Distilled Split Deep Neural Networks for Edge-Assisted Real-Time Systems

Yoshitomo Matsubara
University of California, Irvine
yoshitom@uci.edu

Sabur Baidya
University of California, Irvine
sbaidya@uci.edu

Davide Callegaro
University of California, Irvine
dcallega@uci.edu

Marco Levorato
University of California, Irvine
levorato@uci.edu

Sameer Singh
University of California, Irvine
sameer@uci.edu

ABSTRACT

Offloading the execution of complex Deep Neural Networks (DNNs) models to compute-capable devices at the network edge, that is, edge servers, can significantly reduce capture-to-output delay. However, the communication link between the mobile devices and edge servers can become the bottleneck when channel conditions are poor. We propose a framework to split DNNs for image processing and minimize capture-to-output delay in a wide range of network conditions and computing parameters. The core idea is to split the DNN models into head and tail models, where the two sections are deployed at the mobile device and edge server, respectively. Different from prior literature presenting DNN splitting frameworks, we distill the architecture of the head DNN to reduce its computational complexity and introduce a bottleneck, thus minimizing processing load at the mobile device as well as the amount of wirelessly transferred data. Our results show 98% reduction in used bandwidth and 85% in computation load compared to straightforward splitting.

CCS CONCEPTS

• **Information systems → Mobile information processing systems**; **Multimedia streaming**.

KEYWORDS

Deep neural networks, network distillation, edge computing.

HotEdgeVideo'19, October 21, 2019, Los Cabos, Mexico
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6928-2/19/10…$15.00
https://doi.org/10.1145/3349614.3356022

1 INTRODUCTION

Deep Neural Networks (DNNs) achieve state of the art performance in a broad range of classification, prediction and control problems. However, the computational complexity of DNN models has been growing together with the complexity of the problems they solve. For instance, within the image classification domain, LeNet5, proposed in 1998 [13], consists of 7 layers only, whereas DenseNet, proposed in 2017 [8], has 713 low-level layers. Despite the advances in embedded systems of the recent years, the execution of DNN models in mobile platforms is becoming increasingly problematic, especially for mission critical or time sensitive applications, where the limited processing power and energy supply may degrade the response time of the system and its lifetime.

Offloading data processing tasks to edge servers [4, 16], that is, compute-capable devices located at the network edge, has been proven to be an effective strategy to relieve the computation burden at the mobile devices and reduce capture-to-classification output delay in some applications. However, poor channel conditions, for instance due to interference, contention with other data streams, or degraded signal propagation, may significantly increase the amount of time needed to deliver information-rich data to the edge server.

Recently proposed frameworks [9, 10, 12] *split* DNN models into head and tail sections, deployed at the mobile device and edge server, respectively, to optimize processing load distribution. However, due to structural properties of DNNs for image processing, a straightforward splitting approach may lead to a large portion of the processing load to be pushed to the mobile device, while also resulting in a larger amount of data to be transferred on the network.

The core contribution of this paper is a more refined approach to split DNN models and distribute the computation load for real-time image analysis applications. Specifically, we *distill* the head portion of the DNN model, and introduce a bottleneck within the the distilled head model. This allows the reduction of the computational complexity at the sensor while also reducing the amount of wirelessly transferred

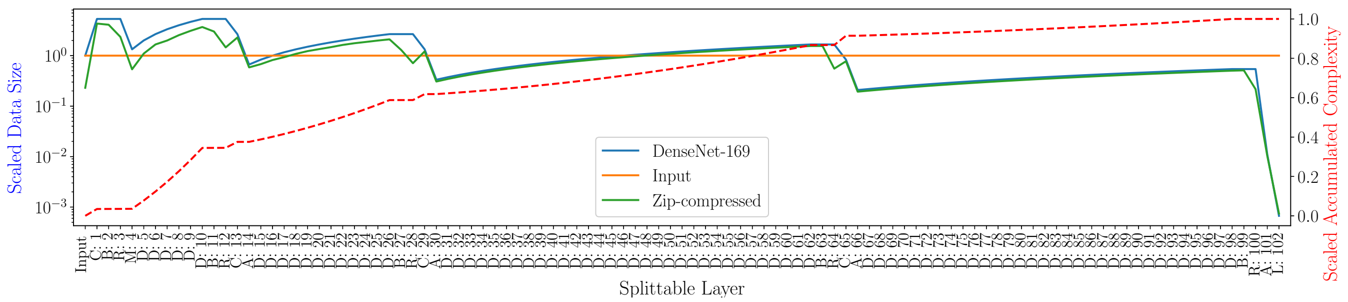


Figure 1: DenseNet-169 as example: Splittable layer-wise scaled output data size (blue and green lines for uncompressed and compressed) defined as the ratio between the size of the layer’s output and input and accumulated computational complexity (red line). C: convolution, B: batch normalization, R: ReLU, M: max pooling, D: (high-level) dense, A: average pooling, and L: linear layers.

data. From a high level perspective, our approach introduces a special case of autoencoder transforming the input signal into the input of a later layer through a bottleneck. We apply this approach to state of the art models and datasets for image classification, and show that it is possible to achieve “compression” up to 1% of the input signal with a complexity 95% smaller than the original head model.

The rest of the paper is organized as follows. Section 2 introduces the problem and discusses structural properties of DNN models for image classification. In Section 3, we present the core contribution of this paper: a DNN splitting strategy where we apply distillation to the head portion and introduce a bottleneck to maximize offloading performance. Section 4 reports the results in terms of overall inference time over suitable embedded computers. Section 5 concludes the paper.

2 PRELIMINARY DISCUSSION

We consider state of the art DNN models for image classification. Specifically, we study: DenseNet-169, -201 [8], ResNet-152 [7] and Inception-v3 [17]. We train and test the models on the CalTech 101 [6] and ImageNet [5] datasets.

We remark that in split DNN strategies, the overall inference time is the sum of three components: the time needed to execute the *head* and *tail* portions of the model – τ_{head} and τ_{tail} respectively – and the time to wirelessly transfer the output of the head model’s last layer to the edge server τ_{data}. We assume that the edge server has a high computation capacity, and seek strategies reducing computation load at the mobile device – that is, the complexity of the head network portion – and the amount of data to be transferred. Pure edge computing can be interpreted as an extreme point of splitting, where the head portion is composed of 0 layers, and τ_{data} is the time to transfer the input image.

Figure 1 shows the scaled size of data to be transferred, expressed as percentage compared to the input size, and the scaled accumulated complexity (number of operations performed up to that layer) for the layers of DenseNet-169 where the model can be split. The trend illustrates the issue: the output of the layers becomes perceivably smaller than the input only in later layers. Thus, reducing τ_{data} would require the – weaker – mobile device to execute most of the model, possibly resulting in an overall larger $\tau_{\text{head}}+\tau_{\text{data}}$ compared to transferring the input image and executing the head portion at the edge server (pure offloading). Importantly, most early layers have an output size larger than the input, and reaching the first point where a reasonable compression is achieved – approximately 33% of the input at layer 30 – corresponds to execute 60% of the total operations composing the whole DNN. The second candidate layer to split the DNN is layer 66, where the output is approximately 21% of the input after an accumulated complexity of 91% of the whole model. Then, the output size slowly increases until the very last layers. Reported in the figure, standard Zip compression allows to reduce the data size of the DNN layers – or of the input – without any loss of classification accuracy. However, almost no compression gain is achieved in “natural” splitting points, and compression does not provide a real advantage.

Intuitively, these trends do not allow an effective splitting strategy in asymmetric systems where the mobile device has a much smaller computational power compared to the edge server. Additionally, an increase in the time needed to transfer data penalizes splitting with respect to pure offloading. Thus, we contend that achieving an advantageous balance between computation at a weaker device and communication over a possibly impaired wireless channel necessitates modifications to the DNN architecture.

3 SPLIT MIMIC DNN MODELS

Our overall objective is to reduce the complexity of head models while minimizing the amount of data transferred from the mobile device to the edge server. To this aim, we

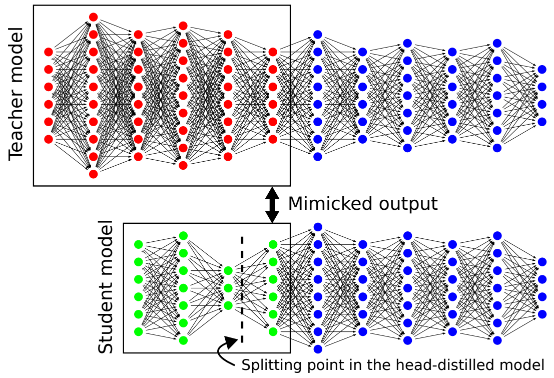


Figure 2: Illustration of head network distillation

use two recent tools: *network distillation* and the introduction of *bottlenecks*. Figure 2 illustrates the modifications in the overall architecture of the DNN. In the experiments shown in this section, the datasets are randomly split into training, validation and test datasets with a ratio 8:1:1, respectively.

Bottlenecks have been recently theoretically shown to promote the DNNs to learn optimal representations [1], thus achieving compression within the model. However, as reported in Table 1, making more aggressive the "natural" bottlenecks directly in the original DNN model resulted in accuracy degradation even for relatively mild compression rates. Moreover, some models, such as Inception-v3, do not present any candidate splitting point. Thus, more substantial modifications to the architecture are needed.

Network Distillation - We propose to "shrink" the head model using network distillation [2, 3, 14, 18], a recently proposed technique to train small "student" networks approximating the output of larger "teacher" models. Interestingly, Ba and Caruana [3] show that student models trained on "soft-labeled" dataset (output of their teacher models) significantly improve prediction performance compared to student models trained on the original ("hard-labeled") training dataset only, *i.e.*, without a teacher model. However, distilling entire DNN models for image analysis to fit the capabilities of mobile devices could degrade their performance. As an indication of this issue, effective small models such as MobileNetV2 [15], a lower complexity model designed to run on mobile devices, achieve a test accuracy of about 71% on CalTech 101 – a significantly worse performance compared to the models built in this paper. Additionally, MobileNet models have a significantly higher complexity – 46% increase – placed at the mobile device compared to our head models.

In the setting considered in this paper, the key advantages of using distillation are: (*a*) properly distilled models often give comparable performance while reducing the number of parameters used in the model and, thus, computation complexity. This will allow us to create efficient distilled head

Table 1: Results on Caltech 101 dataset for DenseNet-169 models redesigned to introduce bottlenecks

Metrics \ 1st Conv	*64 channels	8 channels	4 channels
Test accuracy [%]	84.5	83.9	80.5
Data size [%]	133	16.7	8.33

* number of channels in the original model

models mimicking the original head network; (*b*) student models often avoid overfitting during distillation as the soft-target from a teacher model has a regularization effect [3, 18]; and (*c*) the smaller number of nodes in student models results in a natural reduction of the data to be transferred to the edge server if the splitting point is positioned inside the student model. Moreover, as we will demonstrate later in this section, the more manageable structure of our student models will allow the creation of aggressive bottlenecks.

Figure 2 illustrates the student-teacher distillation approach in the considered split DNN configuration. At first, we split a pretrained DNN model into head (red) and tail (blue) networks. Taking DenseNet-169 as an example, Figure 1 allows the identification of the natural bottleneck points in the original DNN model at the 1st, 2nd and 3rd average pooling layers (layer number 14, 30 and 66). Using our proposed head network distillation, we split a DNN model at a bottleneck point, and build a different smaller student model with which we replace the original head network to reduce computational complexity at the mobile device.

We use Adam [11] to train the student models by minimizing the sum of square error between outputs of teacher and student models, defined as:

$$SSE(X) = \sum_{x \in X} ||t(x) - s(x)||^2, \tag{1}$$

where X is a set of b input RGB images. $t(x)$ and $s(x)$ are (often 3D-shaped) outputs of teacher and student models respectively, and $t(x)$ is fixed given x and treated as a "soft-label" to train the student model $s(x)$. Thus, our objective is to train the student model so that the model can mimic its teacher model i.e. $s(x) \approx t(x)$ given the input x. In the training process, we feed exactly the same input x into both the teacher and student models. The teacher model has been already trained with the dataset, and its model parameters are fixed. The teacher's output $t(x)$ is treated as a "soft-label" (target values), and we update the student model's parameters such that its output $s(x)$ is close to $t(x)$ by minimizing the loss function defined in Eq. (1).

Table 2 reports the accuracy, data size and complexity reduction using the proposed techniques on DenseNet-169 and -201 at different splitting points. The mobile device (MD) complexity reduction granted by the student model is computed

Table 2: Head network distillation results: mimic model without bottleneck – DenseNet-169 and -201

	DenseNet-169	Mimic		
Metrics	Original	1^{st} SP	2^{nd} SP	3^{rd} SP
Test accuracy [%]	84.5	84.0	84.3	83.8
Data size [%]	66.7	66.7	33.3	20.8
MD complexity	1.28×10^9	2.29×10^8	2.53×10^8	1.30×10^9
(Reduction [%])	(0.00)	(82.1)	(88.0)	(58.2)
	DenseNet-201	Mimic		
Metrics	Original	1^{st} SP	2^{nd} SP	3^{rd} SP
Test accuracy [%]	85.2	84.2	84.1	84.3
Data size [%]	66.7	66.7	33.3	29.2
MD complexity	1.28×10^9	2.29×10^8	2.53×10^8	1.51×10^9
(Reduction [%])	(0.00)	(82.1)	(88.0)	(62.4)

as $\left(1-\frac{C_S}{C_T}\right)\times 100$, where C_S and C_T indicate the computational complexity of the student and teacher models, respectively. Note these mimic models do not alter the amount of data transferred to the edge server, as they latch to the original tail network at the bottlenecks already present in the original model. In the tables, we list the output size of the 1^{st} bottleneck in the original models as reference. The head network distillation successfully reduces complexity of the head model while keeping accuracy comparable to the original one irrespective of the splitting point. A slight degradation is perceivable when the student model includes the layers up to the 3^{rd} bottleneck of DenseNet-169, possibly indicating the compression of an excessive portion of the network.

Bottleneck Injection - The student models developed earlier reduce complexity, but still produce an output size which is, at the minimum, around 30% of the input at the 3^{rd} splitting point. We now devise distilled student models where we introduce aggressive bottlenecks to further reduce the amount of data transferred to the edge server. To this aim, in all the considered DNNs, we artificially inject bottlenecks at the very early stages of the *student* models. We emphasize that, thus, the splitting point is inside the student model, rather than at its end, and the edge server will need to execute a portion of the student model.

Table 3 shows even when a DNN model (*e.g.*, ResNet-152 and Inception-v3 models) has no bottleneck point, our proposed approach enables the introduction of an aggressive bottleneck *within* the student model, so that by splitting the student model at that point we reduce both computational complexity on mobile device and transferred data size. Compared to the original models, we achieve a dramatic reduction in both complexity and output data size with at most about 1% accuracy drop. The mimic model has an output size of 1–2% of the input, obtained with a number of operations reduced by up to 95.5% compared to the original head model.

Table 3: Head network distillation results: mimic model with bottleneck

	Mimicked model			
Metrics	DN-169	DN-201	ResN-152	Inc-v3
Test accuracy [%]	83.3 (-1.2)	84.1 (-1.1)	83.2 (-1.1)	85.7 (-0.8)
Data size [%]	1.68	1.68	1.68	1.53
MD complexity	1.22×10^8	1.22×10^8	1.22×10^8	2.11×10^8
(Reduction [%])	(94.2)	(94.2)	(95.5)	(84.4)

Due to the extensive time needed to train the models, we only report preliminary results based on the ImageNet dataset, presenting a more complex classification task due to its size and the large number of classes. The distilled mimic model with bottleneck achieved a data size reduction of 11% of the input, and a complexity reduction of 94%. This result demonstrates that head model compression is possible even in difficult tasks.

Remarks on Object Detection - The proposed approach appears to be an extremely promising technique to balance computation and communication load between weak mobile devices and edge servers. However, its application to object detection – a core vision task – requires some non-trivial extensions. Different from image classification, DNN-based object detectors are required to predict bounding boxes and object class for each box, and the models are often composed of multiple core modules, namely backbone, classifier, and bounding box regressor modules. Taking RetinaNet as an example, the model uses the ResNet architecture as a backbone to extract features for bounding boxes and object class predictions. In addition, the backbone's output includes multiple scaled outputs, which are extracted from different stages in a pipeline of the backbone to inform multi-scale object detection. Note that outputs of the extracted stages are functions of the earlier extracted stages, thus the size and characteristics of backbone's outputs are different to each other. This articulate architecture makes the distillation of head models is highly non-trivial.

4 INFERENCE TIME EVALUATION

We now evaluate complete processing pipelines over a distributed mobile device-edge server system, which is the main focus of this contribution. To this aim, we implement the necessary modules for processing, communication and synchronization within a custom distributed pipeline. The results we present in the following explore an ample range of hardware (see Table 4) and communication data rates to provide an evaluation of the interesting interplay between the delay components. The original model is DenseNet-201, and local computing and pure edge computing *Org. (MD)* and *Org. (ES)*

Table 4: Hardware specifications

Computer	Processor	Speed [GHz]	RAM [GB]
RPI3b+	ARM Cortex A53 (quad-core)	1.2	1
UP Board	Intel Atom x5-Z8350 (quad-core)	1.92	4
Jetson TX2	ARM Cortex-A57 (quad-core) + NVIDIA Denver2 (dual-core)	2.0	8
Laptop	Intel i7-6700HQ (octa-core)	2.6	16

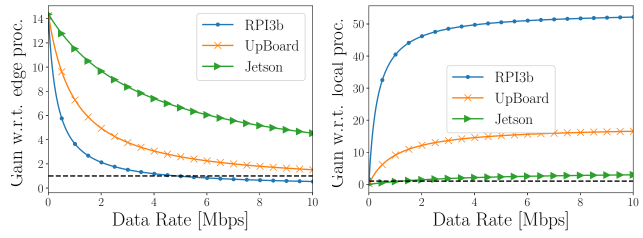
are compared with the mimic model with bottleneck (*Mimic w/B*) at the first splitting point.

The set of plots in Fig. 3 reports the gain – expressed as the ratio between the capture-to-output time T of the *Mimic w/B* model and that of pure offloading (*Org. ES*)/local processing at the mobile device (*Org. MD*) – as a function of data rate in different hardware/network configurations. Figures 3 (a) and (b) show the gain trends for different mobile devices when the edge server is the laptop. Intuitively, the larger the data rate, the smaller the gain with respect to *Org. ES*, as the reduction in τ_{data} granted by the bottleneck decreases compared to the possible disadvantage of executing part of the processing on a slower platform. Note that slower mobile devices emphasize the latter, to the point that the slowest considered embedded device (Raspberry Pi 3) has a gain smaller than 1, that is, the proposed technique leads to larger capture-to-output time compared to *Org. ES* if the mobile device is much weaker than the edge server, and the communication link has high capacity.

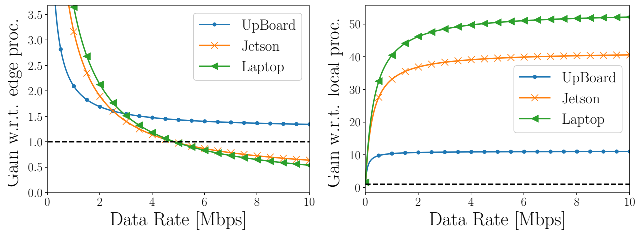
Conversely, a configuration with a strong mobile device emphasizes the general reduction of complexity of the *Mimic w/B*, leading to a substantial gain even when the channel has high capacity. The opposite trend is observed when we measure the gain with respect to *Org. MD*. A larger capacity reduces the time needed to transfer the tensor and increases the gain in *Mimic w/B*. Note that in a range of small channel capacity determined by the strength of the embedded device, the gain is below 1, that is, local processing is a better option.

Similar trends with respect to the data rate are observed in Figures 3 (c) and (d), where the weakest mobile device (Raspberry Pi 3) is used with the available edge servers. Intuitively, this configuration penalizes our approach in comparison with *Org. ES*, as even a small amount of processing positioned at the mobile device may take considerable time. Clearly, this effect is amplified in the presence of a strong edge server, and we observe a reduced range of data rates where our technique provides a gain with respect to *Org. ES*. However, the weak processing capabilities of Raspberry Pi 3 leads to a considerable gain of distributed *Mimic w/B* with respect to local processing in a range of channel capacities.

Overall, *Mimic w/B* provides a substantial gain in configurations where the processing capacity of the mobile device



(a) Gain w.r.t. full offloading, Laptop as ES. (b) Gain w.r.t. local processing, Laptop as ES.



(c) Gain w.r.t. full offloading, RPI3b+ as MD. (d) Gain w.r.t. local processing, RPI3b+ as MD.

Figure 3: Ratio between the total capture-to-output time T of the proposed technique (Mimic w/B) and pure offloading (a) and (c), and local processing (b) and (d) for different hardware configurations (ES: Edge Server, MD: Mobile Device).

and edge server are not excessively different, and the channel conditions are not either excessively large or small data rates. In essence, the proposed approach represents an intermediate option between local processing and edge computing in the range of conditions where these extreme points are operating suboptimally.

5 CONCLUSIONS

In this paper, we proposed an approach to effectively split DNNs in edge-assisted systems. Our technique is based on network distillation, which is applied to the head portion of the split model. The distilled head model is then modified to introduce a bottleneck. Further studies are necessary to understand the general implications of our approach. However, intuition suggests that it could roughly correspond to a special case of autoencoder directly mapping the input of the DNN to the input of a later layer. The student-teacher based approach allows the construction of small models with aggressive bottlenecks. Results on real-world embedded computers identify the range of communication rates in which the proposed technique is effective.

ACKNOWLEDGMENTS

This work was partially supported by DARPA under grant HR00111910001 and Google Cloud Platform research credits.

REFERENCES

[1] Alessandro Achille and Stefano Soatto. 2018. Information dropout: Learning optimal representations through noisy computation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018).

[2] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E Dahl, and Geoffrey E Hinton. 2018. Large scale distributed neural network training through online distillation. In *Sixth International Conference on Learning Representations.*

[3] Jimmy Ba and Rich Caruana. 2014. Do deep nets really need to be deep?. In *Advances in neural information processing systems.* 2654–2662.

[4] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. 2012. Fog computing and its role in the internet of things. In *Proceedings of the first edition of the MCC workshop on Mobile cloud computing.* ACM, 13–16.

[5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition.* Ieee, 248–255.

[6] Li Fei-Fei, Rob Fergus, and Pietro Perona. 2006. One-shot learning of object categories. *IEEE transactions on pattern analysis and machine intelligence* 28, 4 (2006), 594–611.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 770–778.

[8] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely Connected Convolutional Networks. In *CVPR*, Vol. 1. 3.

[9] Hyuk-Jin Jeong, InChang Jeong, Hyeon-Jae Lee, and Soo-Mook Moon. 2018. Computation Offloading for Machine Learning Web Apps in the Edge Server Environment. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS).* IEEE, 1492–1499.

[10] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. 2017. Neurosurgeon: Collaborative Intelligence Between the Cloud and Mobile Edge. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '17).* ACM, New York, NY, USA, 615–629. https://doi.org/10.1145/3037697.3037698

[11] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *Third International Conference on Learning Representations.*

[12] Nicholas D. Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. 2016. DeepX: A Software Accelerator for Low-power Deep Learning Inference on Mobile Devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks (IPSN '16).* IEEE Press, Article 23, 12 pages.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[14] Jinyu Li, Rui Zhao, Jui-Ting Huang, and Yifan Gong. 2014. Learning small-size DNN with output-distribution-based criteria. In *Fifteenth annual conference of the international speech communication association.*

[15] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.* 4510–4520.

[16] Mahadev Satyanarayanan, Victor Bahl, Ramón Caceres, and Nigel Davies. 2009. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing* (2009).

[17] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2818–2826.

[18] Gregor Urban, Krzysztof J Geras, Samira Ebrahimi Kahou, Ozlem Aslan, Shengjie Wang, Rich Caruana, Abdelrahman Mohamed, Matthai Philipose, and Matt Richardson. 2017. Do deep convolutional nets really need to be deep and convolutional?. In *Fifth International Conference on Learning Representations.*