# Unsupervised Aggregation of Deep Models

Amirhossein Afsharrad
Stanford University
afsharrad@stanford.edu

## 1. Introduction

Ensemble methods are a group of algorithms focusing on aggregating multiple base learners to outperform each individual learner. In the context of classification problems, an ensemble algorithm receives the labels generated by multiple classifiers, and outputs one final aggregated label. Many ensemble methods work in a supervised setting, where we have access to a reasonable amount of labeled training data. However, with the labeled data being expensive and difficult to acquire in many applications, it is a reasonable approach to investigate this problem in an unsupervised setting, where we only have a set of unlabeled data.

Another aspect of the aggregation process in ensemble methods is to make decisions based on the structure and performance of each base classifier. As a simple example, if we have 3 classifiers and we know one of them has a 99 percent accuracy while the other two are slightly better than random guessing, we obviously give much more weight to the label produced by the first classifier compared to those of the second and the third classifier. Moreover, our information about the structure and weights of each base classifier can result in a better aggregation. For example, if we have two SVMs, we might be able to use each classifier's weight matrix to come up with a new aggregated set of weights that outperforms each of the base classifiers. However, this is not the case in many applications where we do not have access to anything that lies within the base models. This, for example, can be due to confidentiality of the models, where the model owner will not provide us with the inner structure of their model. Also, the base classifiers may have totally different structures such that even if we know what lies inside them, we can not use the information to improve the aggregation. In these scenarios, we assume a black-box structure for each base classifier, and only rely on the labels that they generate.

Combining the unsupervised setting with the black-box structure, we seem to be quite in the dark to make any useful aggregation. Note that since we do not have access to labeled data, we cannot evaluate the performance of the base classifiers, and it seems that the best we can do is to perform the simple majority voting, and produce the aggregated label as the label supported by the majority of the base classifiers. However, quite surprisingly, with not very strict assumptions, we propose algorithms to outperform the majority vote in this setting.

## 2. Related Work

One of the first works to address the problem of aggregating multiple black-box classifiers with unknown reliabilities to label a set of unlabeled data is [2]. They look at this problem as a simple statistical problem and consider each classifier as a binary symmetric channel, where the output label is the correct with probability $p$ and incorrect with probability $1-p$. The performance probability $p$ is different for each classifier, but is constant for every decision that a specific classifier makes. Other works such as [3] apply the exact same mathematical framework to the binary classification problem. Later, [1] extends the work of [3] to the case of multiclass classification, with an extra assumption of having access to the logits produced by each model, which is a reasonable assumption but violates the pure black-box structure that only assumes access to the produced label. Some other works such as [4] also work on the exact same problem with the same approach, but apply it to other applications. For instance, [4] focuses on this problem in the context of online crowdsourcing. All these works, despite their differences, share the same mathematical structure and assumptions introduced in [3], which is explained in detail in 3.1.

There are other approaches to improve the label accuracies of a base classifier without any labeled data such as self-training, whose theoretical guarantees has been explored in [5]. This is a different approach compared to ours, as it assumes "local-

ity of labels", i.e. if the feature vectors are close in some distance, their corresponding labels must be the same. However, what we propose in this work is based on the "locality of error" assumption, stating that each classifier has similar reliability over close enough feature vectors. This is explained in more detail in 3.2.
In this work, we first use the same framework as [1–4] and apply it to convolutional neural networks for image classification tasks, and then propose a new approach in 3.2 to outperform the classic methods.

## 3. Problem Statement

### 3.1. Uniform Error Model

For simplicity, we consider the binary classification problem with feature space $\mathcal{X} \subseteq \mathbb{R}^d$ and label space $\mathcal{Z} = \{-1, +1\}$. Then, each instance pair $(X, Z) \in \mathcal{X} \times \mathcal{Z}$ is a random variable with probability density function $p(x, z)$ and marginals $P_X(x)$ and $P_Z(z)$. Let $S = \{X_m\}_{m=1}^M$ be a set of unlabeled samples, denoted for simplicity by $X_{1:M}$, with true but unknown labels $Z_{1:M} = \{Z_m\}_{m=1}^M$. Note that $Z_{1:M}$ is not an input of the problem, and we only assume access to a set of unlabeled data, namely $X_{1:M}$. Also, note that the $M$ sample pairs $(X_m, Z_m)$ are I.I.D. realizations from the distribution $p(x, y)$. Let $\{f^k\}_{i=1}^K$ be a set of $K$ binary classifiers of unknown accuracy. Each classifier $f^k : \mathcal{X} \to \mathcal{Z}$ is viewed as a black-box model, and we only assume to have access to the labels they provide on $X_{1:M}$. Let $Y_m^k = f^k(X_m)$ be the label provided by the $k$th classifier on the $m$th sample. The set of all these labels is denoted by $Y_{1:M}^{1:K} = \{Y_m^k | m \in [M], k \in [K]\}$, where $[n] = \{1, \dots, n\}$. We look at $Y_{1:M}^{1:K}$ as a $K \times M$ matrix of labels, with rows and columns corresponding to classifiers and samples respectively. Using only the unlabeled data $X_{1:M}$ and the labels $Y_{1:M}^{1:K}$, we consider the problem of estimating the true labels $Z_{1:M}$.

We measure the performance of each classifier $f_k$ by a parameter $\epsilon_k$, the error probability of the $k$th model, defined as

$$\epsilon_k = \mathbb{P}\left[Y^k \neq Z\right], \tag{1}$$

where $(X, Z)$ is generated according to $p(x, z)$, and $Y^k = f^k(X)$.

In this section, we assume that the classifiers are conditionally independent, in the sense that given the true label $Z$ for a given sample $X$, different classifiers make prediction errors independently. More specifically, we assume

$$\mathbb{P}\left[Y_{1:M}^{1:K} | Z_{1:M}\right] = \prod_{k=1}^{K} \prod_{m=1}^{M} \mathbb{P}\left[Y_m^k | Z_m\right]. \tag{2}$$

This is the most critical assumption in [1–4]. In 3.2, we will discuss the critical shortcomings of this assumption, and propose a novel method to improve the mathematical model of this problem.

We call this structure the uniform error model as it assumes the error for each classifier is uniform over all feature vectors $X$. As a result, in the problem formulation and algorithms to solve the problem, the feature vector $X$ is totally ignored and only $Y$ and $Z$ are considered. In section 3.2, we will generalize this model by taking features into consideration.

As mentioned earlier, the final goal is to estimate $Z_{1:M}$ by observing the labels produced by different classifiers, namely $Y_{1:M}^{1:K}$. Thus, we consider the maximum likelihood estimation problem of $Z_{1:M}$. So the problem of maximizing the log-likelihood can be formulated as

$$\max_{Z_{1:M}} \sum_{k=1}^{K} \sum_{m=1}^{M} \log\left(\mathbb{P}\left[Y_m^k | Z_m\right]\right). \tag{3}$$

The following lemma introduces a more straightforward version of the problem, where one can directly try to solve:

Lemma 1. Let $\gamma_k = \left|\{m | Y_m^k \neq Z_m\}\right|$. Then, the ML problem (3) can be stated as

$$\max_{Z_{1:M}} \sum_{k=1}^{K} \gamma_k \log \epsilon_k + (M - \gamma_k) \log(1 - \epsilon_k). \tag{4}$$

Proof. Omitted. □

Lemme 1 introduces the problem that we would like to solve, but there is one critical challenge left to address, which is the unknown performance of the classifiers. In other words, we do not know the values of $\epsilon_{1:K}$, thus we cannot directly try to solve (4). As suggested by [2,3], we can jointly maximize the likelihood function for $Z_{1:M}$ and $\epsilon_{1:K}$. Thus, the final problem can be stated as

$$\max_{Z_{1:M}, \epsilon_{1:K}} \sum_{k=1}^{K} \gamma_k \log \epsilon_k + (M - \gamma_k) \log(1 - \epsilon_k). \tag{5}$$

### 3.2. Non-Uniform Error Model

#### 3.2.1 Uniform-Error Model Shortcomings

In the uniform error model, we assumed that different classifiers make prediction errors independently.

This assumption is not true in many cases, such as the following two examples:

1. There are easy and difficult examples. In a binary classification problem, some feature vectors $x$ are easy to classify. Easy examples correspond to extreme feature vectors that are classified correctly even by the classifiers with low accuracy, while the difficult examples are hard to classify and need more accurate models to be classified correctly with a high probability. For easy examples, all classifiers have the same (correct) output, so clearly the outputs are correlated.

2. Models can be specialized. Consider a simple classification problem where the inputs are images and we aim to classify whether each image is an animal or not. We may happen to have different models, each specialized to recognize a specific type of animal. For instance, the first model can be trained over a set of (mostly) dogs and random non-animal images, while the second one has observed more horses as animals. This way, if the input is a dog, the first model will produce a correct label with high probability, while the second one has a higher probability to make an error, and the reverse holds if the input is a horse. In this case, the prediction errors made by the two models can well be correlated since when one is most likely to make no error, the other one is likely to make one.

Next, we propose a method to capture this type of complexity and mitigate the effect of error dependence of the models in settings like the above examples.

#### 3.2.2 Problem Setup

The problem settings in terms of inputs and outputs are exactly the same as those of the uniform error model, i.e. we define $X_{1:M}$, $Z_{1:M}$, and $Y_{1:M}^{1:K}$ in the exact same way as in section 3.1. The key difference is the way we handle the error probabilities. We define $\epsilon_k : \mathcal{X} \to [0, 1]$ to be the error probability of the $k$th model when the input is $x \in \mathcal{X}$:

$$\epsilon_k(x) = \mathbb{P}[Y^k \neq Z | X = x] \tag{6}$$

This is a generalization of $\epsilon_k$ defined in (1), where $\epsilon_k$ did not depend on $x$. We argue that, as explained earlier, while it is not quite reasonable to assume independence of errors across different classifiers in general, it is a better assumption when we assume the feature vector $x$ to be given. This means that (2) generalizes to

$$\mathbb{P}\left[Y_{1:M}^{1:K} | Z_{1:M}, X_{1:M}\right] = \prod_{k=1}^{K} \prod_{m=1}^{M} \mathbb{P}\left[Y_m^k | Z_m, X_m\right]. \tag{7}$$

Using a similar notation as in (3), here we wish to maximize the most general form of the likelihood function given by

$$\begin{aligned} &\max_{Z_{1:M}} \mathbb{P}\left[X_{1:M}, Y_{1:M}^{1:K} | Z_{1:M}\right] = \\ &\max_{Z_{1:M}} \mathbb{P}\left[Y_{1:M}^{1:K} | X_{1:M}, Z_{1:M}\right] \mathbb{P}\left[X_{1:M} | Z_{1:M}\right]. \end{aligned} \tag{8}$$

As we have no further information about $X$ and $Z$, we assume $\mathbb{P}[X_{1:M}|Z_{1:M}]$ to be constant, and the maximization problem reduces to

$$\max_{Z_{1:M}} \mathbb{P}\left[Y_{1:M}^{1:K} | Z_{1:M}, X_{1:M}\right] \tag{9}$$

which is equivalent to

$$\max_{Z_{1:M}} \prod_{k=1}^{K} \prod_{m=1}^{M} \mathbb{P}\left[Y_m^k | Z_m, X_m\right]. \tag{10}$$

Next, we need an analogue of Lemma 1 to simplify (10). Lemma 2 provides us with this result:

Lemma 2. The ML problem (10) can be stated as

$$\begin{aligned} \max_{Z_{1:M}} \prod_{m=1}^{M} \prod_{k=1}^{K} &(1 - \epsilon_k(X_m)) \mathbb{1}\left[Y_m^k = Z_m\right] \\ &+ \epsilon_k(X_m) \mathbb{1}\left[Y_m^k \neq Z_m\right]. \end{aligned} \tag{11}$$

Proof. Omitted. □

Similar to the uniform error model case, we face the same challenge, i.e. the problem of not knowing the error probabilities. This time, the problem is even more complicated since we need to deal with $K$ unknown functions $\epsilon_{1:K}(\cdot)$ instead of $K$ unknown constant parameters. Again, similar to the uniform error model case, we will aim to maximize the objective function in (11) jointly over $Z_{1:M}$ and $\epsilon_{1:K}(\cdot)$. Thus, the final problem is

$$\begin{aligned} \max_{Z_{1:M}, \epsilon_{1:K}(\cdot)} \prod_{m=1}^{M} \prod_{k=1}^{K} &(1 - \epsilon_k(X_m)) \mathbb{1}\left[Y_m^k = Z_m\right] \\ &+ \epsilon_k(X_m) \mathbb{1}\left[Y_m^k \neq Z_m\right]. \end{aligned} \tag{12}$$

If we choose each function $\epsilon_k(.)$ arbitrarily, (12) will have a trivial solution with each $\epsilon_k(X_m)$ being zero or one, which is of no use. We need to assume some smoothness constraint over these functions,

and then estimate them. We define the intermediary functions

$$g_k(x) = \ln \frac{1-\epsilon_k(x)}{\epsilon_k(x)}. \tag{13}$$

and estimate them using the Nadaraya-Watson kernel non-parametric regression method

$$\hat{g}(x) = \frac{\sum_{m=1}^{M} y_m K_h(x - x_m)}{\sum_{m=1}^{M} K_h(x - x_m)}, \tag{14}$$

where $K_h : \mathcal{X} \to \mathbb{R}$ is a kernel function, and $h \in \mathbb{R}_+$ is the bandwidth. We now define

$$\eta_m^k = g_k(X_m) = \ln \frac{1-\epsilon_k(X_m)}{\epsilon_k(X_m)}. \tag{15}$$

and state (12) as the final problem

$$\max_{Z_{1:M}, \eta_{1:M}^{1:K}} \sum_{m=1}^{M} \sum_{k=1}^{K} \ln \left[ (1-\epsilon_k(X_m)) \mathbb{1}\left[Y_m^k = Z_m\right] + \epsilon_k(X_m) \mathbb{1}\left[Y_m^k \neq Z_m\right] \right], \tag{16}$$

where the relationship between $g_k(.)$ and $\eta_{1:M}^{1:K}$ is given by (15) and

$$g_k(x) = \frac{\sum_{m=1}^{M} \eta_m^k \tilde{K}\left(\frac{\|x - X_m\|}{h}\right)}{\sum_{m=1}^{M} \tilde{K}\left(\frac{\|x - X_m\|}{h}\right)} \tag{17}$$

and $\tilde{K}(.)$ is a Gaussian kernel.

## 4. Technical Approach

To solve (5), one can search over all $2^M$ choices of $Z_{1:M}$, but since this is computationally infeasible, we use the expectation maximization approach, which was first introduced in [2], where we first solve (5) for a fixed $Z_{1:M}$ for $\epsilon_{1:K}$, and then fix $\epsilon_{1:K}$ and solve for $Z_{1:M}$, and iterate until convergence. The first step solution is given by

$$\epsilon_k = \frac{\gamma_k}{M}, \tag{18}$$

and the second optimization is solved by

$$\hat{Z}_m = \text{sign}\left( \sum_{k=1}^{K} Y_m^k \ln \frac{1-\epsilon_k}{\epsilon_k} \right). \tag{19}$$

This is explained in Algorithm 1.

**Algorithm 1** EM – Uniform Error Model

initiate $Z_{1:M}$
while $Z_{1:M}$ has not converged do
  update $\epsilon_{1:K}$ using (18)
  update $Z_{1:M}$ using (19)
end while
return $Z_{1:M}$

Similarly, to solve (16), we use Algorithm 2.

**Algorithm 2** EM – Non-Uniform Error Model

initiate $Z_{1:M}$
while $Z_{1:M}$ has not converged do
  update $\eta_{1:M}^{1:K}$ by solving (16) for fixed $Z_{1:M}$
  update $Z_{1:M}$ using (19) by letting $\epsilon_k = \epsilon_k(X_m)$
end while
return $Z_{1:M}$

Note that solving for $\eta_{1:M}^{1:K}$ with fixed $Z_{1:M}$ is a convex problem in $\eta_{1:M}^{1:K}$, so it can be solved efficiently.
In order to implement our methods, we use CIFAR10 and ImageNet data. To test for binary classification, we choose different pairs of classes from those datasets and compare the accuracy of our two algorithms. Our baseline is the simple majority vote, and we expect Algorithm 2 to outperform that.

## 5. Results

In this section, we compare our tow algorithms performances with each other and with the baseline majority voting. We use $K = 5$ local models and run $N = 100$ simulations and average the results to generate each plot. We compare the results in different scenarios and discuss each one in a subsection. For simplicity, we call Algorithm 1 and 2 EM0 and EM respectively. EM stands for Expectation Maximization and EM0 is the zero version of EM, where we zero out the effect of the features, namely $X$.

### 5.1. Ideal Scenario

As a first test of our algorithms, we choose a random binary classification set of data from CIFAR10, and generate $Y_{1:M}^{1:K}$ with different chosen $\epsilon_{1:K}(.)$. We manually choose $\epsilon_{1:K}(.)$ functions to be piecewise constant with different values, such that the average error of all models is 0.25. This is the ideal scenario for our algorithm, because the labels are exactly generated as our model describes. Figure 1 shows that our algorithm (EM) performs better than the majority vote, and Algorithm 1 (EM0) performs even worse than majority vote in this case, which confirms what we explained in 3.2.1.

We can interpret the results in Figure 1 in more detail. Note that $m$ is the number of data points. The blue curve, corresponding to the majority vote result, is almost a horizontal line. This makes perfect sense since the performance of the majority vote

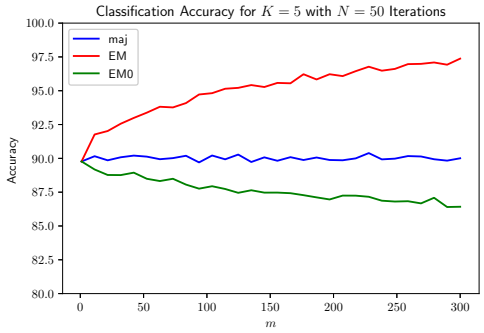


Figure 1. classification accuracy in ideal setting with all local models having same overall accuracy

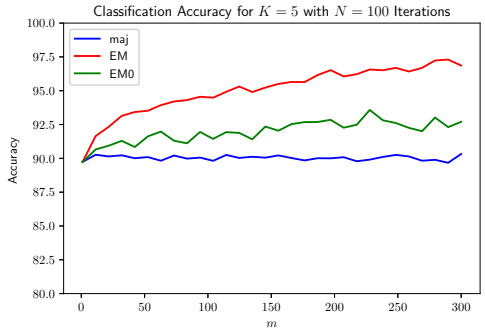


Figure 2. classification accuracy in ideal setting with local models having different overall accuracies and non-constant $\epsilon_k$

does not depend on the number of samples we have. It only depends on how each base classifier works and how accurate it is. In other words, majority vote only considers the local labels for each data point to come up with an aggregated label, so it does not matter how many samples we have. Next, we observe that the red curve is increasing. Note that with $m = 1$, we expect all the algorithms to perform exactly like majority vote since with only one example, we have nothing to rely on except what majority suggests. However, with the increased number of samples, by solving (16), our model learns which model performs better on which part of the feature space, and gives more weight to that model when the input belongs to that part of the feature space. This structure can be learned better when we have more samples, as the model can observe how the similar or different the labels of local models are for different parts of the feature space. This explains why the red curve is increasing and why our model outperforms majority more and more when we have a larger number of samples. Next, we observe that the green curve is below majority. It means that EM0 is performing even worse than the simple majority. This also makes sense because EM0 does not account for the different performances of models for different inputs. It only looks at the overall performance of the local models, and the labels for this simulation are generated in a way that the overall performances are the same (with 75% accuracy). When all the models have the same overall accuracy, EM0 cannot do better than majority, which is intuitively correct since when some models have similar accuracies, there seems to be nothing better than majority. However, when there is underlying difficult or easy examples and we add model specialization (both described in section 3.2), this algorithm can work even worse than majority, which confirms that our novel EM algorithm is capturing the complicated structure of the data.

Before moving on to more realistic cases, we test two more scenarios. First, we change the process of label generation such that two of the local models have an overall accuracy of 90% while the other three have the old 75% accuracy and run our algorithms. The results are shown in Figure 2.

We observe that in this scenario, EM0 outperforms majority but is not as good as EM. This makes perfect sense since the overall performance of models are not the same, and EM0 can recognize which models have a better performance and by giving more weight to them, outperforms majority. However, EM performs better than both since not only it recognizes better models, but it also understands which model is more reliable for which sample, and gives different weights based on the feature vector.

Finally, we consider a case where the performance of all local models are uniform across the feature spaces. This means that no matter what the feature vector is, the $i$th model generates the true label with probability $p_i$, with $p_1 = p_2 = 0.9$ and $p_3 = p_4 = p_5 = 0.75$ for this simulation. The result is shown in Figure 3.

This means that $\epsilon_k(x)$ in (6) is constant, and the performance of models does not depend on the feature vectors. In this case, we expect EM to be similar to EM0 because the advantage of EM is to learn the variations of $\epsilon_k(x)$ when the feature vector $x$ changes. So when $\epsilon_k(x)$ is a constant, EM reduces to EM0. This is confirmed by Figure 3, where we see that the red and green curve coincide (and both outperform majority).

### 5.2. Realistic Scenario

To test our algorithms in the most realistic scenario, we train $K = 5$ deep convolutional neural networks over CIFAR10 and ImageNet datasets for binary classification tasks. To make the classifi-

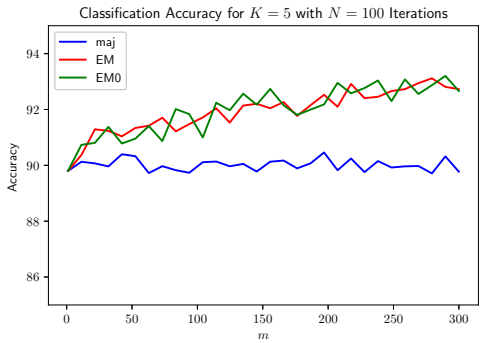


Figure 3. classification accuracy in ideal setting with local models having different overall accuracies and constant $\epsilon_k$

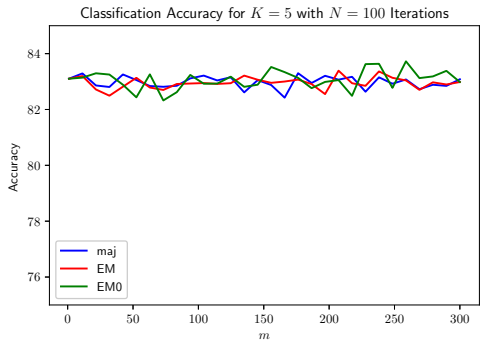


Figure 4. classification accuracy for CNNs with local models having different overall accuracies and non-constant $\epsilon_k$

cation binary, we consider our two classes to be "animal" and "non-animal", and combine multiple classes from original datasets (i.e. multiple animal classes and multiple non-animal classes) to generate the binary dataset. As we discussed before, the most important factor for these models is their accuracy. We fix train these models until they achieve the desired accuracies (which are typically 75% or 90 % for our simulations). We use different structures, ranging from a very simple one layer network without any non-linearity to a complex 10-layer convolutional network with ReLU activation, batch normalization, and dropout. Also, we choose the number of epochs, batch size, and learning rate in different ways to make our models achieve a specific accuracy.

We have run the simulations for dozens of different structures and parameters, but since the results are quite similar, we only report the results for a specific set of $K = 5$ models with the following properties:

- A one-layer network with no non-linearity.
- Two five-layer ReLU networks consisting of three convolutional layers and two fully connected layers.
- Two nine-layer ReLU networks consisting of seven convolutional layers and two fully connected layers.

Since we observe that the results are quite similar for different structures, we do not emphasize on the specific details of the models. One other important specification of our simulations is how we make each model perform better on a specific region of the feature space while having a worse performance on some other region. To do this, we bias the training data for each local model. As a simple example, we make the training data of the first model consist of mostly dogs as animals and cars as non-animals, while the training data of the second model mostly consists of horses and boats as animals and non-animals. If we train these two models such that they both have 75% accuracy, they have the same overall performance but one performs better on a specific part of the feature space (i.e. dogs and cars) while the other one is better on another sub-domain of the feature space (i.e. horses and boats).

First, we run the simulations such that three local models have 75% accuracy and two have 90% accuracy, and we inject the described bias so that models have non-uniform error over the feature space. The results are shown in Figure 4.

The results of Figure 4 seem disappointing as none of our two algorithms is better than majority. However, there is one simple and super-important reason for this. By looking at the optimization problems that EM0 and EM solve, we observe that one of the most crucial mathematical elements of these problems are the pair-wise distance of data points, i.e. images. In our implementations, we are simply using the pixel-wise Euclidean distance of two images, and this is not a good measure of distance for our images. The reason for this is obvious. One can shift one image or rotate it slightly, such that the two images are extremely similar but their pixel-wise distance could grow very large. So we need a better measure of distance. We use two ideas to address this problem:

1. We use color histogram and histogram of oriented gradients (HOG) of the images instead of the images themselves.
2. We use an auto-encoder to encode the images and use the encoded features instead of raw images.

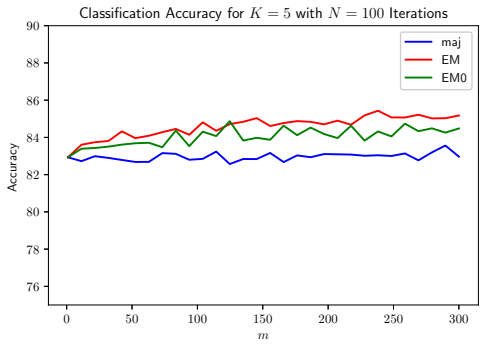


Figure 5. classification accuracy for CNNs with local models having different overall accuracies and non-constant $\epsilon_k$ using color histogram and HOG to measure distance in feature space

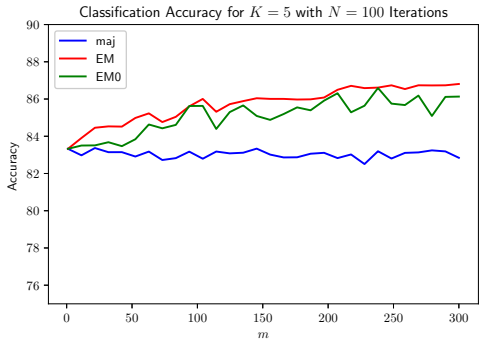


Figure 6. classification accuracy for CNNs with local models having different overall accuracies and non-constant $\epsilon_k$ using auto-encoder to measure distance in feature space

Using the first method (color histogram and HOG), the disappointing results of Figure 4 turns into Figure 5. Note that Figure 5 is a dual of Figure 2 but with a totally realistic implementation. We also use an auto encoder to solve the problem of pixel-wise distance, which results in what is shown in Figure 6. The auto-encoder that we use has a simple eight-layer structure consisting of convolutional and fully connected layers.

We observe that the auto-encoder results are slightly better than those pictured in Figure 5. One reason for this is that auto-encoders can clearly capture a more detailed and meaningful structure of the feature spaces compared to simple color histograms of HOGs. In other words, the output vectors from the auto-encoders are much more meaningful, such that when we use them to compute pairwise distances between images, the distances are also much meaningful and rely much less on random features of the image pixels.

We also generate a dual of Figure 3, where we

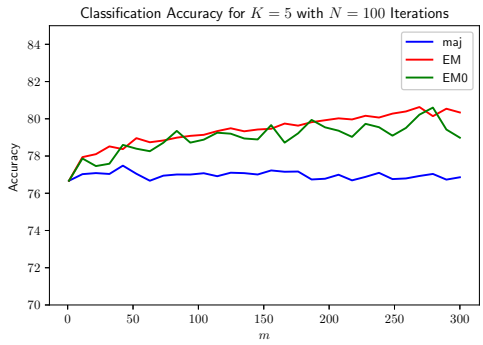


Figure 7. classification accuracy for CNNs with local models having different overall accuracies and almost constant $\epsilon_k$ using auto-encoder to measure distance in feature space

train models with 75% and 90% accuracies and choose the training data for each local model to have the same distribution as the others (so no model is specialized for a specific kind of input). The result is shown in Figure 7.

As we expect, Figure 7 is quite similar to Figure 3 in the sense that EM and EM0 are almost as good as each other. However, while in Figure 3 the two algorithms are essentially producing the same accuracy, in Figure 7 EM is still a little better than EM0. This could be due to other hidden patterns of the data that makes the $\epsilon_k(x)$ functions non-constant. This is actually the very first reason that we designed EM. When it comes to the complicated deep models, there might be unknown underlying structures that make the performances non-uniform, resulting in a non-constant $\epsilon_k(x)$. While we might not be able to find these structures and patterns directly, our proposed algorithm uses them to decide which model performs better for which input, and gives a larger weight to that model for that specific input.

Also, it is worth noting that the improvement we get by using EM or EM0 compared to simple majority vote in the realistic case (with real deep models) is smaller than what we got when we generated the labels in the exact way that our model assumes. This can be observed by comparing Figures 5 and 2, or Figures 7 and 3. This also makes sense since real data and models do not behave exactly the same as what our proposed mathematical model suggests. However, the significant performance of our models suggest that the mathematical propositions are quite sensible and to some extent can explain how the real world models work.

## 6. Conclusion and Future Work

In this work, we focused on the problem of aggregating multiple base classifiers to obtain a better aggregated classification model. We assumed a black-box unsupervised setting, where we only have the input and outputs of the models and we do not know what the true labels of our data are. So we only have a set of unlabeled data along with $K$ classifiers. While one could imagine that we can do nothing better than a majority voting, we proposed two different algorithms based to outperform the simple majority rule. The first one assumes that each model has a constant error probability for all the inputs and by estimating these probabilities, decides which classifier is more accurate. The second model extends this idea by assuming that different models can have different performances based on the input that they are given. Our second algorithm can outperform both the majority and the first algorithm in certain scenarios, and we observed that both of these methods can result in significant improvements when it comes to black-box aggregation of convolutional neural networks.

This work can be extended by analyzing other types of data (other than image), exploring more complicated networks (such as RNNs), or trying to come up with a mathematical proof and guarantees for the performances of the proposed algorithms.

## References


[1] Mehmet Eren Ahsen, Robert M Vogel, and Gustavo A Stolovitzky. Unsupervised evaluation and weighted aggregation of ranked classification predictions. Journal of Machine Learning Research, 20(166):1–40, 2019. 1, 2

[2] A. P. Dawid and A. M. Skene. Maximum likelihood estimation of observer error-rates using the em algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1):20–28, 1979. 1, 2, 4

[3] Fabio Parisi, Francesco Strino, Boaz Nadler, and Yuval Kluger. Ranking and combining multiple predictors without labeled data. Proceedings of the National Academy of Sciences, 111(4):1253–1258, 2014. 1, 2

[4] Filipe Rodrigues, Francisco Pereira, and Bernardete Ribeiro. Learning from multiple annotators: Distinguishing good from random labelers. Pattern Recogn. Lett., 34(12):1428–1436, sep 2013. 1, 2

[5] Colin Wei, Kendrick Shen, Yining Chen, and Tengyu Ma. Theoretical analysis of self-training with deep networks on unlabeled data. In International Conference on Learning Representations, 2021. 1