Redesigning Deep Neural Networks for Resource-Efficient Inference

A Dissertation

Submitted to the Faculty

of

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Doctor of Philosophy

in

Computer Science

by

Xin Dai

January 13, 2022

APPROVED:

Professor Xiangnan Kong
Worcester Polytechnic Institute
Advisor

Professor Liping Liu
Tufts University
Committee Member

Professor Tian Guo
Worcester Polytechnic Institute
Co-advisor

Professor Ziming Zhang
Worcester Polytechnic Institute
Committee Member

Abstract

Deep learning allows mobile applications to provide novel and useful features. However, the current deep inference paradigm is built on top of utilizing server-centric deep learning models, leading to high demand for computational resources. For the resources-constrained platforms such as mobile devices, the model with low computational cost and memory requirement is more desired. Besides the current studies on model compression methods, we addressed the resource-efficiency problem from the following four aspects:

1) Collaborative inference. Traditional paradigms to support mobile deep inference falls into either cloud-based or on-device—both require access to an entire pre-trained model. As such, the efficacy of mobile deep inference is limited by mobile network conditions and computational capacity. In this study we investigate collaborative inference, a means to split inference computation between mobile devices and cloud servers, to address the limitations of traditional inference through techniques such as image compression or model partition.

2) Recurrent attention model (RAM). As an alternative of expensive CNN, RAM was proposed as a computationally efficient model for CV tasks. In this thesis, we investigate the attention model for classification problems involving multiple ROIs. We design a double RNN architecture to disentangle the potential conflict in RAM, and propose a reward mechanism to train the model using the guidance information of ROIs.

3) Dynamic inference, an emerging technique that reduces the computational cost of deep neural networks. One way to achieve dynamic inference is to leverage multi-branch neural networks that apply different computations on input data by following different branches. In this study, we investigate the problem of designing a flexible multi-branch network and early-exiting policies that can adapt to the resource consumption to individual inference request. We propose a lightweight branch structure that also provides fine-grained flexibility for early-exiting and leverages the Markov decision process (MDP) to automatically learn the early-exiting policies.

4) Resource-efficient Multi-Task Learning. Multi-task learning (MTL) is a promising paradigm for improving the test accuracy of deep learning models that have to train with limited datasets. In this study, we investigate the problem of Resource-efficient Multi-Task Learning (MTL), where the goal is to design a resource-friendly model that suits resource-constrained inference environments. We proposed a novel solution for fine-grained parameter sharing, called FiShNet, which can learn how to share parameters directly on the training data. FiShNet can achieve high accuracy comparable to soft-sharing approaches, while only consuming a constant computational and memory cost per task.

Acknowledgements

I would like to express my gratitude to my advisor Dr. Xiangnan Kong and co-advisor Dr. Tian Guo for their excellent guidance, patience, and support for my research. I would like to thank Dr. Liping Liu and Ziming Zhang for serving on my committee. I would like to thank Yixian Huang and Xinlu He for their contributions in my research projects. My thank you also goes to Xinyue Liu, John Boaz Lee, Menghai Pan, Chong Zhou, Mi Feng, Dongsheng Wang, Jianjun Luo, Hang Yin, Wa Gao and Biao Yin for their support and friendship. Last but not the least, I would like to thank my parents for the unconditional love.

My work was supported in part by National Science Foundation through grants IIS-1718310, CNS-1815619.

Contents

Publications

I provide a list of publications I produced during my Ph.D. studies at WPI. In the following list, the first four papers are discussed in this dissertation.

1. **Xin Dai**, Xiangnan Kong, Tian Guo, Xinlu He. Resource-efficient Multi-task Learning. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management, 2021.*

2. **Xin Dai**, Xiangnan Kong, Tian Guo, Yixian Huang. Redesigning Deep Neural Networks for Efficient Mobile-Cloud Collaborative Inference. In *SIAM International Conference on Data Mining, 2021.*

3. **Xin Dai**, Xiangnan Kong, Tian Guo. Learning to Exit with Flexible Multi-Branch Network. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management, 2020.*

4. **Xin Dai**, Xiangnan Kong, Tian Guo, John Boaz Lee, Xinyue Liu, Constance Moore. Recurrent Networks for Guided Multi-Attention Classification. In *Proceedings of the 26th SIGKDD conference on Knowledge Discovery and Data Mining, 2020.*

5. **Xin Dai**, Xiangnan Kong, John Boaz Lee, Xinyue Liu, Constance Moore. Dual-Attention Recurrent Networks for Affine Registration of Neuroimaging Data. In *SIAM International Conference on Data Mining, 2020.*

List of Figures

List of Tables

1

Introduction

1.1 Motivation

Deep convolutional neural networks (CNNs) have achieved good accuracy on computer vision tasks such as image classification [5, 6, 7], object detection [8, 9, 10], semantic segmentation [11, 12, 13]. Such flourishing can be attributed to the deep multi-layer architectures of CNN, which lead to high model capacity and very strong feature extracting ability. The modern CNNs can easily have tens, hundreds and even thousands layers [6]. So the model can fit and learn on large datasets consisting of millions of samples.

However, the accuracy improvement is often accompanied by higher demand for computational resources. For example, the ResNet-50 has 3.8 billion FLOPs for computational cost and 23 million trainable parameters, and the VGG-16 has 15.3 billion FLOPs and 138 million parameters. In many real-world applications, visual recognition tasks are performed on computationally resource-constrained platforms, e.g. mobile phones, robotics, self-driving car. These platforms either are sensitive to energy cost so can't support intensive computations, or have limited memory space so can't accommodate large models. To utilize these resource-intensive models in deployment scenarios such as mobile devices, prior work proposed techniques such as model pruning [14, 15], low-rank factorization [16, 17], knowledge distillation [18, 19], and efficient CNN [20, 21].

The above previous studies mainly focus on the view of models. I considered the efficient inference problem from a complementary direction, i.e. making use of properties of the tasks and datasets. As the basic inductive biases of our works, we have the following assumptions: 1) **Attention assumption**: the goals of some CV tasks are only relevant to the information in one or some regions of the image. 2) **Instance-difficulty assumption**: for some CV tasks, the difficulty of inference is varying from sample to sample. 3) **Multi-task assumption**: some CV tasks can share the same features.

It is not hard to see how the three assumptions are related to the goal of efficient inference. The traditional neural networks apply the same computation on all regions of an image and all instances of a task. But the attention assumption implies that it is not necessary to scan over the entire image for some tasks. And the instance-difficulty assumption implies that we can spend different computational costs on different instances of a task. Further, the multi-task assumption implies the memory space of task-specific models can be saved by sharing parameters among them. However, some multi-task learning (MTL) techniques may sacrifice the computational efficiency to achieve high performance, which inspires us to study the efficient MTL model.

In this thesis, I addressed the efficient inference problem from the following four topics: *(i)* **Mobile-cloud collaborative inference** (based on attention assumption); *(ii)* **Recurrent attention model** (based on attention assumption); *(iii)* **Dynamic inference** (based on instance-difficulty assumption); *(iv)* **Efficient multi-task learning** (based on multi-task assumption). In the next section, simple introductions of the four topics will be presented.

The three assumptions behind the above topics potentially indicate that the models should have some discrete behaviours. E.g. extracting a sub-region from an image, stopping the forward propagation at a layer, or deciding to share a convolutional filter among several task-specific CNNs. So the models proposed in the four topics can be summaized in a paradigm: an architecture consists of an inference module e.g. a classifier, and another module to perform discrete actions. The learning process of the paradigm may mix the standard back propagation with REINFORCE algorithm [22] or reparameterization trick [23], due to the potential needs of handling non-differentiable functions.

1.2 Topics

1.2.1 Topic 1: Mobile-Cloud Collaborative Inference

To leverage deep neural networks to provide novel features, mobile applications either use powerful cloud servers, i.e., *cloud-based inference*, or directly run them on-device, i.e., *mobile-based inference*. Cloud-based inference allows the use of complex models [5, 6, 24, 25] (thus higher inference accuracy), but requires mobile applications to send a non-trivial amount of data over mobile networks. To use mobile-based inference, one needs to use lightweight models [20, 21, 26]; even so mobile-based inference performance can be hindered by limited on-device resources, e.g., CPU and battery life. To address the limitations of cloud-based and mobile-based inference, an inference paradigm called *collaborative inference* was proposed recently [27, 28]. Collaborative inference allows inference execution to be split between mobile devices and cloud servers, and its goal is to achieve the balance between on-device cost and data transmission cost. The current collaborative inference methods are mainly based on model partition, of which the optimal solutions for reducing on-device cost and data transmission cost are usually not the same. In this thesis, we explore a collaborative inference method based on hard attention mechanism.

1.2.2 Topic 2: Recurrent Attention Model

To achieve sublinear computational costs, many attention-based classification techniques (especially hard attention methods) have been proposed [29, 30]. For example, the Recurrent Attention Model (RAM) [29] is an attention-based model, trained using reinforcement learning (RL), which maintains a constant computational cost w.r.t. the size of input image. The very basic assumption behind RAM is the important contents related to a specific visual task are at one or multiple regions of images. The regions of interest (ROIs) can be found by an agent in a recurrent process. At each recurrent step, the sensor of the agent only accesses partial information. But the history of the past steps can be summarized into a hidden state of the agent, i.e. a global representation which indicates the next ROI to look at and the final result of the task to predict. In this thesis, we study the problem of how to alleviate the overfitting issue of recurrent attention models under the multi-ROI setting for small datasets.

1.2.3 Topic 3: Dynamic Inference by Multi-branch Network

Dynamic inference is an emerging technique that aims to reduce the resource consumption, e.g., computational cost, of deep neural network during inference. A prominent approach of dynamic inference centers around the use of multi-branch networks [31, 32, 33], i.e., networks that consists of more than one output layer, for handling the natural difficulty variations exhibited in real-world samples. Ideally, the multi-branch network spends *just enough computation* for each sample, instead of applying the same amount of computation. We refer to the scenarios of using branch classifiers as *early-exiting*. Existing work on *dynamic inference with multi-branch network* [32, 33] mainly focused on improving the accuracies of the branch classifiers, and used handcrafted policies for deciding the exiting branches. For example, Teerapittayanon et al. devised a rule-based policy that uses the entropy of the logits outputted by the branch classifier as an uncertainty measure [31]. One of the limitations of this rule-based policy is the requirement of domain experts in setting the threshold. In this thesis, we explore a multi-branch network architecture with a trainable early-exiting policy.

1.2.4 Topic 4: Resources-efficient Multi-task Learning

Multi-task learning (MTL) is a promising paradigm to improve the test accuracy of deep learning models that have to train with limited datasets. By allowing task-specific models to share useful information with one another, MTL in essence increases the aggregated training datasets for related tasks [34]. Two common ways to support such information sharing are: *(i) hard-sharing* where parameters are shared among the task-specific networks and *(ii) soft-sharing* where feature maps are shared. Concretely, hard-sharing approaches often employ hand-coded policies which result in static and coarse-grained information sharing. In contrast, soft-sharing techniques can learn fine-grained feature sharing directly from multi-task datasets by providing inputs access to fused feature maps. Consequently, MTL models that achieve state-of-the-art accuracy are often ones using soft-sharing. However, the high accuracy of soft-sharing based MTL models comes with the expensive resource requirement in inference phase that grows linearly with the number of tasks. Such high resource requirement of soft-sharing based MTL models impedes the wide deployment to mobile devices. In this thesis, we discuss how to combine the efficiency advantage of hard sharing and the performance advantage of soft sharing.

1.3 Dissertation Outline

The rest of this thesis is organized as follows.

In the Chapter 2, we approach the problem of collaborative inference, by *considering* *the collaboration requirement from the outset and redesigning the deep neural networks.* We proposed a novel collaborative model named CINET, to balance mobile bandwidth consumption, on-device computational cost, and inference accuracy. In short, CINET consists of a lightweight network on device to extract small but important regions of an input image, and a complex network hosted in cloud to perform the cumbersome part of inference.

In the Chapter 3, we focus on the multi-attention classification problem, where each image involves multiple objects, *i.e.* regions of interest (ROIs). The label of an image is determined jointly by multiple ROIs through complex relationships. Under the setting of multiple ROIs, the recurrent attention model is more prone to overfit the data when the training set is small. To handle this challenge, we proposed a model that utilizes the guidance information for multiple ROIs in each image and works well with small training datasets. We designed a new reward mechanism to utilize both the given ROI locations and the label from each training image. We proposed a novel attention model consisting of two separate RNNs that are trained simultaneously.

In the Chapter 4, we investigate the problem of designing a flexible multi-branch network and early-exiting policies that both can be learned in conjunction from the training dataset. First, we formulate the trade-off between classification accuracy and efficiency as the computational-sensitive classification problem. Then we design a novel multi-branch network structure that provides fine-grained flexibility for early-exiting with negligible resource increase. Last, we formulate the early-exiting problem as a Markov decision process (MDP) and use a policy gradient method without sampling to efficiently train good performant policies.

In the Chapter 5, we investigate the problem of *resource-efficient* multi-task learning with the key goal of designing a resource-friendly MTL model that achieves high accuracy comparable to soft-sharing approaches while only consumes constant resource w.r.t. the number of tasks. We formulate the resource-efficient MTL problem as a *fine-grained filter sharing learning* problem, i.e., learning how to share filters at any given convolutional layers among multiple tasks. To solve this problem, We propose a novel architecture called FiShNet which can be directly implemented and trained with existing deep learning frameworks such as Pytorch. Once trained, FiShNet can effectively support single-task inference scenarios by only loading the task-specific network.

In the Chapter 6, we make the conclusion of this dissertation, and provide a direction for future work.

2

CiNet: A Collaboration-aware Deep Neural Network

2.1 Motivation

To leverage deep neural networks to provide novel features, mobile applications either use powerful cloud servers, i.e., *cloud-based inference*, or directly run them on-device, i.e., *mobile-based inference*, as shown in Figure 2.1. Cloud-based inference allows the use of complex models [5, 6, 24, 25] (thus higher inference accuracy), but requires mobile applications to send non-trivial amount of data over mobile networks, leading to high data transmission. To use mobile-based inference, one needs to use mobile-specific models such as MobileNet, SqueezeNet, or ShuffleNet [20, 21, 26]; even so mobile-based inference performance can be hindered by limited on-device resources, e.g., CPU and battery life.

To address the limitations of cloud-based and mobile-based inference, an inference paradigm called *collaborative inference* was proposed recently [27, 28]. Collaborative inference allows inference execution to be split between mobile devices and cloud servers

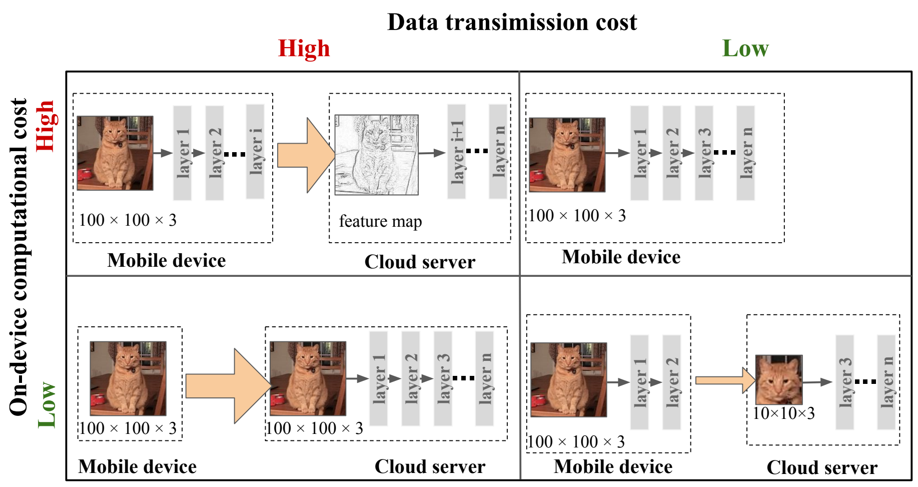


Figure 2.1: ***The problem of mobile-cloud collaborative inference.*** *The goal is to perform image classification on a mobile device by collaborating with a cloud server. The mobile device can send some data to the server to aid in the inference process, which will reduce the computational cost on the mobile device but increase the data transmission cost.*

as demonstrated in Figure 2.1. Prior work on collaborative inference focuses on either reducing network data transmission and the impact on inference accuracy of such reduction or partitioning schemes that split the inference computation across mobile devices and cloud servers [27, 35, 36]. In this work, we approach the problem of collaborative inference from a complementary perspective, by *considering the collaboration requirement from the outset and redesigning the deep neural networks*.

Designing deep learning models that effectively support collaborative inference has the following two key challenges. *First,* the on-device submodel needs to balance mobile bandwidth consumption, on-device computational cost, and inference accuracy. For example, using more complex on-device model structure can effectively reduce the required network data transmission, but can also increase on-device computation. *Second,* both the on-device and cloud submodels should be trained in tandem without requiring additional time-consuming and manual annotations. Prior work on object detection [8, 9, 10] is a potential candidate for detecting image regions to send to the cloud, but often requires access to annotated locations during training [8].

Our design of models that are suitable for collaborative inference is centred around two key insights. *First,* in many real-world scenarios, the results of image classification often only depend on a small image portion. *Second,* the task to identify the important image portion, i.e., extraction, is often easier than the classification. Note our key insights are similar to prior work in dynamic capacity networks [37].

2.2 Problem Formulation

In this section, we first define the problem of *mobile-cloud collaborative inference* and outline key research challenges followed by our design principles.

Mobile-cloud Collaborative Inference. In this chapter, we study the problem of improving the performance, including mobile computational need, mobile data transmission, and inference accuracy, of an emerging paradigm called *mobile-cloud collaborative inference*. At a high level, collaborative inference allows one to split the model computation across mobile devices and cloud servers as demonstrated in Figure 2.1. We approach the problem of efficient collaborative inference by redesigning deep neural networks that are collaboration-aware, unlike existing works in collaborative inference [27, 28, 35, 36]. We focus on the problem of image classification and propose a new neural network design in this work. To use our proposed collaborative inference solution, mobile applications use an on-device sub-model that outputs a smaller-size representation P of the original image I. Afterwards, mobile applications send P to the cloud model server which generates and sends back the predicted label for I.

Key Challenges. One of the key challenges in designing collaborative inference is to achieve low mobile computational and transmission cost *simultaneously* without impacting classification accuracy. Partitioning existing successful deep neural networks, e.g., AlexNet [5], can satisfy accuracy goal but often violate either computational or transmission goals. For example, the first two convolutional layers of AlexNet takes up a large portion of inference computation, making it less ideal to run on mobile devices. Further, the output feature maps of early layers are usually quite large which undermines the mobile-cloud transmission cost.

Design Principles. In designing deep neural networks that are suitable for collaborative inference, we follow the key design principles below: *(i) Reducing mobile computational cost.* The required on-device computation directly impacts the mobile energy consumption, as well as the inference response time. Lower computational cost saves mobile battery life and helps with mobile user experiences. *(ii) Reducing mobile transmission cost.* Similar to computational cost, the required transmitted data also affects mobile energy consumption. Further, it is beneficial to send less data as ways to preserve mobile data plan. *(iii) Achieving comparable classification accuracy.* Last but not the least, we should achieve the computational and transmission goals without sacrificing accuracy as it is important to application utility.

2.3 CiNet: the Proposed Method

2.3.1 Overall Structure

We propose CiNet, an extractor-classifier model that enables efficient mobile-cloud collaborative inference. With low on-device computational complexity and low mobile network data consumption, CiNet can be trained in an end-to-end manner with existing image classification dataset, without additional annotations. As shown in Figure 2.2, the extractor submodel runs on the mobile device and is responsible to extract a smaller-size

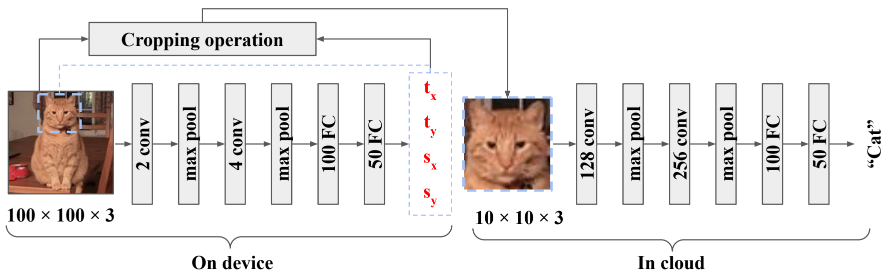


Figure 2.2: *An example of* CINET *structure for mobile-cloud collaborative inference.*

representation P of the original image I. The size of P is predefined during training time, and determines the mobile transmission savings. The classifier submodel runs on the cloud server and is designed to be as complex as needed for the specific classification task to generate labels based on P. In other words, we assume the use of powerful cloud servers that can execute inference requests without imposing latency bottlenecks.

Key Insights. The design of CINET centers two key insights. First, for a specific image classification task, the image usually contains a lot of contents that are irrelevant to its label. In other words, the label-related object may only occupy a small region of the image [29, 30]. For example, to determine whether the image contains a cat or a dog, we often do not have to look at the entire image. Instead, we can only focus on a smaller image region, e.g., faces or eyes. Second, the computational cost to identify the region of important objects can be significantly lower than classification [37]. For example, locating a face in the image often requires fewer filters in the convolutional layers than classifying one. This is because the former only needs to recognize a rough outline whereas the latter requires considering a lot of more details.

2.3.2 On-device Extractor Submodel

The extractor is a convolutional neural network that runs on mobile devices. It consists of several convolutional layers followed by max pooling, hidden fully-connected layers and a final regression layer outputting the transformation parameters $\theta = (t_x, t_y, s_x, s_y)$. Here

t_x and t_y denote a 2D translation, while s_x and x_y denote a 2D scaling. The mapping from the coordinate (x_P, y_P) of the cropped image P to the coordinate (x_I, y_I) of the original image I is parameterized by θ:

$$\begin{bmatrix} x_I \\ y_I \end{bmatrix} = \begin{bmatrix} s_x, 0, t_x \\ 0, s_y, t_y \end{bmatrix} \begin{bmatrix} x_P \\ y_P \\ 1 \end{bmatrix}. \tag{2.1}$$

Based on the above coordinate transformation, CINET performs an image cropping operation on I to obtain the patch P. Then the cloud-based classifier takes P as input and predicts the label of I. Here we focus on describing the extractor network as it directly impacts the mobile computational and transmission cost. To lower the computational cost, we use as few filters as possible in convolutional layers. In our experiments, the extractor network has only two convolutional layers. The first convolutional layer has two filters and the second one has four filters. As demonstrated later in Section 2.4, CINET achieves good classification accuracy even with limited number of convolutional layers. The reason we did not use fully connection layers even though they are often more computational efficient is due to limited mobile memory—fully connected layers have a large number of parameters.

2.3.3 Image Cropping Operations

The On-device extractor yields the transformation parameters θ, indicating which part of the image should be cropped and sent to the cloud-based classifier. The design of the extraction can be viewed as an instance of hard attention mechanism. There are multiple ways to implement hard attention on visual tasks. Here we discuss two approaches to crop the image given θ.

Direct cropping: The transformation parameters θ defines a rectangle region on the original image. The simplest idea is to directly crop this region and then reshape it to our predefined crop shape.

Bilinear sampling function: Given the transformation parameters θ, we can concatenate a spatial transformer layer with bilinear sampling kernel [38] at the end of our extractor submodel. We can then calculate the pixel value $P(x_P, y_P)$ of the cropped patch P with the bilinear mapping from pixel values $I(i, j)$ of original image I:

$$P(x_P, y_P) = \sum_{j=1}^{W} \sum_{i=1}^{H} I(i,j)F(x_I, i)F(y_I, j), \tag{2.2}$$

$$F(a,b) = \max(0, 1 - |a - b|). \tag{2.3}$$

Here W and H are the width and height of the I, and $(x_I\ y_I)$ is the coordinate on I as defined in Equation (2.1). The cropped image P is then sent to the classifier in the cloud. In our current design of CiNet, we adopt this method instead of direct cropping as explained further below in section 2.3.5.

2.3.4 In-cloud Classifier Submodel

The in-cloud classifier takes the cropped image from the mobile device as input and returns the final class of the original image. When designing the in-cloud submodel, we focus on achieving the goal of inference accuracy as it is often safe to assume that cloud servers have ample computational and memory resources. Therefore, one can use existing successful deep neural networks such as AlexNet [5], GoogleNet [25] or ResNet [6] for the in-cloud classifier. When leveraging these existing models, one might need to use an additional image cropping operation, such as the those discussed in Section 2.3.3, to rescale the cropped image to match the predefined input layer size. For simplicity, we designed a new CNN from scratch as shown in Figure 2.2 in CiNet. The input layer of our in-cloud classifier is the same size as the cropped image and therefore only requires a simple identical mapping.

2.3.5 Training Considerations

CiNet consists of the on-device extractor and the in-cloud classifier, forming an end-to-end collaborative neural network. As these two submodels are connected by the image cropping operation, the training algorithm is determined by the chosen cropping operation.

Justifications of Our Image Cropping Choice. Although cropping images with the bilinear sampling function is more complicated than direct cropping, training with it is much easier. This is because the operation is differentiable. In this case, we can train CiNet using the standard back-propagation algorithm under the supervision of image labels [38]. Instead, if choosing to use the direct cropping, one needs to consider and address the problem of propagating the gradient of classification loss from the classifier submodel to the extractor submodel. One possible way is to use the policy gradient method in the form of REINFORCE algorithm [22] to train the extractor, similar to what was proposed by Mnih et al. [29]. However, training with the policy gradient methods can take a long time to converge. As such, we chose to use the bilinear sampling function as the cropping operation when designing CiNet.

Hyperparameters Considerations. In addition to the choice of cropping operations, parameters such as the size of P and specificity of datasets can also complicate the training process. For example, ideally we want to set the size of P as small as possible to reduce the required transmitted data to the cloud. In our current design, we set the size to be

10×10, which is only 1% of the original image size. However, the small extract size means at the early stage of training, P often does not contain meaningful objects. Even worse, for datasets with large black background, e.g., MNIST dataset, it often means sending tensors of zeros to the classifier which further leads to back-propagate gradients of zeros to the extractor. We use two hyper-parameter techniques to mitigate such problems.

- *Weight Initialization.* We initialize all weights in the framework by Gaussian distribution with mean=0 and standard deviation=0.02. However if we use small standard deviation for the final regression layer of the extractor network, the transformation parameters θ of different samples can be very close to each other. Given that we prefer small cropped size, having a larger initial variance of θ could increase the chance to extract the object. As such, it can be helpful to boost the early-stage training. In our experiments, we used 0.2 for standard deviation to initialize the *final* regression layer of the extractor network.
- *Penalty on Scaling Transformation.* When training CINET, we want to avoid the on-device extractor to learn an *easy* way instead of the correct way to extract the objects. In the easy way, the extractor submodel can simply output a large enough scaling transformation that covers the entire original image I. In essence, the cropped image P is merely a downsampled version of I. Such strategies may overfit the training data because important content can be lost in the process of downsampling. In addition, since we chose to design the extractor submodel using very few filters (with the goal to reduce the on-device computational cost), it is therefore more likely for the extractor to learn the easy way. To counter this problem, we add the penalty on scaling transformation into the loss function.

Table 2.1: ***Architecture of CNNs on transformed digit MNIST dataset(left) and Fashion MNIST(right).*** *All CNNs have two fully connected layers with 100 and 50 neurons respectively, before the output layer.*

Method	**Filters in each conv layer**	**Number of conv layers**	**Method**	**Filters in each conv layer**	**Number of conv layers**
CNN1	(128, 256, 256)	3	**CNN1**	(64, 128, 128, 256, 256)	5
CNN2	(64, 128, 256)	3	**CNN2**	(32, 64, 128, 256, 256)	5
CNN3	(32, 64, 128)	3	**CNN3**	(16, 32, 64, 28, 128)	5
CNN4	(8, 64, 128)	3	**CNN4**	(4, 16, 32, 64)	4
CNN5	(8, 16, 64)	3	**CNN5**	(2, 8, 8, 128)	4
CNN6	(4, 8, 32)	3	**-**	-	-
CNN7	(4, 8, 16)	3	**-**	-	-

Table 2.2: ***Architecture of ResNets on CelebA dataset.*** *The ResNets use 1 convolutional layer at beginning, i.e. conv_1, followed by 4 stacks of residual blocks, i.e. conv_2x, ..., conv_5x. The number of filters is doubled at each stack.*

Method	**Number of residual blocks**	**Filters in conv_1 and conv_2x**
ResNet1	(3, 4, 6, 3)	32
ResNet2	(2, 2, 2, 2)	2
ResNet3	(2, 2, 0, 0)	2

2.4 Experimental Evaluations

We evaluate the effectiveness of CINET using three key performance metrics, i.e., classification accuracy, on-device computational cost, and mobile data transmission cost. We compare the performance of CINET to four inference baselines using two transformed MNIST datasets and CelebA dataset (all with JPEG variants). We summarize and highlight our key results below.

- *Accuracy vs. On-device Computational Cost.* Comparing to on-device inference, CINET achieved comparable inference accuracy to the second best CNN while only used 20% computational cost of the fastest CNN. We discuss more details in Section 2.4.2.
- *Accuracy vs. Mobile-Cloud Data Transmission Cost.* CINET significantly reduced the mobile data transmission, incurring only 1% of the cloud-based inference that sent

original image data to the cloud server. When comparing to image compression based techniques including traditional JPEG and DeepN-JPEG [35], CiNet achieved up to 50% higher inference accuracy with similar data transmission cost. More details can be found in Section 2.4.3.

- *On-device Computational Cost vs. Mobile-Cloud Data Transmission Cost.* Comparing to the collaborative inference with model partitions, CiNet incurred lower on-device computation and mobile-cloud data transmission costs for both datasets. We discuss more details in Section 2.4.4.

2.4.1 Experiment Setup

We describe the performance metrics, datasets, models and their hyperparameters, as well as inference baselines we compare to.

Performance Metrics. The computational cost is measured by the number of floating-point multiplication. The data transmission cost is measured by the number of non-zero values sent from the mobile device to the server. We should notice that the three metrics can't be apply for all baselines. For example, on-device inference doesn't send data to sever, so it has no data transmission cost. And in-cloud inference and collaborative inference based on image compression has no or negligible computational cost on device.

Transformed Digit and Fashion MNIST Datasets. We constructed two new datasets from the original digit and fashion MNIST datasets, obtained through TensorFlow and Keras API respectively. Both original MNIST datasets constain 60k images in the training set and 10k in the test data. For each digit/fashion image, we embedded it into the black background of size 100×100 pixels and then performed random scaling and translation of the embedded image. For transformed digit images, we further injected noise that consists of ten 2D sine waves with different phases and frequencies chosen from a Uniform distribution (0, 2π) and a Gaussian distribution ($\mu = 5, \delta = 5$).

CelebA Dataset. We performed experiments on the real-world dataset CelebA. CelebA has 162770 training samples and 19962 test samples. The original CelebA has 40 different labels. In this paper, we only use the "Smiling" label to evaluate our CINET and baselines.

Inference Baselines. We use four different inference approaches, with accompanying convoltuional models summarized in Table 2.1, including two traditional and two collaborative inference mechanisms.

- *On-device inference.* For each transformed MNIST dataset, we trained a series of CNN models with decreasing number of filters, as shown in Table 2.1. We describe common hyperparameter settings below. For CelebA dataset, we compared with three ResNets of different architecture settings, as listed in the Table 2.2, and a simplified MobileNet_V2, with one MobileNet blocks per bottleNeck and expanding rate of 1. These CNN models are assumed to run on mobile devices and are used as baselines for understanding the computational cost and inference accuracy trade-offs.
- *In-cloud inference.* Further, from Table 2.1 we selected the most accurate CNN model, i.e., *CNN1*, for each transformed MNIST dataset, and assume these CNNs to be hosted on the cloud servers. We trained the CNNs on original datasets as well as on the JPEG-decompressed counterparts of original datasets, with both *high* and *low* JPEG quality.
- *Collaborative Inference with DeepN-JPEG [35].* DeepN-JPEG is a neural network-based image compression technique that aims to reduce data transmission while minimizing the impact on accuracy by preserving useful information to classification tasks. To implement DeepN-JPEG, we generated the quantization table used for image compression based on the statistical information of the datasets [35].
- *Collaborative Inference with Model Partitions.* Prior work on model partition techniques focused on identifying the best layer-wise partition point to split the inference

computation across mobile devices and cloud servers [27, 28]. In our evaluations, we tested all possible partition points, i.e., layers, for the most accurate CNNs from Table 2.1. Each partition scheme is labeled as *cut-* followed by the layer name such as *cut-conv3*. A partition scheme defines where the layers are executed. For example, with *cut-conv3* all layers before the third conv layer will be executed on the mobile device and the rest on the cloud server. By evaluating all possible partitions, we can establish the performance of the *optimal* model partition policy, to which we will compare CINET to.

Hyperparameter Settings. Here we introduce the hyperparameters used in our evaluations. *(i) Common settings for* CINET *and baselines*: For experiments on both Transformed Digit and MNIST Datasets, we set the number of epoch = 12, batch size = 64, decay rate of leaning rate is 0.1. We adopt the dropout for all models. For both CINET and CNNs, the size of filter window for convolution is 5×5 and 2×2 for pooling. *(ii)* CINET *setting*: The initial leaning rate = 0.01. On the MNIST and fashion MNIST, The extractor has two convolutional layers, each followed by a max-pooling layer. The convolutional layers have very few filters, which are 2 in the first layer and 4 in the second. There is a fully connected hidden layer consists of 50 neurons before final regression layer. The size of cropped image sent to classifier in the cloud is 10×10. On the CelebA, the extractor is same with the MobileNet baseline. We use bilinear sampling to crop image. The weight of penalty on scaling transformation is 0.1. The classifier of the CINET also has two convolutional layers, of which the number of filters are 128 and 256. The convolutional layers are followed by two fully connected layers, of which the number of neurons are 100 and 50. All the weights in both the extractor and the classifier are initialized from the Gaussian distribution of which the mean is zero and the standard deviation is 0.02, The final regression layer of extractor uses the standard deviation of 0.2. *(iii) Baseline setting*: We set the initial learning to 0.1 for all CNNs listed in Table 2.1 except

Table 2.3: ***Results on transformed MNIST dataset.*** *For in-cloud deployment, the model used is the CNN1 in Table 2.1(left). We use JPEG(H) and JPEG(L) to denote the high quality and low quality used to compressed the images, respectively.*

Deployment	**Method**	**Test Accuracy (%)** Higher better	**Computational cost on device (MFLOPS)** Lower better	**Transmission cost (# non-zero integer)** Lower better
On-device	CNN1	**96.20**	3060	-
	CNN2	93.98	1040	-
	CNN3	93.52	264	-
	CNN4	92.22	162	-
	CNN5	91.73	26	-
	CNN6	86.39	7	-
	CNN7	83.52	5	-
In-cloud	Original	96.20	-	10000
	JPEG(H)	95.79	-	1343
	JPEG(L)	40.15	-	116
	DeepN-JPEG(H)	96.07	-	1298
	DeepN-JPEG(L)	42.20	-	119
Collaborative	**CiNet**	93.74	**1**	**100**

CNN4 and CNN5 on the right tablular; these two CNNs used an initial learning rate is 0.01.

2.4.2 CiNet vs. On-Device Inference

We study the computation and accuracy trade-offs on all three datasets by comparing CiNet to different deep learning models (see Table 2.1 and Table 2.2). To evaluate their computational costs, we assume all CNNs will run on mobile devices, i.e., executing these models will not incur any mobile-cloud data transmission cost.

Table 2.3 compares both on-device computational cost and average inference accuracy between all baseline CNNs and CiNet on transformed MNIST dataset. As expected, the achieved inference accuracy decreases with the computational cost for the baseline CNNs. However, CiNet struck the balance between the on-device computational cost and inference accuracy with the help of the in-cloud classifier submodel. For example, CiNet incurred the lowest computational cost at about 20% compared to the fastest *CNN7*

Table 2.4: ***Results on transformed fashion MNIST dataset.*** *For in-cloud deployment, the model used is the CNN1 in Table 2.1 (right).*

Deployment	**Method**	**Test Accuracy (%)** Higher better	**Computational cost on device (MFLOPS)** Lower better	**transmission cost (# non-zero integer)** Lower better
On-device	CNN1	**82.54**	3060.	-
	CNN2	82.38	1000	-
	CNN3	80.65	122	-
	CNN4	78.63	21	-
	CNN5	66.47	6	-
In-cloud	Original	82.54	-	10000
	JPEG(H)	82.13	-	547
	JPEG(L)	78.58	-	131
	DeepN-JPEG(H)	79.97	-	509
	DeepN-JPEG(L)	77.75	-	152
Collaborative	**CiNet**	82.29	**1**	**100**

and at merely 0.5% compared to *CNN3* whose accuracy was 0.22% lower.

We also observed similar trends when evaluating on the transformed fashion MNIST dataset. Table 2.4 shows that the computational cost of CiNet was only 0.001% and 0.002% of that of CNNs (i.e., *CNN1* and *CNN2*) with comparable inference accuracy, by 0.34% and 0.18%, respectively. Further, CiNet only incurred about 14.7% computational cost compared to the fastest *CNN5* but achieved 16% better accuracy.

Lastly, we compared CiNet to ResNet and MobileNet_V2 on the CelebA dataset, containing more complicated images of human faces. Table 2.5 shows that the computational cost of CiNet was only 0.04% and 0.16% of that of ResNets (i.e., *ResNet1* and *ResNet2*) with comparable inference accuracy, by 1.01% and 0.22%, respectively. Further, CiNet has similar computational cost compared to the fastest on-device model *MobileNet_V2* but achieved 4.06% better accuracy.

In summary, these results support our hypothesis that finding the essential content for classification can be much more computational efficient than directly classifying the entire image. Deploying the extractor submodel instead of a complete CNN model on-device can reduce the computational complexity by two to three orders of magnitude,

Table 2.5: ***Results on CelebA dataset.*** *For in-cloud deployment, the model used is the ResNet1 in Table 2.2.*

Deployment	**Method**	**Test Accuracy (%)** Higher better	**Computational cost on device (MFLOPS)** Lower better	**Transmission cost (# non-zero integer)** Lower better
On-device	ResNet1	**92.04**	21.32	-
	ResNet2	91.25	5.58	-
	ResNet3	87.18	3.14	-
	MobileNet_V2	86.97	0.9	-
In-cloud	Original	92.04	-	38804
	JPEG(H)	91.15	-	3136
	JPEG(L)	87.62	-	1295
	DeepN-JPEG(H)	91.07	-	2980
	DeepN-JPEG(L)	88.44	-	1163
Collaborative	**CiNet**	91.03	**0.9**	**256**

with negligible accuracy loss.

2.4.3 CiNet vs. Image Compression Based Inferences

We focus on evaluating the trade-offs between inference accuracy and mobile-cloud data transmission. The baseline CNNs run in the cloud server and therefore do not incur on-device computational cost.

Table 2.3 compares the performance of *CNN1* from Table 2.1 and CiNet on the transformed digit MNIST dataset. CiNet, with a crop size of 10×10, only incurred 1% of data transmission cost but at the cost of 2.46% lower accuracy when comparing to sending original images to *CNN1*. Further, we compare CiNet with two image compression techniques, i.e., traditional JPEG algorithm and a deep learning based compression called DeepN-JPEG [35]. For JPEG, we chose two quality levels of 50 and 0.5 as the former is a common configuration and the latter achieves the same data transmission cost as CiNet. We then trained the in-cloud *CNN1* with the images that were first compressed with the corresponding JPEG configuration and then decompressed. We followed the same strategy described above to choose two quality levels 50 and 0.15 for DeepN-JPEG and then trained with *CNN1* again.

Table 2.3 shows that using the *CNN1* trained with higher JPEG quality only incurred 10% data transmission cost with a decrease of 0.41% in inference accuracy, when compared to sending the original images. This result supports the common sense that JPEG can preserve the essential information with about 10% compression rate. However, for *CNN1* trained with images of lower JPEG quality, its accuracy was 50% lower than that of CiNet. Such accuracy loss can be attributed to the background 2D sine noises. We can further observe that DeepN-JPEG achieved an increase of 2.2% in accuracy with similar data transmission cost when comparing to JPEG of similar configurations. Similarly, the accuracy of *CNN1* trained with DeepN-JPEG(L) was again 50% lower than CiNet while incurring the same data transmission cost.

We also observed similar results on the transformed fashion MNIST dataset. For example, Table 2.4 shows that the accuracy of CiNet was 0.34% lower than the *CNN1* trained on original image, but with only 0.01% of the data transmission cost. When trained with images compressed and decompressed with higher JPEG quality, the accuracy of *CNN1* was 0.27% lower than that of *CNN1* trained on original images. For *CNN1* trained with lower JPEG quality, it incurred the same data transmission cost but 3.63% lower accuracy than CiNet. Interesting, classification-aware DeepN-JPEG had slightly lower accuracy than JPEG for this dataset. As one of the key differences between these two datasets is the existence of low-frequency noise, such accuracy discrepancy might be caused by DeepN-JPEG's ability to remove such noise.

Lastly, we compared the CiNet to the baselines on the CelebA dataset. Table 2.5 shows that the data transmission cost of CiNet was only 0.006%, 0.081% and 0.085% of that of ResNets trained on original images, and images transmitted by high-quality JPEG and DeepN-JPEG, with neglectable loss of accuracy, by 1.01%, 0.12% and 0.04%, respectively. Using JPEG(L) and DeepN-JPEG(L), the accuracy are 3.41% and 2.59%

lower than CINET, and still have about 5X data transmission costs.

In summary, these results suggest that lower compression quality can lead to the loss of important content for classification. Further, the performance differences can also be attributed to the design goals of JPEG and CINET. JPEG aims at recovering the whole image whereas CINET aims at finding the important content for classification. As such when there are a lot of label-irrelevant contents, JPEG still has to try to recover them whereas CINET can simply avoid them. CINET demonstrated its ability to achieve lower data transmission cost and higher inference accuracy, when compared to image compression based techniques.

2.4.4 CINET vs. Collaborative Inference with Model Partition

We focus on comparing the on-device computation and the data transmission cost between CINET and the optimal model partition.

Table 2.6 shows the performance of *CNN1* from Table 2.1 and CINET on the transformed digit MNIST dataset. As this *CNN1* consists of three convolutional (conv) layers, two fully-connected (fc) hidden layers, and a final classifier layer, we evaluated the performance under all five layer-wise partition schemes. As we can see, the on-device computational cost increased as the partition point moves toward the output layer, with a significant jump from the first partition scheme, i.e., *cut-conv1*, to the next, i.e., *cut-conv2*. The latter in the CNN the partition point is, the more computation it incurs. This is expected as more model layers need to be executed on the mobile device. However, we observe a different trend with the data transmission cost. Specifically, the mobile data transmission cost lowered as the partition point moves toward the output layer. Again, this is expected as feature maps of later conv layers have lower dimension and the last fully connected layer only generates data as little as what are needed for classification labels. In short, the optimal partition point for achieving the lowest computational cost, i.e., *cut-conv1*, and for the lowest data transmission cost, i.e., *cut-fc2*, can not be achieved at the same time under the model partition approach. In constrast, CINET was able to balance these two design goals, achieving low data transmission cost (as low as *cut-fc1*) and an order of magnitude lower computational cost than *cut-conv1*. We observe similar benefits of CINET on the Transformed fashion MNIST dataset.

In summary, CINET demonstrated its ability to balance on-device computation and mobile data transmission cost, when compared to model partition techniques.

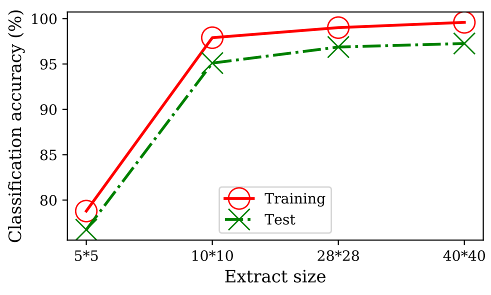
2.4.5 Impact of Hyperparameters

Lastly, we evaluate the impact of two important hyperparameters on CINET's inference accuracy.

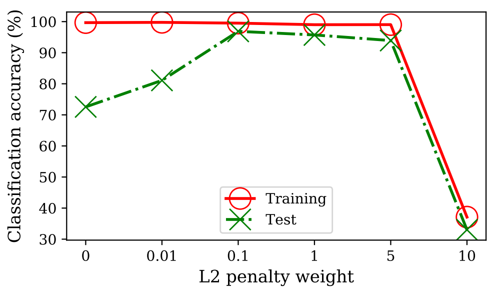
- **Extraction Size** specifies the size of the on-device extractor submodel's output and directly impact the required data to send to the cloud. Figure 2.3(a) shows that both the training and the test accuracy increase with the extraction size. This demonstrates the importance of setting *sufficiently large* extraction size as we observed underfitting with smaller extraction size. However, naively increasing extraction size is not ideal as larger extraction size leads to higher data transmission cost. We also observed that both accuracies plateaued at extraction size of 10×10, which can save up to 16X data transmission cost compared to larger extraction sizes. Our results suggest the need to carefully tune the extraction size to trade-off between data transmission cost and inference accuracy under different application scenarios.
- **L2 Penalty Weight** controls how aggressive the on-device extractor submodel scales up the mapped region in the original image. Figure 2.3(b) shows that models can overfit, indicated by the large gap between training and test accuracy, without using L2 penalty. This is because the on-device submodel is more likely to output a low-resolution version of the original image by scaling transformation to cover the whole

Table 2.6: ***Comparisons with model partitions on the transformed digit MNIST dataset(left) and fashion MNIST(right).*** *We partitioned the CNN, with the structure of each of the* CNN1 *in the left and right of Table 2.1, at all possible layer-wise partition points. The computational (Comp) and data transmission (trans) cost are shown.*

Partition Point	Comp Cost	Trans Cost	Partition Point	Comp Cost	Trans Cost
conv1	32	320000	**conv1**	16	160000
conv2	2040	160000	**conv2**	528	80000
conv3	3060	43200	**conv3**	784	20000
fc1	3060	100	**conv4**	922	12500
fc2	3060	50	**conv5**	1000	4096
-	-	-	**fc1**	1000	100
-	-	-	**fc2**	1000	50
CINET	1	100	**CINET**	1	100



(a) *Extraction size*



(b) *L2 penalty weight.*

Figure 2.3: *Impacts of Hyper-parameters on classification accuracy.*

image. We observe that with the weight of penalty of 0.1, both the training and test accuracy were at their respective high. This is because larger penalty forces the extractor submodel to focus on cropping small region. However, if the penalty is too large, the extractor might have troubles with larger region of interests. In summary, our results suggest the importance of choosing reasonable penalty to help the extractor submodel to crop the important label-related image content.

2.5 Related Work

Collaborative Inference with Model Partition. Han et al. [39] proposed to generate a resource-efficient variant for a given network, then provide a run-time system called MC-

DNN which split the generated network into two fragments and execute them each on mobile device and cloud server. Kang et al. [27] developed Neurosurgeon, an automatic model partition scheme that can adapt to different hardware environments and model structures. Li et al. [28] proposed to quantize the on-device model partition to further reduce the computational and memory requirement on the mobile device. Our design of the collaboration-aware models can be regarded as a way to partition a new model optimally.

Image Compression. JPEG, as one of the widely used image compression algorithm, allows adjusting quality of compression to trade-off between image size and quality. It typically achieves 10:1 compression with little perceptible loss in image quality. A number of recent works explored the use of convolutional neural network [40] or autoencoder [41] as an alternative to compress image data. However, these works often only care about recovering the original image content without concerning the impact on the classification. Recently, Xie et al. proposed to rework traditional image compression algorithm JPEG through modifying quantization table based on the gradients of neural network-based image classifier [36]. Similarly, Liu et al. proposed a dataset aware compression algorithm that adjusts the quantization table with the statistical information of training dataset [35]. Our work shares similar design goal with the recent classification-aware compression algorithm for reducing network data transmission without impacting classification accuracy.

Hard Attention Mechanism. The hard attention mechanism that aims at reducing the computational and memory cost shares similar goals with our work. Minh et al. proposed a recurrent attention model [29] on visual learning tasks that leverages REINFORCE algorithm for training. Similarly, CINET also uses REINFORCE algorithm if the direct cropping operation is used. Jaderberg et al. proposed spatial transformer network (STN) that performs spatial transformation at any feature map [38]. CINET follows similar design when using bilinear image sampling to crop image. However, CiNet differs from STN in that STN was designed to learn invariance to transformation while CiNet focuses on reducing computational and transmission cost for collaborative inference.

3

Recurrent Networks for Guided Multi-Attention Classification

3.1 Motivation

Recurrent Attention Model (RAM) [29] is an attention-based model, trained using reinforcement learning (RL), which maintains a constant computational cost w.r.t. the number of image pixels for image classification. RAM moves its visual attention sensor on the input image and takes a fixed number of glimpses of the image at each step. RAM has demonstrated superior performance on high-resolution image classification tasks, making a strong case for the use of attention-based methods under this setting.

In this chapter, we focus on the multi-attention classification problem, where each image involves multiple objects, *i.e.* regions of interest (ROIs). The label of an image is determined jointly by multiple ROIs through complex relationships. For example, in brain network classification, each fMRI scan contains multiple brain regions whose relationships with each other may be affected by a neurological disease. In order to predict whether a brain network is normal or abnormal, we need to examine the pairwise rela-

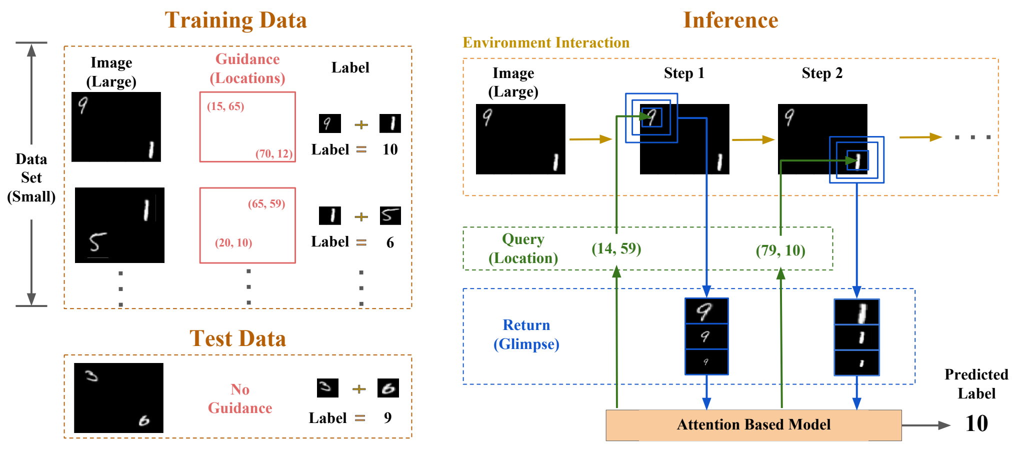


Figure 3.1: *An example of* ***the guided multi-attention classification problem****. Each image contains two written digits (ROIs) at varying locations. The label of the image is determined by the sum of the two digits,* e.g. *the label* $10 = (9 + 1)$*. The locations of the digits are provided as guidance to the system in the* small *training set, but are* not *available during inference. An attention-based model moves its visual sensor (controlled by a policy function) over the image and extracts patches (glimpses) to predict the image label.*

tionships between different brain regions. If we focus on just a single brain region, we may not have enough information to correctly predict the brain network's label. Many other visual recognition tasks also involve multiple ROIs, as illustrated in Figure 3.1.

Current works on attention-based models largely assume that a large-scale training set (*e.g.*, millions of images) is available, making it possible to learn ROI locations automatically. However, in many applications like medical imaging, only a small number of training images are available. Such applications raise two unique challenges for attention-based models: (1) It is usually hard to learn the locations of the ROIs directly from the data. (2) Even if the models manage to find the ROIs given the small number of samples, the models can easily overfit, as demonstrated in Figure 3.2.

One of our key insights is that by learning the locations of the ROIs in addition to the content inside each ROI, an attention-based model can achieve higher accuracy even with small-scale training set. Fortunately, in many applications with a small number of training samples, it is usually possible for human experts to provide the locations of the ROIs, *e.g.*, locations of brain regions. In this paper, we studied a new problem called

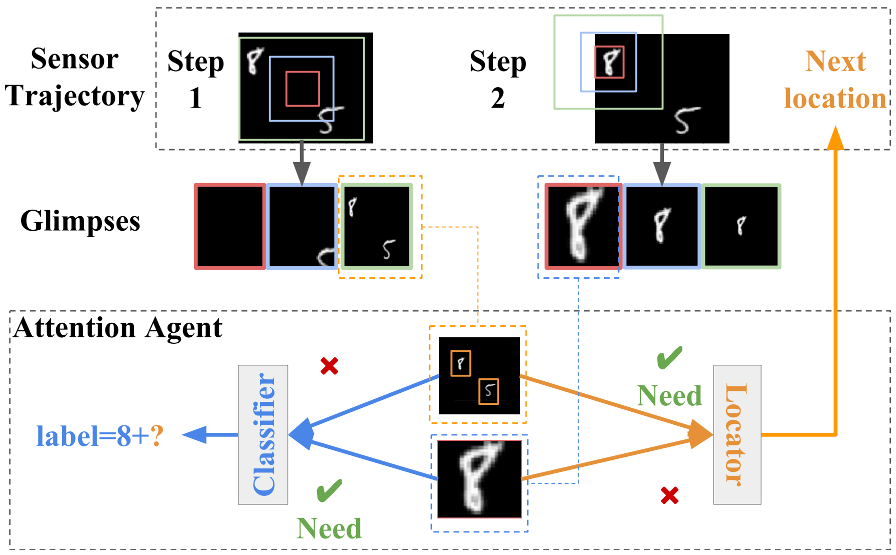


Figure 3.2: ***The unique challenge of attention-based classification with only a small number of training samples.*** *A classifier will overfit if it learns to use the locations instead of the contents of ROIs. To prevent overfitting, a classifier should avoid "memorizing" locations in a low-resolution glimpse and focus on the high-resolution glimpse. Meanwhile, a "locator" network should utilize the low-resolution glimpse to determine where to move the sensor next.*

guided multi-attention classification, as shown in Figure 3.1. The goal of guided multi-attention classification is to train an attention-based model on a small-scale dataset by utilizing the guidance, *i.e.*, the locations of ROIs in each image, to avoid overfitting.

Despite its value and significance, the guided multi-attention classification has not been studied in this context so far. The key research challenges are as follows:

Guidance of Attention: One key problem is how to learn a good policy using the guidance information (*i.e.*, ROIs' locations). Such guidance is *only* available during training which requires careful design to ensure that the model still performs well without it at inference time. Moreover, there can be a large number of possible trajectories covering these ROIs in each training image.

Limited number of samples: Conventional attention-based models usually require a large dataset to train the attention mechanism. With small datasets, the attention-based models can easily overfit by using the locations of ROIs instead of the contents in each region to build a classification model. As shown in Figure 3.2, to avoid overfitting, the classifier of the attention-based model should avoid using the low-resolution glimpse, *i.e.*, containing the ROI locations, but instead focus on the high-resolution glimpse, *i.e.*, containing the content of each ROI. On the other hand, the "locator" network which determines where the sensor should move next, should use the low-resolution glimpse instead.

In this chapter, we propose a model, called Guided Attention Recurrent Network (GARN), for the multi-attention classification problem. Different from existing attention-based methods, GARN utilizes the guidance information for multiple ROIs in each image and works well with small training datasets. We designed a new reward mechanism to utilize both the given ROI locations and the label from each training image. We proposed a novel attention model consisting of two separate RNNs that are trained simultaneously. Empirical studies on three different visual tasks demonstrate that our guided attention approach can effectively boost model performance for multi-attention image classification.

3.2 Problem Formulation

In this section, we formally define the multi-attention classification problem. We are given a small set of N training samples $\mathcal{D} = \{(\mathbf{I}_i, \mathcal{R}_i, y_i)\}_{i=1}^{N}$. Here, $\mathbf{I}_i \in \mathbb{R}^{W\times H\times C}$ denotes the i-th image with dimensions $W \times H \times C$ and label $y_i \in \mathcal{L}$. Furthermore, $\mathcal{L}$ represents the label space, *i.e.*, $\{0, 1\}$ for binary classification, and $\{1, \cdots, N_c\}$ for multi-class classification, where N_c is the number of categories. $\mathcal{R}_i = \{\ell_{ij}\}_{j=1}^{n_i}$ is a set of locations of the ROIs in image $\mathbf{I}_i$. Here $\ell_{ij} = (x_{ij}, y_{ij}) \in \mathbb{R}^2$, where $0 \le x_{ij} \le W$ and $0 \le y_{ij} \le H$, indicates the center of the j-th ROI in the i-th image. The label y_i is only determined by the objects/contents within these ROIs.

Region of Interest (ROI): In the multi-attention classification problem, each ROI is a part of the image that contains information pertinent to the label of the image. For instance, in an fMRI image of the human brain, each ROI is one of the brain regions related to a certain neurological disease.

The goal of multi-attention classification is to learn a model $f : \mathbb{R}^{W \times H \times C} \mapsto \mathcal{L}$. Specifically, we are interested in learning an attention-based model, which interacts with a test image $\mathbf{I}$ that iteratively extracts useful information from a test image through multiple steps. In each step, the attention model obtains a glimpse, *i.e.*, patch, $\mathbf{X}_t$ of the image $\mathbf{I}$ around a queried location. The attention-based model contains a policy function for visual attention $\pi(\mathbf{h}_t) = (x_{t+1}, y_{t+1})$. Here, $\mathbf{h}_t$ represents the hidden state of the model at the t-th step of interaction with the image while (x_{t+1}, y_{t+1}) represents the location where the attention mechanism wants to obtain the next glimpse, at step $t+1$, on the test image $\mathbf{I}$.

In this paper, we focus on studying the guided multi-attention classification problem, which has the following properties: (1) training set size (*i.e.*, $|\mathcal{D}|$) is small; (2) image size is large; (3) the class label of each image is related to multiple ROIs – for instance, the sum (label) of multiple digits (ROIs) in an image, or the correlation (label) between the activities of different brain regions (ROIs) in an fMRI scan; and (4) ground-truth locations of ROIs are only provided for a small training set.

3.3 Our Proposed Method: GARN

3.3.1 RAM Background

Our proposed approach is inspired by the RAM model introduced by Mnih et al. [29]. In RAM, an RL agent interacts with an input image through a sequence of steps. At each step, guided by attention, the agent takes a small patch (or glimpse) of a certain part of the image. The model then updates its internal state with the information provided by the observed glimpse and uses this to decide the next location to focus its attention on. After several steps, the model makes a prediction on the label of the image. Overall, RAM consists of a glimpse network, a core network, a location network, and an action network.

• **Glimpse network** takes a sensor-provided glimpse, $\mathbf{X}_t$, of the input image at time t and encodes it into a "retina-like" glimpse representation, $\mathbf{x}_t$.

• **Core network** is a recurrent neural network. It obtains a new internal state by taking the glimpse representation and combining this with its current internal state. The internal state is a hidden representation which encodes the history of interactions between the agent and the input image.

• **Location network** takes the internal state at time t and outputs a location, ℓ_t, which is where the sensor will be deployed at the next step. Each location, ℓ_t, is assigned a corresponding task-based reward.

• **Action network** takes the internal state at time t as input and generates an action a_t. When RAM is applied to image classification, only the final action, which is used to predict the image label, is utilized. The action earns a reward of 1 if the prediction is correct, otherwise reward is 0.

The t-step agent's interactions with the input image can be denoted as a sequence $S_{1:t} = (\mathbf{x}_1, \ell_1, a_1, \mathbf{x}_2, \ell_2, a_2, \cdots, \mathbf{x}_t)$. RAM learns a function which maps $S_{1:t}$ to a distribution over all possible sensor locations and agent actions. The goal is to learn a policy which determines where to move and what actions to take that maximizes reward.

3.3.2 Dual RNN Structure

Conventional attention-based methods tend to rely on large-scale datasets for training. However, in many real-world applications, such as medical imaging, the number of available images can be relatively small. For instance, the neuroimaging dataset that Zhang et al. [42] studied had less than a hundred samples. As we illustrated in Figure 3.2, training attention-based methods on smaller scale training data leads to some unique challenges.

Our key insight is as follows. Instead of trying to learn the locations of the various

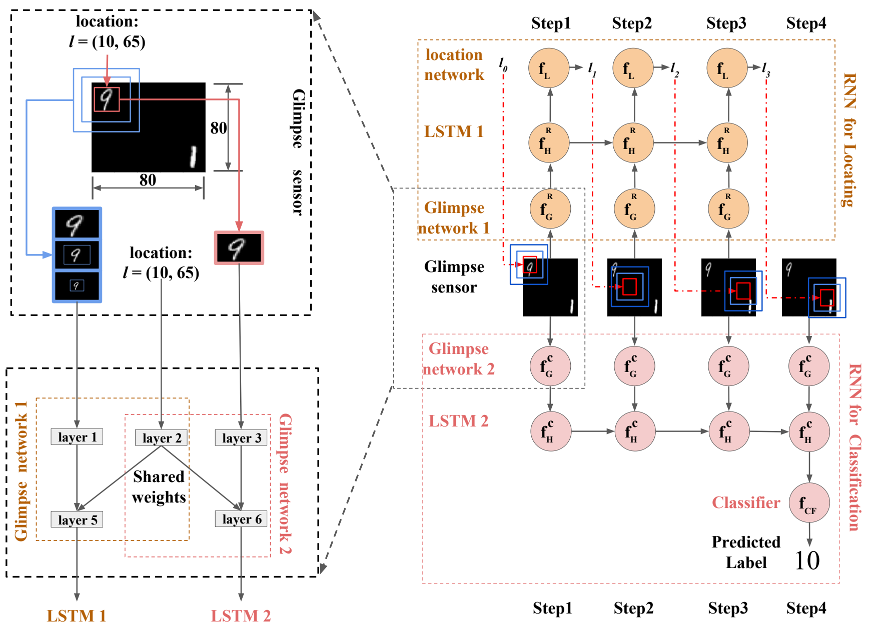


Figure 3.3: ***GARN overview.*** *The proposed GARN model consists of two RNNs, one for locating ROIs and the other for classification. The glimpse sensor extracts several image patches of different scales and feeds them to two glimpse networks, f_G^R and f_G^C. f_G^R is the glimpse network of the RNN which locates ROIs while f_G^C belongs to the classification RNN. The glimpses fed to both f_G^R and f_G^C are from the same location given by the network f_L with a potentially different number of glimpse scales.*

ROIs as well as the relevant content in each of the ROIs using a single network, like conventional approaches, we divide this process into two connected sub-processes. To make the most of the small number of training images and to fully leverage the power of expert-provided guidance (*e.g.*, locations of ROIs), we design a guided multi-attention model with two complementary RNNs (see Figure 3.3). The first RNN is used to locate ROIs in the image while the second one is used solely for classification. While the two RNNs take patches of an image at the same position as input, we expect them to remember different things about the input due to a difference in their function.

We now introduce our proposed model architecture. In the subsequent discussions, we will use the same notations as [29]. Let Linear$(\mathbf{x})$ denote a linear transformation $\mathbf{W}^\top\mathbf{x} + \mathbf{b}$ with weight matrix $\mathbf{W}$ and bias $\mathbf{b}$. On the other hand, Rect$(\mathbf{x}) = \max(\mathbf{x}, 0)$ denotes the ReLU activation.

3.3.2.1 RNN for Locating ROI

Our RNN for locating ROIs consists of four parts: glimpse sensor, glimpse network, core network, and location network.

• **Glimpse sensor:** Given an image $\mathbf{I}$, a location $\ell = (i, j)$ and a glimpse scale s, the sensor extracts s square patches $\mathbf{P}_m$, for $m = 1, \cdots, s$, centered at location (i, j). The side of the $(m+1)^{th}$ patch is twice that of the m^{th} patch. All s patches are then scaled to the smallest size, concatenated, and flattened to a vector $\mathbf{x}$.

• **Glimpse network (f_G^R):** As shown in Figure 3.3, the glimpse network is composed of 3 fully connected (FC) layers: (1) the first FC layer encodes the sensor signal $\mathbf{x}$: $\mathbf{x_h} = \text{Rect}\left(\text{Linear}\left(\mathbf{x}\right)\right)$; (2) the second FC layer encodes the location of the sensor ℓ: $\ell_\mathbf{h} = \text{Rect}(\text{Linear}(\ell))$; (3) the third FC layer encodes the concatenation of $\mathbf{x_h}$ and $\ell_\mathbf{h}$: $\mathbf{g} = \text{Rect}(\text{Linear}(\mathbf{x_h}, \ell_\mathbf{h}))$. The glimpse representation $\mathbf{g}$ is the output of f_G^R.

• **Core network (f_H^R):** Given the glimpse representation $\mathbf{g}_t$ and hidden internal state $\mathbf{h}_t$ at time step t, the core network updates the internal state using the following rule: $f_H(\mathbf{g}_t, \mathbf{h}_t) = \mathbf{h}_{t+1}$. The hidden state $\mathbf{h}_{t+1}$ now encodes the interaction history of the agent up to time t. We use basic LSTM cells to form f_H.

• **Location network (f_L):** At time step t, the next location ℓ_t is stochastically determined by the location network. We assume that ℓ_t is drawn from a 2D Gaussian distribution. The Gaussian distribution's mean vector $\boldsymbol{\mu}$ is outputted by the location network f_L, which is a fully connected layer $\boldsymbol{\mu}_t = \text{Tanh}\left(\text{Linear}\left(\mathbf{h}_t\right)\right)$. The covariance matrix is assumed to be fixed and diagonal.

3.3.2.2 RNN for Classification

This RNN also consists of four parts: glimpse sensor, glimpse network, core network, action network.

• **Glimpse sensor:** It is similar to the glimpse sensor above, and the two sensors look

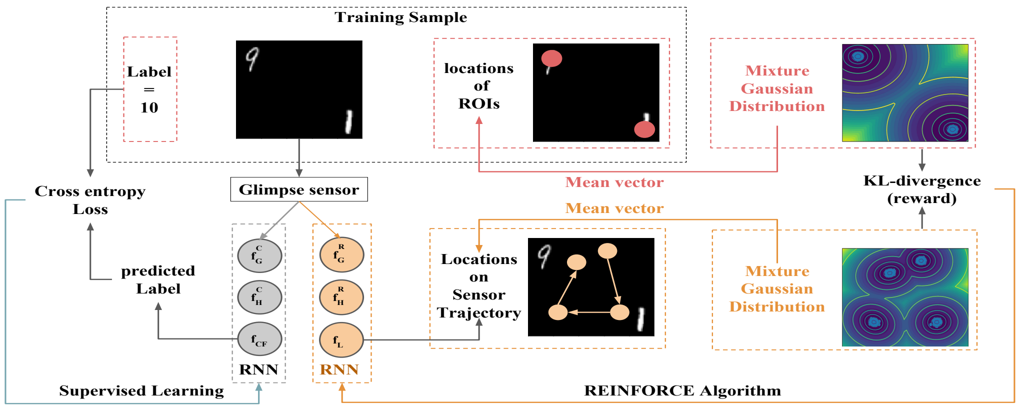


Figure 3.4: ***Training overview.*** *The proposed GARN model consists of two RNNs that are trained simultaneously. The RNN for classification is trained using cross-entropy loss. Meanwhile, we trained the RNN for locating ROIs using the KL divergence between two Mixture Gaussian distributions as the reward for the REINFORCE algorithm.*

at the same position at each step. However, in this paper, we use a dual-scale sensor for classification while a triple-scale sensor is used for finding ROIs. Intuitively, this is because the classifier only needs the higher resolution glimpses while the "locator" RNN may benefit from the lower resolution glimpse which covers a wider area.

• **Glimpse network** (f_G^C)**:** Similar to f_G^R, f_G^C is also composed of three FC layers with similar functions. The FC layer to encode location is shared with f_G^R. However, f_G^C does not share weights with f_G^R for the other two FC layers. This is because the glimpse image here has 1 or 2 scales while f_G^R takes an image with 3 scales.

• **Core network** (f_H^C)**:** The same as f_H^R, but their weights are not shared. f_H^C combines the output of f_G^C at the current step with the previous hidden state to obtain a new hidden state.

• **Action network** (f_{CF})**:** Takes the last hidden state $\mathbf{h}_n^R$ as input and outputs a label prediction. The action network $f_{CF}(\mathbf{h}_n) = \mathbf{a}_p$ is a three-layer fully connected network with ReLU activations for its hidden layers.

3.3.3 Reward and Training

The interaction between our model and an image (Figure 3.4) can be denoted by two sequences. The first, $\mathbf{S}^R_{1:n} = \big(\mathbf{x}^R_1, \ell_1, \mathbf{x}^R_2, \ell_2, \cdots, \mathbf{x}^R_n\big)$, is generated by the RNN for finding ROIs while the second, $\mathbf{S}^C_{1:n} = \big(\mathbf{x}^C_1, \ell_1, \mathbf{x}^C_2, \ell_2, \cdots, \mathbf{x}^C_n, \mathbf{y}\big)$, is encoded by the classification RNN. We can view this as a case of Partially Observable Markov Decision Process [29]. Here, the true state of the environment is static but unknown.

The RNN for classification is trained using cross-entropy loss which is commonly used in supervised learning. Here we mainly discuss the training of the second RNN. We use θ to denote the parameters of the RNN (*i.e.*, f^R_G, f^R_H and f_L). The goal is to learn a policy $\pi(\ell_i|\mathbf{S}^R_{1:i-1};\theta)$ that maximizes the expectation of reward:

$$J(\theta) = \mathbb{E}_{p(\mathbf{S}^R_{1:n};\theta)}\left[\sum_{i=1}^{n} r_{\ell_i|\mathbf{S}^R_{1:i-1}}\right] \tag{3.1}$$

3.3.3.1 Reward

We denote $r_{\ell_i|\mathbf{S}^R_{1:i-1}}$ as the reward for the generated location at the i-th step. Originally, in [29], all rewards $r_{\ell_i|\mathbf{S}_{1:i}}$ are set to 1 if the classification is correct, otherwise a uniform reward of 0 is given. However, such assumptions can be problematic when training with only a small number of images, *e.g.*, the model can get high reward by overfitting the training sample without seeing the true ROIs. To mitigate such problem, we designed a reward function based on the ground truth ROI locations:

1. Construct two mixture Gaussian distributions P_1 and P_2, of which the mean vectors correspond to the locations in f_L and the ground truth locations of ROIs, respectively. The standard deviations are hyperparameters, and we used 0.2 by default.

2. The reward in the Equation (3.1) is the negative of the Kullback-Leibler divergence between P_1 and P_2, which is commonly used for estimating the difference between two

Table 3.1: *Summary of experimental datasets.*

CharacteristicTask	Comparing two digits	Adding two digits	Brain network classification
Dataset size	2k-20k	2k-20k	2k-8k
Feature size	80×80	80×80	$91 \times 91 \times 10$
Number of classes	2	19	2
Ratio of the dominant class	0.5	0.09	0.5
Number of ROIs	2	2	4

distributions.

$$D_{kl}(P_1||P_2) = \sum_i p_1(i) \ln \frac{P_1(i)}{P_2(i)} \tag{3.2}$$

When P_1 is exactly the same as P_2, the KL divergence is 0. Hence, the closer the locations of the glimpses are to the actual ROIs, the higher the reward.

3.3.3.2 Gradient Calculation

We use REINFORCE algorithm [22] to maximize J [29]. The gradient of J can be approximately by:

$$\nabla_\theta J = \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} \nabla_\theta \log \left(\pi \left(\ell_i^j | \mathbf{S}_{1:i-1}^j ; \theta \right) \right) r^j \tag{3.3}$$

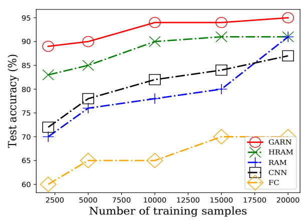
where m denotes the number of episodes and n denotes the total number of steps.

3.4 Experiments

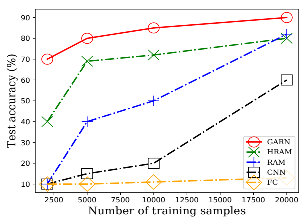
To evaluate our proposed method, GARN, we first conducted experiments on two variants of the MNIST dataset, similar to [30]. We then tested on real-world fMRI data with synthetic regions and labels. More details about each dataset can be found in Table 3.1.

3.4.1 Compared Methods

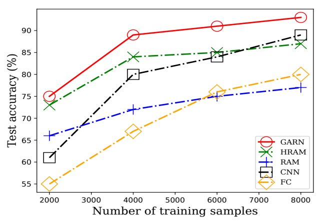
• **Fully Connected Neural Network (FC):** We compare with a fully connected neural network with two hidden layers. The first hidden layer of the FC is composed by 100



(a) *Comparing two digits (Task 1)*



(b) *Adding two digits (Task 2)*



(c) *Brain network classification (Task 3)*

Figure 3.5: ***Performance of multi-attention classification on three different tasks.*** *Undering each setting, the size of test set is same with training set. Our proposed guided attention recurrent network (GARN) achieves up to 30% higher accuracy with a small number of training samples, compared to other baseline models. As the number of training samples increases, our GARN model still outperforms others by 5%.*

neurons, and the second layer by 50. A final classification layer with the appropriate number of outputs is attached at the end.

• **Convolutional Neural Network (CNN):** We designed a CNN that consists of two convolutional layers. Each convolutional layer performs convolution with ReLU activations followed by average pooling. We then connect this to an **FC** network with an architecture that is the same as described above.The convolutional layers have 128 and 256 neurons, respectively. The filter sizes for convolution and pooling are 5×5 and 2×2, respectively.

• **Recurrent Attention Model (RAM):** We built a recurrent attention model based on [29] with a sensor crop size of 20×20 and three glimpse scales. In the glimpse network, we use two fully connected layers which each has 128 neurons to encode the cropped image as well as the location vector. Finally, a third FC layer with 256 neurons is used to encode the glimpse representation. We use a 256-cell LSTM as our core network. The location network has two layers: the hidden layer has 128 neurons, and an output layer with 2 neurons (using tanh activations) indicating the location coordinates. The action network (classifier) is a fully connected network whose architecture is identical to **FC** described above.

• **Recurrent Attention Model with Hints (HRAM):** To demonstrate the usefulness of guidance information, particularly when training with a small dataset, and also for a fair comparison, we implemented a variant of RAM with hints (*i.e.*, guidance information). Architecture-wise, HRAM is identical to RAM. We trained HRAM with the locations of the ROIs with the standard deviation for calculating KL divergence at 0.2.

• **Guided Attention Recurrent Network (GARN):** This is our proposed model which consists of two RNNs. The RNN for locating ROIs consists of a glimpse network, a core network, and a location network. The RNN for classification consists of another glimpse network, another core network, and an action network (*i.e.*, classifier). Each RNN has the same architecture as their counterpart in the baseline RAM. But the RNN for classification only uses one glimpse scale, instead of three, in its glimpse network f_G^C.

In the next section, for all attention baselines and proposed GARN, we use 8 glimpses in the task 1 and 2, and 20 glimpses in the task 3. In the section of parameter discussion, we will try more parameter settings.

3.4.2 Performance Evaluation

We evaluate the performance of GARN and the other methods on three different classification tasks: *comparing two digits*, *adding two digits*, and *brain network classification*. We introduce each task in more detail in the subsequent discussion. However, before we do so we would first like to highlight two important findings in our performance evaluation:

Importance of Guidance Information: We see in Figures 3.5(a)-3.5(c) that, across all three tasks, the methods with guidance information (GARN and HRAM) perform substantially better than others when the number of training samples is small. When the number of training samples start to increase, the other methods close the gap in terms of performance but guidance-based methods are still superior.

Importance of Separating Functions: Here, we see in Figures 3.5(a) and 3.5(b) that

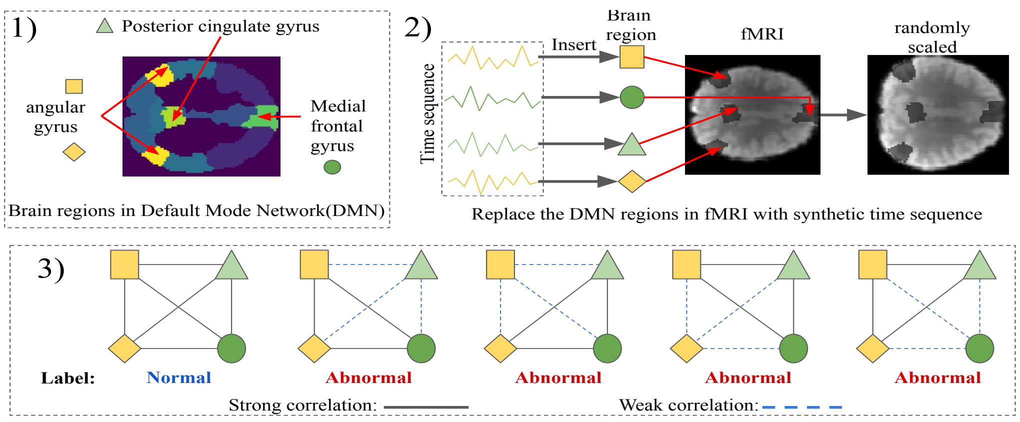


Figure 3.6: *The brain network classification task on fMRI data.*

when we have sufficient training samples, RAM catches up to HRAM. However, we find that across all three tasks GARN still performs the best. This hints at the importance of using two separate networks that each focus on one of the two important functions: locating ROIs and classification.

3.4.2.1 Task 1: Comparing Two Digits

In this task, we constructed a new dataset based on the MNIST dataset. For each sample, we randomly selected two MNIST images, resize them to 14×14, and embedded them into a black background of size 80×80. We randomly sampled two locations around the coordinates (16, 16) and (64, 64) for embedding these two digits. These digits were set to be far-apart in order to force the attention-based methods to learn a policy that has to move for longer distances. We assigned the label 0 to a sample if the digit on the lower right region is larger than the one on the upper left region; otherwise, the label is set to 1.

Figure 3.5(a) compares the test accuracies of our proposed GARN and the four baseline models. When there are only 2k training samples, GARN achieves 6% higher accuracy than the best performing baseline HRAM – RAM modified with additional guidance information. This highlights the importance of designing separate RNNs for locating ROIs location and classification. In addition, the improved test accuracy of HRAM over

RAM, especially for smaller training datasets, highlights the importance of using ROIs' locations during training, whenever possible.

3.4.2.2 Task 2: Adding Two Digits

Next we evaluated our proposed model on determining the sum of two digits embedded in an image. We used the same training images from Task 1 and labeled each sample with one out of 19 possible classes. This task is inherently more difficult than the first task due to the larger number of classes and the fact that images with the same label can look very different, *e.g.*, an image consisting of 1 and 9 and an image of 2 and 8 both have the same label.

In Figure 3.5(b), we demonstrate that GARN outperforms all baselines for training datasets with size ranging from 2k to 20k samples. Interestingly, when there are only 2k training samples, all baselines but HRAM perform poorly – similar to random guessing. HRAM increases the test accuracy by 30%, again indicating the usefulness of providing guidance information in settings when we only have limited data. Lastly, GARN achieves more than 70% test accuracy even with 2k training samples and gradually increases its accuracy to 90% with 20k training samples. Our results indicate that GARN is effective in avoiding overfitting even for relatively complex tasks, with very small number of training samples.

3.4.2.3 Task 3: Brain Network Classification in fMRI

Lastly, we studied the performance of GARN on a brain network classification problem that reflects settings in the real-world. At a high level, this classification task aims to determine whether a human subject has a certain brain disorder (*e.g.*, concussion, bipolar disorder or Alzheimer disease) from fMRI data. An fMRI sample is a 4D image. Essentially, it is a series of 3D brain images captured over time. From a given fMRI sample, we can construct a weighted graph called a functional brain network with nodes in the graph denoting regions and time-series correlations between regions being the weighted edges. Such correlations are calculated from associated time sequences and reflect the functional interactions between brain regions [43]. In this work, we used regions in the Default Mode Network (DMN), one of the most prominent function networks [1]. We designed a classification taks which requires understanding of the relationships between different regions in DMN. Figure 3.6 summarizes the steps in constructing the dataset.

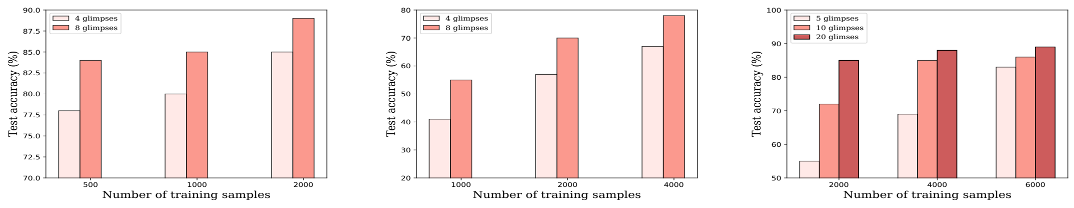
In more details, we constructed a synthetic brain network dataset from real-world fRMI data with 31 samples following these steps:

1. We normalize the brain shape of all subjects by aligning them to the MNI152 standard brain template [2]. This allows us to align all the regions from different fMRI images and helps us identify brain ROIs.

2. For each raw fMRI image, we carefully select six regions of the DMN. These regions are: left/right posterior cingulate gyrus, left/right angular gyrus, and left/right Medial frontal gyrus [44]. We further combine the regions that are visually close to each other, *e.g.*, the left/right posterior cingulate gyrus, and the left/right Medial frontal gyrus.

3. To ensure all four DMN regions are included, we extracted a 3D slice with shape = [width = 91, height = 91, time length = 10] at the position $z = 51$ from each fMRI image. This gives us a total of 31 fMRI images which we used as a basis to construct a larger synthetic dataset. We used two complementary approaches (Figure 3.6-2), *i.e.*, associating each fMRI image with randomly generated time sequences and changing the DMN locations by randomly scaling each fMRI image.

[1]`https://en.wikipedia.org/wiki/Default_mode_network`
[2]`https://fsl.fmrib.ox.ac.uk/fsl/fslwiki/Atlases`



(a) *The task of comparing two digits* **(b)** *The task of adding two digits* **(c)** *The task of classification on fMRI*

Figure 3.7: ***Performance of GARN with different number of glimpses.*** *The number of glimpses heavily depends on the number of ROIs. More glimpses help avoid overfitting, but the benefits decrease as the number of training samples increase.*

4. To determine the label for each new fMRI image, we first built a simple brain network that is a complete graph of four DMN locations. We then calculated the Pearson correlation between each pair of DMN locations based on their time sequences. An fMRI image is labeled as "normal" if all pairwise correlations are higher than 0.6, otherwise it is labeled as "abnormal".

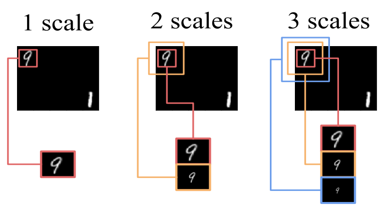
We can see from Figure 3.5(c) that our proposed GARN significantly outperforms all baselines by up to 2%-20% accuracy, even with a small number of training samples.

HRAM achieves about 8% higher accuracy compared to RAM, suggesting the usefulness of utilizing ROIs locations during training. Lastly, neither the CNN nor the FC models work well with small training dataset.

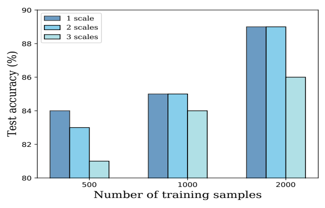
3.4.3 Discussion on Parameters

We evaluated two important hyperparamaters, *i.e.*, the number of glimpses and the number of sensor scales.

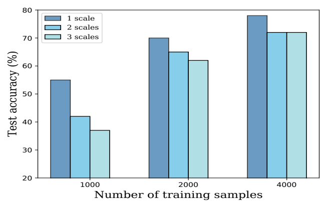
The number of glimpses represents how many chances we give the model to move the sensor around. More glimpses equals a longer sensor trajectory which typically corresponds to a higher likelihood of gathering more information from the image. In Figure 3.7, we compared the test accuracies of models given different numbers of glimpses.



(a) *number of scales*



(b) *The task of comparing two digits*



(c) *The task of adding two digits*

Figure 3.8: ***Performance of GARN with different number of sensor scales.*** *Having smaller number of scales for the classification RNN helps to avoid overfitting with fewer training samples. This also indicates the need for designing two seperate RNNs in multi-attention classification problem.*

For tasks one and two which only contain two ROIs, we set the glimpse number to be four and eight, respectively. For task three, we set the glimpse number to be five, ten, and twenty, respectively. The choices of glimpse numbers are based on the number of ROIs to increase the likelihood of capturing ROIs with stochastically generated locations. In Figure 3.7(a) and Figure 3.7(b), we can see that GARN achieves higher accuracies with eight glimpses than four glimpses. The accuracy gap decreases as the training samples increases. This is likely because the four-glimpse agent has fewer chances of hitting all the ROIs. Figure 3.7(c) shows the impact of different number of glimpses on brain classification task. Given that there are four ROIs in the Default Mode Network, the minimal required number of glimpses is higher than the first two tasks. Having access to more training samples can alleviate the need for more glimpses per sample, as indicated by the shrinking accuracy gaps between ten and twenty glimpses at 8k training samples. Our results suggest that our GARN can effectively avoids overfitting on smaller datasets.

Next we discuss the impact of the number of sensor scales on test accuracy. Recall that our GARN uses two glimpse networks, f_G^R and f_G^C, to locate ROIs and for classification. Each glimpse network can be configured with a different number of sensor scales for each glimpse. We used three scales for f_G^R, similar to the original RAM. We vary the number of sensor scales from one to three for f_G^C which is the agent for classification as demonstrated in Figure 3.8(a).

In Figure 3.8(b) and 3.8(c), we compared the test accuracies for different number of sensor scales. Our results show that for both tasks, using fewer scales under smaller training samples achieve higher test accuracies. This suggests that using more and larger scales may lead to overfitting especially when the training datasets are small. One potential reason is that larger scale contains information, *e.g.*, black background, that is not useful for classification. However, such information can be useful for locating ROIs. This suggests that it is useful to separately configure the number of scales for locating ROIs and classification, as we did in GARN by designing two separate RNNs.

3.5 Related Work

To the best of our knowledge, this work is the first to address the problem of guided multi-attention classification.

Image classification and object recognition: Image classification has become a widely studied topic. Over the past decade, deep neural networks such as CNNs have achieved significant improvement in image classification accuracy [5]. However, these CNNs often incur a disproportionately high computation cost to detect a small object in a large image. A number of works [8, 45, 46] have attempted to address this problem of high computational cost, but in a non end-to-end way. Others [47, 48, 49], on the other hand, have formulated the task of object detection as a decision task, similar to our work.

Classification on fMRI data: The task of classifying fMRI data can be formulated as a special case of multi-object image classification. Most recent work analyzing fMRI study one or more of the following related sub-tasks: brain region detection [50, 51], brain network discovery, and classification [52, 53]. However, neuroimaging datasets are inherently quite challenging to work with due to their high noise, their high dimensionality, and small sample sizes. It was not until very recently that researchers started to propose end-to-end solutions, such as CNN based methods [54] which solve both brain network discovery and classification coherently [55]. Different from existing work, we use a guided attention-based model which can locate brain regions and do classification as well, without requiring additional information such as time sequences from ROIs as input [55].

Attention model: Recently, researchers have begun to explore attention-based deep learning models for visual tasks [30, 49, 56, 57] and natural language processing [58, 59]. Specifically, Mnih et al. [29] proposed the recurrent attention model (RAM) to tackle the issue of high computation complexity when dealing with large images. Other work based on RAM have also tackled the problems of multi-object recognition and depth-based person identification [30, 60]. Most recently, Tariang [61] proposed a recurrent attention model to classify natural images and computer generated images. The structure and training method are similar with [29, 30], while it uses a CNN to implement its glimpse network. Meanwhile, Zhao [62] combined a recurrent convolutional network with recurrent attention for pedestrian attribute recognition, which uses a soft attention mechanism instead of the hard attention used by RAM. Another recent study leveraging the soft attention mechanism is [63], which uses recurrent attention residual modules to refine the feature maps learned by convolutional layers. In the areas of person identification, sequence generation, image generation, some other works [64, 65] are also utilize both attentional processing as well as RNNs.

4

EPNet: Learning to Exit with Flexible Multi-Branch Network

4.1 Motivation

Dynamic inference is an emerging technique that aims to reduce the resource consumption, e.g., computational cost, of deep neural network during inference. A prominent approach of dynamic inference centers around the use of multi-branch networks [31, 32, 33], i.e., networks that consists of more than one output layer, for handling the natural difficulty variations exhibited in real-world samples. Ideally, the multi-branch network spends *just enough computation* for each sample, instead of applying the same amount of computation, as illustrated in Figure 4.1. For example, easier samples can use earlier prediction branches while the harder ones go through the normal forward propagation. We refer to the scenarios of using branch classifiers as *early-exiting*. One of the key challenges in achieving efficient dynamic inference is being able to adapt the resource consumption to individual inference request without impacting the inference accuracy. Existing work on *dynamic inference with multi-branch network* [31, 32, 33] used handcrafted policies

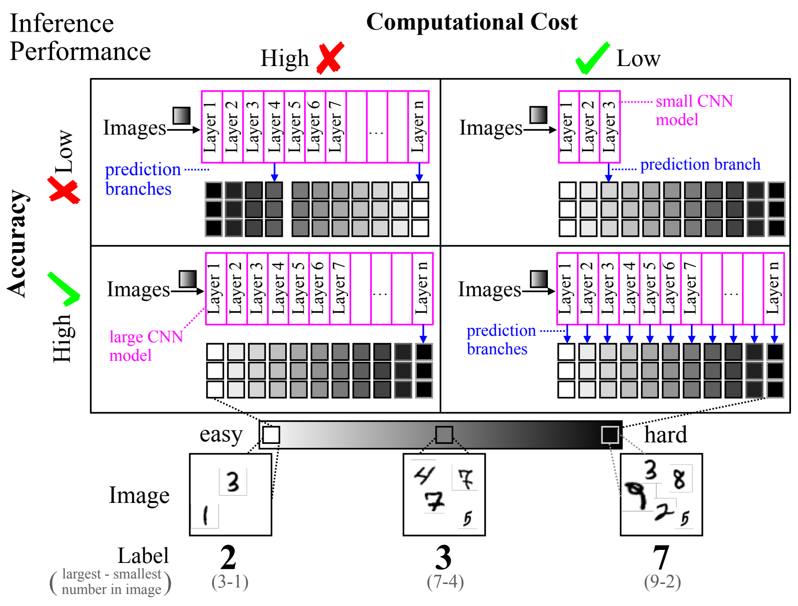


Figure 4.1: *The problem of dynamic inference for image classification tasks.* Our goal is to design a CNN that can adapt its execution to the inference request difficulty, to achieve high inference accuracy and low computational cost.

for deciding the exiting branch per inference, as shown in Figure 4.2. As these policies require domain experts in setting the threshold, the performance is subject to external factors such as resource fluctuation or sample difficulty. Further, prior networks only provide a few adapting options, with a smaller number of branch exits. This limits the network's ability to adapt to smaller changes, i.e., less flexible.

In this work, we investigate the problem of designing a flexible multi-branch network and early-exiting policies that can be learned in conjunction from the training dataset. Designing a flexible multi-branch network requires attending to the tension between the number of branches and the additional parameters and computational cost. In particular, we need an efficient network structure that sufficiently represents the search space of the early-exiting policies, while being mindful about the additional cost associated with the early-exiting controllers. Additionally, when designing the controllers, we need to explicitly consider the trade-off between the classification accuracy and resource efficiency.

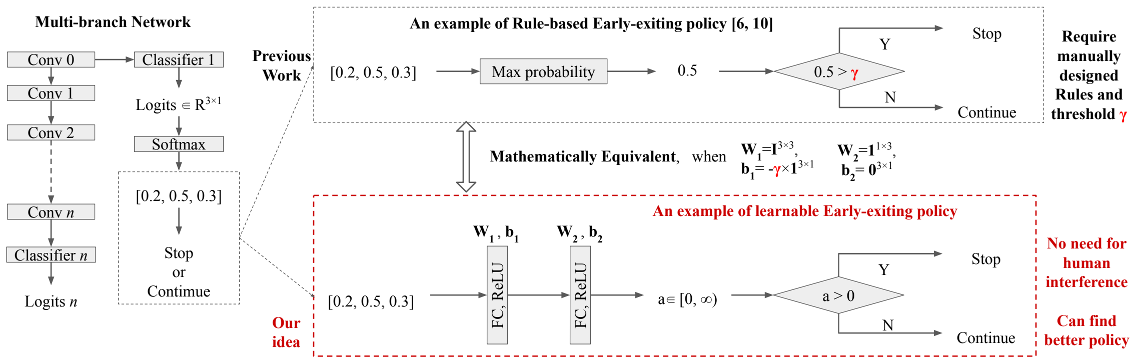


Figure 4.2: *Early-exiting policies for multi-branch networks.* A learning-based policy has two key advantages over the rule-based policies: *(i)* does not require human interference; *(ii)* can lead to better policies.

4.2 Problem Formulation

In this work, we target the dynamic inference problem of image classification. Given a set of samples $\mathcal{D}$=$\{\mathbf{I}_1, \mathbf{I}_2, ..., \mathbf{I}_N\}$, where $\mathbf{I}_i \in \mathbb{R}^{x \times y \times c}$ denotes a image, and a user-defined resource sensitivity value β, our goal is to design a multi-branch model $\mathbf{M}$ that balances the classification accuracy and resource cost trade-offs.

To learn $\mathbf{M}_\theta$ that is parameterized by θ, we design an optimization metric called *benefit score* B as:

$$B(\mathcal{D}, \mathbf{M}_\theta, \beta) = Acc(\mathcal{D}, \mathbf{M}_\theta) - \beta \times C(\mathcal{D}, \mathbf{M}_\theta), \tag{4.1}$$

Here the $Acc(\mathcal{D}, \mathbf{M}_\theta)$ is the average accuracy of $\mathbf{M}_\theta$ on $\mathcal{D}$, and $C(\mathcal{D}, \mathbf{M}_\theta)$ is the average computational cost of $\mathbf{M}_\theta$ on $\mathcal{D}$. The parameters θ is obtained by maximizing the *benefit score* S, and will be used to decide how to perform the classification task. In our formulation, if the resource is ample, we can set β to be 0 which turns to a traditional classification problem. We can set β to a larger value when the resource is more constrained.

4.3 EPNET: the Proposed Method

We first describe the overall architecture and design of our proposed multi-branch models EPNET. We then detail our formulation of the early-exiting problem in Section 4.3.3, followed by how to effectively train the early-exiting controllers in Section 4.3.3.

4.3.1 Model Structure Overview

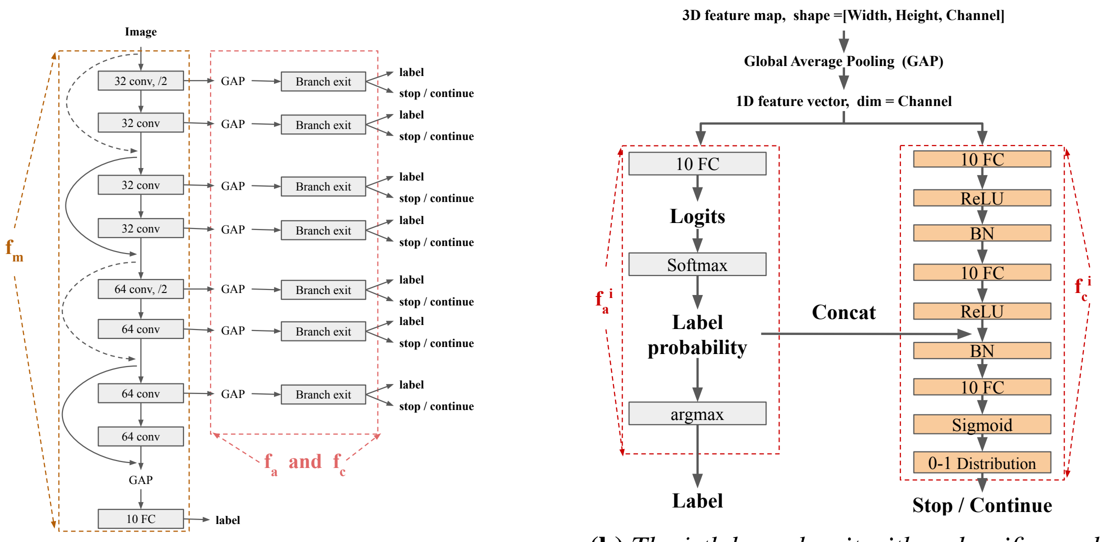
Our proposed EPNET consists of three components: the main branch network f_m, branch classifiers f_a, and early-exiting controllers f_c. Figure 4.3 illustrates an example structure.

4.3.1.1 Main Branch Network

The **main branch network** f_m (the leftmost component in Figure 4.3(a)) takes the image as the input and can produce classification result independently with the final classifier layer. To enable as many exits as possible, we attach additional branch classifiers f_a (described below) at every convolutional layer except the last one. Larger number of branch classifiers provides the early-exiting controllers f_c more flexibility to adapt to dynamic inference environments such as varying sample difficulties and fluctuating system resources.

Additionally, the ideal property of f_m is the monotonically increasing accuracy with the number of the layers, i.e., the branch classifiers attached to the latter layers should have higher accuracy than earlier layers. In this work, we adopt the ResNet [6] as the main branch network architecture. Other potential implementations of f_m include networks with short-cut connections such as DenseNet [7].

In short, the main branch network should be designed around two key principles: *(i)* enabling as many exits as possible; *(ii)* maintaining monotonically accuracy increase with exits.



(a) *Overall network architecture*

(b) *The i-th branch exit with a classifier and a controller.*

Figure 4.3: *An example of our proposed* EPNET*, a multi-branch network with learnable early-exiting policies.* The overall structure consists of three parts: main branch f_m, branch classifiers f_a, and early-exiting controllers f_c.

4.3.1.2 Branch Classifiers

As shown in Figure 4.3(b), we need a **branch classifier** f_a^i for each exit i. In our current implementation, f_a^i is attached to the i-th convolutional layer of f_m. For a main branch network with a total of N convolutional layers, we need a set of branch classifiers f_a represented as $\{f_a^i, i \in \{1, 2, .., N-1\}\}$.

One of the key design challenges for branch classifiers is to balance its resource requirement and classification accuracy. Considering the following example. In order for an exit i to be a valid exit, the total computational cost of exiting through f_a^i should not exceed that of exiting through f_a^{i+1}. This indicates an upper bound of computational budget, e.g., the difference between the two consecutive convolutional layers, when designing the branch classifiers. This computational budget has to be shared with the early-exiting controllers, further restricting our design space.

Similarly, the memory consumption of branch classifiers, i.e., number of parameter, is also a big problem. In classic CNN structure, the 3D feature map outputed by last convolutional layer is flattened and fed to a fully connected layer, where the most of the parameters belong to. This suggests that simply attaching a classifier layer to every convolutional layer may lead the memory consumption to increase by multiple times.

Instead, we design the f_a^i with the structure of GAP-FC-SoftMax. Here the GAP is a global average pooling layer and FC is a fully connected layer. We chose to use the GAP layer because it significantly reduce the resource requirement of the branch classifiers.The input of f_a^i is the 3D activated feature map generated by the i-th convolutional layer.

In short, the branch classifiers should: *(i)* comply to the resource consumption pattern of the main branch network layers; and *(ii)* without impacting the accuracy.

4.3.2 Early-exiting Controllers

Lastly, our EPNET requires a set of **early-exiting controllers** $f_c = \{f_c^i, i \in \{1, 2, .., N - 1\}\}$ that regulates the usage of each exit i.

We design f_c^i as a two-part network, i.e,. f_{in}^i and f_{cat}^i, to preserve the information of features outputted by both the GAP layer and the logits outputted by f_a^i. This allows our controllers to perform at least as well as previously proposed rule-based policy [31, 32, 33]. Both of the f_{in} and f_{cat} are in the form of stacked blocks FC-BN-ReLU, except for the last activation of f_{cat} which should be Sigmoid function. Here BN is a batch normalization layer.

Specifically, f_{in}^i takes the 3D activated feature map generated by the i-th convolutional layer as input and outputs a 1D vector v. This 1D vector v and the logits outputted by f_a^i are concatenated and used as the input to f_{cat}^i who then output a scalar signal $p \in [0, 1]$. From the Bernoulli distribution paramiterized by p, we sample a stopping signal $s \in \{0, 1\}$. If $s = 0$, the forward propagation in main branch f_m will continue until another controller f_c^j at j-th convolutional layer outputs $s = 1$, or reaches the final classifier in f_m. If $s = 1$, the forward propagation is immediately stopped and the model output the label predicted by the current branch classifier f_c^i.

4.3.3 Learning the Early-exiting Policy

We formulate the early-exiting problem as a Markov decision process (MDP) problem $M = (S, A, T, R)$, where the environment is $E = (f_m, f_a, D)$. We describe the *state set* S, *Action set* A, *Transformation table* T and *Reward* R in detail below. The early-exiting policy π can be learned through maximizing the expected reward $E_\pi(R)$, once f_m and f_a are trained.

States set S. We define a state s_i as (m_i, y_i) where m_i is the outputted vector at the GAP layer after the i-th convolutional layer of f_m, and y_i is the logits outputed by f_a^i. Additionally, S contains a distinguish state s_{ab} called absorbing state. The MDP stops when any states transition to s_{ab}. In our case, s_{ab} represents the state when the controller decides to stop and exit from exit i. Lastly, we define the start sate $s_0 = I$ where I denotes an image from D.

Action set A. The MDP only has two actions: ”stop at current exit” or ”continue to forward propagation”. Here we denote it as $A = \{0, 1\}$, where 0 is ”stop” and 1 is ”continue”. Ones the agent takes action $a = 1$, the state transfer to s_{ab}. So given a image x, the trajectory set $\mathcal{T}$ of agent can be denotes as $\mathcal{T} = \{(x, 0^n, 1) | n \in \{0, 1, ..., N-1\}\}$, where 0^n means a succession of 0 of length n, and N is the total number of branch exits.

Transformation table T. $T = \{P(s, a, s') | s, s' \in S, a \in A\}$, where $P(s, a, s')$ is the probability that state s transfer to s' by taking action a. In this study, the T is deterministic so that all $P(s, a, s') = 1$ if $\pi(s, a) = s'$, otherwise $P(s, a, s') = 0$.

Reward R. Given a Image x from D and a cost sensitivity β. If the agent stop at the i-th exit, the trajectory is $\tau = (x, 0^{i-1}, 1)$. The reward $R(\tau)$ is the $Benefit(\{x\}, f_i, \beta)$ defined in Eq 4.1. Here f_i is the sub-network from the input layer of the main network

f_m to the output layer of i-th branch classifier f_a^i.

4.3.4 Training Consideration of the Controllers

The main branch f_m and additional branch classifiers f_a can be trained by simply summing their cross entropy loss together [31]. Here We mainly describe two approaches to train the controllers f_c. We compare their ability to find early-exiting policy in Section 4.4.8.

The **first option** is to leverage REINFORCE algorithm [22] to train the early-exiting controllers as following.

$$\nabla_\theta E_\pi(R) \approx \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} \nabla_\theta \log \left(\pi\left(a_i^j | s_i^j; \theta\right)\right) R^j \tag{4.2}$$

$$\pi\left(a_i | s_i; \theta\right) = \begin{cases} f_c^i(s_i) & a_i = 1, s_i \neq s_{ab} \\ 1 - f_c^i(s_i) & a_i = 0, s_i \neq s_{ab} \\ 1 & a_i = 1, s_i = s_{ab} \\ 0 & a_i = 0, s_i = s_{ab} \end{cases} \tag{4.3}$$

Here m is the number of episode, and n denotes the length of a trajectory, *i.e.*the number of exits. s, a, R are the states, actions, rewards defined in the previous section.

But the classic REINFORCE rule is based on sampling and Markov Chain Monte Carlo approach (MCMC), which could be inefficient in our task. For example, if the dynamic model has 9 additional branches, the trajectory $\tau = (x, 0^9, 1)$ may have very low chance to be sampled. This is because it requires all the controllers to output *continue*. The low sampling efficiency can cause well known drawback of REINFORCE, the high variance of policy gradient.

The **second option**, which we used for training the controllers in this work, is to directly compute the exact gradient of $E_{\pi_\theta}(R)$ as following.

$$\nabla_\theta E_\pi(R) = \sum_{j=1}^{n} \nabla_\theta \prod_{i=1}^{n} \left(\pi\left(a_i^j | s_i^j; \theta\right)\right) R^j \tag{4.4}$$

The gradient computation is feasible because of two important properties of our MDP. *First,* The environment $E = (fm, fa, D)$ is a given and the only randomness comes from the policy π itself. *Second,* given an image and a multi-branch network of N branches, the size of trajectories set is $\mathcal{T} = \{(x, 0^i, 1) | i \in \{0, 1, ..., N-1\}\}$ with size N. In our current design N is bounded by the number of convolutional layers which varies from tens to a couple hundreds given current popular CNNs. As such, N is small enough that we don't need sampling.

4.4 Experimental Evaluations

We evaluate our proposed EPNET with three datasets and compare to two types of baselines. We summarize and highlight our key results below.

• **Accuracy and resource comparisons to baselines.** We show that EPNET outperforms all baselines including ResNet, and both rule-based and manually-tuned early-exiting policies applied to multi-branch models [31, 32, 33].

• **Adaptivity of EPNET.** In Section 4.4.7, we demonstrate EPNET's ability to effectively choose the appropriate branch exit based on both the classification difficulty of samples and the computational cost sensitivity of the resource-constrained platform.

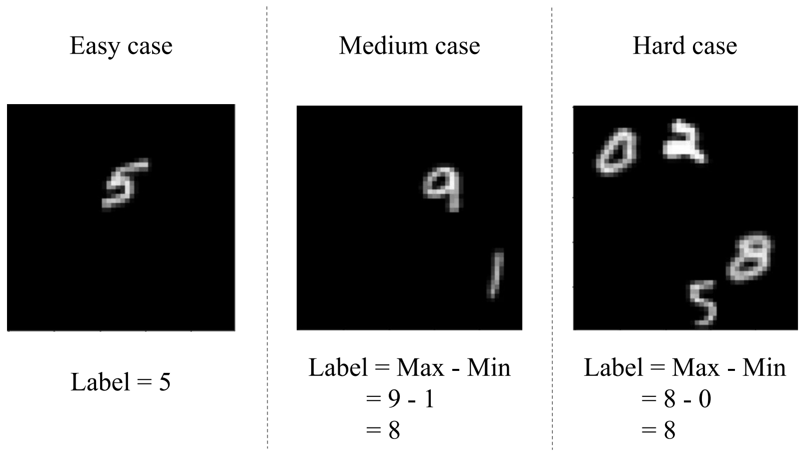


Figure 4.4: *Example of samples in our Max-Min MNIST dataset.* This dataset contains samples of three difficulty levels. The ratios for easy, medium, and hard levels are 2:1:1.

4.4.1 Data Sets

We used the following three widely used datasets for image classification task to evaluate the performance of our EPNET.

• **Max-Min MNIST dataset.** We created this dataset of three difficulty levels, based on the original MNIST, to evaluate the effectiveness of early-exiting policy. Figure 4.4 illustrates corresponding image examples. We constructed the *easy* samples by embedding the original MNIST digits into a 50×50 black background and reusing the original labels. The *medium*-level samples had two digits embedded in the 50×50 black background, with the labels being the absolute difference between the two digits. Finally, the *hard*-level samples had twice as many digits as the *medium*-level ones and were assigned the labels in the same way. For each sample, the locations of digits were generated from the 2D uniform distribution. We used the ratios of *2:1:1* for *easy*, *medium* and *hard* levels for both training and test sets, respectively. The total numbers of training and test images were 60000 and 10000 respectively, the same as the original MNIST dataset.

• **Multi-scale Fashion MNIST.** We created this dataset based on the original Fashion MNIST. Figure 4.5 illustrates corresponding image examples, which could be categorized

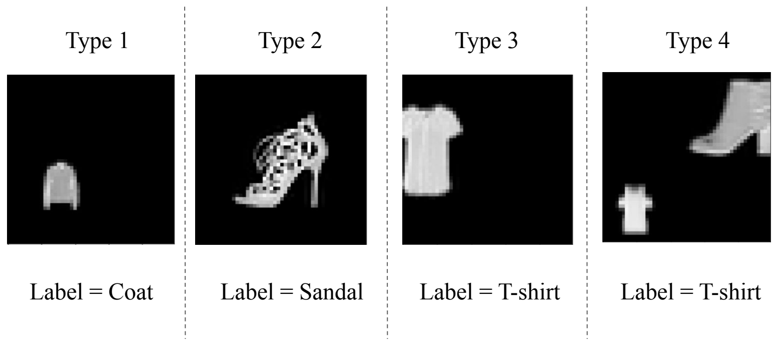


Figure 4.5: *Example of samples in Multi-scale Fashion MNIST dataset.* This dataset contains four different types of samples, with equal proportion.

to four types. We constructed the Type 1 by embedding the original Fashion MNIST objects into a 50×50 black background and reusing the original labels, the objects are scaled to the 0.6 times of its original size. For the Type 2 and 3, the objects are scaled to the 1.5 times of its original size. The Type 4 contains two objects, one is large, another is small, and its label is decided by the small objects. In the Type 1, 3, 4, the locations of object were generated from the 2D uniform distribution. For Type 2, the object is fixed at the center. We used the ratios of *1:1:1:1* for each type, respectively. The total numbers of training and test images were 60000 and 10000 respectively, the same as the original Fashion MNIST dataset.

• **CIFAR-10.** We used the original *CIFAR-10* dataset which consists of 50k training and 10k test images of 32×32 pixels, respectively. For training we adopted the data augmentation technique used in the [6]. Specifically, each image was zero-padded with 4 pixels on each side, then randomly cropped to produce 32×32 resolution. Further, training images were flipped horizontally with 0.5 probability and the pixel values were normalized by subtracting channel means and dividing by channel standard deviations.

4.4.2 Baseline Methods

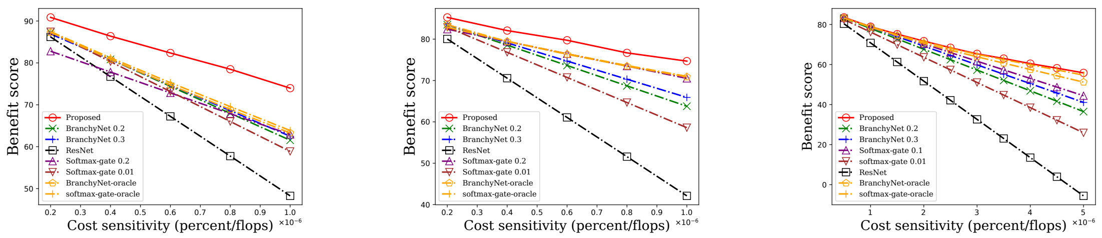
We compared to two types of baselines: *(i)* a state-of-the-art image classification model; and *(ii)* early-exiting policies.

• **ResNet [6].** We not only compared with ResNet, but also used ResNet as the main branch network f_m of proposed EPNET. We made the following changes to the original ResNet structure to account for the image size difference between our chosen datasets and the ImageNet dataset of which ResNet was designed for.

These changes are: *(i)* we removed the first conv layer and the first pooling layer; *(ii)* the first conv layer in the first residual block was performed by stride of 2.

• **BranchyNet [31].** We chose a representative entropy-based early-exiting policy as described in BranchyNet [31]. At a high level, BranchyNet works by comparing the entropy of logits of an exiting branch to a predefined threshold and halting the forward propagation in main branch if the logits entropy is lower. We used the recommended entropy thresholds of 0.2 and 0.3 for the baselines and denoted them as *BranchyNet-0.2* and *BranchyNet-0.3*, respectively. We also compared the BranchyNet with dynamic thresholds, denoted as *BranchyNet-oracle*, which was constructed as following: *(i)* the thresholds are manually tuned; *(ii)* by using the results of our EPNET as a guidance for searching the thresholds.

• **Softmax-gated policy.** This baseline leverages the maximum value of the softmax probability and compares the probability to a given threshold for early-exiting [32, 33]. We denoted the baseline as *Softmax-gate-*γ where γ is the threshold. Given the logits of a branch classifier, if the maximum value of the softmax probability is larger than $1 - \gamma$, the model halt the forward propagation. On each dataset, we pick two thresholds. By the smaller γ the network can achieve the high accuracy close to our EPNET. By the larger γ the network can maintain low computational cost close to EPNET. Similar with BranchyNet, we also compare with Softmax-gate with dynamic thresholds, denoted as



(a) *Max-Min MNIST* **(b)** *Multi-scale Fashion* **(c)** *CIFAR-10*

Figure 4.6: *The benefit score comparison on all three datasets.* We observe that our proposed EPNET outperformed two types of baselines.

Softmax-gate-oracle.

For fair comparisons, our EPNET uses the same structure as its ResNet for its main branch network f_m. Further, EPNET shares the same structure and parameters of f_m and f_a with the BranchyNet and Softmax-gated policy.

4.4.3 Evaluation Methodology

We evaluated the effectiveness of EPNET on all three datasets with following two metrics.

The first metric we chose is the *benefit score* that was defined in the Equation (4.1). This metric allows us to compare our EPNET to baselines in a unified way. In each task, we firstly set the cost sensitivity to $\beta \times 0.01$, where β^{-1} is the order of magnitude of the average comutational cost (FLOPS) of a single convolutional layer in the EPNET used. Then we increase the cost sensitivity each time by adding $\beta \times 0.01$ to the previous cost sensitivity. So we can observe how the methods' performances change against the decreasing available resources. Under each setting of cost sensitivity, we retrain the controllers and keep the rest part of EPNET fixed.

We also used a metric, referred to as *budget-constrained accuracy*, for understanding the effectiveness of early-exiting [32, 33]. To calculate the budget-constrained accuracy, we first define a computational budget and then use it as a barrier for determining the accuracy. For example, in the case of CIFAR-10, we used our EPNET's total computational cost (FLOPS) over the test dataset as the computational budget, and evaluated all baseline models. For baseline models that did not finish all test images within the budget, we assigned labels in an uniformly random way to the remaining test images. We use acc^b to denote the resulting budget-constrained accuracy.

4.4.4 Performances on Max-Min MNIST Dataset

We first describe the network structure and parameter settings we used in EPNET for training on the Max-Min MNIST dataset, followed by the performance comparison to its respective baselines.

Network structure setting. For the main branch network f_m, we used a ResNet with 12 convolutional layers. The first four layers each has 32 filters, followed by another four layers with 64 filters. The last two convolutional layers are of 128 filters. We down-sampled by using a stride of 2 for convolution when the number of filters changed between layers. To construct the early exits, we used a single-layer classifier f_a^i that takes the input of the i-th convolutional layer of f_m. This resulted in a total of 12 potential exits. On i-th exit, The classifier f_a^i is a single fully-connected layer. The controller f_c^i consists of two fully-connected networks f_{in} and f_{cat}, where the f_{in} has 10, 10, 10 units in each layer, and the f_{cat} has 10, 10, 1 units in each layer.

Parameter setting. We adopted the Kaiming initialization [66] and BN [67] without dropout when training the main and branch classifiers f_m and f_a, respectively. We used a mini-batch size of 64 and momentum of 0.9. We set the initial learning rate to be 0.1. We trained the classifiers for a total of 60 epochs. Once the classifiers were trained, we fixed the classifiers and train the controllers f_c. The mini batch size is 64, and the initial learning rate is 0.01. We trained the controllers for a total of 60 epochs as well.

Result and discussion. Figure 4.6(a) compares the benefit score achieved by different baselines and our EPNET.

We have the following three key observations. *First,* the EPNET greatly outperforms all baselines, and the gap of performance grows as the cost sensitivity grows. At beginning the cost sensitivity is small (2×10^{-7}), the EPNET outperforms the best baseline Softmax-gate-oracle by 3.11. When cost sensitivity reaches 1×10^{-6}, the gap between the scores of EPNET and Softmax-gate-oracle increases to 10.17. *Second,* the ability to learn the early-exiting policy from dataset is the reason of the superiority of EPNET. The performances of oracle baselines with dynamic thresholds and the branchyNet-0.2 / 0.3 are very close, indicating the thresholds 0.2 and 0.3 recommended in [31] are suitable for this dataset, while tuning the thresholds can't bring obvious benefit. In contrast, the gaps between the EPNET and the oracle baselines are much larger than the differences among the early-exiting baselines. This results may indicate the EPNET learns much better representation of the confidence of the classification than the rule-based methods. *Third,* The ResNet's benefit score is worst because it only focuses on accuracy while has the highest computational cost.

Table 4.1 shows the comparison on budgeted batch classification. The accuracy of EPNET is 95.51%, which is higher than the best baseline Softmax-gate-0.2 by 7.35%. The BranchyNet-0.2 and 0.3 can't finish within the budget. We have to rise the threshold to 0.95, then the BranchyNet's computational cost meets the limitation of budget. But its accuracy is only 84.13% which is 11% lower than EPNET. The ResNet with same structure of the EPNET's main branch network, can only achieve 46.86% under the limited budget, which is 48.65% lower than the EPNET. Compared with ResNet without budget limitation, the EPNET's accuracy is only 0.18% lower than it, while saves about 50% of computation. Even when the budget limitation is removed for the baselines except for ResNet, their accuracies are all lower than EPNET by at least 1%.

Table 4.1: *Results of budgeted classification.* The budget of each task is the total computational cost of proposed method on the whole dataset. The results are reported as test accuracy constrained by budget (acc^b) (rank), test accuracy (acc*), and whether the classification finished before the budget ran out.

Dataset	Model	acc^b(%)	acc(%)	Completion
Max-Min MNIST	**Proposed**	95.51 (1)	95.51	✓
	BrachyNet-0.2	67.65 (5)	93.71	✗
	BrachyNet-0.3	70.82 (4)	93.03	✗
	BrachyNet-0.95	84.13 (3)	84.13	✓
	softmax-gated-0.2	87.76 (2)	87.76	✓
	softmax-gated-0.01	61.53 (6)	94.59	✗
	ResNet	46.86 (7)	95.69	✗
Multi-Scale Fashion MNIST	**Proposed**	88.68 (1)	88.68	✓
	BrachyNet-0.2	61.09 (5)	88.41	✗
	BrachyNet-0.3	68.11 (4)	87.88	✗
	BrachyNet-0.6	85.68 (2)	85.68	✓
	softmax-gated-0.2	85.38 (3)	85.38	✓
	softmax-gated-0.01	49.88 (6)	88.69	✗
	ResNet	32.14 (7)	89.92	✗
CIFAR-10	**Proposed**	88.61 (1)	88.61	✓
	BrachyNet-0.2	86.54 (4)	88.35	✗
	BrachyNet-0.3	87.83 (2)	87.83	✓
	softmax-gated-0.1	87.53 (3)	87.53	✓
	softmax-gated-0.01	71.72 (5)	88.76	✗
	ResNet	47.82 (6)	89.89	✗

4.4.5 Performances on Multi-scale Fashion MNIST

Next we study the effectiveness of the EPNet on Multi-scale Fashion MNIST dataset.

Network structure setting. We used the same network structure as described in Section 4.4.4 for the Max-Min MNIST dataset.

Parameter setting. We adopted the Kaiming initialization [66] and BN [67] without dropout when training the classifiers (i.e,. f_m and f_a) and the controllers f_c, respectively. For the classifiers, we used a mini-batch size of 128 and momentum of 0.9. We set the initial learning rate to be 0.1 and divide the learning rate by 10 every 100 epochs. We trained the classifiers for a total of 300 epochs. Once the classifiers were trained, we fixed the classifiers and trained the controllers f_c. The mini batch size is 128, and the initial learning rate is 0.01. After 50 epochs the learning rate was reduced to 0.001, then we trained the controllers for another 50 epochs.

Result and discussion. Figure 4.6(b) shows the performances of each model according to the benefit score. Similar with Max-Min MNIST dataset, we make the following two key observations. *First,* the EPNET always outperforms all the baselines and the gaps between EPNET and the baselines are very clear. When the cost sensitivity is 2×10^{-7}, the benefit score of the EPNET is 85.25, and the score of best baseline BranchyNet-oracle is 83.48. When the cost sensitivity reaches 1×10^{-6}, the benefit score of EPNET is 74.68, and the best baseline, Softmax-gate-oracle is 71.01. *Second,* the performances of oracle baselines are almost the same with the Softmax-gate-0.2, indicating 0.2 is already a good threshold. Therefore manually tuning the threshold did not make a big difference. Similar to the discussion of the previous dataset, this observation reflects that the performance of the rule-based polices are limited by their confidence/uncertainty measure.

As show in the Table 4.1, on the budgeted batched classification, the accuracy of EPNET is 88.68$\%$, which is higher than all the baselines by at least 3$\%$. The ResNet with same structure of the EPNET's main branch network, can only achieve 32.14$\%$ under the limited budget, which is 56.54$\%$ lower than the EPNET. Both the branchyNet-0.2 and branchyNet-0.3 can't finish the whole test set before the budget runs out. We have to increase the threshold to 0.6 to fit the budget, and the accuracy of branchyNet-0.2 is 85.68$\%$. Even the budget limitation is removed, branchyNets' accuracies are still lower than EPNET, which may indicate their measurements of confidence, *i.e.*the entropy of logits, may be not feasible in this task.

4.4.6 Performances on CIFAR-10

Lastly we study the effectiveness of the EPNET on CIFAR-10.

Network structure setting. For the main branch network f_m, we used a ResNet with

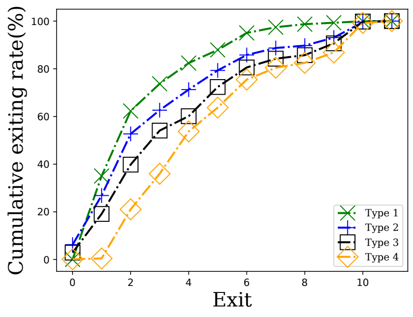
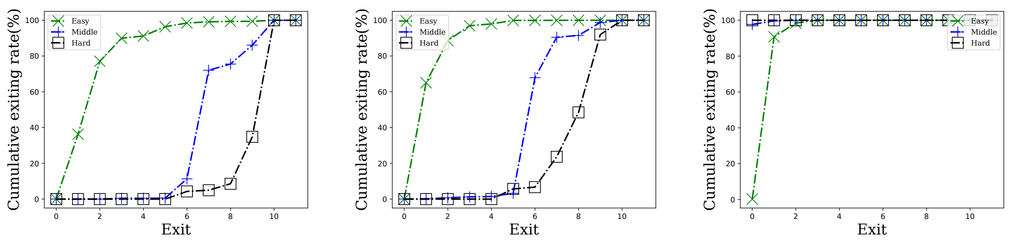


Figure 4.7: *Cumulative exiting rate on the Multi-scale Fashion MNIST.* The cost sensitivity was 2×10^{-7}.



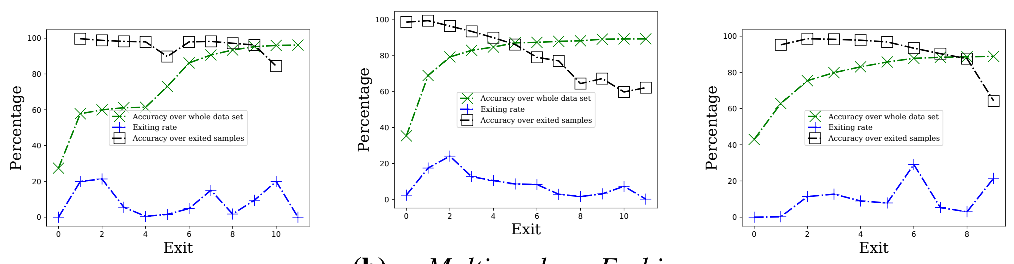
(a) The cost sensitivity is 2×10^{-7} **(b)** The cost sensitivity is 1×10^{-6} **(c)** The cost sensitivity is 2×10^{-6}

Figure 4.8: ***Cumulative exiting rate of samples with varying difficulties on the Max-Min MNIST dataset.***

10 convolutional layers. The structure is similar with the previous task but only has two convolutional layers of 128 filters. For the controller of i-th branch, the f_{in} has 100, 10, 10 units in each layer, and the f_{cat} has 100, 100, 50, 1 units in each layer.

Parameter setting. The most of the parameters are same with the Multi-scale Fashion MNIST. Additionally we used the weight decay of 0.0001 to train the classifiers. Also, for training of controllers, we set the batch size = 64.

Result and discussion: Figure 4.6(c) shows the performance of each model according to the benefit score. The gaps between EPNET and baselines are not as large as on the other datasets. The reason may be the difficulties of the samples are not as obvious as the other two datasets. So we test the models under more settings of cost sensitivity. The key observations are: *First,* even though the sores of all models are very close to each



(a) *Max-Min MNIST* **(b)** *Multi-scale Fashion MNIST* **(c)** *CIFAR-10*

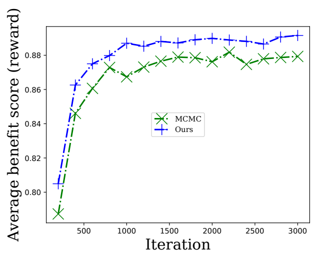
Figure 4.9: *The distribution of accuracy and exiting rate.* The cost sensitivities for each dataset are 5×10^{-7}, 2×10^{-7}, 2×10^{-7}.

Table 4.2: *Performance comparison of different controller layer configurations.*

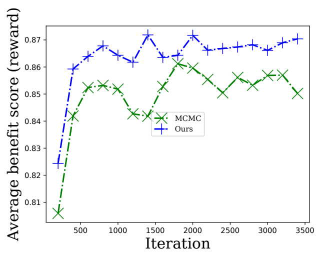
Configuration	Neurons in f_{in}	Neurons in f_{cat}	Benefit Score	Accuracy (%)	Total network Computation (FLOPS)	Controller Computation (FLOPS)
1	[10, 10, 10]	[10, 10, 1]	82.40	88.42	1.203×10^7	0.107×10^4
2	[50, 10, 10]	[50, 50, 1]	83.17	88.45	1.054×10^7	0.724×10^4
3	[100, 10, 10]	[100, 100, 50, 1]	83.51	88.61	1.019×10^7	2.419×10^4
4	None	[100, 100, 50, 1]	82.99	88.37	1.074×10^7	1.700×10^4

other at beginning, the gap between the scores of EPNET and the baselines with fixed thresholds increases quickly. When cost sensitivity reaches 5×10^{-6}, the score gap is up to 10.17. *Second,* even though the oracle baselines are granted "unfair" advantages, *i.e.*their thresholds are tuned by hand and the searching is hinted by the results of EPNET, our EPNET still slightly outperforms them. When the cost sensitivity is small (5×10^{-7}), the EPNET outperforms the best baseline Softmax-gate-oracle by 0.2. Then their gap of benefit score grows to 1.2 as the the cost sensitivity grows to 5×10^{-6}.

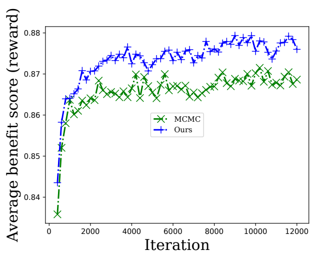
Table 4.1 compares the budget-constrained accuracy and contains the model test accuracy for reference. We observe that among all tested baseline models, only two were able to complete within the computational budget. Even so, *BranchyNet-0.3* and *Softmax-gate-0.1* had 0.78% and 1.08% lower accuracies than our EPNET, respectively. Compared to the *ResNet-11* that shared the same main branch structure, our EPNET achieved 40% higher budget-constrained accuracy. Our results demonstrate the effectiveness of our EPNET in operating with stringent resource.



(a) *Max-Min MNIST*



(b) *Multi-scale Fashion MNIST*



(c) *CIFAR-10*

Figure 4.10: *Average benefit score during the training phase.* The cost sensitivities for each dataset are 5×10^{-7}, 2×10^{-7}, 2×10^{-7}.

4.4.7 Case Study: EPNET's Adaptivity to Classification Difficulty

The desired controller should be able to identify the samples which are easy to be classified at early stage. This property is well supported by the experiment on Max-Min MNIST and Multi-scale Fashion MNIST, which consist of samples of different difficulty levels in term of classification.

Multi-scale Fashion MNIST. In Figure 4.7 we show the cumulative exiting rate over all exits for the Multi-scale Fashion MNIST. The curves that are closer to top-left corner are more tend to exit at earlier branch exits. About 90% of Type 1 samples stop before exit-5. The reason may be the objects in Type 1 are very small, so the shallower layers are sufficient to model them. The 80%-stop point of Type 2 and Type 3 samples are exit-6 and exit-8 respectively. It makes sense because compared with Type 1, the Type 3 samples have larger objects that require down-sampling to capture the objects. Compared with Type 2, the locations of object of Type 3 is random, so it is also more difficult than the Type 2. The Type 4 is at most bottom-right, because the samples are harder than other types due to they have two objects and need to tell which one is larger.

Max-Min MNIST. The similar finding is more obvious on the Max-Min MNIST. Figure 4.8(a) shows when the cost sensitivity is 5×10^{-7}, the easy samples mainly stop at the second, third and fourth exit. The medium samples mainly stop at 8-th exit. The hard samples mainly stop at 10-th and 11-th exit. This result indicates the controller learns to predict the easy samples at shallow stage and leave the hard samples to the deep stage, thus the computational cost is saved. Figure 4.8(a) shows when the cost sensitivity increases to 2.5×10^{-6}, the controller tends to left-shift the exiting distribution of all samples, because the model is more sensitive to the computational cost. As show in the Figure 4.8(c), when the cost sensitivity reaches 5×10^{-6}, something interesting happened. The controller chose to output the hard and medium samples at the first exit, even before the easy cases! The reason is to correctly classify the hard samples, we have to use the deep layers, but the computation cost is much larger than the benefit of correct classification, as the cost sensitivity is large now. So in this case, outputting the hard cases at beginning is reasonable.

All datasets.

For each exit, Figure 4.9 shows its accuracy on whole test set (denoted as acc_w), and on the exited samples (denoted as acc_e), as well as the exiting rate. On all the datasets, the acc_e is much higher than the acc_w at beginning. This reflects that the controller f_c^i can effectively find the samples which the classifier f_a^i can confidently predict, even without knowing the ground truth of sample difficulty. As the depth increases, the acc_w increases while the acc_e drops to below than acc_w. The reason is the easy cases have exited at previous branches, leaving the hard cases to the later branches.

4.4.8 Training Discussion

Now we discuss the training step use different method. As we have statemented, the early exiting problem can be viewed as a reinforcement learning task, and could be solved by classic REINFORCE algorithm based on Markov chain Monte Carlo (MCMC) to estimate the gradients. Here we show the training process using our EPNET and MCMC in the Figure 4.10. As we can see, on three datasets, the training reward of both method increased very quickly at beginning, then slow down. The EPNET always keeps higher reward at each iteration step comparing with the MCMC REINFORCE algorithm. Especially on the Multi-scale Fashion dataset, the reward curve of MCMC REINFORCE algorithm has large fluctuation, even we already add the baseline to reduce the variance of policy gradient. On the contrary, our EPNET not only achieves higher reward, but also has much smaller fluctuation on the reward increasing curve. We think the reason may be our EPNET use the gradient of the exact expectation of reward instead of the estimation based on MCMC.

4.4.9 Parameter Discussion

Lastly, we discuss the impact of the structure of controller on our EPNET performance. We train and test the EPNET on the CIFAR-10 under four configurations of controller, as show in the Table 4.2. Each configuration corresponds to different computational complexities. As we can see, increasing the depth and width of the controllers on each branch lead to higher computational cost of the controller (first three rows). However, the slightly higher computational cost on controller led to two orders of magnitude reduction in computational cost on whole network and better accuracy. This is because more complex controllers are better in learning the early-exiting policies.

Compared with Config 3, *i.e.*the best configuration in Table 4.2, the Config 4 use the similar structure of f_{cat} but without f_{in}, which means the controllers in Config 4 only use the logits to decide to stop or continue, neglecting the original information in the feature vector after GAP layer. So its performance is worse than Config 3. Even though Config 4 uses a larger controller than Config 2, its performance on benefit score, classification accuracy and computational cost are worse than Config 2. This result support our idea that combining the pooled feature vector and logits can improve the controller's performance.

4.5 Related Work

Rule-based Early-exiting. Teerapittayanon et al. proposed a multi-branch network named BranchyNet [31], which added several additional branches on CNNs including LeNet, AlexNet and ResNet. On each branch, the early exiting is controlled by the threshold of logit entropy. Huang et al. proposed a novel model called MSDNet, of which the structure is a stack of multiple DenseNet, for addressing the impacts of the multi-branch structure on the accuracy of branch classifier [32]. MSDNet is designed to provide coarse-level features to earlier branches and reduce the interference between branches. Li et al. further studied the problem of potential negative impacts of gradients from multiple branches and proposed methods to collaboratively improve the training of branches [33]. Both [32] and [33] used the softmax probability for making early-exiting decisions. Our work propose a learning-based early-exiting approach for better adapting to inference environment.

Dynamic Inference on CNNs.

Figurnov et al. proposed a spatial adaptive inference architecture called SACT [68] that can skip convolution within a residual block. Specifically, SACT calculates a halting score during every convolution in a residual block and decides whether to skip the next convolution in the same residual block. Veit et al. proposed a dynamic inference model called ConvNet-AIG [69] that aims to only execute the layers related to the category of input image. Concretely, ConvNet-AIG used a small network as a gated function to decide whether to execute a residual block or just jump over it through the shortcut link. Simiarly, Bengio et al. [70] proposed a method to dropout some units of a layer in neural network. Wang et al. proposed SkipNet [71] that leverages reinforcement learning to identify the suitable shallow networks per sample. Our work focuses on the co-design of a multi-branch network and its early-exiting policy for efficient dynamic inference.

Dynamic Inference on RNNs.

Minh et al. proposed a recurrent attention model (RAM) [29] on visual learning tasks. RAM can learn to only attend to the important regions without scanning the entire image, similar to SACT [68]. On the task of time series classification, Hartvigsen et al. [72] proposed a novel model EARLIEST to jointly minimize the classification error and the execution time of the model. Both RAM and EARLIEST and the works mentioned above [70, 71] are trained by REINFORCE algorithm. Our work also leverages reinforcement learning to obtain the early-exiting policy. As our MDP has much smaller searching space, our proposed controller can be trained in an efficient non-sampling fashion.

5

FiShNet: Fine-Grained Filter Sharing for Resource-Efficient Multi-Task Learning

5.1 Introduction

Multi-task learning (MTL) is a promising paradigm to improving the test accuracy of deep learning models that have to train with limited datasets. Two common ways to support such information sharing are: *(i) hard-sharing* where parameters are shared among the task-specific networks and *(ii) soft-sharing* where feature maps are shared. Concretely, hard-sharing approaches often employ hand-coded policies which result in static and coarse-grained information sharing. In contrast, soft-sharing techniques can learn fine-grained feature sharing directly from multi-task datasets by providing inputs access to fused feature maps.

Consequently, MTL models that achieve state-of-the-art accuracy are often ones using soft-sharing. However, the high accuracy of soft-sharing based MTL models comes with

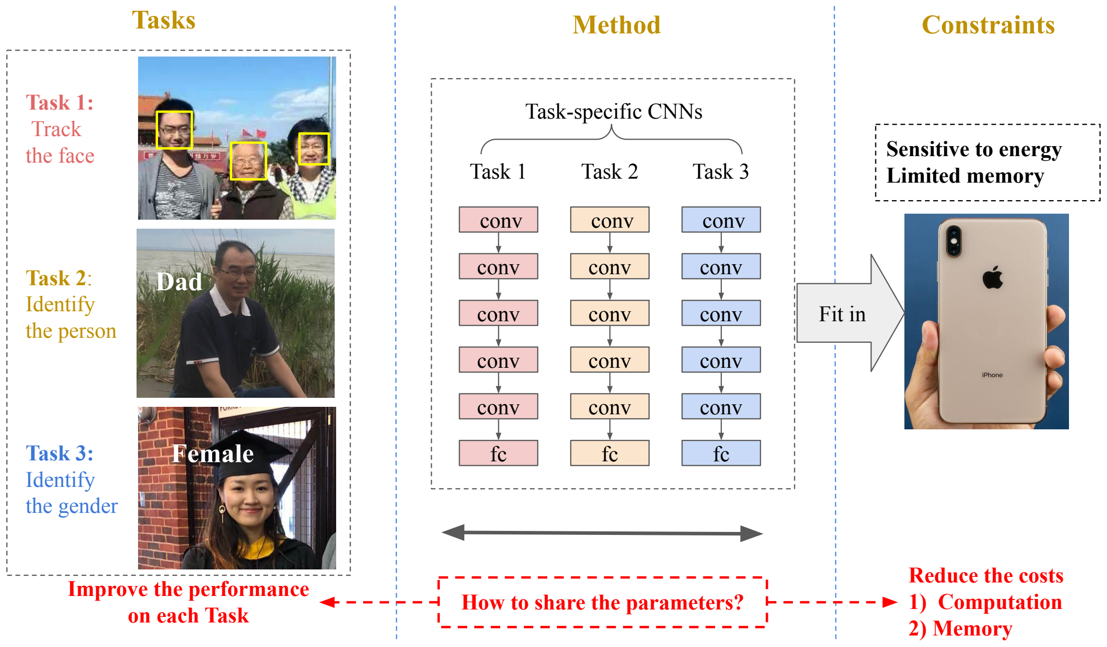


Figure 5.1: ***Resource-efficient MTL.*** *Three tasks are being learned on a multi-task dataset. The goal is to train a model for all three tasks that can be deployed on mobile devices with limited computational resources.*

the expensive resource requirement (both training and inference phases) that grows linearly with the number of tasks. For example, at inference phase, even when the end user is only interested in one out of the many tasks, *all* the task-specific models need to run to produce the inference response. Concurrently, resource-constrained end-user devices such as security camera or mobile devices are emerging as a natural fit for MTL models where high test accuracy is desirable for many related tasks. Figure 5.1 illustrates one such use cases. Unfortunately, the high resource requirement of soft-sharing based MTL models impedes the wide deployment to these resource-constrained devices.

In this paper, we investigate the problem of *resource-efficient* multi-task learning with the key goal of designing a resource-friendly MTL model that achieves high accuracy comparable to soft-sharing approaches while only consumes constant resource w.r.t. the number of tasks. In other words, we aim to incorporate the best of both the soft-sharing (high accuracy) and hard-sharing (low resource consumption) into a new class of models.

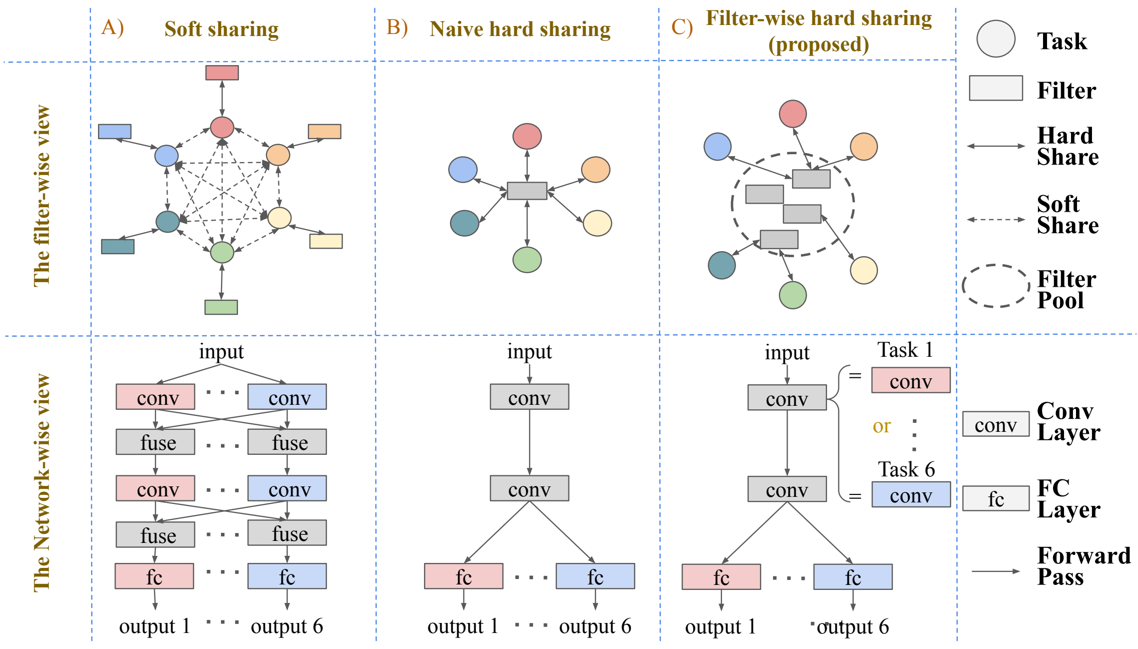


Figure 5.2: ***Comparison of different parameter-sharing methods for multi-task learning.*** *A) soft sharing [1, 2]; B) naive hard sharing [3, 4]; C) filter-wise hard sharing (this paper).*

The key challenge in designing resource-efficient MTL models lies in providing flexible and fine-grained sharing among tasks without incurring undesirable resource overhead. *Fine-grained sharing* and *flexibility* allow enlarging the search space which is useful for improving the potential amount of information sharing among different tasks and for adapting to different data distributions. Traditional hard-sharing approaches often employ coarse-grained layer-based sharing which then result in a fixed sharing structure, e.g., a tree-based structure that allow tasks to share layers that are closer to the input layer as illustrated in Figure 5.2B). Although existing soft-sharing approaches allow finer-grained sharing, they often incur more resource overheads both at the training and inference phases.

To address the above-mentioned challenges, we formulate the resource-efficient MTL problem as a *fine-grained filter sharing learning* problem, i.e., learning how to share filters at any given convolutional layers among multiple tasks. We propose a novel architecture called FISHNET which consists of two parts: the task-specific CNNs parameterized by

ϕ, and the filter allocators, parameterized by θ, that control what parameters to share among tasks-specific CNNs. Figure 5.2C) illustrates an example architecture with six tasks. In particular, the filter allocators learns to select the best filter from each filter pool for each task. Additionally, we introduce three techniques to handle the non-differentiable actions when training the allocators. Our proposed FiShNet architecture can be directly implemented and trained with existing deep learning frameworks such as Pytorch. Once trained, FiShNet can effectively support single-task inference scenarios by only loading the task-specific network.

5.2 Problem Formulation

In this paper, we study the problem of resource-efficient multi-task learning (MTL) on supervised vision tasks. Given a set of N supervised learning tasks $\mathcal{T} = \{t_1, t_2, ..., t_N\}$, each task t_k is associated with a task-specific model M_k and a dataset $\mathcal{D}_k = (\mathcal{X}_k, \mathcal{Y}_k)$, where $\mathcal{X}_k$ is a set of images and $\mathcal{Y}_k$ is a set of labels. Multi-task learning is based on the key assumption that all the tasks or some of them are related. Our goals of resource-efficient MTL are two-folds: 1) to improve the performance on each task by sharing information, i.e. features or parameters, among the $\mathcal{M} = \{M_1, M_2, ..., M_N\}$; 2) to satisfy the resource constraints for each task-specific CNNs. Specifically, we consider the following two constraints: *(i)* The computational complexity of inference on one task should not grows with the number of tasks, i.e. $|\mathcal{T}|$; *(ii)* the total size of all intermediate feature maps outputted by convolutions on one task should not grows with $|\mathcal{T}|$.

In this study, we focus on enabling fine-grained hard sharing across the $\mathcal{M} = \{M_1, M_2, ..., M_N\}$, and assume all M_k has the same architecture. Thus we formulate the resource-efficient MTL as a *fine-grained filter sharing learning problem.*

Specifically, given a multi-task learning problem $\mathcal{P} = (\mathcal{T}, \mathcal{M}, \mathcal{D})$, let's use the f_{ij}^k to denote the $i\text{-}th$ filter in the $j\text{-}th$ convolutional layer of M_k, the task-specific CNN for the task $t_k \in \mathcal{T}$. For all k in $\{1, 2, ..., |\mathcal{T}|\}$, the filters f_{ij}^k are picked from a pool of filters $\mathcal{F}_{ij} = \{f_1, f_2, ...f_N\}$. The goal of filter-level hard sharing is to learn a policy that selects the best filter for each $M_k \in \mathcal{M}$ to improve its performance on its corresponding task $t_k \in \mathcal{T}$.

It is easy to see that the filter-level hard sharing naturally satisfies the resource constraints of the MTL problem. That is, to perform inference on a task t_k, we only need to run a single network M_k consisting of filters chosen by the learned filter selection policy. Consequently the computational complexity and total memory consumption of intermediate feature maps are constant w.r.t $|\mathcal{T}|$. Furthermore, our filter-level sharing can be considered as a generalized version of the traditional hard sharing problem. That is, if M_a and M_b decide to share the parameters between filter f_{ij}^a and f_{ij}^b, they just need to pick the same filter from the pool $\mathcal{F}_{ij}$. The solution space $\mathcal{S}$ of filter-level hard sharing problem is defined on the filter pools $\mathcal{F}_{ij}$ of all filters in architecture of M_k. The traditional tree-shape hard sharing methods belong to a subset of $\mathcal{S}$.

5.3 FISHNET: Resource-efficient MTL via Fine-grained Sharing

5.3.1 Overall Structure

The proposed FISHNET consists of two parts: 1) *The task-specific CNNs*, which have the same architecture, and work for each task as a normal CNN respectively. We only want to share the parameters in convolutional layers, and every CNN should keep its own BN layers and fully-connected layers after convolutions. 2) *The allocator units*, which decide the way of sharing parameters across task-specific CNNs. The goal of this study is to

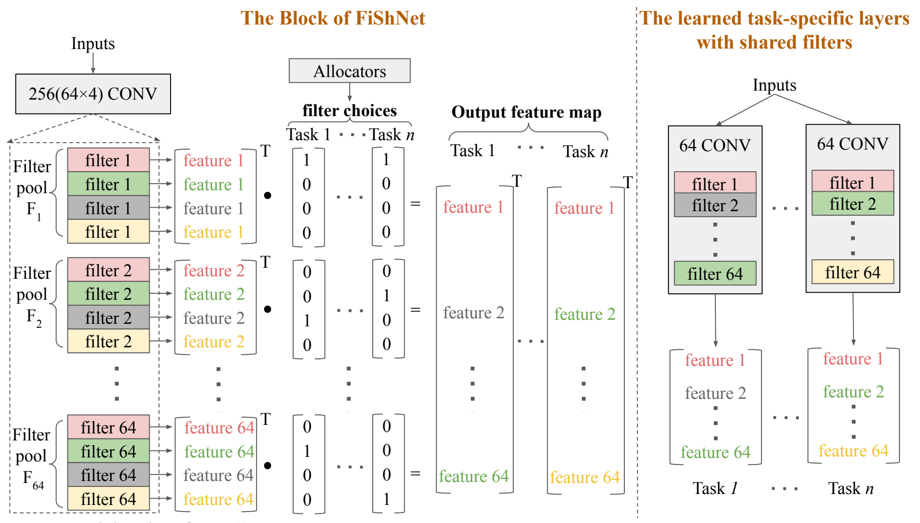


Figure 5.3: ***A block of*** **FISHNET.** *Given n tasks, the block aims to share the filters across the n task-specific convolutional layers, each consists of 64 filters associated with 64 filter pools respectively. In this example we set the pool size = 4. The block has two trainable parts:1) a large convolutional kernel consisting all filters in the pool; 2) the allocators to pick filters (feature maps) for each task-specific layer. By stacking this block, it can learn to share the filters across CNNs of* ***any architecture*** *for* ***any number*** *of tasks in a* ***tunable searching space****.*

design a mechanism that can learn the task-specific CNNs as well as the allocator units directly and synchronously from the training set.

5.3.2 Learning Fine-grained Filter Sharing

In this section, we show the idea that our hard sharing mechanism can be expressed in an equivalent form of normal convolutional networks, which can be easily implemented by open-source deep-learning library e.g. Tensorflow or Pytorch. We will introduce the basic block of FISHNET which is aimed to implement the filter picking for a convolutional layer, as show(shown) in Figure 5.3. In the following notations, we omit the index of the convolutional layer.

Given the task set $\mathcal{T}$, let's start from a convolutional kernel ϕ^k for task t_k. Suppose the $\phi^k \in \mathbb{R}^{C_{in} \times H \times W \times C_{out}}$, it means ϕ^k consists of C_{out} filters $f_o^k \in \mathbb{R}^{C_{in} \times H \times W}$, and $o \in (1, 2, ..., C_{out})$. Here, C_{in} is the number of input channel, H is the height, and

W is the width. Given the channel index o, the filter f_o^k for each task t_k is selected from a filter candidate set $\mathcal{F}_o$. Here we assume the pool size $|\mathcal{F}_o|$ are all the same for $o \in (1, 2, ..., C_{out})$.

To share the parameters across ϕ^1, ϕ^2, ..., $\phi^{|\mathcal{T}|}$, we concat the filters in the $\mathcal{F}_1$, $\mathcal{F}_2$, ..., $\mathcal{F}_{C_{out}}$ together to form a large convolutional kernel $\phi_{\mathcal{F}}$:

$$\phi_{\mathcal{F}} = \bigcup_{o=1}^{C_{out}} \mathcal{F}_o \tag{5.1}$$

It is obvious that the kernel $\phi_{\mathcal{F}}$ contains all candidate filters. Then given an input $\mathbf{X} \in \mathbb{R}^{H \times W \times C_{in}}$, we perform convolution on $\mathbf{X}$ by the kernel $\phi_{\mathcal{F}}$, resulting a feature map $\mathbf{A}$:

$$\mathbf{A} = \phi_{\mathcal{F}} * \mathbf{X} \tag{5.2}$$

Here the $\mathbf{A} \in \mathbb{R}^{H \times W \times C_{out}|\mathcal{F}_o|}$, where every $|\mathcal{F}_o|$ channels are outputted by the filters from the same candidate set $\mathcal{F}_o$. To clarify the further calculation, $\mathbf{A}$ is reshaped to $H \times W \times C_{out} \times |\mathcal{F}_o|$. Then let's consider how to generate the output(feature maps) for each task. For a certain task t_k, there is an allocator unit $\theta_o^k \in \mathbb{R}^{|\mathcal{F}_o|}$ corresponding to each filter candidate set $\mathcal{F}_o$. The $Softmax(\theta_o^k)$ parameterize a $|\mathcal{F}_o|$-way categorical distribution over $\mathcal{F}_o$, from which the choice of filters are sampled. We use the one-hot vector $\mathbf{c}_o^k \in \mathbb{R}^{|\mathcal{F}_o|}$ to denote the choice of filters from $\mathcal{F}_o$ for task t_k:

$$\mathbf{c}_o^k \sim \text{Categorical}(|\mathcal{F}_o|, \theta_o^k) \tag{5.3}$$

Because the choice of the filter is equivalent to the choice of the feature map, the feature map $\mathbf{A}_o^k \in \mathbb{R}^{H \times W}$ for task t_k is calculated by the following way:

$$\mathbf{A}_o^k = \sum_{m=1}^{|\mathcal{F}_o|} c_{om}^k \mathbf{A}_{om} \tag{5.4}$$

Here c_{om}^k is the m-th scalar element in c_o^k, $\mathbf{A}_{om}$ is the 2D slice in $\mathbf{A}$ given the output channel index o and candidate index m in the $|\mathcal{F}_o|$. Once we have the feature maps $\mathbf{A}_o^k$ for task t_k, we can pick the corresponding filters from each candidate set $\mathcal{F}_o$, and form a convolutional kernel ϕ^k for task t_k:

$$\phi^k = \bigcup_{o=1}^{C_{out}} \left\{ f_m | f_m \in \mathcal{F}_o, c_{om}^k = 1 \right\} \tag{5.5}$$

We can see the block of FISHNET has two kinds of parameters: 1) the large convolutional kernel $\phi_{\mathcal{F}}$; 2) the parameters of allocator units $\theta = \{\theta_o^k | o \in \{1, 2, ..., C_{out}\}, k \in \{1, 2, ..., |\mathcal{T}|\}\}$. Once the the $\phi_{\mathcal{F}}$ and θ are trained from dataset, we can pick the trained filters to form a task-specific convolutional layer for each task. By stacking this block, it can learn to share the filters across CNNs of any architecture for any number of tasks in a searching space tuned by the pool size $|\mathcal{F}_o|$. The $\phi_{\mathcal{F}}$ can be simply trained by standard backward propagation. We will discuss how to train the θ_o^k in the next section.

Similarity and difference compared with soft sharing: Although the proposed FISHNET belongs to hard sharing method, its architecture looks more like a soft sharing method in the training phase. If we relax the $|\mathcal{F}_o|-$dimension vector $\mathbf{c}_o^k$ from a one-hot vector to a common vector $\mathbf{w}_o^k \in \mathbb{R}^{|\mathcal{F}_o|}$, FISHNET is somehow similar to the famous Cross-stitch network in the manner of weighted adding the feature maps from all task-specific network (but be attention that they are still very different in the network architecture). From this view, the FISHNET learns a discrete feature fusion in the training phase, and naturally breaks down into common single CNNs for each task in the testing phase.

That's why the FISHNET can learn fine-grained and flexible sharing structure as what soft sharing methods do and still be time and space efficient as naive hard sharing.

5.3.3 Training Considerations

5.3.3.1 Train the Allocator by Gumbel-Softmax Method

Given a filter candidate set $\mathcal{F}_o$, the filter choice $\mathbf{c}_o^k$ for task t_k is drawn from the policy $\pi_{\theta_o^k}$, which parameterized by allocator unit θ_o^k associated with $\mathcal{F}_o$. The sampling in Eq. 5.3 is a non-differentiable operation, so θ can't be directly trained by the standard backward propagation. To solve this problem, we use the Gumbel-Softmax method [23] to train the allocator.

Gumbel-Softmax: The first step, we generate a random vector $\mathbf{g} \in \mathbb{R}^{|\mathcal{F}_o|}$, in which the scalar element g_m, $m \in 1, 2, ..., |\mathcal{F}_o|$ is drawn from a standard Gumbel distribution Gumbel(0,1). Then the choice $\mathbf{c}_o^k \in \mathbb{R}^{|\mathcal{F}_o|}$ can be generated in a alternative but equivalent way:

$$\mathbf{c}_o^k = \text{one-hot}\,(\arg\min_m [g_m + \log \pi_{\theta_o^k}(m)]) \tag{5.6}$$

In the above equation, the argmin is still a non-differentiable operation. According to [23], the one-hot and argmin can be relaxed by using Softmax function with temperature $\tau \in (0, +\infty)$, thus we get a approximate calculation of $\mathbf{c}_o^k$:

$$\mathbf{c}_o^k \approx \text{Softmax}\,(\frac{\mathbf{g} + \log \pi_{\theta_o^k}}{\tau}) \tag{5.7}$$

The higher τ causes the $\mathbf{c}_o^k$ to get closer to the uniform distribution, and the lower τ causes the $\mathbf{c}_o^k$ to get closer to the categorical distribution. By the Eq 5.7, the $\mathbf{c}_o^k$ is differentiable. So we can train the parameters θ_o^k of allocators with backward propagation, as well as the parameters ϕ^k of task-specific networks. As we prefer a deterministic system, the proposed FISHNET will be firstly trained by Gumbel-Softmax for several epochs, then we fix the filter choice $\mathbf{c}_o^k$ calculated by Eq. 5.6, and continue to train the ϕ^k.

5.3.3.2 Alternative Training Methods

In the experiment of this paper, we used the Gumbel-Softmax trick described above to train our FISHNET, due to its simplicity of implementation. Here we also provide another two ways for training.

Network Architecture Searching methods: We can also cast the training of FISHNET in the form of *Network Architecture Searching (NAS)* problem. Under this setting, we prefer to solve it by the gradient-based searching methods e.g. DARTS [73]. The basic idea could be:

1) Divide the training samples to two parts D_{train} and D_{val}.

2) Train the task-specific networks on D_{train} and the allocator units on D_{val}. The two parts can be trained alternately by turns: first fix the allocator units and train the task-specific networks, then fix the task-specific networks and train the allocator units.

3) At last fix the allocator units and train the task-specific networks on both D_{train} and D_{val}.

As following the default setting of DARTS, we can directly use the Softmax probabilities of the allocator weights to train the sharing mechanism. This may allow us to get rid of the static behaviour during training, compared with Gumbel-Softmax trick.

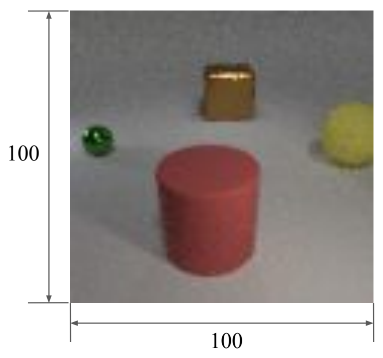
Policy gradients based methods: Another way to handle the non-differentiable sampling operation is the policy gradients methods which are originally and mainly used in the reinforcement learning problems. Here we don't need to define the searching of sharing weights as a Markov process but view it as a simple multi-armed bandit problem. The sampled choices $\mathbf{c}_o^k$ can be viewed as an action, and the associated reward could be the number of correctly predicted tasks. Then the problem can be solved by the REINFORCE rule [22]. However, the policy gradients methods usually have a very high variance in terms of the estimated gradients.

5.3.4 Regularization on Sharing

In the experiment, our FISHNET is solely trained by the summation of losses from different tasks. It implies that, in theory, the FISHNET can possibly learn to not share parameters and converge to exactly the single task networks. As suggested by [74], we can add a regularization term to encourage the FISHNET to share weights. For the filters f_{ij}^{k1} and f_{ij}^{k2} from two tasks t_{k1} and t_{k1}, given the layer index i and channel index j, the regularization may take form of the KL-divergence $KL(\pi_{\theta_{ij}^{k1}}||\pi_{\theta_{ij}^{k1}})$.

However, if the naive hard sharing method can fit the training set, the above regularization term may push the model to share as many filters as possible, so our FISHNET may be downgraded to the naive hard sharing. As we will show later in the experiment, although the naive hard sharing method can fit the training sample very well, its test accuracy may be often lower than the soft sharing methods and FISHNET. This result indicate encouraging sharing may have a potential risk related to the "negative transfer" phenomenon [75]. On the contrary, the sharing policies in the previous studies [2, 74, 76] are also solely trained by task loesses, and usually bring good test performance. This motivates us to use another way for regularization on sharing. As stated in previous sections, given an index i of convolutional layer and index j of filter, the filters f_{ij}^t of each task t, are picked from several candidate sets $\mathcal{F}_{ij}$. If we want to let the CNNs share more or fewer filters, we can simply tune the value of $|\mathcal{F}_{ij}|$. The hyperparameter $|\mathcal{F}_{ij}|$ can only affect the probability of the choice of filters, but not encourage the CNNS to share as many filters as possible, so the MTL model will not reduce to the naive hard sharing model.

Image



Attributes

Shape = {cylinder, cube, sphere}

Color = {red, yellow, green, blue, brown, purple, cyan, gray}

Size = {large, small}

Material = {rubber, metal}

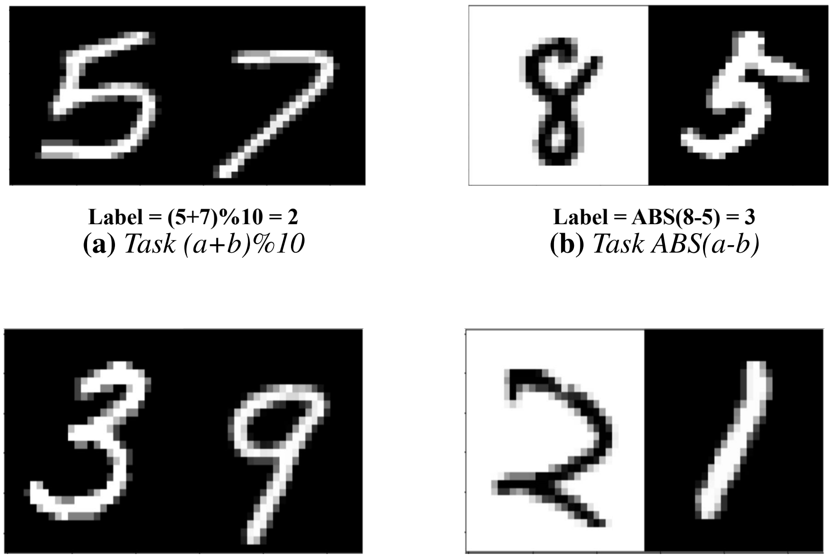
Labels	Explanation
T1: Label = "equal"	#cylinder(warm color) = #sphere(cool color) = 1
T2: Label = "equal"	#small shpere = #large cube = 1
T3: Label = "equal"	#rubber cube = #metal cylinder = 0
T4: Label = 57	Encode [cube, shpere, cylinder, sphere] to 2010 (ternary)
T5: Label = 57	Encode [cube, shpere, cylinder, sphere] to 2010 (ternary)
T6: Label = 45	Encode [cube, shpere, cylinder, sphere] to 1200 (ternary)
T7: Label = 32	Encode [cube, shpere, cylinder, sphere] to 1012 (ternary)
T8: Label = 1	#red cylinder = 1
T9: Label = 2	#yellow sphere + #green sphere = 1+1
T10: Label = 2	#red sylinder + #yellow sphere = 1+1
T11: Label = 0	No cube is in blue or green
T12: Label = 1	#yellow shpere = 1
T13: Label = 1	#red sylinder = 1

Figure 5.4: *An example of the CLEVR-sharing tasks.*

5.4 Experimental Evaluations

5.4.1 Multi-task Learning Datasets

We generate two datasets for evaluating the accuracy and computation complexity of our FiShNet on multi-task learning. First, we leverage and modify the code for CLEVR [77], an image dataset that is widely used in multi-task learning [78] and continual learning [79], and render new CLEVR images. Specifically, for each image, we place four objects at fixed regions. The attributes of object (shape, materials, color, size) are uniformly sampled. Figure 5.4 shows an example image. Based on these new CLEVR images, which we refer to *CLEVR-sharing* data set, we generate four groups of multi-task learning problems which are summarized in Table 5.1. Second, we leverage the original digit MNIST dataset and generate a new dataset, referred to as *MNIST-sharing*, by randomly picking two images from the original dataset and concatenating them together to build a new sample. Based on the *MNIST-sharing* dataset, we generate four tasks each with 2000 training



Label = (5+7)%10 = 2
(a) *Task (a+b)%10*

Label = ABS(8-5) = 3
(b) *Task ABS(a-b)*

Label = MAX(3, 9) = 9
(c) *Task MAX(a,b)*

Label = MIN(2, 1) = 1
(d) *Task MIN(a,b)*

Figure 5.5: ***Examples of four tasks from our* MNIST-sharing *dataset.***

images and 10000 testing images. These four tasks are: *(i)* calculating the sum of two digits and then mod 10; *(ii)* calculating the absolute difference between two digits; *(iii)* calculating the maximum of two digits; *(iv)* calculating the minimum of two digits. For the task 2 and 4, we randomly pick the left or right half of an image to flip its gray degree. Figure 5.5 shows an example image for each task.

5.4.2 Baselines

We compare FISHNET to three categories of approaches: *(i)* hard-sharing multi-task network; *(ii)* soft-sharing multi-task networks [1, 2]; *(iii)* and single-task networks. Moreover, we perform a breakdown study of our proposed FISHNET by comparing it to two weaker versions: *(i)* FISHNET *(random)* where the filters of each task-specific CNNs are randomly chosen from the filter pools when we initialize the model. This baseline is very similar to the Task-Routing model proposed by Strezoski et al. [80]. *(ii)* FISHNET *(*$|\mathcal{F}| = 1$*)* where each of the filter pool has only one filter. so it is very similar to the

Table 5.1: ***Tasks on our* CLEVR-sharing *dataset.*** *We define a total of 16 tasks that form four groups of multi-task learning problems.*

Group	Task ID	Label set	Description
Group 1 & 4	T1	{more, less, equal}	Comparing the # cylinder (warm color) and # sphere (cool color)
Group 1	T2	{more, less, equal}	Comparing the #sphere (small) and #cube (large)
Group 1	T3	{more, less, equal}	Comparing the #cube (rubber) and #cylinder (metal)
Group 1 & 2	T4	{0, 1, 2, ..., 80}	Denote the image by a 4 bit ternary number: encode the objects to {0,1,2} based on their shape
Group 2	T5	{0, 1, 2, ..., 80}	Same with above, but encode the objects based on their shape and color
Group 2	T6	{0, 1, 2, ..., 80}	Same with above, but encode the objects based on their shape and size
Group 2	T7	{0, 1, 2, ..., 80}	Same with above, but encode the objects based on their shape and material
Group 3	T8	{0, 1, 2, 3, 4}	Counting the total # of cube and cylinder, in the color of red or green
Group 3	T9	{0, 1, 2, 3, 4}	Counting the total # of cube and sphere, in the color of yellow or green
Group 3	T10	{0, 1, 2, 3, 4}	Counting the total # of sphere and cylinder, in the color of red or yellow
Group 3	T11	{0, 1, 2, 3, 4}	Counting the # of cube, in the color of blue or green
Group 3	T12	{0, 1, 2, 3, 4}	Counting the # of sphere, in the color of yellow or brown
Group 3	T13	{0, 1, 2, 3, 4}	Counting the # of cylinder, in the color of red or cyan
Group 4	T14	{0, 1, 2, ..., 80}	Same with T4, but the color is sampled from {red, yellow}
Group 4	T15	{more, less, equal}	Same with T2, but the color is sampled from {brown, purple}
Group 4	T16	{more, less, equal}	Same with T3, but the color is sampled from {cyan, blue}

Table 5.2: ***The encoding rule for task group 2:*** *By encoding the each of the four objects in an image to 0,1,2 in the clockwise order, the image is denoted by a 4-bit ternary number which serves as the label. The total number of possible labels is* $3^4 = 81$.

Digits	T4 (shape only)	T5 (shape + color)	T6 (shape + size)	T7 (shape + material)
0	sphere	sphere(warm color), cylinder(cool)	sphere(small), cylinder(large)	sphere(rubber), cylinder(metal)
1	cylinder	cylinder(warm color), cube(cool)	cylinder(small), cube(large)	cylinder(rubber), cube(metal)
2	cube	cube(warm color), sphere(cool)	cube(small), sphere(large)	cube(rubber), sphere(metal)

naive hard sharing, but its task-specific CNNs have their own BN layers which are not shared with each other. When the images of different tasks are generated from the same distribution, FISHNET ($|\mathcal{F}| = 1$) is almost same with the naive hard sharing. So We only compared with this baseline in the experiments where the images of different tasks are generated from different distributions.

For hard-sharing, we adopt a common setting that appears in the comparison in related studies [1, 2, 74, 76, 81] where the convolutional layers, as well as the following BN, ReLU, and pooling layers, are shared across all tasks; and each task has its own fully-connected layers and output layer. For soft-sharing, we compare both the cross-stitch network [1] and the NDDR network [2]. The former is one of the first proposed soft-sharing network for multi-task learning while the latter represents the current state-of-the-art. Both soft-sharing networks let each task has its own network, i.e., the parameters are not shared. For cross-stitch network, the feature maps at certain layer outputted by

Table 5.3: *The results (test accuracy) of all methods (× complexity) on task group 1 (rank)".*

Model	**T1**	**T2**	**T3**	**T4**
FiShNet (×1)	87.90(1)	98.05(1)	80.55(3)	95.20(2)
FiShNet (random) (×1)	68.40(5)	74.20(6)	68.70(5)	89.00(4)
Hard sharing (×1)	70.15(4)	92.90(5)	75.50(4)	85.45(4)
Single nets (×1)	65.90(6)	79.05(5)	62.85(6)	70.25(6)
NDDR (×4)	82.20(3)	94.90(3)	80.65(2)	96.00(1)
Cross-stitch (×4)	87.75(2)	95.30(2)	83.05(1)	93.35(3)

Table 5.4: *The results (test accuracy) of all methods (× complexity) on task group 2(rank)".*

Model	**T4**	**T5**	**T6**	**T7**
FiShNet (×1)	96.40(2)	94.45(2)	93.95(3)	91.20(2)
FiShNet (random) (×1)	91.65(4)	57.20(4)	86.60(4)	73.80(4)
Hard sharing (×1)	78.35(5)	20.25(5)	64.80(5)	29.95(5)
Single nets (×1)	70.25(6)	8.70(6)	57.25(6)	14.05(6)
NDDR (×4)	99.30(1)	98.75(1)	98.45(1)	98.30(1)
Cross-stitch (×4)	94.55(3)	92.20(3)	94.50(2)	90.15(3)

all tasks are weighted added together and feed to the next layer. For NDDR, it uses 1×1 convolution and BN for feature fusion instead of the weighted sum. For single task networks, we train one convolutional network for each task. Tasks within the same group use the same architecture. As these convolutional networks also serve as the backbone for the multi-task networks (hard sharing, soft sharing, and our proposed FiShNet), the corresponding results are insightful baselines.

5.4.3 Hyperparameter Settings

Network architecture: For the *CLEVR-sharing* tasks, we use a Resnet with 10-layer as the single task network and backbones of the MTL sharing models. The number of filters are [8, 8, 16, 32, 64] for the conv layer (or the residual blocks). Each residual block has two conv layers. We set the pool size $|\mathcal{F}|$ of filters equal to the number of tasks $|\mathcal{T}|$ in each group. For the *MNIST-sharing* tasks, we use a simple CNN with 4 conv layers and 2 FC layers. The number of conv filters are [8, 16, 32, 64]. Each conv layer is

Table 5.5: *The results (test accuracy) of all methods (× complexity) on task group 3(rank)".*

Model	T8	T9	T10	T11	T12	T13
FISHNET (×1)	98.80(1)	89.10(1)	84.90(2)	80.90(1)	99.30(1)	86.20(1)
FISHNET ($\lvert\mathcal{F}\rvert = 1$) (×1)	98.35(2)	81.90(2)	85.50(1)	68.10(3)	99.30(1)	72.25(3)
FISHNET (random) (×1)	79.95(4)	72.00(4)	65.65(4)	61.50(4)	45.70(7)	62.50(4)
Hard sharing (×1)	45.80(5)	48.10(5)	51.90(5)	32.05(6)	42.90(4)	37.10(5)
Single nets (×1)	98.15(3)	79.75(3)	78.20(3)	76.15(2)	97.85(3)	74.85(2)
NDDR (×6)	34.85(7)	38.35(7)	36.35(7)	30.60(7)	30.25(6)	27.80(7)
Cross-stitch (×6)	38.85(6)	39.85(6)	50.50(6)	34.30(5)	34.05(5)	38.40(6)

Table 5.6: *The results (test accuracy) of all methods (× complexity) on task group 4(rank)".*

Model	T1	T14	T15	T16
FISHNET (×1)	63.15(4)	89.00(1)	92.90(1)	64.80(2)
FISHNET ($\lvert\mathcal{F}\rvert = 1$)(×1)	63.95(3)	58.40(2)	86.65(3)	63.85(3)
FISHNET (random) (×1)	62.40(5)	52.05(4)	91.10(2)	61.05(4)
Hard sharing (×1)	65.75(2)	16.15(7)	72.10(6)	39.80(7)
Single nets (×1)	60.30(7)	55.02(3)	83.15(5)	69.35(1)
NDDR (×4)	69.55(1)	36.10(5)	85.30(4)	40.75(6)
Cross-stitch (×4)	61.50(6)	28.20(6)	58.15(7)	54.75(5)

followed by a BN [67], ReLU, and max-pooling layer. The number of hidden FC layers are [100, 50]. Each hidden FC layer is followed by a 1D BN and ReLU. We tested the FISHNET using different settings of $\lvert\mathcal{F}\rvert$ which are disscussed detailly in the section 5.4.6. For Cross-stitch and NDDR soft-sharing networks, the feature fusion is performed after every downsampling operation.

Hyperparameter for training: For all tasks, we used the Kaiming initialization [66]. For the *CLEVR-sharing* tasks, we first train the proposed FISHNET by 20 or 40 epochs until the total classification loss is converged to below 1. Then we fix the learned network architecture and train the network by another 20 epochs. The batch size is 50 and the learning rate is 0.01. For FISHNET and all baselines, we set the weight decay to be 0.0001. For the *MNIST-sharing* tasks, we first train the proposed FISHNET by 20 epochs until the total classification loss is converged to below 1. Then we fix the learned network architecture and train the network by another 10 epochs. The batch size is 16 and the

Table 5.7: ***The results (test accuracy) of all methods (× complexity) on MNIST tasks(rank)".***

Model	(a+b)%10	ABS(a,b)	MAX(a,b)	MIN(a,b)
FISHNET (best) (×1)	94.16(1)	92.23(1)	95.53(1)	95.07(1)
FISHNET ($\|\mathcal{F}\| = 4$) (×1)	92.64(4)	89.68(4)	94.14(4)	94.08(4)
FISHNET ($\|\mathcal{F}\| = 2$) (×1)	93.61(3)	92.08(2)	95.18(2)	94.88(2)
FISHNET ($\|\mathcal{F}\| = 1$)(×1)	93.94(2)	90.44(3)	95.13(3)	94.26(3)
FISHNET (random) (×1)	86.83(7)	69.35(8)	91.98(7)	89.52(7)
Hard sharing (×1)	75.04(9)	52.46(9)	88.54(9)	84.75(9)
Single nets (×1)	85.26(8)	74.25(7)	91.37(8)	87.04(8)
NDDR (×1)	87.14(6)	83.79(6)	92.29(6)	91.23(5)
Cross-stitch (×1)	88.09(5)	83.81(5)	92.68(5)	89.68(6)

learning rate is 0.01. There is no weight decay for all methods.

5.4.4 Summary of Key Findings

We compare the performance of FISHNET to all baselines described in Section 5.4.2 on both the test accuracy and the inference computation complexity. Before we go to the details, we summarize our three key findings as follows and will detail the comparison for each multi-task group subsequently.

The results on tasks from the same distribution: When the images of each task are from the same distribution, we find the proposed FISHNET can achieve comparable performance as the state-of-the-art soft sharing methods, i.e. NDDR and cross-stitch. While the computational complexity and memory cost of FISHNET is only $1/n$ of the soft sharing method, here n is the number of tasks in a group.

The results on tasks from different distributions: When the images of each task are from different distributions, we find the FISHNET outperforms all baselines by a wide margin. The performances of the naive hard sharing, cross-stitch and NDDR can be even worse than networks trained on single task, because the problematic effect caused by sharing BN operations [82]. This finding supports our design that the tasks-specific networks should not share the BN layers for tasks from different distributions.

Comparison with other sharing policies: We compared FISHNET with the single task networks, the naive hard sharing (or FISHNET ($|\mathcal{F}| = 1$) when the tasks are from different distribution), and the FISHNET (random). They can be viewed as three trivial cases of FISHNET, which are sharing no convolutional filters, sharing all convolutional filters, and randomly sharing convolutional filters, respectively. In the following four experiments, FISHNET outperforms all of the trivial versions by a wide margin, which provided a concrete evidence that the sharing structure learned by FISHNET is non-trivial.

5.4.5 Performance of *CLEVR-sharing* Tasks

5.4.5.1 Performance on Group 1

In group 1, the tasks are comparing the number of certain classes. The details of the tasks are listed in the Table 5.1. As the original setting of CLEVR, The shape of objects is uniformly sampled from {"sphere", "cylinder", "cube"}. The material is uniformly sampled from {"metal", "rubber"}. The size is uniformly sampled from {"large", "small"}. The color is uniformly sampled from a set of 8 colors. We generate 2000 training images for each tasks, and all tasks share the same 2000 test images.

Results and analysis: The results are shown in Table 5.3. The performances of the proposed FISHNET and the soft sharing models are very close to each other. What's more, the FISHNET achieved highest average accuracy across four tasks, while its computational cost and size of feature maps are as small as the hard sharing network and single task networks, which is only 25% of the soft sharing methods. This reflects the FISHNET can learn flexible sharing structure as the soft sharing models, and keep low computational and memory cost as the naive hard sharing. Besides, the FISHNET's performance on each task is higher than the naive hard sharing by 5%-17%, and higher than FISHNET (random) by 12%-24%. indicating the sharing solution learned by FISHNET is not trivial. The single task networks have the lowest accuracies, indicating the MTL methods can greatly alleviate overfitting on these tasks.

5.4.5.2 Performance on Group 2

In group 2, the images are represented by 4-bit ternary numbers, by encoding objects to 0,1,2 in the clockwise order. The detailed encoding rules are listed in Table 5.2. We use the same training and test sets of group 1 here, but the labels are different.

Results and analysis: The results are shown in Table 5.4. The NDDR outperforms all other methods on these tasks. Our FISHNET is only secondary to NDDR, but notice that FISHNET has only 25% computational and memory cost of NDDR and Cross-stitch. Compared with the FISHNET (random), FISHNET's performance on each task is higher by 4.75%, 37.25%, 7.35%, 17.40%. The accuracy gap between FISHNET and naive hard sharing is up to 74%, which is even larger than that in group 1. Although the accuracies of hard sharing are much lower than other MTL methods, it's still much better than the single task networks. We can see that the tasks in group 2 are harder than group 1, since they are all 81-class classification problems and the networks need to recognize all shapes on all tasks. We think that is the main reason that the accuracy gaps are increased a lot.

5.4.5.3 Performance on Group 3

In group 3, the tasks are counting the number of certain objects. The details of the concerned objects are listed in the Table 5.1. We generated 1000 training samples and 2000 test samples for each task in group 3. This setting is different with the previous two experiments: the tasks' test sets are not shared. The reason is the concerned objects are different across tasks, and we keep the classes balanced in all training sets and test sets. As a result, the samples from different groups are not from the same distribution.

Results and analysis: The results are shown in the Table 5.5. This time our FISHNET outperforms all baselines. The performances of FISHNET and its trivial version FISHNET ($|\mathcal{F}| = 1$) are close to each other on the tasks T2, T10, and T12. However, on the rest tasks FISHNET greatly outperforms the FISHNET ($|\mathcal{F}| = 1$) by 7%, 13%, 11%. Compared with another trivial version FISHNET (random), its accuracies are lower than FISHNET by 12%-44%. We notice that all the hard and soft sharing baselines' performances are even much lower than single task networks. The same phenomenon will be found in the next experiment too. Comparing the results of hard sharing with FISHNET ($|\mathcal{F}| = 1$) which is almost same with hard sharing but doesn't share the BN layers, we can see the problem is caused by the shared BN layers. We will provide a detailed discussion latter in the section 5.4.7.

5.4.5.4 Performance on Group 4

We generated 1000 training samples and 2000 test samples for each task. The tasks in group 4 are almost same with those in group 1, but their colors of objects are from very different distributions. As listed in the Table 5.1, the tasks T14, T15, and T16 share no colors. So it may make this group harder than the previous three groups for multi-task learning.

Results and analysis: The results are shown in the Table 5.6. Once again, our FISHNET outperforms all baselines in terms of average accuracy. Although on the tasks T1 and T16, the performances of FISHNET are not highest, all the methods get very similar accuracies on these two tasks. FISHNET clearly defeats the baselines (except for our trivial versions) by at least 34% and 7% on T14 and T15 respectively. The performances of FISHNET on T14, T15, T16 are higher than FISHNET ($|\mathcal{F}| = 1$) by 30%, 6%, 1%, and slightly lower on T1 by 0.8%. Also, FISHNET outperforms another trivial version FISHNET (random) on all tasks by up to 37%. Again, the naive hard sharing, cross-stitch, and NDDR don't show advantages compared with single-task networks.

Table 5.8: ***The results comparison of the MTL baselines when the BN layers are in the training mode (denoted as modelname+) v.s. in the testing mode.***

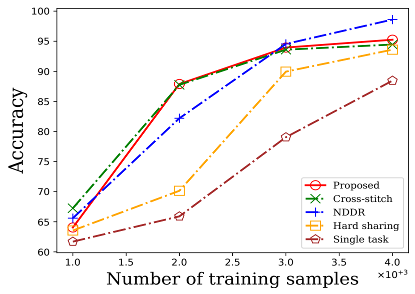
	Group 3 of CLEVR-sharing						Group 4 of CLEVR-sharing				MNIST-sharing			
Model	**T8**	**T9**	**T10**	**T11**	**T12**	**T13**	**T1**	**T14**	**T15**	**T16**	**(a+b)%10**	**ABS(a-b)**	**MAX(a,b)**	**MIN(a,b)**
Hard sharing	45.80	48.10	51.90	32.05	42.90	37.10	65.75	16.15	72.10	39.80	75.04	52.46	88.54	84.75
Hard sharing+	91.65	70.65	77.05	67.20	91.70	72.20	64.20	61.55	74.20	63.70	79.21	58.28	90.06	85.92
Cross-stitch	38.85	39.85	50.50	34.30	34.05	38.40	61.50	28.20	58.15	54.75	88.09	83.81	92.68	89.68
Cross-stitch+	61.55	56.65	55.70	56.60	60.20	56.75	63.30	60.10	83.80	66.40	88.19	83.45	92.37	91.08
NDDR	34.85	38.35	36.35	30.60	30.25	27.80	69.55	36.10	85.30	40.75	87.14	83.79	92.29	91.23
NDDR+	54.05	52.05	49.00	51.05	51.10	49.55	70.15	87.75	83.65	68.95	89.31	84.36	93.00	92.23

5.4.6 Performance of *MNIST-sharing* Tasks

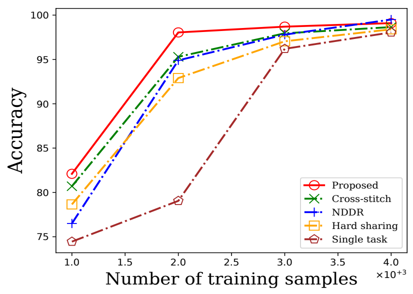
Results and analysis: We evaluate the performance of FISHNET by using different hyperparameter $|\mathcal{F}|$, which is the size of the filter pool. The results are shown in table 5.7. The best result is achieved by setting $|\mathcal{F}| = 2$ for the first two convolutional layers and $|\mathcal{F}| = 4$ for the rest two convolutional layers, and we denote it as FISHNET (best). It outperforms the naive hard sharing, NDDR, and cross-stitch by 5%-19%, 9%-40%, 3%-7% and 4%-11% on the four tasks respectively. The FISHNET ($|\mathcal{F}| = 1$) outperforms the naive hard sharing by 18%, 40%, 6% and 10% on each task, indicating the importance of not sharing BN layers. We also notice the FISHNET ($|\mathcal{F}| = 1$)'s performances are close to the FISHNET (best). The reason may be all the tasks are based on the ten digits serving as the low level features. So sharing all convolutional layers is very close to the perfect policy in this context. However, the FISHNET (best) and FISHNET ($|\mathcal{F}| = 2$) are both slightly better than FISHNET ($|\mathcal{F}| = 1$), showing the FISHNET can learn effective policies for sharing parameters.

5.4.7 Impact of Batch Normalization on MTL

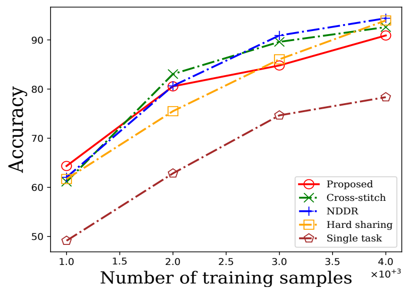
As we have shown in the results of group 3 and 4 on the CLEVR-sharing dataset, the performances of naive hard sharing, cross-stitch, and NDDR are terribly declined and can even be lower than the single task networks. The reason is the images of each task in group 3 and 4 are generated from different distribution, and in the default setting of the



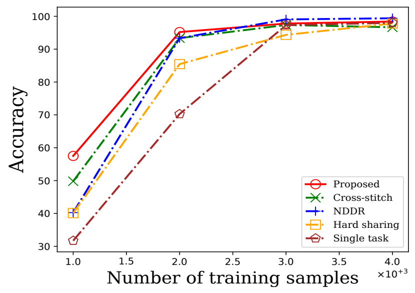
(a) *Results on T1*



(b) *Results on T2*



(c) *Results on T3*



(d) *Results on T4*

Figure 5.6: ***The performances under different number of training samples on T1, T2, T3, T4 in the task group 1 of CLEVR-sharing.***

above methods, the batch normalization operations are shared across tasks. Be attention that although the parameters in BN layers are not shared in the soft sharing methods, the input from a single tasks must run through all the task-specific networks, so the non-parametric normalization is shared by all tasks. However, the BN uses the statistics of the mini-batch for normalization during training, while in the testing uses the statistics tracked in the whole training phase. When the mini-batches from different tasks have very different statistics, the outputs of the BN layer for the same input in the training and testing can be very different! In this case, the lower training loss does not bring lower testing error. The similar finding is also been discovered by Bronskill et al [82]. To clearly demonstrate this problem, we also recorded the testing performances on these baselines in the training mode of BN, and compared them with their original results in testing mode in the table 5.8. On the group 3 of Clevr-sharing tasks, the MTL baselines in the training mode achieve about 20% up to 40% improvements compared with themselves in testing mode. In group 4, the gaps are from 10% up to 50%. We also compared the naive hard sharing with FISHNET ($|\mathcal{F}| = 1$) in table 5.7. It shows the latter using non-shared BN layers can boost the accuracies greatly.

5.4.8 Impact of Training Data Sizes

In this section, we show more details about the comparison of FISHNET and the baselines under the different size of training set. Here we use the tasks in group 1 as an example.

As shown in the Fig 5.6, FISHNET can always achieve comparable accuracies as the soft sharing methods. Especially when the size of the training set is small, e.g. 1000 or 2000, FISHNET may even outperform all baselines. The gap between the naive hard sharing method and FISHNET (and soft sharing methods) is clear when the training samples are less than or equal to 2000. All the MTL methods have clearly better performances than the single tasks networks. But as more training samples are available, all the methods achieve higher and higher accuracies and may converge to the same degree (see the results on T2 and T4). These results may indicate our FISHNET can achieve better performance than the state-of-the-art soft sharing methods when the training set is small, and also keep its computational and memory cost as small as the naive hard sharing and single task networks.

5.5 Related Work

Hard Sharing for Multi-task Learning. Hard sharing is the most common sharing method used in multi-task learning. The very basic example is to sharing all convolutional layers across the task-specific networks but using different fully-connected layers at the end. This method is wildly used as a baseline and lower bound of computational and space cost in recent researches on multi-task learning [1, 2, 74, 76, 81]. Finding an effective parameter sharing structure usually requires handcraft design by human experts. In recent years, some studies tried to search the parameter automatically. Lu et al. [83] proposed a bottom-up searching algorithm to find the sharing structure adapting to a dataset. The basic idea is to start from train a naive hard sharing network, then split the deepest layer into groups and retrain the network, then repeat this process until reaching the input layer. Guo et al. [84] proposed to learn the sharing structure directly from a dataset, which is similar to ours. They formulated the problem in the form of network architecture searching and defined a trainable searching space. Both Lu et al. and Guo et al.'s methods yield tree-shape sharing structures. Bragman et al. [85] proposed a filter grouping method that allocates each convolutional filter to one of the task-specific groups or a task-shared group and can learn a graph-shape sharing structure. This study is similar to our FISHNET, however it mainly focuses on two-tasks scenarios. Recently Sun et al. [74] proposed a novel method named Adashare which can learn a very flexible sharing policy. This work is mainly motivated by the dynamic inference methods [69, 71] which can learn to "jump over" the convolutional layers to reduce the time complexity. In the context of Adashare, the different tasks can learn different "jumping" policies on a shared network, resulting in a graph-shape sharing structure. Strezoski et al. [80] proposed a task routing method applying randomly generated binary masks for different tasks, on the feature maps learned by a single model, which is very similar to our trivial version FISHNET (random). Compared with our FISHNET, the solution space of Adashare and Task routing are restricted within a single CNN, while FISHNET's solution space is tunable as it is controlled by the pool size $|\mathcal{F}_{ij}|$ which is a hyperparameter.

Soft Sharing for Multi-task Learning. Soft sharing is different from hard sharing in that it doesn't share parameters but the learned feature maps across different tasks. Misra et al. [1] proposed the Cross-stitch network, which consists of two parts: the task-specific networks and the cross-stitch units. At certain layers, the cross-stitch units for a given task weighted add the feature maps from all tasks, outputting a mixed feature map and fit it into the next layer of the network of this task. Gao et al. [2] proposed a novel soft sharing method named NDDR-CNN. Different from Cross-stitch, NDDR leverage the 1×1 convolution and batch normalization for feature fusion. In this way, NDDR can fuse the feature maps with different channels, so it enables multi-task learning to be applied on networks with different architectures. Based on NDDR, Gao further et al. proposed a method named MTL-NAS [76]. As shown in its name, this method leveraging the NAS technique to explore the sharing links between feature maps at any possible intermediate layers. Recently, Liu et al. [81] proposed an attention-based method named MTAN aiming to reduce the complexity of soft sharing. MTAN uses a single main network shared by all tasks, which is similar to the hard sharing, but also has a specific attention module for each task, outputting task-specific soft attention masks. Then the masks are applied on the feature maps outputted by the main network, yielding the task-specific representations. Since the attention modules have very small architectures compared with the shared main network, the overall complexity of MTAN can be much lower than the previous soft sharing methods.

Dynamic Inference on Neural Networks. Figurnov et al. proposed a spatial adaptive inference architecture called SACT [68] that can skip convolution within a residual block. Specifically, SACT calculates a halting score during every convolution in a residual block and decides whether to skip the next convolution in the same residual block. Veit et al. proposed a dynamic inference model called ConvNet-AIG [69] that aims to only execute the layers related to the category of the input image. Concretely, ConvNet-AIG used a small network as a gated function to decide whether to execute a residual block or just jump over it through the shortcut link. Similarly, Bengio et al. [70] proposed a method to drop out some units of a layer in a neural network. Wang et al. proposed Skip-

Net [71] that leverages reinforcement learning to identify the suitable shallow networks per sample. Dynamic inference is also used in applications based on RNN. Minh et al. proposed a recurrent attention model (RAM) [29] on visual learning tasks. RAM can learn to only attend to the important regions without scanning the entire image, similar to SACT [68]. On the task of time series classification, Hartvigsen et al. [72] proposed a novel model EARLIEST to jointly minimize the classification error and the execution time of the model. Both RAM and EARLIEST and the works mentioned above [70, 71] are trained by REINFORCE algorithm.

6

Conclusion and future work

6.1 Conclusion

In this dissertation, I study four problems.

In chapter 2, we identified the need to design collaboration-aware deep neural networks for efficient mobile inference. We proposed CINET, deep neural networks for image classification, that leverages key insights of avoiding sending non-essential data to cloud servers to *simultaneously* reduce on-device computational cost, lower mobile network data transmission cost, and maintain high inference accuracy. Our evaluations of CINET on three datasets demonstrated that CINET reduced mobile computation by up to three orders of magnitude, lowered mobile data transmission by 99%, and had small inference accuracy differences of 0.34%-2.46%, compared to four inference approaches including two collaborative inference approaches.

In chapter 3, we first formulated the Guided Multi-Attention Classification problem. We then proposed the use of a guided attention recurrent network (GARN) to solve the problem. Our proposed method addresses the challenges of training with only a small number of samples by effectively leveraging the guidance information in the form of ROI locations. Specifically, GARN learns to identify the locations of ROIs and to perform classifications using two separate RNNs. We performed extensive evaluations on three multi-attention classification tasks. Our results across all three tasks demonstrated that GARN outperforms all baseline models. In particular, when the training set size is limited, we observed up to a 30% increase in performance.

In chapter 4, we co-designed the multi-branch networks and the early-exiting policies in the context of dynamic inference. Our proposed solution, referred to as EPNET, addressed two key challenges, namely *(i)* designing the learning objective to balance both accuracy and efficiency; and *(ii)* explicitly considering the resource overhead associated with the early-exiting policies. Concretely, we designed a lightweight branch structure and cast the early-exiting problem as a Markov decision process. This enables EPNET to make exiting decisions *per convolutional layer* through the learned policy. Comparisons of EPNET on three datasets to two types of baselines demonstrate its efficacy in classification accuracy, adaptivity to sample difficulty, and resource budgets.

In chapter 5, we studied the problem of *Resource-efficient Multi-task Learning* (MLT) with the key goal of designing a resource-friendly MLT model. Our work is primarily motivated by two aspects. First, in many real-world applications, there is an increasing demand of running MTL models on resource-constrained devices such as mobile or IoT devices. Second, existing state-of-the-art MTL models often overlook the resource requirement during inference phase. To satisfy the practical deployment requirements of low resource consumption and high accuracy (even with limited training dataset), we proposed a novel solution called FISHNET for fine-grained parameter sharing. In a nutshell, FISHNET can *learn* how to share parameters directly on training data (akin to soft-sharing) while only consuming a constant computational cost per task (similar to hard-sharing). In other words, FISHNET distills the benefits from both hard-sharing and soft-sharing approaches and enables fine-grained filter sharing among any number of task-specific CNNs of different architectures. Our extensive evaluations on two datasets including the popular CLEVR showed that FISHNET achieves comparable or better accuracy than state-of-the-art soft-sharing approaches on several multi-task problems, while only consuming a fraction of resources. Furthermore, our ablation study showed that FISHNET can effectively learn the policy and that unlike existing MTL sharing (hard and soft) approaches, FISHNET is not subject to the accuracy degradation from the widely used batch normalization technique.

6.2 Future work: Global Filter Sharing for Resource-Efficient Multi-Task Learning

Although the FISHNET can learn hard sharing policy from dataset, there are still two problems remain. First, it can only learn to share the parameters among networks with the same architecture. The second, FISHNET can't share filters at different depths. These two problems may prevent the usage of hard sharing in many scenarios in the real world. Here are two examples:

Example 1. Given a task of telling cat from other animals, and another task of predicting the breed of cat. The second task is more difficult than the first one, so we should use networks of different sizes to fit each of them respectively. Also, the two tasks are related, so sharing parameters or features between the networks for the two tasks may alleviate overfitting.

Example 2. Given two tasks that both require recognizing human faces, the human faces in the images of one task are small and are big in another task. It's not hard to see the similar facial features could be shared by the two task-specific CNNs, but they need to capture the feature at different depths.

A hard sharing method that can solve the above two problems should be able to share parameters among filters of different shapes. Here I present a possible framework for future study: Instead of assigning each filter a candidates pool, we may let all filters in all CNNs share a single pool. The candidates in the pool have the same shape. We keep an allocator and a reshaper module for every filter. The allocator picks a candidate from the pool, as we did in FiShNet. Then the reshaper transforms the shape of the picked candidate to match that of the corresponding filter, and assigns the result to the filter.

References

[1] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. Cross-stitch networks for multi-task learning. In *Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'16)*, pages 3994–4003, 2016. xi, 76, 86, 87, 97

[2] Yuan Gao, Jiayi Ma, Mingbo Zhao, Wei Liu, and Alan L Yuille. Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction. In *Proc. 2019 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'19)*, pages 3205–3214, 2019. xi, 76, 84, 86, 87, 97, 98

[3] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997. xi, 76

[4] Jonathan Baxter. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning*, 28(1):7–39, 1997. xi, 76

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25 (NeurIPS'12)*, pages 1097–1105, 2012. 1, 3, 8, 11, 14, 48

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'16)*, pages 770–778, 2016. 1, 3, 8, 14, 53, 60, 61

[7] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger.

Densely connected convolutional networks. In *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 4700–4708, 2017. 1, 53

[8] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. 2014 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'14)*, pages 580–587, 2014. 1, 9, 10, 48

[9] Ross Girshick. Fast R-CNN. In *Proc. 2015 IEEE Int. Conf. Computer Vision (ICCV'15)*, pages 1440–1448, 2015. 1, 9

[10] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems 28 (NeurIPS'15)*, pages 91–99, 2015. 1, 9

[11] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 1

[12] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017. 1

[13] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proc. 2015 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'15)*, pages 3431–3440, 2015. 1

[14] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and con-

nections for efficient neural network. In *Advances in Neural Information Processing Systems 28 (NeurIPS'15)*, pages 1135–1143, 2015. 2

[15] Vadim Lebedev and Victor Lempitsky. Fast convnets using group-wise brain damage. In *Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'16)*, pages 2554–2564, 2016. 2

[16] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning separable filters. In *Proc. 2013 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'13)*, pages 2754–2761, 2013. 2

[17] Cheng Tai, Tong Xiao, Yi Zhang, Xiaogang Wang, et al. Convolutional neural networks with low-rank regularization. In *Proc. 4th Int. Conf. Learning Representations (ICLR'16)*, 2016. 2

[18] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. In *Advances in Neural Information Processing Systems 28 (NeurIPS'15)Workshop*, 2015. 2

[19] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. In *Proc. 4th Int. Conf. Learning Representations (ICLR'16)*, 2016. 2

[20] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size. *arXiv preprint arXiv:1602.07360*, 2016. 2, 3, 8

[21] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proc. 2018*

IEEE Conf. Computer Vision and Pattern Recognition (CVPR'18), pages 6848–6856, 2018. 2, 3, 8

[22] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992. 3, 15, 40, 57, 84

[23] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. In *Proc. 5th Int. Conf. Learning Representations (ICLR'17)*, 2017. 3, 82

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. 3rd Int. Conf. Learning Representations (ICLR'15)*, 2015. 3, 8

[25] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proc. 2015 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'15)*, pages 1–9, 2015. 3, 8, 14

[26] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 3, 8

[27] Yiping Kang, Johann Hauswald, Cao Gao, Austin Rovinski, Trevor Mudge, Jason Mars, and Lingjia Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *ACM SIGARCH Computer Architecture News*, 45(1):615–629, 2017. 3, 8, 9, 10, 20, 28

[28] Guangli Li, Lei Liu, Xueying Wang, Xiao Dong, Peng Zhao, and Xiaobing Feng. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In *International Conference on Artificial Neural Networks*, pages 402–411. Springer, 2018. 3, 8, 10, 20, 28

[29] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems 27 (NeurIPS'14)*, pages 2204–2212, 2014. 4, 12, 15, 28, 30, 34, 36, 39, 40, 41, 49, 73, 99

[30] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. Multiple object recognition with visual attention. In *Proc. 3rd Int. Conf. Learning Representations (ICLR'15)*, 2016. 4, 12, 40, 49

[31] Surat Teerapittayanon, Bradley McDanel, and Hsiang-Tsung Kung. Branchynet: Fast inference via early exiting from deep neural networks. In *International Conference on Pattern Recognition (ICPR)*, pages 2464–2469. IEEE, 2016. 4, 5, 50, 55, 57, 58, 61, 64, 72

[32] Gao Huang, Danlu Chen, Tianhong Li, Felix Wu, Laurens van der Maaten, and Kilian Weinberger. Multi-scale dense networks for resource efficient image classification. In *Proc. 6th Int. Conf. Learning Representations (ICLR'18)*, 2018. 4, 50, 55, 58, 61, 62, 72

[33] Hao Li, Hong Zhang, Xiaojuan Qi, Ruigang Yang, and Gao Huang. Improved techniques for training adaptive deep networks. In *Proc. 2019 IEEE Int. Conf. on Computer Vision (ICCV'19)*, pages 1891–1900, 2019. 4, 50, 55, 58, 61, 62, 72

[34] Yu Zhang and Qiang Yang. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*, 2017. 5

[35] Zihao Liu, Tao Liu, Wujie Wen, Lei Jiang, Jie Xu, Yanzhi Wang, and Gang Quan. Deepn-jpeg: a deep neural network favorable jpeg-based image compression framework. In *Proceedings of the 55th Annual Design Automation Conference*, pages 1–6, 2018. 9, 10, 18, 19, 23, 28

[36] Xiufeng Xie and Kyu-Han Kim. Source compression with bounded dnn perception loss for iot edge computer vision. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019. 9, 10, 28

[37] Amjad Almahairi, Nicolas Ballas, Tim Cooijmans, Yin Zheng, Hugo Larochelle, and Aaron Courville. Dynamic capacity networks. In *International Conference on Machine Learning*, pages 2549–2558, 2016. 10, 12

[38] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems 28 (NeurIPS'15)*, pages 2017–2025, 2015. 14, 15, 28

[39] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An execution framework for deep neural networks on resource-constrained devices. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 2016. 27

[40] Johannes Ballé, Valero Laparra, and Eero P Simoncelli. End-to-end optimized image compression. In *Proc. 5th Int. Conf. Learning Representations (ICLR'17)*, 2017. 28

[41] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. In *Proc. 5th Int. Conf. Learning Representations (ICLR'17)*, 2017. 28

[42] Jingyuan Zhang, Bokai Cao, Sihong Xie, Chun-Ta Lu, Philip S. Yu, and Ann B. Ragin. Identifying connectivity patterns for brain diseases via multi-side-view guided

deep architectures. In *Proc. 16th SIAM Int. Conf. Data Mining (SDM'16)*, pages 36–44, 2016. 35

[43] Ed Bullmore and Olaf Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature reviews neuroscience*, 10(3):186–198, 2009. 45

[44] Nathalie Tzourio-Mazoyer, Brigitte Landeau, Dimitri Papathanassiou, Fabrice Crivello, Olivier Etard, Nicolas Delcroix, Bernard Mazoyer, and Marc Joliot. Automated anatomical labeling of activations in SPM using a macroscopic anatomical parcellation of the MNI MRI single-subject brain. *Neuroimage*, 15(1):273–289, 2002. 45

[45] Ian J Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, and Vinay Shet. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In *Proc. 2nd Int. Conf. Learning Representations (ICLR'14)*, 2014. 48

[46] Bogdan Alexe, Thomas Deselaers, and Vittorio Ferrari. What is an object? In *Proc. 2010 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'10)*, pages 73–80, 2010. 48

[47] Bogdan Alexe, Nicolas Heess, Yee W Teh, and Vittorio Ferrari. Searching for objects driven by context. In *Advances in Neural Information Processing Systems 25 (NeurIPS'12)*, pages 881–889, 2012. 48

[48] Nicholas J Butko and Javier R Movellan. Optimal scanning for faster object detection. In *Proc. 2009 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'09)*, pages 2751–2758, 2009. 48

[49] Misha Denil, Loris Bazzani, Hugo Larochelle, and Nando de Freitas. Learning where to attend with deep architectures for image tracking. *Neural Computation*, 24(8):2151–2184, 2012. 48, 49

[50] Yudong Zhang, Zhengchao Dong, Preetha Phillips, Shuihua Wang, Genlin Ji, Jiquan Yang, and Ti-Fei Yuan. Detection of subjects and brain regions related to Alzheimer's disease using 3D MRI scans based on eigenbrain and machine learning. *Frontiers in Computational Neuroscience*, 9:66, 2015. 48

[51] Arthur Mensch, Gaël Varoquaux, and Bertrand Thirion. Compressed online dictionary learning for fast resting-state fmri decomposition. In *Proc. 13th IEEE Int. Symposium on Biomedical Imaging (ISBI'16)*, pages 1282–1285, 2016. 48

[52] Luping Zhou, Lei Wang, Lingqiao Liu, Philip Ogunbona, and Dinggang Shen. Discriminative brain effective connectivity analysis for Alzheimer's disease: a kernel learning approach upon sparse Gaussian Bayesian network. In *Proc. 2013 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'13)*, pages 2243–2250, 2013. 48

[53] Tom Brosch, Roger Tam, Alzheimers Disease Neuroimaging Initiative, et al. Manifold learning of brain MRIs by deep learning. In *Proc. 16th Int. Conf. Medical Image Computing and Computer-Assisted Intervention (MICCAI'13)*, pages 633–640, 2013. 48

[54] Dong Nie, Han Zhang, Ehsan Adeli, Luyan Liu, and Dinggang Shen. 3D deep learning for multi-modal imaging-guided survival time prediction of brain tumor patients. In *Proc. 19th Int. Conf. Medical Image Computing and Computer-Assisted Intervention (MICCAI'16)*, pages 212–220, 2016. 49

[55] John Boaz Lee, Xiangnan Kong, Yihan Bao, and Constance Moore. Identifying deep

contrasting networks from time series data: Application to brain network analysis. In *Proc. 17th SIAM Int. Conf. Data Mining (SDM'17)*, pages 543–551, 2017. 49

[56] Hugo Larochelle and Geoffrey E Hinton. Learning to combine foveal glimpses with a third-order Boltzmann machine. In *Advances in Neural Information Processing Systems 23 (NeurIPS'10)*, pages 1243–1251, 2010. 49

[57] Charlie Tang, Nitish Srivastava, and Russ R Salakhutdinov. Learning generative models with visual attention. In *Advances in Neural Information Processing Systems 27 (NeurIPS'14)*, pages 1808–1816, 2014. 49

[58] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *Proc. 32nd Int. Conf. Machine Learning (ICML'15)*, pages 2048–2057, 2015. 49

[59] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *Proc. 3rd Int. Conf. Learning Representations (ICLR'15)*, 2015. 49

[60] Albert Haque, Alexandre Alahi, and Li Fei-Fei. Recurrent attention models for depth-based person identification. In *Proc. 2016 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'16)*, pages 1229–1238, 2016. 49

[61] Diangarti Bhalang Tarianga, Prithviraj Senguptab, Aniket Roy, Rajat Subhra Chakraborty, and Ruchira Naskar. Classification of computer generated and natural images based on efficient deep convolutional recurrent attention model. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019. 49

[62] Xin Zhao, Liufang Sang, Guiguang Ding, Jungong Han, Na Di, and Chenggang Yan. Recurrent attention model for pedestrian attribute recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9275–9282, 2019. 49

[63] Lei Zhu, Zijun Deng, Xiaowei Hu, Chi-Wing Fu, Xuemiao Xu, Jing Qin, and Pheng-Ann Heng. Bidirectional feature pyramid network with recurrent attention residual modules for shadow detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 121–136, 2018. 49

[64] Zhen Zhou, Yan Huang, Wei Wang, Liang Wang, and Tieniu Tan. See the forest for the trees: Joint spatial and temporal recurrent neural networks for video-based person re-identification. In *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 6776–6785, 2017. 49

[65] Jun Liu, Gang Wang, Ping Hu, Ling-Yu Duan, and Alex C Kot. Global context-aware attention LSTM networks for 3D action recognition. In *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'17)*, 2017. 49

[66] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proc. 2015 IEEE Int. Conf. on Computer Vision (ICCV'15)*, pages 1026–1034, 2015. 63, 65, 89

[67] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proc. 32nd Int. Conf. Machine Learning (ICML'15)*, pages 448–456, 2015. 63, 65, 89

[68] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for

residual networks. In *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 1039–1048, 2017. 72, 73, 98, 99

[69] Andreas Veit and Serge Belongie. Convolutional networks with adaptive inference graphs. In *Proc. 2018 the European Conf. on Computer Vision (ECCV)*, pages 3–18, 2018. 72, 97, 98

[70] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. In *Proc. 4th Int. Conf. Learning Representations (ICLR'16) Workshop*, 2015. 72, 73, 98, 99

[71] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proc. 2018 the European Conf. on Computer Vision (ECCV)*, pages 409–424, 2018. 72, 73, 97, 99

[72] Thomas Hartvigsen, Cansu Sen, Xiangnan Kong, and Elke Rundensteiner. Adaptive-halting policy network for early classification. In *Proc. 25th ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD'19)*, pages 101–110, 2019. 73, 99

[73] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *Proc. 7th Int. Conf. Learning Representations (ICLR'19)*, 2019. 83

[74] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *arXiv preprint arXiv:1911.12423*, 2019. 84, 87, 97

[75] Trevor Standley, Amir Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning? In *Proc. 37th Int. Conf. Machine Learning (ICML'20)*, pages 9120–9132. PMLR, 2020. 84

[76] Yuan Gao, Haoping Bai, Zequn Jie, Jiayi Ma, Kui Jia, and Wei Liu. Mtl-nas: Task-agnostic neural architecture search towards general-purpose multi-task learning. In *Proc. 2020 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'20)*, pages 11543–11552, 2020. 84, 87, 97, 98

[77] Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 2901–2910, 2017. 85

[78] Hila Levi and Shimon Ullman. Multi-task learning by a top-down control network. *arXiv preprint arXiv:2002.03335*, 2020. 85

[79] Guy Davidson and Michael C Mozer. Sequential mastery of multiple visual tasks: Networks naturally learn to learn and forget to forget. In *Proc. 2020 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'20)*, pages 9282–9293, 2020. 85

[80] Gjorgji Strezoski, Nanne van Noord, and Marcel Worring. Many task learning with task routing. In *Proc. 2019 IEEE Int. Conf. on Computer Vision (ICCV'19)*, pages 1375–1384, 2019. 86, 97

[81] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proc. 2019 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'19)*, pages 1871–1880, 2019. 87, 97, 98

[82] John Bronskill, Jonathan Gordon, James Requeima, Sebastian Nowozin, and Richard Turner. Tasknorm: Rethinking batch normalization for meta-learning. In *Proc. 37th Int. Conf. Machine Learning (ICML'20)*, pages 1153–1164. PMLR, 2020. 90, 95

[83] Yongxi Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, Tara Javidi, and Rogerio Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. In *Proc. 2017 IEEE Conf. Computer Vision and Pattern Recognition (CVPR'17)*, pages 5334–5343, 2017. 97

[84] Pengsheng Guo, Chen-Yu Lee, and Daniel Ulbricht. Learning to branch for multi-task learning. In *Proc. 37th Int. Conf. Machine Learning (ICML'20)*, pages 3854–3863. PMLR, 2020. 97

[85] Felix J.S. Bragman, Ryutaro Tanno, Sebastien Ourselin, Daniel C. Alexander, and Jorge Cardoso. Stochastic filter groups for multi-task cnns: Learning specialist and generalist convolution kernels. In *Proc. 2019 IEEE Int. Conf. on Computer Vision (ICCV'19)*, October 2019. 97