Regularizing CNN Transfer Learning with Randomised Regression

Yang Zhong Atsuto Maki
Division of Robotics, Perception, and Learning
KTH Royal Institute of Technology, Sweden
{yzhong, atsuto}@kth.se

Abstract

This paper is about regularizing deep convolutional networks (CNNs) based on an adaptive framework for transfer learning with limited training data in the target domain. Recent advances of CNN regularization in this context are commonly due to the use of additional regularization objectives. They guide the training away from the target task using some forms of concrete tasks. Unlike those related approaches, we suggest that an objective without a concrete goal can still serve well as a regularizer. In particular, we demonstrate Pseudo-task Regularization (PtR) which dynamically regularizes a network by simply attempting to regress image representations to pseudo-regression targets during fine-tuning. That is, a CNN is efficiently regularized without additional resources of data or prior domain expertise. In sum, the proposed PtR provides: a) an alternative for network regularization without dependence on the design of concrete regularization objectives or extra annotations; b) a dynamically adjusted and maintained strength of regularization effect by balancing the gradient norms between objectives on-line. Through numerous experiments, surprisingly, the improvements on classification accuracy by PtR are shown greater or on a par to the recent state-of-the-art methods.

1. Introduction

Deep convolutional neural networks (CNNs) have recently advanced the development of computer vision and flourished in many large-scale computer vision applications [3, 22, 23, 16]. Since the introduction of AlexNet [18], deeper and more complex network architectures, such as VGG [31], Inception [33], ResNet [11], and DenseNet [12], have been proposed. In addition, other contributions have been made toward network optimization, which has been helping the performance and efficiency of CNNs, e.g. BatchNorm [13] and MiniBatchSGD [7]. Despite the improved effectiveness by those, one of the known open issues is that CNNs are normally over-parameterized and would demand a large-scale labeled dataset.

It is a common practice to exploit transfer learning which adapts a model pre-trained on a source task to a new target task when given a small amount of labeled dataset. Specifically, by leveraging the transferability of deep features [39] one can map images to a middle or high-level feature through pre-trained model and therewith train target specific classifiers [4, 28, 40], which is often called *feature selection*. It is also viable to *fine-tune* a source model for the target data. As fine-tuning aims to optimize the entire network for the target task, it often achieves higher effectiveness and has therefore been a rule of thumb in CNN transfer learning with a limited amount of domain data [1]. During fine-tuning, a source model needs to be mildly tuned to avoid overfitting due to the fact that deep networks are yet over-parameterized for small-scale target tasks.

One of the challenges during fine-tuning, which this paper also addresses, is to achieve network regularization for an over-parameterized model with limited training samples. In the recent state-of-the-art transfer learning solutions, there is a trend of using an auxiliary training objective in a framework of multi-objective[1] learning for improved regularization [6, 21, 20, 5]. These auxiliary objectives are designed in a concrete and target-specific manner, through which models would enforce certain desired properties that facilitates multiple purposes in the learned image representations. The key to the enhanced regularization on the target task is then attributed to the improved generality learned from the imposed auxiliary objectives through partial or entire source task data. However, the regularization gain comes with a resource-dependent cost of the storage of off-the-shelf predictions for multiple steps of network training [21], selecting qualified labeled data samples from the source domain for a target task [6], using a complex network architecture during training [5], or recalling the source model [20].

From a network training perspective, as another way to understand the regularization, a basic effect of training with

[1]For ease of discussions, we do not distinguish training "objectives" from "tasks"; thus, multi-objective and multi-task learning may be used interchangeably.

a regularization objective could be considered as to *distract* the minimization of the empirical loss (typically, through a structural loss). As a result, the regularization power can also be seen to come from the extra gradients generated by the employed distracting (regularization) objective. These gradients cause useful distortions in the gradient-descent trajectory to force the network to tolerate slightly higher empirical loss in the course of training, which allows for more chances in seeking better optima. Now, if such a distraction effect is the essence to network regularization, it is worthwhile to study whether the regularization objective could have some alternative form rather than being a real and concrete task.

Intuitively, if it is the distraction (rather than the convergence of a regularization task) that is the primary interest, there could be diverse ways to construct a *distractor* which interferes the training of the target task while looking for an improved regularization. One potential approach could be through a *pseudo-task* which neither depends on the above mentioned data and storage availability for multi-task learning, nor a concrete goal as designed in [6, 21, 20, 5].

In this paper, considering image classification tasks in a transfer learning scenario, we aim to device a regularizer which generates distractions while being independent of concrete tasks. Our regularizer simply exploits a *pseudo-task*[2] that injects random noise in the gradients to distract the training on the target tasks, to seek for improvements. Experimental results consistently support our conjecture on various datasets and with different network architectures. The contributions of this paper are:

1. We demonstrate *Pseudo-task Regularization* (**PtR**) that provides an efficient alternative to other recent best regularization based on real and concrete tasks.

2. In PtR, useful gradients for regularization are generated through a pseudo-task, in which we propose to dynamically adjust the strength of the regularization based on the gradient norms of the target objective and the pseudo-task.

The results by those suggest a novel interpretation on the key elements of network regularization for CNN transfer learning, which we hope future research will exploit further to identify the essential requirements for a regularizer.

2. Pseudo-task Regularization

2.1. Overview

Our motivation is to let a CNN learn the representations for a target task while also being distracted so that the

[2]Networks are never able to converge on the pseudo-task as it leverages random regression targets, hence the name pseudo-task. See the details in Sec. 2.2.

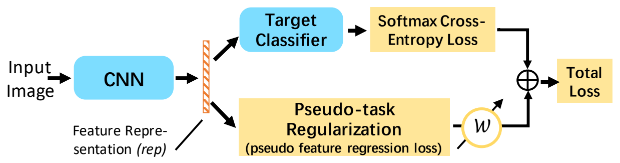
Input Image
CNN
Feature Representation (rep)
Target Classifier
Softmax Cross-Entropy Loss
Pseudo-task Regularization (pseudo feature regression loss)
w
Total Loss

Figure 1. An overview of the proposed Pseudo-task Regularization (PtR). The path linking the blue modules illustrates a vanilla fine-tuning pipeline with a target classifier trained by a cross-entropy loss. The PtR loss, which is connected to the feature representation layer of a CNN, is brought into the network training when the convergence on target task is relatively stable. The total loss therewith is the sum of cross-entropy loss on the target task and the weighted PtR loss. The PtR loss module automatically weights the strength of regularization according to the gradient norm of the target task on the feature layer. The PtR is explained in Algorithm 1 in detail.

learned representations are not excessively target-task specific, leading to the loss of generality. For this purpose, as shown in Figure 1, we choose to exploit a multi-task learning framework which utilizes two training objectives: one is the cross-entropy loss for the target-task classification, the other objective generates distractions to promote the generalization through a pseudo-regression task. We call it Pseudo-task Regularization (PtR).

In using both of the training objectives, a significant aspect of PtR is to balance the impact of the two loss functions. It is reasonable that the distraction should be on a proper level which is not too strong and hinder model convergence, nor is on an ignorable magnitude compared to the gradients of the target classifier. To this end, inspired by [2], we propose to dynamically balance the strengths of gradients from the two losses according to the gradient norms with respect to the image representation during the training process. The training procedure of our adaptive multi-task learning framework is described in Algorithm 1.

2.2. Algorithm

Our method learns image representations on a target task using a pre-trained model in an adaptive multi-objective learning framework, as shown in Algorithm 1. For a training iteration i, it computes the cross-entropy loss $L_{ce}^{(i)}$ and additionally the loss by the randomised regressor, the *Pseudo-task Regularization loss* $L_{PtR}^{(i)}$, starting when $L_{ce}^{(i)}$ falls below a certain threshold of average epoch loss T. (The choice of the threshold is not critical as explained in Figure 2 of the supplementary.) $L_{PtR}^{(i)}$ is calculated by regressing the image representation to a pseudo-regression target. In PtR, we use random regression targets generated on-line such that:

$$L_{PtR}^{(i)} = f_{reg}(rep^{(i)}, t^{(i)}), \tag{1}$$

where rep stands for the activation of the representation layer in a CNN, t for pseudo-regression targets with an

```
Algorithm 1: Training with Pseudo-task Regularization
  Source: a) Off-the-shelf net; b) Labeled data in
          target domain
  Procedure:
  for iteration (batch) $i$ do
      Compute cross-entropy loss $L_{ce}^{(i)}$.
      if $L_{ce}^{(i)}$ far from minimum then
          Back propagate $L_{ce}^{(i)}$ only;
      else
          First, perform the following calculations:
          1. $L_{PtR}^{(i)}$ : the pseudo-regression task loss,
          w.r.t. the regression target $t^{(i)}$ generated
          on-line;
          2. $G_{ce}^{(i)}$ and $G_{PtR}^{(i)}$ : the gradient norms of
          $L_{ce}^{(i)}$ and $L_{PtR}^{(i)}$ w.r.t. $rep^{(i)}$. $rep^{(i)}$ stands for
          the image representations of the batch;
          3. $\overline{G}_{ce}^{(i)}$ and $\overline{G}_{PtR}^{(i)}$ : the average of $G_{ce}^{(i)}$ and
          $G_{PtR}^{(i)}$ over the batch;
          4. Weight $w$: $w = \frac{\overline{G}_{ce}^{(i)}}{\overline{G}_{PtR}^{(i)} \cdot R}$ for a target ratio
          $R$.
          Then, back propagate $L_{ce}^{(i)} + w \cdot L_{PtR}^{(i)}$.
      end
  end
```

equal dimension as rep, and $f_{reg}(\cdot)$ is a regression function for which we consider two popular choices: $L2$ loss and "smooth-L1" (denoted by SML1) loss. Note that the pseudo targets are randomly generated during training so that the training instances are not bounded to the generated regression targets. The details of these targets are described in Section 3.1.

The total loss $L_{total}^{(i)}$ is the weighted sum of the cross-entropy loss and the regression loss:

$$L_{total}^{(i)} = L_{ce}^{(i)} + wL_{PtR}^{(i)}, \tag{2}$$

where w is a coefficient to balance the impact of the distraction regressor, as explained below; $L_{ce}^{(i)}$ and the balanced regression loss $w \cdot L_{PtR}^{(i)}$ are back propagated through the network. Weight decay is omitted for conciseness.

To generate a proper level of distraction for regularization, we first calculate the gradient norms of the cross-entropy loss and those of the regression loss w.r.t. the output feature for each instance, which are denoted by $G_{ce}^{(i)}$ and $G_{PtR}^{(i)}$, respectively (for brevity, the indexes of the training instance in batch i are omitted):

$$G_{ce}^{(i)} = ||\frac{\partial L_{ce}^{(i)}}{\partial rep^{(i)}}||_2, \quad G_{PtR}^{(i)} = ||\frac{\partial L_{PtR}^{(i)}}{\partial rep^{(i)}}||_2. \tag{3}$$

The gradient norms are then averaged over the batch as:

$$\overline{G}_{ce}^{(i)} = \mathbb{E}[G_{ce}^{(i)}], \quad \overline{G}_{PtR}^{(i)} = \mathbb{E}[G_{PtR}^{(i)}]. \tag{4}$$

In order to balance the relative impact of $L_{ce}^{(i)}$ and $L_{PtR}^{(i)}$, we introduce a target gradient norm ratio R. It is defined by the gradient norm ratio of the cross-entropy loss and a desired regression loss in the form of signal-to-noise ratio: $R = \overline{G}_{ce}^{(i)}/w\overline{G}_{PtR}^{(i)}$. Thus, for the gradient norm ratio to satisfy R at iteration i, $L_{PtR}^{(i)}$ needs to be weighted by a factor w:

$$w = \frac{\overline{G}_{ce}^{(i)}}{\overline{G}_{PtR}^{(i)} \cdot R}, \tag{5}$$

which is calculated on-line per batch before back propagation.

It is worth nothing that the common way of balancing losses through a fixed weight (w) would not be an efficient design choice for PtR, as it would never guarantee precise regulation on the gradient norm for regularization. On the contrary, with the use of a dynamic weight PtR is not bound to a particular regularization (regression) loss.

2.3. Encouraging higher variance in gradients

To explore the impact to the gradients by using a randomly varying regression target, we case-study a minimum toy example network which is composed by one hidden neuron (with non-linear activation) connected to one input and one output. The single hidden neuron is denoted by f, whose output $f^{(o)}$ is seen as the feature representation learned by the example network. The input to f, denoted by $f^{(i)}$, is the product of an input x and its learnable weight a on the input path. That is: $f^{(o)} = \delta(f^{(i)})$, $f^{(i)} = a \cdot x$, and $\delta(\cdot)$ represents the ReLU function.

When a regression target t is applied to $f^{(o)}$, the regression loss $E_{reg} = \frac{1}{2}(f^{(o)} - t)^2$. During back propagation, if neuron f is activated, the gradient on a by the chain rule is:

$$\begin{aligned}\frac{\partial E_{reg}}{\partial a} &= \frac{\partial E_{reg}}{\partial f^{(o)}} \cdot \frac{\partial f^{(o)}}{\partial f^{(i)}} \cdot \frac{\partial f^{(i)}}{\partial a} = \mid f^{(o)} - t \mid \cdot x \\ &= \begin{cases}(f^{(o)} - t) \cdot x, & \text{if } (f^{(o)} - t) \geq 0, \\ -(f^{(o)} - t) \cdot x, & \text{otherwise.}\end{cases}\end{aligned} \tag{6}$$

The variance of the gradient of a, if a simplified assumption can hold that x being a constant, $Var(\frac{\partial E_{reg}}{\partial a})$, is determined by that of $f^{(o)}$ and regression target t, such that

$$\text{Var}(\frac{\partial E_{reg}}{\partial a}) = \text{Var}(f^{(o)}) + \text{Var}(t), \tag{7}$$

given that t is a variable independent of $f^{(o)}$.

It can be seen that, if the regularization is achieved through feature norm penalization (e.g., as proposed by

[10]), t in Equation 6 equals a constant of 0. Consequently, in Equation 7, the variance of gradients gets smaller compared to using other regression targets which follow a certain distribution. PtR generates gradients with higher variance using an independent randomly-varying pseudo target. By leveraging them, PtR would explore more local optima to yield higher chances in avoiding saddle points and achieving stronger regularization.

3. Experiments and Results

3.1. Experimental Setup

Datasets. For transfer learning, training CNNs with data across domains has been found to be an important regularization method. However, our experiments focus on a situation where data of other domain is not available. We also focus on a challenging scenario where the training samples are sparse. For this purpose, four commonly used small-scale transfer learning datasets are selected to comparatively evaluate PtR: Flower102 [27], CUB200-2011 [35], MIT67 [30], and Stanford40 [37], two of which represent fine-grained classification tasks of different scenarios. Besides, we also chose 500 identities[3] from the WebFace [38] dataset, denoted by "WebFace500", to evaluate PtR when performing transfer learning from image classification to closed-set face identification with scarce samples per class. Caltech256 [8] was also used for performance evaluation in a general image recognition scenario.

On Flower102 we faithfully follow the data splits for training and testing. On the WebFace500 dataset, each identity has random 20 training images, five validation images, and on average 24 test images. Faces were segmented and normalized to a fixed scale with a face detector [15] before training. From Caltech256, we formed two independent training sets with 30 and 60 training samples per class, respectively, for consistency with [6, 20]. For other datasets, 10% of the training images were randomly separated to form the validation sets for model training.

Training and Evaluation. To augment the training images, we employed random jittering and subtle scaling and rotation perturbations to the training images. We resized images of all involved datasets to 250 $\times$ 250 pixels, and the aspect ratio of the images was retained by applying zero padding all the time. During test time, we averaged over the network responses from the target-task classifiers over ten crops which were sampled from the corners and the centers of originals and the flipped counterparts.

As we consider the vanilla fine-tuning procedure as the baseline, it is very important to ensure that the effectiveness of vanilla fine-tuning is not underestimated. To this end, we carefully selected learning rate schedules for fine-tuning to demonstrate the test accuracy on each dataset with each type of network architecture. To conduct fair comparisons to fine-tuning as much as possible, we also used the same learning rates used by fine-tuning in our dynamic pseudo-task regularization approach; the learning rate schedules were slightly different due to the difference in converging speeds. A learning rate was decreased when the validation loss and validation accuracy stopped progressing and it was decreased twice before model training was terminated. The models trained after the last epoch of their learning rate schedules were always used for performance evaluations.

[3]Random 500 identities that have the most training instances on the WebFace dataset.

Implementation details. Experiments on different datasets shared many common settings. We used the standard SGD optimizer with momentum set to 0.9. The batch size was set to 20 to reduce overfitting as much as possible (unless otherwise stated); weight decay was set to 0.0005 for VGG networks [31] and 0.0001 for ResNet [11] architectures except in a number of ablation studies. The dropout ratio for VGG networks was set to 0.5. Our experiments were implemented with PyTorch [29].

We always started the experiments from an ImageNet [3] pretrained model. As the training data was visited randomly, we ran **five independent runs** and average the results to mitigate the impact of randomness for all the experiments. The classification accuracy was mostly used to compare with related methods except [6].

Other hyperparameters to PtR. In the PtR, the impact of the additional loss is adjusted primarily by the target gradient norm ratio R that controls the interference gradient magnitude. Then, the gradients with respect to each feature dimension are largely determined by the nature of a pseudo-task which we employ in our experiments, such as the distribution of pseudo-regression targets. Without loss of generality we considered the random targets, $t^{(i)}$, following a uniform distribution with a mean value of m such that $m = \mathbb{E}[t_j^{(i)}]$; i.e., for any single regression target $t_j^{(i)}$ in $t^{(i)}$, $t_j^{(i)} \in [0, 2m)$, where j is an instance in batch i ($j \in [1, |i|]$) and $|i|$ is the batch size.

We used independent hold-out sets to efficiently determine R and m (to avoid expensive cross-validation parameter search): for ResNet structures R=3 and m=1 are consistently used on all datasets; for the VGG-16 structure, R varies in the range between 3 and 5, and m around 10 to 15. We chose T=1 as a sensible setting in all the experiments given that the influence by the choice of T is limited. Details are given in Figure 2 of the supplementary.

3.2. Results and Comparisons

As the VGG-16 architecture has been used very commonly in many different transfer learning applications, we first evaluate PtR with it across five different dataset using SML1 and L2 for the regression function respectively, and compare against the fine-tuning baseline. The results are

Table 1. The comparative classification accuracy by the Pseudo-task Regularization (PtR), against vanilla fine-tuning (in column Baseline) with two different choices of regression functions, SML1 and L2, with the VGG-16 architecture. The performance gain brought about by PtR with different regression loss functions are in the two columns in the center under SML1 and L2, respectively. The corresponding error rate reduction values are in the rightmost columns. The standard deviation of each experiment is given in parenthesis.

	Baseline	Regularization Gain		Error Rate Reduction	
		SML1	L2	SML1	L2
Flower102	83.92% (0.36)	2.38% (0.32)	2.61% (0.42)	14.80%	16.23%
CUB200	75.07% (0.26)	3.05% (0.39)	2.84% (0.37)	12.23%	11.39%
MIT67	71.55% (0.38)	1.42% (0.58)	1.39% (0.40)	4.99%	4.89%
Stanford40	76.99% (0.19)	2.50% (0.09)	2.21% (0.16)	10.86%	9.60%
WebFace500	77.54% (0.52)	0.95% (0.56)	0.83% (0.47)	4.23%	3.70%

listed in Table 1.

It is observed that PtR helps improve the vanilla fine-tuning on all the classification tasks tested with two different regression functions; it brings about reasonable and consistent performance gain. On WebFace500 which contains the most training samples, it reduced the error rate for around 4%, but on Flower102 and CUB200 where the training samples are more sparse, it is particularly more effective and reduces the error rates for more than 10%. These results suggest that, on the one hand, collecting more data helps regularization even for small datasets. On the other hand, when the training samples become sparse, PtR manages to keep the learned representations from being excessively target-task specific and further promises the networks to learn more useful representations. The choice of regression function does not appear to be a significant factor as the test accuracy with SML1 and L2 are close; SML1 is used in all the following experiments.

We have also conducted a large number of comparative experiments to recent best performing multi-task/objective based regularization approaches: Joint Training (**JointTrain**) [21], Learning without Forgetting (**LwF**) [21], Borrowing Treasures from the Wealthy (**BTfW**) [6], Inductive Bias (**Ind.Bias**) [20], and Pair-wise Confusion (**PC**) [5]. In addition, we evaluated the regularization through feature norm penalty (denoted by **FNP**) [10] in the context of transfer learning (hyperparameters of FNP were therefore set by using the same procedure of PtR for fairness). We have also compared the impact of weight decay on CUB200 and Caltech256 by disabling weight decay (denoted by "w/o WD"). As we intended to perform all experiments on a single GPU module with 12GB memory, the ResNet-101 was used as a compromise to compare to the results achieved by a special memory-saving version of ResNet-152 in [6]. For fair evaluations, all the corresponding vanilla fine-tuning baseline and improved test accuracy are shown in the following tables together with accuracy gain.

The comparative results on CUB200 dataset are shown in Table 2. It can be seen that the accuracy gain by PtR (with VGG-16) is slightly better than JointTrain where real source data was used for regularization; it also performs better than LwF where off-the-shelf predictions were used. Compared to FNP, PtR outperforms by a visible margin of 1.3%. Although PtR achieved lower accuracy gain than PC, the gaps is not significant regardless of network architecture (around 0.5%). For the absolute accuracy, it is also noticeable that PtR achieved the highest baseline performance as well as that of the optimized models among all the other methods. Weight decay seems not impacting PtR, but the baseline accuracy is 0.7% higher when training without it.

Table 2. Comparing classification accuracy on the CUB200 dataset. All the numerical results are in %. The network and other settings (if any) used by each method are given in parenthesis.

Method	Baseline	Acc.	Gain
JointTrain (VGG-16)	72.1	74.6	2.5
LwF (VGG-16)	72.1	72.3	0.2
PC(VGG-16)	73.3	76.5	**3.2**
PtR (VGG-16)	75.1	78.1	3.0
PC (ResNet-50)	78.2	80.3	**2.1**
FNP (ResNet-50)	80.3	80.6	0.3
PtR (ResNet-50)	80.3	81.9	1.6
PtR (ResNet-50, w/o WD)	81.0	82.0	1.0

On Flower102 dataset, as shown in Table 3, the gain by PtR is larger than PC for 1.4% with the VGG-16 structure; it is just equivalent to that of PC with the ResNet-50 network. FNP brings some regularization margin, but it is 0.3% lower than that of PtR. For PtR, it achieves consistent gain in accuracy with ResNet-50 and ResNet-101, and the depth of the network does not appear to deteriorate the regularization effect. Although we achieved an equally good baseline performance as BTfW (in parenthesis of the bottom row of Table 3), the regularization gain of BTfW is higher than PtR or any other methods. The difference in regularization may suggest that training with sufficient labeled data in a multi-task learning framework is a stronger regularization for transfer learning.

Similar observations can also be found from the results on MIT67 dataset, as shown in Table 4. With the VGG-16 architecture, regularization effect by PtR is again very close to that of JointTrain and outperforms LwF. With the ResNet, the regularization gain by PtR is equivalent to that of Ind.Bias and FNP. BTfW also achieves higher gain than

Table 3. Classification accuracy and accuracy gain (in %) comparisons on the Flower102 dataset. The mean class accuracies used to compare to BTfW [6] are listed in parenthesis.

Method	Baseline	Acc.	Gain
PC(VGG-16)	85.2	86.2	1.0
PtR (VGG-16)	83.9	86.3	**2.4**
BTfW (ResNet-152)	92.3	94.7	**2.4**
PC (ResNet-50)	92.5	93.5	1.0
FNP (ResNet-50)	91.0	91.5	0.5
PtR (ResNet-50)	91.0	91.8	0.8
PtR (ResNet-101)	90.6(92.3)	91.6(93.2)	1.0(0.9)

Table 4. Comparative results on the MIT67 (in %). The mean class accuracies used to compare to BTfW [6] are in parenthesis.

Method	Baseline	Acc.	Gain
JointTrain (VGG-16)	74	75.5	**1.5**
LwF (VGG-16)	74	74.7	0.7
PtR (VGG-16)	71.6	73.0	1.4
BTfW (ResNet-152)	81.7	82.8	**1.1**
Ind.Bias (ResNet-101)	77.5	78	0.5
FNP (ResNet-50)	77.4	78.0	0.6
PtR (ResNet-50)	77.4	77.9	0.5
PtR (ResNet-101)	78.7(78.7)	79.2(79.2)	0.5(0.5)

the other methods with ResNet. We would infer that optimizing a network simultaneously on multiple tasks with sufficient selected real data samples might be more effective than other related methods. As for PtR, it brings about consistent margin over the fine-tuning base-line regardless of the depth of ResNet architecture, which also coincides with the results in Table 3.

The results on Caltech256 dataset are in Table 5. In these experiments, we increased the batch size to 32, which is a value in between of those used by [6, 20], to make fair comparisons as much as possible. Interestingly, we achieved the best baseline accuracy among all the comparing methods with both of the Caltech256 partitions. Consequently, it could be harder for PtR to demonstrate the regularization power in comparison to others because a better generalized baseline usually has a smaller room to improve the generalization. However, we can still observe some similar trends. First, as in the previous experiments, by training a network with sufficient annotated data of multiple classes, BTfW achieves the best regularization gain (around 2.6% with both setups). Second, PtR consistently delivers regularization gain; for both data splits the gains are equivalent, which indicates that PtR would not be so sensitive to the size of training data of each category. The improvement brought about by FNP might be marginal or even unstable given the negative gain on Caltech256-30. The impact of weight decay on the classification accuracy of PtR is not visible.

Through the analysis of the comparative results, we argue that PtR delivers consistent gains, with statistical significance (also see the standard deviation given in Table 1 in the supplementary), which are on a par with the recent state-of-the-art approaches. The compared methods consider using auxiliary objectives attached to concrete tasks while enhancing regularization, but PtR which leverages an additional pseudo-task as a regularizer is free from design of concrete auxiliary tasks and more straightforward. Compared to LwF, the "warmup" training stage and collecting predictions of the target data from an off-the-shelf model is not required by PtR. It is not needed for PtR to remember the "starting point" parameters of an off-the-shelf model as in [20]. PtR is also more efficient than [5] which requires a Siamese network, and it does not depend on annotated data from other domain either as in [6].

Table 5. Mean class accuracies and accuracy gains (in %) on the Caltech256 with two partitions of training data. Bsln is short for baseline. Mean class accuracies were the same as the average classification accuracy over the test set, thus are not given in parenthesis in this table. BTfW [6] was using ResNet-152 while others were using ResNet-101. The network used by each method is not shown in this table for brevity.

	Caltech256-30			Caltech256-60		
Method	Bsln.	Acc.	Gain	Bsln.	Acc.	Gain
BTfW	81.2	83.8	**2.6**	86.4	89.1	**2.7**
Ind.Bias	81.5	83.5	2.0	85.3	86.4	1.1
FNP	84.0	83.8	-0.2	86.8	86.9	0.1
PtR	84.0	84.5	0.5	86.8	87.2	0.4
PtR,w/o WD	84.0	84.5	0.5	86.9	87.2	0.3

4. The Effect of PtR on Predictions

To investigate the impact of PtR, we case-study on the validation set of the CUB200 dataset with ResNet-50 network to explore how the predictions have been altered in comparison to those from vanilla fine-tuning. We base our analysis on the concept of confusion matrix and define a matrix $\mathbf{C}^{(D\times D)}$ (D=200), each row of which contains cumulative predicted probabilities across all the validation samples for different class categories. We compute matrices $\mathbf{C}_{PtR}$ and $\mathbf{C}_{ft}$ for the cases of using PtR and baseline fine-tuned model, respectively. We then sum their diagonal elements into:

$$S_{PtR} = \sum Diag(\mathbf{C}_{PtR}), \quad S_{ft} = \sum Diag(\mathbf{C}_{ft}). \quad (8)$$

Feeding in 584 validation images, we had S_{PtR} = 425 and S_{ft} = 404, which indicates that the pseudo-task regularized model shows more certainty in the correct classes on average than the fine-tuned model. Furthermore, in terms of the average entropy of the predictions, the pseudo-task regularizer reduced it from 1.33 to 1.15 bits. This is due to better regularization that allows the model to eliminate minor probabilities in false predictions, which in turn reduced

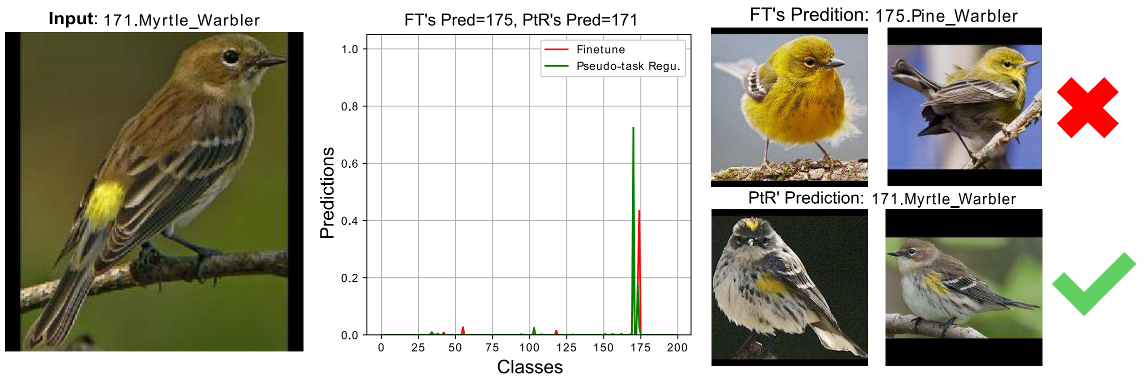

Figure 2. A sample from the validation set of CUB200 that PtR correctly rectified mis-classification caused in the vanilla fine-tuning. Left: An input image. Middle: Categorical distributions made by the baseline model and PtR. The second largest prediction made by FT baseline model is around 30% for Class 171. Right: Two randomly picked training samples of the class predicted by the baseline (top) and two samples of the class predicted by PtR (bottom).

the average entropy. The reduction of entropy also implies that the class predictions have been disambiguated by PtR.

Correspondingly we computed S'_{PtR} and S'_{ft}, the sum of off-diagonal elements of $\mathbf{C}_{PtR}$ and $\mathbf{C}_{ft}$, respectively. We observed that the regularized model tends to make predictions with fewer minor probabilities than the vanilla fine-tuning model. It gives higher certainty to its predictions on the ground truth class given the smaller S'_{PtR}, which is consistent with the reduced entropy in comparison to vanilla fine-tuning.

To further qualitatively study how the predictions made by the vanilla fine-tuning model (baseline) have evolved by PtR, we case study two types of input samples for which:

i) PtR rectified the baseline model's errors, and

ii) PtR mis-classified on the contrary to the baseline.

Namely, i) is *true rectification* and ii) is *false rectification*. Two of these examples are compared in Figure 2 and 3.

It can be seen that PtR has an effect of encouraging the predictions of the instances of other classes which are visually close to the ground truth class[4]. This indicates that the pseudo task regularizer implicitly helps the network focus on and learn to distinguish classes of higher visual similarity, and hence helps the classification accuracy overall (for around 1% on CUB200's validation set). At the same time, from our observations, PtR does not tend to produce so many minor probabilities on other classes which are less similar as the baseline model does. This aids the regularized model to suppress the uncertainties and focus on a few most similar candidate classes.

5. Related Work

Regularization methods that are commonly used for fine-tuning CNNs can be generally grouped into four categories: data perturbation, parameter norm penalty, dropout, and multi-task learning. Image augmentation as a popular form of image perturbation has been proven to be particularly useful to prevent CNNs from overfitting. In this paper, we also assume to perturb our training instances by random augmentations. The supervision signal can be perturbed for better regularization as well. This can be achieved by learning to predict soft targets rather than hard binary ones as in [36]. In this work, label perturbation is not considered so that we can deliver more ablated studies on the effectiveness of auxiliary training objectives.

The parameter norm penalty, or *weight decay* more specifically, has been one of the most common ways of regularization in training deep models. Our PtR leveraged weight decay by default, but we also evaluated PtR without weight decay to study its impact on accuracy. Another method being apparently similar to weight decay is to use feature norm penalty (FPN) or feature contraction on the representation layer of a network [10, 19, 25]. Superficially, FNP would resemble to PtR if the regression target for PtR was towards a static norm of zero without involving randomness. (An additional feature of PtR which should be noted is that it also balances objectives automatically. See Section 2.3 for the technical difference.) *Dropout* is also one of the standard techniques to improve model regularization by temporarily shielding a part of the hidden units in the bottleneck layer and fully connected layers during training [32, 26]. For the VGG-16 structures which we employed in our experiments, dropout was also used after flattened hidden layers.

A more recent approach of regularization in transfer learning has been to train CNNs with an auxiliary task/objective through multi-task learning [21, 6]. These objectives in the recent best performing methods are often designed with the expectation that more generic features are less likely to overfit to the target task in a few different ways. In [6], the network was simultaneously trained by the target data and a number of selected source data samples that

[4]In Figure 2, the second largest prediction made by PtR is at another class of similar appearance; in Figure 3, the wrongly predicted class is also similar to the ground truth.

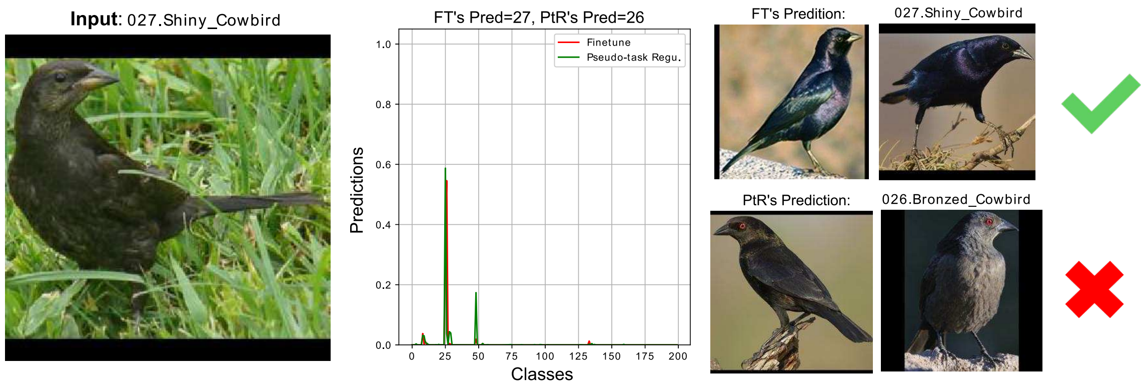

Figure 3. A sample from the validation set of CUB200 that the standard fine-tuning correctly classified but PtR wrongly predicted. See also the caption of Figure 2.

are similar to the target data when viewed in low-level features. Another way to encourage CNN regularization, instead of relying on the availability of foreign data, was to let the model stay not too far from the original structure of the model trained by a large source task. As demonstrated by [21], one can attempt to retain the predictions of the target domain images made by the off-the-shelf source model while learning on the new target task (we acknowledge that the original intention of doing so in [21] was not for regularization). It is therefore reasonable to interpret the use of off-the-shelf predictions as implicitly using the source domain training data. The other way to make the trained models attracted to the original model was to explicitly force the target model being trained to stay in a vicinity of the source model in the weight space; the work in [20] leveraged the inductive bias in a fine-tuning scenario to prevent the learned features from becoming overly specific.

For fine-grained vision tasks, a very recent approach [5] suggested to "confuse" the network by encouraging different class-conditional probability distributions to come closer together, thus reducing the inter-class distance. In the cases where only style transfer is considered, one can attempt to reduce the domain variance through certain metrics (i.e., perform domain adaptation) as in [34, 24].

A common aspect of these aforementioned regularizers is that they depend on a concrete task or objective. But since they are not purposed for explicitly optimizing the target objective, they can all be largely seen as "distractors". The approach studied in this paper also causes distractions, but we suggest that a distractor could work equally effectively without involving a concrete auxiliary objective or any form of source domain data as supervision labels.

6. Conclusions and Discussions

We have introduced a *Pseudo-task Regularization* (PtR) which leverages a multi-task learning framework to generate useful interference by a pseudo-regression task, for improving regularization for transfer learning with limited data samples. The regularization effect from PtR is dynamic in that PtR adjusts the strength of the regularization based on the gradient norms of the target objective and the pseudo-task. Unlike existing approaches, PtR does not depend on a concrete or real regularization objective. Surprisingly, we observed that the performance gain brought about by the simple PtR was on a par or better than the related recent solutions, and therefore we suggest that PtR is available as an efficient alternative to recent best performing regularization methods that are based on concrete objectives.

We attribute the generalization gain from PtR to two aspects while a widely recognized theory on DNN generalization is yet to be established [14, 42]. First, the gradient noise generated by PtR in SGD makes batch-wise gradients noisier and results in smaller equivalent batch size in that respect. The resulting increased quotient of learning rate and (equivalent) batch size have been shown helpful to escape sharp minima [14, 17]. The nosier gradients prolongs training which could in turn encourage networks exploring more and better local minima, hence class disambiguation. Second, the superiority of anisotropic noise of SGD over its isotropic counterpart has been demonstrated [42], and the anisotropic gradient noise generated by PtR could increase the chance to find flatter minima.

Furthermore, it will be also interesting to study the impact of PtR on the calibration performance of the network [9] , and how PtR's gradients interact with those of the baseline as in [41]. This is however beyond the scope of the current work as well as the above mentioned relation to the loss landscape – those are among the topics for future research.

Acknowledgement

We would like to thank Ryuzo Okada and colleagues of Toshiba Corporate R&D Center for funding the research and for collaboration. We also wish to thank Vladimir Li and Matteo Gamba for fruitful discussions, and NVIDIA Corporation for their generous donation of GPUs. The second author is supported by the Swedish Research Council, which is gratefully acknowledged.

References

[1] Hossein Azizpour, Ali Sharif Razavian, Josephine Sullivan, Atsuto Maki, and Stefan Carlsson. Factors of transferability for a generic convnet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2016.

[2] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning (ICML)*, 2018.

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

[4] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference on Machine Learning (ICML)*, 2014.

[5] Abhimanyu Dubey, Otkrist Gupta, Pei Guo, Ramesh Raskar, Ryan Farrell, and Nikhil Naik. Pairwise confusion for fine-grained visual classification. In *European Conference on Computer Vision (ECCV)*, 2018.

[6] Weifeng Ge and Yizhou Yu. Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: training imagenet in 1 hour. *Arxiv preprint 1706.02677*, 2017.

[8] Gregory Griffin, Alex Holub, and Pietro Perona. Caltech-256 object category dataset. Technical report, California Institute of Technology, 2007.

[9] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q. Weinberger. On calibration of modern neural networks. In *International Conference on Machine Learning (ICML)*, 2017.

[10] Bharath Hariharan and Ross Girshick. Low-shot visual recognition by shrinking and hallucinating features. In *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[12] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint 1502.03167*, 2015.

[14] Stanislaw Jastrzebski, Zachary Kenton, Devansh Arpit, Nicolas Ballas, Asja Fischer, Yoshua Bengio, and Amos Storkey. Three factors influencing minima in SGD. *Arxiv 1711.04623*, 2018.

[15] Vahdat Kazemi and Josephine Sullivan. One millisecond face alignment with an ensemble of regression trees. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[16] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: A convolutional network for real-time 6-dof camera relocalization. In *Proc. of the International Conference on Computer Vision (ICCV)*. 2015.

[17] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2017.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 2012.

[19] Vladimir Li and Atsuto Maki. Feature contraction: New convnet regularization in image classification. In *British Machine Vision Conference (BMVC), Newcastle, UK*, 2018.

[20] Xuhong Li, Yves Grandvalet, and Franck Davoine. Explicit inductive bias for transfer learning with convolutional networks. In *International Conference on Machine Learning (ICML)*, 2018.

[21] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *European Conference on Computer Vision (ECCV)*, 2016.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, 2014.

[23] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[24] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael Jordan. Learning transferable features with deep adaptation networks. In *International Conference on Machine Learning (ICML)*, 2015.

[25] Atsuto Maki. Toward principled regularization of deep networks—from weight decay to feature contraction. *Science Robotics*, 4(30), 2019.

[26] Pietro Morerio, Jacopo Cavazza, Riccardo Volpi, Rene Vidal, and Vittorio Murino. Curriculum dropout. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[27] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Proc. of the Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.

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

[29] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[30] Ariadna Quattoni and Antonio Torralba. Recognizing indoor scenes. *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, 2009.

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

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 2014.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *The IEEE International Conference on Computer Vision (ICCV)*, 2014.

[34] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. In *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[35] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, and Serge Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[36] Lingxi Xie, Jingdong Wang, Zhen Wei, Meng Wang, and Qi Tian. Disturblabel: Regularizing cnn on the loss layer. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[37] Bangpeng Yao, Xiaoye Jiang, Aditya Khosla, Andy Lai Lin, Leonidas Guibas, and Li Fei-Fei. Human action recognition by learning bases of action attributes and parts. In *The IEEE International Conference on Computer Vision (ICCV)*, 2011.

[38] Dong Yi, Zhen Lei, Shengcai Liao, and Stan Z Li. Learning face representation from scratch. *Arxiv preprint 1411.7923*, 2014.

[39] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in Neural Information Processing Systems 27*. 2014.

[40] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, 2014.

[41] Xiangyun Zhao, Haoxiang Li, Xiaohui Shen, Xiaodan Liang, and Ying Wu. A modulation module for multi-task learning with applications in image retrieval. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[42] Zhanxing Zhu, Jingfeng Wu, Bing Yu, Lei Wu, and Jinwen Ma. The anisotropic noise in stochastic gradient descent: Its behavior of escaping from sharp minima and regularization effects. In *Proc. of the Interational Conference on Machine Learning (ICML)*, 2019.