TEST-COST-SENSITIVE CONVOLUTIONAL NEURAL NETWORKS WITH EXPERT BRANCHES

Mahdi Naghibi[1], Reza Anvari[1], Ali Forghani[1] and Behrouz Minaei[2]

[1]Faculty of Electrical and Computer Engineering, Malek-Ashtar University of Technology, Iran
[2]Faculty of Computer Engineering, Iran University of Science and Technology, Iran

ABSTRACT

It has been proven that deeper convolutional neural networks (CNN) can result in better accuracy in many problems, but this accuracy comes with a high computational cost. Also, input instances have not the same difficulty. As a solution for accuracy vs. computational cost dilemma, we introduce a new test-cost-sensitive method for convolutional neural networks. This method trains a CNN with a set of auxiliary outputs and expert branches in some middle layers of the network. The expert branches decide to use a shallower part of the network or going deeper to the end, based on the difficulty of input instance. The expert branches learn to determine: is the current network prediction is wrong and if the given instance passed to deeper layers of the network it will generate right output; If not, then the expert branches stop the computation process. The experimental results on standard dataset CIFAR-10 show that the proposed method can train models with lower test-cost and competitive accuracy in comparison with basic models.

KEYWORDS

Test-Cost-Sensitive Learning; Deep Learning; CNN withExpert Branches; Instance-Based Cost

1. INTRODUCTION

Deep convolutional neural networks have produced state-of-the-art results on various benchmarks[1], [2]. Many Researches in the field of convolutional neural networks, practically proved that deeper networks have higher accuracy. Today the state of the art deep CNNs have more than one hundred layers and millions of weights and parameters[3]. This needs a vast amount of computational power and time to execute a network and generate the final output. The high computational cost of these networks can get real systems and applications[4], [5] into trouble. For example, a cloud computing service should process too many requests in every second, or mobile and embedded systems may have not enough power and hardware to run the network for its inputs. So it is very important to reduce the computational cost of networks while keeping their accuracy during the inference. If we consider outputs of each layer of the network as a set of features for the next layer, then computing features of each layer have its own test-cost which a cost-sensitive approach should consider them during computing network output. Figure 1 illustrates the running process of a typical CNN. The model gets an input image and performs some convolution and pooling process layer by layer in the network. Fully connected layers exist at the end of the model which produce the final output for the given instance.

Different methods have been proposed for test-cost reduction and compression of deep convolutional networks. The compression methods try to reduce the number of network parameters, but these approaches do not necessarily make faster networks; because most of the computation of a CNN is related to the convolution operations which cannot be reduced by

DOI: 10.5121/sipij.2019.10502

network compression only. Some recent researches focused on instance-based or input dependent methods which dynamically use a set of models or use some parts of the models to generate the result for a given instance[6]. As we know, even doubling the depth of network will have a small effect on accuracy, and all input instances have not the same difficulty, so many instances can be handled with shallower or simpler models.

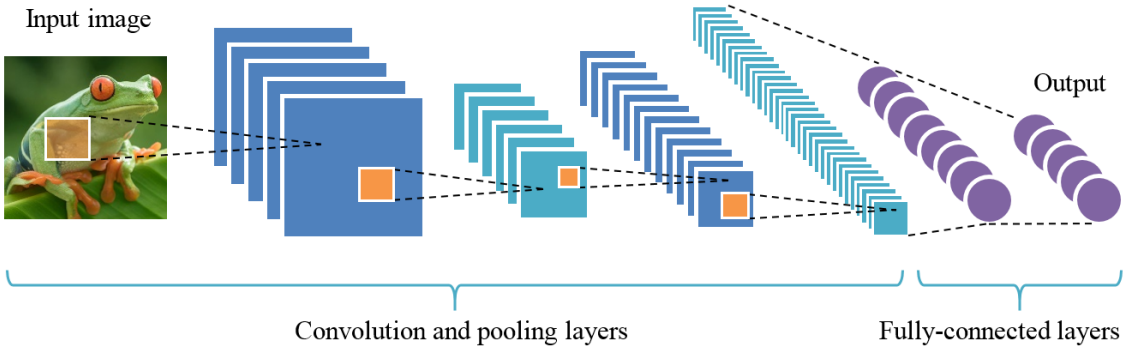

Figure 1. Illustration of deploying a typical CNN model on an input image

Along the line of dynamic and instance-based approaches, in this paper, we propose a new test-cost-sensitive method for deep convolutional networks which can learn to manage the available computational resources in the way that result in faster inference for many input instances. This method uses a set of middle output and expert branches in the convolutional network. When an instance is given to the network input, the computation is started layer by layer to the end of first middle output and expert branch. If the expert branch says that the generated output for the given instance is wrong at this output level but can be corrected in deeper layers of the network, then the running process of the network continues to the higher layers until the next output of the network. For other cases, the expert branches stop the computation process and assign the current output as the final output of the network. In this way, the deeper layers which result in higher computational cost are only used when the expert branch indicates the possibility of improvement in output accuracy, and prevent from useless computational power consumption. This can reduce the overall test-cost and keep the network accuracy at an acceptable level in comparison with the basic model. The experiments on standard datasets show the advantages of the proposed method in comparison with other methods.

The paper continues as follow: in the next section we review the related works in test-cost-sensitive deep learning,section three describes the proposed method in details, in section four we present the experimental results, and section five belongs to conclusions.

2. RELATED WORK

There are various types of costs during a machine learning process [7]. Since computational cost is a real challenge for deep neural networks, researches proposed different methods and approaches to solve it. In this section, we investigate the literature available in this field. These researches may do not use the test-cost-sensitive terminology but are relevant to the current research. The approaches can be categorized into three main categories. The first category belongs to methods that train a new model based on the original one or modify the trained models[8]. Methods of the second category increase the speed of deep networks using advanced computational methods and more efficient using of hardware[9]. Dynamic instance-based approaches are the third category of test-cost-sensitive methods for deep learning which resulted in effective solutions in recent years[6] and the proposed method of this paper belongs to this category. In the following, we describe these approached in more details with some example researches.

2.1. Making a Modified Model

Methods of this approach modify an existing model or learn a new model from scratch to reduce complexity and computational operations of the original model. “Mimicking” network methods train a new shallow network [10] or a “Fitnet”[11], which is called student model. This new model is made from scratch to mimic the behaviour of the original model which is called the teacher model. The newly generated models are more compact, In [10] they are shallower and in [11] models have fewer filters and are fitter. Network decomposition methods [12]–[14] is another group of model modification approaches that use estimation solutions. In these methods, filters are decomposed in the way that increases the total speed of the network but the output of original network layers still estimated well. Older network pruning methods [15]do not consider the computational cost reduction as their goal, but sparsification of the model reduces its complexity which indirectly results in the faster network[16].

2.2. Advanced and Low-Level Computational Methods

Unlike the previous approach, these methods increase the speed of the deep network, without modifying the network structure. One family of methods focus on the way of computing layer outputs, specifically using fast Fourier transform (FFT)[9]. Another family, target the efficient usage of available hardware [17], [18] by low-level parallel computation, efficient memory usage, and low precision arithmetic operations.

2.3. Adaptive Methods

Both of previous approaches have a static behaviour with all of the input instances and cannot allocate the computational resources with an input dependent policy. So there was a lack of test-cost-sensitive approaches that use computational recourse only when it is needed based on the given instance and with a dynamic manner. In recent years some solutions based on this approach have been proposed which we call them adaptive methods. Also, the adaptive methods can be combined with two previous categories of methods and make use of the advantages of both. One main group of researches in adaptive models is network cascades. These methods train a set of deep networks and use them in a cascade fashion. They start with simple models that have lower test-cost and continue the process with more complex networks until reaching an acceptable degree of confidence for the generated output. In this way models with heavier computations is only used for more challenging input instances.

Deep Decision Network proposed in [19] for the classification of images. The method recognizes the hardness of instances and passes more difficult images to subsequent models in the cascade. The method in[20], called convolutional neural networks cascade, proposed for face detection. It operates on versions of the image with different resolutions, rejects the background regions in low-resolution stages and passes some challenging candidates to high-resolution evaluations. DeepPose proposed in [21] makes cascade deep regression framework using a divide and conquer strategy for human pose estimation.

In a different fashion of cascade, the authors of [6] proposed Deep Layer Cascade for the semantic image segmentation problem. Layer cascade, unlike model cascades which use a set of models, trains a single network with some internal branches that generate a degree of confidence for the regions of the image and stop the process for easier parts which are recognized in lower layers of the network and pass harder regions to higher levels of the deep network. The proposed method in this paper is similar with layer cascade method but instead of using middle outputs as the degree of confidence for the regions of the image, we use expert branches which are specially trained to recognize instances that need a deeper process to be categorized correctly. Also, we use the proposed method as a solution for image classification problem.

3. CNNs WITH EXPERT BRANCHES

In this section, we explain the proposed method in more details. It is called CNNs with Expert Branches (CNN-EB). In the following first we investigate the relationship between computational cost in CNNs and test-cost of classification. Then we explain the method details and describe it as an algorithm in the third part of this section.

3.1. Test-Cost in CNNs

The word test-cost comes from medical diagnosis field and means that if we want to do any test on the patient to find the related values of that test, we should consider its cost. Based on this concept we define the test-cost in deep CNNs. The deep learning methods have two main property: automatic learning of features, and a layered process of learning. The specifications of the learning process in deep CNNs mixed test-cost and computation cost concepts. That means in the process of feature extraction and learning in the layers of CNN, each layer gains values of a set of features (test-cost) by means of doing necessary computations (computation cost). That features have more abstraction and representation power in comparison with features in previous layers and can result in more accurate decisions in the CNN model.

In the other words, we can consider a CNN model in the forms of a set of successive layers that each layer is responsible for extraction and computation of a feature set, and this is done by spending the required cost for doing tests and related computations. Also, we can consider the output of a set of network layers which builds a continuous block of the CNN, as the features for the successive building block of the network. Considering this viewpoint, in the next part, we describe a test-cost sensitive method for deep CNNs.

3.2. Model Architecture

The proposed deep CNN model consists of a common convolutional network and two types of augmented branches, which include middle output (or classifier) branches and expert branches. They are paired with each other and operates together on the middle points of the CNN. The output branches are extra output generators that, for example, can recognize the label of input instance in a classification problem. The expert branches look at the data from another view; they decide on passing input instance to higher layers of the network or considering the current generated result of the paired output branch as the final output of the network. To do this, the expert branches are trained to find instances that are recognized wrongly at current level of the network but can be classified correctly in higher levels and successive layers of the deep CNN. The training of expert branches is done based on the extracted features from the instance in concatenation with the result of corresponding paired output branch. This concatenation represents more features available to the expert branch and makes it able to generate more accurate decisions.

Formally we can define the elements of the proposed CNNs with expert branches as follows:

- $L_i = \{l_i^1, l_i^2, \dots l_i^k\}$ set consists of$k$ layer l_i^j,that builds a branch of the network and each$l_i^j \in L_i$is one of the common CNN layer types. L_{bn} contains layers of the base network.

- $\Omega = \{O_1, O_2, \dots O_m\}$set of m output branches, all of them are middle branches except O_m which is the last output of the network. Each branch $O_i$consists of a set of layers L_{O_i}.

- Given the input instance x which has actual output y, The $\hat{y}_i$ is the generated output vector by output branch $O_i$for its input vector $\acute{x}_{bn}^j$:

$$\acute{x}_{bn}^{j} = f_{bn}^{j}\left(x; l_{bn}^{1}, \dots, l_{bn}^{j}\right) \quad (1)$$

and

$$\acute{x}_{O_i}^{k} = f_{O_i}^{k}\left(\acute{x}_{bn}^{j}; l_{O_i}^{1}, \dots, l_{O_i}^{k}\right) \quad (2)$$

where f_{bn}^{j} is the processing function of the base network from layer l_{bn}^{1} to l_{bn}^{j}, and $\acute{x}_{O_i}^{k}$ is the output vector of layer $l_{O_i}^{k}$ of output branch O_i, and $f_{O_i}^{k}$ is the processing function of this branch. Then we have:

$$\hat{y}_i = \sigma\left(\acute{x}_{O_i}^{k}\right) = \frac{exp\left(\acute{x}_{O_i}^{k}\right)}{\sum_{c=1}^{|C|} exp\left(\acute{x}_{O_{i_c}}^{k}\right)} \quad (3)$$

where σ is the softmax function and $|C|$ is the number of dimensions of output y (number of classes in classification problem).

- $E = \{E_1, E_2, \dots E_{m-1}\}$set of *m-1* expert branches. They are experts that decide about continuing the feature extraction process in the higher layers. Each expert branch E_i is paired with an output branch O_i and both are connected to the same point of the base network. E_i consists of a set of the layers L_{E_i}. The last output branch O_m is not paired with an expert branch. Formally we have:

$$\acute{x}_{E_i}^{p} = f_{E_i}^{p}\left(\acute{x}_{bn}^{j}; l_{E_i}^{1}, \dots, l_{E_i}^{p}\right) \quad (4)$$

and

$$\acute{x}_{E_i}^{k} = f_{E_i}^{p,k}\left(\acute{x}_{E_i}^{p} \oplus \acute{x}_{O_i}^{k}; l_{E_i}^{\mathrm{p}}, \dots, l_{E_i}^{k}\right) \quad (5)$$

Where $\acute{x}_{E_i}^{p}$ is the output vector of the middle layer $l_{E_i}^{p}$ of expert branch E_i, and $\acute{x}_{E_i}^{k}$ is the last output vector of this expert branch. $f_{E_i}^{p,k}$ is the processing function from layer $l_{E_i}^{\mathrm{p}}$ to $l_{E_i}^{k}$, and its input $\acute{x}_{E_i}^{p} \oplus \acute{x}_{O_i}^{k}$ is the concatenation of the middle layer's output of branch E_i and output vector of branch O_i. The decision d_i is generated by expert branch E_i using the following formula:

$$\hat{d}_i = \sigma\left(\acute{x}_{E_i}^{k}\right) = \frac{exp\left(\acute{x}_{E_i}^{k}\right)}{\sum_{d=1}^{|D|} exp\left(\acute{x}_{E_{i_d}}^{k}\right)} \quad (6)$$

Where $|D|$ is the number of dimensions of decisions made by the expert, and $D = \{FT, Other\}$ where FT means that the generated output $\hat{y}_i$ for instance x by output branch O_i is false and the true label will be made in the higher output branches of the network, and $Other$ means all of the other cases.

Figure 2 illustrates the architecture of the CNN-EB. The process starts by getting input instance x and continues layer by layer to classifier branches O_i and expert branches E_i. If we get to the last output branch or the "check $\hat{d}_i$" node in the network decides to stops the process then $\hat{y}_i$ is considered as the final output of the network.

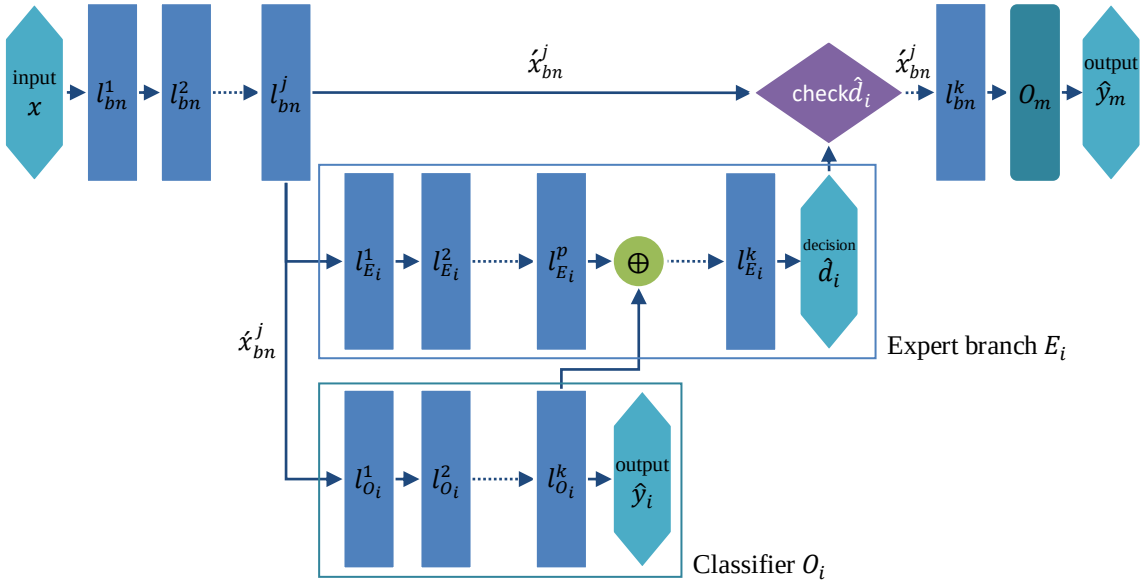

Figure 2 Illustration of the CNN-EB architecture. It includes base branch, expert branches and classifier branches

3.2. Model Algorithm

The pseudocode of generating the output vector for a given instance is illustrated in figure 3.The algorithm gets x and $confidenceThreshold$ as inputs, which are the vector of input instance and the confidence threshold used for decision $\hat{d}_i$, respectively. The processing of the input starts layer by layer in the main branch of the network to $O_r.branchPoint$ which is the position of the next-first output branch O_i and expert branch E_i. Then the output $\hat{y}_i$ is computed for branch O_i, and the decision $\hat{d}_i$ is generated by concatenating $\acute{x}_{E_i}^p$ and $\acute{x}_{O_i}^k$ during the computation of the layers for E_i. If $\hat{d}_i[Other]$ which is decision vector value for $Other$, be higher than $confidenceThreshold$, or we get the last output branch O_m, then the algorithm stops the process and $\hat{y}_i$ is considered as the final output of the network, Otherwise, the process continues for higher layers.

This way, by managing the cost of computing and extracting feature values, the network will be able to generate final output for easy instances at a lower cost, and spend more cost on complex instances and continue computing at higher layers of the network.

4. EXPERIMENTAL STUDY

In this part section, first, we explain the metrics used for comparison of the methods. Then the dataset and settings of experiments are described. After that, the results and their analysis is explained based on the metrics.

4.1. Evaluation Metrics

Based on the cost-sensitive approach of the proposed method described in the previous sections, in addition to evaluating system performance using common standard metrics, the computational costs are also considered. Well-known and standard metrics including Recall, Precision, and Accuracy were used to evaluate the efficiency of the image processing methods. Also, the computational cost of the methods is evaluated on a time-based basis.

Algorithm: Apply the model of CNN with expert branches

input: x: an input instance
$confidenceThreshold$: confidence threshold for decisions
output: $\hat{y}$: generated output vector for the input instance
method: CNN-EB-Apply-Model(x)
$i \leftarrow 1$
$r \leftarrow 1$
$\acute{x} \leftarrow x$
while $r \leq m$ **do** // m is the number of branches
$j \leftarrow O_r.branchPoint$ //position of the next-first output and expert branches
$\acute{x}_{bn}^{j} \leftarrow f_{bn}^{i,j}(\acute{x}; l_{bn}^{i}, \dots, l_{bn}^{j})$ //base network
$\acute{x}_{O_i}^{k} \leftarrow f_{O_i}^{k}(\acute{x}_{bn}^{j}; l_{O_i}^{1}, \dots, l_{O_i}^{k})$ //classifier branch
$\hat{y}_i \leftarrow \sigma(\acute{x}_{O_i}^{k})$
if $r \neq m$ **then**//there are $m-1$ expert branches
$\acute{x}_{E_i}^{p} \leftarrow f_{E_i}^{p}(\acute{x}_{bn}^{j}; l_{E_i}^{1}, \dots, l_{E_i}^{p})$
$\acute{x}_{E_i}^{k} \leftarrow f_{E_i}^{p,k}(\acute{x}_{E_i}^{p} \oplus \acute{x}_{O_i}^{k}; l_{E_i}^{\mathrm{p}}, \dots, l_{E_i}^{k})$
$\hat{d}_i \leftarrow \sigma(\acute{x}_{E_i}^{k})$
end if
if $r = m$ **or** $\hat{d}_i[Other] > confidenceThreshold$ **then**
$\hat{y} \leftarrow \hat{y}_i$
break while
else
$i \leftarrow j + 1$
$r \leftarrow r + 1$
$\acute{x} \leftarrow \acute{x}_{bn}^{j}$
end if
end while
return $\hat{y}$
end method

Figure 3. Pseudocode of Applying the proposed CNN-EB

The metrics are calculated using the following equations:

$$Recall = \frac{tp}{tp + fn} \tag{7}$$

$$Precision = \frac{tp}{tp + fp} \tag{8}$$

$$Accuracy = \frac{tp + tn}{tp + tn + fp + fn} \tag{9}$$

wheretp is true positive, tn is true negative, fp is false positive, and fn is false negative results during the classification process.

4.2. The Dataset

To evaluate the methods we used the CIFAR-10 dataset [22]which is one of the most widely used datasets for image processing research. This dataset contains 60,000 images in 10 different classes. Each class consists of 6,000 images. About 85% of images are used for training and the rest of images is used for testing of models. Table 1 shows the specifications of the CIFAR-10.

Table 1. Specifications of CIFAR-10 dataset used for evaluation of methods

Class	**Train dataset**	**Test dataset**
Airplanes	5,000	1,000
Birds	5,000	1,000
Cars	5,000	1,000
Cats	5,000	1,000
Deers	5,000	1,000
Doges	5,000	1,000
Frogs	5,000	1,000
Horses	5,000	1,000
Ships	5,000	1,000
Trucks	5,000	1,000
Total	50,000	10,000

4.2. Experimental Settings

Figure 4 shows the architecture of the proposed CNN-EB model implemented for image classification. The structure of this model is similar to the Google inception v3 model[23], but we placed the auxiliary branch of the original model after the first inception module which is called "mixed 5b". By doing so, the auxiliary branch is used as the first classifier O_1 which can generate the output for the input instances with much lower cost in comparison with the main output of the model at the end of the network which is the classifier O_2. The expert branch E_1 is also added to the same branchpoint of O_1 in the network. The structure of E_1 is similar to O_1 in addition to a "concat" layer which concatenates middle outputs of branches O_1 and E_1. The output of E_1 is evaluated by "check decision" node in the network, which make the decision to stop the classification or continue the process in higher layers of the model.

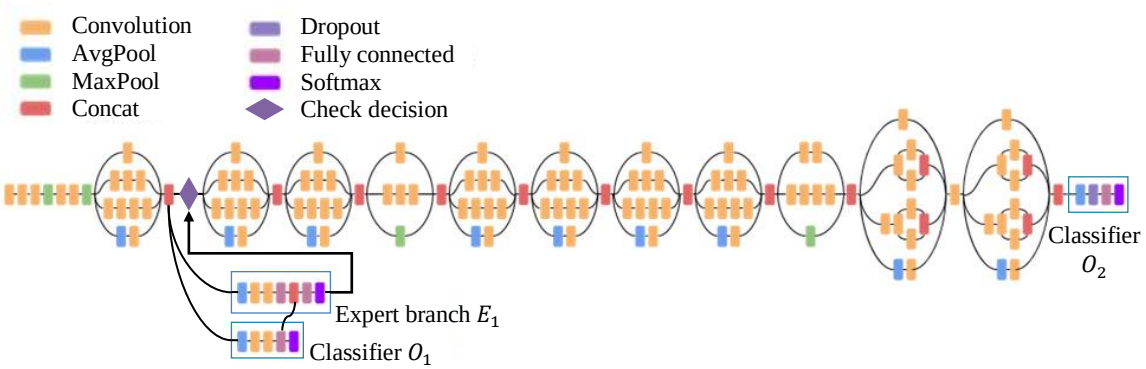

Figure 4. The architecture of implemented proposed CNN-EB method

Three variants of the proposed expert branch method and two basic well-known inception v3 models are compared to specify the characteristics of the proposed method. In the "Auxiliary as Expert Branch" method, the output of the original auxiliary branch of inception v3 which we placed after module 5b, is used as the expert branch decisions by applying some thresholds. This method is very similar to the proposed method in [6] but in a different fashion, we used the output of the auxiliary branch to determine the difficulty of complete instance, not some parts of it. The "Auxiliary+5b as Expert Branch" is implemented by considering auxiliary branch in addition to the inception 5b module as the expert branch of CNN-EB. The "Proposed Expert Branch" is implemented quite similar to the architecture shown in figure 4. The "Inception v3 Auxiliary as Final Classifier" and "Inception v3 Main as Final Classifier" are two basic inception v3 models where in the first model we used the output of auxiliary branch as the final output of the model, and in the second model, the main output of the original inception v3 is used as the final classifier. The TensorFlow[24]which is a well-known machine learning framework is used for the implementation of the models. A machine with Intel Core i7 CPU and Nvidia GeForce GT 740M GPU is used for training of the models.

4.3. Implementation Results

In this section first, we evaluate the performance of the expert branches Apart from the base networks. The FT class is considered as negative and the $Other$ class is considered as positive. Figure 5 shows the precision against the recall of expert branch methods for different $confidenceThreshold$s. Since a majority number of instances belonging to $Other$ class and we considered them as positive samples, the precision of the expert branches is greater than 85% for all of the methods. As we see "Proposed Expert Branch" has higher performance than the other methods and as expected the "Auxiliary+5b as Expert Branch" which has deeper expert branch structure than "Auxiliary as Expert Branch", resulted in more accurate branch model.

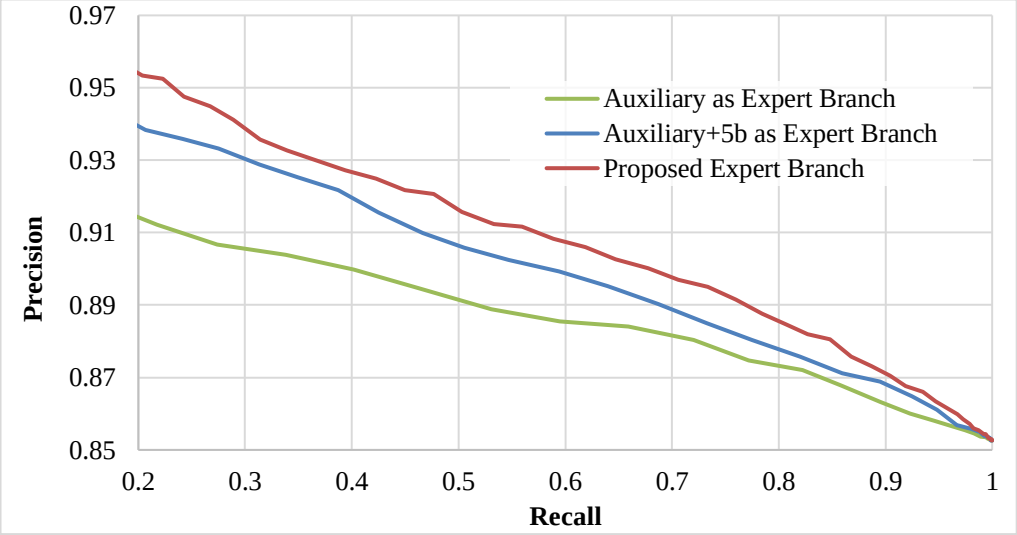

Figure 5. Illustration of precision against recall at various thresholds for expert branches

The ROC curves of the expert branches which shows the true positive rate (TPR) against the false positive rate (FPR) at various $confidenceThreshold$s, is illustrated in figure 6. The "Proposed Expert Branch" has higher curves in comparison with the other methods and same positions as the results of figure 5 are held for ROC curves of the expert branches.

The accuracy against the time at several$confidenceThreshold$s for two basic inception v3 methods and three variants of the proposed CNN with expert branch methods are shown in figure

7. The line between "Inception v3 Auxiliary as Final Classifier" and "Inception v3 Main as Final Classifier" illustrates the imaginary linear growth of accuracy against time for these models.

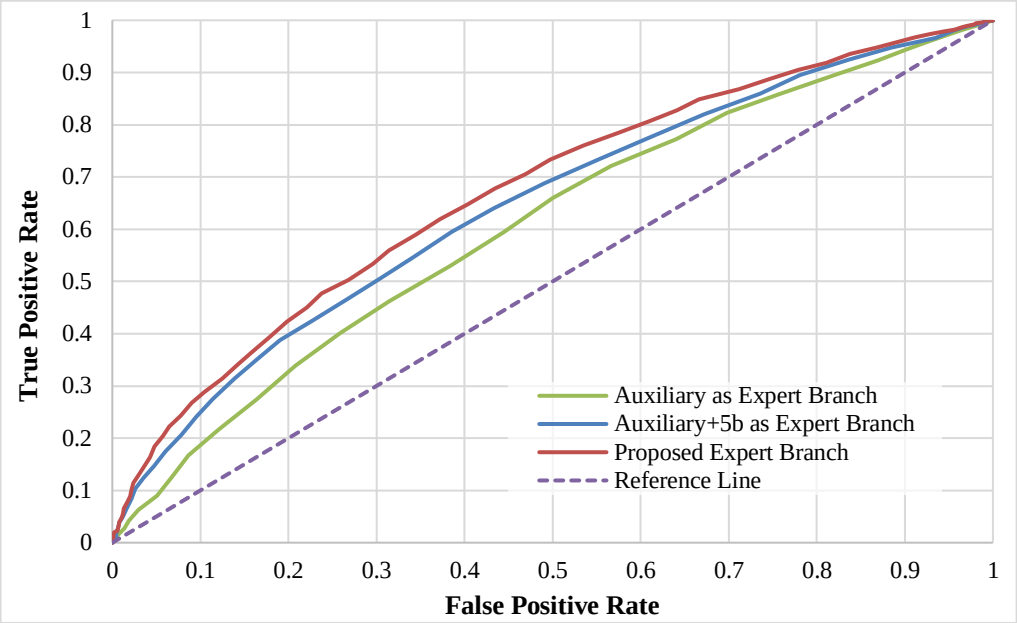

Figure 6. Illustration of The ROC curves for expert branch methods

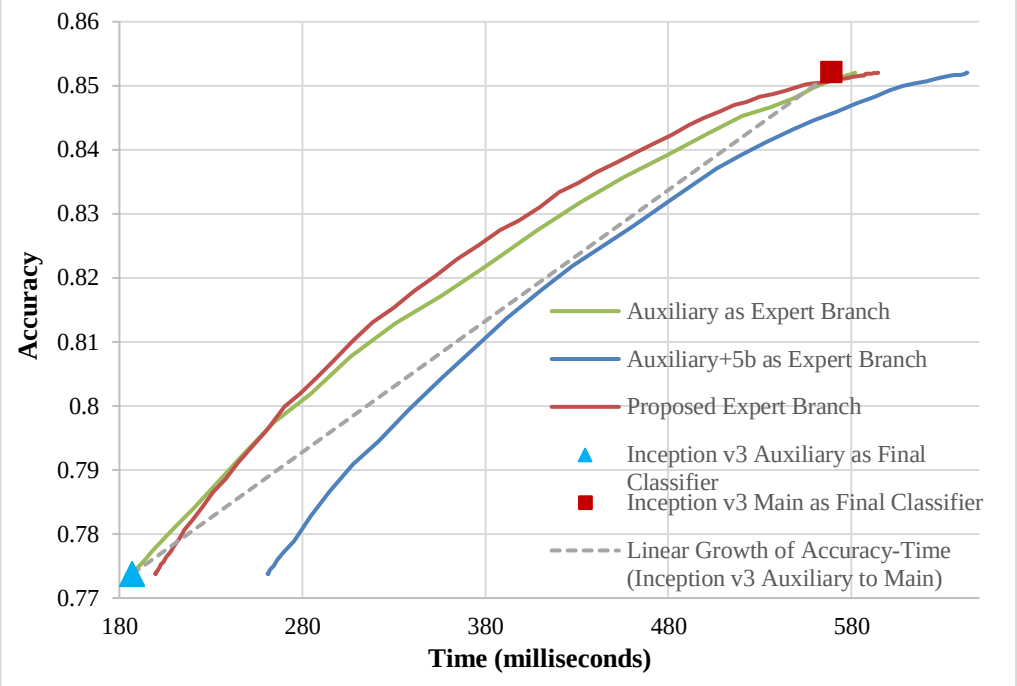

Figure 7. Illustration of accuracy against time at various thresholds for different methods

As can be seen in figure 7 the "Proposed Expert Branch" method has better performance than the other expert branch methods and a small decrease in the accuracy of this method can save a significant amount of processing time and reduce the computational cost of the model. Since The "Auxiliary as Expert Branch" method has lower computational cost in comparison with "Auxiliary+5b as Expert Branch" method, in the most of the cases it can make the same accuracy

with lower cost. The "Proposed Expert Branch" and "Auxiliary as Expert Branch" methods are almost above the imaginary line between two basic inception v3 methods. This indicates that the proposed CNN-EB method is successful in managing the use of computational resources by utilizing the shallower and deeper structure of the network for easier and harder instance, respectively.

Figure 8. Sample results of output classifiers O_1 and O_2 for five CIFAR-10 classes. The left side shows easier instances where both classifiers made true classification, and the right side shows harder instances where only classifier O_2 made the true classification.

To make a visual understanding of easy and hard images for classifiers, figure 8 shows some sample results of output classifiers O_1 and O_2(based on figure 4) for five CIFAR-10 classes. The left side shows easier instances where objects inside the images are clear, with a good position and angle which make it easy for the shallower classifier to predict the true label for these instances. The right side contains harder instances that contain parts of the objects, strange images and multiple objects in the image. Only classifier O_2 that utilizes the deeper structure of the network can generate the true label for hard instances. The illustrated images in figure 8 support the idea of existing easy and hard images in the dataset, and possibility of using cost-sensitive approaches that classify easy instances in shallower and hard instances in deeper layers of the CNN.

Table 2. Comparison of basic and proposed methods based on accuracy and time metrics

Method	**Accuracy**	**Time (msecs)**	**Accuracy Decrease**	**Time Saving**
Inception v3 Auxiliary as Final Classifier	78%	185	-	-
Inception v3 Main as Final Classifier	85%	570	-	-
Auxiliary as Expert Branch	84%	500	1%	14%
Auxiliary+5b as Expert Branch	84%	520	1%	9%
Proposed Expert Branch	**84%**	**450**	**1%**	**21%**
Auxiliary as Expert Branch	83%	430	2%	25%
Auxiliary+5b as Expert Branch	83%	460	2%	19%
Proposed Expert Branch	**83%**	**395**	**2%**	**31%**

Table 2 shows the performance of basic inception v3 methods and variants of the proposed expert branch method based on accuracy and time metrics. As we can see, 1% decrease in the accuracy of "Proposed Expert Branch" model in comparison with basic "Inception v3 Main as Final Classifier", results in 21% time and computational cost saving of the model, and 2% of accuracy decrease makes 31% time-saving. The "Proposed Expert Branch" can save more time than other proposed expert branch method variants with the same accuracy.

5. CONCLUSION

The test-cost of the deep convolutional neural networks is a challenging issue in real-world problems. In this paper, we introduced CNN-EB which is a test-cost-sensitive CNN method that utilizes expert branches to determine the hardness of input instances, and by using shallower layers of the network for easier instances and deeper layers for harder ones, manages the use of available computational resources. The proposed method can be combined with other cost-sensitive CNN methods to make more effective deep models. We implemented the proposed method and compared it with well-known basic method inception v3. The experimental results show that a small decrease in the accuracy of the proposed method in comparison with the basic models will result in significant time and computational resource saving. In order to better evaluate the proposed method, experiments on deeper models can be performed in future work and the efficiency of the proposed method can be investigated for these models.

REFERENCES

[1] S. P. S. Gurjar, S. Gupta, and R. Srivastava, "Automatic Image Annotation Model Using LSTM Approach," Signal Image Process. An Int. J., vol. 8, no. 4, pp. 25–37, Aug. 2017.

[2] S. Maity, M. Abdel-Mottaleb, and S. S. As, "Multimodal Biometrics Recognition from Facial Video via Deep Learning," in Computer Science & Information Technology (CS & IT), 2017, pp. 67–75.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv Prepr. arXiv1512.03385, 2015.

[4] D. Kadam, A. R. Madane, K. Kutty, and B. S.V, "Rain Streaks Elimination Using Image Processing Algorithms," Signal Image Process. An Int. J., vol. 10, no. 03, pp. 21–32, Jun. 2019.

[5] A. Massaro, V. Vitti, and A. Galiano, "Automatic Image Processing Engine Oriented on Quality Control of Electronic Boards," Signal Image Process. An Int. J., vol. 9, no. 2, pp. 01–14, Apr. 2018.

[6] X. Li, Z. Liu, P. Luo, C. Change Loy, and X. Tang, "Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 3193–3202.

[7] M. Naghibi, R. Anvari, A. Forghani, and B. Minaei, "Cost-Sensitive Topical Data Acquisition from the Web," Int. J. Data Min. Knowl. Manag. Process, vol. 09, no. 03, pp. 39–56, May 2019.

[8] A. Polyak and L. Wolf, "Channel-Level Acceleration of Deep Face Representations," Access, IEEE, vol. 3, pp. 2163–2175, 2015.

[9] A. Lavin and S. Gray, "Fast Algorithms for Convolutional Neural Networks," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4013–4021.

[10] J. Ba and R. Caruana, "Do deep nets really need to be deep?," in Advances in neural information processing systems, 2014, pp. 2654–2662.

[11] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," arXiv Prepr. arXiv1412.6550, 2014.

[12] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," 2015.

[13] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in Advances in Neural Information Processing Systems, 2014, pp. 1269–1277.

[14] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," arXiv Prepr. arXiv1405.3866, 2014.

[15] N. Ström, "Sparse connection and pruning in large dynamic artificial neural networks.," in EUROSPEECH, 1997.

[16] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv Prepr. arXiv1207.0580, 2012.

[17] N. Vasilache, J. Johnson, M. Mathieu, S. Chintala, S. Piantino, and Y. LeCun, "Fast convolutional nets with fbfft: A GPU performance evaluation," arXiv Prepr. arXiv1412.7580, 2014.

[18] M. Mathieu, M. Henaff, and Y. LeCun, "Fast training of convolutional networks through FFTs," arXiv Prepr. arXiv1312.5851, 2013.

[19] V. N. Murthy, V. Singh, T. Chen, R. Manmatha, and D. Comaniciu, "Deep decision network for multi-class image classification," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2240–2248.

[20] V. Vanhoucke, A. Senior, and M. Z. Mao, "Improving the speed of neural networks on CPUs," in Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop, 2011, vol. 1.

[21] A. Toshev and C. Szegedy, "Deeppose: Human pose estimation via deep neural networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 1653–1660.

[22] A. Krizhevsky, G. Hinton, and others, "Learning multiple layers of features from tiny images," 2009.

[23] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2818–2826.

[24] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," Mar. 2016.