Optimizing DNN architectures for high speed autonomous navigation in GPS denied environments on edge devices*

Prafull Prakash[1], Chaitanya Murti[1], Saketha Nath[2], and Chiranjib Bhattacharyya[1,3]

[1] Robert Bosch Centre for Cyber-Physical Systems, Indian Institute of Science, {prafull, mchaitanya,chiru}@iisc.ac.in,
[2] Department of Computer Science and Engineering, Indian Institute of Technology, Hyderabad, saketha@iith.ac.in
[3] Department of Computer Science and Automation, Indian Institute of Science

Abstract. We address the challenge of high speed autonomous navigation of micro aerial vehicles (MAVs) using DNNs in GPS-denied environments with limited computational resources; specifically, we use the ODROID XU4 and the Raspberry Pi 3. The high computation costs of using DNNs for inference, particularly in the absence of powerful GPUs, necessitates negotiating a tradeoff between accuracy and inference. We address this tradeoff by employing sparsified neural networks. To obtain such architectures, we propose a novel algorithm to find sparse "sub networks" of existing pre trained models. Contrary to existing pruning-only strategies, our proposal includes a novel exploration step that efficiently searches for a different, but identically sparse, architecture with better generalization abilities. We derive learning theoretic bounds that reinforce our empirical findings that the optimized network achieves comparable generalization to the original network. We show that using our algorithm it is possible to discover models which, on average, have upto 19x fewer parameters than those obtained using existing state of the art pruning methods on autonomous navigation datasets, and achieve upto 6x improvements on inference time compared to existing state of the art shallow models on the ODROID XU4 and Raspberry Pi 3. Last, we demonstrate that our sparsified models can complete autonomous navigation missions with speeds upto 4m/s using the ODROID XU4, which existing state of the art methods fail to do.

1 Introduction

Micro Aerial Vehicles (MAVs), or drones, exemplify the notion of mobile edge computing, as they are required to interact with data from a variety of sources while navigating independently. They are increasingly being used in a variety of applications, including delivery, videography, surveillance, and search-and-rescue

* We gratefully acknowledge the Robert Bosch Centre for Cyber-Physical Systems, Indian Institute of Science, Bangalore, for their support via grant RBCO-0018.

(Giusti et al., 2016). A key requirement for many MAV applications is the ability to react autonomously in a variety of challenging environments, including those where GPS availability is limited. Additionally, drones are required to complete missions as quickly as possible, given their limited power envelope, and the steep power consumption profile of the motors. In particular, this necessitates high speed autonomous navigation to optimize their effectiveness (Boroujerdian et al., 2018). Moreover, because of the aversarial impact of latency on the MAV's decision making, it is better to use onboard compute devices instead of offboard systems, even if the onboard options have limited computing capabilities (Genc et al., 2017); some typical examples of such devices are listed in (Delmerico and Scaramuzza, 2018). Also, in such resource scarce set-ups, monocular cameras are ideal sensors for empowering autonomous navigation on MAVs.

The recent success of deep neural network (DNN) based methods on computer vision tasks has fostered a growing movement towards applying DNNs to empower drones with a variety of autonomous navigation capabilities (Giusti et al., 2016; Loquercio et al., 2018). While DNN models outperform other techniques in terms of accuracy, typically the associated inference time is prohibitive for high speed navigation. As a result, the extremely challenging task of enabling high speed navigation by deploying DNNs on resource constrained platforms becomes crucial to their application on MAVs (Mohta et al., 2018). Significant efforts have gone into developing efficient, customized hardware to enable high speed inference; see (Quigley et al., 2018) and the references therein. Another approach is algorithmic, wherein DNN models are optimized to enable faster inference and lower storage requirements.

This paper proposes using a highly optimized sparse DNN that achieves the right accuracy vs. inference time trade-off as prescribed for high-speed autonomous navigation in such resource-constrained environments. Specifically, our contributions are as follows.

- We empirically establish that existing deep as well as shallow neural network architectures have prohibitively high inference times for deployment on resource constrained platforms like the ODROID XU4 and the Raspberry Pi 3, to enable high speed, vision based autonomous navigation. We observe that unless the inference time is reduced to ≈23ms per image, it is difficult to achieve high-speed navigation.
- The key technical contribution is a novel algorithm, `FeatherDrop`, that efficiently optimizes a given (state-of-the-art) network's structure to achieve the right accuracy vs. inference time trade-off. Unlike existing prune-only algorithms, this algorithm includes a new exploration step that searches for an alternate sub-network with same sparsity but better generalization.
- We analyze the algorithm for convergence and show that it achieves a particular kind of sub-optimality. We also present learning bounds that explicitly illustrate the trade-off with the inference time.
- We empirically establish that the new exploration step can be used along with any existing pruning strategy to improve its efficacy.
- We demonstrate that FeatherNet enables a Parrot Bebop 2 drone to autonomously navigate on semi-structured roads at speeds upto 4 m/s, using

an Odroid XU4. Furthermore, architectures proposed in (Giusti et al., 2016) and (Loquercio et al., 2018) fail to do so. In addition, we also establish the efficacy of our sparsified model on the Raspberry Pi 3.

All proofs, and further experimental detail, is available in the supplementary material[4].

2 Related Work

2.1 Autonomous Navigation of MAVs

Vision Based Autonomous Navigation. Cameras are generally considered to be ideal sensors for autonomous navigation for drones (Giusti et al., 2016). The success of deep learning algorithms for image processing tasks has naturally led to their application toward enabling autonomous navigation capabilities in drones where explicit structure is unavailable, as classical image processing techniques fail in such conditions, see (Giusti et al., 2016; Loquercio et al., 2018) and the references therein.

Low Power MAVs. A crucial drawback to using DNNs on MAVs is the fact that they require significant computational resources. Typically, MAVs are equipped with small, low power compute platforms (Delmerico and Scaramuzza, 2018). However, the limited power available to drones, and the high power demand of motors means that to optimze power usage, it is critical that missions be completed as quickly as possible; an ideal solution to this problem is to enable the drone to navigate autonomously at high speeds (Boroujerdian et al., 2018). When navigation is empowered by DNNs, the flight speed of the MAV is bottlenecked by the inference rate of the DNN (Loianno et al., 2018). Efforts to enable efficient inference for DNNs typically involve developing custom hardware (Quigley et al., 2018). Our approach involves algorithmically improving the inference rate of the DNN using model compression.

2.2 Sparse Architecture Search with Pruning

Pruning refers to the sparsification of the model itself. As noted in (Frankle and Carbin, 2018), for every neural network, there exists an initialization which, when trained therefrom, results in a model with far fewer parameters. The aim of pruning techniques is to find such sparsified models, either by compressing a pre-trained model or by inducing sparsity during training. Moreover, it was proposed in (Liu et al., 2018) that all pruning techniques are essentially algorithms to search for sparsified architectures. Pruning techniques can be further classified into structured pruning and unstructured pruning. Weight pruning refers to processes by which individual weights are removed the model, such as (LeCun

[4] at: https://tinyurl.com/y2dzss44

et al., 1990; Iandola et al., 2016). As noted in (Iandola et al., 2016), unstructured pruning algorithms can achieve remarkably high levels of compression in terms of the number of parameters removed from the model. We don't employ such methods since the drawback to using them is that there is no guarantee that entire filters or neurons are removed; thus, to fully exploit the benefits of highly sparse (but potentially numerous) filters, efficient implementations of sparse matrix multiplications and convolutions are necessary.

Searching for Sparse Architectures. Searching for sparse architectures of neural networks, sometimes referred to as structured pruning (Molchanov et al., 2016), refers to processes by which entire filters or neurons (which we collectively refer to as *artifacts*) are removed from the model to obtain sparse "sub networks" of the original model. The methods proposed in (Li et al., 2016) and similar works search for architectures that meet a preselected level of sparsity; the problems solved by those methods can be formulated as

$$\min \ \{f(\mathbf{w})| \text{ number of artifacts } = K\}$$

for some natural number K. We do not adopt such methods since we know of no rigorous mechanism by which the sparsity level K can be chosen while also satisfying an accuracy constraint. Another approach to solve the structured pruning problem is to automatically search for compressed architectures. This can be formulated as the following optimization problem:

$$\min \{\text{number of artifacs} \mid f(\mathbf{w}) \leq t\}$$

for some test error threshold t, which can be chosen based upon empirically determined constraints. One way to solve such problems is to utilize regularization during training to induce sparsity, such as (Zhang et al., 2018) and similar work; we discount such methods since our goal is to compress existing pre-trained models. Another class of methods iteratively remove filters based on "saliency scores", and then "fine-tune", or retrain, the model, such as (Li et al., 2016; Molchanov et al., 2016). In (Li et al., 2016), filters are pruned based on saliency scores derived from the ℓ_1 norm of the vectorized weights of each filter; note that in the paper, a predefined target architecture is provided, but this can easily be adapted to the setting where an accuracy constraint must be met. In (Molchanov et al., 2016), saliency scores are used to determine which filters can be removed with the least impact on the loss function. Our approach using *basis exploration*, described in Section 3, is by design an algorithm to improve upon architecure search methods; employing our algorithm alongside an iterative architecture search strategy results in models that are at least as sparse as those obtained without using basis exploration.

3 Searching for Sparse DNN Architectures

Motivated by the constraints on accuracy and inference time, we propose a novel technique for optimizing neural network architectures. We observe that

the search for sparse neural network architectures can be broadly thought of as searching for the minimum support of the model. We then propose *Greedy Minimim Support (GMS)*, an iterative algorithm that finds the minimum support of a function, which utilizes a novel exploration step. We adapt the GMS algorithm to finding sparse neural architectures. The specific algorithm thus obtained is called `FeatherDrop`, an iterative pruning algorithm for efficient architecture search. `FeatherDrop` enables significant sparsification, while systematically satisfying an accuracy constraint.

3.1 Preliminaries and Notation

Lower case letters such as, x, denote scalars, bold lower case letters such as, $\mathbf{x}$, denote vectors, uppercase letters like X denote sets, and bold uppercase letters, such as $\mathbf{X}$, denote matrices. $[n] = \{1, ..., n\}$ for $n \in \mathbb{N}$. For a finite set A, let $|A|$ denote the cardinality of A. $\mathbf{x}_i$ denotes the i^{th} entry of $\mathbf{x}$. Let $\nabla_i f(\mathbf{x}) = \frac{\partial f}{\partial \mathbf{x}_i}$ for any $\mathbf{x} \in \mathbb{R}^n$. For $\mathbf{x} \in \mathbb{R}^n$, let $\texttt{sort}_i(\mathbf{x})$ denote the ith largest absolute element of $\mathbf{x}$. Define $\texttt{sparse}_k(\mathbf{x}) \in \mathbb{R}^d$ as a vector with its jth element defined as: $(\texttt{sparse}_k(\mathbf{x}))_j = \begin{cases} \mathbf{x}_j & |\mathbf{x}_j| \geq \texttt{sort}_k(\mathbf{x}) \\ 0 & \text{otherwise} \end{cases}$

We define the loss function of a neural network as follows.

Definition 1. *Let $\mathcal{Y} = \{y_1, .., y_l\}$ be a set of labels for data from set $\mathcal{D}$. Let $g : \mathcal{D} \times \mathcal{W} \to [0, 1]^{|\mathcal{Y}|}$ define the output of the neural network G, with vectorized parameters belonging to $\mathcal{W}$. We define the categorical cross entropy loss of G over samples $\{D_i \in \mathcal{D}\}_{i=1}^N$, given $\mathbf{w} \in \mathcal{W}$ as*

$$f(\mathbf{w}) = -\frac{1}{N|\mathcal{Y}|} \sum_{i=1}^{N} \sum_{j=1}^{|\mathcal{Y}|} \bar{y}_{ij} \log(g_j(\mathbf{w}, D_i)) \tag{1}$$

where each $\bar{y}_{ij}$ is the ground-truth score for class j of sample D_i.

Parameters of filters in convolutional layers and incoming weight vectors in feed forward layers are collectively referred to as *artifacts*. For notational convenience, we assume that the network has N artifacts, each of which has m parameters, thus giving us a network with $n = mN$ parameters and $\mathbf{w} \in \mathbb{R}^n$. Let $\mathbf{w}_{[i]}$ denote the i^{th} block of m parameters corresponding to the i^{th} artifact. Let $\texttt{supp}(\mathbf{w})$ denote the set of indexes for blocks/artifacts with non-zero parameter values in $\mathbf{x}$. Thus $\texttt{supp}(\mathbf{w}) \subseteq [N]$.

3.2 Formulation

We pose the objective of architecture optimization as follows. Given a neural network loss function f defined in Definition 1, and $t \in \mathbb{R}$ such that $t > \min f(\mathbf{w})$ for $\mathbf{w} \in \mathbb{R}^n$, we wish to solve

$$\min_{A \subset [N]} |A| \text{ subject to } \min_{\{\mathbf{w} | \texttt{supp}(\mathbf{w}) = A\}} f(\mathbf{w}) \leq t \tag{OPT}$$

The choice of t is crucial to ensure that the feasibility set of (OPT) is non-empty:

Lemma 1. *Let $\mathbf{w}^{(0)}$ be an unconstrained local minimum of any $f : \mathbb{R}^n \to \mathbb{R}$ and let the gradient of f be L-Lipschitz, and suppose $f_0 = f(\mathbf{w}^{(0)}) > 0$. Then there exists $\mathbf{w} \in \mathbb{R}^n$ such that*

$$f(\mathbf{w}) \leq f_0(1+\delta), \quad \|\mathbf{w}\|_0 = k \in \mathbb{N} \leq n-1,$$

whenever $\delta \geq \delta_0$,

$$\delta_0 = \frac{L}{2}\frac{1}{f_0}\sum_{i=0}^{n-k-1}\left(\mathtt{sort}_{n-i}(\mathbf{w}^{(0)})\right)^2. \tag{2}$$

For a given $t = f_0(1+\delta)$, where $\delta \geq \delta_0$, defined in (2), the optimal A^* of problem OPT satisfies the following property:

$$A \subset [d], |A| < |A^*| \Rightarrow \min_{\{\mathbf{w} | \mathtt{supp}(\mathbf{w}) = A\}} f(\mathbf{w}) > t. \tag{3}$$

Now, define $\mathbf{w}^* \equiv \arg\min_{\{\mathbf{w} | \mathtt{supp}(\mathbf{w}) = A^*\}} f(\mathbf{w})$. To achieve better generalization, from all the optimal solutions, we would like to select that special solution A^* such that:

$$f(\mathbf{w}^*) = \min_{\{\mathbf{w} \mid |\mathtt{supp}(\mathbf{w})| = |A^*|\}} f(\mathbf{w}). \tag{4}$$

In general, verifying whether a candidate $\hat{A}$ is indeed optimal would require solving $O(2^{|\hat{A}|})$ minimization problems. Instead, we opt for a special (sub-optimal) greedy solution: a feasible candidate, $(\mathbf{w}^*, A^* = \mathtt{supp}(\mathbf{w}^*))$ is said to be greedy-optimal iff it satisfies the following:

G1 As a consequence of (3), for all $i \in A^*$, $\min_{\{\mathbf{w} \mid \mathtt{supp}(\mathbf{w}) = A^* - \{i\}\}} f(\mathbf{w}) > f_0(1+\delta)$.

G2 Similarly, a consequence of (4), for every $l \in A^*$ and every $m \in \bar{A^*}$, $f(\mathbf{w}^*) \leq f_{lm} = \min_{\{\mathbf{w} \mid \mathtt{supp}(\mathbf{w}) = A^* - \{l\} \cup \{m\}\}} f(\mathbf{w})$. We call this **Basis Exploration**.

The above conditions suggest that for a greedy optimal point, it is neither possible to improve the sparsity by removing a single artifact nor possible to improve the loss with a strategy of exchanging a single artifact.

3.3 The GMS Algorithm

We use these conditons to derive an iterative algorithm, stated in Algorithm 1, which is gauranteed to converge to a greedy-optimal point. This is because Algorithm 1 does not enter a cycle due to the strict inequality condition for an exchange.

Infact, the proposed algorithm can be efficiently implemented by noting that the identities of the artifacts within a layer do not matter, assuming each layer consists of the same type of artifacts. Hence verifying condition **G1** can be

Algorithm 1: Greedy Minimum Support (GMS)

Input: t
Initialize: Find $\mathbf{w}^*$ a local minimum of

$$\min_{\mathbf{w}} f(\mathbf{w})$$

such that $f(\mathbf{w}^*) \leq t$.
$A = \texttt{supp}(\mathbf{w}^*)$, $\mathbf{w}_{sparse} = \mathbf{w}^*$.
while True **do**
  if there exists $l \in A$ such that $\min_{\texttt{supp}(\mathbf{w})=A-\{l\}} f(\mathbf{w}) \leq t$ **then**
    $\mathbf{w}_{sparse} = \arg\min_{\texttt{supp}(\mathbf{w})=A-\{l\}} f(\mathbf{w})$ and $A \to A - \{l\}$
  else if Find $l \in A$ and $m \in \bar{A}$ such that $f_{lm} \leq t$

$$f_{lm} = \min_{\texttt{supp}(\mathbf{w})=A-\{l\}\cup\{m\}} f(\mathbf{w})$$

  then
    $A \to A - \{l\} \cup \{m\}$ and $\mathbf{w}_{sparse} = \arg\min_{\texttt{supp}(\mathbf{w})=A} f(\mathbf{w})$
  else
    End
  end if
end while
Output: $\mathbf{w}_{sparse}, A$

performed by solving $O(h)$ minimization problems, where h is the number of hidden layers in the network. And, condition **G2** can be verified by solving $O\left(h\left(h-1\right)\right)$ minimization problems.

The pursuit of a greedy-optimal solution is justified because, in the general case, the original problem is NP-hard. The algorithm, as described above, is an instance of an algorithm which searches for a 1-opt solution, as defined in (Park and Boyd, 2018). This greedy search algorithm can be used to search for network architectures, as we show in the next section.

3.4 Applying GMS to Network Architecture Search

We begin with a heuristic for avoiding the $O(h)$ retraining cycles during the pruning phase of the proposed algorithm 1.

For ease of exposition, we define a score function for each artifact to evaluate its importance. These scores are then normalized and the artifact with the lowest score is pruned, instead of $O(h)$ retrainings. More specifically, for any parameter vector $\mathbf{w}$, and the jth artifact in the lth layer, we define

$$\texttt{select}(\mathbf{w}) \equiv \text{argmin}_{j\in\texttt{supp}(\mathbf{w})} \frac{g_j(\mathbf{w})}{\max_{k\in\texttt{Layer}(\mathbf{w},l)} g_k(\mathbf{w})} \tag{5}$$

where $\texttt{Layer}(\mathbf{w}, l)$, is the set of all artifacts in the lth layer and in $\texttt{suppw}$, and $g_j(\mathbf{w})$ is a positive artifact specific score function. We use this normalization as at

least one artifact in each layer has score 1, thereby ensuring that at least 1 artifact remains in each layer. In this paper we experiment with two different score functions. We call $\texttt{select}_{\ell_2}(\mathbf{w})$, the selection criterion obtained by applying the following score function

$$g_j(\mathbf{w}) = \|\mathbf{w}_{(j)}\| \tag{ℓ_2}$$

to equation (5). This criterion can be useful at saddle points where the gradient vanishes. However, whenever the gradient exists then one can build a more refined model of the objective function value, $f(\mathbf{w})$, when we remove the jth artifact through the first order Taylor expansion. Indeed, from the first order Taylor expansion, we obtain the following approximation $f(\mathbf{w} - \mathbf{w}_{(j)}) - f(\mathbf{w}) \approx \langle \mathbf{w}_{(j)}, \nabla_{(j)} f(\mathbf{w}) \rangle$. This motivates the alternate selection criterion, $\texttt{select}_{FS}(\mathbf{w})$, which is defined by using the score function

$$g_j(\mathbf{w}) = \left| \langle \mathbf{w}_{(j)}, \nabla_{(j)} f(\mathbf{w}) \rangle \right| \tag{FS}$$

in (5). We refer to the variant of GMS, that uses (ℓ_2) and (FS) used in lines 5 and 7, as `FeatherDrop`[5].

Learning Bounds

In the reminder of this section we present a learning bound that explicitly incorporates the inference time budget. If N_i is the number of (non-pruned) artifacts in the i^{th} hidden layer, n_i, n_o are the input and output dimensions of the network, then the inference time is proportional to $n_i N_1 + N_1 N_2 + \ldots + N_{D-1} N_D + N_D n_o$, which we assume is less than some budget, B. Then, the following theorem holds.

Theorem 1. *With probability atleast $1 - \delta$, we have:*

$$\mathcal{R}[\mathbf{w}] \leq \hat{\mathcal{R}}[\mathbf{w}] + 2\sqrt{\frac{4^d \log n_i R \Pi_{l=1}^{d} B_l}{m}} + 3\sqrt{\frac{1}{2m}\left(\log\frac{2}{\delta} + d\log\frac{B}{2}\right)}, \tag{6}$$

where $\|W_l\|^2_{1_{N_l},\infty} \leq B_l, \mathcal{R}, \hat{\mathcal{R}}$ denote the true, emprical risk with employing the pruned classifier respectively. Here, $\|x\|_{1_N}$ denotes the sum of the absolute values of the largest (in magnitude) N entries of x. If $N = dim(x)$, then this is same as 1-norm. For a matrix W, $\|W\|_{1_N,\infty}$ denotes the maximum of the 1_N-norms of the columns of W.

This learning bound provides the following insights into the generalization ability of the pruned classifier:

1. As inference rate increases (B decreases), the confidence terms decrease. Hence the gaurantees become better and better as long as the empirical risk is upper bounded (like in our algorithm by t).
2. The above phenomenon has more pronounced effect for deep networks. In other words, the befenits of pruning are more prominent for deep networks.

[5] In the analogy of flight of birds, we drop unnecessary feathers, hence `FeatherDrop`

4 Experimental Results

In this section, we motivate and detail our experiments. We investigate two different areas, namely the efficacy of the pruning algorithm, and the utility of compressed models for autonomous navigation.

4.1 Experimental Methodology

Our aim is to address the question- can drones be made to autonomously navigate at high speeds of upto 4m/sec in GPS denied environments, using CPU on lightweight compute platforms? Here, we detail how we established inference time constraints on the Odroid XU4 and Raspberry Pi 3, our reactive navigation scheme, and our dataset collection scheme.

Establishing inference time contraints In particular, we have used Odroid XU4 for running algorithms, and Parrot Bebop 2 drone as our testing platform. Parrot Bebop 2 driver running in ROS on the Odroid XU4 board receives data from the drone at 30Hz i.e. approximately every 33ms (Fig. 4(a)). Using the CvBridge package in ROS, the raw data conversion to OpenCv image takes on average 10ms (Fig. 4(b)) on Odroid XU4. This leaves approximately 23ms to process each frame for autonomous navigation.

Thus, we cast the problem as searching for sufficiently accurate neural networks which require less than 23ms for inference, using only the Odroid XU4 CPU. Our approach to finding such networks is to search for a sparse subnetwork of existing neural network architectures that are capable of satisfying our given test-set accuracy requirements. We conduct similar experiments with the Raspberry Pi 3, as detailed in our supplemental material.

Navigation Technique We use the strategy proposed in (Giusti et al., 2016). Utilizing the semi-structured settings of urban roads, we formulate the problem of autonomous navigation as a three class classification problem, where each input image is labelled as- (1) Turn Left, (2) Go Straight, or (3) Turn Right, corresponding to a control command taken when an image is classified as such. For further details, refer to the supplemental material.

Dataset Collection We collected two datasets. We construct the first dataset, called CampusRoads, out of images of roads in the university campus, annotated with TurnRight, TurnLeft, or GoStraight. We collected a total of 60042 frames, of which 40200 are for training, 9000 are for validation of architecture search algorithms, and 10842 are for testing. We construct a similar dataset of the data provided in (Giusti et al., 2016), which we call Forest Trails (20316, 3765 and 5361 frames for training, validation and testing). For a more detailed description of our data collection process, refer to the supplemental material.

4.2 Experimental Setup

Training and Sparsification of Models We train our models, and sparsify them on desktop computers equipped with an Nvidia 1080Ti and an Intel Xeon

workstation processor. The models are trained on the CampusRoads and ForestTrails datasets we have constructed.

Computational Board We use the ODROID XU4 and Raspberry Pi 3 as our compute platforms. The ODROID XU4 and Raspberry Pi 3 are both small, low power computing devices (Delmerico and Scaramuzza, 2018), and are thus ideal for use on MAVs.

Navigation Stack We use the Parrot Bebop 2 as our MAV. These MAVs are retail products, and more importantly, have drivers that integrate with the Robot Operating System (ROS). The rate of image acquisition from the drone is limited to 30Hz. Thus, we achieve autonomous navigation at high speeds of upto 4m/sec by optimizing the entire stack to run at rate at least as high as 30Hz. For a full description of our navigation stack, and the establishment of inference rate constraints, refer to the supplementary material.

All accuracies are reported on the testing set defined in Section 3. Lastly, we conduct our flight tests at two locations in the university campus.

4.3 Comparison of `FeatherDrop` with Other Pruning Schemes

In this section, we establish the effectiveness of `FeatherDrop` as compared to the algorithms proposed in (Molchanov et al., 2016) and (Li et al., 2016). We begin with pre trained models with the architecture proposed in (Giusti et al., 2016), trained separately on the CampusRoads and ForestTrails datasets. We then apply a variety of architecture search algorithms to obtain sparsified architectures. The algorithms we chose for comparision are `FeatherDrop` , `FeatherDrop` without basis exploration, (Molchanov et al., 2016; Li et al., 2016), as well as randomly selecting artifacts for pruning to obtain sparsified architectures. In order to demonstrate the utility of basis exploration, we append the random pruning strategy and the strategy proposed in (Li et al., 2016) with basis exploration. We then compare the degree of sparsification otbained by models trained on the CampusRoads and ForestTrails datasets for different values of t chosen a priori.

Results and Discussion We compare the extent of pruning for different constraint values of t as test set accuracy using `FeatherDrop` algorithm versus naive strategy of pruning without any basis exploration. The results are tabulated in Table 1.

- Table 1 shows that `FeatherDrop` leads to much more pruned arhcitectures than the naive pruning technique without basis exploration, random pruning without basis exploration, and the algorithms proposed in (Li et al., 2016) without basis exploration and (Molchanov et al., 2016).
- We observe that using basis exploration significantly improves the sparsification when applied to models pruned using (Li et al., 2016), and random pruning. Appending both the random strategy and the strategy in (Li et al., 2016) with basis exploration results in a significant increase in sparsification. This supports our conjecture that any iterative weight-based pruning scheme can be improved, potentially significantly, by utilizing basis exploration.

Test set accuracy \dataset	`FeatherDrop`	**Naive strategy**	**Random without BE**	**Random with BE**	**Li et al 2016**	**Li et al 2016 with BE**	**Molchanov et al 2016**
92% on CR	**98.69**	96.24	89.85	98.57	85.85	93.05	93.43
93% on CR	**97.98**	87.08	36.79	89.11	72.42	91.75	93.43
94% on CR	**96.68**	79.69	0.41	90.86	5.54	72.82	66.81
95% on CR	**94.74**	3.41	1.19	90.42	1.58	70.05	66.81
81% on FT	87.29	69.98	4.59	84.54	3.56	**90.04**	9.96
82% on FT	**80.08**	63.18	1.98	72.59	9.1	70.02	2.19
83% on FT	82.52	16.77	8.65	60.15	3.96	**90.5**	1.4
84% on FT	71.72	1.62	0.83	56.09	0.40	**87.51**	1.01
85% on FT	**36.06**	3.41	3.06	3.45	0.39	0.79	0.0

Table 1: Percentage reduction in the number of parameters obtained by different iterative pruning algorithms. BE refers to Basis Exploration, CR refers to CampusRoads, FT refers to ForestTrails.

– We notice `FeatherDrop` is comparably less susceptible to change in extent of pruning with change in t constraint. In particular, as t increases we obtain the same pruned architectures although the number of basis exploration steps increase.

4.4 Using Sparse Models for Autonomous Navigation

The goal of these experiments is to demonstrate that drones are capable of autonomously navigating roads at speeds upto 4m/s with sparsified models.

1. We show that existing DNN architectures for autonomous navigation propsed in (Giusti et al., 2016) and (Loquercio et al., 2018) are unsuitable for deployment on the ODROID XU4 and the Raspberry Pi 3. Then, we show that the sparsified model is able to achieve sufficient inference rates on the ODROID XU4 and the Raspberry Pi 3 (see supplemental material).
2. We compare the performance of the networks proposed in (Giusti et al., 2016) and (Loquercio et al., 2018) with the sparsified architectures obtained using `FeatherDrop`. Specifically, we deploy each model on the ODROID XU4, and attempt to complete two challenging road following tasks wherein the roads have unclear boundaries, while flying at speeds upto 4m/s.

Results and Discussion

1. We compared the inference rates of the models proposed in (Giusti et al., 2016) and (Loquercio et al., 2018) with the sparsified model we obtained using `FeatherDrop` on both the ODROID XU4 and the Raspberry Pi 3. We observed that on average the former two models achieve 61ms and 50ms inference times, and thus fail to meet the inference rate constraints established previously. Moreover, we observe that the sparsifed model obtained with `FeatherDrop` achieves an average inference time of 10ms, a 5x increase in inference rate over (Loquercio et al., 2018) and a 6x increase in inference rate over the model proposed in (Giusti et al., 2016) on the ODROID XU4;

similarly, the sparsified model achieves a 12ms inference time on the Raspberry Pi 3 (5x and 4x improvements). For a more detailed description of these results, refer to the supplementary material.

2. Last, we compared the performance of each network on challenging autonomous road following missions at speeds upto 4m/s with an ODROID XU4 using a Parrot Bebop2 drone as the platform. The missions we chose were to follow two challenging roads inside the university campus- one with two turns and the other as an unseen straight road. We observe that the models proposed in (Giusti et al., 2016) and (Loquercio et al., 2018) are incapable of regularly completing missions, whereas the sparsified model obtained using `FeatherDrop` finished all but 1 trial. This corresponds to failures occurring only when the entire system is unable to keep up with the inference rate requirement and significant environmental noise. These results are tabulated in Table 2.

Network	Mission Status on Unseen Straight Road	Mission Status on Road with 2 Turns
(Giusti et al., 2016)	1	0
(Loquercio et al., 2018)	0	0
Sparse Model	**3**	**2**

Table 2: Comparison of mission success rates (3 attempts) of our sparsified architecture with architectures proposed in (Giusti et al., 2016) and (Loquercio et al., 2018) using an ODROID XU4 for computation.

5 Conclusions

In this work, we have addressed the problem of autonomous navigation of MAVs at high speeds using DNNs in GPS-denied environments with limited computational resources. This problem naturally requires trading off accuracy and inference rate; our proposed solution negotiates this tradeoff by deploying neural networks with sparse architectures with good generalization ability. We obtain such models by searching for "sub networks" of existing pre trained models. We propose a novel algorithm to find such sparse sub networks which utilizes an exploration step. We show that this algorithm achieves superior sparsity to state of the art iterative pruning methods. Furthermore, we show that our algorithm can be used to improve the efficacy of any iterative pruning method. Lastly, we demonstrate that small, sparsified models obtained by applying our network search algorithm to a network based on (Giusti et al., 2016), are able to complete real world, autonomous flight missions at high speed that models proposed in (Giusti et al., 2016) and (Loquercio et al., 2018) cannot.

References

Boroujerdian, B., Genc, H., Krishnan, S., Faust, A., Reddi, V.J.: Why compute matters for uav energy efficiency? (2018)

Delmerico, J., Scaramuzza, D.: A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. Memory 10, 20 (2018)

Frankle, J., Carbin, M.: The lottery ticket hypothesis: Training pruned neural networks. arXiv preprint arXiv:1803.03635 (2018)

Genc, H., Zu, Y., Chin, T.W., Halpern, M., Reddi, V.J.: Flying iot: Toward low-power vision in the sky. IEEE Micro 37(6), 40–51 (2017)

Giusti, A., Guzzi, J., Cireşan, D.C., He, F.L., Rodríguez, J.P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Di Caro, G., et al.: A machine learning approach to visual perception of forest trails for mobile robots. IEEE Robotics and Automation Letters 1(2), 661–667 (2016)

Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K.: Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. arXiv preprint arXiv:1602.07360 (2016)

LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in neural information processing systems. pp. 598–605 (1990)

Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P.: Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)

Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. arXiv preprint arXiv:1810.05270 (2018)

Loianno, G., Scaramuzza, D., Kumar, V.: Special issue on high-speed vision-based autonomous navigation of uavs. Journal of Field Robotics 35(1), 3–4 (2018)

Loquercio, A., Maqueda, A.I., del Blanco, C.R., Scaramuzza, D.: Dronet: Learning to fly by driving. IEEE Robotics and Automation Letters 3(2), 1088–1095 (2018)

Mohta, K., Sun, K., Liu, S., Watterson, M., Pfrommer, B., Svacha, J., Mulgaonkar, Y., Taylor, C.J., Kumar, V.: Experiments in fast, autonomous, gps-denied quadrotor flight. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). pp. 7832–7839. IEEE (2018)

Molchanov, P., Tyree, S., Karras, T., Aila, T., Kautz, J.: Pruning convolutional neural networks for resource efficient transfer learning. arXiv preprint arXiv:1611.06440 (2016)

Park, J., Boyd, S.: A semidefinite programming method for integer convex quadratic minimization. Optimization Letters 12(3), 499–518 (2018)

Quigley, M., Mohta, K., Shivakumar, S.S., Watterson, M., Mulgaonkar, Y., Arguedas, M., Sun, K., Liu, S., Pfrommer, B., Kumar, V., et al.: The open vision computer: An integrated sensing and compute system for mobile robots. arXiv preprint arXiv:1809.07674 (2018)

Zhang, D., Wang, H., Figueiredo, M., Balzano, L.: Learning to share: Simultaneous parameter tying and sparsification in deep learning (2018)