A Crowd-based Image Learning Framework using Edge Computing for Smart City Applications

George Constantinou, Abdullah Alfarrarjeh, Seon Ho Kim, Cyrus Shahabi
IMSC, University of Southern California
Los Angeles, CA 90089, USA
{*gconstan, alfarrar, seonkim, shahabi*}*@usc.edu*

Gowri Sankar Ramachandran, Bhaskar Krishnamachari
CCI, University of Southern California
Los Angeles, CA 90089, USA
{*gsramach, bkrishna*}*@usc.edu*

***Abstract*—Smart city applications covering a wide area such as traffic monitoring and pothole detection are gradually adopting more image machine learning algorithms utilizing ubiquitous camera sensors. To support such applications, an edge computing paradigm focuses on processing large amount of multimedia data at the edge to offload processing cost and reduce long-distance traffic and latency. However, existing edge computing approaches rely on pre-trained static models and are limited in supporting diverse classes of edge devices as well as learning models to support them. This research proposes a novel crowd-based learning framework which allows edge devices with diverse resource capabilities to perform machine learning towards the realization of image-based smart city applications. The intelligent retraining algorithm allows sharing key visual features to achieve a higher accuracy based on the temporal and geospatial uniqueness. Our evaluation shows the trade-off between accuracy and the resource constraints of the edge devices, while the model re-sizing option enables running machine learning models on edge devices with high flexibility.**

***Keywords*-Edge Devices; Smart Cities; Machine Learning; Crowdsourcing; Crowd-based Learning**

I. Introduction

Edge computing (EC) paradigm focuses on processing information close to the data source to reduce long-distance traffic and latency. Applications in the context of smart cities are expected to benefit from the EC due to the increased deployment of smart sensors such as IoT devices [1] and CCTV cameras [2], [3]. Such deployments provide application stakeholders with the ability to recognize objects and events of interests by processing media content close to the data source. Besides, the platforms such as MediaQ [4] provide crowdsourcing of media content, wherein the users can collect and share media content in a centralized media repository for creating a geospatial media library [5]–[7] for smart cities. Crowdsourced image data is typically used to improve the algorithms that detect potholes [3], graffiti [8], [9], and traffic flow [10], to name a few. In all these applications, the edge devices such as smartphones, CCTV, drones, police cars or Raspberry PIs are equipped with high-resolution cameras, can capture videos with high frame rate (ranging from 30FPS to 60FPS). Recent advances in neural networks along with the processing power of edge devices have made it possible to run deep learning models locally on edge device, which provides a great potential to utilize lots of mobile edge devices in public domain for urban applications.

However, there exist challenges. Heterogeneous devices exhibit different capabilities concerning processing power, memory, communication bandwidth, and battery capacity, which influences the inference time of machine learning (ML) algorithms and subsequently the performance of applications. To demonstrate it, we performed object detection using identical software developed using YOLO [11] library on Raspberry Pi, Desktop computer without GPU, and a server machine with GPU to study the performance of various classes of platforms. As shown in Table I, an edge server with GPU outperforms the other platforms, even though all platforms are capable of performing object detection. Although the devices are capable of detecting objects using the given model, the processing time depends on the resource capacity of the device which may limit the practical application of the model.

Table I: Object Detection Time on Various Platforms

Platform	CPU (GHz)	GPU	Processing Time (s)
Raspberry Pi	0.6	No	360
PC	1.596	No	29
Low-end server	1.386	Yes	0.2

One fundamental challenge is the uncertainty and heterogeneity of participating edge devices. Most of the existing frameworks train a single model on the server side and then distribute it to edge devices [11]. This approach works well in the case when the model owner is well aware of the processing and communication capabilities of the participating edge devices. However, in many cases, the model owner may not be the edge device owner, which is especially true for crowd-based and incentive-driven smart city applications [1]; thus the processing power of edge devices is unknown. Having a single model for a diverse set of edge devices with different processing capabilities introduces new limitations because high-end devices can run a more complex version of the model which potentially can provide more accurate results, or on the other side of the spectrum, a low-end device can run a simpler version of the model much faster but with less accurate results. In addition, edge devices are generating new multimedia data, which can be used to retrain the model for fine-tuning the inference accuracy. Existing frameworks lack appropriate mechanisms to integrate big datasets from the edge devices themselves into the retraining process for quality and performance enhancement.

To address the above challenges, we integrate crowdsourcing, ML, and EC in one framework dubbed as "*crowd-based learning framework using edge computing*". The framework enhances the ML algorithm of interest by utilizing the crowdsourced data collected by edge devices. One straightforward approach for crowd-based learning framework is to exploit all of the crowdsourced data; however such mechanism is infeasible in image ML due to network constraints, the high computational requirements at the server and the potential degradation of the accuracy of learning model[1]. Consequently, we propose a distributed selection algorithm that prioritizes the crowdsourced data, transfers only a selected subset of data, and still efficiently upgrades the learning model at the server end.

The proposed generic crowd-based learning framework for EC applications makes the following key contributions:

- An EC framework based on the client-server architecture with built-in support for dispatching ML models to edge devices based on the resource capacity and the bandwidth availability.
- Classification, creation, and maintenance of image ML models for a wide array of edge devices with heterogeneous resource and bandwidth capacities.
- Model enhancement algorithm which takes into account the geospatial and temporal properties of the newly collected data at edge devices to intelligently decide when to retrain the model.

II. Related Work

This section reviews the literature and explains how our approach moves the state-of-the-art efforts in the EC domain.

Balan *et al.* [12] propose the concept of cyber foraging for resource-constrained mobile platforms. The idea of cyber foraging refers to the offloading of computation and storage-heavy tasks to nearest resource-rich servers. Our approach applies the principles of cyber foraging to resource-constrained mobile and IoT platforms. Flinn *et al.* [13] contribute Spectra for offloading computation based on the resource availability of the neighboring servers. Spectra deals with the trade-off among performance, energy consumption, and the Quality-of-Service requirements. Unlike Spectra, our work focuses on reducing the bandwidth requirement when offloading media streams. Pillai *et al.* [14] introduce Sprout, which provides a framework for gesture-driven gaming applications, which parallelizes the media processing by distributing the computations to nearby servers with a focus on increasing the throughput while minimizing the latency. Sprout concentrates on offloading challenges in a static environment while our work concentrates on dynamic applications with evolving processing and storage requirements.

[1]Several factors might cause the degradation of the accuracy of learning model (e.g., the crowdsourced data might be of low-quality, or the new data might change the distribution of the training dataset which may generate a biased learning model).

Iida *et al.* [15] present GPUrpc, which is a remote procedure call extension for offloading computation to powerful Graphics Processing Units (GPU). GPUrpc is tested on wired networks for data mining and image processing application, whereas our approach is in the context of wireless networks with severe resource constraints. Ra *et al.* [16] introduce Odessa, a lightweight run-time for mobile devices, which offloads computation-intensive tasks to Internet-connected servers and parallelize the computation tasks in multi-core processing platforms. Odessa continuously monitors the resource availability of the platform, and adaptively offload computations and parallelize computations to improve accuracy and responsiveness. Saurez *et al.* [17] contribute a programming infrastructure to efficiently off-load computations using APIs while providing support for dynamically adding or removing computation platforms in the fog.

EC for IoT: Due to the resource-constrained nature of IoT platforms, some solutions have been proposed to offload the computation processes to an edge server [18]–[21]. An offloading approach is introduced for environmental monitoring application, in which the computations are transformed into a lightweight process and offloaded to an edge server which is close to the data sources [18]. M. Satyanarayanan *et al.* [19] highlight increasing use of video cameras in smart spaces and contribute GigaSight for searching media data at Internet-scale using cloudlets, which sits between the mobile phones and cloud, for bandwidth optimization. Renart *et al.* [20] propose an edge processing framework for the IoT that schedules the tasks based on the location of the source device and the availability of computation resources.

Crowdsourcing, Crowd Learning, and Crowd-based Learning: Crowdsourcing [22] is the practice of engaging a "crowd" or group of people for performing a task (e.g., translating an article and labeling an image). Due to the ubiquity of smartphones, crowdsourcing has been extended to spatial crowdsourcing [23]–[25] which requires workers to be physically present at the location of a task for performing it (e.g., asking a journalist to record an event occurring at a certain place). With the increasing commercial adoption of both crowdsourcing (e.g., Amazon Mechanical Turk [26]) and spatial crowdsourcing (e.g., TaskRabbit [27]) in the industry, researchers started investigating incentive-based learning algorithms to maximize the total reward of workers exploiting the historical activities. This problem is known as crowd learning [28], [29]. In this paper, we focus on integrating crowdsourcing, ML, and EC seamlessly. In

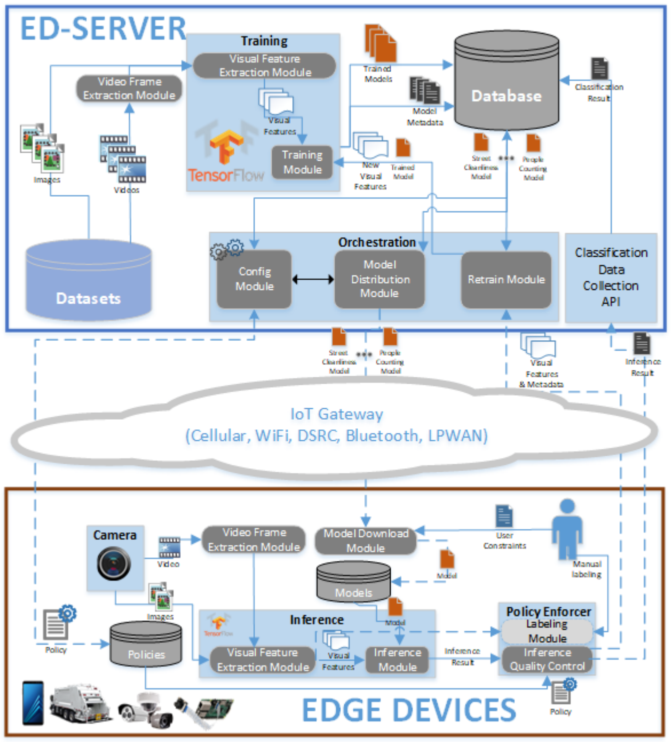


Figure 1: The Design of the Proposed Framework

particular, crowd-based learning refers to the mechanism of evolving a supervised ML algorithm (e.g., a learning model for classification or object detection) by exploiting the power of crowdsourced data (i.e., data collected by a crowd) and the crowd feedback.

III. Proposed Framework

Figure 1 presents our proposed framework in two layers: 1) the devices at the edge (denoted as Edge Devices) and 2) the edge device server (denoted as ED-Server). The communication between the edge server and the edge device is realized through conventional technologies such as cellular, WiFi, DSRC, and LPWAN.

A. Edge Devices

An edge device is capable of collecting media content from a camera, pre-processing the data, performing local inference using a ML model provided by the edge server, extracting visual features vectors (VFVs) and metadata based on the policy set by the edge server, labeling, and transmitting the metadata and the VFVs to the ED-Server. In the framework, an edge device joins the framework and downloads a suitable model based on its resource capacity and the accuracy goals from the edge server. The central edge server maintains a list of models capable of running on a wide variety of edge devices. Conventional approaches use a pre-trained model and do not continuously upgrade the model for accuracy and performance because of the bandwidth and energy requirements associated with the transmission of the entire media content to the edge server for enhancing the accuracy of the model. Example applications in the context of smart cities we are currently working with the City of Los Angeles include: the detection of waste dumped objects [2], graffiti [8], road damages and potholes [3], traffic flow [10], and natural disasters [30]. Users and devices participating in reporting the presence of graffiti, potholes, and garbage on streets have been relying on a static model provided by the edge server. To increase the detection accuracy and the resource efficiency, we introduce a feature detection algorithm on the edge devices, which detects the key features relevant for the application and sends it to the edge server for reinforcement learning to continuously improve the accuracy of models. Note that the devices reporting the presence of graffiti, pothole or garbage currently do not send the entire media content due to the bandwidth limitation and energy consumption, which means the models are not dynamically trained using the real-world data generated by the application. The core components of an edge devices are discussed below:

1) Model download module: This module communicates the device capability and user constraints with the edge server and downloads an ML model that can operate within the resource capacity of the device. Each device, when running the crowd-based learning framework, maintains a resource profile which is used to identify the suitable model for the edge device. The user or the owner of the edge device may configure the edge device to determine the resource allocation for the crowd-based learning framework. For example, the user constraints may include the maximum memory space a model can consume, the maximum time to run an inference, the energy budget per inference, etc.

2) Video frame extraction module: This module is responsible for capturing video content and convert it into a series of keyframes [31]. Extracted frames can be fed into the inference module to identify desired objects or classes.

3) Visual feature extraction module: During phase one of the inference, the module resizes the raw image to the trained input resolution and feeds the resized image to the trained model which uses the layer just before the final output layer (that actually does classification) to extract VFV values.

4) Inference module: During phase two of the inference, the VFVs extracted in the previous step, are fed into the inference module, which uses the last layer of the trained model to identify the desired object or feature.

5) Inference quality control module: This module decides whether to submit the inference results and the extracted VFVs to the edge server. The decision is enforced by the policy module which dictates when data need to be transmitted to the server. For example, a quality control policy can specify that only the inference results that are above a certain threshold are reported to the edge server and/or the feature vectors of the images captured in a particular location or time can be uploaded. Such a policy allows the edge device to preserve bandwidth and energy consumption, while the server is able to control the quality and semantic importance of the crowdsourced results.

6) Policy module: The policy module maintains the desired policies set by the ED-Server based on the resource capacity. This module ensures that the results generated by edge devices are of high quality, as the reporting of sub-optimal results may result in bandwidth wastage while adding little or no value to the overall application. In addition, the policies provide the flexibility to the server to orchestrate the selection of quality, volume, and priority of the extracted VFVs. For example, the policy can allow some devices to prioritize sending their data based on image location, time, utility (e.g., can retrain two classification tasks with the same image) or semantic (e.g., underrepresented labels). Besides, a policy can specify which features to send based on their size: low, medium, high-resolution image or size of the feature vectors.

B. ED-Server

The ED-server is a central repository consisting of a set of ML models and an algorithm to create models. Media content from various online sources can be used to generate a (static) model for the edge devices. Typically, the edge server maintains a generic model for processing media content. In our framework, we introduce a closed loop training system to fine-tune the models using real-world data on the edge server for a wide class of edge devices. An edge device that joins the application network requests the edge server to provide a model for a particular application by announcing its location and the resource capacity. Based on the information received from the edge device, the server dispatches a suitable model. Whenever the edge device captures media content, a feature extraction algorithm is executed to collect relevant features. Instead of transmitting the entire media content over a resource-constrained network, our framework extracts the key features necessary to improve the accuracy of the model and send it to the edge server only when they satisfy the rule set by the edge server's policy configuration module. Through continuous feedback, our edge server maintains a set of models with different resource requirements and accuracy levels to support a wide array of edge devices with varying resource capacities and inference quality.

1) Model distribution module: This module processes the metadata provided by an edge device when it joins the crowd-based learning framework. Based on the resource capacities and the additional constraints enforced by the user, the model distribution module dispatches a suitable model and associated policy configuration to the edge device.

2) Training module: This module is responsible for creating models by using existing datasets. A set of labeled images or video frames are fed to one of the convolutional neural network (CNN) architectures (e.g., Caffe [32], TensorFlow [33]), to train an initial model. Our implementation used TensorFlow for the extraction of VFVs and training.

3) Database: The ED-Server maintains a database as the storage layer for trained models, model metadata, classes of edge devices, VFVs, inference results and other ED and model metadata.

4) Retrain module: This module determines when to initiate retraining. The VFVs collected from edge devices are fed to the retrain module to determine the uniqueness of the data. When the newly submitted VFV is unique, it is added to the dataset for retraining. This module ensures that the models maintained for each type of edge devices are of high quality by continuously upgrading the models through the labeled data submitted by edge devices.

5) Model metadata: Along with a trained model, its model metadata is exported and stored. Model metadata contains valuable information that is used to distribute or retrain the model. For example, a $spatial - coverage$ field is used to prioritize VFV collection from under-represented locations (more details in Section IV-B), a $labels$' counts field to collect under-represented classes, the $FLOPS$ field can provide insight about the model complexity and a rough estimate of an inference time.

IV. Crowd-based Learning

Our framework uses an available set of images (referenced as $\mathcal{D}_i$) for training an initial model (referenced as $\mathcal{M}$) to be distributed to all edge devices. Subsequently, edge devices perform two tasks: collecting new images (a.k.a. crowdsourced images) (referenced as $\mathcal{D}_c$) and analyzing the content of each image (i.e., $d \in \mathcal{D}_c$) using the equipped model $\mathcal{M}$ to predict a label describing the image. Over time, the edge devices can aggregate a large amount of images ($\mathcal{D}_c$) that can be used to evolve $\mathcal{M}$ and create a better model (referenced as $\mathcal{M}'$). One straightforward approach is to use all of the new images $\mathcal{D}_c$ for obtaining $\mathcal{M}'$. However, this approach may suffer from multiple challenges: 1) high network bandwidth requirement to transmit the whole $\mathcal{D}_c$, 2) potential degradation of model accuracy (e.g., due to including bad quality images), and 3) long computational processing time required for creating $\mathcal{M}'$ using $\mathcal{D}_i + \mathcal{D}_c$ at the edge server. Subsequently, our framework aims at selecting a subset of $\mathcal{D}_c$ (referenced as $\mathcal{D}'_c$) (i.e., $\mathcal{D}'_c \subseteq \mathcal{D}_c$) that can be used to evolve $\mathcal{M}$ to an efficient $\mathcal{M}'$. Towards this, there are two paradigms: centralized (i.e., the selection process is performed on the edge server) and decentralized (i.e., the selection process is performed on the edge devices). We focus on the decentralized paradigm which utilizes the model metadata for selecting appropriate images which can potentially enhance the model. In what follows, we investigate several metrics that affect the selection procedure.

A. Image Quality

Since the framework includes heterogeneous types of edge devices (i.e., equipped with various types of cameras), the quality of the captured images vary. The quality of an image can be characterized by different specifications including illumination and image resolution. In particular, an extreme illumination potentially affects the model negatively. On the one hand, a very high illumination on an image blurs the image, while a very low illumination makes the image vague and black. Thus, both cases make the image content unclear and harden visual perception. Hence, edge devices can discard the images with such very high or low illumination. On the other hand, image resolution also potentially affects the model efficiency. Since the model metadata contains the image resolution of $\mathcal{D}_i$, each image available at an edge devise needs to be processed to get a resolution similar to the one in the model metadata.

B. Image Metadata

Edge devices typically contain GPS receiver and digital compass to estimate the geospatial location of a captured image. Therefore, the images are automatically tagged with spatiotemporal metadata (i.e., location, viewing direction, and time). Such metadata effectively enable the framework to select a subset of images to expand the spatial and temporal coverage of $\mathcal{D}_i$. Since the model metadata include information about both the spatial and temporal coverage of $\mathcal{D}_i$, the edge devices can identify the images of $\mathcal{D}_c$ that are tagged with locations or timestamps not covered by $\mathcal{D}_i$.

At the edge server, the spatial coverage of $\mathcal{D}_i$ can be calculated using the Grid index structure. In particular, the global area of $\mathcal{D}_i$ can be divided into grid cells. Thereafter, the coverage of each cell is represented either by a Boolean or percentage value. The Boolean value indicates whether the cell contains at least one image while the percentage represents how much area is visually covered by the images contained in that cell[2]. Similarly, the edge server can also measure the temporal coverage of $\mathcal{D}_i$. Thereafter, the server augments both the temporal and spatial coverage in the model metadata. Subsequently, the edges use the spatial and temporal coverage of $\mathcal{D}_i$ to prioritize the images that are excluded by the temporal and spatial coverage of $\mathcal{D}_i$.

C. Image Label

Using $\mathcal{M}$, an edge device can predict a label to describe the content of each image $d \in \mathcal{D}_c$. The label is also associated with a confidence score indicating the prediction certainty by the model. The user can provide a feedback on the predicted label and in such cases, the user's feedback will be used alternatively if the edge devices are operated by human (e.g., smartphones). The image label feedback is used in selecting $\mathcal{D}'_c$ to enhance $\mathcal{M}$. Since the model metadata, which is stored at each edge, includes the image distribution of $\mathcal{D}_i$ among the various labels, the edge can select a subset of $\mathcal{D}_c$ that is associated with labels that are less-available in $\mathcal{D}_i$. Furthermore, the edge may choose only "certainly labeled" images (i.e., minimizing the noise in the newly created model) by discarding the images associate with low-confidence scores.

Edge devices may contain multiple trained models for various smart-city applications. In this case, after evaluating the above metrics for $\mathcal{D}_c$, the image subset which is qualified is prioritized based on the metrics of multiple models.

V. Experiments

A. Datasets

To demonstrate the effectiveness of our proposed framework we used a real-world dataset: a labeled collection of Los Angeles County street images. We investigated an approach for the automation of street scene classification based on their cleanliness level using real geo-tagged images from the Los Angeles Sanitation Department (LASAN). Each image has geotagged metadata, i.e., the location where the image was taken is known. The dataset contains 42,331 images with five distinct labels: 14,495 bulky items, 7,120 illegal dumping, 7,007 encampment, 6,982 overgrown vegetation, and 6,727 clean[3]. We denote the Street Cleanliness dataset as $\mathcal{D}^{SC}$, which represents an unbalanced real-world data in our experiments. Note that the unbalanced dataset may generate a biased learning model.

In addition, we included the Caltech 256 [37] dataset, denoted as $\mathcal{D}^{CAL256}$, which contains 256 labels, 30,608 images in total, with a minimum of 80 images per label and 119 on average. $\mathcal{D}^{CAL256}$ represents a well-defined dataset for retraining in our experiments.

In our experiments, we trained models to classify predefined objects/classes in images. To save computing power in terms of GPU-hours, instead of training new models from scratch, we used *transfer learning*. More specifically, we extract the VFVs from powerful models pre-trained on ImageNet dataset and train a softmax layer on top. We used three pre-trained models to compare our results: Inception V3 [38], MobileNet V1 [39], and V2 [40]. Although Inception V3 is not designed for mobile and embedded devices, we include it in our experiments for comparison.

B. Model Retraining

Nowadays, a large amount of imagery information is captured in a continuous and streaming fashion. Once a model is trained, it must be adapted to capture changes over time. To better illustrate the importance of retraining consider the following example: *Smart city deployments starting to use video streams captured by sanitary trucks [41] to train models in order to automate the prioritization of street*

[2]The spatial coverage of a cell for $\mathcal{D}_i$ can be measured using a spatial coverage model [34] that utilizes various image metadata such as geo-location, viewing direction, and spatial extent (referred to as image field of view [35] or scene location [36]).

[3]The Dataset and Classification for Identifying the Level of Street Cleanliness is adopted from [2].

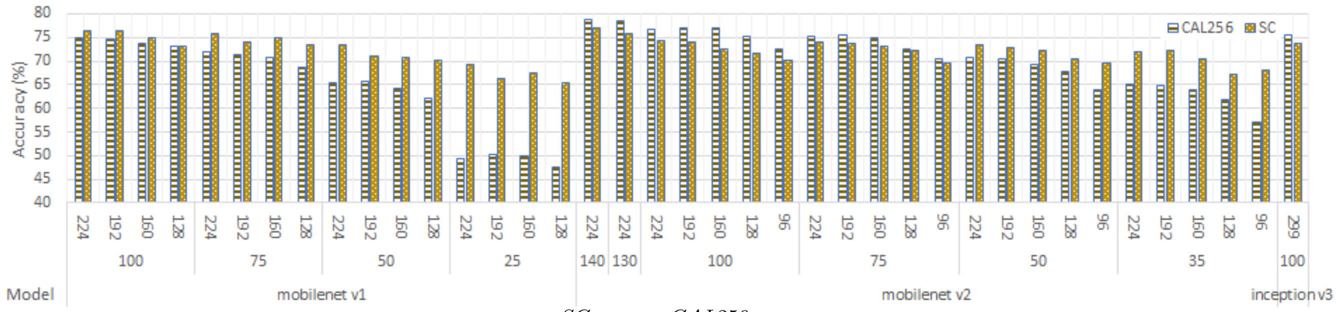


Figure 2: Accuracy for $\mathcal{D}^{SC}$ and $\mathcal{D}^{CAL256}$ Dataset for Various Models

cleaning based on the cleanliness level of the streets. The model predicts five classes: clean, bulky items, encampment, overgrown vegetation, and illegal dumping [2]. The city plans to release the model as part of a mobile application to the public to collect more data for improving the efficiency of their cleaning process. Sanitary trucks are typically driven around the city during specific working hours of the day. In addition, parts of the city may not be covered because some trucks lack video cameras. In such a scenario, if the model is fed with new data obtained by the public, that differs significantly from the data used for training, (e.g., obtained by different viewing angle, during different hours of the day, etc.), the model could be made more robust through retraining. In the following experiments, we retrain our models from scratch. Unless otherwise noted, we use the initial training dataset $\mathcal{D}_i$ along with the new dataset obtained by edge devices $\mathcal{D}'_c$, as the new training dataset (i.e., $\mathcal{D}$ = $\mathcal{D}_i \bigcup \mathcal{D}'_c$).

1) Width Multiplier and Resolution vs. Accuracy: MobileNets allow tuning width multiplier α and input resolution ρ hyper parameters which in turn control the model size and complexity. Here, the role of width multiplier is to thin the network connectivity uniformly at each layer in the underlying neural network. Specifically, the input channels M and output channels N at a given layer, become αM and αN, respectively. This, in effect, reduces the number of parameters and computational cost by roughly α^2. The input resolution hyper parameter ρ is applied to the input image of the CNN and the internal representation at each layer, reducing computational cost by roughly ρ^2 [39]. For Mobilenet V1, we varied the width multiplier $\alpha \in$ (100%, 75%, 50%, 25%) and input resolution $\rho \in$ (224, 192, 160, 128). Similarly, for Mobilenet V2, we varied the width multiplier $\alpha \in$ (140%, 130%, 100%, 75%, 50%, 35%) and

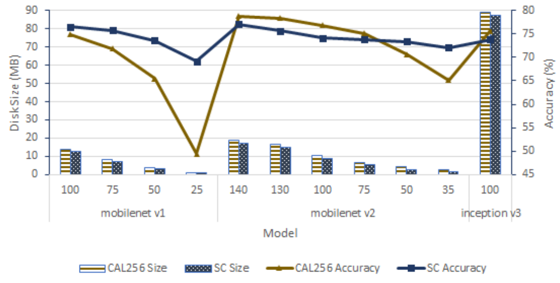


Figure 3: Model Size vs. Accuracy for $\mathcal{D}^{SC}$ and $\mathcal{D}^{CAL256}$ Datasets for Various Models

input resolution $\rho \in$ (224, 192, 160, 128, 96). Inception V3 has a fixed input resolution of 299.

Figure 2 plots the test accuracy (y-axis) for $\mathcal{D}^{SC}$ and $\mathcal{D}^{CAL256}$ datasets on 10% of stratified images, after training using 5-fold cross validation on the different models (x-axis). The results showed that the accuracy highly depends on width multiplier α and input resolution ρ. A given model flavor may not be suitable for an arbitrary edge device as some models may require more resources for inferencing, demonstrating a clear trade-off between model complexity, accuracy, and the resource capacity of the edge devices. For example, the $\mathcal{D}^{SC}$ had the best accuracy of 77.08% when trained on MobileNet V2, $\alpha = 1.4$, $\rho = 224$, and the worst accuracy of 65.32% when trained on MobileNet V1, $\alpha = 0.25$, $\rho = 128$. Although most users strive for the most accurate model, the resource constraints of an edge device might render them impracticable. In such cases, an alternative model such as MobileNet V1, $\alpha = 0.5$, $\rho = 128$, can be chosen, which is less accurate (70.07%) but less computationally demanding. In addition, each model configuration affects the $\mathcal{D}^{CAL256}$ and $\mathcal{D}^{SC}$ datasets differently. For example, MobileNet V1, $\alpha = 0.5$, the $\mathcal{D}^{CAL256}$ dataset degraded the accuracy significantly compared to its best case (i.e., a 37% decrease on average), whereas the $\mathcal{D}^{SC}$ dataset was less affected for the same configuration (i.e., a 12.5% decrease on average).

2) Model Size vs. Accuracy: Figure 3 plots the accuracy compared to the model disk size[4] (storage space on device). A more complex model (with a higher width multiplier) resulted in a larger model size and a higher inference accuracy. Although disk space might not be a concern (most devices have plenty of available disk capacity), large neural network models have larger memory footprint, require more time to persist in and out of an edge device's memory and demand more bandwidth for model distribution. Hence, it is often desirable to keep the model as small as possible, while not sacrificing much accuracy. As shown in Figure 3, the $\mathcal{D}^{SC}$ trained on MobileNet V1, $\alpha = 1$ achieved 76.5% accuracy with a model that requires 13MB disk space. MobileNet V1, $\alpha = 0.25$ achieved the accuracy of 69.14%

[4]The input resolution α does not affect the model disk space. Hence we omit it from the graph for brevity. Here, the accuracy corresponds to the highest image resolution (i.e., 224 for MobileNets and 299 for Inception V3).

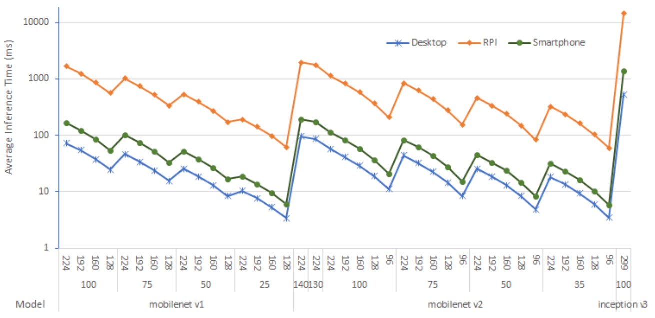


Figure 4: Inference Time vs. Model for $\mathcal{D}^{SC}$ Dataset

with 860KB, a 9.6% accuracy loss for 93% disk space reduction. Also in some cases, the size does not increase the accuracy. For example, despite that Inception V3 generated the largest models of 89MB and 87MB for $\mathcal{D}^{CAL256}$ and $\mathcal{D}^{SC}$, respectively, their accuracy is lower than the best of MobileNets. Hence, a model distribution middleware is particularly important to dispatch the right model based on the resource capacity of the devices.

3) Inference Time vs. Model: After generating models for $\mathcal{D}^{SC}$ and $\mathcal{D}^{CAL256}$ datasets, we measured the average time to perform an inference on various edge devices which have different resource capacities. The results for $\mathcal{D}^{SC}$[5] are shown in Figure 4. We ran each model on 50 random inputs and plotted the mean inference time in milliseconds and logarithmic scale, required to perform each prediction. Raspberry Pi had limited resources compared to desktop class devices, requiring thousands of milliseconds inference time, and on average it was *1.5x* order of magnitude slower compared to desktop class devices. The desktop class devices were capable of processing the VFVs faster (tens of milliseconds in most cases) for models with various complexities and image sizes. Our framework dispatches the suitable model, with prior knowledge of resource capacities of edge devices and their performance capabilities, which not only accounts for the resource consumption but also aims to achieve the best result possible within the available resource budget.

4) Feature Size vs. Accuracy: A large number of edge devices run on battery and extending their lifetime is crucial to ensure longevity and to reduce maintenance cost. In our framework, we categorize power consumption in two main categories: 1) CPU/GPU processing and 2) data transmission. The former includes the power consumed, for example, to capture an image, load a model, extract VFVs and infer the class of the images, whereas the latter includes the power consumed to download the model, send the raw image or VFVs and inference results to the ED-Server through a communication channel (WiFi, Bluetooth, etc).

Uploading only VFVs instead of raw images from edge devices to the server is crucial in saving bandwidth and battery power. Besides, a crowd-based learning mechanism to submit VFVs is an essential part of the model retraining module. Figure 5 plots the average size in KB (left y-axis),

[5]Similar trends were observed for $\mathcal{D}^{CAL256}$ dataset so it was omitted.

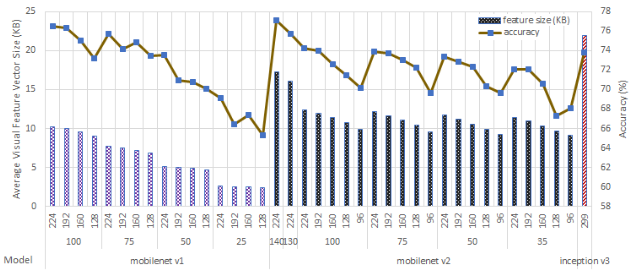


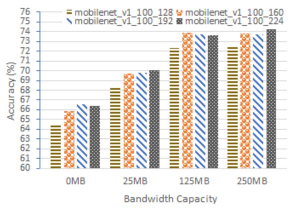
Figure 5: Average Feature Size vs. Accuracy for $\mathcal{D}^{SC}$

of the extracted VFVs and their corresponding accuracy (right y-axis). For example, the average size of the VFV for MobileNet V1, $\alpha = 1$, $\rho = 224$ was around 10KB. Usually, the larger the size of the VFVs, the higher the accuracy, because they carry a more detailed summary of the image.

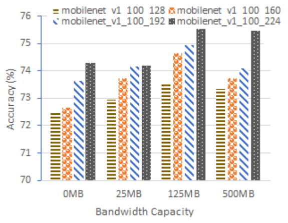
To make our analysis simpler and widely applicable, instead of limiting the power consumption which is device-specific, we impose limitations on the total consumed bandwidth capacity which is more generic, and investigate how the bandwidth limitation affects the accuracy of the model.

Figures 6a and 6b depict the trade-off between bandwidth and accuracy for four different models. For each dataset $\mathcal{D}^{CAL256}$ and $\mathcal{D}^{SC}$, we first split each to 10% test ($\mathcal{D}^{CAL256_{test}}$ and $\mathcal{D}^{SC_{test}}$, respectively) and 10% train stratified subsets ($\mathcal{D}_i^{CAL256}$ and $\mathcal{D}_i^{SC}$, respectively). Then, according to the total bandwidth limitation, the train dataset are augmented to include labeled VFVs uploaded from edge devices (i.e., $\mathcal{D}_i^{CAL256} \bigcup \mathcal{D}_c^{CAL256'}$ and $\mathcal{D}_i^{SC} \bigcup \mathcal{D}_c^{SC'}$, respectively). For a given available upload size, the device can send more or less VFVs for a given model, depending on their size. For example, with 0MB available bandwidth capacity, no VFVs are sent and all models are trained on the 10% initial training dataset. For 5MB, the training dataset for the models $mobilenet\_v1\_100\_128$, $mobilenet\_v1\_100\_160$, $mobilenet\_v1\_100\_192$, $mobilenet\_v1\_100\_224$ are augmented with 588, 550, 524, 505 new VFVs respectively, for 25MB, they are augmented with 2939, 2756, 2626, 2529 new VFVs and so on. We ensure that at the maximum bandwidth capacity, all models receive the 100% of dataset's VFVs, that is $\mathcal{D}_c^{CAL256'} = \mathcal{D}_c^{CAL256}$ and $\mathcal{D}_c^{SC'} = \mathcal{D}_c^{SC}$.

Figure 6a and 6b shows how the accuracy is affected by different bandwidth capacities. The accuracy of the model improved with the increase in VFV set, but it came at the cost of high bandwidth. Figure 6a and 6b shows that the devices with the higher bandwidth can send more VFVs to the edge devices for retraining the model to enhance accuracy. This is more apparent for the $\mathcal{D}^{CAL256}$ dataset which includes 256 labels and requires more VFVs to classify the test images correctly. Interestingly, at 125MB, the accuracy of less complex models is higher. This is because the server can receive a larger number of VFVs, which although they contain a more compact summary, can help to distinguish the labels more accurately. The accuracy of the model settled at bandwidth above 25MB to a steady state since the size of the



(a) Bandwidth vs. Accuracy on $\mathcal{D}^{CAL256}$ Dataset



(b) Bandwidth vs. Accuracy on $\mathcal{D}^{SC}$ Dataset

Figure 6: Bandwidth vs. Accuracy

VFV set did not influence the accuracy when the edge device sent 125MB worth of VFVs for retraining. Consequently, the retraining and model dispatching channel decisions for the edge devices should be taken based on the bandwidth capacity and the number of labels in the dataset. In summary, our server maintains a set of models which are suitable for devices with various bandwidth capacities. When the edge device operating in a given bandwidth capacity submits a VFV to the server, the server uses the received VFV set to enhance the model by initiating the retraining sequence.
Our approach is therefore capable of not only performing inference on resource-constrained devices such as Raspberry Pi and medium-end smartphones but also provides the ability to enhance the models for devices with limited bandwidth. In some cases, an edge device may have processing capacity but limited bandwidth. Such scenarios require models that are targeted for bandwidth-constrained devices rather then processing-constrained devices.

5) *Location based retraining:* ML models deployed on edge devices for Smart City applications should have the flexibility to evolve and adapt the model to account for newly fed data. Due to the dynamic nature of the environment, and the vast amount of images and videos captured by edge devices, a VFV selection mechanism should be deployed which will orchestrate the feature acquisition process. One such feature selection mechanism is based on the location where the visual data were captured. The policies enforced by the ED-Server orchestrator on the edge devices prioritize the collection of VFVs at under-represented locations to capture unique information. Intuitively, the infrastructure and landscape vary significantly in different parts of the city, resulting in images with different backgrounds, unique architectural styles, and VFVs.

In our experiment, we used the $\mathcal{D}^{SC}$ dataset to highlight the importance of location-based retraining. We analyzed the location metadata of $\mathcal{D}^{SC}$ and removed all images the fall within the Downtown Los Angeles (DTLA) area (referred as $\mathcal{D}^{SC}_{DTLA}$). The architecture of DTLA is significantly different compared to other locations in Los Angeles. The dataset $\mathcal{D}^{SC}_{DTLA}$ is then split to 50%-50% stratified train $\mathcal{D}^{SC_{train}}_{DTLA}$ and test $\mathcal{D}^{SC_{test}}_{DTLA}$ datasets. Subsequently, we trained two models: a model $\mathcal{M}1$ on the dataset which does not include images from DTLA at all, i.e., $\mathcal{D}^{SC} \setminus \mathcal{D}^{SC}_{DTLA}$ and a model $\mathcal{M}2$ that includes the train dataset from DTLA, i.e., $\mathcal{D}^{SC} \setminus \mathcal{D}^{SC_{test}}_{DTLA}$. After training, both models were tested on

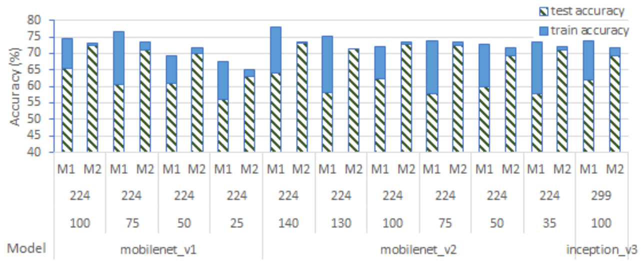


Figure 7: Location-based Feature Selection ($\mathcal{M}1$ is trained on $\mathcal{D}^{SC} \setminus \mathcal{D}^{SC}_{DTLA}$, while $\mathcal{M}2$ is trained on $\mathcal{D}^{SC} \setminus \mathcal{D}^{SC_{test}}_{DTLA}$)

the unseen dataset SC^{test}_{DTLA}. Our results depicted in Figure 7 show that under-represented regions (such as DTLA) significantly affect the accuracy for all variations of $\mathcal{M}1$; sometimes there is more than 15% drop in accuracy, while $\mathcal{M}2$ is more robust, with no significant drop. Although we do not include experiments for time-based image selection, we expect that the temporal information of collected images is symmetrically important. Hence, smart city crowd-based learning applications should not only collect image data, but also enrich the collected datasets with location and time information, to enable spatiotemporal VFV selection.

VI. Conclusion

This paper presented a crowd-based learning framework, which combines the principles of EC with ML to enable heterogeneous edge devices to participate in and contribute to smart city applications. The proposed framework dispatches the ML model to the edge device based on its resource capacity. Furthermore, the proposed intelligent retraining algorithm enables the edge device to share new information back to the edge server to enhance the model by assessing the novelty of the data, by assessing the uniqueness of the geospatial and temporal information at the edge device. To the best of our knowledge, the crowd-based learning framework presented in this paper is the first with built-in support for heterogeneous edge devices and model retraining based on uniqueness. Results show the trade-off between inference accuracy and the resource constraints of the edge devices. By reducing the model complexity, the inference accuracy and performance varies, which shows that the models can be tuned to support wide classes of edge devices. As part of future work, we plan to focus on a) testing the proposed crowd-based learning framework in real-world application scenarios in the City of Los Angeles, b) expanding the model repository with support for more classes of edge devices, and c) integrating this solution in our visionary framework (Translational Visual Data Platform, TVDP [42]) for smart cities.

Acknowledgment

This research has been supported in part by NSF grant CNS-1461963, the USC Integrated Media Systems Center, the Annenberg Foundation, and unrestricted cash gifts from Google, Oracle and the USC Viterbi Center for Cyber-Physical Systems and the Internet of Things (CCI).

References

[1] B. Krishnamachari, J. Power, S. H. Kim, and C. Shahabi, "I3: An iot marketplace for smart communities," in *MobiSys*. ACM, 2018, pp. 498–499.

[2] A. Alfarrarjeh, S. H. Kim, S. Agrawal, M. Ashok, S. Y. Kim, and C. Shahabi, "Image classification to determine the level of street cleanliness: A case study," in *BigMM*. IEEE, 2018, pp. 1–5.

[3] A. Alfarrarjeh, D. Trivedi, S. H. Kim, and C. Shahabi, "A deep learning approach for road damage detection from smartphone images," in *Big Data*. IEEE, 2018, pp. 5201–5204.

[4] S. H. Kim, Y. Lu, G. Constantinou, C. Shahabi, G. Wang, and R. Zimmermann, "MediaQ: Mobile Multimedia Management System," in *MMSys*. ACM, 2014, pp. 224–235.

[5] Y. Cai, Y. Lu, S. H. Kim, L. Nocera, and C. Shahabi, "Gift: A geospatial image and video filtering tool for computer vision applications with geo-tagged mobile videos," in *ICMEW*. IEEE, 2015, pp. 1–6.

[6] Y. Lu, C. Shahabi, and S. H. Kim, "Efficient indexing and retrieval of large-scale geo-tagged video databases," *GeoInformatica*, vol. 20, no. 4, pp. 829–857, 2016.

[7] A. Alfarrarjeh, C. Shahabi, and S. H. Kim, "Hybrid indexes for spatial-visual search," in *Thematic Workshops of ACM MM 2017*. ACM, 2017, pp. 75–83.

[8] A. Parra, M. Boutin, and E. J. Delp, "Automatic gang graffiti recognition and interpretation," *Journal of Electronic Imaging*, vol. 26, no. 5, p. 051409, 2017.

[9] A. Alfarrarjeh, D. Trivedi, S. H. Kim, H. Park, C. Huang, and C. Shahabi, "Recognizing material of a covered object: A case study with graffiti," in *ICIP*. IEEE, 2019.

[10] S. H. Kim, J. Shi, A. Alfarrarjeh, D. Xu, Y. Tan, and C. Shahabi, "Real-time traffic video analysis using intel viewmont coprocessor," in *DNIS*. Springer, 2013, pp. 150–160.

[11] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[12] R. Balan, J. Flinn, M. Satyanarayanan, S. Sinnamohideen, and H. Yang, "The case for cyber foraging," in *ACM SIGOPS EW*. ACM, 2002, pp. 87–92.

[13] J. Flinn, S. Park, and M. Satyanarayanan, "Balancing performance, energy, and quality in pervasive computing," in *ICDCS*. IEEE, 2002, pp. 217–226.

[14] P. S. Pillai, L. B. Mummert, S. W. Schlosser, R. Sukthankar, and C. J. Helfrich, "Slipstream: Scalable low-latency interactive perception on streaming data," in *NOSSDAV*. ACM, 2009, pp. 43–48.

[15] Y. Iida, Y. Fujii, T. Azumi, N. Nishio, and S. Kato, "Gpurpc: Exploring transparent access to remote gpus," *TECS*, vol. 16, no. 1, pp. 17:1–17:25, 2016.

[16] M. Ra, A. Sheth, L. Mummert, P. Pillai, D. Wetherall, and R. Govindan, "Odessa: Enabling interactive perception applications on mobile devices," in *MobiSys*. ACM, 2011, pp. 43–56.

[17] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *DEBS*. ACM, 2016, pp. 258–269.

[18] V. Cozzolino, A. Y. Ding, J. Ott, and D. Kutscher, "Enabling fine-grained edge offloading for iot," in *SIGCOMM Posters and Demos*. ACM, 2017, pp. 124–126.

[19] M. Satyanarayanan, P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, W. Hu, and B. Amos, "Edge analytics in the internet of things," *IEEE Pervasive Computing*, vol. 14, no. 2, pp. 24–31, 2015.

[20] E. G. Renart, J. Diaz-Montes, and M. Parashar, "Data-driven stream processing at the edge," in *ICFEC*, 2017, pp. 31–40.

[21] S. G. Ahmad, C. S. Liew, M. M. Rafique, E. U. Munir, and S. U. Khan, "Data-intensive workflow optimization based on application task graph partitioning in heterogeneous computing systems," in *BdCloud*, 2014, pp. 129–136.

[22] D. C. Brabham, "Crowdsourcing as a model for problem solving: An introduction and cases," *Convergence*, vol. 14, no. 1, pp. 75–90, 2008.

[23] L. Kazemi and C. Shahabi, "Geocrowd: enabling query answering with spatial crowdsourcing," in *SIGSPATIAL GIS*. ACM, 2012, pp. 189–198.

[24] D. Deng, C. Shahabi, and L. Zhu, "Task matching and scheduling for multiple workers in spatial crowdsourcing," in *SIGSPATIAL GIS*. ACM, 2015, p. 21.

[25] A. Alfarrarjeh, T. Emrich, and C. Shahabi, "Scalable spatial crowdsourcing: A study of distributed algorithms," in *MDM*. IEEE, 2015, pp. 134–144.

[26] "Amazon mechanical turk," [Last Accessed: Dec. 20, 2018]. [Online]. Available: http://www.mturk.com/

[27] "Taskrabbit," [Last Accessed: Dec. 20, 2018]. [Online]. Available: https://www.taskrabbit.com/

[28] N. Padhariya and K. Raichura, "Crowdlearning: An incentive-based learning platform for crowd," in *IC3*. IEEE, 2014, pp. 44–49.

[29] Y. Liu and M. Liu, "Crowd learning: improving online decision making using crowdsourced data," in *IJCAI*. AAAI Press, 2017, pp. 317–323.

[30] A. Alfarrarjeh, S. Agrawal, S. H. Kim, and C. Shahabi, "Geo-spatial multimedia sentiment analysis in disasters," in *DSAA*. IEEE, 2017, pp. 193–202.

[31] W. Wolf, "Key frame selection by motion analysis," in *ICASSP*, vol. 2. IEEE, 1996, pp. 1228–1231.

[32] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *ACM MM*. ACM, 2014, pp. 675–678.

[33] M. Abadi, P. Barham, J. Chen *et al.*, "Tensorflow: a system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[34] A. Alfarrarjeh, S. H. Kim, A. Deshmukh, S. Rajan, Y. Lu, and C. Shahabi, "Spatial coverage measurement of geo-tagged visual data: A database approach," in *BigMM*. IEEE, 2018, pp. 1–8.

[35] S. A. Ay, R. Zimmermann, and S. H. Kim, "Viewable Scene Modeling for Geospatial Video Search," in *ACM MM*. ACM, 2008, pp. 309–318.

[36] A. Alfarrarjeh, S. H. Kim, S. Rajan, A. Deshmukh, and C. Shahabi, "A data-centric approach for image scene localization," in *Big Data*. IEEE, 2018.

[37] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," California Institute of Technology, Tech. Rep. 7694, 2007. [Online]. Available: http://authors.library.caltech.edu/7694

[38] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, in *CVPR*, 2016, pp. 2818–2826.

[39] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017.

[40] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018.

[41] C. Mertz, S. Varadharajan, S. Jose, K. Sharma, L. Wander, and J. Wang, "City-wide road distress monitoring with smartphones," in *ITS World Congress*, 2014, pp. 1–9.

[42] S. H. Kim, A. Alfarrarjeh, G. Constantinou, and C. Shahabi, "TVDP: Translational visual data platform for smart cities," in *ICDEW*. IEEE, 2019, pp. 45–52.