Machine Learning Hw3: Semi Supervised Learning b03901036 陳柏文

Collaborator: b03901037 鄭宇强, b03901137 張致剛

1. **Supervised learning**:

(Use only labeled data to train a model, record its performance, and describe your method)

(a.) Using augmented data: In order to increase the amount of training data, we augment them via keras.preprocessing.image.ImageDataGenerator class, so that my model would never see twice the exact same picture. This helps prevent overfitting and helps the model generalize better. I add the Before adding augmented data, I only get an accuracy of 0.3268 (on Kaggle public test set). However, after adding that, I get an accuracy of 0.4426 (on Kaggle public test set).

(b.) Using of Layer-sequential unit-variance (LSUV) initialization

Before using LSUV initialization, it's hard to train the model. (The accuracy of the model might be stuck at 0.1, which probably means the model has zero output, so that only class zero is predicted.) After using LSUV, model converges faster and has better performance, which I get an accuracy of 0.5459 on Kaggle public test set. The method consists of the two steps. First, pre-initialize weights of each convolution or inner-product layer with orthonormal matrices. Second, proceed from the first to the final layer, normalizing the variance of the output of each layer to be equal to one.

LSUV initialization is described in: Mishkin, D. and Matas, J.,(2015). All you need is a good init. ICLR 2016.

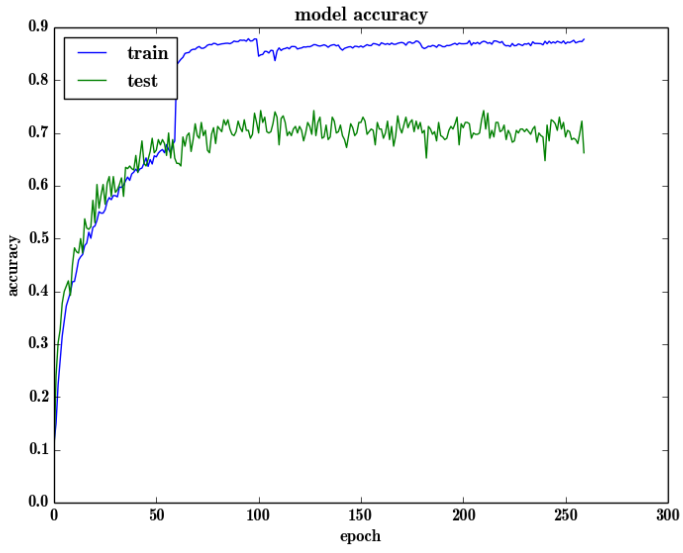
(c.) Using more layer

I construct a deeper model based on the structure of VGG which is hard to train, but with LSUV, training becomes much easier. I add dropout to partial convolution layer. For this deeper model, I get an accuracy of 0.6649, which outperforms the baseline.

2. **Semi-supervised learning (1)**:

(Use whole data to train a model, record its performance, and describe your method)

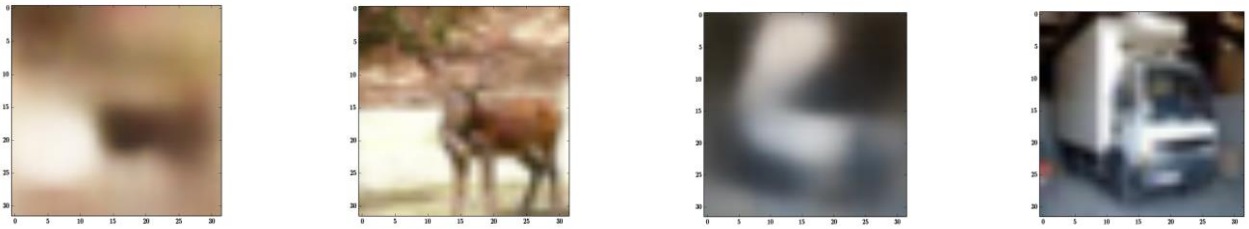
I add only partial unlabeled data which has more confident on its prediction, that means it has probability above 0.8 on the class it predicted. And I get an accuracy of 0.7348 on Kaggle private test set by this model. For the following picture, the blue curve indicates the accuracy on training set and the green curve indicates the accuracy on validation set, which is a spilt of size 400 from original 5000-sample training dataset. We can see that actually the improvement on the validation set is stable and grow slowly. On the other hand, the accuracy improves a lot quite on the training set upon the start of self-training. This is quite reasonable since that self-training reinforce the performance on those images having higher confidence in theirs predciction.



3. **Semi-supervised learning (2)**:

(Use another method, record its performance, and describe your method)

I use variation auto-encoder (VAE) to get the feature of each image and I use SVM (based on sklearn library) to predict the classes based on the codes got from encoder. The variation auto-encoder is constructed with convolution layer and deconvolution layer. And I found that with the dimension of the middle layer (which is the dimension of latent variables) becoming larger, the output of the VAE is closer to ground truth. The following pictures are output of VAE and ground truth for latent variables dimension equals to 120.



I get an accuracy of 0.4522 on Kaggle private test set by this model.

4. Compare and analyze your results

I get the best performance on method 1, but I think there is still space for improvement for method 2. And I also find a model called Contrastive Pessimistic Likelihood Estimation (CPLE) (based on - but not equivalent to - Loog, 2015), a `safe' framework applicable for all classifiers which can yield prediction probabilities (safe here means that the model trained on both labelled and unlabelled data should not be worse than models trained only on the labelled data). Maybe replacing this model for SVM can have better performance.

Current semi-supervised learning approaches require strong assumptions, and perform badly if those assumptions are violated (e.g. low density assumption, clustering assumption). In some cases, they can perform worse than a supervised classifier trained only on the labeled exampels. Furthermore, the vast majority require O(N^2) memory.

(Loog, 2015) has suggested an elegant framework (called Contrastive Pessimistic Likelihood Estimation / CPLE) which only uses assumptions intrinsic to the chosen classifier, and thus allows choosing likelihood-based classifiers which fit the domain / data distribution at hand, and can work even if some of the assumptions mentioned above are violated. The idea is to pessimistically assign soft labels to the unlabelled data, such that the improvement over the supervised version is minimal (i.e. assume the worst case for the unknown labels).