2021 IEEE 14th International Conference on Cloud Computing (CLOUD)

The Case for Adaptive Deep Neural Networks in Edge Computing

Francis McNamee*, Schahram Dustdar‡, Peter Kilpatrick*,
Weisong Shi§, Ivor Spence*, and Blesson Varghese*
*Queen's University Belfast, UK
‡TU Wien, Austria
§Wayne State University, USA
E-mail: fmcnamee04@qub.ac.uk, dustdar@infosys.tuwien.ac.at, p.kilpatrick@qub.ac.uk,
weisong@wayne.edu, i.spence@qub.ac.uk, b.varghese@qub.ac.uk (Corresponding e-mail)

***Abstract*—Deep Neural Networks (DNNs) are an application class that benefit from being distributed across the edge and cloud. A DNN is partitioned such that specific layers of the DNN are deployed onto the edge and the cloud to meet performance and privacy objectives. However, there is limited understanding of: whether and how evolving operational conditions (increased CPU and memory utilization at the edge or reduced data transfer rates between the edge and cloud) affect the performance of already deployed DNNs, and whether a new partition configuration is required to maximize performance. A DNN that adapts to changing operational conditions is referred to as an 'adaptive DNN'. This paper investigates whether there is a case for adaptive DNNs by considering four questions: (i) Are DNNs sensitive to operational conditions? (ii) How sensitive are DNNs to operational conditions? (iii) Do individual or a combination of operational conditions equally affect DNNs? (iv) Is DNN partitioning sensitive to hardware architectures? The exploration is carried out in the context of 8 pre-trained DNN models and the results presented are from analyzing nearly 8 million data points. The results highlight that network conditions affect DNN performance more than CPU or memory related operational conditions. Repartitioning is noted to provide a performance gain in a number of cases, but a specific trend is not noted in relation to the underlying hardware architecture. Nonetheless, the need for adaptive DNNs is confirmed.**

I. Introduction

Edge computing envisions that compute resources located or placed at the edge of the network, such as routers, gateways or dedicated micro data centers, may be used for running certain application services closer to the end-user device where data is generated [1]–[4]. Processing data at the edge provides opportunities for making applications more responsive by reducing end-to-end latencies, performance efficient by reducing ingress bandwidth demand beyond the edge resource, and privacy-sensitive by selectively releasing data beyond the edge.

Many performance-critical and privacy-sensitive applications are demonstrated to benefit from edge computing - for example, cognitive wearable assistance [5], image and video analytics [6], connected and autonomous vehicles [7] and privacy preserving denaturing [8]. These applications take advantage of the edge by distributing services of the application across the edge and the cloud.

One reason the above applications lend themselves to take advantage of the edge is because they are underpinned by Deep Neural Networks (DNNs). A DNN is a sequence of multiple layers (each layer is a collection of neurons) that carry out functions, such as convolution, pooling or activation [9], [10]. Therefore, the layers of a DNN can be distributed in a specific manner across the edge and the cloud to reduce inference times, reduce the volume of data transferred to the cloud from a sensor or end-device, or to not release sensitive data beyond the edge [11], [12]. This is achieved by DNN partitioning - splitting the DNN into two sequential DNNs at a specific layer (the layer at which the DNN is partitioned is referred to as the partitioning point). The partitions can then be distributed across the cloud and the edge.

DNN partitioning for performance efficiency is an avenue that has been reported in edge computing literature. There are multiple techniques for partitioning DNNs, such as using estimation-based [6], [13], [14], structural modification-based [15], [16], and measurement-based techniques [17]. These techniques identify an optimal partitioning point based on the characteristics of the layers of a DNN and operational conditions, such as resource utilization or network conditions. It is generally understood that distributed DNN execution has specific performance advantages.

However, there is limited understanding of whether and how evolving operational conditions at the edge and hardware architectures affect the performance of the already deployed DNNs and raises an important question on whether a new partitioning configuration is required. If new system conditions affect performance, then the DNN will need to be repartitioned to maximize its performance under the new operational conditions or to suit new architectures. Such a DNN that adapts to operational conditions is referred to in this paper as an 'adaptive DNN' (the process is referred to as 'adaptivity').

Therefore, this paper sets out to investigate *whether there is a case for adaptive DNNs in edge computing*. In doing so, the following four questions are considered:

(**Q1**) *Are DNNs sensitive to operational conditions?* This question addresses whether operational conditions, such as CPU and memory stress on edge resources or the network data transfer rate between the cloud and the edge, affect the performance of a distributed DNN.

(**Q2**) *How sensitive are DNNs to operational conditions?* If

2159-6190/21/$31.00 ©2021 IEEE
DOI 10.1109/CLOUD53861.2021.00017

Q1 is true, then the aim is to answer a second question - how different would the partition configurations and performance be given a change in the operational environment?

(**Q3**) *Do individual or a combination of operational conditions equally affect DNNs?* In this paper, the operational conditions are explored individually (only varying a single operational condition and by not explicitly influencing the other conditions) and in combination (multiple operational conditions are varied) to identify their sensitivity for DNN performance. For example, it is important to understand whether the partitioning point of a DNN changes when operational conditions change individually or in combination.

(**Q4**) *Is DNN partitioning sensitive to hardware architectures?* In this paper, multiple hardware architectures, namely Intel and Arm processors are employed across the cloud and edge. Observations from these are valuable when deploying distributed DNNs in an architecture rich and heterogeneous distributed computing environment.

The questions raised above have not been considered in the existing literature but it is essential that they are understood within the context of edge computing to further explore adaptive DNNs and maximally leverage the benefits of using the edge. This paper presents a first such exploratory study to address the above questions by developing a practical methodology to benchmark DNNs across cloud-edge resources. Experimental studies are carried out on 8 DNNs by examining different operational conditions for CPU stress, memory stress and network data transfer rates.

The results presented are obtained from a cloud and edge lab-based experimental platform by analyzing nearly 8 million data points. The key observation is that DNN performance is sensitive to operational conditions, both individual and in combination. Network conditions have more impact on DNN performance than CPU stress and memory stress individually. Repartitioning can provide performance gains, but there are DNNs that are not significantly impacted. Operational conditions in combination affect DNN performance more than individually. Thus, there is a case for repartitioning, i.e., the need for adaptive DNNs.

The rest of this paper is organized as follows. Section II provides a background to DNN (re)partitioning. Section III presents the methodology used in this paper. Section IV provides a discussion on the results.Section V presents related research. Section VI concludes this paper.

II. Background

A DNN consists of an input layer, multiple hidden layers, and an output layer (each layer is a collection of neurons) [9], [10]. There are different types of DNN layers, which include: (1) Fully-connected layers, (2) Convolution layers, (3) Pooling layers, (4) Activation layers, and (5) Softmax layers.

Eight DNNs are considered and are shown in Table I. The size of a trained model and its corresponding weights, the number of DNN layers, the number of valid partitioning points, and the type of the DNN is shown. These models are obtained from the Keras library (https://keras.io).

TABLE I: Pre-trained DNN models available from Keras that is used in this paper; Type: S - sequential, N - non-sequential

DNN Model	Size (MB)	Layers	Partition points	Type
VGG16 [18]	527	23	22	S
VGG19 [18]	548	26	25	S
MobileNet [19]	16	93	92	S
AlexNet [20]	110	25	24	S
DenseNet [21]	31	429	22	N
ResNet50 [22]	98	177	23	N
ResNet50V2 [22]	98	192	16	N
LeNet [23]	7	11	10	S

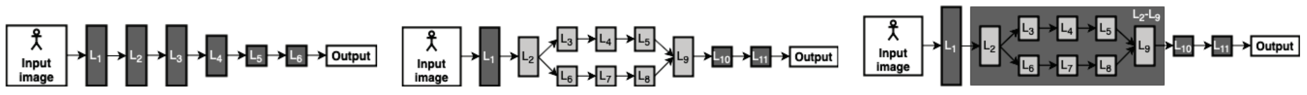
There are two types of DNNs - sequential and non-sequential, represented as S and N, respectively in Table I. In a sequential DNN the input of one layer is connected to the next in a linear manner (Figure 1a shows an example with six layers). A non-sequential DNN on the other hand will have layers that may be connected to two or more layers (refer Figure 1b for an example with 11 layers). Hence, there are multiple paths that connect the first and last layer of the DNN.

Distributing a sequential DNN across the cloud and the edge is straightforward. The DNN can be partitioned at a suitable layer that would yield maximum performance (for example, lowest end-to-end latency and/or least amount of data transferred from the edge to the cloud).

Distributing a non-sequential DNN requires additional pre-processing. This is to ensure that a parallel path in a DNN is not partitioned as it may lead to synchronization issues that will incur communication overheads [6]. Partitioning is avoided on the parallel paths by grouping parallel layers as a single entity, referred to as a block of layers (refer to Figure 1b and Figure 1c for an example - Layers 2-9 are treated as a single entity). Therefore, the number of partitioning points is reduced. For example, in Table I ResNet50V2 has 192 layers, but with only 16 suitable partitioning points.

The layer at which a DNN needs to be partitioned may depend on the characteristics of the DNN. For example, the layer at which the DNN is partitioned may be based on creating partitions that will result in the lowest execution time and the least volume of data that will be transferred between the edge and the cloud. Such a partitioning approach based on the DNN characteristics will create ideal DNN partitions.

However, would an ideal partition be the most suitable for a given set of operational conditions, such as utilization of edge resources or network state between the cloud and edge? In addition, if an ideal partition were deployed and the operational conditions changed, would a more context-driven partition improve the overall performance of the DNN? For example, the edge may execute multiple workloads, resulting in increasing CPU utilization, which may affect the edge partition running on the network. Alternatively, the network between the edge and the cloud may be congested resulting in sub-optimal performance due to communication overheads. In such cases, the DNN performance may be sensitive to the operational conditions. As already highlighted in the previous section, it is currently not fully understood how sensitive DNNs are to operational conditions.



(a) Sequential DNN (b) Non-sequential DNN (c) Non-sequential DNN after pre-processing

Fig. 1: Sequential and non-sequential DNNs

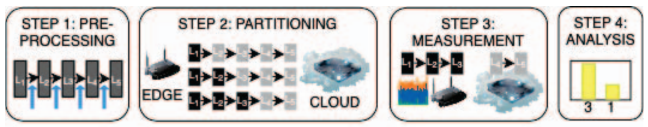


Fig. 2: Exhaustive benchmarking method adopted for obtaining data on performance partitioning points of DNNs

III. Methodology

The aim of this paper is to carry out exploratory research to address the questions raised in Section I. Therefore a methodology for measuring and benchmarking the DNN partitions to collect data relevant to the individual layers/blocks under varying operational conditions is required. This section, presents the methodology adopted and the practical technique used for measuring and benchmarking as shown in Figure 2.

The methodology adopted for obtaining data from the DNNs shown in Table I has the following four steps:

Step 1 - Pre-process the DNN and identify suitable partitioning points: As discussed in the previous section, a DNN may be a sequential or non-sequential DNN. All layers of a sequential DNN can be partitioned. However, a non-sequential DNN may have parallel paths and therefore there will be fewer suitable partitioning points. In this step, the DNN is pre-processed to identify the valid partitioning points of the DNN. As shown in Table I non-sequential DNNs have fewer partitioning points than the number of layers; a block of layers will need to be treated as a single entity to avoid synchronization issues when partitioning parallel paths [6].

Step 2 - Partition the DNN across all suitable partitioning points: The DNN is then partitioned across all the above identified partitioning points to ensure that all possible combinations of partitions are available for benchmarking.

Step 3 - Measure the performance on the edge and cloud resource for varying operational conditions: The partitioned DNNs are executed on the edge and cloud resource to measure the end-to-end latencies of discrete individual layers and/or a block of layers of the DNN. For example, consider a sequential DNN with five layers. Then the DNN will be benchmarked for the following 4 combination of partitions: Layer 1 on the edge and Layer 2-5 on the cloud, Layer 1-2 on the edge and Layer 3-5 on the cloud, Layer 1-3 on the edge and Layer 4-5 on the cloud, and Layer 1-4 on the edge and Layer 5 on the cloud. The most performance efficient partition will have the lowest end-to-end latency (which is the sum of the compute time of the partitions on the edge and cloud and the communication time of data from the edge to the cloud). The average of 10 executions of the combination of partitions is noted.

For CPU and memory stress on the edge, explicit stress is created on the resource using the Linux tool, namely `stress`, and the network data transfer rate is controlled using the Linux traffic control tool, namely `tc`. The network data transfer rates are based on different speeds observed in the wide-area network - 50Mb/s is a fast connection available to small businesses in the UK, 25Mb/s is equivalent to the average UK household data download speed, and 10Mb/s is equivalent to the speeds of a more busy network.

The measurement step is time consuming in that 1000 executions need to be considered ($5\times5\times4\times10$). One execution of the different DNNs required between 1-7 minutes depending on the depth of the neural network. Between 1-5 whole days were required for recording the measurements of individual DNNs on each experimental platform considered in the paper.

Step 4 - Analyze recorded measurements to identify performance efficient partitions for different scenarios: The measurements obtained from benchmarking are used to determine the most performance efficient partitions. The data is aggregated across multiple dimensions and presented in the next section.

IV. Experimental Studies

In this section, the experimental platform used for pursuing studies to identify scenarios when a DNN is sensitive to adaptivity and the results obtained from the study is presented.

A. Experimental Platform

The experiments were carried out on four different platforms comprising a cloud and edge processor as shown in Table II.

The benchmarking methodology is implemented in Python and requires Tensorflow 1.5+. Tensorflow [24] is used to execute the pre-trained DNNs (Table I) provided by Keras. NumPy is used for processing multi-dimensional arrays.

The different levels of: (i) CPU stress considered are 0%, 22%, 45%, 67%, and 90%, (ii) memory stress considered are 0%, 22%, 45%, 67%, and 90%, and (iii) network data transfer rates considered are 10Mb/s, 25Mb/s, 37.5Mb/s and 50Mb/s. However, some of the experiments considered will only present results for fewer stress values and data transfer rates for a meaningful representation of the results.

B. Results

Performance efficiency is measured as the lowest end-to-end latency when the DNN partitions are executed across the cloud-edge for a single test image of 150KB size. The end-to-end latency values are the average of ten experimental runs.

Experimental results are presented to highlight that:

(i) DNNs are sensitive to operational conditions (to address Q1 posed in Section I). For this the percentage of performance

TABLE II: The four experimental platforms comprising a cloud and edge processor employed in the research

Platform	OS	Cloud				Edge			
		Processor	Cores	Frequency	Memory	Processor	Cores	Frequency	Memory
P_1	Ubuntu 18.02 LTS	Intel Xeon E5	4	2.3 GHz	8 GB	ARM Cortex-A72	1	2.3 GHz	2 GB
P_2	Ubuntu 18.02 LTS	ARM Cortex-A72	8	2.3 GHz	16 GB	ARM Cortex-A72	1	2.3 GHz	2 GB
P_3	Ubuntu 18.02 LTS	ARM Cortex-A72	8	2.3 GHz	16 GB	Intel Xeon E5	2	2.3 GHz	2 GB
P_4	Ubuntu 18.02 LTS	Intel Xeon E5	4	2.3 GHz	8 GB	Intel Xeon E5	2	2.3 GHz	2 GB

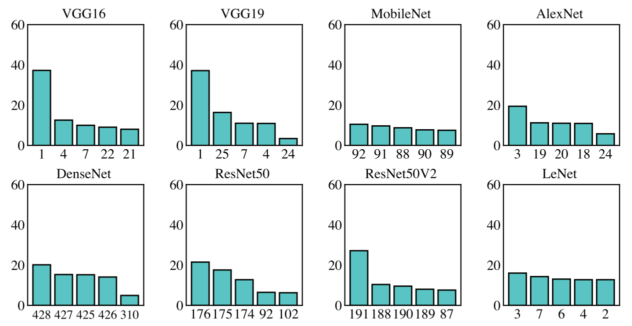


Fig. 3: Percentage of performance efficient partitioning points for DNNs across all combinations of CPU stress, memory stress and data transfer rate on all experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

efficient partitioning points for DNNs across all combinations of CPU stress, memory stress and network data transfer rates is presented. It is noted that all eight DNNs considered have scope for adaptivity across all potential combinations of CPU stress, memory stress and data transfer rates.

(ii) A performance gain is observed when repartitioning under different operational conditions (to address Q2 from Section I). This is explored in the context of individual operational conditions. It is noted that although there are performance gains in a number of cases, the overheads in repartitioning and deployment may offset the gain (may depend on the input not considered in this paper).

(iii) Both individual and a combination of operational conditions affect DNNs (to address Q3). Network conditions affect DNN performance more directly than CPU and memory stress individually (network conditions are an important consideration for performance efficiency in connected and autonomous vehicles [7]). There is more impact on DNN performance when a combination of operational conditions are considered.

(iv) DNN partitioning is sensitive to the hardware architecture employed on the cloud and the edge (to address Q4). The effect of all and individual operational conditions on each experimental platform is considered. The levels of adaptivity observed vary depending on the network and experimental platform. This highlights the importance of taking the platform into account for partitioning, although a specific correlation between the hardware employed on the cloud/edge and its influence on the partitioning point is not noted.

1) General Observation on DNN Sensitivity to Operational Conditions: Figure 3 shows the percentage of the top 5 performance efficient partitioning points for all combinations of operational conditions considered – CPU stress, memory stress and network data transfer rates on all experimental platforms P_1, P_2, P_3 and P_4. The x-axis shows the partitioning point (the layer after which the DNN is partitioned; for example, a number 81 corresponds to the first partition having layers 1-81 on the edge, and the remaining layers from Layer 82 will be a second partition that is executed on the cloud).

Consider the partition points for VGG16. Regardless of the operational conditions three partitioning points (Layer 1, 4 and 7) result in over 95% of the most performance efficient partitions. In most cases for VGG16, it is highly probable that there will be three options for partitioning. For instance, if the current partition is at Layer 1, then either that is still the most optimal partition, or performance can be optimized by repartitioning with Layer 4 or 7 as the partitioning point.

Similarly for VGG19 there are three main partitioning points, namely Layer 1, 25 and 7. However, nearly 10% of operational conditions have Layer 4 as the optimal partition point. Take ResNet50 as an example, then the partitioning points that result in efficient partitions are nearly 50% of the time at Layer 176 or 175. However, for another 20% of the cases the partitioning points are at Layer 174 and 92. This is just one evidence of the variation in the partitioning points if the DNN adapts to the changing operational condition.

2) General Observation on DNN Adaptivity and Hardware Architectures: Figure 4 to Figure 7 shows the percentage of the top 5 performance efficient partitioning points for all combinations of operational conditions considered – CPU stress, memory stress and network data transfer rates on the experimental platforms P_1, P_2, P_3 and P_4 respectively.

To demonstrate the sensitivity to edge hardware architectures, we compare the results from platforms P_1 and P_4 (same Intel processor on the cloud, but different edge processors; Figure 4 and 7) and from P_2 and P_3 (same ARM processor on the cloud, but different edge processors; Figure 5 and 6).

Consider AlexNet and ResNet50 on P_1 and P_4. For both DNNs the dominant partitioning points on the Arm edge processor are early in the network (Layer 3 for AlexNet and Layer 1 for ResNet50), but on the Intel edge processor the dominant partitioning points are further down in the network. Similarly, consider VGG16 and VGG19 on P_2 and P_3. For both DNNs the dominant partitioning points on the Arm edge processor are earlier in the network, where as on the Intel edge processor appears further down in the network.

The results suggest that there is a limited influence of the cloud processor on the partitioning point at the edge.

3) DNN Adaptivity and Individual Operational Conditions: The effect of CPU stress, memory stress and network data transfer rate individually on DNN adaptivity is considered. The results presented for varying individual operational conditions do not additionally stress other conditions. For example, the experiments for identifying the sensitivity of CPU stress to

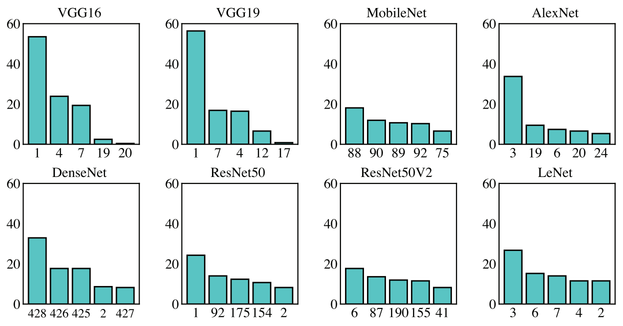


Fig. 4: Percentage of performance efficient partitioning points for DNNs across all combinations of CPU stress, memory stress and data transfer rate on experimental platform P_1. X-axis shows the partitioning point.

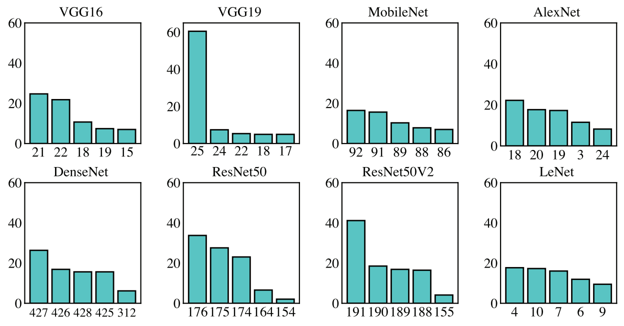


Fig. 6: Percentage of performance efficient partitioning points for DNNs across all combinations of CPU stress, memory stress and data transfer rate on experimental platform P_3. X-axis shows the partitioning point.

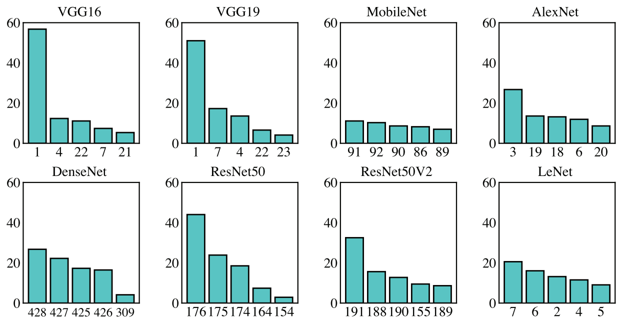


Fig. 5: Percentage of performance efficient partitioning points for DNNs across all combinations of CPU stress, memory stress and data transfer rate on experimental platform P_2. X-axis shows the partitioning point.

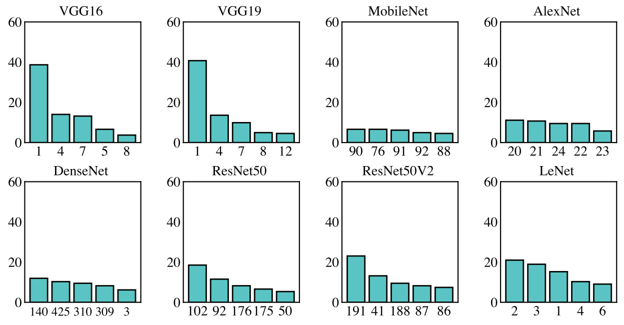


Fig. 7: Percentage of performance efficient partitioning points for DNNs across all combinations of CPU stress, memory stress and data transfer rate on experimental platform P_4. X-axis shows the partitioning point.

DNN adaptivity has no additional memory stress applied and has the maximum network data transfer rate.

Two comparisons are relevant. Firstly, comparing DNN sensitivity to CPU stress, memory stress and network data transfer rates individually against Figure 3. This will highlight the effect of individual operational conditions on DNN partitioning points against all combinations of operational conditions. Secondly, comparing the graphs on the impact of DNN partitioning points on CPU stress, memory stress and network data transfer rates against each other. This will highlight the sensitivity of DNNs to specific operational conditions.

Sensitivity to CPU stress: Figure 8 shows the percentage of the top performance efficient partitioning points for different values of CPU stress on the edge, when there is no additional memory stress and the network data transfer rate is 50Mb/s. Comparing with Figure 3, it is noted that AlexNet has only 3 optimal partitioning points with Layer 3 the most frequent and VGG19 has only 4 optimal partitioning points. There are fewer optimal partitioning points for these DNNs for different CPU stress; when using the Intel processor on the edge, the number of partitioning points increased for these networks (results not shown since the graphs are exhaustive). The other partitioning points, for example layers 150 and 149 of AlexNet in Figure 8 are sub-optimal partitioning points in decreasing order. For VGG19 it is noted that the partitioning point that is most prominent across the search space (Layer 1; Figure 3) is the optimal partition point under maximum CPU stress (most layers should be on the cloud) and remains the optimal partitioning point, regardless of the underlying hardware architecture. DenseNet demonstrates the impact of CPU stress on optimal partitioning points effectively. A number of layers are sensitive to CPU stress, but are less sensitive to memory stress and network data transfer rates.

Sensitivity to memory stress: Figure 9 shows the percentage of the top performance efficient partitioning points for different values of memory stress on the edge, when no additional CPU stress is applied and the network data transfer rate is 50Mb/s. Similar to the sensitivity to CPU stress, VGG16 and VGG19 show little sensitivity, with layer 1 being the most frequent optimal partition point. AlexNet shows the same optimal partitioning points as in Figure 8. VGG16 and ResNet50 have a similar profile as seen in Figure 8. Both CPU stress and memory stress have a similar diversity of partitioning points.

Sensitivity to network data transfer rate: Figure 10 shows the percentage of the top performance efficient partitioning points for different network data transfer rates when no additional CPU or memory stress is applied on the edge. Two layers, Layer 1 and Layer 7 are prominent partitioning points

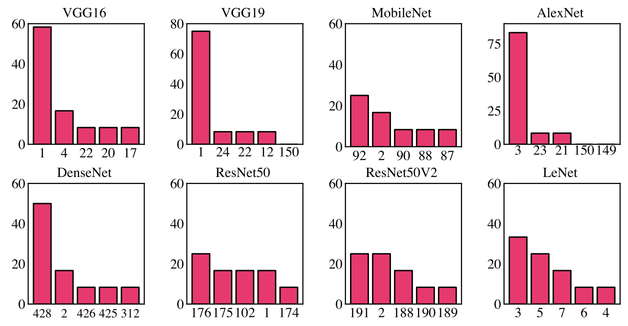


Fig. 8: Percentage of performance efficient partitioning points for the DNNs for different CPU stress when no additional memory stress is applied on the edge and the network data transfer rate is 50Mb/s on all experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

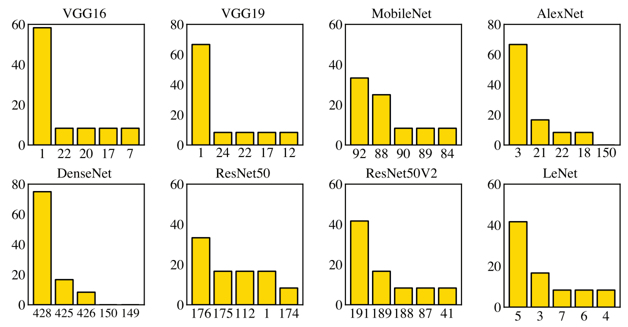


Fig. 9: Percentage of performance efficient partitioning points for the DNNs for different memory stress when no additional CPU stress is applied on the edge and the network data transfer rate is 50Mb/s on all experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

for VGG16. In the case of AlexNet, Layer 23 appears as an optimal partitioning point although it does not appear as a top five across all combinations of the operational conditions. The optimal partitioning point Layer 428 in Figure 3 becomes even more prominent for DenseNet in Figure 10.

The comparison of the DNNs among the three individual operational conditions on all experimental platforms is summarized in Table III. When CPU stress is considered MobileNet, ResNet50, ResNet50V2 and DenseNet are adaptive on all experimental platforms. Similarly, for memory stress MobileNet, ResNet50, ResNet50V2 and LeNet are adaptive to all experimental platforms and for network data transfer ResNet50V2, ResNet50 and MobileNet are adaptive to all experimental platforms. All networks show sensitivity to individual operational conditions across all experimental platforms, however the VGG networks usually perform best when layer 1 is chosen as the partition point. The choice of processor on the edge affects the optimal partitioning points for all the networks, with the optimal partitioning point moving deeper into the network when an Intel architecture is used. This highlights that the underlying hardware influences the partitioning point in response to individual operational

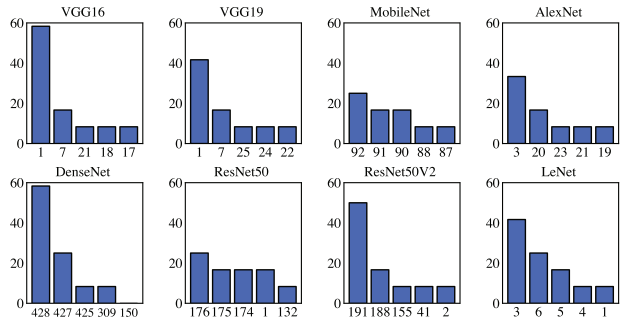


Fig. 10: Percentage of performance efficient partitioning points for the DNNs for different network data transfer rates when no additional CPU or memory stress is applied on the edge on all experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

conditions. However, specific patterns of the sensitivity to hardware architectures cannot be obtained.

Performance Gain: Table IV shows the performance gain of repartitioning DNNs for different CPU stress levels on experimental platform P_1 (similar results obtained from other platforms but not presented). The table shows the end-to-end latency (in seconds) and the partitioning layer for 0%, 45% and 90% CPU stress at the edge. The partitioning layer is the same for 45% and 90% as it is for when there is no CPU stress. The general trend is that the end-to-end latency increases with increasing CPU stress. The table then shows the end-to-end latency of the best DNN partition at a different partition layer for 45% and 90% CPU stress (shown in the table as best partition). The performance gain of the best partition over using static partitioning (best partitioning point when CPU stress is 0%) is indicated in the table.

Consider DenseNet as an example in Table IV. The performance gain is immediately evident. If the original partition is used (at Layer 428; optimal partition when CPU stress is 0%), then the end-to-end latency of this distributed DNN would be 2.254 seconds when the CPU stress is 90%. However, repartitioning at Layer 2, results in a DNN with 1.296 seconds end-to-end latency (42.50% performance gain). A smaller gain of 17.89% is noted when the DNN is repartitioned at Layer 2 if there is 45% CPU stress. The performance gain is an indicator of the benefit of repartitioning.

Table V shows the performance gain of repartitioning DNNs for different memory stress levels. The end-to-end latency and the partitioning layer for 0%, 45% and 90% memory stress at the edge is shown. The partitioning layer is the same for 45% and 90% as it is for when there is no memory stress. The general trend is that the end-to-end latency increases with more memory stress. The table then shows the end-to-end latency of the best DNN partition at a different partition layer for 45% and 90% memory stress (shown in table as best partition). The performance gain in using the best partition layer over a static partitioning layer (best partitioning point when memory stress is 0%) is indicated. A noteworthy gain from repartitioning is

TABLE III: DNN sensitivity to individual operational conditions on experimental platforms, P_1–P_4; Y - Yes, N - No

DNN Model	CPU Stress				Memory Stress				Network Data Transfer			
	P_1	P_2	P_3	P_4	P_1	P_2	P_3	P_4	P_1	P_2	P_3	P_4
ResNet50V2	N	Y	Y	Y	Y	Y	N	Y	Y	Y	Y	Y
MobileNet	Y	Y	Y	Y	N	N	Y	Y	Y	Y	Y	Y
ResNet50	Y	Y	Y	Y	Y	N	Y	Y	Y	N	Y	Y
VGG16	Y	N	Y	N	Y	N	Y	Y	Y	N	Y	Y
VGG19	Y	N	Y	Y	Y	N	Y	Y	Y	N	Y	Y
DenseNet	Y	Y	Y	Y	N	Y	Y	Y	N	Y	Y	Y
AlexNet	N	N	N	Y	N	N	Y	Y	Y	Y	Y	Y
LeNet	N	Y	Y	Y	Y	Y	Y	Y	N	Y	Y	Y

TABLE IV: Effect of CPU stress on end-to-end latency (seconds) on experimental platform P_1; partitioning layer is shown in brackets beside the end-to-end latency values. End-to-end latency is shown for 0%, 45% and 90% CPU stress when partitioning layer is the same as for CPU stress is 0%. End-to-end latency for the best DNN partition when CPU stress is 45% and 90% along with the partitioning layer is shown. The performance gain of best partition is shown. AlexNet, ResNet50V2 and LeNet are not included as they are not sensitive to CPU stress (refer to Table III).

DNN Model	0% (best partition)	45%	45% (best partition)	Gain (%)	90%	90% (best partition)	Gain (%)
VGG16	1.048 (1)	3.236 (1)	2.022 (4)	37.51	3.260 (1)	2.349 (4)	27.94
VGG19	1.261 (1)	3.395 (1)	2.417 (4)	28.81	3.468 (1)	2.564 (1)	26.10
MobileNet	0.280 (88)	0.524 (88)	0.291 (83)	44.47	0.554 (88)	0.370 (78)	33.21
DenseNet	1.179 (428)	1.554 (428)	1.276 (2)	17.89	2.254 (428)	1.296 (2)	42.50
ResNet50	0.952 (112)	1.046 (112)	0.960 (1)	8.22	1.827 (112)	0.953 (5)	47.84

TABLE V: Effect of memory stress on end-to-end latency (seconds) on experimental platform P_1; partitioning layer is shown in brackets beside the end-to-end latency values. End-to-end latency is shown for 0%, 45% and 90% memory stress for the partitioning layer when memory stress is 0%. End-to-end latency for the best DNN partition when memory stress is 45% and 90% along with the partitioning layer is shown. The performance gain of the best partition is shown. MobileNet, AlexNet and DenseNet are not included as they are not sensitive to memory stress (refer to Table III).

DNN Model	0% (best partition)	45%	45% (best partition)	Gain (%)	90%	90% (best partition)	Gain (%)
VGG16	1.048 (1)	1.897 (1)	1.817 (4)	4.22	2.059 (1)	2.059 (1)	0
VGG19	1.261 (1)	3.389 (1)	2.040 (4)	39.81	3.385 (1)	2.039 (1)	39.76
ResNet50	0.952 (112)	1.054 (112)	0.945 (1)	10.34	1.048 (112)	0.941 (1)	10.31
ResNet50V2	0.678 (2)	0.684 (2)	0.676 (1)	1.17	2.094 (2)	0.753 (87)	64.04
LeNet	0.008 (3)	0.010 (3)	0.009 (6)	10.00	0.012 (3)	0.011 (7)	8.33

observed for VGG19 and ResNet50V2. The DNNs are less sensitive to memory stress than CPU stress.

Table VI shows the performance gain of repartitioning DNNs for different network data transfer rates. The table shows the end-to-end latency (in seconds) and the partitioning layer for 50Mb/s, 25Mb/s, and 10Mb/s between the edge and the cloud. The partitioning layer is the same for 25Mb/s and 10Mb/s as it is when there is maximum available bandwidth. The general trend is that the end-to-end latency increases with decreasing data transfer rates. The table then shows the end-to-end latency of the best DNN partition at a different partition layer for 25Mb/s and 10Mb/s (shown in the table as best partition). The performance gain of employing the best partition over using a static partitioning layer (best partitioning point when network data transfer rate is 50Mb/s) is highlighted.

It is immediately inferred that the performance gain for the selected DNNs in response to different network data transfer rates is greater than for CPU or memory stress. AlexNet, that was not sensitive to CPU and memory stress, is more sensitive to changing network conditions and benefits from repartitioning; up to 37.67% gains are noted. Although VGG16

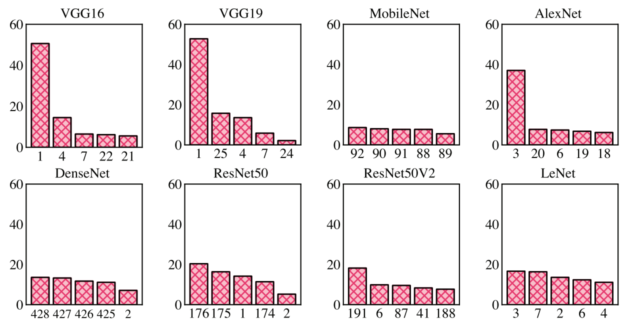


Fig. 11: Percentage of performance efficient partitioning points for the DNNs when there is edge CPU stress of 90% for different memory stress on the edge and network data transfer rates between the edge and the cloud on experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

is sensitive to network conditions, there are instances when there is no performance gain.

4) DNN Adaptivity and a Combination of Operational Conditions: Figure 11 and Figure 12 show the percentage of

TABLE VI: Effect of network data transfer rates between the edge and the cloud on end-to-end latency (seconds) when using experimental platform P_1; partitioning layer is shown in brackets beside the end-to-end latency values. End-to-end latency is shown for 10Mb/s, 25Mb/s and 50Mb/s data transfer rate (partitioning layer is the same as when transfer is 50Mb/s). End-to-end latency for the best DNN partition when network data transfer rate is 25Mb/s and 10Mb/s along with the partitioning layer is shown. The performance gain of the best partition is shown. LeNet and DenseNet are not shown as they are not sensitive to network transfer rates (refer to Table III).

DNN Model	50Mb/s (best partition)	25Mb/s	25Mb/s (best partition)	Gain (%)	10Mb/s	10Mb/s (best partition)	Gain (%)
VGG16	1.048 (1)	1.131 (1)	1.131 (1)	0	3.606 (1)	3.004 (6)	16.70
VGG19	1.261 (1)	5.585 (1)	2.489 (7)	55.43	14.307 (1)	3.254 (7)	77.25
MobileNet	0.280 (88)	0.319 (88)	0.293 (90)	8.15	0.384 (88)	0.346 (90)	9.89
AlexNet	0.107 (3)	0.446 (3)	0.278 (23)	37.67	0.516 (3)	0.345 (10)	33.14
ResNet50	0.952 (112)	1.174 (112)	0.971 (1)	17.29	1.632 (112)	1.196 (1)	26.72
ResNet50V2	0.678 (2)	0.721 (2)	0.717 (1)	0.55	2.295 (2)	0.974 (155)	57.56

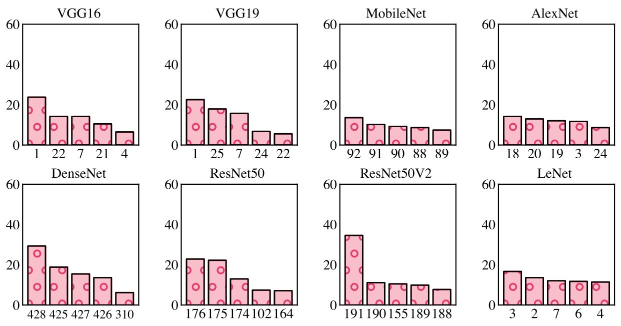


Fig. 12: Percentage of performance efficient partitioning points for the DNNs when there is 0% edge CPU stress for different memory stress on the edge and network data transfer rates between the edge and the cloud on all experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

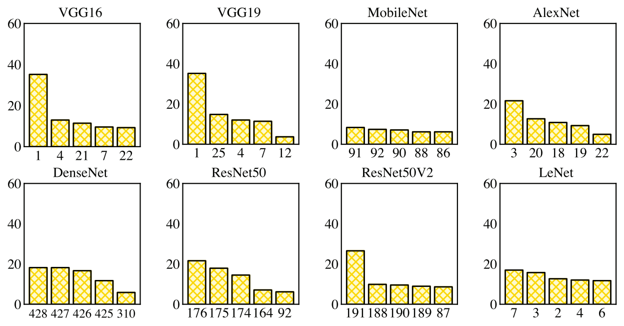


Fig. 13: Percentage of performance efficient partitioning points for the DNNs for a maximum memory stress of 90% at the edge for different CPU stress on the edge and network data transfer rate on all experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

the top five performance efficient partitioning points for the DNNs when there is a CPU stress of 90% and of 0% on the edge, respectively, when there is different memory stress at the edge and data transfer rates between the edge and the cloud. The general observation is that partitions with more layers on the cloud are appropriate when CPU stress is maximum. For example, consider AlexNet. The two prominent partitioning layers when CPU stress is at a minimum are 18 and 20. However, when CPU stress is at 90% the optimal partitioning point is layer 3. A few observations from the results are that: (i) for VGG16 layer 1 and layer 22 are usually the prominent performance efficient partitioning points when CPU stress is 0% meanwhile layer 4 becomes a prominent partitioning point when CPU stress is 100%, (ii) AlexNet becomes less sensitive to increased CPU stress as Layer 3 becomes a prominent partitioning point, (iii) the optimal partitioning point for DenseNet is Layer 428 for both minimum and maximum CPU stress, even as network and memory performance vary.

Figure 13 and Figure 14 show the percentage of the top five performance efficient partitioning points for the DNNs when there is memory stress of 90% and 0% on the edge, respectively, for different CPU stress at the edge and data transfer rates between the edge and cloud. The general observation is that there are fewer changes to the optimal partitioning points. Memory stress has limited impact on partitioning.

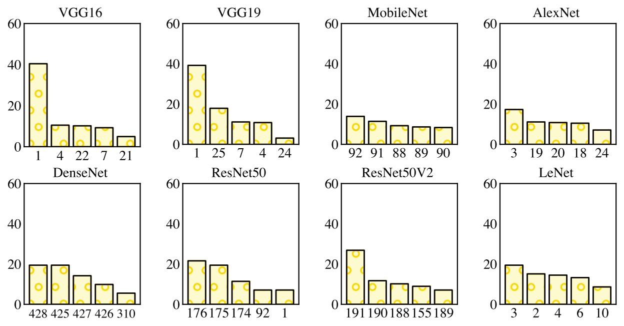


Fig. 14: Percentage of performance efficient partitioning points for the DNNs for a minimum memory stress of 0% at the edge for different CPU stress on the edge and network data transfer rate on all experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

Figure 15 and Figure 16 show the percentage of performance efficient partitioning points for the DNNs when the network data transfer rate between the edge and the cloud is 50Mb/s and 10Mb/s under different CPU and memory stress levels on the edge. The graphs highlight that DNNs are sensitive to a combination of operational conditions. Although individual operational conditions, such as CPU or memory stress may not affect DNNs substantially, the combination of

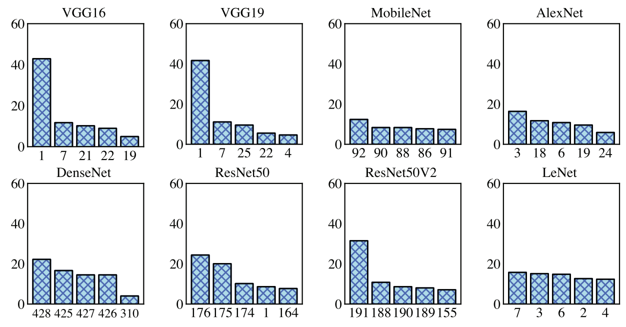


Fig. 15: Percentage of performance efficient partitioning points for the DNNs when the network data transfer rate between the edge and the cloud is 50Mb/s under different CPU and memory stress on the edge on all experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

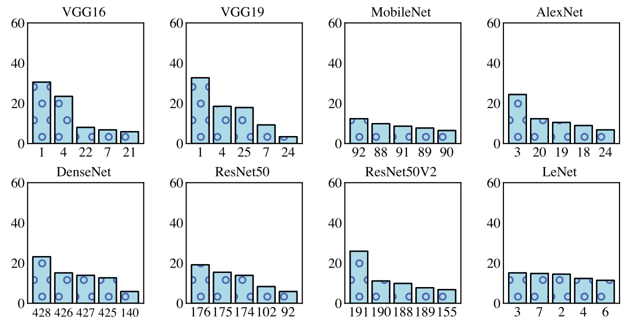


Fig. 16: Percentage of performance efficient partitioning points for the DNNs when the network data transfer rate between the edge and the cloud is 10Mb/s under different CPU and memory stress on the edge on all experimental platforms P_1, P_2, P_3 and P_4. X-axis shows the partitioning point.

operational conditions makes a case for adaptive DNNs.

The results highlight that DNNs are sensitive to operational conditions and hence are amenable to repartitioning. However, it is observed that DNNs are more sensitive to network data transfer than CPU or memory stress when considered individually. A performance gain is observed when DNNs are repartitioned. A case for DNN adaptivity is stronger when a combination of operational conditions are considered. Although DNNs are sensitive to underlying hardware architectures, a specific pattern was not noted. These highlight the importance of application adaptivity that need the edge.

V. Related Work

Adaptivity in DNNs has not been presented as a singular concept in the existing literature, rather is a reference to multi-faceted aspects of DNN execution. In this section, the presentation of two classes of adaptive DNNs is highlighted. The first is adaptivity in the context of native DNN execution on a resource and the second in relation to DNN partitioning.

Adaptivity in the context of native DNN execution on a device or a server is considered in three different ways. Firstly, DNN adaptivity for executing pre-trained models, for example in the acoustic context by taking into account the speaker or environment [25]. Typically, the DNNs are fine-tuned to provide a higher quality result measured by accuracy [26]. Three types of adaptive approaches are considered, namely input feature transformation, direct adaptation by transforming DNN parameters, and using auxiliary context features [25]. In this case, adaptivity refers to taking the acoustic environment or user into account for maximizing DNN performance.

Secondly, adaptivity is considered to choose a DNN model for inference from a portfolio of models using a learning approach to maximize estimation accuracy for a given input [27].

Thirdly, adaptivity is in the context of model compression to execute DNN models on resource constrained devices [28]. The weight matrices of fully connected and convolutional neural networks are compressed without losing significant accuracy. The approach uses a disciplined approach using singular value decomposition to prune fully-connected structures, which is what is referred to as being adaptive.

Adaptivity in DNN partitioning is also presented in the literature. With the emergence of edge computing, DNN partitioning and distribution across devices, edge and cloud resources for inference has become an important avenue of research. Literature would suggest that adaptivity in the context of partitioning and distributing DNNs across multiple resources is not only about improving accuracy, but about optimizing end-to-end latencies. In other words, adaptivity is understood as making DNN partitions suitable for the operational context.

There are numerous DNN partitioning methods presented in the literature. These methods include, estimation-based techniques [6], [13], [14], structural modification-based techniques [15], [16], and measurement-based techniques [17]. These techniques identify an optimal partitioning point based on the characteristics of the DNN layers and operational conditions, such as resource utilization or network conditions.

The sensitivity of DNNs to operational conditions of the edge resource (resource utilization) and the network state between the edge and the cloud for repartitioning is understood in a limited way. There is recent research that focuses on adaptivity for DNN partitioning [29]. However, the main consideration is performance-awareness for which metrics, such as per layer latency, are used to inform DNN partitioning. A combination of an estimation-based approach is used for finding the partitioning point. An early exit strategy is employed for further optimizing performance. The research is pursued in the context of initial partitioning and the effect of evolving operational conditions are not explored.

The impact of network conditions for DNN repartitioning has been considered [30]. The network is assumed to have two states - lightly and heavily loaded states. However, how sensitive are DNNs to operational conditions (both resource specific and network specific) is not considered.

Similarly, there is research that considers adaptivity in the context of partitioning output data from layers using a compression-based technique, referred to as the compressed sparse row scheme [31]. The output matrices of a layer are partitioned into dense and sparse partitions. This compression relies on network conditions (thus considered as adaptive).

VI. Conclusions

Performance-critical and privacy-sensitive applications benefit from edge computing by distributing selected services of an application closer to where data is generated. This allows applications to pre-process and selectively release data to make the overall application more responsive and privacy-aware. Deep Neural Networks (DNNs) are a class of applications that naturally lend to distribution across the cloud and edge given that they are organized as a sequence of layers. The distribution of a DNN is achieved by partitioning it at a layer that would maximize its performance while taking operational conditions into account (CPU/memory stress on the edge or network data transfer rates between the edge and cloud).

There is limited understanding of how evolving operational conditions might affect the performance of a distributed DNN and whether a new partition is required to optimize the overall performance. If operational conditions affect DNNs, then they will need to be repartitioned (a new layer at which the DNN can be partitioned). A DNN that adapts to the operational conditions is defined in this paper as an 'adaptive DNN'.

This paper set out to investigate *whether there is a case for adaptive DNNs in edge computing*. An exploratory exercise was carried out by benchmarking 8 pre-trained production DNNs for different operational conditions, such as CPU/memory stress and network data transfer rates. The results presented were obtained from a cloud-edge lab-based experimental platform by analyzing nearly 8 million data points. The key observations are that DNNs are sensitive to both individual and a combination of operational conditions. When considering individual operational conditions, network conditions have a more substantial impact on DNN performance than CPU/memory stress on an edge node. Repartitioning can provide performance gains for certain DNNs considered. Operational conditions in combination have a more significant impact on DNN repartitioning than individual operational conditions. This paper concludes that there is a case for adaptive DNNs at the edge.

A limitations of the current exploration is that the execution of the DNNs on CPUs is assumed. There is a compelling case for using hardware accelerator platforms, such as GPUs, TPUs and ASICs, which will be considered in the future.

Acknowledgment

Dr Blesson Varghese is supported by a Royal Society Short Industry Fellowship and funds from Rakuten Mobile, Japan.

References

[1] B. Varghese *et al.*, "Challenges and Opportunities in Edge Computing," in *Proc. of the IEEE Int. Conf. on Smart Cloud*, 2016, pp. 20–26.

[2] W. Shi *et al.*, "Edge Computing: Vision and Challenges," *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, 2016.

[3] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, 2017.

[4] B. Varghese *et al.*, "Next Generation Cloud Computing: New Trends and Research Directions," *Future Generation Computer Systems*, vol. 79, no. 3, pp. 849–861, 2018.

[5] K. Ha *et al.*, "Towards Wearable Cognitive Assistance," in *Proc. of the 12th Conf. on Mobile Sys., Applications, and Services*, 2014, p. 68–81.

[6] K. J. Hsu *et al.*, "Couper: DNN Model Slicing for Visual Analytics Containers at the Edge," in *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, 11 2019, pp. 179–194.

[7] L. Liu *et al.*, "A Comparison of Communication Mechanisms in Vehicular Edge Computing," in *Proceedings of the 3rd USENIX Workshop on Hot Topics in Edge Computing*, 2020.

[8] M. Satyanarayanan *et al.*, "The Seminal Role of Edge-Native Applications," in *Proceedings of the IEEE International Conference on Edge Computing*, 2019, pp. 33–40.

[9] Y. LeCun *et al.*, "Deep Learning," *Nature*, vol. 521, p. 436–444, 2015.

[10] J. Schmidhuber, "Deep Learning in Neural Networks: An Overview," *Neural Networks*, vol. 61, p. 85–117, 2015.

[11] Z. Zhou *et al.*, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1738–1762, 2019.

[12] X. Wang *et al.*, "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey," *IEEE Comms. Surveys Tutorials*, 2020.

[13] Y. Kang *et al.*, "Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge," in *Proc. of the 22nd Int. Conf. on Architectural Support for Programming Languages and Operating Systems*, 2017.

[14] M. Xu *et al.*, "DeepWear: Adaptive Local Offloading for On-Wearable Deep Learning," *IEEE Transactions on Mobile Computing*, vol. 19, no. 2, pp. 314–330, 2020.

[15] Z. Zhao *et al.*, "DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348–2359, 2018.

[16] J. Mao *et al.*, "MoDNN: Local Distributed Mobile Computing System for Deep Neural Network," in *Proceedings of the Design, Automation Test in Europe Conference Exhibition*, 2017, pp. 1396–1401.

[17] S. Yi *et al.*, "LAVEA: Latency-Aware Video Analytics on Edge Computing Platform," in *Proceedings of the IEEE 37th International Conference on Distributed Computing Systems*, 2017, pp. 2573–2574.

[18] K. Simonyan *et al.*, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2015.

[19] A. G. Howard *et al.*, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 2017.

[20] J. Deng *et al.*, "ImageNet: A Large-Scale Hierarchical Image Database," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[21] G. Huang *et al.*, "Densely Connected Convolutional Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2261–2269.

[22] K. He *et al.*, "Deep Residual Learning for Image Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[23] Y. LeCun *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[24] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. [Online]. https://www.tensorflow.org/

[25] M. Delcroix *et al.*, "Context Adaptive Deep Neural Networks for Fast Acoustic Model Adaptation in Noisy Conditions," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2016, pp. 5270–5274.

[26] Y. Miao *et al.*, "Speaker Adaptive Training of Deep Neural Network Acoustic Models Using I-Vectors," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 23, no. 11, pp. 1938–1949, 2015.

[27] B. Taylor *et al.*, "Adaptive Deep Learning Model Selection on Embedded Systems," *SIGPLAN Notices*, vol. 53, no. 6, p. 31–43, 2018.

[28] R. Wang, "Adaptive Solution to Compress Deep Neural Networks for Resource-constrained Devices," Master's thesis, University of California Riverside, 2019. [Online]. https://escholarship.org/uc/item/7xb9j5mc

[29] H. Wang *et al.*, "ADDA: Adaptive Distributed DNN Inference Acceleration in Edge Computing Environment," in *Proc. of the IEEE 25th Int. Conf. on Parallel and Distributed Systems*, 2019, pp. 438–445.

[30] C. Hu *et al.*, "Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge," in *Proceedings of the IEEE Conference on Computer Communications*, 2019, pp. 1423–1431.

[31] T. Mohammed *et al.*, "Distributed Inference Acceleration with Adaptive DNN Partitioning and Offloading," in *Proceedings of the IEEE Conference on Computer Communications*, 2020.