Project 2 explanations and insights

Andrés Delicado & Pablo Ares

1. GPU or CPU

Graphic Processor Units (GPU) are a good alternative to usual CPU´s. GPUs are very good for performing a huge amount of simple mathematical computations, which is precisely what the filters of Convolutional networks need. So, we have implemented a code to select, if available, the cuda GPU of NVIDIA (by means of google collab) as it will support much better the huge number of simple computations, speeding up our processing.

2. Lenet5

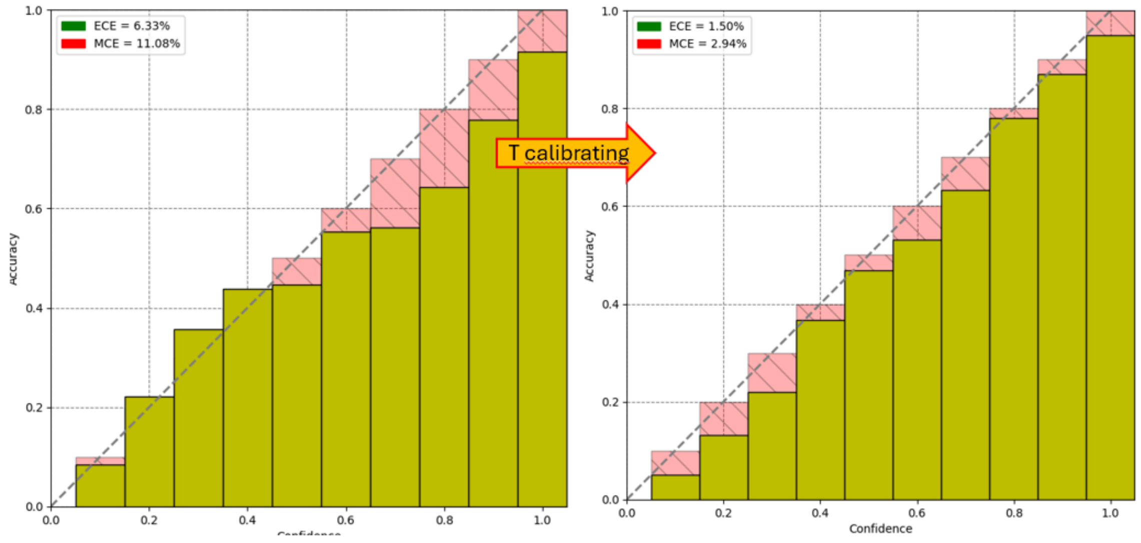
Lenet5 provides some benefits and disadvantages. The main benefits are that it is **efficient** and **relatively** simple, this means it requires fewer computational resources to train and use, making it suitable for applications with limited resources. In addition, it's good **generalization**.

Its main disadvantage comes precisely from its advantage and is that as Lenet5 is so simple it has some limitations on the **problem complexity**, being inefficient for complex problems.

As the structure of the CNN was fixed, our main investigations regarding the model architecture were focused on which are the better activation functions between layers, which resulted in Sigmoid for the first layer, and ReLu for the next ones. Also, at the beginning we didn´t implement batch normalization and then we implemented it and the accuracy increased, however, as the paper "*On calibration on modern neural networks*" says, including batch normalization made the ECE increase.

Binary cross entropy was found again to be the best loss function to train our model and **Adam** as optimizer. With this setup, our model performs in the following way:

Model	Accuracy	ECE	MCE	time (seconds)
Lenet5 without calibration	80.90%	6.5%	11.85%	82.75
Lenet5 calibrated	80.90%	1.5%	2.94%	82.75



3. Useful functions

-bins_composition: The main motivation for creating this function was that it makes much more accurate ECE computations. This function groups the predictions into bins, calculates the accuracy and average confidence for each bin, and returns these results along with the bin boundaries and the indices of bins assigned to each prediction. The best way we found to do this task was using binned=np.digitize to assign each prediction in y_pred_probs to the corresponding bin based on their values and the boundaries defined in bins. binned will contain the indices of the bins to which the predictions were assigned. With that, we can use the index selector binned=bin to identify the predictions we are interested in each bin. We compute the **accuracy** of the bin by dividing the sum of true labels (converted to one-hot encoding) by the size of the bin and the **average confidence** of the predictions in the bin by dividing the sum of prediction probabilities by the size of the bin.

-reliability_histogram: after creating a reliability graph by using only the calibration curve of sklearn, although it yields useful results, we thought that we could greatly improve visualization of the reliability graph, which is a fundamental point for the calibration analysis. So, we thought that we could use the bins division made by the previous function to create a histogram with the accuracy and confidence of each bin.

This function helps us by plotting the statistical accuracy as a function of confidence (estimated accuracy), we can visualize how well the confidence reflects the true accuracy. In this way we can see how well the predicted confidence scores hold up against their actual accuracy (the ideal result would be that the bins reach the line y=x, that is that the accuracy and confidence are identical at each bin).

Also, we see that we need no more than 250 interactions till the Tscaling factor converges and also the loss.

4. ECE calculation and additional indicators

Using the bins_composition function, we divided the prediction of the model into several bins, each one with its accuracy and confidence. Our final ECE computation was done by adding the absolute difference between the bins' confidence and accuracy, each one multiplied by the amount of samples of the bin (bin_size, which we did not do on the previous calculation) and divided by the total number of samples.

We wanted to include some additional indicators, and we included the MCE, which is computed at the same time as the ECE by simply using the max function of python, to see if the maximum calibration error also decreases with our T scaling. Moreover, we included the R2 as an additional indirect measure. The insights of this measure are explained in the code script, but basically are measuring how well our calibration curve fits the line y=x that would mean perfect calibration.

5. T scaling optimization

The optimization of T_scaling is achieved through the gradient descent process implemented by the LBFGS optimizer. In each iteration, the gradient of the loss with respect to T_scaling is calculated, and T_scaling is updated in the direction that minimizes the loss. We use the LBFGS optimizer with a line search function to obtain the best T scaling value that improves the most the reliability of our net. Initially, we set the Tscaling value to 1 and use nn.Parameter to let Pytorch know it is a parameter to optimize. After this we have several stages:

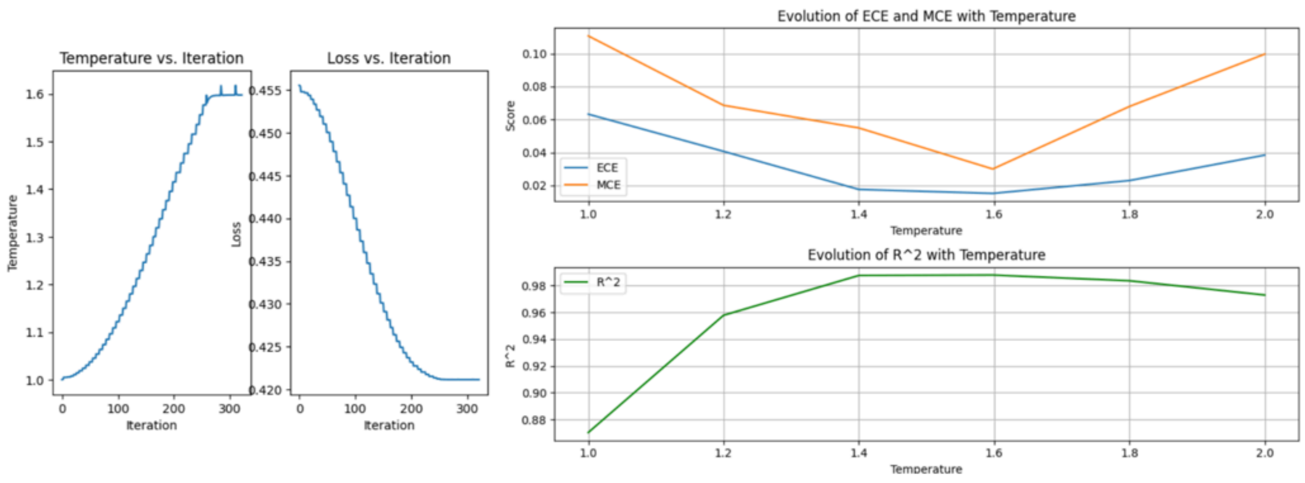
An **evaluation loop**, in which the test dataset (testloader) is iterated through, and the model's predictions for each image are obtained. These predictions are stored in logits_list, and the mapped labels are stored in labels_list.

And an **Evaluation Function (eval):** the function _eval() calculates the loss using the T_calibrating function to apply temperature adjustment (temperature) to the model's outputs (logits_list). It then computes the loss using the previously defined loss criterion. It also logs the current temperature and loss in lists (temps and losses, respectively).

At the end, the temperature is optimized by calling the optimizer.step(_eval) method to perform an optimization iteration using the LBFGS algorithm. In each iteration, the temperature parameter is updated to minimize the loss calculated by the _eval function.

The final T calibrating factor for Lenet5 is 1.598, which minimizes the ECE and MCE, and also maximizes the R2 (the R2 is similar in some other places but this is due to the fact that R2 is an indirect measure, so we are not maximizing it explicitly).

Temperature optimization to reduce the loss of the evaluation function and ECE, MCE and R2 as a function of temperature:

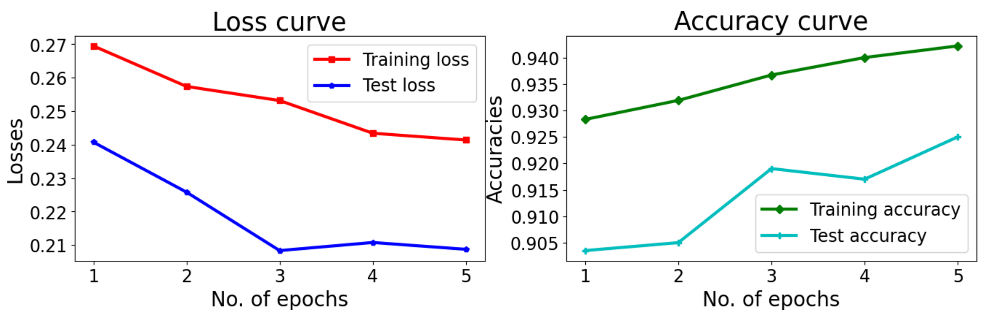


6. AlexNet as pre-trained model

We want to compare the results we have obtained with the ones of a pre-trained AlexNet model. We need to do some adaptations, so the AlexNet fits our data correctly:

- The most important part is to add some Linear layers at the end that reduce the dimensionality gradually till we end up with the number of outputs as classes that we want.
- Also, we found that a good implementation was freezing the pre-trained layers to prevent the pre-trained weights from being modified too much during training with your dataset. This means that these layers will not be updated during the training process, which helps retain the previously learned knowledge by the pre-trained model. We do this by going through all the layers using *for param in alexnet.parameters()* and setting *param.requires_grad=False* to avoid computing the gradients during training. So, in conclusion, when we train we compute the weights of the last layers we added, but keeping the knowledge from the pre-trained AlexNet model on its original layers.
- We needed to resize the train and test loaders.

The training history is the following (with 5 epochs):

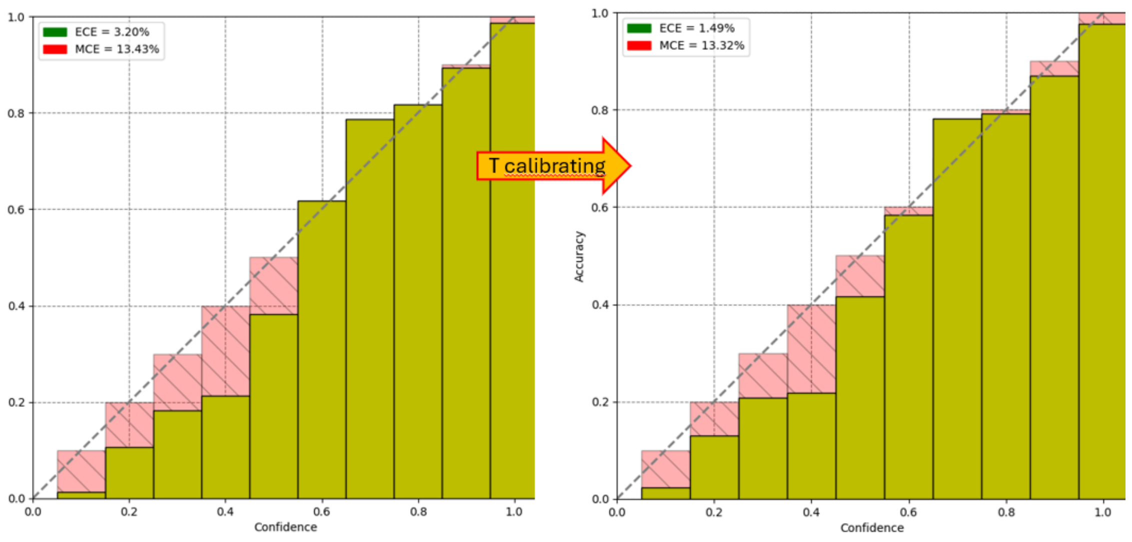


After making a small train of the model with our data (although the training time is much larger than with Lenet5 for smaller amount of data), with the objective that it learns it and adapts to our dataset, we end up having a better model than the Lenet5 from scratch that we made:

Model	Accuracy	ECE	MCE	Correctly predicted (out of 16)	Time (seconds)
Lenet5 calibrated	80.90%	1.50%	2.94%	15	82.75
AlexNet calibrated	92.50%	1.49%	13.32%	13/14	3977.18

A huge issue:

The Temperature factor for this model is 0.794. Once calibrated, the **ECE** for both models was 1.50 %, however the Lenet5 has significantly less **MCE** (3% vs 13%), indeed, we have observed that sometimes it has too much confidence in its predictions, which later result to be wrong. This means that it is very confident in its predictions although they are wrong, which is a huge problem and a handicap of this model, making it difficult to trust its predictions and compare it with other models. We optimized its temperature factor, but applying Temperature Scaling to AlexNet, which worked very well for Lenet5, did reduce its ECE, but the MCE almost did not change. So, we still have a great Calibration error for some predictions, which could be problematic for its interpretability and reliability. The **time** needed for the little training that the AlexNet model needed was 47 times larger than for the other smaller model (67 minutes vs 1.5 minutes), but we think it is a price worth paying to obtain a 10% more accuracy, which could be critical in some contexts. We can see how the T calibration does very little:



In conclusion, nowadays we have very well-made models that have been previously trained with a huge amount of data that we, as single normal users, are not able to do. Using this pre-trained models is a good idea for further tasks, we only need to do some adaptations of the layers so they can process our data and we obtain the results we want (mainly adapting the last layer to the number of outputs we want) and a little training on our dataset to learn it (specially the last layers we added), to obtain quite good results. Indeed, better results than creating a Neural Network ourselves and with less effort.

So, in tasks where a very high accuracy is critical and we want to maximize it, we would use as a start point a pre-trained model that we will adapt to make the job we need.

6. Final conclusions

We have learned the importance of the calibration of a model, that was something unknown for us. The ECE measures how well the predicted probabilities of a model are calibrated. This could not seem very problematic, but a bad calibration error has several huge problems such as: **Overconfidence** (or underconfidence), if a model is poorly calibrated, it may overestimate or underestimate the certainty of its predictions. For example, it may assign a high probability to an incorrect class. This could also make the user of our model not trust the model. Also, it makes it much more difficult for **comparing models** , as the predicted probabilities may not be comparable to each other.

Moreover, we see that using temperature scaling for the logits of our model reduces both the expected calibration error and the maximum calibration error. In our case reducing the ECE from 6.33% to 1.50% and the MCE from 11.08% (which means a huge error in a bin) to 2.94%, which is a much more smooth error and easy to deal with. The best part of all this, is that the accuracy of our model stays exactly the same after applying temperature calibration. This is because we are simply scaling the logits before passing them to the sofmax function (the one that makes the final prediction), but the dominant class of each logit vector is exactly the same, as we are scaling by the same factor all the logits of each class, so the bigger one will still be the same one and the prediction will be the same one.

To sum up, we are able to significantly reduce the ECE and MCE without affecting the accuracy, which translates into a better model, more reliable and easy to compare with others. We have learned the importance of the ECE and we will consider it on further models, also we have learned a good tool to improve the calibration of our model and how to use it. Additionally, the exploration of a pre-trained model showed us that it is better to adapt a pre-trained model for our problem than creating a new model for our explicit job, yielding a higher accuracy. However we need to be careful with the overconfidence of the predictions of this pretrained model.