A Mixup training

In this appendix, we provide an extended explanation to the one in section 4, regarding the reasons why Mixup should not necessarily provide calibrated distributions. Mixup has its fundamentals in vicinal risk minimization (VRM) [2], which is derived as a solution to the limitations present in ERM [31,37,32]. The ideal learning signal in a frequentist paradigm should be provided by the gradient of the expected value of a loss function over the underlying probability density $P(x,t)$,

$$\mathcal{R}(\theta) = \int L(g_\theta(x), t)\, \mathrm{d}P(x,t) \tag{5}$$

which in practice is impossible as we only have access to an i.i.d. sample $\mathcal{O}$. As a consequence, we attempt to minimize the expected value of the loss function over the empirical distribution $P_d(x,t)$, a process known as empirical risk minimization (ERM). This distribution is given by Dirac delta distribution centred at the observed points:

$$P_d(x,t) = \frac{1}{N}\sum_i \delta(x_i, t_i) \tag{6}$$

Thus, ERM clearly lacks of support in many different parts of the input space, which makes this learning paradigm present some limitations such as over/underfitting, memorization [36], or sensitivity to adversarial examples [28]. VRM is proposed to solve this lack of support in the input manifold. To achieve this goal, the Dirac Delta distribution is substituted with a *vicinity* distribution, which aims at exploring different parts of the input space in the vicinity of the observed set $\mathcal{O}$. For instance, a vicinity distribution can be implemented as a Gaussian centred at each sample x_i. In practice, we then sample from this Gaussian distribution and recover an unbiased estimate of $\mathcal{R}(\theta)$ computed with this new set of generated samples, which is used in conjunction with stochastic gradient guided learning algorithms. Thus, any DA technique, such as Gaussian noise addition, can be understood under the VRM paradigm.

The main motivation behind Mixup is that DA techniques assume that the samples in the vicinity distribution belong to the same class. For that reason, Mixup vicinity distribution is defined as the expected value of a linear interpolation between two input samples and their corresponding labels [37]. This interpolation is parameterized by a coefficient γ which is drawn from a beta distribution. An unbiased estimate of $\mathcal{R}(\theta)$ can be obtained by evaluating the average loss function on a set of samples drawn from this distribution as follows:

$$\begin{aligned} \gamma &\sim \mathrm{Beta}(\alpha, \alpha) \\ x &= \gamma \cdot x_1 + (1-\gamma)\cdot x_2 \\ t &= \gamma \cdot t_1 + (1-\gamma)\cdot t_2 \end{aligned} \tag{7}$$

As a consequence, training with Mixup smooths the predictions performed by a model in the intersection between samples from the unknown distribution $P(x,t)$. However, even if this might reduce high-oscillations in the predictions performed in these regions of the feature space, or smooth the ultimate confidence assigned to these regions, this only ensures that the model will be less overconfident, which does not necessarily mean that the ultimate probability distribution will be calibrated. This is because Mixup only ensures a linear-soft transition between the confidence assigned by the model in different parts of the input space. As a consequence, only if the data distribution presents a linear relation between their corresponding classes, one could expect the ultimate distribution to be calibrated. It is clear that Mixup interpolation does not consider the proportion of samples present in the input distribution, which is at the core of a proper calibration. In the experimental section we show that some models trained with Mixup do not necessarily improve the calibration, as recently noted [30]. In fact, our results show that Mixup can highly degrade calibration in many cases.

B Measuring Calibration

Calibration can be measured in different ways, each one with their own properties. While some metrics are directly PSR, such as the Brier score (BS)[4] or the logarithmic score (NLL)[4], averaged over empirical samples; some others are merely measures of calibration, such as the expected calibration error (ECE) [8] and the maximum calibration error (MCE)[8]. Given a set of M samples, each of these metrics can be computed in the following way:

$$\begin{aligned}
\texttt{BS} &= \frac{1}{M}\sum_{n}(g_\theta(x_n) - t_n)^2 \\
\texttt{NLL} &= -\frac{1}{M}\sum_{n} t_n \cdot \log[g_\theta(x_n)] \\
\texttt{ECE} &= \sum_{j}\frac{|B_j|}{M}|\text{acc}(B_j) - \text{conf}(B_j)| \\
\texttt{MCE} &= \max_{j}|\text{acc}(B_j) - \text{conf}(B_j)|
\end{aligned} \tag{8}$$

where the $[0,1]$ confidence range is equally divided into j bins B_j. In each of these bins, the accuracy (acc) and the average confidence (conf) of the samples that lie in that particular bin are computed. In this case t_n represents a one-hot vector for the class probability k.

Note, for instance, that the NLL score highly penalizes important errors (i.e. extreme and wrong probabilities), but it is not able of separating which part of those errors are due to discrimination and which to calibration. This means that the NLL will always be penalized under non-separable data manifolds even though we face the ideal situation in which the model has recovered the data generation

distribution (thus, when it presents perfect calibration and has recovered data discrimination). In this situation, a perfect model will present $\texttt{ECE} = 0$ because ECE is just a measure of calibration.

On the other hand, while ECE is not sensible to high extreme errors made in only one sample (assigning 1.0 confidence towards an incorrect class), NLL penalizes this error by assigning an infinity score. In this sense, NLL is much more sensitive to strong overconfidence error than the rest of the calibration performance metrics and can be more useful in applications where overconfidence errors must be avoided in a very restrictive way.

C Further Loss Analysis

In this appendix, we provide additional analysis of the proposed loss function. First, note that while the CE loss aims at pushing the probability of a given sample x_i towards 1.0 confidence of belonging to its associated class t_i, our loss encourages the model to auto-adjust its confidence depending on the accuracy of each batch of data being forwarded through the model. It is clear that this loss function and the CE play different, and opposite, roles regarding the probabilistic information that the model should provide. For this reason, we might think that the combination of both losses could lead to a suboptimal result, as each of the losses pushes in opposite directions. In order words, both losses play a give-and-take game. However, note that learning signals provided by the losses are somehow complementary. At the beginning of the learning processes, when the network is initialized at random, the network typically parameterizes a quasi-constant output distribution, and the accuracy provided by the model is near to that of a prior classifier. Thus, the learning signal provided by the ARC loss is negligible as compared to the one provided by the CE. On the other hand, when the optimization of the CE stalls, then the ARC loss plays its role by adjusting the ultimate confidences if they are uncalibrated. This trade-off between ARC and CE can be seen as a type of regularizer of the CE by ARC, preventing CE to reach discrimination without taking care of calibration. Our loss will not let the CE push the probability towards extreme 1.0 values.

Moreover, it should be noted that this cost presents other desirable properties that aim at improving regularization. First, consider a set of samples lying in the confidence range $[0.6, 0.7]$. If the accuracy of these samples is located in this range then our loss function will encourage the model to adjust them to be as close as possible to the accuracy. Second, if the accuracy provided by the model has a value over this range, e.g 80%, then the model will raise these confidences to recover a calibrated model. It should be noted that in this case, our loss function will not change the accuracy as we are just pushing upwards the confidence of the samples which are originally correctly/incorrectly assigned, and thus the decision of which class should be assigned to each sample remains intact. Third, consider the same set of samples but with a provided accuracy of 40%. Our loss function will encourage these set of samples to reduce its confidences. It is clear that reducing this confidence has to be done at the cost of raising the

confidence towards other classes. By doing this, we have a chance of changing the decision made by the model towards another class, thus helping to improve the discrimination of the model and consequently raising the accuracy.

On the other hand, the idea of experimenting with the two variants of our loss named $\texttt{ARC\_V}_1$ and $\texttt{ARC\_V}_2$ is based on the following observation. The only difference between the two variants is whether we force the average confidence of a set of samples to match the accuracy, as performed by $\texttt{ARC\_V}_1$, or we force each individual sample to match the accuracy, as done by $\texttt{ARC\_V}_2$. $\texttt{ARC\_V}_2$ is proposed to avoid solutions in which the set of confidences assigned by the model presents high variance. This will avoid solutions in which, for instance, the network present a 90% accuracy on a set of samples, and the model assigns 0.8 confidence to half of the samples and 1.0 to the other half. In such a setting, the loss being minimized will be 0, but the ultimate goal will not be achieved. The possibility of computing our loss over separate bins is incorporated to reduce this effect. However, in practice, we expect both losses to work, as the ideal behaviour of a good representation as learned by a model should be to map all the samples of a given class to the same (ideally linearly separable) representation. If this happens, the aforementioned variance on the confidence assigned by the model is reduced.

D Additional Experimental Details

Datasets We choose datasets to evaluate our approach. We rely on classical benchmarks such as (number of classes into the brackets) CIFAR100 (100)[18], CIFAR10 (10)[17], SVHN (10)[23], and we also evaluate our model on more realistic problems such as the ones provided by Caltech-Birds (200)[34], Standford-Cars (196)[16]. These datasets are made up of bigger and more realistic images, and a padding preprocessing must be done. Due to computational restrictions, we did not evaluate our model on ImageNet.

Models. We evaluate our model on several state-of-the-art configurations of computer vision neural networks, over the mentioned datasets: Residual Networks [9], Wide Residual Networks [35] and Densely Connected Neural Networks [14]. Moreover, for each variant, we evaluate a model with and without Dropout. We find this interesting because a dropout model can be used to quantify uncertainties [6,15]. For the ResNet we add a Dropout layer after the whole network. We set the Dropout values according to the ones provided in the original works, or the model implementations, except for the ResNet where we use a 0.5 Dropout rate. We use the pre-trained models on ImageNet provided by the PyTorch API for Birds and Cars datasets. On these pre-trained models, we add a Dropout layer at the end. Models are optimized with stochastic gradient descent with momentum and by placing a Gaussian prior over the parameters. The precision of this Gaussian prior is set accordingly to the provided implementations. For all the databases except Birds and Cars we use a learning rate starting from 0.1. For Birds and Cars the initial learning rate is set to 0.01. We use step learning

rate scheduler that varies depending on the model. Additional details can be found in the code.

Data Augmentation Hyperparameters: Regarding Mixup hyperparameters we used the ones provided in the original work. On the datasets where these techniques were not evaluated, we searched for the optimal value on a validation set. This hyperparameter is then fixed for the rest of the experiments carried out. More details on Github.

ARC Hyperparameters: Our loss hyperparameters: β, the number of bins and the type of cost used (V_1-V_2) were searched using a validation set with the ResNet-18 for all the models with and without Dropout. This is because we wanted to extract conclusions on a possible good configuration of our loss function and to do that we need to do a big battery of experiments (we trained more than 1000 Neural Networks to evaluate the loss); and this big experimental search came at the cost of computational restrictions.

As explained in the experimental section, this allows us to conclude that $\texttt{ARC\_V}_1$ and bins $M = 1$ are a reliable choice of the hyperparameters. Our search include all the possible combinations of: loss $\texttt{ARC\_V}_1$ and $\texttt{ARC\_V}_2$; number of bins: $M = 1$, $M = 15$ and $M = \{5, 15, 30\}$ (for this one the loss is computed three times, one per each value of M, and the three losses are then averaged); and evaluation of the ARC loss over the Mixup image $\tilde{x}$ or the separate images x_1 and x_2. This experiment was essential to validate our claim regarding data uncertainty and calibration, as exposed in section 4. We run experiments over all these combinations, searching for the optimal β value. We select the value of β that provides good accuracy with low calibration error. In some cases we found this hyperparameter to be 40 times greater than the CE loss, see details on Github. This enhances the beneficial influence that our loss function can have in several problems.

Note that this way of searching for hyperparameters is not optimal. In general, the extrapolated hyperparameter performed well in the rest of the models as detailed in the models on GitHub. However, sometimes, we experimented accuracy degradation in the training set. This is because a pathological solution of optimizing the ARC loss is by setting the parameters to output the data prior probability. This solution evaluates the ARC loss to 0, but at the cost of parameterizing a useless prior classifier. As an example consider, for instance, that on the ResNet-18 we found that the optimal hyperparameter was $\beta = 42$, but when training a DenseNet-121 this hyperparameter degraded the accuracy over the training set at the cost of providing perfect calibration. When this effect was observed we just picked the next hyperparameter that provided the next top performance over the ResNet-18; until the training accuracy was not degraded.

On the other hand, in CIFAR100 with Mixup we found this way of searching for the hyperparameter not to be as effective for the models without dropout. As provided in the specific results for each model trained on CIFAR100 in the tables provided in Github, we can see that all the models except ResNet-18 improve calibration when using Mixup. Thus, we cannot expect the hyperparameter to

extrapolate as with other datasets. This was observed by training any of the deeper models with a validation set. To solve this, we simply perform a hyperparameter search over one of the deeper models in which Mixup showed great calibration performance, and use this parameter with the rest of the models. Due to computational limitations, we did not perform such an exhaustive search as we did with the ResNet-18, and just select a subset of the hyperparameters based on the previous wider analysis performed over the ResNet-18.

D.1 Additional Results

We finally include additional results in our experiments. Table 3 show average results and table 4 show best performing model result for other calibration metrics. We can see that the results extrapolate from those in the experimental section, showing the improvement achieved by A+M.

Finally, table 5 shows the results of applying `ARC` loss only to a validation set that is uncalibrated. Surprisingly, the `DNN` can increase the accuracy of this validation set without using the `CE`, instead of relaxing the confidences. This shows the great ability of `DNN` to overfit, and manifest the unpredictable behaviour of these models when used in probabilistic machine learning. This motivates the search of new losses that can encourage these powerful models to better represent the underlying distribution, and thus move them towards a better generalization, mandatory for critical applications.

Table 3. This table shows different calibration metrics for average results. ACC in (%), `MCE` in (%), `BS` $\times 100$ and `NLL`

	CIFAR10				CIFAR100				SVHN				Birds				Cars			
Model	ACC	MCE	BS	NLL	ACC	MCE	BS	NLL	ACC	MCE	BS	NLL	ACC	MCE	BS	NLL	ACC	MCE	BS	NLL
B	94.76	2.16	0.86	0.23	77.21	5.20	0.35	1.08	96.32	1.86	0.62	0.17	78.51	0.58	0.17	1.04	86.74	0.56	0.10	0.52
B+M	96.01	2.88	0.65	0.18	80.04	0.67	0.29	0.79	96.41	2.76	0.63	0.18	79.63	1.49	0.18	1.11	86.67	1.81	0.13	0.71
M	94.24	1.06	0.89	0.21	72.68	0.61	0.38	0.98	96.28	1.12	0.61	0.17	78.78	0.46	0.17	1.02	86.83	0.59	0.10	0.52
M+M	91.90	2.73	1.33	0.32	78.52	1.18	0.31	0.86	96.59	1.46	0.59	0.16	79.99	1.23	0.17	1.07	86.03	1.28	0.12	0.67
A	94.85	2.13	0.85	0.23	77.04	4.78	0.35	1.05	96.26	1.16	0.62	0.17	78.52	0.65	0.17	1.04	87.78	1.32	0.10	0.51
A+M	95.90	0.78	0.67	0.17	79.84	0.54	0.29	0.80	96.02	1.11	0.64	0.16	79.74	0.65	0.15	0.82	89.63	1.70	0.08	0.44

Table 4. This table shows different calibration metrics for the best model per task and technique. `ACC` in (%), `MCE` in (%), `BS` $\times 100$ and `NLL`

	CIFAR10				CIFAR100				SVHN				Birds				Cars			
Model	ACC	MCE	BS	NLL	ACC	MCE	BS	NLL	ACC	MCE	BS	NLL	ACC	MCE	BS	NLL	ACC	MCE	BS	NLL
B	95.35	1.23	0.65	0.15	79.79	2.36	0.29	0.81	97.07	0.18	0.48	0.12	80.31	1.01	0.16	0.98	89.13	0.42	0.089	0.45
B+M	97.19	3.11	0.47	0.14	82.34	0.46	0.26	0.70	96.97	2.55	0.53	0.16	82.09	1.12	0.15	0.97	89.45	1.84	0.110	0.61
M	95.58	0.46	0.67	0.15	74.98	1.18	0.36	0.92	96.90	0.34	0.49	0.13	80.64	0.99	0.16	0.97	89.40	0.35	0.087	0.44
M+M	97.02	0.73	0.45	0.11	81.31	0.70	0.28	0.74	97.17	1.36	0.50	0.14	82.41	1.05	0.15	0.96	88.47	1.14	0.105	0.59
A	95.99	1.02	0.62	0.14	80.77	2.25	0.28	0.79	97.08	0.17	0.47	0.12	80.32	1.17	0.16	0.99	90.09	1.21	0.080	0.42
A+M	97.09	0.39	0.48	0.12	82.02	0.31	0.26	0.72	96.82	1.75	0.51	0.14	82.45	0.34	0.13	0.69	91.13	0.91	0.078	0.42

Table 5. This table shows the results of applying the `ARC` loss just to a validation set.

	CIFAR100				CIFAR10				SVHN			
	validation		test		validation		test		validation		test	
β	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE	ACC	ECE
0.5	82.74	4.32	77.97	9.48	97.28	1.29	95.29	2.82	98.50	1.21	96.45	2.32
1.0	86.46	4.29	78.74	10.65	97.92	0.88	94.92	3.39	98.86	0.54	96.48	2.33
2.0	91.26	2.17	79.49	9.19	99.08	0.27	95.25	3.05	99.08	0.33	96.54	2.37
4.0	94.30	1.61	79.61	8.86	99.74	0.21	95.60	2.63	99.24	0.24	96.50	2.33
8.0	96.26	1.08	78.85	9.73	99.84	0.18	95.58	2.72	99.32	0.18	96.67	2.13