 7))
plt.savefig("5.png", bbox_inches='tight')
plt.show()

# !cp /content/5.png /content/drive/MyDrive/VGG16/5.png
!cp /content/5.png /content/drive/MyDrive/VGG11/5.png
```