Deep Collaborative Optimization Techniques

Van-Nhiem, Tran

Department of Computer Science and Information Engineering, National Central University, Taoyuan 32001

tvnhiemhmus@g.ncu.edu.tw

1 Introduction:

Since 2012 after AlexNet [1] wining the image classification achieved top-5 error of 15.3% of image classification using deep learning, nowadays deep learning is the most effective solution for task on supervise and unsupervised machine learning challenges tasks. Deep learning methods have made an impressive breakthrough performance in a wide variety of applications, and the three main important fields include computer vision, natural language processing, and speech recognition. Deep learning has been the exponential growth and great attention and tremendous investments on many scientist, laboratories and organization all around the world. Furthermore, the existing of mobiles and integrated embedded device more hundreds of billion devices around the worlds. Deep learning techniques have been transformed how mobile and embedded devices interpret and react smarter and efficiency to the world. The growth of the future of machine learning on embedded system and mobile device will need more addition breakthrough in efficient of learning algorithm and hardware development and software development. This study will aim try to create the optimal and efficient learning algorithm to deploy on mobile and embedded system. Neural networks are the core of deep learning, which is a subset of machine learning (ML) and the architecture inspired by the structure of the human brain. They transform an input through a series of comprises of neurons layers, each neuron has a set of weights, and the neurons belonging to different layers are connected. The final layer outputs are the answer, for example, whether an object is recognized or not. In practice, there are two stages of deep learning. The first stage of deep learning is training which is when the neural network learns to extract features automatically from the input data. The outcome of the training phase is a neural network with weights configured so that it works for the images it was trained on, called the neural network model. Many image databases public such as Coco [2] for object segmentation, Cifar10 [3] for small image classification, ImageNet [4] contains 10s of millions of classified images. The second stage, called inference, is deploying the network onto the embedded device that will run the vision application, for example, self-driving car with multi object tracking, smart camera surveillance control system, etc. During the inference phase, a new image is an input to the neural network, and it outputs a prediction of the answer.

Convolutional Neural Networks (CNNs) is a type of neural network closely correlated with computer vision, which is the part of artificial intelligence that helps computers see, recognize, and process images. CNNs use a structure originally motivated by the part of the brain visual cortex that processes visual information. When it comes to recognizing faces CNNs also be almost as good as humans, and in situations that involve noticing fine-grained characteristics – to identify a particular breed of dog or species of bird, for example CNNs have demonstrated accuracy that's even better than humans, CNN architecture is one of the active research areas since 2012 with many breakthroughs in model architecture design is shown in **Figure 1**. Outside of computer vision, neural networks (NNs) on natural language processing, because they're good at parsing sentences and predicting patterns in word usage. They're used for drug discovery and toxicology because they're good at predicting the interactions between molecules and biological proteins, and for games like chess and Go because they're good at evaluating positions and suggesting the next moves and many more applications in different fields implemented neural network in general and CNNs in specific. Deep Neural Networks (DNNs) showed many achieve state of the art out human performance in many applications. However, the explosive of increasing the model size and computation cost these are really challenges of deploying neural network without powerful machine. Traditionally, CNNs are typically trained on a high-performance CPU/GPU decision making has been concentrated in the cloud, server or at the center. This places huge strain on the technical and commercial of big techs company and organization. As example the computation expensive by using high performance GPU to run CNNs model demonstrated in **Figure 2**, showing correlation between model parameter or model size and the model memory usage of training on NVIDIA GPU Titan Xp. With the new era of machine learning many researchers and engineers try to bring Deep learning to deploy in many low power devices such as mobile and embedded devices instead of cloud or high-performance computation resource device such GPU.

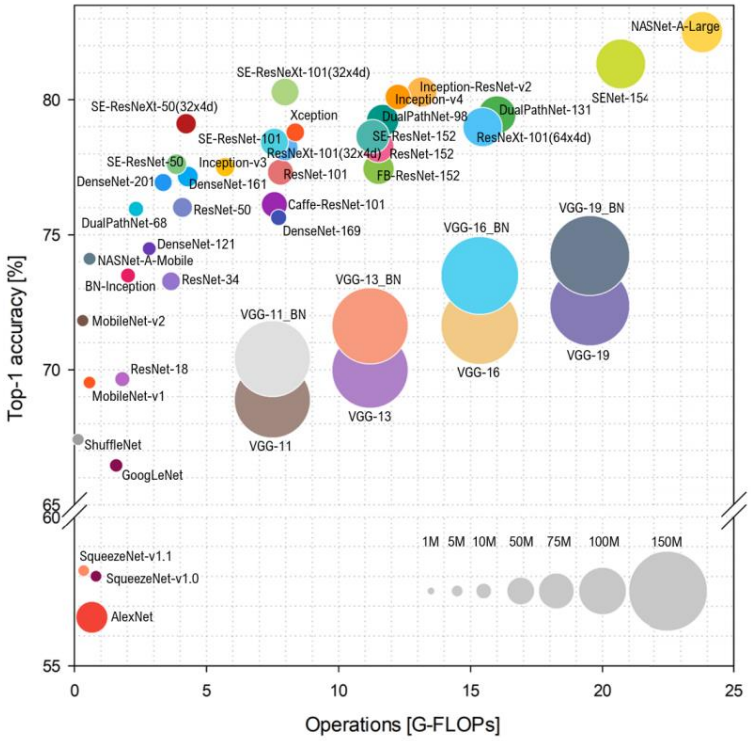

Figure 1. List of state-of-the-art convolution neural networks correlate with error in image recognition in ImageNet dataset since 2012 to 2018 (Bianco et al [5])

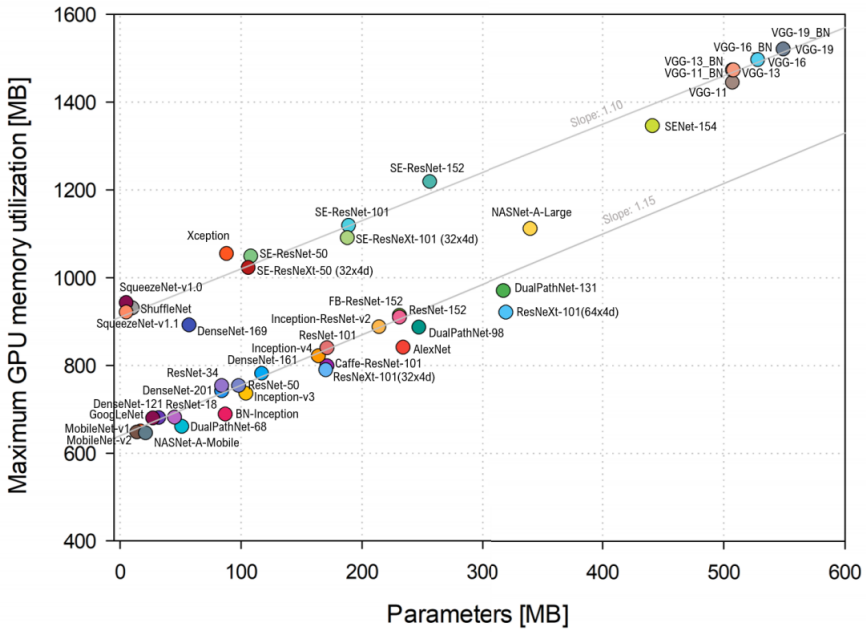

Figure 2. Plot of the initial static allocation of the model parameters and the total memory utilization with batch size 1 on the Titan Xp. (Bianco et al [5]).

For machine learning to fulfill its promise in many mobiles and embedded devices there are existing many challenges we need to address. These fundamental challenges include memory-constrained, low latency, privacy, bandwidth, power consumption, and real-time performance, hardware accelerator and many more. The concept of pushing computing closer to where sensors gather data is a central point of modern embedded systems at the edge of the network. To enable intelligence and autonomy at the edge, many applications from automated machinery and industrial robots on a factory floor to self-guided vacuums in the home, to an agricultural tractor in the field these applications mostly require the processing must happen locally. According to Arm the security surveillance [6], the IP camera market is on pace for an annual growth rate of 20% through to 2021 reaching over 500 million shipped units, and personal robots are growing at a staggering 75% expected to reach 2 million units shipped in 2021, the smart home market is growing at 14% to 88 million in 2021, and other devices like drones, augmented reality (AR) mixed reality (MR) equipment and action cameras are shifting from emerging to widely adopted in terms market. Furthermore, the number of chips of embedded device show the billions of chips increasing every year in the future is shown in **Figure 3.** This shows the exponential growth and the important future development and requirement and attention of our research in order to adapt and respond to the future technology.

In specific one area of the fields, embedded vision has proved to enhance solutions in a broad range of markets. Understanding of deep learning efficiency in design architecture combined with increasing embedded compute performance will lead to explosive growth in embedded vision products. From a technical viewpoint, the sub-problems that embedded vision can be divided into images capturing high-quality and video, analyzing them, identifying and recognizing objects, and be able to give a response to human users in real-time. In traditional computer vision, feature extraction is key, and algorithms for detecting features such as edges, corners, and objects are based on mathematical models, object detection and recognition are difficult, if not impossible to solve using classic rules-based programming. Furthermore, to implement traditional computer vision and manually decide which features to extract and which algorithm to use for the specific class of objects these are really challenges by time consuming, and human expert requirements. On the other hand, modern deep learning can address issue much more efficient in many ways, but the complexity and hardware resources requirement for running modern deep learning still big challenges of embedded vision device. For example, running a CNNs still takes quite a bit of memory and a fairly high degree of computational complexity. AlexNet [1], for example, which is one of the earliest CNN's developed, requires 1.45 billion operations per input image, with weights size 240 MB. It can be really difficult and inefficient to deploy the original model into the embedded device

with the memory constrained existing. The popular embedded hardware currently uses on many applications such as Raspberry pi bases on Arm CPU cortex A and Jetson Nano base on NVIDIA GPU.

For running the original CNNs, even the recent development of CNN architecture with a lightweight model like Resnet 50 [7] that already exceeds thee storage limit by 100x, MobileNetV2 [8] exceed by 22x time on Raspberry Pi 1 model only has 256 MB of memory. Even seem impossible to run CNNs on embedded Arm microcontroller cortex-M4 to M7 with 16kB to 2MB of flash memory and SRAM from 16kB to 1MB is shown in **Table 1** as an example. If not impossible, for an embedded system to meet these kinds of processing and memory requirements. But, by making some changes to the way CNNs operate.

Table 1: Series available STM32 microcontrollers and their characteristics (*STM32 Electronic [9]*)

Name	Architecture	Flash (kB)	RAM (kB)	CPU (MHz)	Price (USD)
STM32F030CC	Cortex-M0	256	32	48	1.0
STM32L476	Cortex-M4	1024	128	80	5.0
STM32F746	Cortex-M7	1024	1024	216	7.5
STM32H743ZI	Cortex-H7	2048	1024	400	9.0

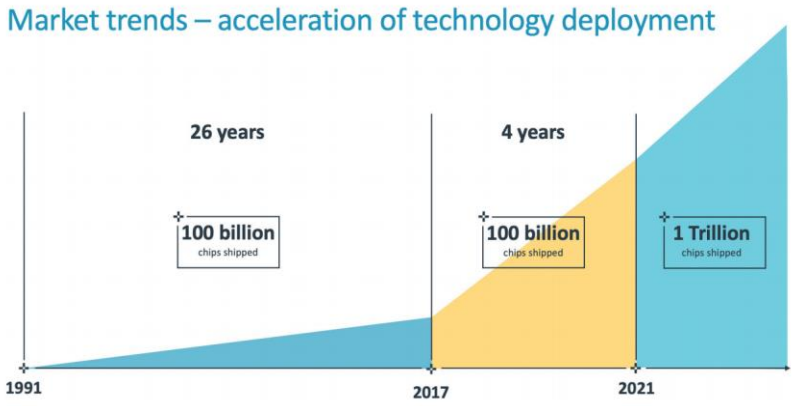

Figure 3. Market trends and the acceleration of technology deployment (Arm white paper [6])

In this study, we propose method can make them more compact and bring them within reach of the existing of current hardware resources deploying advanced deep learning algorithms on edge devices especially for computer vision applications like autonomous vehicles and IoT requires special capabilities. our goals are to create practical neural-network based algorithms to address the issue of deploying large model size can't fit into memory of embedded device. We proposed the machine

learning algorithm base on combination knowledge distillation and deep model compression of CNNs model. The main concept of our algorithm first using knowledge distillation which focus on techniques train the large and complex network on ensemble model which can extract important features from the given data. The knowledge from the complex model that call teacher model can transfer the knowledge to new smaller model call student model to obtain high-accuracy. Second, we jointly train deep model with different deep compression technique in different parts of model including parameter pruning, clustering, aware quantization during training and post training quantization. The main advantage of the algorithm extremely compresses the deep CNNs model with optimization model by knowledge distillation and jointly deep compression. The algorithm makes CNNs model to become fewer parameter to simulate the original large model and accelerate the performance with less efficient parameter. The most important we can bring the state of the art CNNs model to the small from hundred to several MB to KB scale fit into the SRAM of small embedded device such Raspberry Pi 0, 1 and Arm microcontroller cortex-M series.

2. Related Work

2.1 Knowledge Distillation.

Knowledge distillation is a model compression method that was first proposed by Bucila et al. [10] and generalized by Hinton et al. [11]. In general, it can be described as a special instance of learning with privileged information, e.g. Vapnik & Izmailov [12] In distillation knowledge or the extraction of features of a given supervised dataset is transferred from the large complex model call the teacher model to the smaller model call student model that mimic teacher model by minimizing a loss function with the target is the distribution class probabilities predicted by teacher model. Because of the issue of the probability distribution of correct class at significant high probability and very low with other class probabilities almost close to 0. Hinton et al. [11] proposed the term “softmax temperature” that improve base on the standard softmax function is express in Eq. (1) the probability p of the class of i base on softmax temperature term calculated from the logit z express as Eq. (2):

$$\frac{\exp(z_j)}{\sum_j \exp(z_j)} \tag{1}$$

$$pi = \frac{\exp(\frac{z_i}{T})}{\sum_j \exp(\frac{z_j}{T})} \tag{2}$$

where T is the temperature parameter usually set to 1, using T at higher value to produce a softer probability distribution over classes.

2.2 Model compression techniques

2.2.1 Model Pruning

Model pruning consists in fine tuning the weights in various connection matrices of a pre-trained network while keeping only the relevant ones as the example shown in **Figure 4.** After training a network using a standard method, the small weights (in absolute value) below a threshold are removed. The original dense network is transformed into a sparse one preserving the most important connections. Han et al. [13] have experimented with this method reducing the number of parameters by 9 times and 13 times for the AlexNet [1] and the VGG16 [14] models respectively. Zhu et al. [15] proposed method effectiveness of pruning algorithm by obtaining sparsity of model with small sparsity then gradual increasing sparsity on each layer of the model added the binary mask variable which is the same size and shape as the layer's weight tensor and determine which of the weight participate in the forward execute is shown in **Figure 5**. Furthermore, in their experiment the pruning model can achieve 10x reduction in number of non-zero parameter with minima loss in accuracy.

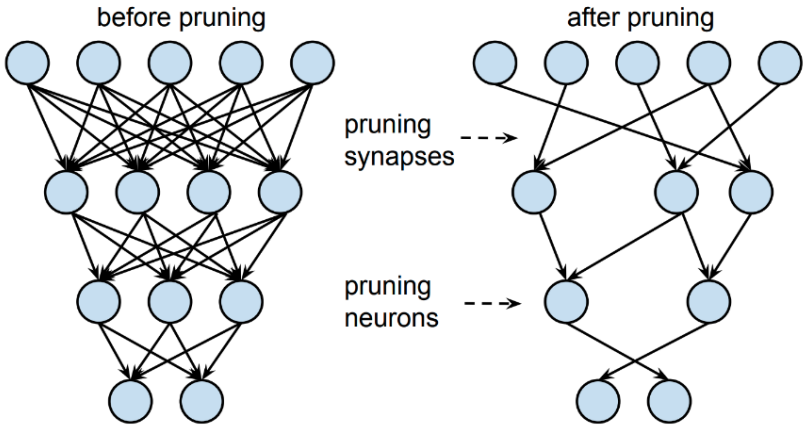

Figure 4. synapses and neurons before and after pruning (Han et al.[13])

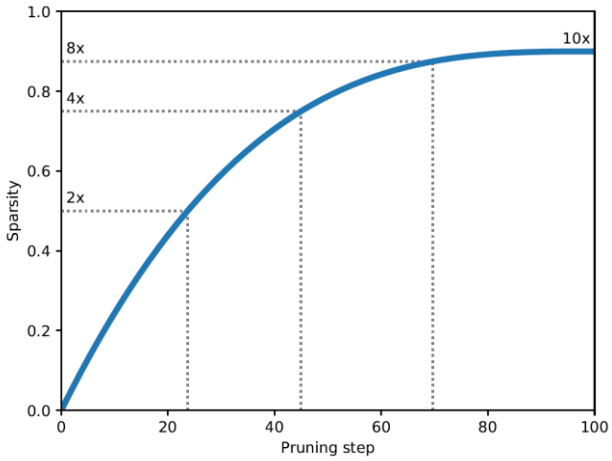

Figure 5. The demonstration of model gradual pruning magnitude algorithm increasing from initial value to the final value. (Zhu et al. [15])

2.2.2 Model weight clustering method:

Han et al. [13] applied the weight sharing base on k-means clustering to identify the shared weights for each layer, all the weights at the same cluster will share the same weight. Sharing weight between each layer by centroid initialization method including forgy, density-based, and linear are demonstrated in **Figure 6.** show the experiment result in weight's distribution on AlexNet [1] of 1 convolution layer. (Forgy) random initialization by choosing k observation of dataset and uses as the initial centroid. (Density-based) initialization linearly spaces the CDF of the weights in the y-axis then determines the intersection horizontal with CDF. Finally, determination the vertical intersection on the x-axis, and this point become the centroid. (Linear) by initialization linearly spaces the centroids between the [min, max] of the original weights. In the experiment on AlexNet [1] of three methods compared the accuracy Han pointed out the linear initialization works best and the k-means clustering of the weight for each layer expressed in Eq. (3) to minimize the within-cluster sum of squares, the partition n original weight W= {w_1, w_2, … w_n} into k clusters C= {c_1, c_2, … c_k} the n much larger than k n>>k.

$$\underset{C}{\arg min} \sum_{i=1}^{K} \sum_{w \in ci} |w - ci|^2 \tag{3}$$

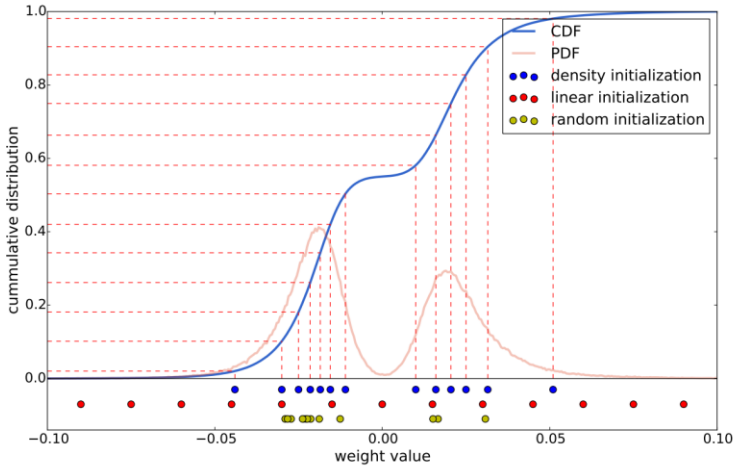

Figure 6. Three different centroids initialization method apply in weight clustering

2.2.3 Model quantization

The model quantization is to shrink the representation floating number at higher or 32-bit to the smaller at 8 bit or smaller by storing min and max for each layer, then compressing each float value from 32-bit to 8-bit or smaller representing the closest real number in the linear set of 256 for 8-bit within range the expression of quantization can be formularized in mathematic express in Eq. (4). (5). **Figure 7.** showing the example of shrinking the value from 32-bit to 8-bit representation. Fan, Stock et al. [18] showed by applying quantize a different random subset of weight during each forward is express in Eq. (6) and the resulting of replace the original weight W with noisy matrix w_{noise}, during the forward pass to compute a noisy output y_{noise} is express in Eq.(7) allowing for unbiased gradients to flow through the other weights. Controlling the amount of noise and its form allows for extreme compression rates while maintaining the performance of the original model. The quantization noise technique makes model size even further by more than 15x compare to int4 quantization with 8x reduction size. In their experiment showed Quant-Noise to PQ leads to new state-of-the-art trade-offs between accuracy and model size. For computer vision, their study report 80.0% top-1 accuracy on ImageNet [4] by compressing an EfficientNet-B3[19] to 3.3 MB; the demonstration and schematic of quantize noise is shown in **Figure 8.**

$$s = \frac{(Wmax - Wmin)}{2^n} \; ; \; z = round(Wmin/s) \tag{4}$$

$$quantized_{value} = \frac{float_{value}}{s} \tag{5}$$

where S is the scale floating point at n-bit value, z is shift bias term, $quantized_{value}$ is the float point value quantize.

$$\delta(b_{kl}|J) = \begin{cases} \varphi b_{kl} \; if\,(k,l) \in J, \\ b_{kl} \;\; otherwise \end{cases} \tag{6}$$

where b_{kl} is the target quantize compute block with m x q dimension, $\delta(b_{kl}|J)$ is the the transform adding distortion or noise, φ is the noise function acting on compute block, $J \in \{(k,l)\}$ for $1 \leq k \leq m$, $1 \leq l \leq m$.

$$w_{noise} = (\delta(b_{kl}|J))_{kl}\,; \;\; Y_{noise} = w_{noise}x \tag{7}$$

where x is an input vector, w_{noise} is the result adding noise during forward pass, Y_{noise} is the noisy output.

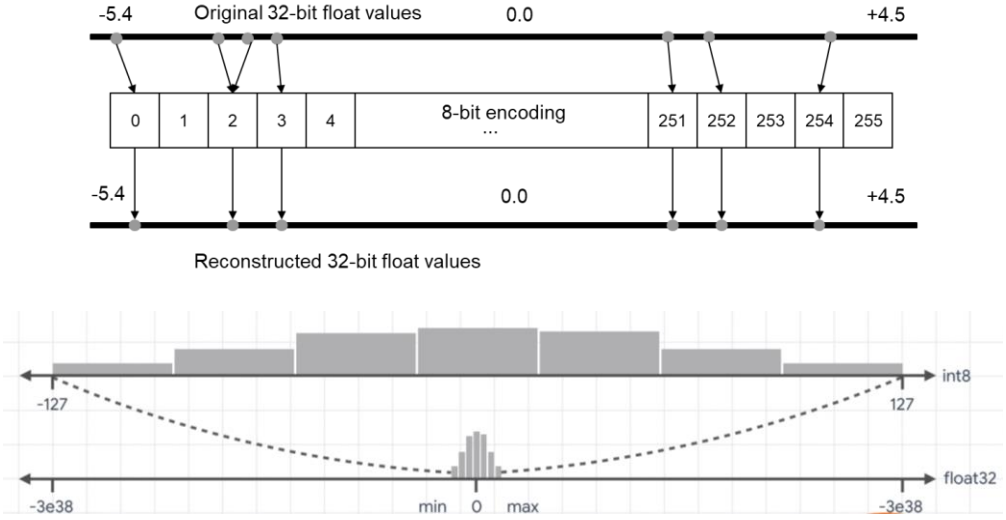

Figure 7. The demonstrate of the linear quantization compressing 32-bit to 8-bit representation

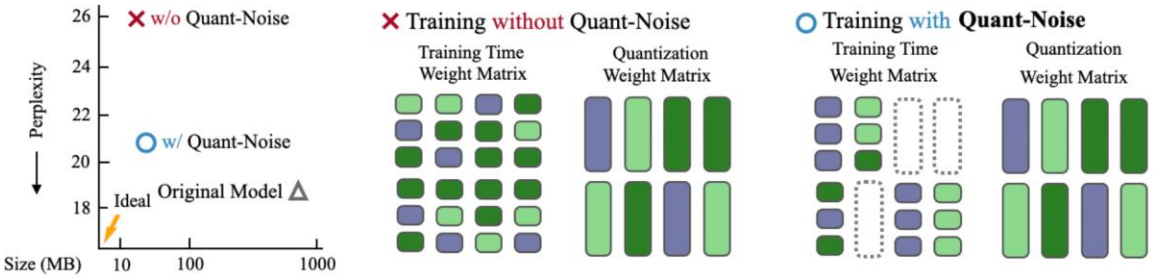

Figure 8. Quant-Noise train models to be resilient to inference-time quantization by mimicking the effect of quantization method during training time. (Fan, Stock et al. [18])

3. Proposal method

3.1 End to end approach computer vision on embedded devices.

Understand the fundamental image classification models that are optimized for running on embedded devices, offer fast classification without needing to go to the cloud or server. Techniques that enable energy-efficient processing of CNNs on embedded devices with limited memory without sacrificing application accuracy or increasing hardware cost are critical to the wide deployment of embedded vision. We proposed a method call end to end approach computer vision domain on embedded devices the flow chart is shown in **Figure 9.** Our approach from the step of developing target applications to algorithm development and deployment in the specific application on different embedded device platform. At our pipeline end to end approaching, the first stage is the selection of open-source machine learning framework that enable specific neural network algorithm in high-level languages train on large of our custom collected data sets and public data sets. The popular deep learning frameworks include TensorFLow, PyTorch, and Caffe. In the implementation of these frameworks, the output will be the network with the weight configure that works for the dataset it was trained on also the similar target datasets. The second stage is the embedded applications, our target tries to develop iris identification by using low power hardware with low cost and efficient performance and energy consumption. The third and fourth stage are our development optimization algorithm that transforms the large and deep CNNs matching with the capability of embedded device, rather than the capacity of the server or work station. The fifth and final stage takes the well optimize algorithm after training on our lab custom collected dataset and public datasets and convert it into the embedded system compile platform and testing reference running on devices with various different hardware platforms focus on energy consumption, latency, and real-time performance efficiency.

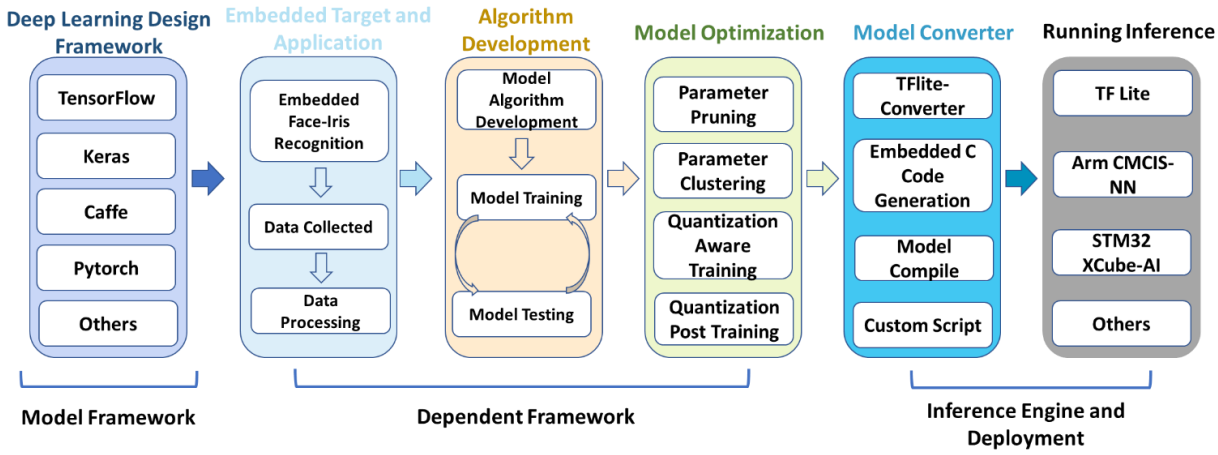

Figure 9. Flow chart of end to end approaching deep learning on embedded applications

3.2 Development optimization algorithm (DCCDKA).

At the dependent framework in our end to end approach, our approach optimization algorithm calls deep collaborative optimization on knowledge distillation is shown in **Figure 10**. first using knowledge distillation focus on techniques to train the large and complex network on ensemble model which can extract important features from the given data. The knowledge from the complex model that calls teacher model teacher with meaningful feature extraction from the large trained dataset can transfer to the new smaller model call student model also train on a similar dataset. Moreover, both the teacher model and student jointly apply collaborative optimization techniques at each convolution layer is shown in part collaborative optimization in **Figure. 10** is the combination of layer pruning then clustering and finally quantization layers. The model pruning algorithm will implement the gradual pruning algorithm Zhu et al. [15]. the pruning magnitude is increased from an initial sparsity s_i value to a final sparsity s_f value, and the during pruning schedule the rate of pruning increasing will strongly correlate with the learning rate schedule during training the network. throughout of n pruning steps, starting at training step t_0 and with sparsity frequency Δt:

$$s_t = s_f + (s_i + s_f)(1 - \frac{t-t_0}{n\Delta t})^3 \text{ with } t \in \{t_0 + t_0 + \Delta t, \dots t_0 + n\Delta t\} \quad (8)$$

where s_t is the sparsity as step t, s_i and s_f is the initial and final sparsity, t is the training step, Δt is the sparsity frequency.

After the prune layer, the weight will be cluster into the different groups by initialization the cluster number using the k-mean algorithm, the linear initialize the centroid between max and min value of the original weight. All the weights in the same cluster will share the same weight. The prune layer will have n original weight W= (w_1, w_2, w_3… w_n) all the prune weight then clusters into m cluster k = {k_1, k_2, k_3…, k_m} with m much smaller than n, m<<n. the value of each cluster will be minimizing by the sum of squares.

$$\underset{C}{\arg min} \sum_{i=1}^{m} \sum_{w \in ki} |w - ki|^2 \tag{9}$$

where w is the original weight of the layer, m is the number of clusters, the k is the weight of each cluster.

Finally, the weight in each cluster will apply the fixed-point scalar quantization adding noise method adaptive from Fan, Stock et al. [18], the scale factor s and the bias z as mentioned in section 2 Eq. (4). The single floating-point weights are replaced by n bit fixed-point numbers, the weights are rounded to one of 2^n express in Eq. (10) to the nearest neighbour in the cluster, and the noise function quantize at intx-bit replace the weight with the rounding output is express in Eq. (11).

$$c = \left(round\left(\frac{w_{kl}}{scale} + z\right) - z\right) \times s \tag{10}$$

$$\varphi_{intx}(w) = \left(round\left(\frac{w}{scale} + z\right) - z\right) \times scale \tag{11}$$

where c is the rounding function, s, z is the scale factor, and the bias updated during training, w is the weight of layer.

The distillation loss in the algorithm the loss refers to the loss we train the original model with. Because the teacher and student model will be trained on parallel, the deep teacher network will train and optimize by using categorical cross-entropy loss is express in Eq. (12) then the pretrained teacher model with the output the softmax with the temperature output layers that mentioned section 2 Eq. (2).

$$CE = -\log\left(\frac{e^{s_p}}{\sum_j^C e^{s_j}}\right) \tag{12}$$

The student will give two outputs. The first output is softmax with the temperature term as the same as the teacher model and another output is the standard softmax function with the temperature =1 The back propagation only on the student model because the teacher model is a pre-trained one and we are distilling the teacher model to student. The first loss function calculation to optimize the student model correlate with teacher model that is described in the cross-entropy loss between teacher's soft targets (softmax of teacher with temperature at T) and the student's soft predictions

(student model at the same temperature) is expressed in Eq. (13). The second loss function is the cross-entropy loss between the student's hard prediction and the ground truth output label is expressed in Eq. (14). The final loss is the obtained by adding the both, The standard loss will be calculated between input and parameter of the student parameter $L(X_{input}, W)$ is expressed in Eq. (15)..

$$L1 = H(\sigma(Z_t; T = \tau), \sigma(Zs, T = \tau)) \tag{13}$$

where the $\sigma(Z_t; T = \tau)$ softmax output of the teacher model at $T = \tau$, $\sigma(Zs, T = \tau)$ is the softmax output of the student model at $T = \tau$.

$$L2 = H(Y, \sigma(Zs, T = 1)) \tag{14}$$

where σ(Zs;T=1)->standard softmax output of student model, Y is the truth output labels.

$$\mathrm{L}(\mathrm{X}_{\mathrm{input}}, \mathrm{W}) = H\big(\sigma(Z_t; T = \tau), \sigma(Zs, T = \tau)\big) + H(Y, \sigma(Zs, T = 1)) \tag{15}$$

where X_{input} is the input, W are the student model parameters, Y is the ground truth label, H is the cross-entropy loss function, σ is the softmax function parameterized by the temperature T . Zs and Z_t are the logits of the student and teacher model.

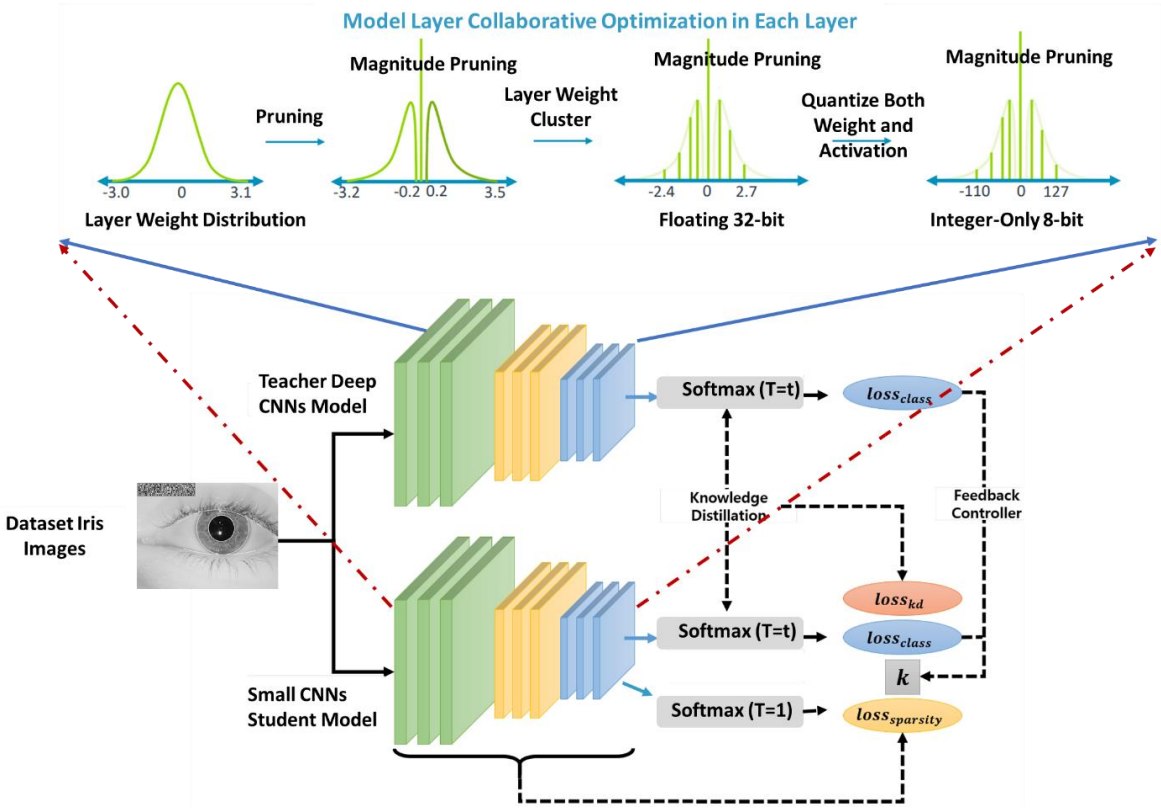

Figure 10. Deep collaborative compression distillation knowledge algorithm

Reference:

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

[2] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Doll´ar, and C. L. Zitnick. Microsoft coco: Common objects in context. In European conference on computer vision, pages 740–755. Springer, 2014.

[3] Alex Krizhevsky, Learning Multiple Layers of Features from Tiny Images 2009. URL http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf .

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. FeiFei. Imagenet: A large-scale hierarchical image database. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 248–255. IEEE, 2009.

[5] Simone Bianco, Rémi Cadène, Luigi Celona, Paolo Napoletano. Benchmark Analysis of Representative Deep Neural Network Architectures. IEEE Access 6: 64270-64277 (2018).

[6] Radhika Jagtap, Alessandro Grande. Adding Intelligent Vision to Your Next Embedded Product, Arm white paper 2019. URL https://www.arm.com/resources/white-paper/embedded-intelligent-vision .

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Conference on Computer Vision and Pattern Recognition, pages 4510–4520, 2018.

[9] STMicroelectronics URL https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html.

[10] [Bucila, Caruana, and Niculescu-Mizil 2006] Bucila, C.; Caruana, R.; and Niculescu-Mizil, A. 2006. Model compression. In SIGKDD, 535–541. ACM.

[11] [Hinton, Vinyals, and Dean 2015] Hinton, G.; Vinyals, O.; and Dean, J. 2015. Distilling the knowledge in a neural network. In NIPS Deep Learning and Representation Learning Workshop.

[12] [Lopez-Paz et al. 2015] Lopez-Paz, D.; Bottou, L.; Scholkopf, B.; and Vapnik, V. 2015. Unifying distillation and privileged information. arXiv preprint arXiv:1511.03643.

[13] Song Han, Huizi Mao, and William J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. CoRR, abs/1510.00149, 2015. URL http://arxiv.org/abs/1510.00149.

[14] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[15] Michael Zhu, Suyog Gupta: To prune, or not to prune: exploring the efficacy of pruning for model compression. CoRR abs/1710.01878 (2017).

[16] Jacob, B., et al. (2017). "Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference." CoRR **abs/1712.05877**.

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

[18] Fan, A., et al. (2020). "Training with Quantization Noise for Extreme Model Compression." CoRR abs/2004.07320.

[19] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks, 2019.

[20] CASIA-IrisV4 is released on Biometrics Ideal Test (12/14/2010) URL http://biometrics.idealtest.org/

[21] Proença, Hugo and Alexandre, Luís, UBIRIS: A noisy iris image database Inproceedings {ProencaAlexandre2005} Proceed. of ICIAP 2005 - Intern. Confer. on Image Analysis and Processing. URL http://iris.di.ubi.pt/index_arquivos/Page374.html