Effective Deep-Learning-Based Depth Data Analysis on Low-Power Hardware for Supporting Elderly Care

Christopher Pramerdorfer
Cogvis & TU Wien
Vienna, Austria
pramerdorfer@cogvis.at

Rainer Planinc
Cogvis
Vienna, Austria
planinc@cogvis.at

Martin Kampel
TU Wien
Vienna, Austria
kampel@cvl.tuwien.ac.at

Abstract

We present a detailed technical insight into a commercial vision-based sensor for monitoring residents in elderly care facilities and alerting caretakers in case of dangerous situations such as falls or residents not returning to their beds during nighttime. We focus on aspects that enable deep-learning-based object classification in realtime on low-end ARM-based hardware, which is prerequisite for a solution that is performant yet affordable, low-power, and unobtrusive. To this end, we introduce an efficient vision pipeline that maps the input depth data to concise virtual top-views. These views are then processed by a set of convolutional neural networks, with a scheduler selecting the most appropriate one based on the current operating conditions and available hardware resources. In order to overcome the challenge of acquiring large amounts of training data in this privacy-critical environment, we pretrain these networks on a large set of synthetic depth data. These concepts are general and applicable to similar vision tasks.

1. Introduction

Due to the ongoing demographic change in developed countries, there is a rising demand for solutions that can support caretakers in elderly care facilities. Solutions that alert caretakers in case of relevant events such as frail people getting up from their beds ensure that residents receive help as quickly as possible while also reducing the need for continuous checks by the caretakers [10].

In this paper, we present a detailed technical overview of a commercial product we have developed for this purpose. In contrast to more established solutions such as panic buttons or pressure mats [10], our system is based on computer vision and requires no user intervention or maintenance once installed on a wall or ceiling in the room to be monitored. It combines the functionality of these solutions, namely fall detection and getup detection, with additional features such as alerting caretakers if residents do not return to their bed at night. The system analyzes data from a depth sensor for improved privacy over color cameras and reliable monitoring regardless of the lighting conditions, including at night. Configuration and monitoring is done using a web-platform, and alerts can be sent by various means, such as preexisting alerting infrastructure or text messages.

Given the purpose of our system, reliable event detection is crucial as e.g. missing a fall can have serious health implications. This requires state-of-the-art computer vision methods, which calls for deep-learning-based solutions [4]. However, such solutions are computationally expensive and require large datasets for training. Both aspects can prevent the use of deep learning in commercial vision systems due to resource constraints, and because acquiring large datasets for commercial purposes is labor-intensive and expensive, particularly in privacy-critical environments.

This paper presents design considerations that have allowed us to overcome both these problems and have enabled us to deploy a deep-learning-based computer vision pipeline for realtime analysis on a low-end ARM-based single-board computer, namely the Raspberry Pi 3. This is a major and stable platform that is readily available, inexpensive, small, and has a low power consumption, enabling small, unobtrusive, and affordable solutions that integrate this platform. However, the platform is slow even by ARM CPU standards, imposing tight restrictions on the algorithms used. The computer and depth sensor are integrated in a small custom casing that is shown in Figure 1.

We demonstrate empirically that practical deep learning for realtime applications is possible on this platform by combining concise top-view representations of objects with efficient neural network architectures and resource-based network scheduling. Furthermore, we present a two-stage approach to network training that overcomes the lack of large amounts of training data by pretraining on synthetic depth samples. These design aspects are general and can be applied to other vision problems and hardware platforms.

This paper is structured as follows. Section 2 reviews the

Figure 1. The custom casing (width: 25 cm).

pertinent literature on efficient neural networks, synthetic depth data, and related solutions for elderly care facilities. An overview of the main computational steps is presented in Section 3. Section 4 presents the object classification step, which is the focus of this paper, in more detail. Section 5 covers data acquisition, Section 6 presents the experiments and results, and Section 7 concludes the paper.

2. Related work

Efficient neural network architectures. Efficiency in deep learning is often ignored in favor of further performance improvements. GoogLeNet [19] was one of the first architectures that focus on both aspects. It has since been improved several times and combined with the popular ResNet architecture [18, 4].

Architectures that are efficient enough to run on low-end hardware have been presented only recently, for instance MobileNet [6] and ShuffleNet [21]. These architectures utilize efficient forms of convolutions, exploiting that these operations can be factorized. MobileNet utilizes pairs of depthwise and pointwise convolutions, whereas ShuffleNet is built upon units that combine a depthwise convolution layer with two pointwise convolution layers, and a layer that shuffles layers in groups. These units include a shortcut path for improved information flow, similarly to [4]. MobileNet v2 [15] adopts similar units albeit without channel shuffling. ShuffleNet v2 improves the original units and is more effective than MobileNet v2 [9]. The network architectures we utilize are based on this architecture.

Another approach for improving efficiency is quantization. This involves performing calculations in low-precision floating point or integer arithmetic if possible, which can lead to significant speed improvements depending on the capabilities of the target hardware [7]. Pruning unimportant parts of trained models is another option [5].

Synthetic depth data. A seminal work on synthetic depth data is [16], which demonstrates the potential of machine learning on large synthetic depth datasets. We follow a similar approach but utilize deep learning for classification instead of random forests, and finetune our models on a smaller realistic dataset for improved performance. [1] and [13] also utilize synthetic depth data, for fall detection and hand pose estimation, respectively. There are only few public datasets, with the most comprehensive one we are aware of being SURREAL [20]. SUNCG is another popular example, however the dataset appears to be no longer available due to copyright issues.

Active and assisted living. Research on technologies for supporting elderly people seemingly focus on fall detection. Depth data are popular for this purpose. An example is [14], which presents an efficient motion detection algorithm that we adopt. The authors employ virtual top-views [3], from which they compute basic features for classification using a random forest. We also employ such top-views but process them like images using powerful convolutional neural networks. [1] also detects falls via motion detection and random forest classification but the method is not view-based. One of the first such works was [2], which employs HU features and SVM-based classification. [11] reviews older color-based methods.

Commercial solutions for elderly care. There are various technical solutions available on the market that aim to support caretakers in elderly care facilities. Panic buttons are a popular example, however they require a worn device and user intervention. Most passive products are relatively simple mechanical solutions such as floor or bed mats that react to pressure. Vision-based products that provide additional features and require no user intervention have entered the market only recently. There are currently less than five companies that provide such solutions. Our product has been on the European market since 2018.

3. System overview

All computational steps are optimized for depth data. In addition to the aforementioned advantages, depth data enable efficient 3D scene reconstruction, which we utilize for data conversion to virtual top-views that can be processed effectively by convolutional neural networks.

3.1. Computational steps

Figure 2 lists the main computational steps in our system. The input depth map from the sensor is first processed using an efficient motion detection algorithm optimized for depth data. We adapted the algorithm presented in [14] for this purpose, replacing the original sensor noise model to reflect the properties of our depth sensor.

This follows object segmentation. For efficiency, we utilize a binary top-view representation of the scene that encodes the location and geometry of all moving objects is a concise way. This representation is obtained by converting

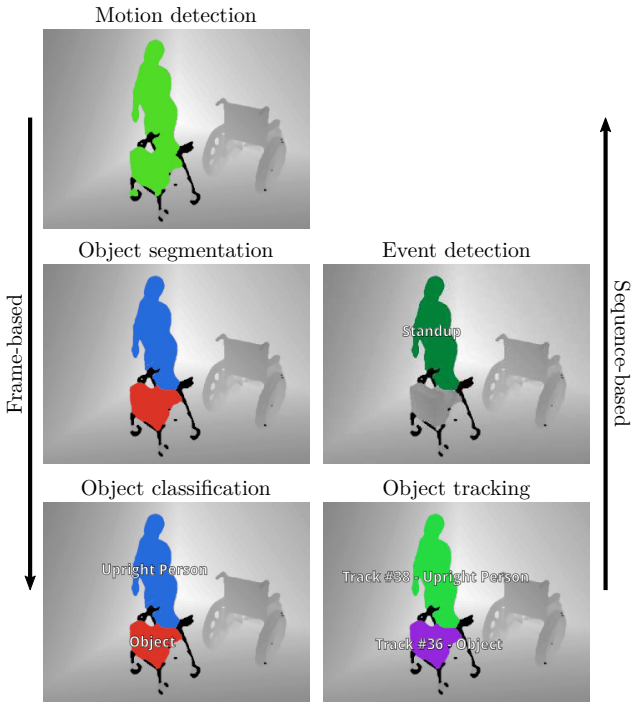

Figure 2. The main computational steps in our system.

all motion pixels to top-view coordinates as detailed in Section 4.1. On this basis, occupied coordinates are assigned to tracks based on the position, velocity, and geometry of all currently tracked objects. Occupied coordinates that are left unassigned are subject to connected component analysis, and the resulting components spawn new tracks.

The next step, and the focus of this paper, is object classification. This entails mapping each segmented object to a virtual top-view representation that can be classified effectively using convolutional neural networks. Section 4 describes this step in detail.

The remaining steps are object tracking, which involves updating the location, velocity, and geometrical properties of all tracked objects, as well as event detection. Events are detected via temporal analysis of all tracked persons, considering both location and state classes predicted by the classifier (e.g. *upright person* or *person on floor*). Both stages assume a relatively constant framerate of around 7.5 frames per second. We found this target framerate to be a good compromise in terms of measurement frequency vs. measurement accuracy, given the available computational resources. More specifically, this framerate allows for reliable tracking and robustness to temporary classification errors due to occlusions or inaccurate motion detection.

3.2. Resource consumption

We set an average CPU load limit of 60% overall during normal operation when determining resource budgets. This ensures long-term system stability, limits power consumption and heat production, and leaves sufficient headroom for temporary background tasks and short load spikes. Around 80% of the corresponding CPU cycles are available to the computational steps described in this paper.

The computational step we found most important to the overall performance is object classification. We thus assign the majority of available resources to this stage, as convolutional neural networks scale well in this regard [4, 15, 9]. More specifically, we assign around 50% of these resources to the neural networks. Motion detection and segmentation may consume around 20% resources each on average in practice, with the rest being required by the tracking and event detection stages. We note that these are soft limits that may be exceeded temporarily.

4. Object classification

The object classification stage takes the current depth map and a label map that encodes which pixels belong to which segmented object, and outputs a vector of class scores for each object. This is illustrated in Figure 3, which highlights the main steps involved in this process.

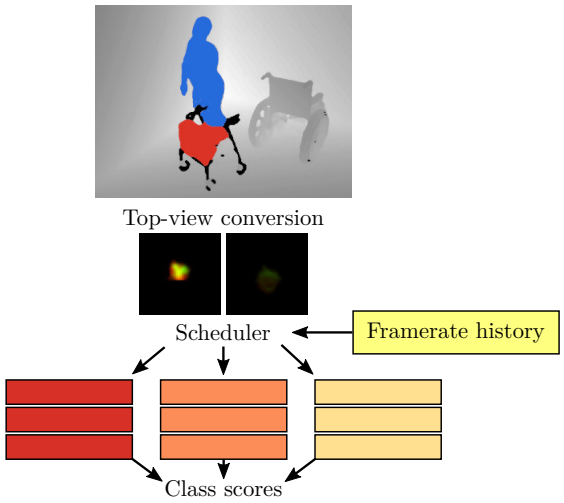

Figure 3. The proposed object classification pipeline.

The combination of top-view analysis, efficient neural network architectures, and a scheduler that selects the optimal network given the currently available resources allows us to perform effective deep-learning-based object classification in realtime on low-end hardware.

4.1. Top-view conversion

For top-view conversion, all pixels that belong to any object are mapped to camera coordinates, which is trivial as the depth map pixels encode distances from the sensor. These coordinates are then mapped to a world coordinate system whose (x, y) plane coincides with the floor plane.

The extrinsic camera parameters required in this step are estimated automatically during system initialization. They are obtained by detecting the floor plane via RANSAC-based plane fitting, similarly to [14]. This conversion to world coordinates makes analysis invariant to the sensor position and orientation, facilitating system installation.[1]

For each object, the corresponding point cloud is then converted to two top-view representations. For this purpose, the x and y coordinates of each point are first divided by a factor λ that determines the resolution of the resulting views, and then rounded to integers (i_x, i_y). We set $\lambda = 5$ cm, which preserves a sufficient amount of spatial information while resulting in views that are small enough to process quickly. This process generally maps multiple points to the same top-view coordinates (i_x, i_y).

We generate two top-views, a height-map and an occupancy-map [3], by merging such points in two different ways. For the former, we store the maximum z coordinates, which encode distances from the floor. Height-maps thus encode object geometry in a concise way. For occupancy-maps, we count the number of points that map to each (i_x, i_y). The result is weighted based on the distance of (i_x, i_y) from the camera, as the number of image pixels per unit area decreases quadratically with distance. Occupancy-maps thus indicate the object density and represent information that is complementary to height-maps. We next refine both top-views similarly to [3], smoothing and then thresholding them based on occupancy to remove noise due to motion detection errors.

For classification, we treat both views as image channels, stacking them to obtain a single two-channel image. We classify the views of all objects in a single batch as this is more efficient than classifying each view individually, as shown in Section 6.2. In order to obtain views of a consistent size, which is required for batch processing, we center each view based on the centroid location according to the occupancy data and then extract a center-crop of size 40×40, padding with zeros if necessary. At $\lambda = 5$ cm this ensures that objects up to an extent of 2 by 2 meters are fully captured, which includes most people regardless of pose and most movable indoor objects.

4.2. Classifier scheduling

Classification utilizes multiple neural network classifiers that vary in terms of speed and accuracy, with slower classifiers being more accurate. The job of the *classifier scheduler* is to select the slowest and thus most accurate classifier possible considering the available time budget and the current batch size b. The time budget is derived from the predefined resource consumption targets. For this purpose, the scheduler maintains a so-called framerate history, a lookup table that maps from b the average time it takes each classifier to classify batches of size b. This lookup table is loaded from a template at startup and adapted by the scheduler over time based on continuous classifier speed measurements.

We note that this approach is generally applicable whenever the available computational resources vary over time, and is not tied to specific network architectures.

4.3. Network architectures

Our network architectures are based on ShuffleNet v2, a state-of-the-art architecture that achieves high performance while being efficient and optimized for ARM platforms [9]. We adapt this architecture as follows. First, we remove the max-pooling layer to preserve spatial information initially as our inputs have size 40×40 whereas the original architecture was designed for size 224×224. To achieve the efficiency required for realtime analysis, we reduce the number of feature maps produced by the convolutional layers significantly. This results in *narrow* versions of the original ShuffleNet v2 architecture, which we call *NSNets*. We use three versions of this architecture that represent optimal trade-offs between performance and target speed at a given batch size, as detailed in Section 6.2. These are named NSNet-S (slow), NSNet-M (medium), and NSNet-F (fast).

Table 1. Architectural differences between our NSNet variants and ShuffleNet v2. The values represent the number of feature maps apart from the max-pooling layer, where they denote the stride.

Layer	NSNet-S	NSNet-M	NSNet-F	ShuffleNet
Conv1	24	16	16	24
Pool	-	-	-	2
Stage2	48	32	16	48
Stage3	72	64	24	96
Stage4	144	96	32	192
Conv5	512	512	512	1024
Params	172k	143k	34k	358k

Table 1 summarizes the differences between these NSNet variants and compares them to the original ShuffleNet v2 architecture with complexity 0.5, the smallest official version [9]. At the given input size, NSNet-S has around half the number of parameters of ShuffleNet v2 despite the less aggressive pooling.

5. Training data acquisition

While large amounts of training data are required for optimal model performance in deep learning, there are no public datasets available that reflect the task at hand (let alone ones with commercial licenses). Acquiring large amounts of data that reflect this task is particularly challenging, as this would require recording people in their homes as well as their caretakers, which is problematic with respect to privacy and data protection rules.

[1] This conversion is also required for object segmentation. For efficiency, conversion is carried out once and the results are shared.

To overcome this problem, we have instead adopted a two-tier strategy for data acquisition and training that involves pretraining on synthetic depth data followed by finetuning on a smaller set of real samples collected by our systems in case of detected events.

5.1. Synthetic data for pretraining

Generating a synthetic depth data sample involves five main steps, which are illustrated in Figure 4.

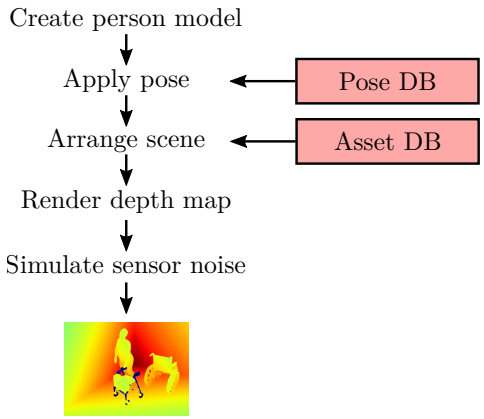

Figure 4. Main steps of our synthetic data rendering pipeline.

First, a commercial 3D modeling software is used to automatically generate a realistic 3D model of a person. This includes randomizing properties such as gender, height, weight, age, body tone, and facial expression, such that persons vary widely in appearance.

Second, a random pose is sampled from our pose database that comprises thousands of realistic body poses, and applied to the person model. These poses were obtained via motion capturing of actors that were simulating various relevant activities and actions such as walking, sitting, lying on a bed, and falling on the floor.

The person model is positioned randomly inside an empty room, which is then filled with objects from our asset database that consists of hundreds of 3D models. The object selection and arrangement is random but considers the body pose and location to ensure that the resulting scene is realistic. For instance, if the label associated with the sampled person pose is *sitting*, a suitable sitting accommodation is selected from the database and positioned in the scene accordingly. This is illustrated in Figure 5.

After the scene has been arranged, a virtual camera is positioned at a random position and orientation, and a depth map of the scene is rendered. In addition, a label map is generated that can be used to extract individual objects.

The rendered depth map lacks sensor noise and other errors, which is unrealistic. In the final step, we thus simulate such sensor errors using the method presented in [12], which we adapted to better reflect the error characteristics of the depth sensor model we use.

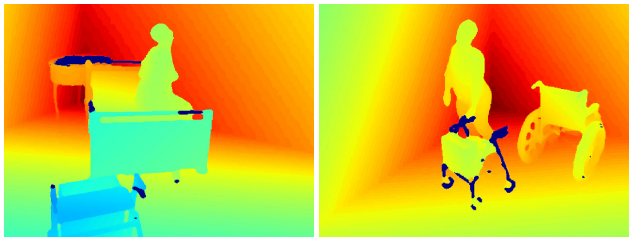

Figure 5. Example depth maps from our synthetic dataset, color-coded for visualization purposes.

This pipeline has allowed us to generate tens of thousands of depth maps depicting synthetic yet realistic scenes. Two of these depth maps are illustrated in Figure 5.

5.2. Event data for finetuning

After pretraining, our models are finetuned on a realistic but smaller dataset of top-views. These top-views are sent to our servers in an anonymized and encrypted form whenever an event is detected by our systems (e.g. if a person falls down on the floor). Figure 6 shows such samples, highlighting their level of abstraction. This makes identifying persons impossible, ensuring the user's privacy.

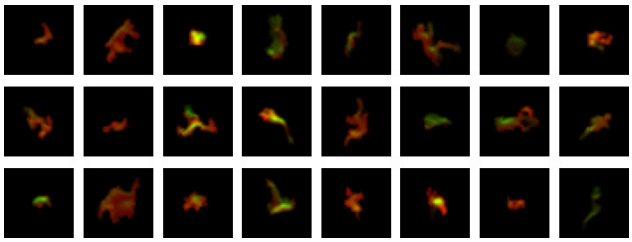

Figure 6. Visualizations of example depth maps from our realistic dataset (red: height data, green: occupancy data).

Employees assign class labels to these samples based on a visual inspection and anonymized information that is also available to the customers on the web-platform. Customers who do not want to participate can opt-out of data acquisition at any time. We emphasize that there is no continuous data recording or transmission; samples are generated only in case of detected events. The number of samples per sensor varies from less than one per week to few samples per day, depending on the operating conditions.

We utilize samples with labels *upright person*, *sitting person*, *resting person*, *squatting person*, *person on floor*, *person in other pose*, and *object*. The resulting dataset consists of about 21000 samples of people in different poses as well as various kinds of objects. The dataset is challenging as the subjects are often captured only partially due to occlusions or inaccurate motion detection. The classes are imbalanced, for instance almost 40% of all samples are labeled as *object* while around 5% are *squatting persons*.

6. Experiments

We compare our NSNet network architectures to popular alternatives, namely ResNet-18 [4] and ShuffleNet v2 [9] at complexity $\alpha = 1$ (standard) and $\alpha = 0.5$ (the fastest official variant). The former architecture is included as a reference, as ResNet is arguably the standard architecture if resource limitations are not an issue. ShuffleNet v2 is the architecture NSNets derive from. We adapt the ResNet architecture to 40×40 sized inputs by replacing the initial 7×7 convolution at stride 2 with a 3×3 convolution at stride 1 and removing the subsequent max-pooling layer. This preserves spatial resolution initially. For ShuffleNet v2 we remove the first max-pooling layer for the same reason.

6.1. Network training

We first pretrain each network from scratch on our synthetic dataset, and then finetune it on the smaller real dataset. For parameter optimization we minimize the cross-entropy loss using the improved version of Adam [8] and the one-cycle policy for adaptive learning rates [17]. In both iterations, we split the datasets into training and validation sets for hyperparameter optimization, and train until the validation accuracy saturates. To account for imbalanced class distributions, we compute stratified splits and employ oversampling of the training set. For training data augmentation we utilize a combination of random affine transformations, random crops, and horizontal flipping.

6.2. Inference speed

We first compare the inference speed of all architectures on the target hardware, a Raspberry Pi 3B, using an optimized build of TensorFlow 2.1. To facilitate comparisons, we benchmark solely network inference speeds on an otherwise idle system, not the whole computational pipeline. We do so at different batch sizes, which usually vary between 1 and 5 in practice depending on how many objects are tracked. At each batch size, we run the prediction 100 times and report median prediction framerates on this basis.

Figure 7 summarizes the results. Apart from ResNet, all networks tested run at over 15 frames per second (fps) when processing single samples. The framerates decrease slower than linearly at larger batch sizes, mainly due to the load being split across the four CPU cores available.

Based on these results and considering our target framerate of 7.5 fps, it may appear that even the comparatively complex ShuffleNet v2 architecture with $\alpha = 1.0$ is fast enough. However, this is not the case as (i) the CPU cores are not at idle in practice and (ii) due to the resource consumption limits covered in Section 3.2. For these reasons, networks must score around 30 fps according to Figure 7 to fulfill our requirements on efficiency.

Our NSNet variants achieve this for batch sizes up to 2, 4, and 6, respectively. Scheduling acts accordingly and by default assigns batches with size 1 or 2 to NSNet-S, those with size 3 or 4 to NSNet-M, and larger ones to NSNet-F.

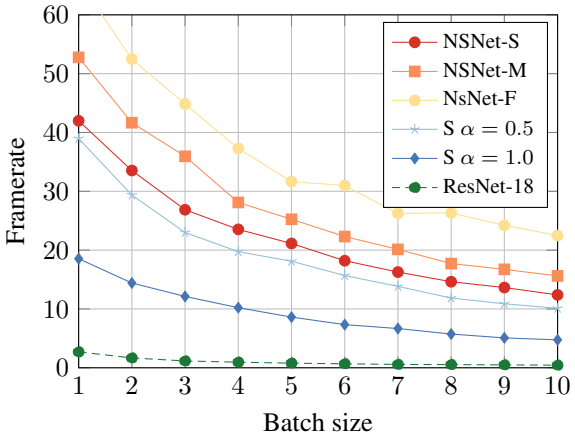

Figure 7. Speed of different network architectures on the target hardware as a function of batch size, in frames per second. S is ShuffleNet v2 without max-pooling at complexity α.

We note that the level of optimization of deep learning libraries and versions varies, particularly on ARM. The capabilities of ARM SOCs also varies widely. For instance, [9] report a single-sample speed of 57 fps for ShuffleNet v2 at $\alpha = 0.5$ with larger images (but with max-pooling) on a more powerful Qualcomm Snapdragon 810 platform.

6.3. Classification accuracy

Figure 8 compares the classification performances and relates them to the single-sample inference speeds in order to highlight speed vs. accuracy tradeoffs.

NSNet-S performs comparably to ShuffleNet v2 at $\alpha = 0.5$ while being slightly more efficient. NSNet-M is around 2% less accurate than NSNet-S while being 25% faster. NSNet-F scales virtually identically with respect to NSNet-M, with the single-sample framerate being 25% higher and the accuracy being 2% lower. These accuracy gains may seem small but result in significant improvements in event detection performance, as these gains are on challenging samples that may cause missed events or false alarms.

The speed vs. accuracy curve of NSNets is comparable to ShuffleNet v2 and MobileNet v2 on ImageNet in similar accuracy regions [9, 15]; about twice the computational budget is required for around 5% increased accuracy. ShuffleNet v2 at $\alpha = 1.0$ and ResNet-18 outperform our NSNets as expected but are too inefficient for our application.

7. Conclusion

We have presented a technical overview of a commercial vision-based system for monitoring residents of elderly care facilities, with a focus on effective object classification on low-power hardware. We have discussed the key design

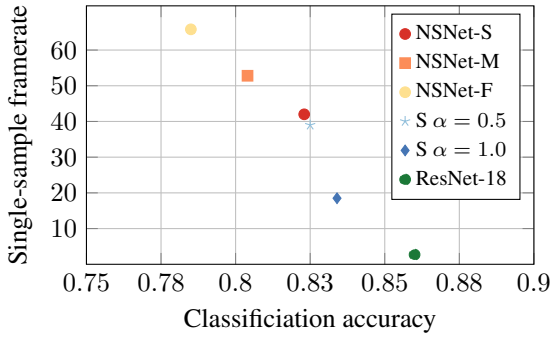

Figure 8. Classification accuracies and single-sample framerates of all network architectures compared.

choices that enable deep-learning-based object classification in depth data in realtime, namely view-based analysis, efficient network architectures, as well as resource-based scheduling in order to optimize the tradeoff between classification accuracy and resource consumption. These design choices are general and can be adapted to overcome similar resource bottlenecks in other vision applications. Furthermore, we have presented our method of two-stage network training using a combination of synthetic and realistic data in order to overcome challenges in dataset acquisition.

References

[1] Ahmed Abobakr, Mohammed Hossny, and Saeid Nahavandi. A Skeleton-free Fall Detection System from Depth Images using Random Decision Forest. *IEEE Systems Journal*, 12(3):2994–3005, 2017. 2

[2] Rachit Dubey, Bingbing Ni, and Pierre Moulin. A Depth Camera Based Fall Recognition System for the Elderly. In *International Conference Image Analysis and Recognition*, pages 106–113, 2012. 2

[3] Michael Harville and Dalong Li. Fast, Integrated Person Tracking and Activity Recognition with Plan-view Templates from a Single Stereo Camera. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2004. 2, 4

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. In *European Conference on Computer Vision (ECCV)*, pages 630–645, 2016. 1, 2, 3, 6

[5] Yihui He, Xiangyu Zhang, and Jian Sun. Channel Pruning for Accelerating Very Deep Neural Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1389–1397, 2017. 2

[6] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv preprint arXiv:1704.04861*, 2017. 2

[7] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew G. Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. *CoRR*, abs/1712.05877, 2017. 2

[8] Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *CoRR*, abs/1711.05101, 2019. 6

[9] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *European Conference on Computer Vision (ECCV)*, pages 116–131, 2018. 2, 3, 4, 6

[10] Rana Mostaghel. Innovation and Technology for the Elderly: Systematic Literature Review. *Journal of Business Research*, 69(11):4896–4900, 2016. 1

[11] Muhammad Mubashir, Ling Shao, and Luke Seed. A Survey on Fall Detection: Principles and Approaches. *Neurocomputing*, 100:144–152, 2013. 2

[12] Chuong V Nguyen, Shahram Izadi, and David Lovell. Modeling Kinect Sensor Noise for Improved 3D Reconstruction and Tracking. In *International Conference on 3D Imaging, Modeling, Processing, Visualization & Transmission*, pages 524–530, 2012. 5

[13] Markus Oberweger, Gernot Riegler, Paul Wohlhart, and Vincent Lepetit. Efficiently Creating 3D Training Data for Fine Hand Pose Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4957–4965, 2016. 2

[14] Christopher Pramerdorfer, Rainer Planinc, Mark Van Loock, David Fankhauser, Martin Kampel, and Michael Brandstötter. Fall Detection Based on Depth-data in Practice. In *European Conference on Computer Vision (ECCV) Workshops*, pages 195–208, 2016. 2, 4

[15] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018. 2, 3, 6

[16] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time Human Pose Recognition in Parts from Single Depth Images. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1297–1304, 2011. 2

[17] Leslie N. Smith and Nicholay Topin. Super-Convergence: Very Fast Training of Residual Networks Using Large Learning Rates. *CoRR*, abs/1708.07120, 2017. 6

[18] Christian Szegedy, Sergey Ioffe, and Vincent Vanhoucke. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. *CoRR*, abs/1602.07261, 2016. 2

[19] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *CoRR*, abs/1409.4842, 2014. 2

[20] Gül Varol, Javier Romero, Xavier Martin, Naureen Mahmood, Michael J. Black, Ivan Laptev, and Cordelia Schmid. Learning from Synthetic Humans. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2

[21] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018. 2