CS725: Foundations of Machine Learning

Project Report: Reproducing Calibration Results for Deep Neural Networks (Guo et al., 2017)

Team Members:
25M0742 Madhur Thakkar
25M0744 Monil Manish Desai
25M0770 Anurag Tripathi
25M0745 Aditya Gaikwad

November 25, 2025

Abstract

This project aims to reproduce the findings of the paper *"On Calibration of Modern Neural Networks"* by Guo et al. (2017). We investigate the phenomenon where modern deep neural networks, despite achieving high classification accuracy, tend to be overconfident in their predictions, i.e the average confidence comes out to be higher than average accuracy. We implement and compare four calibration methods—Histogram Binning, Isotonic Regression, Vector Scaling, and Temperature Scaling across multiple datasets such as CIFAR-10, CIFAR-100, Stanford Cars, and Birds using various Neural Networks such as ResNet-56, ResNet-164, DenseNet-190, WideResNet-28-10, MobileNetV2 and InceptionV3. We even extend the analysis by examining the effects of Label Smoothing as a calibration technique which is pre-processing compared to the methods in the paper which are post-processing. Our results confirm that Temperature Scaling is the most effective while being an efficient method for calibrating modern neural networks.

Contents

Chapter 1

Introduction

1.1 Problem Statement

In real-world applications of machine learning, such as autonomous driving and medical diagnosis, a model needs to be not only accurate but also calibrated. A model is *calibrated* if its predicted probability confidence aligns with its empirical accuracy. For example, if a model predicts a set of images with 80% confidence, 80% of those predictions should effectively be correct.

Guo et al. (2017) identified a troubling trend: while modern neural networks (like ResNets) have become deeper and more accurate than older models (like LeNet), they have simultaneously become significantly miscalibrated.

1.2 Understanding the Paper & Miscalibration

The paper highlights that a single factor is not the sole contributor but a combination of architectural choices common in modern deep learning contribute to the miscalibration of the models:

- **Depth and Width:** Deeper and wider networks have higher capacity, allowing them to minimize Training Negative Log Likelihood (NLL) which penalizes lower confidences aggressively, often pushing probabilities towards 1 even when test error stops improving leading to overconfidence.
- **Normalization:** Techniques like Batch Normalization improves the speed of convergence but can disrupt the scale of logits.
- **Weight Decay:** (Rarely used in modern neural networks) A lack of heavy regularization allows models to fit the training data distribution too closely.

The authors propose **Temperature Scaling**, a simple post-hoc method, as a universal solution to fix this issue without affecting the model's accuracy.

1.3 Github Link

The codebase can be found on this link: `https://github.com/monil2003/FML_Project`

Chapter 2

Methodology

2.1 Datasets and Models

To ensure our replication covers the variety of circumstances (depth, width, and complexity) described in the paper, we utilized the following diverse Model-Dataset pairs:

Dataset	**Model**	**Type**	**Characteristics**
CIFAR-100	ResNet-164	Deep	High depth, residual connections
CIFAR-100	DenseNet-190	Deep	Dense connections, high capacity
CIFAR-100	WideResNet-28-10	Wide	Increased channel width
CIFAR-100	ResNet-56	Standard	Moderate depth
CIFAR-10	ResNet-164	Deep	High accuracy baseline
CIFAR-10	ResNet-56	Standard	Standard benchmark
Stanford Cars	MobileNetV2	Fine-grained	Lightweight architecture
CUB-200 (Birds)	InceptionV3	Fine-grained	Complex topology, pre-trained

Table 2.1: Summary of Models and Datasets used for Calibration

2.2 Calibration Metrics and Methods

The metric that we used for evaluation is the **Expected Calibration Error (ECE)**. ECE partitions predictions into M equally spaced bins and calculates the weighted average of the difference between accuracy and confidence in each bin:

$$ECE = \sum_{m=1}^{M} \frac{|B_m|}{n} |acc(B_m) - conf(B_m)| \tag{2.1}$$

We implemented and compared four post-hoc calibration methods:

2.2.1 Histogram Binning (Non-Parametric)

This method divides the output confidence probabilities into bins and assigns a calibrated probability θ_i to each bin based on the average accuracy of samples within that bin.
Limitations: It requires choosing the number of bins and changes the predicted probabilities for all classes, which can hamper classification accuracy.

2.2.2 Isotonic Regression (Non-Parametric)

This method fits a non-decreasing function f to transform uncalibrated outputs. It optimizes (minimizes) the square error $\sum(f(p_i) - y_i)^2$.
Limitations: Like histogram binning, it is strictly monotonic but can be prone to overfitting on small validation sets.

2.2.3 Vector Scaling (Parametric)

A generalization of Platt Scaling where a linear transformation is applied to the logits vector **z** before the softmax:

$$\hat{q} = \sigma(\mathbf{W}\mathbf{z} + \mathbf{b}) \tag{2.2}$$

where **W** is a diagonal matrix.
Limitations: As shown in our results, the increased number of parameters often leads to overfitting due to increase in complexity, degrading calibration on the test set.

2.2.4 Temperature Scaling (Proposed Method)

This is a simplified version of Platt Scaling using a single scalar parameter $T > 0$ for all classes:

$$\hat{q}_i = \max \sigma(\mathbf{z}_i / T) \tag{2.3}$$

T is optimized to minimize NLL on a held-out validation set.
Advantages: It preserves the rank order of logits (highest logit is still highest), meaning **the model's accuracy remains unchanged**. It effectively "smoothens" the softmax distribution, countering overconfidence.

2.3 Implementation Details

Our code workflow followed these steps for each method:

1. **Data Preparation:** Split the test sets into a Validation Set (for learning parameters like T or bins) and a Test Set (for final ECE evaluation).
2. **Optimization:** We used grid search (for TS) and in-built libraries (for Vector/Isotonic) to minimize NLL or MSE on the validation split.
3. **Evaluation:** Computed ECE before and after calibration on the unseen Test Set.

2.4 Code Explanations

2.4.1 Histogram Binning Implementation

The implementation consists of two main phases: **fitting** and **calibration**.

- **Bin Definition:** We divide the probability space $[0, 1]$ into M fixed-width intervals using `np.linspace`.
- **Fitting:** We use `np.digitize` to assign samples to bins. For each bin, we calculate θ_m as the mean of the binary ground truth labels.
- **Calibration:** During inference, the raw probability is replaced by the stored accuracy value θ_m corresponding to its bin index.

2.4.2 Isotonic Regression

Isotonic Regression is implemented using the sklearn.isotonic library. The process involves three main steps: initialization, data preparation, and transformation.

Initialization

Isotonic Regression fits a monotonic function to the validation data. To handle test inputs outside the observed range, we set out_of_bounds='clip', ensuring values are clipped to the nearest boundary.

```
ir = IsotonicRegression(out_of_bounds='clip')
```

Data Preparation

Using flatten() to convert 2D outputs to 1D.

```
ir.fit(probs.flatten(), ground_truth.flatten())
```

Calibration

Each probability is mapped to its calibrated value on the monotonic curve using **transform**.

```
calibrated_probs = ir.transform(test_probs.flatten())
```

2.4.3 Temperature Scaling Implementation Details

Temperature Scaling is implemented as a PyTorch model wrapper (nn.Module) and requires an optimization loop. The process involves three components: parameter initialization, logic modification, and optimization.

Parameter Definition and Logic

A single scalar temperature is introduced for gradient updates (commonly initialized to 1.0 or 1.5). The forward pass divides logits by this scalar before applying softmax.

```
self.temperature = nn.Parameter(torch.ones(1) * 1.5)
def forward(self, input):
    return input / self.temperature
```

Optimization Routine

The optimal temperature is learned on the validation set using optim.LBFGS, as it is efficient on convex, low-dimensional problems. A closure function recomputes the loss (NLL) at each step.

```
optimizer = optim.LBFGS([self.temperature], lr=0.01)
def closure():
    optimizer.zero_grad()
    loss = nll_criterion(self.forward(logits), labels)
    loss.backward()
    return loss
optimizer.step(closure)
```

Efficiency

This method adds negligible inference overhead ($O(1)$ per class) and trains quickly, since only one parameter is optimized without backpropagation through the full network.

2.4.4 Vector Scaling Implementation Details

Vector Scaling generalizes Temperature Scaling by using Linear regression technique for calibration, it introduces two learnable vectors: weights ($\mathbf{W}$) and biases ($\mathbf{b}$), each of size K (number of classes).

Parameter Definition

In PyTorch, these are treated as parameters. Weights are initialized to ones and biases to zeros, starting from a neutral state.

```
self.weights = nn.Parameter(torch.ones(num_classes))
self.bias = nn.Parameter(torch.zeros(num_classes))
```

Forward Pass

Each class logit is calibrated independently using

$$\hat{q} = \mathbf{W}\mathbf{z} + \mathbf{b} \tag{2.4}$$

```
def forward(self, logits):
    return logits * self.weights + self.bias
```

Optimization

Training uses `optim.LBFGS` to minimize NLL, similar to Temperature Scaling. However, complexity grows with $2K$ parameters (e.g., 200 for CIFAR-100), making the method prone to overfitting on small validation sets.

```
optimizer = optim.LBFGS([self.weights, self.bias], lr=0.01)
```

2.5 Extra Work: Label Smoothing

In addition to the post-processing methods analyzed by Guo et al., we implemented **Label Smoothing** as an intrinsic regularization technique during training. This method was not part of the original calibration study but is relevant to modern calibration analysis.

Standard Cross-Entropy (CE) loss forces the model to predict the one-hot target distribution, which encourages the logits of the correct class to diverge to infinity, leading to overconfidence. We modified the training loss to use soft targets. For a ground truth label y, the smoothed target distribution y^{LS} is defined as:

$$y_k^{\text{LS}} = \begin{cases} 1-\alpha & \text{if } k = y \\ \frac{\alpha}{K-1} & \text{if } k \neq y \end{cases} \tag{2.5}$$

where α is the smoothing parameter and K is the number of classes. The loss is then computed as the Kullback-Leibler (KL) divergence between the predicted log-probabilities and this smoothed distribution:

$$\mathcal{L}_{LS} = D_{KL}(y^{\text{LS}}||\sigma_{\text{SM}}(\mathbf{z})) \tag{2.6}$$

By preventing the model from assigning full probability mass to a single class, label smoothing acts as a regularizer that can implicitly improve calibration (or at least reduce overconfidence) before any post-processing is applied. We hypothesized that retraining a model with LS might prevent miscalibration. We fine-tuned ResNet-56 on CIFAR-10 using Label Smoothing ($\alpha =$ 0.1).

Chapter 3

Results

3.1 Calibration Performance

The consolidated results of our experiments are presented in Table 3.1.

Dataset	Model	Uncalibrated ECE	Temp. Scaling (TS)	Vector Scaling	Hist. Binning	Isotonic Reg.
CIFAR-100	ResNet-164	15.07%	**1.36%**	1.99%	1.33%	1.90%
CIFAR-100	ResNet-56	15.87%	**2.68%**	3.45%	2.03%	1.94%
CIFAR-100	DenseNet-190	7.34%	**2.88%**	3.45%	1.01%	1.61%
CIFAR-100	WideResNet	6.45%	**3.53%**	4.27%	1.53%	1.32%
CIFAR-10	ResNet-164	3.75%	**0.95%**	1.18%	0.79%	0.95%
CIFAR-10	ResNet-56	3.96%	**1.21%**	1.63%	0.358%	0.84%
Stanford Cars	MobileNetV2	9.42%	**2.20%**	2.94%	1.26%	1.52%
Birds 400	InceptionV3	0.50%	0.85%	0.50%	0.354%	0.82%

Table 3.1: Comparison of ECE (%) across all models and calibration methods.

3.2 Visualizations

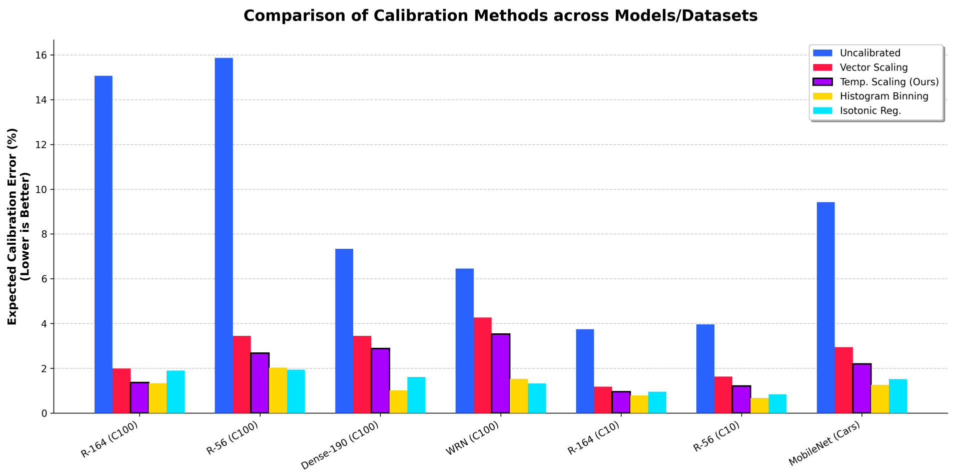


Figure 3.1: Bar chart comparing the ECE reduction across different methods. Temperature Scaling (Blue) consistently provides excellent reduction compared to the uncalibrated baseline (Gray) and outperforms Vector Scaling (Orange).

3.3 Reliability Diagrams

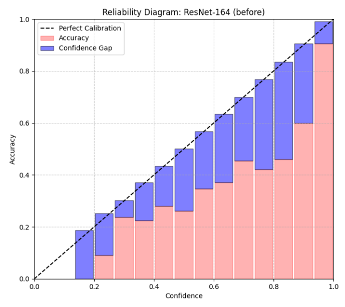
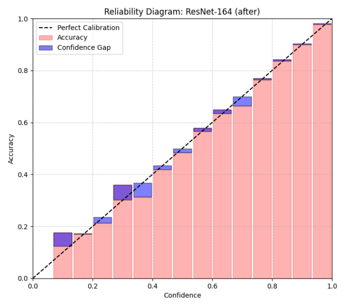
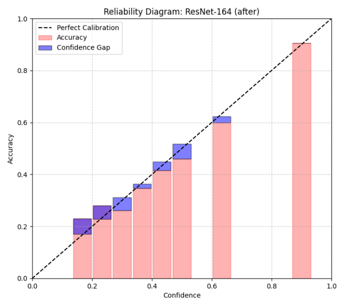


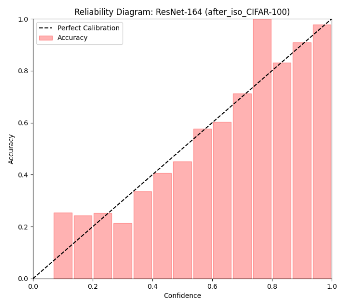
Figure 3.2: ResNet-164 on CIFAR-100 (uncalibrated).



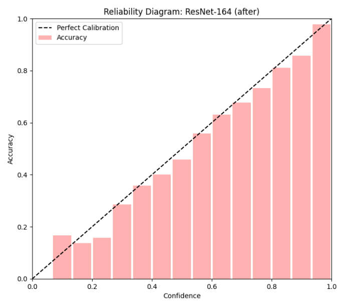
(a) Temp. Scaling



(b) Histogram Binning



(c) Isotonic Regression



(d) Vector Scaling

Figure 3.3: ResNet-164 on CIFAR-100 (post-hoc methods).

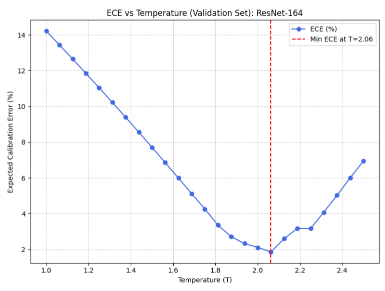


Figure 3.4: ResNet-164 on CIFAR-100: ECE vs Temperature.

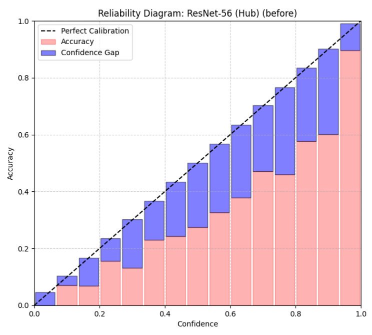
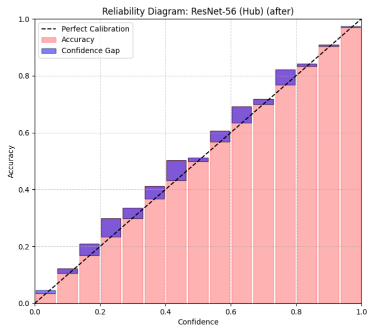
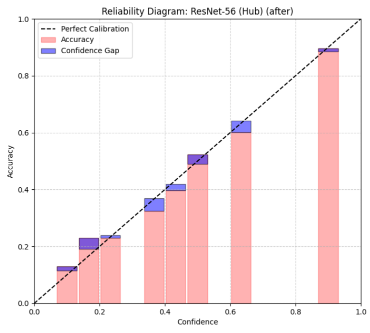


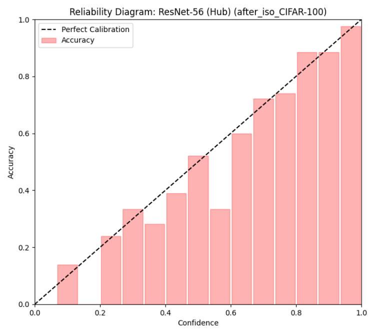
Figure 3.5: ResNet-56 on CIFAR-100 (uncalibrated).



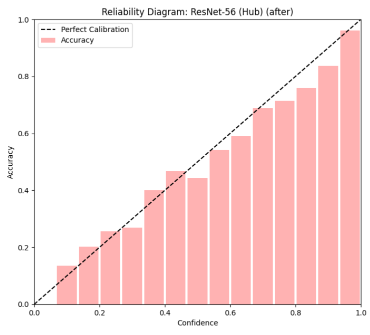
(a) Temp. Scaling



(b) Histogram Binning



(c) Isotonic Regression



(d) Vector Scaling

Figure 3.6: ResNet-56 on CIFAR-100 (post-hoc methods).

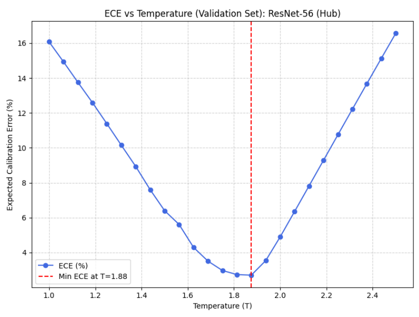


Figure 3.7: ResNet-56 on CIFAR-100: ECE vs Temperature.

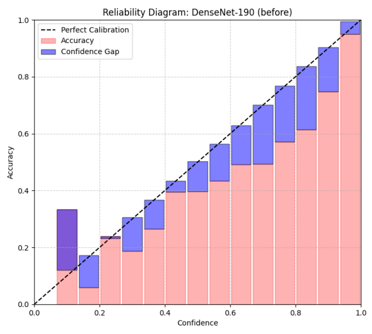
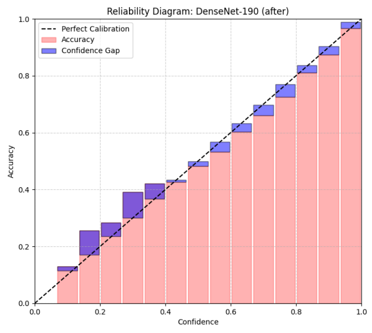
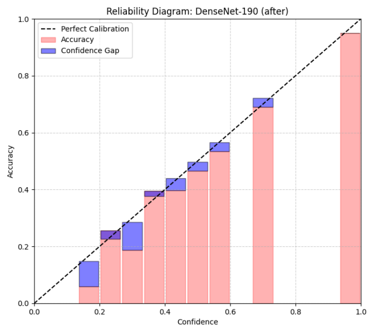


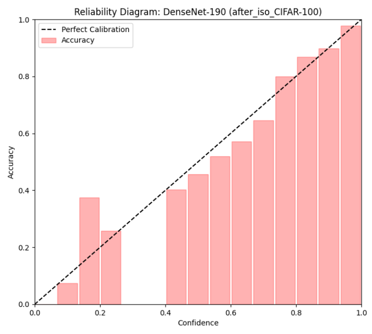
Figure 3.8: DenseNet-190 on CIFAR-100 (uncalibrated).



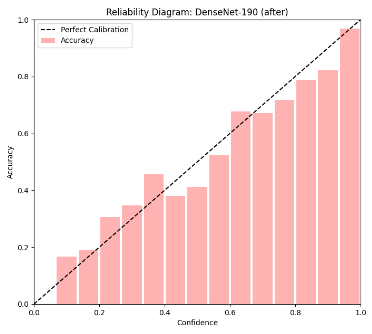
(a) Temp. Scaling



(b) Histogram Binning



(c) Isotonic Regression



(d) Vector Scaling

Figure 3.9: DenseNet-190 on CIFAR-100 (post-hoc methods).

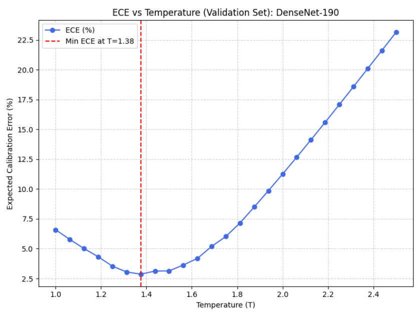


Figure 3.10: DenseNet-190 on CIFAR-100: ECE vs Temperature.

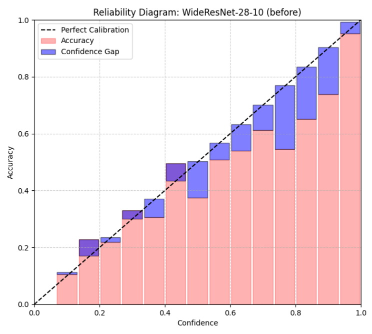
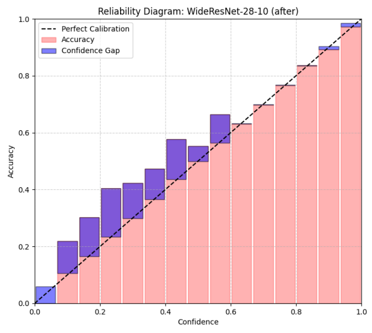
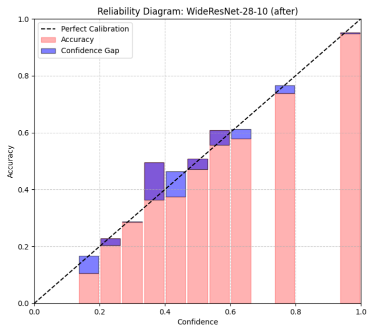


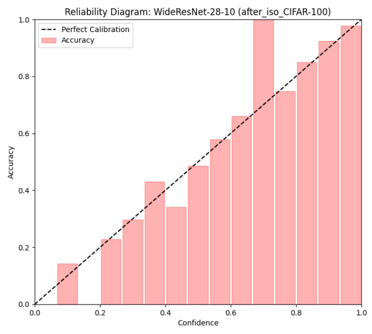
Figure 3.11: WideResNet-28-10 on CIFAR-100 (uncalibrated).



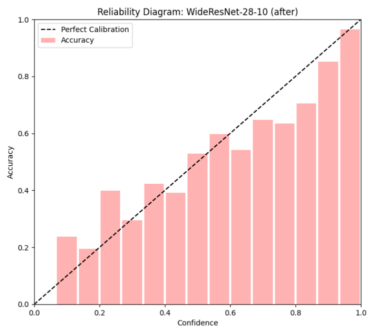
(a) Temp. Scaling



(b) Histogram Binning



(c) Isotonic Regression



(d) Vector Scaling

Figure 3.12: WideResNet-28-10 on CIFAR-100 (post-hoc methods).

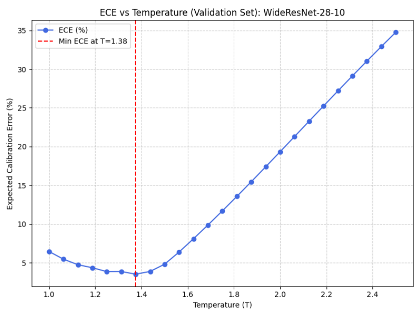


Figure 3.13: WideResNet-28-10 on CIFAR-100: ECE vs Temperature.

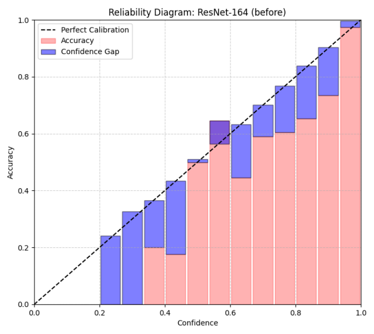
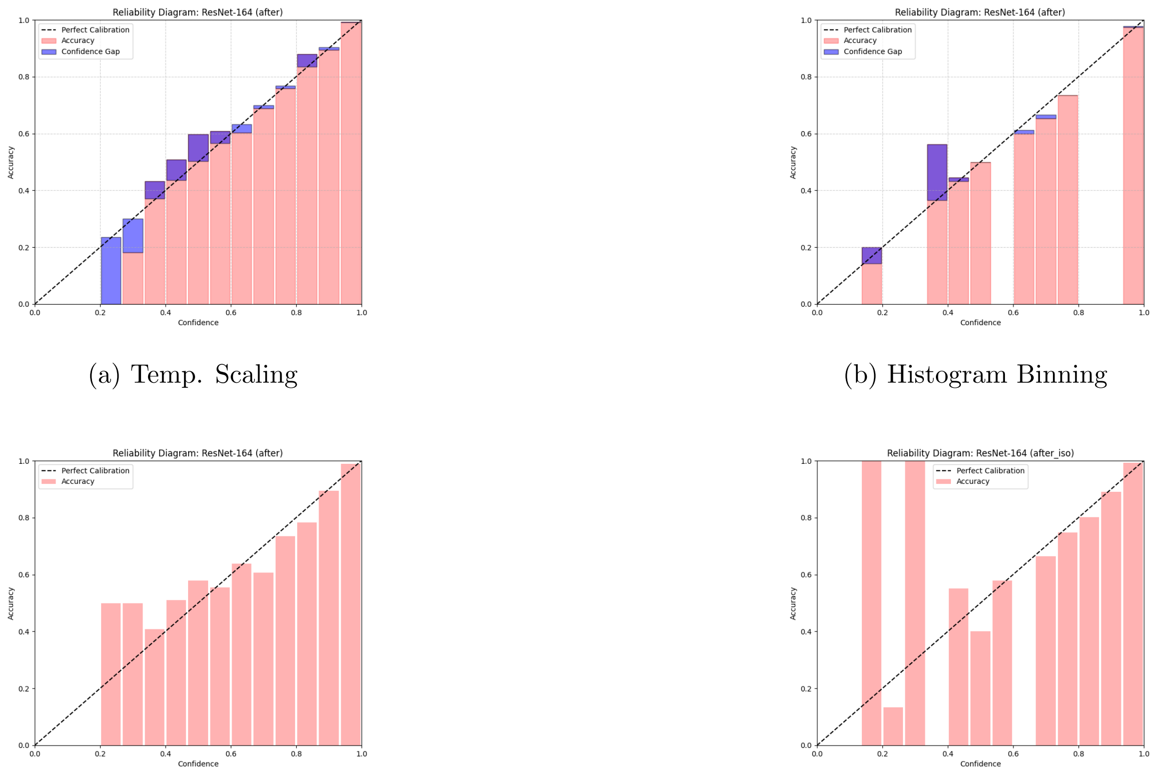


Figure 3.14: ResNet-164 on CIFAR-10 (uncalibrated).



(a) Temp. Scaling

(b) Histogram Binning

(c) Vector Scaling

(d) Isotonic Regression

Figure 3.15: ResNet-164 on CIFAR-10 (post-hoc methods).

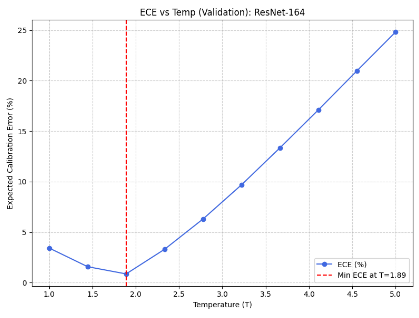


Figure 3.16: ResNet-164 on CIFAR-10: ECE vs Temperature.

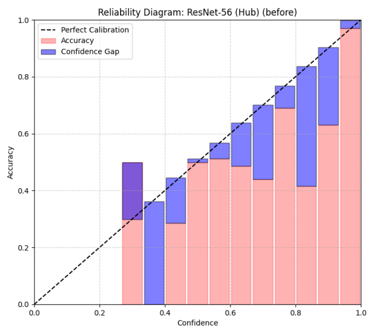
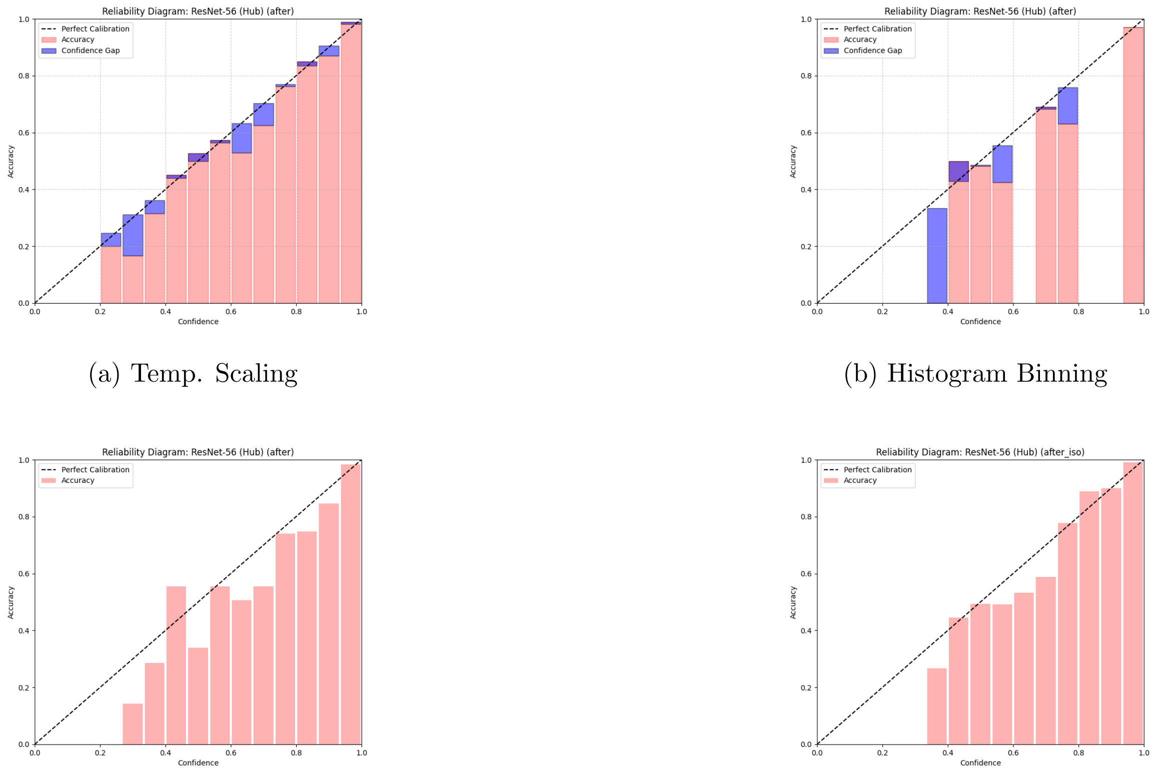


Figure 3.17: ResNet-56 on CIFAR-10 (uncalibrated).



(a) Temp. Scaling

(b) Histogram Binning

(c) Vector Scaling

(d) Isotonic (not available)

Figure 3.18: ResNet-56 on CIFAR-10 (post-hoc methods).

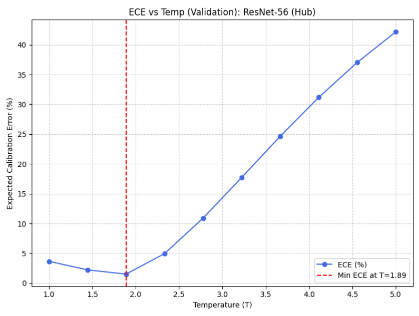


Figure 3.19: ResNet-56 on CIFAR-10: ECE vs Temperature.

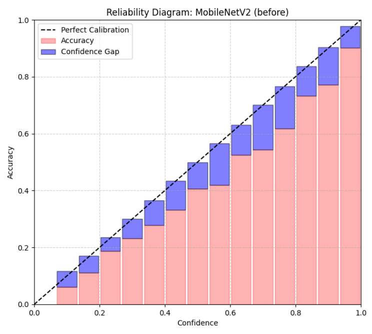
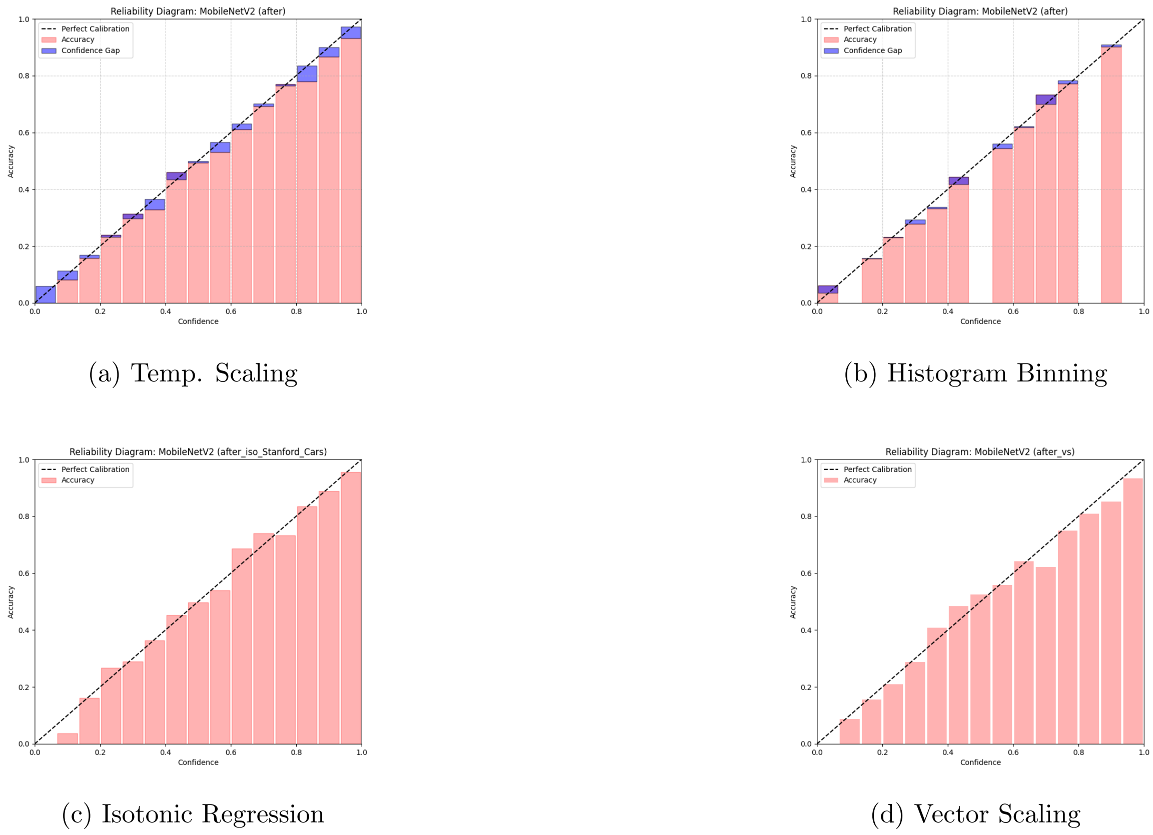


Figure 3.20: MobileNetV2 on Stanford Cars (uncalibrated).



(a) Temp. Scaling

(b) Histogram Binning

(c) Isotonic Regression

(d) Vector Scaling

Figure 3.21: MobileNetV2 on Stanford Cars (post-hoc methods).

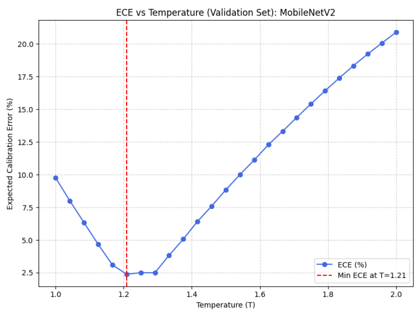


Figure 3.22: MobileNetV2 on Stanford Cars: ECE vs Temperature.

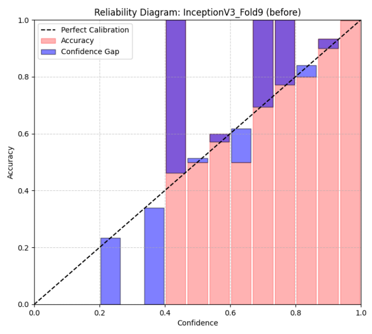
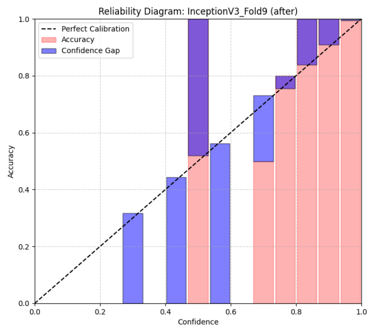
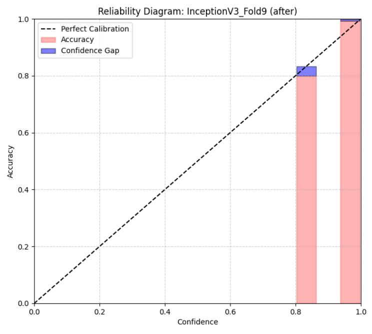


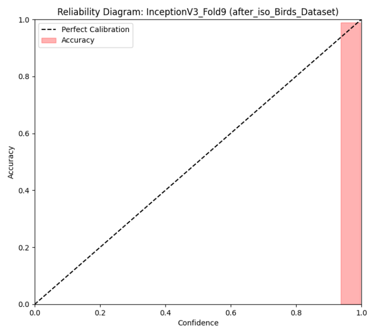
Figure 3.23: InceptionV3 on Birds400 (uncalibrated).



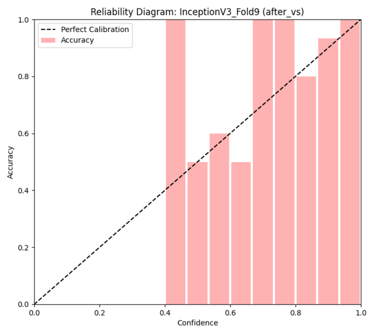
(a) Temp. Scaling



(b) Histogram Binning



(c) Isotonic Regression



(d) Vector Scaling

Figure 3.24: InceptionV3 on Birds400 (post-hoc methods).

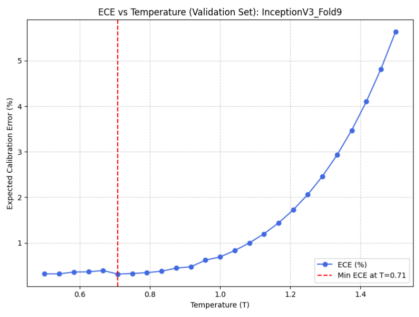


Figure 3.25: InceptionV3 on Birds400: ECE vs Temperature.

3.4 Analysis of Results

- **Confirmation of Hypothesis:** Consistent with Guo et al., the uncalibrated deep networks (ResNet-164, ResNet-56) exhibited high ECE (approx. 15% on CIFAR-100), confirming they are miscalibrated.
- **Performance of Temperature Scaling:** TS drastically reduced ECE across all miscalibrated models (e.g., reducing ResNet-164 error from 15.07% to 1.36%). Ideally, it matched or neared the performance of non-parametric methods (Histogram Binning) without the complexity of bin selection.
- **Failure of Vector Scaling:** Vector scaling consistently performed worse than TS (e.g., 4.27% vs 3.53% on WideResNet). This confirms the paper's finding that Vector Scaling tends to overfit to the validation set due to having more parameters.
- **The "Birds" Exception:** The InceptionV3 model was already perfectly calibrated (ECE 0.50%). Post-hoc methods slightly degraded performance, which show that calibration should only be applied when miscalibration is detected.

3.5 Results of Extra Work: Label Smoothing

Our experiment with Label Smoothing (LS) gave the following result:

- **Accuracy Drop:** Fine-tuning the pre-trained ResNet-56 with LS caused accuracy to drop from 94.12% to 92.90%.
- **Calibration degradation:** Consequently, the ECE increased from 3.96% (uncalibrated) to 6.05% (LS).

This shows that attempting to calibrate a pre-trained model via re-training (LS) is destructive compared to the safe, post-hoc nature of Temperature Scaling.

Chapter 4

Conclusion

Our project successfully replicated the key results of Guo et al. (2017). We demonstrated that modern deep networks suffer from poor calibration due to increased depth and width. Through our implementation, we verified that **Temperature Scaling** is the most desired solution. It consistently effectively reduces ECE across varied architectures (ResNet, DenseNet, MobileNet) without degrading classification accuracy, unlike Label Smoothing or Histogram Binning. This fulfils the goal of the FML project by validating the practical importance of calibration in deep learning.

Bibliography

[1] Guo, C., Pleiss, G., Sun, Y., & Weinberger, K. Q. (2017). *On Calibration of Modern Neural Networks*. International Conference on Machine Learning (ICML).