Towards Network-accelerated ML-based Distributed Computer Vision Systems

Hisham Siddique, Miguel Neves, Carson Kuzniar, Israat Haque
Faculty of Computer Science, Dalhousie University
Halifax, Nova Scotia
{hisham.siddique, mg478789, Carson.Kuzniar, israat}@dal.ca

***Abstract*—Computer vision is a crucial component in many modern applications (e.g., medical image analysis, environmental monitoring and self-driving cars). However, their stringent computational, latency and bandwidth requirements still pose a huge challenge to system architects, which must seek for alternatives to both the limited resources (e.g., low-end CPU) on client devices and the hurdles of moving data from clients to cloud/edge servers for analysis. In this work, we advocate for the usage of emerging programmable network devices to speed up ML-based computer vision tasks, particularly image classification, on resource constrained environments. To take the first step towards this new paradigm, we propose NetPixel, a framework that enables P4-programmable switches to classify images in real-time, accurately and at scale. We implemented a prototype of NetPixel in a software switch to show its feasibility and conducted a preliminary evaluation on widely adopted datasets. Our results show that NetPixel can classify images with an accuracy within 8% that of a server-based implementation even for shallow classifiers and low-resolution images.**

***Index Terms*—Distributed systems, machine learning, network protocol, P4, computer vision.**

I. Introduction

The proliferation of latency- and safety-critical IoT applications like AR/VR, surveillance, or autonomous vehicles and the evolution of 5G networking have fueled the post-cloud computing paradigm, edge computing [1][2][3]. Edge computing brings computing resources closer to end users for a better quality of experience due to faster response times. However, that comes at the price of reduced processing capacity on servers. Researchers have tried to alleviate resource limitations through: i) carefully distributing tasks among end and edge nodes [4]; or ii) augmenting edge servers with domain-specific accelerators (e.g., GPUs for ML operations) [5] in an attempt to not overwhelm the increasingly scarce edge fleet.

With the advent of software-defined networking and programmable network devices (e.g., programmable switches, smart NICs), it also becomes possible to accelerate or offload data processing to this hardware. For example, recent Tofino-based switches [6] can provide orders of magnitude higher throughput and lower latency than servers and thus emerge as a feasible solution to alleviate the edge burden. Previous work has exploited programmable switches to speed up a variety of applications including machine learning [7], caching [8], lock management [9], and even edge detection [10].

In this work, we advocate for using programmable network devices to accelerate a more complex computer vision application, namely, image classification. Image classification is a fundamental computer vision (CV) task needed in many latency- and safety-critical applications [11], [12], [13]. We argue that network devices sit between low-end appliances and edge servers and thus become a natural choice to offload or accelerate the intended processing. To support our claims and show the feasibility of in-network image classification, we present NetPixel, a framework that allows P4-enabled devices to classify images in real-time, accurately, and at scale.

We compare the performance of NetPixel against a baseline Python implementation over several real datasets (e.g., MNIST, ImageNet). Our results show that the performance degradation (mainly due to approximate computations) is marginal. In particular, the accuracy loss is below 8% in the worst case; these results help to pave the way for offloading or accelerating image/video-based latency-critical edge applications to on-path network devices. To summarize, our contributions through this work include:

- Designing NetPixel, a novel *system* that allows accurate image classification on a programmable network device for latency-critical applications.
- Designing a *protocol* for sending images as chunks through multiple packets to allow efficient feature calculation and subsequent classification.
- Evaluating our system against a baseline server implementation and demonstrating its *competitive accuracy*.

Organization. The rest of this paper is organized as follows: in Section II, we provide the necessary background on PISA architecture and the P4 programming language upon which NetPixel has been developed. The same section also delves into the motivation behind NetPixel alongside some of its implementation challenges. Section III presents the description of NetPixel's design. NetPixel is then evaluated against a baseline Python implementation using three different image datasets in Section IV. We discuss works closely related to ours in Section VI followed by concluding remarks in Section VII.

II. Background and motivation

This section provides the necessary background to understand the proposed system. We also outline the motivation and challenges for designing NetPixel.

A. Computer vision and distributed systems

Computer vision constitutes a variety of tasks that help machines gain higher-level understanding of image data or their sequences (i.e., videos). Among the most common computer vision tasks, one can find image classification, segmentation and object detection [14]. In this work, we focus on the former as it is also a basis for the other tasks. Computer vision-based systems have drastically improved their capabilities over the last decade mostly thanks to the adoption of machine learning algorithms (e.g., decision trees, deep neural networks) [15], [16], [17]. However, the ever increasing computational demand of these algorithms (currently in the order of $O(n^5)$) [18] has also led to the need of (at least partially) offloading image data processing from low-end client devices to more powerful cloud/edge servers. As a result, system architects strike to find a balance between the latency and bandwidth requirements of current computer vision applications and the constrained resources on mobile and IoT gadgets.

B. Motivating scenarios

Next, we outline a few driving scenarios to push image classification towards emerging programmable network devices as an efficient solution to the trade-off faced by computer vision system architects.

Real-time image recognition. Timely image recognition is a critical task for applications such as AR/VR, self-driving cars and smart factories [10]. However, executing heavy image processing tasks on low-end devices (e.g., an electric car or a robotic arm) remains difficult today due to their processing and energy constraints. Although cloud/edge servers could offer better capacities, they usually incur unacceptably higher network latency. In this context, programmable switches can offer a high-speed alternative that is also less hops away.

Image/frame filtering. Camera deployments (e.g., for surveillance, traffic or environmental monitoring) are becoming ubiquitous especially in urban areas [19]. However, the growing number of always-on cameras can collectively generate hundreds of gigabytes of data every second, which easily overload a shared network infrastructure [19], [11]. An intuitive alternative to this problem is filtering out images/frames that do not contain relevant information (e.g., performing classification near where the images are produced) to avoid unnecessary transfers [20]. Such pruning can also reduce the high compute load on edge servers, enabling more video feeds to be processed on these boxes or reducing the amount of resources to be provisioned. In this sense, programmable network devices are a sweet spot for image/frame filtering as they sit in-between low-end IoT cameras and edge servers and can provide up to terabits per second throughput.

C. Programmable switches

NetPixel has been developed targeting recent programmable network elements, e.g., switches and SmartNICs. In-network computing has become popular in recent years, influenced by advancements in networking device capabilities and the broad paradigm of computation at the edge of the network.

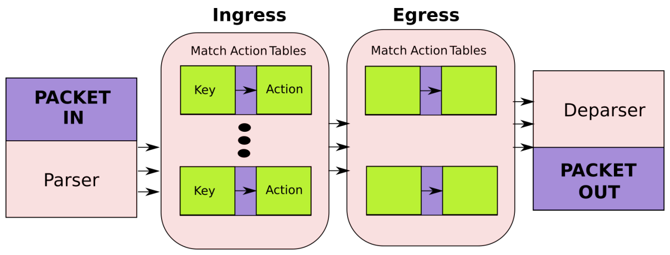

Fig. 1: PISA architecture model.

The standard *de facto* programming language for networking devices is P4 [21]. The P4 language is based on the *Protocol Independent Switch Architecture (PISA)* [22], and contains constructs to facilitate the programming of packet processing tasks (e.g., parsers, match-action tables, headers and metadata). Figure 1 summarizes the PISA architecture model. PISA-based devices include multiple packet processing pipelines. Packets traditionally flow sequentially through the ingress pipeline, followed by the egress pipeline, and then forwarded back to the network. A pipeline accounts for the bulk of packet processing and contains one or more match-action tables.

As packets arrive at the switch, the parser extracts information from the header fields, including traditional physical, network, and transport layer headers as well as application-specified custom headers. The match-action tables are then used to match these header fields' values and perform a subsequent action set. Packet header values may also be modified or manipulated during ingress processing, alongside packet metadata and on-board registers, constructing a memory space that can span multiple pipeline stages. Depending on the application complexity, packets may be recirculated through designated ports, carrying information through rewritten header values.

D. Challenges and opportunities

While P4 supports simple mathematical and logical operations such as addition and xor, it is not possible to perform more complex ones (e.g., division and logarithm) using basic language constructs [23]. Also, P4 does not support floating-point arithmetic, which is inevitable in image classification tasks. We overcome these limitations by computing complex values as approximations and converting floating-point numbers into their fixed-point representation. Additional constraints may also apply depending on the target device. For example, hardware switches may impose restrictions on the number of bits that can be parsed from a packet to achieve line rates. For those cases, programmers may need custom solutions (e.g., recirculating a packet to inspect its payload).

III. NetPixel design

A. Overview

Figure 2 shows the overview of NetPixel, which consists of a gateway switch and a network controller. Next, we discuss each component in detail.

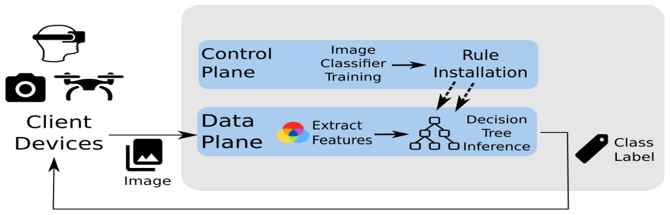

Fig. 2: NetPixel Overview

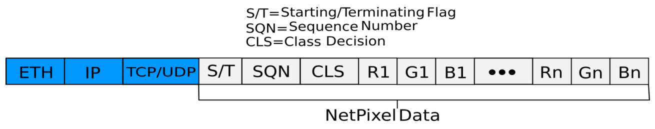

Fig. 3: Packet Format

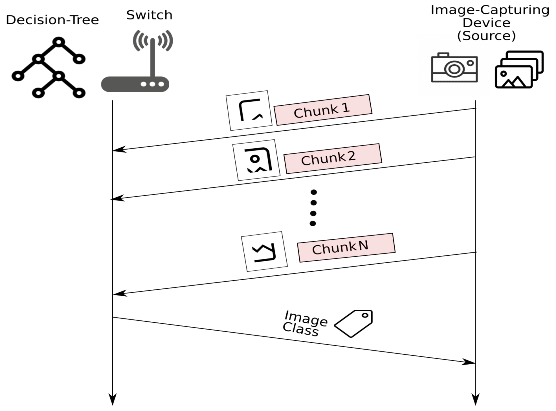

Fig. 4: NetPixel protocol

Switch. The switch is responsible for extracting pixel information stored in each packet, calculating the various features that will be used for image classification, and effectively classifying images themselves. These pixels are sent over the network as packets, based on the protocol described in Section III-B. Squared chunks (of n pixels) are created, where each chunk is sent in a separate packet[1].

As we discuss in Section IV-E, the size of the chunk impacts NetPixel's performance. Based on our experimental findings, we select 3x3 as the preferred chunk size for NetPixel's design. This is the chunk size used throughout the evaluations performed in this paper. Our selection is derived from the slightly better accuracy provided by this size as well as the reduction in per-packet calculations performed on the switch, relative to larger sizes. However, network constraints (e.g., packets-per-second throughput is limited by hardware) may require clients to send larger chunk sizes in order to send images faster (i.e., using fewer packets).

Once pixel values are extracted from packets, the RGB components of each pixel (each component is an 8-bit value) are used to calculate the various features required to classify an image. Based on the literature [24], we have selected seven main features: the number of edge segments and distinct colors, the contrast and average brightness, and the ratios of low, mid and high intensity pixels in the image. Each feature is discussed in detail in Section III-C.

Lastly, NetPixel applies the calculated features to a classifier that labels the image and sends the result back to the client device. Note that the image data can be either discarded or sent to an edge server for further processing. We have selected decision trees as image classifiers due to their intuitive mapping to a match-action pipeline. However, our approach is orthogonal to the type of classifier and could be extended to support different algorithms (e.g., SVM, K-means, quantized neural networks [23]. We leave the investigation of NetPixel performance under different classifiers as future work. Section III-D gives more details on how we structure our decision tree.

[1]Client devices can apply a downscaling technique (e.g., discarding a few pixels) if an image size is not a multiple of n.

Controller. The core function of NetPixel is dependent on the rules of the decision tree. The network controller is responsible for the training of the classifier model, based on the CART algorithm [25], and installing the rules pertaining to the model on the switch. The controller must read and convert the rules to entries for the match action table. These entries contain the feature values as keys and subsequent class decision for that particular combination. Once these rules are installed on the switch, the latter may begin its inference tasks.

B. *Network protocol*

NetPixel has its own protocol for transmitting images to the switch, which works at the application layer (on top of TCP/UDP). As can be seen in Figure 4, the image is sent as squared chunks of n pixels, describing the whole image. These packets contain the color information in the form of RGB values, for the pixels in that chunk. This is depicted in Figure 3, which also shows the other header fields of the NetPixel protocol. `S/T` are flags to indicate start and terminating packets (i.e., the first and last packet containing data associated with the same picture); `SQN` stands for sequence number and can be used for reliable transmission in case of UDP packets; `CLS` stores the final class decision (i.e., a label) once the image is classified. This field is empty in all packets except for the terminating one, which carries the class decision back to the client device[2]. Finally, `R1`, `G1`, `B1`, ..., `Gn`, `Bn` is the sequence of n pixels represented by their RGB components. The chunks begin at the top-left corner of the image and continue in a horizontal fashion for the first row of chunks, followed by the next row until the whole image has been transmitted. NetPixel is designed to perform scale invariant to image sizes and thus does not require a fixed number of pixels or chunks. Once all chunks have been received by the switch, inference is performed using the decision tree implemented in P4. The resulting class decision is then written on to the final packet in the appropriate field. This packet is then retransmitted back

[2]We also envision NetPixel to be used as part of other applications (e.g., frame counting or filtering). In this case, terminating packets could be sent to other devices such as an edge server or even discarded.

Feature	Formula
Number of colors	$\|C\|$
Ratio of Pixels with low intensity	$(\sum_{i=1}^{s} x_i)/s : 0 < I(x_i) < 85$
Ratio of Pixels with mid intensity	$(\sum_{i=1}^{s} x_i)/s : 86 < I(x_i) < 171$
Ratio of Pixels with high intensity	$(\sum_{i=1}^{s} x_i)/s : 172 < I(x_i) < 255$
Contrast	$(I_{max} - I_{min})/(I_{max} + I_{min})$
Avg brightness	$(\sum_{i=1}^{s} I(x_i))/s$
Number of edge segments	$\sum_{j=2}^{h} 1 : L_j L_{j-1} < 0$

TABLE I: Supported features. Symbols: x_i - a pixel; $I(x)$ - intensity of pixel x; I_{max}, I_{min} - highest and lowest intensity pixels, respectively; L_j - value of Laplacian filter when applied to chunk j; s - image size; h - number of chunks in an image; C - set of distinct colors in an image.

to the source device, which then utilizes the class decision according to its application requirements.

C. Image features

This section describes the features NetPixel uses to classify an image, which are outlined in Table I. NetPixel applies the same operations to all pixels in a chunk. Total number of colors is calculated using the RGB components of a pixel, while the rest of the features assume gray-scale intensities. NetPixel converts pixel values from RGB to gray-scale according to Equation 1, where I is the resulting intensity value.

$$I = 0.299 * R + 0.587 * G + 0.114 * B \tag{1}$$

Number of colors. This feature describes the number of distinct colors present in an image. To calculate it, NetPixel maintains both a bloom filter to keep track of colors previously seen and a counter to count new ones. The bloom filter is indexed by a hash of the RGB components from a pixel while entries are single-bit values indicating the presence of a color. In the worst case, the number of entries should be large enough to accommodate hashes for all possible combinations of RGB values without collisions, which is impractical as the RGB palette has more than 16M colors (∼2MB of memory for calculating number of colors of a single image). We overcome that issue with three observations: i) the usage of multiple hash functions can decrease the number of filter entries while keeping the same accuracy level [26]; ii) in practice, most of the images contain less than 1M different colors[3]; and iii) ML-based classifiers can afford small inaccuracies on the feature values while still working properly. With those observations on hand, we were able to reduce filter sizes to less than 1M entries (∼120KB of memory per image).

Ratio of pixels with low, mid and high intensity. By definition, intensity refers to the pixel values in the gray-scale colorspace (i.e., a single value ranging from 0 to 255 and calculated based on RGB components according to Equation 1). NetPixel divides the gray-scale colorspace into three ranges: 0-85 for low, 86-170 for mid, and 171-255 for high-intensity pixels, and counts the number of pixels in each range. NetPixel stores a counter for each of these three features. All three counters are then divided by the image size once the terminating packet is received to obtain the desired ratios.

Contrast and average brightness. The contrast of an image entails the differences between its low and high intensity pixels, and is calculated as $(I_{max} - I_{min})/(I_{max} + I_{min})$, where I_{max} and I_{min} represent the highest and lowest intensities among all pixels. Overall, high contrast images have a large range of intensities (i.e., large difference between I_{min} and I_{max}). NetPixel compares each pixel intensity with current minimum and maximum and updates these values if required. It then calculates contrast once all pixels have been processed. Average brightness, on its turn, refers to the average intensity of all image pixels. NetPixel keeps a counter to accumulate pixel intensities and divides it by the image size to compute average brightness. The rolling values (three in total) required for these two features are stored on switch registers, and the final calculations for the feature values performed upon the arrival of the terminating packet.

Number of edge segments. NetPixel uses a Laplacian filter to calculate the number of edge segments in an image. The filter, whose definition depends on its size [27], has the same dimension as the image chunk and works by looking for differences in pixel intensities among chunks. For that, NetPixel compares the Laplacian filter value from two consecutive packets looking for zero-crossings (i.e., transitions from positive to negative values or vice-versa), and counts the number of crossings among all chunks. This approach has two sources of inaccuracies. First, as we only check Laplacian values between a chunk and its predecessor, it is not possible to detect horizontal edge segments (or differences in pixel intensities between consecutive vertical chunks). Second, we only apply the Laplacian filter on different chunks, meaning its sliding window is always equal to the filter (or chunk) dimension. As a result, we may not be able to detect segments that are fully contained inside a chunk. We tested the decrease in accuracy caused by these two simplifications for the datasets used in our experiments (see Section IV) and found it to be smaller than 5% in the worst case though.

D. Pipeline structure

The complete pipeline of NetPixel, as shown in Figure 5 in terms of how each packet is used to build up the features for each image, consists of a number of stages. Each of these stages is described in detail below.

First, the red, green and blue components of each pixel are combined and used in a hash function to determine whether they represent a new color or not. A bloom filter is used to store the set of previously seen colors. Next, each of the RGB pixels is converted into its equivalent gray-scale value using Equation 1. Each gray-scale value is then compared to the aforementioned ranges and the appropriate counter is incremented to compute intensity ratios. Next, the gray-scale values are compared to the current minimum and maximum to check whether these values need to be updated as part of the contrast calculation. The intensity values are also added to the

[3]We observed that empirically for the datasets used in our experiments.

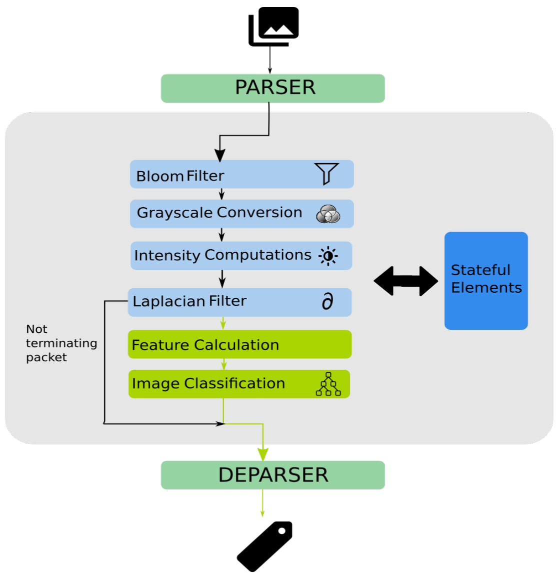

Fig. 5: NetPixel pipeline.

Dataset	Image size	Training images	# of labels
MNIST	28x28x1	60000	10
CalTech101	Variable	9200	101
CalTech256	Variable	30000	256
ImageNet	Variable	20000	100

TABLE II: Evaluated datasets.

Dataset	DT-Python	NetPixel
MNIST	92.45%	85.00%
CalTech101	96.50%	92.78%
CalTech256	91.11%	87.28%
ImageNet	90.36%	86.73%

TABLE III: Classification accuracy for different datasets.

summation of all intensities, which will later be used for the average brightness feature. These are all done for each pixel in the chunk. Finally, the Laplacian operator, L, is applied to the gray-scale pixel values of each chunk and, if the packet is not a terminating one, it is ready to be deparsed (or discarded). In case it is a terminating packet, values from the registers are used to consolidate the final values for the desired features.

Once all features are calculated, their values are used as inputs to the image classifier. NetPixel uses a decision tree to classify images, which is deployed as a single match-action table matching all features described in Section III-C. Resulting actions then set an appropriate class (or label) according to the matched rule. While different designs are possible (e.g., deploying each feature [23] or tree level [7] in a separate table), as we show in our experiments the number of rules required for achieving a decent accuracy when classifying an image is relatively low compared to the number of rules in current packet classifiers (which can easily contain hundreds of thousands of rules), and thus could be considered as an alternative to reduce the resource consumption (e.g., number of match-action stages used) particularly on hardware devices.

Each rule in the NetPixel classifier encodes a set of constraints on one or more features, representing a path (from root to leaf) in the decision tree. If the same feature is constrained more than once in a path, the NetPixel controller translates the multiple constraints into a single one representing their intersection. Likewise, if a feature is not constrained, the NetPixel controller sets its range to [0, T], where T is the largest value the feature can assume. Currently, NetPixel uses range rules to classify images, though these can be converted into ternary or LPM rules by breaking a range into multiple entries for devices that do not support range-based matches [28], [23]. We leave the investigation of classifier optimizations and alternative classifier designs as future work.

IV. Evaluation

A. Setup

We have implemented a prototype of NetPixel including three components: i) a data plane written in P4 (around 1.8K lines of code), targeting the v1model architecture (we use the BMv2 software switch in our experiments); ii) a Python-based network controller (around 300 lines of code), which trains the ML classifier and converts decision rules into the switch pipeline configuration; and iii) a client library (also written in Python) that receives an image and encapsulates it into NetPixel packets. Our code is available at [29]. Both the network controller and the client library convert all floating-point values into their fixed-point representation (we use 28 and 4 bits for the integer and fractional parts, respectively) before sending them to the switch[4]. For the evaluation of our system, we use a host equipped with a 4-core Intel Core i5-7400 CPU and paired with 8GB of RAM, which runs our Python scripts and the BMv2 software switch. The same host was used to run the DT-Python baseline system against which NetPixel is compared. The main goal of our evaluation is to assess the feasibility and functionality of our design. We leave a thorough performance analysis (including detailed latency and throughput experiments) as part of our future work[5].

Datasets. We used four different datasets for evaluating NetPixel, whose attributes are detailed in Table II. MNIST [30] contains gray-scale images of handwritten digits. CalTech101 [31] and CalTech256 [32] contain variable size color images with 101 and 256 classes, respectively, while ImageNet [33] is a database of 14 million variable size images. There are around 21000 classes in this database, from which we have selected 100.

B. Overall performance

Table III presents the accuracy of NetPixel on the chosen datasets. The results are based on 10-fold cross-validation

[4]We assume all values are at most 32 bits long, though our implementation can be extended to support bigger values.

[5]We are currently porting NetPixel implementation to the Tofino architecture.

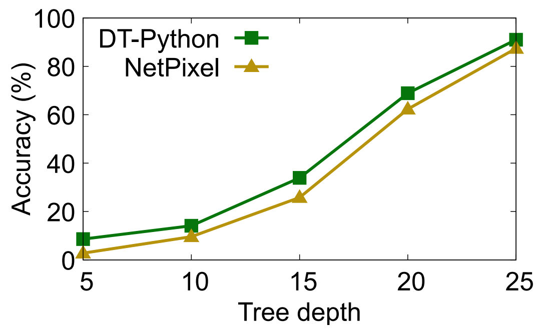

Fig. 6: Accuracy of NetPixel when varying tree depth

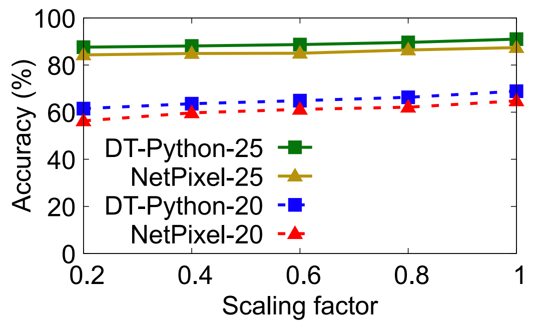

Fig. 8: Accuracy with different image scales

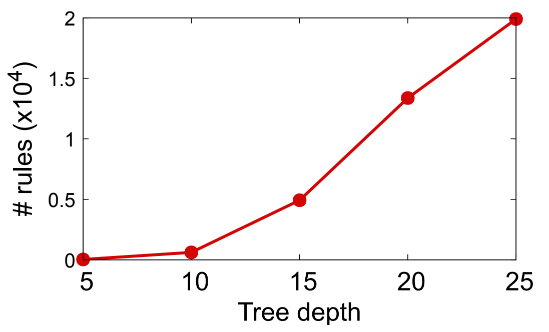

Fig. 7: Memory footprint in terms of decision tree rules

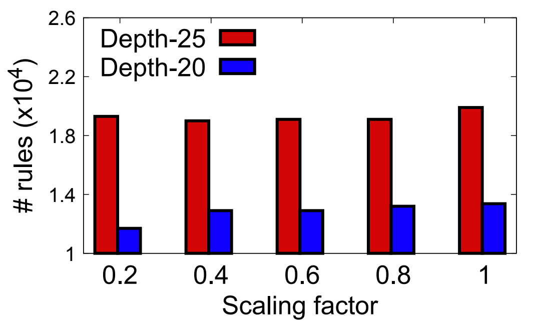

Fig. 9: Memory footprint with varying image scales

with a tree depth of 25. In general, we observe an accuracy discrepancy of 5% to 7.45% between NetPixel and the baseline. This result is expected as we consider an approximation to alleviate the limitation of P4 and PISA on supporting division and floating-point operations. Among all datasets, CalTech101 has the best accuracy while MNIST has the worst. The latter dataset has the lowest number of classes when compared to the others; however, the similarities in the features like the total number of colors, average brightness, and contrast impact the performance, i.e., the role of these features in the consolidation of decision boundaries diminish. By extension, it would be appropriate to assume that similar effects would be seen for datasets where classes contain a high level of resemblance to each other. On the other hand, CalTech101 and Caltech256 both have a wide range of classes that allows the chosen features to establish definitive decision boundaries among the classes. Finally, the image classes in ImageNet vary less compared to CalTech datasets resulting in a lower accuracy than CalTech256.

C. *Impact of tree depth*

In this evaluation, we use the Caltech256 dataset to observe the effects of tree depths on NetPixel's accuracy and memory footprint, which are shown in Fig. 6 and Fig. 7 respectively. We denote memory usage by the number of rules generated by the decision tree since each rule is stored on the switch as a match-action table entry. Overall, the higher the number of rules the larger the memory footprint. While the tree depth is predefined before classifier training, the number of nodes and consequently the number of rules within that depth may still change, as the experiments reflect. As expected, the accuracy increases with the increasing tree depth with the sharpest incline from depths 15 to 20. An accuracy of 80% or more requires a tree depth of around 23 for Caltech256 as more leaf nodes are required to achieve high accuracy. However, that demand impacts memory usage (Fig. 7). In particular, there is not a drastic change in the number of rules between depths 5 and 10; after that, the number of rules increases almost linearly as depth increases. Note that the increase is not exponential with the tree depth as the tree does not use the maximum depth in every branch because of having alternatives.

D. *Impact of image resolution*

We again use the Caltech256 dataset to observe NetPixel's sensitivity to image size and measure accuracy and memory usage (see Fig. 8 and Fig. 9). In order to observe the effect of varying image sizes, we scaled images on the client-side using a Bicubic interpolation before sending them to the classifying switch. Interestingly, the image size does not have a significant impact on the accuracy compared to the tree depth. We believe the chosen features are scale invariants that can consolidate decision boundaries even for smaller images. We observe a similar performance trend in the case of memory usage. In particular, the memory usage remains mostly unchanged with the varying image sizes for Depth-25, while we observe a 14% increase in memory usage when the scaling factor changes from 0.2 to 1 for Depth-20. This is due to more nodes being required for the larger image size within the maximum tree depth of 20, i.e., more rules are generated to establish decision boundaries between classes. However, for Depth-25, the nodes are enough

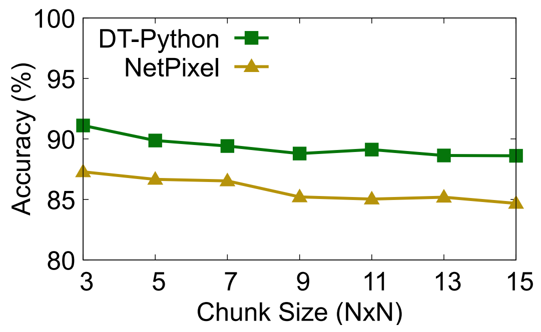

Fig. 10: Accuracy with varying chunk sizes

in number to establish definitive decision boundaries even with varying image sizes.

E. Impact of chunk size

We also evaluated the effect of different chunk sizes on the accuracy of NetPixel's classification. We use the CalTech256 dataset for this test. As can be seen in Figure 10, the accuracy slightly decreases beyond the standard 3x3 chunk-size we have used throughout this paper and the largest chunk size of 15x15 pixels. Chunk-sizes are mostly relevant to the *number of edges* feature, with larger chunk-sizes detecting fewer numbers of edges in an image. This creates a lower range of possible edges in an image and thus makes it harder for the classifer to establish definitive decision boundaries in some cases. However, since this affects a single feature only, the decrease in accuracy is not significant.

V. Discussion

In this section, we outline how NetPixel can be extended in several ways.

Hardware implementation. As part of our ongoing work, we are implementing NetPixel in a programmable ASIC-based switch (e.g., Intel Tofino [6]). In addition to the challenges described in Section II-D, Tofino-based switches also pose extra resource limitations such as a restricted number of TCAM entries and pipeline stages. To overcome these limitations, we consider applying feature importance analysis techniques [34] to simplify our ML model while keeping its accuracy high.

Neural network based ML models. NetPixel was originally designed based on decision trees due to their straightforward implementation in programmable network devices. However, deep neural network (DNN) based image classification algorithms (e.g., convolutional neural networks) have shown higher performance, i.e., accuracy. We plan to extend the current NetPixel design to accommodate the capabilities of neural networks such as convolutional neural networks. One alternative is converting a trained DNN into a decision tree or simplifying the NN architecture using binarization and pruning methods.

Other applications. Although we targeted image classification in this work, the overall principles of NetPixel, i.e., in-network computing applied to machine learning and image/video data, are not restricted to this task. Indeed, we envision extending our work to encompass other applications such as the ones involving object detection and image segmentation tasks.

VI. Related Work

In the following, we briefly outline and contrast existing work with NetPixel.

Image classification. The existing image classification designs mainly differ in feature selection, and image classification techniques [35]. Many solutions (e.g., [24], [36]) have adopted decision trees due to their simplicity, accuracy, and interpretability properties. Other works explore image classification on the Raspberry Pi as an edge device, either through various ML algorithms [37] or well-known deep learning frameworks [38].

Kong *et al.* [39] present a framework that constitutes face recognition and image classification taking advantage of an edge server through the use of multiple CNN frameworks. This system is used to classify mask-wearers in a pandemic environment. A similar method for the classification of crops was proposed in [40], running inference on a nearby server using the TensorFlow framework. Avgeris *et al.* [41] present a system for fire detection in rural areas, making use of an image classification system with data captured from low-powered IoT devices and/or cameras. However, this system makes considerable use of the resources at a cloud server. Despite the varied nature of these works, they do not delve into the potential for programmable network devices to deploy a scalable and real-time image classification framework like NetPixel.

In-network computing. Computation on networking devices (e.g., switches, smart NICs) has seen a massive surge with the advent of programmable networking hardware, enabling both the offload and acceleration of a number of applications [23], [8], [9]. Similar to NetPixel, Xiong *et al.* [23] and Busse-Grawitz *et al.* [7] propose implementing decision tree-based classifiers in programmable network devices. However, their focus is on networking applications (e.g., flow classification), and none of them explore the challenges intrinsic to image classification tasks (e.g., calculating image features). Glebke *et al.* [10] implement an in-network edge detection application using P4 and a smartNIC without considering image classification. NetPixel, with its low-latency framework, can be a viable pathway to performing tasks such as object detection and image classification at near-distance networking devices and further extend to other tasks.

VII. Conclusion

In conclusion, we have presented NetPixel, a novel in-network image classification system for latency-critical and bandwidth-hungry applications in edge computing. NetPixel infers a class based on extracted features and decision tree rules deployed on a programmable network device. We implemented a prototype of our system using the BMv2 software switch and tested it on a wide range of real images. The extensive evaluation results show that NetPixel offers competitive accuracy compared to a standard server-based implementation while can process images directly at the programmable networking hardware.

Acknowledgments. We thank the anonymous reviewers for their valuable feedback. This research is supported by CFI JELF and NSERC Discovery Grants.

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE internet of things journal*, vol. 3, no. 5, pp. 637–646, 2016.

[2] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, 2016.

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

[4] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 587–597, 2018.

[5] K. Karras, E. Pallis, G. Mastorakis, Y. Nikoloudakis, J. M. Batalla, C. X. Mavromoustakis, and E. Markakis, "A hardware acceleration platform for ai-based inference at the edge," *Circuits, Systems, and Signal Processing*, vol. 39, no. 2, pp. 1059–1070, Feb 2020. [Online]. Available: https://doi.org/10.1007/s00034-019-01226-7

[6] "Intel tofino 2 p4 programmability with more bandwidth," 2018, accessed, 2021-03-19. [Online]. Available: https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series.html

[7] C. Busse-Grawitz, R. Meier, A. Dietmüller, T. Bühler, and L. Vanbever, "pforest: In-network inference with random forests," *CoRR*, vol. abs/1909.05680, 2019.

[8] X. Jin, X. Li, H. Zhang, R. Soulé, J. Lee, N. Foster, C. Kim, and I. Stoica, "Netcache: Balancing key-value stores with fast in-network caching," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: ACM, 2017, p. 121136.

[9] Z. Yu, Y. Zhang, V. Braverman, M. Chowdhury, and X. Jin, "Netlock: Fast, centralized lock management using programmable switches," in *Proceedings of the ACM SIGCOMM*, ser. SIGCOMM '20. New York, NY, USA: ACM, 2020, p. 126138.

[10] R. Glebke, J. Krude, I. Kunze, J. Rüth, F. Senger, and K. Wehrle, "Towards executing computer vision functionality on programmable network devices," in *Proceedings of the 1st ACM CoNEXT Workshop on Emerging In-Network Computing Paradigms*, ser. ENCP '19. New York, NY, USA: ACM, 2019, p. 1520.

[11] C. Canel, T. Kim, G. Zhou, C. Li, H. Lim, D. G. Andersen, M. Kaminsky, and S. R. Dulloor, "Scaling video analytics on constrained edge nodes," *arXiv preprint arXiv:1905.13536*, 2019.

[12] M. Wang and W. Deng, "Deep face recognition: A survey," *Neurocomputing*, vol. 429, pp. 215–244, 2021. [Online]. Available: https://doi.org/10.1016/j.neucom.2020.10.081

[13] "Autonomous cars will generate more than 300 tb of data per year," 2017, accessed, 2021-03-19. [Online]. Available: https://www.tuxera.com/blog/autonomous-cars-300-tb-of-data-per-year/

[14] D. Forsyth and J. Ponce, *Computer Vision: A Modern Approach. (Second edition)*. Prentice Hall, Nov. 2011. [Online]. Available: https://hal.inria.fr/hal-01063327

[15] S. R. Safavian and D. Landgrebe, "A survey of decision tree classifier methodology," *IEEE transactions on systems, man, and cybernetics*, vol. 21, no. 3, pp. 660–674, 1991.

[16] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3642–3649.

[17] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in *Twenty-second international joint conference on artificial intelligence*, 2011.

[18] K. Fredenslund, "Computational complexity of neural networks." [Online]. Available: https://kasperfred.com/series/introduction-to-neural-networks/computational-complexity-of-neural-networks

[19] V. Kolar, I. T. Haque, V. P. Munishwar, and N. B. Abu-Ghazaleh, "Ctcv: Coordinated transport of correlated videos in smart camera networks," in *24th International Conference on Network Protocols (ICNP)*. IEEE, 2016.

[20] Y. Li, A. Padmanabhan, P. Zhao, Y. Wang, G. H. Xu, and R. Netravali, "Reducto: On-camera filtering for resource-efficient real-time video analytics," in *Proceedings of ACM SIGCOMM*, ser. SIGCOMM '20. New York, NY, USA: ACM, 2020, p. 359376.

[21] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, p. 8795, Jul. 2014.

[22] P. Bosshart, G. Gibb, H.-S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 99110. [Online]. Available: https://doi.org/10.1145/2486001.2486011

[23] Z. Xiong and N. Zilberman, "Do switches dream of machine learning? toward in-network classification," in *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, ser. HotNets '19. New York, NY, USA: ACM, 2019, p. 2533.

[24] P. Surynek and I. LuksovÃ¡, "Automated Classification of Bitmap Images using Decision Trees," *Polibits*, pp. 11 – 18, 12 2011.

[25] L. Brieman, J. Friedman, R. Olshen, C. Stone, D. Steinberg, and P. Colla, "Cart: Classification and regression trees," 1995.

[26] P. C. Dillinger and P. Manolios, "Bloom filters in probabilistic verification," in *International Conference on Formal Methods in Computer-Aided Design*. Springer, 2004, pp. 367–381.

[27] D. Marr and E. Hildreth, "Theory of edge detection," *Proceedings of the Royal Society B, London*, vol. 207, pp. 187–217, 1980.

[28] V. Demianiuk and K. Kogan, "How to deal with range-based packet classifiers," in *Proceedings of the 2019 ACM Symposium on SDN Research*, ser. SOSR '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 2935.

[29] "Netpixel," https://github.com/PINetDalhousie/netpixel.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories," in *2004 conference on computer vision and pattern recognition workshop*. IEEE, 2004, pp. 178–178.

[32] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," 2007.

[33] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[34] J. Cai, J. Luo, S. Wang, and S. Yang, "Feature selection in machine learning: A new perspective," *Neurocomputing*, vol. 300, pp. 70–79, 2018.

[35] D. Lu and Q. Weng, "A survey of image classification methods and techniques for improving classification performance," *International Journal of Remote Sensing*, vol. 28, no. 5, pp. 823–870, 2007.

[36] N. Frosst and G. Hinton, "Distilling a neural network into a soft decision tree," *arXiv preprint arXiv:1711.09784*, 2017.

[37] S. Abdel Magid, F. Petrini, and B. Dezfouli, "Image classification on iot edge devices: profiling and modeling," *Cluster Computing*, vol. 23, no. 2, pp. 1025–1043, Jun 2020. [Online]. Available: https://doi.org/10.1007/s10586-019-02971-9

[38] E. Kristiani, C.-T. Yang, and C.-Y. Huang, "isec: An optimized deep learning model for image classification on edge computing," *IEEE Access*, vol. 8, pp. 27 267–27 276, 2020.

[39] X. Kong, K. Wang, S. Wang, X. Wang, X. Jiang, Y. Guo, G. Shen, X. Chen, and Q. Ni, "Real-time mask identification for covid-19: An edge computing-based deep learning framework," *IEEE Internet of Things Journal*, pp. 1–1, 2021.

[40] M. D. Yang, H. H. Tseng, Y. C. Hsu, and W. C. Tseng, "Real-time crop classification using edge computing and deep learning," in *2020 IEEE 17th Annual Consumer Communications Networking Conference (CCNC)*, 2020, pp. 1–4.

[41] M. Avgeris, D. Spatharakis, D. Dechouniotis, N. Kalatzis, I. Roussaki, and S. Papavassiliou, "Where there is fire there is smoke: A scalable edge computing framework for early fire detection," *Sensors*, vol. 19, no. 3, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/3/639