ProbabilityCNN: Environment Adaptive Model Without Finetuning

Anonymous Author(s)

Abstract

We can meet thousands of objects through years, i.e., different kind of animals, plants, and various people, which requires a general model to classify all these objects. On the other hand, in a specific environment, like our home or office, only less than ten objects appear, such as family members, pets, and furniture. Thus a special model recognizing a few objects could be sufficient. Using a special model in this environment could lead automatically to a much higher accuracy since there is a reduction in the number of classes. However, using special models has an intrinsic problem that there are so many environments and mind-bogglingly many combinations of objects, which makes it infeasible to train special models one-by-one. In the contrast, the general model could be trained once and used everywhere. The general model also has another benefit that it has seen much more images than a special model and a special model derived from the general model could have much higher accuracy than the one trained from scratch. The problem is, given various combination of classes and distribution, could we train a general model and *derive specialized models from it efficiently for all combinations?* The intuitive idea would be transfer learning, in which the general model is fine-tuned to adapt to the new dataset. However, considering the time latency and energy consumption of finetuning a model, we argue that avoiding finetuning is critical in the efficient derivation of specialized models. Instead, we brought up *probability layer*, which could adapt the general model to all combinations of classes and distribution efficiently without any retraining. Through experiments, it was shown that probability layer performs equivalent or even better than transfer learning and complements other acceleration methods to achieve an increase in accuracy and a decrease in computation.

1 Introduction

Convolutional neural network (CNN) plays an important role in image recognition. In 2012, AlexNet [12] achieved a top-5 error of 17% on ImageNet [3], while previous method could only achieve a top-5 error of 25.7%. Since then, CNNs have become the dominant method and main research direction in image recognition. In 2015, ResNet [9] achieves a top-5 error of 3.57%. Considering that the estimated human classification error on ImageNet is 5.1% [15], we can conclude that CNNs have gained the ability to perform better than human.

However, all of these results are gained in the lab setting, which overlooks the environment information that we can use in real life. According to our experience, the appearance of classes has strong temporal and spatial locality. Even if we can meet thousands of objects through years, only less than ten objects would appear in a particular environment, like our home or office. Experiments on videos of day-to-day life from Youtube [17] shows that 10 objects comprised 90% of all objects in 85% time. While we can train a general model classifying thousands of classes in the lab, a special model for tens of objects would be sufficient. This reduction in the number of classes could lead to a boost in accuracy automatically. For example, if we randomly guess from 1000 classes, the accuracy is 0.1%, while the accuracy for randomly guessing from 10 classes is 10%. In other words, even if we have changed nothing in the model architecture, the reduction in the number of classes could naturally provide us a higher accuracy. Thus it is desirable to incorporate the environment information and use a specialized model instead of a general one. The obstacle to this fascinating plan is the mind-bogglingly huge number of specialized scenarios. Considering the combinations of classes, if we take 10 classes out of 100 classes, there would be $1.73 * 10^{14}$ combinations. If the class distribution is also taken into consideration, this number could increase further. In order to use specialized models, we must know how to get them efficiently on various combinations of classes and different class distributions.

Transfer learning is the dominant method of conquering various problems for implementing CNNs in the real world. In transfer learning, we will have a source domain and a target domain. A model will be pretrained on the source domain and the last few layers will be retrained on the target domain such that the model can handle testing data from the target domain [4, 7, 14, 17, 23]. In this process, various problems could be solved, such as lacking training data, using unlabeled data [4, 13], and allowing quick runtime domain adaption [7, 17]. Transfer learning has also shown improvement in accuracy compared with training a model from scratch [14, 23] since the model has seen more images in the process of transfer learning. Due to the effectiveness in all these domains, transfer learning has become the method off the top of the head. To the best of our knowledge, transfer learning is the only method used to utilize the scenario we described.

Although transfer learning can bring in many benefits, it also requires expertise in training models and has intrinsic obstacles in automatic retraining. First, the general practice in transfer learning is to freeze the parameters in leading layers and retrain the last few layers, while existing papers [23] show that freezing leading layers may have a strong negative effect on the accuracy. Without testing various settings, it is hard to know how many layers should be frozen. Second, the choice of training step depends on the dataset and a unique setting is hard to find, which makes automatical transfer learning very hard, if not impossible. For example, in the training process of VggNet [18], a shallow model needs to be trained to initialize the weights in a deeper model and the learning rate is also carefully selected for several times. There is also a long-lasting debate for the best choice of hyper-parameters for various architectures and datasets [2, 6, 8, 19, 22]. All of these evidence shows that finetuning a model is not a straightforward task and expertise in the training process is required, which obstacles the automatical finetuning. Third, due to the requirement for short latency, the model may not converge and dramatical variance in accuracy may appear, even with the same model and dataset. Given all these intrinsic problems to implement transfer learning, a natural question would be whether we can find a method to use the scenario *without retraining*?

To use the scenario without retraining, we brought up *probability layer* which can avoid retraining totally and perform better than transfer learning. The benefit contains three folds. First, probability layer can make use of the scenario without any retraining, which would save lots of time. Second, probability layer can make the prediction based on the knowledge of original model and performs better than training from scratch. Third, through series of experiments, probability layer is shown to have equivalent or even better performance than transfer learning. Through experiments, we show that probability layer is a better method than transfer learning on making use of various kind of scenarios.

2 Related Works

Many papers have talked about the difference between testing CNN models in the lab setting and in the wild. In lab setting, the prevalent benchmarks, like ImageNet [3] and CIFAR100 [11], all assumes equal number of images for each class. However, in a more realistic setting [7, 17], different classes may have dramatically different frequency and the distribution may also change over time. The question we want to answer is that, given an environment with a number of classes and distribution, how to use this information? All of the existing works [7, 17] only use transfer learning to retrain the last few layers, which would introduce a huge amount of energy consumption and time latency. Our method will make use of the distribution efficiently without finetuning, thus avoid additional energy-consumption and time latency.

Unbalanced datasets focus on the problem that training dataset has different distribution with the testing dataset, which is related to our scenario. One key point of the unbalanced dataset is that the minor classes in the training dataset cannot gain same attention as the major classes, thus oversampling on minor classes or undersampling on major classes will be implemented to generate a balanced dataset [**?** **?** 21]. Our scenario is different since we assume that we have a model pretrained on a balanced dataset and in the testing time the class distribution changes. Another key point of the unbalanced dataset is that in the running time, as the distribution changes, the model will keep updating as new inputs come and finetuning is constantly needed. Our method avoids retrain at all and could achieve the same or better performance by only changing a few parameters according to the distribution automatically. Thus energy efficiency and speedup could be achieved by our solution.

Transfer learning has shown benefit in domain adaption and currently is the dominant method, if not the only one, for domain adaption. To solve the problem that the testing dataset is small, we use transfer learning [14]. To use unsupervised dataset [4, 13], we use transfer learning. Assume we have a large model handling 1000 classes and in an environment where only 10 classes appearing, we still use transfer learning [7, 17]. Transfer learning has become the only method off the top of the head when we consider the change in classes. Although transfer learning shows various benefit, it also has intrinsic shortage residing in the process of training a CNN model. First, it is hard to decide how many layers we should freeze. Published papers [23] reported that freezing fewer layers leads to better performance on different domains and freezing more layers lead to a better performance on similar domains since the co-adapted interactions between layers will be kept. However, it is still hard to decide whether two domains are similar enough and the exact effect of freezing a various number of layers. Second, it is hard to decide the hyper-parameters unless actually trying different settings. When choosing epoch numbers, it is hard to predict whether the model will converge or collapse after a pre-chosen number of epochs. The choice of learning rate also depends on both model and dataset. The choice of hyper-parameter needs expertise in finetuning, which is lacked by the automatical product. Third, the long latency and energy consumption of training a model obstacle the transfer learning on energy-efficient devices, especially in an environment that class number and distributions keep changing. Our method can avoid retraining at all while adapting to the new dataset, thus all the inconvenience related to retraining is avoided naturally.

3 Probability Layer

In this section, we introduce the probability layer and the PCNN. Before proceeding further, we introduce the notation that will be used in the rest of the paper.

Notation. A CNN takes as input an image X and outputs a vector of estimated probability for each class

$$\vec{p} = (p_1, p_2, ..., p_n) \tag{1}$$

, where n is the number of classes, and select the class with highest estimated probability

$$\underset{i}{\text{argmax}}\ p_i \tag{2}$$

as the true class. Here we treat the whole CNN model as a classifier which gives the estimated probability p_i for the label i given the input image X

$$p_i = P(i|X) \tag{3}$$

. After adding the probability layer, the original prediction $P(i|X)$ will be rescaled and we use $P_t(i|X)$ to denote the rescaled probability. Additionally, we define $P(i)$ and $P_t(i)$ for class distribution in training data and testing data respectively.

Key Assumption. The main difference between the proposed layer and the original CNNs is that we take into consideration the environment information. In original CNNs, the prediction for each image will be made individually, assuming a sequence of images is independent and identically distributed (*i.i.d*). However, in real life, this assumption does not hold and strong spatial and temporal locality may exist. In particular, we assume that in the environment the number of classes, class type, and class distribution is fixed while a huge difference between training dataset and testing dataset exists. This is a reasonable assumption on the environment in real life. Experiments on videos of day-to-day life from Youtube [17] shows that 10 objects comprised 90% of all objects in 85% time.

In this section, we present the *probability layer*. Probability layer is an extra layer after the CNN model, rescaling the output of softmax layer. Rescaling is a topic in statistics [16]. To the best of our knowledge, we are the first to discuss rescaling in CNN context. The outputs of original CNNs predict the probability for each class and the probability layer will adjust this prediction based on the difference of class distribution in training and testing dataset. In particular, for classes with different distributions in training and testing dataset, the probability layer will rescale the corresponding outputs from softmax layer according to the difference in distribution. For other classes with same distribution in both training and testing dataset, the outputs of the proposed layer are equal to the outputs of the softmax layer.

The probability layer will take as inputs the originally predicted probability, class distribution in training dataset, as well as the distribution in testing dataset, and output a vector for the rescaled prediction. The first input is the prediction $P(\vec{\cdot|}X)$ from the softmax layer, which represents the originally predicted probability for each class from the original CNNs. The second input is a vector of class distribution $P(\vec{\cdot})$ in training dataset and the third one is a vector of class distribution $P_t(\vec{\cdot})$ in testing dataset. The probability layer will rescale the predicted probability in $P(\vec{\cdot|}X)$ element wisely and produce as output a vector $P_t(\vec{\cdot|}X)$ for the rescaled prediction of each class.

Formally, let the outputs of CNNs with and without rescaling are

$$P_t(i|X) = \frac{P_t(X|i) \cdot P_t(i)}{P_t(x)} \tag{4}$$

and

$$P(i|X) = \frac{P(X|i) \cdot P(i)}{P(x)} \tag{5}$$

respectively. Here $P_t(i)$ means the class distribution in testing dataset and $P_t(i|X)$ represents the predicted probability for class i after the probability layer. We assume that $P_t(X|i)$ equals $P(X|i)$ approximately, where $P(X|i)$ is the distribution of image data for class i. This assumption makes sense since, for a class i, the selection of input x is random. Transforming equation 4 and 5 into

$$P_t(X|i) = \frac{P_t(i|X) \cdot P_t(X)}{P_t(i)} \tag{6}$$

and

$$P(X|i) = \frac{P(i|X) \cdot P(X)}{P(i)} \tag{7}$$

and following our assumption that $P_t(X|i) = P(X|i)$, we can derive that

$$P_t(i|X) = \frac{P_t(i)}{P(i)} \cdot P(i|X) \cdot P(X) \tag{8}$$

. Considering $\sum_{i=1}^{n} P_t(i|X) = 1$, we can get that $P(X) = 1/(\sum_{j=1}^{n} \frac{P_t(j)}{P(j)} \cdot P(j|X))$. Finally, the rescaling formular is

$$P_t(i|X) = \frac{\frac{P_t(i)}{P(i)} \cdot P(i|X)}{\sum_{j=1}^{n} \frac{P_t(i)}{P(j)} \cdot P(j|X)} \tag{9}$$

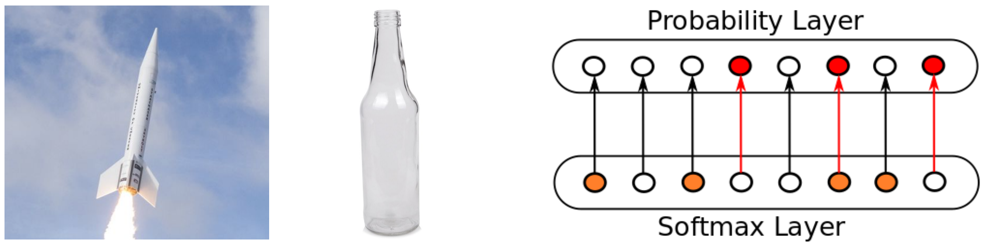
To give probability layer the ability to detect new classes, we choose not to rescale the outputs from softmax layer when the original model has strong confidence in its prediction and set the formula of probability layer as

$$P_t(i|X) = \frac{\frac{P_t(i)}{P(i)} \cdot P(i|X)}{\sum_{j=1}^{n} \frac{P_t(i)}{P(j)} \cdot P(j|X)} \cdot I_{\{P(i|X)<\omega\}} + P(i|X) \cdot I_{\{P(i|X)>=\omega\}} \tag{10}$$

, where ω is the threshold when we should trust the original prediction and I_X is the indicator function such that

$$I_X(x) = \begin{cases} 1, & \text{if } x \in X \\ 0, & o.w. \end{cases} \tag{11}$$

. If a model have a strong confidence in its prediction, the accuracy would be much higher than the model's average accuracy. Our experiments show that CNNs will give most of the images high



(a) Rocket **(b)** Bottle **(c)** Computation Intuition

Figure 1: Probability Layer Intuition. Fig .1a and fig. 1b give example of similar classes and fig. 1c shows how probability layer and softmax layer take effect together.

predicted probability and the accuracy of these images will exceed average accuracy a lot. Probability layer helps when the original model is confused on the prediction and will not interfere the decision when the original model has confidence in its prediction.

Probability layer helps by using environment information that original CNNs does not use. When human recognizes an object, both vision and environment information will be used, i.e., what we have seen recently and which objects may appear here. However, CNNs can only make use of visual information while discarding environment information, which makes it extremely difficult to distinguish classes with similar shapes. For example, fig. 1a and 1b shows images from CIFAR100 for bottle and rocket respectively. It is hard to distinguish these two classes only from images while environment information can easily rule out rocket in most scenarios.

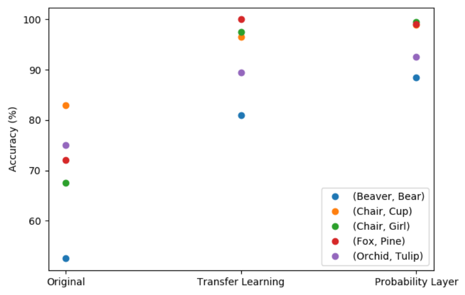
Fig. 1 gives intuition on how probability layer utilize environment information. In fig. 1c, the lower row represents the outputs from softmax layer and the upper row represents the probability layer. The orange nodes stand for the classes with high predicted probability in softmax layer and the red nodes stand for the suggestion from the environment. The prediction from probability layer will be selected from the intersection of the set of red nodes and orange nodes, which rules out confusing classes for CNNs.

4 Experiments

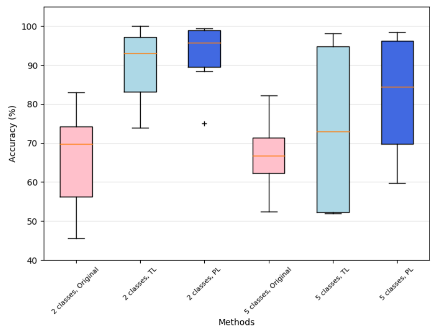
We implemented probability layer on DenseNet [10] and evaluated the CNN model with probability layer, i.e., *PCNN*, on specialized datasets with various number of classes and class distribution. We choose DenseNet as the base model since it is the state-of-the-art and has similar results with other state-of-the-art models, i.e., GoogLeNet [20] and ResNet [9]. We reimplemented the DenseNet on Tensorflow [1] and trained the model on CIFAR100 [11] from scratch. The *specialized dataset* is generated from CIFAR100 [11], which originally has 100 classes with equal weight.

Transfer Learning. Following the published practice [4, 7, 14, 17, 23], all fully-connected layers after convolutional layers are finetuned on the generated dataset with same class distribution as the testing dataset. To eliminate the effect of the epoch, we used three different epochs, i.e., 1, 5, and 30. The maximum epoch is 30 because latency and energy efficiency are considered.

To show the potential of PCNN, we evaluate probability layer on specialized datasets composed of a various number of classes and class distributions. We begin our experiments by showing that different class combinations have significant influence over the accuracy and probability layer can bring in benefit and perform better than transfer learning on all class combinations. Then we show that probability layer can improve accuracy for any number of classes and make use of unbalanced distribution. We proceed to show that most of the images will have predicted probability higher than 75%, which are called *high-confidence images*, and the accuracy among these high-confidence images is 84%. Finally, we demonstrate that probability layer is complementary to acceleration methods through combining probability layer with spatial redundancy elimination approach [5] to achieve decreased computation and improved accuracy.



(a) Concrete examples



(b) Summary of class effect

Figure 2: Class effect on the original model, the model after finetuning, and the model with probability layer.

4.1 Class effect

We explore the effect of class combinations and compare the performance of probability layer and transfer learning on various class combinations. Fixing the number of classes as two, we randomly selected five class combinations and compare the original model, model after transfer learning, and the model with probability layer, see fig. 2a. We see that with the same model, different class combinations may have dramatically different accuracy. While the original model can achieve a accuracy as 83% for classifying chair and cup, the accuracy for beaver and bear is only 52.5%. We can also observe that both transfer learning and probability layer can bring in benefit no matter which two classes compose the dataset while probability layer generally performs better than transfer learning. In the randomly selected five class combinations, probability layer can provide an additional benefit of 2.16% on average. Note that for fox and pine, transfer learning provides a slightly better accuracy than probability layer. We believe that this is due to randomness, especially when considering that transfer learning performs better on all other class combinations.

Fig. 2b gives a summary of the performance of all three models on the specialized dataset with two and five classes. The figure shows the result from 100 randomly selected class combinations. We see that, on the original model, while the average accuracy for classifying two classes is 70%, the minimum accuracy could be 45% and the maximum accuracy can reach 83%, which shows that the accuracy of classifying two classes could change dramatically even using the same model. This observation also holds for transfer learning and probability layer. We can also note that the average accuracy of probability layer is much higher than transfer learning.

4.2 Results on specialized datasets with only majority classes

We explore probability layer's ability on making use of environment when there are only a few majority classes. For each number of classes, we randomly sampled 100 subsets of classes and present the average accuracy in fig. 3. We see that significant benefit has been achieved by probability layer for all number of classes. When there are 5 classes, an increase of more than 20% can be achieved without any finetuning. Another point worth noting is that the benefit diminishes gradually as the number of classes increases. Even if there are 40 classes, a benefit over 10% could still be observed. This shows a significant advantage over previously published results [17], in which no benefit exists when there are more than 15 classes.

We compare our results with transfer learning in fig. 3. For every specialized dataset, we finetune the model for 5 rounds on a training dataset with the same class combinations and class distribution as the testing dataset. For all selected class numbers, probability layer performs better than transfer learning. This advantage of probability layer over transfer learning increases as the number of classes increase. We contribute this phenomenon to the fact that PCNN has seen more images than transfer learning. Existing papers [23] has also reported that transfer learning may destroy the co-adaption between layers and deteriorate the performance on prediction. Another point worth noting is that when the number of classes increases over 90, retraining would bring worse accuracy than the original

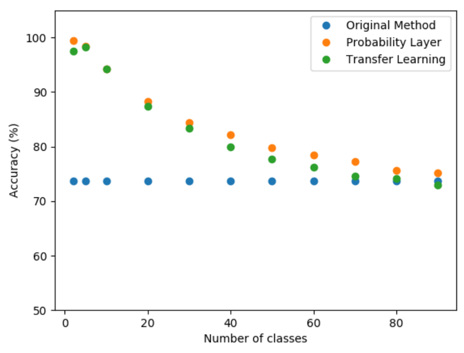


Figure 3: Performance on different number of classes

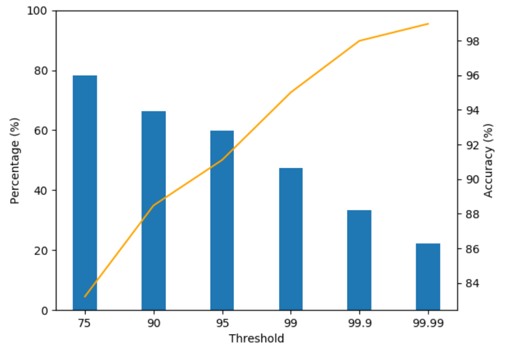


Figure 4: Threshold

model without retraining. In contrast, the probability layer can still bring 2% advantage over the original model. We believe the reason is that the deterioration of co-adaption between layers leads to a decrease in accuracy and the reduction in the number of classes cannot make up this deterioration when the number of classes is 90, which is almost same as the original class numbers. Probability layer does not need finetuning and thus avoid this problem. All these observations indicate that probability layer has better ability in using various environment.

4.3 High confidence, high accuracy

We justify our design of threshold in probability layer by exhibiting the percentage of images with predicted probability layer higher than the various threshold and the accuracy of these images, see fig. 4. While the original DenseNet has top-1 accuracy as 73%, the accuracy increase to 83% when we set the threshold ω to be 75%. When we increase furtherly the threshold ω to be 99%, the accuracy would increase to 95.01%. Thus when a model has strong confidence in its prediction, we should better believe in the model instead of rescaling.

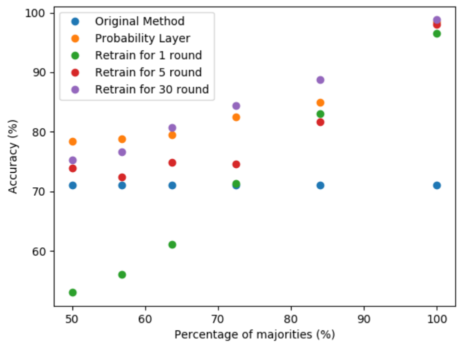
We also note that a large portion of images can get a high predicted probability from the original model. For instance, there are more than 60% images get predicted probability higher than 95%. For these images that the original model has strong confidence in its prediction, the probability layer will not interfere with the decision. The probability layer will step in when the original model is not sure and give suggestion to the probability layer according to the environment information.

4.4 Results on specialized datasets with both majority and minority classes

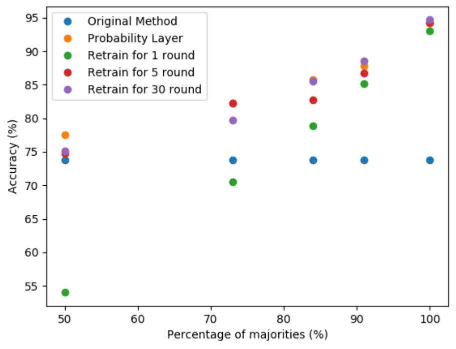
We evaluate the probability layer on the noisy environment, in which a few major classes occupies most of the images while a huge number of minority classes also appear. Fig. 5a and fig. 5b shows the results when the numbers of majority classes are five and ten respectively. While the weight of majority classes increases from 50% to 100%, probability layer performs consistently better than the transfer learning for 5 or 1 rounds. Even if we finetune the model for 30 rounds, probability layer still performs better when the weight of major classes is relatively small. When the weights increase further, the advantage of finetuning for 30 rounds is at most 0.5%. Considering the intensive energy-consumption and time latency of retraining for 30 rounds, this benefit of 0.5% advantage is not so significant, especially on devices which require the real-time response and has strict energy budget. We should also note that retraining for 1 rounds would make the accuracy to be much lower than original model when the weight of major classes is less than 75%. Since we need to choose a method before we start in automatically using environment information, we can conclude that probability layer is the most suitable method.

4.5 Combining acceleration methods

A promising way to achieve high speed up and accuracy is to combine acceleration methods with probability layer. For this to succeed, the acceleration methods should utilize different types of redundancy in the network. In this section, we verify that probability layer can be combined with an



(a) Five classes with various distribution



(b) Ten classes with various distribution

Figure 5: Probability layer on denseNet

acceleration method of using spatial redundancy, *PerforatedCNN* [5], to achieve high speed up while increasing top-1 accuracy.

We reimplemented the PerforatedCNN [5] on DenseNet. PerforatedCNN makes use of spatial redundancy in the image by skipping evaluation in some of the spatial positions. Different from other methods in using spatial redundancy, i.e., increasing strides, PerforatedCNN will interpolate these skipped positions using nearest neighborhood, such that the output size will be unchanged. In this way, the architecture remains same and no finetuning is needed. The shortage of PerforatedCNN is that it may introduce a huge decrease in accuracy. Our experiments show that this drawback of PerforatedCNN could be made up by probability layer. Thus combining probability layer with other acceleration methods can both decrease computation and increase accuracy.

We first apply the probability layer to the network. Then we apply the spatial redundancy elimination methods to this network. In the whole process, no finetuning is needed. The PerforatedCNN is tested at the theoretical speedup level of 2x. The testing dataset contains 5 randomly selected classes with equal frequency. The results are presented in the table 1. Due to class effect, the original model will give a top-1 accuracy of 67.9%, which is slightly lower than the average accuracy of DenseNet on CIFAR100. With the probability layer, the model without finetuning can increase the top-1 accuracy dramatically to be 98.4%. The PerforatedCNN will give a top-1 accuracy of 48.19% if we choose the theoretical speedup level of 2x, which is similar to the results reported in PerforatedCNN [5]. Adding the proposed method, the PerforatedCNN can give a top-1 accuracy of 92.20% while decreasing computation by half, which shows that probability layer complements spatial redundancy elimination methods perfectly and provides a promising perspective of combining probability layer with other acceleration methods.

Table 1: Summary

Method	Mult. ↓	Top-1 Accuracy
Original Model	1.0x	67.79%
Probability Layer	1.0x	98.4%
Perforation	2.0x	48.19%
Combined Method	2.0x	92.20%

5 Conclusion

We have presented probability layer which exploits runtime environment information to increase prediction accuracy. Probability layer requires only a single extra layer after the general CNN models with unnoticeable extra computation and obtains a boost in accuracy. Compared to transfer learning, probability layer achieves comparable or even better accuracy boost, avoids retraining, and maintain the ability to detect new classes. Additionally, probability layer can be combined with acceleration methods which exploit various types of network redundancy to achieve further speedups.

References

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[4] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[5] Mikhail Figurnov, Aizhan Ibraimova, Dmitry P Vetrov, and Pushmeet Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions. In *Advances in Neural Information Processing Systems*, pages 947–955, 2016.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[7] Seungyeop Han, Haichen Shen, Matthai Philipose, Sharad Agarwal, Alec Wolman, and Arvind Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pages 123–136. ACM, 2016.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

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

[10] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017.

[11] A Krizhevsky and G Hinton. Learning multiple layers of features from tiny images. *Technical report, University of Toronto*, 1, 01 2009.

[12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[13] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[14] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*, pages 1717–1724. IEEE, 2014.

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[16] Marco Saerens, Patrice Latinne, and Christine Decaestecker. Adjusting the outputs of a classifier to new a priori probabilities: a simple procedure. *Neural computation*, 14(1):21–41, 2002.

[17] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. Fast video classification via adaptive cascading of deep models. *arXiv preprint*, 2017.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[19] Samuel L Smith, Pieter-Jan Kindermans, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017.

[20] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, et al. Going deeper with convolutions. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Cvpr, 2015.

[21] Shuo Wang, Leandro L Minku, and Xin Yao. Dealing with multiple classes in online class imbalance learning. In *IJCAI*, pages 2118–2124, 2016.

[22] D Randall Wilson and Tony R Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[23] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.