Final Course Project 2021 - ML Course

Deep Learning ensemble methods

Submitted by:
Roy Poliansky: 209220789
Ron Edri: 207912007

Task goal

The goal of the final exercise is to evaluate the performance of the Deep Learning ensemble methods that were published in the professional literature that weren't covered during the course.

Stage 1: Selecting the algorithm for evaluation

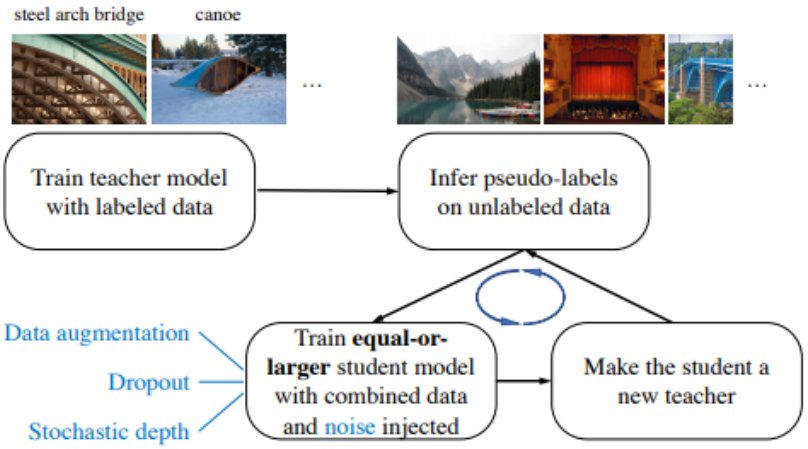
We choose the algorithm from the list, the algorithm shown in the article "*Self-training with noisy student improves image net classification*" wrote by Xie, Qizhe, et al. and presented in the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020.

Description of the algorithm

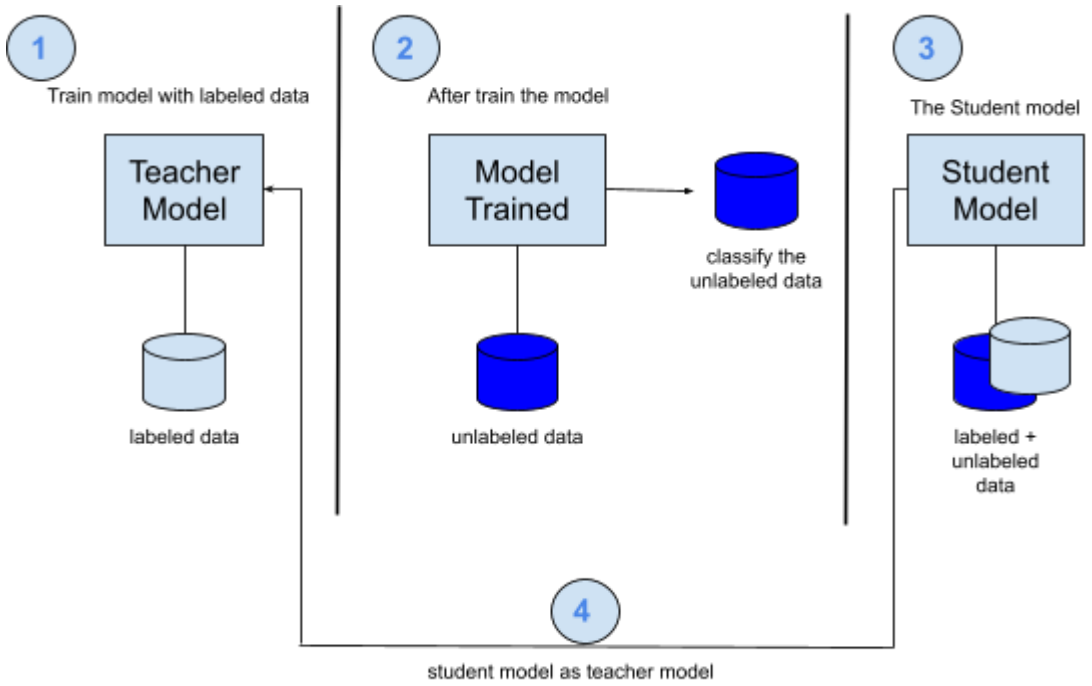
Noisy Student Training is a semi-supervised learning method which achieves 88.4% top-1 accuracy on ImageNet (SOTA) and surprising gains on robustness and adversarial benchmarks.

Noisy Student Training is based on the self-training framework and trained with 4 simple steps:

1. Train a classifier on labeled data (teacher).
2. Infer labels on a much larger unlabeled dataset.
3. Train a larger classifier on the combined set, adding noise (noisy student).
4. (Optional) Go to step 2, with student as teacher.



* An illustration of Noisy Student Training through four simple steps. We use two types of noise: model noise (Dropout, Stochastic Depth) and input noise (data augmentation, such as RandAugment).



One can view Noisy Student as a form of self-training, because the model generates pseudo-labels with which it retrains itself to improve performance. A surprising property of Noisy Student Training is that the trained models work extremely well on robustness test sets for which it was not optimized, including ImageNet-A, ImageNet-C, and ImageNet-P. We hypothesize that the noise added during training not only helps with the learning, but also makes the model more robust.

Noisy Student uses the idea of “knowledge distillation” (process of transferring knowledge from a large model to a smaller one) to knowledge expansion.

Advantages of the algorithm

As we said above, the main benefits of the model related to the previous works are the robustness and adversarial benchmarks. The use of model noise and input noise makes the model more robust to changes. In addition, this method can also deal with adversarial attack and beats previous SOTA on the ImageNet 2012 ILSVRC challenge problem.

Disadvantages of the algorithm

We train the model many times and therefore the runtime high, moreover a large amount of unlabeled data is necessary for better performance.

Conclusion

The authors proposed an innovative learning algorithm, and also studied the method toughly by well-designed experiments, quantizing the performance of each part.
This teacher-student iterative training method is similar to data augmenting techniques. It is very amazing that this method can have such good results, since the source of the knowledge requires only a small dataset (compared to its performance).
Noisy student self-training allows us to train a large model, so extraordinary performance is possible, the iterative noisy training process and progressive model size help generalize the student model. Like GAN based data augmentation, this kind of methods make self learning and lifelong learning possible.

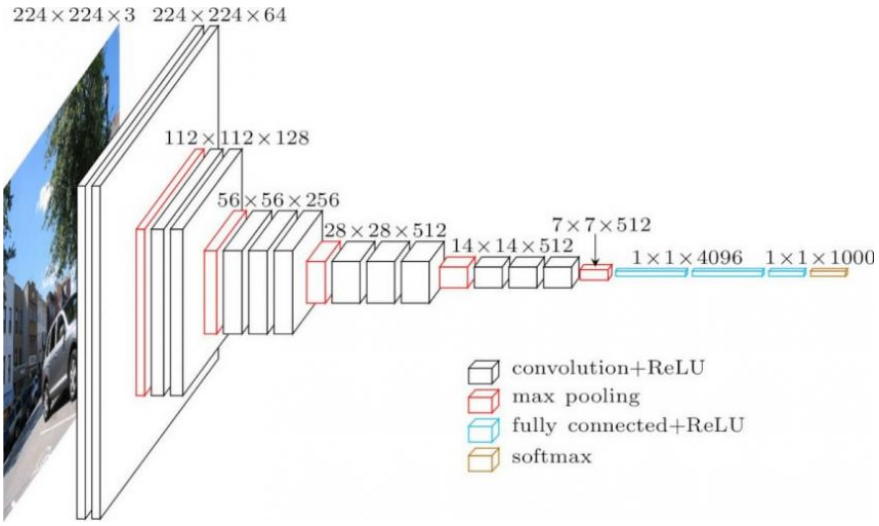
Stage 2: Suggesting an improvement:

We suggest splitting the unlabeled data so more iterations of the student-teacher cycle can occur with the same size of data. For example if we have 1000 instances of unlabeled data, instead of feeding all 1000 items into the teacher to label them and use them to train the student. We feed only 500 into the teacher to label, use the generated data to train the student, and then use the student to label the remaining 500 items and use them to train a new student.
The logic behind our improvement is to use steps when using the unlabeled data, and to not “burn it out”. This results in a final model that is trained on a more accurately labeled data.

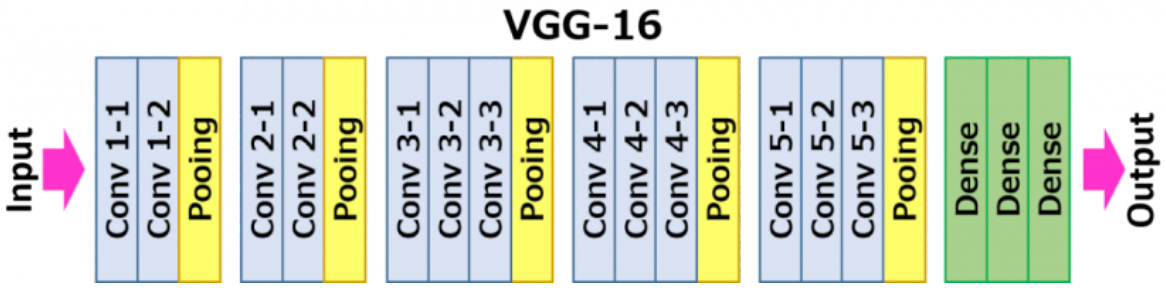
Stage 3: Selecting a well-known algorithm for comparison:

To compare our models, we selected the VGG16 model, The VGG-16 network is one of the most popular pre-trained models for image classification. Introduced in the famous ILSVRC 2014 Conference. Developed at the Visual Graphics Group at the University of Oxford, VGG-16 beat the then standard of AlexNet and was quickly adopted by researchers and the industry for their image Classification Tasks.

Here is the architecture of VGG-16:



And here is a more intuitive layout of the VGG-16 Model:



The following are the layers of the model:

- Convolutional Layers = 13
- Pooling Layers = 5
- Dense Layers = 3

Let us explore the layers in detail:

1. Input: Image of dimensions (224, 224, 3).
2. Convolution Layer Conv1:
    - Conv1-1: 64 filters
    - Conv1-2: 64 filters and Max Pooling
    - Image dimensions: (224, 224)
3. Convolution layer Conv2: Now, we increase the filters to 128
    - Input Image dimensions: (112,112)

    - Conv2-1: 128 filters
    - Conv2-2: 128 filters and Max Pooling
4. Convolution Layer Conv3: Again, double the filters to 256, and now add another convolution layer
    - Input Image dimensions: (56,56)
    - Conv3-1: 256 filters
    - Conv3-2: 256 filters
    - Conv3-3: 256 filters and Max Pooling
5. Convolution Layer Conv4: Similar to Conv3, but now with 512 filters
    - Input Image dimensions: (28, 28)
    - Conv4-1: 512 filters
    - Conv4-2: 512 filters
    - Conv4-3: 512 filters and Max Pooling
6. Convolution Layer Conv5: Same as Conv4
    - Input Image dimensions: (14, 14)
    - Conv5-1: 512 filters
    - Conv5-2: 512 filters
    - Conv5-3: 512 filters and Max Pooling
    - The output dimensions here are (7, 7). At this point, we flatten the output of this layer to generate a feature vector
7. Fully Connected/Dense FC1: 4096 nodes, generating a feature vector of size(1, 4096)
8. Fully ConnectedDense FC2: 4096 nodes generating a feature vector of size(1, 4096)
9. Fully Connected /Dense FC3: 4096 nodes, generating 1000 channels for 1000 classes. This is then passed on to a Softmax activation function
10. Output layer

Stage 4: Evaluating the algorithms:

As required, we used external 10-fold cross validation, in addition to an internal 3-fold cross validation for hyperparameter optimization.

Hyperparameter optimization

We optimized 4 different hyperparameters:

- Learning Rate - learning rate can be crucial, especially when dealing with both labeled and unlabeled data
- Batch Size - different batch sizes may be required for different datasets
- Pooling - max/average pooling layers
- Weights - initialized weights for the network

The metric for evaluating the optimal hyperparameters is the accuracy of the model.

The reasons behind choosing these hyper params:

1. learning rate: the learning rate is affecting the weights update of the teacher, similar to the learning rate in the baseline model. So it makes sense to try different values for the learning rate.
2. batch size: its good practice to try the model learning on different batch sizes to find a good balance between the computation time and the accuracy of the gradient update.
3. Pooling - max/average pooling layers - we try to run the base model with max/average to find the best pooling layers.
4. Weights - the initial of weights is very important, we try to initial with ‘imagenet’ weights or None.

Datasets

We collect 20 datasets of images with different shape and number of classes. Some datasets were divided to small parts.

We show in a table the metadata description of datasets:

Ind	Dataset Name	Shape	Number of Samples	Classes
1	Beans	(32, 32, 3)	1295	3
2	Casava	(64, 64, 3)	9430	5
3	Cifar100_1	(32, 32, 3)	12000	20
4	Cifar100_2	(32, 32, 3)	12000	20
5	Cifar100_3	(32, 32, 3)	12000	20
6	Cifar100_4	(32, 32, 3)	12000	20
7	Cifar100_5	(32, 32, 3)	12000	20
8	Cmater	(32, 32, 3)	6000	10
9	comp_1	(32, 32, 3)	1500	9
10	comp_2	(32, 32, 3)	1500	9
11	comp_3	(32, 32, 3)	1500	9
12	Oxford_1	(64, 64, 3)	3493	51
13	Oxford_2	(64, 64, 3)	4696	51
14	Rps	(100, 100, 3)	2892	3
15	Coloret	(64, 64, 3)	5000	8
16	shvn_1	(32,32,3)	51765	4
17	shvn_2	(32,32,3)	34565	4
18	stl_1	(96,96,3)	5200	4
19	stl_2	(96,96,3)	5200	4
20	stl_3	(96,96,3)	5200	4

* All results are documented in the attached CSV file.

Stage 5: Statistical significance testing:

We ran a Friedman test on the results of the experiments over the 20 datasets and the three algorithms. We choose AUC as the metric for the test.

What is *friedman test* ?

The Friedman test is the nonparametric version of the repeated measure's analysis of variance test, or repeated measure's ANOVA. The test can be thought of as a generalization of the Kruskal-Wallis H Test to more than two samples.

Results:

Friedman test p-value = 0.0367, therefore for alpha=0.05 we can reject the null Hypothesis. So we reject the null hypothesis, and we ran post-hoc tests.
We chose the Nemenyi Post-Hoc test to compare the performance of the algorithms two algorithms at a time. The results of the Nemenyi test are provided owing table

	baseModel	ArticleModel	ImproveArticleModel
baseModel	1.000000	0.33901	0.0334141
ArticleModel	0.339010	1.00000	0.339010
ImproveArticleModel	0.0334141	0.33901	1.000000

We can see from the results that for alpha=0.05 our improved algorithm outperforms the algorithm from the article significantly.

Stage 6: Conclusions:

In this project, we ran experiment on three algorithms in order to show the improvements we compare them on 20 datasets, on the metrics of Accuracy, TPR, FPR, Precision, AUC, Area under the Precision-Recall, training time, Inference time for 1000 instances.

The baseline algorithm (VGG16):

VGG16 is a classic deep model for image classification, we use it as base model. We found that the training run time, and the inference time of this model are fastest, but the accuracy is less than the others models.

Pseudocode - model structure

```
base_model = VGG16(include_top=False, weights=self.weight, pooling=self.pooling, input_shape=self.input_shape)
model = Sequential()
model.add(base_model)
model.add(Dense(self.classes, activation='softmax'))
return model
```

<u>The algorithm of the article:</u>

The accuracy of this model on the most datasets are better than the base model.

Pseudocode - teacher/student model structure

```
base_model = K.applications.EfficientNetB3(include_top=False, weights='imagenet', drop_connect_rate=0.4)
resize = K.Sequential([
  K.layers.experimental.preprocessing.Resizing(self.input_shape[0], self.input_shape[1])
])
model= K.Sequential()
model.add(resize)
model.add(base_model)
model.add(K.layers.Flatten())
model.add(K.layers.Dense(512, activation=('relu')))
model.add(K.layers.Dropout(0.2))
model.add(K.layers.Dense(256, activation=('relu')))
model.add(K.layers.Dropout(0.2))
model.add(K.layers.Dense(self.classes, activation=('softmax')))
return model
```

<u>The algorithm of the improvement article:</u>

This is the model that present the best results.

Pseudocode - teacher/student model structure (same as article model)

```
base_model = K.applications.EfficientNetB3(include_top=False, weights='imagenet', drop_connect_rate=0.4)
resize = K.Sequential([
  K.layers.experimental.preprocessing.Resizing(self.input_shape[0], self.input_shape[1])
])
model= K.Sequential()
model.add(resize)
model.add(base_model)
model.add(K.layers.Flatten())
model.add(K.layers.Dense(512, activation=('relu')))
model.add(K.layers.Dropout(0.2))
model.add(K.layers.Dense(256, activation=('relu')))
model.add(K.layers.Dropout(0.2))
model.add(K.layers.Dense(self.classes, activation=('softmax')))
return model
```

* The results of all models include in file name's (results.xlsx) in GitHub