CiNet: Redesigning Deep Neural Networks for Efficient Mobile-Cloud Collaborative Inference

Xin Dai* Xiangnan Kong* Tian Guo* Yixian Huang*

Abstract

Deep neural networks are increasingly used in end devices such as mobile phones to support novel features, e.g., image classification. Traditional paradigms to support mobile deep inference fall 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. *Collaborative inference*, a means to splitting inference computation between mobile devices and cloud servers, was proposed to address the limitations of traditional inference through techniques such as image compression or model partition.

In this paper, we improve the performance of collaborative inference from a complementary direction, i.e., through redesigning deep neural networks to satisfy the collaboration requirement from the outset. Specifically, we describe the design of a collaboration-aware convolutional neural network, referred to as CiNet, for image classification. CiNet consists of a mobile-side extractor submodel that outputs a small yet relevant patch of the image and a cloud-based submodel that classifies on the image patch.

We evaluated the efficiency of CiNet in terms of inference accuracy, computational cost and mobile data transmission on three datasets. Our results demonstrate that CiNet achieved comparable inference accuracy while incurring orders of magnitude less computational cost and 99% less transmitted data, when comparing to both traditional and collaborative inference approaches.

1 Introduction

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 1. Cloud-based inference allows the use of complex models [7, 12, 17, 18] (thus higher inference accuracy), but requires mobile applications to send non-trivial amount of data over mobile networks, leading

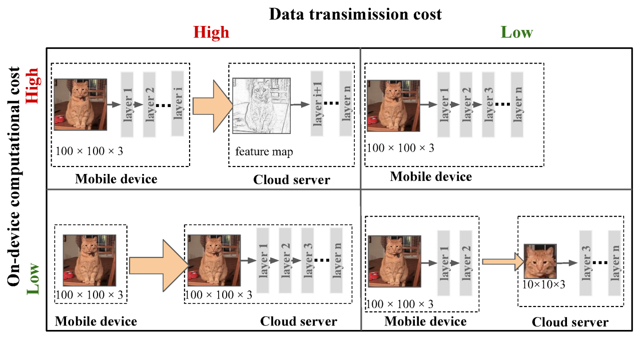


Figure 1: ***The problem of mobile-cloud collaborative inference.*** *The goal is to perform image classification on a mobile device by collaborating with a cloud server. The mobile device can send some data to the server to aid in the inference process, which will reduce the computational cost on the mobile device but increase the data transmission cost.*

to high data transmission. To use mobile-based inference, one needs to use mobile-specific models such as MobileNet, SqueezeNet, or ShuffleNet [8, 9, 22]; 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 [11, 13]. Collaborative inference allows inference execution to be split between mobile devices and cloud servers as demonstrated in Figure 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 [11, 14, 21]. 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 ef-

*Worcester Polytechnic Institute

fectively 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 [4, 5, 16] is a potential candidate for detecting image regions to send to the cloud, but often requires access to annotated locations during training [5].

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 [1]. We make the following main contributions.

- We identify the need and the key principles to redesign deep neural networks for achieving efficient inference under the collaborative inference paradigm. For example, the performance of existing collaborative inference approaches are constrained by the deep learning models and often can not *simultaneously* achieve equally important performance goals such as low on-device computation, low mobile data transmission, and high inference accuracy.

- We describe the design of a collaboration-aware model for image classification called CiNet that works within the limitations of mobile devices and achieves comparable inference accuracy to complex cloud-based models. On the mobile side, CiNet consists of an extractor submodel that generates a predefined data grid from the content-locations of original image. The values on the grid are resampled at the corresponding locations on the original image and are used as the input to the cloud-based classifier. In short, CiNet efficiently splits computation across mobile devices and cloud servers with low transmission cost, and can be trained in an end-to-end fashion using the standard backpropagation with only the image labels.
- We evaluated CiNet on three datasets and compared against four inference mechanisms including cloud-based, mobile-based, and existing collaborative inference. Our results show that CiNet reduced mobile computational cost by up to three orders of magnitude, lowered mobile data transmission by 99%, and achieved similar inference accuracy with 0.34%-2.46% differences.

Although we only focus on the images having only one ROI in this paper, the experiment shows the ROI can be detected with low computational cost compared with classification itself. It implies the idea proposed in this paper can be extended to more complex images having multiple ROIs.

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 paper, 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. We approach the problem of efficient collaborative inference by redesigning deep neural networks that are collaboration-aware, unlike existing works in collaborative inference [11, 13, 14, 21]. 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 [12], 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

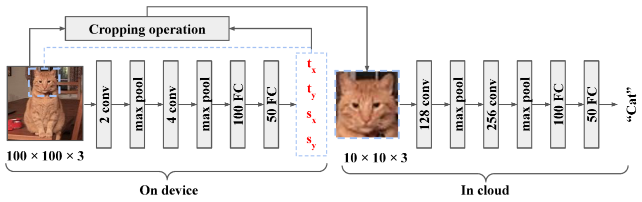


Figure 2: ***An example of*** **CiNet** ***structure for mobile-cloud collaborative inference.***

goals without sacrificing accuracy as it is important to application utility.

3 CiNet: A Collaboration-aware Deep Neural Network

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, the extractor submodel runs on the mobile device and is responsible to extract a smaller-size 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 [2, 15]. 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 [1]. 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.

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:

$$\begin{bmatrix} x_I \\ y_I \end{bmatrix} = \begin{bmatrix} s_x, 0, t_x \\ 0, s_y, t_y \end{bmatrix} \begin{bmatrix} x_P \\ y_P \\ 1 \end{bmatrix}. \tag{3.1}$$

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 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.

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 [10] 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:

$$P(x_P, y_P) = \sum_{j=1}^{W} \sum_{i=1}^{H} I(i, j) F(x_I, i) F(y_I, j), \tag{3.2}$$

$$F(a, b) = \max(0, 1 - |a - b|). \tag{3.3}$$

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 (3.1). 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 3.5.

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 [12], GoogleNet [18] or ResNet [7] 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 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 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.

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 [10]. 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 [20] to train the extractor, similar to what was proposed by Mnih et al. [15]. 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 hyperparameter techniques to mitigate such problems.

Table 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
CNN1	(128, 256, 256)	3
CNN2	(64, 128, 256)	3
CNN3	(32, 64, 128)	3
CNN4	(8, 64, 128)	3
CNN5	(8, 16, 64)	3
CNN6	(4, 8, 32)	3
CNN7	(4, 8, 16)	3

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

- *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.

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

Table 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

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 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 [14], CiNet achieved up to 50% higher inference accuracy with similar data transmission cost. More details can be found in Section 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 4.4.

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 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 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, 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 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 [14].* 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 [14].
- *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 [11, 13]. In our evaluations, we tested all possible partition points, i.e., layers, for the most accurate CNNs from Table 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

Table 3: ***Results on transformed MNIST dataset.*** *For in-cloud deployment, the model used is the CNN1 in Table 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**

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

Table 4: ***Results on transformed fashion MNIST dataset.*** *For in-cloud deployment, the model used is the CNN1 in Table 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**

Table 5: ***Results on CelebA dataset.*** *For in-cloud deployment, the model used is the ResNet1 in Table 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**

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 1 and Table 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 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* 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 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 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, with negligible accuracy loss.

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

Table 3 compares the performance of *CNN1* from Table 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 [14]. 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 3 shows that using the *CNN1* trained with higher JPEG quality only incurred 10% data transmission cost with 0.41% lower 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 2.2% better 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 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 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.

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 6 shows the performance of *CNN1* from Table 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.

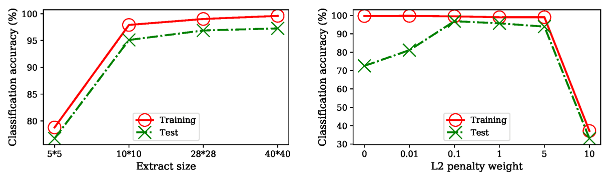
4.5 Impact of Hyperparameters Lastly, we evaluate the impact of two important hyperparameters on CiNet's inference accuracy.

- **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 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 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 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.

Table 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 1, at all possible layer-wise partition points. The computational (Comp) and data transmission (trans) cost are shown.*

Partition Point	Comp Cost	Trans Cost
conv1	32	320000
conv2	2040	160000
conv3	3060	43200
fc1	3060	100
fc2	3060	50
-	-	-
-	-	-
CiNet	1	100

Partition Point	Comp Cost	Trans Cost
conv1	16	160000
conv2	528	80000
conv3	784	20000
conv4	922	12500
conv5	1000	4096
fc1	1000	100
fc2	1000	50
CiNet	1	100



(a) *Extraction size* **(b)** *L2 penalty weight.*

Figure 3: *Impact of HyperParameters.*

5 Related Work

Collaborative Inference with Model Partition. Han et al. [6] proposed to generate a resource-efficient variant for a given network, then provide a run-time system called MCDNN which split the generated network into two fragments and execute them each on mobile device and cloud server. Kang et al. [11] developed Neurosurgeon, an automatic model partition scheme that can adapt to different hardware environments and model structures. Li et al. [13] 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 [3] or autoencoder [19] 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 [21]. Similarly, Liu et al. proposed a dataset aware compression algorithm that adjusts the quantization table with the statistical information of training dataset [14]. 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 [15] 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 [10]. 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.

6 Conclusion

In this work, 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.

7 Acknowledgement

This work was supported in part by NSF Grant CNS-1815619.

References

[1] A. Almahairi, N. Ballas, T. Cooijmans, Y. Zheng, H. Larochelle, and A. Courville. Dynamic capacity networks. In *ICML*, 2016.

[2] J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. In *ICLR*, 2016.

[3] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. In *ICLR*, 2017.

[4] R. Girshick. Fast R-CNN. In *ICCV*, 2015.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014.

[6] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. Mcdnn: An execution framework for deep neural networks on resource-constrained devices. In *MobiSys*, 2016.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[8] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.

[9] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5 MB model size. *arXiv:1602.07360*, 2016.

[10] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *NeurIPS*, 2015.

[11] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang. Neurosurgeon: Collaborative intelligence between the cloud and mobile edge. *In ASPLOS*, 2017.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[13] G. Li, L. Liu, X. Wang, X. Dong, P. Zhao, and X. Feng. Auto-tuning neural network quantization framework for collaborative inference between the cloud and edge. In *ICANN*, 2018.

[14] Z. Liu, T. Liu, W. Wen, L. Jiang, J. Xu, Y. Wang, and G. Quan. Deepn-jpeg: a deep neural network favorable jpeg-based image compression framework. In *DAC*, 2018.

[15] V. Mnih, N. Heess, A. Graves, et al. Recurrent models of visual attention. In *NeurIPS*, 2014.

[16] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015.

[17] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015.

[19] L. Theis, W. Shi, A. Cunningham, and F. Huszár. Lossy image compression with compressive autoencoders. In *ICLR*, 2017.

[20] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

[21] X. Xie and K.-H. Kim. Source compression with bounded dnn perception loss for iot edge computer vision. In *MobiCom*, 2019.

[22] X. Zhang, X. Zhou, M. Lin, and J. Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*, 2018.