

Cost-Sensitive Deep Learning with Layer-Wise Cost Estimation

Yu-An Chung and Hsuan-Tien Lin

Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan

b01902040@ntu.edu.tw, htlin@csie.ntu.edu.tw

Abstract

While deep neural networks have succeeded in several visual applications, such as object recognition, detection, and localization, by reaching very high classification accuracies, it is important to note that many real-world applications demand varying costs for different types of misclassification errors, thus requiring cost-sensitive classification algorithms. Current models of deep neural networks for cost-sensitive classification are restricted to some specific network structures and limited depth. In this paper, we propose a novel framework that can be applied to deep neural networks with any structure to facilitate their learning of meaningful representations for cost-sensitive classification problems. Furthermore, the framework allows end-to-end training of deeper networks directly. The framework is designed by augmenting auxiliary neurons to the output of each hidden layer for layer-wise cost estimation, and including the total estimation loss within the optimization objective. Experimental results on public benchmark visual data sets with two cost information settings demonstrate that the proposed framework outperforms state-of-the-art cost-sensitive deep learning models.

1. Introduction

Deep learning has shown great success on a broad range of visual applications such as object recognition [1, 2, 3], detection [4, 5, 6], localization [7, 8], and video classification [9, 10, 11]. Problems in such applications belong to a large class of regular classification in which the main evaluation metric is accuracy, implying each type of misclassification error is penalized equally.

Nevertheless, using accuracy as the evaluation metric for learning does not always produce the most useful classification system in the real world. In fact, many real-world applications [12, 13, 14, 15, 16], including vision related, demand varying costs for different types of misclassification errors. For example, different costs are useful for building a realistic face recognition system [15, 17, 18, 19], in which a government staff being misrecognized as an impostor causes only a slight inconvenience; however, an imposter misrecognized as a staff can result in serious damage. Even in a simple digit recognition task, varying costs can be helpful in representing the nature of the task, as it is common and understandable to classify an ill-written 7 as 1 but classifying a 7 as a 4 would be laughable. Such real-world applications call for cost-sensitive classification algorithms, which aim to identify the best classifier under the application-demanded costs.

Much research effort has been made to study cost-sensitive classification algorithms. In [20, 21, 22], the researchers proposed to equip probabilistic classifiers with Bayes decision theory to enable the classifiers to consider the cost information

during prediction. Some other studies extended existing cost-insensitive classification algorithms to be cost-sensitive, such as support vector machine [23] or neural network [20, 24]. Recently, as deep neural networks (DNN) have become state-of-the-art on a broad range of machine learning applications [25, 26, 2, 27, 3], researchers are attempting to incorporate cost information into training DNN [28].

One successful DNN for cost-sensitive classification, called Cost-Sensitive DNN (CSDNN), has been recently proposed in [28]. The training process of CSDNN consists of two steps. The first step is to initialize the DNN by layer-wise pretraining using a cost-sensitive variant of the conventional auto-encoder [29]. The second step involves the fine-tuning of the DNN with a cost-sensitive loss function. The final CSDNN is thus cost-sensitive in both pretraining and training stages, and is shown to be a state-of-the-art algorithm that outperforms other existing cost-sensitive classification algorithms and some deep learning alternatives.

While CSDNN is state-of-the-art, its design is based on the conventional fully-connected DNN with sigmoid activation functions and experiences two issues. First, the design restricts the applicability to more modern structures such as convolutional [30, 31] and pooling layers. Second, the sigmoid activation function suffers from the problem of diminishing gradients when the network deepens, even after careful pretraining.

In this paper, we resolve these issues by proposing a novel framework for cost-sensitive deep learning. To build a cost-sensitive DNN for a K -class cost-sensitive classification problem, the proposed framework replaces the layer-wise pretraining step with layer-wise cost estimation, in which K additional neurons are added to the output of each hidden layer. These K additional neurons serve as auxiliary units that help the DNN learn meaningful representations towards estimating the costs in each layer. Experiments conducted on four benchmark visual data sets and two cost information settings validate that the proposed framework outperforms CSDNN. Furthermore, we show that the proposed framework can be easily and effectively attached to deep neural networks with ReLU [32] activation functions or convolutional neural networks like AlexNet [31], demonstrating that the framework provides a more general end-to-end solution for cost-sensitive deep learning than CSDNN. The benefits of performance and generality make the proposed framework a favorable choice in practice.

The remainder of the paper is organized as follows. In Section 2, we formally define the cost-sensitive classification problem and introduce related works. Then, we illustrate our proposed framework in Section 3, and validate the framework with vision-related data sets in Section 4. Finally, we conclude in Section 5.

2. Preliminary

We start by formalizing the regular (cost-insensitive) classification problem and extend it to the cost-sensitive setting in Section 2.1. We then introduce some important deep learning research conducted to tackle cost-sensitive classification in Section 2.2.

2.1. Cost-Sensitive Classification

In a K -class regular classification problem, a size- N training set $S = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$ is given, where each input vector \mathbf{x}_n is within an input space $\mathcal{X} \subseteq \mathbb{R}^D$, and each label y_n is within a label space $\mathcal{Y} = \{1, 2, \dots, K\}$. Regular classification aims at using S to train a classifier $g: \mathcal{X} \rightarrow \mathcal{Y}$ such that the expected error $\mathbb{I}[y \neq g(\mathbf{x})]$ on the test examples (\mathbf{x}, y) is small.¹ That is, each type of misclassification error is charged with the same penalty.

Cost-sensitive classification extends regular classification by penalizing each type of misclassification error differently according to some given costs. We consider a general cost-vector setting [20, 23] when designing the proposed framework. The cost-vector setting represents the cost information by coupling an additional cost vector $\mathbf{c} \in [0, \infty)^K$ with each example (\mathbf{x}, y) , where the k -th component $\mathbf{c}[k]$ of the cost vector \mathbf{c} denotes the cost of predicting \mathbf{x} as class k , and naturally $\mathbf{c}[y] = 0$. Consider a cost-sensitive training set $S_c = \{(\mathbf{x}_n, y_n, \mathbf{c}_n)\}_{n=1}^N$, cost-sensitive classification aims at using S_c to train a classifier $g_c: \mathcal{X} \rightarrow \mathcal{Y}$ such that the expected cost $\mathbf{c}[g_c(\mathbf{x})]$ on the test examples $(\mathbf{x}, y, \mathbf{c})$ is small.

A special case of the cost-vector setting is the cost-matrix setting, where the cost information is encoded by a $K \times K$ cost matrix \mathbf{C} and each entry $\mathbf{C}(y, k) \in [0, \infty)$ indicates the cost for predicting a class- y example as class k . The information within the cost matrix can be simply cast as the cost vectors by defining the cost vector in $(\mathbf{x}, y, \mathbf{c})$ as the y -th row of the cost matrix \mathbf{C} . The cost-matrix setting, albeit less general, allows real-world applications to specify their demanded costs more easily. We follow many earlier cost-sensitive classification works [20, 21, 33, 23, 28] to take the cost-matrix setting when conducting benchmark experiments.

2.2. Deep Learning for Cost-Sensitive Classification

Nowadays, most DNNs are designed to solve regular classification problems [2, 27, 3]. Those DNNs usually consist of several hidden layers with a softmax layer of K neurons at the end. Each input vector \mathbf{x} propagates through different hidden layers and is transformed into different levels of latent representations. The softmax layer converts the last latent representation into per-class probability estimation, and takes the class with the highest estimated probability as the prediction $g(\mathbf{x})$ of the network.

On the other hand, only few works have explored cost-sensitive classification using shallow or deep neural networks. [24] studied how sampling and threshold-moving can make the neural network cost-sensitive to tackle the class imbalance problem; [20] proposed four approaches of modifying neural networks for cost-sensitivity. Nevertheless, both works focused on relatively shallow networks and thus can hardly be viewed as general cost-sensitive deep learning algorithms.

In [28], a cost-sensitive deep learning algorithm called Cost-Sensitive DNN (CSDNN) was proposed. In terms of the

¹The boolean operation $\mathbb{I}[\cdot]$ is 1 if the condition is true, and 0 otherwise.

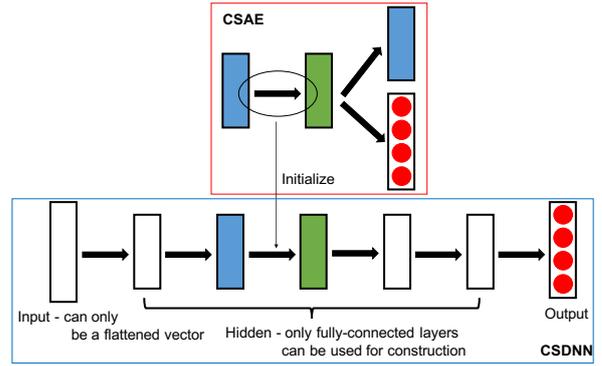


Figure 1: CS AE pretraining for CSDNN [28]

network structure, CSDNN starts with a regular DNN with fully-connected layers, but replaces the softmax layer at the end of the DNN by a cost-estimation layer. Each of the K neurons in the cost-estimation layer provides per-class cost estimation with regression instead of per-class probability estimation. Then, the class with the lowest estimated cost can be naturally taken as the prediction $g_c(\mathbf{x})$ of the network. [28] proposed to train the structure with a cost-sensitive loss function L_{OSR} on the cost-estimation layer.²

[28] then found that the performance of the network can be further improved by careful pretraining, and proposed a Cost-Sensitive Auto-Encoder (CSAE) to pretrain the structure above in a layer-wise manner. CSAE operates similar to a conventional auto-encoder [29], which is a shallow neural network that maps any input \mathbf{x} to a representation such that the output $\tilde{\mathbf{x}}$ is a close reconstruction of the original input. The reconstruction error is commonly measured by cross-entropy loss, denoted by L_{CE} . What makes CSAE different is that the shallow network is augmented with K additional output neurons for cost estimation. That is, CSAE attempts to not only reconstruct \mathbf{x} but also digest the cost information by estimating the cost vector \mathbf{c} . The attempt is represented with a mixture loss $(1-\beta) \cdot L_{CE} + \beta \cdot L_{OSR}$ with a balancing coefficient $\beta \in [0, 1]$ on the output layer of CSAE. When $\beta = 0$, CSAE degrades to a conventional auto-encoder.

Figure 1 illustrates how CSAE is used to pretrain CSDNN. With the pretraining, each layer of (initial) latent representations in CSDNN carries some ability to estimate the costs. That is, the pretraining makes the latent representations *cost-aware*. [28] reported that such initialization indeed allows CSDNN to converge to a better optima and to reach state-of-the-art performance.

3. Proposed Framework

While CSDNN is state-of-the-art, its design is based on fully-connected layers with sigmoid activation functions and thus suffers from some issues. Next, we discuss the issues behind CSDNN that motivate us to study a general framework that allows conducting cost-sensitive deep learning more effectively and effortlessly in Section 3.1. Then we present our proposed framework in Section 3.2.

²The term L_{OSR} stands for One-Sided Regression and roots from a cost-sensitive SVM work [23]. Details are omitted here for lack of space.

3.1. Motivation

Arguably the key idea within CSDNN is pretraining with CSAE. To understand the issues behind CSDNN, we first review the necessity of pretraining for general deep learning. In earlier years, neural networks used sigmoid or hyperbolic-tangent activation functions for non-linear transformation in the hidden layers [34, 35, 36, 29]. Both functions, which exhibit flatness in part of their curves, can cause the gradients of the network to be small. As the depth of the network increases, the small gradients in the latter layers of the network make the gradients in the earlier layers even smaller during back-propagation, a phenomenon known as the diminishing gradients. The phenomenon results in poor local optima of the entire network, and restricts the depth of earlier neural networks [37]. [36, 29] tackled the diminishing-gradient problem by proposing a greedy layer-wise pretraining strategy with Restricted Boltzmann Machines and auto-encoders for initializing the weights of DNN. Pretraining helped mitigate the problem to some degree, but the problem would resurface as the network deepens if we stick with the same activation functions.

In recent years, another route to resolve the diminishing-gradient problem is to consider other activation functions, such as the rectifier linear unit (ReLU) [32]. As ReLU does not suffer from the diminishing-gradient problem as much as sigmoid or hyperbolic-tangent activation functions, pretraining is no longer necessary [38]. Nowadays, ReLU and many of its variants [39, 40] become the mainstream activation functions in modern deep learning studies [41, 3].

CSDNN [28] intended to conduct cost-sensitive deep learning by mimicking what [29] did for regular deep learning: using sigmoid activation functions, and adopting greedy layer-wise pretraining. Thus, CSDNN carries the same problem of diminishing gradients when the network deepens, as our experimental results in Section 4 will demonstrate. To keep cost-sensitive deep learning up to date with modern deep learning studies, it is then necessary to conduct cost-sensitive deep learning with other routes, such as adopting ReLU and removing the pretraining stage.

Nevertheless, directly removing the pretraining stage in CSDNN throws away one important benefit of CSAE in making the latent representations cost-aware. Next, we present our proposed framework to rescue the benefit. As we shall demonstrate later, the proposed framework carries an additional advantage of being generally applicable to a wide range of network structures and activation functions, including but not limited to ReLU.

3.2. Layer-Wise Cost Estimation

Our key goal is to construct a DNN that can simultaneously enjoy the benefit of cost-aware representation extraction (similar to that provided by CSAE), and the flexibility of using any structures. CSAE achieved cost-aware representation extraction by using K additional neurons in the auto-encoder for cost estimation. Our key idea is to also use K additional neurons for cost estimation, but instead of adding them to the auto-encoders that are separated from the DNN, we propose to directly put K neurons into each layer of the DNN. That is, we propose to replace CSAEs by “merging” their additional neurons with the DNN of our interest. The proposed structure is illustrated with Figure 2. By dressing the original DNN with K additional neurons in each layer that serve as auxiliary outputs, the extracted latent representations carry some ability to estimate the costs, thus achieving cost-aware representation extraction almost effortlessly.

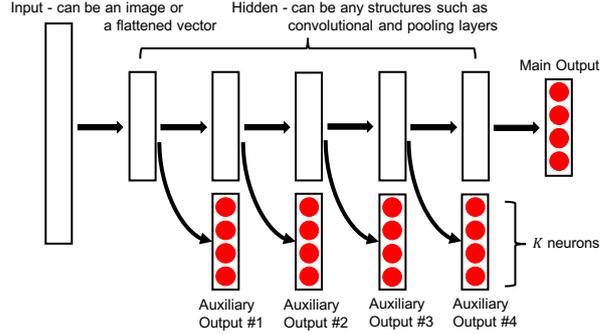


Figure 2: a DNN with five hidden layers dressed with the proposed Auxiliary Internal Targets (AuxIT) framework

As shown in Figure 2, in addition to augmenting K additional neurons to each layer of the DNN, we follow CSDNN and replace the output layer of the DNN with a cost-estimation layer. Then, the only remaining task is to train the “upgraded” DNN with a proper loss function. We consider a simple weighted-mixture loss function of the main one-sided regression loss function at the output layer, and the auxiliary one-sided regression loss functions at the hidden layers. In particular, let $L_{\text{OSR}}^{(i)}$ denote the auxiliary loss function for the output of the i -th hidden layer and $L_{\text{OSR}}^{(*)}$ denote the main loss function at the output layer, we train the upgraded DNN with the loss

$$\sum_{i=1}^{H-1} \alpha_i \cdot L_{\text{OSR}}^{(i)} + L_{\text{OSR}}^{(*)}, \quad (1)$$

where H is the number of hidden layers in the DNN, and α_i is the balancing coefficient for $L_{\text{OSR}}^{(i)}$.³

With the proposed structural add-ons and the mixture loss function, we are now ready to present the full framework in Algorithm 1. The framework will be named as Auxiliary Internal Targets (AuxIT). While the novel framework appears simple, it carries many practical benefits. With the framework, we can now flexibly use ReLU or other activation functions and thus avoid diminishing-gradient problem. We can also build cost-sensitive DNN with any structures, such as image inputs with convolutional and pooling layers. Furthermore, we can apply this framework directly and effortlessly on any state-of-the-art DNN structures such as VGG [2] and ResNet [3] for solving large-scale cost-sensitive classification problems.

4. Experiments

Three sets of experiments are conducted to validate the usefulness of the proposed AuxIT framework.

4.1. Setup

Four benchmark visual data sets are used throughout the three experiments: MNIST, CIFAR-10 [42], CIFAR-100 [42], and Caltech-256 [43]. MNIST belongs to handwritten digit recognition task where each example is a 28×28 gray-scale digit image; CIFAR-10 is a well-known image recognition data set

³There is no need to consider $L_{\text{OSR}}^{(H)}$ for the outputs of the last hidden layer, as the main loss function $L_{\text{OSR}}^{(*)}$ readily conducts cost estimation.

Algorithm 1 Auxiliary Internal Targets (AuxIT)

- Input:** your favorite regular DNN or any off-the-shelf one [31, 2, 27, 3] with H hidden layers; balancing coefficients $\{\alpha_i\}_{i=1}^{H-1}$
- 1: Replace the softmax layer at the end of DNN with K regression neurons and loss function $L_{\text{OSR}}^{(*)}$
 - 2: **for** $i = 1, 2, \dots, H - 1$ **do**
 - 3: Add K additional regression neurons with loss function $L_{\text{OSR}}^{(i)}$ to the output of the i -th hidden layer and connect them fully to the i -th hidden layer
 - 4: **end for**
 - 5: Train the new DNN by back-propagation on (1)
-

with 10 classes where the size of each image is $3 \times 32 \times 32$ (3 for RGB); CIFAR-100 is just like CIFAR-10, except it has 100 classes; Caltech-256 is another popular object recognition data set that contains 256 classes, and the size of each RGB image is roughly 300×400 in average. For all data sets, the training and testing splits follow the source websites; the input vectors in training set are linearly scaled to $[0, 1]$, and the input vectors in the testing sets are scaled accordingly. We use MNIST, CIFAR-10, and CIFAR-100 in our first two experiments, and Caltech-256 in our last experiment.

The four data sets were originally collected for regular (cost-insensitive) classification and thus contain no cost information. We adopt the most frequently-used benchmark in cost-sensitive learning, the randomized proportional setup [33], to generate the costs. For a regular data set $S = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$, the setup first generates a $K \times K$ matrix \mathbf{C} , and sets the diagonal entries $\mathbf{C}(y, y)$ to 0 while sampling the non-diagonal entries $\mathbf{C}(y, k)$ uniformly from $[0, 10 \frac{|n: y_n=k|}{|n: y_n=y|}]$. Then, for each example (\mathbf{x}_n, y_n) in S , its cost vector \mathbf{c}_n is defined as the y_n -th row of matrix \mathbf{C} . The randomized proportional setup generates the cost information that takes the class distribution of the data set into account, charging a higher cost (in expectation) for misclassifying a minority class, and can thus be used to deal with imbalanced classification problems.

Arguably one of the most important use of cost-sensitive classification is to deal with imbalanced data sets. Nevertheless, the first three data sets MNIST, CIFAR-10, and CIFAR-100 are somewhat balanced, and the randomized proportional setup may generate similar cost for each type of misclassification error. To better meet the real-world usage scenario and increase the diversity of data sets, we further conduct experiments to evaluate the algorithms with imbalanced data sets. In particular, for each of the first three data sets MNIST, CIFAR-10, and CIFAR-100, we construct a variant data set by randomly picking 40% of the classes and removing 70% of the examples that belong to those 40% classes. We will name these imbalanced variants as $\text{MNIST}_{\text{imb}}$, $\text{CIFAR-10}_{\text{imb}}$, and $\text{CIFAR-100}_{\text{imb}}$, respectively.

Our first experiment in Section 4.2 intends to investigate the relationship between the balancing coefficient α_i in (1) for using AuxIT and the performance. Our second experiment in Section 4.3 compares DNN equipped with AuxIT framework with state-of-the-art CSDNN [28] to show the usefulness of AuxIT. For the first and the second experiments, the cost information was generated by the randomized proportional setup. Interestingly, there exists a top-down hierarchical tree structure for the 256 classes in Caltech-256 data set, from that hierarchy tree, we can easily define the cost information based on the closeness

between each class. Therefore, our last experiment, presented in Section 4.4, was an interesting cost-sensitive classification problem where the cost information was based on the category closeness. We apply AuxIT to the well-known AlexNet [31] to tackle this challenging cost-sensitive classification task and reports the achieved performance to encourage future researchers to continue to work on this task. In each of the three experiments, we will describe the goal of the experiment, present the experimental results, and provide discussions and conclusions.

4.2. How does α_i affect AuxIT?

In our proposed Auxiliary Internal Targets (AuxIT) framework, K additional neurons are added in parallel to each of the hidden layer in DNN. As an example \mathbf{x} propagates through the network, in addition to the final prediction layer, the DNN also outputs K values in each hidden layer. Same with the final prediction layer, these additional K neurons in each hidden layer also aim to estimate the per-class costs, and are coupled with L_{OSR} . The final objective function for optimizing the entire DNN turns out to be a weighted sum of the main one-sided loss for the final prediction layer and the auxiliary one-sided loss for all hidden layers, and has the form (1).

In this experiment, we would like to investigate the relationship between the selection of α_i in (1) and the performance (average test costs) of AuxIT framework. To simplify the experiment, we keep all coefficients α_i to identical values, that is, $\alpha_1 = \alpha_2 = \dots = \alpha_{H-1} = \alpha$, and (1) becomes:

$$\alpha \cdot \sum_{i=1}^{H-1} L_{\text{OSR}}^{(i)} + L_{\text{OSR}}^{(*)}, \quad (2)$$

and we increase the value of α from 0 to 1 by a step 0.1. MNIST, CIFAR-10, CIFAR-100 and their imbalanced variants are used in this experiment, and their cost information is generated by randomized proportional setup described in Section 4.1. We constructed fully-connected DNN with varying numbers of hidden layers $H = \{1, 2, 3, 4, 5\}$, where each hidden layer consists of 1024 neurons. Note that our proposed AuxIT framework can be applied to DNN consists of any kind of layers, but since our goal in current experiment is not to pursue the best performance but to investigate more about AuxIT, we choose to use only fully-connected layers for constructing DNN in order to reduce the amount of hyper-parameters.

The experimental results are shown in Figure 3. For each data set, we plot 5 curves (because we tested with 5 kinds of numbers of hidden layers), where the x-axis is the value of α , and the y-axis is the corresponding average test costs achieved. Note that when $\alpha = 0$, it means that the DNN does not make use of AuxIT framework. From the six figures, no matter how many hidden layers there are, roughly U-shaped curves could be observed, and the lowest average test costs were achieved when α fell in the range $0.2 \sim 0.5$, implying that α within this range best balanced (2). Another phenomenon we can observe from these six figures is that, when the number of hidden layers increased from 1 to 3, the performance was improved as the entire curve moved downward; but when the number of hidden layers continued to increase from 3 to 5, the performance could not be further improved and even got worse. Such phenomenon could probably be attributed to overfitting - the training examples of MNIST, CIFAR-10, and CIFAR-100 are only 60K, 50K, and 50K, respectively, and their imbalanced counterparts contain even less examples due to the removals, so using more than 3 hidden layers would be overkilling.

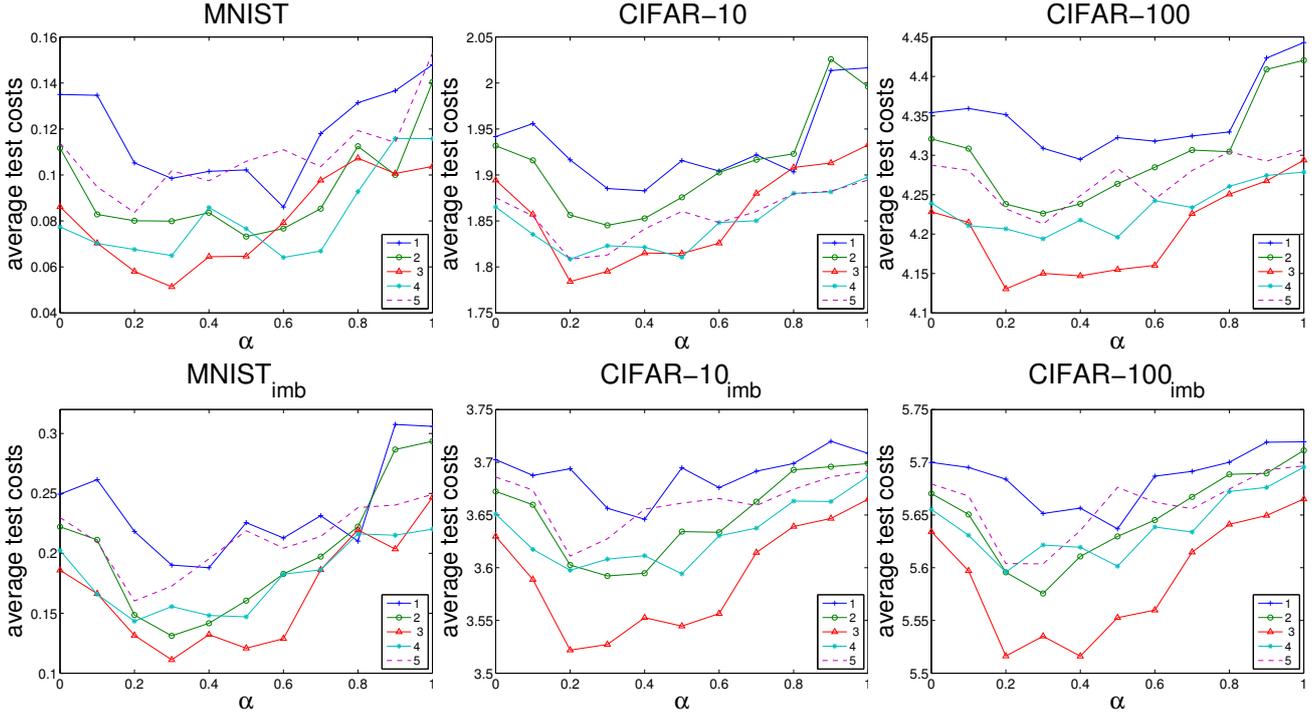


Figure 3: The figure shows the results of our first experiment described in Section 4.2. The six sub-figures show the relationship between the selection of α in (2) for using AuxIT framework and the performance achieved on MNIST, CIFAR-10, CIFAR-100, MNIST_{imb}, CIFAR-10_{imb}, and CIFAR-100_{imb}. Each figure contains five curves, the numbers in the legend are the number of hidden layers, and each number corresponds to a curve. The x-axis is the value of α , and the y-axis is the corresponding average test costs.

4.3. Compare with state-of-the-art

In this experiment, we build two DNNs with and without AuxIT framework and compare them to state-of-the-art Cost-sensitive Deep Neural Network (CSDNN) [28]. We emphasize the two major drawbacks of CSDNN here:

1. CSDNN uses sigmoid functions for non-linear transformations, and this will eventually results in diminishing gradients when the network grows deeper.
2. CSDNN can be applied to DNN that consists of only fully-connected layers, this puts limits on its potential to be extended and applied to more challenging tasks that require modern neural components such as convolution and pooling layers.

To give CSDNN a fair chance of comparison, the two DNNs we build also consist of only fully-connected layers, and ReLU is used as activation function. The first DNN is equipped with AuxIT by setting $\alpha_i = 0.2$, as 0.2 was found to be one of the best value balancing for (2) in Section 4.2, we will refer to this DNN as AuxDNN. The second DNN, which will be referred to as NaiveDNN, did not make use of AuxIT and was directly optimized by L_{OSR} , it is equivalent to setting $\alpha = 0$ in (2).

The experimental results are displayed in Figure 4. The x-axis is the number of hidden layers and the y-axis is the corresponding average test costs achieved. As we can observe from Figure 4, when the number of hidden layers was less than or equal to three, CSDNN outperformed NaiveDNN probably because CSAE were doing cost-aware feature extraction relatively well, which accorded to the experimental results in [28].

When the number of hidden layers exceeded three, all of the three models began to suffer from overfitting, causing their average test costs to increase. However, by looking at CSDNN and NaiveDNN, it was interesting to observe that although the average test costs of both models increased, the extent of increment of CSDNN was larger than that of NaiveDNN. We inferred that this phenomenon was ascribed to the diminishing gradients caused by sigmoid functions used in CSDNN, and although CSAE had done their best to mitigate this problem when the network was relatively shallow, CSDNN can still not escape the fate of diminishing gradients when the network grew deeper. This phenomenon could not be observed in [28] because the deepest network they built had only three hidden layers. As for AuxDNN, it significantly outperformed both CSDNN and NaiveDNN regardless of the number of hidden layers, this further demonstrated the usefulness of our proposed AuxIT framework.

To have a more comprehensive understanding on what the auxiliary outputs are doing, we also record the layer-wise one-sided training loss of AuxDNN with three hidden layers and plot the learning curve in Figure 5 (we only plot the results of CIFAR-10, as the results of MNIST and CIFAR-100 are similar to CIFAR-10). Since it's AuxDNN with three hidden layers, there would be two auxiliary outputs and one main output at the end. From the figure, we can observe that the earlier layers primarily serve as assistance, as their training loss decrease much slower than the last layer, which is the main target.

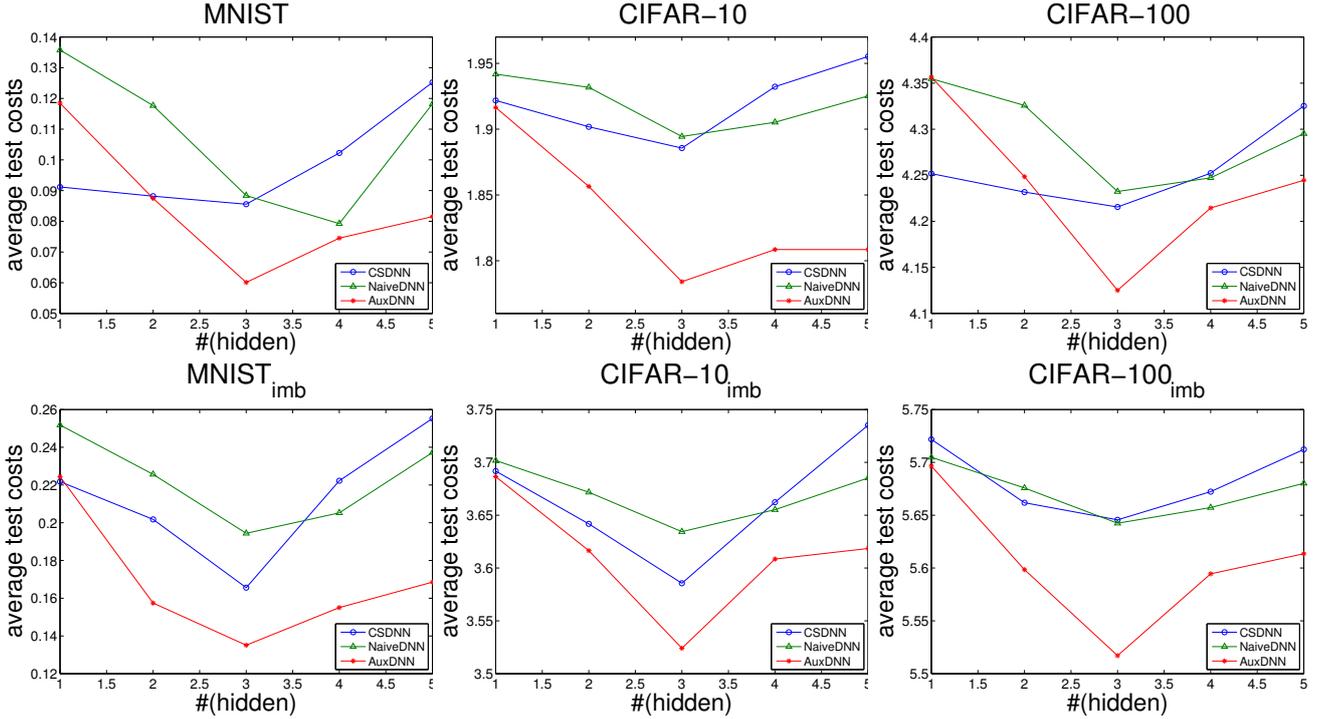


Figure 4: This figure shows the results of our second experiment described in Section 4.3. The six sub-figures display the performance of the three competing DNNs on MNIST, CIFAR-10, CIFAR-100, MNIST_{imb}, CIFAR-10_{imb}, and CIFAR-100_{imb}, where each curve corresponds to one competitor. The x-axis is the number of hidden layers used to construct the DNN, and the y-axis is the corresponding average test costs.

4.4. Category Closeness as Cost Information

In Section 4.2 and 4.3, the cost information for all data sets are generated by randomized proportional setup. In our third experiment, instead of randomized proportional setup, we use a property featured in Caltech-256 data set to generate a much more interesting and realistic cost information. In Caltech-256, the 256 classes present a top-down hierarchical tree structure. Since the 256 classes present a hierarchy tree, the cost of classifying a class-A instance as class B could be interestingly and intuitively defined by the distance between class A and class B within the hierarchy tree. For example, in Figure 6, the cost of classifying a baseball bat as a baseball glove is 2, since it takes only two steps to go from baseball bat to baseball glove through the tree paths; while the cost of classifying a baseball bat as a tennis racket is 4.

We apply the AuxIT framework to the well-known AlexNet [31] to tackle this cost-sensitive classification problem. AlexNet consists of five convolutional layers and three fully-connected layers. We use the off-the-shelf implementation of AlexNet provided in [44], but replace the last softmax layer with regression outputs for per-class cost estimation so that we can apply L_{OSR} and the AuxIT framework. There are around 30K images in Caltech-256, we randomly split 20K images for training and the rest 10K images for testing. As there are 60 million parameters to be learned in AlexNet, a training set contains only 20K images turns out to be insufficient if we train the entire AlexNet on Caltech-256 from scratch. To deal with this problem, we first take AlexNet pretrained on ImageNet [45] provided in [44] as initialization, then replace

the softmax layer with regression outputs, and finally fine-tune the AlexNet on Caltech-256. The final AlexNet equipped with AuxIT framework is shown in Figure 7, and will be referred to as AuxAlexNet. One may wonder that how come a DNN trained for a regular classification task can be transferred to a cost-sensitive classification one. However, this turns out to be not a problem, and will be discussed later.

We compare AuxAlexNet with the original AlexNet, which is set as the cost-blind deep learning baseline. The original AlexNet is also first pretrained on ImageNet, then fine-tuned on Caltech-256. Similar to the first experiment in Section 4.2, we will also try varying values for α to observe how α interacts with the performance of AuxAlexNet. The experimental results are shown in Figure 8. As we can see from the results, AuxAlexNet achieved very different average test costs with different α , but still outperformed the original AlexNet in most cases. It is actually not surprising to see that the transfer learning from a regular classification task to a cost-sensitive classification one works quite well. Recall that the function of DNN is feature extraction, and the cost information in a cost-sensitive classification task just further asks the DNN to pay more attention on some specific features to prevent misclassifying those examples that belong to some really important classes (will need to pay relatively high cost if misclassify them). In this experiment, we give an example of how to apply the proposed AuxIT framework to a modern network to tackle cost-sensitive classification problem, and the experimental results on Caltech-256 with category closeness as cost information validate the usefulness of the AuxIT framework.

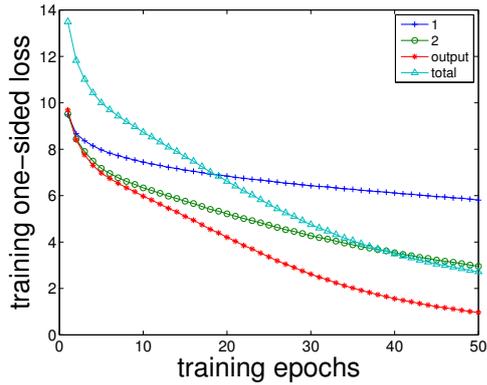


Figure 5: This figure shows the learning process of AuxDNN with three hidden layers on CIFAR-10. The blue and green represent the training loss of the first and second hidden layers, the red curve represents the training loss of the output layer, and the aquamarine curve represents the total loss. The x-axis is the training epochs, and the y-axis is the corresponding one-sided loss.

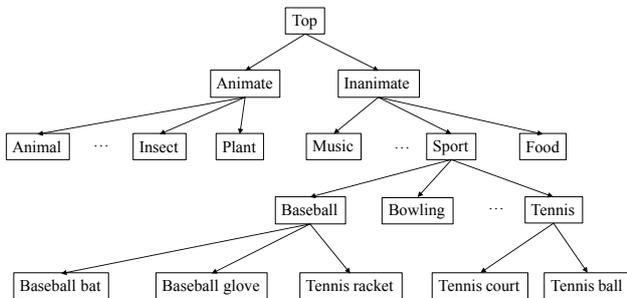


Figure 6: The figure shows a part of the hierarchy tree structure of the 256 classes in Caltech-256. The cost information is defined by the distance between classes. For example, the cost of classifying a baseball bat as a baseball glove is 2 since it takes only two steps to go from baseball bat to baseball glove according to the tree paths; while the cost of classifying a baseball bat as a tennis racket would be 4.

5. Conclusion and Future Work

We propose a novel framework Auxiliary Internal Targets (AuxIT) for general end-to-end cost-sensitive deep learning. Different from the previous approaches, the framework can be applied to DNN that consists of any structures to tackle challenging cost-sensitive classification problems. Extensive experimental results on MNIST, CIFAR-10, CIFAR-100, and Caltech-256 with two cost information settings demonstrate the usefulness of the proposed framework for making any advanced DNN models cost-sensitive. In the future, we will build a deeper network with AuxIT framework to tackle ImageNet cost-sensitive classification problem.

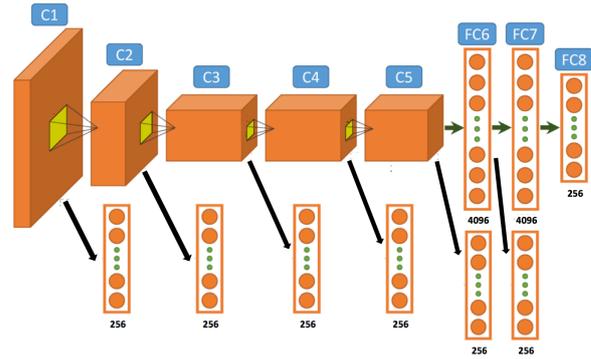


Figure 7: This figure shows how to apply the AuxIT framework to AlexNet. AlexNet consists of five convolutional layers and three fully-connected layers, where the last softmax layer is replaced with regression outputs for per-class cost estimation. For each hidden layer in AlexNet, 256 additional neurons are added as auxiliary per-class cost estimation. This network is referred to as AuxAlexNet in the experiment.

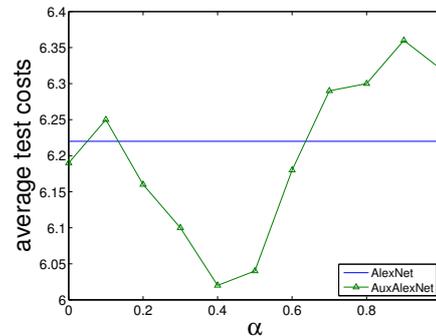


Figure 8: This figure shows the results of our third experiment described in Section 4.4. The x-axis is the value of α in (2) and is specific to AuxAlexNet, which is represented by the green curve, and the y-axis is the corresponding average test costs. The blue curve represents the cost-blind AlexNet and has nothing to do with α .

6. References

- [1] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *CVPR*, 2012.
- [2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint arXiv:1512.03385*, 2015.
- [4] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *NIPS*, 2013.
- [5] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *CVPR*, 2014.
- [6] Y. Zhang, K. Sohn, R. Villegas, G. Pan, and H. Lee, "Improving object detection with deep convolutional networks via bayesian optimization and structured prediction," in *CVPR*, 2015.
- [7] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *arXiv preprint arXiv:1312.6229*, 2013.

- [8] D. Lin, X. Shen, C. Lu, and J. Jia, "Deep lac: Deep localization, alignment and classification for fine-grained recognition," in *CVPR*, 2015.
- [9] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *CVPR*, 2014.
- [10] H. Ye, Z. Wu, R.-W. Zhao, X. Wang, Y.-G. Jiang, and X. Xue, "Evaluating two-stream cnn for video classification," in *ICMR*, 2015.
- [11] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: Deep networks for video classification," in *CVPR*, 2015.
- [12] M. Tan, "Cost-sensitive learning of classification knowledge and its applications in robotics," *Machine Learning*, vol. 13, no. 1, pp. 7–33, 1993.
- [13] P. K. Chan and S. J. Stolfo, "Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection," in *KDD*, 1998.
- [14] W. Fan, W. Lee, S. J. Stolfo, and M. Miller, "A multiple model cost-sensitive approach for intrusion detection," in *ECML*, 2000.
- [15] Y. Zhang and Z.-H. Zhou, "Cost-sensitive face recognition," *TPAMI*, vol. 32, no. 10, pp. 1758–1769, 2010.
- [16] T.-K. Jan, H.-T. Lin, H.-P. Chen, T.-C. Chern, C.-Y. Huang, B.-C. Wen, C.-W. Chung, Y.-J. Li, Y.-C. Chuang, L.-L. Li, Y.-J. Chan, J.-K. Wang, Y.-L. Wang, C.-H. Lin, and D.-W. Wang, "Cost-sensitive classification on pathogen species of bacterial meningitis by Surface Enhanced Raman Scattering," in *BIBM*, 2011.
- [17] J. Lu and Y.-P. Tan, "Cost-sensitive subspace learning for face recognition," in *CVPR*, 2010.
- [18] L. Zhang, H. Li, X. Zhou, B. Huang, and L. Shang, "Cost-sensitive sequential three-way decision for face recognition," in *RSEISP*, 2014.
- [19] G. Zhang, H. Sun, Z. Ji, Y.-H. Yuan, and Q. Sun, "Cost-sensitive dictionary learning for face recognition," *Pattern Recognition*, vol. 60, pp. 613–629, 2016.
- [20] M. Kukar and I. Kononenko, "Cost-sensitive learning with neural networks," in *ECAI*, 1998.
- [21] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive," in *KDD*, 1999.
- [22] B. Zadrozny and C. Elkan, "Learning and making decisions when costs and probabilities are both unknown," in *KDD*, 2001.
- [23] H.-H. Tu and H.-T. Lin, "One-sided support vector regression for multi-class cost-sensitive classification," in *ICML*, 2010.
- [24] Z.-H. Zhou and X.-Y. Liu, "Training cost-sensitive neural networks with methods addressing the class imbalance problem," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 63–77, 2006.
- [25] G. E. Dahl, D. Yu, L. Deng, and A. Acero, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 30–42, 2012.
- [26] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.
- [27] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015.
- [28] Y.-A. Chung, H.-T. Lin, and S.-W. Yang, "Cost-aware pre-training for multiclass cost-sensitive deep learning," in *IJCAI*, 2016.
- [29] Y. Bengio, "Learning deep architectures for ai," *Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
- [30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [31] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.
- [32] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, 2010.
- [33] N. Abe, B. Zadrozny, and J. Langford, "An iterative method for multi-class cost-sensitive learning," in *KDD*, 2004.
- [34] J. E. Dayhoff, *Neural Network Architectures: An Introduction*. Van Nostrand Reinhold Co., 1990.
- [35] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back, "Face recognition: A convolutional neural-network approach," *IEEE transactions on neural networks*, vol. 8, no. 1, pp. 98–113, 1997.
- [36] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [37] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.
- [38] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *AISTATS*, 2011.
- [39] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.
- [40] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *ICCV*, 2015.
- [41] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *ICML, Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.
- [42] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," *Master's thesis, Department of Computer Science, University of Toronto*, 2009.
- [43] G. Griffin, A. Holub, and P. Perona, "Caltech-256 object category dataset," *California Institute of Technology*, 2007.
- [44] W. Ding, R. Wang, F. Mao, and G. Taylor, "Theano-based large-scale visual recognition with multiple gpus," *arXiv preprint arXiv:1412.2302*, 2014.
- [45] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.