Robotics and Autonomous Systems 142 (2021) 103780

Contents lists available at ScienceDirect

Robotics and Autonomous Systems

journal homepage: www.elsevier.com/locate/robot

# A controlled investigation of behaviorally-cloned deep neural network behaviors in an autonomous steering task

Michael Teti [a,*], William Edward Hahn [a], Shawn Martin [b], Christopher Teti [c], Elan Barenholtz [a]

[a] Center for Complex Systems and Brain Sciences, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, USA
[b] College of Computer and Information Science, Northeastern University, 360 Huntington Ave, Boston, MA 02115, USA
[c] Department of Ocean and Mechanical Engineering, Florida Atlantic University, 777 Glades Road, Boca Raton, FL 33431, USA

ARTICLE INFO

*Article history:*
Received 24 December 2019
Received in revised form 28 August 2020
Accepted 29 March 2021
Available online 19 April 2021

*Keywords:*
End-to-end control systems
Deep neural networks
Autonomous vehicles
Imitation learning
Behavioral cloning

ABSTRACT

Imitation learning (IL) is a popular method used to train machine learning models that are capable of acting on their environment based on expert examples. Two types of IL models are inverse reinforcement learning (IRL) and behavioral cloning (BC). Models trained under IRL traditionally perform better than those trained under BC due to compounding covariate shift associated with the latter, which typically requires algorithms such as DAGGer to help compensate for this. More recently, however, deep learning architectures with increased generalization performance have been developed, which may help to alleviate the problem of compounding covariate shift and allow researchers to take advantage of the simplicity of BC. Despite these developments, recent studies on BC in sub-scale autonomous robots employ relatively primitive convolutional networks without such tools as batch normalization and skip connections, and it is difficult to judge their networks' performance relative to others due to drastically different training and testing conditions. Here, we examine how an array of artificial neural networks, chosen to reflect more recent architectural choices available, behave in a highly controlled IL task – navigating around a small, indoor racetrack – upon being embedded in a sub-scale RC vehicle as an end-to-end steering system. For our main findings, we report the lap completion rate and path smoothness of each network under the exact same conditions as it controls the vehicle on the track. To supplement these findings, we also measure each network's bias toward the distribution of the training actions and develop a method to highlight regions of a given input image that are deemed 'important' to a given network. We observe that most of the more recent neural networks perform reasonably well during testing, as opposed to the more primitive networks which did not perform as well. For these reasons and others, we identify VGG-16 and AlexNet – out of the networks tested here – as attractive candidate architectures for such tasks.



## 1. Introduction

Imitation learning (IL) is a common paradigm used to train autonomous robotic vehicles in recent years. Under the IL umbrella there exist two common approaches: 'inverse reinforcement learning' (IRL) [1,2] and 'behavioral cloning' (BC) [3]. IRL attempts to learn a reward function under which an expert's behavior is optimal. This approach prioritizes entire trajectories over others [4], and, thus, has been shown to be relatively immune to compounding covariate shift [5,6], in which deviations between the training and test set accrue as an autonomous robot's trajectory becomes increasingly different from the expert's trajectory it was trained on. One disadvantage of this class of models, however, is that they are typically susceptible to erroneously attributing certain instantaneous actions as stemming from a long-term goal, particularly when there is a long time horizon. Furthermore, when training IRL models, care must often be taken to prevent the model from converging on degenerate solutions, which can lead to catastrophic failures.

BC, which is an alternative class of IL models that predates IRL, uses supervised learning in order to associate states and actions as provided by an 'expert'. While relatively simple to train [4], models trained under BC are highly vulnerable to the problem of compounding covariate shift. Typically, these models show a failure to generalize from the human-generated dataset to the novel trajectories, leading to poor performance at test time. As a result, deployed BC typically relies on methods such as Dataset Aggregation (DAGGer) [6], or one of its variants, in which the expert watches the learner perform the task for some brief time

* Corresponding author.
*E-mail address:* mteti@fau.edu (M. Teti).

https://doi.org/10.1016/j.robot.2021.103780
0921-8890/

while recording its preferred action at every timestep for the learner to train on later [7]. Of course, DAGGer requires substantially more supervision than standard BC which can significantly increase training time and effort.

Of these two classes of IL models, IRL models typically outperform BC models due to BC's compounding covariate shift problem, and, as a result, IRL has steadily become the more popular IL algorithm in recent decades. Over roughly the same timeline, artificial neural networks (ANNs) – particularly Deep Convolutional Neural Networks (DCNNs) – have become an extremely popular machine learning algorithm for image classification, feature extraction, object detection, and reinforcement learning, among others. Consequently, most recent instantiations of IL use a DCNN under an IRL framework, and, thus, DCNNs trained under a BC framework are relatively scarce. However, larger, more complex, and more powerful networks that are capable of increased generalization and representational ability have been developed over the last few years. These modern architectures (e.g. Inception, VGG, ResNet, etc.) may exhibit an increased ability to generalize to new inputs and, possibly as a result, decreased susceptibility to the problem of compounding covariate shift which plagued earlier models trained under BC. This would allow those developing IL systems in autonomous robots to take advantage of the simplicity of BC for certain tasks and may warrant a second look at the BC paradigm.

Implementations using an ANN trained under BC to map sensor data (e.g. frames captured by a forward-facing camera) to a steering value in a real-world robot are few in number[1] [3,8–13]. These implementations exhibited good driving performance, but all of them reported the performance of just a single hand-designed ANN of relatively primitive design (we discuss this more thoroughly in Section 2). These custom ANNs used in these studies were presumably designed for the particular environment and task at hand, which differed drastically between studies. As a result, it is difficult to get an accurate assessment of how an architecture deployed in one study performed relative to that in another. Therefore, we set out to perform a systematic comparison among contemporary models in an autonomous task, something that we have yet to find reported in the literature on behavioral cloning in robotics. In addition, it is unknown how well more recent architectures (e.g. VGG, ResNet, Inception, etc.) might perform while controlling a sub-scale robot, particularly in relation to competing architectures, or how specific architectural choices that are currently available (e.g. batch normalization, skip connections, etc.) influence this performance.

In contrast, the present study, which is intended to complement these previous works, observed how each architecture performed in the exact same conditions, which should allow for a more accurate estimate of each one's behavior relative to the others and provide insight into what particular factors of the environment influence this behavior. Specifically, we investigate the performance of several state-of-the-art neural network architectures in a basic imitation learning task in which a small, indoor robot is trained to successfully complete a lap on a simple path using the input from a monocular camera. The models were trained on steering decisions of human operators who navigated the track. This simple paradigm allowed us to compare behavior in a *highly controlled* setting with clear metrics of performance, something that has been lacking in much of the literature. We compared several performance metrics – including lap completion rate, path smoothness, and input pixel importance – of multiple network architectures, which were chosen to reflect the diverse types of architectures employed in recent years (as well as some older ones). We employed a straightforward BC approach to training the models; the critical variable was the choice of learning architecture. These included: (1) a three hidden layer fully-connected network, (2) a simple convolutional neural network (CNN) with two convolutional/pooling layers followed by two fully-connected layers, (3) AlexNet, [14], (4) a version of VGG-16 [15] with a slightly reduced number of filters in each layer, (5) Inception-V3 [16], (6) a version of the ResNet architecture [17] which we refer to as ResNet-26, and (7) a Long Short-Term Memory (LSTM) network [18]. The details of each network are described below in detail. Each network's architecture, as well as the training procedure and videos of each network steering the vehicle autonomously over different trials can be viewed at https://github.com/mpcrlab/DNN_Rover.

A sub-goal of the current study was to determine how different types of image modalities compare to each other in end-to-end training of an autonomous robotics system. For example, how helpful is it to include color information, which typically includes more information than grayscale? To assess this, we included three input types used as training and test data for the different models: (1) single grayscale video frame, (2) single color video frame, and (3) the current grayscale video frame plus several grayscale video frames from previous timepoints, concatenated along the channel dimension (which we term 'framestack' throughout the rest of this report) as input to each different network. These three input classes were chosen to determine whether spatial, color, or temporal information is more useful for such tasks, a consideration when designing low-power, smaller systems that may not be able to afford to utilize all three feature modalities. The framestack approach provides a method for including temporal sequence information in a simpler manner than typical approaches, such as recurrent neural networks. This allowed us to test the role of temporal information, as [12] recommend in the conclusion of their study, without modifying the structure of the network relative to the other input types.

## 2. Related work

The development of an autonomous robot that can learn, end-to-end, from human driving behavior has been a highly sought goal for decades. Here, we discuss some of the significant historical developments toward this goal. The ALVINN project of Pomerleau [3] in the late 1980s was the first encouraging example of such a vehicle. They used an ANN, consisting of a single, fully-connected hidden layer to map inputs from a camera and laser range finder to a continuous steering angle. This model was used to successfully drive a full-sized vehicle on limited road conditions, a remarkable achievement for its time, though at a very slow speed for public roads. This was probably due to the limited computing resources available at that time, which also influenced the size of the ANN they were able to use. In addition, the ALVINN project, and some others to follow, did not include detailed analysis of the vehicle's behavior/performance and instead simply showed the vehicle driving on various roads.

The next significant contribution was LeCun et al.'s [9] early 2000s DARPA Autonomous Vehicle, or DAVE, a sub-scale, radio-controlled (RC) car that was trained with human driving data to output a discrete left, right, or forward steering direction given an input from two forward-facing cameras. The goal of this project was to develop a vehicle that could navigate autonomously through outdoor terrain containing many different obstacles using a CNN that consisted of four hidden layers. Similar to the current study, their vehicle was designed to perform three discrete actions at a constant speed: move forward, turn left, and turn right. However, similar to the ALVINN project, the goals of

[1] Although [3] and [8] employ full-scale vehicles, the networks used were trained via behavioral cloning and the task was similar to other previous works mentioned. Therefore, we believe they are relevant enough to include here.

DAVE were to determine whether it was possible for any neural network to successfully navigate a vehicle, rather than comparing different architectures. The project was ultimately deemed unsuccessful on the terrain tests, as the vehicle had difficulty navigating through previously unseen environments and coping with noisy sensor data. This is likely due to the fact that the ANN used in DAVE – although larger and more advanced than that used in ALVINN – was still extremely small in comparison to contemporary models. Lecun et al. do not report any results concerning DAVE's performance in the actual obstacle course after training, only the classification scores from the validation set. As a result, it is difficult to assess their network's efficacy in a BC setting where the vehicle's actions are changing the subsequent inputs.

Building on the work of [9], Bojarski et al. [8] developed DAVE-2, a full-scale vehicle equipped with their PilotNet CNN [19], that took in input images from three forward-facing cameras and output a continuous steering angle. PilotNet, which consisted of five convolutional layers followed by three fully-connected layers, was designed by empirically trying different layer configurations and observing the relative performance on the training set [19]. After PilotNet was trained, it was first tested on videos collected from the vehicle, and once it passed these simulation tests, it was deployed in the vehicle on the road. To analyze the performance of their network on the road, they computed the percent of time out of total time driven the vehicle was being controlled autonomously (i.e. a human was not intervening), which they estimate at approximately 98%. They also utilized the method developed in [20] to illustrate the most salient portions of a given input image in determining steering angles. While the performance and behavior of DAVE-2 was investigated more fully than that of ALVINN and DAVE, the level of analysis of driving performance was limited to a simple measure of successful and unsuccessful driving, based on the determination of the human operator.

Pan et al. [10] used a sub-scale RC car to compare driving behavior of 'online' (training with DAGGer) vs 'batch' (training without DAGGer) training of a custom CNN, which mapped input images and current wheel speeds to a steering angle and throttle value. The vehicle was trained and tested on the same oval-shaped dirt track, and the authors reported the average number of times it was able to complete one minute of driving around the track out of the total number of test runs. The online training methods all achieved 100% success rates, while the batch methods all did significantly worse. This study was similar to the current approach in that it included a controlled investigation into two different training algorithms. The critical difference is that Pan et al. report the performance of a single, custom ANN trained with DAGGer vs. without DAGGer, while we report the performance of multiple different ANNs under the same training regime and with the hope of providing insight into specific architectural choices that influenced the robot's performance.

More recently, Codevilla et al. [11] deployed a sub-scale vehicle controlled by a CNN in different urban environments. As in [9] and the present study, the vehicle was capable of performing three discrete actions. The network used in this work was composed of 8 convolutional layers followed by two fully-connected layers. It was trained to map input images from three forward-facing cameras, as in [8], to a discrete steering direction and a throttle value. In parallel to this CNN, two separate fully-connected networks took as input sensor measurements (e.g. wheel speeds, acceleration, etc.), as well as a 'command' vector which represented short-term goals. The outputs of the CNN and the two fully-connected networks were then combined into a single feature vector and sent to three further fully-connected layers to determine the appropriate action. Since the main contribution of this work was to introduce the concept of the command vector, the results reported compare the performance relative to different methods of incorporating this vector into the network under the same conditions as the training set.

The work of Toromanoff et al. [12] was designed to observe the performance of a DCNN – inspired by the network in [8] and of similar architecture – as it controlled a full-scale vehicle around a custom test track containing obstacles. Like the work of [3] and [8], the DCNN was designed to output a continuous steering angle from an input image, which they obtain with a single fisheye camera. Before deploying the DCNN in the vehicle on the test track, they performed multiple tests on a simulation until adequate performance was reached. They also performed two significant experiments using this simulation regarding the quality of training data used to train their DCNN. Namely, they observe the effect the bias present in the label distribution (i.e. most steering angles are very small due to the vehicle traveling straight an inordinate amount of the time) has on the DCNN's ability to control the vehicle once trained, and they observe how effectual bagging – averaging the weights of networks each trained on different datasets – can be on the simulation task as well. Upon testing the DCNN in the vehicle on the test track, they reported that the DCNN was very sensitive to the camera calibration and performed poorly at first, but, after adjusting for this, the vehicle was successfully driven on a few private roads. To conclude, they speculate that more recent advancements in ANNs, such as batch normalization, skip connections, and temporal inputs, could possibly be utilized in future work to increase the performance of their network. Like many of the other studies discussed above, no analysis of the driving behavior on the actual track was provided; only performance on the simulation was reported — which evidently differed greatly from the actual conditions, judging by the initial performance on the track relative to the simulation. The current study emphasizes the need for a controlled, systematic investigation into the behavior of such algorithms when they are embedded in actual vehicles, not just in simulation.

Most recently, Chowdhuri et al. [13] used a sub-scale RC vehicle to develop a multi-modal, multi-task DCNN, which they call Z2Color. Their network, which consisted of two convolution-max pooling-batch normalization layers followed by two fully-connected layers, was designed to take as input 4 images – one current and one past image from two forward-facing cameras – and output a throttle value and continuous steering angle. The main contribution of this work was to incorporate different behavioral priors into the decision-making process of the DCNN so that the steering/throttle value that was output would depend on that behavioral prior. To evaluate their Z2Color network's performance while controlling the vehicle in different suburban and offroad environments, they use the same metric as in [8] and report 92.68% average autonomy on these tests vs. 84.27% autonomy for the multi-task network they built upon.

Every one of the DNNs used in these previous works could be considered primitive in at least one way. For example, all of these used a much more shallow DNN than what would be considered standard at the time of their development. Furthermore, DNN layers such as batch normalization and skip connections, which are considered standard in current DNNs, were used by one [13] and none of the studies above, respectively. This alone is not necessarily an issue, as the networks employed performed reasonably well in their respective tests. However, it is unknown whether more recent network architectures containing these types of tools would have performed better or would not have needed additional training algorithms such as DAGGer to reach adequate performance, as none of these studies tested their architecture against standard ones. For example, the network used by Chowdhuri et al. [13] contained only two convolutional layers and two fully-connected layers, which was small even compared to the

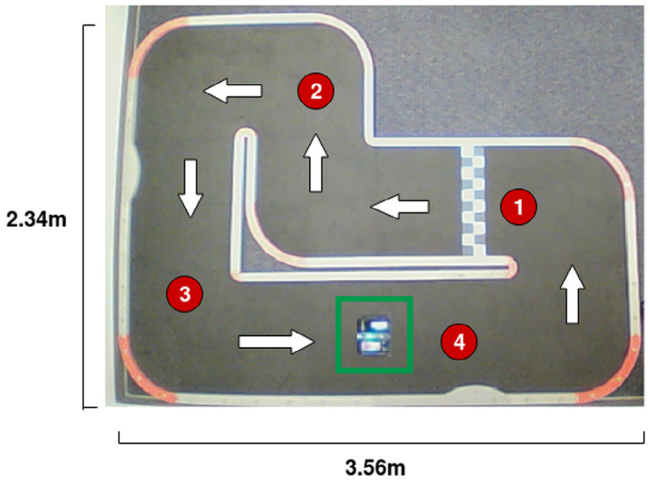


**Fig. 1.** The track used to train and test each network. The vehicle was trained and tested on its ability to navigate the track successfully in the direction indicated by the white arrows. The four test positions are indicated by the red circles, and the vehicle is contained within the green box. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

rest of the studies highlighted here, yet, this network seemed to obtain similar or better performance than many of the others. Perhaps this might be due to the inclusion of batch normalization layers into the network developed in that study, as no other study described above used that technique, or perhaps it was due to their use of DAGGer, which some of the other studies did not use. Before building more complex and elaborate control algorithms/networks centered around a particular architecture – as was done in some of the studies above – it is first necessary to investigate how different types of network architectures will behave relative to other architectures while performing the same exact task.

## 3. Methods

### 3.1. Experimental setup

To test each network architecture in an autonomous navigation task, we used a 3.56 m × 2.34 m L-shaped foam racetrack [21] and a Brookstone Rover 2.0 [22] (Fig. 1). Each 240 × 320 color video frame (Fig. 2) collected by the vehicle's single, built-in camera (which was set to collect 30 frames per second) was sent over Wi-Fi[2] to a computer running Ubuntu 18.04 with two GeForce GTX 1080 TI GPUs, 16GB of DDR4 DRAM, and an Intel(R) Core(TM) i5-6500 CPU (Fig. 3).

### 3.2. Training protocol

To create a supervised dataset on which to train each network, multiple people drove the vehicle a single direction around the track (Fig. 1). We recorded each video frame along with the action – left pivot, right pivot, forward, or backward – the human performed at that frame. The dataset, which totaled approximately 250,000 frames and their respective labels, was composed of a validation dataset of ∼7000 frames, which was taken from a completely separate test run than those used in training, and a training dataset of ∼243,000 frames. The validation set was used to test the network every 100th training iteration.

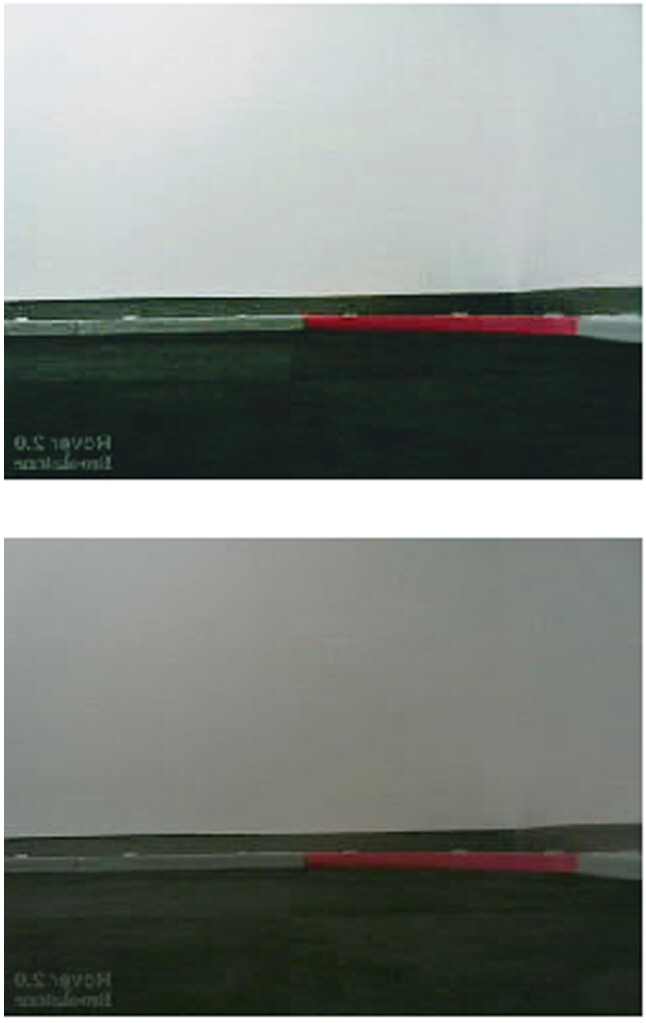

**Fig. 2.** A sample frame from the vehicle's camera taken approximately from its position in Fig. 1. The top frame was taken under the high-light condition, and the bottom was taken under the low-light condition. As can be seen in these images, the vehicle's camera was very narrow and could only capture the scene directly in front of it, which added difficulty to the task. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

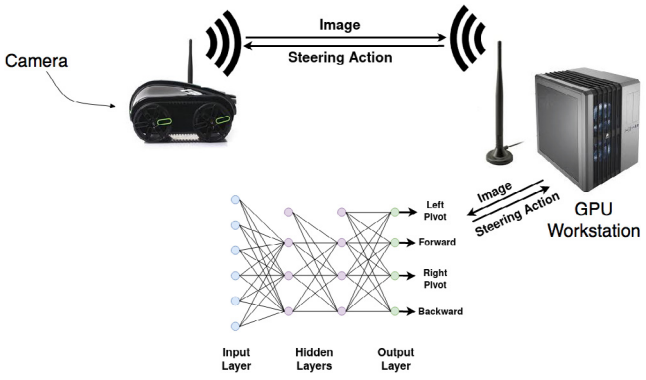


**Fig. 3.** A diagram of the setup. The robot sent its camera images to the GPU workstation, where each image was input to the model to obtain the output steering action, which was sent back to the robot.

[2] To ensure that latency was not an issue, the round-trip time was monitored throughout this study, and it was typically within 15–30 ms, exceeding 35 ms occasionally.

Each network was tested and trained on these same validation and training sets for 6000 training iterations. The number of training iterations was chosen such that slower-learning networks would have sufficient training time to learn while still controlling for the number of training iterations, as many networks converged well before this point but did not overfit (as determined by the relation between the training and validation loss curves over training). Each training iteration began with a random video frame and the subsequent 79 video frames (80 in total) from the training set.

Once the batch was randomly chosen from the training set, each frame was cropped by removing the top 110 rows (making the images 130 × 320 × 3 pixels) like in [12], and further operations were performed depending on the image processing method being employed. These operations are described below:

#### 3.2.1. Grayscale

For the video frames in the grayscale method, each frame was made grayscale and instance normalization [23] was subsequently employed on each individual frame.

#### 3.2.2. Color

For the color class, instance normalization [23] was used on each color frame.

#### 3.2.3. Grayscale framestack

Each video frame in this method began with the same operations as in the grayscale class. Each $\text{frame}_t$ in the batch was then paired with the $\text{frame}_{t-5}$ and $\text{frame}_{t-15}$ along the channel dimension. The human action at $\text{frame}_t$ was used as the label for the framestack training example. The intervals were chosen empirically by trying many different values and observing how well the trained vehicle navigated the track. These intervals are likely dependent, at least to some extent, on the frame rate of the camera(s), as well as the top speed of the vehicle. There is some existing research in which temporal correlations in video data were exploited in a similar manner [24].

After performing the appropriate operations, each image was copied, and white noise was added to each of the copies,[3] such that the network would get both an unaltered and noisy version of the same image with the same label in a single training batch. The peak signal-to-noise ratio (PSNR) was computed over 500 random frames and their noise-augmented counterparts, and the average for each frame was 10.0 dB. The batch was then sent to the neural network to continue the training iteration. Each network's weights were optimized using Tensorflow's Adam optimizer and a learning rate of 3e-5.

### 3.3. Architectures tested

We tested seven different models, described below. (A description of each model's layer architecture is included in Table 2). For the CNNs besides ResNet-26, these were initialized with random values from a uniform distribution without scaling variance as described in [25], and weights in all fully-connected layers were initialized with random values taken from a truncated normal distribution as in [26]. The weights in the convolutional filters in ResNet-26 were initialized with random values such that the variance of the inputs would be constant as in all other networks' convolution layers, but, instead of taking these values from a uniform distribution, they were taken from a truncated normal distribution as in [27]. A weight decay [28] of 0.001 was employed in all convolution and fully-connected layers. Dropout [29] of fully-connected nodes, which is used to reduce overfitting, was utilized in all networks except ResNet-26 with different dropout probabilities depending on the architecture, as described in more detail below. The output layer of every network consisted of four fully-connected nodes and a softmax activation function [30].

#### 3.3.1. Fully-connected network

Perhaps the most basic deep artificial neural network, the fully-connected network consists of the input layer, three hidden layers, and the output layer. The input layer contains 124,800 input nodes for color images (130 × 320 × 3). Each hidden layer contains 64 nodes, which are each connected to every node in the previous and subsequent layers, with a hyperbolic tangent activation function [31], $\ell_2$ regularization [32], and weight decay of 0.001 [28]. Each layer of the network has a dropout probability of 0.5 during training and 0.0 for testing.

#### 3.3.2. 2-layer CNN

This architecture is perhaps one of the simplest convolutional networks, with just two convolutional layers and two fully-connected hidden layers. $\ell_2$ regularization [32] is used in both convolution layers. After each convolution layer, 2 × 2 max pooling [33,34] with stride 2 and local response normalization [14], which encourages sparsity, were applied. Dropout was used on both fully-connected layers with a probability of 0.5 for training and 0.0 for testing.

#### 3.3.3. Alexnet

The AlexNet architecture, which was developed around 2011 by Krizhevsky et al. [35] and is rather simple by contemporary standards, consists of five convolutional layers followed by two fully-connected layers. The overall architecture was hand-tailored to the ImageNet dataset [36], and Krizhevsky et al. note that alterations to it, such as removing a single convolutional layer, resulted in a significant drop in accuracy. AlexNet uses max pooling after the first two convolution layers and local response normalization, which allows nodes in different feature maps but at the same spatial location to compete based on their activations. The result of this is that the features become decorrelated, and it has also been shown that this layer helps regularize the weights. Dropout was used in both fully-connected layers with a probability of 0.5 during training and 0.0 during testing.

#### 3.3.4. VGG-16

The VGG architecture [15] attempts to address the issue of choosing different stride and filter sizes based on a particular dataset by using a stride of one and a filter size of 3 × 3 for all convolution layers. Thus, this style of architecture reduces the number of hyper-parameters – despite its greater depth – than AlexNet by stacking "building blocks of the same shape . . . which increases simplicity and reduces the chance of over-adapting the architecture to a specific dataset" [37]. The VGG-16 architecture contains 13 convolutional layers overall, with max-pooling layers following the 2$^{\text{nd}}$, 4th, 7th, 10th, and 13th layers. VGG-16, like AlexNet, also features two fully-connected hidden layers immediately before the output layer, but, unlike AlexNet, these layers use a rectified linear activation function. Each of these fully-connected layers use a dropout probability of 0.5 during training and 0.0 during testing. This architecture is rather large in terms of the number of parameters even by today's standards. Here, for ease of computation, we slightly reduce the number of filters and nodes in the convolutional and fully-connected layers, respectively.

[3] We also tried to augment each data batch by flipping each image with respect to the vertical axis and changing the label accordingly, but this caused the vehicle's movement to be less continuous and its accuracy worse.

#### 3.3.5. Inception-v3

In contrast to the VGG-style architectures, the Inception architecture [16,38,39] contains hand-crafted topologies with many varying hyper-parameters while still exhibiting low model complexity [37]. These architectures, including Inception-V3, which we use here, all operate on the principle of splitting the feature map outputs of certain layers into multiple different streams of operations (represented by the dashed lines in Table 2) and subsequently merging their outputs together via concatenation. This network also employs global average pooling after the last convolutional layer before the fully-connected output layer. Dropout is applied to the feature vector produced by global average pooling with a probability of 0.4 during training and 0.0 during testing.

#### 3.3.6. Resnet-26

The ResNet architecture [17] builds off of the splitting/merging strategy of the Inception architectures and the simple, block-template style of VGG nets. These networks are composed of "residual blocks", where a template of convolutions is repeated a set number of times, and after each repetition, the features that served as input to that specific repetition are added to the output of the repetition. After every convolutional layer, batch normalization [40] and a rectified linear activation function [41] are applied to the output (except when adding the identity of the previous block, in which the activation function comes after the addition). Like Inception-V3, this network uses global average pooling on the last convolutional layer, which helps to reduce the number of weights drastically without loss of performance. The model complexity, measured in FLOPs and number of parameters, of ResNets is extremely low relative to other CNNs (Table 2).

#### 3.3.7. LSTM

Each node in a typical LSTM network has four gates: input, output, and input modulation which use the sigmoid activation function as in [42], and the forget gate which uses the hyperbolic tangent activation function [31]. These gates work in conjunction with each other to help regulate which information enters the cell state, which is able to hold long-term dependencies, and a hidden state, which captures the short-term dependencies. The typical input for such networks is an $m \times n$ matrix, where each row is the next timestep in the data and each column is a dimension in those timesteps. Here, we treat each image as this matrix, as though the rows of the image are the timesteps with $320 \times 3$ dimensions for color and grayscale framestack images and 320 dimensions for grayscale images. We do this by concatenating the input channels along the column dimension.[4] The network we use here has two hidden LSTM layers each containing 500 nodes, where each layer is essentially comprised of four fully-connected layers representing each gate. Each node in the first of these hidden layers outputs a sequence of hidden states corresponding to the number of time-steps (image rows), whereas each node in the second hidden layer returns one output for the whole sequence it was given.

### 3.4. Testing protocol

After each network completed training, it was then used to control the vehicle autonomously at a constant speed around the track. To measure the performance of each network, the vehicle was placed at four different positions around the track (Fig. 1) and driven autonomously by the respective network for 10 trials at each position, totaling 40 test trials per network. The reason for starting the vehicle at four separate positions on the track was to ensure that the performance was not dependent on a single, arbitrary starting position, as our pilot trials indicated that the vehicle's trajectory around the track was often distinct up to some degree when the vehicle was set to start at one position vs another, even when the same network was controlling it.

Every time the testing position was changed (every 10 trials), the vehicle's batteries, which were new and unused at the start of this research, were replaced with fully-charged ones. A camera, which was fixed to the ceiling and faced down toward the track, was used to film each test run. Along with this video, the time of each trial was taken, and it was recorded whether the vehicle successfully completed a single lap or not. A trial was ended when one of four circumstances occurred: (1) the vehicle completed a lap and made it back to its starting position, (2) the vehicle turned around and went the wrong direction three times in the same trial (most models eventually turned back around and righted themselves when this occurred), (3) the vehicle hit the wall and/or became stuck, or (4) the vehicle became stuck in an oscillatory back-and-forth motion without making net progress on the track for 10 consecutive seconds.

Using the protocol above, each network was tested in two separate test trials. The first testing trial, which we hereafter refer to as Lighting Tests, was designed to isolate performance differences based only on differences in brightness, which is important for autonomous robots. During the Lighting Tests, the same track shape that was used in the training data was also used in the testing phase, but five of the ten trials at each position were performed under a high-light condition (all lights on; Fig. 2, top) and five with a low-light condition (one-third of the lights on, Fig. 2, bottom).[5]

In the second testing trial, which we term Lighting and Object Tests, each network was tested under the same protocol but only using the input image method that enabled it to obtain the best performance in the Lighting Tests (e.g. grayscale, color, or framestack). During these tests, the entire layout of the room was rearranged (such that different objects were in the vehicle's point-of-view but beyond the track's walls). The track was also configured in an oval shape instead of the L-shape, four diverse objects (pictured in Fig. 4) which were not present in the training data were placed randomly on the track,[6] and a different vehicle of the same make and model of the training vehicle was used. Finally, just as in the Lighting Tests, five trials were under a high-light condition and five under a low-light condition. For each of these test trials, a random number generator was used to determine (1) the number of objects placed on the track, (2) how far along the lap each object should be placed, (3) where each object was positioned relative to the middle of the path, and (4) how much the object was rotated. All of these parameters regarding object placement were consistent across all networks tested. This test phase was performed to try and capture the base level of performance each network could achieve while operating under relatively different conditions than what it was trained on.

Since our experiments are performed in an extremely controlled environment, we are able to control these variables (e.g. brightness levels, object placement, etc.) precisely and test each model under the *exact same* conditions, which would have been extremely difficult or even impossible in a typical outdoor setting.

### 3.5. Lap completion rate

To measure lap completion rate, or success rate, during the testing phases, we use the equation,

$$\text{lap completion rate} = \frac{\#\text{ of trials with lap completed}}{40} \tag{1}$$

[4] It is worth noting that we initially attempted concatenation along the row dimension with poor results.

[5] The training dataset contained only a mid-light condition.

[6] During training no objects were placed on the track.


**Fig. 4.** The four objects used in the second testing trial of this research.

This was used as the primary metric to determine how each network performed on this task.

### 3.6. Path analysis

To further explore each network's effect on the vehicle's path, we used the videos taken by the ceiling camera and employed an object-tracking algorithm to determine the location of the center of the vehicle during every trial in the Lighting Tests.[7] These coordinates were used to indicate where the vehicle had traveled over its 40 trials and its location at each timestep during each trial.

One metric that may be used to compare autonomous performance across different models is the smoothness of the trajectory. In autonomous robots, a smoother path is generally preferable in terms of the efficiency of a vehicle. In addition, path smoothness could effect our measures of performance as well as low path smoothness may cause a given network to become more susceptible to compounding covariate shift because greater path fluctuations may increase the chance that the vehicle will enter into a novel path. Alternatively, lack of smoothness could *result* in compounding covariate shift, as a vehicle that has entered into a novel path will likely exhibit less smooth driving behavior as it is forced to predict the steering angle from a novel image.

To this end, we developed a metric which we show can be interpreted as the smoothness of a trajectory given the coordinates obtained with the object-tracking algorithm, which we refer to as lateral movement (as it is not a strict measure of smoothness). This method uses using visual information because our vehicle did not support the addition of an accelerometer (although it could easily be modified to measure smoothness from an accelerometer), and, since our vehicle was designed to operate indoors, we were able to affix a camera to the ceiling and capture the entire path. First, we manually determined the coordinates at the center of the entire path along the track, which served as a reference to measure the position of the vehicle as it traversed the track. Next, the points along the vehicle's trajectory were ordered by the timestep in which they were captured by the overhead camera. The euclidean distance between the center of the vehicle and the minimum distance to the reference path was then calculated at each timestep. Finally, we calculated the mean absolute difference between the distance value at each timestep $t$ and that from timesteps $t-1$ through $t-30$, which indicates how much the vehicle's path fluctuated side-to-side during this time. We choose to look 30 frames back because the overhead camera's frame rate was 30 fps, which meant that these lateral movements were computed over one second intervals. We chose this interval – which corresponded to about 0.6m of movement – because it was short enough that it would capture sharp side-to-side movements, and it would not capture natural drift over longer distances. Since we are taking the difference between the distance from reference (essentially how much the vehicle changed laterally) at time t and that over the recent past, an abrupt change in lateral position in a short time span would cause this metric to be higher. Conversely, if the vehicle's position did not change very much laterally over the recent past, this value would be lower.

### 3.7. Pixel importance

In order to gain further insight into the observed differences between tested models, we assessed which portions of the image each model deemed more important in order to make its behavioral decisions. To do so, we utilized a novel method loosely based on [43] in which we systematically 'flipped' the values of each pixel value in the input images, one by one, and observed the corresponding difference in the model outputs compared to the unaltered image. This serves to determine what pixels/regions of the image were more important in the network's classification. This method bears some similarity to other recently developed methods designed to make neural networks more interpretable — often using methods such as deconvolutional layers [44] or layer-wise relevance propagation (LRP) [45]. However, these other methods do not generalize well to all neural network architectures and their results are not always readily interpretable [46]. The method introduced here provides a simple and easily interpretable method for localizing the information in images across all models. The procedure was performed over 50 randomly-selected images from the validation set and consisted of five steps: (1) preprocess and perform inference on a given test image and record the output layer values and the action with the highest output value, (2) loop through every pixel in the image and maximally flip its value (i.e. pixel values $\geq$128 were given the value 0 and those $<$128 were given the value 255), (3) perform preprocessing and inference on the image with the altered pixel, and (4) calculate the MSE between the output layer values associated with the altered image and the unaltered image and record it along with the altered pixel's location. We use the MSE for each pixel location in step 4 to create a heatmap in the image space that illustrated how important each pixel of the image was in changing the output. During each iteration of the loop, we also record whether the flipped pixel at the respective location caused the network to output a different action than when it was given the unaltered image.

In a separate calculation from the heatmap generation, we report the mean softmax probability of the chosen action given 5000 random, unperturbed input images. This allowed us to roughly observe how easy it might have been to change a particular network's output action when the perturbed images were input.

### 3.8. Pruning analysis

When developing modern DCNNs with the intent of embedding them into mobile or edge applications, the power consumption and size of the network is of great concern due to the resource constraints of current edge processors (e.g. FPGA, NVIDIA Jetson, etc.). As a result, various methods of decreasing

[7] We did not film the Lighting and Object Tests.

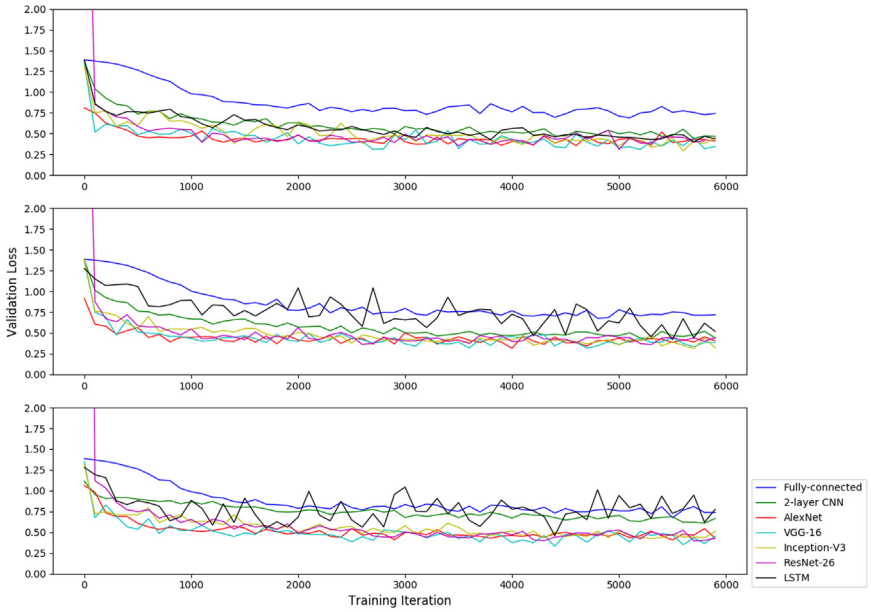


**Fig. 5.** Each network's validation loss over training with single grayscale frame (top), single color frame (middle), and grayscale framestack (bottom) as input. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

the size and latency of trained networks have been developed, such as weight quantization and/or pruning, designing more efficient networks, and knowledge distillation (for a recent review, see [47]). Recent evidence indicates that it is often possible to prune the vast majority of a network's weights with a minor loss in performance, and these pruned sub-networks can typically be identified even before any training [48] or after training without the need for retraining once the network is pruned [49]. Here, we use the Triangular pruning method proposed in [49] to prune each network that was tested in the Lighting and Object Tests. We choose $T_{min}$ and $T_{max}$ such that the difference in validation accuracy was $\leq$5% *and* the maximum change in any output node's softmax activation did not change by more than a value of 0.2 on average over the validation set when each network was given the same image. By ensuring the output layers are similar between the pruned and un-pruned network when given the same images, this method also resembles knowledge distillation techniques, where a smaller model is made to approximate the outputs of a larger model. Thus, we were able to find the sub-network that was "driving" the output of the original network.

### 3.9. Action bias

One factor that is influential on generalization ability is the ability of a network to avoid overfitting the training set. This is made more difficult when the frequencies of the labels – steering actions is in this case – in the training set are very unbalanced (e.g. the forward action appeared much more often than any other action in the training set). To address this, Toromanoff et al. [12] introduced a method of randomly discarding different percentages of the straight steering angles to make the dataset more balanced. Here, we do not take any preventative measures to help relieve this massive imbalance, as we would like to understand whether certain networks are able to handle this imbalance and achieve relatively good generalization performance without such measures. To this end, we examined the bias weights of the output layer compared against the actual distribution of labels in the dataset. The bias weight of a node can be thought of as the node's output in the absence of any external input, either through zeroed-out weights or inputs. By looking at these bias weights of the output layer (where each node represents a certain steering action), we were able to observe how active each of these nodes would be (i.e. how likely it would be for a certain action to be predicted relative to the other actions) before any input was sent through the network. Ideally, the distribution of these four output bias weights should not be too similar to the distribution of the actions in the training set. If they were, this could be a possible indication that the network overfit to the training set and, thus, would have trouble generalizing to an environment or path with a different distribution of actions.

### 3.10. Inference rate

Finally, we observe the average number of inferences per second each network performed on images of the size we use. This was calculated by sending 1000 images – which were already loaded in an array in Python – sequentially to the network to predict the action, obtaining the elapsed time over all images, and dividing by 1000. This procedure was performed five times for each network, and the average of these five trials is reported. This method was intended to isolate just the latency introduced by the model, so the predicted actions were not used to steer the vehicle and input frames were not collected by the camera.

## 4. Results

### 4.1. Validation performance

Fig. 5 shows the validation loss of each model over the entire course of training for each of the three input types. For the single grayscale frame and single color frame conditions (top and middle), all of the models converged to moderate losses except for the fully-connected network, which yielded a loss that was approximately 2$\times$ higher upon model convergence than all others. For the grayscale framestack input (bottom), the four contemporary CNNs show significantly reduced loss upon convergence compared with the three other models.

The confusion matrices computed from the validation set did not differ very dramatically between models, with the exception of the fully-connected network and 2-layer CNN (Fig. 6). One

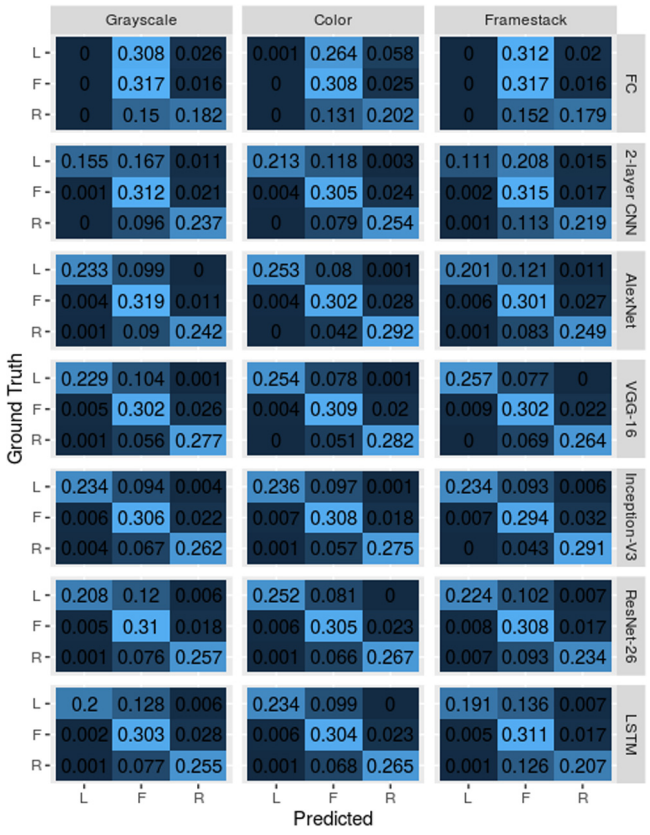


**Fig. 6.** Confusion matrices computed from the validation set.

thing that stands out in these confusion matrices is that in all networks the highest mis-classified pair of actions was when the ground truth action was "left" and the predicted action was "forward", which is probably due to the high class imbalance between the two. During training, we did not place much emphasis on the validation accuracy because the validation set was made with several different drivers, many of whom had a different driving style/strategy. This meant that there were certain points on the track (often the turns) which different drivers handled differently, and, as a result there were validation images that were roughly the same but had different ground truth actions.

By observing the t-SNE [50] embeddings of each network's last hidden layer activations when given images from the validation set, there appears to be a good degree of variability between networks regarding the separation between classes (Fig. 7). For example, the embeddings for each action were highly overlapping with respect to the fully-connected network and 2-layer CNN, while in Inception-V3, LSTM, and ResNet-26 there was a moderate amount of separation. Finally, AlexNet and VGG-16 exhibited the most separation between classes.

### 4.2. Lap completion rate

#### 4.2.1. Lighting tests

Fig. 8 (top) presents the lap completion rate (i.e. the percentage of trials in which the lap was completed) across all of the tested architectures during the Lighting Tests. Overall, the convolutional neural networks and LSTM vastly outperformed the fully-connected network. Within the contemporary CNNs, most of them achieved reasonably good success rates (∼95%) with at least one data type, with the exception of ResNet-26. However, only AlexNet, trained on single color video frames, achieved a perfect success rate over 40 trials. VGG-16 was found to be the most robust to the input class, as its lap completion rate was equally high for all three input image types.

Across the different data types, the color single frame yielded the best overall performance across models while the grayscale framestack yielded dramatically worse performance across most models.

#### 4.2.2. Lighting and object tests

The success rates achieved by each network in the Lighting and Object Tests were much lower than those in the Lighting Tests (Fig. 8, bottom). AlexNet exhibited the best success rate during this phase, completing 55% of the 40 laps, followed by VGG-16 which completed 45% of the 40 test laps (Fig. 8, bottom). ResNet-26 only completed 25%, or 10, of its test laps.

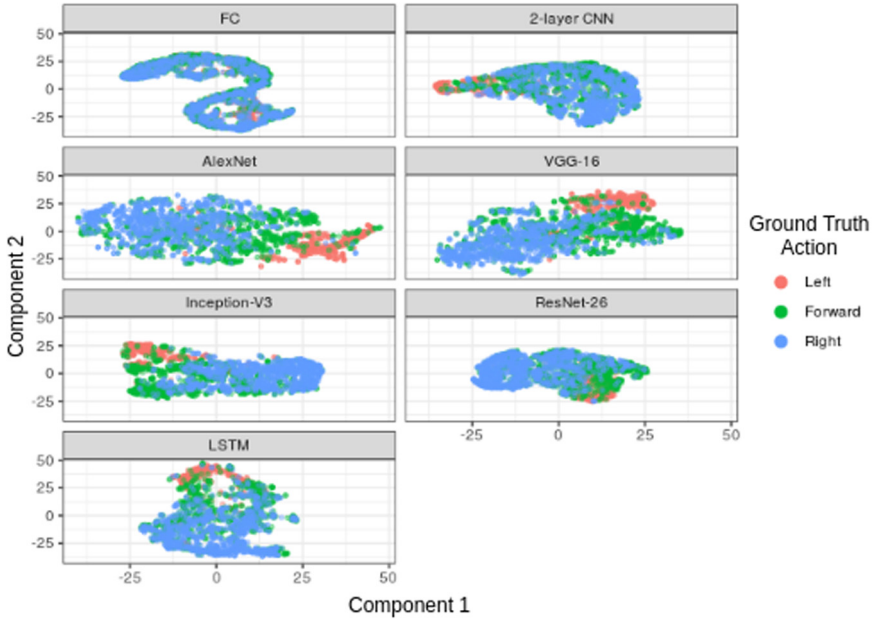


**Fig. 7.** t-SNE representations of each network's activations at the last hidden layer given random images from the validation set. Component 1 and Component 2 refer to the two t-SNE components the activations were mapped onto.

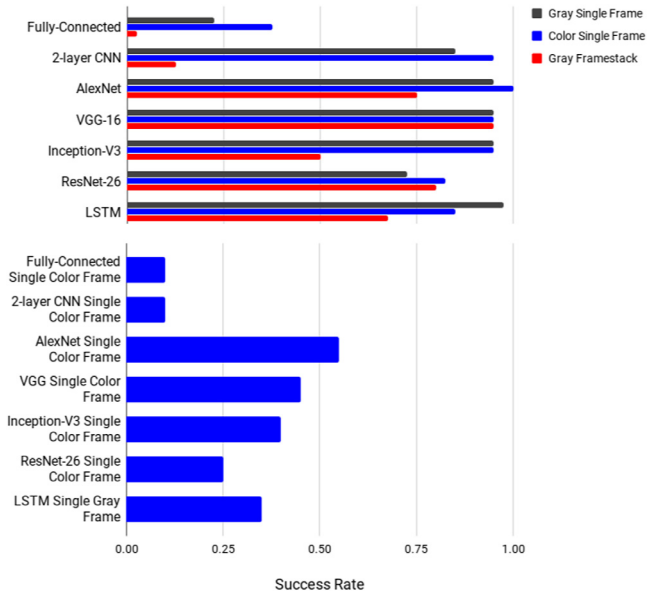


**Fig. 8.** The success rate of each network during the Lighting Tests (top) and the Lighting and Object Tests (bottom).

**Table 1**
Summary of spatial information analysis.

| Network | % Altered actions | Output MSE | Softmax Prob. |
|---|---|---|---|
| FC | 0.00% | $4.08 \times 10^{-6}$ | 0.80 |
| 2-layer CNN | 0.07% | $4.00 \times 10^{-4}$ | 0.84 |
| AlexNet | 2.11% | $1.30 \times 10^{-3}$ | 0.91 |
| VGG-16 | 2.11% | $2.00 \times 10^{-3}$ | 0.92 |
| Inception-V3 | 1.07% | $6.30 \times 10^{-3}$ | 0.91 |
| ResNet-26 | 0.64% | $1.00 \times 10^{-3}$ | 0.85 |
| LSTM | 0.00% | $9.87 \times 10^{-6}$ | 0.86 |

### 4.3. Validation loss and success rate

Figs. 9 and 10 show how the validation loss of a particular model was related to the performance in the Lighting Tests and Lighting and Object Tests, respectively. As can be seen, many model/input types with similar validation losses (those between .4 and .45) demonstrate widely variable success rates in both the Lighting Tests and the Lighting and Object Tests. For example, Inception-V3 trained on grayscale framestack input had the same validation loss as VGG-16 trained on the same input, but the former's success rate in the Lighting Tests was 50% and the latter's was 95%. Furthermore, AlexNet trained on single color frames achieved perfect success in the Lighting Tests, yet this network had one of the worst validation losses (Fig. 9). Similar conclusions for the Lighting and Object Tests can be drawn in Fig. 10, as many of the same models (i.e. 2-layer CNN, LSTM, ResNet-26, and AlexNet) performed very differently despite having a similar mean validation loss of ∼0.43.

### 4.4. Path analysis

An illustration of representative paths taken by highlighted networks during a single trial can be seen in Fig. 11.

Fig. 12 depicts the results of our Lateral Movement analysis. The fully-connected network and Inception-V3 had the highest mean lateral movement over all trials in the Lighting Test, while the LSTM and ResNet-26 exhibited the smallest lateral movement on average. VGG-16 and AlexNet, which were the best-performing networks in both tests, as well as the 2-layer

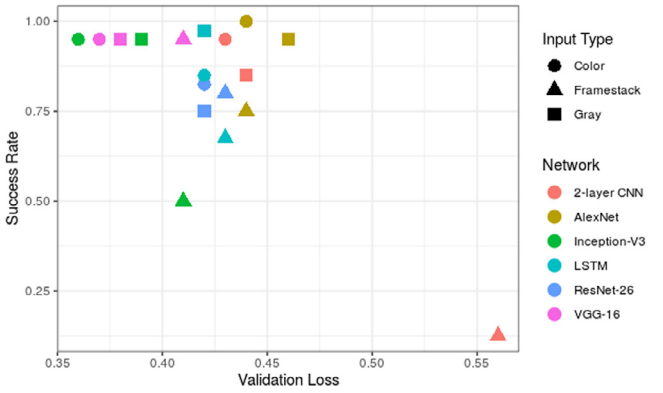


**Fig. 9.** The success rate during the Lighting Tests as a function of the validation loss achieved by each network (fully-connected network not shown).

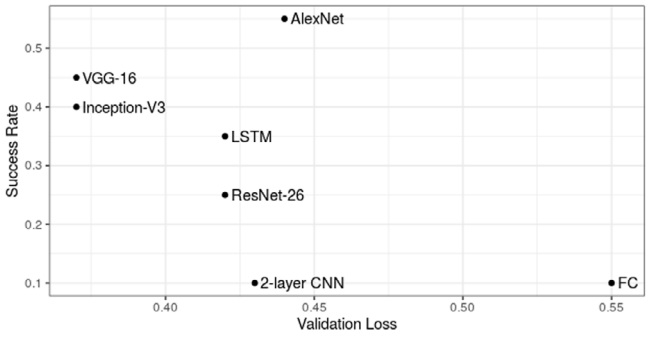


**Fig. 10.** Success rate of each network in the Lighting and Object Tests as a function of the validation loss achieved by each network.

CNN all exhibited moderate lateral movement on average, and they are likely not significantly different judging by the error bars.

We were also able to observe how the vehicle's lateral movement over a given trial run (Fig. 13), and how it changed based on the starting position. As is evident from these plots, many of the models exhibited similar amounts of lateral movement in their paths. The fully-connected network and Inception-V3 typically traded off as the network with the highest lateral movement at different points in time. Specifically, Inception-V3's lateral movement increased dramatically toward the end of the trials that started at position 1, and it also exhibited much higher lateral movement throughout the first half of trials beginning at position 3. It also appears that this was influenced by specific points on the track, because Inception-V3's fluctuations occur at relatively confined timestep intervals, as opposed to those exhibited by the fully-connected, which are typically higher over the entire trial.

### 4.5. Pixel importance

Table 1 illustrates the results of the pixel flipping analysis described above. The action decisions of the two non-convolutional networks – fully-connected and LSTM – were completely unaffected by the flipped pixels (i.e. no pixel flip caused the network to change its action decision). Regarding the convolutional networks tested, the models that exhibited higher performance in both the Lighting Tests and the Lighting and Object Tests (i.e. AlexNet, and VGG-16) generally had more action decisions changed due to flipped pixels. Furthermore, the MSE between the output layers associated with the image containing the altered pixel and the unaltered image showed a similar trend, as the fully-connected and LSTM networks showed little difference. The convolutional networks, on average, contained MSEs that were

**Table 2**
A detailed description of the architecture of each network used in this research and its number of FLOPs and parameters.

| Fully Connected | 2-layer CNN | AlexNet | VGG-16 | Inception-V3 | ResNet-26 | LSTM |
|---|---|---|---|---|---|---|
| 64-d fc, tanh, L2 reg. | 3 × 3, 32, relu, L2 reg. | 11 × 11, 96, stride 4, relu | 3 × 3, 45, relu | 7 × 7, 64, stride 2, relu | 7 × 7, 32, stride 2, relu | 500-d LSTM, keep seq. |
| dropout, 0.5 | 2 × 2 max pool, stride 2 | 3 × 3 max pool, stride 2 | 3 × 3 max pool, stride 2 | 3 × 3 max pool, stride 2 | 3 × 3 max pool, stride 2 | 500-d LSTM |
| 64-d fc, tanh, L2 reg. | Local response normalization | Local response normalization | 2 × 2 max pool, stride 2 | Local response normalization | batch normalization | 4-d fc, softmax |
| dropout, 0.5 | | | 3 × 3, 120, relu | 1 × 1, 64, relu | residual block | |
| 64-d fc, tanh, L2 reg. | 3 × 3, 64, relu, L2 reg. | 5 × 5, 256, relu | 3 × 3, 120 | 3 × 3, 192 | | |
| dropout, 0.5 | 2 × 2 max pool, stride 2 | 3 × 3 max pool, stride 2 | 2 × 2 max pool, stride 2 | Local response normalization | [3×3, 32, relu; 3×3, 32, relu] × 4 | |
| 4-d fc, softmax | Local response normalization | 3 × 3, 384, relu | 3 × 3, 200, relu | 3 × 3 max pool, stride 2 | | |
| | | 3 × 3, 384, relu | 3 × 3, 200, relu | 1 × 1, 64, relu / 1 × 1, 96, relu / 1 × 1, 16, relu / 3 × 3 max pool, stride 1 | residual block | |
| | 128-d fc, tanh | 3 × 3, 256, relu | 3 × 3, 200, relu | 3 × 3, 128, relu / 5 × 5, 32, relu / 1 × 1, 32, relu | | |
| | dropout, 0.5 | 3 × 3 max pool, stride 2 | 2 × 2 max pool, stride 2 | Concatenate channels | [3×3, 64, relu; 3×3, 64, relu] × 4 | |
| | 256-d fc, tanh | Local response normalization | 3 × 3, 450, relu | 1 × 1, 128, relu / 1 × 1, 128, relu / 1 × 1, 32, relu / 3 × 3 max pool stride 1 | | |
| | dropout, 0.5 | | 3 × 3, 450 | 3 × 3, 192, relu / 5 × 5, 96, relu / 1 × 1, 64, relu | residual block | |
| | 4-d fc, softmax | 4096-d fc, tanh | 3 × 3, 450, relu | Concatenate channels | | |
| | | dropout, 0.5 | 2 × 2 max pool, stride 2 | 3 × 3 max pool, stride 2 | [3×3, 128, relu; 3×3, 128, relu] × 4 | |
| | | 4096-d fc, tanh | 3 × 3, 450, relu | 1 × 1, 192, relu / 1 × 1, 96, relu / 1 × 1, 16, relu / 3 × 3 max pool, stride 2 | | |
| | | dropout, 0.5 | 3 × 3, 450 | 3 × 3, 208, relu / 5 × 5, 48, relu / 1 × 1, 64, relu | Global avg pool | |
| | | 4-d fc, softmax | 3 × 3, 450, relu | Concatenate channels | 4-d fc, softmax | |
| | | | 2 × 2 max pool, stride 2 | 1 × 1, 160, relu / 1 × 1, 112, relu / 1 × 1, 24, relu / 3 × 3 max pool stride 1 | | |
| | | | 3500-d fc, relu | 3 × 3, 224, relu / 5 × 5, 64, relu / 1 × 1, 64, relu | | |
| | | | dropout, 0.5 | Concatenate channels | | |
| | | | 3500-d fc, relu | 1 × 1, 128, relu / 1 × 1, 128, relu / 1 × 1, 24, relu / 3 × 3 max pool stride 1 | | |
| | | | dropout, 0.5 | 3 × 3, 256, relu / 5 × 5, 64, relu / 1 × 1, 64, relu | | |
| | | | 4-d fc, softmax | Concatenate channels | | |
| | | | | 1 × 1, 112, relu / 1 × 1, 144, relu / 1 × 1, 32, relu / 3 × 3 max pool stride 1 | | |
| | | | | 3 × 3, 288, relu / 5 × 5, 64, relu / 1 × 1, 64, relu | | |
| | | | | Concatenate channels | | |
| | | | | 1 × 1, 256, relu / 1 × 1, 160, relu / 1 × 1, 32, relu / 3 × 3 max pool stride 1 | | |
| | | | | 3 × 3, 320, relu / 5 × 5, 128, relu / 1 × 1, 128, relu | | |
| | | | | Concatenate channels | | |
| | | | | 3 × 3 max pool, stride 2 | | |
| | | | | 1 × 1, 256, relu / 1 × 1, 160, relu / 1 × 1, 32, relu / 3 × 3 max pool, stride 1 | | |
| | | | | 3 × 3, 320, relu / 5 × 5, 128, relu / 1 × 1, 128, relu | | |
| | | | | Concatenate channels | | |
| | | | | 1 × 1, 384, relu / 1 × 1, 192, relu / 1 × 1, 48, relu / 3 × 3 max pool, stride 1 | | |
| | | | | 3 × 3, 384, relu / 5 × 5, 128, relu / 1 × 1, 128, relu | | |
| | | | | Concatenate channels | | |
| | | | | Global avg. pool | | |
| | | | | 4-d fc, softmax | | |
| $8.0 \times 10^6$ params | $2.17 \times 10^7$ params | $7.30 \times 10^7$ params | $1.02 \times 10^8$ params | $6.74 \times 10^6$ params | $1.46 \times 10^6$ params | $4.93 \times 10^6$ params |
| $5.60 \times 10^7$ FLOPs | $5.46 \times 10^8$ FLOPs | $2.42 \times 10^9$ FLOPs | $1.87 \times 10^{10}$ FLOPs | $2.84 \times 10^9$ FLOPs | $1.06 \times 10^9$ FLOPs | $1.29 \times 10^9$ FLOPs |

much greater than the fully-connected-based networks. Although the convolutional networks had more actions altered when given altered images, the output softmax probability in these networks was much higher on average, indicating that these networks were more 'confident' in their outputs.

Fig. 14 depicts representative examples of some of the heatmaps constructed for single images using the MSE values for each pixel when flipped. Although VGG-16 (Fig. 14, top left) and AlexNet (Fig. 14, top right) had the most action decisions affected by flipped pixels per image on average, these pixels tended to lie in a very confined region of the image. This was not true for the LSTM (middle left) or the 2-layer CNN (middle right), as the information they attended to was fairly distributed, although the 2-layer CNN was affected more by pixels that did not correspond to 'useful' features of the scene (e.g. objects outside of the track or parts of the room's wall such as the rubber baseboard). One important observation is that the heatmaps generated with ResNet-26 were often localized to features peculiar to this particular track and setup (i.e. the pink color of the track's walls on the turns) while for the other convolutional networks (i.e. AlexNet, VGG-16, and Inception-V3) this occurred less often. This indicates that the ResNet-26 architecture would likely be less able to generalize to new tracks and/or settings, which is in contrast to these other networks whose heatmaps are localized on more general features such as the ground in front of them or the edge where the floor meets the track wall.

### 4.6. Pruning analysis

The distribution of each network's weights before pruning was performed can be seen in Fig. 15 while Fig. 16 shows the per-and post . We observed the highest amount of pruning with VGG-16 (67% pruned) and the lowest with the Fully-Connected network (21%). There was a generally positive relationship between the number of weights pruned and the original number of weights in a network, although Inception-V3 and ResNet-26 had around 50% of their weights pruned despite their low number of weights to begin with. By looking at Fig. 16, there does not seem to be much of a correlation between the number of weights in the network (either pruned or unpruned) and the success rate of a given network. For example, VGG-16 and AlexNet had the most weights and performed the best, but the 2-layer CNN and fully-connected network had the second and third most weights, respectively, but were lower-performing networks.

### 4.7. Action bias

While AlexNet and VGG-16, which exhibited good performance relative to others, had output bias weight distributions that were relatively distinct from the distribution of actions in the training set, so did the 2-layer CNN and fully-connected network, which did not perform as well 17. On the other hand, Inception-V3 trained on color images had a distribution that was relatively similar to the action distribution in the training set when compared to other networks, although it appeared less similar than the Inception-V3 network trained on framestack inputs and at most as similar as the Inception-V3 network trained on grayscale frames. For example, the output bias weights for the 'forward' and 'right' nodes in the Inception-V3 network using grayscale inputs were closer to the proportions of 'forward' and 'right' actions, respectively, in the training set than they were in the Inception-V3 network using color inputs. In the ResNet-26 network, the bias weight corresponding to the 'forward' output node was much higher than the other bias weights in this layer (and most similar to the proportion of 'forward' actions in the training set), even when compared to most other networks. Overall, these results illustrate that examination of the output bias weights is one possible way to identify overfitting for applications where the training label distribution is highly-skewed such as this one, but it is obviously not the only predictor of overfitting, as there are many possible causes of this.

### 4.8. Inference rate

The Fully-Connected network was able to perform the most inferences $s^{-1}$ by far (729.83 inferences $s^{-1}$ for color images; Fig. 18), while the LSTM performed just 18 inferences $s^{-1}$ on images with three channels. Generally, the larger, more advanced networks exhibited decreased inference rates than the smaller, more primitive ones as would be expected (with the exception of ResNet-26 due to its relatively efficient architecture).

## 5. Discussion

Recent work on autonomous vehicles using ANNs trained under BC indicates that compounding covariate shift might not be as great a problem for larger ANN architectures or ones that use more recent developments like batch normalization. Inspired by these findings, the current study presents the first systematic assessment and comparison of multiple neural network models in an experimentally-controlled BC autonomous steering task. Overall, we observe that a network's performance on these tests is influenced by a combination of different factors, specifically path smoothness and output bias. We also find that a given network's performance is not closely related to the shear number of weights in the network, indicating that more specific architectural choices may have more of an impact. Below, we first discuss some more general observations from our experiments and delve into their possible causes before examining each network individually in an attempt to provide some insight into each one's performance.

### 5.1. Framestack inputs exhibit high complexity

One of the main, and somewhat surprising, observations is that the framestack data did not improve performance but actually impaired it. We believe that this poor performance may be due to the fact that a stack of three frames may carry more complexity than a single frame or three color channels, which may require networks to have greater representational ability. To test this, we calculated the Structural Similarity Index (SSIM) [51] — which is commonly used as a measure of image similarity — between each channel in the color images and each channel in the framestack images respectively, averaged the three, then took the average over 1000 random images. The average SSIM between color image channels was 0.94, and the average SSIM between framestack channels was 0.68. Therefore, it is possible to conclude that the gray framestack images performed poorly relative to the other image types, and extremely poorly in some of the simpler networks, because the channels in this type of image are not as correlated as those in color images, adding complexity. This is due to the fact that it is impossible to drive exactly the same as in the training dataset, meaning the past frames will never be exactly the same as those the network was trained on, and the network will need to generalize even further to overcome this. This is why the confusion matrices given framestack inputs are not much worse than those using the other inputs, but the performance on the track given these inputs is far worse. Furthermore, this is also why most of the more recent, contemporary networks perform better than the fully-connected network and 2-layer CNN when using these inputs. In future work, CNN-LSTM networks could be used to compare different architectures on spatio-temporal inputs rather than by using the framestack approach we used

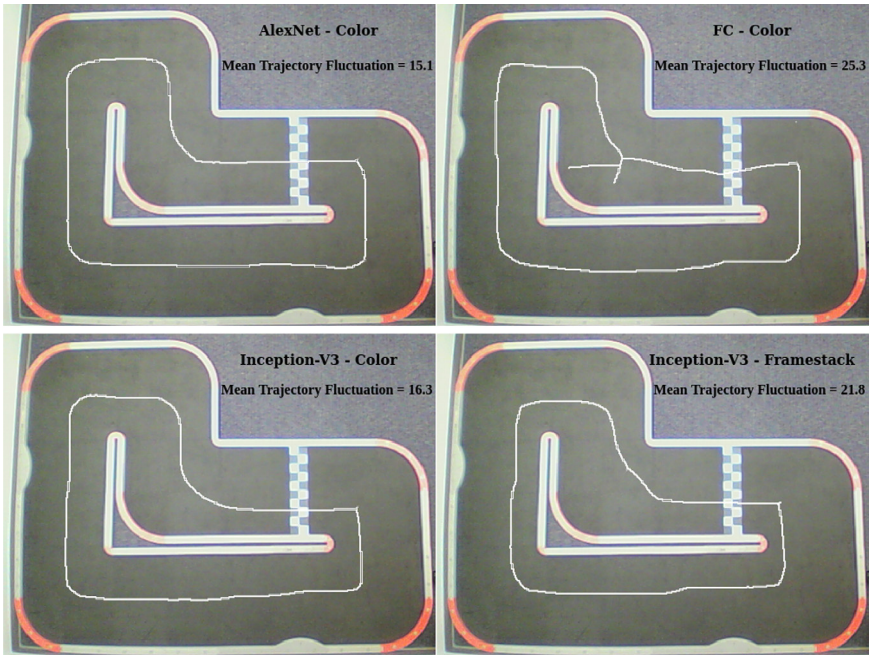


**Fig. 11.** Each pane shows the path of the vehicle as it was driven by the listed model during a single trial of the Lighting Test. Mean trajectory fluctuation numbers were computed with the method described in the trajectory fluctuation methods.

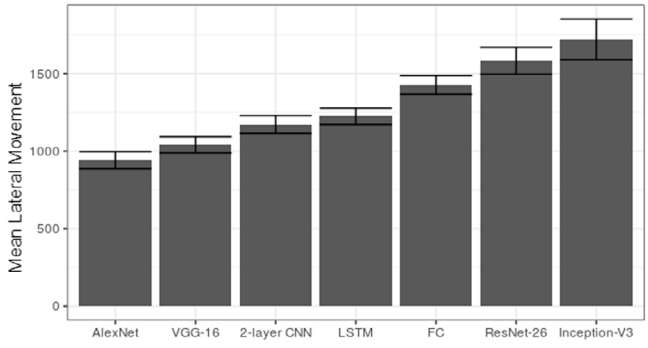


**Fig. 12.** Mean lateral movement of each network tested in the Lighting and Object Tests as defined in Section 3.6. The error bars represent standard errors.

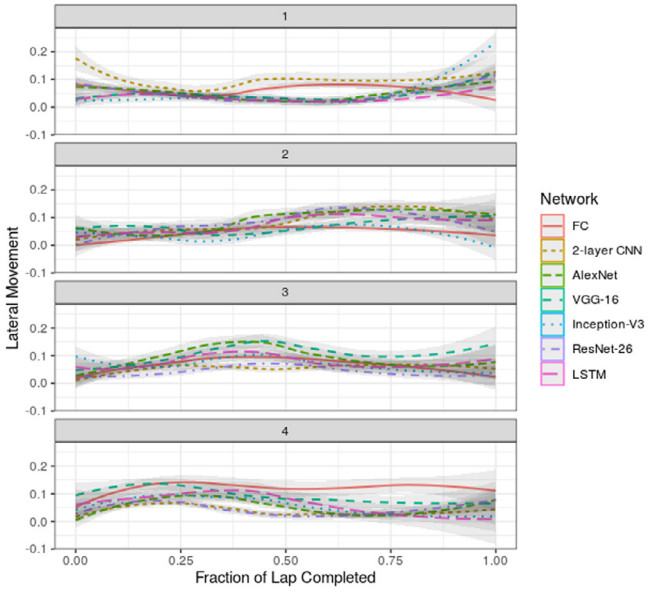


**Fig. 13.** Lateral movement (as calculated via the procedure outlined in 3.6) over the track length for all trials in the Lighting Tests. Shaded areas represent 95% confidence intervals. The data is plotted with ggplot's loess function [52].

here. This would probably lead to better performance on this input type because the temporal patterns will be learned on the features extracted by the CNN as opposed to the raw input images.

### 5.2. Fully-connected and 2-layer CNN

Given the architecture, size, and validation loss/accuracy of this network (as illustrated by its loss curve and confusion matrix), its poor performance is not particularly surprising. This network's confusion matrix indicates that it often predicted a steering action of left or right at points when it was supposed to go straight, which would either cause it to turn completely around and start going the wrong direction, hit the wall, or travel on a path with decreased smoothness (all three of which would greatly increase the chance of a failed trial). Furthermore, Fig. 7 indicates that the last hidden layer's activations were completely overlapping for all classes. Based on these results, it is rather obvious that the fully-connected architecture here did not contain the necessary size/complexity capable of performing reasonably well at this task. Specifically, the extreme bottleneck from the input layer to the first hidden layer probably led to high information loss and decreased generalization ability, something which has been studied in bottleneck autoencoders [53]. In [8, 10–12], and [13] this bottleneck (i.e. the ratio of nodes in the flattened fully-connected layer to those in the fully-connected layer) was around 1, 50, 137,[8] 1, and 5, respectively, while in this architecture, it was 1950.

[8] This network not only took in raw images, but low-dimensional representations in a parallel stream. It is possible the network may have learned to rely more on the low-dimensional representation than the raw image, which would allow for a higher bottleneck, but this was not examined.

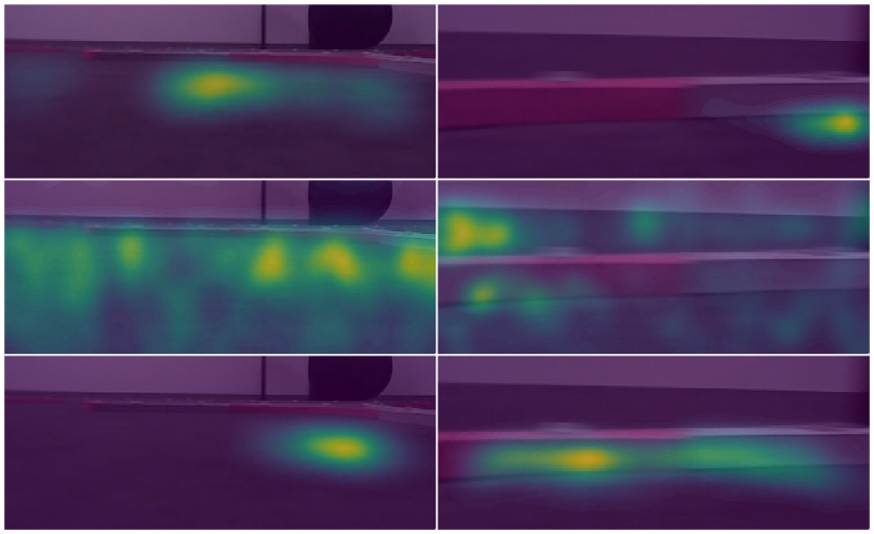

**Fig. 14.** Representative heatmaps depicting how much each pixel, when maximally flipped, changed the output of the network. The three heatmaps in the left column were created using the same image where the desired action was forward and the networks displayed are VGG-16 (top), LSTM (middle), and Inception-V3 (bottom). The three in the right column were all created from an image which had a desired action of right, and the networks displayed are AlexNet (top), 2-layer CNN (middle), and ResNet-26 (bottom). Each individual heatmap's values were scaled between 0 and 255, so intensities cannot be compared between heatmaps here. Note: the black shape present in the top right corner in the three heatmaps in the left column was an object placed outside of the track and not related to the heatmap generation. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

The 2-layer CNN architecture's performance was around average or better regarding many of the observed metrics – including validation loss/accuracy, lateral movement, pixel flipping, and Lighting Test success rate – despite being relatively small and shallow compared to most other networks. One observation which stands out is the wide disparity between its performance on the Lighting Tests and that on the Lighting and Object Tests. This network has some similarities to the fully-connected network, both in terms of the number of parameters (Fig. 16) but also in the information flow within it. With just two convolutional layers, the first fully-connected layer in this network – which contained 128 nodes – received a tensor of size $32 \times 64 \times 64$ (or 131,072 when flattened). This extreme bottleneck was of size 1024, and, like the fully connected network, it most likely lead to high information loss. The network was able to overcome this bottleneck (and its effects) in the presence of different lighting conditions, but it appeared to be too drastic to allow for good performance in the Lighting and Object Test. Furthermore, the bottleneck in this network and the fully-connected network were similar, but the one present in this network was less severe due to the sizes of the pre and post layers where the bottleneck occurred, as well as the fact that the inputs to the post layer were filtered by two convolutional layers in this network as opposed to raw input values as in the fully-connected network. This probably afforded the 2-layer CNN additional performance – relative to the fully-connected network – during the Lighting Tests, and it might explain why these two networks exhibited very similar behavior, both quantitatively (number of flipped pixels that altered an action, mean softmax probability, and Lighting and Object Test success rate) and qualitatively (focusing on irrelevant regions in the heatmaps).

### 5.3. Alexnet and VGG-16

The AlexNet and VGG-16 architectures were the two highest performing networks in both the Lighting Tests and the Lighting and Object Tests, and their behavior was very similar in many other ways as well. Architecturally, these two networks most closely resemble DCNN architectures used in recent studies under an imitation learning paradigm (e.g. [8,10–13]). Specifically, the

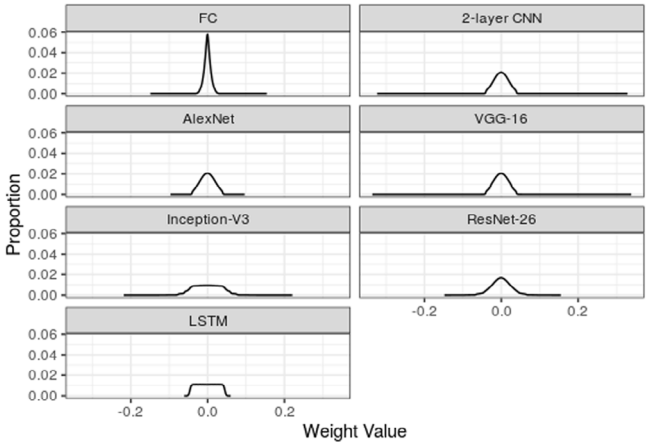


**Fig. 15.** Weight distributions of each network in the Lighting and Object Tests.

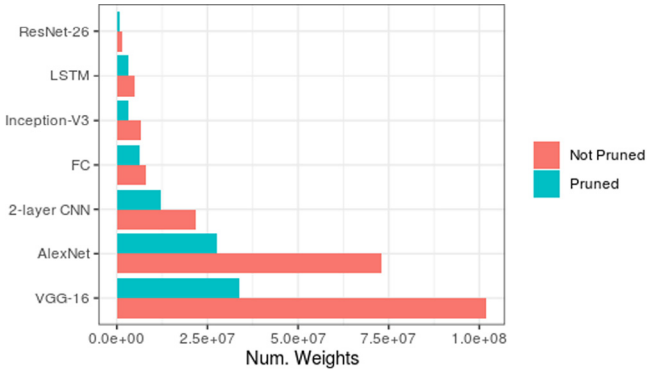


**Fig. 16.** The number of weights in the un-pruned networks and the corresponding pruned networks which were used in the Lighting and Object Tests.

architectures used in these studies contain more than a few layers but not a large number of layers by any means, do not use skip

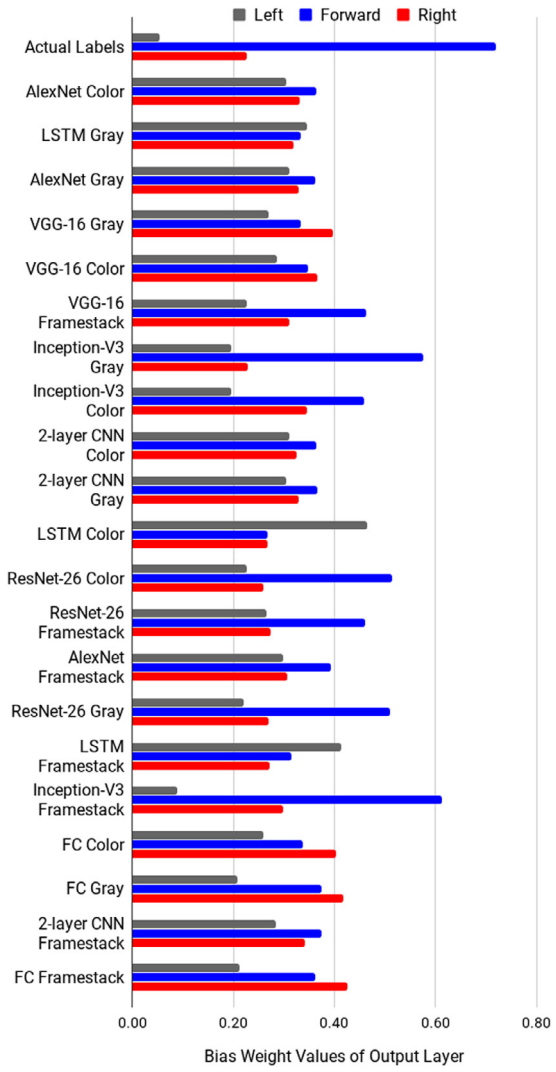


**Fig. 17.** The bias weights of each network's output layer, and the actual distribution of the training dataset's labels (top). The backward action was not included due to its very low frequency relative to the other actions. The bias weights were normalized here such that for each network they added up to 1, and the networks are shown in descending order of performance starting from the top. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

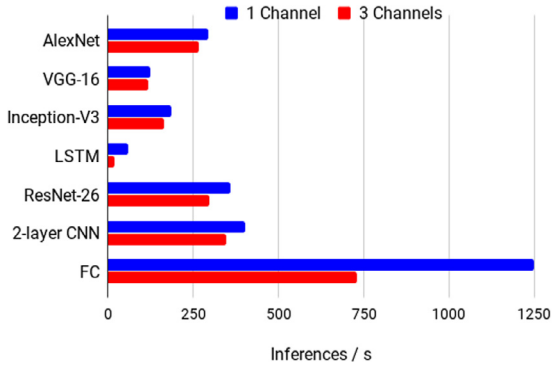


**Fig. 18.** The number of inferences each network was able to perform on color images. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

connections or global average pooling, and have at least two (often a few more) fully-connected layers. Even though AlexNet and VGG-16 were very similar in many ways, VGG-16 contained almost 30 million more parameters, yet it achieved worse success rates in the tests, despite performing significantly better in image classification tasks [54]. We identify three key differences between these two networks which help explain this performance difference. First, the VGG-16 network we use here contains relu non-linearities in the fully-connected layers, whereas the AlexNet network uses tanh. The relu non-linearity has been shown to accelerate the training process because it does not produce saturating neurons [35], but it has also been shown to lead to 'dead' neurons. This could be problematic for VGG-16 because dead neurons in the fully-connected layers would prohibit the gradient from back-propagating throughout the rest of the network. We used a relatively small learning rate, which would have helped to lessen the chance of dead neurons, but this phenomenon may have played some role in VGG-16's performance. Another important difference between the two architectures is the size of the bottleneck between the last convolutional layer and the first fully-connected layer, as discussed previously. The bottleneck in AlexNet was 2.5, and that in VGG-16 was 5.14. While these are both well within the range of those observed in the previous studies, it is quite possible that this conferred an advantage on AlexNet in this task. One final difference is that AlexNet employs local response normalization in addition to weight decay (as described in Section 3.3) while VGG-16 only uses weight decay. Local response normalization helps to decorrelate and regularize the features of a given layer, which likely explains why AlexNet was able to perform better with fewer filters and layers. Fig. 15 illustrates that AlexNet and VGG-16 had very similar weight distributions, except that the tail of VGG-16's distribution extended much farther out, which could indicate some degree of overfitting.

### 5.4. Inception-v3 and resnet-26

The Inception-V3 architecture obtained above average success rates on both test phases, even with a relatively low number of parameters. One behavior to note is this network's high lateral movement relative to the other networks (Fig. 12). Given that this network exhibited similar behavior to the best-performing networks (i.e. AlexNet and VGG-16) regarding validation loss/confusion matrix, mean softmax probability, and heatmap appearance, why, then, does this network exhibit such extreme path fluctuations and worse performance on the Lighting and Object Tests? We believe this is because Inception-V3 employs a global average pooling layer followed by the linear classifier. This is desirable in image classification networks, where the goal is to detect if there is an object anywhere in the image, but in tasks such as robotic applications which involve navigating through environments that are often complex, it is detrimental because it essentially causes the loss of all spatial information and, thus, introduces added difficulty onto the classifier in trying to predict the correct steering action. In fact, the authors are not aware of any recent works which use a sub-scale, remote-controlled vehicle in an end-to-end autonomous driving/steering task (e.g. [8,10–12], etc.) that also use a global average pooling layer. Instead, they use multiple fully-connected layers between the final convolutional layer and the linear classifier. Previously, the limitations of global average pooling have been discussed by Szegedy et al. [16] and Yang et al. [55] – albeit briefly and without empirical evidence, who point out global average pooling's difficulties when applied to transfer learning scenarios. Conversely, in robotic applications where compounding covariate shift is an issue and the task involves navigation, the reason to use fully-connected layers in place of global average pooling has not been

stated explicitly until this report. Finally, we would note that the Inception-V3 network trained on single color images obtained one of the higher lap completion rates in both the Lighting Tests and the Lighting and Object Tests, but unlike the other top performers AlexNet and VGG-16, this network experienced only about half of the altered actions as these other networks in the pixel-flipping experiment. It is possible that this is due to the global average pooling layer, but future studies will be needed to fully understand this phenomenon, perhaps in the context of adversarial examples.

Despite exhibiting above average validation performance and decreased lateral movement, the ResNet-26 network was one of the worst-performing networks with respect to most other metrics, which is initially surprising given that this architecture is often used in many state-of-the-art image classification, object segmentation/detection, and image generation networks. Similar to the Inception-V3 network, this poor performance is likely due to the fact that ResNet-26 uses global average pooling, which would cause a loss of spatial information. As a result, both networks had large bias weights corresponding to the forward node, which may have been an attempt to deal with the loss of spatial information by taking advantage of the action imbalance in the training set. ResNet-26 often used the pink color of the track's walls on the turns as a cue to turn (as discussed in Section 4.5), which is one possible indication that it may have overfit to the training dataset and would not be able to generalize to new tracks very well (for example those without a pink-colored border on the turns). One major difference between the Inception-V3 and ResNet-26 architectures is the number of convolutional filters in each network, particularly in the last convolutional layer before global average pooling. In the Inception-V3 architecture, this number was 1024, and in the ResNet-26 architecture, it was just 128. This would not only have caused Inception-V3 to have a far greater representational capacity, but its feature vectors would be more separable with respect to different classes due to the shear difference in the number of features, as we observed in Fig. 7. This may be why Inception-V3 was able to obtain a higher success rate than ResNet-26 despite its much more extreme path fluctuations. Possible future studies might include observing how the number of feature maps in the layer preceding global average pooling affects driving performance, if the addition of a fully-connected layer after global average pooling and before the linear classifier – as Szegedy et al. [16] recommend for transfer learning – is able to increase performance, or how the ResNet architecture performs with global average pooling vs with a fully-connected layer in its place.

### 5.5. LSTM

The LSTM network performed around average regarding most of the metrics – including success rates – despite exhibiting a below average confusion matrix. One interesting result was this network's superior performance on the grayscale inputs compared to the color frames, something not seen in other networks. This may be explained quite simply with the aid of Fig. 18. On images the same size and type as those coming through the vehicle's camera, the LSTM used here was only able to perform 18 inferences $s^{-1}$ on images with three channels, such as the single color frame and the gray framestack, but for the single gray image, it was able to perform 60 inferences $s^{-1}$. Since the vehicle operated at 30 FPS during the course of this research, it was impossible for the LSTM using color and framestack inputs to produce inferences fast enough, which probably caused the network to miss frames.

Given the fact that this network and the fully-connected network both use only fully-connected layers, it is at first surprising why their performance is so different in many ways. One reason this might be the case is that in this network the rows of the image are treated as timesteps, which means that each node is responding to each row at a time until it has reached the bottom of the image. In this way, it is similar to a convolutional network with a $1 \times 320$ filter (at least in the grayscale image), except that the hidden activation is determined via various gating mechanisms as opposed to a weighted sum followed by an activation function and/or normalization. We believe that this allowed the LSTM to perform better than the fully-connected network, as it was able to incorporate information in the input locally and had a smaller bottleneck.

## 6. Conclusion

This work consists of the first experimentally-controlled comparison of multiple deep learning architectures, trained under BC, while they control the steering of a vehicle. We found that most of the more recent and advanced architectures that were tested performed well, contrary to the more primitive networks, which illustrates that they are able to generalize enough to overcome compounding covariate shift introduced by BC, even under different lighting changes and when tasked to avoid obstacles, despite not being trained to do so. Furthermore, the higher performing networks showed more consistency in their path between trials and also converged onto similar paths as the other high-performing networks, and they also exhibited much smoother paths within trials. Specifically, the AlexNet architecture is an attractive candidate for such tasks, as it was the best performing network on both test phases, had a very consistent and smooth path relative to the others, showed low bias toward the training label distribution, and was able to be pruned significantly without much loss of accuracy or change in output values. Overall, we identify the bottleneck between convolutional and fully-connected layers, use of global average pooling, and use of normalization as critical architectural features in determining behavior in tasks such as this. It is evident from these findings that architectural features that perform well on strict classification tasks may not necessarily be best suited for IL tasks (as others have previously pointed out), and, as these networks are increasingly deployed in IL systems capable of acting on their environment, it is crucial that we keep this in mind when designing them. We hope this work helps to motivate further studies aimed at examining how subtle architectural differences may influence driving behavior when they are embedded in physical vehicles in more complex environments. Finally, we hope that these experimental results help to complement previous results obtained in real-world settings, and we also note that further confirmation of these insights is necessary in more complex settings, which we leave to future work.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Acknowledgments

The authors gratefully acknowledge the support of NVIDIA Corporation with the donation of the GPU hardware used for this research. The authors would also like to thank Levi Stein for his hard work and creativity throughout this research.

## References


[1] Andrew Y. Ng, Stuart J. Russell, et al., Algorithms for inverse reinforcement learning, in: Icml, 2000, pp. 663–670.

[2] Stuart Russell, Learning agents for uncertain environments, in: Proceedings of the Eleventh Annual Conference on Computational Learning Theory, ACM, 1998, pp. 101–103.

[3] Dean A. Pomerleau, Alvinn: An autonomous land vehicle in a neural network, in: Advances in Neural Information Processing Systems, 1989, pp. 305–313.

[4] Jonathan Ho, Stefano Ermon, Generative adversarial imitation learning, 2016, CoRR, abs/1606.03476.

[5] Stéphane Ross, Drew Bagnell, Efficient reductions for imitation learning, in: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 2010, pp. 661–668.

[6] Stéphane Ross, Geoffrey Gordon, Drew Bagnell, A reduction of imitation learning and structured prediction to no-regret online learning, in: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, 2011, pp. 627–635.

[7] J. Andrew Bagnell, An Invitation to Imitation, Technical Report, Carnegie-Mellon University Robotics Institute, 2015.

[8] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al., End to end learning for self-driving cars, 2016, arXiv preprint arXiv:1604.07316.

[9] Y. Lecun, E. Cosatto, J. Ben, U. Muller, B. Flepp, Dave: Autonomous Off-Road Vehicle Control Using End-To-End Learning, DARPA-IPTO Final Report, 2004.

[10] Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, Evangelos Theodorou, Byron Boots, Agile off-road autonomous driving using end-to-end deep imitation learning, 2, 2017, arXiv preprint arXiv:1709.07174.

[11] Felipe Codevilla, Matthias Miiller, Antonio López, Vladlen Koltun, Alexey Dosovitskiy, End-to-end driving via conditional imitation learning, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 1–9.

[12] Marin Toromanoff, Emilie Wirbel, Frédéric Wilhelm, Camilo Vejarano, Xavier Perrotton, Fabien Moutarde, End to end vehicle lateral control using a single fisheye camera, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2018, pp. 3613–3619.

[13] Sauhaarda Chowdhuri, Tushar Pankaj, Karl Zipser, MultiNet: Multi-modal multi-task learning for autonomous driving, in: 2019 IEEE Winter Conference on Applications of Computer Vision, WACV, IEEE, 2019, pp. 1496–1504.

[14] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.

[15] Karen Simonyan, Andrew Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, CoRR, abs/1409.1556.

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, Going deeper with convolutions, 2014, CoRR, abs/1409.4842.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep residual learning for image recognition, 2015, CoRR, abs/1512.03385.

[18] Sepp Hochreiter, Jürgen Schmidhuber, LSTM Can solve hard long time lag problems, in: Advances in Neural Information Processing Systems, 1997, pp. 473–479.

[19] Mariusz Bojarski, Philip Yeres, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Lawrence Jackel, Urs Muller, Explaining how a deep neural network trained with end-to-end learning steers a car, 2017, arXiv preprint arXiv:1704.07911.

[20] Mariusz Bojarski, Anna Choromanska, Krzysztof Choromanski, Bernhard Firner, Larry J Ackel, Urs Muller, Phil Yeres, Karol Zieba, Visualbackprop: Efficient visualization of cnns for autonomous driving, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 1–8.

[21] Kittrich Corporation, RCP-Tracks, 2017.

[22] Brookstone LLC, Rover 2.0 app-controlled wireless spy tank, 2017.

[23] Dmitry Ulyanov, Andrea Vedaldi, Victor S. Lempitsky, Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis, 2017, CoRR, abs/1701.02096.

[24] Yingwei Pan, Yehao Li, Ting Yao, Tao Mei, Houqiang Li, Yong Rui, Learning deep intrinsic video representation by exploring temporal coherence and graph structure, in: IJCAI, 2016, pp. 3832–3838.

[25] David Sussillo, Random walks: Training very deep nonlinear feed-forward networks with smart initialization, 2014, CoRR, abs/1412.6558.

[26] Felix A. Gers, Jürgen Schmidhuber, Fred Cummins, Learning to forget: Continual prediction with LSTM, Neural Comput. 12 (10) (2000) 2451–2471.

[27] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015, CoRR, abs/1502.01852.

[28] Anders Krogh, John A. Hertz, A simple weight decay can improve generalization, in: Advances in Neural Information Processing Systems, 1992, pp. 950–957.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.

[30] Rob A. Dunne, Norm A. Campbell, On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function, in: Proc. 8th Aust. Conf. on the Neural Networks, Vol. 181, Melbourne, 1997, p. 185.

[31] Barry L. Kalman, Stan C. Kwasny, Why tanh: choosing a sigmoidal function, in: Neural Networks, 1992. IJCNN., International Joint Conference on, Vol. 4, IEEE, 1992, pp. 578–581.

[32] Twan van Laarhoven, L2 regularization versus batch and weight normalization, 2017, CoRR, abs/1706.05350.

[33] Maximilian Riesenhuber, Tomaso Poggio, Hierarchical models of object recognition in cortex, Nature Neurosci. 2 (11) (1999) 1019.

[34] Heiko Wersing, Edgar Körner, Learning optimized features for hierarchical models of invariant object recognition, Neural Comput. 15 (7) (2003) 1559–1588.

[35] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252.

[37] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, Kaiming He, Aggregated residual transformations for deep neural networks, 2016, CoRR, abs/1611.05431.

[38] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, Inception-v4, inception-ResNet and the impact of residual connections on learning, 2016, CoRR, abs/1602.07261.

[39] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, Rethinking the inception architecture for computer vision, 2015, CoRR, abs/1512.00567.

[40] Sergey Ioffe, Christian Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015, CoRR, abs/1502.03167.

[41] George E. Dahl, Tara N. Sainath, Geoffrey E. Hinton, Improving deep neural networks for LVCSR using rectified linear units and dropout, in: Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on, IEEE, 2013, pp. 8609–8613.

[42] Bekir Karlik, A. Vehbi Olgac, Performance analysis of various activation functions in generalized MLP architectures of neural networks, Int. J. Artif. Intell. Expert Syst. 1 (4) (2011) 111–122.

[43] Jiawei Su, Danilo Vasconcellos Vargas, Kouichi Sakurai, One pixel attack for fooling deep neural networks, 2017, CoRR, abs/1710.08864.

[44] Matthew D Zeiler, Dilip Krishnan, Graham W Taylor, Rob Fergus, Deconvolutional networks, in: 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE, 2010, pp. 2528–2535.

[45] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, Wojciech Samek, On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation, PLoS One 10 (7) (2015) e0130140.

[46] Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Methods for interpreting and understanding deep neural networks, 2017, CoRR, abs/1706.07979.

[47] Jiasi Chen, Xukan Ran, Deep learning with edge computing: A review, Proc. IEEE 107 (8) (2019) 1655–1674.

[48] Jonathan Frankle, Michael Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, 2018, arXiv preprint arXiv:1803.03635.

[49] Amir H. Ashouri, Tarek S. Abdelrahman, Alwyn Dos Remedios, Retraining-free methods for fast on-the-fly pruning of convolutional neural networks, Neurocomputing 370 (2019) 56–69.

[50] Laurens van der Maaten, Geoffrey Hinton, Visualizing data using t-SNE, J. Mach. Learn. Res. 9 (Nov) (2008) 2579–2605.

[51] Zhou Wang, Alan C Bovik, Hamid R Sheikh, Eero P Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process. 13 (4) (2004) 600–612.

[52] Hadley Wickham, Maintainer Hadley Wickham, The ggplot package, 2007.

[53] Parth Gupta, Rafael E. Banchs, Paolo Rosso, Squeezing bottlenecks: exploring the limits of autoencoder semantic representation capabilities, Neurocomputing 175 (2016) 1001–1008.

[54] Alfredo Canziani, Adam Paszke, Eugenio Culurciello, An analysis of deep neural network models for practical applications, 2016, arXiv preprint arXiv:1605.07678.

[55] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, Ziyu Wang, Deep fried convnets, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1476–1483.

**Michael Teti** holds a B.S. in Biology from Florida Atlantic University in Boca Raton, Florida, where he is currently pursuing a Ph.D. in the Center for Complex Systems and Brain Sciences. His work has focused on Deep Learning and sparse coding applications to autonomous navigation and object recognition. He is interested in human and animal development and is currently researching how these might apply to robotics engineering.