Contents lists available at ScienceDirect

Neurocomputing

journal homepage: www.elsevier.com/locate/neucom

Multi-target deep neural networks: Theoretical analysis and implementation

Zeng Zeng[a], Nanying Liang[a,*], Xulei Yang[b], Steven Hoi[c]

[a] Institute for Infocomm Research, 1 Fusionopolis Way, #21-01 Connexis (South Tower), 138632 Singapore
[b] Institute of High Performance Computing, 1 Fusionopolis Way, #16-16 Connexis, 138632 Singapore
[c] Singapore Management University, 81 Victoria Street, 188065 Singapore

ARTICLE INFO

Article history:
Received 8 June 2017
Revised 11 August 2017
Accepted 31 August 2017
Available online 8 September 2017

Communicated by Prof. Zidong Wang

Keywords:
Deep neural networks
Multi-target deep learning
Object detection
Segmentation
Learning path

ABSTRACT

In this work, we propose a novel deep neural network referred to as Multi-Target Deep Neural Network (MT-DNN). We theoretically prove that different stable target models with shared learning paths are stable and can achieve optimal solutions respectively. Based on GoogleNet, we design a single model with three different targets, one for classification, one for regression, and one for masks that is composed of 256×256 sub-models. Unlike bounding boxes used in ImageNet, our single model can draw the shapes of target objects, and in the meanwhile, classify the objects and calculate their sizes. We validate our single MT-DNN model via rigorous experiments and prove that the multiple targets can boost each other to achieve optimization solutions.

1. Introduction

Deep neural networks have shown great promise in many practical applications. State-of-the-art performance has been reported in several domains, ranging from image classification [1], speech recognition [2], to text processing [3], playing Atari games [4]. The latest and greatest honor of deep neural networks belongs to AlphaGo [5] that has defeated Lee Sedol, one of the best human professional Go game players in the world, a feat previously thought to be at least a decade away.

Deep neural networks are networks of neurons, which can execute different simple functions and are connected following predefined topologies [6]. Unlike the networks we are familiar with, e.g., mobile networks, computer networks, sensor networks, which have multiple entries and multiple exits, deep neural networks have only one entry, where data can be poured into the networks, and one exit, where the target functions can obtain the results. All the neurons in the networks learn synchronously (the neurons within the same layers) or asynchronously (the neurons in different layers) to achieve optimal solutions of single target models [7]. Hence, from the view of layers, deep neural networks are end-to-end links, instead of networks.

In our real world, we have many multi-label problems [8]. In the famous ImageNet data set, millions of images have got at least one label [1]. Initially, ImageNet required the competitors to identify which class the target image belongs to within 1*k* or 22*k* known categories. Little by little, ImageNet starts to provide images with additional locations of target objects that are indicated by bounding boxes, and then, the competitions become solving multi-label problems. Now, the most successful solutions are to deliberately integrate the different labels into a single target function, and then using deep neural networks to achieve optimization results of the targets [8,9]. However, no matter how many labels the target models may have, the deep neural networks are end-to-end links, solving "Single-Target" problems, instead of end-to-ends networks that can solve "Multi-Target" problems.

Inspired by communication networks [6,10,11], we propose a novel learning network, Multi-Target Deep Neural Networks (MT-DNN), based on which we can construct scalable deep neural networks. In each *Source–Destination* pair, there is at least one learning path, through which the source data can be transformed into some kinds of values that can be used in the destination as the target. Some learning paths may be shared by different Source–Destination pairs where some *shared* features can be extracted by different target models. We theoretically prove the stabilities of

* Corresponding author.
E-mail addresses: zengz@i2r.a-star.edu.sg (Z. Zeng), liangny@i2r.a-star.edu.sg, nanying@gmail.com (N. Liang), yangx@ihpc.a-star.edu.sg (X. Yang), chhoi@smu.edu.sg (S. Hoi).

http://dx.doi.org/10.1016/j.neucom.2017.08.044
0925-2312/© 2017 Elsevier B.V. All rights reserved.

MT-DNN and prove that all the target learning paths can converge to their optimal solutions, respectively.

Based on the concept of MT-DNN, we design a novel model that has three branches above the main layers of GoogleNet [12]. Branch 1 is aiming at object classifications, branch 2 is used to calculate the size of the objects, and branch 3 is composed of $W \times H$ sub-models working as compound eyes of bees. The target of each sub-model is to figure out whether the corresponding pixel in the image belongs to the object or not. Unlike ImageNet that uses bounding boxes to indicate the locations and sizes of objects, we use masks with size of $W \times H$, where $mask[w, h] = 1$ if the point of $image[w,\ h]$ belongs to the target object and $mask[w, h] = 0$ otherwise. If we set $W = H = 256$, that means 65,536 sub-models have to be trained. We carry out rigorous experiments with respect to several influencing conditions and prove that our MT-DNN with multi-targets are stable and convergent, and can solve some problems that single-target models cannot do.

1.1. Our contributions

The specific contributions of this work are as follows: a). We propose the concept of MT-DNN; b). We theoretically demonstrate that multiple targets can converge respectively by using Stochastic Gradient Descent (SGD) [5,8,12]. Adjusting the learning rate of each target, we can roughly guarantee all the targets can converge synchronously; c). We present a study case to describe the proposed MT-DNN and design a single model with three different target models; d) We carry out series of experiments to examine the performance of the model and demonstrate that multiple targets can boost each other to achieve optimization solutions. It is the first time in the domain that a single model can identify objects, obtain their sizes, and point out their locations and shapes at the same time.

The rest of the paper is organized as follows. Section 2 discusses relevant research work. Section 3 illustrates the main definitions and theorems. In Section 4, we describe a study case for ease of understanding, and discuss the main target functions. In Section 5, we show the results of our experiments. We conclude our work and discuss some future work in Section 6.

2. Related work

The winner of ILSVRC 14 is “GoogleNet”, which is a 22 layers deep network [12]. The main hallmark of this architecture is the improved utilization of the computing resources inside the network by a carefully crafted design. Besides of the top layer that is a softmax function used to calculate the logarithm loss and the beginning of back-forwards learning, there are two more same branches below that carry out the same procedures in different lower layers. In this way, the depth and width of the network are increased while the computational budget is kept constant. This crafted design can help to converge faster, but contributes little to the final accuracy. Although there are three result outputs in GoogleNet, the outputs are aiming at the same target and hence, in our definition, it is still an end-to-end deep neural network.

The problem of object classification is the recognition of the class of an object belongs to. CNNs represent the-state-of-the-art approaches to address this problem. However, to solve this problem alone cannot fully fulfill the requirement in some other real applications. For example, it could be favorable to extraction the location and size information about the object in addition to its class information at the same time, which represents a multiple targets application scenario.

In order to achieve $f_1(x_2),\ f_2(x_2),\ \ldots,$ minimized (or maximized) at the same time, linear integration of multiple models into a single target optimization problem is applied [13,14], i.e., these functions can be added together with some trade-off parameters, λ, as: $minimized:\ \ f_{all} = \lambda_1 f_1(x_1) + \lambda_2 f_2(x_2) + \cdots$. One type of the method applied in deep neural networks is named as shared computation of convolutions that has been attracting increasing attention for efficient, yet accurate, visual recognition [9,15–19]. In [7], Andrew *et. al.* used linear combination to integrate sparsity and reconstruction models as the target optimization problem with $\lambda_1 = 1$ and $\lambda_2 = 0.1$. They trained their network to obtain 15.8% accuracy in recognizing 22,000 object categories from ImageNet, a leap of 70% relative improvement over the previous state-of-the-art. In [8], Ren et al. proposed Faster R-CNN model that can obtain very high object classification accuracy, while detect the positions of the objects with low errors. They define a loss function, $L_{cls}(p)$, where p is object's probability, for object classification, and a loss function, $L_{reg}(t)$, where t is Euclidean position of object, for position detection, respectively. Then, they obtain the optimization problem $L(p, t) = L_{cls}(p) + \lambda L_{reg}(t)$ with λ set to be 10, and hence, both $L_{cls}(p)$ and $L_{reg}(t)$ are roughly equally weighted. In ILSVRC and COCO 2015 competitions, the model is the foundation of the 1*st*-place winning entries in several tracks.

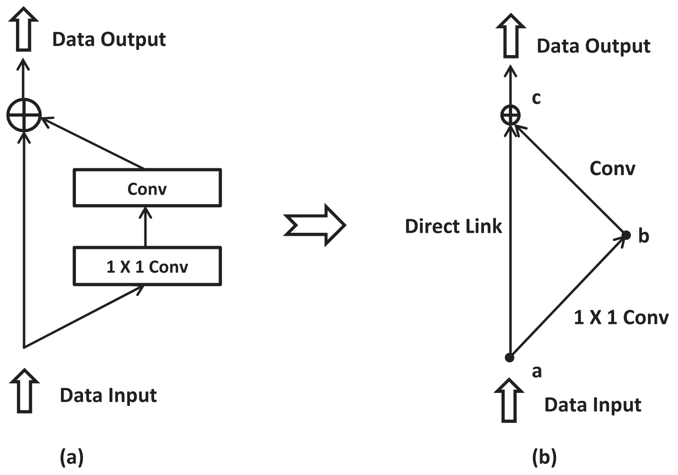

Fig. 1. Transformations of layers to learning links.

3. Concept of multi-target deep neural network

Layers of neural networks consist of neurons that are connected in pre-defined topologies and have one data input and one data output as shown in Fig. 1(a). Neural network layers can “learn” from batches of data by Stochastic Gradient Descent (SGD) functions and update their own parameters through back-forward in up-to-down fashion [5,16]. Referring to Fig. 1, we can observe that layer $1 \times 1\,Conv$, referred to as i, can be transformed to a link $(a \rightarrow b)$ or l_i and the function of the layer can be denoted as $out_i = f_{a \rightarrow b}(in_i)$ or $out_i = f_{l_i}(in_i)$, where in_i and out_i are layer i's data input and data output, respectively. Layer i can be transferred to link l_i with a mapping function f_{l_i} from $in_i \mapsto out_i$. When two links l_i and l_j need to be combined to a single link, we define two different combination functions: element-wise addition denoted as $l_i \oplus l_j \mapsto l_a$ and concatenation addition denoted as $l_i \otimes l_j \mapsto l_a$. For example, in Fig. 1(a), layer $1 \times 1\,Conv$ and layer $Conv$ can be transferred into learning link $(a \rightarrow b)$ and learning link $(b \rightarrow c)$, respectively as shown in Fig. 1(b).

AlphaGo has two deep neural networks: policy network and value network. We transfer the networks into two independent learning paths as shown in Fig. 2(a). Now, if we merge the two learning paths together, we can have many potential topologies. In Fig. 2(b), there are two main learning paths that are left one ($S \Rightarrow n \Rightarrow D_1$) and right one ($S \Rightarrow n \Rightarrow D_2$). We can observe that

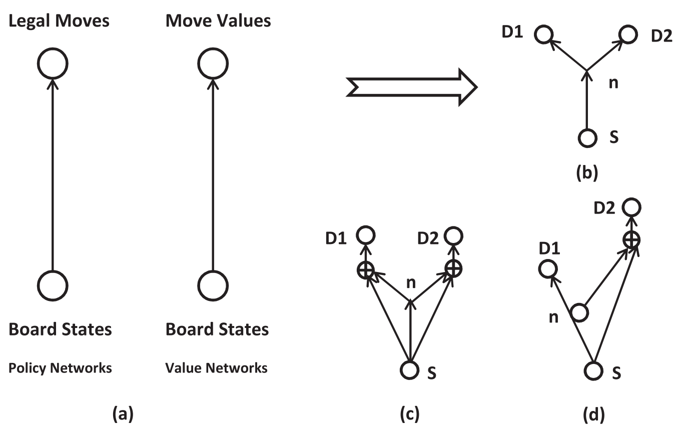

Fig. 2. Illustration of merging multiple learning paths. (a) We use policy network and value network as examples of two independent deep neural networks which share the same inputs. They are annotated as two independent learning paths. (b) A two-target learning network is derived after merging policy network and value network via their shared learning layers. (c) is an extension of (b) by supplying the input sources together with the shared features to the learning branches. (d) is a possible variance of (c) by removing the shared features from learning branch D1.

the learning path $(S \Rightarrow n)$ is shared by path $(S \Rightarrow D_1)$ and $(S \Rightarrow D_2)$. This topology has been used in [9,15–18] with the name of shared computation of convolutions. However, these learning paths are trained in alterative fashion. In this work, one of the learning paths is trained first and then, the training procedure stops to start training another learning path. The training phases will be repeated till some threshold achieved. Please note that in our proposal, different learning paths in the same network will be trained synchronously. Topology of Fig. 2(b) can be further extended to Fig. 2(c), in which there is a learning path $(S \Rightarrow n)$ shared by the two main learning paths. The shared learning path can learn the shared features belonging to both learning paths. Besides the shared learning path, the main learning paths have independent learning paths $(S \Rightarrow D_1)$ and $(S \Rightarrow D_2)$, respectively, that can discover the features unique to the main learning paths only. In Fig. 2(d), there is an interesting learning path $(n \Rightarrow D_2)$ whose source node is in the middle of another main learning path $(S \Rightarrow D_1)$. In this topology, main learning path $(S \Rightarrow D_2)$ can learn some features from $(S \Rightarrow D_1)$, but not vice versa. Without $(n \Rightarrow D_2)$, the two main learning paths are independent to each other.

The advantages of merging two learning paths together are twofold: First, by merging learning paths, the learning features are shared by different targets and thus it can reduce the feature computational cost; Secondly, the optimization solution is achieved by considering the constrains of all learning targets without the need of an explicit global loss function. In the following, we shall present some theoretical analysis of learning networks.

3.1. Theoretical analysis

At first, we present Theorem 1 that can guarantee the stability of learning paths that have some shared learning path(s).

Theorem 1. *Given the same data input S, if independent learning path $(S \Rightarrow D_1)$ is stable and can achieve optimal solution $f_1(S|\theta_1^*)$, and another independent learning path $(S \Rightarrow D_2)$ is stable and can achieve optimal solution $f_2(S|\theta_2^*)$, then the joint network of the two learning paths as shown in Fig. 2(b), is also stable.*

Proof. We assume that although the two learning paths are independently, they have some same structure of layers as in [5,9,15–18]. Then, we have the learning function of $(S \Rightarrow n)$ in Fig. 2(b) as $f_s(S|\theta_s)$, and the branches of the learning paths are $f_l(S_n|\theta_l)$ and $f_r(S_n|\theta_r)$, respectively, where S_n is the data output at node n. Obviously, $S_n = f_s(S|\theta_s)$ and hence, we obtain the function of independent learning path $(S \Rightarrow D_1)$ is $f_1(S|\theta_1) = f_l(f_s(S|\theta_s)|\theta_l)$ and the function of independent learning path $(S \Rightarrow D_2)$ is $f_2(S|\theta_2) = f_r(f_s(S|\theta_s)|\theta_r)$.

Since we assume that the independent learning paths $(S \Rightarrow D_1)$ and $(S \Rightarrow D_2)$ are stable and can achieve optimal solutions, the learning procedures by SGD are $\theta_l \leftarrow \theta_l - \eta_l \frac{\partial f_1}{\partial \theta_l}$ and $\theta_r \leftarrow \theta_r - \eta_r \frac{\partial f_2}{\partial \theta_r}$, respectively, where η_l and η_r are their learning rates and can be variant [5]. Hence, we obtain that $\frac{\partial f_1}{\partial \theta_l} \rightarrow 0$, $\frac{\partial f_2}{\partial \theta_r} \rightarrow 0$ and $\theta_l \rightarrow \theta_l^*$, $\theta_r \rightarrow \theta_r^*$. When the two learning paths are joint, the learning path $(S \Rightarrow D_1)$ has been divided into two parts: $(S \Rightarrow n)$ and $(n \Rightarrow D_1)$ and then, we denote θ_l^1 and θ_l^2 for these two parts, respectively, as the target parameters. Again, if we consider learning path $(S \Rightarrow D_2)$, we can obtain θ_r^1 and θ_r^2, for $(S \Rightarrow n)$ and $(n \Rightarrow D_2)$, respectively. For the left branch in Fig. 2(b), we can have $\theta_l^2 \leftarrow \theta_l^2 - \eta_l \frac{\partial f_1}{\theta_l^2}$ and for the right branch, we have $\theta_r^2 \leftarrow \theta_r^2 - \eta_r \frac{\partial f_2}{\theta_r^2}$. Since we assume that the independently learning path $(S \Rightarrow D_1)$ has the same structure of shared learning path $(S \Rightarrow D_1)$, and the same to the second learning path, we have $\theta_l^2 = \theta_l$ and $\theta_r^2 = \theta_r$. Then, we can obtain $\frac{\partial f_1}{\theta_l^2} \propto \frac{\partial f_1}{\theta_l} \rightarrow 0$ and $\frac{\partial f_2}{\theta_r^2} \propto \frac{\partial f_2}{\theta_r} \rightarrow 0$. Obviously, learning paths $(n \Rightarrow D_1)$ and $(n \Rightarrow D_2)$ are independent and stable, and they can learn without any interactivities.

Now we consider the shared learning path $(S \Rightarrow n)$. If we let one branch trained only, say left branch, we can have the learning procedure as $\theta_s \leftarrow \theta_s - \eta_l \frac{\partial f_1}{\partial \theta_s}$ and it is stable. For right branch, we also obtain $\theta_s \leftarrow \theta_s - \eta_r \frac{\partial f_2}{\partial \theta_s}$. If we train the two branches together, we shall have:

$$\theta_s \leftarrow \theta_s - \left(\eta_l \frac{\partial f_1}{\partial \theta_s} + \eta_r \frac{\partial f_2}{\partial \theta_s} \right). \tag{1}$$

From Eq. (1), we can observe that different learning paths update the shared paths independently and have no idea that there exists other learning paths updating the parameters too. In deep neural networks, there are dropout layers that can randomly select part of neurons to update and keep the rest unchanged [12]. Furthermore, there are ReLU layers that are sensitive to some output greater than some thresholds [20] and have no activities if the output are less than the thresholds. By using dropout layers, we can differentiate the neurons and make them sensitive to different features. By using ReLU layers, we can isolate some neurons that have no relations with the target models. Hence, we assume that during the back-forward procedures, in the shared learning path $(S \Rightarrow n)$ in Fig. 2(b), left branch updates θ_l^1 and right branch updates θ_r^1, respectively, instead of all of θ_s. Obviously, $\theta_l^1 \subset \theta_s$, $\theta_r^1 \subset \theta_s$ hold and learning procedures are $\theta_l^1 \leftarrow \theta_l^1 - \eta_l \frac{\partial f_1}{\partial \theta_l^1}$ and $\theta_r^1 \leftarrow \theta_r^1 - \eta_r \frac{\partial f_2}{\partial \theta_r^1}$. Then, Eq. (1) can be transformed to $\theta_s \leftarrow \theta_s - (\eta_l \frac{\partial f_1}{\partial \theta_l^1} + \eta_r \frac{\partial f_2}{\partial \theta_r^1})$. If $\theta_c = \theta_l^1 \cap \theta_r^1 \neq \emptyset$ holds, that means θ_c are sensitive to the features common to the target models. Hence, through shared learning paths, different learning paths can learn the features unique to themselves, discover the shared features together, and ignore the unrelated features.

Since the two independent learning paths are stable, we have $\frac{\partial f_1}{\partial \theta_s} \propto \frac{\partial f_1}{\partial \theta_l^1} \rightarrow 0$ and $\frac{\partial f_2}{\partial \theta_s} \propto \frac{\partial f_2}{\partial \theta_r^1} \rightarrow 0$. Hence, the joint network of two independent learning paths is stable too. □

Theorem 1 can guarantee that if two independent learning paths are stable and have some same layers, they can be merged to a single joint stable neural network.

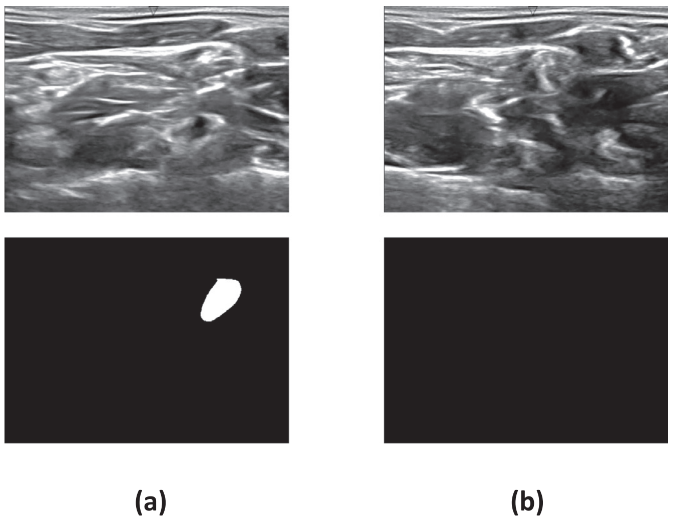

Fig. 3. Examples of ultrasound images and their corresponding mask labels. (a) an ultrasound images containing nerve structure (above) and its according nerve mask (below). (b) an ultrasound image containing no nerve structure (above) and its according nerve mask (below) showing no object was identified.

Theorem 2. *In a multi-target learning network, each independent single learning path i can achieve its optimal solution to the target function f_i.*

Proof. We assume, without loss of generality, a multi-target learning network consists of a learning path ($S \Rightarrow D_1$) with target function f_1 and a learning path ($S \Rightarrow D_2$) with target function f_2. These two learning paths share ($S \Rightarrow n$), as shown in Fig. 2(b).

Now, let's consider the left branch in Fig. 2(b). When the branch becomes stable, we have $\frac{\partial f_1(S|\theta_b)}{\partial \theta_b^*} = 0$ for learning path ($n \Rightarrow D_1$), and $\frac{\partial f_1(S|\theta_s)}{\partial \theta_s^*} = \frac{\partial f_1(S|\theta_l^1)}{\partial \theta_l^{1*}} = 0$ for learning path ($S \Rightarrow n$). For the right branch, we have the similar solution $\frac{\partial f_2(S|\theta_b)}{\partial \theta_b^*} = 0$ and $\frac{\partial f_2(S|\theta_s)}{\partial \theta_s^*} = \frac{\partial f_2(S|\theta_r^1)}{\partial \theta_r^{1*}} = 0$, where θ_l^1 and θ_r^1 are parameter sets belonging to left and right learning path, respectively. Hence, f_1 and f_2 achieve their own optimal solutions in this learning network without imposing any formality on their relationship. □

Theorem 2 can guarantee that true optimal solutions to multiple task-dependent loss functions are obtained, while the strategy of a simple linear combination of them $f_{all}(S|\theta) = \lambda_1 f_1(S|\theta) + \lambda_2 f_2(S|\theta)$ doesn't guarantee this. This makes MT-DNN more significant than multi-task models. In the proposed MT-DNN, different learning paths may have different learning rates η_i. These learning rates can control the convergent rates of the corresponding learning paths and in the meanwhile, deliberately adjusted η_i can avoid the situations that some learning paths converge too fast, while some others learn too slowly.

4. Case study

We are interested in a competition "Ultrasound Nerve Segmentation" on Kaggle [21], in which, there are thousands of ultrasound images of patients provided with the mask labels created by experienced doctors. Examples with and without nerve in an ultrasound image are given in Fig. 3. Fig. 3(a) shows an ultrasound image containing nerve structure (above) and its according nerve mask which indicate the position and the shape of the nerve structure is provided (below). Fig. 3(b) shows an ultrasound image without any identified nerve structures. Then, we have the problem statement: Given an ultrasound image, the proposed models could generate a mask image as output. If there is no nerve structure found in the image, a mask with all black should be provided. Otherwise, the position and the shape of the nerve structure should be labeled as white pixels in the mask image. In the dataset, the images are all with the size of 580×420 pixels in TIFF format. In the training dataset, there are 5635 patients' images with corresponding masks provided, among which there are 2323 images with positive masks and 3312 images are normal.

We choose this dataset used in our experiments due to the following reasons: a) There is no research work on shape detection in ultrasound images by using deep neural networks and it is challenging; b) This problem can be easily transformed into multi-target problem; c) This problem is a classic problem and the research work can be extended to many other domains, such as X-ray detections in hospitals, fault detections in industry, etc.

4.1. Training dataset preparations

In the dataset provided, the images are all black-white and have only one channel. Color image files have (*r, g, b*) three channels [22]. We use the TIFF data as *g* channel, and set *r* channel to be 0 and *b* channel to be 255. By adding extra channels into the images, we can have color images as shown in Fig. 8(a). In [5], the authors also added constant planes with 0 and 1 into their data source as independent channels.

Through analysis, we find that although the original images are 580×420, the nerve structures' sizes are all within 256×256 and then, we set the size of our scanner to be 256×256, in order to minimize the computational requirement. By using rotating and cropping, we obtain millions of small images and attempt to avoid the notorious overfitting [9].

4.2. Implementation of our proposed neural networks

Unlike faster R-CNN in [8] that proposed thousands of region anchors in a single image, we focus on the design of a fixed size scanner that can be used to scan much bigger images, in order find the potential objects, their locations and shapes. By using the scanner, we can know in the scanned part of the images, whether there is an object. If yes, what the size of the object is, and what the shape of the object is. Compared with the work in [8], our solution is more scalable and much easier for implementations. Furthermore, our model can predict the shapes of objects, while the work only provide bounding boxes that may cover the objects.

In the scanner, there is a learning network which has one data input and three target models. The first target is to identify whether there is nerve structure or not in the images. This target is the same as ImageNet and can be considered as classification problem. The second one is to tell the sizes of the nerve structures or the ratio of the number of pixels belonging to objects and the number of pixels that have been scanned. This goal can be considered as regression problem. The third target is to present the masks with pixels labeled as 0 or 1 and we can consider this one as classification problem too. The three different targets have different models as shown in the following.

Model of Target 1: For object classification, we adopt the widely used *Log-likelihood Loss* function [8,20,23–25]. This is equivalent to maximizing the likelihood of the data set $\mathcal{D}$ under the model parameterized by θ. We have the likelihood $\mathcal{L}$ and the loss of Target 1, $\mathcal{T}_1$, as follows:

$$\mathcal{T}_1(\mathcal{D}|\theta) = -\mathcal{L}(\mathcal{D}|\theta) = -\sum_{i=0}^{|\mathcal{D}|} \log(P(Y = y^{(i)}|x^{(i)}), \theta), \qquad (2)$$

where normally $\theta = \{W, b\}$ [1,8,24,25].

Model of Target 2: This target is to estimate the size of object in the range of [0, 1]. If there is no object, the value of size $s = 0$. If

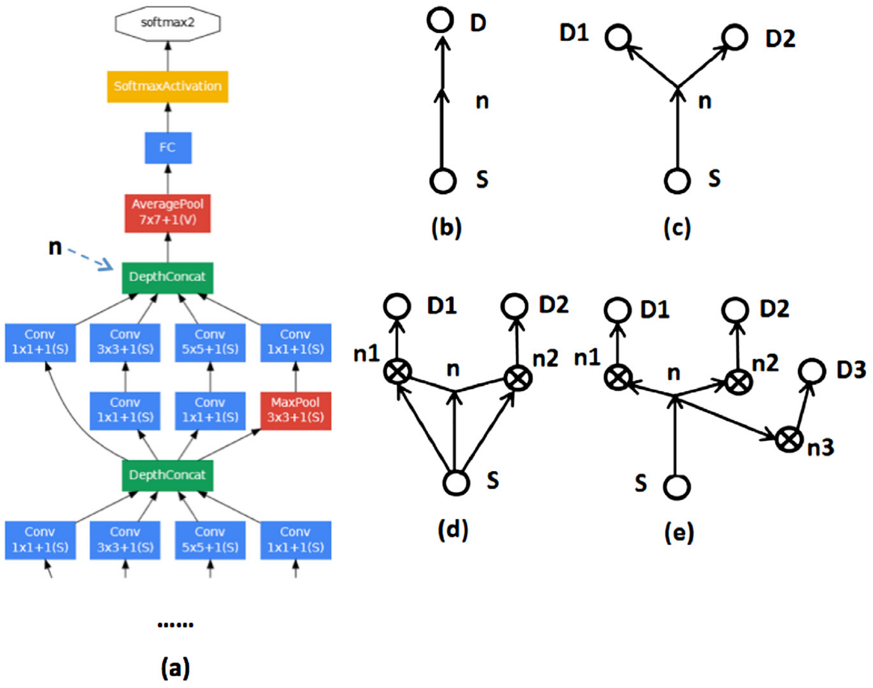

Fig. 4. MT-DNNs based on GoogleNet. (a) The GoogleNet model. (b) GoogleNet model is annotated as learning paths. (c) A two-target learning network is derived by adding a second learning path to (b) model. (d) is a variant of (c) by adding input source paths to the learning branches. (e) A three-target learning network is derived by adding a third learning path to (c) model.

there is only an object with black background, the value is $s = 1$. If there is an object with some background (noise), the value is in the range of (0, 1). We use mean squared error (MSE) [5,26] as the loss function. Then we have the loss of Target 2, $\mathcal{T}_2$, as follows:

$$\mathcal{T}_2(\mathcal{D}|\theta) = \frac{1}{|\mathcal{D}|}\sum_{i=0}^{|\mathcal{D}|}(\hat{s}^{(i)} - s^{(i)})^2, \tag{3}$$

where $\hat{s}$ is the predicted value with given θ.

Model of Target 3: This target is to predict the shape of the object. In a TIFF image file, if the pixel's value is 0, the point is black. If the value is 255, it is white. We normalize pixels' values to 0 and 1 in the mask images. We assume the shape of our scanner is (W_s, H_s), then we have $W_s \times H_s$-dimension vector with value of 0 or 1. We use Sigmoid Cross Entropy Loss function as the target function [27]. Then, we obtain the loss of Target 3, $\mathcal{T}_3$, in the following:

$$\mathcal{T}_3(\mathcal{D}|\theta) = \frac{1}{|\mathcal{D}|}\sum_{i=0}^{|\mathcal{D}|}[p^{(i)}\log\hat{p}^{(i)} + (1 - p^{(i)}\log(1 - \hat{p}^{(i)})], \tag{4}$$

where $\hat{p}^{(i)}$ is the estimated probability that the target pixel i in the matrix is 1.

4.3. Design of MT-DNN

Convolutional Neural Networks (CNN) and Full-Connected Networks (FC) are two main layers of deep neural networks. Along with CNN, there are other layers, such as pooling (max, average, etc.) [28], ReLU, which can help to discovery significant features. Normally, multiple layers of FCs are used, in order to select the features that are useful to the final results. Hence, no matter how deep or how complicated a neural network may be, we functionally divide the network into two learning paths: ($S \Rightarrow n$) that mainly extract and discover the features and ($n \Rightarrow D$) that mainly select from available features to minimize the target loss models. So, GoogleNet can be transformed from Fig. 4(a) to (b), while at node n, GoogleNet's learning path has been divided into two connected learning paths. As we mentioned before, Target 1 is the same to GoogleNet, we set D_1 in Fig. 4(c) is equal to D in (b). From n, we have a new branch ($n \Rightarrow D_2$) that is a duplication of ($n \Rightarrow D$) with D replaced by D_2, Target 2. Then, we have the learning network with Target 1 and Target 2 as shown in Fig. 4(c).

We may consider that besides the shared features learned through path ($S \Rightarrow n$), each target shall learn the features that are belonging to their models only, we can add an independently learning path to each target as we discussed above. As shown in Fig. 4(d), based on Fig. 4(c) we add extra paths ($S \Rightarrow n_1$) and ($S \Rightarrow n_2$) for Target 1 and 2, respectively, which are duplications of path ($S \Rightarrow n$) in Fig. 4(b). Here we use lines without arrow to denote the links that can pass data only.

In order to improve the performance of each learning path, we can design the layers of individual learning path, instead of duplicating other learning path in this case. Then, the topologies of learning networks will be more complicated and it is worth our research in the near future.

5. Experimental results

In this section, we will carry out serials of experiments to evaluate our proposed MT-DNN. The models in the Experiment-1 and Experiment-2 are trained from scratch and the weights of parameters are initiated randomly. The main shared structure ($S \Rightarrow n$) is based on GoogleNet. We chose the parameter for our proposed algorithm the same as those used by GoogLeNet: momentum = 0.9, weight decay = 0.0002, learning rate = 0.01, and batch size = 32. In Experiment-1, we slightly modify GoogleNet with two more additional layers and name this new model as GoogleNet Learning Net (GLN) that is with the two first branches in the following list. In Experiment-2, we further merge GLN with Auto-Encoder [29] and name this new model as GLN_{auto} that have three main learning branches as:

1. Branch 1: Classification, the same to GoogleNets final structure.
2. Branch 2: Regression, three layers of fully-connected neural networks.

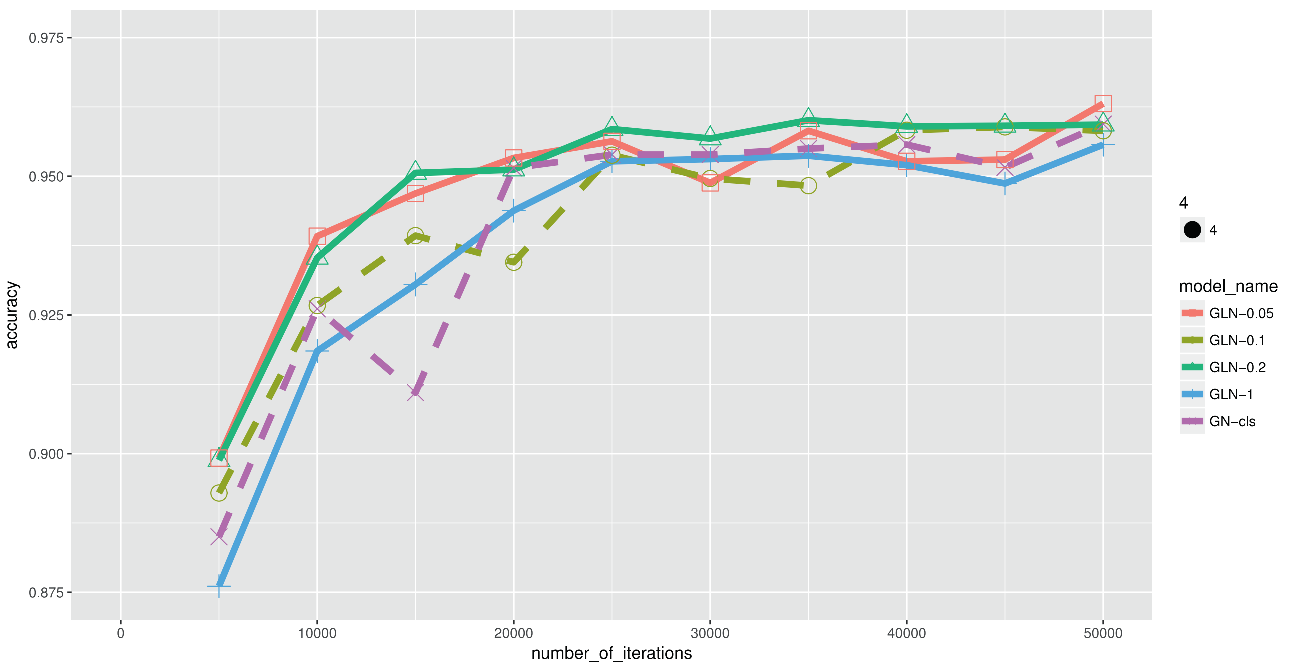

Fig. 5. Comparisons of D_1 branch of our modified *GLN* models with GN_{cls}, w.r.t. accuracy.

3. Branch 3: Mask, five groups of deconvolution CNN to remap features back to an output of the same size as original input images.

5.1. Experiment settings

We comprehensively evaluate our methods on a server equipped with Dual 8-Core Intel@Xeon Processors 2.4 GHz, 128 GB memory, 4 × 3TB Enterprise SATA3 hard disk, and 4 × nVidia Titan X 12GB GDDR5 GPU cards. The OS is Ubuntu 14.04 with *Cuda, cudnnlib, Caffe* [30], installed. The learning rate is 0.01 throughout of our experiments.

5.2. Experiment-1

Based on Fig. 4(a) and (b), we add a new average pooling layer and full-connected layer above node *n*. In Fig. 4(c), the number of output of D_1 branch is 2 that are probabilities of two classifications, respectively. The number of output of D_2 branch is 1 that is the size of object in the range of [0, 1]. We also change GoogleNet as the learning path ($S \Rightarrow D_1$) which has two outputs and uses log-likelihood loss as the loss function (2). We denote this modified GoogleNet as GN_{cls}. In order to compare the performance of $\mathcal{T}_2$, we design another model based on GoogleNet too that has one output and uses loss function (3) of MSE as learning path ($S \Rightarrow D_2$). This model is referred to as GN_{rgs}.

In Fig. 4(c), for D_1, we use (2) as the target loss function, and for D_2, we use (3) as the target loss function. We fix $\lambda_1 = 1$ and variate λ_2 to be 0.05, 0.1, 0.2, 1, and these models are denoted as $GLN_{0.05}, GLN_{0.1}, GLN_{0.2}, GLN_1$, respectively. We compare the results of D_1 branch with GN_{cls} and present the results of D_2 branch of $GLN_{0.05}$ under comparisons of GN_{rgs}. Please note that GN_{cls} and GN_{rgs} are two different models and have to be trained independently. It takes around 8 h to train each model under consideration for 50,400 training iterations.

Now, we carry out experiments on models $GLN_{0.1}, GLN_{0.2}$, and GLN_1, in order to examine the effects of weights on the performance of multiple learning paths. In these experiments, we record the testing results of the models very 5000 iterations by using validation dataset. In Fig. 5, we show the accuracy of the models on the tests. In Fig. 6, we present the MSE of *GLN* models and that of GN_{cls} in the experiments. From these results, we can observe that the branches of our *GLN* models with variant weights can both converge to their own optimal solutions synchronously. To our surprise, till 50,400 training iterations, D_1 branch and D_2 branch of $GLN_{0.2}$ perform even better than GN_{cls} and GN_{rgs}, respectively. It seems that when $\lambda = 0.2$, the two learning paths of *GLN* can converge with nearly the same rates and hence, they can help each other learning faster. After some training iterations, we can expect that all the models can achieve their optimal solutions eventually.

Practically, these experiments prove our Theorem 1 and Theorem 2: our proposed MT-DNN with multiple targets are stable and different targets can converge to their optimal solutions independently. In the following, we shall combine GoogleNet with Auto-Encoder and construct more complicated learning networks with three different targets of (2), (3), and (4).

5.3. Experiment-2

In the previous experiments, we can observe that D_1 branch of our model can tell us what kinds of the objects are and D_2 branch can figure out the sizes of the objects. In the following experiments, we attempt to draw the shapes of objects. In order to obtain the masks of objects in the images, based on learning network as shown in Fig. 4(c), we add a new learning path ($n \Rightarrow D_3$) from node *n* and obtain the learning network in Fig. 4(e). The function of learning path ($S \Rightarrow D_3$) can be described as: given an image, the learning path shall tell us whether the target pixel in the image belongs to the target object or not.

In our experiments, we set $W_s = H_s = 256$ and then, we have $256 \times 256 = 65{,}536$ pixels under consideration. We observe that at point *n* in GoogleNet, the images have been transformed into 1024 small matrixes with the size of 8×8. It is worth noting that these matrixes are extracted features and have no information on positions. Based on these features, D_1 and D_2 branches can figure out the objects' classifications and their sizes respectively, but cannot tell us the locations of the objects within the images. In order to detect target objects, numerous research works have been carried

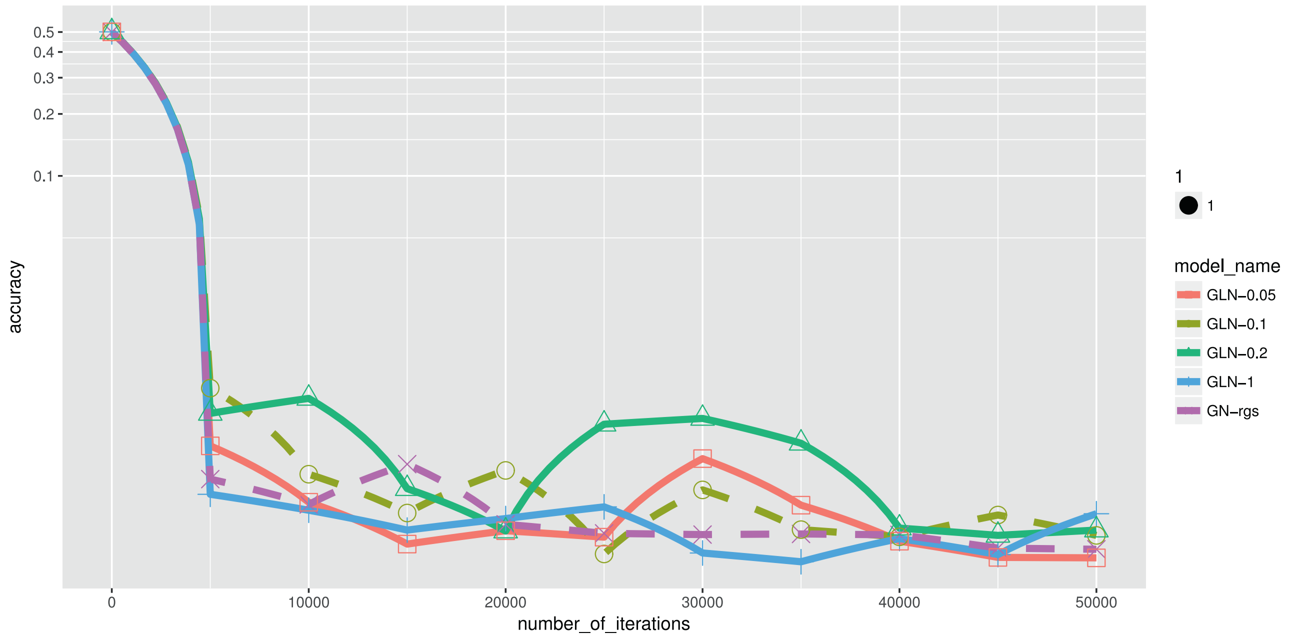

Fig. 6. Comparisons of D_2 branch of our modified *GLN* models with GN_{rgs}, w.r.t. MSE.

out in both academia and industry [8,9]. The discussion on this part is beyond the scope of this work that focuses on efficiency and convergency of MT-DNN only. We will explore the performance of $(S \Rightarrow D_3)$ and compare our methods with other famous algorithms in our future work.

Based on the features extracted by D_1 and D_2 branches, we add a new learning path with the hypothesis that the features can be mapped into the channels and the numbers of sequential channels can present the positions of the features extracted from the images. For example, in MNIST competitions, the images of size 32×32 can be reshaped into $C \times H \times W$ as $1 \times 32 \times 32$, where C is the number of channel. By using Auto-Encoder [29], we can transform the pixels into channels as $724 \times 1 \times 1$ where channel 1 presents pixel in position (1, 1) of the original image. In this way, we can obtain the features and we also can reserve the positions based on channel numbers.

From node n in Fig. 4(c), we add 3 more convolutional neural network layers following by ReLU and pooling layers. These layers transform the features at node n from $1,024 \times 8 \times 8$ to $16,384 \times 1 \times 1$. In this way, all features have been mapped into channels and the channel numbers can be mapped into the pixel positions in the images. Following these layers, there are 5 full connected neural network layers working as Auto-Encoder that map the 16,384 channels into 65,536 channels. In the last layer, we reshape the channels into $1 \times 256 \times 256$ and reconstruct the channels into image matrix. We compare the outputs with the mask labels and use (4) as the loss function. In this model, each pixel is handled by an independent sub-model that can judge whether the pixel belongs to the target object or not. All the sub-models can be trained synchronously.

Again, we extract the learning path $(S \Rightarrow D_3)$ from the model and construct an independent model as the benchmark model, which is referred to as GN_{auto}. We present the results in Fig. 7. From this figure, we can observe that the loss of GN_{auto} with single target drops from 0.7 to 0.17 from iteration 0 to the first validation, but after that the loss is vibrating with very small variance (less than 5×10^{-4}). Obviously, the model learns nothing from the training procedures. On the contrary, D_3 branch of our learning network converges from 0.7 to 0.12 step by step. We can antici-

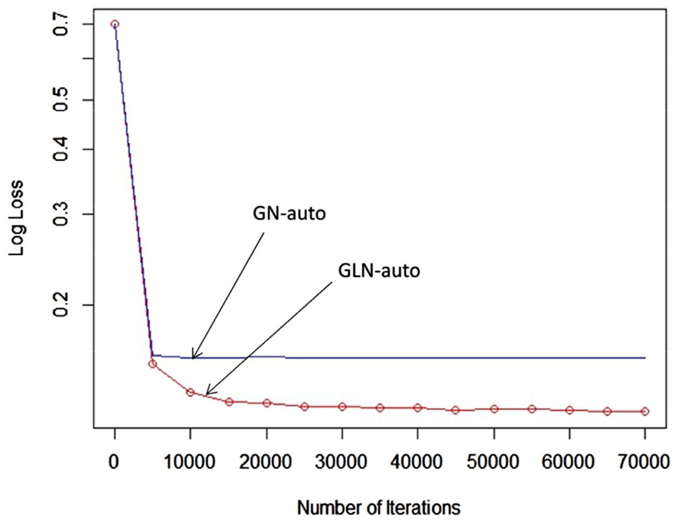

Fig. 7. Comparisons of D_3 branch of our GLN_{auto} models with GN_{auto}, w.r.t. log-likelihood loss.

pate that with more iterations, our model can converge to some optimal point and the system is stable.

Unlike the previous experiments, without D_1 and D_2 branches, the learning path $(S \Rightarrow D_3)$ performs very poorly. It is reasonable, since $(S \Rightarrow D_1)$ and $(S \Rightarrow D_2)$ can help extracting the features through the learning path $(S \Rightarrow n)$. Then, the following full-connected layers can find the patterns based on the features that exist in different channels. A good classifier shall discover the same feature-based patterns no matter of the objects' positions, where there are some cropped images from one original image with the corresponding mask. When we slide the scanning window on the original images, we found the features slide among the channels too. Hence, if the features slide among the channels, we also know that the object slide in the image too. Based on the sliding features among channels, we can determine the position of target object. GN_{auto} has no features of target object that would have been ob-

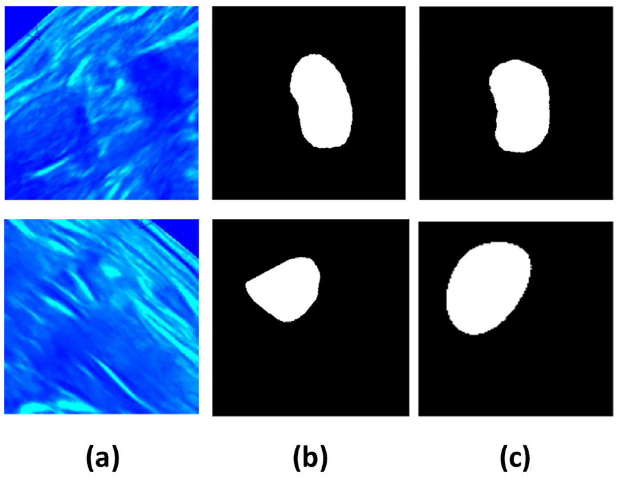

Fig. 8. The prediction of D_3 branch of our proposed learning network GLN_{auto}. (a) Two examples of the input ultrasound images; (b) The according ground-true mask labels for the input images; (c) The predicted mask of D_3 branch of our GLN_{auto} model.

tained by learning paths ($n \Rightarrow D_1$) and ($n \Rightarrow D_2$), without which GN_{auto} cannot learn the positions of target objects.

In Fig. 8, we present some examples of the predicted masks of D_3 branch, where Fig. 8(a) are input images and (b) are label masks that can indicate the object's shape and location within the image, and (c) are the output masks of D_3 branch of the model. From these figures, we can observe that (c) are very similar to (b) and our hypothesis works!

6. Conclusions and future work

In this work we have proposed novel multi-target deep neural networks, referred to as MT-DNN. Unlike multi-task methods or shared computation of convolutions in the literature that have single target model only, MT-DNN can handle several different targets at the same time. Based on GoogleNet we design a single model with three different targets, one is for classification that can tell whether there is a nerve structure inside or not, one is for regression that can figure out what the size of the nerve structure is, and the rest one is for masks that is composed of 65,536 sub-models. For each pixel in the image with the size of 256×256, there is a corresponding sub-model that can just figure out whether the pixel belongs to the nerve structure or not. Furthermore, we find that without the help of target one and two, target three cannot converge. That means the proposed MT-DNN can solve some problems that single-target model cannot achieve.

In our experiments, we noticed that when multiple branches are combined together, the models may become too big to be supported by single-GPU based computing platforms and novel divide-and-conquer distributed computing methods for deep learning should be proposed. In our future work, we plan to extend MT-DNN for processing 3D images based on 3D-CNN neural network architectures for biomedical image applications. Furthermore, MT-DNN can also be applied to industry domains by providing AI-based prognosis and diagnosis solutions for vehicle/aircraft maintenance.

Acknowledgment

The work was supported by Singapore-China NRF-NSFC Grant [No. NRF2016NRF-NSFC001-111]

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (2015) 211–252.
[2] G. Dahl, D. Yu, L. Deng, A. Acero, Context-dependent pre-trained deep neural networks for large vocabulary speech recognition, IEEE Trans. Audio, Speech, Lang. Process. (2012) 30–42.
[3] Y. Bengio, R. Ducharme, P. Vincent, C. Jauvin, A neural probabilistic language model, J. Mach. Learn. Res. 3 (2003) 1137–1155.
[4] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.
[5] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of go with deep neural networks and tree search, Nature 529 (2016) 484–489.
[6] Z. Zeng, V. Bharadwaj, Design and performance evaluation of queue-and-rate-adjustment dynamic load balancing policies for distributed networks, IEEE Trans. Comput. 55 (11) (2006) 1410–1422.
[7] Q.V. Le, Building high-level features using large scale unsupervised learning, in: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, 2013, pp. 8595–8598, doi:10.1109/ICASSP.2013.6639343.
[8] S. Ren, K. He, R. Girshick, J. Sun, Faster r-cnn: towards real-time object detection with region proposal networks, IEEE Trans. Pattern Anal. Mach. Intell. PP (99) (2016). 1–1
[9] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. Lecun, Overfeat: integrated recognition, localization and detection using convolutional networks, CoRR (2013). http://arxiv.org/abs/1312.6229.
[10] D. Bertsekas, R. Gallager, Data Networks (2Nd Ed.), Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1992.
[11] Z. Zeng, V. Bharadwaj, Design and analysis of a non-preemptive decentralized load balancing algorithm for multi-class jobs in distributed networks, Comput. Commun. 27 (2004) 679–694.
[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 1–9.
[13] M. Avriel, Nonlinear Programming: Analysis and Methods, Prentice-Hall, Englewood Cliffs, NJ, 1976.
[14] D.P. Bertsekas, Nonlinear Programming, Belmont, Mass. Athena Scientific, 1995. Lccopycat
[15] K. He, X. Zhang, S. Ren, J. Sun, Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition, Springer International Publishing, Cham, pp. 346–361.
[16] R. Girshick, Fast r-cnn, in: 2015 IEEE International Conference on Computer Vision (ICCV), 2015, pp. 1440–1448.
[17] E. Shelhamer, J. Long, T. Darrell, Fully convolutional networks for semantic segmentation, IEEE Trans. Pattern Anal. Mach. Intell. PP (99) (2016). 1–1
[18] J. Dai, K. He, J. Sun, Convolutional feature masking for joint object and stuff segmentation, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 3992–4000.
[19] D. Eigen, R. Fergus, Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture, in: The IEEE International Conference on Computer Vision (ICCV), 2015, pp. 2650–2658.
[20] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: F. Pereira, C.J.C. Burges, L. Bottou, K.Q. Weinberger (Eds.), Advances in Neural Information Processing Systems 25, Curran Associates, Inc., 2012, pp. 1097–1105.
[21] K. Competitions, Ultrasound nerve segmentation, 2016. https://www.kaggle.com/c/ultrasound-nerve-segmentation.
[22] P. Dollar, Z. Tu, P. Perona, S. Belongie, Integral channel features, in: Proceedings of the British Machine Vision Conference, BMVA Press, 2009, pp. 91.1–91.11. Doi:10.5244/C.23.91
[23] B. Settles, Active Learning Literature Survey, Computer Sciences Technical Report, University of Wisconsin–Madison, 2009.
[24] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, CoRR (2014). http://arxiv.org/abs/1409.1556.
[25] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770–778, doi:10.1109/CVPR.2016.90.
[26] Y. Ephraim, D. Malah, Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator, IEEE Trans. Acoust. 32 (1984) 1109–1121.
[27] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, Pierre-Antoine Manzagol, Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion, J. Mach. Learn. Res. 11 (2010) 3371–3408.
[28] K. Jarrett, K. Kavukcuoglu, M. Ranzato, Y. LeCun, What is the best multi-stage architecture for object recognition? in: ICCV, IEEE, 2009, pp. 2146–2153.
[29] L. Deng, M. Seltzer, D. Yu, A. Acero, A.-r. Mohamed, G. Hinton, Binary coding of speech spectrograms using a deep auto-encoder, in: Interspeech 2010, International Speech Communication Association, 2010.

[30] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R.B. Girshick, S. Guadarrama, T. Darrell, Caffe: convolutional architecture for fast feature embedding, CoRR (2014). http://arxiv.org/abs/1408.5093.

Dr. Zeng Zeng received the Ph.D. degree in electrical and computer engineering from the National University of Singapore, Singapore, in 2005, and the B.S. and M.S. degrees in automatic control from the Huazhong University of Science and Technology, Wuhan, China, in 1997 and 2000, respectively. Currently, he works as a Scientist III in Data Analytics Department, I2R, A*Star, Singapore. From 2011 to 2014, he worked as a Senior Research Fellow with the National University of Singapore, and he was the Founder of GoGoWise Cloud Education Pte. Ltd., Singapore. From 2005 to 2011, he worked as an Associate Professor in Computer and Communication School, Hunan University, China. From 2008 to 2011, he was a Senior Engineer and Senior Consultant of CSR Zhuzhou Institute Co. Ltd, Hunan, China. In the meanwhile, he was a Senior member of IEC High Speed Train Group and participated in the draft proposals of IEC-61375, IEC-62580, etc. His research interests include distributed/parallel computing systems, data stream analysis, deep learning, multimedia storage systems, wireless sensor networks, onboard fault diagnosis and fault pre-alerting, and controller area networks.

Dr. Nanying Liang received the Ph.D. degree in Electrical and Electronic Engineering from Nanyang Technological University, Singapore in 2007, and B.E. degree in Biomedical Engineering from Jilin University, China in 2002. She is a scientist in Data Analytics Department, Institute for Infocomm Research, A*STAR, Singapore. From 2007–2009, she worked as a post-doctor in INRIA-LORIA, France and in VTT, Finland. Her research interests are in Neural Network, Deep Learning, Bioinformatics, Biomedical Image Analysis and Healthcare Data Analytics.

Dr. Yang Xulei (IEEE Senior Member) is a research scientist from Institute of High Performance Computing, A*STAR, Singpaore, with many years of research experiences in image analysis and machine learning. His current research interests focus on deep learning for biomedical image analysis. He has published more than 50 scientific papers and international patents.

Dr. Steven Hoi is an Associate Professor in the School of Information Systems (SIS), Singapore Management University (SMU), Singapore. Prior to joining SMU, he was a tenured Associate Professor at the School of Computer Engineering of the Nanyang Technological University (NTU), Singapore. He received his Bachelor degree from Tsinghua University, and his Master and Ph.D degrees from the Chinese University of Hong Kong. His research interests include large-scale machine learning (online learning and deep learning) with application to tackling big data analytics challenges across a wide range of real-world applications, including multimedia retrieval, social media, web search and information retrieval, computer vision and pattern recognition, computational finance, cyber security, mobile and software data mining, etc. He has published over 150 papers in premier conferences and journals, and served as an organizer, area chair, senior PC, TPC member, editors, and referee for many top conferences and premier journals. He is the recipient of the Lee Kong Chian Fellowship Award due to his research excellence. Currently he is the Editor in Chief of Neurocomputing, a premier journal for neural networks & deep learning.