Signal Compression for Efficient Partitioning of Deep Neural Networks

Flávio Brito, Ingrid Nascimento, Luan Goncalves, Silvia Lins, Neiva Linder and Aldebaro Klautau

***Abstract*— Fifth generation (5G) mobile networks are adopting several techniques to provide higher data rates while meeting strict latency requirements. Advanced compression techniques and distributed data processing are among them supporting cost efficient network deployments. Also, the usage of machine learning techniques to optimize telecommunication systems is gaining momentum, specially after promising results with deep learning models. Following this trend, there are investigations towards splitting the processing of such models between different network nodes, making these applications more suitable and adaptable to scenarios with low processing capacity nodes. Therefore, in this work we investigate and propose different compression techniques for efficient partitioning of deep neural networks. We combine several splits with quantization and Huffman coding compression algorithms, providing insights on the configurations with the best performance. We compress the output score of the model to reduce the overhead of transmitting such scores through the network and evaluate how the accuracy is affected by this compression, and which compression technique provides the best performance.**

***Keywords*— Deep Learning, 5G, Telecommunications, Split Research, Compression**

I. Introduction

Fifth-generation mobile networks promote cost-efficient deployments embracing more adaptable network architectures. Centralized Radio Access Networks (C-RANs) come into sight in this context providing increased flexibility and lower cost of deployment. On the other hand, it imposes strict requirements in the fronthaul links (i.e. the connection between the centralized baseband processing and the radio nodes at the network edge).

To address such requirements, several techniques are being evaluated and adopted in C-RAN scenarios towards reducing the overhead and optimizing data transmission. One of the challenges in C-RAN is the increased bandwidth required for transmission in fronthaul links. In this sense, several works evaluate the efficiency and computational cost of different compression methods for fronthaul traffic.

A. Compression methods for C-RAN scenarios

In [1] the authors propose a low-latency baseband signal compression algorithm that adopts resampling, block scaling, Scalar Quantization (SQ) and Huffman coding to reduce the required bandwidth for fronthaul data transmission. Also, the authors in [2] use the same block structure as described in [1] but use Vector Quantization (VQ) instead of SQ. Using VQ decreases the Error Vector Magnitude (EVM) but with the cost of higher computational burden in comparison to SQ.

The authors in [3] use the same structure from [1] and [2] but adopt Trellis Coded Quantization (TCQ) [4] which gives a lower EVM than SQ with a lower computational burden than VQ. In contrast with the works [1], [2] and [3], the authors in [5] use Linear Predictive Coding (LPC) to predict the n_{th} sample of an OFDM symbol and quantize the error prediction with SQ, compressing it with Huffman Coding. The LPC-based technique was also improved in [6] with the implementation of high rate adaptation.

The inherent complexity associated with 5G scenarios is also challenging in C-RAN architectures, motivating the exploration of Machine Learning (ML) techniques towards making mobile network deployments more cost-efficient. One promising ML technique under intense investigation in this sense by the research community is Deep Learning (DL).

B. Deep Learning for 5G

The authors in [7] proposed an adaptive beam management scheme based on deep learning. Also, the authors in [8] developed a dynamic network slicing technique for short term traffic prediction, applying deep learning techniques. The authors in [9] use deep learning for power allocation to reduce the effects of imperfect SIC (Successive Interference Cancellation) for the NOMA (Non-Orthogonal Multiple Access) system. Deep learning was also applied successfully in 5G networks security [10] and congestion control fields [11].

1) Challenges in Deep Learning adoption: An issue regarding the adoption of deep learning techniques is that some of them require high computational capacity devices to perform real-time processing. If low capacity nodes are adopted, it may either negatively impact accuracy or the application latency and response time. To overcome the computational cost and latency impact, there are works assessing where is the best place to deploy and process the neural networks [12]. In another study [13] the authors propose a split to the neural network model in order to perform its processing in different network nodes. Also, another work regarding neural network splitting is described in [14], where the authors investigate three possible options: processing performed entirely at the edge; entirely at the cloud, or splitting the model to deploy the initial part of the neural model at the edge and the final part at the cloud.

2) Compression and Partitioning of Deep Neural Networks: To continue evolving the investigation in this field, this work

LASSE-5G & IoT research Group, Federal University of Pará (UFPA), Belém-PA, Brazil, Ericsson Research, Kista, Sweden, E-mails: flavio.brito, ingrid.nascimento, luan.goncalves@itec.ufpa.br silvia.lins, neiva.linder@ericsson.com, aldebaro@ufpa.br

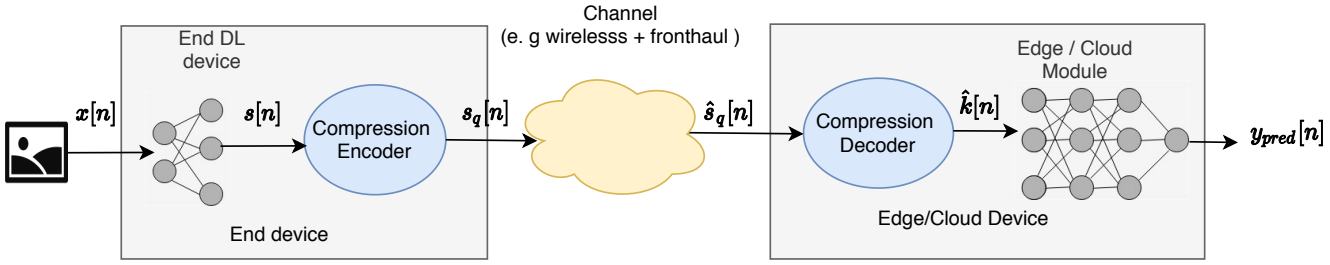


Fig. 1: System model. Our proposed work is to split and compress the intermediate output of the model.

proposes a framework for the deployment of the deep neural network models based on both splitting and compression techniques. We first split the neural networks into two parts and then we apply compression algorithms in the scores (i.e intermediate output of the neural network). To summarize, our contribution is described as follows:

- We propose a framework for deep learning deployment adopting splitting and compression algorithms.
- We apply different compression techniques based on quantization and Huffman coding in order to reduce the total of bits sent to the cloud.

The rest of this paper is organized as follows: Section II describes the system model proposed for the deep neural network partitioning and compression. Section III details the experiments performed to assess its performance. Section IV presents the results obtained with the proposed solution and Section V provides the conclusions of the paper.

II. System model

Fig. 1 describes the model of the system used in this work. As can been seen, the original deep learning model is split into two parts: the end device and the edge/cloud device. We adopt images as the system input, represented by the variable $x[n]$. The variable $x[n]$ is the input for the first part of the (partitioned) deep neural model, deployed at the end device. The variable variable $s[n]$ represents the output of this model, which is only the intermediate output. Before sending the variable $s[n]$ to the cloud, $s[n]$ is compressed used a combination of quantization and Huffman encoding resulting in the compressed signal $s_q[n]$, which is sent to the cloud.

At the cloud side, the compressed signal $\hat{s}_q[n]$ (which is the signal $s_q[n]$ plus some noise) is decompressed by the quantization and Huffman decoder block resulting in the signal $\hat{k}[n]$. Finally, the signal $\hat{k}[n]$ is used as input to the Cloud DL module, resulting in the predicted signal $\hat{y}_{pred}[n]$.

In this work, we evaluate two compression algorithms for different split configurations. The first compression is composed of a scalar quantization and Huffman coding whereas the second compression system is composed of the TCQ and Huffman coding. Also, for this work, the serial convolutional network VGG16 [15] is used in the experiment.

III. Experiment description

In this section, we describe the experiment scenario. First of all, the dataset of our experiment is a smaller version of the dog-vs-cat Kaggle dataset [16] which is composed of two classes: cat and dog. The training set is composed of 100 images of cats and 100 images of dogs and the test set are composed of 20 images of cats and 20 images of dogs.

The model which will be split is the VGG16 which is composed of 13 convolutional layers, 5 pooling layers and 3 dense layers. In this experiment, we use Keras to load and process the architecture. The Keras model is trained in the ImageNet dataset and, therefore, the output layer is composed of 1000 neurons. In this work, we replace the number of neurons in each dense layer in a way that the first dense layer is composed of 4096 neurons, the second dense layer is composed of 1024 neurons; the third is composed of 512 neurons. Also, we added a fourth dense layer which is the output layer and is composed of 2 neurons, one for each class. All the dense layers, except the output layer, uses the ReLu activation function and the output layer uses the softmax function. We use this new VGG16 configuration and train again the model using our smaller version of the dog-vs-cat dataset. Before the training process, the kernel and the bias weights of all the convolutional layers were freeze in order to guarantee that only the dense layer will be updated in the training process. In the training process, the training dataset is divided into train and validation sets where 70% of the original training dataset is designed to the train set and 30% of the test dataset is designed to the validation set.

The Compression system is composed of two blocks: the first one is the quantization block. In this work, we use two quantization techniques: Scalar Quantization (SQ) and Trellis Coded Quantization (TCQ). The second block is the Huffman algorithm which is responsible for reducing the necessary amount of bits that will be transmitted to the cloud. In this experiment, we are considering that there is no impact w.r.t. to noise added from the transmission to the cloud, i.e. $s_q[n] = \hat{s}_q[n]$.

For the split configuration, in this work we have chosen nine different configurations. Table I describes each split configuration. For the first split configuration (ID = 1) for example, only one Layer (input layer) is deployed at the edge where the rest is deployed at the cloud. For the second split configuration (ID = 2), the two first layers of the VGG16 (input layer and the first convolutional layer) are deployed at the edge and the other layers are deployed at the cloud.

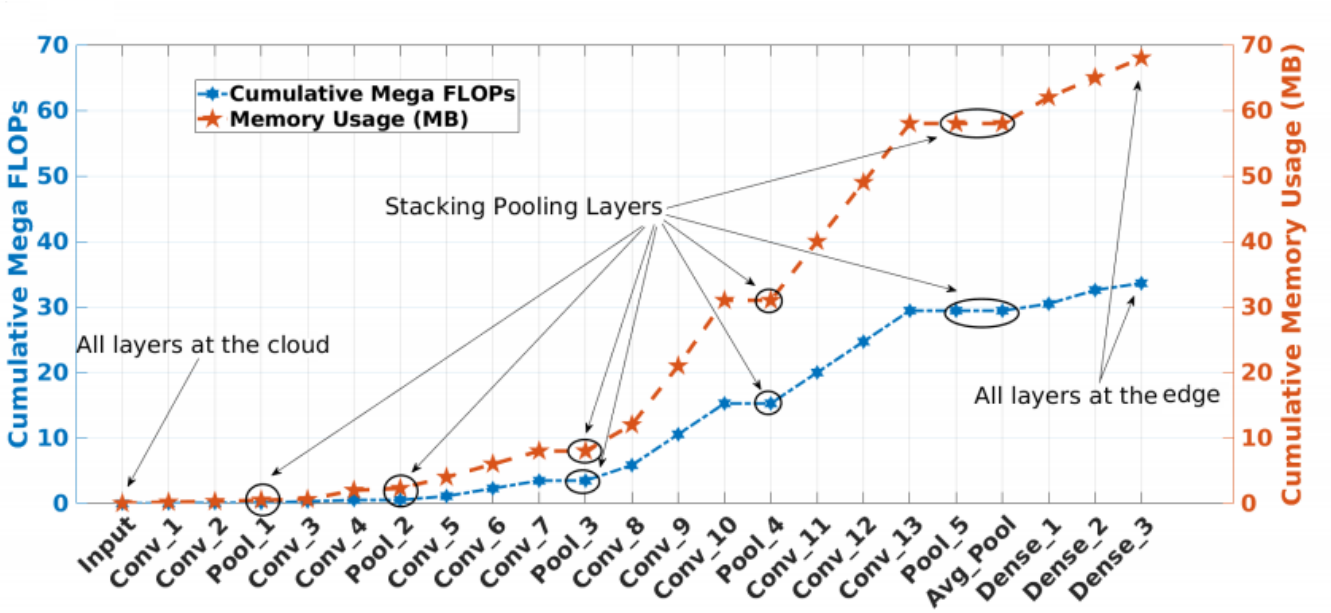


Fig. 2: Accumulative Mega Flops at the edge. The first layer is when all the model is deployed at the cloud where the last layer is when all the model is deployed at the edge. The cumulative curve does not increase when we stack pooling layers as shown in the graph.

Split configuration ID	Layers at the edge	Layers at the cloud
1	1	22
2	2	21
3	3	20
4	4	19
5	5	18
6	6	17
7	7	16
8	8	15
9	9	14

Table I: Split configuration of the VGG16.

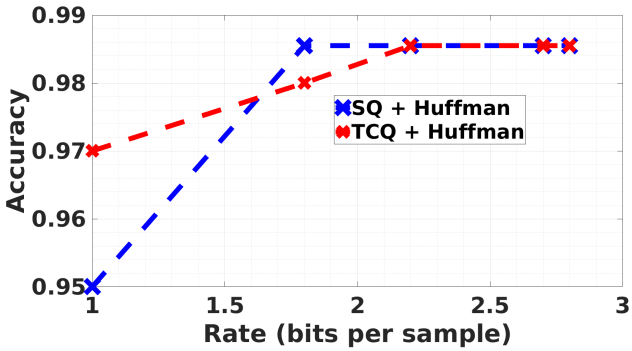


Fig. 3: Accuracy results when the first convolutional layer is deployed at the end device and the output scores are compressed using Trellis Coded Quantization with Huffman Coding vs. Scalar Quantization with Huffman coding.

IV. Results

The first results discussed here are regarding the Floating-Point Operations per Second (FLOPs) needed to process the neural network. Fig. 2 shows the cumulative Mega FLOPs required at the edge for the different split configurations. The x-axis shows how many layers are deployed at the edge and the y-axis shows the cumulative Mega FLOPs required. It is important to notice that the Mega FLOPs does not increase so much when pooling layers are stacked at the edge. However, convolutional layers require more FLOPS, and a linear increase is noticed when that layer is deployed at the edge.

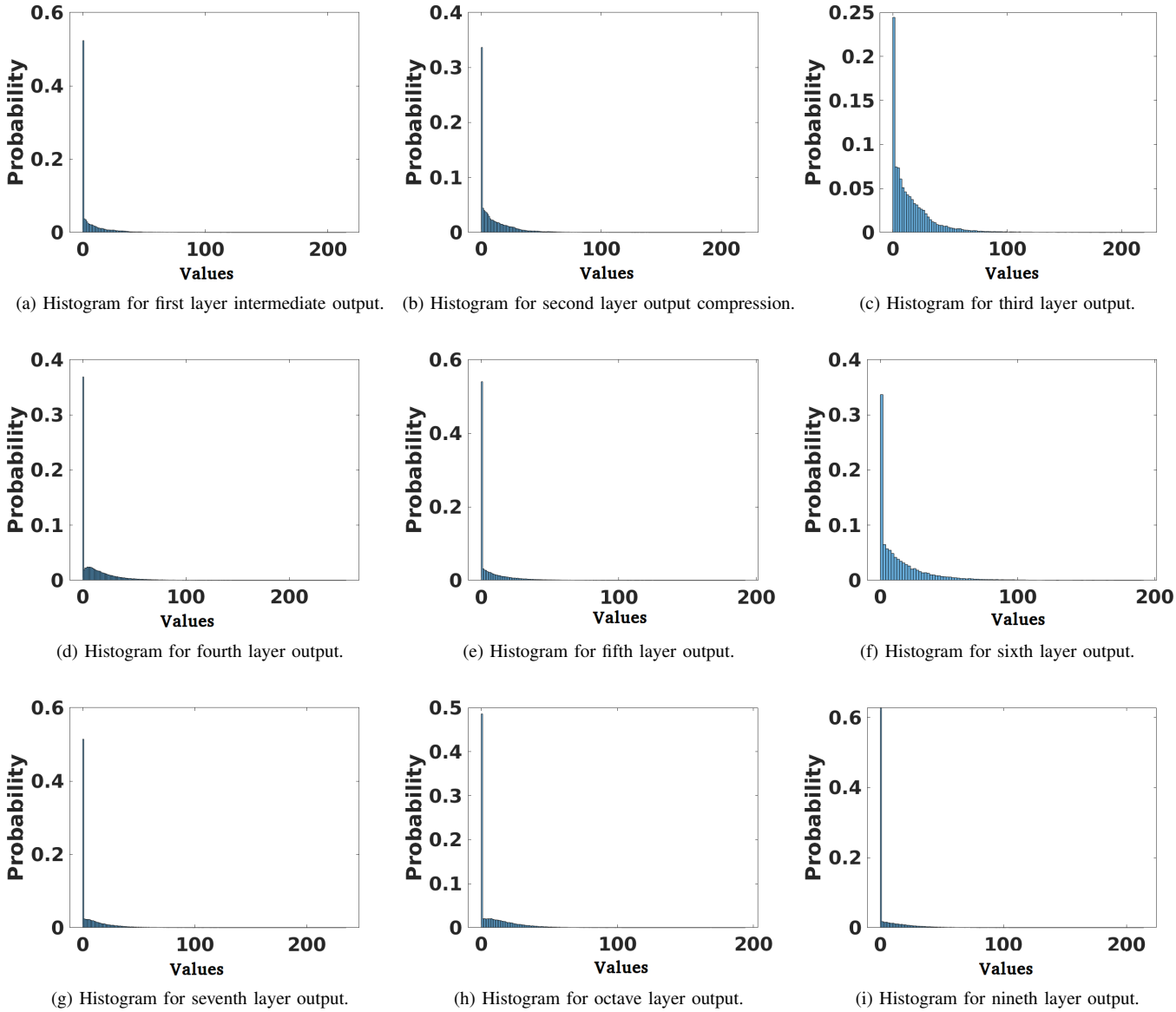
Fig. 3 shows the accuracy when we deploy the first convolutional layer at the end device. As can be seen in this example, using Scalar Quantization at the quantization block results in faster convergence of the system with an accuracy of 0.98. Therefore, using SQ can be a good choice for the compression method.

Regarding the histogram pattern of the scores, Fig. 4 shows the histogram of all nine intermediate outputs for a system with a quantizer of 8 bits. As can been seen, in all the histograms, the probability distribution is a function that concentrates almost all the values around zero. The zero concentrated values in the histogram can explain the high compression rate achieved in the experiments.

It is relevant to evaluate the entropy and the Signal to Noise Quantization Ratio (SQNR) performance for the 8 bits quantizer using scalar quantization. As can been seen, Fig. 5 shows that, for each layer used in the experiment, we achieve at least an SQNR of 30 dB, which means that a low

V. Conclusions

This work assessed the joint usage of split processing and compression techniques at the intermediate outputs of deep convolutional neural network models. The results showed that it is possible to achieve high compression rates with minor impact on the model accuracy.



(a) Histogram for first layer intermediate output. (b) Histogram for second layer output compression. (c) Histogram for third layer output.

(d) Histogram for fourth layer output. (e) Histogram for fifth layer output. (f) Histogram for sixth layer output.

(g) Histogram for seventh layer output. (h) Histogram for octave layer output. (i) Histogram for nineth layer output.

Fig. 4: Histogram of each experiment using the system with a quantizer of 8 bits.

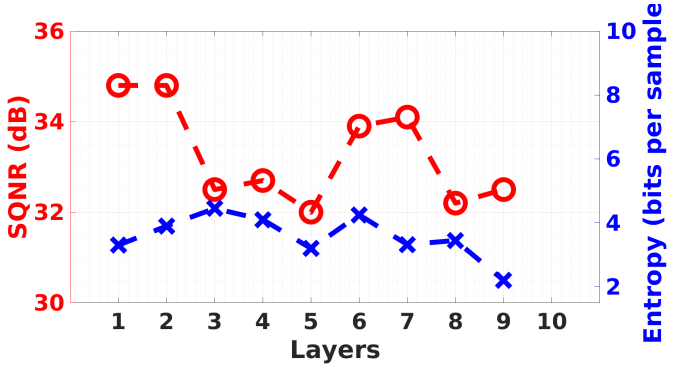


Fig. 5: Entropy and Signal to Noise Quantization Ratio (SQNR) performance for each layer.

Our approach was the following:

- We used a small version of the dog-vs-cat dataset contains 100 images of dog and cat for training (50 images of cats and 50 images of dogs). For testing, we used a test dataset containing 20 images of cats and 20 images of dogs.
- Also, we used python2.7 equipped with Keras in order to load the VGG16 model and we changed the dense layers of the model, and, in the output, layer, we use only 2 neurons.
- We froze the convolutional learnable parameters and we trained the modified VGG16 on this dataset. We set 70% of the training images for the trains set and the remaining 30% of the images were used in the validation set. We also used a small portion of the training set to generate the uniform codebook.
- In the test step, we used python to load the model (and for splitting). We used Matlab to quantize, get the histogram,

and perform the entropy calculation. In this step, we used a quantizer with 2,3,4,5,6,7 and 8 bits.

- After the Matlab step, the python was used again to set the Matlab quantized signal as the input to the cloud model.
- In our approach, we consider that there is no noise between the signal output of the edge and input of the cloud. Therefore, we consider that $s_q[n] = \hat{s}[n]$.

As future work, the following steps are envisioned:

- Using another quantization algorithm (e.g. vector quantization). This could result in a more accurate model but has as drawback the increased computational complexity.
- Evaluate other serial models like the well know AlexNet and VGG19 in order to generalize our results.
- Evaluate non-serial models like Residual Net and Dense Nets in order to evaluate how the intermediate output will be affected by the non-linear system.
- Evaluate the compression of the weights and bias of each neural network in order to reduce the total size of the edge model.
- Use the Lloyd algorithm in order to optimize the codebook parameters to get higher accuracy.

ACKNOWLEDGMENT

This work was supported by the Innovation Center, Ericsson Telecomunicações S.A, CNPq and Capes Foundation, Brazil.

REFERÊNCIAS

[1] Bin Guo, Wei Cao, An Tao, and Dragan Samardzija. LTE/LTE-A signal compression on the CPRI interface. *Bell Labs Technical Journal*, 18(2):117–133, 2013.

[2] Hongbo Si, Boon Loong Ng, Md Saifur Rahman, and Jianzhong Zhang. A novel and efficient vector quantization based CPRI compression algorithm. *IEEE Transactions on vehicular technology*, 66(8):7061–7071, 2017.

[3] Flávio Brito, Miguel Berg, Chenguang Lu, Leonardo Ramalho, Ilan Sousa, and Aldebaro Klautau. A Fronthaul Signal Compression Method Based on Trellis Coded Quantization. In *2019 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6. IEEE, 2019.

[4] Michael W Marcellin and Thomas R Fischer. Trellis coded quantization of memoryless and Gauss-Markov sources. *IEEE transactions on communications*, 38(1):82–93, 1990.

[5] Leonardo Ramalho, Maria Nilma Fonseca, Aldebaro Klautau, Chenguang Lu, Miguel Berg, Elmar Trojer, and Stefan Höst. An LPC-based fronthaul compression scheme. *IEEE Communications Letters*, 21(2):318–321, 2016.

[6] Leonardo Ramalho, Igor Freire, Chenguang Lu, Miguel Berg, and Aldebaro Klautau. Improved LPC-based fronthaul compression with high rate adaptation resolution. *IEEE Communications Letters*, 22(3):458–461, 2018.

[7] Woongsoo Na, Byungjun Bae, Sukhee Cho, and Nayeon Kim. Deep-learning Based Adaptive Beam Management Technique for Mobile High-speed 5G mmWave Networks. In *2019 IEEE 9th International Conference on Consumer Electronics (ICCE-Berlin)*, pages 149–151. IEEE, 2019.

[8] Qize Guo, Rentao Gu, Zihao Wang, Tianyi Zhao, Yuefeng Ji, Jian Kong, Riti Gour, and Jason P Jue. Proactive Dynamic Network Slicing with Deep Learning Based Short-Term Traffic Prediction for 5G Transport Network. In *2019 Optical Fiber Communications Conference and Exhibition (OFC)*, pages 1–3. IEEE, 2019.

[9] Worawit Saetan and Sakchai Thipchaksurat. Application of Deep Learning to Energy-Efficient Power Allocation Scheme for 5G SC-NOMA System with Imperfect SIC. In *2019 16th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*, pages 661–664. IEEE, 2019.

[10] Nagarathna Ravi, P Vimala Rani, and S Mercy Shalinie. Secure Deep Neural (SeDeN) Framework for 5G Wireless Networks. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6. IEEE, 2019.

[11] Ingrid Nascimento, Ricardo Souza, Silvia Lins, Andrey Silva, and Aldebaro Klautau. Deep reinforcement learning applied to congestion control in fronthaul networks. In *2019 IEEE Latin-American Conference on Communications (LATINCOM)*, pages 1–6. IEEE, 2019.

[12] Diego Dantas, Kaio Forte Carnot Braun, Andrey Silva Flavio Brito, Neiva Linder Silvia Lin, and Aldebaro Klautau. Testbed for Connected Artificial Intelligence using Unmanned Aerial Vehicles and Convolutional Pose Machines. 2019.

[13] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Distributed deep neural networks over the cloud, the edge and end devices. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 328–339. IEEE, 2017.

[14] Jong Hwan Ko, Taesik Na, Mohammad Faisal Amir, and Saibal Mukhopadhyay. Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE, 2018.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[16] Dogs vs. Cats: Create an algorithm to distinguish dogs from cats. `https://www.kaggle.com/c/dogs-vs-cats`, 2014. [Online; accessed 3-April-2020].