IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received November 8, 2019, accepted November 26, 2019, date of publication January 3, 2020, date of current version January 14, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2963912

Urban Intelligence With Deep Edges

GARY WHITE AND SIOBHÁN CLARKE
School of Computer Science and Statistics, Trinity College Dublin, Dublin 2, D02 W272 Ireland
Corresponding author: Gary White (whiteg5@scss.tcd.ie)

This work was supported by the Science Foundation Ireland (SFI) under Grant 13/IA/1885.

ABSTRACT With the increased accuracy available from state of the art deep learning models and new embedded devices at the edge of the network capable of running and updating these models there is potential for urban intelligence at the edge of the network. The physical proximity of these edge devices will allow for intelligent reasoning one hop away from data generation. This will allow a range of modern urban reasoning applications that require reduced latency and jitter such as remote surgery, vehicle collision detection and augmented reality. The traffic flow from IoT devices to the cloud will also be reduced as with the increased accuracy from deep learning models only a subset of the data will need to be reported after a first pass analysis. However, the training time of deep learning models can be long, taking weeks on multiple desktop GPUs for large datasets. In this paper we show how transfer learning can be used to update the last layers of pre-trained models at the edge of the network, dramatically reducing the training time and allowing the model to perform new tasks without data ever having to be sent to the cloud. This will also improve the users' privacy, which is a key requirement for urban intelligence applications with the introduction of GDPR. We compare our approach to alternative IoT urban intelligence architectures such as cloud-based architectures and deep learning algorithms trained only on local data.

INDEX TERMS Edge computing, transfer learning, deep learning, urban intelligence, IoT, QoS.

I. INTRODUCTION

Urban intelligence can provide insight into a number of the major problems being faced by our cities, such as air pollution, increased mobile communication network demand, traffic congestion and water floods, etc., through the use of data from IoT sensors and intelligent analytics on the data. This can allow for the efficient running of a smart city, making the most of the resources available. Problems in smart cities such as increased network demand and crowd congestion are set to get increasingly difficult with the UN predicting 60% of people globally will live in cities with at least half a million inhabitants by 2030 [1]. This will exacerbate the current problems in our cities, so it will be critical to mange these problems with intelligent urban reasoning algorithms and a suitable deployment architecture. IoT devices will enable access to a range of devices and information that was not available to urban authorities before. This additional information from air pollution sensors, CCTV cameras, vehicles and so on will allow the development of more advanced urban reasoning applications [2].

The associate editor coordinating the review of this manuscript and approving it for publication was Hao Ji.

The development and rising popularity of bandwidth-heavy applications such as self-driving cars and CCTV footage will mean that smart cities in the future will have much more data being generated. As some of these applications such as self-driving cars have strict tolerable delay requirements there is a need to be able to avoid congestion on the core network. The number of IoT devices is also set to increase, with forecasts predicting 25-50 billion IoT devices by 2020 [3]. Smart cities of the future will need edge computing, which is an architecture with additional computing resources made available at the edge of the network close to the end-devices [4], [5]. The physical proximity of the devices will reduce the latency to support applications with strict QoS requirements such as augmented reality and vehicle collision detection. Applications such as vehicle collision detection are safety critical as they can seriously injure or kill a person if the algorithm fails to work properly. Deep neural networks have shown increased accuracy in a number of recent benchmark competitions in machine learning and pattern recognition [6]. In previous work we have shown how the combination of deep neural network models one hop away from data generation enables a range of new urban reasoning applications and services in smart cities [7]. This architecture

TABLE 1. Urban intelligence applications.

Urban Application	Data Type	Traffic Rate	Tolerable Delay	Number of IoT Devices	Criticality
Waste Management [10]	Historical Data	>= 100 MB per day	> 30 mins	>= 1000-1million per city	Low
Structural Health [10]	Historical Data	>= 10 MB per day	> 30 mins	>= 10-1000 per building	High
Air Quality Monitoring [10]	Historical Data	>= 10 MB per day	> 5 mins	>= 1000-1million per city	High
Noise Monitoring [10]	Historical Data	>= 100 MB per day	> 5 mins	>= 1000-1million per city	Medium
Wearable IoT	Stream Data	<= 1 GB per device	> 5 mins	>= 1-10 per person	Medium
Traffic Congestion [10]	Historical Data	>= 100 MB per day	> 5 mins	>= 1000-1million per city	Low
Smart Parking [10]	Event Data	>= 10 MB per day	> 1 min	>= 1000-1million per city	Low
Smart Home [10]	Stream/ Massive Data	>= 10 MB per house per day	1 s - 10 mins	>= 10 - 100 per house	Medium
Smart Energy [11]	Stream/ Massive Data	>= 100 GB per day	10 ms - 10mins	>= 1 million per grid	Medium
Remote Surgery [12]	Stream/ Massive Data	>= 50 MB per second	<= 100 ms	>= 1-10 per surgery	High
Augmented Reality [13]	Stream/ Massive Data	>= 100 MB per second	<= 10 ms	> 200,000 globally	Low
Autonomous Vehicles [13]	Stream/ Massive Data	>= 100 GB per vehicle per day	<= 10 ms	>= 50-200 per vehicle	High

is called "deep edges" as it combines the accuracy of deep neural networks and the reduced delay of edge networks. In this paper, we extend this initial proposal and show how transfer learning can be used to update pretrained deep neural network models at the edge. We also include a detailed analysis of the requirements of current urban intelligence applications and deep learning algorithms that can be applied to urban intelligence problems.

With the recent success of deep learning a number of pre-trained models are now available to users who may not have the data or computational capability to train them. The computational capability at the edge of the network is limited compared to the cloud. Machine learning libraries such as Tensorflow have set up hubs that allow for the easy sharing of pre-trained models.[1] This allows easy access to large models that can take a very long time to train. For example, the VGG-16 model used in the results section of this paper took 2-3 weeks to train fully using four Nvidia Black GPUs on the ImageNet dataset [8]. Recent transfer learning techniques such as feature extraction and fine tuning allow us to use and update the model at the edge of the network. As we update only the last few layers of the network, we can download the model to an edge device, such as a Jetson Tx2 and update the model for domain-specific tasks using local data that never leaves the edge device. For example, we can update the pre-trained VGG-16 model using transfer learning for a number of classification tasks at the edge using local data [9]. By using the pre-trained model and transfer learning with local images from the camera a more accurate model can be obtained with less training time and increased accuracy compared to just training the model on the local dataset. The ability to do all the training at the edge also increases privacy as none of the local images leave the edge.

The remainder of the paper is organised as follows: Section II highlights urban intelligence applications and the recent challenges for reduced delay and increased bandwidth in urban applications such as augmented reality and autonomous vehicles. This section also shows the variety of deep learning models that have recently been used for urban intelligence applications. Section III describes the deep edge architecture and how this can allow for alternative training methods such as transfer learning at the edge of the network. Section IV describes the experimental setup and Section V presents the results of those experiments. Section VI concludes the paper.

[1] https://www.tensorflow.org/hub

TABLE 2. Computing layer characteristics.

Charachteristics	IoT	Edge	Cloud
Deployment	Distributed	Distributed	Centralised
Components	Physical Devices	Edge nodes	Virtual
Computational	Very Limited	Limited	Unlimited
Storage	Small	Limited	Unlimited
Response Time	NA	Fast	Slow
Big Data	Source	Process	Process
QoS	NA	High	Medium
Energy	Low	Low	High

II. URBAN INTELLIGENCE AT THE EDGE

Urban computing and intelligence can bridge the gap of pervasive computing, intelligent computing, cooperative communication, mass data management technologies and artificial intelligence to improve urban environments and quality of life in smart city systems. These applications can have a range of demands in traffic rate, tolerable delay and criticality. Modern applications with high criticality and low tolerable delay show the need for accurate models at the edge of the network.

A. EDGE COMPUTING

Table 1 shows a selection of urban applications and illustrates significant differences in their requirements in terms of tolerable delay and traffic rate. Traditional smart city applications that researchers envisioned five years ago had tolerable delays of over one minute [10]. This typically involved applications such as air quality monitoring, traffic congestion and structural health that have a larger tolerable delay suitable for a cloud architecture. However, more recent urban applications such as augmented reality, smart energy and autonomous vehicles have much stricter tolerable delays, which have only become achievable with recent advances in edge computing. Table 2 highlights the major difference between the computing architectures in terms of their computation, storage and QoS characteristics. In this section we

describe the benefits that edge computing can bring to current urban intelligence problems.

1) LOW LATENCY

Recent urban intelligence applications such as autonomous vehicles not only increase the network demand but also require urban reasoning services that can provide information about congestion and collision detection [14]. Applications, such as urban congestion that do not have strict requirements with latency can be deployed in a typical cloud-based architecture, but applications, such as augmented reality have strict requirements for low delay and jitter. There are a growing number of urban reasoning applications, as shown in Table 1, that require this additional level of low latency and jitter to provide effective applications such as remote surgery, smart grid and collision detection [15].

2) BANDWIDTH-INTENSIVE APPLICATIONS

Modern cloud-based urban intelligence applications such as CCTV pedestrian detection and users transmitting 1080p videos are bandwidth intensive. The cumulative data rate for these data heavy applications could quickly saturate the metropolitan area network if growth continues at the current rate: 12,000 users in a city streaming 1080p video requires a network that can deliver 100 gigabits per second and a million users would require a network capable of transmitting 8.5 terabits per second [15]. An edge computing framework such as Gigasight can be used to stop flooding in the network, as video from a mobile device only travels as far as the nearest edge device [16]. Computer visions algorithms can run on the edge device in almost real time to allow only the results of the processing e.g., recognised faces, vehicle accidents, content tags, etc., along with the metadata e.g., size, date captured, capture location, etc., to the cloud. This will result in a massive reduction in data sent to the cloud by three to six of magnitude.

3) MASKING CLOUD OUTAGES

Cloud providers have become increasingly reliable over the years however a total reliance on cloud computing can increase the applications vulnerability to cloud outages. This can be caused by a wide variety of factors that can be outside the cloud providers control such as a failure in the network infrastructure or a cyber-attack such as denial of service being carried out against the provider [17]. The failure of a cloud provider can also happen internally due to the failure of a cluster of machines or through human error as in the case where Amazon S3 web services failed due to a typo [18]. To prevent the loss of data during a web service failure, the edge device can act as a proxy for the cloud to back up the data and perform the critical functionalities needed by the user on local data [15]. This allows the application to correctly function and provide services even during a cloud outage. When access to the cloud is fixed, the actions performed on the edge devices and data stored can be propagated to the cloud for reconciliation.

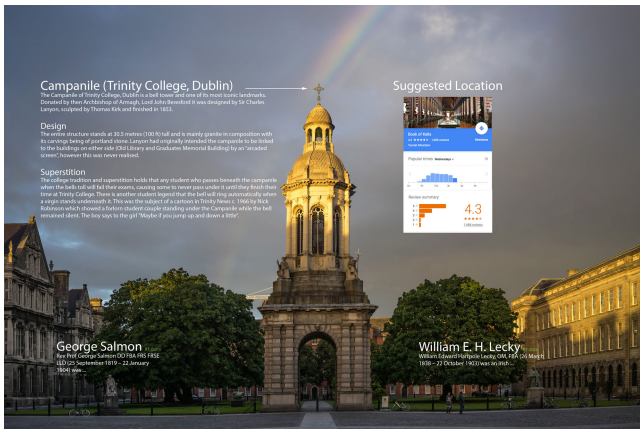


FIGURE 1. Augmented reality for urban visualisation. [22].

4) PRIVACY-POLICY ENFORCEMENT

Privacy-policy enforcement has become increasingly important with the ratification of the General Data Protection Regulation (GDPR) on the 25^{th} May 2018 as part of the EU Data Protection Directive. This has allowed users to have much more control over the data that is being stored about them and who will receive access to this data [19]. With a number of other regions in the world expected to enforce similar legislation privacy-policy enforcement will become increasingly important [20]. Citizens in a smart city should be able to delete data that they deem to be highly sensitive or private. Service providers should use data aggregated across multiple people at certain times of the day and denatured images such as faces being blurred. Cloud-based architectures for urban reasoning make this extremely difficult as the data can be transported from an IoT device to a data centre located in any part of the world. Using an edge device can solve some of these problems by running trusted software modules called privacy mediators that execute on an edge device and provide denaturing of data one hop away from the source [21]. The edge device can provide a clear natural boundary of trust with a foundation of scalable and secure privacy that aligns with the users boundaries of trust, while still allowing for urban computing and intelligence.

5) AUGMENTED REALITY FOR URBAN VISUALISATION

Augmented reality and IoT are a highly compatible combination as IoT is a combination of physical objects with virtual representation and augmented reality superimposes additional virtual information to users about smart objects and service in the user's view of the world [22]. Augmented reality can use local information beacons and sensors to provide context-aware suggestions that can be displayed to users [23]. Figure 1 shows a use case where a tourist may want information about the buildings and statues that they are seeing as well as recommendations for places to visit next. This information can be provided through augmented reality, which also gives the opportunity for highly creative applications such as the statues coming to life to explain their achievements to the user. There are a number of strict

TABLE 3. Deep learning for urban intelligence.

Model	Category	Learning Model	Typical Input Data	Urban Applications
AE	Generative	Unsupervised	Various	Object tracking [25], QoS Prediction [26], Detection of road traffic accidents [27]
RNN	Discriminative	Supervised	Serial, time series	Car Park Occupancy [28], Energy Demand [29], Metro Density [30], Waste Generation [31]
RBM	Generative	Unsupervised, Supervised	Various	Urban Object Recognition [32], Intrusion Detection [33], Object Classifcation [34], City Event detection from Twitter [35]
DBN	Generative	Unsupervised, Supervised	Various	Network Traffic Assignment [36], Indoor Localisation [37], Facial Expression Recognition [38],
LSTM	Discriminative	Supervised	Serial, time series	Traffic flow [39], QoS forecasting [40], Air Quality [41], Particule Matter Forecasting [42], Energy Load Forecasting [43],
CNN	Discriminative	Supervised	2D (image, sound, etc)	License Plate Recognition [44], Pedestrian Detection [45], Object Detection [46], Street Cleaning [47], Noise Recognition [48], Predictive Driving Control [49]
VAE	Generative	Semi-supervised	Various	House Number Classification [50], Motion Prediction [50], Anomaly Detection [51]
GAN	Hybrid	Semi-supervised	Various	Path Planning [52], IoT Wireless Classifcation [53], Adversarial Video Classification [54], Vehicle Re-identification [55], Intelligent Transport System Analytics [56]
DQN	Value-based	Reinforcement Learning	Various	Rounting for Crowd Management [57], Energy Efficient Data Collection [58], Low Latency Microgrid Communication [59], Resource Allocation at the Edge [60]
A3C	Actor-critic	Reinforcement Learning	Various	Adaptive Video Streaming [61], Network Topology Mangement [62], Content Caching [63], Service Selection [64], Traffic Signal Control [65]

QoS requirements needed for AR applications as they require reduced latency and jitter meaning they need to be deployed at the edge of the network.

B. DEEP LEARNING

Table 3 shows the variety of deep learning models and the applications to which they can be applied. These models have become very popular in recent years as they have shown increased accuracy using big data, with most of the research being published in the last three years. In this subsection we give an overview of several common deep learning architectures as well as the urban intelligence applications to which they have been applied.

1) AUTOENCODER (AE)

An AE is a type of deep learning algorithm that consists of an input layer, an output layer and one or more hidden layers connecting them, with the output layer having the same number of nodes as the input layer. AEs have two main components an encoder that receives the input data and transforms it to a lower dimensional latent variable and a decoder that uses this latent variable to try to reconstruct the original input. AEs are useful for unsupervised learning tasks as the training procedure tries to minimise the error between the input and output, which allows it to notice any unexpected changes in the output. This is useful for a number of applications in urban intelligence such as detection of road traffic accidents [26], fault diagnosis in city devices and machines and object tracking [24]. They can also be used to make accurate QoS predictions at the edge of the network [25], which allows for the composition of more reliable urban applications that have stricter QoS requirements in terms of delay, throughput and jitter.

2) RECURRENT NEURAL NETWORK (RNN)

There are many tasks in a smart city that have a sequence or time series component such as traffic congestion, electricity grid demand, network quality etc. In such applications, a deep learning model is needed that is capable of learning these dependencies, as a traditional feed-forward neural network assumes no time dependency between input and output layers. RNNs and subsequently LSTMs as an extension have been developed to address this issue. The data used to train an RNN consists of both the current training data and the feedback loop in the RNN network that returns the current output as an input for the next data point. The network is trained using an extension of the original backpropagation algorithm, called backpropogation through time [65]. RNNs have been used for a number of sequence prediction tasks such as car park occupancy prediction [27], energy demand prediction [28], metro density prediction [29] and waste generation prediction [30].

3) RESTRICTED BOLTZMANN MACHINE (RBM)

RBM are a stochastic neural network consisting of two main layers: a visible layer that has the input that is known through the data and a hidden layer that has the latent variables. The restricted in a Restricted Boltzmann Machine refers to the connectivity of the neurons compared to a traditional Boltzmann machine. RBMs build a bipartite graph where each visible neuron is connected to all hidden neurons and each hidden neurons is connected to all visible neurons. The restriction is that there is no connection between any two units on the same layer. RBMs also contain a bias unit that is connected to all the visible and hidden neurons. RBMs are used as a building block to create Deep Belief Networks. They can be used in a variety of urban intelligence applications, such as urban object recognition [31], intrusion detection [32], object classification [33] and city event detection from Twitter [34].

4) DEEP BELIEF NETWORK (DBN)

DBNs are a type of generative neural network composed of multiple layers of latent variables with connections between the layers but not between units in the same layer and a visible layer corresponding to the inputs [66]. DBN can be used in a range of applications to extract a hierarchical representation of the training data. Training a DBN is performed in a layer by layer fashion. By adding a classification layer such as softmax to the network it can also be used for prediction tasks. This makes it useful for a range of urban intelligence application such as network traffic assignment [35], indoor localisation [36] and facial expression recognition [37].

5) LONG SHORT TERM MEMORY (LSTM)

LSTMs developed from RNNs due to some of the training problems. LSTMs improved on the initial RNN architecture by saving and retrieving data over a long period with an in built error carousel and an explicit gating mechanism [67].

This change in training method addressed many of the difficulties of training RNNs, where the dynamics of backpropagation error would increase or decrease to the point that the gradients in the RNN would explode or vanish. LSTMs also introduced a new gating mechanisms compared to the basic RNN approach that would compute a weighted sum of the inputs and apply a nonlinear function. This gating mechanism allows the LSTM to decide whether or not to keep the existing memory or to overwrite it. This means that if the LSTM unit detects an important feature from an input sequence, this feature will be carried over a long distance and used to capture long-distance dependencies. This allows them to be used for a number of forecasting applications such as traffic flow prediction [38], QoS forecasting [39], air quality prediction [40], particle matter forecasting [41] and energy load forecasting [42].

6) CONVOLUTIONAL NEURAL NETWORK (CNN)

For image and sound based tasks, standard deep neural networks with dense connections between the layers are hard to train and encounter a number of problems when trying to scale. Traditional dense connections do not have the translation-invariance property, so any slight change in the size or placement of an image would be difficult to detect. CNNs solve this problem by using multiple hidden convolutional layers that extract high level features. Convolutional layers consist of learnable parameters, called filters that go through the whole image (e.g., in an image, it goes across the width and length) and calculates the inner product of the input and the filter. This leads to a feature map of the filter. Another building block of convolution networks, which can be seen in the VGG-16 model in Figure 6, is a pooling layer, which operates on the feature maps. The pooling layer reduces the spatial size of the representation to reduce the chance of overfitting and cuts down the number of parameters. Max pooling is an approach that divides the space into individual regions and picks the maximum value for each region. CNNs also make heavy use of the ReLu activation function of the form $f(x) = max(0, x)$ as this leads to faster training without effecting the accuracy of the network [68]. CNNs have a huge range of applications in urban intelligence, with some of the most recent papers published in the last three years using CNNs for license plate recognition [43], pedestrian detection [44] (tested on our local images in Figure 2), object detection [45], street cleaning [46], noise recognition [47] and predictive driving control [48].

7) VARIATIONAL AUTOENCODER (VAE)

VAEs are a generative neural network model that have weak assumptions about the structure of data, while allowing for fast training through the use of backpropagation [69]. They can also be used for semi-supervised learning making them useful for urban applications with diverse and scarcely labelled data [70]. Such examples include house number classification [49], motion prediction [49] and anomaly detection [50].

FIGURE 2. Pedestrian detection using pre-trained model.

8) GENERATIVE ADVERSARIAL NETWORK (GAN)

GANs consist of two neural networks, a generative and discriminative network that compete against each other to produce synthetic high quality data [71]. The generator is responsible for generating new data after the distribution has been learned from the training dataset. The discriminator is then responsible for discriminating between the real data from the training set and the fake data being produced by the generator. The generative network is trained to produce data that is deceiving for the discriminator. Thus the two networks are in competition as training occurs. The objective function used to train GANs is based on minimax games where one network tries to maximise the value of a function and the other tries to minimise it. GANs have a wide range of urban applications such as path planning [51], IoT wireless classification [52], adversarial video classification [53], vehicle re-identification [54] and intelligent transport system analytics [55].

9) DEEP-Q-NETWORK (DQN)

DQN is a deep reinforcement learning algorithm that builds from Q-learning, where the goal was learn a policy, which tells an agent what action to take under certain circumstances by learning a table of states and actions $Q(s, a)$. The DQN approach uses a neural network instead of the Q learning table to approximate a large number of Q values. The neural network is referred to as the approximation function denoted as $Q(s, a; \theta)$, where θ represents the trainable weights of the network [72]. DQN has huge potential for controlling a number of services in a smart city as it can continuously update to new data. In the last two years it has already been applied to routing for crowd management [56], energy efficient data collection [57], low latency microgrid communication [58] and resource allocation at the edge [59].

10) ASYNCHRONOUS ACTOR-CRITIC AGENTS (A3C)

A3C has recently been introduced as a competitor to DQN [73]. A3C uses an alternative approach to DQN that utilises multiple incarnations of an agent represented by a

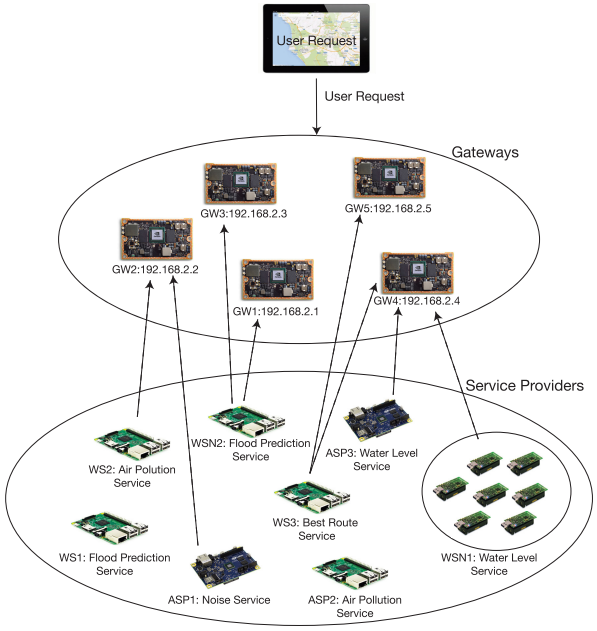


FIGURE 3. Small scale demonstration.

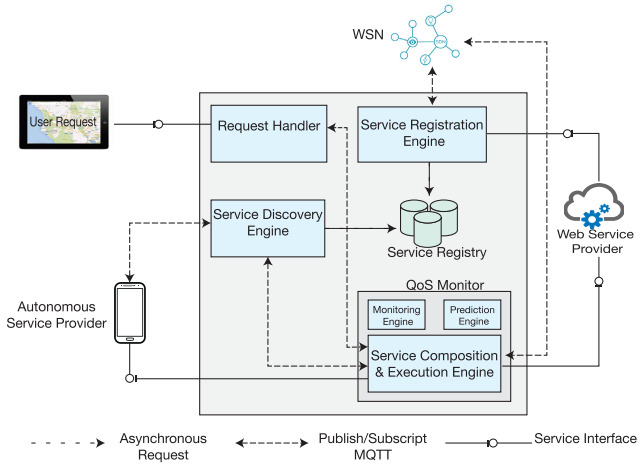


FIGURE 4. Middleware architecture.

neural network interacting with an environment to learn more effectively. A3C contains multiple worker agents in a global network, which each have their own network parameters. Each of these agents can interact with an individual copy of the environment in parallel with other agents who have their own separate environments. This allows for increased speedup and also increases the overall experience available for training as agents can be more diverse. These networks have already been applied to adaptive video streaming [60], network topology management [61], content caching [62], service selection [63] and traffic signal control [64].

III. DEEP EDGES

Deep edges combine the increased accuracy of deep learning models with the reduced latency, increased bandwidth and privacy of edge networks [7]. The ability to train and update these models at the edge of the network has not been possible before as devices capable of updating and training these networks at the edge have only been developed in recent years, such as the Nvidia Jetson Tx2. These edge device can provide effective communication and increased AI at the edge by arranging them as a network as shown in Figure 3. At the edge of the network these devices can use data from the large number of service providers in a smart city to add auxiliary information to their models. Here the embedded GPU devices (Jetson Tx2) have a number of services registered on them that provide real-time data for air pollution, water levels and traffic information in the city.

The additional processing power available from the embedded GPUs at the edge compared to traditional IoT gateways (e.g., Raspberry Pis and Intel Galileos) allows the devices to run more complex prediction models for the QoS of services in the environment [74], [25], [39]. These predictions can be used as part of a middleware architecture to compose more reliable services even in dynamic environments. Figure 4 shows a middleware architecture that allows the QoS predictions in a prediction engine to be used by a Service Composition & Execution Engine (SCEE). The main components are the Request Handler (RH), the Service Registration Engine (SRE), the Service Discovery Engine (SDE), the QoS Monitor and the Service Composition & Execution Engine. The RH establishes a request/response communication channel with the user and forwards the request to the other middleware components. The SRE registers the available services in the environment using consumer feedback [75]. The SDE uses the backward-planning algorithm to identify the concrete services, which can be used to satisfy the request and sends this list of services to the SCEE. The QoS monitor is used to monitor these services and can forecast when a service is about to degrade in quality [39] and predict possible candidate services to switch to when this degradation happens [74], [25]. The SCEE will use these services to create a response for the request using a stigmergic service composition algorithm [76]. This improves the QoS for service-based urban intelligence applications.

A. TRANSFER LEARNING

The goal of transfer learning is to improve learning in the target task by levering knowledge from the source task [77]. It can provide several benefits, such as improving baseline performance, speeding up overall model development and training time and also getting an overall improved and superior model performance compared to building the model from scratch. This is especially important in deep learning models that have a large training time. Transfer learning can be applied to all the Deep Learning Models discussed in Table 3. Access to open evaluation datasets, such as the Caltech Pedestrian Dataset [78] makes these models more easily available. Table 4 shows the five best performing models for pedestrian detection on this dataset, with code available.

TABLE 4. Pedestrian detection models.

Rank	Method	Reasonable Miss Rate	Extra Training Data	Paper Title	Year	Paper	Code
1	CSP + CityPersons dataset	3.8	Yes	High-level Semantic Feature Detection: A New Perspective for Pedestrian Detection	2019	[80]	https://github.com/liuwei16/CSP
2	RepLoss + CityPersons dataset	4	Yes	Repulsion Loss: Detecting Pedestrians in a Crowd	2017	[81]	https://github.com/bailvwangzi/repulsion_loss_ssd
3	CSP	4.5	No	High-level Semantic Feature Detection: A New Perspective for Pedestrian Detection	2019	[80]	https://github.com/liuwei16/CSP
4	ALFNet + CityPersons dataset	4.5	Yes	Learning Efficient Single-stage Pedestrian Detectors by Asymptotic Localization Fitting	2018	[82]	https://github.com/VideoObjectSearch/ALFNet
5	RepLoss	5	No	Repulsion Loss: Detecting Pedestrians in a Crowd	2017	[83]	https://github.com/bailvwangzi/repulsion_loss_ssd

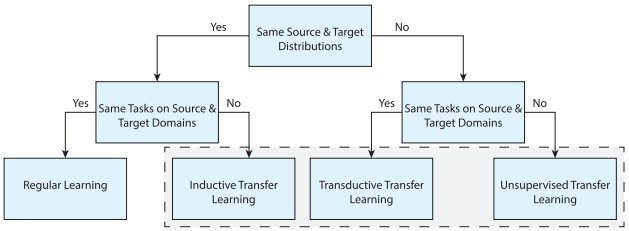
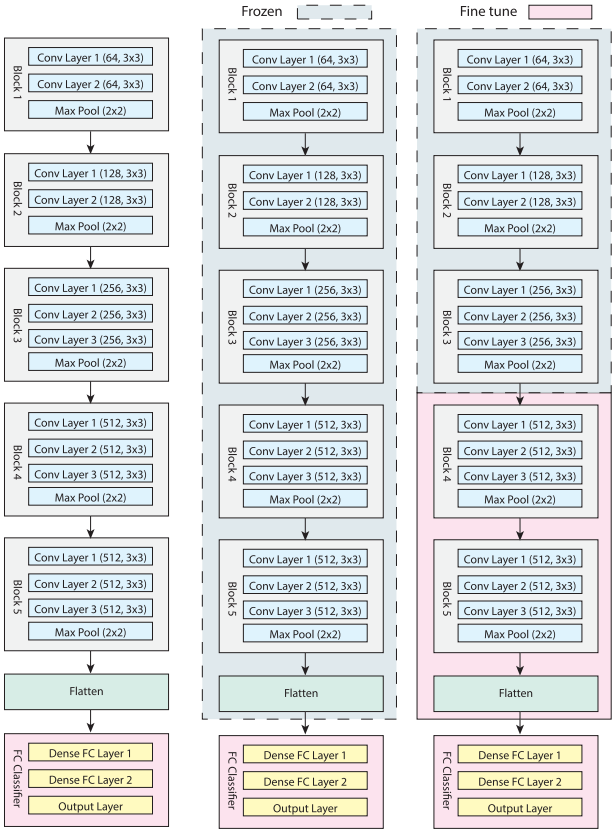


FIGURE 5. Transfer learning categories.

Figure 5 shows the different categories of transfer learning based on the relationship between the source and target distributions. The most simple case is regular learning where the source and target have the same distributions and are required to perform the same tasks. When the source and target have the same distribution or are in the same domain but the tasks that they are required to perform are different this is called inductive transfer learning. This category can further be broken down depending upon whether the source domains contain labelled data or not: if a lot of labelled data in the source domain are available then it is multi-task learning and if there is no labelled data from the source domain then it is self-taught learning [83]. When the source and target distribution are not the same but the tasks are similar it is called transductive transfer learning. In this situation, no labelled data in the target domain are available while a lot of data in the source domains are available. The final category is unsupervised transfer learning where there is a difference in both the source and target distribution and tasks. This category focuses on solving unsupervised tasks in the target domain such as clustering [84] and dimensionality reduction [85], with no labelled data available in the source and target domains in training. A linear cost function can be used to minimise the difference between the source and target domain distribution in unsupervised transfer learning [86], [87]. In this paper we focus on two specific transfer learning techniques of feature extraction and fine-tuning. We also use data augmentation to increase the amount of training data.

1) DATA AUGMENTATION

Generating artificial data based on existing observations is a technique in machine learning to control overfitting, improve model accuracy and generalisation [88]. The idea behind this technique also known as data augmentation is that we follow a set process, taking existing data, such as images from our training set and applying some image transformation operations on them, such as translation, zooming, shearing and rotation to produce a new set of alternative images. The randomness of the process means that we do not get the same



(a) VGG-16 Model [8] (b) VGG-16 Model Feature Extractor (c) VGG-16 Model Fine Tuning

FIGURE 6. Deep transfer learning approaches.

images each time. This helps to stop the deep learning model from overfitting on the local training data. In our experiments we use the ImageDataGenerator class[2] from Keras to provide a number of transformations for generating new images, such as: zooming, rotation, translation, randomly flipping images horizontally, and filling new pixels with their nearest surround pixels.

2) FEATURE EXTRACTION

Deep learning models are layered architectures that learn different features at different layers. These layers are finally connected to a last layer, which is usually fully connected in the case of classification to get the final output. This layered architecture allows us to utilize pre-trained networks without a final layer as a fixed feature-extractor for other tasks. Figure 6 shows the different transfer learning

[2]https://keras.io/preprocessing/image/#imagedatagenerator-class

approaches that can be applied to this task. Figure 6a shows the VGG-16 model that can be downloaded directly to an edge device. Figure 6b shows the VGG model as a feature extractor where we freeze (fix weights and don't train) all the blocks of convolutions layers and flattening layer in the dashed blue box. We only update the fully connected classifier block at the end of the model. This allows the new model in this case to transform the image from a new domain task into a large dimension vector based on hidden states, thus enabling us to extract features from a new domain task using the source domain. This is one of the most widely used methods of transfer learning using deep neural networks.

3) FINE TUNING

In fine tuning the weights of the last few layers of the network are updated as shown in Figure 6c with a pink box and trained as well as the fully connected layers at the end of the model, for the classification task. This means that this method is slightly more resource intensive as we have to train some of the previous layers. As deep neural networks are layered with the initial layers capturing the most basic features such as edges and the later layers capturing more specific details about the task, we can freeze some of the first blocks and update the later ones. In Figure 6c we freeze the first three layers in the dashed blue box, while fine tuning the last two blocks to suit the task. This allows us to use the knowledge in terms of the overall architecture of the network and use its states as the starting point for our retraining step allowing us to achieve better performance in less time.

One of the problems with updating a model using fine-tuning is that that some of the parameters in the non-frozen layers have to be updated to solve the new problem. This can overwrite parameters that the network has learned before and lead to ''catastrophic forgetting'' of the knowledge that was previously acquired [89], [90]. Previous approaches to preventing catastrophic forgetting have used an ensemble of neural networks. When there is a new task, the algorithm creates a new network and shares the representation between the tasks [91], [92]. However, these approaches are not suitable for the edge due to the space and complexity restrictions as the number of networks increases linearly with the number of new tasks to be learned.

In this paper instead of using a traditional optimisation approach, such as RMSProp [93] we use a cyclical learning rate (CLR) [94]. Specifically, we use an exponential cyclical learning rate (ECLR), which can be seen in Figure 7. The learning rate is varied between the base learning rate and the max learning rate, with the value changing from one bound to the other after 500 training iterations. Updating the layers at different learning rates allows the network to initially come out of any saddle points or local minima in the base network. The max learning weight then decreases exponentially as the number of training iterations increase to avoid drastic updates, which can lead to divergent behaviour.

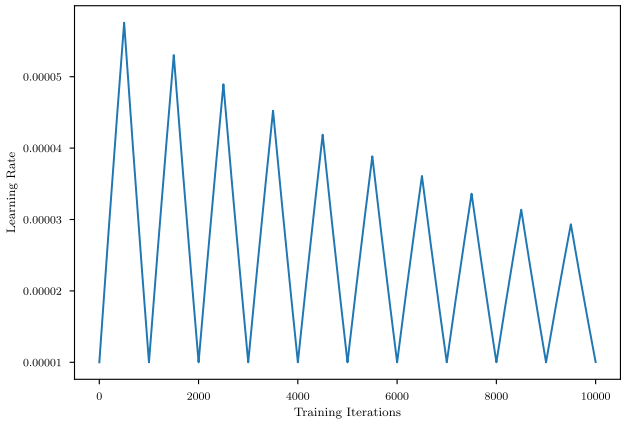


FIGURE 7. Exponential cyclical learning rate.

IV. EXPERIMENTAL SETUP

In our experiments, we test the effect that different network topologies have on the performance of the packet loss and response time. We consider a number of devices from alternative architectures including local edge devices and cloud-based approaches. A Nvidia Jetson Tx2 is used as an edge device connected to a router one hope away, a Rapberry Pi 3 Model B+ (RPi) is connected in a mobile ad hoc network one hop away and a desktop computer is connected to an alternative network three hops away. We consider a range of more traditional cloud-based approaches using a number of Amazon EC2 instances in five locations: Dublin, Paris, Frankfurt, Bahrain and Seoul. This provides a global perspective of response times from data centres located around the world. We conduct the network test in Trinity College Dublin and record the round trip time (RTT) between the client and server obtained through ICMP ping messages. To obtain a comprehensive distribution of the response time values 5000 ping messages are sent to each of the servers at one second intervals. The device are connect using a Wi-Fi-based network and the RTT is recorded in milliseconds.

We also implement the neural network architectures in Figure 6 and show how the different training methods effect the accuracy of the final model. We test how the training and validation loss improve over the epochs by using image augmentation. We use the Dogs vs. Cats[3] dataset to evaluate the accuracy of our transfer learning algorithms. We also create a basic CNN model trained only on the dogs vs cats dataset to allow for comparison between a traditional learning approach and transfer learning. The basic CNN model has three convolutional layers of size 16, 64 and 128 with max pooling after each convolution. This is then followed by a flattening layer and two fully connected layers. The effects of transfer learning are also tested on two other much deeper models, Inception ResNet V2 [95] and Xception [96].

[3]https://www.kaggle.com/c/dogs-vs-cats/

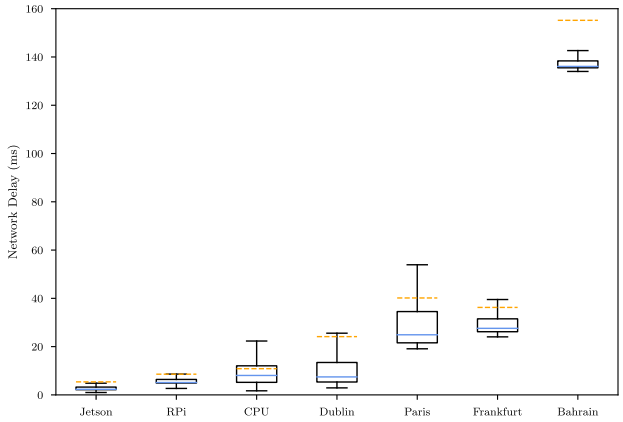


FIGURE 8. Network delay times.

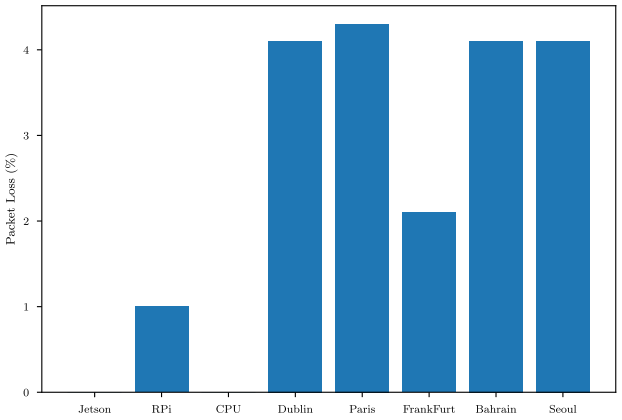


FIGURE 9. Network packet loss.

A. METRICS

We use a number of classification metrics to comprehensively show how the classification models perform. The metrics are based on the number of True Positives (TP), False Positives (FP), True Negatives (TN) and False Negatives (FN) and are defined as:

$$Accuracy = \frac{True\ Positive + True\ Negative}{TP + TN + FP + FN} \quad (1)$$

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \quad (2)$$

$$TPR/Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \quad (3)$$

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \quad (4)$$

We also use a receiver operating curve (ROC) that provides a clear visual illustration of the overall performance as well as the area under the curve (AUC), which measures the degree of separability (how much a model can distinguish between classes). A higher AUC means the model is better at classification, with an AUC near 1 showing a good measure of separability, while a measure of 0.5 shows no class separation. The curve uses the true positive rate (TPR) defined previously and the false positive rate (FPR) defined as:

$$FPR = \frac{False\ Positive}{False\ Positive + True\ Negative} \quad (5)$$

The training time per epoch of each of the models is also evaluated on a range of devices: a Titan V with 5120 CUDA Cores and 12GB RAM requiring 250W power, a Jetson Tx2 with 256 CUDA Cores and 8GB RAM requiring 12W, and a Raspberry Pi 3 Model B with a Broadcom VideoCore IV GPU and 1GB of RAM requiring 1.2W.

V. RESULTS

A. NETWORK DELAY

Figure 8 shows the network delay for the device configurations that were described in Section IV. A box plot is used to show the distribution of network delay for each of the configurations. The blue line in the box plot indicates the median response time and the average delay is indicated by the dashed orange line. Figure 8 shows that the best performing configuration is the Jetson connected one hop away from with a median delay of 2.3ms and mean of 5.39ms. The raspberry pi (RPi) connected one hop away in a MANET has a slightly longer response time with a median delay of 5.1ms and mean of 8.5ms. The local desktop (CPU) that is connected to a different network has a slightly increased delay with a median of 8.0ms and mean of 10.8ms. The data centres are then used to evaluate the impact of using a cloud architecture in different locations. The Amazon Dublin instance has the lowest response time with a median of 7.5ms and mean of 24.1ms, this can be seen as a special case of a cloud provider as the data centre is located close to the Trinity College Dublin, where the test was conducted. However, even in this special case the average delay is larger than the tolerable delay required by applications such as augmented reality. To fully evaluate the impact of the data centre location we evaluate the response time of four other cloud locations two in Europe (Paris, Frankfurt), one in the Middle East (Bahrain) and one in Asia Pacific (Seoul).

The median response time for the data centre in Paris was 24.905ms with a mean of 40.2ms and the data center in Frankfurt has a median delay of 27.5ms with a mean of 36.2ms. However, in terms of geographical locations this is still quite optimistic as Dublin is relatively close with good communication links. The data centre in Bahrain had a much increased response time with a median of 136.0ms and a mean of 155.2ms. This can be seen be the noticeable jump in the results in Figure 8. The results for Seoul are not plotted in this figure as they are much larger with a median of 334.1ms and a mean of 342.8ms.

B. PACKET LOSS

Figure 9 shows the percentage of the total 5000 packets that were lost during the experimentation. The first noticeable difference is the change in packet loss between the first three local configurations and the cloud-based approaches. The local approaches of Jetson and CPU both have a 0% packet loss with the raspberry pi having a 1% packet loss. The packet loss for the data centre-based approaches were

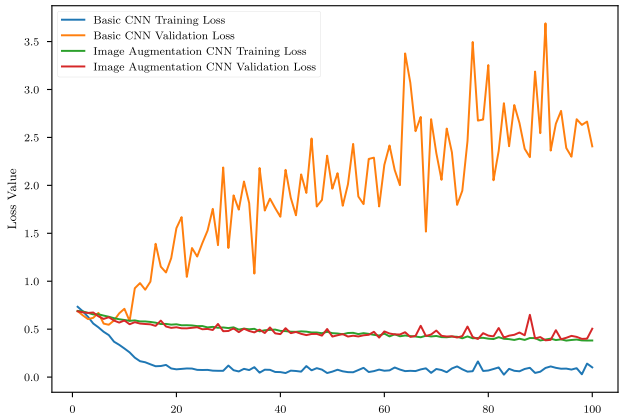


FIGURE 10. Augmented data loss.

4.1% for Dublin, 4.3% for Paris, 2.1% for Frankfurt, 4.0% for Bahrain and 4.1% for Seoul. The packet loss matters for streaming services such as collision detection or CCTV footage analysis as a drop in one of the frames can cause a person or object not to be detected. For critical applications such as collision avoidance in autonomous vehicles it is important to have a low packet loss to avoid frames not being processed.

C. DATA AUGMENTATION

One of the problems with training on data at the edge is that it can lead to overfitting on the training data. This can lead to models achieving very low training loss on the training data set with a larger validation loss when the model is used on the validation data set. Figure 10 shows this clearly with the Basic CNN model achieving the best training loss, however we can see that the validation loss increases suggesting that the model is overfitting on the training data. This figure shows the benefit of data augmentation as the model that is trained using image augmentation has almost the same validation and training loss. This shows that the model is not overfitting on the training data and is able to perform better on the unseen validation test set compared to the basic model.

D. TRANSFER LEARNING ACCURACY

Table 5 and Figure 11 show the overall results of the classification models. Table 5 show the improvement in accuracy, precision, recall and F1 score achieved by using transfer learning with pretrained models over training a model from scratch. Image augmentation shows less improvement on the feature extraction model compared to the Basic CNN model. This could be because many of the lower layers of this model are frozen, which may help to stop overfitting on the training data.

The feature extraction and fine tuning of the pretrained VGG model are both improved through the use of the Exponential cyclical learning rate (ECLR) compared to the existing RMSProp method. The fine tuning model for VGG, which updates the weights of the last two convolutional blocks as well as the fully connected dense layers at the end of the

TABLE 5. Transfer learning results.

Model	Accuracy	Precision	Recall	F1 score
Basic CNN	0.779	0.779	0.779	0.779
Image Augmentation CNN	0.798	0.817	0.798	0.795
Feature Extraction VGG	0.883	0.887	0.883	0.882
Feature Extraction Image Augmentation VGG	0.889	0.892	0.889	0.888
Feature Extraction Image Augmentation VGG ECLR	0.891	0.892	0.891	0.891
Fine Tuning Image Augmentation VGG	0.954	0.954	0.954	0.954
Fine Tuning Image Augmentation VGG ECLR	0.956	0.956	0.956	0.956
Featue Extraction Image Augmentation Xception ECLR	0.957	0.957	0.957	0.957
Fine Tuning Image Augmentation InceptionResNet V2	0.958	0.958	0.958	0.958
Fine Tuning Image Augmentation InceptionResNet V2 ECLR	0.959	0.959	0.959	0.959
Feature Extraction Image Augmentation Xception	0.960	0.960	0.960	0.960
Feature Extraction Image Augmentation InceptionResNet V2	0.961	0.961	0.961	0.961
Fine Tuning Image Augmentation Xception	0.964	0.964	0.964	0.964
Feature Extraction Image Augmentation InceptionResNet V2 ECLR	0.965	0.965	0.965	0.965
Fine Tuning Image Augmentation Xception ECLR	0.967	0.967	0.967	0.967

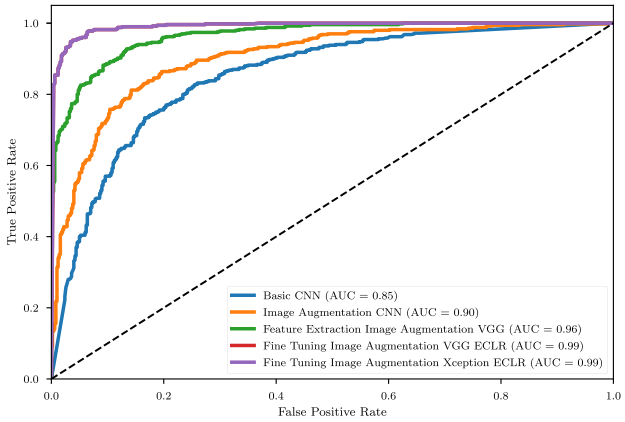


FIGURE 11. Receiver operating curve (ROC) curve.

model as shown in Figure 6c shows further improvement in classification accuracy compared to the feature extraction and basic CNN model. In Table 5 we can see that accuracy has improved to 0.956 using fine tuning with ECLR from 0.779 in the basic model a 0.177 increase.

Using larger pretrained models can help to further improve the final accuracy of the model after transfer learning. The Xception and InceptionResNet model show further improvement compared to the fine tuned VGG model with ECLR. The two most accurate models both use ECLR with the most accurate model being the Fine Tuning Image Augmentation Xception ECLR model. This model achieves an F1 score of 0.967 compared to 0.956 for the VGG model a 0.011 increase.

Figure 11 gives an intuitive view into the improvement of the model using image augmentation, feature extraction and fine tuning for the models. We can see how the AUC as shown in the legend increases from the basic model with AUC = 0.85 to the fine tuning VGG model with AUC = 0.99. The most accurate Xception model in Table 5 also achieve an AUC = 0.99. Figure 11 shows the benefit of transfer learning in this case as the transfer learning models achieve better baseline performance at the start, better efficiency gains (higher slope) and better final performance as shown by the AUC value.

E. TRANSFER LEARNING PARAMETERS

Table 6 shows the number of total parameters, trainable parameters and depth for each of the models. For the Basic CNN model we have 3,706,113 parameters and have to train all of these from scratch. Although image augmentation

TABLE 6. Transfer learning parameters.

Model	Total Parameters	Trainable Parameters	Depth	Titan V	Jetson Tx2	Raspberry Pi
Basic CNN	3,706,113	3,706,113	9	2	20	N/A
Image Augmentation CNN	3,706,113	3,706,113	9	10	36	N/A
Feature Extraction VGG	19,172,673	4,457,985	23	1	30	N/A
Feature Extraction Image Augmentation VGG	19,172,673	4,457,985	23	11	87	N/A
Fine Tuning Image Augmentation VGG	19,172,673	17,437,185	23	11	121	N/A
Fine Tuning Image Augmentation InceptionResNet V2	61,678,305	7,341,569	572	12	N/A	N/A
Feature Extraction Image Augmentation Xception	47,339,561	26,478,081	126	10	N/A	N/A
Feature Extraction Image Augmentation InceptionResNet V2	61,678,305	11,644,385	572	10	N/A	N/A
Fine Tuning Image Augmentation Xception	47,339,561	33,266,465	126	12	N/A	N/A

does not increase the number of parameters, it does increase the training time per epoch by increasing the size of the training data set. Transfer learning gives access to a much larger model, such as VGG with 19,172,673 total parameters and various amounts of trainable parameters depending on whether the base model is used for feature extraction or fine tuning. The trainable parameters for the feature extraction model are 4,457,985, fine tuning increases the amount of parameters dramatically to 17,437,185. Xception and InceptionResNetV2 increases the total number of parameters available for transfer to 47,339,561 and 61,678,305. These models are very deep with Xception having 126 layers and InceptionResNetV2 having 572 layers.

Table 6 also show the training time per epoch for a range of device categories. Titan V is a cloud GPU that requires a large amount of power, Jetson Tx2 is an embedded GPU edge device and Raspberry Pi is a low powered IoT device. The Raspberry Pi is not capable of training the Basic CNN or performing feature extraction or fine tuning due to lack of resources such as only having 1GB memory. This is indicated by N/A in the table. The Jetson Tx2 edge device is capable of training the Basic CNN and performing transfer learning on the VGG model. However, when the models get very deep (>125 layers) the Jetson recieves a resource exhaustion error for limited memory. The Tintan V cloud GPU is capable of training and updating all the models. The additional trainable parameters can lead to a better final model as seen in Figure 11 and Table 5 but come at the cost of increased training resources. There is a diminishing return on the number of parameters with the Fine Tuning Image Augmentation VGG model capable of being updated at the edge with 19,172,673 parameters achieving an F1 Score of 0.956 and the best performing Fine Tuning Image Augmentation Xception with 47,339,561 parameters and an F1 Score of 0.967. In Figure 11 these models both have an AUC of 0.99.

VI. CONCLUSION

The results of the experiments have provided a number of insights into using a deep edge architecture. The network delay experiments showed a dramatic reduction to an average of 5.39ms on the Jetson at the edge compared to the cloud approaches in Paris that were 40.2ms and Bahrain that were 155.2ms. The low network delay and packet loss at the edge allows for a range of modern urban reasoning applications such as collision detection and augmented reality. This improves the privacy and security of the data used to create the model.

IoT devices are not suitable for training or updating large deep learning models. The training of the large models that can be used for transfer learning should take place in the cloud with access to large GPUs such as a Titan V that can train a model in reduced time as shown by the results in Table 6. Edge devices such as a Jetson Tx2 can be used to update these models at the edge one hop away from data generation. This allows for increased security and privacy of local user data when updating the model. As show in Section II there are a range of urban intelligence applications that can benefit from the reduced latency at the edge combined with the improvements in accuracy that we have shown using transfer learning with an exponential cyclical learning rate.

ACKNOWLEDGMENT

The Jetson Tx2 and Titan V used for this research were donated by the NVIDIA Corporation.

REFERENCES

[1] *The world's cities in 2016*, United Nations, New York, NY, USA, 2016.
[2] H. E. A. Schaffers, "Smart cities and the future Internet: Towards cooperation frameworks for open innovation," in *The Future Internet*. Berlin, Germany: Springer, 2011, pp. 431–446.
[3] H. Bauer, M. Patel, and J. Veira, *The Internet of Things: Sizing Up the Opportunity*. New York, NY, USA: McKinsey, 2014.
[4] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies, "The case for VM-based cloudlets in mobile computing," *IEEE Pervasive Comput.*, vol. 8, no. 4, pp. 14–23, Oct. 2009.
[5] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, *Fog Computing: A Platform for Internet of Things and Analytics*. Cham, Switzerland: Springer, 2014, pp. 169–186.
[6] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Netw.*, vol. 61, pp. 85–117, Jan. 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002135
[7] G. White and S. Clarke, "Smart cities with deep edges," in *Proc. ECML PKDD Workshops*. Cham, Switzerland: Springer, 2019, pp. 53–64.
[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*. [Online]. Available: https://arxiv.org/abs/1409.1556
[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. CVPR*, 2009.
[10] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of Things for smart cities," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22–32, Feb. 2014.
[11] S. A. Salam, S. A. Mahmud, G. M. Khan, and H. S. Al-Raweshidy, "M2M communication in smart grids: Implementation scenarios and performance analysis," in *Proc. IEEE Wireless Commun. Netw. Conf. Workshops (WCNCW)*, Apr. 2012, pp. 142–147.
[12] M. Perez, S. Xu, S. Chauhan, A. Tanaka, K. Simpson, H. Abdul-Muhsin, and R. Smith, "Impact of delay on telesurgical performance: Study on the robotic simulator dv-trainer," *Int. J. Comput. Assist. Radiol. Surgery*, vol. 11, no. 4, pp. 581–587, 2016.

[13] G. Netw, "Unlocking commercial opportunities from 4G evolution to 5G," GSMA, London, U.K., Tech. Rep., 2016.

[14] T. A. S. Nielsen and S. Haustein, "On sceptics and enthusiasts: What are the expectations towards self-driving cars?" *Transp. Policy*, vol. 66, pp. 49–55, Aug. 2018.

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

[16] P. Simoens, Y. Xiao, P. Pillai, Z. Chen, K. Ha, and M. Satyanarayanan, "Scalable crowd-sourcing of video from mobile devices," in *Proc. 11th Annu. Int. Conf. Mobile Syst., Appl., Services (MobiSys)*. New York, NY, USA: ACM, 2013, pp. 139–152, doi: 10.1145/2462456.2464440.

[17] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-defined networking (SDN) and distributed denial of service (DDoS) attacks in cloud computing environments: A survey, some research issues, and challenges," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, pp. 602–622, 1st Quart., 2016.

[18] S. Gibbs. *Typo Blamed for Amazon's Internet-Crippling Outage*. Accessed: Jan. 4, 2020. [Online]. Available: https://www.theguardian.com/technology/2017/mar/03/typo-blamed-amazon-web-services-internet-outage

[19] S. Kyriazakos *et al.*, "eWALL: An open-source cloud-based eHealth platform for creating home caring environments for older adults living with chronic diseases or frailty," *Wireless Pers. Commun.*, vol. 97, no. 2, pp. 1835–1875, Nov. 2017.

[20] J. Albrecht, "How the GDPR will change the world," *Eur. Data Protection Law Rev.*, vol. 2, no. 3, pp. 287–289, 2016.

[21] N. Davies, N. Taft, M. Satyanarayanan, S. Clinch, and B. Amos, "Privacy mediators: Helping IoT cross the chasm," in *Proc. HotMobile*. New York, NY, USA: ACM, 2016, pp. 39–44.

[22] G. White, C. Cabrera, A. Palade, and S. Clarke, "Augmented reality in IoT," in *Proc. Service-Oriented Comput. (ICSOC) Workshops*. Cham, Switzerland: Springer, 2019, pp. 149–160.

[23] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui, "Mobile augmented reality survey: From where we are to where we go," *IEEE Access*, vol. 5, pp. 6917–6950, 2017.

[24] L. Wang, T. Liu, G. Wang, K. L. Chan, and Q. Yang, "Video tracking using learned hierarchical features," *IEEE Trans. Image Process.*, vol. 24, no. 4, pp. 1424–1435, Apr. 2015.

[25] G. White, A. Palade, C. Cabrera, and S. Clarke, "Autoencoders for QoS prediction at the edge," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, Kyoto, Japan, Mar. 2019.

[26] D. Singh and C. K. Mohan, "Deep spatio-temporal representation for detection of road accidents using stacked autoencoder," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 3, pp. 879–887, Mar. 2019.

[27] A. Camero, J. Toutouh, D. H. Stolfi, and E. Alba, "Evolutionary deep learning for car park occupancy prediction in smart cities," in *Learning and Intelligent Optimization*. Cham, Switzerland: Springer, 2019, pp. 386–401.

[28] M. S. Munir, S. F. Abedin, M. G. R. Alam, D. H. Kim, and C. S. Hong, "RNN based energy demand prediction for smart-home in smart-grid framework," in *Proc. Korea Softw. Congr.*, 2017, pp. 437–439.

[29] V. C. Liang, R. T. Ma, W. S. Ng, L. Wang, M. Winslett, H. Wu, S. Ying, and Z. Zhang, "Mercury: Metro density prediction with recurrent neural network on streaming CDR data," in *Proc. IEEE 32nd Int. Conf. Data Eng. (ICDE)*, May 2016, pp. 1374–1377.

[30] A. Camero, J. Toutouh, J. Ferrer, and E. Alba, "Waste generation prediction in smart cities through deep neuroevolution," in *Proc. Ibero-Amer. Congr. Inf. Manage. Big Data*. Springer, 2018, pp. 192–204.

[31] J. Aryal and R. Dutta, "Smart city and geospatiality: Hobart deeply learned," in *Proc. 31st IEEE Int. Conf. Data Eng. Workshops*, 2015, pp. 108–109.

[32] A. Elsaeidy, K. S. Munasinghe, D. Sharma, and A. Jamalipour, "Intrusion detection in smart cities using Restricted Boltzmann Machines," *J. Netw. Comput. Appl.*, vol. 135, pp. 76–83, Jun. 2019.

[33] N. Sun, G. Han, K. Du, J. Liu, and X. Li, "Person/vehicle classification based on deep belief networks," in *Proc. 10th Int. Conf. Natural Comput. (ICNC)*, Aug. 2014, pp. 113–117.

[34] N. Farajidavar, S. Kolozali, and P. Barnaghi, "A deep multi-view learning framework for city event extraction from twitter data streams," 2017, *arXiv:1705.09975*. [Online]. Available: https://arxiv.org/abs/1705.09975

[35] J. Yang, Y. Han, Y. Wang, B. Jiang, Z. Lv, and H. Song, "Optimization of real-time traffic network assignment based on IoT data using DBN and clustering model in smart city," *Future Gener. Comput. Syst.*, to be published.

[36] J. Luo and H. Gao, "Deep belief networks for fingerprinting indoor localization using ultrawideband technology," *Int. J. Distrib. Sensor Netw.*, vol. 12, no. 1, Jan. 2016, Art. no. 5840916.

[37] M. Z. Uddin, M. M. Hassan, A. Almogren, A. Alamri, M. Alrubaian, and G. Fortino, "Facial expression recognition utilizing local direction-based robust features and deep belief network," *IEEE Access*, vol. 5, pp. 4525–4536, 2017.

[38] Y. Tian and L. Pan, "Predicting short-term traffic flow by long short-term memory recurrent neural network," in *Proc. IEEE Int. Conf. Smart City/SocialCom/SustainCom (SmartCity)*, Dec. 2015, pp. 153–158.

[39] G. White, A. Palade, and S. Clarke, "Forecasting QOS attributes using LSTM networks," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2018.

[40] I. Kök, M. U. Şimşek, and S. Özdemir, "A deep learning model for air quality prediction in smart cities," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, Dec. 2017, pp. 1983–1990.

[41] C.-J. Huang and P.-H. Kuo, "A deep cnn-lstm model for particulate matter ($PM_{2.5}$) forecasting in smart cities," *Sensors*, vol. 18, no. 7, p. 2220, Jul. 2018.

[42] W. Kong, Z. Y. Dong, Y. Jia, D. J. Hill, Y. Xu, and Y. Zhang, "Short-term residential load forecasting based on LSTM recurrent neural network," *IEEE Trans. Smart Grid*, vol. 10, no. 1, pp. 841–851, Jan. 2019.

[43] R. Polishetty, M. Roopaei, and P. Rad, "A next-generation secure cloud-based deep learning license plate recognition for smart cities," in *Proc. 15th IEEE Int. Conf. Mach. Learn. Appl. (ICMLA)*, Dec. 2016, pp. 286–293.

[44] J. Lwowski, P. Kolar, P. Benavidez, P. Rad, J. J. Prevost, and M. Jamshidi, "Pedestrian detection system for smart communities using deep convolutional neural networks," in *Proc. 12th Syst. Syst. Eng. Conf. (SoSE)*, 2017, pp. 1–6.

[45] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia, "Multi-view 3D object detection network for autonomous driving," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 1907–1915.

[46] C. Balchandani, R. K. Hatwar, P. Makkar, Y. Shah, P. Yelure, and M. Eirinaki, "A deep learning framework for smart street cleaning," in *Proc. IEEE 3rd Int. Conf. Big Data Comput. Service Appl. (BigDataService)*, Apr. 2017, pp. 112–117.

[47] J. Cao, M. Cao, J. Wang, C. Yin, D. Wang, and P.-P. Vidal, "Urban noise recognition with convolutional neural network," *Multimed Tools Appl*, vol. 78, no. 20, pp. 29021–29041, Oct. 2019.

[48] P. Drews, G. Williams, B. Goldfain, E. A. Theodorou, and J. M. Rehg, "Aggressive deep driving: Model predictive control with a CNN cost model," 2017, *arXiv:1707.05303*. [Online]. Available: https://arxiv.org/abs/1707.05303

[49] J. Walker, C. Doersch, A. Gupta, and M. Hebert, "An uncertain future: Forecasting from static images using variational autoencoders," in *Proc. Eur. Conf. Comput. Vis.* Amsterdam, The Netherlands: Springer, 2016, pp. 835–851.

[50] J. Sun, X. Wang, N. Xiong, and J. Shao, "Learning sparse representation with variational auto–encoder for anomaly detection," *IEEE Access*, vol. 6, pp. 33353–33361, 2018.

[51] M. Mohammadi, A. Al-Fuqaha, and J.-S. Oh, "Path planning in support of smart mobility applications using generative adversarial networks," 2018, *arXiv:1804.08396*. [Online]. Available: https://arxiv.org/abs/1804.08396

[52] C. Zhao, M. Shi, Z. Cai, and C. Chen, "Research on the open–categorical classification of the Internet-of-Things based on generative adversarial networks," *Appl. Sci.*, vol. 8, no. 12, p. 2351, Nov. 2018.

[53] S. Li, A. Neupane, S. Paul, C. Song, S. V. Krishnamurthy, A. K. R. Chowdhury, and A. Swami, "Adversarial perturbations against real-time video classification systems," 2018, *arXiv:1807.00458*. [Online]. Available: https://arxiv.org/abs/1807.00458

[54] Y. Zhou and L. Shao, "Aware attentive multi-view inference for vehicle re-identification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 6489–6498.

[55] A. Ferdowsi, U. Challita, and W. Saad, "Deep learning for reliable mobile edge analytics in intelligent transportation systems," 2017, *arXiv:1712.04135*. [Online]. Available: https://arxiv.org/abs/1712.04135

[56] L. Zhao, J. Wang, J. Liu, and N. Kato, "Routing for crowd management in smart cities: A deep reinforcement learning perspective," *IEEE Commun. Mag.*, vol. 57, no. 4, pp. 88–93, Apr. 2019.

[57] B. Zhang, C. H. Liu, J. Tang, Z. Xu, J. Ma, and W. Wang, "Learning–based energy–efficient data collection by unmanned vehicles in smart cities," *IEEE Trans. Ind. Informat.*, vol. 14, no. 4, pp. 1666–1676, Apr. 2018.

[58] M. Elsayed and M. Erol-Kantarci, "Deep Q-learning for low-latency tactile applications: Microgrid communications," in *Proc. IEEE Int. Conf. Commun., Control, Comput. Technol. Smart Grids (SmartGridComm)*, Oct. 2018, pp. 1–6.

[59] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerg. Topics Comput.*, to be published.
[60] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2017, pp. 197–210.
[61] S. Salman, C. Streiffer, H. Chen, T. Benson, and A. Kadav, "DeepConf: Automating data center network topologies management with machine learning," in *Proc. Workshop Netw. Meets AI & ML*, 2018, pp. 8–14.
[62] C. Zhong, M. C. Gursoy, and S. Velipasalar, "A deep reinforcement learning-based framework for content caching," in *Proc. 52nd Annu. Conf. Inf. Sci. Syst. (CISS)*, 2018, pp. 1–6.
[63] K. Baek and I.-Y. Ko, "Spatio-cohesive service selection using machine learning in dynamic IoT environments," in *Proc. Int. Conf. Web Eng.* Cáceres, Spain: Springer, 2018, pp. 366–374.
[64] T. Tan, F. Bao, Y. Deng, A. Jin, Q. Dai, and J. Wang, "Cooperative deep reinforcement learning for large-scale traffic grid signal control," *IEEE Trans. Cybern.*, to be published.
[65] P. Werbos, "Backpropagation through time: What it does and how to do it," *Proc. IEEE*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.
[66] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, Jul. 2006.
[67] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.
[68] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 1097–1105.
[69] C. Doersch, "Tutorial on variational autoencoders," 2016, *arXiv:1606.05908*. [Online]. Available: https://arxiv.org/abs/1606.05908
[70] D. P. Kingma, S. Mohamed, D. J. Rezende, and M. Welling, "Semi-supervised learning with deep generative models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 3581–3589.
[71] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Proc. Adv. Neural Inf. Process. Syst.*, 2014, pp. 2672–2680.
[72] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy, "Deep exploration via bootstrapped DQN," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4026–4034.
[73] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.
[74] G. White, A. Palade, C. Cabrera, and S. Clarke, "IoTPredict: Collaborative QoS prediction in IoT," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. (PerCom)*, Athens, Greece, Mar. 2018.
[75] C. Cabrera, A. Palade, G. White, and S. Clarke, "Services in IoT: A service planning model based on consumer feedback," in *Proc. Int. Conf. Service-Oriented Comput.* Hangzhou, China: Springer, 2018, pp. 304–313.
[76] A. Palade, C. Cabrera, G. White, and S. Clarke, "Stigmergic service composition and adaptation in mobile environments," in *Proc. Int. Conf. Service-Oriented Comput.* Hangzhou, China: Springer, 2018, pp. 618–633.
[77] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of Research on Emerging Trends and Applications of Machine Learning*. Hershey, PA, USA: IGI Global, 2010, pp. 242–264.
[78] P. Dollár, C. Wojek, B. Schiele, and P. Perona, "Pedestrian detection: A benchmark," in *Proc. CVPR*, Jun. 2009.
[79] W. Liu, S. Liao, W. Ren, W. Hu, and Y. Yu, "High-level semantic feature detection: A new perspective for pedestrian detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019.
[80] X. Wang, T. Xiao, Y. Jiang, S. Shao, J. Sun, and C. Shen, "Repulsion loss: Detecting pedestrians in a crowd," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 7774–7783.
[81] W. Liu, S. Liao, W. Hu, X. Liang, and X. Chen, "Learning efficient single-stage pedestrian detectors by asymptotic localization fitting," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 618–634.
[82] X. Wang, T. Xiao, Y. Jiang, S. Shao, J. Sun, and C. Shen, "Repulsion loss: Detecting pedestrians in a crowd," Nov. 2017, *arXiv:1711.07752*. [Online]. Available: https://arxiv.org/abs/1711.07752
[83] J. Han, J. Pei, and M. Kamber, *Data Mining: Concepts and Techniques*. Amsterdam, The Netherlands: Elsevier, 2011.
[84] W. Dai, Q. Yang, G.-R. Xue, and Y. Yu, "Self-taught clustering," in *Proc. 25th Int. Conf. Mach. Learn.*, 2008, pp. 200–207.
[85] Z. Wang, Y. Song, and C. Zhang, "Transferred dimensionality reduction," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Antwerp, Belgium: Springer, 2008, pp. 550–565.
[86] S. Nejatian, V. Rezaie, H. Parvin, M. Pirbonyeh, K. Bagherifard, and S. K. S. Yusof, "An innovative linear unsupervised space adjustment by keeping low-level spatial data structure," *Knowl. Inf. Syst.*, vol. 59, no. 2, pp. 437–464, May 2019, doi: 10.1007/s10115-018-1216-8.
[87] A. Pirbonyeh, V. Rezaie, H. Parvin, S. Nejatian, and M. Mehrabi, "A linear unsupervised transfer learning by preservation of cluster-and-neighborhood data organization," *Pattern Anal. Appl.*, vol. 22, no. 3, pp. 1149–1160, Aug. 2019, doi: 10.1007/s10044-018-0753-9.
[88] M. D. Bloice, C. Stocker, and A. Holzinger, "Augmentor: An image augmentation library for machine learning," 2017, *arXiv:1708.04680*. [Online]. Available: https://arxiv.org/abs/1708.04680
[89] R. French, "Catastrophic forgetting in connectionist networks," *Trends Cognit. Sci.*, vol. 3, no. 4, pp. 128–135, Apr. 1999.
[90] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," Aug. 2017, *arXiv:1708.02072*. [Online]. Available: https://arxiv.org/abs/1708.02072
[91] S.-W. Lee, C.-Y. Lee, D.-H. Kwak, J. Kim, J. Kim, and B.-T. Zhang, "Dual-memory deep learning architectures for lifelong learning of everyday human behaviors," in *Proc. 25th Int. Joint Conf. Artif. Intell. (IJCAI)*. 2016, pp. 1669–1675. [Online]. Available: http://dl.acm.org/citation.cfm?id=3060832.3060854
[92] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," 2016, *arXiv:1606.04671*. [Online]. Available: https://arxiv.org/abs/1606.04671
[93] T. Tieleman and G. Hinton, "Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude," in *Proc. Neural Netw. Mach. Learn. (COURSERA)*, 2012.
[94] L. N. Smith, "Cyclical learning rates for training neural networks," in *Proc. IEEE Winter Conf. Appl. Comput. Vis. (WACV)*, Mar. 2017, pp. 464–472.
[95] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-ResNet and the impact of residual connections on learning," Feb. 2016, *arXiv:1602.07261*. [Online]. Available: https://arxiv.org/abs/1602.07261
[96] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," Oct. 2016, *arXiv:1610.02357*. [Online]. Available: https://arxiv.org/abs/1610.02357

GARY WHITE received the B.A.I. and M.A.I. degrees in computer engineering from the Trinity College Dublin, Dublin, Ireland, where he is currently pursuing the Ph.D. degree in computer science. His research interests include QoS, machine learning, and edge computing. His research is funded by Science Foundation Ireland (SFI).

SIOBHÁN CLARKE received the B.Sc. and Ph.D. degrees in computer science from Dublin City University. She is currently a Professor and a Fellow of the Trinity College Dublin, where she leads the Distributed Systems Group and the Director of Future Cities, the Trinity Centre for Smart and Sustainable Cities. Her research interests are in software engineering models for the dynamic provision of smart and dynamic software services to urban stakeholders in large-scale and mobile environments.