,

Seeking agreement

an ensemble approach to epistemic uncertainty

Anonymous Author(s)
Affiliation
Address
email

Abstract

The abstract paragraph should be indented ½ inch (3 picas) on both the left- and right-hand margins. Use 10 point type, with a vertical spacing (leading) of 11 points. The word **Abstract** must be centered, bold, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

1 Introduction

Quantifying prediction uncertainty is an active area of research in machine learning and elsewhere [1, 12, 8, 4, 3, 9]. A useful way to reason about uncertainty is to divide it into *aleatoric* vs *epistemic* uncertainty.We demonstrate this partition by considering our model to be a DNN for binary classification. We consider a single example x our goal is to preict the associated label $y \in \{-1, +1\}$.

Suppose the learned prediction function f (the weights of the NN) is a fixed function $f : X \to \mathbb{R}$. The aleatoric, ot intrinsic uncertainty is captured by the conditional probability $p_a = P(y = +1|f(x) > a)$. This so-called "calibration function" is "intrinsic" because it does not depend on the training set.

On the other hand the epistemic uncertainty orresponds to *model uncertainty* i.e. the uncertainty the we have in the model

knowledge as to which function f is best.

divide prediction uncertainty is to divide preidction uncertainty it into aleatory vs. epistemic uncertainties. We briefly describe what this means in the context of binary classification using a NN. Denote a binary example by (x, y). Let M

useful way to distinguish between different types of uncertainty is to differentiate aleatory or irreducible uncertainty from epistemological or reducible uncertainty. Consider predicting the outcome of flipping a coin whose (unknown) probability of heads is $p = 1/4$. If we know p we should always predict "tails" and will be incorrect with probability $1/4$. This uncertainty is irreducible, because no amount of additional data will change it. On the other hand, if we flip the coin n times and find that m of the outcomes were "heads" we can estimate the true value of p to be approximately $\frac{m}{n} \pm \frac{1}{\sqrt{n}}$. The range $\pm\frac{1}{\sqrt{n}}$ is the epistemological or reduxible uncertainty, because it relates to our *knowledge* regarding p which can be improved by increasing n.

In this paper we show how epistemological uncertainty in DNN can be reduced using a simpl ensemble based method. Our approach is based on projecting the model uncrtainty onto instances and measuring the resulting per-nstance uncertainty.

Yoav till here.

Modern DNN have increasingly complex architectures involving tens of millions of parameters. Experience shows that increasing the depth and width of the DNN generally leads to better performance.

Submitted to 37th Conference on Neural Information Processing Systems (NeurIPS 2023). Do not distribute.

The question we ask is whether increasing the number of parameters is equally important for every test example. We carried some experiments using CIFAR-10. Not surprisingly, we find that different examples require different levels of complexity.

What *is* surprising, however, is the fact that for *most* examples a very simple network suffices, only a small fraction of the examples benefits from complex networks. We call examples of the first type *easy* and those of the second type *hard*. our results suggest that the hardness of an example is an *inherent* property of the example and depends only weakly on the type and architecture of the network.

Prior work on this subject [] is based on the amount of training required to get the example labeled correctly. The problem with this measure is that it requires knowing the correct label and therefor cannot be used at test time. We suggest an ensemble-based approach to measure hardness that does not require knowing the true label.

2 Related Work

[14]introduced a framework to deal with multiclass classification task in a one-vs-rest(OVR) way, which is similar to ours that treat each 10-class classification task as 10 binary classification tasks. It also trains multiple classifiers. However, its classifiers are not independent and they work in a stacked order and each classifier is responsible for one class. This paper also pay attention to easy classes and confusing classes and it uses the classes' easiness to arrange the order of the classifiers. The key differences to our work are(Need further verify, I haven't carefully read the details yet): (1)It need labels to define the confusing classes. (2)Whether an example is confusing only matters in the training stage, it no longer cares if it's confusing in the inferencing stage.

[16] focus on building hash for images to make search and retrieve of the images efficient. And it pay attention to the easy and hard examples. However, it needs labels to tell which one is easy or hard. And it only focus on the easiness on training process.

[13] This paper mentioned a concept: Focal Loss(Not originated in this paper). It also focus on the uncertainty of the classifier's prediction. But simply using the difference between the predicted score and ground truth as the measure of uncertainty which is very different from ours.

I will look for more paper focusing on easy-confusing examples. Till now, I find most works that pay attention to the easiness and confusion of examples are only trying to make use of it in the training stage to improve the training process. After they get the trained framework, they no longer care about whether an example is easy or confusing on the inference stage, but to simply predict a label in the classical way. But our framework can distinguishing the hardness of the examples on the inference stage, and is able to tell the uncertainty quantitatively.

- [4]: Add dropout in the inference stage. This can generate an 'ensemble' without really training multiple predictors. However, this paper doesn't propose novel ways to make use of the ensemble outputs. Actually the mc-dropout method can be combined with ours.
- [11] It does three things: (1) Directly predict uncertainty as part of the predictor output(i.e, the predictor's output should be a normal distribution with mean value and uncertainty rather than just a single value) and properly add this uncertainty into the loss function. (2)Use adversarial training (3)Train an ensemble to estimate the predicted mean value and uncertainty. Difference to ours: (1)The way to make use of the ensemble output is different from us: treat the ensemble as a uniformly-weighted gaussian mixture. (2)Doesn't mention predicting a SET.
- [5] Constructs a method to find a proper threshold for confidence-rate(Can be almost any function) so that only the confident predictions will be made(Say don't know for unconfident). Difference to ours: (1)doesn't use ensemble(The author mentioned ensemble saying using ensemble costs too much). (2) Doesn't mention predicting a SET. Seems its way to choose threshold can be applied to our method. We can compare our results with this by regarding the Set size $\neq 1$ as 'don't know'.
- [7] Following the work in [5], this paper trys to find a good method to measure the confidence-rate. It also uses an ensemble, but the members in the ensemble are not trained independently but rather predictors obtained in different epochs in the training process.

- [6] Designed a way to train predictors that output uncertainty directly. Neither use ensemble nor mention prediction set. We can compare our results with this by regarding the Set size $\neq 1$ as 'don't know'.

3 Theory

Many learning algorithms are based on finding the single best rule. Ensemble methods such as Bagging and Random Forests are based on combining many rules, each of which is close to otimal. Averaging/voting across the rules results in a more stable and more accurate rule. We use a similar logic for classsification with one important twist: when the ratio between +1 and -1 predictions is close to 50%/50% our combination rule outputs 0, which can be interpreted as "I don't know" (IDK).

Figure 1 is a sketch of ensemble based classification. The central assumption is that models that are in the close the best model, in terms of the probability of disagreement, are almost as accurate as the best model. models. This implies that under slightly different training conditions that sub-optimal model can become the best. We call this set of models the "support set" and divide the instance space into the "easy" instances, on which all models in the support set agree. The "hard" instances are those that are not "easy". The "hardness" of hard examples is measured by the the difference between the probability [1] of the two parts to which the instance divides the sample.

An ensamble is a sample from the support set. Procedurally, it is generated by perturbing the training process. In this paper we consider three ways of perturbing the training process, a few more possibilities are listed in the conclusions.

1. **Bootstrap** perturb the training process by selecting from the size n training set a new training set of size n by sampling from the n times with replacement.
2. **Random starting point** Choose the starting point for gradient descent multiple times independently at random.
3. **Architecture** Choose a different architecture for each ensemble member.

We divide our discussion to the We use the binary classification problem as a building block of our solution for multi-label prediction.

3.1 Binary Case

In a standard binary classification task, given an input **x**, the predictor will output either $+1$ or -1. In our framework the model can predict 0 in addition to -1 and $+1$, where 0 stands for "hard instance" or "I don't know the label".

To quantify

e don't use the binary classification or the logit value generate from the score, instad, we use the score by a DNN.

Usually in a binary classification task, a neural networks predictor calculates a real value. In the training stage this value can be further transfered by a sigmoid funtion to a real value ranging from 0 to 1 and then pluged into the loss function; while in the inferencing stage, this value can be compared with 0 to decide whether this example belongs to the positive class or the negative class. Denote this value calculated by predictor $f(\cdot, \mathcal{P})$ as $O_f(\cdot, \mathcal{P})$. If $O_f(x, \mathcal{P})$ is larger than 0, the example will be classified into the positive class, otherwise the negative class.

Assumption As we change the pertubation $\mathcal{P}$ to get $O_f(x, \mathcal{P}_1)$, $O_f(x, \mathcal{P}_2)$,$\cdots$, $O_f(x, \mathcal{P}_i)$,$\cdots$,$O_f(x, \mathcal{P}_E)$, the values $O_f(x, \mathcal{P}_i)$ obeys a normal distribution $\mathcal{N}(\mu, \sigma^2)$.

The mean value μ should be positive if this example belongs to the positive class and negative if it belongs to the negative class. After training an ensemble $f(\cdot, \mathcal{P}_1), f(\cdot, \mathcal{P}_2), \cdots, f(\cdot, \mathcal{P}_i), \cdots, f(\cdot, \mathcal{P}_E)$, we can esitimate how confident we are to say μ is positive or negative based on $O_f(x, \mathcal{P}_1), O_f(x, \mathcal{P}_2), \cdots, O_f(x, \mathcal{P}_i), \cdots, O_f(x, \mathcal{P}_E)$. This can be done by *t-test*. The null hypothesis is $\mu = 0$, and the t-statistic is:

$$t = \frac{\overline{O}_f(x, \mathcal{P})}{S/\sqrt{E}} \tag{1}$$

[1] Assuming some fixed prior distribution.

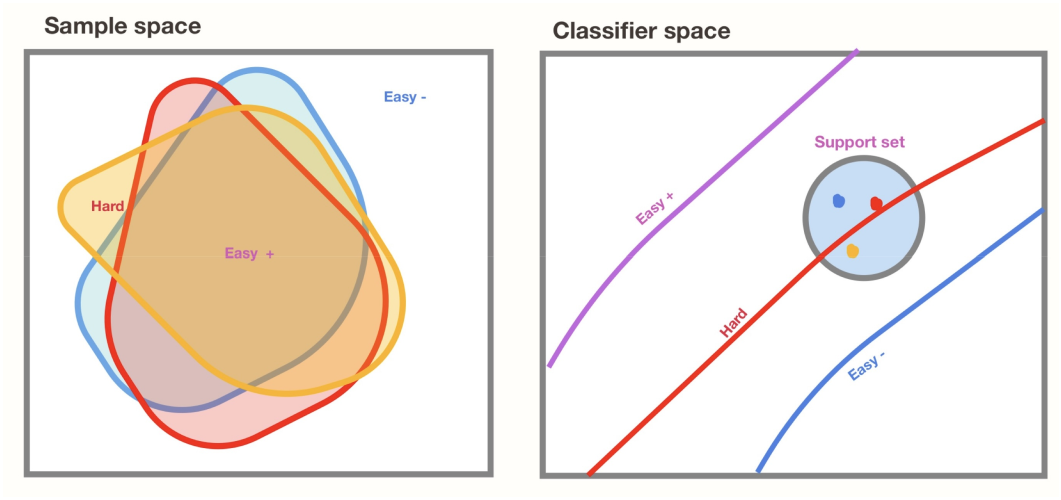


Figure 1: **Duality between classifiers and examples** Each point in the sample space figure corresponds to a sample, the three shaded sets correspponds to an ensemble of 3 classification rules with similar training error. “Easy -” is an instance on which the ensemble predicts unanimously “-”, while “Easy+” is an instance on which the unanimous prediction is “+”. It is this unanimity that makes the instance “easy”. The hard instance is one on which classifiers disagree. The classifier space (parameter space), demonstrates the same relationships between instances and classifiers, but here each classifier is a point and the instances define boundaries between classifiers that predict “+’ or “-” on the instance. THe *support set* is the set of classifiers whose performance is close to optimal. The three dots correspond to the ensemble of three classifiers. Finally, easy instances are ones on which the rules in the support set are unanimous, while the hard example splits the support set in two.

where

$$\overline{O}_f\left(x,\mathcal{P}\right) = \tfrac{1}{E}\sum_{i=1}^{E} O_f\left(x,\mathcal{P}_i\right) \tag{2}$$

$$S^2 = \tfrac{1}{E-1}\sum_{i=1}^{E}\left(O_f\left(x,\mathcal{P}_i\right) - \overline{O}_f\left(x,\mathcal{P}\right)\right)^2 \tag{3}$$

denote the p-value of the test as p, we define our confidence as:

$$confidence = -sign(\overline{O}_f\left(x,\mathcal{P}\right)) * log(p) \tag{4}$$

If the predictors are very confident that the example belongs to the positive class, the confidence should be a very large positive value; if the predictors are confident of the negtatice class, it should be a very small negative value. If the predictors are unsure, the confidence value would be close to 0. We can set a threshold to discriminate class+ and class-. Further, with this confidence value, we can set one threshold for confidence class+ and another thereshold for confident class-. In the regime between the two thresholds, the predictors will say "I don’t know".

3.2 Multiclass Case

In a multiclass classification task, $\mathbf{O}_f\left(x,\mathcal{P}\right)$ is no longer a real value but a R^C vector, C is the number of classes. In the training stage, a softmax can be operated on $\mathbf{O}_f$ and further used in the loss function. In the inferencing stage, $\max_j \mathbf{O}_{f,i}$ will be taken as the final prediction result, where $\mathbf{O}_{f,j}$ refers to the jth dimension of the $\mathbf{O}_f$ vector.

To apply our *t-test*-based method, we take each multiclass classification task as multiple binary classification tasks. Based on $\mathbf{O}_{f,j}\left(x,\mathcal{P}_1\right),\cdots,\mathbf{O}_{f,j}\left(x,\mathcal{P}_i\right),\cdots,\mathbf{O}_{f,j}\left(x,\mathcal{P}_E\right)$, we can calculate the t-statistic t_j and get the p-value p_j and finally the confidence value $confidence_j$ for class j.

We can have different ways to make use of the confidence values, for example:

1. The class with the largest confident value will be the predicted class.

2. Instead of predicting one class, the class with the largest confident value and classes whose confidence is no smaller than the largest by a certain amount will form a prediction set.

3. We can set a threshold as in the binary case. If $\mathit{confidence}_j$ is smaller than the threshold, then the example doesn't belong to class j, otherwise the predictors think the example belongs to class j. Similarly, we can set two thresholds for confident of class j, confident not class j and "I don't know".

The second and the third ways bring a problem: the example may belong to multiple classes or none of the classes. When this happens, it suggests that this example is a "hard" example, because it makes the predictors confused among these classes. The experiment section shows that this confusion indeed happens for human. Therefore, it can be benifit to allow the predictors to say that they are confused and further careful actions are needed to make a decision(For example, call stronger predictors to classify the example), rather than simply give a prediction without confidence.

Let $\mathcal{D}$ be a dataset containing $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$,where $\mathbf{x}$ is the oberserved feature vector, y is the label in a classification task or a numerical value in a regression task. For a predictor with a specific architecture f (For example, a ResNet),we can apply a pertubation $\mathcal{P}$ in the training process of f and get a predictor $f(\cdot, \mathcal{P})$. When a new observed feature vector $textbfx$ comes, the predictor gives a prediction $f(\mathbf{x}, \mathcal{P})$. Since there are many different kinds of predictors, we can also train a set of predictors with different architectures $f_1, f_2, \cdots, f_i, \cdots, f_K$

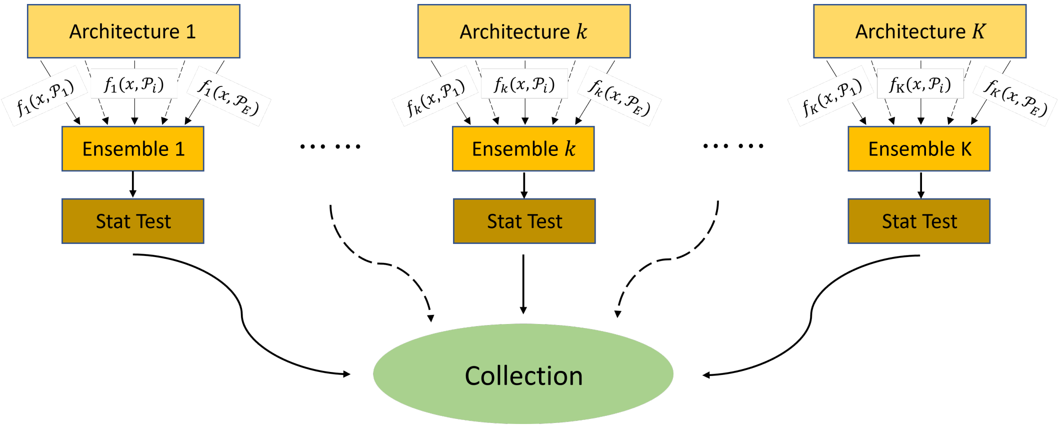


Figure 2: **Structure of Our Framework**

3.3 Three types of Pertubations: BootStrap, Architecture Starting Points and Architecture

There are two common ways to pertubate the training process:

Bootstrap We can construct a training set $\mathcal{T}$ by randomly draw M examples $\{(\mathbf{x}_i, y_i)\}_{i=1}^M$ from $\mathcal{D}$ with replacement. We can sample a set a $\mathcal{T}_1, \mathcal{T}_2, \cdots, \mathcal{T}_i, \cdots, \mathcal{T}_E$ independently, and train a f on each of the training set. This will generate a set of predictors $f(\cdot, \mathcal{T}_1), f(\cdot, \mathcal{T}_2), \cdots, f(\cdot, \mathcal{T}_i), \cdots, f(\cdot, \mathcal{T}_E)$ with the same architecture but different parameters.

Random Starting Point We take the entire dataset $\mathcal{D}$ as the training set, yet change the initialization of the paremeters in the architecture f(For example, use different random seeds to initialize the neural networks). With different starting points $\mathcal{S}_1, \mathcal{S}_2, \cdots, \mathcal{S}_i, \cdots, \mathcal{S}_E$ of the parameters, we can train a set of predictors $f(\cdot, \mathcal{S}_1), f(\cdot, \mathcal{S}_2), \cdots, f(\cdot, \mathcal{S}_i), \cdots, f(\cdot, \mathcal{S}_E)$

3.4 Why Ensemble Works

This section is the analysis of why the aggregation of the ensemble works better than a single predictor.

3.4.1 For Regression Tasks

In regression, a common aggregation method is to calculate the average of the ensemble outputs known as *Bagging*, which is:

$$f^{A}\left(x\right)=\frac{1}{E}\sum_{i=1}^{E}f\left(x,\mathcal{P}_{i}\right) \tag{5}$$

as $E\to\infty$, we have:

$$f^{A*}\left(x\right)=\mathcal{E}_{\mathcal{P}}\left[f\left(x,\mathcal{P}\right)\right] \tag{6}$$

The expectation of the squared error for a single model is:

$$\begin{aligned}\mathcal{E}_{\mathcal{P}}\left[\left(y-f\left(\mathbf{x},\mathcal{P}\right)\right)^{2}\right]&=y^{2}-2y\mathcal{E}_{\mathcal{P}}\left[f\left(x,\mathcal{P}\right)\right]+\mathcal{E}_{\mathcal{P}}\left[f^{2}\left(x,\mathcal{P}\right)\right]\\&=y^{2}-2y\mathcal{E}_{\mathcal{P}}\left[f\left(x,\mathcal{P}\right)\right]+\mathcal{E}_{\mathcal{P}}^{2}\left[f\left(x,\mathcal{P}\right)\right]+\mathcal{E}_{\mathcal{P}}\left[f^{2}\left(x,\mathcal{P}\right)\right]-\mathcal{E}_{\mathcal{P}}^{2}\left[f\left(x,\mathcal{P}\right)\right]\\&=\left(y-\mathcal{E}_{\mathcal{P}}\left[f\left(x,\mathcal{P}\right)\right]\right)^{2}+\mathcal{E}_{\mathcal{P}}\left[f^{2}\left(x,\mathcal{P}\right)\right]-\mathcal{E}_{\mathcal{P}}^{2}\left[f\left(x,\mathcal{P}\right)\right]\\&=\left(y-f^{A*}\left(x\right)\right)^{2}+Var\left(f\left(x,\mathcal{P}\right)\right)\end{aligned} \tag{7}$$

The equation 7 shows that the expectation of the squared error for a single predictor is larger than the squared error of the aggregation. And the difference is just the variance of the predictor outputs over the pertubation.

3.4.2 For Classification Tasks

Assume we have C classes in a classification task, for a given **x**, use $P(j|\mathbf{x})$ to denote the "Ground Truth" probability that **x** belongs to class j, we will further discuss what the "Ground Truth" means in later section, here let's just simply take it as the true probability. Given an input **x**, a single predictor will give a prediction among classes $1,2,\cdots,j,\cdots,C$. With an ensemble, we can measure how often the predictors will give the predictions, which is:

$$Q(j|\mathbf{x})=\frac{1}{E}\sum_{i=1}^{E}I(f(\mathbf{x},\mathcal{P}_{i})==j) \tag{8}$$

where $I(\cdot)$ is the indicator function. Again, as $E\to\infty$, we have:

$$Q^{*}(j|\mathbf{x})=\mathcal{E}_{\mathcal{P}}[I(f(\mathbf{x},\mathcal{P})==j)] \tag{9}$$

A common method to make use of the ensemble know as *Majority Vote* is picking $\operatorname{argmax}_{1\le j\le C}Q^{*}(j|\mathbf{x})$ as the final classification output.

The expectation of a single predictor being correct is:

$$\sum_{j=1}^{C}Q^{*}(j|\mathbf{x})P(j|\mathbf{x}) \tag{10}$$

However, as [2] indicates, if the ensemble of predictors is order-correct, which means:

$$\operatorname*{argmax}_{1\le j\le C}Q^{*}(j|\mathbf{x})=\operatorname*{argmax}_{1\le j\le C}P(j|\mathbf{x}) \tag{11}$$

Not necessarily to be accurate, the ensemble's expectation of being correct is $\max_{1\le j\le C}P(j|\mathbf{x})$, which is no worse than a single predictor. [2] gives an example where $P(1|\mathbf{x})=0.9$, $P(2|\mathbf{x})=0.1$ and $Q*(1|\mathbf{x})=0.6$, $Q*(2|\mathbf{x})=0.4$. The expectation of a single predictor being correct is 0.58, but for the ensemble, it's 0.9

4 Method

5 Experiment Result

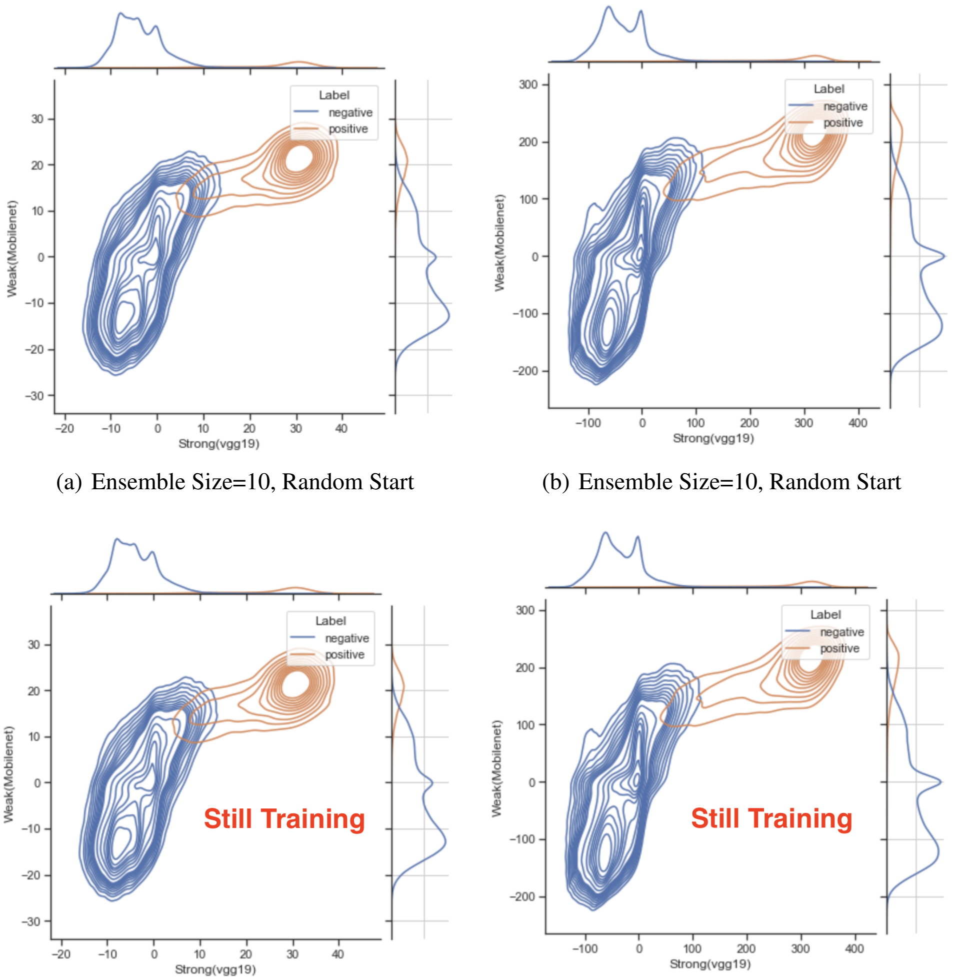
This section shows the experiment results on *CIFAR10*[10] and *CIFAR10H*[15]. *CIFAR10* includes 50000 image-label pairs for training and 10000 image-label pairs for testing. *CIFAR10H* includes extra information for each image which is the frequencies of each label to be chosen as the true label by a group of human labeler. The architectures we used include: ShuffleNet, ShuffleNetV2, MobileNetV2, Regnetx, MobileNet, Efficientnetb0, GoogleNet, Densenet121, Resnext29, ResNet18, SeNet18, SimpleDLA, VGG19, DPN92(Will add citition for these architectures). Table[**?**] shows the basic information of these architectures.

Table 1: **Basic Properties of the Architectures**: Parameter numbers refers to the number of trainable parameters in each architecture. Forward Time refers to the averaged forward propagation time for one CIFAR image. We tested these values on 3 different devices: Nvidia RTX 3080Ti, Nvidia RTX 3060 and 11th Gen Intel(R) Core(TM) i5-11400F CPU.The last two columns show the accuracy of single predictors trained with different pertubation methods. Bootstrap samples 30000 examples from the 50000 training data.

	Parameter	Forward Time(ms)			Single Predictor Accuracy	
	Numbers	RTX 3080Ti	RTX 3060	CPU	Bootstrap	Random Start
shufflenet	925,618	2.016	7.815	13.995	0.88802	0.895520
shufflenetv2	1,263,854	1.867	7.341	5.931	0.88535	0.890430
mobilenetv2	2,296,922	1.717	6.694	7.592	0.89370	0.889671
regnetx	2,321,946	2.115	7.400	8.260	0.93330	0.938220
mobilenet	3,217,226	0.777	2.981	3.424	0.87632	0.873603
efficientnetb0	3,599,686	2.474	10.068	10.929	0.88585	0.896610
googlenet	6,166,250	2.990	10.722	30.060	0.93784	0.947831
densenet121	6,956,298	4.905	18.762	22.279	0.93735	0.945434
resnext29	9,128,778	1.335	4.723	24.190	0.93952	0.948639
resnet18	11,173,962	0.930	3.375	7.130	0.93688	0.948173
senet18	11,260,354	1.417	5.282	8.088	0.93591	0.945290
simpledla	15,142,970	1.769	6.415	12.596	0.93045	0.943860
vgg19	20,040,522	0.855	3.154	6.660	0.91733	0.931955
dpn92	34,236,634	4.712	17.596	58.489	0.94122	0.948508

5.1 Compare the Strong and Weak Architectures

As explained in section 3.2, each multiclass example can be transformed to multiple binary case. In our settings, each image can be divided to 10 image-class pairs. Each pair will have a label $+$ if the image belongs to this class, or $-$ if it doesn't.To compare the difference of the confidence values by weak and strong architectures, we assign a $[S, W]$ coordinates for each image-class pair. The S value represents the confidence value for a strong architecture, while the W represents a week. In this way, the 10000 test examples will be 100000 points on the $S - W$ plane. Figure 3 shows the density distribution of these points with KDE plots.



(a) Ensemble Size=10, Random Start

(b) Ensemble Size=10, Random Start

(c) Ensemble Size=10, BootStrap

(d) Ensemble Size=100,BootStrap

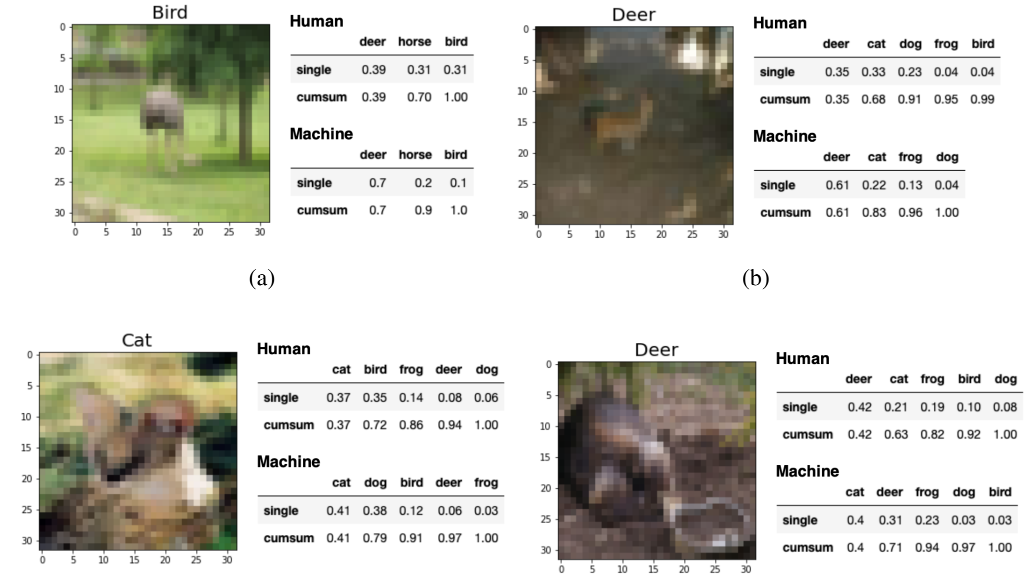
Figure 3: **KDE Plot of Confidence of Strong and Weak Architectures**: The strong architecture is Vgg19, the weak architecture is MobileNet

5.2 Compare Human Results with Machine Results

With our t-test based method, we can have different ways to combine the outputs of different architectures. Here we illustrate two ways and compare their results with human prediction.

5.2.1 Combination Way 1

Following the framework shown in 3.2, we use the 3rd way in section 3.2 and only set a single threshold for each architecture and allow prediction sets rather than only a single class as the output. Then for each image, each architecture will give a prediction set. We regard every class in the set as getting one vote from the architecture. Combine the sets by all the architectures, we get a collection. In this collection, each class among the 10 will get zero or one or multiple votes from the architectures. Based on the frequencies of each class's votes, we can measure how likely it is for each class to be chosen as a final prediction. With CIFAR10H, we can compare the results of human labeler and our machine predictors. Figure 4 illustrates some examples that human and machine have the same confusion.



(a)

Human

	deer	horse	bird
single	0.39	0.31	0.31
cumsum	0.39	0.70	1.00

Machine

	deer	horse	bird
single	0.7	0.2	0.1
cumsum	0.7	0.9	1.0

(b)

Human

	deer	cat	dog	frog	bird
single	0.35	0.33	0.23	0.04	0.04
cumsum	0.35	0.68	0.91	0.95	0.99

Machine

	deer	cat	frog	dog
single	0.61	0.22	0.13	0.04
cumsum	0.61	0.83	0.96	1.00

(c)

Human

	cat	bird	frog	deer	dog
single	0.37	0.35	0.14	0.08	0.06
cumsum	0.37	0.72	0.86	0.94	1.00

Machine

	cat	dog	bird	deer	frog
single	0.41	0.38	0.12	0.06	0.03
cumsum	0.41	0.79	0.91	0.97	1.00

(d)

Human

	deer	cat	frog	bird	dog
single	0.42	0.21	0.19	0.10	0.08
cumsum	0.42	0.63	0.82	0.92	1.00

Machine

	cat	deer	frog	dog	bird
single	0.4	0.31	0.23	0.03	0.03
cumsum	0.4	0.71	0.94	0.97	1.00

Figure 4: **Examples That Human and Machine have the Same Confusion**: The tables shows the frequencies of votes of human labeler and machine predictors. The rows with index 'single' are the frequencies for each single class, the unshown classes all have zero frequencies. The rows with index 'cumsum' are the culmulative sum of the above row.

For each image, we can use entropy of the frequency distribution by human/machine to measure how confusing it is for human/machine. We can use L1 distance between the frequency distributions by human and machine to measure their difference. We can denote each image as a point in the $[X, Y]$ plane where the X axis is the machine entropy, and the Y axis is the L1 distance between human and machine. A density plot is shown in Figure 5.2.1

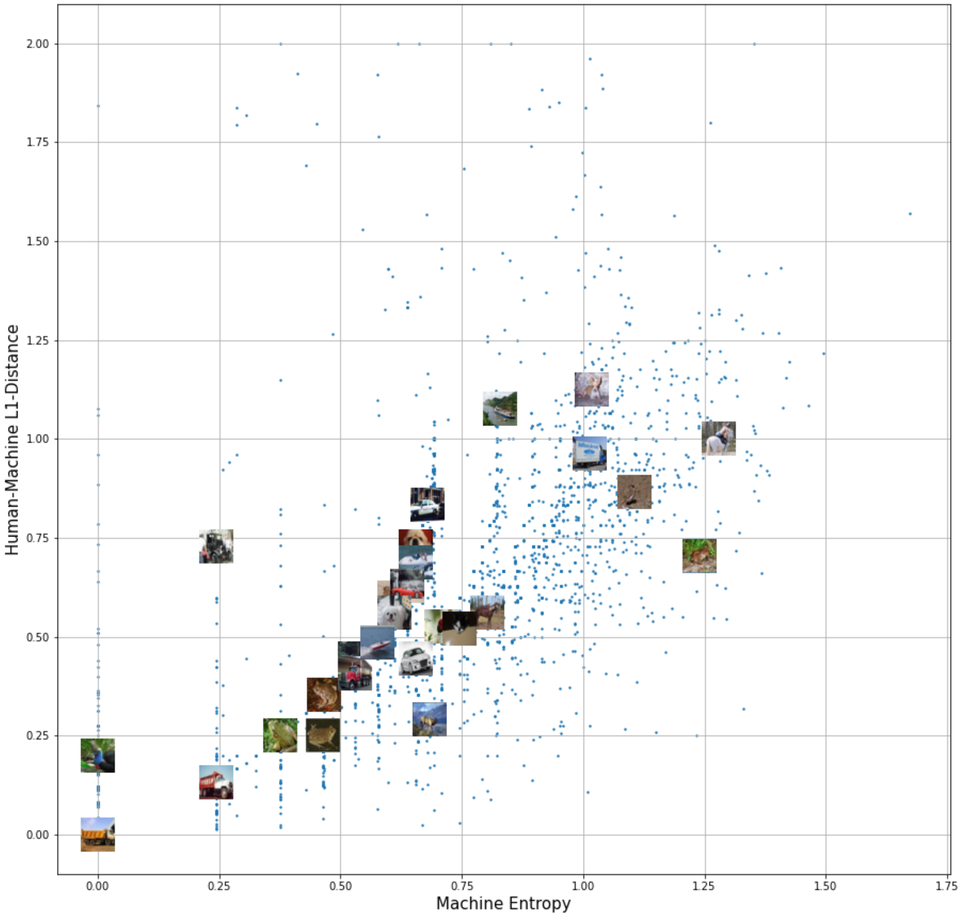


Figure 5: **Entropy-L1 Scatter Plot**

5.2.2 Combination Way 2

Using the 3rd way in section 3.2 and set two thresholds for each architecture to separate ’Confident Positive(+)’, ’I don’t know(0)’, and ’Confident Negative(-)’. For each test instance, each class gets K votes in $+, 0, -$ from the K architectures. If a class has more than T votes of $+$ and 0 vote of $-$, then we say this class belong to the prediction set of this instance.

We compare the results of this combination way with the human results in the following way: for a test instance, we have the machine generated prediction set and each class’s frequencies to be chosen by human. we first sum up the human frequencies of the classes in the prediction set to measure how much the machine results can match the human’s judgements. Then we penalty large set size by minus the set size divided by the number of classes C.

With this combination method. A prediction set whose size is 1 can get a score ranging from -0.1 to 0.9, size 2 can get -0.2 to 0.8 and size 3 from -0.3 to 0.7. Figure**??** and Figure7 shows the culmulative distribution function of the distribution of this score under different set sizes.

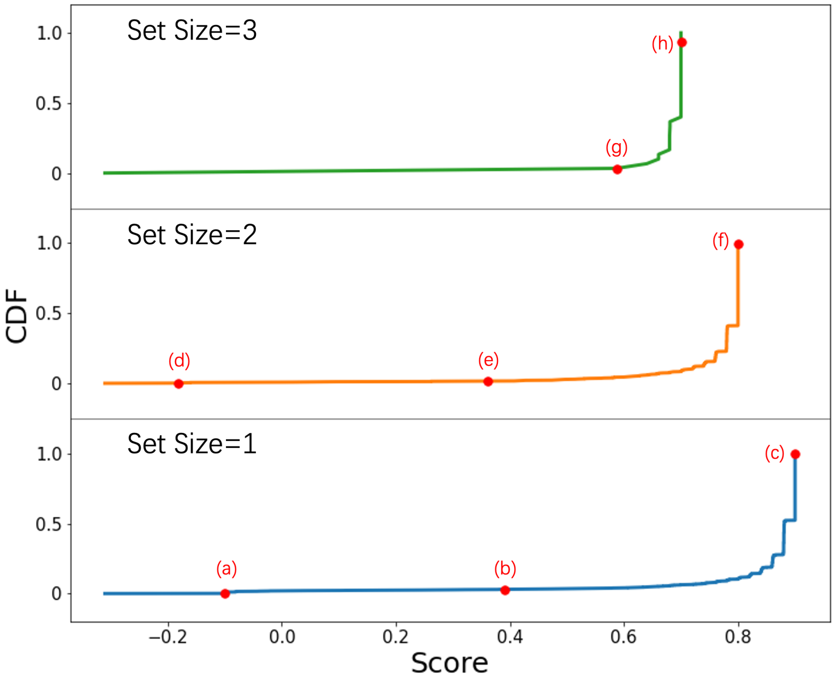
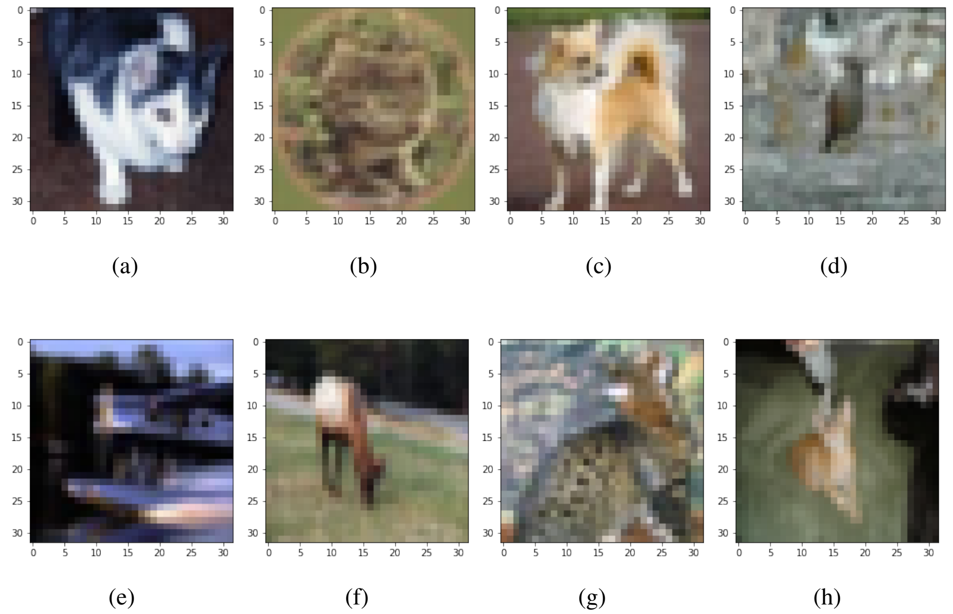


Figure 6: **CDF of The Score for Different Set Sizes** To choose the 2 thresholds to separate $+, 0, -$, we first get the culmulative distribution function(CDF) of positive binary instances and complementary culmulative distribution function(CCDF) of negative binary instances. Set the confidence score where negative CCDF value = 10% and the score where positive CDF value=90% as the 2 thresholds. $T = 1$, which is the least thresholds of $+$ votes.

Table 2: **Running Time of Weak-Strong Flow**: The weak model is MobileNet, the strong model is DPN92, the ensemble size for both architectures are 10. Batch Size indicates how how many images are processed parallelly by the weak-strong flow. The unit for all values is ms.

Batch Size	Weak Run	Weak T-Test	Strong Run	Strong T-Test	Weak-Strong Flow Total
16	0.54016	0.04063	3.38590	0.03598	4.00267
64	0.32618	0.01138	2.32941	0.00957	2.67653
256	0.42272	0.00420	2.83185	0.00308	3.26185
1024	0.33215	0.00329	2.68724	0.00153	3.02421



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 7: **Examples of Human and Machine Classification** (a)**Human**: Cat(1.0). **Machine**: Dog. (b)**Human**: Frog(0.49), Cat(0.47), Bird(0.04). **Machine**: Frog. (c)**Human**: Dog(1.0). **Machine**: Dog. (d)**Human**: Bird(0.92), Ship(0.02), Frog(0.02), Dog(0.02), Cat(0.02). **Machine**: Cat, Dear. (e)**Human**: Airplane(0.56), Ship(0.44). **Machine**: Truck, Airplane. (f)**Human**: Deer(0.6), Horse(0.4). **Machine**: Deer, Horse. (g)**Human**: Cat(0.72), Frog(0.09), Deer(0.09), Bird(0.08), Airplane(0.02). **Machine**: Cat, Frog, Bird. (h)**Human**: Cat(0.65), Dog(0.21), Deer(0.14). **Machine**: Dog, Deer, Cat

5.3 Example: Weak-Strong Flow

To see how the above finds can be applied, we establish a weak-strong flow where we consturct an ensemble with a weak yet fast architecture and another ensemble with a strong yet slow architecture. For each coming image, the weak ensemble labels $+$, $-$, or 0 for each class using the 3rd way in section 3.2 with two thresholds, where $+$ represents being confident of belonging to this class, $-$ for confident of not belonging to this class, or 0 for "I don't know" . If there is only one class with the label $+$, then this class will be the final result. Otherwise we call the strong the strong ensemble and make a prediction with the 1st way in section 3.2.

The accuracy of this weak-strong flow is 0000. To compare, the accuracy for purly running an easy ensemble with the 1st way in section 3.2 is 00000. The accuracy of running the strong ensemble only is 0000000

Table 2 and 3 show the running time of the weak-strong flow and only running the strong ensemble.

Table 3: **Running Time of the Strong Ensemble Only**: The strong model is DPN92, the ensemble size is 10. Batch Size indicates how how many images are processed parallelly. The unit for all values is ms

Batch Size	Strong Only Run	Strong Only T-Test	Strong Only Total
16	5.93767	0.03345	5.97112
64	4.93924	0.00943	4.94867
256	5.70755	0.00375	5.71130
1024	7.01784	0.00224	7.02008

References

[1] Leo Breiman. Fitting additive models to regression data: Diagnostics and alternative views. *Computational Statistics & Data Analysis*, 15(1):13–46, 1993.

[2] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[3] Armen Der Kiureghian and Ove Ditlevsen. Aleatory or epistemic? does it matter? *Structural safety*, 31(2):105–112, 2009.

[4] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[5] Yonatan Geifman and Ran El-Yaniv. Selective classification for deep neural networks. *Advances in neural information processing systems*, 30, 2017.

[6] Yonatan Geifman and Ran El-Yaniv. Selectivenet: A deep neural network with an integrated reject option. In *International conference on machine learning*, pages 2151–2159. PMLR, 2019.

[7] Yonatan Geifman, Guy Uziel, and Ran El-Yaniv. Bias-reduced uncertainty estimation for deep neural classifiers. *arXiv preprint arXiv:1805.08206*, 2018.

[8] Stephen C Hora. Aleatory and epistemic uncertainty in probability elicitation with an example from hazardous waste management. *Reliability Engineering & System Safety*, 54(2-3):217–223, 1996.

[9] Eyke Hüllermeier and Willem Waegeman. Aleatoric and epistemic uncertainty in machine learning: A tutorial introduction. *arXiv preprint arXiv:1910.09457*, 5, 2019.

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[11] Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. *Advances in neural information processing systems*, 30, 2017.

[12] Lihong Li, Michael L Littman, and Thomas J Walsh. Knows what it knows: a framework for self-aware learning. In *Proceedings of the 25th international conference on Machine learning*, pages 568–575, 2008.

[13] Chen Liu, Xiaomeng Dong, Michael Potter, Hsi-Ming Chang, and Ravi Soni. Adversarial focal loss: Asking your discriminator for hard examples. *arXiv preprint arXiv:2207.07739*, 2022.

[14] Weiwei Liu, Ivor W Tsang, and Klaus-Robert Müller. An easy-to-hard learning paradigm for multiple classes and multiple labels. *The Journal of Machine Learning Research*, 18, 2017.

[15] Joshua C Peterson, Ruairidh M Battleday, Thomas L Griffiths, and Olga Russakovsky. Human uncertainty makes classification more robust. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9617–9626, 2019.

[16] Cheng Yan, Guansong Pang, Xiao Bai, Chunhua Shen, Jun Zhou, and Edwin Hancock. Deep hashing by discriminating hard examples. In *Proceedings of the 27th ACM international conference on multimedia*, pages 1535–1542, 2019.