Calibration Tech Report

Mukund Sudarshan
Cornell University

Xiaoyan Wu
Cornell University

1 Introduction

Modern machine learning techniques achieve unprecedented accuracy on many tasks, even surpassing human performance on some. However, in practice accuracy often is not the only requirement for a machine learning model to be successful. In scenarios where predictions can have a high impact such as in military or medical domains, the error of such a model may still be high enough that always trusting the models predictions is too risky, and could even lead to loss of life. Alternatively, a machine learning algorithm may be used as part of an integrated system where other components depend on good confidence estimates, for example in visual localization (Kendall and Cipolla, 2015). Ideally, any machine learning algorithm used in these settings should be equipped with confidence estimates, that measure how likely the model is to be correct. This enables an algorithm to be trusted only when its confidence is high, and checked by a human in the loop when it is low.

In recent years, new techniques based on deep learning have transformed the field by achieving much better generalization performance on a wide variety of tasks, ranging from image recognition to natural language processing. Neural networks do output a probability distribution over the possible classes of each input, and historically these probabilities were well-calibrated. We will demonstrate, however, that as neural network models have become deeper, these distributions have become less representative of the actually correctness probabilities of the samples.Neural networks are extremely overconfident in their predictions, which could introduce signficiant amounts of error into realworld systems.

This gives rise to a dichotomy, where practitioners must choose between models with well-calibrated con- fidence estimates or better accuracy. The models based on Gaussian processes and random forests, for example, are more aware of being wrong, but at least in settings where deep learning works well are likely to be wrong more often. On the other hand, deep neural networks are much more accurate, but are not able to identify when predictions may be wrong. In order for deep learning systems to work reliably in real-world settings, the probability measures must be calibrated to the accuracy of the network.

2 Motivation

Measuring Overconfidence: A natural definition of model confidence is that a model predicting class c on a set of points with probability α should get, in expectation, $100\alpha\%$ of these points correct. Intuitively, a model that outputs a probability of 60% should have a roughly 60% chance of being correct. More formally, suppose X_α is the set of all possible test points that the model makes predictions for at probability level α. Let $\delta(\mathbf{x}^*)$ denote whether the model correctly classifies a test point $\mathbf{x}^*$. Then, the accuracy of a well-calibrated model on test points drawn from X_α should be α in expectation:

$$E[\delta(\mathbf{x}^*)]_{x\sim X_\alpha} = \alpha \tag{1}$$

$$\alpha - E[\delta(\mathbf{x}^*)]_{x\sim X_\alpha} = 0 \tag{2}$$

Note that the above should hold for a calibrated model at all values of α. Furthermore, since the sets X_α are all disjoint, we can sum over all test points rather the individual α sets. For any $\mathbf{x}^*$, denote by $p(y^*|x^*)$ the probability output by the model. That is, if $x^* \in X_\alpha$, then $p(y^*|x^*) = \alpha$. Let P_D be the unknown data distribution. The above facts imply that the following should hold for a well calibrated model:

$$E[p(y^*|x^*) - \delta(\mathbf{x}^*)]_{x\sim X_\alpha} = 0 \tag{3}$$

While this expectation cannot be computed in practice, the weak law of large numbers we can estimate it empirically on a large test set D^*:

$$OC(D^*;M) = \frac{1}{|D^*|} \sum_{\mathbf{x}^* \in D^*} p(y^*|x^*) - \delta(\mathbf{x}^*) \quad (4)$$

$$= \frac{1}{|D^*|} \sum_{\mathbf{x}^* \in D^*} p(y^*|x^*) - \frac{1}{|D^*|} \sum_{\mathbf{x}^* \in D^*} \delta(\mathbf{x}^*) \quad (5)$$

If we distribute the sum, we note that this is just the average prediction confidence on D^*: $\bar{c}(D^*)$ minus the accuracy on D^*: $\bar{a}(D^*)$:

$$OC(D^*;M) = \bar{c}(D^*) - \bar{a}(D^*) \quad (6)$$

We call the above function the *overconfidence* of a model M on a test dataset D^*. Intuitively, the above derivation implies that if a model with average confidence $\bar{c}(D^*)$ were well calibrated, it should obtain accuracy $\bar{a}(D^*)$ as well. Thus, positive values of overconfidence imply that, on average, the model is predicting probabilities that overstate the true correctness probability. Similarly, negative values of overconfidence correspond to *underconfidence* the model produces probabilities that underestimate the true correctness probabilities.

Single networks are overconfident: In standard classification problems, the final layer of a neural network is a softmax that outputs a probability distribution over the possible classes:

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad (7)$$

It is tempting to use these numerical outputs in (0,1) as confidence estimates for example, interpreting an output of 0.8 in some class as a confidence of 80% in that class. Other papers have observed that modern deep archi- tectures give rise to poor confidence estimates Gal and Ghahramani (2016). While historically these probabilities have been well calibrated [JRG: cite Caruana], in Figure 1 [insert figures 1 and 2 from old paper] we demonstrate that, as network depth increases, so too does overconfidence. In particular, one artifact of training with the negative log-likelihood loss on the ground truth labels is that these probabilities can easily be saturated by a high capacity model. By 110 layers, the majority of test points are predicted with 99% confidence or above, despite the error on these points being much higher than 1%. Indeed, the average confidence of the 110 layer network is nearly 90%, substantially larger than the accuracy of the model at 75.6%. Using these outputs as measurements of probabilities would give a false sense of accuracy for an error-prone model.

3 Methods

3.1 Temperature Scaling

The general idea behind temperature scaling is to divide each input to the softmax layer by a particular constant before computing the softmax scores. [GP: Cite "Distilling the Knowledge in a Neural Network". Also read it for more context on the temperature transformation.] Shown below is this simple transformation:

$$\sigma(z)_j = \frac{e^{z_j/T}}{\sum_{k=1}^{K} e^{z_k/T}} \quad (8)$$

Diving each input by a constant factor T decreases the magnitude of larger logits by a greater amount than smaller ones. [GP: Sort of. Again, read "Distilling" to see how they discuss the temperature trick. They describe it as "softening" the logits.] This means that if the neural network is extremely certain about an input belonging to a certain class, it will become less sure. Note that as $T \to \infty$, the probability of choosing any class is equally likely, and when $T \to 0$, one class will be chosen with probability 1, while the others will never be chosen. [GP: This is not super necessary, and might be potentially confusing.]

3.2 Platt Scaling

Platt scaling is yet another technique to transform the outputs of a classification model into class probabilities. [GP: Cite the paper.] It works by fitting a logistic regression model to a classifier's scores. If a data point x is predicted to be a member of class $f(x) = c$, then the probability estimates produced by Platt scaling can be given by:

$$P(y=1|x) = \frac{1}{1+\exp(af(x)+b)} \quad (9)$$

where a and b are parameters learned by the logistic regression model. Note that this formula only applies to the binary case. In the multi-class case, we simply run $k-1$ one-vs-all classifiers, where k is the total number of classes [GP: Is this true? Have others extended this to a multiclass setting?]

3.3 Isotonic Regression

Isotonic Regression is a powerful calibration method that can correct any monotonic distortion. [GP: Cite. Also, highlight differences between this and Platt scaling.] The only restriction is that the mapping function be isotonic (monotonically increasing). That is, given the predictions f_i from a model and the true targets y_i, the basic assumption in Isotonic Regression is that:

$$y_i = m(f_i) + \varepsilon_i \tag{10}$$

where m is an isotonic (monotonically increasing) function. Then, given a train set (f_i, y_i), the Isotonic Regression problem is finding the isotonic function $\hat{m}$ such that

$$m = \arg\min_z \sum (y_i - z(f_i))^2 \tag{11}$$

Similar to Platt Scaling, Isotonic Regression only works in binary cases, so we need to run $k-1$ one-vs-all classifiers, where k is the total number of classes [GP: Again, citation needed for this. Are there other tricks for this in a multi-class setting?]

4 Datasets

4.1 Cifar 10 and Cifar 100

For these experiments, we used CIFAR-10 and CIFAR-100, two standard image classification datasets. They each contain 60,000 images. CIFAR-10 has 10 classes with 6,000 images in each class while CIFAR-100 has 100 classes with 600 images in each class.

[GP: For CIFAR10 Platt Scaling - I'm a little worried that there are no points with confidence < 0.5. Similarly, I'm worried for CIFAR100 Platt Scaling that there are no points with confidence > 0.8. Might be a bug?]

4.2 Google Streetview

5 Results

6 Conclusions

7 Figures

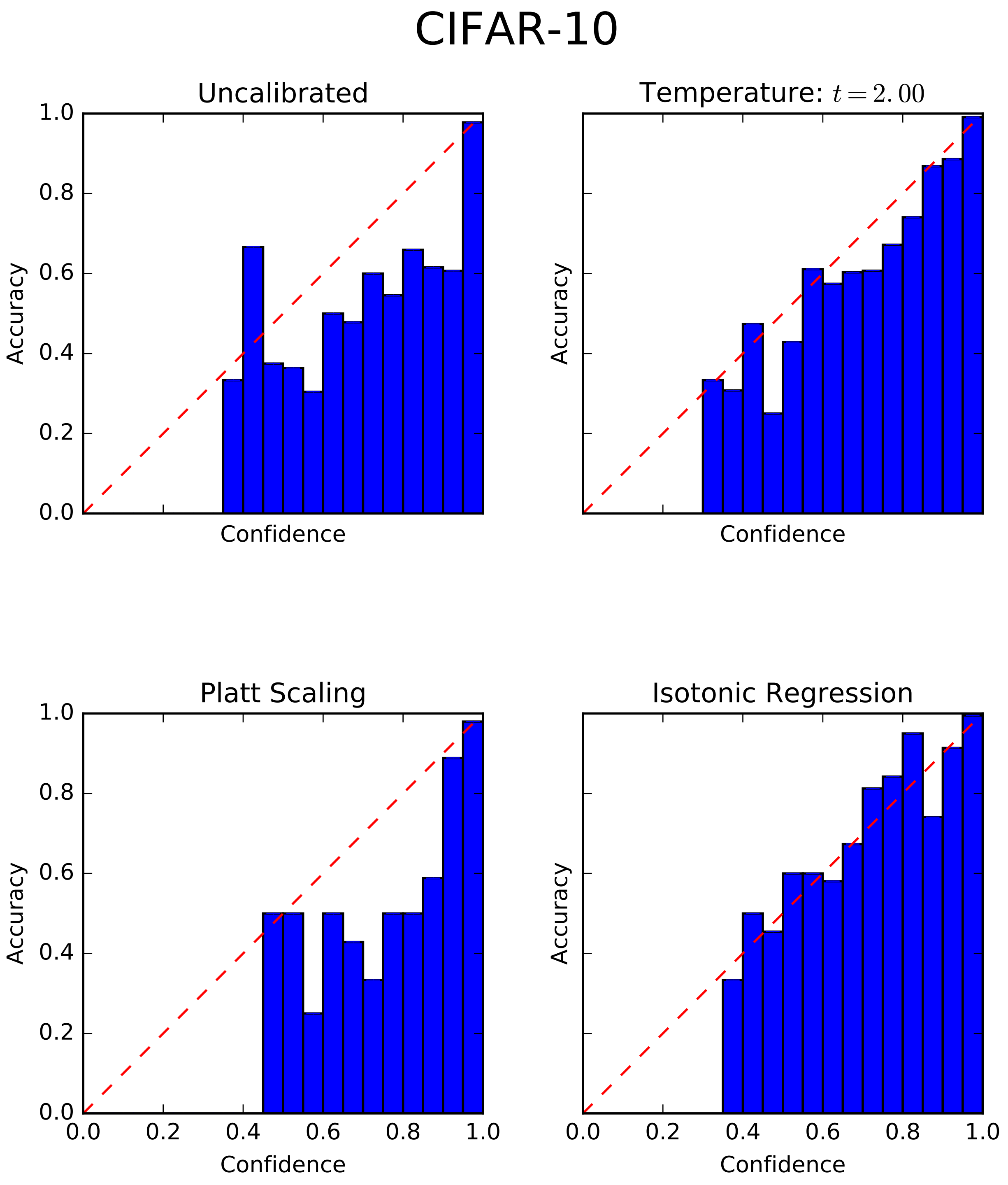

Figure 1: CIFAR-10

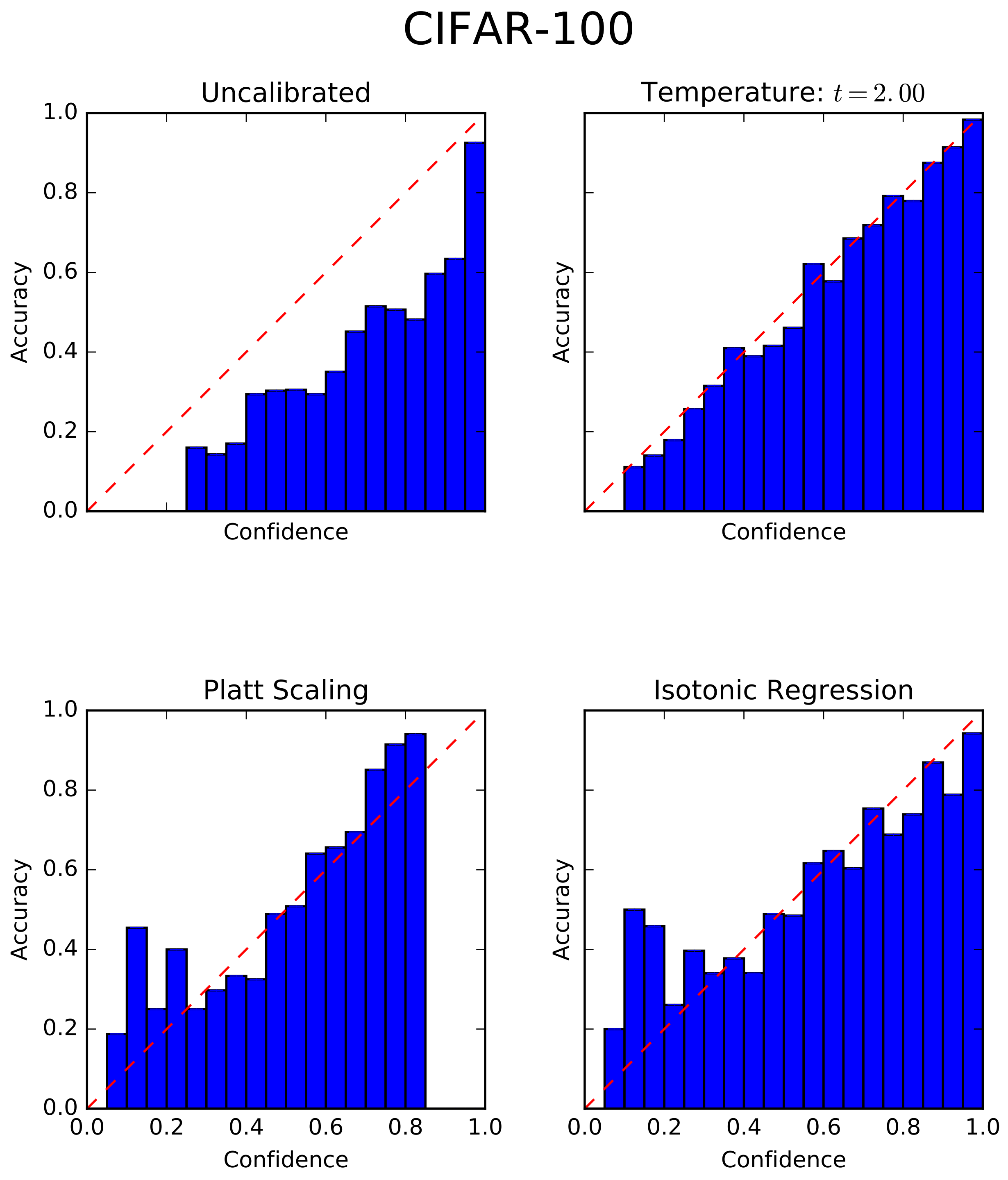

Figure 2: CIFAR-10

References