Inference Time Optimization Using BranchyNet Partitioning

Roberto G. Pacheco * and Rodrigo S. Couto*

* Universidade Federal do Rio de Janeiro - PEE/COPPE/GTA - Rio de Janeiro, RJ, Brazil

Email: pacheco@gta.ufrj.br, rodrigo@gta.ufrj.br

Abstract—Deep Neural Network (DNN) inference requires high computation power, which generally involves a cloud infrastructure. However, sending raw data to the cloud can increase the inference time due to the communication delay. To reduce this delay, the first DNN layers can be executed at an edge infrastructure and the remaining ones at the cloud. Depending on which layers are processed at the edge, the amount of data can be highly reduced. However, executing layers at the edge can increase the processing delay. A partitioning problem tries to address this trade-off, choosing the set of layers to be executed at the edge to minimize the inference time. In this work, we address the problem of partitioning a BranchyNet, which is a DNN type where the inference can stop at the middle layers. We show that this partitioning can be treated as the shortest path problem, and thus solved in polynomial time [1].

I. Introduction

Deep Neural Networks (DNN) are largely employed in machine learning applications, such as computer vision and speech recognition. DNN is composed of neuron layers, where each neuron receives inputs and generates a non-linear output. In summary, a DNN architecture is composed of an input layer, a sequence of middle layers, and an output layer. For image classification, DNN inference executes a feed-forward algorithm to label an input image into one of the predefined classes. In this algorithm, each layer receives the output data from the prior layer, executes a computation, and then propagates its output data to the next layer. DNN inference executes this algorithm from the input layer through the middle layers, until it reaches the output layer, which generates the probability for each predefined class [1].

Traditionally, DNNs can be deployed on end devices (e.g., smartphones and personal assistants) or on a cloud server [2], [3]. DNN inference generally requires high computational power, and its execution on resource-constrained end devices can result in a prohibitive processing delay. DNN inference can thus be executed in a cloud computing infrastructure, which is generally equipped with computational resources to accelerate the processing, such as GPUs (Graphics Processing Unit). In cloud-based solutions, end devices gather the data and transmit it to the cloud server, which executes DNN inference. This adds a data communication delay, which is affected by the network behavior between the end device and the cloud, increasing the inference time. This is a severe problem since recent DNN applications, such as cognitive assistance and intelligent vehicles, require high responsiveness [4]. It is necessary to reduce communication and processing delays to achieve high responsiveness. The former is the time required to send data, through the Internet, from the end device to the cloud server. The second one is the time to perform the inference itself, related to the employed hardware. Edge computing emerges as an alternative to reduce the communication delay imposed by cloud computing [2]. This paradigm consists of deploying computational resources at the edge of the Internet (i.e., close to end devices), reducing the communication delay. Edge servers can be installed in locations such as cellular base stations and Wi-Fi access points. Nevertheless, edge devices provide a computational capacity significantly lower than the cloud, which adds processing delay. Therefore, edge computing reduces communication delay but increases the processing delay as compared to the cloud. Thus, considering responsiveness, there is a clear trade-off between the communication delay and the processing one.

In literature, there are proposals to handle each one of the delays related to DNN inference. To reduce processing delay, BranchyNet proposes classifying an input sample at the middle layers if a certain confidence level is achieved [5]. Regarding communication delay, DNN partitioning proposes to compute the first DNN layers at the edge device and the other ones at the cloud server. This proposal is based on the fact that the communication delay to send data from middle layers is significantly lower than the delay to send a raw image [6]. This paper combines BranchyNet and DNN partitioning to evaluate the trade-off between processing and communication delay. To address this trade-off, this paper formalizes an optimization problem whose objective is to find an optimal partition that minimizes the inference time for a BranchyNet. This optimization problem depends not only on network bandwidth and the computational power of edge and cloud, but also aspects inherent to input data, such as image quality. To this end, we model the inference time for BranchyNet. Then, to minimize the inference time, we show the equivalency between BranchyNet partitioning and the shortest path problem. Thus, we can derive a globally optimal solution in polynomial-time.

This paper is organized as follows. Section II reviews related works about DNN partitioning. Section III presents basics concepts of BranchyNet. Section IV models the infer-

ence time for this DNN type. Then, Section V formalized the BranchyNet partitioning problem. The experiments are shown in Section VI. Finally, Section VII concludes this paper and suggests future directions.

II. RELATED WORK

To accelerate DNN inference, prior works study how to partition a DNN between edge devices and the cloud server. Neurosurgeon [3] constructs performance prediction models for DNNs. It allows to estimate the processing delay at the edge device and the cloud server. Then, these prediction models are combined with wireless network conditions to dynamically select the best partition. However, Neurosurgeon is limited to chain-topology DNNs. To address this limitation, DADS [6] (Dynamic Adaptive DNN Surgery) optimally partitions a general DAG (Directed Acyclic Graph) topology DNN. To this end, DADS treats the partitioning problem as a min-cut problem. These papers propose DNN partitioning methods, considering DNNs with no side branches.

Li et al [7] propose a partitioning method to BranchyNets [5] framework. However, different from our work, they use the Branchynet framework only for choosing the DNN size. Instead of using a confidence level threshold in each side branch, their proposal uses a brute force search to choose the branch and the partition decision that achieves a given latency requirement and maximizes the inference accuracy. This method may be unfeasible for increasingly deeper DNNs. We thus assume that confidence level thresholds are well-chosen before the execution of our partitioning method, guaranteeing a high accuracy level. Hence, given the BranchyNet architecture with its thresholds, our paper optimally partitions a BranchyNet to minimize inference time. Moreover, this paper is the first work to model the inference time for a BranchyNet, considering the probability that a sample is classified at the side branch also as a factor that impacts the inference time. Then, we convert the BranchyNet partitioning problem into a shortest path problem, which can be feasible to increasingly deeper DNNs.

III. BRANCHYNET

BranchyNet is a DNN architecture, whose goal is to accelerate inference. This architecture is based on the idea that features extracted in the first layers can label correctly a large number of samples on a dataset. To this end, BranchyNet proposes to modify an original DNN architecture, inserting side branches at the middle layers. These side branches allow an input sample to be classified at middle layers, instead of the output one as in regular DNNs. BranchyNet can use entropy as an uncertainty metric to compute the confidence level of sample classification, to decide if the inference can stop or not at the middle layers [5].

Figure 1 illustrates a generic BranchyNet with $N-1$ side branches. In this figure, the nodes v_1 to v_N represent layers of the main branch, and b_1 to b_{N-1} refers to the side branches inserted at those middle layers. In summary, these layers can be of three types: convolutional (conv), max-pooling

(max-pool) and fully-connected (fc). The convolutional layers consist of a set of filters, whose components are learnable parameters during the training process. Each filter is responsible to generate a set of output features using convolutional operations. The max-pooling layers provide robustness to noise in output features of a convolutional layer. To this end, max-pooling layers get the maximum value of a predefined window. The output fully-connected layer receives the features extracted by the previous convolutional layers and generates a probability vector, containing the probability that a sample belongs to each predefined class.

Once trained, BranchyNet receives an image, which is processed, layer by layer, until a side branch is reached. Then, on the side branch, it computes the confidence level of sample classification based on the probability vector and verifies if this confidence level is less than a threshold. If so, the inference finishes and the class inferred is the class with the highest probability. Hence, this sample is not processed by any next layer, reducing the number of processed layers and thus the processing delay. Otherwise, the sample is processed by the next layers of the main branch until the next side branch is reached. Then, the whole procedure is performed for this branch. If the sample is not classified in any branch, the inference ends when the output layer is reached.

When the majority of samples on a dataset cannot be classified at side branches, executing BranchyNet inference at an edge device can introduce processing delay. To avoid that, it is necessary to determine which layer the DNN partitioning must occur to minimize inference time. This decision should take into account the probability of classifying at side branches, the network conditions, and the processing capacity of edge and cloud hardware. Therefore, this work formalizes a BranchyNet partitioning problem, giving the model defined next.
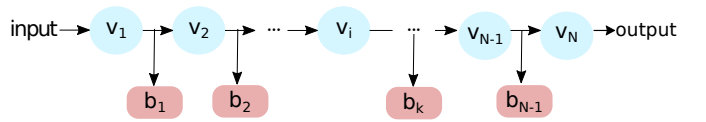

Fig. 1. An illustration of a general BranchyNet.

IV. PARTITIONING MODEL

In this section, we model the inference time for BranchyNet partitioning. To this end, we represent BranchyNet as a graph and define the BranchyNet partitioning problem.

A. BranchyNet Graph

A DNN can be modeled as a DAG $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. The set $\mathcal{V}$ contains vertices of $\mathcal{G}$. Each vertex $v_1, \cdots, v_{|\mathcal{V}|}$ represents the layers in a DNN. For instance, the vertices v_1 and $v_{|\mathcal{V}|}$ represent the input and output layers, respectively. The set $\mathcal{E} = \{e_{ij} = (v_i, v_j) | v_i, v_j \in \mathcal{V}\}$ contains the links[2] of the graph. The link $(v_i, v_j) \in \mathcal{E}$ exists if and only if the output

---

[2]In this paper, we use the word "link" to denote an edge in a graph, to avoid misunderstanding with edge computing.

data of layer v_i feeds the input of the layer v_j. Thus, the layer v_i is processed before v_j.

A BranchyNet can be modeled as a DAG since it is a DNN. According to Figure 1, a BranchyNet is characterized by inserting a side branch $b_i \in \mathcal{B}$ between the middle layers of the main branch, where $\mathcal{B}$ is the set of side branches. Therefore, we can summarize the BranchyNet architecture into two components: the main branch and side branches. In this paper, the main branch is modeled as a chain graph, denoted by $\mathcal{P}_{|\mathcal{V}'|}$, where the sub-index $|\mathcal{V}'|$ indicates the number of vertices in $\mathcal{P}_{|\mathcal{V}'|}$. In a chain graph, each vertex v_i has only one outgoing link, representing its connection to v_{i+1}, for all $v_i \in \mathcal{V}'$, since $(v_i, v_{i+1}) \in \mathcal{E}$. Let $\mathcal{G}_{\text{BDNN}}$ be a DAG of BranchyNet. To model the graph $\mathcal{G}_{\text{BDNN}}$, we introduce the vertices $b_i \in \mathcal{B}$ into the graph $\mathcal{P}_{|\mathcal{V}'|}$ of the main branch. In addition, we replace the link (v_i, v_{i+1}) with the links (v_i, b_i) and (b_i, v_{i+1}). In other words, we replace the outgoing link from vertex v_i to its neighbor v_{i+1} with a link from v_i to the side branch vertex b_i, and we add a link from b_i to v_{i+1}. Thus, at this stage, a BranchyNet also can be modeled by a chain graph denoted by $\mathcal{P}_{|\mathcal{V}' \cup \mathcal{B}|}$.

B. BranchyNet Partitioning

BranchyNet partitioning problem consists of choosing which layer sends its output data from an edge device to the cloud. In this section, we approach this problem as a graph partitioning problem. Given a $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ graph and a positive integer K, the graph partitioning finds vertex sets $\mathcal{V}_1, \cdots, \mathcal{V}_K$, such that $\bigcup_{i=1}^{K} \mathcal{V}_i = \mathcal{V}$ and $\bigcap_{i=1}^{K} \mathcal{V}_i = \emptyset$, which means that each node $v_i \in \mathcal{V}$ belongs to only one of these subsets.

BranchyNet partitioning splits the set $\mathcal{V}$ of a BranchyNet graph $\mathcal{G}_{\text{BDNN}}$ into two (i.e., $K = 2$) disjoint subsets $\mathcal{V}_e$ and $\mathcal{V}_c$. The vertices $v_1, \cdots, v_{|\mathcal{V}_e|}$ represent BranchyNet layers processed at the edge. The vertices $v_i \in \mathcal{V}_c$ are the layers processed in the cloud. As in graph partitioning, each vertex belongs to only one subset thus the layer v_i is processed only at the edge device or at the cloud server, which means that $\mathcal{V}_e \cap \mathcal{V}_c = \emptyset$. As explained in Section IV-A, we model the main branch of Branchynet as a chain graph. Thereby, formally, the partitioning task must find a layer $v_s \in \mathcal{V}'$ that defines $\mathcal{V}_e$ set that splits $\mathcal{G}$ into two parts. The partitioning layer v_s is the last one to be processed in the edge. Hence, the layers from v_1 to v_s are processed at edge. Then, the edge device sends the output data of v_s to the cloud that processes the next layers.

Once the partitioning layer v_s is found, we can determine which set each layer belongs to. Formally, the set processed at the edge is $\mathcal{V}_e = \mathcal{V}' \cup \mathcal{B}$, where $\mathcal{V}' = \{v_1, \cdots, v_s\}$ and $\mathcal{B} = \{b_1, \cdots, b_{s-1}\}$. Hence, the set processed in the cloud is $\mathcal{V}_c = \{v_{s+1}, \cdots, v_{|\mathcal{V}|}\}$. It is important to note, in this paper, that no side branch is processed in the cloud, hence no vertex $b_i \in \mathcal{B}$ belongs to the set $\mathcal{V}_c$. Therefore, all vertices b_i posterior to v_s are discarded. It occurs because, in the cloud, the reduction of processing delays provided by classifying a sample at the side branch is negligible when compared with the time required to execute all layers of the main branch. Moreover, this reduction is significantly lower than the communication delay between

the edge and the cloud. Figure 2 shows different partitioning scenarios for a BranchyNet composed of four layers.

In Figure 2, the gray vertices represent the layers processed at the edge device, the orange vertex corresponds to the partitioning layer v_s and the blue vertices refer to layers processed at the cloud server. Figure 2(a) illustrates an example where all layers are processed in the edge, so the partitioning layer is v_4. Thus, we have $\mathcal{V}_e = \{v_1, v_2, v_3, v_4, b_1, b_2, b_3\}$ and $\mathcal{V}_c = \emptyset$. In this case, no data is sent to the cloud. On the other hand, Figure 2(b) shows cloud-only processing where all layers are processed at the cloud, so $\mathcal{V}_e = \emptyset$ and $\mathcal{V}_c = \{v_1, v_2, v_3, v_4\}$. In this case, the output data size sent to the cloud corresponds to the raw input data size. Finally, Figure 2(c) shows an example where partition occurs in the layer v_2, so $\mathcal{V}_e = \{v_1, b_1, v_2\}$ and $\mathcal{V}_c = \{v_3, v_4\}$. In this case, the output data of layer v_2 is sent from an edge device to the cloud.

Generally, the partitioning splits a BranchyNet to follow an objective, such as reducing inference time, saving bandwidth, maximizing inference accuracy, or even reducing energy consumption. In this paper, our goal is to minimize the inference time. We derive next a model to estimate the inference time.

C. Estimation of Inference Time

In DNN applications with edge computing, end-devices send input data to the edge device. When the DNN has no side branches, this data is processed by all layers placed at the edge. The edge device sends the output data of the partitioning layer v_s to the cloud, which is responsible to process the remaining layers. Hence, in this DNN, the inference time depends on the processing delay in the edge and the cloud, as well as their communication delay.

Edge devices and cloud servers differ significantly regarding their computational power and thus have different processing delays. Hence, the processing time for a given layer depends on where it is computed. Let t_i^e and t_i^c be the processing time to compute the layer v_i at the edge and the cloud, respectively. The total processing delay at the edge, when the DNN has no side branches, is

$$T_e = \sum_{i|v_i \in \mathcal{V}_e} t_i^e, \tag{1}$$

and the processing time to compute all the remaining layers at cloud is given by

$$T_c = \sum_{i|v_i \in \mathcal{V}_c} t_i^c. \tag{2}$$

The communication delay depends on the output data size of the partitioning layer and the network bandwidth. The output data size generated by each DNN layer presents a non-monotonic behavior, which means each layer produces output data with different sizes. Therefore, there are layers closer to the input layer that generate less data than a deeper layer (i.e., closer to output layer), resulting in higher communication delay. For each layer v_i, we can define the communication time as $t_i^{\text{net}} = \frac{\alpha_i}{B}$ where α_i is the output data size of layer v_i and B is the network bandwidth. At this stage, we can define
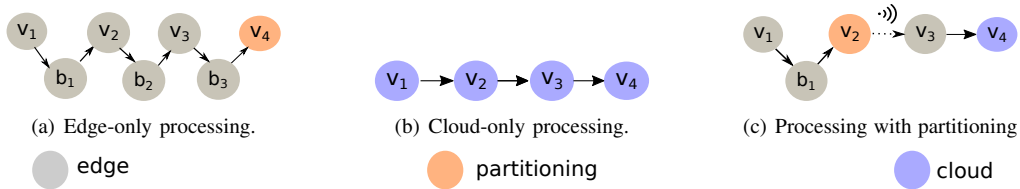
Fig. 2. Possible scenarios for DNN processing.

$\boldsymbol{t_e} = [t_1^e, \cdots, t_{|\mathcal{V}|}^e]$ and $\boldsymbol{t_c} = [t_1^c, \cdots, t_{|\mathcal{V}|}^c]$ as parameters related to hardware resources and $\boldsymbol{t}_{\text{net}} = [t_1^{\text{net}}, \cdots, t_{|\mathcal{V}|}^{\text{net}}]$ related to network condition. Additionally, we can assign a 3-tuple $(t_i^e, t_i^{\text{net}}, t_i^c)$ to each vertex v_i of a DNN with no side branches or for the main branch in a BranchyNet $\mathcal{G}_{\text{BDNN}}$.

In a DNN with no side branches, the inference time is the sum of the total processing delay (i.e. T_e and T_c) with the communication delay (i.e., t_s^{net}), as shown in Equation 3 [6].

$$T_{inf}^{DNN} = T_e + t_s^{\text{net}} + T_c \qquad (3)$$

If we consider a DNN with side branches (i.e., a BranchyNet), an inference can stop at one of these branches, if the classification achieves a certain confidence criterion. To model the inference time for DNN with side branches, we divide our analysis into a particular case and a generic one.

1) Particular Case: We first consider a particular BranchyNet, with only one side branch b_k, placed in the output of any middle layer v_k, where $1 \leq k \leq (|\mathcal{V}| - 1)$. As BranchyNet inference algorithm described in Section III, in this case, when a sample reaches this side branch b_k, it can be classified and exits the BranchyNet at that middle layer or is processed by the next layers until reach the output layer. To model this two possible outcomes, we define a Bernoulli random variable X_k that takes on 1 if a sample is classified at the side branch b_k with probability $\mathsf{P}[X_k = 1] = p_k$, where $p_k \in [0, 1]$. Otherwise, $X_k = 0$ if the sample is process by the next layers to side branch b_k with $\mathsf{P}[X_k = 0] = 1 - p_k$.

2) General Case: We generalize the previous analysis, considering a BranchyNet with $|\mathcal{V}| - 1$ side branches, as shown in Figure 1. As in particular case, we define a Bernoulli random variable X_k for each side branch $b_k \in \mathcal{B}$, resulting in a sequence of Bernoulli random variables $\boldsymbol{X} = (X_1, X_2, \cdots, X_{|\mathcal{V}|-1})$. For a sample to be classified at side branch b_k, this sample cannot meet the confidence criterion in any of the $k - 1$ previous side branches. Thus, the Bernoulli random variable X_k takes on value 1 for the first time at side branch b_k after $k - 1$ trials. To model it, we define a random variable Y that represents the number of side branches, which have already been processed before the current side branch b_k, and it cannot classify the sample. Then, the probability of random variable Y as follows:

$$p_Y(k) = \mathsf{P}[Y = k] = p_k \prod_{i=1}^{k-1}(1 - p_i). \qquad (4)$$

Considering a batch of input data samples, we can compute the expected value of the inference time. As the cloud has no

side branches, if the partitioning occurs earlier than the first side branch, the inference time is modeled by Equation 3 as a DNN with no side branches. If the partitioning occurs after the side branch, the sample can be classified by this branch, and thus it is not processed by the remaining layers. In this case, in a BranchyNet with only one side branch, the expected value of the inference time is

$$\mathbb{E}[T_{inf}^{\text{BDNN}}(k)] = \sum_{\substack{i \mid v_i \in \mathcal{V}_e \\ i \leq k}} t_i^e + (1 - p_Y(k)) \left(\sum_{\substack{i \mid v_i \in \mathcal{V}_e \\ i \leq k}} t_i^e + t_s^{\text{net}} + T_c \right). \qquad (5)$$

Equation 5 shows that the edge device always processes the layers before the side branch b_k. However, the processing and communication delays of any remaining layers are weighted by the probability of classifying the input data at side branch. In an extreme case, where the input samples are always classified at the side branch, which means $p = 1$, Equation 5 considers neither the communication delay nor the processing delay for the remaining layers. On the other extreme, if the inference never stops at a side branch (i.e., $p = 0$), Equation 5 is equal to Equation 3. At this stage, according to the partitioning layer position, the expected value of the inference time can be modeled as follows:

$$\mathbb{E}[T_{inf}(k)] = \begin{cases} T_{inf}^{DNN}, & \forall v_k, v_s \mid s < k. \\ \mathbb{E}[T_{inf}^{\text{BDNN}}], & \text{otherwise.} \end{cases} \qquad (6)$$

V. BRANCHYNET PARTITIONING OPTIMIZATION

In this section, our goal is to determine the partitions $\mathcal{V}_e$, $\mathcal{V}_c$ that minimize the inference time of Equation 6, given the input parameters $\boldsymbol{t_e}$, $\boldsymbol{t_c}$, $\boldsymbol{t}_{\text{net}}$, B and $p_Y(k)$. As defined in Section IV-A, the main branch of BranchyNet is modeled as a chain graph $\mathcal{P}_{|\mathcal{V}'|}$. Hence, if we find a partitioning layer v_s that minimize $\mathbb{E}[T_{inf}(k)]$, we can determine the partitions $\mathcal{V}_e$ and $\mathcal{V}_c$. To this end, we propose to construct a new graph $\mathcal{G}'_{\text{BDNN}}$ based on the BranchyNet graph. These graph $\mathcal{G}'_{\text{BDNN}}$ allows to associate one delay of the a 3-tuple $(t_i^e, t_i^c, t_i^{\text{net}})$ to each link, as described Section IV-C. The delay associated with each link (v_i, v_j) depends on where the vertex v_i is processed. When then we show that the partitioning problem can be considered as the shortest path problem in $\mathcal{G}'_{\text{BDNN}}$. To build $\mathcal{G}'_{\text{BDNN}}$, we create two disjoint chain graphs $\mathcal{P}^e_{|\mathcal{V}' \cup \mathcal{B}|}$ and $\mathcal{P}^c_{|\mathcal{V}'|}$, such as those of Figures 2(a) and 2(b), respectively. The vertices of $\mathcal{P}^e_{|\mathcal{V}' \cup \mathcal{B}|}$ and $\mathcal{P}^c_{|\mathcal{V}'|}$ represent the layers processed in the edge in the cloud, respectively. Then, we assign a weight $\omega_{(v_i, v_j)}$ to each link (v_i, v_j) of the graphs $\mathcal{P}^e_{|\mathcal{V}' \cup \mathcal{B}|}$ and $\mathcal{P}^c_{|\mathcal{V}'|}$, representing
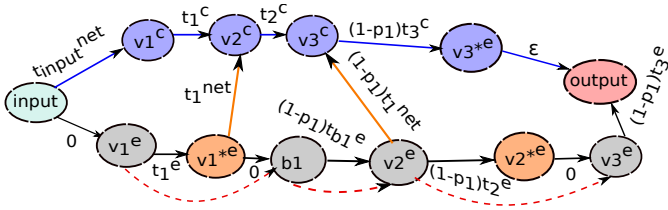
Fig. 3. Graph representation $\mathcal{G}'_{\text{BDNN}}$ of 3-layer BranchyNet.

the processing time to compute v_i at the edge (i.e., t_i^e) and at cloud (i.e., t_i^c), respectively. Figure 3 shows the graph $\mathcal{G}'_{\text{BDNN}}$ constructed based on a BranchyNet constituted by the main branch with three layers and one side branch inserted after the first layer of the main branch. In this figure, each gray and blue vertices represents a layer processed at the edge and cloud, respectively. The dashed red and blue links correspond to links of graphs $\mathcal{P}^e_{|\mathcal{V}'\cup\mathcal{B}|}$ and $\mathcal{P}^c_{|\mathcal{V}'|}$.

The graph $\mathcal{G}'_{\text{BDNN}}$ must model three possibilities: edge-only processing, cloud-only processing, and processing with partitioning. To model cloud-only and edge-only, we introduce two virtual vertices called $input$ and $output$ and then, we add links (input, v_1^c) and (input, v_1^e) into graph $\mathcal{G}'_{\text{BDNN}}$, illustrated by a blue and black links in Figure 3. Then, we assign weights to these links that are related to communication delay in cloud-only and edge-only processing. The weight $\omega_{(\text{input},v_i^c)}$ corresponds to communication time to upload a raw input sample to the cloud denoted by $t^{\text{net}}_{\text{input}}$ in cloud-only processing. In edge-only, the weight $\omega_{(\text{input},v_i^e)} = 0$ since there is not communication delay. Figure 3 shows that the path between $input$ and $output$ using only the red dashed and blue links computes the inference time for edge-only and cloud-only processing, respectively. To model the processing with partitioning in graph $\mathcal{G}'_{\text{BDNN}}$, for all $v_i^e \in \mathcal{V}'$, we introduce an auxiliary vertex $v_i^{*e} \in \mathcal{V}_A$ (i.e., orange vertices in Figure 3), where $\mathcal{V}_A$ is the set of auxiliary vertices. Then, we add a link (v_i^e, v_i^{*e}) and replace (v_i^e, v_{i+1}^e) (i.e., red dashed links in Figure 3) to (v_i^{*e}, v_{i+1}^e). In Figure 3, the red dashed links (v_1^e, b_1) are replaced to the black links (v_1^e, v_1^{*e}) and (v_1^{*e}, b_1).

To model the communication between edge and cloud, we add a link, whose the weight corresponds to the communication time to send output data of partitioning layer $v_s \in \mathcal{V}'$ placed at edge to $v_{s+1} \in \mathcal{V}_c$ placed at cloud denoted by t_s^{net}. In this figure, the orange link, such as (v_1^{*e}, v_2^e), represents the communication between edge side and cloud. To avoid ambiguity in the choice of the shortest path when the probability $p_Y(k) = 1$, we add a virtual vertex v_3^{*c} as successor of v_3^c and predecessor of vertex $output$. Then, we assign the weight ϵ to the link $(v_3^{*c}, output)$. The weight ϵ must be a very small value, to not interfere with the result of the shortest path problem. If the probability $p_Y(k) = 0$, which means the probability that any sample is classified by side branch b_1, the graph $\mathcal{G}'_{\text{BDNN}}$ represents a regular DNN.

The weights are assigned to links of $\mathcal{G}'_{\text{BDNN}}$ as follows:

$$\omega_{(v_i,v_j)} = \begin{cases} t_i^e, & \text{if } v_i \in \mathcal{V}_e \\ t_i^c, & \text{if } v_i \in \mathcal{V}_c \\ t_i^{\text{net}}, & \text{if } (v_i \in \mathcal{V}_A,\ v_j \in \mathcal{V}_c) \vee (v_i = \text{input}) \\ \epsilon, & \text{if } v_i \in \mathcal{V}_c,\ v_j = \text{output}, \\ 0, & \text{if } v_i \in \mathcal{V}_A,\ v_j \in \mathcal{V}_e \end{cases} \tag{7}$$

To model the expected value of inference time in a BranchyNet, the weights $\omega^{\text{BDNN}}_{(v_i,v_j)}$ assigned to the links in $\mathcal{G}'_{\text{BDNN}}$ is weighted by the probability $p_Y(k)$ that the sample is classified at side branch b_k. Thereby, as higher the probability that a sample is classified at the side branch, less significant are the weights of links after the side branch. Thus, the weights assigned to links in a Branchynet is given by

$$\omega^{\text{BDNN}}_{(v_i,v_j)}(k) = p_Y(k)\omega_{(v_i,v_j)}. \tag{8}$$

Our goal is to determine the path with the minimum cost that connects the virtual vertices $input$ and $output$. The cost of the path is defined as the sum of the weights associated with the links in graph $\mathcal{G}'_{\text{BDNN}}$. At this point, we show the equivalency between the BranchyNet partitioning problem and the shortest path problem. Given two vertices, the shortest path problem finds a forward path that connects these two vertices with minimum cost. In this problem, these two vertices are $input$ and $output$. In Figure 3, if the layers of $\mathcal{V}_e$ are contained in the shortest path, it means that processing strategy is edge-only. In this case, the total cost correspondents to T_e, as shown in Equation 1. However, if all vertices $v_i \in \mathcal{V}_c$ is contained in the shortest path, it means cloud-only processing. The cost of shortest path is thus T_c, as defined in Equation 2. Otherwise, if the vertices in shortest path belongs to $\mathcal{V}_e$ as well as $\mathcal{V}_c$, then partitioning occurs. The total cost is given by Equation 6. For instance, in Figure 3, assuming the partitioning layer is vertex v_2, the shortest path is (input, v_1^e), (v_1^e, v_1^{*e}), (v_1^{*e}, b_1), (b_1, v_2^e), (v_2^e, v_2^{*e}), (v_2^e, v_3^c), (v_3^c, v_3^{*c}) and $(v_3^{*c}, output)$. Thus, the layers v_1 through v_2 are processed at the edge and belong to set $\mathcal{V}_e$ then the edge sends output data of v_2 to the cloud, which, in turn, processes the v_3 that belongs to set $\mathcal{V}_c$.

The shortest path problem is a well-known problem that can be solved in polynomial-time. In this work, Dijkstra's algorithm is used to find the shortest path with the computational complexity of $\mathcal{O}(m + n\log(n))$, where m and n are the number of links and vertices in $\mathcal{G}'_{\text{BDNN}}$, respectively.

VI. EVALUATION

The experiments present a sensitivity analysis evaluating the impacts of input parameters described in Section IV-C, such as t_i^c, t_i^e, B and $p_Y(k)$, in inference time, under different network bandwidth. Then, we analyze the partitioning layer choice under different processing capacities of the cloud server and edge device. To this end, we implement a BranchyNet called B-AlexNet [5]. The B-AlexNet is composed of a standard AlexNet architecture as the main branch with one side branch inserted after the first middle layer of the main branch.

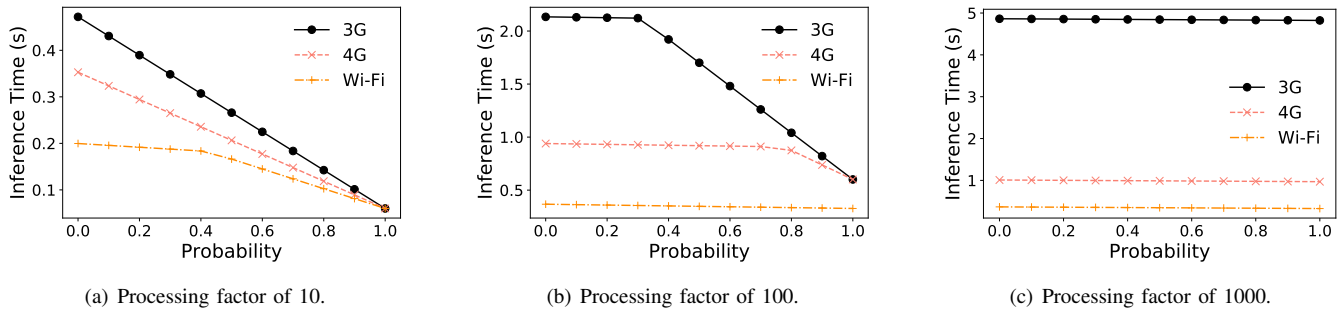(a) Processing factor of 10. (b) Processing factor of 100. (c) Processing factor of 1000.

Fig. 4. Inference time according to the probability of classifying a sample for different wireless technology and processing factors γ

The choice of only one side branch aims to simplify the experimental analysis, and the side branch position is chosen to avoid unnecessary processing in the edge. First, we have to obtain those input parameters. The parameters $\mathcal{G}'_{\text{BDNN}}$ and the output data size α_i for each layer are obtained after the definition of B-AlexNet as BranchyNet architecture. To obtain the communication time t_i^{net}, we consider that the edge device uses wireless technology to send data to the cloud. Besides, we assume that the bottleneck is the access network. We thus use average uplink rates of 1.10, 5.85, 18.80 Mbps, which corresponds to 3G, 4G, and Wi-Fi, respectively. These average uplink rates used in this work are based on the values presented in [6]. To obtain t_c, we measure the processing time for each layer of B-AlexNet, using Google Colaboratory.[3] Google Colaboratory is a cloud computing service for machine learning, equipped with Intel 2-core Xeon(R)@ 2.20GHz processor, 12 GB VRAM and GPU Tesla K80. We define the processing time at edge for each layer as $t_i^e = \gamma \cdot t_i^c$ where $\gamma \in \mathbb{N}_{>1}$ is a proportionality factor that indicates the ratio between processing time at cloud and edge. We use this factor to span different edge hardware types in our evaluation. For example, Jetson TX2[4] module developed by NVIDIA can represent an edge device equipped with high processing power and thus a low γ. On the other hand, Raspberry Pi[5] can represent a resource-constraint device, with a high γ.

Figure 4 shows the impact of probability in expected value of inference time, given by Equation 5, for three processing factors: 10, 100 and 1000. These results are obtained based on the solution of our optimization problem when varying the probability that a sample is classified at the side branch. For each processing factor, we present results for 3G, 4G and Wi-Fi. Figure 4 shows that, when the edge has high processing power, the probability has a severe impact on inference time. Also note that, for each processing time, the y-axis has a different scale, meaning that a high edge processing power leads to an overall low inference time.

Considering a given processing factor, Figures 4(a) and 4(b) show that networks with lower bandwidth are more affected by probability. For example, the 3G results in Figure 4(a)

show that the inference time reduces 87.27% if we compare a case where the probability is zero with the case where the probability is one. On the other hand, this difference is 82.98% and 70%, for 4G and Wi-Fi, respectively. Figure 4(a) also shows that, when the probability is one, all network technologies have the same inference time. In this case, this result is expected since all samples are classified at the side branch. Although the probability also affects inference time in Figure 4(b), we can note that, for each technology, the y-axis remains constant for a given range of probability. This explained since, as compared to the case of Figure 4(a), the edge has lower processing power. Hence, for low probabilities, the optimization problem chooses cloud-only processing since the major part of the samples are not classified on the side branch. As cloud-only processing does not have side branches, the inference time is not affected by the probability. After a given probability value, the problem begins to choose partitioning solutions, where the edge is involved, and thus the probability begins to affect the inference time. For example, for 3G, the inference time only starts to decrease when the probability is higher than 0.3. For 4G, this value is 0.8 since, as compared to 3G, the bandwidth is higher, and thus it is more interesting to send raw data to the cloud for a large probability range. Finally, the Wi-Fi results of Figure 4(b) shows that it is always interesting to perform cloud-only processing due to its high bandwidth. Figure 4(b) shows an extreme situation where the probability does not affect the inference time. This behavior happens because the edge has low processing power, and thus it is always interesting to perform cloud-only processing.

Using the same scenario of Figure 4, we vary the processing factor γ and analyze which layer the optimization problem chooses as the partitioning one. Figure 5(a) and 5(b) shows the chosen layer for different processing factors, when using 3G and 4G, respectively. Each curve represents a given probability of classifying the sample at a side branch. This behavior is expected since an edge with a lower processing power means that it is more interesting to process the layers in the cloud. Figure 5(a) shows that as the processing factor increases, the chosen partitioning layer moves toward to input layer. For example, assuming a probability $p = 0.8$, when γ changes from 500 to 600, Figure 5(a) shows that the partitioning layer changes from $conv1$ to $input$, which means cloud-only

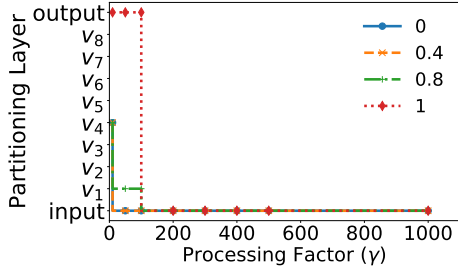processing. When comparing Figure 5(b) with Figure 5(a),
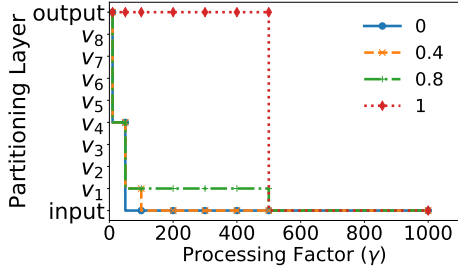
(a) Partitioning Layer according to factor γ with 3G.

(b) Partitioning Layer according to factor γ with 4G.

Fig. 5.   Partitioning layer for different processing factors.

we note that, for 4G, the problem starts to choose cloud-only processing for a lower processing factor (i.e., a higher edge processing power). This confirms the trend observed in Figure 4, where, for a higher average uplink rate, the problems tend to choose cloud-only processing. Finally, Figure 5 also confirms the behavior of Figure 4 where the probability influences the choice of the partitioning layer, thus impacting the inference time.

In practice, the probability of classifying an input image at a side branch is a parameter related to aspects inherent of input data that depends on numerous factors. One of these factors is image quality. To show that image quality affects the probability, and thus the partitioning decision, we train a B-AlexNet for image classification, using a cat-and-dog dataset [8]. This dataset is composed of images of dogs and cats in different environments without any distortion. Once trained, we apply a batch with 48 samples with different levels of Gaussian blur. This experiment implements Gaussian filters with dimensions 5, 15, 65 to represent images with low, intermediate, and high distortion, respectively. We use these images as samples to perform inferences to classify if an animal is a dog or a cat. Figure 6 shows the probability of classifying an input image according to the entropy threshold. This figure shows that as distortion level increases, the probability that a sample is classified at a side branch decreases. This is true since images with higher distortion levels tend to have a higher uncertainty in the inference, resulting in a lower probability that it is classified at a side branch.

VII. CONCLUSION

In this paper, we accelerate DNN inference, partitioning DNN between the edge device and cloud server, to minimize
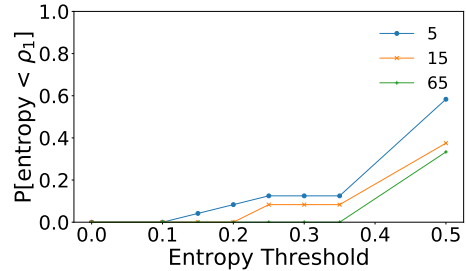

Fig. 6.   Probability of side branch classification under different distortion levels in B-AlexNet.

the inference time. Different from a traditional DNN, the BranchyNet has side branches that allow the inference to stop at the middle layers, which can reduce the inference time. Hence, to partition a BranchyNet we have to take into account the probability that inferences stop at a side branch. To address this problem, we model the expected value of inference time as a function of different factors, such as processing and communication delay and the probability of classification at a side branch. We also model the BranchyNet as a graph, showing that the minimization of inference time can be solved as the shortest path problem. Hence, this problem can be solved in polynomial-time, using Dijkstra's algorithm. We evaluate our problem using a sensitivity analysis where we vary the probability of side branch classification and the processing power in the edge. From the results obtained, we demonstrate that probability affects partitioning layer choice, hence impacts the inference time. Thus, estimating the probability allows improving the partitioning decision as network conditions and computational resources. Moreover, evaluations show that processing strategy changes according to the probability. Thus, this paper also introduces the probability as a factor to be considered in BranchyNet partitioning. As future work, our first goal is to extend our proposal to handle also DAG topology DNN. Moreover, we will investigate heuristics for side branch placement, to attempt also accuracy requirement.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[2] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.

[3] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," in *ACM Computer Architecture News (SIGARCH)*, vol. 45, no. 1, 2017, pp. 615–629.

[4] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are fpgas suitable for edge computing?" in *USENIX Workshop on Hot Topics on Edge Computing (HotEdge)*, 2018.

[5] S. Teerapittayanon, B. McDanel, and H.-T. Kung, "Branchynet: Fast inference via early exiting from deep neural networks," in *IEEE International Conference on Pattern Recognition (ICPR)*, 2016, pp. 2464–2469.

[6] C. Hu, W. Bao, D. Wang, and F. Liu, "Dynamic adaptive DNN surgery for inference acceleration on the edge," in *IEEE Conference on Computer Communications (INFOCOM)*, 2019, pp. 1423–1431.

[7] E. Li, L. Zeng, Z. Zhou, and X. Chen, "Edge ai: On-demand accelerating deep neural network inference via edge computing," *IEEE Transactions on Wireless Communications*, 2019.

[8] S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *IEEE International Conference on Quality of Multimedia Experience (QoMEX)*, 2016, pp. 1–6.