Ensemble methods for image classification

Yaqi Zhang[*]
Department of Mechanical Engineering
UW-Madison
Madison, WI 53705
zhang623@wisc.edu

Chengning Zhang
Department of Statistics
UW-Madion
Madison, WI 53705
czhang449@wisc.edu

Zijian Ni
Department of Statistics
UW-Madion
Madison, WI 53705
zni25@wisc.edu

Youran Qi
Department of Statistics
UW-Madion
Madison, WI 53705
yqi@stat.wisc.edu

Na Li
Department of Animal Sciences
UW-Madion
Madison, WI 53705
nli37@wisc.edu

Abstract

As one of the core problems in computer vision (CV), image classification has a large variety of practical applications. (Deep) Convolution neural network have made a lot of achievements in this field. Ensemble methods are learning algorightms that construct a set of classifiers to get a better performance. In the project, we studied the performance of different ensemble methods including bagging (with majority voting and weighted average), AdaBoost, SAMME, stacking, super learner and snapshot ensemble of both normal CNN and residual neural network. For CNN, we not only compared the relative performance of different ensemble methods, but also compared the performance of ensembles with different number of base learners. For residual neural network, we studied the ensembles of different residule neural networks. Finally, we stuied the ensemble of networks with different structures through combining the models of residual neural network with different depths.

1 Introduction

1.1 Motivation

Our motivation for this project are listed as follows

- Gain some experience on practical machine learning problems;
- Have a better understanding of how and why ensemble methods work;
- Compare the performance of different ensemble methods/techniques;

1.2 Related work

Ensemble methods in neutral networks have been widely used in solving modern machine learning problems [1, 2]. The ensemble learning methods make predictions by combining training results from several baseline models through some rules. Compared with single training method, the ensemble method demonstrated better prediction performance in practice and gains increased popularity in various fields. The relative performances of different learners depend on the model assumptions and the true data generating distributions. However, real sample data used for model training are usually

[*] Corresponding author: Yaqi Zhang <zhang623@wisc.edu>

limited by its sample size, dimensionality, representation of the population and the bias-variance trade-off of the model [3]. Thus, it is impossible to know which learner performs best given the finite data set and prediction problems [4]. The idea of ensemble learning, which combines different predictors demonstrated great advantage in prediction accuracy and are well studied and applied in many fields recently.

The most widely used ensemble techniques are bagging [5] and boosting [6]. Bagging reduces the variances of the strong learners by manipulating the training examples through bootstrap aggregation. Boosting, on the other hand, boost the capacity of the weak learners. Adaboost, developed by Freund and Schapire [7] manipulates the training set by minimizing the weighted errors. Boosting has been reported to be applied in convolutional neural networks and create strong ensembles [8]. SAMME - Stagewise Additive Modeling proposed by Zhu et al. [9] lifted the restriction of Adaboost to binary classification by introducing a multi-class exponential loss function into the algorithm. Wolpert [10] and Breiman [11] reported a linear combination strategy called stacking to train models. Super Learner developed by van der Laan et al. [12] is a further extension of stacking generalization with a cross-validation based ensemble framework. It achieves the optimal combination of a collection of prediction algorithms by minimizing the cross-validated risk. Super Learner has gained great popularity in a wide range of topics and achieved excellent or best performance, such as image classification [3], spatial prediction [13], online ensemble learning [14], clinic trails [12] and survival analysis [15].

In recent years, deep learning neutral networks shows great success in improving the predictive performance by modeling average of multiple stochastically trained networks [3]. The averaging idea has won almost all machine learning challenges across different areas such as image classification [16], imageNet detection [17] and localization [18]. Deep neural networks demonstrate high variance and low bias due to their high capacity and flexibility [3]. Even though ensemble method has been well applied in solving deep learning network problems, most published literatures only focus on the design of network structures simply by averaging ensembles [3]. Therefore, the behavior of ensemble learning with deep networks is still not well understood. Our work will investigate and compare different ensemble methods using (deep) convolution (CNN) or residual neural network (RNN) with different structures and depths to find the best one suitable for image classification. In addition, prior studies only focused on improving the generalization performance, few addressed the cost of training or test time. Several recent studies tried to transfer the knowledge of ensembles into one single model to reduce the test time cost [19, 20].

1.3 Outline

The report is organized as follows. *Section 2* introduces the general CNN model and residual neural network which are picked in our project be the base learners. Numerical experiments are described in *section 3*. In it, *section 3.2* introduces CNN model used for classification and metal model used in stacking and super learner. Ensemble results are listed and compared in *section 3.3*. *Section 4* is reserved for discussion and conclusion.

2 Background

2.1 Convolution neural network and residual neural network

2.1.1 Convolution neural network

When the inputs to the neural network are large images, the traditional fully-connected neural networks are no longer appropriate because there will be too many parameters [21]. For example, if the input is a 28×28 image with only one channel and the output is a 8×8 image with only one channel, then we need $(28 \times 28) \times (8 \times 8) = 50176$ parameters, if we simply regard the input as a 784 dimensional vector and the output as a 64 dimensional vector and fully connect them. In order to handle the large images, we need to use the convolutional neural networks, whose primary goal is to reduce the number of parameters.

A convolutional neural network may consist of three kinds of layers - the convolutional layer, the pooling layer and the fully-connected layer (see Fig. 1).

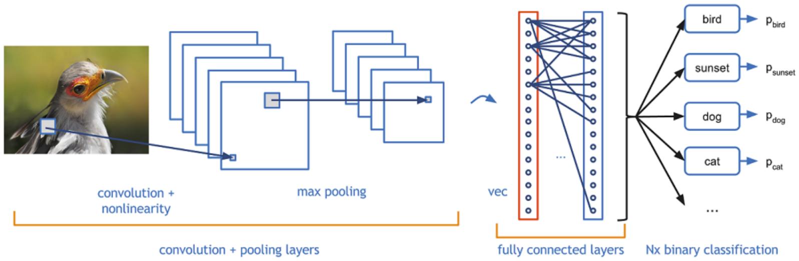

Figure 1: General structure of CNN [22]

The key component of a convolutional layer is a filter that can be slided on the image. For an $H \times W$ image with D channels, an $u \times v$ filter only connects with the $u \times v \times D$ pixels, where u, v are much less than H, W respectively. In this case, there are only $u \times v \times D + 1$ parameters, including a bias parameter. The filter fully connects with all the D channels but within a single channel, it only connects with $u \times v$ pixels, which is called "local connectivity". The filter has different parameters for different channels but when we slide the filter on the image with given strides s and r, the parameters remain the same, which is called "parameter sharing". Sliding a filter on the image yields a single feature map. In practice, we often use d filters to get d feature maps, where $d > 1$. We then can regard the feature maps as an $(H-u)/s+1$ by $(W-v)/r+1$ image with d channels and stack another convolutional layer or a pooling layer on it. Mathematically, we have

$$a_{i,j,k} = \phi\Big(\sum_{i'=1}^{u}\sum_{j'=1}^{v}\sum_{k'=1}^{D} A_{i'+(i-1)s,j'+(j-1)r,k'} w_{i',j',k',k} + b_k\Big), \tag{1}$$

where $i = 1, \dots, (H-u)/s+1$, $j = 1, \dots, (W-v)/r+1$, $k = 1, \dots, d$, $a_{i,j,k}$ is the value at location (i,j) in the kth feature map, $A_{\tilde{i},\tilde{j},\tilde{k}}$ is the value at location $(\tilde{i},\tilde{j})$ in the $\tilde{k}$th channel, $w_{i',j',k',k}$ and b_k are the parameters we need to learn and ϕ is usually the ReLU function.

The goal of the pooling layer is to reduce the size of the feature maps. The pooling layer is actually destructive so some scholars recommend not to use them. Mathematically, an $U \times V$ max-pooling layer with strides S and R can be expressed as follows:

$$B_{i,j,k} = \max_{i'=1,\dots,U,j'=1,\dots,V}\{a_{i'+(i-1)S,j'+(j-1)R,k}\}, \tag{2}$$

where $B_{i,j,k}$ is the value at location (i,j) in the kth pooled feature map.

Finally, we will also stack several fully-connected layers on the top, where we simply stretch the last set of feature maps to make them form a vector.

2.1.2 Residual neural network

Two problems in deep learning are

1. Gradient vanishing and exploding
2. Degradation.

To large extent, the first problem has been solved by the batch normalization [23] and the second problem has been solved by the residual structure [24].

Degradation means that when we have deeper neural networks, the accuracy first gets saturated and then degrades rapidly [24]. If we only consider the ability to represent a complex relationship, a deeper neural network is always better than a shallower neural network. A shallower network can always be viewed as a special version of the deeper network because when we let the extra layers in the deeper network be identity mappings, the deeper network reduces to the shallower one. However, even though the deeper networks are more "general", they are actually much harder to train, which explains why adding layers decreases the accuracy instead of increasing it, i.e., why degradation happens.

The degradation phenomenon indicates that it is actually much harder than we thought for a deep neural network with multiple nonlinear layers to approximate the identity mappings [24]. The idea of the residual structure is that we assist a deep neural network to do this - instead of asking it to approximate an identity mapping, we ask it to approximate zero because we hypothesize that the latter is easier. Even if the optimal mapping is only close to an identity mapping, but not exactly the identity mapping, it should still be easier to approximate the perturbations with reference to an identity mapping, than to approximate the optimal mapping from scratch [24]. Mathematically, the residual structure can be expressed as follows:

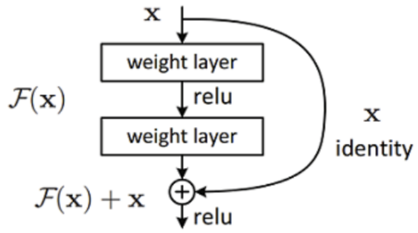
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2: Residule block [22]

$$a^{(l+1)} = a^{(l)} + \mathcal{F}_\theta(a^{(l)}), \tag{3}$$

where $a^{(l)}$ is the input to the residual structure, $a^{(l+1)}$ is the output from the residual structure, and θ represents the parameters that need to be learned. $\mathcal{F}_\theta$ can consist of multiple layers but no matter what its structure is, it is only about the residual $a^{(l+1)} - a^{(l)}$, i.e., the model only needs to learn the residual.

2.2 Ensemble methods

2.2.1 Bagging

One useful approach for constructing ensembles is to manipulate the training examples. The most straightforward way of manipulating the training set is called Bagging. On each run, Bagging presents the learning algorithm with a training set that consists of a sample of m training examples drawn randomly with replacement from the original training set of m items. Such a training set is called a bootstrap replicate of the original training set. and the technique is called bootstrap aggregation from which the term Bagging is derived [5]. Each bootstrap replicate contains, on the average, 63.2% of the original training set, with several training examples appearing multiple times.

In the backpropagation algorithm for training neural networks, the initial weights of the network are set randomly. If the algorithm is applied to the same training examples but with different initial weights, the resulting classifier can be quite different [25]. In our experiments, we also tried bagging without random picking training set which means every base learner is trained with the same training data set. In this approach, model diversity is injected by the randomness in the training procedure.

After we obtain "diverse" and "accurate" base classifiers, one intuitive question would be how to combine all base classifiers to increase performance. In our project two methods are used: **weighted averaging** and **majority voting**. However, before introduce the two methods used in our project, It is necessary to introduce the **unweighted averaging** first, which is most common ensemble method for neural network.

Unweighted averaging is the most common ensemble approach for neural networks. It takes unweighted average of the output score/probability for all the base learners, and reports it as the predicted score/probability.

Due to the high capacity of deep neural networks, simple unweighted averaging improves the performance substantively. Taking the average of multiple networks reduces the variance, as deep ANNs have high variance and low bias. If the models are uncorrelated enough, the variance of models could be dramatically reduced by averaging. This idea inspires Random Forest [26], which builds less correlated trees by bootstrapping observations and sampling features. We could average either directly the score output, or the predicted probability after softmax transformation:

$$p_{ij} = softmax(\vec{s_i})[j] = \frac{\vec{s_i}[j]}{\sum_{k=1}^{K} \exp\left(s_i[k]\right)} \tag{4}$$

where score vector $\vec{s_i}$ is the output from the last layer of the neural network for $i-th$ unit, $\vec{s_i}[k]$ is the score corresponding to $k-th$ class/label, and p_{ij} is the predicted probability for unit i in class j. It is more reasonable to average after the softmax transformation, as the scores might have varying scales of magnitude across the base learners, as the score output from different network might be in different magnitude. Indeed, adding a constant to scores for all the classes leaves predicted probability unchanged. In our project, we use weighted averaging, which is an improved version of unweighted averaging.

Weighted average The only difference between weighted averaging and unweighted averaging is that we assign weights to the base classifier according to the prediction accuracy in the training set when making final prediction. The intuition is that base classifiers which have higher accuracy in training set should have more weights in future prediction.

Majority voting Majority voting is similar to unweighted averaging. But instead of averaging over the output probability, it counts the votes of all the predicted labels from the base learners, and makes a final prediction using label with most votes. Or equivalently, it takes an unweighted average using the label from base learners and chooses the label with the largest value.

2.2.2 Boosting-AdaBoost and SAMME

The second method for manipulating the training set is illustrated by the AdaBoost algorithm, developed by Freund and Schapire [7]. Like Bagging, AdaBoost manipulates the training examples to generate multiple hypotheses. AdaBoost maintains a set of weights over the training examples. In each iteration l, the learning algorithm is invoked to minimize the weighted error on the training set, and it returns an hypothesis h_l. The weighted error of h_l is computed and applied to update the weights on the training examples. The effect of the change in weights is to place more weight on training examples that were misclassified by h_l and less weight on examples that were correctly classified. In subsequent iterations, therefore, AdaBoost constructs progressively more difficult learning problems. AdaBoost algorithm proceeds as follows:

```
1 Initialize the observation weights $w_i = 1/n, i = 1, 2, ..., n$
2 For $m = 1$ to $M$:
    (a) Fit a classifier $T^{(m)}(x)$ to the training data using weights $w_i$.
    (b) Compute
        $err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}(c_i \neq T^{(m)}(x)) / \sum_{i=1}^{n} w_i$
    (c) Compute
        $\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}}.$     (5)
    (d) Set
        $w_i \longleftarrow w_i \cdot \exp(\alpha^{(m)} \cdot \mathbb{I}(c_i \neq T^{(m)}(x))), i = 1, 2, \ldots, m$
    (e) Re-normalize $w_i$.
3 Output $C(x) = \arg\max \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(x) = k)$.
```

The original adaboost algorithm is restricted to binary classification. Ji Zhu et al. [9] proposed an algorithm called SAMME - Stagewise Additive Modeling using a Multi-class Exponential loss function. SAMME is very similar to AdaBoost with a major difference in 5. The equation of α^m is changed into

$$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + log(K-1), \tag{6}$$

where K is the number of classes. Now in order for α^m to be positive, we only need $(1 - err^m) > 1/K$. As a consequence, the new algorithm puts more weight on the misclassified data points in

(2d) than AdaBoost, and the new algorithm also combines weak classifiers a little differently from AdaBoost. It is worth noting that when $K = 2$, ASMME reduces to AdaBoost.

We note that with enough training of the picked hypothesises (e.g. normal CNN and Residual NN), the error rate of each base learner is less than 0.5. So the original AdaBoost algorithm is still applicable. We implemented both AdaBoost and ASMME in the experiments.

2.2.3 Stacking and super learner

Stacking, also called Super Learning or Stacked Regression, is a class of algorithms that involves training a second-level "meta-learner" to find the optimal combination of the base learners. Unlike bagging and boosting, the goal in stacking is to ensemble strong, diverse sets of learners together. In other words, after we have a list of base classifiers, we use the output from the base classifiers as input, then train a network to get final output.

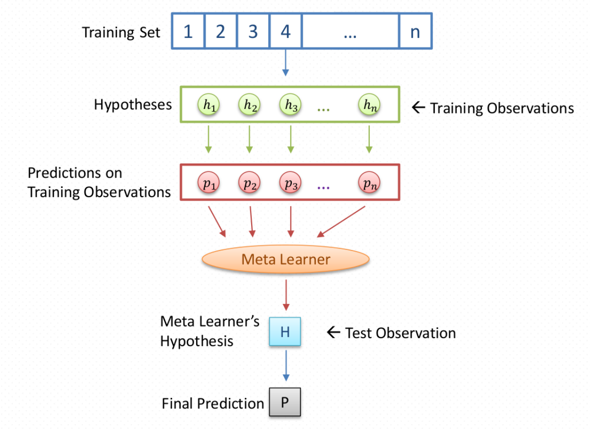

Figure 3: The stacking framework

The underlying idea of stacking is to learn whether training data have been properly learned, If a particular base-classifier incorrectly learned a certain region of the feature space, the second tier (meta) classifier may be able to detect this undesired behavior. Along with the learned behaviors of other classifiers, it can correct such improper training.

Unlike voting, stacking is a different approach for combining models. The difference between the stacking framework and voting is that: (1) In contrast to stacking, no learning takes place at the meta-level when combining classifiers by a voting scheme. (2) Label that is most often assigned to a particular instance is chosen as the correct prediction when using voting.

Super learner is an extension of stacking. It is a cross-validation based ensemble framework, which minimizes cross-validated risk for the combination. We implemented both stacking and super learner. The meta-model is described in *section 3.2.2*. And for stacking, we trained the meta-model with the original training set. For super learner, we split the original training samples to $45,000$ training data and $5,000$ validation data. And the validation set is used to train the meta-model.

2.2.4 Snapshot ensemble

The last ensemble method employed in our project is called Snapshot ensemble developed by Huang et al. [28]. This method focuses on the seemingly-contradictory goal of learning an ensemble of multiple neural networks without incurring any additional training costs. This goal can be achieved by manipulating the learning rate. It leverages the non-convex nature of neural networks and the ability of SGD to converge to and escape from local minima on demand. Instead of training M neural networks independently from scratch, it lets SGD converge M times to local minima along its optimization path. Each time the model converges, it saves the weights and add the corresponding network to ensemble. Then restart the optimization with a large learning rate to escape the current local minimum. More specifically, it adopts the cycling procedure suggested by Loshchilov & Hutter [29], in which the learning rate is abruptly raised and then quickly lowered to follow a cosine function (see Fig. 4). Because the final ensemble consists of snapshots of the optimization path, This approach is referred as Snapshot Ensembling (see Fig. 5). The original paper claimed that the training time for the entire ensemble is identical to the time required to train a single traditional model. However in our experiments (see *section 3*), we found that this claim might be questionable.

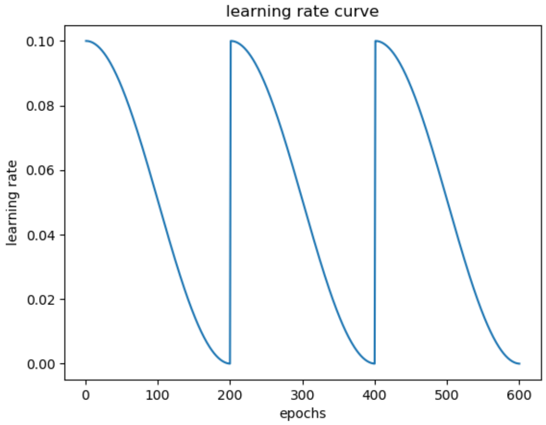

Figure 4: learning rate cycling procedure

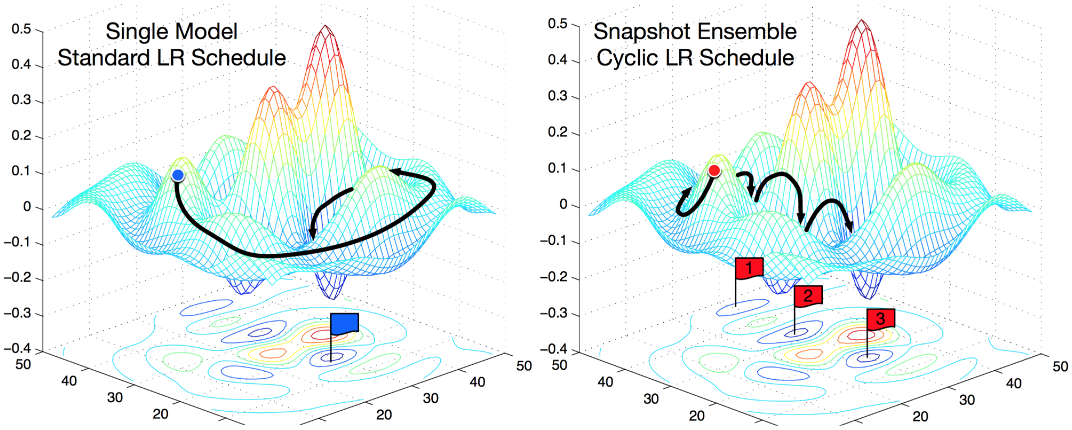

Figure 5: **Left**: Illustration of SGD optimization with a typical learning rate schedule. The model converges to a minimum at the end of training. **Right**: Illustration of Snapshot Ensembling. The model undergoes several learning rate annealing cycles, converging to and escaping from multiple local minima. It takes a snapshot at each minimum for test-time ensembling. [28]

3 Experiment

3.1 Cifar-10

We use CIFAR-10 [30], a common benchmark in machine learning for image recognition, to learn and evaluate our models.

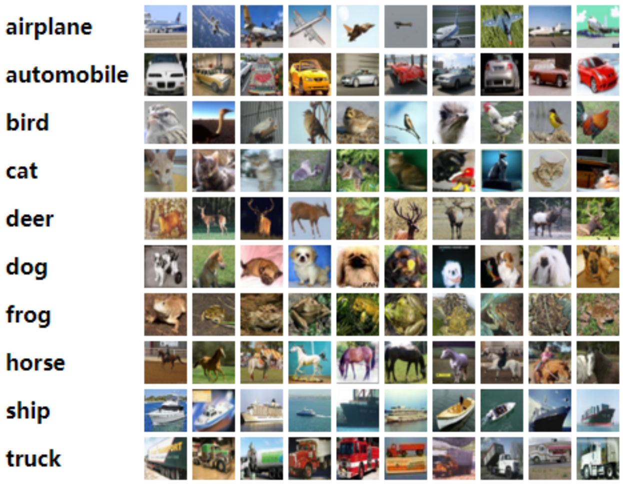

Figure 6: Cifar10 image examples

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The class labels are: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck (see Fig. 6).

3.2 Training

3.2.1 Normal convolution neural network

The CNN model used in our experiments is described below (see Fig. 7). We call this the normal CNN model. In the first block of the convolutional neural network, we first start with a convolution layer with 32 filters to get 32 feature maps for each original input graph. Then we apply another convolution layer again with 32 filters. Next, we use a max pooling layer to reduce dimensionality of each feature map but retains the most important information, and a dropout layer to randomly set 25% of outputs from pooling layer to be zeros.

Using outputs from the first block as inputs, the second block has similar structure as well. We again add two convolution layers but with 64 filters. Then we apply a max pooling layer and a dropout layer with 25% drop rate.

In the last block, we first add a flatten layer to transform the input image matrices into data vectors. Then we use one dense layer with ReLU activation function to summarize data into only 512 variables for dimension reduction. Following that, we use a dropout layer with 50% drop rate to prevent overfitting. Finally, we add a dense layer with Softmax activation function to summarize data into 10 variables, corresponding to the probabilities of 10 classes.

The size of all filters are 3×3 pixels. The activation function of all convolution layers are ReLU. The size of all pooling windows are 2×2 pixels.

3.2.2 Meta model in stacking and super learner

The meta-model combines outputs from different base learners and calculate a final output probability vector. In our case, the outputs of models $(M_1, ..., M_L)$ based on original data, which are the inputs of meta-model, are 10-dimensional vectors representing estimated probabilities for 10 classes. That is, the dimension of input values are $10 \times L$. Our meta-model adds one hidden layer H with dimension $10 \times L$ and one output layer O with dimension 10. We use ReLU between M and H and Softmax between H and O as activation functions. Categorical cross-entropy is used to evaluate model accuracy.

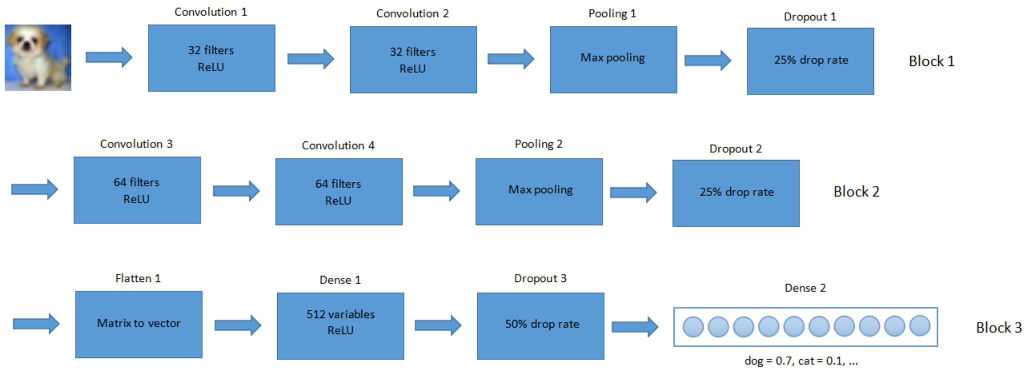

Figure 7: CNN model structure

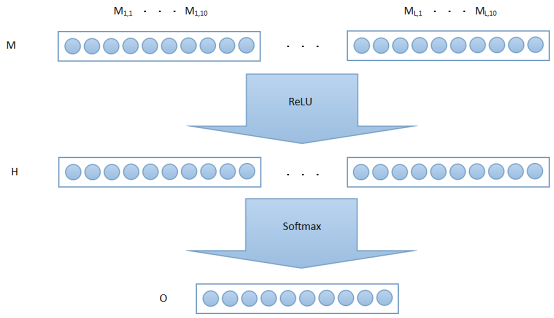

Figure 8: Meta-model structure

3.3 Result

Section 3.3.1 and *section 3.3.2* studied the ensemble methods of normal CNN and Residual NN (with different depths) respectively. *Section 3.3.3* studied the ensemble of base learners with different structures.

All the models are implemented and tested using Keras [31] (version 2.1.2) with tensorflow [32] (version 1.4.0) as backend. You can download the code from the below Github page.

`https://github.com/zhangyaqi1989/Ensemble-Methods-for-Image-Classification`

In tables of the below sections, Bagging (random + mv) stands for bagging method with randomly pick training set and using majority vote to combine base learners. Bagging (random + wa) stands for bagging method with randomly pick training set and using weighted average to combine base learners. Bagging (all + mv) stands for bagging method with the same training set and using majority vote to combine base learners. Bagging (all + mv) stands for bagging method with the same training set and using majority vote to combine base learners. Bagging (all + wa) stands for bagging method with the same training set and using weighted average to combine base learners.

3.3.1 Ensemble of normal CNN

In this part, we tested the performance of different ensemble methods using different numbers of normal CNN base learners. Snapshot ensemble is trained using $100 \times L$ epochs (L is the number of base learners) with $\alpha_0 = 0.0001$. ASMME and AdaBoost both sample $3m$ ($m = 50,000$ is the size of training set) training points for each training stage. And both of them use 40 epochs to train a base learner. All the other ensemble methods use 100 epochs to train a base learner.

Table 1: CNN with 3 base learners

Ensemble method	Best base learner accuracy	Ensemble accuracy	Increased accuracy
Bagging (random+mv)	79.68%	82.00%	2.32%
Bagging (random+wa)	79.68%	82.82%	**3.14%**
Bagging (all+mv)	80.40%	82.29%	1.89%
Bagging (all+wa)	80.40%	82.72%	2.32%
SAMME (3m)	79.87%	82.43%	2.56%
Adaboost (3m)	80.11%	82.45%	2.34%
Stacking	80.05%	82.84%	2.79%
Super learner	80.06%	**82.95%**	2.89%
Snapshot	80.15%	82.00%	1.85%

Table 2: CNN with 5 base learners

Ensemble method	Best base learner accuracy	Ensemble accuracy	Increased accuracy
Bagging (random+mv)	79.76%	82.36%	2.60%
Bagging (random+wa)	79.76%	82.74%	2.98%
Bagging (all+mv)	80.89%	82.93%	2.04%
Bagging (all+wa)	80.89%	83.47%	2.58%
SAMME (3m)	80.03%	83.34%	**3.31%**
Adaboost (3m)	80.35%	82.95%	2.60%
Stacking	80.77%	**83.64%**	2.87%
Super learner	80.06%	82.83%	2.77%
Snapshot	80.66%	82.04%	1.38%

Table 3: CNN with 7 base learners

Ensemble method	Best base learner accuracy	Ensemble accuracy	Increased accuracy
Bagging (random+mv)	80.04%	82.54%	2.50%
Bagging (random+wa)	80.04%	82.91%	2.87%
Bagging (all+mv)	80.55%	83.42%	2.87%
Bagging (all+wa)	80.55%	83.62%	3.07%
SAMME (3m)	79.96%	83.08%	3.12%
Adaboost (3m)	79.93%	83.24%	3.31%
Stacking	80.64%	**84.12%**	**3.48%**
Super learner	80.34%	83.32%	2.98%
Snapshot	80.96%	82.53%	1.57%

From the results listed above (Table 1, 2, 3), we can see weighted averaging performs better than majority voting as an approach for combining base learners. Among all the ensemble methods, stacking works best (super learner is specical kind of stacking).

Also comparing results of ensemble with different number of base learners, we found more base learners lead to better performance which is reasonable. Snapshot ensemble does not perform as well as other ensemble methods due to its relatively low classifier diversity. Since it uses a single training to generate all the base learners.

3.3.2 Ensemble of residual neural network

In this part, we tested the performance of different ensemble methods on different structures of residual neural networks. Snapshot ensemble is trained using $200 \times L$ epochs with $\alpha_0 = 0.002$. ASMME and AdaBoost both sample $3m$ ($m = 50,000$ is the size of training set) training points for each training stage. And both of them use 70 epochs to train a base learner. We found that sampling $3m$ in Adaboost and SAMME with 70 epochs training is not enough. So we also tried sampling m in

AdaBoost and SAMME with 200 epochs training. All the other ensemble methods use 200 epochs to train a base learner. In Table 4, ResNet20 stands for residual network with depth = 20.

Table 4: ResNet20 with 3 base learners

Ensemble method	Best base learner accuracy	Ensemble accuracy	Increased accuracy
Bagging (random+mv)	92.14%	93.43%	1.29%
Bagging (random+wa)	92.14%	93.70%	1.56%
Bagging (all+mv)	92.05%	93.33%	1.28%
Bagging (all+wa)	92.05%	93.67%	1.62%
SAMME (3m)	88.42%	91.68%	3.26%
AdaBoost (3m)	88.23%	91.82%	**3.59%**
SAMME (m)	91.22%	93.00%	1.78%
AdaBoost (m)	90.92%	92.79%	1.87%
Stacking	92.23%	**93.71%**	1.48%
Super learner	91.91%	93.37%	1.46%
Snapshot	89.13%	90.38%	1.25%

Table 5: ResNet32 with 3 base learners

Ensemble method	Best base learner accuracy	Ensemble accuracy	Increased accuracy
Bagging (random+mv)	91.96%	93.41%	1.45%
Bagging (random+wa)	91.96%	93.50%	1.54%
Bagging (all+mv)	92.30%	93.49%	1.19%
Bagging (all+wa)	92.30%	93.80%	1.50%
SAMME (3m)	85.58%	89.37%	**3.79%**
AdaBoost (3m)	88.58%	92.01%	3.43%
SAMME (m)	90.95%	93.05%	2.10%
AdaBoost (m)	91.11%	93.08%	1.97%
Stacking	92.46%	**94.10%**	1.64%
Super learner	92.53%	93.88%	1.35%
Snapshot	89.50%	90.83%	1.33%

Table 6: ResNet44 with 3 base learners

Ensemble method	Best base learner accuracy	Ensemble accuracy	Increased accuracy
Bagging (random+mv)	92.60%	93.80%	1.20%
Bagging (random+wa)	92.60%	93.99%	1.39%
Bagging (all+mv)	92.86%	94.10%	1.24%
Bagging (all+wa)	92.86%	94.36%	1.50%
SAMME (3m)	85.76%	89.28%	**3.52%**
AdaBoost (3m)	88.51%	91.71%	3.20%
SAMME (m)	91.34%	93.31%	1.97%
AdaBoost (m)	91.20%	93.28%	2.08%
Stacking	92.63%	**94.32%**	1.69%
Super learner	91.73%	93.64%	1.91%
Snapshot	89.47%	90.39%	0.92%

Table 7: ResNet56 with 3 base learners

Ensemble method	Best base learner accuracy	Ensemble accuracy	Increased accuracy
Bagging (random+mv)	92.64%	93.82%	1.18%
Bagging (random+wa)	92.64%	94.10%	1.46%
Bagging (all+mv)	92.75%	93.71%	0.96%
Bagging (all+wa)	92.75%	94.02%	1.27%
SAMME (3m)	88.75%	92.35%	**3.60%**
AdaBoost (3m)	88.22%	91.39%	3.17%
SAMME (m)	91.16%	93.09%	1.93%
AdaBoost (m)	90.92%	93.14%	2.22%
Stacking	92.71%	**94.45%**	1.74%
Super learner	92.68%	94.18%	1.50%
Snapshot	88.83%	90.21%	1.38%

From the results listed above (Table 4, 5, 6, 7), we can see weighted averaging performs better than majority voting as an approach for combining base learners. Among all the ensemble methods, stacking works best.

Also comparing results of ensemble with different depth of residual NN, we found deeper residual neural network tends to have higher predict accuracy which agrees with the theory. Same as CNN, snapshot ensemble does not perform as well as other ensemble methods due to its relatively low classifier diversity. Since it uses a single training to generate all the base learners and its learning rate strategy might be too simplified for training residual neural network.

3.3.3 Ensemble of combining residual neural network with different depths

In this part, we loaded the saved models trained in *section 3.3.2* randomly and use stacking ensemble to combine the models. We list some of the results below (RN stands for ResNet).

Table 8: Stacking ensemble of 3 base learners with different structures

Combination	Best base learner accuracy	Ensemble accuracy
1 RN20, 1 RN32, 1RN56	92.53%	93.97%
2 RN44, 1RN56	92.63%	**94.27%**
2 RN20, 1RN110	92.54%	94.15%

Table 9: Stacking ensemble of 5 base learners with different structures

Combination	Best base learner accuracy	Ensemble accuracy
3 RN32, 1 RN44, 1 RN56	92.63%	**94.64%**
1 RN20, 2 RN32, 1 RN44, 1RN56	93.15%	94.48%
1 RN20, 2 RN32, 2 RN44	92.86%	94.51%

From Table 8 and Table 9, we can see combining base learners with different structure or more base learners in an ensemble can lead to good performance.

4 Conclusions and limitations

4.1 Limitations and future work

Due to the limited computing power we own, we did not conduct any kinds of parameter tuning (e.g. like changing the filter size in CNN). Also since ensemble method is computationally expensive (e.g. it takes 20+ hours to train an ensemble for ResNet44 on a machine with a GTX 1080), each case was tested only once without averaging. In the future, we planned to (1) test ensemble methods on deeper residual neural network (e.g. ResNet118); (2) try ensemble methods on other types of deep neural network (e.g. VGG, LeNet); (3) try other ensemble methods (e.g. bayes optimal classifier)

4.2 Discussion and conclusion

In this project, we implemented sevaral ensemble methods including bagging, AdaBoost, ASMME, stacking, super learner and snapshot ensemble; applied them to both normal CNN and residual neural network with different depth; compared the relative performance of different ensemble methods. We found

1. Ensembles can often perform better than any single classifier;
2. Combining more base learners tend to have better performance;
3. Weighted averaging is superior to majority voting in combining base learners;
4. Stacking usually has better performance than bagging, AdaBoost, ASMME and snapshot ensemble; And the meta-model training in stacking is much more efficient than training a base learner. So stacking achieves good balance between performance and efficiency.
5. Combining models with different structures is worth to try;

References

[1] Hansen, L. K., & Salamon, P. (1990). Neural Network Ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12, 993-1001.

[2] Krogh, A., & Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In149 G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in neural information processing systems*, pp. 231-238. Cambridge, MA: MIT Press.151

[3] Ju, C., Bibaut, A., & van der Laan, M. J. (2017). The Relative Performance of Ensemble Methods with Deep Convolutional Neural Networks for Image Classification. arXiv preprint arXiv:1704.01664.

[4] Van der Laan, M. J., Polley, E. C., & Hubbard, A. E. (2007). Super learner. Statistical applications in genetics and molecular biology, 6(1).

[5] Breiman, L. (1996a). Bagging predictors. Machine learning, 24(2), 123-140.

[6] Friedman, 2001

[7] Freund, Y., & Schapire, R. E. (1995). A desicion-theoretic generalization of on-line learning and an application to boosting. European conference on computational learning theory. pp. 23-37. Springer, Berlin, Heidelberg.

[8] Moghimi, M., Belongie, S. J., Saberian, M. J., Yang, J., Vasconcelos, N., & Li, L. J. (2016). Boosted Convolutional Neural Networks. BMVC.

[9] Zhu, J., Rosset, S., Zou, H., & Hastie, T. (2006). Multi-class adaboost. Ann Arbor, 1001(48109), 1612.

[10] Wolpert, D. H. (1992). Stacked generalization. Neural networks, 5(2), 241-259.

[11] Breiman, L. (1996b). Stacked regressions. Machine learning, 24(1), 49-64.

[12] Sinisi, S. E., Polley, E. C., Petersen, M. L., Rhee, S. Y., & van der Laan, M. J. (2007). Super learning: an application to the prediction of HIV-1 drug resistance. Statistical applications in genetics and molecular biology, 6(1).

[13] Davies, M. M., & van der Laan, M. J. (2016). Optimal spatial prediction using ensemble machine learning. The international journal of biostatistics, 12(1), 179-201.

[14] Benkeser, D., Ju, C., Lendle, S., & van der Laan, M. (2017). Online cross-validation-based ensemble learning. *Statistics in Medicine*.

[15] Hothorn, T., Bühlmann, P., Dudoit, S., Molinaro, A., & Van Der Laan, M. J. (2005). Survival ensembles. Biostatistics, 7(3), 355-373.

[16] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In149 G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), Advances in neural information processing systems, pp. 1097-1105. Cambridge, MA: MIT Press.151

[17] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770-778.

[18] Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

[19] Buciluă, C., Caruana, R., & Niculescu-Mizil, A. (2006, August). Model compression. Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 535-541). ACM.

[20] Hinton, G., Vinyals, O.,& Dean, J. (2015). Distilling the knowledge in a neural network. arXiv preprint arXiv:1503.02531.

[21] http://cs231n.github.io/convolutional-networks/

[22] https://adeshpande3.github.io/adeshpande3.github.io/

[23] Ioffe, S., & Szegedy, C. (2015, June). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *In International Conference on Machine Learning* (pp. 448-456).

[24] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[25] Kolen, J. F., & Pollack, J. B. (1991). Back propagation is sensitive to initial conditions. *Advances in Neural Information Processing Systems*, Vol. 3, pp. 860-867 San Francisco, CA. Morgan Kaufmann.

[26] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.

[27] M. J. van der Laan, E. C. Polley & A. E. Hubbard. Super learner. Statistical applications in genetics and molecular biology, 6(1), 2007.

[28] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles: Train 1, get M for free. *ICLR submission*, 2017.

[29] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with restarts. arXiv preprint arXiv:1608.03983, 2016.

[30] Alex Krizhevsky (2009) Learning Multiple Layers of Features from Tiny Images.

[31] https://github.com/keras-team/keras/

[32] https://www.tensorflow.org/