Learning from Noisy Labels with Deep Neural Networks

Sainbayar Sukhbaatar
Dept. of Computer Science,
Courant Institute,
New York University
sainbayar@cs.nyu.edu

Rob Fergus
Facebook AI Research &
Dept. of Computer Science,
Courant Institute
New York University
fergus@cs.nyu.edu

Abstract

We propose several simple approaches to training deep neural networks on data with noisy labels. We introduce an extra noise layer into the network which adapts the network outputs to match the noisy label distribution. The parameters of this noise layer can be estimated as part of the training process and involve simple modifications to current training infrastructures for deep networks. We demonstrate the approaches on several datasets, including large scale experiments on the ImageNet classification benchmark, showing how additional noisy data can improve state-of-the-art recognition models.

1 Introduction

In recent years, deep learning methods have shown impressive results on image classification tasks. However, this achievement is only possible because of large amount of labeled images. Labeling images by hand is a laborious task and takes a lot of time and money. An alternative approach is to generate labels automatically. This includes user tags from social web sites and keywords from image search engines. Considering the abundance of such noisy labels, it is important to find a way to utilize them in deep learning. Unfortunately, those labels are very noisy and unlikely to help training deep networks without additional tricks.

Our goal is to study the effect label noise on deep networks, and explore simple ways of improvement. We focus on the robustness of deep networks instead of data cleaning methods, which are well studied and can be used together with robust models directly. Although many noise robust classifiers are proposed so far, there are not many works on training deep networks on noisy labeled data, especially on large scale datasets.

Our contribution in this paper is a novel way of modifying deep learning models so they can be effectively trained on data with high level of label noise. The modification is simply done by adding a linear layer on top of the softmax layer, which makes it easy to implement. This additional layer changes the output from the network to give better match to the noisy labels. Also, it is possible to learn the noise distribution directly from the noisy data. Using real-world image classification tasks, we demonstrate that the model actually works very well in practice. We even show that random images without labels (complete noise) can improve the classification performance.

2 Related Work

In any classification model, degradation of performance is inevitable when there is noise in training labels [13, 15]. A simple approach to handle noisy labels is a data preprocessing stage, where labels suspected to be incorrect are removed or corrected [1, 3]. However, a weakness of this approach is the difficulty of distinguishing informative hard samples from harmful mislabeled ones [6]. Instead, in this paper, we focus on models robust to presence of label noise.

The effect of label noise is well studied in common classifiers (e.g., SVMs, kNN, logistic regression), and their label noise robust variants have been proposed. See [5] for comprehensive review. A more recent work [2] proposed a generic unbiased estimator for binary classification with noisy labels. They employ a surrogate cost function that can be expressed by a weighted sum of the original cost functions, and gave theoretical bounds on the performance. In this paper, we will also consider this idea and extend it multiclass.

A cost function similar to ours is proposed in [2] to make logistic regression robust to label noise. They also proposed a learning algorithm for noise parameters. However, we consider deep networks, a more powerful and complex classifier than logistic regression, and propose a different learning algorithm for noise parameters that is more suited for back-propagation training.

Considering the recent success of deep learning [8, 17, 16], there are very few works about deep learning from noisy labels. In [11, 9], noise modeling is incorporated to neural network in the same way as our proposed model. However, only binary classification is considered in [11], and [9] assumed symmetric label noise (noise is independent of the true label). Therefore, there is only a single noise parameter, which can be tuned by cross-validation. In this paper, we consider multiclass classification and assume more realistic asymmetric label noise, which makes it impossible to use cross-validation to adjust noise parameters (there can be a million parameters).

3 Approach

In this paper, we consider two approaches to make an existing classification model, which we call the *base model*, robust against noisy labels: bottom-up and top-down noise models. In the bottom-up model, we add an additional layer to the model that changes the label probabilities output by the base model so it would better match to noisy labels. Top-down model, on other hand, changes given noisy labels before feeding them to the base model. Both models require a noise model for training, so we will give an easy way to estimate noise levels using clean data. Also, it is possible to learn noise distribution from noisy data in the bottom-up model. Although only deep neural networks are used in our experiments, the both approaches can be applied to any classification model with a cross entropy cost.

3.1 Bottom-up Noise Model

We assume that label noise is random conditioned on the true class, but independent of the input $\mathbf{x}$ (see [10] for more detail about this type of noise). Based on this assumption, we add an additional layer to a deep network (see Figure 1) that changes its output so it would better match to the noisy labels. The weights of this layer corresponds to the probabilities of a certain class being mislabeled to another class. Because those probabilities are often unknown, we will show how estimate them from additional clean data, or from the noisy data itself.

Let D be the true data distribution generating correctly labeled samples $(\mathbf{x}, y^*)$, where $\mathbf{x}$ is an input vector and y^* is the corresponding label. However, we only observe noisy labeled samples $(\mathbf{x}, \tilde{y})$ that generated from a some noisy distribution $\tilde{D}$. We assume that the label noise is random conditioned on the true labels. Then, the noise distribution can be parameterized by a matrix $Q = \{q_{ji}\}$: $q_{ji} := p(\tilde{y} = j | y^* = i)$. Q is a probability matrix because its elements are positive and each column sums to one. The probability of input $\mathbf{x}$ being labeled as j in $\tilde{D}$ is given by

$$p(\tilde{y} = j|\mathbf{x}, \theta) = \sum_i p(\tilde{y} = j|y^* = i)p(y^* = i|\mathbf{x}) = \sum_i q_{ji} p(y^* = i|\mathbf{x}, \theta). \tag{1}$$

where $p(y^* = i|\mathbf{x}, \theta)$ is the probabilistic output of the base model with parameters θ. If the true noise distribution is known, we can modify this for noisy labeled data. During training, Q will act as an adapter that transforms the model's output to better match the noisy labels.

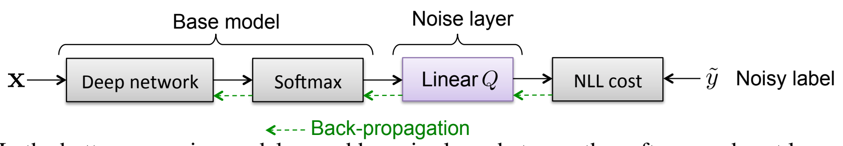


Figure 1: In the bottom-up noise model, we add a noise layer between the softmax and cost layers. The noise layer is a special linear layer with weights equal to the noise distribution. It changes output probabilities from the base model into a distribution that better matches to noisy labels.

If the base model is a neural network with a softmax output layer, this modification can be simply done by adding a *noise layer* on top of the softmax layer, as shown in Figure 1. The role of the noise layer is to perform operations in Eqn. 1. Therefore, the noise layer is a normal linear layer except there is no bias and its weights are constrained between 0 and 1 because they represent conditional probabilities q_{ji}. Also, weights coming from a single node should sum to one because $\sum_j q_{ji} = 1$.

When training labels are clean, we would set the weight matrix of the noise layer to identity, which is equivalent to not having the noise layer. For noisy labels, we would use noise distribution Q instead of the identity matrix. Because the Q matrix is linear, we can pass gradients through it during training and perform back-propagation to train the rest of the network. As before, the learning objective is to maximize the log likelihood over N samples, but now the objective incorporates the noise matrix Q:

$$\mathcal{L}(\theta) = \frac{1}{N}\sum_{n=1}^{N} \log p(\tilde{y} = \tilde{y}_n | \mathbf{x}_n, \theta) = \frac{1}{N}\sum_{n=1}^{N} \log\left(\sum_i q_{\tilde{y}_n i}\, p(y^* = i | \mathbf{x}_n, \theta)\right) \tag{2}$$

3.2 Estimating Noise Distribution Using Clean Data

In the bottom-up noise model, we need to know the noise distribution Q in order to train the network. Unfortunately, Q is often unknown for real-world tasks. Because the number of free parameters in Q is the square of the number of classes, it is impossible to find Q using cross-validation methods for large-scale datasets. However, we can get unbiased estimation of Q if some clean data is available in addition to the noisy data, which is often the case. The idea is to use the clean data to get the confusion matrix of a some pre-trained model. We also can measure the confusion matrix on the noisy data. Then, the difference between those two confusion matrices should be the noise distribution Q. This is because the model's mistakes on the noisy data is a combination of two types of mistakes: (1) mistakes of the model and (2) mistakes in noisy labels. The statistics of the first type mistakes can be measured by the clean data, and the second type mistakes correspond to the noise distribution.

Let us formulate this more precisely. We have clear data D^* and noisy data $\tilde{D}$. The goal is to estimate the noise distribution of $\tilde{D}$. We also need a some pre-trained model M, which could have been trained on either noisy or clear data (but it should be separate from D and $\tilde{D}$). Let C^* and $\tilde{C}$ be the confusion matrices of M on the two data types

$$D^* \xrightarrow{M} C^* : \;\; c^*_{ij} = p(y = i | y^* = j, M) \quad \text{and} \quad \tilde{D} \xrightarrow{M} \tilde{C} : \;\; \tilde{c}_{ij} = p(y = i | \tilde{y} = j, M).$$

Then, the relation between those two is: $\sum_i c^*_{ki} r_{ij} = \tilde{c}_{kj}$ for $\forall j, k$, where r_{ij} denotes $p(y^* = i | \tilde{y} = j)$. If we use a matrix form, we can write:

$$C^* R = \tilde{C} \implies R = {C^*}^{-1}\tilde{C}, \tag{3}$$

when C^* is an invertible matrix. If computed R has negative values, it should be projected back to the subspace of probability matrices. Finally, we can compute Q from R using Bayes' rule

$$p(\tilde{y} = j | y^* = i) = \frac{p(y^* = i | \tilde{y} = j) p(\tilde{y} = j)}{p(y^* = i)} \quad \implies \quad q_{ji} = \frac{r_{ij} p(\tilde{y} = j)}{p(y^* = i)}. \tag{4}$$

We can measure $p(\tilde{y} = j)$ from the data, and $p(y^* = i)$ can be computed from the fact $\sum_j q_{ji} = 1$.

In case when the number of clean samples is small relative to the number of elements in R, using the inverse of C^* is not good idea because the inverse operation is unstable in presence of noise. Instead we can put a sparsity prior on R and solve the following optimization problem

$$\min_R (0.5\|C^* R - \tilde{C}\|^2 + \lambda |R|) \tag{5}$$

under constraint that R should be a probability matrix. Here λ is a hyper-parameter controlling the L1 sparsity of R, which can determined by cross-validation. We can effectively solve this optimization with simple gradient descend method. Such sparse prior over R and Q is useful because in real-world data classes are likely only to be mislabeled with a small set of other classes.

3.3 Learning Noise Distribution From Noisy Data

In practice, the noise distribution Q is often unknown and we might not have clear data from which to estimate it. If we only have noisy training data, then we have to learn $\hat{Q}$, an approximation of Q, from the noisy data itself. Since the elements of $\hat{Q}$ correspond the weights of the noise layer in our model, it can be learned in the same way as other weights using back-propagation. However, weight matrix $\hat{Q}$ have to be projected back to the subspace of probability matrices after each update.

A problem with this is that there is no guarantee $\hat{Q}$ would converge true Q. The combined model can estimate the noise Q via the product of $\hat{Q}$ and C, the base model confusion matrix: $\hat{Q}C = Q$. Thus, if the base model is powerful enough, it can learn the noise distribution and $\hat{Q}$ will be an identity matrix. To prevent this, we add a regularizer $tr(\hat{Q})$ to the objective, which in turn makes the base model less noisy, so encouraging $\hat{Q}$ to converge to Q, see Theorem 1.

Theorem 1. *In the following optimization problem, the only global minimum is $\hat{Q} = Q$ and $C = I$. (where Q, $\hat{Q}$ and C are probability matrices).*

$$\underset{\hat{Q},C}{\textit{minimize}} \quad tr(\hat{Q}) \quad \textit{subject to} \quad \hat{Q}C = Q,\ \hat{q}_{ii} > \hat{q}_{ij},\ q_{ii} > q_{ij} \quad \textit{for}\ \forall i, j \neq i.$$

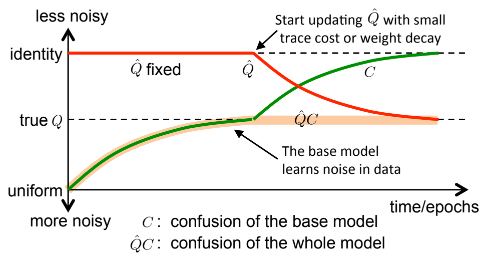
Proof. Let Q^* be the global solution. Then

$$tr(Q) = tr(Q^*C) = \sum_i(\sum_j q^*_{ij}c_{ji}) \leq \sum_i(\sum_j q^*_{ii}c_{ji}) = \sum_i q^*_{ii}(\sum_j c_{ji}) = \sum_i q^*_{ii} = tr(Q^*)$$

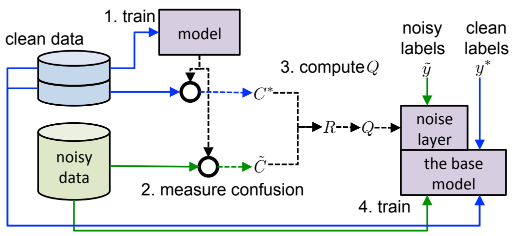
The equality will only hold true only when $C = I$. Therefore, $Q^* = \hat{Q}$. □

Before the model is converged, $\hat{Q}C$ may only approximately equal Q, but empirically we show that the $tr(\hat{Q})$ regularization works well. In practice, we use weight decay on $\hat{Q}$ since it is already implemented in most deep learning packages and has the same effect.

3.4 Training a Bottom-up Model



(a)



(b)

Figure 2: (a) The training sequence when learning from noisy data alone. The noise matrix $\hat{Q}$ (red) is initially set to the identity, while the base model (green) is trained, inadvertently learning the noise in the data. Then we start updating $\hat{Q}$ also (with regularization) and this captures the noise properties of the data, leaving the model to make "clean" predictions. (b) Training when we have noisy and clean data available. 1: We train a model one on the clean data. 2 & 3: Comparing the confusion matrices on clean/noisy data, we compute Q. 4: We train a new model with the noise layer Q fixed.

Noisy labels only: We first consider the case where we only have noisy labeled training data. We start by initializing the base model in usual way (there are no special requirements). Initially, we fix $\hat{Q} = I$ during training until the validation error stops decreasing. At this point, the base model could have learned the noise in the training data, which is not what we want. Therefore, we start updating $\hat{Q}$ along with the rest of the network, using weight decay to push $\hat{Q}$ toward Q. As $\hat{Q}$ gets more noisy, it should absorb the noise from the base model, thus making the base model more accurate. However, too large weight decay (or trace cost) would make $\hat{Q}$ noisier than the true Q, which will hurt the performance. Therefore, we have to find right amount of weight decay using cross validation. We continue the training until the validation error stops decreasing to prevent overfitting. This procedure is illustrated in figure 2(a). If we want to make prediction or test the model on clear data, the noise layer should be removed (or set to identity I), but for noisy labeled validation data we use the learned $\hat{Q}$.

Noisy and clean data: If some clean training data is available, we can use it to estimate the noise distribution Q of the noisy data, using the method described in section 3.2. We first train a model (which can be a normal deep network) on a subset of the clean data. Then, we measure the confusion matrices on the both clean (must exclude the subset used for training to avoid bias) and noisy data. From the difference between those two matrices, we compute an estimate of the noise distribution Q using Eqns. 3 & 4. This can then be used to train a new model on both clean and noisy dataset. Figure 2(b) shows the training scheme. A variant of this procedure involves one further stage where the initial estimate of Q is refined using the procedure in Section 3.3.

3.5 Top-down Noise Model

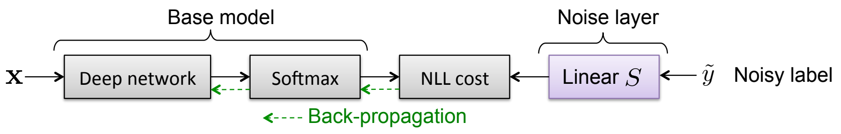


Figure 3: In the top-down noise model, noisy labels are converted by matrix S before being used by the base model for training.

Instead of modifying the model to match with noisy labels, we can change the noisy labels so that it should produce an unbiased classification. Given a noisy label is i, we replace it with vector label $\mathbf{s}_i$ (see Figure 3). Let S be the conversion matrix consisting from column vectors $\mathbf{s}_i$. The learning objective is still to maximize the log likelihood, but now noisy labels are converted by matrix S.

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{K} s_{i\tilde{y}_n} \log p(y = i|\mathbf{x}_n, \theta), \tag{6}$$

where K is the number of classes. Note the difference between Eqn. 6 and 2 is that the sum over classes being inside or outside the log operator. Unfortunately, we cannot optimize this objective with respect to S, because there is a degenerate solution where S converts all labels to one class and the model always predicts that class. Therefore, we cannot learn S directly from the noisy data.

An unbiased estimator for binary classification is introduced in [12], where the cost function is replaced by a surrogate objective that combines the two costs (costs for class +1 and -1) with coefficients that depends on the noise level. We can view this as replacing noisy labels with different label vectors. If we generalize this surrogate function to multi classes, we can see that S should be the inverse of the noise distribution Q (or at least QS should be equal to an identity plus some constant). However, the inverse operation is unstable with respect to noise, and also it usually contain negative elements when Q is a probability matrix, which can make Eqn. 6 diverge to infinity.

As there is no practical way to learn S reliably, in our experiments we fix it to $\alpha \cdot I + (1-\alpha)/K \cdot \mathbf{1}$, where $\mathbf{1}$ is a $K \times K$ matrix of ones. The hyperparameter α is selected by cross-validation.

3.6 Reweighting of Noisy Data

In some experiments in this paper, we have clean data in addition to noisy data. If we just mix them, we lose the valuable information about which labels are trustworthy. A simple way to express the relative confidence between the two sets of data is to down-weight the noisy examples, relative to the clean, in the loss function, as shown in Eqn. 7. This trick can be combined with both the bottom-up and top-down noise models described above. The objective is to maximize

$$\mathcal{L}(\theta) = \frac{1}{N_c + N_n} \left(\sum_{n=1}^{N_c} \log p(y = y_n|\mathbf{x}_n, \theta) + \gamma \sum_{n=1}^{N_n} \log p(\tilde{y} = \tilde{y}_n|\tilde{\mathbf{x}}_n, \theta) \right) \tag{7}$$

where N_c and N_n are the number of clean and noisy samples respectively. The hyper-parameter γ is the weight on noisy labels and is set by cross validation.

4 Experiments

In this section, we empirically examine the robustness of deep networks with and without noise modeling. We experiment on several different image classification datasets with label noise. As the base model, we use convolutional deep networks because they produce state-of-art performance on many image classification tasks.

This section consists from two parts. In the first part, we perform controlled experiments by deliberately adding label noise to clean datasets. This is done by randomly changing some labels in the training data according to some known noise distribution. This allows us to check if the learned noise distribution is close the ground-truth noise distribution. First, we experiment on the Google street-view house number dataset (SVHN) [14], which consists of 32x32 images of house number digits captured from Google Streetview. It has about 600k images for training and 26k images for testing. Next, we experiment on CIFAR-10 [7], a more challenging dataset consisting from 60k small images of 10 object categories. The both datasets are hand-labeled, so all labels are clean.

In the second part, we show more realistic experiments using two datasets with inherent label noise. The first dataset consist from clean images from CIFAR-10 dataset and noisy images from Tiny Images dataset [18]. The second dataset consists of clean images from ImageNet [4] and noisy images downloaded from web search engines. In the both datasets, we do not know the true noise distribution of noisy labels.

4.1 Deliberate Label Noise

We synthesize noisy data from clean data by deliberately changing some of the labels. Original label i is randomly changed to j with fixed probability q_{ji}. An example of an noise distribution $Q = \{q_{ji}\}$ we use is shown in Figure 4(c). By changing the probability on the diagonal, we can generate datasets with different noise levels. The labels of test images are left unperturbed.

We use a publicly available fast GPU code[1] for training deep networks. As the base model, we use their "18% model" with three convolutional layers (layers-18pct.cfg) for both SVHN and CIFAR-10 experiments. No data augmentation is done. The only data preprocessing was the contrast normalization for SVHN images.

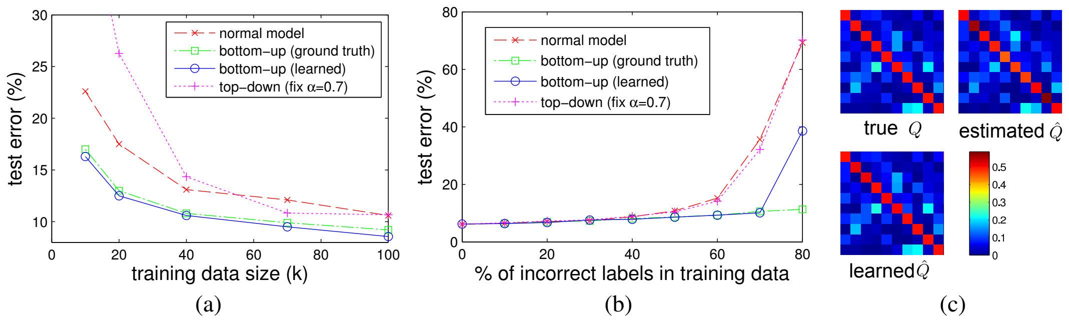


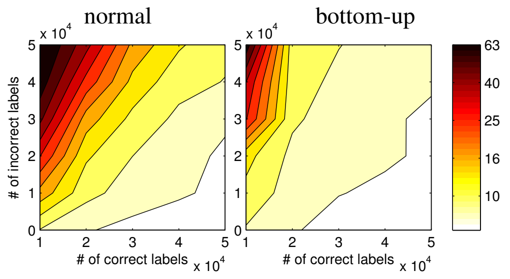
Figure 4: (a) Test errors on SVHN dataset when noise level is 50% (b) Test errors when trained on 100k samples. (c) The true noise distribution Q for 50% noise compared with learned and estimated $\hat{Q}$.

SVHN noisy only: When training a bottom-up model with SVHN data, we fix $\hat{Q}$ to identity for the first five epochs. Then, $\hat{Q}$ is updated with weight decay 0.05 for 100 epochs. Figure 4(a) and (b) shows the test errors for different training data sizes and different noise levels. Compared with a normal deep network, the bottom-up model always achieves better accuracy. However, the top-down model do not work well for any value of α (even using the true Q does not improve).

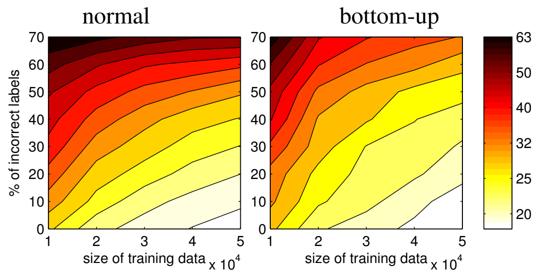
In Figure 4(a) and (b), we also plot error rates for a bottom-up model trained using the true noise distribution Q. We see that it performs as well as the learned $\hat{Q}$, showing that our proposed method for learning the noise distribution from data is effective. Figure 4(c) shows one such learned $\hat{Q}$ alongside the ground truth Q used to generate the noisy data. We can see that the difference between them is negligible.

Figure 5(a) shows the effect of label noise on performance in more detail for SVHN data. For a normal deep network, the performance drops quickly as the number of incorrect labels increases. In contrast, the bottom-up model shows more robustness against incorrect labels. For example, in the normal model, training on 50k correct + 40k incorrect labels is the same as training on 10k correct

[1]`https://code.google.com/p/cuda-convnet/`



(a) SVHN



(b) CIFAR-10

Figure 5: The effect of incorrect labels on the test error (%) for an unmodified network ("normal") and one with our bottom-up noise layer appended.

labels. However, we can achieve the same performance using 30k correct and 50k incorrect labels with the bottom-up model.

CIFAR-10 noisy only: We perform the same experiments as for SVHN on the CIFAR-10 dataset, varying the training data size and noise level. We fix $\hat{Q}$ to identity for the first 50 epochs of training and then run for another 70 epochs updating $\hat{Q}$ with weight decay of 0.05 or 0.1 (selected on validation set). The results are shown in Figure 5(b). Again, the bottom-up model is more robust to label noise, compared to the unmodified model. The difference is especially large for high noise levels and large training sets, which shows the scalability of the bottom-up model.

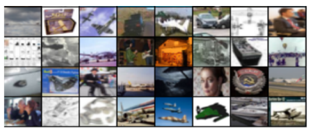
CIFAR-10 clean + noisy: Next, we consider the scenario where both noisy and clean data are available. We now are able to use the approach from Section 3.2 to estimate the noise distribution Q using clean/noisy data, instead of learning it. We prepare two subsets of data: 20k clean images for estimating Q, and 30k noisy images for training the final model. First, we train a normal network with 10k clean data, which gives 30% test error. Then, we measure two confusion matrices, one on the other 10k clean data and the other on 30k noisy data. From these we compute an estimated $\hat{Q}$, which is used to train a new bottom-up model using only noisy data. An estimated $\hat{Q}$ for 50% noise is shown in Figure 4(c), where it is very close to the ground truth. Table 1 compares the classification performance using learned and estimated $\hat{Q}$ for two different noise levels within the 30k noisy set. The estimated $\hat{Q}$ is as effective as the true Q, even at high noise levels.

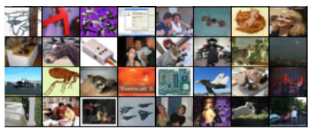
Model	normal	true Q	learned $\hat{Q}$	estimated $\hat{Q}$
Test error (50% noise)	38%	28%	30%	29%
Test error (70% noise)	60%	35%	40%	35%

Table 1: Test errors when trained on 30k images from CIFAR-10 with different noise levels

4.2 CIFAR-10 + Tiny Images

CIFAR-10 was originally created by cleaning up a subset of Tiny Images, a dataset of loosely labeled 80M 32x32 images (more than half of labels are estimated to be incorrect). In the process of hand picking CIFAR-10 images, some images were excluded because they did not show the object clearly or were labeled falsely. Our training data consists of two subsets: 50k clean labeled images from CIFAR-10 training data and 150k noisy labeled images from the excluded set of Tiny Images. As shown in Figure 6, those extra training images have very noisy labels, and often contain objects not in the 10 categories.


airplane


cat

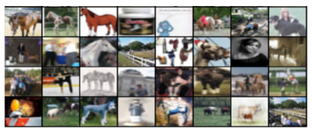

horse

Figure 6: Sample images from the extra training data

We use a model architecture similar to the Krizhevsky's "18% model", but with more feature maps in the three layers $(32, 32, 64 \rightarrow 64, 128, 128)$ to accomodate the larger training set. No data augmentation is used. During training, we alternate mini batches from two data sources (when the current mini batch is from clear data, the noise modeling is removed from the network). The noise matrix is fixed to identity during the first 50 epochs of the bottom-up model. Then the noise matrix is updated simultaneously with no weight decay. In the top-down model, however, the noise matrix is always fixed to $\alpha = 0.5$. The network is evaluated on the original CIFAR-10 test set (which has clear labels).

Table 2 shows test errors for several noise modeling approaches. In most cases, weighting the noisy labels with $\gamma = 0.2$ helps performance, as shown in Figure 7 and row 4 of Table 2. The bottom-up model performance is mediocre, most likely due to the large fraction of outside images (i.e., showing objects not in the 10 categories) present in the Tiny Images, which violate the noise model. By contrast, the top-down model imposes a more uniform label distribution on these outside images, and so works well in practice. We test this hypothesis by training with a new 150k image set, randomly drawn from the entire Tiny Image dataset (not just the excluded set) and having uniform labels (equivalent to top-down with $\alpha = 0.1$) and we see that these also give good test performance. Although we can treat outside images as a separate class, assigning them uniform labels works better in practice. It can be considered as novel way of regularizing deep networks by random images, which are cheap to obtain. However, it is important that random images are not constrained within training categories.

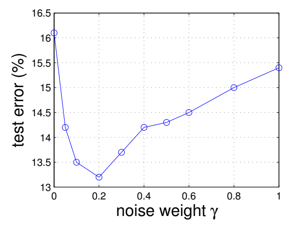


Figure 7: Test error dependency on noise weight γ

Model	Extra data	Noisy weight γ	Test error
Conv. net	-	-	16.1%
Conv. net	150k noisy	1	15.4%
Conv. net		0.2	13.2%
Bottom-up		0.2	13.2%
Top-down		0.4	12.5%
Conv. net	150k random	0.2	13.8%

Table 2: Test error on CIFAR-10 + Tiny images.

4.3 ImageNet + Web Image Search

We perform a large-scale experiment using the ImageNet 2012 dataset which has 1.3M image with clean labels over 1000 classes. We obtain a further noisy set of 1.4M images, scraped from Internet image search engines using the 1k ImageNet keywords. Overlapping images between the two sets were removed from the noisy set using cross-correlation. We trained the model of Krizhevsky *et al.* [8] on the clean and combined datasets, with several types of noise modeling. Table 3 shows that training with the combined dataset does not improve the performance compared to training on the clean dataset. But just weighting the noisy labels (using $\gamma = 0.1$) gives an improvement of 1.4%. This can be improved slightly with the bottom-up noise model. However, the absolute performance achieved with our techniques and the 1.4M additional noisy data equals the performance when training on an *additional 15M clean images* from ImageNet 2011 (row 3). This demonstrates that noisy data can be very beneficial for training. Note that no model averaging is done in those experiments.

Model	Extra data	Noisy weight γ	Top 5 val. error
Krizhevsky et al. [8]	-	-	18.2%
Krizhevsky et al. [8]	15M full ImageNet	-	16.6%
Conv. net	-	-	18.0%
Conv. net	1.4M noisy images from Internet	1	18.1%
Conv. net		0.1	16.7%
Bottom-up (learned)		0.1	16.5%
Bottom-up (estimated)		0.2	16.6%

Table 3: Error rates of different models on validation images of ImageNet

5 Conclusion

In this paper, we proposed two models for learning from noisy labeled data with deep networks, which can be implemented with minimal effort in existing deep learning implementations. Our experiments show that this model can reliably learn the noise distribution from data, under reasonable conditions. In addition, a simple technique to estimate the noise distribution using clean data is proposed. We also found that, if noisy and clean training sets exist, simply down-weighting the noisy set can help significantly. Both these techniques were demonstrated on large scale experiments, showing significant gains over training on clean data alone. Another surprising finding was that random images can be used to regularize deep networks and improve the performance significantly.

References

[1] R. Barandela and E. Gasca. Decontamination of training samples for supervised pattern recognition methods. In *Advances in Pattern Recognition*, volume 1876 of *Lecture Notes in Computer Science*, pages 621–630. Springer, 2000.

[2] J. Bootkrajang and A. Kabn. Label-noise robust logistic regression and its applications. In *Machine Learning and Knowledge Discovery in Databases*, volume 7523 of *Lecture Notes in Computer Science*, pages 143–158. Springer, 2012.

[3] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255, June 2009.

[5] B. Frenay and M. Verleysen. Classification in the presence of label noise: A survey. *Neural Networks and Learning Systems, IEEE Transactions on*, 25(5):845–869, May 2014.

[6] I. Guyon, N. Matic, and V. Vapnik. Discovering informative patterns and data cleaning. In *Advances in Knowledge Discovery and Data Mining*, pages 181–203. 1996.

[7] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. *Computer Science Department, University of Toronto, Tech. Rep*, 2009.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. 2012.

[9] J. Larsen, L. Nonboe, M. Hintz-Madsen, and L. Hansen. Design of robust neural network classifiers. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 2, pages 1205–1208 vol.2, May 1998.

[10] N. D. Lawrence and B. Schölkopf. Estimating a kernel fisher discriminant in the presence of label noise. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 306–313, 2001.

[11] V. Mnih and G. Hinton. Learning to label aerial images from noisy data. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 567–574, 2012.

[12] N. Natarajan, I. Dhillon, P. Ravikumar, and A. Tewari. Learning with noisy labels. In *Advances in Neural Information Processing Systems 26*, pages 1196–1204. 2013.

[13] D. Nettleton, A. Orriols-Puig, and A. Fornells. A study of the effect of different types of noise on the precision of supervised learning techniques. *Artificial Intelligence Review*, 33(4):275–306, 2010.

[14] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[15] M. Pechenizkiy, A. Tsymbal, S. Puuronen, and O. Pechenizkiy. Class noise and supervised learning in medical domains: The effect of feature extraction. In *Computer-Based Medical Systems, 2006. CBMS 2006. 19th IEEE International Symposium on*, pages 708–713, 2006.

[16] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations (ICLR 2014)*, April 2014.

[17] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. DeepFace: Closing the Gap to Human-Level Performance in Face Verification. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[18] A. Torralba, R. Fergus, and W. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1958–1970, Nov 2008.