DEEP LEARNING ON THE EDGE: MODEL PARTITIONING, CACHING, AND COMPRESSION

DEEP LEARNING ON THE EDGE: MODEL PARTITIONING, CACHING, AND COMPRESSION

BY

YIHAO FANG, M.Eng.

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTING AND SOFTWARE

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

Doctor of Philosophy (2020) McMaster University

(Computing and Software) Hamilton, Ontario, Canada

TITLE:	Deep Learning on the Edge: Model Partitioning, Caching, and Compression
AUTHOR:	Yihao Fang M.Eng. (Software Engineering), McMaster University, Hamilton, Canada
SUPERVISOR:	Rong Zheng
NUMBER OF PAGES:	xv, 131

Lay Abstract

Edge artificial intelligence (EI) has attracted much attention in recent years. EI is a new computing paradigm where artificial intelligence (e.g. deep learning) algorithms are distributed among edge nodes and end devices of computer networks. There are many merits in EI such as shorter latency, better privacy, and autonomy. These advantages motivate us to contribute to EI by developing intelligent solutions including partitioning, caching, and compression.

Abstract

With the recent advancement in deep learning, there has been increasing interest to apply deep learning algorithms to mobile edge devices (e.g. wireless access points, mobile phones, and self-driving vehicles). Such devices are closer to end-users and data sources compared to cloud data centers, therefore deep learning on the edge leads to several merits: 1) reduce communication overhead (e.g. latency), 2) preserve data privacy (e.g. not leaking sensitive information to cloud service providers), and 3) promote autonomy without the need of continuous network connectivity. However, it also comes with a trade-off that deep learning on the edge often results in less prediction accuracy or longer inference time. How to optimize such a trade-off has drawn a lot of attention among the machine learning and systems research communities. Those communities have explored three main directions: partitioning, caching, and compression to solve the problem.

Deep learning model partitioning works in distributed and parallel computing by leveraging computation units (e.g. edge nodes and end devices) of different capabilities to achieve the best of both worlds (accuracy and latency), but the inference time of partitioning is nevertheless lower bounded by the smallest of inference times on edge nodes (or end devices).

In contrast, model caching is not limited by such a lower bound. There are two

trends of studies in caching, 1) caching the prediction results on the edge node or end device, and 2) caching a partition or less complex model on the edge node or end device. Caching the prediction results usually compromises accuracy, since a mapping function (e.g. a hash function) from the inputs to the cached results often cannot match a complex function given by a full-size neural network. On the other hand, caching a model's partition does not sacrifice accuracy, if we employ a proper partition selection policy.

Model compression reduces deep learning model size by e.g. pruning neural network edges or quantizing network parameters. A reduced model has a smaller size and fewer operations to compute on the edge nodes or end device. However, compression usually sacrifices prediction accuracy in exchange for shorter inference time.

In this thesis, our contributions to partitioning, caching, and compression are covered with experiments on state-of-the-art deep learning models. In partitioning, we propose TeamNet based on competitive and selective learning schemes. Experiments using MNIST and CIFAR-10 datasets show that on Raspberry Pi and Jetson TX2 (with TensorFlow), TeamNet shortens neural network inference as much as 53% without compromising predictive accuracy.

In caching, we propose CacheNet, which caches low-complexity models on end devices and high-complexity (or full) models on edge or cloud servers. Experiments using CIFAR-10 and FVG have shown on Raspberry Pi, Jetson Nano, and Jetson TX2 (with TensorFlow Lite and NCNN), CacheNet is 58–217% faster than baseline approaches that run inference tasks on end devices or edge servers alone.

In compression, we propose the logographic subword model for compression

in machine translation. Experiments demonstrate that in the tasks of English-Chinese/Chinese-English translation, logographic subword model reduces training and inference time by 11–77% with Theano and Torch. We demonstrate our approaches are promising for applying deep learning models on the mobile edge.

Acknowledgements

I wish to express my deepest gratitude to Professor Rong Zheng, my Ph.D. supervisor, for all her support towards my research accomplishment. Her editorial vigilance and supportive nature most influence me. I feel grateful for her giving me a chance to continue my Ph.D. study at the most difficult moment.

I want to thank my advisory committee, Dr. Frantisek Franek, Dr. Emil Sekerinski, for their excellent constructive insights and invaluable feedback. I would like to thank my family for their understanding, support and love.

Contents

List of Figures

List of Tables

Notation, Definitions, and Abbreviations

Abbreviations

AI	Artificial intelligence
EI	Edge artificial intelligence
MLP	Multi-layer perceptron
CNN	Convolutional neural network
RNN	Recurrent neural network
MoE	Mixture of experts
VAE	variational autoencoder
IoT	Internet of things
NIST	National institute of standards and technology
MNIST	The modified NIST dataset of handwritten digits

CIFAR	Canadian institute for advanced research
CIFAR-10	A CIFAR dataset of color images in 10 different classes
CIFAR-100	A CIFAR dataset of color images in 100 different classes
FVG	The frontal-view gait dataset
BLEU	The bilingual evaluation understudy score
GPU	Graphics processing unit
TPU	Tensor processing unit
FPGA	Field-programmable gate array
TCP	Transmission control protocol
MPI	Message passing interface

Chapter 1

Introduction

Artificial intelligence (AI) is built upon algorithms that learn from a large amount of data, find useful patterns, and make predictions when given new data. During this process, it requires powerful and reliable computing resources to perform computation on the data, especially in cases of unstructured data like pictures and natural languages. Typically cloud computing serves this need, thanks to centralized data centers. However, challenges arise as more and more data need to be uploaded to the cloud in return for intelligence. The latency incurred in transferring data to cloud is not negligible and can be even harmful in some situations (e.g. autonomous driving). With the rapid proliferation of IoT devices, smartphones, and assisted driving or autonomous vehicles, there are growing interests to unlock the computing power at the edge of networks. As edge nodes (or end devices) are physically closer to (sensor) data sources and end-users, computation and data storage on edge nodes (and end devices) can relieve the bandwidth (capacity) burden of network infrastructures. Furthermore, response times can be substantially shortened due to proximity, which makes many real-time applications possible.

Edge computing is a distributed computing paradigm that brings computation and data storage closer to the location where they are generated and needed. An important research topic in edge computing studies AI algorithms on the edge (EI). With the extraordinary success of deep learning in recent years (a sub-field of AI), EI has attracted more and more attention in recent years. In computer vision, by running deep learning model in the cloud, objects (such as human, animal and transportation vehicles) can be detected and identified. But it is unclear whether real-time detection and classification on the edge are feasible. In natural language processing, translations of texts from one language to another have been done successfully by offloading the tasks to cloud. It remains to be investigated whether offline translation on end device can achieve the same quality.

Real-time inference on the edge is essential for human-in-the-loop interactive applications as response time is critical for good user experience. Excessive communication latency in data transfer renders cloud computing unsuitable for such applications. Furthermore, stable network connections are not always available between end devices and the cloud causing large jitters in interactions.

Apart from network connectivity and latency, data privacy is another important aspect to consider. In cloud computing, once a user's data is stored in the cloud, she has little control over where the data is and who has access to it. Although approaches exist to prevent data leakage in cloud services, they often need to trade-off computation time and inference accuracy with privacy preservation resulting in unacceptable utility to end users. Moving deep learning models to the edge enables local data analysis without leaking any sensitive information to third parties.

Lastly, autonomous robotic applications is an important driver for EI due to their

safety critical nature. A self-driving vehicle needs to operate safely even when network connectivity is unavailable due to poor network coverage or shadowing effects from obstructions in its environment. Pushing computation closer to data sources, such as on-board camera and LiDAR sensors is essential for these applications.

Compared to cloud data centers, edge devices usually have low computation resources and storage capabilities. Unfortunately, today's deep learning models have high space and time complexity making it challenging to run inference tasks on edge devices in real time. Though many efforts and much progress have been made in the research communities to mitigate this problem, with techniques such as deep learning model partitioning, caching, and compression, there is still much room to realize real-time EI. The main objective of the thesis is thus to develop effective mechanisms to accelerate edge computation for deep learning models. In particular, we investigate model partitioning and model caching approaches that take advantage of distributed computing resources at the edge. We also develop a language model that reduces the number of output classes resulting smaller network sizes in neural translation tasks.

Before elaborating on our contributions in each of these aspects, we first give a brief introduction to deep learning and various AI tasks where deep learning algorithms have been successfully applied. Then, classes of techniques to enable deep learning on edge devices are briefly reviewed.

1.1 Deep Learning

Deep Learning is a sub-area in machine learning. From the learning-task points of view, there are supervised learning, unsupervised learning and reinforcement learning. There are two main categories of models, discriminative models, and generative

models. Discriminative models are widely used in classification, while generative models are mostly applied to regression. Multi-layer perceptrons (MLP), recurrent neural networks (RNN), convolutional neural networks (CNN) and memory networks are mostly studied in the field of deep learning.

Deep learning has attracted much attention in different fields such as handwritten digit recognition, face recognition, and machine translation. They can be mainly classified into two group: computer vision and natural language processing. Apart from those topics, robust AI is a newly raised topic and has drawn much attention for the past few years. Robustness is mainly measured by uncertainty such as entropy and mutual information.

Our approaches in model partitioning, caching and compression are evaluated with the state-of-the-art deep learning models in the experiments. While partitioning, caching, and compression will be elaborated in the later chapters, an brief introduction of deep learning models are given in this section.

1.1.1 Handwritten Digit Recognition

Handwritten digit recognition is the task of recognizing human handwritten Arabic digits. Its applications are including but not limited to handwritten courtesy amount recognition in back cheques (Miah *et al.*, 2015) and numerical field extraction from handwritten incoming mail documents (Chatelain *et al.*, 2006). For the past decades, this task has drawn much research. On the MNIST dataset (which is a benchmark handwritten digit recognition dataset), various models have been proposed such as multi-layer perceptrons (MLP) (Cireşan *et al.*, 2011) and constitutional neural networks (CNN) (LeCun *et al.*, 1999; Ciresan *et al.*, 2011).

The best results on the MNIST dataset were attained by multi-column deep neural networks (MCDNN) (Cireşan *et al.*, 2012) and DropConnect neural networks (Wan *et al.*, 2013) with respectively the accuracy of 99.77% and 99.79%. MCDNN is inspired by sparsely connected neural layers found in mammals, in which multiple neural columns are trained to be the experts on differently preprocessed inputs. By taking the average of those neural columns' outputs, MCDNN decreases the error rate by 30-40% as opposed to those of the other DNN approaches.

DropConnect (Wan *et al.*, 2013), similar to Dropout (Hinton *et al.*, 2012), introduces dynamic sparsity to the model in order to prevent overfitting. However, unlike Dropout, DropConnect sets a randomly selected subset of weights (rather than activations) to zero. On the MNIST dataset, Li Wan and his colleagues observe that DropConnect neural networks converge slower than Dropout's, but end with a lower error rate.

On low-power devices, faster computation is crucial at both training and inference. It is beneficial to constrain weights to binary values (e.g. -1 or 1) during the forward and backward propagations, as simple accumulations are much faster than multiply-accumulate operations. BinaryConnect (Courbariaux *et al.*, 2015), which is that kind of neural networks, retains precision on the permutation-invariant MNIST dataset with a comparable error rate 1.01%, while poses great advantage on specialized low-power device.

1.1.2 Object Classification

Object classification is the task of identifying object classes in the images such as classes of airplanes, cars, dogs, and cats. There are two important datasets in this task: CIFAR-10 and CIFAR-100. CIFAR-10 consists of 60000 32-by-32 color images with

50000 for training and 10000 set aside for testing. There are 10 mutually exclusive classes with 6000 images per class. On this benchmark dataset, many approaches have been proposed such as sum-product networks (SPN) (Gens and Domingos, 2012), ReNet (Visin *et al.*, 2015), network in network (NIN) (Lin *et al.*, 2013), densely connected convolutional networks (DenseNet) (Huang *et al.*, 2017), and shake-shake regularization (Gastaldi, 2017).

SPNs are an expressive (Delalleau and Bengio, 2011) and tractable (Poon and Domingos, 2011) deep architecture with full probabilistic semantics. Robert Gens and Pedro Domingos (Gens and Domingos, 2012) first brought SPNs to discriminative learning with the propose of an efficient backpropagation-style algorithm. They tested their algorithm on the CIFAR-10 dataset and achieved the accuracy of 83.96% which was the best results at the time.

ReNet (Visin *et al.*, 2015) is a recurrent neural network architecture for object recognition, which replaces the convolution and pooling layer with four recurrent neural networks sweeping horizontally and vertically in both directions. The learning algorithm Adam (Kingma and Ba, 2014) was used and dropout was applied. On the CIFAR-10 dataset, ReNet achieved the accuracy of 87.65%. That suggests RNNs is a viable alternative to CNNs, which dominates object classification and most of the other computer vision tasks.

Unlike conventional convolutional layer, NIN (Lin *et al.*, 2013) uses micro neural networks (nonlinear function approximators) within the receptive field to abstract the data living on a nonlinear manifold. The resulting structure is stack-able and sub-sampling layers can be added in between as those in other CNNs. Experiments on CIFAR-10 and CIFAR-100 datasets show that NIN outperformed other approaches

at the time with respectively the accuracy of 91.19% and 64.32%.

In DenseNet (Huang *et al.*, 2017), short paths were created to connect the feature-maps of all preceding layers to the current layer, in order to alleviate the vanishing gradient problem. Different from ResNet (He *et al.*, 2016), DenseNet establishes direct connections where the feature-maps were sent as inputs to the subsequent layers. On CIFAR-10 and CIFAR-100, DenseNet's accuracy of 96.54% and 82.82% outperforms what ResNet achieved.

Shake-shake regularization (Gastaldi, 2017) alleviates the overfit problem by replacing the standard summation of parallel branches with a stochastic affine combination. It improves published results on CIFAR-10 and CIFAR-100 to the accuracy of 97.24% and 84.15%.

1.1.3 Face Recognition

Faces are biometric identities of human beings. Well-designed face biometric systems should stand the test of any imposters with any deliberate attacks. The importance of face recognition has drawn much research. Many neural network architectures have been proposed such as DeepFace (Taigman *et al.*, 2014), VGGFace (Parkhi *et al.*, 2015), FaceNet (Schroff *et al.*, 2015), SphereFace (Liu *et al.*, 2017) and VGGFace2 (Cao *et al.*, 2018). There have been found several large-scale face verification and identification datasets such as labeled faces in the wild (LFW) (Huang *et al.*, 2008), FGLFW (Deng *et al.*, 2017), IJB-A (Best-Rowden *et al.*, 2014), FaceCrub (Ng and Winkler, 2014), CASIA-WebFace (Yi *et al.*, 2014), and Megaface (Miller *et al.*, 2015). LFW contains more than 13000 images of faces with 1680 of the people pictured having two or more distinct photos. FGLFW is a renovation of LFW. Different from

LFW, 3000 similarly-looking face pairs are deliberately rather than randomly selected by human crowdsourcing. Megaface contains 4.7 million photos with 672057 Unique Identities. There are about 7 photos per person.

A face recognition pipeline conventionally consists of four stages: detection, alignment, representation and classification. DeepFace (Taigman *et al.*, 2014) improves the alignment step by using an explicit 3D modeling, and employs a generalized face representation learned from the larger Social Face Classification (SFC) dataset which includes 4.4 million labeled faces. With the above improvements, DeepFace acquired an accuracy of 97.35% on the Labeled Faces in the Wild (LFW) dataset.

In VGGFace2 (Cao *et al.*, 2018), ResNet-50 (with and without Squeeze-and-Excitation (Hu *et al.*, 2018) (SE) blocks) are trained and evaluated. A Squeeze-and-Excitation block captures channel-wise statistics using average pooling, then learns channel-wise dependencies though sigmoid and ReLU (Nair and Hinton, 2010) activation. The resulted vector multiplies and scales the input block to introduce dynamics conditioned on the input. When the false positive rate (FAR) equals to 0.001, ResNet-50 with the SE block achieves 90.4% on the IJB-A dataset versus 89.5% without the SE block; on the IJB-B dataset, ResNet-50 with the SE block is 1% better as well with 88.8% versus 87.8%.

1.1.4 Machine Translation

Cho et al (Cho *et al.*, 2014) first proposed the RNN seq2seq model by modeling it as an RNN encoder-decoder architecture, with the encoder transforming an input sentence into a context vector and the decoder mapping the context vector to an output sentence (the translation hypotheses). Bahdanau et al. (Bahdanau *et al.*, 2014)

further improved RNN seq2seq models by making the encoder as a bidirectional gate recurrent unit (GRU) and binding the attentional mechanism to the decoder. To avoid overfitting in RNN seq2seq models, Gal and Ghahramani (Gal and Ghahramani, 2016) proposed to apply the variational inference based dropout technique to the model. To speed up convergence in training, Ba and his colleagues (Ba *et al.*, 2016) introduced layer normalization to stabilize state dynamics in RNNs. Salimans and Kingma (Salimans and Kingma, 2016) proposed weight normalization that reparameterizes weight vectors from their direction. To increase the model depth, Zhou et al. (Zhou *et al.*, 2016) proposed fast-forward connections where the shortest paths do not depend on any recurrent calculations. Wu et al. (Wu *et al.*, 2016b) introduced the bidirectional stacked encoder and Barone and his colleague (Barone *et al.*, 2017) proposed a BiDeep RNN by replacing the GRU cells of a stacked encoder with multi-layer transition cells. Deviating from RNN seq2seq models, Gehring et al. (Gehring *et al.*, 2017) proposed convolutional seq2seq model where the encoder and decoder were fully replaced by convolutional neural networks (CNN). Their approach allows much faster training while retaining the BLEU scores closely comparable to those obtained with RNN seq2seq models.

Sennrich and Haddow (Sennrich and Haddow, 2016) generalized the embedding layer to support linguistic features such as morphological features, part-of-speech tags, and syntactic dependency labels. Press and Wolf (Press and Wolf, 2016) proposed tier embedding and argued that weight tying reduces the size of neural translation models. However, no attempt has been made to reduce the size of the target dictionary through word embedding. Significant reduction in model complexity is expected considering words are on the order of hundreds of thousands or more in a typical dictionary.

Sennrich et al. (Sennrich *et al.*, 2015) proposed to segment words of the source and target sentences into smaller subword units using byte pair encoding (BPE) compression (Gage, 1994). They showed an improvement in the BLEU scores of 1.1 and 1.3 for English-German and English-Russian translations, respectively. Despite its advantage, BPE splits an alphabetic word to multiple-letter groups, and thus it is intrinsically not applicable to logographic languages such as Chinese, Chorti, and Demotic (Ancient Egyptian) where a word is a glyph rather than alphabetic letters. García-Martínez et al. (García-Martínez *et al.*, 2016) proposed to decompose words morphologically and grammatically into factored representations such as lemmas, part-of-speech tag, tense, person, gender, and number. Their approach reduced training time and out of vocabulary (OOV) rates with improved translation performance, but also introduces unnecessary grammatical dependencies, (e.g. there are hundreds of tenseless languages), and is not optimized in all scenarios.

1.2 Edge Artificial Intelligence

A new computing paradigm, edge artificial intelligence (EI), has emerged from the demand of AI (e.g. deep learning algorithms) on the edge. There are various approaches to EI, including but not limited to model partitioning, caching, model compression, and hardware acceleration.

1.2.1 Model Partitioning

Model partitioning is a method that splits the computation flow of a task into parts that can be executed on different computational devices. It is an attractive solution

to accelerating deep learning inference, when trade-offs exist between incurring too much communication overhead to offload all computation to the cloud and too much computation time if model prediction is left on local devices entirely.

Neurosurgeon (Kang *et al.*, 2017), a partitioning algorithm, was introduced to automatically partition the DNN computation between mobile devices and data center at the granularity of neural network layers. It was examined that there existed a best partition point, which improved both latency and energy consumptions. Based on this observation, a prediction model was proposed in the Neurosurgeon to dynamically select the best partition point in different DNN topology. Although this approach is empirically demonstrated, the limitation is also obvious: the prediction model relies on a network connection. In the absence of that, the prediction cannot be performed between the cloud and the mobile devices, therefore, the scope of application is restricted.

Distributed Deep Neural Networks (DDNNs) (Teerapittayanon *et al.*, 2017) was designed to perform fast and localized inference using shallow portions of the neural network at the end devices. Using an exit point after device inference, the output would be classified before sending back to the cloud. When multiple end devices presented, an aggregation method would gather all output from each end device in order to perform classification, where an exit was determined. If the classification could not be made when the sample was not confident, it would escalate to a higher exit point (e.g. the edge exit) in the hierarchy until the last exit (the cloud exit) which always performed classification. By allowing multiple exit points, DDNNs could reduce the communication cost significantly. However, when the exit point was made, the neural network was trained based on the collected sample. Any new data from

the end devices will not be learned by the trained neural network furthermore. Lack of transferable learning ability will be difficult for DDNNs to adapt to new local environments and changing circumstances.

Ko et. al (Ko *et al.*, 2018) proposed the edge-host partitioning of convolutional neural networks. The number of operations, memory access and feature size per layer are carefully measured. Furthermore, the correlation between energy consumption and throughput were thoroughly tested for various wireless protocol including wifi, Bluetooth, and Bluetooth low energy (BLE). The novelty of this work is that it evaluated both the lossless and lossy compression methods for encoding the output feature space being transmitted to the cloud. However, although AlexNet, VGG, and ResNet are measured, there were not experiments and discussions showing that their approach can be generalized to more complex (state-of-the-art) neural networks such as neural networks with shake-shake regularization, implicit generative models and neural networks with attention.

1.2.2 Model Caching

Model caching exploits and reuses cachable components or partitions of a deep learning model. DeepCache (Xu *et al.*, 2018) is a model caching approach for deep learning inference in continuous mobile vision. It using diamond search (Xu *et al.*, 2018) breaks down an input video frame into smaller blocks and discovers similar blocks between consecutive frames. Video temporal locality is discovered and exploited by DeepCache. Regions of reusable results are propagated in DeepCache by exploiting the model's internal structure.

Guo et al. proposed FoggyCache Guo *et al.* (2018) for cross-device approximate

computation reuse. Previously computed outputs are reused in FoggyCache by harnessing the "equivalence" between different input values. Adaptive locality sensitive hashing (A-LSH) and homogenized k-nearest neighbors (H-kNN) were introduced to achieve constant lookup, high-quality reuse, and tunable accuracy guarantee.

1.2.3 Model Compression

Model compression shortens inference time through computation reduction on a deep learning model. There are two groups of compression methods that are most relevant. They are quantization and tensor decomposition.

Quantization

Quantization methods reduce network sizes by lowering weight precision (e.g. from 32-bit floating points to 8-bit integers) with minimal loss of accuracies. Fixed point quantization, vector quantization, and product quantization are the most-studied quantization methods in compression.

Motivated by the significant reduction in model inference cost, Wu et al. (Wu *et al.*, 2016b) posed additional constraints to deep LSTM stacks during training, so that the resulting model is quantizable at inference with minimal loss of accuracy. Fixed point quantization was applied to replace floating point operations with 8-bit or 16-bit integer operations.

There are other earlier studies on CNN model quantization such as fixed-point quantization (Courbariaux *et al.*, 2014; Gupta *et al.*, 2015; Lin *et al.*, 2016a) and vector quantization (Gong *et al.*, 2014). For compressing word embedding, Suzuki, and Nagata (Suzuki and Nagata, 2016) applied block-wise k-means post-processing to

SkipGram with negative sampling (SGNS).

Tensor Decomposition

Tensor Decomposition methods reduce network size by replacing large weight matrix (tensor) with multiple much smaller matrixes (tensors), considering neural networks are in fact composed of weight matrixes (tensors)

Low-rank matrix factorization reduces the number of parameters by replacing a weight matrix with the product of multiple lower rank matrixes. Low-rank technique (Sainath *et al.*, 2013) was first applied to the final weight layer of deep neural networks considering the majority of parameters are in the final weight layer for both acoustic modeling and language modeling. Their experiments demonstrate that a low-rank factorization reduces the number of parameters of the network by $30 - 50\%$ without a significant loss in accuracy.

Tucker decomposition is also known as a higher-order principal component analysis (PCA). Tucker decomposition decomposes a tensor into a core tensor and multiple factor matrices (principal components) by minimizing the difference between the original tensor and the reconstructed one from the core tensor and principal components. Tucker-tensor-train-model-compression (T3MC) (Chen *et al.*, 2017) and one-shot whole network compression (Kim *et al.*, 2015) are two successful Tucker decomposition applications to deep model compression.

Tensor train (TT) decomposition decomposes a tensor to a train of much smaller tensors. For the Very Deep VGG networks (Novikov *et al.*, 2015), it is reported that TT decomposition reduces the size of the fully-connected layer up to 200000 times achieving the compression factor of the whole network up to 7 times.

Knowledge Distillation

Knowledge distillation methods reduce the model sizes by distilling knowledge from a larger teacher network or an ensemble of teacher networks into a smaller student network. Hinton et. al. (Hinton *et al.*, 2015) proposed to distill the knowledge by adding a temperature term to the final softmax function. They reported that the distilled single model performs only slightly worse than the average predictions of 10 models in an ensemble. Sau et. al. (Sau and Balasubramanian, 2016) proposed to distill knowledge from noisy teachers. Instead of learning from an ensemble of teachers, they proposed to simulate the effect of an ensemble by injecting noise and perturbing the outputs of a teacher. Codistillation, introduced by Anil et. al., restored the symmetry of Hinton et. al.'s approach. In the codistillation's architecture, a neural network can be either the teacher of other students and the student of other teachers at the same time (Anil *et al.*, 2018).

1.2.4 Hardware Acceleration

Graphics processing units (GPU) were first used in hardware acceleration of deep learning models (deep neural networks). GPU hardware such as Nvidia P100, GTX 1080ti, and Jetson TX1/TX2 are widely used among data centers, edge nodes and end devices. Tensor processing units (TPU), which are application-specific integrated circuits (ASIC), were developed specifically for acceleration of neural network inference in data centers. With an 8-bit matrix multiplication unit, TPU offers a peak throughput of 92 TeraOps per second. (Jouppi *et al.*, 2017) Edge TPUs, on the other hand, designed for Internet of thing (IoT) devices, enable high-performance neural network inference on the edge. Field-programmable gate array (FPGA), offering superior

energy efficiency and irregular parallelism, is a great fit for hardware acceleration with extreme customizability. (Nurvitadhi *et al.*, 2017)

Software Libraries on Hardware: DeepX (Lane *et al.*, 2016) significantly reduces the latency of fully-connected layers by decomposing deep neural networks into unit-blocks which are more efficiently run by various mobile processors such as GPUs and CPUs. DeepMon (Huynh *et al.*, 2017) efficiently offloads convolutional layers to mobile GPUs with tests on phones with Adreno and Mali GPUs. TensorFlow (Abadi *et al.*, 2016), operating on heterogeneous embedded devices such as NVIDIA Jetson TX2 and Raspberry Pi, represents neural networks as the data flow graphs and maps nodes of the graphs to heterogeneous mobile processors.

1.3 Main Contributions and Thesis Organization

The thesis contributes to advancing the state-of-the-art EI techniques in devising efficient models with small sizes and low computation complexity. The motivating applications, the models and key achievements are highlighted in Figure 1.1.

For model partitioning, we propose TeamNet, which trains multiple models that can be executed in parallel on distributed devices on the edge. TeamNet is a federation of experts, each specializing in a particular subset of knowledge associated with target tasks. Given that knowledge representation is in itself an active area of research, it is non-trivial to qualify the amount of knowledge of a task and partition it equally among expert models. Furthermore, one important question is to determine which expert model has the expected knowledge partition (and thus can be trusted for a specific input data). This essentially implies that *a model needs to know what*

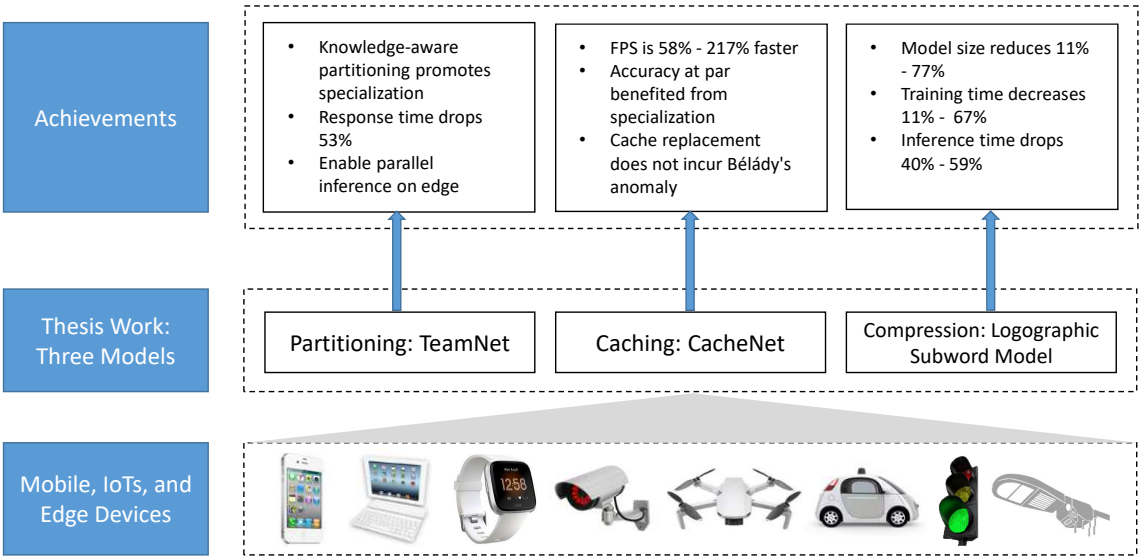


Figure 1.1: TeamNet, CacheNet, and Logographic Subword Model are applicable to various connected mobile, IoT, and edge devices, such as smartphones, smart cameras, drones, self-driving vehicles, and smart roadway lights.

it does not know. TeamNet addresses this problem by measuring the predictive uncertainty of individual expert models. We develop a unique training procedure that enables knowledge specialization among experts. Experiments show that by promoting specialization, TeamNet achieves even slightly better predictive accuracy than baseline models. Furthermore, TeamNet can significantly shorten inference time by as large as 57%.

In analogy to caching in memory hierarchy, we introduce CacheNet, which caches a portion of a neural network model for unstructured data such as images and audio signals on edge or end devices. Similar to TeamNet, CacheNet builds upon the division of a neural network into multiple smaller specialized partitions during training. Unlike TeamNet, the right partition (model) for caching is selected and cached during inference. To do so, CacheNet encodes high-dimensional unstructured data into low

dimension vector representations. The low dimension representation becomes the hint to select a particular partition to carry out inference on a particular unstructured data input. Due to temporal locality that naturally occur in streaming data, such a selection tends to be valid for subsequent data over a period of time.

For model compression, we propose a novel language model, namely, logographic subword model, which significantly reduces the number of classes in the final prediction. In language modeling and acoustic modeling, there exist a large number of output classes. They contribute to higher model complexity in the number of weights and biases in the output layer. ASM reduces the number of output classes by mapping a single word or phrase to multiple subwords, which can be shared among words and phrases. As experiments demonstrate, model sizes can be reduced by up to 77% with inference time improved by up to 59%.

The rest of the thesis is organized as follows:

- **Chapter 2:** The partitioning framework TeamNet is described followed by the evaluation in a distributed network environment.

- **Chapter 3:** The caching framework CacheNet is elaborated and in-depth experiments are given with two benchmark datasets (including CIFAR-10 and FVG

- **Chapter 4:** The logographic subword model in compression is elaborated followed by thorough evaluation in machine translation. (Zhang *et al.*, 2019)) and on four typical edge devices (including Jetson TX2, Jetson Nano, Raspberry Pi, and the Android smartphone).

- **Chapter 5:** The conclusion to the thesis and the future research are discussed.

Chapter 2

TeamNet: Knowledge-Aware Partitioning for Collaborative Inference

This chapter is reproduced from "TeamNet: A Collaborative Inference Framework on the Edge", Yihao Fang, Ziyi Jin, and Rong Zheng, published in IEEE International Conference on Distributed Computing Systems (ICDCS), Dallas, Texas, USA, 2019. The author of this thesis is the first author and the main contributor of this publication.

Abstract

With significant increases in wireless link capacity, edge devices are more connected than ever, which makes possible forming artificial neural network (ANN) federations on the connected edge devices. Partition is the key to the success of distributed ANN inference while unsolved because of the unclear knowledge representation in most of the ANN models. We propose a novel partition approach (TeamNet) based on the psychologically-plausible competitive and selective learning schemes while evaluating its performance carefully with thorough comparisons to other existing distributed machine learning approaches. Our experiments demonstrate that TeamNet with sockets and transmission control protocol (TCP) significantly outperforms sophisticated message passing interface (MPI) approaches and the state-of-the-art mixture of experts (MoE) approaches. The response time of ANN inference is shortened by as much as 53% without compromising predictive accuracy. TeamNet is promising for having distributed ANN inference on connected edge devices and forming edge intelligence for future applications.

2.1 Introduction

Recent advances in the Internet of Things and wireless communications witness the arrival of a new computing paradigm, in which computation and data storage is distributed among multiple end-user devices and near-user edge devices. Thousands of distributed smart edge devices such as smart cameras and smart routers are connected seamlessly through protocols such as 5G, LTE, and WiFi, enabling distributed intelligence.

Deep neural networks have been successfully applied to many domains such as computer vision and natural language processing. In a large-scale image recognition task, deep neural networks normally consume tens of gigabytes in RAM and significant computing power on GPUs and CPUs. Their resource consumption can be accommodated by a cloud environment, but typically cannot be handled by individual edge devices due to their limited processing power and memory. Take the processing power as an example. There are 256 NVIDIA CUDA cores on a Jetson TX2 board. In contrast, the Helios cluster hosted by Compute Canada, a consortium of high-performance computing data centers, has 120 NVIDIA K20 GPUs (with 2496 CUDA cores each) and 96 K80 GPUs (with 4992 CUDA cores each) – roughly 3042 times more cores than a Jetson TX2 board.

Shallow neural networks alone generally have poorer predictive accuracy compared to state-of-the-art (SOTA) deep neural networks, but they allow real-time prediction on the edge devices, which make them more suitable for edge computing. A question arises that *if it is possible to train and coordinate the inference of multiple shallow neural networks (each running on an edge device) to have comparable performance as or even outperform a single SOTA deep neural network in prediction.*

In this work, we propose a novel partition approach called *TeamNet* based on competitive and selective learning. TeamNet is inspired by phenomena in human society, where knowledge is naturally partitioned among people with each individual specializing in only one or a few subject domains. Analogous to the human society, shallow neural networks in TeamNet do not learn the knowledge of the entire dataset, instead, they only master one subset of it. During inference, knowing what they do know and what they know by estimating predictive uncertainties, TeamNet devices

running shallow models aggregate their predictions to form final decisions distributively and collaboratively. Unlike existing computation partition approaches (Kang *et al.*, 2017; Teerapittayanon *et al.*, 2016) that decide where computation should be done for *pre-trained* deep neural network models, TeamNet is a fundamentally different approach to partition by training shallower models using the *similar but downsized architecture* of a given SOTA deep model. Both the number of shallower models and the SOTA deep model can be specified by users.

We have implemented TeamNet using TensorFlow and CUDA on two types of representative edge devices, i.e., Jetson TX2 and Raspberry Pi 3 Model B+. For comparison, we have also parallelized baseline models using the message passing interface (MPI) and implemented the SOTA mixture of experts (MoE) approaches. Experiments using MNIST and CIFAR-10 datasets show that TeamNet can shorten inference latency by as much as 53% compared to baseline neural networks without compromising the predictive accuracy. Therefore, TeamNet is promising for enabling distributed edge intelligence for compute-intensive applications.

The rest of the chapter is organized as follows. Section 2.2 describes the related work. System architecture is presented in Section 2.3. Section 2.4 provides more details of TeamNet's training algorithms. Section 2.5 described TeamNet's inference. In Section 2.6, we present implementation details, experimental setups and results. Conclusions and future work are given in Section 2.7.

2.2 Related Work

In this section, we provide an overview of related work. There are two groups of work that are relevant, namely, computation partition and the mixture of experts.

Computation Partition of Neural Networks Computation partition splits a known model represented as a data flow graph into two or more parts and executes them on edge and in the cloud (Kang *et al.*, 2017). For deep forward neural networks, the partition is generally performed between layers, which trade-offs extra data transfer time and less total computation time (due to offloading). Recently, two interesting variants of computation partition have been proposed. In (Ko *et al.*, 2018), Ko et al. considered lossless and lossy compression of the output features of an intermediate layer before transmitting them to the cloud. In (Teerapittayanon *et al.*, 2017), Distributed Deep Neural Networks (DDNNs) was designed to perform fast and localized inference using shallow portions of a neural network on edge devices. Using an exit point after device inference, an output is classified locally. When multiple end devices presented, an aggregation method would gather all output from each end device in order to perform classification, where an exit was determined. If the classification could not be made due to low confidence, the task is escalated to a higher exit point (e.g. the edge exit) in the hierarchy until the last exit (the cloud exit). With multiple exit points, DDNNs can significantly reduce communication costs.

Computation partition approaches take a trained DNN model and divide the pipeline among devices with different processing capabilities. Therefore, how partitions can be made is inherently constrained by the structure of the existing model. In contrast, in TeamNet, we train smaller and specialized expert models.

Mixture of Experts (MoE) Adaptive Mixtures of Local Experts (Jacobs *et al.*, 1991) was proposed to combine multiple feed-forward network experts with an adaptive gating network (also feed-forward network). In this architecture, all experts receive the same input and have the same number of outputs. The gating network receives

the same input as the expert networks' and outputs a stochastic switch with the probabilities (gate values) that the switch uses to select the outputs from experts. The mixture of experts is partitionable in nature. It is feasible to deploy experts and the gating network respectively to multiple edge devices.

Sparsely-Gated Mixture-of-Experts (SG-MoE) (Shazeer *et al.*, 2017) was introduced to increase model capacity through a sparse combination of multiple neural network experts. Different from Jacobs's approach (Jacobs *et al.*, 1991), the gating network uses noisy top-K gating that keeps only the top k gate values and set the rest gate values to zero.

Since MoE has been introduced more than two decades ago, it has been the topic of much research. Different MoE architectures have been proposed such as a hierarchical structure (Jordan and Jacobs, 1994; Bishop and Svenskn, 2002; Yao *et al.*, 2009), infinite number of experts (Rasmussen and Ghahramani, 2002), and sequential increment of experts (Aljundi *et al.*, 2017). However, a majority of the approaches target specific kinds of expert models such as Gaussian, GP, SVM, etc and thus are not architecture independent. They fail to capture recent advances in DNN models. SG-MoE can work with different NN architecture but its training process is not optimized to have experts of comparable capacity. As will be demonstrated through experiments in Section 2.6, this leads to degraded inference performance.

2.3 System Architecture

During the training stage, TeamNet takes a neural network architecture, the number of experts K, and training data as input and produce K expert models by semantically partitioning the dataset's knowledge as illustrated in Figure 2.1. The number of

parameters of the expert models is hyperparameters that can be tuned using grid search or other automated machine learning approaches. TeamNet can be thought of as a black box from the developers' point of view. As an example, we input to TeamNet the convolutional neural network (CNN) architecture with 26 hidden layers and then ask TeamNet to generate 4 expert models according to the CIFAR-10 dataset. The outputs of TeamNet are 4 CNN models each with 8 layers, and those models could collaborate with one another on connected edge devices.

At run-time, TeamNet expert models are deployed on edge devices connected through a wireless network. For ease of presentation, we assume each edge device only runs one model and each device has its own sensor input (e.g., visual or audio data). As illustrated in Figure 2.1, in Step 1, upon a sensing event, an edge device broadcasts the sensor data to all peer edge devices in the network (Step 2). In Step 3, all edge devices will execute their own local models in parallel and each produces an uncertainty measure. Finally, in Step 4 and 5, all the results are being gathered and the output with the least uncertainty will be taken as the final result. This last step can be done distributedly, e.g., using a leader election protocol, or done centrally by sending the results along with the uncertainty measures to a designated device.

2.4 Training TeamNet

In this section, we present the details of the algorithm to train TeamNet. The key challenge is the "richer gets richer" phenomenon. Specifically, regardless of how the expert models are initialized, they are predisposed to *biases* – some expert is confident about more input data than the others. Without any correction to the initial bias, the expert that is confident about more input data will be trained with more data

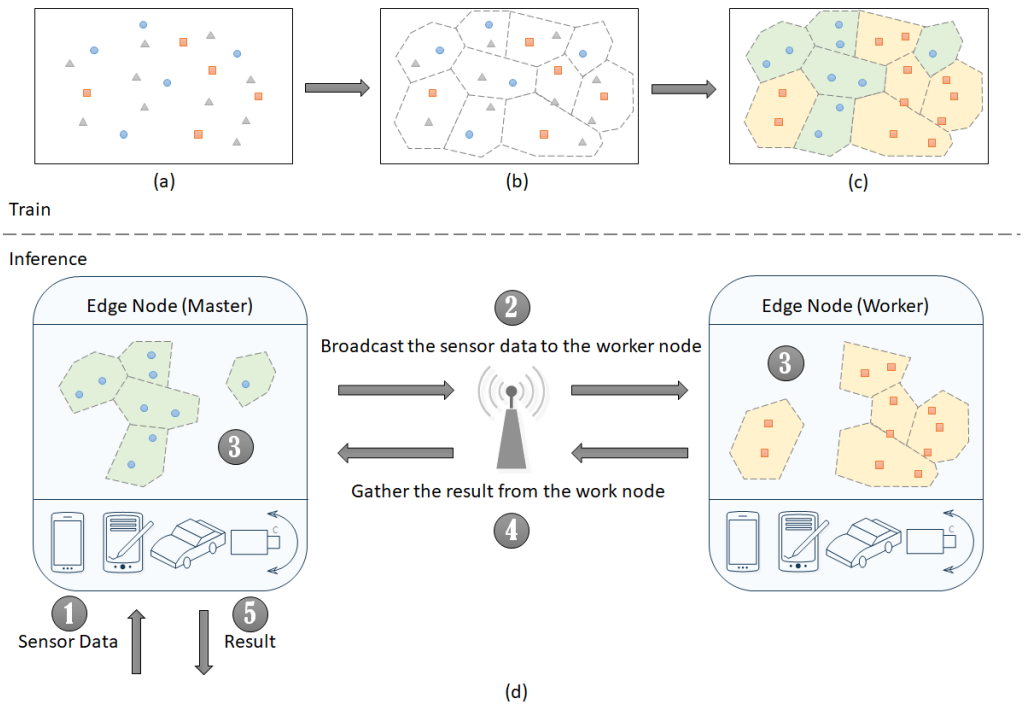


Figure 2.1: TeamNet's training and inference are illustrated on two edge nodes: (a) Initially, each expert has very limited knowledge about the dataset but each is randomly more certain of some data than the others. (e.g. One expert is more certain of the square data points, while the other expert is more certain of the circle data points. However, neither of them has knowledge of the triangle data points.) (b) With this preference, it is more likely for each expert to select the more certain data to learn, respectively. (c) Gradually, with the help of gradient descent, each expert has learned its more certain data from the entire dataset. (d) At the inference stage, each expert is deployed on one edge node, and performs inference concurrently.

leading to skewed partitions of the training data. In the worst case, some experts are subject to little training data and learn nothing.

2.4.1 Overview

Consider K experts, each modeled as a function $f(x;\theta_i)$, parameterized by θ_i, $i = 1, 2, \ldots, K$. For multi-class classification problem with C categories, $f(x;\theta_i)$ can be

viewed as the parameters of a multinomial distribution of C classes. Let $p(\hat{y} = c|x, \theta_i)$ be the predictive probability of output $c = 1, 2, \ldots, C$ for input $x \in X$ from Expert i. The predictive entropy of Expert i of input x is defined as,

$$H(\hat{y}|x, \theta_i) := -\sum_c p(\hat{y} = c|x, \theta_i) \log p(\hat{y} = c|x, \theta_i).$$

Predictive entropy is closely related to perplexity. The predictive entropy of a model reflects its "uncertainty" of a data instance drawn from the same distribution of the training data.

The primary objective of TeamNet training is to divide the input data $\mathcal{D}$ into K partitions $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_K$ such that for any $(x, y) \in \mathcal{D}_i$, Expert i can predict the correct label y and has the least predictive entropy for x among all experts with high likelihood. The secondary objective is to have $|\mathcal{D}_i| \approx \frac{|\mathcal{D}|}{K}, i = 1, 2, \ldots, K$, where $|\mathcal{D}|$ is the cardinality of set $\mathcal{D}$. Thus, TeamNet is both *localized* and *implicit*. *Localized* means that each expert specializes in a subset of the data, and *implicit* means that we do not partition the data explicitly but let the experts *compete* with one another to divide them (roughly equally). The rationale behind equal partitions of training data is to have experts of similar capacity. Otherwise, some experts may be under-fitted while others are over-fitted. One may argue equally divided training data does not imply equally partitioned models. This is indeed true especially in the case of unbalanced training data and will be considered in our future work. In this work, we assume the training data is balanced.

The neural network used to train the experts is given in Figure 2.2. All expert networks are initialized with random weights. Training is done in multiple epochs. In each epoch, the training data is first reshuffled and then divided into equal-sized

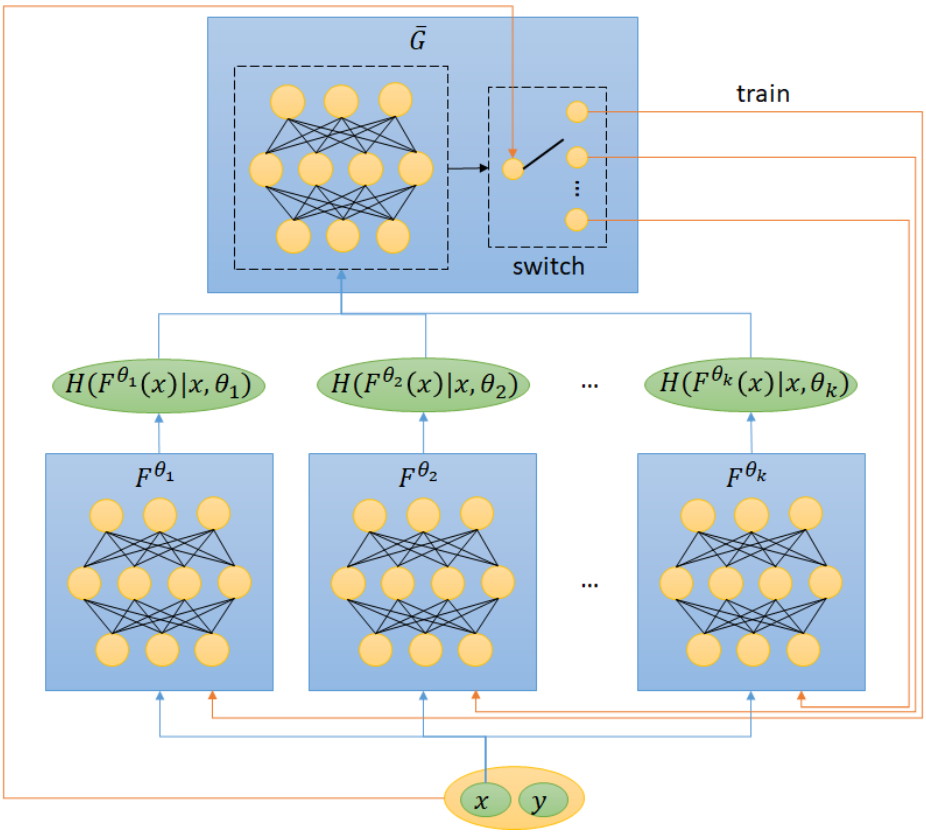


Figure 2.2: At the training stage, the gate $\bar{G}$ decides which data example should be assigned to which expert to learn. Such a decision is based on the uncertainty estimation of all experts for the particular data example.

batches. Each batch of data is given to all experts to evaluate their respective predictive entropy values. The calculated predictive entropy values serve as inputs to the gate network $\bar{G}$, which are then trained to assign data samples in the batch to different experts. After the assignment is done, the weights of the expert networks are updated using back-propagation with their own partitions. The procedure is summarized in Algorithm 1.

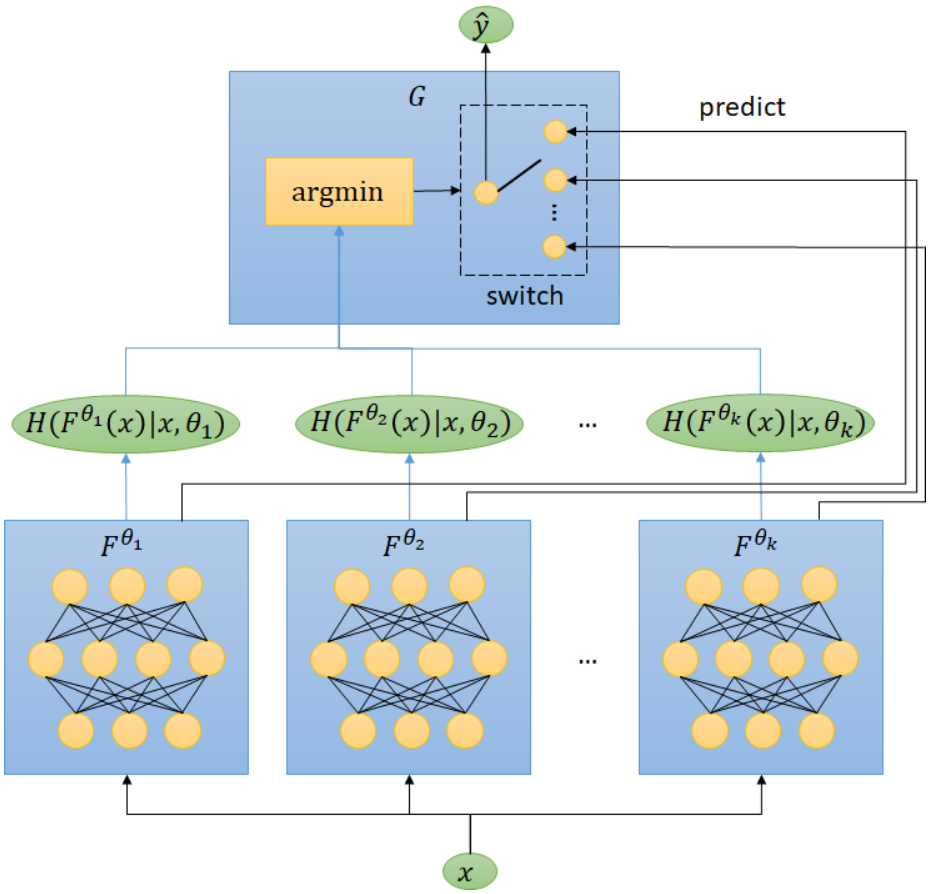


Figure 2.3: At the inference stage, it is sufficient to have the arg min function to determine which expert is least uncertain of the given data input. TeamNet's inference time is generally faster than other mixture of experts (MoE) approaches since its gate is much simpler.

2.4.2 Dynamic Gating

The gate function $\bar{G}$ splits the current batch into K partitions for training. When experts are biased, direct application of the arg min gate would result in unevenly split data. As the experts become less biased, $\bar{G}$ should approach an arg min gate, where the training sample is given to the model with the least uncertainty. Therefore, it is important to update $\bar{G}$ dynamically based on some measure of the biases of the

Algorithm 1 Training TeamNet

▷ Let r be the number of epochs
procedure TRAIN($\theta, \eta, \epsilon, K, r$)
    Shuffle the dataset
    Repeat the dataset for r times
    Get the first batch β
    while $|\beta| > 0$ **do**
        $\mathbf{H} \leftarrow H(\hat{y}|x, \theta_i), \forall i \in \{1...K\}, \forall x \in \beta$
        $\bar{G}^{\beta} \leftarrow$ GATE_TRAIN($\beta, \mathbf{H}, \eta, \epsilon, K$)
        EXPERT_TRAIN ($\beta, \bar{G}^{\beta}, \theta, \eta, K$)
        Get the next batch β
    end while
end procedure

experts.

We define $\bar{G}$ as:

$$\bar{G}(x, \delta) := \arg\min_i \delta_i \cdot H(\hat{y}|x, \theta_i), \tag{2.4.1}$$

where $\delta = (\delta_1, \delta_2, \ldots, \delta_K)$ are control variables to be determined. Let $G(x) := \arg\min_i H(\hat{y}|x, \theta_i)$ denote the assignment of the arg min gate for input x.

Given the current batch β and δ, let

$$\gamma_i = \frac{\sum_{x \in \beta} \mathbb{1}_{G(x)=i}}{|\beta|}, \tag{2.4.2}$$

and

$$\bar{\gamma}_i(\delta) = \frac{\sum_{x \in \beta} \mathbb{1}_{\bar{G}(x,\delta)=i}}{|\beta|}, i = 1, 2, \ldots, K, \tag{2.4.3}$$

where $\mathbb{1}$ denotes the Kronecker delta function. In other words, γ_i and $\bar{\gamma}_i(\delta)$ are the portion of data in batch β that are assigned to Expert i according to gate G and $\bar{G}$, respectively. Under the assumption that the data in β is balanced, the bias in Expert i can then be characterized by $\gamma_i - \frac{1}{K}$.

Clearly, $\gamma_i - \frac{1}{K} \equiv 0$ for all experts implies the batch are divided equally by the expert models trained thus far. Otherwise, by adjusting δ_i's, we can "correct" the biases by assigning more data to the expert who would have received less training data according to gate G. Thus, we formulate the following optimization problem,

$$\min_{\delta} \sum_{i=1}^{K} \left| \bar{\gamma}_i(\delta) - \left(\frac{1}{K} - a \cdot (\gamma_i - \frac{1}{K}) \right) \right|, \tag{2.4.4}$$

where $0 < a < 1$ is a hyperparameter analogous to the gain of a proportional controller (Figure 2.4).

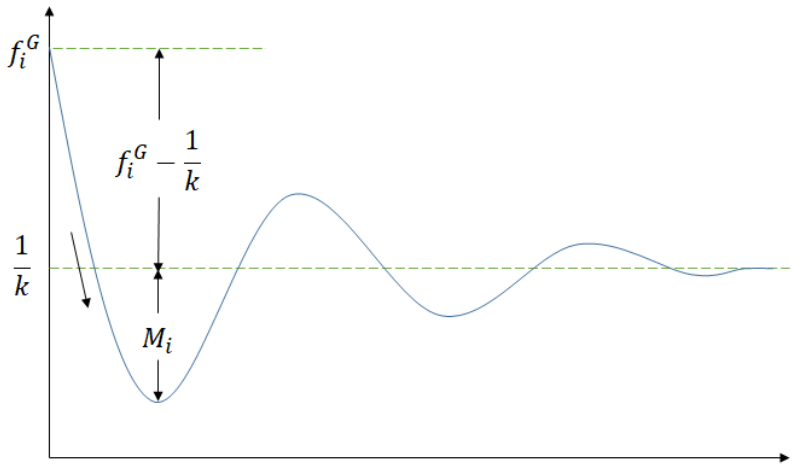


Figure 2.4: The term γ_i is defined to reflect the bias from the past. Correcting it induces a momentum (momentum-caused displacement) $a \cdot (\gamma_i - \frac{1}{K})$. The momentum counteracts the bias which has been incurred by the past, and brings γ_i to the mean (set point) $\frac{1}{K}$ eventually.

The optimization in (2.4.4) cannot be solved directly since there is no closed-form expression for $\bar{\gamma}_i$'s for arbitrary K. Instead, we represent the possible values of δ parametrically.

Let $E(x) = \frac{1}{K} \sum_{i=1}^{K} H(\hat{y}|x, \theta_i)$ be the mean entropy and $D(x) = \frac{1}{K} \sum_{i=1}^{K} |H(\hat{y}|x, \theta_i) - E(x)|$ be the absolute deviation of the entropy of instance x from K experts. We

introduce Δ, the average normalized absolute deviation of batch β as

$$\Delta = \frac{1}{|\beta|} \sum_{x \in \beta} \frac{D(x)}{E(x)}.$$

In other words, Δ measures how “diverse" the uncertainty of different expert models is.

Next, we represent Δ by

$$\delta = 1 + \Delta \cdot W(z, \Theta),$$

where z is a vector of length N randomly drawn from a uniform distribution $\mathcal{U}(-1, 1)$ and W is a multilayer perceptron (MLP) function parametrized by Θ. Essentially, we transform the problem of solving for δ into estimating parameters Θ. Doing so allows us to evaluate the gradients of Θ with respect to the objective function in (2.4.4) by approximating $\arg\min$ in $\bar{G}$ with a continuous function. The gradients are then used to update Θ until convergence.

To this end, we summarize the procedure to find $\bar{G}$ for batch β in Algorithm 2. The proof sketch of its convergence is given in Appendix 2.8.

Soft Argument of the Minimum The $\arg\min$ function in (2.4.1) is not differentiable. Gradients cannot be propagated back to Θ. In order to calculate the partial derivatives with respect to Θ, the $\arg\min$ function must be softened (Chapelle and Wu, 2010), defined by:

$$\text{soft}\arg\min(x) = \sum_i \frac{e^{-bx_i}}{\sum_j e^{-bx_j}} i \tag{2.4.5}$$

Algorithm 2 Finding Gate $\bar{G}$

▷ Let β be a mini-batch with size n
▷ Let $\mathbf{H}$ be the matrix of $H(\hat{y}|x, \theta_i)$ for all i in $1...K$ for all x in β
▷ Let Δ be the relative mean absolute derivation of $\mathbf{H}$
▷ Let $\bar{G}^{\beta}$ be the vector of $\bar{G}(x, \delta)$ for all x in β

procedure GATE_TRAIN($\beta, \mathbf{H}, \eta, \epsilon, K$)
  Calculate Δ for β
  Create $z \sim \mathcal{U}(-1, 1)$ ▷ z is a latent variable
  Calculate γ_i for all i in $1...K$
  while $J > \epsilon$ **do**
    $\Phi \leftarrow W(z, \Theta)$ ▷ W transforms z into Φ
    $\delta \leftarrow 1 + \Phi \cdot \Delta$
    $\bar{G}^{\beta} \leftarrow \arg\min \delta \odot \mathbf{H}$
    Calculate $\bar{\gamma}_i(\delta)$ for all i in $1...K$
    $J \leftarrow \frac{1}{K} \sum_{i=1}^{K} \left| \bar{\gamma}_i(\delta) - \left(\frac{1}{K} - a \cdot (\gamma_i - \frac{1}{K}) \right) \right|$
    $\Theta \leftarrow \Theta - \eta \nabla_{\Theta} J$ ▷ Θ descends w.r.t. gradients
  end while
  return $\bar{G}^{\beta}$
end procedure

Meta-Estimator An over-large b in (2.4.5) leads to an over-steep slope in gradient descent, while an over-small b leads to an over-gentle one. Neither benefits gradient propagation. To solve this problem, the scalar b is found by optimizing a meta-estimator (neural network) with the objective that minimizes the distance between a small ϵ and the expectation of $\bar{G}(x, \delta)$ to its closest integer. The introduction of ϵ avoids from happening of an over-steep slope and smoothens the loss surface. Formally, the objective function is written as follows:

$$\min_{b} \left| \frac{\sum_{x \in \beta} \min_{i=1,2,...,K} \left| \bar{G}(x, \delta) - i \right|}{|\beta|} - \epsilon \right| \tag{2.4.6}$$

Kronecker Delta's Approximation The Kronecker delta function $\mathbb{1}$ in (2.4.3) is not differentiable. Thus, instead, $\mathbb{1}$ is replaced by a differentiable approximation,

denoted by:

$$\mathbb{1}_{\bar{G}(x,\delta)=i} \approx \tanh\left(c \cdot \mathrm{ReLU}(0.5 - |\bar{G}(x,\delta) - i|)\right), \quad i = 1, 2, \ldots, K, \tag{2.4.7}$$

where the rectified linear unit (ReLU) (Nair and Hinton, 2010) is given by $\mathrm{ReLU}(x) = \max(x, 0)$, and the hyperbolic tangent function is defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$.

The differentiable approximation in (2.4.7) is the combination of shifting ($\bar{G}(x,\delta) - i$), ramping (ReLU), and discretization (tanh). In our experiments in Section 2.6, the constant c is set to 10 to satisfy the needs of discretization while letting gradients propagate through. The constant 0.5 is added to approximate rounding.

2.4.3 Expert Trainer

The training process of experts is guided by gate $\bar{G}$. The gate decides which expert to learn a particular data example in each batch. A mini-batch β is then partitioned into K subsets $\beta_1, \beta_2, ..., \beta_K$. Gradients of Expert i (i in $\{1, ..., K\}$) are calculated with respect to the loss from the data examples in batch β_i. The parameters of Expert i (noted by θ_i) are then updated by subtracting the product of the learning rate η and the normalized gradients of the batch (batch normalization) (Ioffe and Szegedy, 2015). Details of the process are given in Algorithm 3.

At Line 4, the formula $\sum_c y \log f(x; \theta_i)$ gives the cross entropy loss (objective) for optimization. Only batch β_i is used to update the parameters of Expert i. No expert learns from all data examples in β.

Algorithm 3 Training Experts

▷ Let β be a mini-batch with size n
▷ Let $\bar{G}^{\beta}$ be the vector of $\bar{G}(x, \delta)$ for all x in β
procedure EXPERT_TRAIN($\beta, \bar{G}^{\beta}, \theta, \eta, K$)
  $\beta_1, \beta_2, ..., \beta_K \leftarrow$ group β by $\bar{G}^{\beta}$
  for i **in** $1...K$ **do** ▷ in parallel
    $\theta_i \leftarrow \theta_i - \eta \frac{1}{|\beta_i|} \sum_{(x,y)\in\beta_i} \nabla_{\theta_i} \sum_c y \log f(x; \theta_i)$
  end for
end procedure

2.5 TeamNet Inference

Once the experts are trained, in the inference phase, given a new instance x, each expert makes its own prediction and computes the respective predictive entropy. A gate function is then applied to the predictive entropy and selects the prediction of the expert with the least uncertainty as the final output. This procedure is illustrated in Figure 2.3.

The arg min gate in Figure 2.3 is not the only way to combine outputs of multiple expert models. For example, one can take the (weighted) majority vote from all experts as in ensemble learning(Littlestone and Warmuth, 1994). However, since the experts are trained to highly specialize on a subset of the data, considering the prediction of "non-expert" can be detrimental.

2.6 Performance Evaluation

In this section, we evaluate the proposed algorithms using two image datasets, namely, the MNIST handwritten digit and the CIFAR-10 datasets. In addition to testing TeamNet's predictive accuracy, we also evaluate its inference time, memory usage and CPU/GPU usage for image classification tasks. Its performance is compared with

three different message passing interface (MPI) implementations to parallelize baseline models and the state-of-the-art Sparsely-Gated Mixture-of-Experts (SG-MoE).

2.6.1 Implementation

In the experiment, communication among the edge devices is done through TCP sockets over WiFi. Each edge device runs a listening socket to accept incoming data. Two types of edge devices (Jetson TX2 and Raspberry Pi 3 Model B+) are employed with TensorFlow, CUDA and cuDNN pre-installed. Trained experts (neural networks) are executed on the TensorFlow framework at the inference stage.

Message Passing Inference (MPI) Two types of neural networks are evaluated in the experiments: MLP and convolutional neural network (CNN) with the Shake-Shake regularization. In the first case, matrix (weights) multiplication can be split among multiple edge nodes using the MPI protocol (MPI-Matrix). In the second case, there are two main branches in the Shake-Shake CNN, which can be split into two edge nodes and coordinated through the MPI protocol (MPI-Branch). Therefore, MPI-Branch is only evaluated in experiments employing two edge devices. Alternatively, we can distribute convolutional kernels and their associated computation onto multiple edge devices. This approach is called (MPI-Kernel). It can be tested with multiple edge devices.

All MPI approaches are executed on the TensorFlow framework at the inference stage.

Sparsely-Gated Mixture-of-Experts (SG-MoE) SG-MoE requires both experts and a gate to be trained together as a joint architecture. Similar to TeamNet,

SG-MoE is trained with the TensorFlow framework on NVIDIA 1080TI GPUs. At the inference stage, each expert is executed on one edge node, and the gate is placed on one of the edge nodes.

Two protocols are evaluated for communication among SG-MoE experts, namely, gRPC, an open source remote procedure call (RPC) system, and MPI. The respective SG-MoE implementations are called SG-MoE-G and SG-MoE-M.

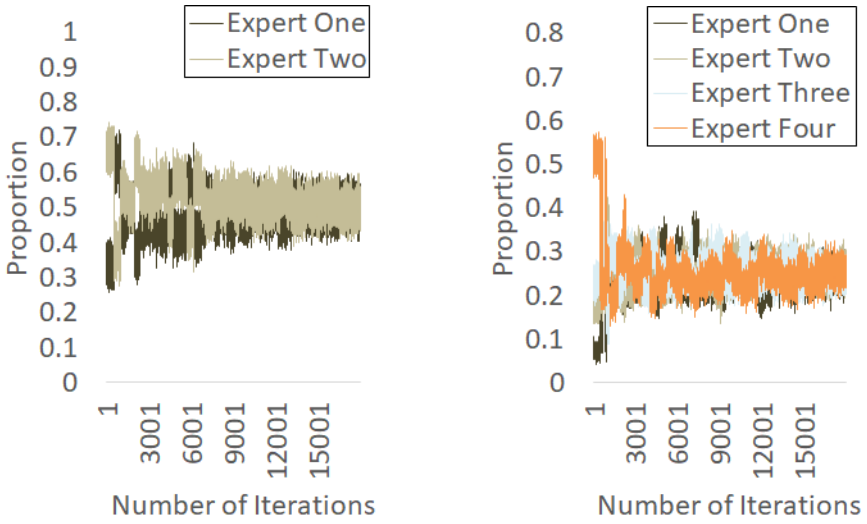
2.6.2 Experimental Setup

In the experiments, two datasets are evaluated: MNIST and CIFAR-10. MNIST is a dataset of handwritten digits on grey color images. There are 10 classes in total from digit zero to digit nine. It consists of a training set of 60000 images and a test set of 10000 images.

CIFAR-10 (Krizhevsky *et al.*, 2010) is a benchmark dataset in image classification. There are 10 classes in the dataset such as the airplane, automobile, bird, and dog. It consists of 60000 32-by-32 color images, with 50000 images for training and 10000 images set aside for evaluation. CNN allows fast inference and is commonly used in image classification such as the CIFAR-10 image classification task.

2.6.3 Handwritten Digit Recognition

Multi-layer perceptron (MLP) classifiers are trained with the MNIST dataset to recognize handwritten digits on images. With TeamNet, four 2-layer (4xMLP-2) and two 4-layer multi-layer perceptrons (2xMLP-4) are trained respectively. They collaborate with each other in prediction using $\arg\min$ gate in Figure 2.3. For comparison, an 8-layer MLP (MLP-8) is trained on the same dataset with the same



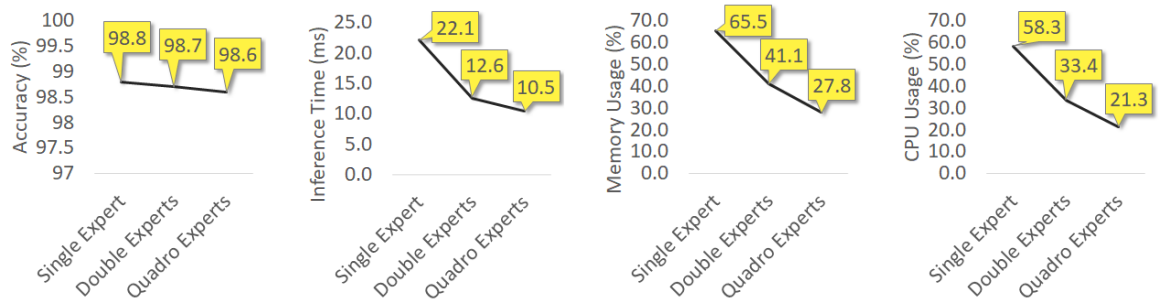
(a) Double Experts (b) Quadro Experts

Figure 2.5: TeamNet's convergence is evaluated for handwritten digit recognition by monitoring at each iteration the proportion of data assigned to each expert. (a) With two experts, the proportion deviates from the set point (0.5) initially and converges to it at about the 12000^{th} iteration. (b) With four experts, the proportion fluctuates at the beginning, but finally reaches the set point (0.25) at about the 15000^{th} iteration.

number of epochs as the baseline model.

The first experiment is conducted on the edge device: Raspberry Pi 3 Model B+. Accuracy, inference time, memory and CPU usages are measured for all three scenarios. From Figure 2.6, we observe that TeamNet with quadro experts (4xMLP-2) has the shortest inference time, lowest memory and CPU consumption due to a smaller model size on each edge device. At the same time, the predictive accuracy is not compromised of both 4xMLP-2 (quadro experts) and 2xMLP-4 (double experts).

The second experiment compares TeamNet with MPI in the context of distributed edge computing (Table 2.1). Jetson devices participate in the experiment, with WiFi connecting with each other. Accuracy, inference time, memory, CPU and GPU usages are measured during the experiment. The results show that TeamNet far excels MPI in



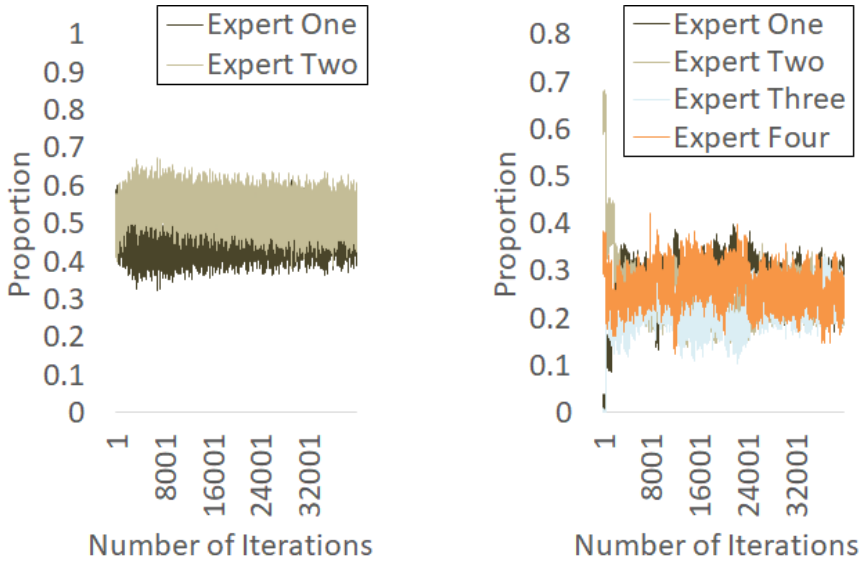
(a) Accuracy (b) Inference Time (c) Memory (d) CPU Usage

Figure 2.6: TeamNet's performance is evaluated on Raspberry Pi 3 Model B+ for handwritten digit recognition. With more experts in TeamNet, inference becomes faster, and memory and CPU consumption become smaller on the edge node. The accuracy is generally not compromised.

inference time. Inference time with MPI is even slower than the one with the baseline model. The reason is that MPI requires frequent communication among Jetson devices per each matrix multiplication, while TeamNet requires only communication at first and at last. WiFi is much slower than the infinite band (mostly used in the data center). It turns out to be overkill when the WiFi communication between edge devices is over frequent in parallel computing. Hence, TeamNet is a better architecture for distributed edge computing.

In the third experiment, TeamNet is compared with two SG-MoE approaches: SG-MoE with gRPC and SG-MoE with MPI. Accuracy, inference time, memory, CPU and GPU usages are measured during the experiment. TeamNet is more accurate than SG-MoE most of the time since SG-MoE adopts random data assignment and does not promote specialization on the experts (Table 2.1).

The fourth experiment is to test whether the proportion of data assigned to each expert eventually converges to the set point. Figure 2.5a shows that the proportion deviates from the set point (0.5) initially, and as the training goes, the proportion



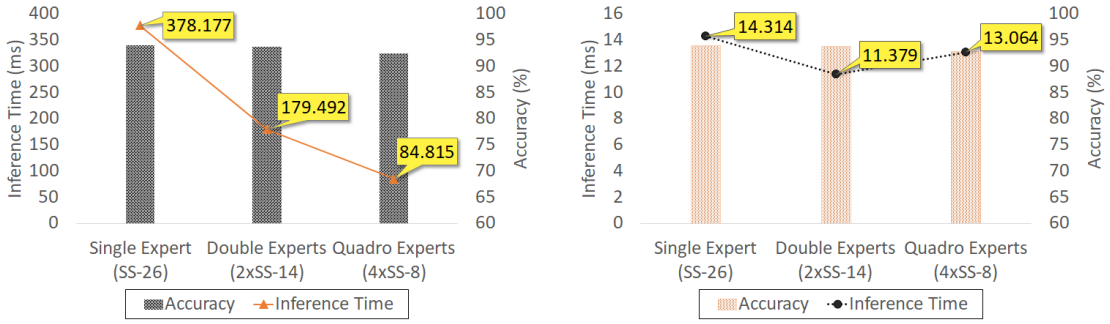
(a) Double Experts (b) Quadro Experts

Figure 2.7: TeamNet's convergence is evaluated for the image classification task by monitoring at each iteration the proportion of data assigned to each expert. (a) The proportion is initially close to the set point (0.5) by luck, but it deviates from the set point (0.5) very fast since both experts had limited knowledge of the dataset at this stage and cannot have a clear judgment on uncertainty. As the training goes, both experts become certain of an equal amount of data, and the proportion becomes closer and closer (and converges) to the set point (0.5). (b) With four experts in TeamNet, the proportion deviates from the set point (0.25) at the beginning but converges to it finally at about the 32000^{th} iteration.

gets closer and closer to the set point (0.5) and eventually converges to it. Figure 2.5b demonstrates that with four experts in TeamNet, the proportion fluctuates at the beginning but eventually reaches the set point (0.25).

2.6.4 Image Classification

In the first experiment, we evaluate the performance of our algorithm on CNNs with the Shake-Shake regularization. With TeamNet, four 8-layer (4xSS-8) and two 14-layer

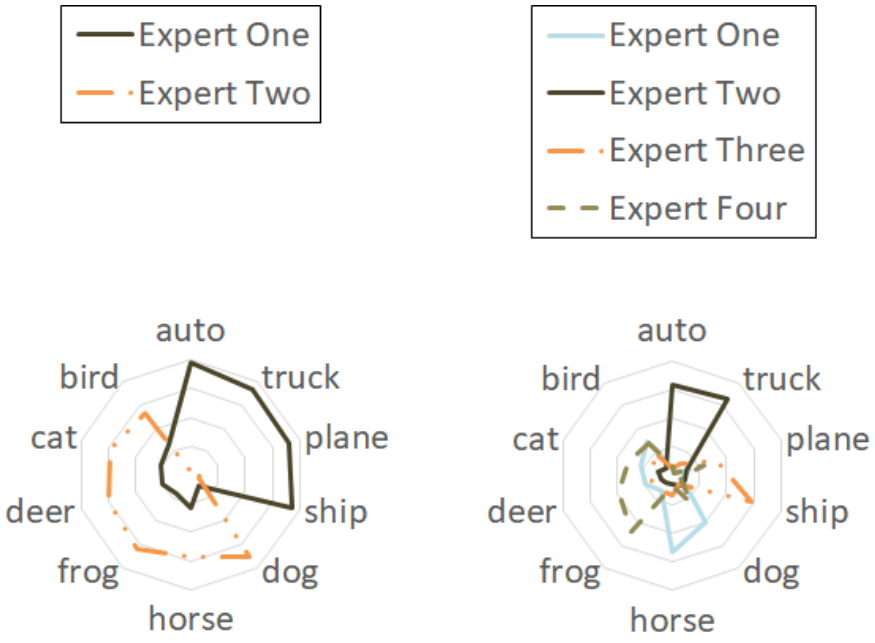


(a) Jetson TX2 CPU only (b) Jetson TX2 GPU and CPU

Figure 2.8: TeamNet's performance is evaluated on both Jetson CPUs and GPUs for the image classification task. (a) On Jetson CPUs, inference becomes faster with more experts in TeamNet, while accuracy is generally not compromised. (b) On Jetson GPUs, the fastest inference is acheived when there are two experts in TeamNet. Because there is a fixed communication cost over the WiFi network, a shorter time cannot be achieved when the communication latency is too close to the actual latency caused by the computation unless the model size is significantly larger and more computation is needed.

Shake-Shake CNNs (2xSS-14) are trained respectively. Predictive accuracy, inference time, memory, CPU and GPU usages on Jetson TX2 are compared to those of the 26-layer Shake-Shake CNN (SS-26) model. The experiments demonstrate that TeamNet nearly halves the inference time on Jetsons with CPU only, while does not sacrifice accuracy (Figure 2.8a). On Jetson GPUs, 2xSS-14 achieves the fastest inference. The performance gain from smaller model size is overwhelmed by the communication cost when 8-layer and 14-layer Shake-Shake models are running on Jetson GPUs (Figure 2.8b).

In the second experiment, TeamNet is compared with MPI and Sparsely-Gated Mixture-of-Experts (SG-MoE) (Shazeer *et al.*, 2017) on the Jetson TX2 edge devices. The experiment shows that TeamNet has much shorter inference time than that of MPI, and more accurate than SG-MoE. Faster inference time is benefited by infrequent



(a) Double Experts (b) Quadro Experts

Figure 2.9: Specialization is emphasized in TeamNet. (a) With two experts in TeamNet, Expert One is more certain of machines such as airplanes, automobiles and trucks, while Expert Two is more certain of animals such as cats and dogs. (b) With four experts in TeamNet, Expert One and Expert Four are more certain of animals, while Expert Two and Expert Three are more certain of machines.

communication among edge devices, and better accuracy is owing to the specialization of each expert. In SG-MoE, data examples are randomly assigned to experts for learning, thus specialization is not being emphasized. On the other hand, TeamNet assigns the most certain data examples to the experts for learning, thus experts will only learn the ones which they are most familiar with. That is how the specialization being emphasized in TeamNet and why TeamNet outperforms SG-MoE (Table 2.2).

The third experiment evaluates whether the proportion of data assigned to each expert converges eventually to the set point. Figure 2.7a demonstrates that the proportion is close to the set point (0.5) at the beginning while very fast deviates

from it. The reason is that both experts have limited knowledge of the dataset at this stage and cannot have a clear judgment of uncertainty. As the training goes, both experts know more about the dataset, and the proportion is closer and closer to the set point. Figure 2.7b shows that when there are four experts in TeamNet, the proportion converges to the set point (0.25) eventually at about 32000^{th} iteration.

In the fourth experiment, we further investigate the effects of specialization which is forced by the training algorithm. The experiment is performed on the CIFAR-10 dataset. There are 10 classes in the dataset such as airplanes, birds, and cats. When there are two experts in TeamNet, we observe that Expert One is more certain of machines such as airplanes, automobiles, ships, and trucks; on the other hand, Expert Two knows more about animals such as birds, cats, and horses (Figure 2.9a). When there are four experts in TeamNet, we observe that Expert One and Expert Four are more certain of animals, and each masters half of this category; on the other hand, Expert Two and Expert Three are more certain of machines: Expert Two has more knowledge about trucks and automobiles, while Expert Three knows more about ships and airplanes (Figure 2.9b).

2.7 Conclusion

In this paper, we propose TeamNet, a novel framework for collaborative inference among storage-and-compute-limited edge devices. Unlike existing computation-partition solutions that take a pre-trained model and determine the best partitions between edge devices and the cloud, TeamNet is a *model partition* approach by training multiple small *specialized* models. It is proven to work well with the commonly used neural networks (CNNs and MLPs) on different numbers and types of edge devices.

Extensive experiments have demonstrated that TeamNet can be executed on low-end edge devices such as Raspberry Pi and Jetson TX2 devices with superior performance than the baseline solution and the state-of-the-art MoE approach in inference time with marginal degradation in inference accuracy.

Currently, TeamNet is trained with the objective of equally partitioning training samples among experts. As part of future work, we will explore other objective functions especially those can adapt to the imbalances among different classes in training data.

2.8 Appendix A: Proof Sketch of Convergence

In this section, we analyze the convergence of Algorithm 2 based on some simplifying assumptions. Experimental results in Section 2.6 have shown empirically the procedure converges.

Assumption 1. *The expert model trained with $x\%$ of training data has the least uncertainty of $x\%$ of test instances among all expert models.*

Assumption 2. *Each batch in training is sufficiently random.*

Assumption 3. *Solution to* (2.4.4) *gives*

$$\sum_{i=1}^{K} \left| \bar{\gamma}_i(\delta) - \left(\frac{1}{K} - a \cdot (\gamma_i - \frac{1}{K}) \right) \right| = 0, \tag{2.8.1}$$

Proof. From Assumption 3, we have

$$\bar{\gamma}_i(\delta) \approx \frac{1}{K} - a \cdot (\gamma_i - \frac{1}{K}), \tag{2.8.2}$$

for all $i = 1, ..., K$. Let $|\beta|$ be the number of examples in a batch, and $n_{i,l}$ be the number of examples in the l^{th} batch used to train Expert i. Consider the L^{th} batch.

From Assumption 1,

$$\gamma_{i,L} = \frac{\sum_{l=1}^{L-1} n_{i,l}}{(L-1) \cdot |\beta|} \tag{2.8.3}$$

From (2.8.2), we have

$$\bar{\gamma}_{i,L}(\delta) = \frac{1}{K} - a \cdot (\gamma_{i,L} - \frac{1}{K}), \tag{2.8.4}$$

where $\bar{\gamma}_{i,L}(\delta)$ is the percentage of samples in the Lth batch ith expert is least uncertain of according to $\bar{G}$. Since

$$n_{i,L} = \bar{\gamma}_{i,L} \cdot |\beta|, \tag{2.8.5}$$

we have

$$\begin{aligned} \gamma_{i,L+1} &= \frac{\sum_{l=1}^{L-1} n_{i,l} + n_{i,L}}{L \cdot |\beta|} \\ &= \frac{\gamma_{i,L} \cdot (L-1) + \frac{1}{K} - a \cdot (\gamma_{i,L} - \frac{1}{K})}{L}. \end{aligned}$$

Subtracting $\frac{1}{K}$ from both sides and with further simplification, we have

$$\begin{aligned} \gamma_{i,L+1} - \frac{1}{K} &\approx \frac{\gamma_{i,L} \cdot (L-1) + \frac{1}{K} - a \cdot (\gamma_{i,L} - \frac{1}{K}) - \frac{L}{K}}{L} \\ &= \frac{(\gamma_{i,L} - \frac{1}{K})(L-1) - a \cdot (\gamma_{i,L} - \frac{1}{K})}{L} \\ &= \frac{L-1}{L}(1 - \frac{a}{L-1})(\gamma_{i,L} - \frac{1}{K}) \end{aligned}$$

Therefore,

$$\begin{aligned}\left|\gamma_{i,L+1} - \frac{1}{K}\right| &= \frac{L-1}{L}(1 - \frac{a}{L-1})\left|\gamma_{i,L} - \frac{1}{K}\right| \\ &= \frac{1}{L}\prod_{l}\left(1 - \frac{a}{l-1}\right) \times \left|\gamma_{i,1} - \frac{1}{K}\right|.\end{aligned}$$

With $a > 0$, clearly, as $L \to \infty$, $\gamma_{i,L+1} \to \frac{1}{K}$ In other words, the training data will be *equally* partitioned among experts. □

Table 2.1: TeamNet is compared with the baseline, a MPI approach, and SG-MoE with gRPC and MPI for handwritten digit recognition. (a) On Jetson CPUs, TeamNet is faster than other approaches while the accuracy is not compromised. MPI requires frequent communication among Jetson devices, thus it is slower than other approaches. (b) On Jetson GPUs, TeamNet is still faster than MPI and the SG-MoE approaches. However, TeamNet is not better than the baseline. The root cause is that there is a fixed cost over the WiFi communication. The performance gain from a smaller model is overwhelmed by the communication cost, especially when the device's computing power is significantly larger than the computation needed by the model.

(a) Jetson TX2 CPU only

	Baseline	**Double Nodes**				**Quadro Nodes**			
		TeamNet	***MPI-Matrix***	***SG-MoE-G***	***SG-MoE-M***	***TeamNet***	***MPI-Matrix***	***SG-MoE-G***	***SG-MoE-M***
Accuracy (%)	98.8	**98.7**	**98.7**	98.6	98.6	**98.7**	**98.7**	98.5	98.5
Inference Time (ms)	3.4	**3.2**	108.2	5.9	6.9	**3.3**	189.0	4.1	10.3
Memory Usage (%)	8.2	6.0	8.55	7.8	9.1	4.4	6.78	6.1	6.6
CPU Usage (%)	55.3	30.7	31.9	21.9	54.3	21.2	34.6	11.5	48.2

(b) Jetson TX2 GPU and CPU

	Baseline	**Double Nodes**				**Quadro Nodes**			
		TeamNet	***MPI-Matrix***	***SG-MoE-G***	***SG-MoE-M***	***TeamNet***	***MPI-Matrix***	***SG-MoE-G***	***SG-MoE-M***
Accuracy (%)	98.8	**98.8**	**98.8**	98.7	98.6	98.7	**98.8**	98.5	98.5
Inference Time (ms)	0.3	**1.5**	104.8	5.8	3.2	**2.6**	187.7	4.5	6.9
Memory Usage (%)	10.0	9.9	15.1	15.3	15.1	8.3	14.5	13.9	14.0
CPU Usage (%)	37.0	21.7	30.8	15.1	49.5	15.9	34.7	9.2	42.5
GPU Usage (%)	5.0	3.8	1.9	3.4	3.0	2.8	6.8	1.6	1.6

Table 2.2: TeamNet is compared with the baseline model, two MPI approaches, and SG-MoE with gRPC and MPI for the image classification task. TeamNet is faster and it consumes fewer resources on each edge device than the baseline model. (a) On Jetson CPUs, SG-MoE with gPRC is slightly faster than TeamNet, but generally less accurate. MPI approaches are slightly more accurate than TeamNet but much slower. (b) On Jetson GPUs, TeamNet is the fastest approach among the others, though MPI approaches are slightly more accurate.

(a) Jetson TX2 CPU only

	Baseline	**Double Nodes**					**Quadro Nodes**			
		TeamNet	***MPI-Kernel***	***MPI-Branch***	***SG-MoE-G***	***SG-MoE-M***	***TeamNet***	***MPI-Kernel***	***SG-MoE-G***	***SG-MoE-M***
Accuracy (%)	94.0	93.7	**93.9**	**93.9**	89.7	90.1	92.4	**93.6**	87.1	87.8
Inference Time (ms)	378.2	179.5	2684.3	1227.8	**157.3**	192.4	84.8	6722.7	**67.8**	71.6
Memory Usage (%)	27.3	20.9	18.6	11.8	15.0	15.6	18.5	14.9	10.7	14.1
CPU Usage (%)	95.4	91.9	40.3	47.6	45.7	75.7	82.0	34.5	21.4	57.7

(b) Jetson TX2 GPU and CPU

	Baseline	**Double Nodes**					**Quadro Nodes**			
		TeamNet	***MPI-Kernel***	***MPI-Branch***	***SG-MoE-G***	***SG-MoE-M***	***TeamNet***	***MPI-Kernel***	***SG-MoE-G***	***SG-MoE-M***
Accuracy (%)	93.9	93.8	93.9	**94.0**	89.4	89.0	92.8	**93.5**	87.3	87.3
Inference Time (ms)	14.3	**11.4**	2611.7	1002.7	31.7	29.4	**13.1**	7062.9	30.6	29.5
Memory Usage (%)	43.8	36.8	27.5	26.3	26.8	26.1	33.5	22.6	22.6	22.7
CPU Usage (%)	24.3	20.3	33.4	35.4	16.9	43.7	16.2	34.2	9.4	46.3
GPU Usage (%)	34.6	26.7	2.3	2.7	15.2	14.2	18.1	1.3	5.3	5.3

Chapter 3

CacheNet: An Information Maximizing Caching Framework

This chapter is reproduced from "CacheNet: A Model Caching Framework for Deep Learning Inference on the Edge", Yihao Fang, Shervin Manzuri Shalmani, and Rong Zheng, submitted on May 20, 2020 to IEEE Transactions on Mobile Computing (TMC). The author of this thesis is the first author and the main contributor of this publication.

Abstract

The success of deep neural networks (DNN) in machine perception applications such as image classification and speech recognition comes at the cost of high computation and storage complexity. Inference of uncompressed large scale DNN models can only run in the cloud with extra communication latency back and forth between cloud and end devices, while compressed DNN models achieve real-time inference on end devices at the price of lower predictive accuracy. In order to have the best of both worlds (latency and accuracy), we propose CacheNet, a model caching framework. CacheNet caches low-complexity models on end devices and high-complexity (or full) models on edge or cloud servers. By exploiting temporal locality in streaming data, high cache hit and consequently shorter latency can be achieved with no or only marginal decrease in prediction accuracy. Experiments on CIFAR-10 and FVG have shown CacheNet is $58-217\%$ faster than baseline approaches that run inference tasks on end devices or edge servers alone.

3.1 Introduction

In recent years, deep neural networks (DNN) have achieved tremendous successes in perception applications such as image classification, speech recognition, target tracking and machine translation. In many cases, they outperform human beings in accuracy. However, such high accuracy comes at the cost of high computation and storage complexity due to large model sizes. For instance, ResNet-152 contains 152 layers and over 60M parameters. Inference using such large-scale DNN models cannot be accomplished on end devices with limited computation power and storage

in real-time. As a result, many model compression techniques have been proposed to reduce the size of DNN networks often at the expense of prediction performance (Lin *et al.*, 2016b; Sainath *et al.*, 2013). Therefore, application developers face a dilemma to choose between a highly accurate model that can only run in the cloud with extra communication latency of uploading raw input data and getting the results back, or local execution of compressed models with reduced accuracy.

Is it possible to get the best of both worlds? In other words, can we achieve a good trade-off between latency and prediction accuracy? This question has to some degree been answered by partitioning approaches (Kang *et al.*, 2017; Teerapittayanon *et al.*, 2017; Fang *et al.*, 2019). They mainly fall into two paradigms: 1) *model partitioning*: concurrent computing among edge nodes and/or end devices (Fang *et al.*, 2019), which collaboratively performs inference in parallel per a particular sensor input; 2) *computation partitioning*: partition between edge and cloud, which take a pre-trained deep model and decide at run-time based on computation capability of local and cloud compute nodes and communication overheads where portions of computation should reside (Kang *et al.*, 2017). The inference time of both paradigms is clearly lower bounded by the smaller (or smallest) of inference times on the end device and a cloud node (or on all end devices/edge nodes). Furthermore, as per computation partitioning, since DNN models tend to be sequential, the possible ways of partitioning are limited.

In this work, we take a drastically new approach in addressing the trade-off between latency and prediction accuracy of DNNs. Our approach is motivated by two observations of perception applications with inputs from natural scenes or human interactions. First, despite the fact that such applications may need to handle a

large number of input classes over time, the classes of inputs commonly encountered can be much smaller. For instance, an average English speaking person uses about 4000 words in daily life out of 171,476 words listed in the second edition of Oxford English Dictionary. Secondly, there exists strong temporal locality in terms of the types of inputs encountered in a short period of time. This is especially true for vision processing where rich redundancy exists among consecutive video frames (Xu *et al.*, 2018; Chen *et al.*, 2015; Huynh *et al.*, 2017; Mathur *et al.*, 2017).

To exploit these two properties, we propose *CacheNet*, a model caching framework for deep learning inference on edge. CacheNet is inspired by caching in the memory hierarchy. In computer architecture, the memory hierarchy separates computer storage (e.g., register, cache, random access memory, etc.) based on response time (Mutlu and Subramanian, 2015). Caching increases data retrieval performance (e.g. faster response time) by reusing previously retrieved and computed data in the storage. Analogous to the memory hierarchy, end devices are closer to data sources and thus have faster response time but lower storage capacity; while an edge server has more storage capacity but relatively longer network latency. However, unlike the memory hierarchy that only stores data, CacheNet stores DNN models. To mitigate the limited computation power on end devices, only down-sized models with high confidence in the current input data are stored. Thanks to the temporal locality and the small number of frequently observed classes, the cached model only needs to be replaced infrequently.

In short, CacheNet combines model partitioning with caching. Instead of training a single large-scale model, CacheNet generates multiple small submodels each capturing a partition of the knowledge represented by the large model. In the proposed architecture,

the end device is responsible for selecting a locally cached model and performing the inference; whereas the edge server stores the baseline model and submodels, and is responsible to handle "cache misses" when there are sufficient changes in input data. CacheNet is agnostic to the architecture of a baseline deep model. Both the number of submodels and the baseline deep model can be specified by users.

We have implemented CacheNet in TensorFlow, TensorFlow Lite and NCNN. Here, TensorFlow is a high-performance framework for neural network training, while TensorFlow Lite and NCNN are lightweight inference framework optimized for edge computing. CacheNet has been evaluated on a variety of end devices and two different datasets (CIFAR-10 Krizhevsky *et al.* (2009) and FVG Zhang *et al.* (2019)). We found that CacheNet outperforms end-device-only and edge-server-only approaches in inference time without compromising inference accuracy. For CIFAR-10, CacheNet is 2.2 times faster than the end-device-only approach and 58% faster than edge-server-only; for FVG, it is 1.5 times faster than end-device-only and 71% faster than edge-server-only.

The rest of the paper is organized as follows. Section 3.2 describes related work to CacheNet from two perspectives: caching and partitioning. An overview of our approach is given in Section 3.3 from requirements to system level design. In Section 3.4, we elaborate on aspects of training CacheNet and formalize CacheNet mathematically. Details of inference is provided in Section 3.5 from *partition selection* on the edge server to *cache replacement* on end devices. Section 3.6 provides evaluations of CacheNet on multiple end devices including Jetson TX2, Jetson Nano, and Raspberry Pi 4. The conclusion and future works are stated in Section 3.7.

3.2 Related Work

Existing algorithmic approaches to accelerate machine learning inference on end devices mainly fall into three categories, namely: i) model compression, ii) computation partitioning, and iii) reduction of computation in machine learning pipelines. The three categories of approaches are orthogonal to one another and can be applied jointly. Among the three, the latter two are closer to CacheNet and will be discussed in further details in this section.

3.2.1 Computation Partitioning

Partitioning splits a known neural network model into multiple parts to be executed either sequentially or concurrently on the edge and cloud. It can be performed between layers. By trading off between the time offloading computation to the cloud with the time spent in local computation on edge, a shorter latency could be achieved (Kang *et al.*, 2017).

A more sophisticated computation partitioning was proposed in distributed deep neural networks (DDNNs) (Teerapittayanon *et al.*, 2017). DDNN was designed to perform fast and localized inference using shallow portions of a neural network on end devices. Using an exit point after device inference, an output is classified locally. If the classification cannot be made due to low confidence, the task is escalated to a higher exit point (e.g. the edge exit) in the hierarchy until the last exit (the cloud exit). With multiple exit points, DDNNs can significantly reduce communication costs.

TeamNet (Fang *et al.*, 2019) takes a different approach for computation partitioning. Rather than dividing a pre-trained neural network structurally, it explores knowledge specialization and trains multiple small NNs through competitive and selective learning.

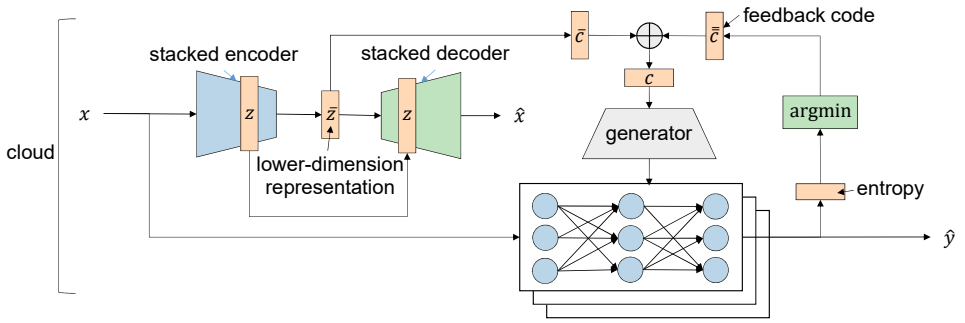
During inference, the NNs are executed in parallel on cooperative end devices. By decision-level fusion, a master node (either one of the end devices or a edge/cloud node) outputs the final inference results. Since computation partitioning in TeamNet is done at model level, it is also considered a model partitioning approach.

CacheNet bears similarity with TeamNet in training multiple shallower models to represent the knowledge of a single deep model. However, unlike TeamNet that requires concurrent execution of the shallower models, CacheNet utilizes a "selector" to determine the suitable shallow model based on input data. In CacheNet, when a cache hit occurs, the inference is performed on the end device only. The overall inference time is reduced by the indexability of specialized submodels and running the suitable submodel locally most of the time.

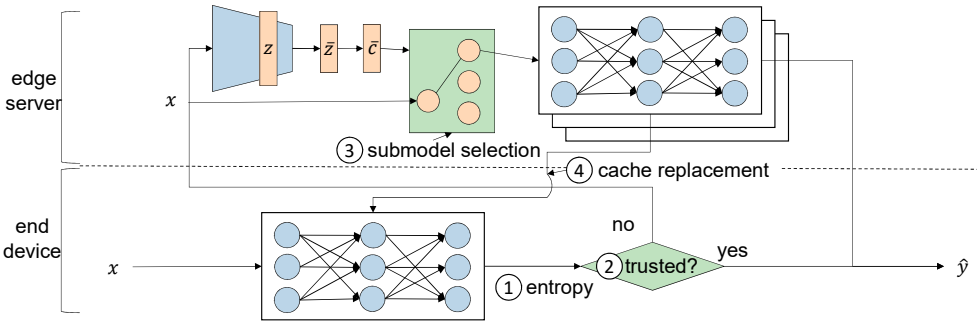
3.2.2 Computation Reduction

Exploiting the existence of the temporal locality in input data, several works reduce DNN inference time by reusing all or part of previous computation results.

Glimpse is a continuous, real-time object recognition system for camera-equipped mobile devices (Chen *et al.*, 2015). In Glimpse, object recognition tasks are executed on local devices when the communication latency between the server and mobile device is higher than a frame-time. In addition to using a reduced model for faster local inference, Glimpse uses an active cache of video frames on the mobile device. A subset of the frames in the active cache is used to track objects on the mobile, using (stale) hints about objects that arrive from the server from time to time. In (Xu *et al.*, 2018), Xu *et al.* proposed DeepCache, a principled cache design for deep learning inference in continuous mobile vision. It breaks down an input video frame into



(a) Train



(b) Inference

Figure 3.1: (a) CacheNet first partitions a neural network into multiple smaller specialized neural networks in the cloud. (b) Owing to the temporal locality that exists in the video, the smaller specialized neural network will work well on consecutive frames over a short period. An abrupt change of frame induces higher entropy and triggers cache replacement.

smaller blocks and discovers similar blocks between consecutive frames using diamond search (Xu *et al.*, 2018). Computation on reusable regions (e.g., feature maps) can thus be cached and propagated through subsequent layers without further processing. In (Apicharttrisorn *et al.*, 2019), to reduce energy drain while maintaining good object tracking precision, the authors develop a software framework called MARLIN. MARLIN only uses a DNN as needed, to detect new objects or recapture objects that significantly change in appearance. It employs lightweight methods in between

DNN executions to track the detected objects with high fidelity. Alternatively, we can view MARLIN as reuse the detection and classification results by associating detected objects across multiple frames. In Guo *et al.* (2018), Guo et al. proposed FoggyCache for cross-device approximate computation reuse. FoggyCache reuses previously computed outputs by harnessing the "equivalence" between different input values. Content lookup and high quality reuse are achieved by the adoption of adaptive locality sensitive hashing (A-LSH) and homogenized k-nearest neighbors (H-kNN). Harnessing reuse opportunities translates to reduced computation latency and energy consumption.

All afore-mentioned approaches are orthogonal to CacheNet. In DeepCache and MARLIN, a full-fledged deep model is still needed on an end device and thus the worst-case execution time is not reduced. This is in contrast with CacheNet, which only runs reduced submodels locally.

3.3 System Design

CacheNet is a distributed inference framework on edge. Its training phase happens in the cloud and the inference is a collaboration between the edge server and the end device. The intuition behind CacheNet is dividing a neural network's knowledge into multiple specialized partitions (neural networks). These specialized partitions are generally a few times smaller than the original neural network, and only the specialized partition is transferred to the end device for inference. From the end device's perspective, it caches only a times smaller and specialized partition of the knowledge, and thus its inference is times faster than the original ones.

The challenges of partitioning are two folds: 1) each partition must be sufficiently

specialized and the combination (collaboration) of all partitions must behave roughly equivalently to the original neural network; 2) There must be a selector that picks the right partition given a specific hint at a time. The first challenge was mostly solved by TeamNet (Fang *et al.*, 2019), while the second one has not been solved by any approaches at this point.

In order to solve the second challenge, it is necessary to formalize the hint as a specific representation. Inspired by coding theory, a code vector is a good representation as long as the mutual information between the code vector and the input image is maximized at the training phase. Although we have the hint representation, it is still difficult to associate the representation with a specific portion of the knowledge. To do so, we introduce a generator that generates the neural network's parameters accordingly to the given code representation.

Thus, in training (Figure 3.1a), we need to 1) maximize the mutual information between the code representation and the input image; 2) better associate the code representation with the specialized neural network partition; 3) train each partition with respect to their output entropy, which has been demonstrated practical in TeamNet (Fang *et al.*, 2019). CacheNet's system design is therefore conducted simultaneously with respect to the above objectives.

During inference (Figure 3.1b), CacheNet should infer the code representation from a particular input image, and then the code representation will be used as a hint to tell which specialized partition to be cached on the end device. Without the need to transfer the input frames to the edge server every time, inference latency can be shortened. The accuracy is generally not sacrificed, because there exists a temporal locality on consecutive frames most of the time. As long as there is not an abrupt

change of the scene, a specialized partition should work well; otherwise (e.g., in regard to edited clips from multiple cameras, or a fast-moving object/camera Kwon and Lee (2008)), a cache replacement should be triggered, considering a partition only holds a subset of the knowledge.

3.4 Training CacheNet

As illustrated in Figure 3.1a, to train CacheNet submodels, we need to first divide the input data into partitions[1]. The index associated with a partition is taken as an input to a neural network generator to produce the corresponding submodel for the partition. The encoder that maps input data to partition indices and the submodels will be optimized jointly. Next, we discuss the steps in detail.

3.4.1 Stacked Information Maximizing Variational Autoencoder (S-InfoVAE)

The purpose of this step is to map input data into a low dimension space for further partitioning. The low-dimension representation should preserve the proximity among data and allow "reconstruction" of the orignal data.

Variational Bayesian autoencoder was proposed by Kingma and Welling (Diederik *et al.*, 2014). The basic idea is to find a lower-dimension latent variable underlying the corresponding input distribution. Let z denote the latent variable and x represent the input variable. Consider a dataset $D = \{X, Y\}$, where X is drawn independently from an input probability distribution $p_D(x)$. Suppose that $p_\xi(z)$ (the prior distribution

[1]The partitions are overlapping as will be discussed in Section 3.4.2.

of z) and the conditional probability distribution $p_\xi(x|z)$ are both parameterized by a neural network with parameters ξ. One can find the optimal parameters ξ by maximizing the log-likelihood as:

$$\mathbb{E}_{p_D(x)}\left[\log p_\xi(x)\right] = \mathbb{E}_{p_D(x)}\left[\log \mathbb{E}_{p_\xi(z)}\left[p_\xi(x|z)\right]\right]. \tag{3.4.1}$$

However, the integral of the marginal likelihood $p_\xi(x)$ is generally intractable even for a moderately complex neural network with a single non-linear hidden layer. A possible approach (Diederik *et al.*, 2014) is to rewrite $\log p_\xi(x)$:

$$\log p_\xi(x) = D_{KL}(q_\psi(z|x)||p_\xi(z|x)) + \mathcal{L}(\xi, \psi; x), \tag{3.4.2}$$

where

$$\mathcal{L}(\xi, \psi; x) = -D_{KL}(q_\psi(z|x)||p_\xi(z)) + E_{q_\psi(z|x)} \log p_\xi(x|z). \tag{3.4.3}$$

Since Kullback-Leibler divergence is always non-negative, $\mathcal{L}(\xi, \psi; x)$ is a lower bound of $\log p_\xi(x)$, namely,

$$\mathcal{L}(\xi, \psi; x) \leq \log p_\xi(x). \tag{3.4.4}$$

By maximizing the lower bound $\mathcal{L}(\xi, \psi; x)$, the log likelihood $\log p_\xi(x)$ is maximized as well. However, since the latent variable z is of lower dimension than the input variable x, any optimization against x may be magnified compared to z. To counteract the imbalance problem, Zhao et al. (Zhao *et al.*, 2019) propose to put more weight on z. Let $\mathcal{L}(\xi, \psi)$ be the expectation of $\mathcal{L}(\xi, \psi; x)$ with respect to the input distribution

$p_D(x)$. We then have,

$$\begin{aligned}\mathcal{L}(\xi,\psi) =& E_{p_D(x)}\mathcal{L}(\xi,\psi;x)\\ =& - D_{KL}(q_\psi(x,z)||p_\xi(x,z))\\ =& - D_{KL}\left(q_\psi(z)||p_\xi(z)\right)\\ & - \mathbb{E}_{p_\xi(z)}\left[D_{KL}\left(q_\psi(x|z)||p_\xi(x|z)\right)\right].\end{aligned} \tag{3.4.5}$$

To put more weights on z, one needs to add i) a scaling parameter to the Kullback-Leibler divergence between $q_\psi(z)$ and $p_\xi(z)$, and ii) a term of mutual information between x and z (Zhao *et al.*, 2019):

$$\begin{aligned}\mathcal{L}^*(\xi,\psi) =& - \lambda D_{KL}\left(q_\psi(z)||p_\xi(z)\right)\\ & - \mathbb{E}_{p_\xi(z)}\left[D_{KL}\left(q_\psi(x|z)||p_\xi(x|z)\right)\right]\\ & + \alpha I_{q_\psi(x,z)}(x;z).\end{aligned} \tag{3.4.6}$$

In practice, $\mathcal{L}^*(\xi,\psi)$ can be rewritten into (3.4.7) for more effective optimization (Zhao *et al.*, 2019):

$$\begin{aligned}\mathcal{L}^*(\xi,\psi) =& \mathbb{E}_{p_D(x)}\mathbb{E}_{q_\psi(z|x)}\left[\log p_\xi(x|z)\right]\\ & - (1-\alpha)\mathbb{E}_{p_D(x)}D_{KL}(q_\psi(z|x)||p_\xi(z))\\ & - (\alpha+\lambda-1)D_{MMD}(q_\psi(z)||p_\xi(z)),\end{aligned} \tag{3.4.7}$$

where $D_{MMD}(q_\psi(z)||p_\xi(z))$ is the maximum-mean discrepancy between $q_\psi(z)$ and

$p_\xi(z)$.

Experiments show that when the latent variable z is of far lower dimension than the input variable x, the lower bound $\mathcal{L}^*(\xi, \psi)$ can not properly converge. To deal with this problem, we propose the S-InfoVAE by keeping z at a relative high dimension and introducing a second latent variable $\bar{z}$ of dimension two. The corresponding parameters (or equivalently the neural networks) of the two latency variables are stage-wisely optimized. Formally, the second optimization objective is defined as follows:

$$\bar{\mathcal{L}}^*(\bar{\xi}, \bar{\psi}) = E_{p_{\bar{\psi}}(z)} \mathcal{L}(\bar{\xi}, \bar{\psi}; z) \tag{3.4.8}$$

3.4.2 Indexability of Low-dimension Representation

To divide data into overlapping partitions, sophisticated indexes are needed. Let K be the total number of submodels, an input parameter of CacheNet. Each input sample in D is associated with one or more indices chosen from 1 to K and will be used to train the corresponding submodel(s). By allowing multiple indices per data sample or equivalently shared training data, we facilitate knowledge sharing across submodels. In this step, we determine the indices of input data solely based on the low-dimension representations from the S-InfoVAE. In subsequent sections, we will also incorporate feedback from the resulting submodels in the form of uncertainty.

Recall that $\bar{z}$'s are 2D vectors. To calculate the angular distance between the

vector $\bar{z} = [\bar{z}_1 \bar{z}_2]$ and the x-axis, the arctan trigonometric function is applied:

$$\theta = \begin{cases} \arctan \dfrac{\bar{z}_2}{\bar{z}_1} & \bar{z}_1 > 0 \\ \arctan \dfrac{\bar{z}_2}{\bar{z}_1} + \pi & \bar{z}_1 < 0, \bar{z}_2 \geq 0 \\ \arctan \dfrac{\bar{z}_2}{\bar{z}_1} - \pi & \bar{z}_1 < 0, \bar{z}_2 < 0 \\ \dfrac{\pi}{2} & \bar{z}_1 = 0, \bar{z}_2 > 0 \\ -\dfrac{\pi}{2} & \bar{z}_1 = 0, \bar{z}_2 < 0 \\ 0 & \bar{z}_1 = 0, \bar{z}_2 = 0. \end{cases} \tag{3.4.9}$$

For better convergence, a small noise term ϵ is added to the θ. To keep the resulting angles between 0 and 2π, a modulo function is applied as follows:

$$\tilde{\theta} = (\theta + \epsilon) \mod 2\pi. \tag{3.4.10}$$

For K partitions where each partition roughly occupies a region of $\frac{2\pi}{K}$, the midpoint of the k^{th} partition is given by $\frac{2\pi\left(k-\frac{1}{2}\right)}{K}$, for $k = 1, \ldots, K$. Let ζ be a vector of all such midpoints, namely:

$$\zeta = [\zeta_1 \ldots \zeta_K], \quad \zeta_k = \frac{2\pi\left(k - \frac{1}{2}\right)}{K}. \tag{3.4.11}$$

We wish to assign input samples to partitions based on their closeness to the K midpoints in polar coordinates. One straightforward approach is via a 1-nearest neighbor search, namely, finding k that minimizes $\min\left(|\tilde{\theta} - \zeta_k|, 2\pi - |\tilde{\theta} - \zeta_k|\right)$. Doing so will result in a one-hot vector with one for the kth element and zeros for all other

elements. Instead, we choose to define a *soft* code $\bar{c}$ as,

$$\bar{c} = \sum_{n=-1}^{n=1} \exp\left(-\frac{(\zeta - \tilde{\theta} + 2\pi n)^2}{2\sigma^2}\right), \tag{3.4.12}$$

where σ is a parameter that controls the speed of decay as $\tilde{\theta}$ deviates from the midpoints. Clearly, each element of $\bar{c}$ is between 0 and 1, and the maximum value occurs at $k = \arg\min_k \left(\min\left(|\tilde{\theta} - \zeta_k|, 2\pi - |\tilde{\theta} - \zeta_k|\right)\right)$.

With the soft code $\bar{c}$ of some input x and a threshold τ, we can determine which partition(s) it belongs to as $\{k|\bar{c}_k \geq \tau\}$. Plugging (3.4.11) and (3.4.12), we have $c_k \geq \tau$ if the following condition holds,

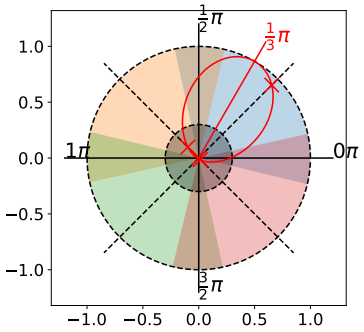
$$\frac{2\pi\left(k - \frac{1}{2}\right)}{K} - \sigma\sqrt{-2\log\tau} \leq \tilde{\theta} \leq \frac{2\pi\left(k - \frac{1}{2}\right)}{K} + \sigma\sqrt{-2\log\tau}.$$

In other words, we can view mapping to soft codes along with a suitable choice of τ and σ, having the effect of dividing the polar coordinate space into K overlapping sectors with width $2\sigma\sqrt{-2\log\tau}$. An example of four partitions is given in Figure 3.2. When $\bar{z}$ of an input x falls into the overlapping area of sectors i and j, we view it as contributing to the training of submodel i and j. Let γ be the overlapping ratio (normalized by 2π). σ can thus be determined by,

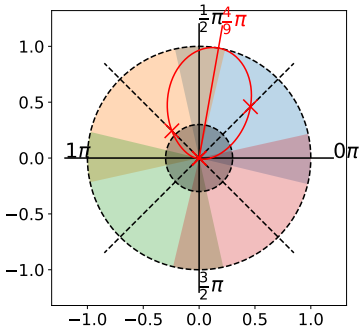
$$\sigma = \sqrt{-\frac{\pi^2(1+\gamma)^2}{2K^2\log\tau}} \tag{3.4.13}$$

In Figure 3.2, γ is set to 30% and τ equals to 0.3. When $\tilde{\theta}$ equals to $\frac{1}{3}\pi$, which is outside of the overlapping region (Figure 3.2a), the data point only contributes to the training of one submodel. When $\tilde{\theta}$ equals to $\frac{4}{9}\pi$, which is in between two midpoints $\frac{1}{4}\pi$

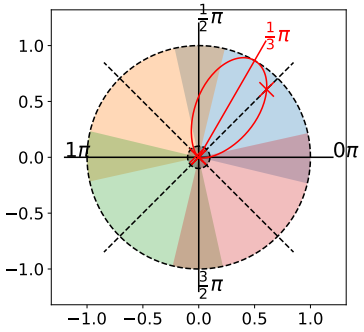
and $\frac{3}{4}\pi$, the data point contributes to the training of the two corresponding submodels.



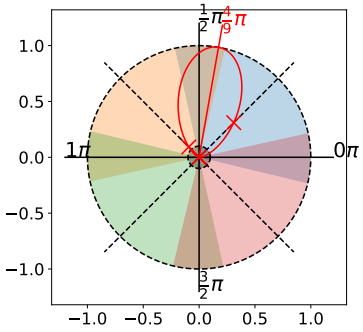
(a) Vector not in any overlap region (τ equals to 0.3)



(b) Vector in an overlap region (τ equals to 0.3)



(c) Vector not in any overlap region (τ equals to 0.1)



(d) Vector in an overlap region (τ equals to 0.1)

Figure 3.2: The red straight line denotes the angle $\tilde{\theta}$, with the red curve indicating the amount of decay from the maximum 1 to the minimum 0 while moving away from $\tilde{\theta}$. The red cross maker demonstrates a value on the midpoint, with in the brighter area telling it is above the selection threshold while in the darker area telling below the selection threshold.

3.4.3 Consideration of Model Uncertainty

The soft code $\bar{c}$ utilizes the angular proximity of input data in a 2D representation. However, partitioning based on the soft code alone does not always imply the trained

model is more specialized. The predictive uncertainty of a trained model with respect to the input data is also indicative of how much the model has "specialized" on the data. Intuitively, if a model is specialized on one partition of the input space, it should have a lower predictive uncertainty on the prediction of the data in the partition, but higher uncertainty on other data. In (Fang *et al.*, 2019), we found that the entropy computed from the softmax output of a neural network model is a good surrogate for the uncertainty of the model on the data. Formally, we denote $H(\hat{y}_k|x, \phi_k)$ the entropy of the k^{th} submodel parameterized by ϕ_k with respect to the input x,

$$H(\hat{y}_k|x, \phi_k) = -\sum_c p(\hat{y}_k = c|x, \phi_k) \log p(\hat{y}_k = c|x, \phi_k), \tag{3.4.14}$$

where $p(\hat{y}_k = c|x, \phi_k)$ is the predictive probability of output $c = 1, 2, ..., C$ for input x from submodel k.

To encourage the assignment of x to a submodel that has the lowest predictive uncertainty, we introduce a K-dimension vector $\bar{\bar{c}}$ as follows:

$$\bar{\bar{c}} = [\bar{\bar{c}}_1 \dots \bar{\bar{c}}_K], \quad \bar{\bar{c}}_k = \begin{cases} \tau & i = \arg\min_k H(\hat{y}_k|x, \phi_k) \\ 0 & otherwise. \end{cases} \tag{3.4.15}$$

Clearly, $\bar{\bar{c}}$ is a one-hot vector scaled by τ.

3.4.4 Partition of Input Data

To this end, we have obtained two K-dimension codes $\bar{c}$ and $\bar{\bar{c}}$ for each input data x. To decide the final partition of input data, we should take both into account. This

can done by a simple linear combination:

$$c = \alpha\bar{c} + (1-\alpha)\bar{\bar{c}}. \tag{3.4.16}$$

In the experiments, we set $\alpha = \frac{1}{2}$.

Let $\mathcal{P}(x) = \left\{k | c_k \geq \frac{\tau}{2}\right\}$ denote the indices of partitions (submodels) that input x contributes to. Clearly, $\mathcal{P}(x)$ cannot be an empty set since its respective $\bar{\bar{c}}$ contains one element that equals to τ. In the case that the cardinality of $\mathcal{P}(x)$ is greater than one, this implies that x will be used to train multiple submodels.

3.4.5 Neural Network Generator

The architecture of the generator network is illustrated in Figure 3.3. A neural network generator G takes an element k in $\mathcal{P}(x)$ (being converted to a one-hot vector) as input and generates the parameters ϕ_k of the kth submodel. CacheNet is agnostic to the target neural network architecture, which is decided by the target application. For example, for image classification, Shake-Shake (Gastaldi, 2017) has been shown to perform well across several datasets. Given K, we scale down the target neural network architecture to have reduced capacity.

Suppose $\hat{y}_k$ is the prediction of the kth submodel for x, noted by $\hat{y}_k = F(x; \phi_k)$. To avoid overfitting, we allow parameter sharing across the submodels. The proportion of parameters to be shared, the depth and the width of the shared networks are hyper-parameters to be determined by the neural network structure of the submodels. For an input data x and its label y, we first compute $\mathcal{P}(x)$. The cross-entropy loss for

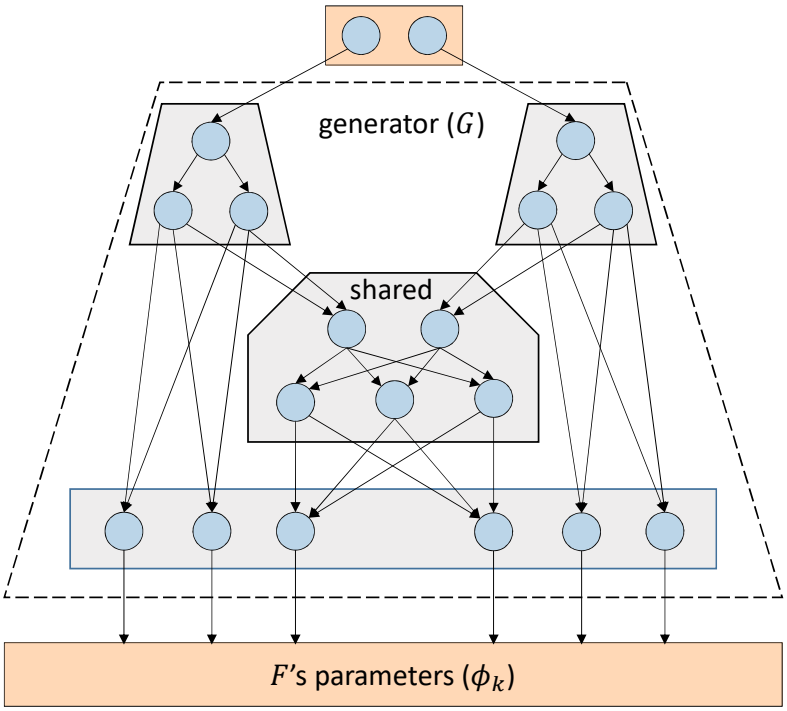


Figure 3.3: The generator G takes a one-hot vector δ_i as input and generates the parameters of the i^{th} partition. Values (either 0 or 1) of each dimension in δ_i are used to deactivate or activate a corresponding branch.

classification is given by,

$$J_F(x, y) = \sum_{k \in \mathcal{P}(x)} H(\hat{y}_k, y) \tag{3.4.17}$$

Minimizing $E_{p_D(x)} J_F(x, y)$ leads to a more accurate prediction with respect to the dataset.

3.4.6 Training Algorithm

In CachNet, there are three networks that need to be trained, namely, the stacked encoder, the stacked decoder and the generator network. Since the output of the

stacked encoder contributes to the input of the generator network, they need to be trained jointly.

The lower-dimension representation $\bar{z}$ is the most informative of a particular input x if two lower bounds $\mathcal{L}^*(\xi, \psi)$ and $\bar{\mathcal{L}}^*(\bar{\xi}, \bar{\psi})$ are maximized, and a submodel's predictions are the most accurate if $E_{p_D(x)} J_F(x, y)$ is minimized. Thus, the minimization objective J should be $E_{p_D(x)} J_F(x, y)$ added to the negation of $\mathcal{L}^*(\xi, \psi)$ and $\bar{\mathcal{L}}^*(\bar{\xi}, \bar{\psi})$:

$$J = E_{p_D(x)} J_F(x, y) - \mathcal{L}^*(\xi, \psi) - \bar{\mathcal{L}}^*(\bar{\xi}, \bar{\psi}). \tag{3.4.18}$$

To better converge, $E_{p_D(x)} J_F(x, y)$, $\mathcal{L}^*(\xi, \psi)$, and $\bar{\mathcal{L}}^*(\bar{\xi}, \bar{\psi})$ are optimized stage-wisely and batch-wisely. Let $J^{(i)}$ be J with respect to a batch $(X^{(i)}, Y^{(i)})$ drawn from the dataset D. Suppose the generator G is parameterized by χ, and κ is the set of $\{\xi, \psi, \bar{\xi}, \bar{\psi}, \chi\}$. The training algorithm should iteratively apply gradient updates to κ (or χ) with respect to the loss function $J^{(i)}$ and descend to a minimum of J (as shown in Algorithm 4).

Algorithm 4 Training CacheNet

▷ let η be the learning rate
▷ let ν be the epoch stopping gradient updates in $\xi, \psi, \bar{\xi}, \bar{\psi}$
procedure TRAIN(η, ν)
  while $J^{(i)}$ is decreasing **do**
    draw the next batch $(X^{(i)}, Y^{(i)})$ from D
    if #epoch $< \nu$ **then**
      $\kappa \leftarrow \kappa - \eta \nabla_\kappa J^{(i)}$
    else
      $\chi \leftarrow \chi - \eta \nabla_\chi J^{(i)}$
    end if
  end while
end procedure

3.5 CacheNet Inference

With CacheNet, inference on end devices is accelerated by caching submodels of lower computation complexity. Depending on storage availability, one or multiple submodels can be stored on end devices. At any time, only one submodel is *active* and is used to make predictions. Given an input data sample x, the active submodel k outputs $\hat{y}$, the label of x and the predictive entropy $H(\hat{y}|x, \phi_k)$. If the entropy is above a certain threshold, $\hat{y}$ will be returned. Otherwise, two situations may arise, i) x is better handled by another cached submodel, and ii) x is better handled by a submodel not in cache. The latter case is called a *cache miss*. Like caching in memory hierarchy, CacheNet needs to handle cache misses by replacing a cached "item" (model). However, unique to CacheNet, the newly cached "item" is not the input data but a suitable model.

3.5.1 Submodel Selection

In Section 3.4, a K-dimension code $\bar{c}$ is computed for each input data sample using S-InfoVAE and the subsequent mapping in polar coordinates. In the training stage, $\bar{c}$ contributes to the input to the generator network that generates the parameters of respective submodels. In the inference stage, $\bar{c}$ can be used to select the submodel to make prediction given an input data sample. In particular, the joint optimization of S-InfoVAE, generator network and submodels aligns the output of S-InfoVAE with the submodel that has lowest predictive uncertainty. Thus, we can simply select the submodel whose index corresponds to the largest element in $\bar{c}$. Note in the inference stage, we do not need to calculate the predictive uncertainty for each submodel. Instead, only one submodel is applied. This is one of the key differences between

CacheNet and the work in (Fang *et al.*, 2019). S-InfoVAE can be executed on the end device or on the edge server. In the former case, extra storage and computation overhead are introduced. In the latter case, submodel storage and selection are delegated to the edge server.

3.5.2 Cache Replacement

When the predictive entropy is below a preconfigured threshold using the active submodel, the input data x is sent to the edge server, which will perform inference on behalf of the end device. Additionally, by submodel selection, the edge server determines a suitable model for x. A cache miss occurs on the end device. The newly selected submodel will be downloaded to the device to replace an existing model. Here, we adopt the Least Recently Used (LRU) policy and select the model that is least recently used. By the virtue of LRU, such a policy does not suffer from Bélády's anomaly. In other words, as the cache size increases, the cache miss rate does not increase. More detailed proof can be found in Appredix 3.8.

3.6 Evaluation

In this section, we evaluate CacheNet with two different real-world datasets (the CIFAR-10 Krizhevsky *et al.* (2009) and the Frontal View Gait (FVG) dataset Zhang *et al.* (2019)), and test CacheNet's performance with respectively two different neural network models (Shake-Shake Gastaldi (2017) and ResNet He *et al.* (2016)).

3.6.1 Datasets

CIFAR-10 CIFAR-10 Krizhevsky *et al.* (2009) is a benchmark dataset for image classification, comprised of $60,000$, 32×32 colored images and 10 classes (such as automobile, bird and horse) in total. Although CIFAR-10 is not a video dataset and is an image classification dataset, image classification is still a valid scenario if it is in a video processing pipeline (e.g. where the background has been removed previously from the video). In this case, temporal locality still applies while consecutive images would be less redundant owing to the earlier steps in the pipeline. For example, a horse (possibly shot in different angles with different scales) in the video is still likely to appear multiple times in the sequence, even when the background has been removed (e.g. object detection).

For fair evaluation, test images are not supposed to be seen during training. Thus, we set aside $10,000$ images for testing. To simulate temporal locality in a video pipeline, the synthesized image sequence in testing is composed of a sample of the $10,000$ images in the way that images with the same label are concatenated together.

To reduce overfitting, data augmentation techniques are used, including: 1) random cropping and 2) random flipping. Apart from data augmentation, Shake-Shake regularization has been applied to reduce overfitting Gastaldi (2017), and batch normalization to reduce internal covariate shift Ioffe and Szegedy (2015).

FVG FVG is a person re-identification dataset, first introduced in Zhang *et al.* (2019), as a collection of frontal walking videos from 226 subjects. In total it contains $2,856$ videos at 15 frames per second with a resolution of 1920×1080.

In contrast to other person re-identification datasets in surveillance settings, FVG

is the first to focus on the frontal view. This makes it useful for two reasons: (i) It contains temporal locality in the form of a fixed background and the same subject walking towards the camera, which can be leveraged for caching. (ii) Having a frontal view means that it contains minimal gait cues.

To reduce the chance of overfitting and improve generalization ability we use data augmentation techniques Shorten and Khoshgoftaar (2019) on this dataset as well. We first oversample the images by interpolating between existing frames. This technique preserves the extrinsic distribution while allowing us to experiment with cache performance by varying the degree of temporal locality. Additionally, in the original dataset the average frame rate of each video is 15 frames per second. That is only half of the frame rate of a HD video (generally 30-60 frames per second). Since each video sample is of the subject walking straight towards the camera from a distance, it contains intrinsic depth information that can be utilized to synthesize intermediate frames. As such, we use DAIN Bao *et al.* (2019), a state of the art approach that leverages the depth information to interpolate between the frames.

3.6.2 Experimental Setup

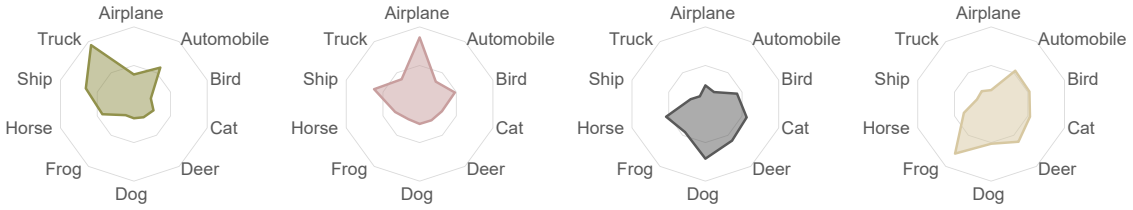
CacheNet's performance is evaluated on two different datasets (CIFAR-10 and FVG), three end devices (Jetson TX2, Jetson Nano, and Raspberry Pi 4) and two deep learning frameworks (NCNN and TensorFlow Lite). There are two baselines to compare with: a) running a full model (Shake-Shake-26 or ResNet-50) on an end devices (*Device*), and b) offloading the full model onto an edge server (*Edge*). Different thresholds are evaluated to better trade off hit rate against accuracy: for CIFAR-10, they are 0.5, 0.6, 0.7, 0.75, and 0.8; for FVG, they are 1.5, 2.0, 2.3, 2.5, and 2.7. (Here,

A larger threshold in FVG is caused by more classes (neurons) at the output layers.) Furthermore, on the FVG dataset, we evaluate two video frame rates 15 FPS and 30 FPS (at inference) with both trained at 60 FPS (by using data augmentation).

The number of submodels K is set to 4 in the experiment. For a possible convergence, CacheNet is trained on TensorFlow with 4 NVIDIA 1080TI graphic cards. Per CIFAR-10, CacheNet partitions Shake-Shake-26 (with 26 layers) into 4 Shake-Shake-8 (with 8 layers) neural network submodels for caching; per FVG, CacheNet partitions ResNet-50 into 4 ResNet-20 (but with fewer channels per layer).

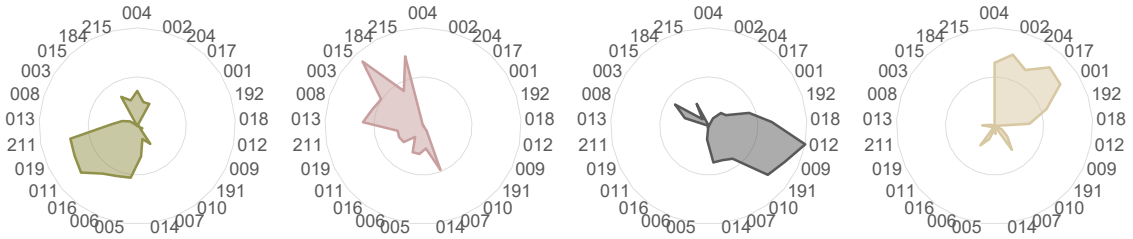
CacheNet's inference is distributed between the edge server and the end device in the experiment. One submodel is cached and runs on the end device, while submodel storage and selection are delegated to the edge server. End devices are evaluated with limited storage to mimic that of end devices such as security cameras. One Intel Xeon CPU core is enabled on the edge server to representatively simulate those of most of WiFi access points (e.g. a 500 megahertz MIPS processor on the Arlo SmartHub) with generally limited compute power. There is sufficient storage on the edge server comparable to that of WiFi access points (e.g. a 128 gigabyte SD card on the eufy HomeBase and a 2 terabyte USB hard drive onto the Arlo SmartHub). End devices are connected to the edge server through a WiFi router, via WiFi 5G (802.11ac) and an Ethernet cable, respectively.

TensorFlow submodels from training were converted to NCNN and TensorFlow Lite submodels and stored on the edge server. Whenever a submodel is needed, the end device initiates an HTTP/1.1 request to the edge server, and then the chosen submodel on the edge server is encoded in an HTTP/1.1 and protobuf message then sent back to the end device. OpenCV is also used in the experiment to read a testing



(a) Partition A (b) Partition B (c) Partition C (d) Partition D

Figure 3.4: For CIFAR-10, partition A is more specialized in trucks and automobiles; partition B can predict airplanes and ships better; partition C is more certain of the horse, dog, and cat classes; partition D knows more about frogs and deer.



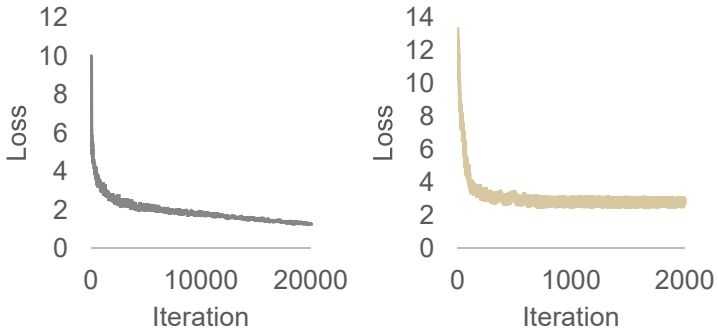
(a) Partition A (b) Partition B (c) Partition C (d) Partition D

Figure 3.5: For FVG, partition A is more certain of person identifier (PID) 211, 019, 011, 016, 006, and 005; partition B is specialized in PID 013, 008, 003, 015, and 215; partition C knows more about PID 010, 191, 009, 012, and 018; partition D is more certain of PID 004, 002, 204, 017, and 001.

image sequence (video) into the memory and convert them into tensors.

3.6.3 Results

Specialization Specialization is crucial for caching because a non-specialized partition cannot match the full model's performance by any chance even for a smaller subset of input. There are two aspects we would investigate: (a) whether similar input images are mapped to the same partition; (b) whether input images are partitioned roughly evenly to fully utilize the capacities of all submodels, considering both CIFAR-10 and FVG are approximately balanced datasets.

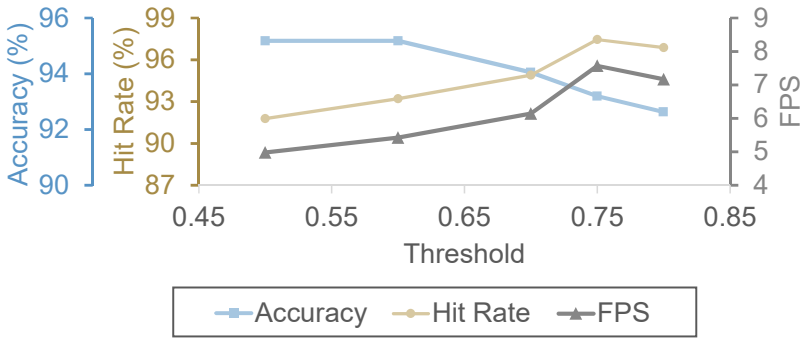


(a) CIFAR-10 (b) FVG

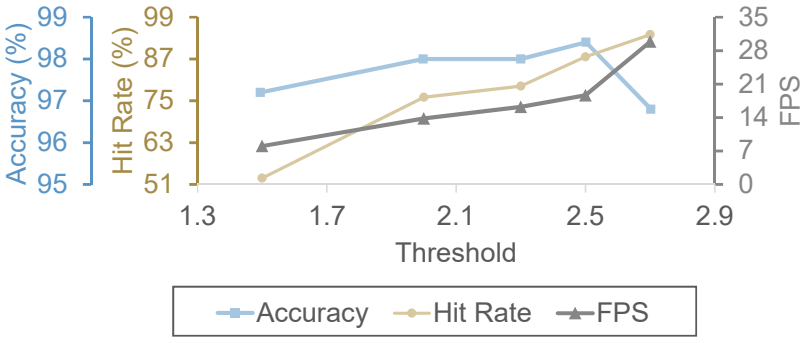
Figure 3.6: CIFAR-10's and FVG's losses both start high but converge closer and closer to zero.

Figure 3.4 and 3.5 illustrate the number of input images per class being mapped (by S-InfoVAE) to a particular partition. They answer most of our concerns: (a) A partition roughly covers most of similar input images from the same class. e.g. for CIFAR-10, partition A is more specialized in trucks and automobiles; partition B knows better airplanes and ships; for FVG, partition A is more certain of person identifier (PID) 211, 019, 011, 016, 006, and 005; and partition B is specialized in PID 013, 008, 003, 015, and 215. (b) In both cases of CIFAR-10 or FVG, the areas (Figure 3.4 and 3.5) that partitions occupy are roughly even. It implies the total number of (image) instances they span are approximately the same.

Convergence Not all neural networks converge. Thus, whether CacheNet is useful depends on whether it converges or not per the particular dataset. In CIFAR-10 and FVG, we can see (Figure 3.6) that their losses both start high but converge closer and closer to zero. Since FVG is a smaller dataset compared to CIFAR-10, CacheNet with FVG converges faster (in fewer iterations) than CIFAR-10.



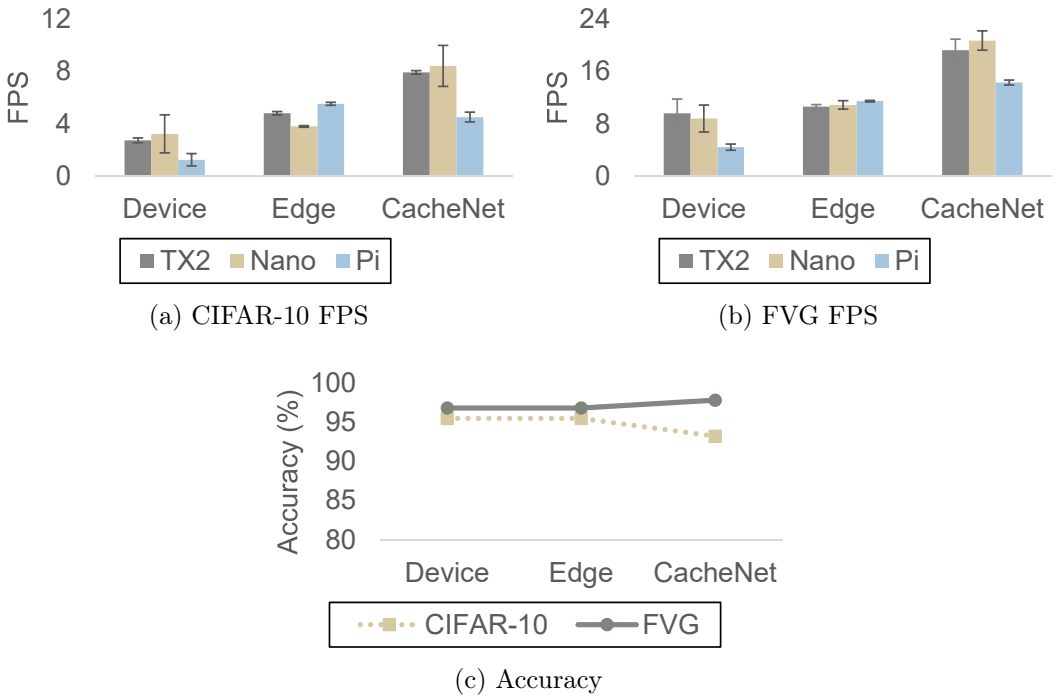
(a) CIFAR-10



(b) FVG

Figure 3.7: FPS and hit rate increase most of the time as the preconfigured threshold increases. Accuracy generally decreases because predictions of less certainty are considered valid. When multiple submodels outperforming the full model (in the FVG dataset), there is a small peak observed before the accuracy declines.

Cache replacement As it is discussed in Section 3.5.2, if the predictive entropy is below a preconfigured threshold, the inference is performed locally; otherwise, it is done remotely on the edge server. Figure 3.7a and 3.7b demonstrate that the FPS increases as the threshold increased most of the time for both CIFAR-10 and FVG. The reason is that the hit rate is generally higher when the threshold is higher. Fewer cache replacement is needed and more and more images are being processed locally, which speeds up the inference. On the other hand, Figure 3.7a and 3.7b show that a



(a) CIFAR-10 FPS

(b) FVG FPS

(c) Accuracy

Figure 3.8: Medians are taken and standard deviations are plotted as error bars. CacheNet is faster than the other two baselines, while accuracy is comparable to the full model.

higher hit rate generally comes at the cost of lower accuracy. It is because a higher threshold allows prediction with higher entropy (uncertainty) to become valid. Higher entropy predictions are of lower quality that decrease the overall accuracy. We find that in practice, it is a trade-off between hit rate and accuracy.

Comparison to baselines A comparison between CacheNet and the other two baselines (*Device* and *Edge*) are shown in Figure 3.8a and 3.8b. Medians (of all the scenarios) are taken and standard deviations are plotted as error bars in those figures. For CacheNet, preconfigured threshold 0.75 and 2.5 are chosen respectively per CIFAR-10 and FVG to the best extend to trade off hit rate against accuracy. As

visualized on those figures, CacheNet is much faster than the other two baselines: for CIFAR-10, 3.2X of *Device* and 1.6X of *Edge*; for FVG, 2.5X of *Device* and 1.7X of *Edge*. At the same time, the accuracy of CacheNet is comparable with that of the full model, with only a slight drop on CIFAR-10, but increasing a bit on FVG.

More details are given in Table 3.1–3.6. CacheNet generally works better on end devices with more computing power such as Jetson TX2 and Jetson Nano. Offloading to the edge server (*Edge*) releases end devices' burden thus CPU usages are lowest among three. However, it also implies that the computing power on the end device has not been fully utilized. Memory usages fall into a similar pattern as that of CPU usages. If we divide elapsed time into the time that is run on the end device and that is performed on the edge server (including time for upload and download), we observe that CacheNet distributes the total (computation) time between the end device and the edge server, while the other two baselines are not taking the advantages of distributed computing, that either runs locally (*Device*) or computes on the edge server most of the time (*Edge*).

Comparison across frameworks NCNN and TensorFlow Lite are both lightweight deep learning framework tailored for embedded devices with limited compute power, memory and storage. A comparison between TensorFlow Lite and NCNN are given in Table 3.1–3.6. CacheNet with NCNN and TensorFlow Lite both outperform the baselines. NCNN is slightly more efficient than TensorFlow Lite for both CIFAR-10 and FVG, while TensorFlow Lite consumes far less memory than NCNN.

Comparison across devices From Figure 3.8, we observe that CacheNet performs better on end devices with higher compute power such as Jetson TX2 and Jetson

Table 3.1: Experimental Results with CIFAR-10 on Jetson TX2, Jetson Nano, and Raspberry Pi 4 - NCNN

	Jetson TX2			**Jetson Nano**			**Raspberry Pi 4**		
	Device	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***
FPS	2.85	4.89	**8.02**	4.25	3.83	**9.53**	1.57	**5.60**	4.77
Accuracy (%)	95.47	95.47	93.20	95.47	95.47	93.20	95.47	95.47	93.20
CPU (%)	84.53	4.25	60.26	96.83	5.96	57.23	98.65	1.21	63.75
Memory (Mb)	610.71	1.86	198.76	863.14	1.94	241.80	875.53	0.91	201.75
Time (s)	124.06	72.16	**44.03**	83.06	92.20	**37.04**	224.15	**63.02**	74.03
├**Device (s)**	124.06	0.80	26.25	83.06	0.66	17.89	224.15	0.63	42.28
└**Edge (s)**	0.00	71.37	17.79	0.00	91.54	19.14	0.00	62.39	31.75

Nano. Raspberry Pi incurs more time on submodel inference, which leads to lower FPS. Detailed numerical comparisons can be found in Table 3.1–3.6.

3.7 Conclusion

In this paper, we proposed CacheNet, a neural network model caching mechanism for edge computing. In CacheNet, an edge (cloud) server is responsible for the storage and selection of neural network partitions, while an end device with a cached partition performs inferencing most of the time.

Three key features enable CacheNet to achieve short end-to-end latency without much compromise in prediction accuracy: 1) Caching avoids the communication latency between an end device and edge (cloud) server whenever there is a cache hit; 2) specialized cached partitions do not sacrifice prediction accuracy if properly trained and selected; 3) the computation and storage complexities of cached model partitions are smaller rather than those of a full model.

Table 3.2: Experimental Results with CIFAR-10 on Jetson TX2, Jetson Nano, and Raspberry Pi 4 - TensorFlow Lite

	Jetson TX2			**Jetson Nano**			**Raspberry Pi 4**		
	Device	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***
FPS	2.59	4.71	**7.83**	2.19	3.74	**7.31**	0.90	**5.44**	4.24
Accuracy (%)	95.47	95.47	93.20	95.47	95.47	93.20	95.47	95.47	93.20
CPU (%)	77.26	4.40	52.37	79.92	5.90	56.82	74.29	1.66	53.45
Memory (Mb)	213.42	29.93	113.96	226.37	108.73	133.23	210.12	106.98	99.98
Time (s)	136.05	74.91	**45.08**	161.04	94.29	**48.30**	390.31	**64.90**	83.25
├Device (s)	136.05	0.56	29.00	161.04	0.83	34.16	390.31	0.55	60.36
└Edge (s)	0.00	74.35	16.07	0.00	93.46	14.13	0.00	64.35	22.89

In future works, we plan to experiment with more datasets and neural network models using CacheNet. The two-level caching idea can be further extended to consider a hierarchy of caches, e.g., distributed among end devices, edge nodes and cloud servers. Another line of research is to apply neural architecture search to CacheNet to improve its adaptability to different types of neural networks.

3.8 Appendix A: Absence of Bélády's Anomaly

Bélády's anomaly is the phenomenon that a larger cache incurs more cache misses than a smaller one. In CacheNet, there are two possible ways to take advantage of a larger cache size: 1) each individual submodel being cached has a larger capacity (i.e., deeper); 2) more submodels are being cached on an end device. If both do not result in fewer cache hits, we can conclude that Bélády's anomaly does not occur in CacheNet.

Table 3.3: Experimental Results with FVG (15 FPS) on Jetson TX2, Jetson Nano, and Raspberry Pi 4 - NCNN

	Jetson TX2			**Jetson Nano**			**Raspberry Pi 4**		
	Device	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***
FPS	11.36	10.40	**20.80**	10.41	10.40	**22.70**	5.10	11.36	**14.70**
Accuracy (%)	97.20	97.20	98.40	97.20	97.20	98.40	97.20	97.20	98.40
CPU (%)	96.05	8.27	22.35	95.22	11.54	23.37	96.32	4.23	24.37
Memory (Mb)	312.35	7.19	10.55	436.91	7.75	11.57	454.79	6.72	8.04
Time (s)	22.01	24.05	**12.02**	24.02	24.04	**11.01**	49.04	22.01	**17.01**
├Device (s)	22.01	0.72	2.31	24.02	0.87	1.70	49.04	0.48	4.62
└Edge (s)	0.00	23.32	9.71	0.00	23.17	9.31	0.00	21.53	12.38

3.8.1 Larger Capacity

A submodel with a larger capacity is defined as follows. Given any sequence $X = x_1, x_2, \ldots, x_N$ of images, audio clips etc. Let $\Phi = \phi^{(1)}, \phi^{(2)}, \ldots, \phi^{(Q)}$ be an sequence of submodel instances for caching, with respect to 1) their depths $d^{(1)} < d^{(2)} < \ldots < d^{(Q)}$, 2) any layer in $\phi^{(1)}$ contained by $\phi^{(2)}$, ..., and any layer in $\phi^{(Q-1)}$ contained by $\phi^{(Q)}$. According to the capacity theorem Cohen *et al.* (2016), submodel instance $\phi^{(1)}$ expresses less functions than $\phi^{(2)}$, ..., and $\phi^{(Q-1)}$ less functions than $\phi^{(Q)}$.

Let $H(\hat{y}|x_i, \phi^{(j)})$ be the predictive entropy given any input $x_i, i = 1, 2, \ldots, N$ and any submodel instance $\phi^{(j)}, j = 1, 2, \ldots, Q$. For a predefined threshold T, if $H(\hat{y}|x_i, \phi^{(j)}) < T$, we say it is a cache hit, else it is a cache miss.

Theorem 1. *Let $M(X, \phi^{(j)})$ be the number of misses (faults) given the input sequence X and the submodel instance $\phi^{(j)}, j = 1, 2, \ldots, Q$. Then $M(X, \phi^{(1)}) \geq M(X, \phi^{(2)}) \geq \ldots \geq M(X, \phi^{(Q)})$*

Proof. We can prove this theorem by induction.

Table 3.4: Experimental Results with FVG (15 FPS) on Jetson TX2, Jetson Nano, and Raspberry Pi 4 - TensorFlow Lite

	Jetson TX2			**Jetson Nano**			**Raspberry Pi 4**		
	Device	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***
FPS	7.65	10.72	**20.65**	7.01	10.65	**21.47**	3.75	10.71	**16.02**
Accuracy (%)	97.20	97.20	98.40	97.20	97.20	98.40	97.20	97.20	98.40
CPU (%)	69.51	8.63	18.31	72.55	10.77	21.11	62.75	4.89	18.32
Memory (Mb)	194.68	10.61	16.16	181.79	99.08	16.95	190.74	96.64	10.42
Time (s)	32.70	23.33	**12.11**	35.67	23.48	**11.64**	66.65	23.33	**15.60**
├Device (s)	32.70	0.49	2.42	35.67	0.74	2.87	66.65	0.80	4.59
└Edge (s)	0.00	22.84	9.68	0.00	22.74	8.78	0.00	22.53	11.01

1) Base case: if $X = x_1$, both $\phi^{(j)}$ and $\phi^{(j+1)}$ incurs a cache miss on x_1, thus, $M(X, \phi^{(j)}) = M(X, \phi^{(j+1)})$

2) Induction hypothesis: we need to show if $X = x_1, \ldots, x_i$, $M(X, \phi^{(j)}) \geq M(X, \phi^{(j+1)})$ for an arbitrary j, when $X = x_1, \ldots, x_{i+1}$, $M(X, \phi^{(j)}) \geq M(X, \phi^{(j+1)})$ also holds.

a) If the newly input x_{i+1} incurs a cache hit on the submodel instance $\phi^{(j)}$, there should be also a cache hit on $\phi^{(j+1)}$. This claim relies on the capacity theorem Cohen *et al.* (2016) that the submodel instance $\phi^{(j+1)}$ has more functional expressibility than $\phi^{(j)}$. By definition, the submodel instance $\phi^{(j)}$ can be embedded in $\phi^{(j+1)}$. The submodel instance $\phi^{(j+1)}$'s additional layers can be made as an identity for $x_1, \ldots, x_{i+1}$'s intermediate outputs. Thus, the claim holds.

b) If the new input x_{i+1} incurs a cache miss on $\phi^{(j)}$, there may be a cache hit or cache miss on $\phi^{(j+1)}$. Since the submodel instance $\phi^{(j)}$ is embedded in $\phi^{(j+1)}$, and $\phi^{(j+1)}$'s additional layers are made as an identity for $x_1, \ldots, x_i$'s intermediate outputs.

Table 3.5: Experimental Results with FVG (30 FPS) on Jetson TX2, Jetson Nano, and Raspberry Pi 4 - NCNN

	Jetson TX2			**Jetson Nano**			**Raspberry Pi 4**		
	Device	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***
FPS	11.62	11.09	**17.84**	10.86	11.88	**19.98**	5.05	11.62	**13.89**
Accuracy (%)	96.40	96.40	97.20	96.40	96.40	97.20	96.40	96.40	97.20
CPU (%)	96.94	8.46	22.78	96.91	11.84	24.05	98.10	4.43	25.30
Memory (Mb)	310.16	12.76	9.00	455.06	12.79	11.52	454.23	11.48	9.35
Time (s)	43.02	45.08	**28.03**	46.03	42.07	**25.03**	99.04	43.01	**36.01**
├Device (s)	43.02	0.87	3.77	46.03	0.65	3.21	99.04	0.22	8.52
└Edge (s)	0.00	44.21	24.26	0.00	41.42	21.82	0.00	42.79	27.49

The additional layers of $\phi^{(j+1)}$ may have the additional capacity to represent x_{i+1}'s function.

In either case, $M(X, \phi^{(j)}) \geq M(X, \phi^{(j+1)})$ for an arbitrary j. The induction hypothesis holds. □

3.8.2 More Submodels

When there are multiple submodels to cache on an end device, a cache miss happens if the predictive entropy of the current submodel is less than the threshold T and there is no suitable submodel (which is decided by the S-InfoVAE on the end device) currently stored on the end device.

Theorem 2. *Let k ($1 \leq k \leq K$) be the number of submodels cached on an end device. Let $\bar{M}(X, k)$ be the number of misses (faults) given the input sequence X. Then, under the LRU cache replacement policy, $\bar{M}(X, 1) \geq \bar{M}(X, 2) \geq \cdots \geq \bar{M}(X, K)$.*

Proof. We can prove this theorem by induction.

Table 3.6: Experimental Results with FVG (30 FPS) on Jetson TX2, Jetson Nano, and Raspberry Pi 4 - TensorFlow Lite

	Jetson TX2			**Jetson Nano**			**Raspberry Pi 4**		
	Device	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***	***Device***	***Edge***	***CacheNet***
FPS	7.82	10.49	**17.76**	7.15	11.08	**19.43**	3.43	11.53	**13.49**
Accuracy (%)	96.40	96.40	97.20	96.40	96.40	97.20	96.40	96.40	97.20
CPU (%)	69.98	8.60	19.89	73.13	11.52	21.24	63.68	4.52	17.17
Memory (Mb)	197.03	10.34	16.64	189.10	99.05	16.09	191.69	6.79	11.23
Time (s)	63.90	47.67	**28.15**	69.89	45.13	**25.73**	145.61	43.35	**37.06**
├Device (s)	63.90	0.69	4.97	69.89	0.99	5.57	145.61	0.19	9.29
└Edge (s)	0.00	46.98	23.18	0.00	44.14	20.16	0.00	43.16	27.78

1) Base case: if $X = x_1$, both k and $k+1$ cached submodels incur a cache miss on x_1, thus, $\bar{M}(X,k) = \bar{M}(X,k+1)$

2) Induction hypothesis: we need to show if $X = x_1, \ldots, x_i$, $\bar{M}(X,k) \geq \bar{M}(X,k+1)$ for an arbitrary k, when $X = x_1, \ldots, x_{i+1}$, $\bar{M}(X,k) \geq \bar{M}(X,k+1)$ also holds.

a) If the newly input x_{i+1} incurs a cache hit on k cached submodels, there should be also a cache hit on $k+1$ cached submodels, because the k cached submodels are always embedded in the $k+1$ submodels under the least recently used (LRU) policy.

b) If the newly input x_{i+1} incurs a cache miss on k cached submodels, there may be a cache hit or cache miss on $k+1$ cached submodels, because the k submodels are embedded in the $k+1$ submodels, the one more submodel in the cached $k+1$ submodels may cause the hit or not depending on whether it matches the index given by S-InfoVAE.

No matter in either case, $\bar{M}(X,k) \geq \bar{M}(X,k+1)$ for an arbitrary k. The induction hypothesis holds. Thus, the theorem holds. □

To this end, we conclude when individual submodels have larger capacity or more submodels can be cached on an end device, CacheNet always has higher or the same hit rates. In other words, it does not suffer from Bélády's anomaly.

Chapter 4

Logographic Subword Model: Compression for Machine Translation

This chapter is reproduced from "Logographic Subword Model for Neural Machine Translation", Yihao Fang, Rong Zheng, and Xiaodan Zhu, published in International Conference on Computational Linguistics and Machine Translation (ICCLMT), Tokyo, Japan, 2019. The author of this thesis is the first author and the main contributor of this publication.

Abstract

A logographic subword model is proposed to reinterpret logograms as abstract subwords for neural machine translation. Our approach drastically reduces the size of an artificial neural network, while maintaining comparable BLEU scores as those attained with the baseline RNN and CNN seq2seq models. The smaller model size also leads to shorter training and inference time. Experiments demonstrate that in the tasks of English-Chinese/Chinese-English translation, the reduction of those aspects can be from 11% to as high as 77%. Compared to previous subword models, abstract subwords can be applied to various logographic languages. Considering most of the logographic languages are ancient and very low resource languages, these advantages are very desirable for archaeological computational linguistic applications such as a resource-limited offline hand-held Demotic-English translator.

4.1 Introduction

The sequence-to-sequence (seq2seq) models have been widely adopted in machine translation tasks. There are two important types of seq2seq models: the recurrent neural network (RNN) based seq2seq models (Cho *et al.*, 2014; Sutskever *et al.*, 2014) and the convolutional neural network (CNN) based sequence-to-sequence models (Gehring *et al.*, 2017). Both types of seq2seq models have gained extensive attention and motivated many efforts to make them faster, smaller, and more accurate, such as tied embedding (Press and Wolf, 2016), layer normalisation (Ba *et al.*, 2016), weight normalisation (Salimans and Kingma, 2016), and subword byte pair encoding (BPE) (Sennrich *et al.*, 2015; Gage, 1994), among others. Compared to the RNN seq2seq

models, the CNN seq2seq models substitute RNN components with CNN and allow much faster training while retaining the BLEU scores closely comparable to those obtained with RNN seq2seq models.

Despite the success of seq2seq models in machine translation, their high computing complexity still strongly limits their applications such as those running on offline hand-held translators. An efficient approach to reducing models' complexity is compacting their output layers. A cumbersome output layer significantly increases the number of parameters in the model and consequently slows down model inference. Furthermore, the amount of gradient calculation in training grows as the number of model parameters increases. In machine translation, the size of the output layer is often proportional to the size of the target dictionary. For instance, if there are one million words in the target dictionary, 64 examples in a batch, and 50 words in a sentence, then there are $64 \times 1M$ units in the output layer of each RNN decoder cell, and $64 \times 50 \times 1M$ units in the output layer of a CNN decoder. Compacting the target dictionary consequently reduces the model size and speeds up model training and inference.

It is non-trivial to design a new approach that can compact the target dictionary and is directly applicable to different logographic languages without sacrificing performance. In this paper, we propose a logographic subword model that represents logograms as multiple "abstract subwords" (code symbols), with an encoder and decoder transforming logograms to abstract subwords and subwords to logograms. The encoder quantizes and decomposes the embeddings of logograms to multiple abstract subwords (code symbols). Word embedding or equivalent vector representation of words (Pennington *et al.*, 2014; Mikolov *et al.*, 2013) preserves the closeness between word pairs through their distances in the vector space. Quantization identifies common semantic components (abstract

subwords) among logograms. Two quantizers are examined in the experiments: the state of the art locally optimized product quantization (LOPQ) (Kalantidis and Avrithis, 2014) and a novel density aware product quantization (DAPQ). Quantizing embeddings of logograms helps logograms with close meanings be more likely to share common abstract subwords in one or more dimensions. By sharing, the number of abstract subwords can be significantly reduced in the target directory. Furthermore, only infrequent words are decomposed into code symbols. Taking word frequency into consideration avoids unnecessary elongation of the source and target sentences.

Using the proposed model, the sizes of RNN and CNN sequence-to-sequence models are reduced by 37% and 77% respectively in an English-to-Chinese translation task without sacrificing performance. The training times are about 11% and 73% shorter; the inference time is nearly halved in RNN and 36% shorter in CNN.

Considering many of the logographic languages are ancient and low resource languages, these advantages are also desirable for archaeological applications. Also, reduction in model sizes is useful for resource-limited applications such as those running on hand-held devices. Furthermore, while we discuss the proposed models in the context of translation, the methods have implications for other tasks of NLP involving predicting tokens, e.g., language modeling, summarization, and image captioning.

The rest of the paper is organized as follows. Section 4.2 describes the related work in neural machine translation. The proposed approach is presented in Section 4.3. Section 4.4 provides more details about product quantization and the proposed density aware approach. In Section 4.5, we discuss more the encoding algorithm. The experimental setups and experimental results are given in Section 4.6, and conclusions and future work are discussed in Section 4.7.

4.2 Related Work

Cho et al (Cho *et al.*, 2014) first proposed the RNN seq2seq model by modeling it as an RNN encoder-decoder architecture, with the encoder transforming an input sentence into a context vector and the decoder mapping the context vector to an output sentence (the translation hypotheses). Bahdanau et al. (Bahdanau *et al.*, 2014) further improved RNN seq2seq models by making the encoder as a bidirectional gate recurrent unit (GRU) and binding the attentional mechanism to the decoder. To avoid overfitting in RNN seq2seq models, Gal and Ghahramani (Gal and Ghahramani, 2016) proposed to apply the variational inference based dropout technique to the model. To speed up convergence in training, Ba and his colleagues (Ba *et al.*, 2016) introduced layer normalization to stabilize state dynamics in RNNs. Salimans and Kingma (Salimans and Kingma, 2016) proposed weight normalization that reparameterizes weight vectors from their direction. To increase the model depth, Zhou et al. (Zhou *et al.*, 2016) proposed fast-forward connections where the shortest paths do not depend on any recurrent calculations. Wu et al. (Wu *et al.*, 2016b) introduced the bidirectional stacked encoder and Barone and his colleague (Barone *et al.*, 2017) proposed a BiDeep RNN by replacing the GRU cells of a stacked encoder with multi-layer transition cells. Deviating from RNN seq2seq models, Gehring et al. (Gehring *et al.*, 2017) proposed convolutional seq2seq model where the encoder and decoder were fully replaced by convolutional neural networks (CNN). Their approach allows much faster training while retaining the BLEU scores closely comparable to those obtained with RNN seq2seq models.

Embedding Sennrich and Haddow (Sennrich and Haddow, 2016) generalized the embedding layer to support linguistic features such as morphological features, part-of-speech tags, and syntactic dependency labels. Press and Wolf (Press and Wolf, 2016) proposed tier embedding and argued that weight tying reduces the size of neural translation models. However, no attempt has been made to reduce the size of the target dictionary through word embedding. Significant reduction in model complexity is expected considering words are on the order of hundreds of thousands or more in a typical dictionary.

Decomposition Sennrich et al. (Sennrich *et al.*, 2015) proposed to segment words of source and target sentences into smaller subword units using byte pair encoding (BPE) compression (Gage, 1994). They showed an improvement in the BLEU scores of 1.1 and 1.3 for English-German and English-Russian translations, respectively. Despite its advantage, BPE splits an alphabetic word to multiple letter groups, and thus it is intrinsically not applicable to logographic languages such as Chinese, Chorti, and Demotic (Ancient Egyptian) where a word is a glyph rather than alphabetic letters. García-Martínez et al. (García-Martínez *et al.*, 2016) proposed to decompose words morphologically and grammatically into factored representations such as lemmas, part-of-speech tag, tense, person, gender, and number. Their approach reduced training time and out of vocabulary (OOV) rates with improved translation performance, but also introduces unnecessary grammatical dependencies, (e.g. there are hundreds of tenseless languages), and is not optimized in all scenarios.

To the best of our knowledge, our work is the first to explore an abstract subword representation for logographic languages. The abstract representation makes it applicable to different logographic languages. It is a very desirable feature especially for

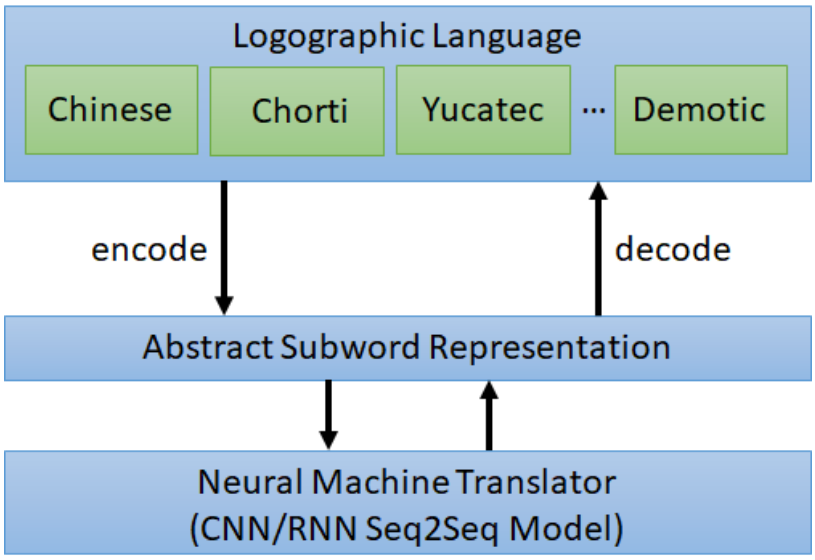


Figure 4.1: Abstract subwords are code symbols which are independent on a particular language. They directly participate in machine translation on behave of words of the logographic language.

most of the low resource logographic languages.

4.3 System Architecture

The proposed logographic subword model consists of an encoder and a decoder. The encoder transforms a word into multiple abstract subwords (code symbols), and the decoder transforms multiple abstract subwords into a word (Figure 4.1). Only abstract subwords directly participate in the training of the sequence-to-sequence models. This additional layer of abstraction reduces the model size, because the smaller dictionary that the abstract subwords form results in an output layer with the smaller number of units in the neural network.

Abstract subwords are code symbols which are independent on a particular language. These symbols are shared among words, and thus there will be much fewer symbols

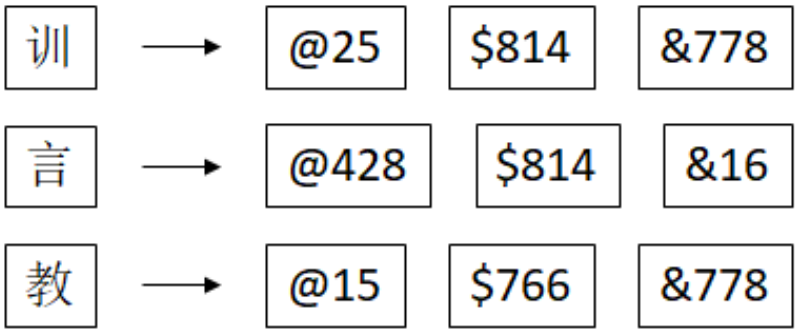


Figure 4.2: The encoder converts the word "train" in the corpus to three new abstract subwords (symbols) "@25", "$814" and "&778". "Train" and "talk" share the second symbol "$814", and "train" and "teach" share the third symbol "&778". Through sharing, the number of distinct symbols in the dictionary can be greatly reduced.

in both the source and target dictionaries (Figure 4.2). Symbols are created from a quantizer's codebook. The code at the first dimension is left padded an "@" and the code at the second dimension is left padded a "$", etc.

4.3.1 Encoder

The encoder quantizes and decomposes the embeddings of words into abstract subwords. Embedding maps words to vectors of real numbers. Similarities among words in the corpus are reflected to a large extent by Euclidean distances, which preserve the relationship between pairs of words. However, machine translation is generally modeled as a classification problem rather than a regression problem. Consequently, vectors of real numbers need to be represented by vectors of distinct symbols. This can be accomplished via quantization.

Quantization transforms vectors of real numbers to code symbols. In contrast to the initial word identifiers (before embedding), code symbols embed the information about word similarities. Furthermore, the code symbols can be far fewer than the

word identifiers, and this alleviates the classification burden and reduces the size of the neural network. To retain the translation performance, quantization also needs to enforce a one-to-one correspondence between words and vectors of code symbols.

In order not to significantly increase the length of a long sentence, the decomposition procedure ensures that only infrequent words are replaced by their quantized vectors of code symbols. Code symbols (abstract subwords) participate in the training of the sequence-to-sequence model, as opposed to word identifiers.

The encoder transforms words in the corpus to fewer distinct symbols (abstract subwords) resulting in a smaller dictionary. A smaller target dictionary leads to a smaller model size. With the smaller number of weights and bias, less gradient calculation is needed, and training is sped up. On the other hand, with the smaller number of weights and bias, there are fewer operations during inference, and inference time is shortened.

More details of quantization and decomposition will be covered in Section 4.4 and Section 4.5 respectively.

4.3.2 Decoder

Since code symbols are used in training, actual words are unknown to the neural network. Inference outputs are sequences of code symbols (abstract subwords). Symbol-level prediction errors may make it impossible to exactly locate a word by the sequence of decoded symbols. We notice that most of the time, such errors tend to be minor, with only one of symbols in the sequence incorrectly predicted. In order to restore the original symbols and locate the right word, we can apply the nearest neighbor search (Jegou *et al.*, 2011) to efficiently decode code symbols to words of the logographic

language even in presence of prediction errors.

Experiments show that the translation performance (BLEU score) with our approach actually matches that without any preprocessing, since the nearest neighbor search to some extent rectifies prediction errors, and with the smaller number of output classes (neurons), the resulting neural network has lower complexity.

4.4 Product Quantization

Quantization is an important component of the encoder. In our experiments, product quantization methods are examined. Product quantization was first proposed by Jégou and his colleagues (Jegou *et al.*, 2011) for nearest neighbor search. They decomposed the space into lower dimension subspaces and perform quantization on each subspace separately. Consider a space of n dimensions evenly divided into m subspaces each of n/m dimensions. Quantization is then performed on each of the m subspaces separately and maps vectors in each subspace to codes in each sub-codebook.

Formally, let X be the set of all vectors in the space, and x is a vector in X. Assume that x can be evenly divided into m subvectors, noted by $u_i(x)$, where $i \in I = \{1, ..., m\}$. We have that x is the concatenation of all $u_i(x)$, noted by

$$x = u_1(x)||...||u_m(x),$$

where symbol $||$ stands for concatenation. With the above definitions, subspace X_i can be defined as the set of all $u_i(x)$ for all $x \in X$, noted by

$$X_i := \{u_i(x)|x \in X\}, \forall i \in I$$

The m subspaces are quantized separately. Subquantizer q_i of subspace X_i maps $u_i(x)$ to a reference vector in $C_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,k_i}\}$, where k_i is the size of the codebook C_i for X_i. Here, q_i defines a partition of X_i, namely,

$$S_{i,j} := \{u_i(x) | x \in X \text{ and } q_i(u_i(x)) = c_{i,j}\},$$

where $i \in I, j \in J_i = \{1, 2, \ldots, k_i\}$.

Let $q(x)$ be the concatenation of all the reference vectors, noted by

$$q(x) = q_1(u_1(x))||...||q_m(u_m(x))$$

The quality of a quantizer is usually measured by the mean squared error between the input vector x and its reproduction value $q(x)$,

$$E_{PQ} = \sum_{x \in X} ||x - q(x)||^2$$

For a quantizer to be optimal, it should quantize the subvectors of $x \in X$ to their nearest centroids, namely,

$$c_{i,j} = \frac{1}{|S_{i,j}|} \sum_{u_i(x) \in S_{i,j}} u_i(x)$$

4.4.1 Centroid Initialization and Gaussian Kernel Density

Product quantization (Jegou *et al.*, 2011) is an effective quantization method. However, the design of the quantizer is application dependent. With respect to the GloVe (Pennington *et al.*, 2014) or Word2Vec (Mikolov *et al.*, 2013) vector spaces, our

experiments show that higher frequency words tend to have higher densities around in the space. Since high-frequency words are more likely to appear in the target sentences, intuitively, making higher frequency words more distinguishable helps improving translation performance. Thus, we propose a new density aware product quantization (DAPQ), which is a variant of product quantization customized for word embedding and machine translation.

First, we estimate the Gaussian kernel density for every subspace. The log density is used in calculating the initial centroids for the k-means++ algorithm (Arthur and Vassilvitskii, 2007). Initially, the first centroid is selected uniformly at random from X_i, and subsequent centroids are chosen from any vector z in X_i with probability:

$$p_i(z) = \frac{\rho(z)D(z)^2}{\sum_{z \in X_i} \rho(z)D(z)^2}, z \in X_i, i \in I, \tag{4.4.1}$$

where $\rho(z)$ is the Gaussian kernel density of z and $D(z)$ denotes the distance from z to its nearest centroid. The density term $\rho(z)$ allows more centroids to be initially assigned to denser areas. Consequently, dense words, which are also frequent words, are more likely to be grouped into distinct clusters in subspaces. Doing so increases the chances frequent words being correctly decoded in the target sentences.

4.4.2 The Number of Clusters and Degree of Distinctness

Quantization locally minimizes the distortion in each subspace. However, there are a number of global hyperparameters that remain to be decided. In product quantization, there are two important hyperparameters: *the number of subspaces* m and *the number of clusters in each subspace* k. It is observed that the selections of m and k affect the translation performance (BLEU scores), but their effects are indirect and not

analytically tractable. However, we observe that a more distinct correspondence (e.g. the most distinct one-to-one correspondence) tends to improve translation performance. Thus, it is necessary to introduce a metric to measure the *degree of distinctness* DoD (Figure 4.4). Intuitively, larger k tends to increase the distinctness and consequently translation performance. Therefore, it is reasonable to use DoD as a performance metric to find the best number of clusters k in each subspace.

Degree of Distinctness

DoD affects the reversibility of quantization. It actually tells how unique the code vectors are. Uniqueness is critical, e.g., if both "mile" and "kilometer" are encoded to the same code vector, ambiguity will arise. It is not reversible no matter selecting either one (e.g., based on weights), and it will hurt the translation quality. Our experiment demonstrates that the degree of distinctness (uniqueness) actually significantly impacts the final BLEU score in machine translation. Thus, it is a crucial metric to consider in designing the encoding algorithm.

Formally, let W be the set of all the words from the logographic dictionary and Q is the set of all the distinct code vectors generated by quantization. DoD $\mathfrak{D}$ is defined as e to the power of the multiplication of a scaler b and the result of 1 subtracted by the cardinality of set W divided by that of set Q, noted by:

$$\mathfrak{D} = e^{b(1-\frac{|W|}{|Q|})}, \tag{4.4.2}$$

where $|\cdot|$ is the cardinality of a set.

Optimizing the Number of Clusters in each Subspace

As the number of clusters increases, DoD increases at the cost of a slightly larger dictionary. Denote $D_{trgt} \in [0, 1]$ be the target DoD. To find the value of k, starting from an initial value $\lceil \sqrt[m]{|X|} \rceil$, we incrementally search with step size η until the resulting DoD reaches D_{trgt}.

More generally, the number of clusters can be different from subspace to subspace. Let k_i be the number of clusters in subspace X_i with an initial value of $\lceil \sqrt[m]{|X|} \rceil$. Finding the optimal k_i's is an integer programming problem. We adopt a simple heuristic similar to coordinate descent by increasing the number of clusters in one subspace at a time. Specifically, for each i, we compute ΔD_i as the change in DoD when increasing k_i by η while keeping the others the same. Let $i^* = \arg\max_{i \in I} \Delta D_i$. Then, we update $k_{i^*} = k_{i^*} + \eta$. The process is repeated until D_{tgrt} is met.

4.4.3 Reduction in Target Dictionary

The target dictionary size is given by $\sum_i k_i$. If $D_{tgrt} = 1$, it is easy to show that $\prod_i k_i \geq |X|$. This implies that $\sum_i k_i \geq m \sqrt[m]{|X|}$. Therefore, the maximum reduction in vocabulary size is $\frac{\sum_i k_i \geq m \sqrt[m]{|X|}}{|X|}$. Empirical results show that k_i's found by the heuristics discussed previously are mostly close to their initial values $\sqrt[m]{|X|}$, resulting significant reduction in dictionary size. For example, for $|X| = 64000$ and $m = 3$, the maximum reduction is $64000/3 * 40 = 533$.

4.5 Decomposition

Decomposition is another part of the encoder. Quantizing words to multiple code symbols can significantly reduce the target dictionary size. However, doing so for each word of a sentence would unnecessarily increase the length of the sentence. Experiments show that longer sentences would adversely affect BLEU scores, training and inference times. On the other hand, a smaller dictionary is beneficial to these metrics. Clearly, there exists a trade-off between the dictionary size and sentence length.

To address this problem, the decomposer measures the frequency of words in the corpus, defined as the number of occurrences of a word in the corpus divided by the total number of words in the corpus. It is observed that most words in the corpus are infrequent words, while most sentences are largely composed of frequent words. Therefore, it is sufficient and beneficial to only decompose infrequent words to their vectors of code symbols, (to reduce the dictionary size but not to significant increase sentence length).

We introduce parameter f_{ct} as the cut-off frequency to judge whether a word is an infrequent word or not. If a word's frequency is larger than the cut-off frequency, it is a frequent word, or else an infrequent word. As shown in Figure 4.3, a larger cut-off frequency leads to a smaller target dictionary size, but a longer sentence on average, and vice versa.

In summary, the proposed encoder (as described in Algorithm 5) takes as input the set of all words W, the corpus of all sentence pairs C, the number of partitions m, the distinctness objective $\mathfrak{D}_{trgt}$, the learning rate η, and the cutoff frequency f_{ct}. It calculates the (GloVe (Pennington *et al.*, 2014)) embedding of W and assigns it to X.

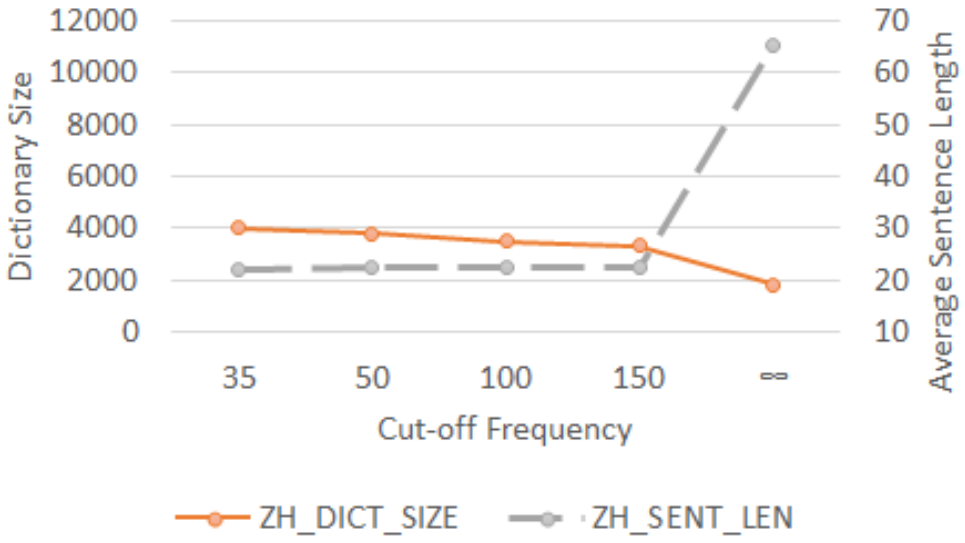


Figure 4.3: For the logographic language Chinese, as the cut-off frequency increases, the number of symbols decreases while the average sentence length increases. The BLEU score, training, and inference are benefited from the smaller dictionary and shorter sentences. There is a trade-off between them since they increase/decrease in the opposite directions with respect to the cut-off frequency.

It initiates the degree of distinctness $\mathfrak{D}$ and the number of clusters in each subspace k. Here, k is initialized as $\lceil \sqrt[n]{|X|} \rceil$, and increases iteratively until the target degree of distinctness is reached. Product quantization takes k as the hyper-parameter and outputs the set of code symbols with minimum distortion. Decomposition decomposes only infrequent words to their code symbols without over elongation of the sentences based on the cut-off frequency f_{ct}.

4.6 Evaluation

In this section, we evaluate the translation performance of the proposed logographic subword model in both the RNN and CNN sequence-to-sequence translators.

Algorithm 5 Logographic Encoder

procedure ENCODE($W, C, m, \mathfrak{D}_{trgt}, \eta, f_{ct}$)
  $X \leftarrow$ EMBEDDING(W)
  $\mathfrak{D} \leftarrow 0$
  $k \leftarrow \lceil \sqrt[m]{|X|} \rceil$
  while $\mathfrak{D} < \mathfrak{D}_{trgt}$ **do**
    $Q \leftarrow$ QUANTIZE(X, m, k)
    $\mathfrak{D} \leftarrow e^{1-\frac{|W|}{|Q|}}$
    $k \leftarrow k + \eta$
  end while
  for w **in** C **do**
    if $w.f < f_{ct}$ **then**
      DECOMPOSE(w, Q)
    end if
  end for
end procedure

4.6.1 Experiment Setup

The United Nation Parallel Corpus (Ziemski *et al.*, 2016) is used for model training and testing. English (alphabetic language) and Chinese (logographic language) translations are evaluated for both the RNN and CNN translators. The first 500000 pair-wise aligned sentences are taken from the corpus's training dataset, among which sentences longer than 40 words are not selected, (in order to simplify the model complexity brought unnecessarily to the experiments). This results in 259644 pair-wise aligned sentences for English and Chinese. To be consistent with the training dataset, sentences longer than 40 words are also not selected from the corpus's development and test dataset, and there are remaining 1928 and 1948 (out of 4000) pair-wise aligned sentences for development and test respectively. In total, there are 61168 and 31700 tokens for English and Chinese respectively in those sentences.

We evaluate the baseline model, and our logographic subword model with a LOPQ

Table 4.1: Terms used in the proposed system

Term	**Meaning**
W	the set of all words
X	the set of all word vectors
Q	the set of all code vectors
m	the number of partitions
k	the number of clusters per subspace
$\mathfrak{D}$	the degree of distinctness
η	the learning rate
C	the corpus of all sentence pairs
$\mathfrak{D}_{trgt}$	the distinctness objective
f_{ct}	the cutoff frequency

and DAPQ quantizer (L-SW-LO and L-SW-DA respectively). In the encoder of the logographic subword model, GloVe (Pennington *et al.*, 2014) is used to construct embeddings of logograms, wherein every word vector has 6 dimensions. The LOPQ and DAPQ quantizers are both outputting 3-dimension code vectors. Code vectors are formatted to abstract subword (symbol) vectors as illustrated in Figure 4.2. In these experiments, $\mathfrak{D}_{trgt}$ are all set to 1, thus there exists a one-to-one correspondence between a logographic word and the vector of abstract subwords (code symbols), and the encoding is fully reversible.

The RNN seq2seq model is trained and evaluated on Nematus (Sennrich *et al.*, 2017) and Theano (Al-Rfou *et al.*, 2016). It is a 2-layer bidirectional seq2seq model with each GRU cell having 1024 units. Input embedding to both encoder and decoder cells have 512 dimensions. In the experiments with L-SW-LO and L-SW-DA, every infrequent word is replaced by 3 code symbols. Since infrequent words rarely appear in sentences, the sentence lengths do not increase substantially. Thus, in those experiments, the cut-off lengths are set to 50. Theano is used to evaluate the proposed approach with the RNN seq2seq model. A Nvidia GTX 1080ti GPU is assigned to each translation

task for both training and inference.

The CNN seq2seq model is trained and evaluated on Fairseq (Gehring *et al.*, 2017, 2016) and Torch (Collobert *et al.*, 2011). Both the encoder and decoder in the CNN seq2seq model are fully convolutional. The target dictionary size decides the model complexity and how the hyperparameters are set. Since there are larger target dictionaries in the experiments with the baseline model, we define 6 convolutional layers with 768 channels in the first four layers and 1024 channels in the last two. The kernel sizes are set to 3 for the first five layers and 1 for the last layer. Since there are smaller dictionaries for L-SW-LO and L-SW-DA, we define 4 convolutional layers with 384 channels in the first two layers and 512 channels in the last two. The kernel sizes are set to 5 for the first two layers, 3 for the third layer and 1 for the last layer. Input embedding to both encoder and decoder have 128 dimensions. Cut-off lengths are set to 50 for all language pairs. Torch is used to evaluate the proposed approach with the CNN seq2seq model. Two Nvidia GTX 1080ti GPUs are assigned to each translation task for both training and inference.

4.6.2 Results

We examine how our approach improves the performance of both the RNN seq2seq model (Cho *et al.*, 2014; Bahdanau *et al.*, 2014) and the CNN seq2seq model (Gehring *et al.*, 2017). During encoding, every logographic word in those sentences is replaced by their corresponding abstract subword (symbol) vectors. New dictionaries are created from those encoded sentences. This results in much smaller dictionaries for Chinese (as indicated by the TrgtV column in Table 4.2). We then train both the RNN and CNN sequence-to-sequence models with them.

Table 4.2: "PrePr": preprocessing method;"TrgtV": the number of tokens in the target dictionary; "NN": the sizes of the neural networks; Per RNN, "T" is the elapsed time of 15 epochs of training on one GTX1080TI GPU; per CNN, "T" is the elapsed time of 18 epochs of training on two GTX1080TI GPUs. "t" is the average inference time for the translation per each sentence. "L-SW-LO" stands for the logographic subword model with LOPQ encoding. "L-SW-DA" stands for the logographic subword model with DAPQ encoding.

(a) RNN Seq2Seq Model

L	PrePr	TrgtV	NN(Mb)	T(hr)	t(ms)	BLEU
EN-ZH	-	31702	466.6	16.45	241.79	**50.31**
	L-SW-LO	4997	236.4	15.39	128.53	49.95
	L-SW-DA	**3492**	**228.1**	**13.45**	**124.43**	49.63
ZH-EN	-	61170	527.0	16.81	320.84	48.73
	L-SW-LO	7844	253.7	**14.93**	137.58	48.59
	L-SW-DA	**6632**	**234.3**	14.94	**130.92**	**49.23**

(b) CNN Seq2Seq Model

L	PrePr	TrgtV	NN(Mb)	T(hr)	t(ms)	BLEU
EN-ZH	-	31703	501.8	2.54	35.93	**49.22**
	L-SW-LO	4998	112.2	0.98	**21.05**	47.07
	L-SW-DA	**3493**	**107.8**	**0.87**	**21.05**	47.47
ZH-EN	-	61171	532.8	3.06	34.91	48.21
	L-SW-LO	7845	115.5	1.01	**21.05**	47.39
	L-SW-DA	**6633**	**111.4**	**1.00**	22.07	**48.22**

During testing, BLEU scores are evaluated by comparing the predicted sentences with the candidates. It is observed that BLEU scores are comparable to the baseline model and slightly better in the Chinese-English translation.

Model sizes (NN), training time (T) and inference time (t) are measured as shown in Table 4.2. We observe that in the English-to-Chinese translation task, the model sizes are reduced by 37% and 77% respectively with RNN and CNN; the training time is 11% and 73% shorter; and the inference time is nearly halved in RNN and 36% shorter in CNN.

Table 4.3: In the task of Chinese-English translation, the BPE approach cannot identify subwords in Chinese logograms and consequently hurts the final BLEU score.

PrePr	**NN(Mb)**	**T(hr)**	**t(ms)**	**BLEU**
SW-BPE	347.2	**14.59**	169.92	47.66
L-SW-LO	253.7	14.93	137.58	48.59
L-SW-DA	**234.3**	14.94	**130.92**	**49.23**

We compare the logographic subword approaches with the subword BPE approach for Chinese-English translation (Table 4.3). BPE cannot identify subwords in Chinese logograms. Consequently, it hurts the final BLEU score. Our approaches fit better to logographic languages in this direction. In the direction from English to Chinese, we did not observe the proposed models are better than BPE-based models.

Reversibility is critical to translation performance. We explore various values of DoD $\mathfrak{D}$ (at cut-off frequency $f_{ct} = \infty$) and evaluate the translation performance and model sizes in the CNN seq2seq model (Figure 4.4). Experiments are conducted for language pairs with the same hyper-parameters except different $\mathfrak{D}$ values. As expected, the BLEU score increases as $\mathfrak{D}$ increases and reaches the maximum when $\mathfrak{D}$ equals to 1, since the encoding is fully reversible at this point. Figure 4.4 demonstrates the correlation between DoD and the BLEU score at $\mathfrak{D} = e^{0.5(1-\frac{|W|}{|Q|})}$ ($b = 0.5$ in Equation 4.4.2); BLEU is proportional to $\mathfrak{D}$ in both translation tasks.

4.7 Conclusion

A logographic subword model is proposed, with an encoder and decoder transforming logograms to abstract subwords and subwords to logograms. The encoder quantizes and decomposes the embeddings of logograms. By sharing common abstract subwords (code symbols), quantization reduces the dictionary size without sacrificing translation

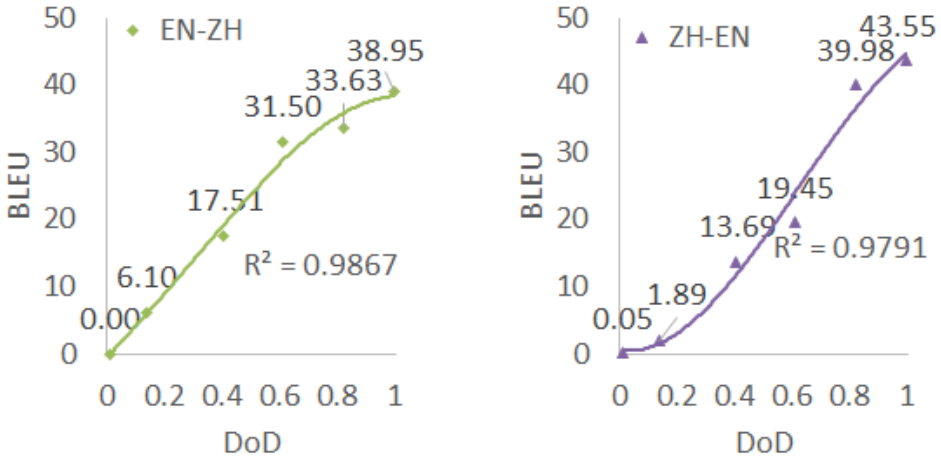


Figure 4.4: There exists a correlation between the *degree of distinctness* $\mathfrak{D}$ and the BLEU score at $\mathfrak{D} = e^{0.5(1-\frac{|W|}{|Q|})}$ on a CNN seq2seq model. The BLEU score is polynomially proportional to $\mathfrak{D}$ for both language pairs.

performance. A new metric, *degree of distinctness*, is proposed to quantify the effect of distinctness and reversibility.

The proposed approach has been shown with experiments to reduce model sizes as well as shorten training and inference time for both RNN and CNN sequence-to-sequence models. It is promising for reducing the complexity of other computationally expensive NLP problems with potential impact on large-dictionary real-time offline applications such as translation or dialog systems on offline mobile platforms.

As future work, we will build a faster and more accurate encoder and decoder, and explore the use of abstract subwords in other logographic languages such as Chorti and Demotic. Those are important since the proposed techniques help to understand different logographic languages from the subword perspective.

Chapter 5

Conclusion

Many merits arise to real-time applications when deep learning is on the edge such as shorter latency, better privacy, and autonomy. Edge artificial intelligence (EI) methods arise in minimizing the difference between the current situation and future vision including but not limited to model partitioning, caching, and compression. For partitioning, we proposed TeamNet which is a knowledge-aware partitioning approach for collaborative inference on mobile edge. For caching, we proposed CacheNet, an information maximizing framework specialized to deep learning model caching on the edge. For compression in machine translation, we proposed logographic subword model which enables a more compact dictionary (vocabulary) to RNN and CNN sequence-to-sequence models.

For future research, we first highlight deep insights and potential improvements to the current approaches, and then we introduce new directions in the field of EI.

5.1 Deep Insights and Improvements

5.1.1 Partitioning and Caching in Sequence Learning

Uncertainty (entropy) has been deeply integrated into the architectures of TeamNet and CacheNet. However, unlike the entropy of a single class prediction, the uncertainty (entropy) of a sequence is a joint entropy of multiple class predictions. Computation of joint entropy is intractable with a large number of classes and even a moderate length sequence. Without solving this problem, it is impractical to extend TeamNet and CacheNet to natural language processing tasks such as machine translation and question answering. Although joint entropy is not likely computable directly, an approximation may be found.

The uncertainty of a sequence X_1, X_2, ..., X_n can be quantified by the joint entropy $H(X_1, X_2, ..., X_n)$ of all the items (random variables) in the sequence. By Shannon, the definition of the joint entropy involves the calculation of the joint probabilities $p(X_1, X_2, ..., X_n)$ of the all the possible sequences, noted by:

$$H(X_1, X_2, ..., X_n) = -\sum p(X_1, X_2, ..., X_n) \log p(X_1, X_2, ..., X_n)$$

Joint entropy is particularly important for the measurement of uncertainty in applications involving sequences such as machine translation, text summarization and video captioning. However, the computation of joint entropy is very expensive. For example, in machine translation, there are usually more than 100000 words in the English dictionary and 10 to 20 words in the sentence (sequence). The number of combinations of all the choices will easily exceed 100000^{10}. It poses a huge burden to the computation of joint entropy and prohibits its application to uncertainty estimation

in deep learning.

In order to solve the above problem, many attempts have been proposed such as the bounds of joint entropy Madiman and Tetali (2007) and other approximation approaches Wu *et al.* (2016a); Radicchi and Castellano (2018), but the computations are still intense. There is a need for simple approximations.

We observe that in Machine Translation, most of the item probabilities are near zero. Near-zero item probabilities contribute very little to the joint probability and the joint entropy of the sequence.

Assume the Markov property applies, according to the chain rule, the joint probability of the sequence $p(X_1, X_2, ..., X_n)$ equals to the product of the conditional probability $p(X_i|X_{i-1})$ of all items, where i is $1, 2, ..., n$.

$$p(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} p(X_i|X_{i-1}) \tag{5.1.1}$$

Any of the $p(X_i|X_{i-1})$ near zero leads to the joint probability $p(X_1, X_2, ..., X_n)$ near zero. Consequently, this contributes very little to the joint entropy $H(X_1, X_2, ..., X_n)$. Since their contributions are little, we can group them into a residual $\bar{H}$ in the calculation.

In order to calculate the residual $\bar{H}$, we assume the joint probability of those sequence follows uniform distribution. Let $\bar{p}(X_i|X_{i-1})$ be the mean conditional probability of items in those sequences. The mean joint probability of those sequences $\bar{p}\bar{p}(X_1, X_2, ..., X_n)$ is equal to the product of the conditional probabilities of all the items, noted by:

$$\bar{p}(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} \bar{p}(X_i|X_{i-1}) \tag{5.1.2}$$

Since it is easy to calculate the number of combinations N in total, the residual $\bar{H}$ is approximately equal to the negation of the total number N multiplied by mean joint probability multiplied by the logarithm of mean joint probability, noted by:

$$\bar{H} \approx -N\bar{p}(X_1, X_2, ..., X_n) \log \bar{p}(X_1, X_2, ..., X_n) \tag{5.1.3}$$

We define a set Ω with all the sequences with noticeable joint probabilities. The joint entropy is then approximately the negation of the sum of the joint probability of those sequences multiplied by the logarithm of the joint probability added by the residual $\bar{H}$, noted by:

$$H(X_1, X_2, ..., X_n) \approx -\sum_{\Omega} p(X_1, X_2, ..., X_n) \log p(X_1, X_2, ..., X_n) + \bar{H} \tag{5.1.4}$$

The benefits are obvious. We only have to calculate a few joint probabilities, considering there are only a few sequences with noticeable joint probabilities but they contribute most to the joint entropy.

5.1.2 Partitioning in a Subjective and Unlabelled Task

TeamNet is applicable to supervised learning tasks but not unsupervised ones. Is it necessary to label all the instances in an unlabelled dataset to train TeamNet, especially for subjective predictions such as predicting the sentiments of music or text messages?

Sentiment analysis aims to recognize the sentiment polarity through natural language processing. Traditional sentiment analysis identifies either positive or negative

polarity from the text. The work in Fang *et al.* (2015) and Hu *et al.* (2009) goes beyond binary classifications. The authors proposed to recognize specific emotions through natural language processing on the lyrics.

Russell's circumplex model of emotion Russell (1980) has been widely adopted by many other researchers Paltoglou and Thelwall (2012); Gobron *et al.* (2010) in the field of affective computing. The advantage of Russell's model is that they represented emotions as a coordinate in a two-dimensional space with valence as the x-axis and arousal as the y-axis. Any emotion in Russell's model corresponds to the combination of the degrees of valence and arousal.

Consider the incorporation between the machine learning classifiers and Russell's Circumplex model. Fang *et al.* (2015) proposed to define emotional states on Russell's model and each state stands for a coordinate subspace in Russell's two-dimensional space. The advantage of their definition is that the output class labels of the classifiers can be directly associated with specific emotions in Russell's model. Support vector machine (SVM) is used in Fang *et al.* (2015), and it has been shown to effectively recognize emotional states given the n-gram language model of the lyrics. In natural language processing, the n-gram model has been successfully used in tasks such as machine translation Crego *et al.* (2005) and sentiment analysis Kouloumpis *et al.* (2011); He *et al.* (2008). The SVM model was trained from a corpus labeled by crowd workers.

The disadvantages of Fang *et al.* (2015)'s approach are that 1) the lyrics to label were randomly selected, and 2) SVM is unsuitable for a subject-specific sentiment analysis task. TeamNet inherits characteristics of mixture of experts (MoE), and trains experts for a specific subject or group of subjects.

A possible approach is that TeamNet is initially trained by a smaller set of labeled lyrics and then it judges which unlabeled lyric is worthy of labeling based on uncertainty. (A possible way may be that only a very uncertain lyric for all the experts is worth to label.) The labeling is performed on a crowdsourcing (or crowdsensing) platform. Fewer labels often lead to lower costs to task owners.

5.2 Future Research Directions

Partitioning with Neural Architecture Search: How smaller can a partition of TeamNet be compared to the full model? Shall a partition be of similar architecture as the full model? Those questions may be answered by the recent advances in neural architecture search (NAS). NAS approaches have shown to outperform hand-crafted architectures on several tasks, such as image classification Real *et al.* (2019) and semantic segmentation Liu *et al.* (2019). NAS in model partitioning is an interesting direction to explore.

Hierarchical Partitioning as a Soft Decision Tree: Decision trees have been shown to be advantageous in interpretability and accuracy. Recent research in decision trees and neural networks has demonstrated the feasibility of combining both types of models, such as neural-backed decision tree Wan *et al.* (2020). TeamNet, as a partitioning approach, inheriting characteristics of MoE models, is similar to a decision tree hierarchy. Is it possible to take one step further to make it an interpretable soft decision tree? If so, more AI tasks become provable rather than empirical. A rigorous partitioning approach also improves the reliability of EI applications.

Partitioning with Conditional Variational Autoencoder: It has been observed in CacheNet (a combination of caching and partitioning) that variational autoencoder (with maximizing mutual information between the input and the latent variable) helps to diversify a dataset into specific subsets. If the dimension of latent variables is properly chosen, a roughly balanced partition can be achieved. Conditional variational autoencoder allows a decoder to generate an output with respect to a particular condition. The combination of a condition and mutual information gives more control to the partitioning process.

Compression with Gradient-Optimizing Product Quantization: Product quantization has shown a significant reduction in the dictionary size. It is possible to further improve the compression ratio if product quantization is applied to all the network parameters. The challenge is that it is hard to scale up with a heuristic quantization algorithm. Combination with a gradient descent algorithm in optimization allows a thorough quantization throughout all neural network parameters.

Edge Artificial Intelligence in Self-Driving and Driving-Assistance: Artificial neural networks have been successfully applied to self-driving and driving-assistance tasks such as lane change prediction Dou *et al.* (2018); Scheel *et al.* (2019) and lane detection Zou *et al.* (2019). Model partitioning, caching, and compression help reduce latency substantially and enable real-time prediction and detection for self-driving vehicles. In such applications, the input data is sequential in nature, e.g., multiple vision or LADAR frames in lane change prediction. Thus, it is important to extend EI approaches to be handle sequence data.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., *et al.* (2016). Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283.

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., *et al.* (2016). Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, **472**, 473.

Aljundi, R., Chakravarty, P., and Tuytelaars, T. (2017). Expert gate: Lifelong learning with a network of experts. In *CVPR*, pages 7120–7129.

Anil, R., Pereyra, G., Passos, A., Ormandi, R., Dahl, G. E., and Hinton, G. E. (2018). Large scale distributed neural network training through online distillation. *arXiv preprint arXiv:1804.03235*.

Apicharttrisorn, K., Ran, X., Chen, J., Krishnamurthy, S. V., and Roy-Chowdhury, A. K. (2019). Frugal following: Power thrifty object detection and tracking for mobile augmented reality. In *Proceedings of the 17th Conference on Embedded Networked Sensor Systems*, pages 96–109.

Arthur, D. and Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *arXiv preprint arXiv:1607.06450*.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bao, W., Lai, W.-S., Ma, C., Zhang, X., Gao, Z., and Yang, M.-H. (2019). Depth-aware video frame interpolation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3703–3712.

Barone, A. V. M., Helcl, J., Sennrich, R., Haddow, B., and Birch, A. (2017). Deep architectures for neural machine translation. *arXiv preprint arXiv:1707.07631*.

Best-Rowden, L., Han, H., Otto, C., Klare, B. F., and Jain, A. K. (2014). Unconstrained face recognition: Identifying a person of interest from a media collection. *IEEE Transactions on Information Forensics and Security*, **9**(12), 2144–2157.

Bishop, C. M. and Svenskn, M. (2002). Bayesian hierarchical mixtures of experts. In *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*, pages 57–64. Morgan Kaufmann Publishers Inc.

Cao, Q., Shen, L., Xie, W., Parkhi, O. M., and Zisserman, A. (2018). Vggface2: A dataset for recognising faces across pose and age. In *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018)*, pages 67–74. IEEE.

Chapelle, O. and Wu, M. (2010). Gradient descent optimization of smoothed information retrieval metrics. *Information retrieval*, **13**(3), 216–235.

Chatelain, C., Heutte, L., and Paquet, T. (2006). Segmentation-driven recognition applied to numerical field extraction from handwritten incoming mail documents. In *International Workshop on Document Analysis Systems*, pages 564–575. Springer.

Chen, C., Batselier, K., and Wong, N. (2017). A novel tensor-based model compression method via tucker and tensor train decompositions. In *2017 IEEE 26th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pages 1–3. IEEE.

Chen, T. Y.-H., Ravindranath, L., Deng, S., Bahl, P., and Balakrishnan, H. (2015). Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168. ACM.

Cho, K., Van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*.

Ciresan, D. C., Meier, U., Masci, J., Maria Gambardella, L., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237. Barcelona, Spain.

Cireşan, D. C., Meier, U., Gambardella, L. M., and Schmidhuber, J. (2011). Handwritten digit recognition with a committee of deep neural nets on gpus. *arXiv preprint arXiv:1103.4487*.

Cohen, N., Sharir, O., and Shashua, A. (2016). On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728.

Collobert, R., Kavukcuoglu, K., and Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS workshop*, number EPFL-CONF-192376.

Courbariaux, M., Bengio, Y., and David, J. (2014). Low precision arithmetic for deep learning. *CoRR, abs/1412.7024*, **4**.

Courbariaux, M., Bengio, Y., and David, J.-P. (2015). Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131.

Crego, J. M., Mariño, J. B., and Gispert, A. d. (2005). An n-gram-based statistical machine translation decoder. In *Ninth European Conference on Speech Communication and Technology*.

Delalleau, O. and Bengio, Y. (2011). Shallow vs. deep sum-product networks. In *Advances in Neural Information Processing Systems*, pages 666–674.

Deng, W., Hu, J., Zhang, N., Chen, B., and Guo, J. (2017). Fine-grained face verification: Fglfw database, baselines, and human-dcmn partnership. *Pattern Recognition*, **66**, 63–73.

Diederik, P. K., Welling, M., *et al.* (2014). Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dou, Y., Fang, Y., Hu, C., Zheng, R., and Yan, F. (2018). Gated branch neural network for mandatory lane changing suggestion at the on-ramps of highway. *IET Intelligent Transport Systems*, **13**(1), 48–54.

Fang, Y., Barone, M., and Woolhouse, M. (2015). Sentiment analysis of mandarin pop lyrics using multi-emotion profiles. In *Seminar on Cognitively Based Music Informatics Research*.

Fang, Y., Jin, Z., and Zheng, R. (2019). Teamnet: A collaborative inference framework on the edge. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1487–1496. IEEE.

Gage, P. (1994). A new algorithm for data compression. *C Users J.*, **12**(2), 23–38.

Gal, Y. and Ghahramani, Z. (2016). A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.

García-Martínez, M., Barrault, L., and Bougares, F. (2016). Factored neural machine translation. *arXiv preprint arXiv:1609.04621*.

Gastaldi, X. (2017). Shake-shake regularization. *arXiv preprint arXiv:1705.07485*.

Gehring, J., Auli, M., Grangier, D., and Dauphin, Y. N. (2016). A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*.

Gens, R. and Domingos, P. (2012). Discriminative learning of sum-product networks. In *Advances in Neural Information Processing Systems*, pages 3239–3247.

Gobron, S., Ahn, J., Paltoglou, G., Thelwall, M., and Thalmann, D. (2010). From sentence to emotion: a real-time three-dimensional graphics metaphor of emotions extracted from text. *The Visual Computer*, **26**(6-8), 505–519.

Gong, Y., Liu, L., Yang, M., and Bourdev, L. (2014). Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

Guo, P., Hu, B., Li, R., and Hu, W. (2018). Foggycache: Cross-device approximate computation reuse. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 19–34.

Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. (2015). Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746.

He, H., Jin, J., Xiong, Y., Chen, B., Sun, W., and Zhao, L. (2008). Language feature mining for music emotion classification via supervised learning from lyrics. In *International Symposium on Intelligence Computation and Applications*, pages 426–435. Springer.

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

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141.

Hu, X., Downie, J. S., and Ehmann, A. F. (2009). Lyric text mining in music mood classification. *American music*, **183**(5,049), 2–209.

Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

Huang, G. B., Mattar, M., Berg, T., and Learned-Miller, E. (2008). Labeled faces in the wild: A database forstudying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*.

Huynh, L. N., Lee, Y., and Balan, R. K. (2017). Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 82–95. ACM.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive mixtures of local experts. *Neural computation*, **3**(1), 79–87.

Jegou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, **33**(1), 117–128.

Jordan, M. I. and Jacobs, R. A. (1994). Hierarchical mixtures of experts and the em algorithm. *Neural computation*, **6**(2), 181–214.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., *et al.* (2017). In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 1–12. IEEE.

Kalantidis, Y. and Avrithis, Y. (2014). Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2321–2328.

Kang, Y., Hauswald, J., Gao, C., Rovinski, A., Mudge, T., Mars, J., and Tang, L. (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*, pages 615–629. ACM.

Kim, Y.-D., Park, E., Yoo, S., Choi, T., Yang, L., and Shin, D. (2015). Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Ko, J. H., Na, T., Amir, M. F., and Mukhopadhyay, S. (2018). Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms. *arXiv preprint arXiv:1802.03835*.

Kouloumpis, E., Wilson, T., and Moore, J. (2011). Twitter sentiment analysis: The good the bad and the omg! In *Fifth International AAAI conference on weblogs and social media*.

Krizhevsky, A., Hinton, G., *et al.* (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

Krizhevsky, A., Nair, V., and Hinton, G. (2010). Cifar-10 (canadian institute for advanced research). *URL http://www. cs. toronto. edu/kriz/cifar. html*.

Kwon, J. and Lee, K. M. (2008). Tracking of abrupt motion using wang-landau monte carlo estimation. In *European conference on computer vision*, pages 387–400. Springer.

Lane, N. D., Bhattacharya, S., Georgiev, P., Forlivesi, C., Jiao, L., Qendro, L., and Kawsar, F. (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*, page 23. IEEE Press.

LeCun, Y., Haffner, P., Bottou, L., and Bengio, Y. (1999). Object recognition with gradient-based learning. In *Shape, contour and grouping in computer vision*, pages 319–345. Springer.

Lin, D., Talathi, S., and Annapureddy, S. (2016a). Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858.

Lin, D., Talathi, S., and Annapureddy, S. (2016b). Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858.

Lin, M., Chen, Q., and Yan, S. (2013). Network in network. *arXiv preprint arXiv:1312.4400*.

Littlestone, N. and Warmuth, M. K. (1994). The weighted majority algorithm. *Information and computation*, **108**(2), 212–261.

Liu, C., Chen, L.-C., Schroff, F., Adam, H., Hua, W., Yuille, A. L., and Fei-Fei, L. (2019). Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 82–92.

Liu, W., Wen, Y., Yu, Z., Li, M., Raj, B., and Song, L. (2017). Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 212–220.

Madiman, M. and Tetali, P. (2007). Sandwich bounds for joint entropy. In *Information Theory, 2007. ISIT 2007. IEEE International Symposium on*, pages 511–515. IEEE.

Mathur, A., Lane, N. D., Bhattacharya, S., Boran, A., Forlivesi, C., and Kawsar, F. (2017). Deepeye: Resource efficient local execution of multiple deep vision models using wearable commodity hardware. In *Proceedings of the 15th Annual*

International Conference on Mobile Systems, Applications, and Services, pages 68–81. ACM.

Miah, M. B. A., Yousuf, M. A., Mia, M. S., and Miya, M. P. (2015). Handwritten courtesy amount and signature recognition on bank cheque using neural network. *International Journal of Computer Applications*, **118**(5).

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Miller, D., Brossard, E., Seitz, S., and Kemelmacher-Shlizerman, I. (2015). Megaface: A million faces for recognition at scale. *arXiv preprint arXiv:1505.02108*.

Mutlu, O. and Subramanian, L. (2015). Research problems and opportunities in memory systems. *Supercomputing frontiers and innovations*, **1**(3), 19–55.

Nair, V. and Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814.

Ng, H.-W. and Winkler, S. (2014). A data-driven approach to cleaning large face datasets. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 343–347. IEEE.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. (2015). Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450.

Nurvitadhi, E., Venkatesh, G., Sim, J., Marr, D., Huang, R., Ong Gee Hock, J., Liew, Y. T., Srivatsan, K., Moss, D., Subhaschandra, S., *et al.* (2017). Can fpgas beat

gpus in accelerating next-generation deep neural networks? In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 5–14. ACM.

Paltoglou, G. and Thelwall, M. (2012). Seeing stars of valence and arousal in blog posts. *IEEE Transactions on Affective Computing*, **4**(1), 116–123.

Parkhi, O. M., Vedaldi, A., Zisserman, A., *et al.* (2015). Deep face recognition. In *BMVC*, volume 1, page 6.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Poon, H. and Domingos, P. (2011). Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE.

Press, O. and Wolf, L. (2016). Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*.

Radicchi, F. and Castellano, C. (2018). Uncertainty reduction for stochastic processes on complex networks. *Physical Review Letters*, **120**(19), 198301.

Rasmussen, C. E. and Ghahramani, Z. (2002). Infinite mixtures of gaussian process experts. In *Advances in neural information processing systems*, pages 881–888.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. (2019). Aging evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence*.

Russell, J. A. (1980). A circumplex model of affect. *Journal of personality and social psychology*, **39**(6), 1161.

Sainath, T. N., Kingsbury, B., Sindhwani, V., Arisoy, E., and Ramabhadran, B. (2013). Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE.

Salimans, T. and Kingma, D. P. (2016). Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909.

Sau, B. B. and Balasubramanian, V. N. (2016). Deep model compression: Distilling knowledge from noisy teachers. *arXiv preprint arXiv:1610.09650*.

Scheel, O., Nagaraja, N. S., Schwarz, L., Navab, N., and Tombari, F. (2019). Attention-based lane change prediction. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8655–8661. IEEE.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

Sennrich, R. and Haddow, B. (2016). Linguistic input features improve neural machine translation. *arXiv preprint arXiv:1606.02892*.

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*.

Sennrich, R., Firat, O., Cho, K., Birch, A., Haddow, B., Hitschler, J., Junczys-Dowmunt, M., Läubli, S., Barone, A. V. M., Mokry, J., *et al.* (2017). Nematus: a toolkit for neural machine translation. *arXiv preprint arXiv:1703.04357*.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*.

Shorten, C. and Khoshgoftaar, T. M. (2019). A survey on image data augmentation for deep learning. *Journal of Big Data*, **6**(1), 60.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Suzuki, J. and Nagata, M. (2016). Learning compact neural word embeddings by parameter space sharing. In *IJCAI*, pages 2046–2052.

Taigman, Y., Yang, M., Ranzato, M., and Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708.

Teerapittayanon, S., McDanel, B., and Kung, H. (2016). Branchynet: Fast inference via early exiting from deep neural networks. In *Pattern Recognition (ICPR), 2016 23rd International Conference on*, pages 2464–2469. IEEE.

Teerapittayanon, S., McDanel, B., and Kung, H. (2017). Distributed deep neural networks over the cloud, the edge and end devices. In *Distributed Computing*

Systems (ICDCS), 2017 IEEE 37th International Conference on, pages 328–339. IEEE.

Visin, F., Kastner, K., Cho, K., Matteucci, M., Courville, A., and Bengio, Y. (2015). Renet: A recurrent neural network based alternative to convolutional networks. *arXiv preprint arXiv:1505.00393*.

Wan, A., Dunlap, L., Ho, D., Yin, J., Lee, S., Jin, H., Petryk, S., Bargal, S. A., and Gonzalez, J. E. (2020). Nbdt: Neural-backed decision trees. *arXiv preprint arXiv:2004.00221*.

Wan, L., Zeiler, M., Zhang, S., Le Cun, Y., and Fergus, R. (2013). Regularization of neural networks using dropconnect. In *International conference on machine learning*, pages 1058–1066.

Wu, J., Gupta, S., and Bajaj, C. (2016a). Higher order mutual information approximation for feature selection. *arXiv preprint arXiv:1612.00554*.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., *et al.* (2016b). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.

Xu, M., Zhu, M., Liu, Y., Lin, F. X., and Liu, X. (2018). Deepcache: principled cache for mobile deep vision. In *Proceedings of the 24th Annual International Conference on Mobile Computing and Networking*, pages 129–144. ACM.

Yao, B., Walther, D., Beck, D., and Fei-Fei, L. (2009). Hierarchical mixture of

classification experts uncovers interactions between brain regions. In *Advances in Neural Information Processing Systems*, pages 2178–2186.

Yi, D., Lei, Z., Liao, S., and Li, S. Z. (2014). Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*.

Zhang, Z., Tran, L., Yin, X., Atoum, Y., Wan, J., Wang, N., and Liu, X. (2019). Gait recognition via disentangled representation learning. In *In Proceeding of IEEE Computer Vision and Pattern Recognition*, Long Beach, CA.

Zhao, S., Song, J., and Ermon, S. (2019). Infovae: Balancing learning and inference in variational autoencoders. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5885–5892.

Zhou, J., Cao, Y., Wang, X., Li, P., and Xu, W. (2016). Deep recurrent models with fast-forward connections for neural machine translation. *arXiv preprint arXiv:1606.04199*.

Ziemski, M., Junczys-Dowmunt, M., and Pouliquen, B. (2016). The united nations parallel corpus v1. 0. In *LREC*.

Zou, Q., Jiang, H., Dai, Q., Yue, Y., Chen, L., and Wang, Q. (2019). Robust lane detection from continuous driving scenes using deep neural networks. *IEEE Transactions on Vehicular Technology*.