Vehicle Recognition Using Deep Learning on Stanford Image-Set

Tyler Bryk
Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, New Jersey, USA
tbryk@stevens.edu

Michael Eng
Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, New Jersey, USA
meng1@stevens.edu

Gabriella Tantillo
Electrical and Computer Engineering
Stevens Institute of Technology
Hoboken, New Jersey, USA
gtantill@stevens.edu

***Abstract*—This document addresses the construction of deep learning algorithms to classify images containing vehicles. A dataset including 16,000 labeled vehicle images was used to train three different neural network classifiers. The NN models included a Visual Geometry Group (VGG) Model, Residual NN, and a Dense Convolutional NN. The training data underwent various transforms and feature extraction methods prior to training, all of which will be discussed in this paper. After training the three models, different performance tests were conducted to determine the fit of each model. The results section offers evidence that it is possible to successfully train deep neural networks for image classification.**

I. Introduction

A popular application of machine learning is image processing. Image processing and classification is vital when it comes to autonomous vehicles, medical imaging, and security. Image processing is also used across many everyday uses such as image organization, social media, marketing campaigns, and photo enhancements. For this project, image classification will be used to identify the vehicular class of automobiles. This application of machine learning has the potential to be exceptionally useful for different roadway facilities such as toll plazas and parking garages, which charge different rates based on vehicular class. The project will consist of three different classifiers which map automobile images to one of 196 vehicle classes.

II. Related Work

Machine learning algorithms that classify cars into different categories are popular across many disciplines. The government, insurance companies, security systems all have practical uses for machine learning algorithms that identify the make and model of cars. Similar applications are any other image classifier. Algorithmically, determining whether a picture is of a dog or a cat is similar to classifying the type of vehicle. The related works section of this project will focus on car image classifications, because it falls underneath the larger umbrella of image classification. The Stanford Cars Dataset was last updated 2 years ago, demonstrating that vehicular classification is a popular area of machine learning applications.

Car classifications based off of images have been popular for quite some time. This first article related to this project was written in 2000, "Algorithms for Vehicle Classification" [4]. The authors' goal was to classify vehicles into two different subcategories: cars, and trucks. They wanted this information because it could aid in traffic pattern design, and was a better alternative than using people to count cars and trucks going through an intersection. A clustering algorithm was used to classify the vehicles passing through an intersection. The classifications relied on the parameters of height and length of vehicles passing. The results of the model were 99% accurate, with the shortcomings being short trucks.

10 years later in 2010, car classifications were still being performed for similar applications. The article "Vehicle Classification Based on Soft Computing Algorithms [1]" goal was to classify vehicles by either sedan, van, or truck. The authors of this article used four different classifiers. K Nearest Neighbors algorithm (KNN), Artificial Neural Network (ANN), Decision Tree (DT), and Random Forest (RF). They found that the ANN and RF classifiers resulted in the best accuracy. The optimized model had an accuracy of 95.8% which was the ANN algorithm. This article demonstrates how machine learning algorithms had improved over a 10 year span. The classifications were no longer just limited to 2-dimensional parameters.

III. Dataset Description

In terms of project data, Stanford's Cars Dataset is used, which contains over 16,000 images of cars and their respective labels which compromise 196 different features, based on the vehicles year, make, and model. The dataset is already balanced by class, meaning that there are an equal number of samples per each of the 196 classes. Additionally, the data has already been split into training and testing sets following a 1:1 ratio (8K training and 8K testing data).

As far as data transformation is concerned, the images were resized to a uniform 256x256 size, and the RGB pixel values were normalized using PyTorch's built-in feature normalizer. This function finds the average pixel value across all images in a set, and then subtracts that calculated average from the training images. No further transformations have been completed at this time, but dimensionality reduction is certainly a consideration for the future.

IV. Machine Learning Algorithms

As far as machine learning algorithms are concerned, three different deep learning neural network variations will be developed. Given that this is an image classification project, the data input vectors are highly dimensional, so using a neural network which can handle large inputs is ideal. The three algorithms of choice are the visual geometry group (VGG), residual neural network, and dense convolutional neural network.

A. Visual Geometry Group (VGG)

The Visual Geometry Group (VGG) algorithm, a successor to the AlexNet, was chosen as the first model to be implemented for this project. The model was selected because of its simplicit design and modest accuracy. In terms of architecture, it can include up to 11, 16, or 19 hidden layers, and it uses a convolutional receptive field of size 3x3. The hidden layers are split into 6 groups, the first 5 contain 2 to 3 convolution layers and one pooling layer. Then, the last one contains 3 fully connected layers [4].

The VGG model was implemented in a Jupyter Notebook environment which was running on an Nvidia Tesla K80 GPU accelerator. The PyTorch Torchvision library was used for the actual implementation of this algorithm. The data was loaded in from cloud files and transformed. The transformation included a simple image resize to a standardized 256x256 size, and then image normalization using a three-channel mean and standard deviation; finally, the images were converted to tensor objects. The normalization values are in the format of (mean[ch1], mean[ch2], mean[ch3]) and (std[ch1], std[ch2], std[ch3]). Originally, a mean of 0.5 and standard deviation of 0.25 was used across all channels. However, during implementation, it was later discovered that the suggested PyTorch values were optimal. A mean of (0.485, 0.456, 0.406), and standard deviation of (0.229, 0.224, 0.225) were used across the three channels.

The VGG was loaded in as a pre-trained model which simply means that it was already trained on the famous ImageNet dataset, and then exported to a PyTorch model file. After loading the pre-trained model, the architecture was defined to yield 196 outputs since there were 196 different vehicle classes. The model parameters were also initialized in this stage as well: learning rate of 0.01, momentum of 0.90, cross-entropy loss function, and lr-scheduler with stochastic gradient descent. Next, the model was trained using a GPU accelerator which took around 6 hours to complete. Afterwards, the in-sample accuracy was recorded as 94.118% and the out-of-sample accuracy was 67.753%.

Despite having a large training time, the model parameters were still tuned in order to achieve optimal results. When tuning a model, there are a lot of different techniques that can be applied from manipulating the training data with preprocessing to adjusting the model parameters. As far as modifying the dataset goes, both the training and testing datasets were already balanced, so the most that could be done was to apply different transformations. After trial and error with different transformations such as center cropping, padding, and random filtering, it was concluded that the best accuracy was achieved with a simple resize and image normalization. For the normalization means and standard deviations, two approaches were tried. First, a generic mean of 0.5 and std. of 0.25 was trialed. Then, a second trial was conducted where the PyTorch suggested mean and std. were used for each channel (specific values are noted above). The results showed that using PyTorch's mean and std. were optimal.

In terms of model tuning, a few different parameters were varied which included the model complexity, number of training iterations, learning rate, and momentum. The PyTorch VGG model comes in a few different flavors: 11-layer, 16-layer, and 19-layer architectures. The VGG also comes with the option of being pre-trained. For the model architecture, the 19-layer version was chosen, and it had the best accuracy of the three configurations. When testing the models as pretrained, having that flag set to 'true' yielded nearly a 4% testing accuracy boost in all cases, so that flag was left on for the remainder of the tuning process. The results for training different layer sizes are shown in Fig 1.

Model	Pretrained	In-Sample Accuracy	Out-of-Sample Accuracy
VGG-11	TRUE	93.178%	62.790%
VGG-11		93.335%	59.389%
VGG-16	TRUE	93.689%	64.479%
VGG-16		93.490%	61.578%
VGG-19	TRUE	94.118%	67.753%
VGG-19		94.067%	63.876%

Fig. 1. VGG Accuracy vs. Layer Size

The number of training iterations was also tested, with a range of 5 to 15, and the optimal number was determined to be 10 iterations as shown in Fig 2.

Iterations	Train Acc.	Test Acc.
5	88.723%	64.487%
7	92.390%	65.980%
10	94.118%	67.753%
12	94.129%	67.738%
15	94.298%	67.690%

Fig. 2. VGG Accuracy vs. Number of Iterations

Similarly, the learning rate and momentum parameters were varied from 0.001 to 0.5, and 0.80 to 0.95, respectively. Ultimately, the optimal parameters were determined to be lr=0.01 and momentum=0.90 (Fig 3). At this point, the model tuning was considered to be completed.

Learning Rate	Momentum	Test Acc.
0.001	0.90	65.783%
0.01	0.90	67.753%
0.10	0.90	66.839%
0.50	0.90	64.720%
0.01	0.80	66.712%
0.01	0.85	67.283%
0.01	0.90	67.753%
0.01	0.95	67.599%

Fig. 3. VGG Accuracy vs. Parameters

B. Residual Neural Network

The residual neural network, or ResNet for short, was chosen as the second algorithm for this project for a number of reasons. First, the ResNet has the ability to re-propagate signals through layers and also skip through layers by using 'Identity Shortcut Connections' [3]. This is exceptionally useful because it allows ResNets to handle and solve the vanishing gradient problem [6]. The vanishing gradient problem is a phenomena where the backpropagation gradient becomes infinitely small as it travels back to earlier layers. Ultimately, the ResNet can alleviate this issue by skipping through layers by using "highway" connections - a special type of connection between perceptrons that enables signal to pass straight through if certain criteria is met. Without getting into the semantics of ResNets too much, it should be mentioned that they are well known for being able to model the learning process of large multi-layer networks by simply using a few compact layers. For that reason, this algorithm was selected.

The residual neural network was implemented in the same Jupyter Notebook environment as the VGG model, and it also extended the PyTorch Torchvision library. The data was loaded in from cloud files, and transformed following the same parameters and criteria as shown in the VGG model. The transformation included a simple image resize to a standardized 256x256 size, and then image normalization using a three-channel mean and standard deviation; finally, the images were converted to tensor objects. The ResNet was loaded in as a pre-trained model. Then, the model architecture was defined to yield 196 outputs. The model parameters were also initialized in this stage as well: learning rate of 0.01, momentum of 0.90, cross-entropy loss function, and lr-scheduler with stochastic gradient descent. Next, the model was trained using a GPU accelerator which took around 3 hours to complete. Afterwards, the in-sample accuracy was recorded as 99.293% and the out-of-sample accuracy was 75.928%.

The ResNet model had a much better training time of about 3 hours, which is a 50% reduction from the VGG model. Regardless, the training time still made it difficult to tune the model. As mentioned previously, the data set did not require any further transformations or cleaning, so all tuning took place in the form of model architecture and parameters. A few different parameters were varied which included the model complexity, number of training iterations, learning rate, and momentum. The PyTorch ResNet model comes in a few different flavors: 18-layer, 34-layer, and 50-layer architectures (101 and 152 configurations exist as well, but were simply too complex for this project). The ResNet also comes with the option of being pre-trained. For the model architecture, the 50-layer version was initially chosen, but it was computationally expensive and the training results were poor. To simplify the model complexity, both the 34 and 18-layer models were trained, and the 34-layer model had the best accuracy of the three configurations. When testing the models as pretrained, having that flag set to 'true' yielded nearly an 8% testing accuracy boost in all cases, so that flag was left on for the remainder of the tuning process. The results for training different layer sizes are shown Fig 4.

Model	Pretrained	In-Sample Accuracy	Out-of-Sample Accuracy
ResNet-18	TRUE	99.024%	74.106%
ResNet-18		97.932%	66.923%
ResNet-34	TRUE	99.293%	75.928%
ResNet-34		98.102%	67.829%
ResNet-50	TRUE	99.002%	73.273%
ResNet-50		97.402%	66.358%

Fig. 4. ResNet Accuracy vs. Layer Size

The model was tuned for a varying number of iterations ranging from 5 to 15. The result indicated that the model converged around 12 iterations, and diminishing returns were achieved afterwards. The iteration optimization is shown in Fig 5.

Iterations	Train Acc.	Test Acc.
5	97.839%	72.698%
7	98.286%	73.964%
10	99.034%	75.194%
12	99.293%	75.928%
15	99.853%	75.463%

Fig. 5. ResNet Accuracy vs. Number of Iterations

Lastly, the learning rate and momentum parameters were varied from 0.001 to 0.5, and 0.80 to 0.95, respectively. Ultimately, the optimal parameters were determined to be lr=0.01 and momentum=0.90 which is shown in Fig 6. At this point, the model tuning was considered to be completed.

C. Dense Convolutional Network

The Densely-Connected Convolutional Neural Network (DenseNet) was chosen as the third algorithm for this project.

Learning Rate	Momentum	Test Acc.
0.001	0.90	71.798%
0.01	0.90	75.928%
0.10	0.90	73.980%
0.50	0.90	70.587%
0.01	0.80	75.819%
0.01	0.85	75.893%
0.01	0.90	75.928%
0.01	0.95	75.910%

Fig. 6. ResNet Accuracy vs. Parameters

The DenseNet is similar to the ResNet in many ways, however, the DenseNet is a newer, more robust algorithm specifically designed for image classification. Similar to ResNets, the DenseNet alleviates the vanishing gradient problem, and it also strengthens feature propagation. On the other hand, DenseNets promote feature reuse because of their feature mapping schemas which recycle features from layer to layer [4]. Additionally, DenseNets inherently have less parameters than other neural networks which makes them easier to optimize. Based on the computational advancement and simplicit design, this algorithm was chosen as the final model for this project.

The implementation process for the DeseNet was nearly identical to that of the other algorithms. The data was loaded in from cloud files and transformed using the same image resize and normalization parameters as performed in the VGG and ResNet. The DenseNet model was also loaded from PyTorch, and the architecture was configured so that the model would have 196 outputs. Additionally, the same optimizer, lr-scheduler, and cross-entropy loss function was used. The model parameters initially started out with a learning rate of 0.01 and momentum of 0.90. Due to the size and complexity of this DenseNet model, a GPU accelerator could not be used because it would simply overflow the CUDA memory. Instead, a standard CPU processor was used to train the model which took around 8 hours. Once tested, the in-sample accuracy was 99.840%, and the out-of-sample accuracy was 78.845%.

The parameter tuning process for this model was slightly less extensive due to the time complexity of the model. Being that 8 hours were required to train the model after each change in parameter, only a limited number of parameters were tested, which included learning rate and momentum. The DenseNet model comes in a few different flavors: 121-layer, 161-layer, 169-layer, and 201-layer architectures. The DenseNet also comes with the option of being pre-trained. For the model architecture, the 121-layer version was initially chosen, and that ultimately ended up performing the best. The higher-layer models were trained, the 161 and 169-layer models yielded accuracies that were nearly identical to that of the 121-layer, but the model complexity was large, and the training time was very costly. The 201-layer version of the model was simply too large for the computer and resulted in a CUDA out of memory error. When testing the models as pre-trained, having that flag set to 'true' yielded nearly a 6% testing accuracy boost in all cases, so that flag was left on for the remainder of the tuning process. The results for training different layer sizes are shown below.

Model	Pretrained	In-Sample Accuracy	Out-of-Sample Accuracy
DNet-121	TRUE	99.840%	78.821%
DNet-121		99.589%	72.279%
DNet-161	TRUE	99.829%	78.845%
DNet-161		99.490%	72.965%
DNet-169	TRUE	99.702%	78.863%
DNet-169		99.427%	73.380%

Fig. 7. DenseNet Accuracy vs. Layer Size

The number of training iterations was also tested, but with a smaller range, and the optimal number was determined to be 10 iterations as shown in Fig 8.

Iterations	Train Acc.	Test Acc.
5	97.328%	74.903%
7	98.961%	77.189%
10	99.840%	78.821%
12	99.365%	78.721%
15	99.026%	78.401%

Fig. 8. DenseNet Accuracy vs. Number of Iterations

Similarly, the model parameters were tuned and the optimal learning rate was chosen as 0.01 and momentum as 0.90 (Fig 9). At this point, the model was considered to be fully optimized given its complexity.

Learning Rate	Momentum	Test Acc.
0.001	0.90	76.721%
0.01	0.90	78.821%
0.10	0.90	75.019%
0.50	0.90	73.982%
0.01	0.80	77.982%
0.01	0.85	78.238%
0.01	0.90	78.821%
0.01	0.95	78.589%

Fig. 9. DenseNet Accuracy vs. Parameters

D. AlexNet Model

The AlexNet is another CNN algorithm which was implemented as a bonus. The Alexnet is one of the first image classification algorithms in its class. It predates the three algorithms that were implemented prior, and set new standards for image classification. It set the standard for image classification models to be required to classify high resolution images and also to classify off-center objects. It was implemented and became popular in 2012 by winning the ImageNet competition that year [1]. AlexNet architecture consists of eight layers with 5 convolution layers and 3 fully connected layers [5]. The AlexNet algorithm uses a GPU accelerator, but was able to increase speed by allowing for training on multiple GPUs, and reduce error by using overlapping. As earth-shattering as AlexNet was in 2012, it's standards of accuracy do not compare to the models that are being used today. It is a small algorithm that works quickly but can yield results less than 50% accuracy, which is no longer suitable for image classification. For this specific application of the AlexNet to the Stanford Cars data set, the resultant test accuracy was 40.592%.

V. Results

Throughout the implementation process, the models were tuned and the optimized parameters were found by using accuracy as the primary performance metric. The accuracy of each model was calculated by recording the percentage of correct classifications made on both the training and testing data sets. Of the four classification algorithms that were implemented, the optimized instances of each model were compared. Numerically, the algorithms were compared by the computed accuracy of the model. To start, Alexnet had a higher training accuracy than VGG-19, but since the testing accuracy is the more important metric for determining model performance, AlexaNet was deemed to be the most inaccurate algorithm. This makes sense as the AlexNet was one of the first convolutional neural networks in its class. The figure below shows that the training accuracies are much higher than the testing accuracies for each model, which is expected behavior of most machine learning algorithms. In terms of testing accuracy we can see that DenseNet-121 had the highest testing accuracy with 78.845%, which is very good for image classification. ResNet-34 followed behind with 75.928% testing accuracy. VGG-19 was next with 67.753% and as mentioned, AlexNet with the lowest testing accuracy of 40.592%. As a flat comparison we can conclude that DenseNet-121 has the best performance in terms of accuracy. Image classification models have other points of comparison including computational cost and speed. AlexNet ran the fastest, taking about an hour and a half on the GPU, VGG-19 ran for about 6 hours, ResNet-34 ran for 3 hours and, DenseNet-121 ran for about 8 hours. The majority of the runtime of the algorithms is spent on the model training part, and not so much on the testing. It is important to note that VGG-11 ran faster than ResNet-34 but VGG-16 and VGG-19 ran slower, due to computational costs. The more complex training process a model goes through, the longer it is going to run for. DenseNet goes through many complex layers and has the longest run time, but it is also the most accurate. We can see that training complexity correlates with run time. It is also interesting to note that rank in performance follows the age of the model. DenseNet came out recently and has the highest performance, because it has a new set of standards and must perform better than previous models in order to not be obsolete before it is even introduced.

Model	Train Acc.	Test Acc.
AlexNet	94.957%	40.592%
VGG-19	94.118%	67.753%
ResNet-34	99.293%	75.928%
DenseNet-121	99.829%	78.845%

Fig. 10. Best Model Accuracy

VI. Future Research

The top performing image classification models that have been optimized on a data set have only achieved accuracies of around 90%. The best model for this application performs at 78.895% accuracy. In the future, this accuracy could be optimized even further by fine tuning the parameters. A solution beyond the 90% accuracy would be for another image classification model to come along. The next model to come out is going to be an improvement, which is the easiest way to achieve a higher accuracy on the Stanford cars dataset. The best path to follow for future research would simply be to look out for new deep learning convolutional neural networks, as they will offer the best immediate accuracy boost. Specifically for this project, if more time was allotted, then a wider range of parameters could have been tuned during the optimization stage. As GPU acceleration becomes better and CNN models become more accurate, this project will inherently see an increase in accuracy and a reduction in computational cost.

VII. Conclusion

Image processing, a trivial task for humans, is almost impossible to program into a computer using conventional methods. Neural networks are able to solve this problem by taking inspiration from how the human brain processes information. Neural networks are not a new concept and have been proposed as far back as the 1940s, however in the early 2010s, researchers attacked the idea armed with faster computers and GPUs and proposed deep neural networks. The difference between deep neural networks and the original artificial neural networks is simply the amount of hidden layers between the input and output; if there is a substantial amount of hidden layers, the deeper the neural network and the better it will be at classifying an image of, say a vehicle.

To classify the vehicles from the Stanford Dataset, a collection of common Convolutional Neural Networks were compiled. In chronological order, which also happens to be the order of accuracy, the CNNs that were implemented were Alexnet, VGG-19, ResNet-34 and DenseNet-121. The results conclude that although all algorithms were able to, with a reasonable degree of accuracy, classify a new image of a vehicle given it is one of the 196 models, the DenseNet was significantly more effective. DenseNet's accuracy came at a cost as it was also equally more computationally intensive and took longer to train.

The ResNet that was used at the halfway point of the project was concluded to be both more accurate and more time efficient than the newly implemented VGG. In addition, Principal component Analysis or matrix factorization was discussed during the mid-stage report as a means to reduce the number of inputs, however convolutional neural networks include convolution layers to reduce inputs in their first few layers, rendering PCA obsolete. Overall, it was concluded that the accuracy and model complexity usually increases over time as new ideas come to life.

REFERENCES

[1] Dalka, Piotr, and Andrzej Czyżewski. "Vehicle Classification Based on Soft Computing Algorithms." SpringerLink, Springer, Berlin, Heidelberg, 28 June 2010, link.springer.com/

[2] Han, Jiawei, and Jian Pei. "Dimensionality Reduction." Dimensionality Reduction - an Overview — ScienceDirect Topics, Science Direct, 2012, www.sciencedirect.com/topics/computer-science/dimensionality-reduction.

[3] He, Kaiming, et al. "Deep Residual Learning for Image Recognition." ArXiv.org, 10 Dec. 2015, arxiv.org/abs/1512.03385.

[4] Huang, Gao, et al. "Densely Connected Convolutional Networks." ArXiv.org, 28 Jan. 2018, arxiv.org/abs/1608.06993.

[5] P. Domingos, P. Domingos, et al. "Deep Learning Applications and Challenges in Big Data Analytics." Journal of Big Data, SpringerOpen, 1 Jan. 1970, journalofbigdata.springeropen.com/articles/10.1186/s40537-014-0007-7.

[6] Papanikolopoulos, N, and S Gutpe. "Algorithms for Vehicle Classification." University of Minnesota, University of Minnesota Dept. of Computer Science and Engineering Artificial Intelligence, Robotics and Vision Laboratory Minneapolis, MN 55455, 200AD, www.its.umn.edu/Publications/ResearchReports/pdfdownload.pl?id=1696.

[7] Shaikh, Faizan. "Inception Network: Implementation Of GoogleNet In Keras." Analytics Vidhya, 14 May 2020, www.analyticsvidhya.com/blog/2018/10/understanding-inception-network-from-scratch/.

[8] Yu, Shucheng. "CPE695 Lecture Slides." Stevens Institute of Technology, 2020.