NEURAL NETWORK ARCHITECTURE SEARCH AND MODEL COMPRESSION FOR FAST PREDICTION OF UAS TRAFFIC DENSITY

Zhenhang Zhang, Chen Luo*, M. Cenk Gursoy, Qinru Qiu, Department of Electrical Engineering & Computer Science, Syracuse University, Syracuse, NY 13244*

Carlos Caicedo, School of Information Studies, Syracuse University, Syracuse, NY 13244

Adrian Solomon, Franco Basti, Thales Diigital Aviiation Customer Success and Innovation, Thales USA, Arlington, VA 22202

Abstract

The number of daily small Unmanned Aircraft Systems (sUAS) operations in uncontrolled low altitude airspace is expected to reach into the millions in the future. This makes UAS traffic density prediction a critical and challenging problem. In an Unmanned Aircraft System Traffic Management (UTM) framework, an accurate traffic prediction model is the key to manage congestion in very dense areas and allow us to evaluate different mission and resource provision plans in search for the best solution. In our previous work, a deep neural network (DNN) model has been proposed that predicts the instantaneous traffic density based on mission schedule information. However, one of the main drawbacks of the DNN is the high computational cost, which prevents us from applying the model to search for the best mission plan or best locations of launching and landing zones, because it requires exponentially large numbers of predictions based on different input combinations. In this paper, we aim to reduce the complexity of the neural network model. A neural architecture optimization framework that searches for the best compression ratio for each layer is developed. Overall, we are able to reduce the size of the traffic prediction model by 50%. Furthermore, because the pruning adds more regularization on the model and reduces the potential of overfitting, the compressed model also achieves small improvements in the prediction accuracy.

Introduction

New applications and services based on small Unmanned Aircraft Systems (sUAS) have gradually been introduced into urban city environments. The number of daily sUAS operations in uncontrolled low altitude airspace is expected to reach into the millions in the future. Complicated and high density UAS traffic imposes a significant burden on air traffic management, city planning and communication resource allocation. Fast and accurate UAS traffic density prediction is necessary for centralized management to coordinate and control UAS missions to avoid potential conflicts, ensure timely completion of missions and enhance operational safety through guaranteed connectivity to communication networks.

The UAS traffic density at a future time T can be considered as a function of the current UAS traffic density distribution, the flight environment (e.g. the location of the no-fly-zones), and the schedule of the future UAS missions within the target air space up to time T. Recent research showed that conventional neural networks with at least one hidden layer satisfy the universal approximation property [1] in that they can approximate an arbitrary continuous or measurable function given enough number of neurons in the hidden layer. In our previous work [2] a DNN model has been developed to predict the traffic density. Compared to previous machine learning based traffic predictors, our DNN takes the flight environment and detailed UAS mission launch information as the inputs, hence it can be generalized to different air spaces as long as the trajectory of each UAS is routed using the same algorithm. In other words, it will need no “down time” after a change of the no-fly-zone or the launching/landing zone information. The predicted traffic has high correlation to the actual traffic.

However, the complexity of the DNN model increases when more environment dynamics are considered. The traffic predictor is used in the inner loop of many dynamic traffic management applications, such as selecting the landing and launching area, scheduling a mission request, and allocating frequency band resources for UAV communications. Each of such tasks is a combinatorial optimization problem, where the optimal solution

This work is partially supported by the National Science Foundation I/UCRC Center for Alternative Sustainable and Intelligent Computing (ASIC).

* These authors contributed equally to this work.

cannot be found analytically. To efficiently explore the design space to search for the optimal configuration, the traffic predictor must be able to process many potential traffic scenarios in a short period of time. Therefore, in addition to accuracy, low complexity and low cost are other critical requirements for the traffic predictor.

The goal of this work is to compress the traffic prediction model to a smaller size without sacrificing the prediction accuracy. A network architecture search (NAS) method is developed to find the optimal DNN architecture in the spatial-temporal domain for better prediction performance and robustness. The overall framework is based on an architecture prediction model that predicts the accuracy of a compressed traffic density predictor. With the architecture search framework, we have reduced the size of the original traffic prediction model by 50% without losing accuracy.

Background and Related Works

Unmanned aerial vehicles have played an increasingly important role in many areas [3][4][5][6]. As one of the key factors in the design of UTM system, UAS traffic density prediction based on implementations that use deep learning methods has attracted great interest. [7][8][9][10].

In our previous work, a DNN model has been trained to predict the UAV traffic density with a correlation up to 0.945 and to predict traffic hot-spots with the Area Under the Receiver Operating Characteristics (AUROC) score up to 0.95. In order to consider the long, medium and short-term impact of the scheduled UAV missions, the model architecture shown in Fig. 1, with each term contains two convolutional neural networks (CNNs) in total six was used to extract features from the future mission plans for up to 6 cycles. And each cycle represents 10 simulation cycles which is 10 seconds for real time. Another CNN is needed to process the current traffic density map. Finally, a multi-layer fully connected network is used to fuse the information from different channels to form a vector embedding and a de-convolutional neural network is used to transform the vector into a 2D traffic map. The model has 10.6MB weight parameters and requires 7.85GOP (Giga Operation) computations for each prediction.

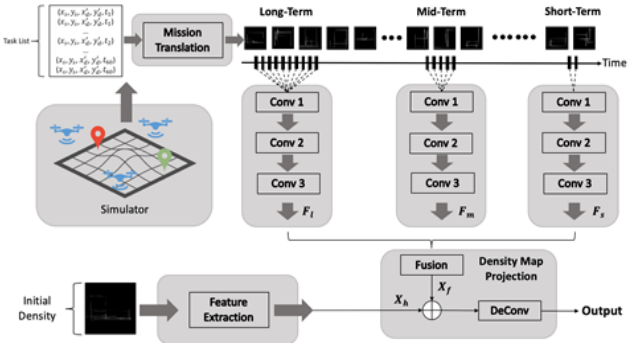


Fig. 1. Mission-Aware Spatio-Temporal Model Architecture.

Recent works have shown that weight pruning techniques [11][12][13] can significantly reduce the size of DNNs without loss of accuracy. [14] proposes a Structured Sparsity Learning (SSL) method to regularize the structures (i.e., filters, channels, filter shapes, and layer depth) of DNNs, and learn a compact structure from a bigger DNN to reduce computation cost. [15][16] proposed methods to overcome pruning ratio limitations. Other model compression methods, such as connection pruning [17][18] and low rank approximation [19][20], have also been proposed.

Most of the previous works aim at removing links and neurons in the given network until the pre-specified prune ratio is reached. How much can be pruned without affecting the performance of the model is unknown in advance. Different prune ratios must be tried until the right one is found. As each DNN layer can be pruned with a different ratio, the possible number of combinations increases exponentially with the size of the network. While pruning a network requires iterative training and fine tuning, it obviously is not feasible to try each possible prune configuration.

Our solution in this work is to train an architecture prediction model that predicts the performance of the compressed model based on the combination of its layer-wise compression ratio. The architecture prediction model allows us to quickly estimate the performance (e.g., traffic prediction accuracy) of any compressed model without lengthy pruning, and consequently narrows down the search space. The predictor allows us to find a good combination of prune ratios of different layers that reduces the size of the traffic predictor while maintaining its accuracy.

Model Reduction and Architecture Search

Motivations

Our traffic prediction network extracts mission information using 6 convolutional neural networks (CNNs) as shown in Fig. 1. Each network processes the set of missions scheduled to launch in a certain time period. All of them have the structure that is given in TABLE I. The missions scheduled to launch in the near future and far future play different roles in shaping the traffic density, therefore, although initialized with the same architecture, the weight parameters of the 6 CNNs diverge after training.

TABLE I. Architecture information of the original traffic predictor

Layer	Filter size	#of filters	Stride
1	4 x 4	64	2
2	2 x 2	128	1
3	2 x 2	256	1

Although the model gives high quality traffic density prediction, the complexity is also extremely high. Most of the computation is on the 6 CNNs for mission feature extraction. Our goal is to reduce the size of the CNNs to lower the model complexity and accelerate the prediction without sacrificing its prediction accuracy. This is achieved by pruning (unnecessary) filters in the CNN layers. Filter pruning is chosen here because it results in a dense weight matrix, thereby facilitating GPU acceleration. Furthermore, pruning a filter in layer i results in the removal of a channel in layer i +1 without creating sparsity.

Because their input has different importance to the prediction, it is natural to expect that the 6 CNNs should be compressed in different ways. Moreover, the layers in the CNNs must be pruned differently. For example, some layers play a dominant role in the prediction process. When the number of filters of these layers is relatively low, the model's final performance will become extremely poor. The overall complexity of the compressed model is determined by the prune ratios of each of the 3 layers in those 6 CNNs. Fig. 2 shows the relation between the prediction accuracy of the compressed model and its actual model size. Generally, a smaller size indicates a higher compression ratio. As we can see that a smaller model does not always have worse accuracy and vice versa. This is because the same compression ratio may correspond to different neural architectures, depending on how those 18 layers in Fig. 1 are pruned. To find the best architecture is to find the optimal combination of the 18 prune ratio variables. We also need to point out that, the accuracy of the compressed model is not necessarily lower than the original model. As we will show in the experimental results, the compressed model may slightly outperform the original model, as the highly regulated structure helps it to avoid overfitting.

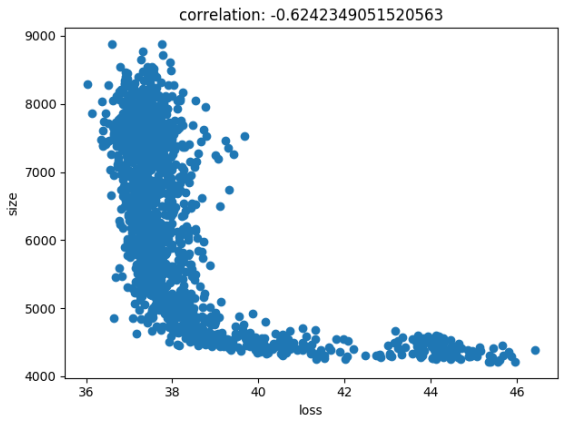


Fig. 2. Prediction accuracy versus compression ratio.

Exhaustively evaluating each possible compressed architecture to find its accuracy is not realistic as pruning a DNN is extremely time consuming. In this work we develop another DNN that predicts the accuracy of the compressed architecture to accelerate the design space exploration. We refer to it as the architecture predictor to distinguish from the traffic prediction model to be pruned.

Architecture Prediction Model

We expect that the prediction accuracy is a function of the combination of layer-wise prune ratios. A 5-layer DNN is trained to approximate this relationship. The structure of the DNN is shown in Fig. 3, where N is the batch size. The size of each layer is labeled in the figure. The input of the model is the vector of 18 prune ratios, and the output is the

accuracy of the compressed traffic predictor, measured by the mean square error of the traffic prediction.

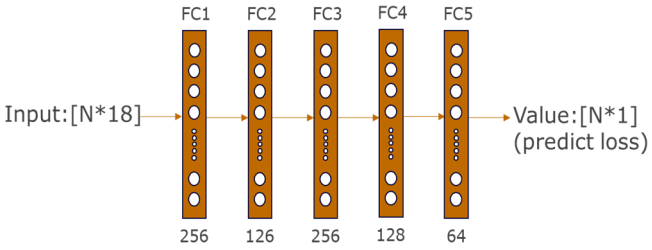


Fig. 3. Architecture prediction model.

The architecture predictor is trained using supervised training. To generate the training data, different prune ratio combinations are sampled and the original traffic predictor is pruned/trained accordingly. The resulting compressed model is tested and its mean square prediction error is used as the target value for the training. As we can see, generating each data point for the training set requires us to train a compressed model, hence is time-consuming. However, as we will show in the experimental results, the DNN can generalize the relation between the architecture and the model performance, and extend it to other prune ratio combinations that is not in the training set. While the training set is just a subset of the architecture space, the model can be used to explore the entire architecture space, which is much larger than the training set.

Training of the architecture predictor

The architecture predictor is slightly different from a regular regression model. The goal of the conventional regression models is to maximize the prediction accuracy; hence they usually use mean square error as the loss function. The architecture predictor is used to compare different compressed architectures (that have similar size) to find the one with the best accuracy. The absolute value of those architectures' accuracy is not important, as long as their relative order is predicted correctly. In other words, it is more important to maintain a high correlation between the predicted accuracy and target accuracy than minimizing the absolute difference between these two. Therefore, during the training, the following three techniques were adopted to improve the correlation between the predicted and the target accuracy.

Adding correlation to the loss function

For each training batch, after the forward pass, the correlation between the model's predicted value $\hat{y}$ and the target value y is calculated using the following equation.

$$corr = \frac{cov(\hat{y},y)}{\sqrt{var(\hat{y})*var(y)}} \quad (1)$$

The loss function of the training is a weighted combination of the mean square error (MSE) loss and the correlation loss as specified in Equation (2):

$$loss = \alpha * MSE + \beta * (1 - corr), \quad (2)$$

where α and β are hyper-parameters. Smaller α and larger β would cause the model to focus on increasing the overall correlation, while larger α and smaller β would cause the model to focus on reducing the overall loss.

Using the new loss function, the correlation between the prediction and the target value is improved. Although the model may lose some prediction accuracy, this is acceptable in our application.

Adjusting training data distribution

We found that adding data with different features as much as possible helps to improve the quality of the model. For instance, in some extreme cases, the randomly sampled compressed architecture has convolutional layers that have only one filter left. The accuracy of such traffic prediction network is obviously very poor. Although these architectures do not have real application significance, including them in the training set can help the model better understand the role of different convolutional layers in the training process.

Training with mixed batch size

In the training process, the architecture predictor be optimized one time in one epoch and the batch size is a crucial factor. We dynamically change the batch size as the epoch increases during training. The batch size increases sequentially. For example, the first epoch's batch size is 4, the batch size of the second epoch is 10, and the batch size of the third epoch is 90.

We use *partial correlation* to refer to the prediction and target value correlation in the same batch and use *overall correlation* to refer to the

correlation in the entire training set. We found that using a fixed batch size, no matter how large or small, it is hard to balance the overall and partial correlation. When the batch size is small, we get very high partial correlation. However, the model does not give consistent prediction from one batch to another, hence the overall correlation is poor. When the batch size is large, the overall correlation is improved. However, a single outlier does not affect the overall correlation significantly, and this is reflected as the poor partial correlation. Therefore, a mixed batch size is adopted. This method can integrate the advantages of large and small batch sizes and consider the partial correlation while improving the overall correlation. Since this method requires constant batch size changes, the model also needs to train more epochs to achieve the best results.

Using the above three methods, we finally increased the model correlation from 0.71 to 0.92 and even reached 0.96 in the validation set.

Incremental training to specific corner

Since our purpose is to reduce the size of the traffic predictor as much as possible while retaining its accuracy, we are interested more in the data points located at the lower left corner of the size versus accuracy (or loss) design space shown in Fig. 2. The architectures located in this corner have smaller size and higher accuracy, therefore they are referred to as *effective architectures*. And the architectures located outside this corner are referred to as *ineffective architectures* as they have larger size and/or lower accuracy. During the training process, we try to specialize the model to give better prediction to the data points corresponding to the effective architectures. This is achieved by adding more data points corresponding to the effective architectures to the training set.

However, to distinguish between an effective architecture and an ineffective architecture is not easy. Given a randomly sampled prune vector, while the size of the compressed model can be easily estimated, its accuracy is unknown unless we actually compressed the mode, trained and tested it. Unfortunately, there are more ineffective architectures than effective architecture. Since an ineffective architecture does not help the training process as much as an effective architecture, much of the effort in generating the training data will be wasted. To address this problem, we apply incremental training as described in Fig. 4. Incremental training accelerates the model's learning speed for the feature that we need by adding the validated model prediction data into the training set. First, we randomly sample 300-400 compressed architectures, train and test them to generate the initial training set. The architecture prediction model is pre-trained on the initial training set. Taking advantage of the pre-trained model, we then predict the accuracy of another set of randomly sampled architectures, and select the architecture whose accuracy is lower than a certain threshold. The corresponding compressed architecture will be generated and tested. These new data will form a new training batch to further refine the model. This procedure is repeated iteratively until the overall correlation achieves a predefined threshold. Totally 2300 training architecture data is generated in this work.

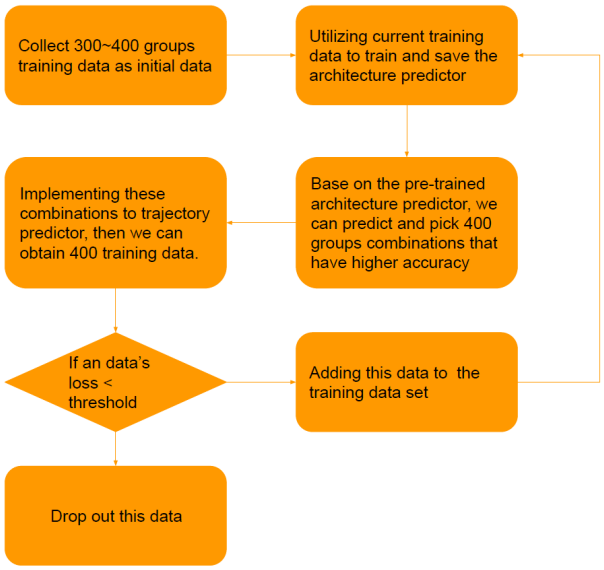


Fig. 4. Flow of incremental training.

Experimental Results

Original model vs. compressed model

Using the trained architecture predictor, we predicted the accuracy of 30 million random architectures with a size below a given threshold. 30 architectures with the best predicted size vs. accuracy tradeoffs were selected and implemented. The green points in Fig. 5 show the size vs. accuracy tradeoffs of those architectures. The blue points in the figure give

the size vs. accuracy of the 2300 training architectures. As we can see, since the 30 selected architecture have been filtered by the predictor, in average, they have

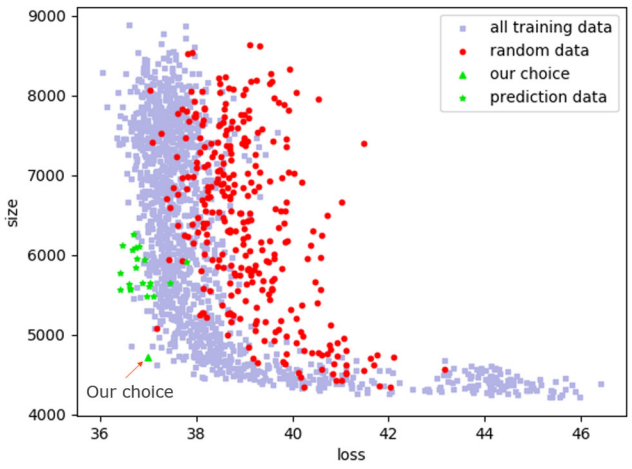


Fig. 5. Size-accuracy tradeoffs of the 30 selected architecture.

better accuracy than the training architecture. The point located at the lower left corner is the best architecture.

TABLE II shows the percentage reduction of size, accuracy and inference time of the best selected architecture compared to the original model. As we can see, the compressed model can slightly improve the accuracy (negative accuracy reduction) even though the size is compressed by 55%. The reason that the accuracy is slightly improved is because the compressed architecture has less weight parameters and hence is less likely to do overfitting. The inference speed is also greatly improved due to the reduced complexity. This means that our compressed model consumes fewer resources to achieve the same prediction effect.

TABLE II. Original Model vs. Compressed Model

	Original Model	**Compressed Model**	**Comparison**
Model size	10,562 KB	4,713 KB	-55.38%
Loss	37.575	37.413	-0.431%
Time (200 Samples)	0.068 s	0.025 s	-63.24%

Comparison with randomly found architecture

Searching for the optimal architecture by exhaustively evaluating different structures may be feasible for simple and small neural networks, since the possible architectures are limited and the training of a small neural network does not take too much time. However, when the network size gets increased, this is no longer practical. In order to find the optimal size and accuracy tradeoff, a much larger number of compressed architectures must be sampled and tested due to the exponentially increased search space, and each one requires longer training and evaluation times.

The biggest advantage of our approach is that the architecture prediction model can be trained based on a limited number of compressed architectures and it can generalize the learned architecture-accuracy relationship to a wider range. In our work, a total of 2300 compressed architectures were generated to train the model. The average mean square error (MSE) loss between the predicted value and label is only 0.14, and the correlation can reach 0.926. Using trained models, we can evaluate tens of millions of possible compressed architectures within minutes.

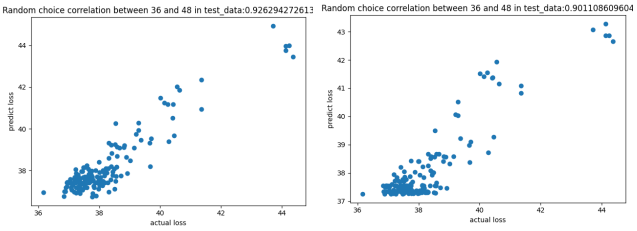
TABLE III. Original Model vs. Compressed Model (Random search)

Best Random Result			
	Original Model	**Compressed Model**	**Comparison**
Model size	10,562 KB	4,490 KB	-57.49%
Loss	37.575	39.645	+5.221%
Minimum Loss Result			
	Original Model	**Compressed Model**	**Comparison**
Model size	10,562 KB	8,286 KB	-21.55%
Loss	37.575	36.038	-4.265%
Minimum Size Result			
	Original Model	**Compressed Model**	**Comparison**
Model size	10,562 KB	4,210 KB	-60.14%
Loss	37.575	45.361	+17.16%

We randomly selected 300 architectures and plot them in the size-accuracy space in Fig. 5 using red color. As we can see, compared to random compressed architecture, the size-accuracy plot of the training data

that we generated for the incremental training process (i.e. data points in grey) has already shifted to the lower left side notably. This means these training architectures exhibit better size-accuracy tradeoff. And the green data points, which are filtered by the predictor, are further improved compared to the training data. We also selected 3 of those random architectures with the best size-accuracy trade off. Their reduction of its size, accuracy and inference time compared to the original model is reported in TABLE III. It shows that at the similar compress ratio, the randomly selected architecture has much lower accuracy.

Impact of correlation loss



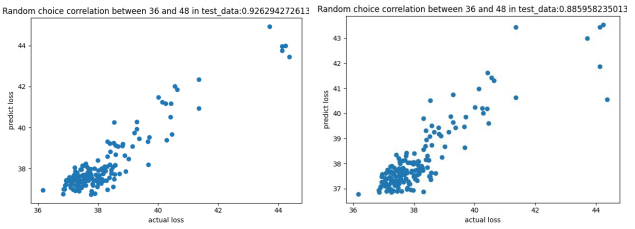
(a) W. correlation **(b) W/O. correlation**

Fig. 7. Loss Function Representative Correlation Result.

As mentioned above, we modified the loss function of the architecture predictor by adding the correlation loss. The modified loss function can directly boost the correlation between the predicted and actual value. Fig. 7 gives the scatter plots between the prediction and the target value for the architecture predictor with and without the correlation. As we can see, if only MSE loss is considered in training, when the target accuracy (loss) is high (low), the model cannot distinguish the performance of different architectures and will simply predict the same value. This obviously will not help us select the optimal architecture.

Impact of different batch size

As we discussed before, the batch size also has an impact on the quality of the trained model and it helps to improve data point distribution. Fig. 6 shows the relation between the predicted value and target value, when trained with a mixed batch and a constant batch with size N=90. As we can see, the model trained with mixed batch size has better correlation. Actually, the model trained mixed batch size has correlation as high as 0.9262. And the models trained fixed batch with size N=10, 50, and 90, have correlations 0.898, 0.879 and 0.885, respectively.



(a) **(b)**

Fig. 6. Impact of batch size (a) Mixed (b) N = 90

Conclusions

In this paper we proposed a technique to search for the best prune strategy for the UAV traffic density prediction model to facilitate fast traffic prediction. With the help of the proposed architecture prediction model, we can efficiently evaluate the performance of many different compressed architectures in the design space and select the optimal one We found out that adding correlation into the loss function, dynamically adding new data during the training phase and training with a mixed sized batch can significantly improve the correlation of our architecture prediction model. In our results, we achieved a reduction in the size of the original traffic prediction model by 55% while keeping similar accuracy as the original model. At the same time, the model execution time speeds up by 60%.

References

[1] Csáji, Balázs Csanád. "Approximation with artificial neural networks." Faculty of Sciences, Etvs Lornd University, Hungary 24.48 (2001): 7.

[2] Z. Zhao, C. Luo, F. Basti, M. C. Gursoy, C. Caicedo, and Q. Qiu, "Machine Learning-Based Traffic Management Model for UAS Instantaneous Density Prediction in an Urban Area." 2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC). IEEE, 2020.

[3] A. Puri, “A survey of unmanned aerial vehicles (uav) for traffic surveillance,” Department of

computer science and engineering, University of South Florida, pp. 1–29, 2005.

[4] S. G. Gupta, M. M. Ghonge, and P. Jawandhiya, “Review of unmanned aircraft system (uas),” International journal of advanced research in computer engineering & technology (IJARCET), vol. 2, no. 4, pp. 1646– 1658, 2013.

[5] P. Liu, A. Y. Chen, Y.-N. Huang, J.-Y. Han, J.-S. Lai, S.-C. Kang, T.-H. Wu, M.-C. Wen, M.-H. Tsai et al., “A review of rotorcraft unmanned aerial vehicle (uav) developments and applications in civil engineering,” Smart Struct. Syst, vol. 13, no. 6, pp. 1065–1094, 2014.

[6] F. Nex and F. Remondino, “Uav for 3d mapping applications: a review,” Applied geomatics, vol. 6, no. 1, pp. 1–15, 2014.

[7] Formosa, Nicolette, et al. "Predicting real-time traffic conflicts using deep learning." Accident Analysis & Prevention 136 (2020): 105429.

[8] Yao, Huaxiu, et al. "Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction." Proceedings of the AAAI conference on artificial intelligence. Vol. 33. No. 01. 2019.

[9] T. Zhao, Y. Xu, M. Monfort, W. Choi, C. Baker, Y. Zhao, Y. Wang, and Y. N. Wu, “Multi-agent tensor fusion for contextual trajectory prediction,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 12 126–12 134.

[10] Z. Shi, M. Xu, Q. Pan, B. Yan, and H. Zhang, “Lstm-based flight trajectory prediction,” in 2018 International Joint Conference on Neural Networks (IJCNN). IEEE, 2018, pp. 1–8.

[11] X. Dai, H. Yin, and N. K. Jha, “Nest: A neural network synthesis tool based on a grow-and-prune paradigm,” arXiv preprint arXiv:1711.02017, 2017.

[12] S. Han, J. Pool, J. Tran, and W. Dally, “Learning both weights and connections for efficient neural network,” in Advances in Neural Information Processing Systems (NIPS), pp. 1135–1143, 2015.

[13] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding,” International Conference on Learning Representations (ICLR), 2016.

[14] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, “Learning structured sparsity in deep neural networks,” in Advances in Neural Information Processing Systems, pp. 2074–2082, 2016.

[15] Zhang, Tianyun, et al. "Adam-admm: A unified, systematic framework of structured weight pruning for dnns." arXiv preprint arXiv:1807.11091 2.3 (2018).

[16] Zhang, Tianyun, et al. "Structadmm: A systematic, high-efficiency framework of structured weight pruning for dnns." arXiv preprint arXiv:1807.11091 (2018).

[17] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In Advances in Neural Information Processing Systems, pages 1135–1143. 2015.

[18] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, 2015.

[19] Misha Denil, Babak Shakibi, Laurent Dinh, Marc' Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In Advances in Neural Information Processing Systems, pages 2148–2156. 2013.

[20] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In Advances in Neural Information Processing Systems, pages 1269–1277. 2014.