Ensemble strategies II – Output aggregation

Based on: "Performance on Ensembles with DCNN for Image Classification" [1]

https://arxiv.org/abs/1704.01664

This section explores the second decision to make when building an ensemble.

That is, we will answer the question:

How the output from the individual learners should be aggregated?

Table of Contents

Notes

- All the individual learners of the ensembles are trained separately.
- All these approaches are way older than ensemble aware training, which is 'recent'.
- Ensemble aware approaches are discussed in the next section

The paper faces this issue by searching the best ensemble strategy. However, in the main section we saw how an ensemble strategy should not be exclusive of the aggregation method of the output. For consistency with the paper we will keep calling ensemble strategy to the aggregation method, but keep in mind we are just in one of the decisions that defines a strategy.

Statements

The performance of a single learner depends on how effective its searching strategy is in approximating the optimal predictor defined by the true data generating distribution

The performance of various learners will depend, in theory, on the model assumptions and the true data-generating distributions. In practice, it will depend on the sample size, dimensionality, and the bias-variance trade-off of the model

Motivation

The motivation of the paper on which this work is based is because there was no previous research on ensembles of convolutional neural networks.

What have previous work on ensembles done so far? A naïve average of the result of each individual learner to enhance performance (recap ambiguity decomposition)**.** This averaging is **not data adaptive** and thus is vulnerable to a bad \`library\` of learners – understanding by library as a set of neural networks. Indeed, it works well for networks with similar structure and comparable performance, BUT it is sensible to the presence of excessively biased base learners.

Objective

The objective of the paper is to get insight of the performance of different ensembling strategies.

The objective of this work is to demystify what each of the strategies is doing and elaborate a nice visualization and concise conclusion on each of them.

The continuation of this work covers the paper [2] where more interesting strategies are taken, since they will consider understanding ensemble models as a first-class problem. However, during the strategies covered here as mentioned, ensemble will refer to the problem of combining results of already trained models.

1. Unweighted (naïve) average

As mentioned before, this is the most common ensemble approach for neural networks.

It simply takes the (unweighted) average of the output for all the base learners and reports it as the predicted output score. The average can be performed in the scores of each learner or on the probabilities of each member, yielding to two type of averaging, scores averaging and soft voting, respectively.

> ***Deep neural networks have a really low bias*** *due to their high capacity. However, that* ***also*** *induce a* ***great variance*** *in these models. The motivation behind averaging comes precisely from reducing this variance. The more the individual learners are uncorrelated, the more the variance is reduced.*

Scores Averaging

We know from [2], and will be detailed later, that a better result is achieved if averaging on the scores. This is initially counter-intuitive since we are trying to optimize the best output probabilities. Figure 1 and Eq *1* shows the case for averaging on the scores, while Figure 2 and *Eq 2* shows the case for averaging in the scores.

$$p_{i,j} = softmax\left(\frac{(\vec{s}_i)[j]}{\sum_{c=1}^{C}(\vec{s}_i)[c]}\right)$$

Eq 1

Also, note that **scores averaging assumes that the models are calibrated**. This means that the scale in which the network output the scores are equivalent among them, and no single networks dominates over the rest.

<u>Note1</u>: in the figures, each network outputs a vector with the length equal to the total number of classes. The horizontal vectors that are represented before each averaging step correspond to the scores/probabilities for each class by all the networks of the ensemble.

<u>Note2</u>: the shaded grey blocks just represent the first input $\vec{x}_1$ to the model, which is the one being tracked for a clearer visualization. The rest of the inputs (1D arrays this case (X)) will behave the same.

<u>Note3</u>: the vector y_{true} consists on a one hot vector representing the correct class to predict.

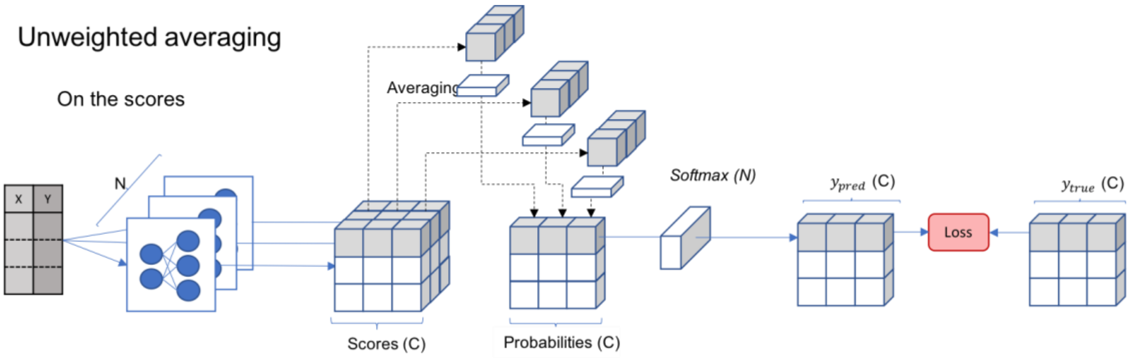


Figure 1. Unweighted averaging on the scores

Soft Voting

As mentioned above, it is more reasonable to average after the softmax transformation, since they will all be in the same scale, between 0 and 1, and we don't need to assume the networks are calibrated.

$$p_{i,j} = softmax(\vec{s}_i)[j] = \frac{(\vec{s}_i)[j]}{\sum_{c=1}^{C} exp\big((\vec{s}_i)[c]\big)} \qquad \text{Eq 2}$$

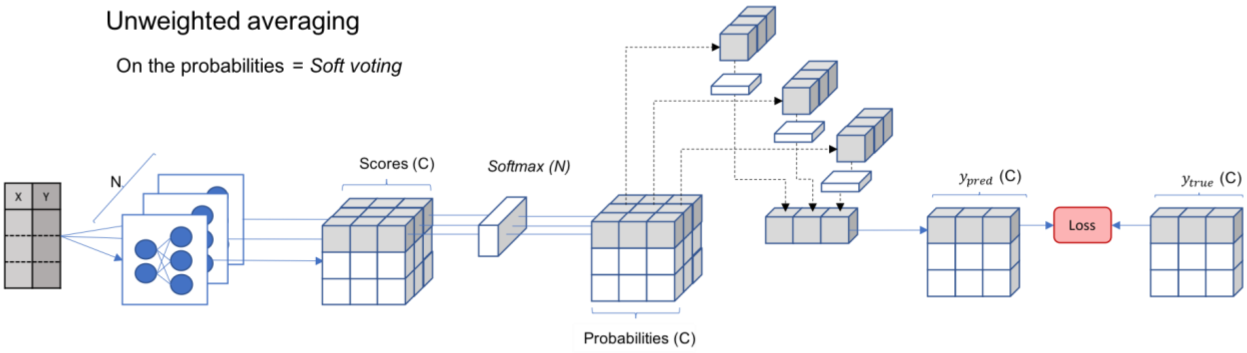


Figure 2. Unweighted averaging on the probabilities

The immediate **drawbacks** of naïve averaging come naturally from the weaknesses of the pure concept of averaging data. When the library contains heterogeneous models, it is vulnerable to weak learners, and sensitive to over-confident candidates. To address this particular issue, we find the next strategy.

2. Majority voting

Majority voting, instead of averaging, counts the predicted labels and output the label with most votes.

Majority voting is therefore less sensitive to the output from a single model. However, it would still be dominated if the library contains multiple similar and dependent base learners. Another clear weakness is the loss of information by keeping only the most voted one.

***Shallow neural networks**, with less capacity, **gives more disperse predictions** than deeper architectures. Then, in the case of majority voting, a library of shallow networks is preferred.*

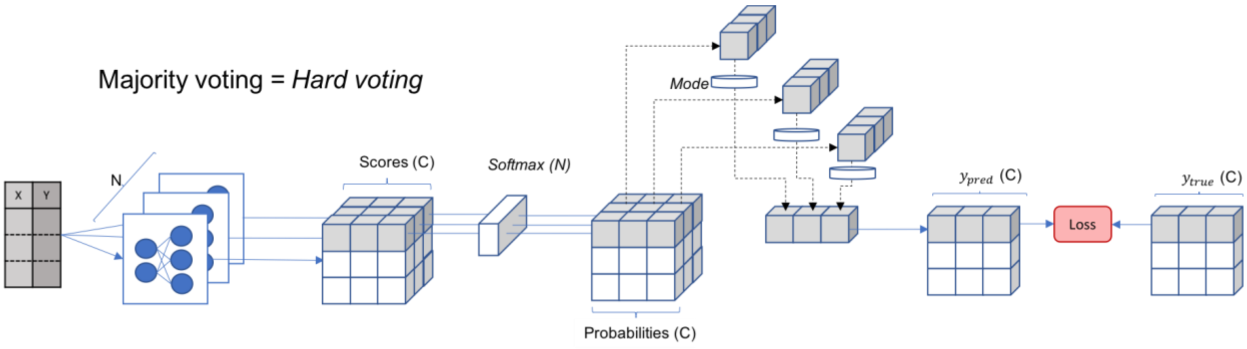


Figure 3. Majority voting

3. Weighted averaging – Stacked Generalization

Stacking models works by deducing the biases of the generalizers with respect to a provided learning set. The idea of stacking is to 'stack' the predictions by a linear combination of weights:

$$f_{stack}(x) = \sum_{i=1}^{M} a_i \cdot f_i(x) \qquad \textit{Eq 3}$$

The values of the weights are precisely what has to be learned by the meta-learner to combine the outputs in the most efficient way.

4. Super Learner: cross-validation based stacking

Super Learner (SL) [3] is an extension of stacking. It is a cross-validation (CV) based ensemble methods, which minimizes cross-validated risk for the combination of the individual learners.

Intuitively, SL combines the base learners by CV. To better visualize this (more complex) procedure, let's first exposure how the training of each of the model is perform, then let's see one forward pass for 1 particular model, and finally understand the forward pass over the entire ensemble.

To be consistent with [1], let's say we have m models, and we split the dataset into V folds. The training step, represented in Figure 4, will divide the dataset into the V folds and train each learner of the library with each of the constructed datasets.

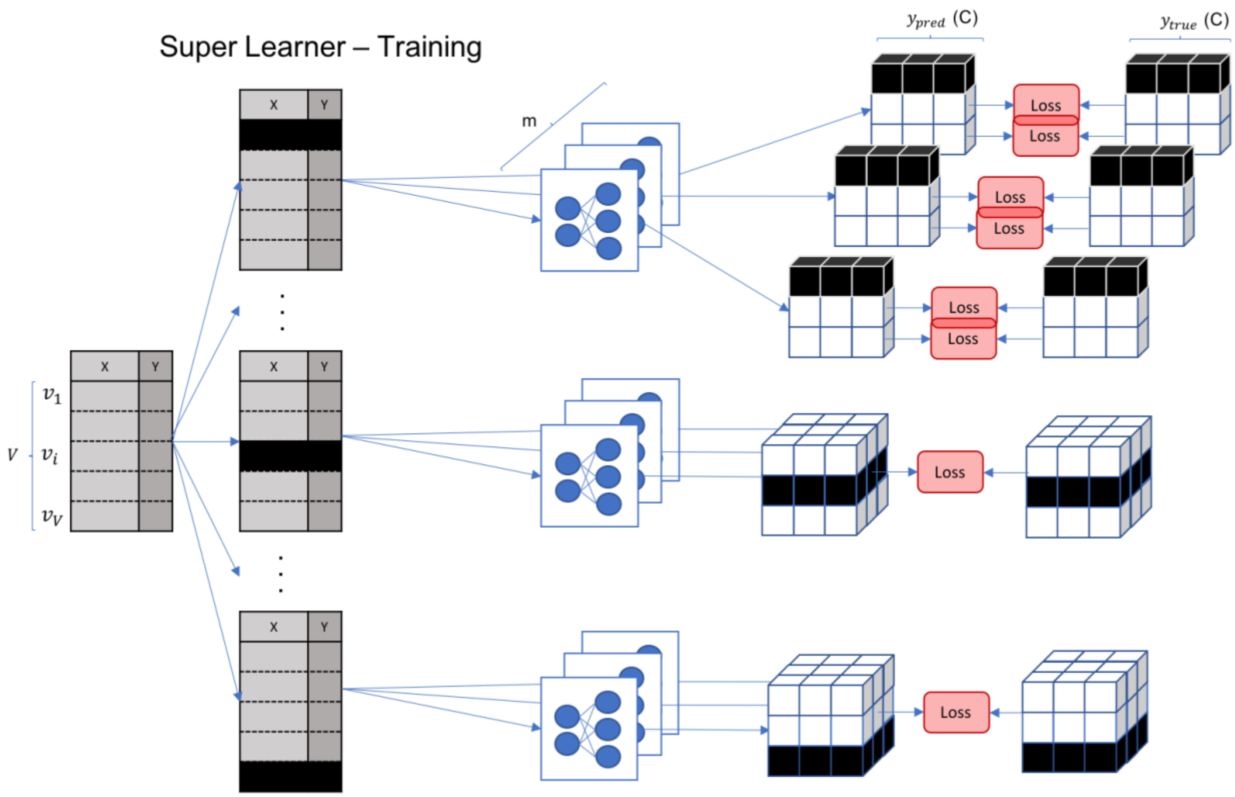
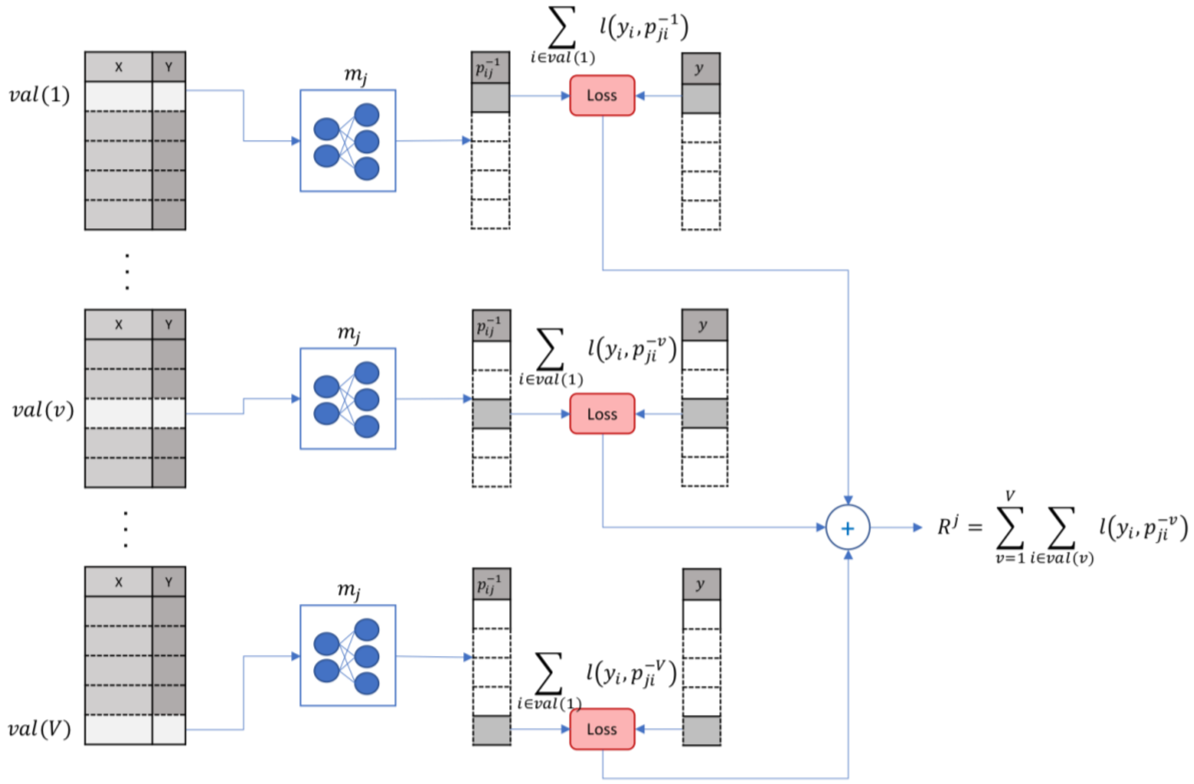


Figure 4. Training on every individual for the Super Learner ensemble method

That is, each learner will have a copy of itself, trained on the entire dataset except the validation fold v_i.

Next, let's take a look at Figure 5, where we represent one of the m learners, let's say network j, and perform a single forward pass to define a loss for a particular network.



$\sum_{i \in val(1)} l(y_i, p_{ji}^{-1})$
X
Y
$val(1)$
m_j
p_{ij}^{-1}
y
Loss
X
Y
$val(v)$
m_j
p_{ij}^{-1}
$\sum_{i \in val(1)} l(y_i, p_{ji}^{-v})$
y
Loss
+
$R^j = \sum_{v=1}^{V} \sum_{i \in val(v)} l(y_i, p_{ji}^{-v})$
X
Y
m_j
p_{ij}^{-1}
y
$\sum_{i \in val(1)} l(y_i, p_{ji}^{-V})$
$val(V)$
Loss

Figure 5. Forward pass on a single learner of the Super Learner ensemble

This time, for the learner that was trained using all the data set except the samples within the fold v, those particular samples are the input to get the probabilities of the learner. Note how in the sum, the $i \in val(v)$ refers to this. We can now compute a lost for that output fold compared with the correspondent fold of true values, to get the loss for that learner trained in all the dataset except that fold.

Now, we can extend to calculate the loss for the entire learner j. To do it, we again need to sum the loss calculated for each of the folds, resulting on R^j, the loss for learner j.

Intuitively, the last expansion will be towards calculating the loss for the entire ensemble. To it, we need to modify the value corresponding to the loss of each fold, since now it will be an average between the rest of the learners of the ensemble. Furthermore, we have not walk this far to do a simple average, we want to be the expansion of the stack method mentioned at the beginning of the explanation. Therefore, we will perform a **parametrized** average.

In other words, we will substitute p_{ij}^{-v} by $\sum_{j=1}^{m} a_j \cdot p_{ij}^{-v}$, yielding to the loss for the ensemble prediction:

$$R(\vec{a}) = \sum_{v=1}^{V} \sum_{i \in val(v)} l\left(y_i, \sum_{j=1}^{m} a_j \cdot p_{ij}^{-v}\right) \qquad \textit{Eq 4}$$

Ideally, we want out meta-learner to find the optimal vector of parameters that minimizes the ensemble loss. Being the optimal vector:

$$\vec{a} = arg \min_{\vec{a}} \ R(\vec{a}) \qquad Eq\ 5$$

Let's go one more step beyond and apply what we have just seen into the C-class classification problem to help as visualize how the vector $\vec{a}$ is incorporated into the big picture, along with the rest of the learners of the ensemble, see Figure 6.

Ensembling on scores levels, as we mentioned before is more optimal, will yield to the following expression, that is also depicted in Figure 6.

Being $p_i^z(\vec{a})$ the ensemble prediction for the i-th simple for the c-th class with weight vector $\vec{a}$:

$$p_i^z(\vec{a}) = -\log\left(\frac{exp\left(\sum_{j=1}^{m} a_j \cdot s_i[j,z]\right)}{\sum_{c=1}^{C} exp\left(\sum_{j=1}^{m} a_j \cdot s_i[j,c]\right)}\right) \qquad Eq\ 6$$

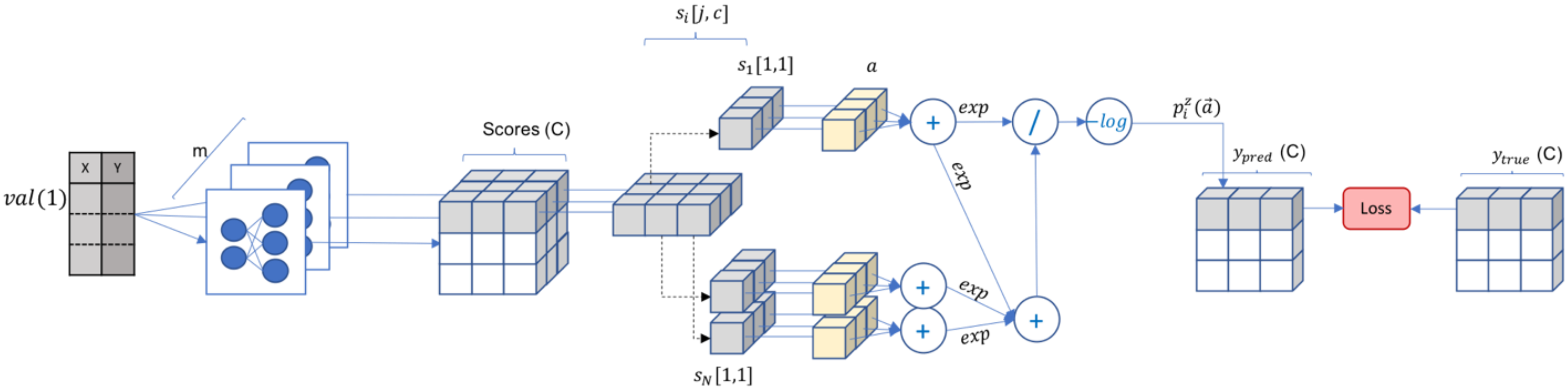


Figure 6. Super Learner Prediction

For simplicity, we have shaded the fold being predicted and remove the rest after the scores volume. Notice how the scores $s_i[j,c]$ for a given input sample consist on a m by C matrix, standing for the score of the j-th model to the class c.

4.1. Comments on Super Learner approach

Impractical on Deep Learning Models

It is very intuitive to see that the number of computations to perform the SL approach has increased considerably. During the training phase, each individual learner has to be trained V times!

This results completely **impractical when the dataset is big, and/or the library is large**.

Usually the validation set is set aside instead of performing a cross-validation, to assess and tune the models for deep learning. This way, we could optimize a single-split cross-validation loss instead to get the ensemble weights.

SL as 1x1 Convolution

There are many perspectives of neural networks considered to be ensemble learning. Dropout, for instance, could be seen as training multiple base learners and ensembling them during prediction. Another example is ResNet architecture, that could be understood as ensemble of shallow networks.

SL computes an honest ensemble weight based on the validation set.

A validation set is commonly used in Deep Learning for tuning some of the hyperparameters. For most image classification data sets, the validation set is very large in order to make this validation stable. Authors of [1] conjecture that the potential of the validation information has not been fully exploited yet.

SL could be considered as a neural network with a 1x1 convolution over the validation set, with the scores of the base learners as input.

Therefore, it learns the $1x1xm$ kernel by back-propagation in an end-to-end model, represented in Figure 7:

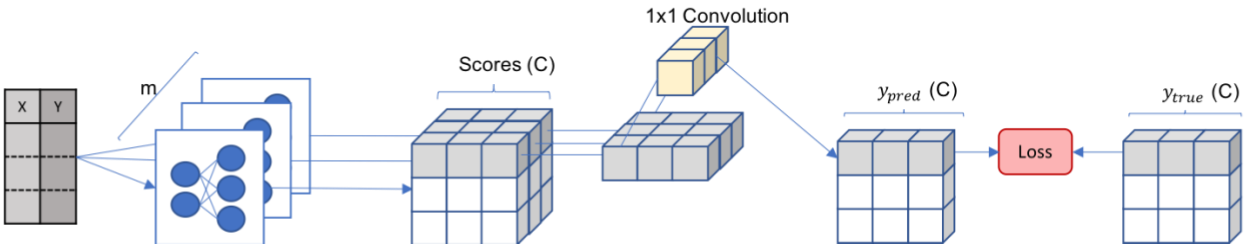


Figure 7. Super Learner as a 1x1 Convolution

5. Discussion

Unweighted averaging provides good results when the performance of the individual learners is comparable, outperforming majority voting in almost all the experiments run on [1]. However, it is proved to be sensitive to over-confident models.

The Super Leaner addressed this issue by simply optimizing a weight on the validation set in a data-adaptive manner.

Super Learner has been initially presented as a cross-validation based ensemble method. However, since CNN are computationally intensive and that validation sets are typically large in computer vision tasks, the validation set is set aside and is used by the neural networks for computing the weights of Super Learner in the single-split cross-validation method proposed immediately after.

Works Cited

[1] C. Ju and M. van der Laan, "The Relative Performance of Ensemble Methods with Deep Convolutional Neural Networks for Image Classification," 2017.

[2] S. Lee, S. Purushwalkam, M. Cogswell, D. Crandall and D. Batra, "Why M Heads are Better than One: Training a Diverse Ensemble of Deep Networks," 2015.

[3] M. van der Laan and E. Polley, "Super Learner," *Statistical Applications in Genetics and Molecular Biology,* vol. 6, no. 1, 2007.