Partitioning of CNN Models for Execution on Fog Devices*

Swarnava Dey
TCS Research and Innovation
Kolkata, West Bengal, India
swarnava.dey@tcs.com

Arijit Mukherjee
TCS Research and Innovation
Kolkata, West Bengal, India
mukherjee.arijit@tcs.com

Arpan Pal
TCS Research and Innovation
Kolkata, West Bengal, India
arpan.pal@tcs.com

Balamuralidhar P
TCS Research and Innovation
Bangalore, Karnataka, India
balamurali.p@tcs.com

ABSTRACT

Fog Computing has in recent times captured the imagination of industrial and research organizations working on various aspects of connected livelihood and governance of smart cities. Improvements in deep neural networks imply extensive use of such models for analytics and inferencing on large volume of data, including sensor observations, images, speech. A growing need for such inferencing to be run on devices closer to the data sources, i.e. devices which reside at the edge of the network, popularly known as *fog devices* exists, in order to reduce the upstream network traffic. However, being computationally constrained in nature, executing complex deep inferencing models on such devices has been proved difficult. This has led to several new approaches to partition/distribute the computation and/or data over multiple fog devices. In this paper we propose a novel depth-wise input partitioning scheme for CNN models and experimentally prove that it achieves better performance compared to row/column or grid based schemes.

CCS CONCEPTS

• **Computing methodologies** → *MapReduce algorithms*; • **Computer systems organization** → *Cloud computing*; *Neural networks*;

KEYWORDS

CNN, distributed, Edge, Fog, Cloud, DCNN, convolution, parallel

ACM Reference Format:
Swarnava Dey, Arijit Mukherjee, Arpan Pal, and Balamuralidhar P. 2018. Partitioning of CNN Models for Execution on Fog Devices. In *The 1st ACM International Workshop on Smart Cities and Fog Computing (CitiFog'18), November 4, 2018, Shenzhen, China.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3277893.3277899

*Produces the permission block, and copyright information

CitiFog'18, November 4, 2018, Shenzhen, China

ACM ISBN 978-1-4503-6051-7/18/11…$15.00
https://doi.org/10.1145/3277893.3277899

1 INTRODUCTION

In recent years industries and research organizations have heavily invested in Fog Computing where computational methods are placed closer to the data sources at the edge of the network. Data analytic applications processing large volume of sensor data, images, videos, sounds etc. to generate inferences are primary candidate applications for such a processing architecture as processing the data closer to the source ensures less data traffic upstream. Example implementations of data analytic applications in Smart City are available in smart city transport systems [11], smart city healthcare [22, 23], detection of illegal garbage dumping [3] and several others. We redirect the reader to a recent survey [14] that highlights challenges and opportunities in *Artificial Intelligence(AI)*-based frameworks for smart cities. It is noteworthy that many of the above mentioned and several other data analytic applications for smart city are adopting Deep Learning (DL)/Inferencing techniques due to availability of state of the art (SoA) learning models ready for transfer learning and fine tuning, resulting in faster time to market. One of the major challenges of running top of the line deep models like Inception, Resnet, VGG in common edge/fog devices are the computational and memory requirements for each of the models. In our experiments, we have found that the Inception V3 model [28] can not be loaded into the available memory without allocating a USB based swap space in the Raspberry Pi 3 board and it takes nearly *five* seconds to classify a single image; and the issues are similar in most of the commonly used models. In this work, we propose a method to run deep inference operation of Convolutional Neural Networks (CNN) [16] on a set of fog devices for achieving high speed inferencing. CNNs are *de facto* techniques for image classification and have recently been used for speech and sensor data as well [13]. Though the concept of collaborative edge execution of CNN is introduced earlier by Mao *et al.* [20], our work extends the SoA through the following major contributions:1) a novel depth-wise input partitioning scheme that removes the overhead associated with earlier row/column and grid partitioning schemes, 2) a highlighted role of input and output depth of current convolutional layers (CLs) in the speedup achieved by distributed execution and 3) demonstration of its effect on distributed execution through extensive simulations with realistic workloads.We also prove our partitioning scheme on Inception V3 CLs on a real system based on Raspberry pi 3 and TensorFlow [8] to achieve 3 times speedup. The current paper is organized as follows: Section 2 gives a brief overview of current state of development in Edge Computing and

DCNN acceleration. Section 3 presents the methodology of implementing the proposed depth-wise partitioning scheme for DCNNs, Section 3.4 outlines the simulation and experimental results, and finally Section 4 concludes the paper.

2 RELATED WORK

2.1 Edge Computing: Current State

In [26], Satyanarayanan highlights how factors including low latency (through physical proximity), data reduction, scalability , privacy and security are driving the emergence of Edge Computing. The initial challenges in Fog and Edge Computing were mainly service standardization, integration of third party vendors, handling of confidential data (trust and privacy) and monetization. Those aspects are now getting standardized through initiatives like European Telecommunications Standards Institute (ETSI) [9] Multi-access Edge Computing (MEC) [10], OpenFog Consortium [12]. As a concrete example, section A.4 of *GS MEC-IEG 004* of ETSI MEC describes the detailed requirements and architecture of a massive sensor data pre-processing and data processing scenario in a Smart City. The OpenFog Consortium is bringing in the benefits of AI and 5G connectivity to the Fog computing ecosystem.

In the middleware level, systems like *ECHO* [24], are providing *end-to-end* hybrid *Edge-Fog-Cloud* middleware for dataflow management, resource discovery, runtime monitoring of services. In [29], open source softwares like TensorFlow, kubernetes and Docker are used for DL, Fog cluster management and resource virtualization, to build a Fog middleware.

2.2 DCNN acceleration and Distributed Execution

DL inference in constrained devices is an area of active interest since last few years. The approaches include layer compression and layerwise partitioning of DNNs [15, 18, 30], new methods including the use of FFT for reducing computation load [19], use of depth wise separable convolutions with smaller kernel sizes [4, 7]. Though focussed on FPGA, in [1] a comprehensive overview of several techhniques for CNN accelaration is provided. Partitioning of workload is a well employed technique that renders embedded systems responsive [6] and our focus for this work was to find effective mechanism for partitioning input data of the CLs of CNNs in a lossless fashion, that can augment any of these above mentioned techniques.

We have identified *two* earlier works that comprehensively treat distributed execution of convolution and CNN CLs. In [17], spatial convolution is distributed by partitioning the images among a set of parallel computers. Image is partitioned using a novel heuristic partitioning strategy that is better than row/column based partitioning and more powerful than the grid based partitioning. A detailed performance modeling is done to derive the speedup and experiments are done to validate the model. In [20], CNN is distributed into real mobile phones by image partitioning and model compression. A adaptive partitioning strategy is devised that partition along the shorter edge , resulting in lesser overhead pixels to be exchanged. The major contribution here is the identification that the mobile device shut down their transceivers when not required and hence there is a need to include this in the analysis of distribution speedup.

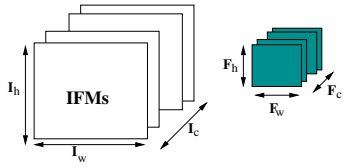


Figure 1: Input feature maps and one sample filter

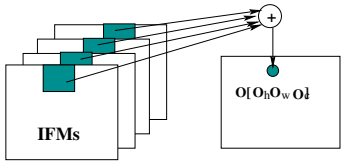


Figure 2: Convol. to generate one feature of an OFM

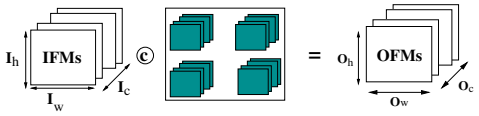


Figure 3: Each filter generate one OFM channel.

None of these prior works analyze the effect of handling high number input and output channels, which is the *de facto* standard in current SoA CNNs. Based on that we propose a non-loss depthwise image partitioning scheme that results in more speedup and renders such CNNs suitable for running in Fog/Edge grids.

3 NOVEL PARTITIONING MODEL

In this section we develop a theoretical model for distributed execution of the CLs used in DCNNs, especially considering the high number of input and output depth channels used in established CNN models.

3.1 Standard Convolutional Layers

A typical convolutional layer (CL) in CNNs operate on a set of 2D matrices to produce another set of 2D matrices.The matrices are often referred to as Feature Maps(FMs) and the number of FMs in input /output space are called channels or depth. The input FMs (IFMs) are convoluted with a set of filter kernels to generate the output FMs (OFMs).

Fig. 1 depicts a $I_h \times I_w \times I_c$ IFM and a single $F_h \times F_w \times F_c$ filter where *h, w, c* denotes height, width and depth channels. The dimensions of 2D filters are usually much smaller than the 2D IFMs. Each filter kernel has same depth as the input and generates *one OFM from all IFMs.*

Fig. 2 shows the calculation of a single feature value of OFM at a particular depth and Fig. 3 show calculation of all OFMs.

- o_h, o_w, o_c are given indices in height, width and depth dimensions of the OFMs, shaped $O_h \times O_w \times O_c$,
- f_h, f_w are the indices along height, width of the filter kernels and
- i_h, i_w, i_c are given indices in height, width and depth dimensions of the IFM.

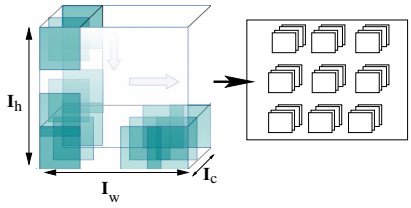


Figure 4: Implementation of the convol operation

```
Image patch d0  Conv 1 is:
[[1. 3. 9. 5.]  [[ 25.  25.  44.  29.]
 [6. 2. 1. 2.]   [ 78. 128. 127.  83.]
 [5. 6. 7. 4.]   [ 62.  90.  81.  46.]
 [1. 1. 5. 4.]]  [ 58.  89.  95.  65.]]
Image patch d1  Conv 2 is:
[[2. 4. 2. 6.]  [[ 55.  59.  91.  42.]
 [4. 9. 5. 7.]   [ 52. 102. 110.  65.]
 [3. 1. 6. 4.]   [ 83.  67. 146.  45.]
 [0. 3. 1. 9.]]  [ 12.  53.  50.  45.]]
filter d0 is:   Composite Conv is:
[[4. 5. 5.]     [[ 80.  84. 135.  71.]
 [1. 3. 0.]      [130. 230. 237. 148.]
 [1. 2. 5.]]     [145. 157. 227.  91.]
filter d1 is:    [ 70. 142. 145. 110.]]
[[2. 1. 6.]
 [2. 3. 1.]
 [4. 0. 5.]]
```

Figure 5: Sample 4x4 IFM and 2x2 Filter with depth 2

```
Kernel  Patches
[[4.]  [[[[0. 0. 0. 0. 0. 0. 0. 0. 1. 2. 3. 4. 0. 0. 6. 4. 2. 9.]
 [2.]     [0. 0. 0. 0. 0. 0. 0. 1. 2. 3. 4. 9. 2. 6. 4. 2. 9. 1. 5.]
 [5.]     [0. 0. 0. 0. 0. 0. 0. 3. 4. 9. 2. 5. 6. 2. 9. 1. 5. 2. 7.]
 [1.]     [0. 0. 0. 0. 0. 0. 0. 9. 2. 5. 6. 0. 0. 1. 5. 2. 7. 0. 0.]]
 [5.]
 [6.]    [[0. 0. 1. 2. 3. 4. 0. 0. 6. 4. 2. 9. 0. 0. 5. 3. 6. 1.]
 [1.]     [1. 2. 3. 4. 9. 2. 6. 4. 2. 9. 1. 5. 5. 3. 6. 1. 7. 6.]
 [2.]     [3. 4. 9. 2. 5. 6. 2. 9. 1. 5. 2. 7. 6. 1. 7. 6. 4. 4.]
 [3.]     [9. 2. 5. 6. 0. 0. 1. 5. 2. 7. 0. 0. 7. 6. 4. 4. 0. 0.]]
 [3.]
 [0.]    [[0. 0. 6. 4. 2. 9. 0. 0. 5. 3. 6. 1. 0. 0. 1. 0. 1. 3.]
 [1.]     [6. 4. 2. 9. 1. 5. 5. 3. 6. 1. 7. 6. 1. 0. 1. 3. 5. 1.]
 [1.]     [2. 9. 1. 5. 2. 7. 6. 1. 7. 6. 4. 4. 1. 3. 5. 1. 4. 9.]
 [4.]     [1. 5. 2. 7. 0. 0. 7. 6. 4. 4. 0. 0. 5. 1. 4. 9. 0. 0.]]
 [2.]
 [0.]    [[0. 0. 5. 3. 6. 1. 0. 0. 1. 0. 1. 3. 0. 0. 0. 0. 0. 0.]
 [5.]     [5. 3. 6. 1. 7. 6. 1. 0. 1. 3. 5. 1. 0. 0. 0. 0. 0. 0.]
 [5.]]    [6. 1. 7. 6. 4. 4. 1. 3. 5. 1. 4. 9. 0. 0. 0. 0. 0. 0.]
          [7. 6. 4. 4. 0. 0. 5. 1. 4. 9. 0. 0. 0. 0. 0. 0. 0. 0.]]]]
```

Figure 6: Sample 4x4 IFM and 2x2 Filter with depth 2

The formula for convolution calculation of a standard CL is given in Eq. 1:

$$OFM[o_h, o_w, o_c] = \sum_{i_c=1}^{I_c} \sum_{f_h=1}^{F_h} \sum_{f_w=1}^{F_w} F[f_h, f_w, i_c, o_c] \cdot IFM[o_h + f_h - 1, i_w + f_w - 1, i_c] \quad (1)$$

Eq. 1 captures a 3D CL operation where each filter is placed at a particular location of the IFM and filter contents are element-wise multiplied with the corresponding contents of the IFM, at a matching depth (2D spatial convolutions). The products are then added along the depth axis (linear projections). The filter is scanned through the whole IFM in both vertical and horizontal directions and from each location of the IFM, where filter is placed, an 3D IFM patch is extracted. This operation is shown in Fig. 4. The number of IFM patches extracted is exactly $O_h \times O_w$. These patches are flattened into an 1D array of dimension $F_h \times F_w \times F_c$, where elements are arranged from top left location of the patch, taken depth-wise, proceeding horizontally. Each 3D filter is flattened in the same way. Fig. 5 and Fig. 6 shows such flattened patches and filters.

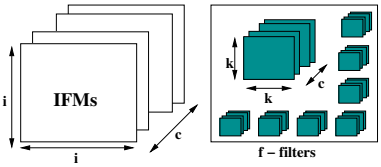


Figure 7: Input feature maps and one sample filter

These two arrays are now multiplied element-wise and summed up. The time complexity of CL operation can be specified as $O_h \cdot O_w \cdot F_h \cdot F_w \cdot F_c \cdot O_c$, for all the filters. Without loss of generality, we will proceed with the assumptions that 1) the IFMs have same spatial dimensions as the OFMs, 2) the IFMs are square shaped - transforming the time complexity above to $I^2 \cdot K^2 \cdot c \cdot f$, where I,K,c,f are input spatial dimensions, filter spatial dimensions, number of input channels and number output channels respectively. Fig. 7 portrays such labeling for ease of understanding and further calculation.

3.2 Partitioning Scheme

Earlier works use row/column and grid based spatial image partitioning strategies for distributed execution of convolutions. An associated problem in that scheme is sending some extra entries along with the partitioned image. This overhead is due to the *stride* of the filter movement where the filter overlaps with the adjacent IFM entries that fall in another spatial partition of IFM. In [17] this overhead is calculated as 7.8% for row based partitioning and 3.9% for grid based partitioning and 4.1% for heuristic partitioning proposed by the authors. In [20] an adaptive partitioning method is used with capacity based load assignment to worker nodes, where the partitioning is along the relatively larger dimension. In state of the art CNNs the *depth dimension* has become important for achieving high accuracy [27]. The prior works that look into distributed execution of image operations, like convolution on networked server and mobile grids, did not analyze the effect of handling high number input and output channels. Along with the increase in depth channels the kernel size is reduced and often kept around 3×3 in SoA CNNs, rendering the earlier analyses outdated. As an instance the table 1 presents few notable CLs in Inception V3 [28].

Table 1: Convolutional layers of Inception V3 showing

Conv#	InCh	OutCh	IDim	KDim
conv_1_1	32	32	149	3
conv_2_2	32	64	147	3
conv_4_4	80	192	73	3
mixed_3	288	384	35	3
mixed_8	768	192	17	1

In our work we split an image and a filter along depth dimension for distribution among the Fog resources. Moreover we also distribute all the filters to the resources. This results in a non-loss partitioning and is different in principal from depthwise separable convolutions popularized in [4, 7], where the filter kernel is decomposed into bases filters and computational efficiency is achieved.

3.3 Performance Modeling: Distributed CL

To model the performance of distributed execution of CLs, we define one computation step as per Eq. 2, where a 2D spatial convolution is performed for a particular input and an output channel. This is a subset of Eq. 1, where convolution is performed for all input channels of a particular output channel.

$$OFM_{Int1}[o_h, o_w, o_c] = \sum_{f_h=1}^{K} \sum_{f_w=1}^{F_w} F[f_h, f_w, i_c, o_c] \cdot I[o_h + f_h - 1, o_w + f_w - 1, i_c] \quad (2)$$

As we are doing input IFM partitioning on both in and out channel level, we need to define another basic computation step where the 2D spatial convolutions generated from Eq. 2 are summed up along the depth axis. Such a step is defined in Eq. 3:

$$OFM_{Int2}[o_h, o_w, o_c] = \sum_{i_c=1}^{I_c} OFM_{Int1}[o_h, o_w, i_c] \quad (3)$$

Let us consider that the computation steps in Eq. 2 and takes γ time to execute and *one* summation between *two* OFM_{Int1} layers, as defined in Eq. 3, takes λ time to execute. The time taken to execute CL operation in a standalone mode is given by Eq. 4:

$$T_{cls} = I^2 \cdot K^2 \cdot c \cdot f \cdot \gamma \quad (4)$$

For determining the data transfer time we use the model defined in [17]. Considering the setup time for channel establishment for a single MTU packet (M bytes), transfer as α and per byte transmission time as β, the time taken to transmit P bytes of data is given by:

$$T_{comm} = \frac{P}{M} \cdot \alpha + P \cdot \beta \quad (5)$$

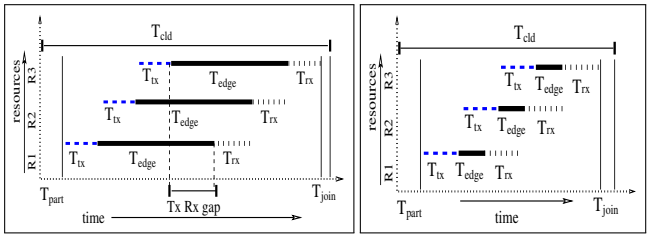
The distributed setup includes a master Edge device that partitions and distributes data (IFMs and Filters) and finally merges the partial solutions from the Edge workers. We partition the input data based on both input and output channels into n edge workers. We assume a homogeneous set of resources where the IFMs are equally distributed. This is a practicable assumption as lightweight virtual machines like containers [21], dockers are available for edge platforms [5] and can be extended to a capacity based partitioning [2] in future.

The distributed execution would require time to 1) create partitioned IFMs along the channels (T_{part}) at the master Edge, 2) send partitioned IFMs to the Edge workers (T_{tx}), 3) time to calculate the intermediate OFMs at all Edge worker T_{edge} , 4) time to send back intermediate OFMs to the master Edge (T_{rx}) and 5) merging the intermediate OFMs input channel-wise (Eq. 3) and concatenation of OFMs output channel-wise (T_{join}).We follow the distributed execution model used in [17], where two different scenarios may happen based on the communication and computation times.

As depicted in Fig. 8a, the time taken to execute CL operation in a distributed fashion is given by Eq. 6:

$$T_{cld} = T_{part} + n \cdot (T_{tx} + T_{rx}) + T_{edge} + T_{join} \quad (6)$$

When the number of Edge workers (N) increase and the computation time T_{edge} is comparatively small, the communication time



(a) Dominating computation **(b) Higher communication**

Figure 8: various distribution scenarios

determines the overall execution time of distributed CL and is specified as:

$$T'_{cld} = T_{part} + n \cdot (T_{tx} + T_{rx}) + T_{join} \quad (7)$$

Next we derive the three major contributors among these timings, analyzing their operations:

3.3.1 T_{tx}. Given the sizes of the IFMs and filters and the fact that we actually send $\frac{c}{n}$ input and $\frac{f}{n}$ output channels to each worker, Eq.1 5 can be rewritten as:

$$T_{tx} = \frac{I^2}{M} \cdot \frac{c}{n} \cdot \alpha + I^2 \cdot \frac{c}{n} \cdot \beta + \frac{K^2}{M} \cdot \frac{c}{n} \cdot \frac{f}{n} \cdot \beta + K^2 \cdot \frac{c}{n} \cdot \frac{f}{n} \cdot \beta \quad (8)$$

3.3.2 T_{edge}. The time to compute at each Edge is summation of the time to do 2D spatial convolution for selected input channels of an IFM with set of filters assigned to that edge.

$$T_{edge} = \frac{I^2 \cdot K^2}{M} \cdot \frac{c}{n} \cdot \frac{f}{n} \cdot \gamma + \frac{c}{n} \cdot \frac{f}{n} \cdot \lambda \quad (9)$$

3.3.3 T_{rx}. Each Edge worker sends back $\frac{f}{n}$ 2D OFMs:

$$T_{rx} = \frac{I^2}{M} \cdot \frac{f}{n} \cdot \alpha + I^2 \cdot \frac{f}{n} \cdot \beta \quad (10)$$

We ignore the partitioning and joining time as those are very small compared to the above terms.

Based on the above analysis and using Eq. 6, the total execution time of distributed CL (refer to Eq. 6 can be specified as:

$$T_{cld} = n \cdot (\frac{I^2}{M} \cdot \frac{c}{n} \cdot \alpha + I^2 \cdot \frac{c}{n} \cdot \beta + \frac{K^2}{M} \cdot \frac{c}{n} \cdot \frac{f}{n} \cdot \alpha + K^2 \cdot \frac{c}{n} \cdot \frac{f}{n} \cdot \beta) + \frac{I^2 \cdot K^2}{M} \cdot \frac{c}{n} \cdot \frac{f}{n} \cdot \gamma + \frac{c}{n} \cdot \frac{f}{n} \cdot \lambda + \frac{I^2}{M} \cdot \frac{f}{n} \cdot \alpha + I^2 \cdot \frac{f}{n} \cdot \beta \quad (11)$$

To derive the speedup S as $\frac{T_{cls}}{T_{cld}}$ we simplify Eq. 11 to: $\frac{K^2 cf\alpha}{n} \cdot (\frac{\alpha}{M} + \beta) + I^2 c \cdot (\frac{\alpha}{M} + \beta) + I^2 c \cdot (\frac{\alpha}{M} + \beta)$, ignoring the term denoting computation time when n gets large (refer to Eq. 7). This can be again simplified to $I^2 \cdot (c + f) \cdot (\frac{\alpha}{M} + \beta)$ as the kernel size used in state of the art CNNs are kept very small (around 3×3) to reduce computation requirements, rendering the terms with K^2 negligible compared to the IFMs. The maximum speedup achievable by adding more and more Edge workers for distributed CL execution is thus bounded by:

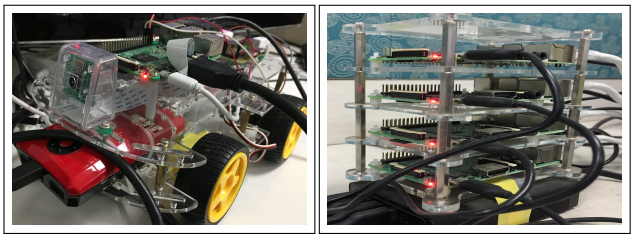
$$S = \frac{K^2 \cdot c \cdot f \cdot \gamma}{(c + f) \cdot (\frac{\alpha}{M} + \beta)} \quad (12)$$

From Eq. 12, it can be inferred that the speedup limit is not only dependent on the ratio of computation time to communication time, as shown in [17], but also on the input and output depth. The input and output depth has maximum effect on S when both are equal.

In the next section we validate our distribution scheme and the speedup achieved with real workloads of publicly available CNNs that demonstrated high accuracy in ILSVRC [25].

3.4 Results and Discussion

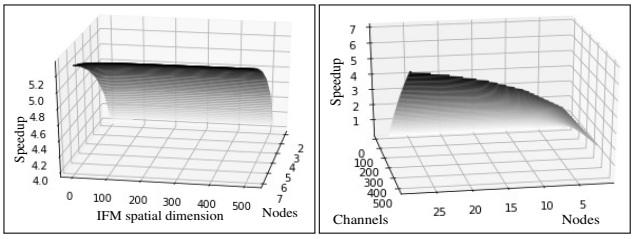
In the following model validation experiments we first gather the parameters γ, α, β etc. defined in section 3.3. We calculate the computation step (refer to Eq. 2) by measuring the full CL operation and then dividing by the operations (*input* × *kernel*×*in channels*). The computation time γ, estimated by us was around $0.001036\mu s$ The communication parameters were estimated by sending and receiving data between the Rpi boards over a WiFi connection (802.11 n, infrastructure). The estimated communication parameters are: $\alpha \simeq 300\mu s$, $\beta \simeq 0.412\mu s$ and $M \simeq 2304$ bytes. In future we would like to experiment with Wifi Direct for higher communication speed, as done in [20].



(a) RaspberryPi-3 master **(b) R-Pi-3 Fog cluster**

Figure 9: Experimental setup for parameter gathering

3.4.1 Distributed Execution. Fig. 10a shows that there is not much effect of the IFM spatial size on the speedup, however the number of depth channels has huge effect (Fig. 10b) on it. Fig. 10b also shows that for more that *ten* Edge workers, the speedup does not improve. This is in agreement with Eq. 12.



(a) Effect of Image Size **(b) Nature of Speedup**

Figure 10: Effect of general parameters

11a, 11b and 11c show the speedup achieved in the Inception V3 CLs as the number of channels increase, with no speedup when the depth is less than *60*. The speedup achieved is always driven by the depth, not by the IFM size (refer to table 1). Almost similar results are obtained while running VGG16 workloads as shown in 11d, 11e and 11f.

Based on the above findings we develop an algorithm that calculates the estimated speedup (ref Eq. 12), based on the then compute capacity of slave Edge devices, available network bandwidth and the input and output depth of the CL that is being executed. The partitioned execution of a CL is only triggered if the estimated speedup is above a threshold value. As an example, in our experimental setup, *we partition only if the number of channels exceed 60*. In a setup of Fog devices created using WiFi connected Rpi 3 boards, as shown in Fig. 9, we split CLs of Inception V3 by replacing the standard *conv_2d* TensorFlow functions into separate spatial *conv_2d* functions and merge the intermediate results as per the method given in section 3.3 to achieve the following preliminary results:

Table 2: Preliminary Results

Convolution#	Standalone time	Distributed time
conv_2_2	202.104 ms	97.4 ms
conv_4_4	293.079 ms	102.06 ms
conv_mixed3	183.872 ms	94.96 ms

4 CONCLUSIONS

Use of constrained fog/edge devices located at the edge of the network has become a reality for processing various forms of data closer to the source leading to smart inferencing in analytics related to several aspects of IoT and smart cities. Researchers are working on different schemes for utilizing such devices for useful computation. However, the constrained nature of these devices in terms of computational power and memory makes such usage restrictive. In order to overcome such shortfalls, we have proposed in this paper a novel depth-wise partitioning scheme of the inputs to popular CNN models that overcomes the difficulties of the current state of the art of row/column and grid partitioning schemes. Additionally, we highlight the role of input and output depth of current convolutional layers in the speedup achieved by distributed execution and prove our approach via extensive simulations with realistic workloads from the SoA CNN models. We expect that the data path optimization scheme presented here for a distributed computing hierarchy, will be equally applicable to parallel implementation of DCNN in FPGAs and other accelerators. In future we would like to work with the methods for automated partitioning of other DNN variants.

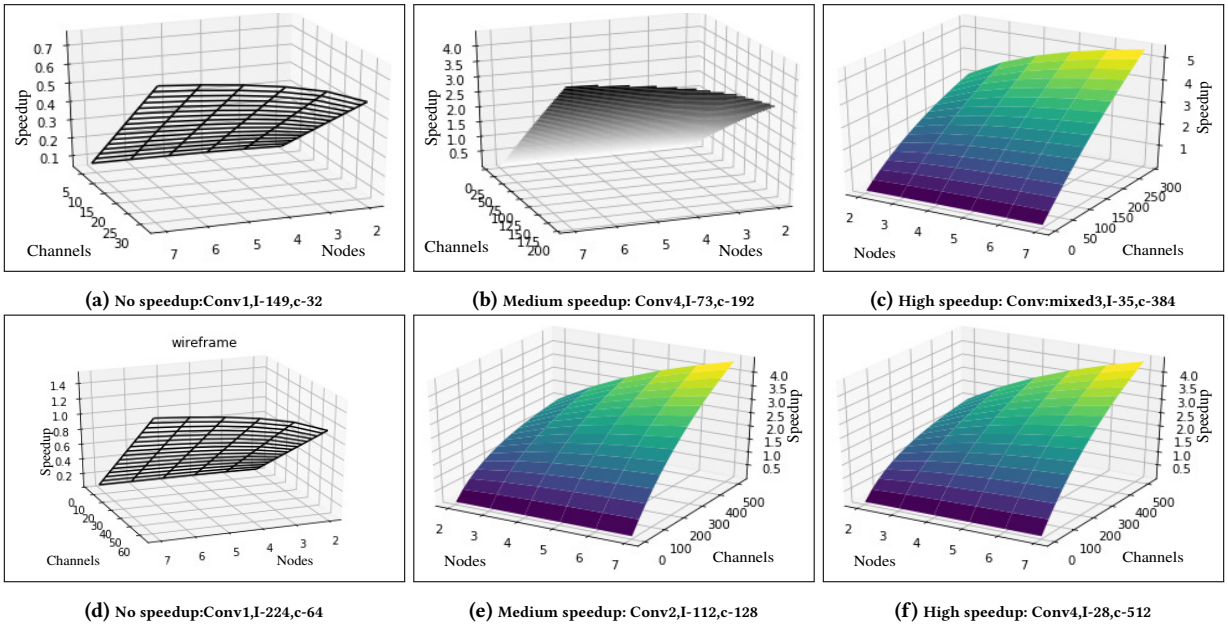
REFERENCES

[1] K. Abdelouahab, M. Pelcat, J. Serot, and F. Berry. 2018. Accelerating CNN inference on FPGAs: A Survey. *ArXiv e-prints* (May 2018). arXiv:cs.DC/1806.01683

[2] Ansuman Banerjee, Himadri Sekhar Paul, Arijit Mukherjee, Swarnava Dey, and Pubali Datta. 2014. A Framework for Speculative Scheduling and Device Selection for Task Execution on a Mobile Cloud. In *Adaptive Resource Management and Scheduling for Cloud Computing*. Springer International Publishing, Cham, 36–51.

[3] H. Begur, M. Dhawade, N. Gaur, P. Dureja, J. Gao, M. Mahmoud, J. Huang, S. Chen, and X. Ding. 2017. An edge-based smart mobile service system for illegal dumping detection and monitoring in San Jose. In *2017 IEEE SmartWorld, Ubiquitous Intelligence Computing, Advanced Trusted Computed, Scalable Computing Communications, Cloud Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. 1–6.

[4] François Chollet. 2016. Xception: Deep Learning with Depthwise Separable Convolutions. *CoRR* abs/1610.02357 (2016).



(a) No speedup:Conv1,I-149,c-32 (b) Medium speedup: Conv4,I-73,c-192 (c) High speedup: Conv:mixed3,I-35,c-384

(d) No speedup:Conv1,I-224,c-64 (e) Medium speedup: Conv2,I-112,c-128 (f) High speedup: Conv4,I-28,c-512

Figure 11: Inception V3 workloads: speedup

[5] desktopcontainers. 2017. Raspberry Pi Virtual Machine using QEMU inside Docker Container. Retrieved August 16, 2018 from https://store.docker.com/community/images/desktopcontainers/raspberrypi

[6] S. Dey and R. Dasgupta. 2009. Fast Boot User Experience Using Adaptive Storage Partitioning. In *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. 113–118. https://doi.org/10.1109/ComputationWorld.2009.121

[7] Andrew G. Howard et. al. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017).

[8] Martín Abadi et. al. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/ Software available from tensorflow.org.

[9] ETSI. 2018. ETSI - European Telecommunications Standards Institute. Retrieved August 16, 2018 from http://www.etsi.org

[10] ETSI. 2018. Multi-access Edge Computing. Retrieved August 16, 2018 from http://www.etsi.org/technologies-clusters/technologies/multi-access-edge-computing

[11] Aidin Ferdowsi, Ursula Challita, and Walid Saad. 2017. Deep Learning for Reliable Mobile Edge Analytics in Intelligent Transportation Systems. *CoRR* abs/1712.04135 (2017).

[12] OpenFog Consortium Architecture Working Group. 2018. Enabling advanced IoT,5G, AI with fog computing. Retrieved August 16, 2018 from www.openfogconsortium.org

[13] G. Keren and B. Schuller. 2016. Convolutional RNN: An enhanced model for extracting features from sequential data. In *2016 International Joint Conference on Neural Networks (IJCNN)*. 3412–3419.

[14] S. Khan, D. Paul, P. Momtahan, and M. Aloqaily. 2018. Artificial intelligence framework for smart city microgrids: State of the art, challenges, and opportunities. In *2018 Third International Conference on Fog and Mobile Edge Computing (FMEC)*. 283–288.

[15] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. DeepX: A Software Accelerator for Low-power Deep Learning Inference on Mobile Devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks (IPSN '16)*. IEEE Press, Piscataway, NJ, USA, Article 23, 12 pages. http://dl.acm.org/citation.cfm?id=2959355.2959378

[16] Yann LeCun and Yoshua Bengio. 1998. The Handbook of Brain Theory and Neural Networks. MIT Press, Cambridge, MA, USA, Chapter Convolutional Networks for Images, Speech, and Time Series, 255–258. http://dl.acm.org/citation.cfm?id=303568.303704

[17] Chi-kin Lee and Mounir Hamdi. 1995. Parallel Image Processing Applications on a Network of Workstations. *Parallel Comput.* 21, 1 (Jan. 1995), 137–160.

[18] En Li, Zhi Zhou, and Xu Chen. 2018. Edge Intelligence: On-Demand Deep Learning Model Co-Inference with Device-Edge Synergy. In *Proceedings of the 2018 Workshop on Mobile Edge Communications (MECOMM'18)*. ACM, New York, NY, USA, 31–36. https://doi.org/10.1145/3229556.3229562

[19] Sheng Lin, Ning Liu, Mahdi Nazemi, Hongjia Li, Caiwen Ding, Yanzhi Wang, and Massoud Pedram. 2018. FFT-based deep learning deployment in embedded systems. *2018 Design, Automation and Test in Europe Conference & Exhibition (DATE)* (2018), 1045–1050.

[20] J. Mao, X. Chen, K. W. Nixon, C. Krieger, and Y. Chen. 2017. MoDNN: Local distributed mobile computing system for Deep Neural Network. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*. 1396–1401.

[21] Dirk Merkel. 2014. Docker: Lightweight Linux Containers for Consistent Development and Deployment. *Linux J.* 2014, 239 (March 2014).

[22] T. Muhammed, R. Mehmood, A. Albeshri, and I. Katib. 2018. UbeHealth: A Personalized Ubiquitous Cloud and Edge-Enabled Networked Healthcare System for Smart Cities. *IEEE Access* 6 (2018), 32258–32285.

[23] Alex Adim Obinikpo and Burak Kantarci. 2017. Big Sensed Data Meets Deep Learning for Smarter Health Care in Smart Cities. *Journal of Sensor and Actuator Networks* 6, 4 (2017).

[24] Pushkara *et al.* Ravindra. 2017. ECHO: An Adaptive Orchestration Platform for Hybrid Dataflows across Cloud and Edge. In *Service-Oriented Computing*. Springer International Publishing, Cham, 395–410.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *Int. J. Comput. Vision* 115, 3 (Dec. 2015), 211–252. https://doi.org/10.1007/s11263-015-0816-y

[26] Mahadev Satyanarayanan. 2017. The Emergence of Edge Computing. *Computer* 50, 1 (Jan. 2017), 30–39.

[27] K. Simonyan and A. Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

[28] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. 2016. Rethinking the Inception Architecture for Computer Vision. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2818–2826.

[29] Pei-Hsuan Tsai, Hua-Jun Hong, An-Chieh Cheng, and Cheng-Hsin Hsu. 2017. Distributed analytics in fog computing platforms using tensorflow and kubernetes. *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)* (2017), 145–150.

[30] Mengwei Xu, Feng Qian, and Saumay Pushp. 2017. Enabling Cooperative Inference of Deep Learning on Wearables and Smartphones. *CoRR* abs/1712.03073 (2017). arXiv:1712.03073 http://arxiv.org/abs/1712.03073