An Evaluation of the Application of Various Models in Autonomous Vehicles

Eric Chen
Stanford University
erichchen@stanford.edu

Emily Xia
Stanford University
emxia18@stanford.edu

Bruno Dumont
Stanford University
bdumont@stanford.edu

Abstract

In this project, we compare the effectiveness of several machine learning models such as MLPs, ConvNets, ResNets, and more to accomplish tasks that are regularly encountered by autonomous vehicles on the road. In particular, we evaluate the models on classification (determining whether a road image contains a car) and object detection (determining the location of a car within an image). Both of these tasks are crucial for any autonomous driving system, and it's We will evaluate the models' performances on the validation set against each other, and select the best one to run on the test set. We'll also evaluate the efficacy of fine-tuning existing models like ResNet50, VGGNet, and AlexNet to our dataset and compare them to the various models we built from scratch. We obtain our training, validation, and testing data from a wide range of image datasets including the Vehicle Detection Image Set, Car Object Detection, and Coco. Our project provides critical insight into the applicability of various models in autonomous vehicles, especially in such a field where safety is crucial.

1. Introduction

Image recognition, especially in the domain of self-driving cars, plays a pivotal role in various sectors ranging from automotive safety to autonomous driving systems. As the self-driving vehicle industry continues to evolve, the demand for accurate and efficient methods for identifying and classifying vehicles from images becomes increasingly crucial. The ability to segment and identify cars and car-related objects in an image holds immense value.

In this paper, we explore image recognition for self-driving cars. Our primary objective is to explore and compare the performance of various deep learning architectures such as linear classifiers, fully connected networks, convolutional networks (ConvNets), residual networks (ResNets), and more in accurately classifying and boxing road images. We aim to provide statistics about the performance and particular attributes of different models for this task. For instance, we are curious about what different models pay attention to when identifying a car out of an image—as could be analyzed with a heat map.

Through this comparative analysis, we address critical questions about the application of different deep-learning architectures in self-driving vehicles.

In the following sections, we will delve into the methods employed, data used, experimental setups, and results obtained from our model training, culminating in a comprehensive evaluation of the different architectures.

1.1. Problem Statement

We're currently focused on analyzing how different models perform tasks related to self-driving cars. Specifically, we want to compare how different deep learning designs handle image classification and object detection tasks.

2. Related Works

There has been significant progress in the field of image recognition for self-driving cars.

In 2021, Rao et al. achieved 86% accuracy in recognizing the make and model of a car [1]. They utilized a Single Shot Detector (SSD)-YOLOV5 for car recognition and a deep residual network (ResNet) for classification purposes. They also employed two different classification models: VGG16 and ResNet34. VGG16 is a CNN made up of convolution layers with filter sizes of 3×3 with stride 1 and a max-pooling layer of 2×2 with stride 2. ResNet34 is a 34-layer convolutional neural network that uses residual blocks with skip connections. In both models, the ReLU activation function is used in all hidden layers and the final result is followed by a softmax activation function to categorize. Both models were optimzied using SGD. Overall, VGG16 gave a 71% test accuracy and ResNet34 was 86%.

A year later, in 2022, another IEEE paper published by Kumar et al. combined neural networks and cost-effective cameras to build a self-driving car prototype [2]. Their prototype successfully recognized lanes, traffic signals, and obstructions along with implementation of trajectory planning and steering control. In their project, the robot contained a Raspberry Pi as the brain, a Raspberry Pi camera, and a Motor driver. An Image Folder with images collected by the

camera named as per the timestamp and a Log File of the steering angle values are used to train the model. The model is then trained by a Convolutional 2D layer, Flatten layer, Dense layer with the Adam method for optimization. The end result of their project was a robot that could drive and function on its own without any human input. It was able to find its own path on the road (a small road built by the researchers). Additionally, it was able to detect traffic signals and other barriers and act accordingly. For our project, we similarly want to examine the applications of advancements in image recognition in improving self-driving cars.

In 2023, Khan et al. analyzed two state-of-the-art object detection methods: You Only Look Once (YOLO) and Faster Region CNN (Faster R-CNN) [3]. Khan et al. created a hybrid model combining the boundary box selection capabilities of YOLO with the region of interest (RoI) pooling from Faster R-CNN resulting in improved segmentation and classification accuracy. The research compared 2 models of YOLO (v5 and v7), the R-CNN, and the hybrid approach based on two metrics mean average precision (mAP) and inference time. The proposed hybrid model achieves the highest mAP values compared to all other models when the inference time is between 4 ms and 44 ms. The research shows that combining the strengths of different models can be very effective in some tasks.

Similarly, we wish to similarly analyze an array of different models and point out the strengths and weaknesses of models to eventually build a fast and accurate model or combination of models. Though many of the models discussed above seem to have relatively high accuracies (e.g. 86%), in the context of self driving cars, this might not be enough. If our vehicles only correctly identified other cars 86% of the time, this still leaves much room for error, and this error can be incredibly harmful. Clearly, there is still much room for improvement in this regard. Therefore, our project aims to close this gap by taking state-of-the-art models such as the ones mentioned above, and determine which combination of models an give the highest accuracy.

2.1. Project Targets

For our project milestone, our goal was to explore the methods discussed in class for image classification. To do this, we utilize the Vehicle Detection Image Set (VDIS) [4]. Our aim for this milestone is to evaluate the following models on our data-set: linear classifiers, fully-connected networks, and convolutional neural networks.

We also intend to advance to the more complex task of object detection within an image—something crucial for self-driving cars to do in real-time. To do so, we'll use several models like AlexNet, ResNet, and VGGNet as backbones and fine tune them to our object detection task[5, 6, 7]. We would also like to try our hand at training our own object detection model and comparing our results with these fine-tuned models. We'll train our detection models on a combination of the Coco and Car Object Detection (COD) datasets [8, 9].

3. Methods

3.1. Image Classification

In the first part of our project we compared the efficacy of different classifier models on the Vehicle Detection Image Set (VDIS) [4]. We first implemented different models studied in class (Linear Softmax, Fully-Connected, and Convolutional Network). We trained the models to detect whether an image contains a car. We performed a train-val-test split of $60\%, 5\%, 35\%$ on the dataset. Our goal was to develop models that achieved near-perfect performance on the dataset, as the field of self-driving cars demands extremely high accuracy. Since we are working with a binary classifier we expect even simple models to have relatively high accuracy.

Our linear model flattened the inputs and performed a single matrix multiplication (with a $(64\times64\times3)\times2$ matrix), then a softmax operation. For our optimization we used the Adam optimizer with 1×10^{-3} learning rate.

Our fully-connected model flattened the inputs, then passed it through three hidden layers of sizes $5000, 10000, 200$. At each layer before the output, we perform a Batchnorm and a ReLU after the affine operation. We used the same optimizer for this model.

Our convolutional model leverages 6 convolutions, each followed by a batchnorm and a ReLU. After every other convolution, we add a 2×2 maxpool operation. Our first convolution uses a 5×5 kernel, while all others use 3×3 kernels. They all have stride 1 and padded so that the dimensions are unchanged. Our first two convolutions both have 16 output channels, our second two have 32 output channels, and our final two have 64 output channels. Following our convolutional layers, we flatten the output and pass it through two hidden layers of 1000 neurons each, taking Batchnorm and ReLU after each hidden layer. We once again used the Adam optimizer.

We trained the models until their training accuracy plateaued (60 batch iterations for linear and fully connected, and 100 for convolutional).

Additionally, we compared the accuracies of pre-trained models we learned about in class (Adam, VGG-16 and ResNet-50). We altered the final layers of the model to output a binary class selection and trained the models using SGD + Momentum to fine-tune the class selection in our data set. We also transformed all images to be 224x224 so that the pre-trained models could work correctly.

AlexNet [5] is a simple convolutional neural network and was one of the first successful image classification architectures to be trained on GPU. VGG-16 [7] is a newer and more

robust convolutional neural network architecture. ResNet 50 [6]is a convolutional network that makes the use of residual connections to ease the training of deeper networks. All these models have been pre-trained on millions of images and only their last layer has been changed to adapt to the new data-set. In particular, we loaded the pre-trained models directly from the Pytorch library [10].

3.2. Object Detection

For the bulk of our project, we expanded beyond a simple binary classifier and compared the ability of models to detect desired objects. Since we are mostly concerned with the usage of AI in the context of self-driving cars, we tested the ability of models to properly detect where cars are in a given image.

Our dataset consisted of everyday road images: some with cars, some without cars. Each image was paired with bounded box coordinates: $(x_{min}, y_{min}, x_{max}, y_{max})$. For images without cars, these coordinates defaulted to $(0, 0, 0, 0)$ to indicate the lack of a car. We then trained various models on this data to compare their abilities at object detection.

To adapt our models to do object detection, we changed the output of the last layer of every model to be 4 dimensional and used mean squared error as our loss function.

We first implemented the same convolutional neural network as our image detection with the only change at the last fully connected layer. After that, we used ResNet50 and AlexNet as two backbones for the model. For these models, we appended a single convolutional layer to the last convolutional layer of the backbone. This convolutional layer with 256 3×3 kernels, and a stride and padding of 1. This convolutional layer is followed by a 8×8 average pool and two linear layers of with activation dimensions of 1024 and 4. For these backbone models, we froze all the gradients within the backbones and only allowed backpropagation to affect the added layers.

As this is no longer a simple binary classification problem with a clear right and wrong, we had to create a new way of calculating the accuracy of each model to compare results. Here, it became important to differentiate between images with and without cars. When the original area was not zero (that is, there was a car in the image), the calculated value for that data point was the intersection of the true and predicted boxes divided by the union of the two areas. In other words, if we let T_b be the true box and P_b be the predicted box, When the original area was zero (no car in the image) we needed to take a different approach since no matter what the car output the intersection would be always zero and so the accuracy would not correctly reflect the model's capabilities. Hence we decided to take $\frac{1}{1+Area(P_b)}$ to be the accuracy value on this case. Thus, we have

$$Acc(T_b, P_b) = \begin{cases} \frac{Area(T_b \cap P_b)}{Area(T_b \cup P_b)} & \text{if } Area(T_b) > 0 \\ \frac{1}{1+Area(P_b)} & \text{if } Area(T_b) = 0 \end{cases} \quad (1)$$

This would penalize any nonzero predicted area with a lower accuracy the larger the predicted area is. Our accuracies were calculated by averaging over all of these accuracies for every data point.

However, we saw that our models could not handle simultaneously boxing cars and outputting zero-area boxes for images with no cars. Indeed, as we can see in Figure 1, the loss was wildly oscillating as the model trained—even with tiny learning rates on the order of 10^{-8}, meaning that something was going very wrong with the model. We suspect that this oscillating loss is due to the zero-car images forcing the model to output tiny boxes, while the one-car images kept pushing the sizes of the output boxes back up. This back-and-forth process resulted in horrendous accuracy and no convergence.

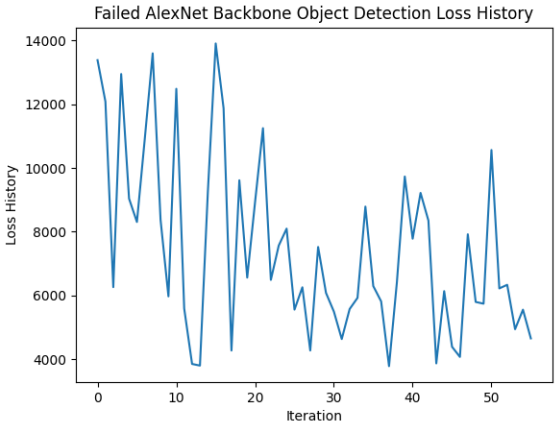


Figure 1. Loss of AlexNet Backbone Model, trained on both 0- and 1-car images for two epochs.

As such, we removed the images where there were no cars present and only included images where there was one car present. On this smaller dataset, we trained a ConvNet and two networks that used AlexNet [5] and ResNet18 [6] as backbones.

4. Dataset

For our project milestone, we utilize the Vehicle Detection Image Set, which contains over $17,000$ $64 \times 64 \times 3$ images depicting road scenes with and without cars [4].

For the object detection portion of our project, we utilized the COCO: Common Objects in Context dataset [8]. This dataset contains a variety of everyday images with labels and coordinates for bounding boxes on each labeled

Figure 2. Image 94: bbox = [356.23,275.22,42.87,38.46]

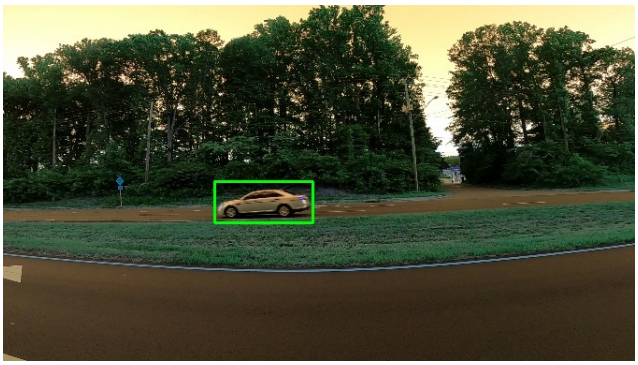


Figure 3. Image from COD, with ground-truth box

item of interest. To download and use the COCO dataset, we utilized FiftyOne an open-source tool to aid in downloading and visualizing datasets.

Of interest for our scenario, are images containing cars. It is rather convenient that COCO allows for the specification of specific image types when downloading the data. Therefore, our dataset consists of images with cars and bounding box coordinates for the car or cars in each image (bounding box coordinates in the form of (x, y) of the lower corner + the width and height). For example, 2 is a part of the dataset with an id-label of 94.

Additionally, we combined this database with another database we found on Kaggle: Car Object Detection (COD) [9]. This dataset contains over 1000 images of cars from all views and coordinates to bound the car and its location. These coordinates are in the form of (xmin, ymin, xmax, ymax) 3.

To combine the datasets properly, we modified the bounding boxes within the COCO dataset to resemble the coordinates from the COD dataset. Additionally, because images were all different sizes when inputted, we scaled each image to be 256×256 and then scaled the boxes accordingly. This ensured that the images inputted into the model were all of the same size.

By combining the two datasets, we were able to train and test on 932 images. Of these 932 images, 646 did not contain cars. This is largely due to the fact that the COD dataset used raw footage off of roads. Therefore, there was a considerable amount of time where no cars were present on these roads. Additionally, some of the images from the COD dataset contained more than one car. As not to confuse the model too much, we decided to remove such images. We'll refer to the combined dataset as CocoCOD.

Because the two dataset images have differing dimensionality, to run them through the same model, we first reshape all images to be $3 \times 256 \times 256$, scaling the bounding box dimensions accordingly.

5. Results/Evaluation

5.1. Binary Classification

Table 1 shows the final testing accuracies of each of our models. We found that all models performed well on the VDIS dataset, with 94% testing accuracy for the fully connected network and 98% accuracy for the ConvNet. Even the linear softmax classifier achieves 80% accuracy.

Model	Validation Accuracy
Linear Classifier	0.80
Fully Connected	0.94
CovnNet	0.98
AlexNet	0.99
VGG-16	0.98
ResNet60	0.98

Table 1. Testing Accuracies of Models on VDIS

Even with unusually high performance across all models, we can see a clear improvement from 80% to 98% due to the activation and normalization techniques utilized in the fully connected and convolutional networks

We can see that even our simple model with a linear classifier performed considerably better than random chance. We can also see that adding more hidden layers with nonlinearity and normalization techniques utilized had a significant improvement in accuracy. As expected the convolution model performed the best with less than 1/3 of the mistakes of the fully connected model.

For each of the three models, we graphed the change in training accuracy, validation accuracy, and training loss, as shown in Figures 7. We see that all models have a high amount of fluctuation in their training accuracy, but the validation accuracy is a lot more stable. We suspect that the training accuracy fluctuation is due to a very ill-behaved loss function, where a step in the gradient direction will often overshoot the local minimum. We also suspect that the comparative smoothness of the validation accuracy is that the models are powerful enough to generalize quickly, and most of the epochs are spent focusing on very small details in the training data.

Similarly, when comparing the validation accuracies, we see that the validation accuracy of both the Full Connected Model and the CNN converged rather quickly, but the linear model showed repeated fluctuation.

Additionally, we can see that our fine-tuned models achieved high accuracy very fast. This is due to the fact that models already learned to identify the most important features of the image and only needed to learn how to see which features were relevant to the car. Additionally, very complex models seemed to perform slightly worse than simpler models. We suspect that since this task is not that complex very deep models have a harder time identifying what parts of the image are relevant. This can be further analyzed by looking at the heat maps for the models.

Figure 4. Alex Net Heat Map

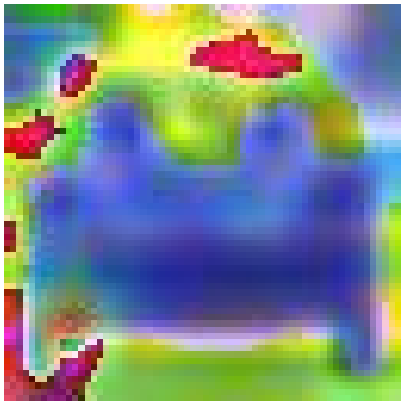


Figure 5. VGG Heat Map

As we can see, AlexNet had a bigger focus on important areas on on the car's boundary specially around the wheels which are represented by a strong red color. However, on more complex models we can see that the specificity was diluted and on the resnet50 we can see almost no red color showing that the model tried to identify the image as a whole and not focus on important areas. This might be a reason why AlexNet performed better than the other models.

]

Figure 6. ResNet50 Heat Map

5.2. Object Detection

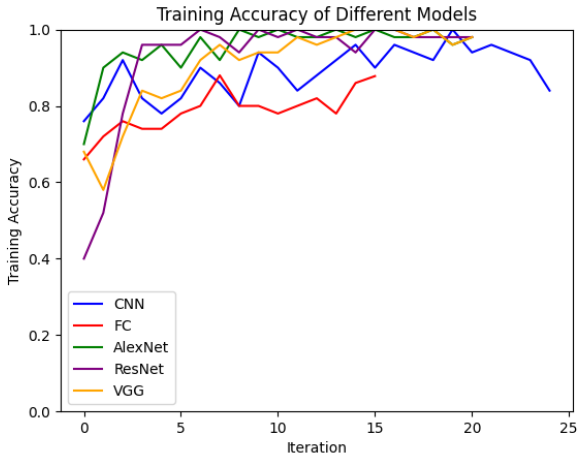
On our object detection task, we saw a different pattern. This task is considerably more complex than just the image classification and as a result, our accuracy was significantly lower.

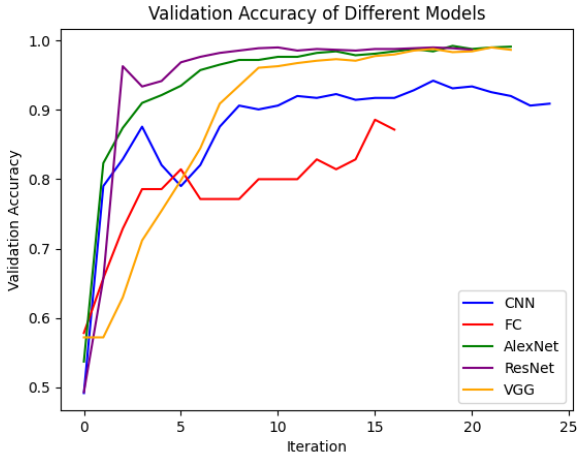
From our graphs, we can see that the ResNet18 backbone model didn't perform well with the validation accuracies oscillating below 0.2. The AlexNet backbone model performed better with stable accuracies around 0.4. Our best-performing model was the CNN with a more stable validation accuracy around 0.6.

Comparing ResNet18 with AlexNet reveals that the that performed better in the image classification task also performed better in the object detection task. However, comparing our CNN with AlexNet we see that even though AlexNet performed better in the image classification task our CNN performed much better in the object detection task. We hypothesize that this is due to the AlexNet model being pre-trained on a different task with different loss functions. Our CNN model was trained from scratch for the object detection task and only on this data set, however, the AlexNet model was trained on the image classification task for different data sets. This wasn't an issue on image classification tasks as seen in the performance of the models above however it proved it be a problem in object detection tasks.

However, we see that in almost all cases, the general area in which the model generates a box is very close to the area with the car. We believe that we only attain a lower accuracy because our accuracy function heavily penalizes any difference between the model's outputted box and the ground truth, even if they generally box the same area. We display an example of this from the AlexNet backbone model in Figure 11,

Another failure point could be a lack of data. Due to the structure of the Car Object Detection Dataset [9], there was overwhelmingly more images of roads without cars than





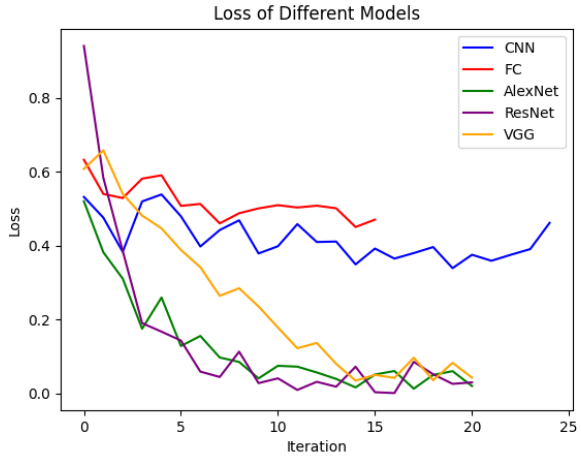
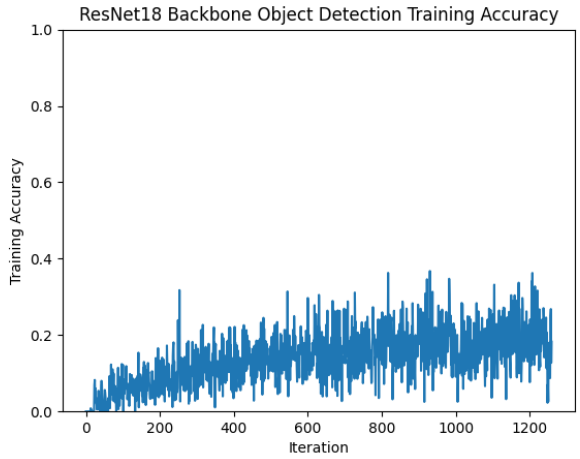
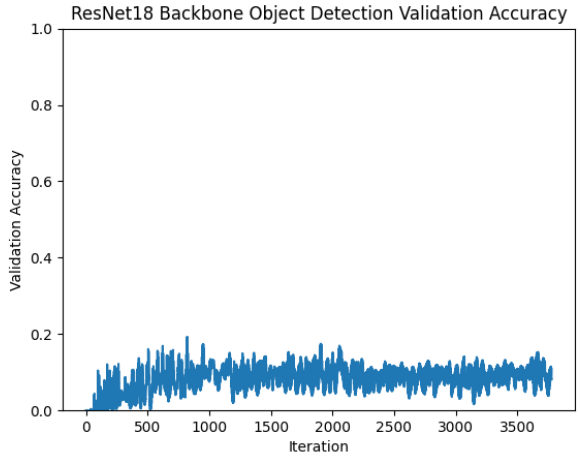


Figure 7. Training statistics of different classification models.





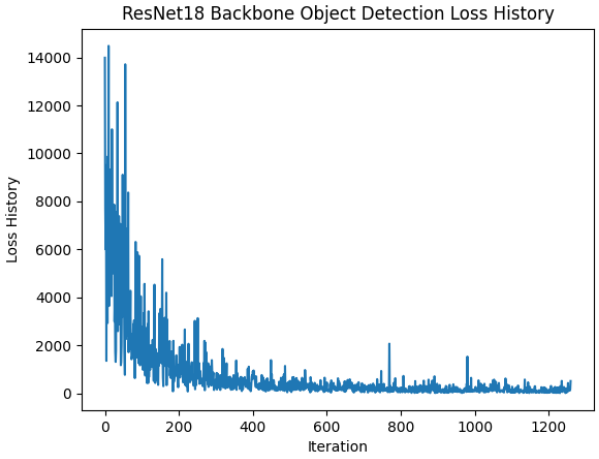
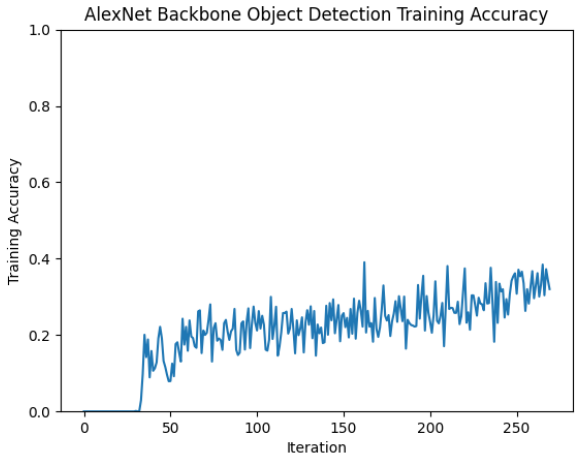


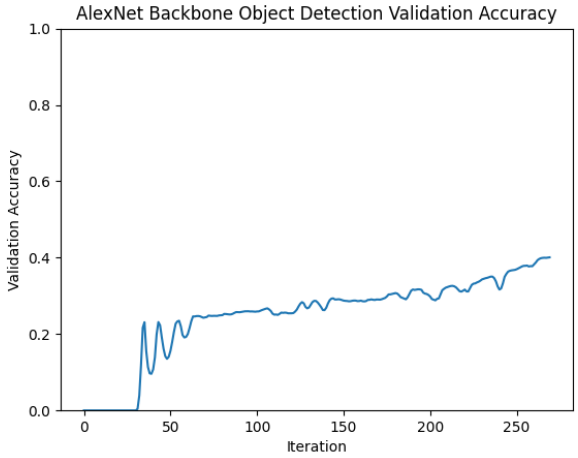
Figure 8. Accuracies and Loss for ResNet18 on CocoCOD

that with cars.

6. Conclusion

By comparing state-of-the-art models such as AlexNet, ResNet, etc. on simple yet fundamental aspects of com-





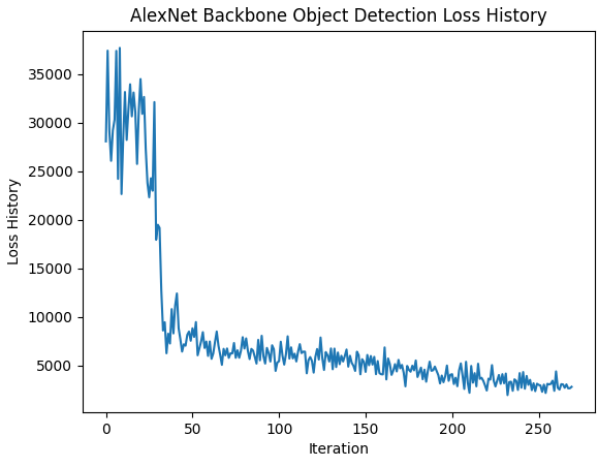
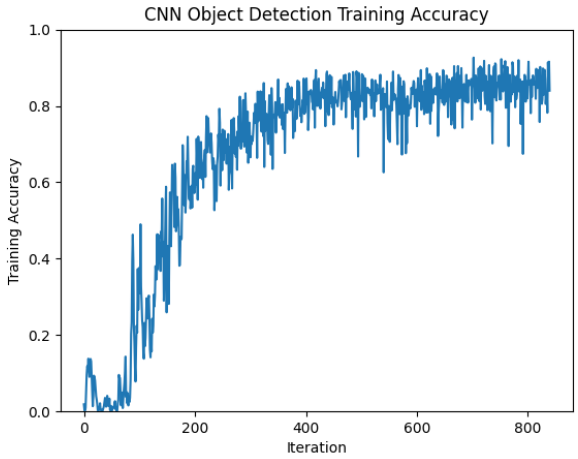
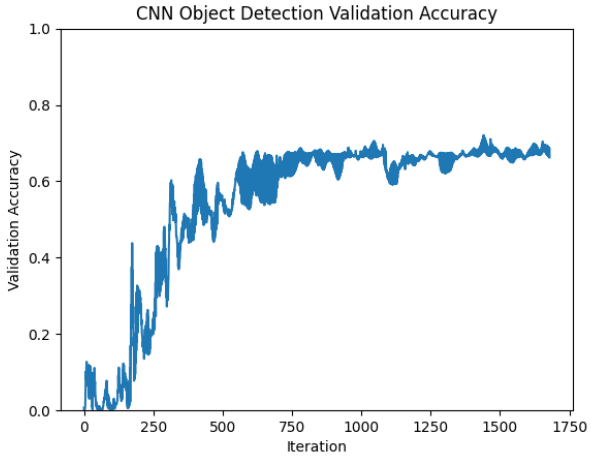


Figure 9. Accuracies and Loss for AlexNet





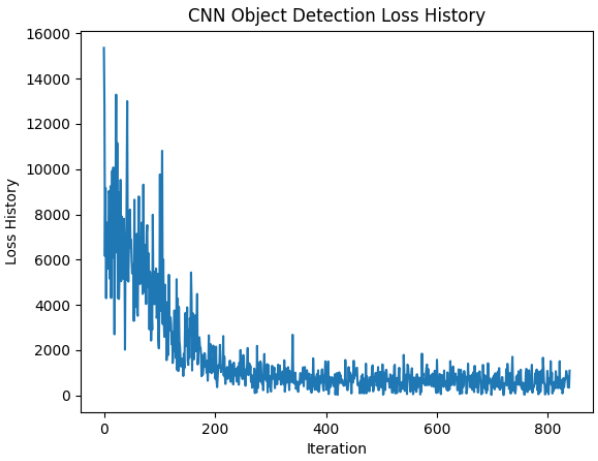


Figure 10. Accuracies and Loss for ConvNet

puter vision in self-driving cars, we elucidate the efficacy of various architectures on problems faced in autonomous driving. From the binary classification problem, we can see that AlexNet's sensitivity to the boundary of images (as displayed in our heat maps) helped it best succeed in this classification process. For the object detection problem, we see that the CNN gave the highest results, of an accuracy of around 0.6. Additionally, though ResNet 18 is very power-

Figure 11. AlexNet Backbone's prediction for an image in the COD dataset

ful in classification settings, they did not perform as well as the vanilla CNNs in object detection.

We emphasize that near-perfect accuracy is necessary for this field. With many other uses of machine learning, minor errors may be negligible. If ChatGPT made an error in summarizing a paper, this may be inconvenient for the user but nevertheless relatively harmless. However, in the context of self-driving cars, these tiny chances of error still have the potential to cause considerable degrees of harm. If a self-driving car is unable to properly classify a human, for example, this may result in the car running into humans or other vehicles on accident, causing unimaginable harm. As such, it's reassuring that our most advanced models were able to attain near-perfect validation accuracy.

This issue is what our future works could aim to address. How can we improve these models to say with certain that such harmful mistakes will be avoided? How can we determine when a model is strong enough to be operating on a car driving on a road?

Another future direction we could explore is real-time video analysis. Real self-driving cars would need to interpret video from the world rather than just one stationary image. It would be rather illuminating to test the ability of current models to identify and detect objects on a moving road, and maybe even calculate the speed at which certain objects are moving in real time.

Though our experiments show relatively high success in the current ability of models to classify and detect objects relevant to self-driving cars, we can clearly see that there is much room for progress. By identifying the gaps in our research and building upon what we've discovered through our experiments, we come one step closer to making self-driving cars a reality for society.

7. Authors and Contributions

Eric Chen, Emily Xia, Bruno Dumont: These authors contributed equally to this work, though each focusing on different areas.

Eric Chen set up the hardware used for the project (AWS EC2 GPUs). He created the code for running the models as well as the architecture for the models themselves. He ran all of the models and collected their loss history, and training/validation accuracies. He collaborated with Bruno Dumont to draw heat maps for the various models on the binary classification task.

Emily Xia found, downloaded, processed, and combined the two datasets for the object detection problem. She also trained models for the binary classification problem and generated graphs and plots for both parts of the methods. Additionally, she wrote parts of the paper (literature review, dataset, methodology, conclusion).

Bruno Dumont worked on writing the methods part of the paper, half of the literature review, and part of the results. He also worked on developing some of the models used in both tasks and creating the accuracy function for object detection.

References

[1] S. S. Kumar, K. S. Kumar, G. Prakasha, H. Teja, V. Shrinidhi, *et al.*, "Self-driving car using neural networks and computer vision," in *2022 International Interdisciplinary Humanitarian Conference for Sustainability (IIHC)*, pp. 1200–1204, IEEE, 2022.

[2] A. S. Rao, S. Sapna, T. Akshay, A. S. Shenoy, B. Adithya, and A. Dias, "Identification of car make and model using deep learning and computer vision techniques," in *2022 International Conference on Artificial Intelligence and Data Engineering (AIDE)*, pp. 202–207, IEEE, 2022.

[3] S. A. Khan, H. J. Lee, and H. Lim, "Enhancing object detection in self-driving cars using a hybrid approach," *Electronics*, vol. 12, no. 13, p. 2768, 2023.

[4] B. Dincer, "Vehicle detection image set." `https://www.kaggle.com/datasets/brsdincer/vehicle-detection-image-set/data`.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, p. 84–90, may 2017.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[8] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Mi-

crosoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755, Springer, 2014.

[9] E. Zhang, "Car object detection." https://www.kaggle.com/datasets/sshikamaru/car-object-detection/discussion/366832.

[10] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.