Multi-layer Radial Basis Function Networks for Out-of-distribution Detection

Supplementary Material

6. Initialization Details

For layer ℓ with N centroids, k-means clustering with N centroids was used to find the initialization positions for the centroids. This can be done quickly for a large batch of data using web-scale k-means clustering with k-means++ initialization [1, 38]. Then for each datapoint in the initialization batch, the distance to its closest centroid is computed, producing a vector in $\mathbb{R}^n$, where n is the number of datapoints in the batch. Similarly, for each centroid, the distance to its closest datapoint was computed, producing a vector in $\mathbb{R}^N$. Note that distances were computed using $\|\cdot\|_k$, which is also used in the RBFN forward pass.

To set $\beta^+_{c(\ell)}$, we take inspiration from the Gaussian distribution. In the Gaussian distribution, 95% of datapoints fall within 2 standard deviations of the mean. This can be represented as $2\sigma = d$, or equivalently, $4\sigma^2 = d^2$. Since $\frac{1}{\sigma^2} \propto \beta^+_{c(\ell)}$, we should expect to set $\beta^+_{c(\ell)}$ proportional to $\frac{4}{d^2}$.

Returning to our two vectors in $\mathbb{R}^n$ and $\mathbb{R}^N$, call the larger of the 95% quantile of each of these metrics d. We can use this d as an estimate of the necessary width of a Gaussian distribution, and thus set $\beta^+_{c(\ell)} = \frac{4}{d^2}$. Note that we initialize every inverse-width in a layer to the same value, $\frac{4}{d^2}$. In our code, where $\beta^+_{c(\ell)} = \texttt{softplus}(\beta_{c(\ell)})$, we actually initialize $\beta_{c(\ell)} = \texttt{inverse-softplus}(\frac{4}{d^2})$.

To see this procedure work in code, please see our code in section 9.

7. Moons

To enable development and achieve a simple baseline, we employed a 4-class moons problem. Similar datasets have been used in other papers on uncertainty quantification due to their ease of visualization.

The 4-class moons dataset was generated from scikit-learn's make moons function with noise parameter 0.2, which produces a 2-class moons problem [32]. Two extra classes were generated by shifting the base 2-class moons problem by 2 units in each direction. Overall, the dataset had 1000 training datapoints and 500 testing datapoints split equally among each class. The dataset was normalized prior to training.

We trained a 3-layer MLRBFN with $N = 50$ centroids in the first and second layers and $N = C = 4$ centroids in the third layer. A projection dimension of $o = 100$ was used in the first and second layers. k was set to 2 for all layers. The Adam optimizer was used with a learning rate of 10^{-3} [20]. The network was trained for 250 epochs with a batch size of 100.

Figure 1 demonstrates the performance of our MLRBFN on the 4-class moons problem. Datapoints are scattered and colored according to their true class. Color in the background represents the class prediction while shading represents the confidence of the network, with white being low confidence and dark being high confidence. It is clear that the network can classify the dataset correctly ($> 99.5\%$ training and testing accuracy) and is only confident near the training data manifold.

8. Results Continued

8.1. Tabular Results

In this section, we include tabular results referred to in the main body of the paper. These can be seen in Table 1, Table 2, Table 3, and Table 4.

In the main body, we perform a deep analysis of Table 1 and Table 2, so here we discuss the other two tables. From Table 3, it is clear that MLRBFNs produce significantly better results than MSP and are competitive with other presented methods. Some other methods, like RMDS, have significantly better performance across all datasets, but these methods can be incorporated with MLRBFNs to produce even better performance.

From Table 4, MLRBFN results are significantly better than MSP: MSP has a 100% false positive rate across all datasets, while MLRBFNs reduce this significantly. Again, MLRBFNs can be combined with other methods presented in the table to produce improved performance.

8.2. OOD Detection with Increasing Number of Classification Layers

Finally, we tested how the number of trainable "head" layers affects OOD detection performance in DNNs and MLRBFNs. To do this, we trained networks with 1, 2, 3, or 4 head layers on CIFAR10 features extracted by CLIP, and measured AUROC on near-OOD and far-OOD datasets. Results are shown in Figure 5. As can be seen, MLRBFN performance slightly increases as the number of layers increases, while DNNs with ReLU activation functions produce worse performance as the number of layers increases. This may be a sign that feature collapse can occur rapidly, as the DNN's trainable "head" layers map all inputs to one of the training classes even when DNNs use powerful frozen feature extractors. MLRBFNs are able to achieve similar accuracy to DNNs but do not have degrading robustness to OOD inputs.

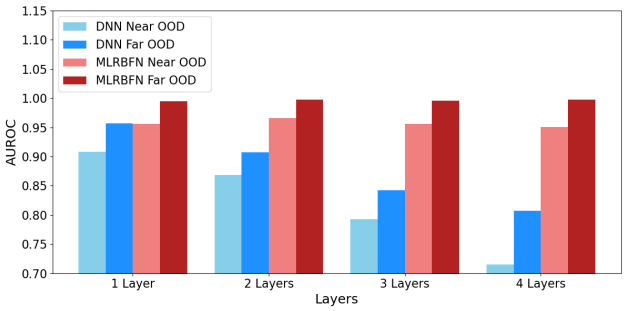

Figure 5. AUROC for near- and far-OOD tasks on the CIFAR10 dataset. MSP was used for DNN OOD detection. As the number of classification layers increases, MLRBFNs maintain high AUROC for the OOD detection task, while DNNs have decreasing AUROC. This supports MLRBFNs as networks which are naturally capable of detecting OOD inputs.

Table 1. Comparing the performance of MLRBFNs to published benchmarks and existing methods in the OpenOOD v1.5 study. In this table, we use AUROC as the metric for OOD detection. Whenever possible, we report the average number and the corresponding standard deviation obtained from 3 training runs. In almost all cases, MLRBFNs using the ResNet18 (for CIFAR10/CIFAR100/ImageNet200) and ResNet50 (for ImageNet1K) backbones are competitive with the average performance of methods from the OpenOOD v1.5 study. MLRBFNs using DINOv2 and CLIP as feature extracts generally outperform the best performing methods from OpenOOD v1.5.

	CIFAR10			**CIFAR100**			**ImageNet200**			**ImageNet1K**		
	Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.
- Multi-layer Radial Basis Function Networks												
ResNet	88.88$_{(\pm 0.26)}$	90.01$_{(\pm 0.58)}$	94.63$_{(\pm 0.19)}$	77.30$_{(\pm 0.49)}$	80.51$_{(\pm 1.18)}$	69.64$_{(\pm 0.30)}$	81.02	89.10	84.50	74.30	86.60	70.72
DINOv2	96.01$_{(\pm 0.34)}$	97.60$_{(\pm 0.64)}$	97.79$_{(\pm 0.04)}$	89.18$_{(\pm 0.73)}$	93.18$_{(\pm 1.17)}$	85.25$_{(\pm 0.32)}$	90.24$_{(\pm 0.22)}$	96.82$_{(\pm 0.52)}$	94.61$_{(\pm 0.37)}$	78.84$_{(\pm 0.25)}$	92.57$_{(\pm 0.21)}$	78.62$_{(\pm 0.12)}$
CLIP	95.75$_{(\pm 0.35)}$	98.46$_{(\pm 0.20)}$	95.78$_{(\pm 0.16)}$	86.53$_{(\pm 0.84)}$	95.96$_{(\pm 1.35)}$	76.04$_{(\pm 0.17)}$	85.05$_{(\pm 0.42)}$	94.78$_{(\pm 0.42)}$	90.97$_{(\pm 0.07)}$	72.03$_{(\pm 0.51)}$	88.50$_{(\pm 0.32)}$	72.16$_{(\pm 0.11)}$
- OpenOOD v1.5 Benchmarks												
Best Post-hoc	90.64$_{(\pm 0.20)}$	93.48$_{(\pm 0.24)}$	95.06$_{(\pm 0.30)}$	81.05$_{(\pm 0.07)}$	82.92$_{(\pm 0.42)}$	77.25$_{(\pm 0.10)}$	83.69$_{(\pm 0.04)}$	93.90$_{(\pm 0.27)}$	86.37$_{(\pm 0.08)}$	78.17	95.74	76.18
Best Mod. Training	92.68$_{(\pm 0.27)}$	96.74$_{(\pm 0.06)}$	95.35$_{(\pm 0.52)}$	80.93$_{(\pm 0.29)}$	88.40$_{(\pm 0.13)}$	77.20$_{(\pm 0.10)}$	82.66$_{(\pm 0.15)}$	94.49$_{(\pm 0.07)}$	86.37$_{(\pm 0.16)}$	76.52	92.18	76.55
Best Outlier Exp.	94.82$_{(\pm 0.21)}$	96.00$_{(\pm 0.13)}$	94.95$_{(\pm 0.04)}$	88.30$_{(\pm 0.10)}$	81.41$_{(\pm 1.49)}$	76.84$_{(\pm 0.42)}$	84.84$_{(\pm 0.16)}$	89.02$_{(\pm 0.18)}$	86.12$_{(\pm 0.07)}$	N/A	N/A	N/A
Avg. Post-hoc	79.18	83.24	95.06	73.16	76.43	77.25	75.88	84.02	86.37	71.11	85.36	76.18
Avg. Mod. Training	88.06	91.91	94.15	76.28	78.88	73.74	79.44	90.47	85.52	72.96	87.35	75.36
Avg. Outlier Exp.	91.12	93.25	94.12	81.09	78.03	74.84	81.35	87.08	81.44	N/A	N/A	N/A

Table 2. Comparing the performance of MLRBFNs to published benchmarks and existing methods. In this table, we use AUROC as the metric for OOD detection. Whenever applicable, we report the average number and the corresponding standard deviation obtained from 3 training runs. The table demonstrates that MLRBFNs outperform MSP in all cases (emphasized in red), meaning that the outputs of MLRBFNs are better suited for OOD detection than standard DNNs. This result supports our goal of building a neural network architecture which is better able to inherently detect OOD inputs than modern DNNs.

		CIFAR10			**CIFAR100**			**ImageNet200**			**ImageNet1K**		
		Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.
- Multi-layer Radial Basis Function Networks													
MLRBFN	DINOv2	96.01$_{(\pm 0.34)}$	97.60$_{(\pm 0.64)}$	97.79$_{(\pm 0.04)}$	89.18$_{(\pm 0.73)}$	93.18$_{(\pm 1.17)}$	85.25$_{(\pm 0.32)}$	90.24$_{(\pm 0.22)}$	96.82$_{(\pm 0.52)}$	94.61$_{(\pm 0.37)}$	78.84$_{(\pm 0.25)}$	92.57$_{(\pm 0.21)}$	78.62$_{(\pm 0.12)}$
	CLIP	95.75$_{(\pm 0.35)}$	98.46$_{(\pm 0.20)}$	95.78$_{(\pm 0.16)}$	86.53$_{(\pm 0.84)}$	95.96$_{(\pm 1.35)}$	76.04$_{(\pm 0.17)}$	85.05$_{(\pm 0.42)}$	94.78$_{(\pm 0.42)}$	90.97$_{(\pm 0.07)}$	72.03$_{(\pm 0.51)}$	88.50$_{(\pm 0.32)}$	72.16$_{(\pm 0.11)}$
- Foundation Model Benchmarks													
MSP	DINOv2	86.74$_{(\pm 0.50)}$	90.30$_{(\pm 1.59)}$	98.13$_{(\pm 0.09)}$	83.85$_{(\pm 0.46)}$	87.28$_{(\pm 0.88)}$	87.62$_{(\pm 0.17)}$	79.09$_{(\pm 0.33)}$	86.25$_{(\pm 0.18)}$	94.95$_{(\pm 0.08)}$	67.80$_{(\pm 0.38)}$	72.99$_{(\pm 0.46)}$	77.94$_{(\pm 0.17)}$
	CLIP	77.67$_{(\pm 0.24)}$	76.56$_{(\pm 1.80)}$	95.92$_{(\pm 0.09)}$	72.85$_{(\pm 0.67)}$	72.08$_{(\pm 1.12)}$	79.14$_{(\pm 0.17)}$	75.55$_{(\pm 0.60)}$	83.51$_{(\pm 0.91)}$	92.73$_{(\pm 0.10)}$	65.04$_{(\pm 0.32)}$	69.84$_{(\pm 0.67)}$	72.21$_{(\pm 0.11)}$
ODIN	DINOv2	96.27$_{(\pm 0.27)}$	97.16$_{(\pm 0.51)}$	98.13$_{(\pm 0.09)}$	85.70$_{(\pm 0.54)}$	89.06$_{(\pm 1.18)}$	87.62$_{(\pm 0.17)}$	81.54$_{(\pm 0.48)}$	84.49$_{(\pm 0.27)}$	94.96$_{(\pm 0.08)}$	55.61$_{(\pm 1.70)}$	49.15$_{(\pm 2.14)}$	78.42$_{(\pm 0.12)}$
	CLIP	83.56$_{(\pm 1.07)}$	74.32$_{(\pm 2.48)}$	95.92$_{(\pm 0.09)}$	64.73$_{(\pm 1.81)}$	53.04$_{(\pm 0.79)}$	79.16$_{(\pm 0.17)}$	69.18$_{(\pm 1.20)}$	73.42$_{(\pm 1.88)}$	92.74$_{(\pm 0.10)}$	61.51$_{(\pm 0.94)}$	63.16$_{(\pm 2.32)}$	72.74$_{(\pm 0.15)}$
GEN	DINOv2	96.98$_{(\pm 0.18)}$	97.85$_{(\pm 0.30)}$	98.13$_{(\pm 0.09)}$	88.43$_{(\pm 0.20)}$	91.73$_{(\pm 0.78)}$	87.62$_{(\pm 0.17)}$	88.03$_{(\pm 0.39)}$	94.06$_{(\pm 0.08)}$	94.95$_{(\pm 0.08)}$	70.17$_{(\pm 0.56)}$	75.17$_{(\pm 0.53)}$	77.94$_{(\pm 0.17)}$
	CLIP	89.19$_{(\pm 0.23)}$	87.01$_{(\pm 1.26)}$	95.92$_{(\pm 0.09)}$	74.49$_{(\pm 0.65)}$	71.62$_{(\pm 0.90)}$	79.14$_{(\pm 0.17)}$	80.68$_{(\pm 0.68)}$	88.12$_{(\pm 0.67)}$	92.73$_{(\pm 0.10)}$	68.50$_{(\pm 0.40)}$	75.33$_{(\pm 1.16)}$	72.21$_{(\pm 0.11)}$
SHEIP	DINOv2	95.24$_{(\pm 0.06)}$	97.49$_{(\pm 0.06)}$	98.13$_{(\pm 0.09)}$	94.06$_{(\pm 0.04)}$	95.49$_{(\pm 0.39)}$	87.62$_{(\pm 0.17)}$	91.78$_{(\pm 0.04)}$	98.30$_{(\pm 0.07)}$	94.95$_{(\pm 0.08)}$	81.78$_{(\pm 0.17)}$	94.83$_{(\pm 0.11)}$	77.94$_{(\pm 0.17)}$
	CLIP	93.53$_{(\pm 0.07)}$	98.95$_{(\pm 0.06)}$	95.92$_{(\pm 0.09)}$	88.74$_{(\pm 0.18)}$	97.44$_{(\pm 0.20)}$	79.14$_{(\pm 0.17)}$	80.15$_{(\pm 0.17)}$	89.78$_{(\pm 0.48)}$	92.73$_{(\pm 0.10)}$	66.77$_{(\pm 0.44)}$	79.56$_{(\pm 0.73)}$	72.21$_{(\pm 0.11)}$
SHEE	DINOv2	97.62$_{(\pm 0.01)}$	98.74$_{(\pm 0.05)}$	98.13$_{(\pm 0.09)}$	94.89$_{(\pm 0.06)}$	96.41$_{(\pm 0.07)}$	87.62$_{(\pm 0.17)}$	92.60$_{(\pm 0.04)}$	97.83$_{(\pm 0.08)}$	94.95$_{(\pm 0.08)}$	83.58$_{(\pm 0.11)}$	94.10$_{(\pm 0.07)}$	77.94$_{(\pm 0.17)}$
	CLIP	94.72$_{(\pm 0.05)}$	99.10$_{(\pm 0.01)}$	95.92$_{(\pm 0.09)}$	89.74$_{(\pm 0.15)}$	98.03$_{(\pm 0.18)}$	79.14$_{(\pm 0.17)}$	85.57$_{(\pm 0.07)}$	93.41$_{(\pm 0.31)}$	92.73$_{(\pm 0.10)}$	73.97$_{(\pm 0.31)}$	85.44$_{(\pm 0.26)}$	72.21$_{(\pm 0.11)}$
KNN	DINOv2	95.57$_{(\pm 0.00)}$	96.06$_{(\pm 0.00)}$	98.13$_{(\pm 0.09)}$	91.44$_{(\pm 0.00)}$	86.57$_{(\pm 0.00)}$	87.62$_{(\pm 0.17)}$	81.51$_{(\pm 0.00)}$	96.20$_{(\pm 0.00)}$	94.95$_{(\pm 0.08)}$	77.53$_{(\pm 0.00)}$	94.65$_{(\pm 0.00)}$	77.94$_{(\pm 0.17)}$
	CLIP	90.71$_{(\pm 0.00)}$	97.82$_{(\pm 0.00)}$	95.92$_{(\pm 0.09)}$	82.14$_{(\pm 0.00)}$	91.39$_{(\pm 0.00)}$	79.14$_{(\pm 0.17)}$	68.36$_{(\pm 0.00)}$	72.28$_{(\pm 0.00)}$	92.73$_{(\pm 0.10)}$	57.80$_{(\pm 0.00)}$	61.12$_{(\pm 0.00)}$	72.21$_{(\pm 0.11)}$
RMDS	DINOv2	98.27$_{(\pm 0.00)}$	99.19$_{(\pm 0.00)}$	98.13$_{(\pm 0.09)}$	95.16$_{(\pm 0.00)}$	97.53$_{(\pm 0.00)}$	87.62$_{(\pm 0.17)}$	94.77$_{(\pm 0.00)}$	98.89$_{(\pm 0.00)}$	94.95$_{(\pm 0.08)}$	83.79$_{(\pm 0.00)}$	96.31$_{(\pm 0.00)}$	77.94$_{(\pm 0.17)}$
	CLIP	95.73$_{(\pm 0.00)}$	97.81$_{(\pm 0.00)}$	95.92$_{(\pm 0.09)}$	93.23$_{(\pm 0.00)}$	98.81$_{(\pm 0.00)}$	79.14$_{(\pm 0.17)}$	92.09$_{(\pm 0.00)}$	97.88$_{(\pm 0.00)}$	92.73$_{(\pm 0.10)}$	78.21$_{(\pm 0.00)}$	91.50$_{(\pm 0.00)}$	72.21$_{(\pm 0.11)}$

Table 3. Comparing the performance of MLRBFNs to published benchmarks and existing methods. In this table, we use AUPR Out as the metric for OOD detection. Whenever applicable, we report the average number and the corresponding standard deviation obtained from 3 training runs.

		CIFAR10			CIFAR100			ImageNet200			ImageNet1K		
		Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.
- Multi-layer Radial Basis Function Networks													
MLRBFN	DINOv2	95.77 (±0.41)	98.20 (±0.60)	97.79 (±0.04)	86.47 (±1.13)	93.80 (±1.23)	85.25 (±0.32)	93.50 (±0.17)	95.83 (±0.62)	94.61 (±0.37)	58.82 (±1.07)	69.61 (±0.62)	78.62 (±0.12)
	CLIP	95.27 (±0.51)	98.65 (±0.30)	95.78 (±0.16)	83.36 (±1.41)	95.92 (±1.70)	76.04 (±0.17)	88.56 (±0.24)	93.55 (±0.76)	90.97 (±0.07)	47.53 (±0.91)	61.72 (±1.21)	72.16 (±0.11)
- Foundation Model Benchmarks													
MSP	DINOv2	91.17 (±0.35)	94.80 (±0.68)	98.13 (±0.09)	78.91 (±0.65)	87.86 (±0.87)	87.62 (±0.17)	88.75 (±0.23)	88.85 (±0.09)	94.95 (±0.08)	43.90 (±0.31)	34.34 (±0.45)	77.94 (±0.17)
	CLIP	82.40 (±0.15)	85.64 (±0.97)	95.92 (±0.09)	65.07 (±0.48)	75.14 (±0.76)	79.14 (±0.17)	83.38 (±0.29)	83.36 (±0.51)	92.73 (±0.10)	40.26 (±0.24)	29.96 (±0.54)	72.21 (±0.11)
ODIN	DINOv2	95.24 (±0.48)	96.94 (±0.68)	98.13 (±0.09)	80.47 (±0.54)	88.82 (±1.41)	87.62 (±0.17)	84.27 (±0.16)	79.72 (±0.46)	94.96 (±0.08)	33.94 (±0.89)	16.96 (±0.69)	78.42 (±0.12)
	CLIP	80.52 (±1.62)	77.14 (±2.07)	95.92 (±0.09)	57.75 (±1.48)	62.89 (±0.44)	79.16 (±0.17)	73.26 (±0.85)	66.92 (±1.71)	92.74 (±0.10)	36.44 (±0.97)	23.54 (±1.62)	72.74 (±0.15)
GEN	DINOv2	96.30 (±0.24)	97.93 (±0.33)	98.13 (±0.09)	84.12 (±0.32)	91.88 (±0.94)	87.62 (±0.17)	91.45 (±0.14)	92.34 (±0.12)	94.95 (±0.08)	43.91 (±0.81)	33.94 (±0.77)	77.94 (±0.17)
	CLIP	87.69 (±0.20)	88.66 (±1.00)	95.92 (±0.09)	64.77 (±0.51)	73.29 (±0.48)	79.14 (±0.17)	84.16 (±0.57)	85.00 (±1.00)	92.73 (±0.10)	42.15 (±0.45)	35.34 (±1.83)	72.21 (±0.11)
SHEIP	DINOv2	95.01 (±0.08)	98.20 (±0.09)	98.13 (±0.09)	93.60 (±0.08)	97.05 (±0.28)	87.62 (±0.17)	94.45 (±0.08)	97.71 (±0.18)	94.95 (±0.08)	63.19 (±0.55)	76.19 (±0.40)	77.94 (±0.17)
	CLIP	93.08 (±0.07)	98.69 (±0.05)	95.92 (±0.09)	87.61 (±0.23)	97.50 (±0.24)	79.14 (±0.17)	81.93 (±0.28)	85.46 (±0.62)	92.73 (±0.10)	41.70 (±0.37)	37.70 (±1.12)	72.21 (±0.11)
SHEE	DINOv2	97.65 (±0.02)	99.30 (±0.05)	98.13 (±0.09)	94.06 (±0.09)	97.30 (±0.06)	87.62 (±0.17)	94.44 (±0.05)	96.62 (±0.20)	94.95 (±0.08)	64.35 (±0.53)	70.89 (±0.62)	77.94 (±0.17)
	CLIP	94.11 (±0.06)	98.76 (±0.03)	95.92 (±0.09)	88.66 (±0.24)	97.64 (±0.28)	79.14 (±0.17)	86.44 (±0.26)	90.38 (±0.46)	92.73 (±0.10)	47.25 (±0.39)	46.42 (±0.73)	72.21 (±0.11)
KNN	DINOv2	95.37 (±0.00)	97.76 (±0.00)	98.13 (±0.09)	91.23 (±0.00)	91.47 (±0.00)	87.62 (±0.17)	86.08 (±0.00)	96.14 (±0.00)	94.95 (±0.08)	60.15 (±0.00)	80.16 (±0.00)	77.94 (±0.17)
	CLIP	90.51 (±0.00)	97.64 (±0.00)	95.92 (±0.09)	82.59 (±0.00)	91.52 (±0.00)	79.14 (±0.17)	69.15 (±0.00)	61.81 (±0.00)	92.73 (±0.10)	35.04 (±0.00)	21.54 (±0.00)	72.21 (±0.11)
RMDS	DINOv2	97.89 (±0.00)	99.17 (±0.00)	98.13 (±0.09)	94.14 (±0.00)	98.29 (±0.00)	87.62 (±0.17)	97.16 (±0.00)	98.63 (±0.00)	94.95 (±0.08)	74.46 (±0.00)	85.21 (±0.00)	77.94 (±0.17)
	CLIP	94.79 (±0.00)	97.08 (±0.00)	95.92 (±0.09)	92.57 (±0.00)	98.91 (±0.00)	79.14 (±0.17)	94.45 (±0.00)	97.27 (±0.00)	92.73 (±0.10)	55.24 (±0.00)	63.57 (±0.00)	72.21 (±0.11)

Table 4. Comparing the performance of MLRBFNs to published benchmarks and existing methods. In this table, we use FPR@95 (False Positive Rate at 95% True Positive Rate) as the metric for OOD detection. As such, unlike the previous tables, smaller numbers indicate better performance here. Whenever applicable, we report the average number and the corresponding standard deviation obtained from 3 training runs.

		CIFAR10			CIFAR100			ImageNet200			ImageNet1K		
		Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.	Near-OOD	Far-OOD	ID Acc.
- Multi-layer Radial Basis Function Networks													
MLRBFN	DINOv2	17.99 (±2.08)	9.51 (±2.08)	97.79 (±0.04)	43.39 (±2.41)	25.31 (±4.31)	85.25 (±0.32)	49.55 (±1.07)	11.71 (±2.13)	94.61 (±0.37)	67.08 (±1.03)	26.92 (±1.48)	78.62 (±0.12)
	CLIP	16.81 (±1.78)	6.36 (±1.30)	95.78 (±0.16)	46.16 (±1.87)	14.21 (±3.57)	76.04 (±0.17)	60.18 (±1.89)	22.03 (±2.23)	90.97 (±0.07)	76.68 (±1.06)	45.61 (±2.35)	72.16 (±0.11)
- Foundation Model Benchmarks													
MSP	DINOv2	100.00 (±0.00)	100.00 (±0.00)	98.13 (±0.09)	100.00 (±0.00)	45.78 (±1.41)	87.62 (±0.17)	100.00 (±0.00)	100.00 (±0.00)	94.95 (±0.08)	100.00 (±0.00)	100.00 (±0.00)	77.94 (±0.17)
	CLIP	100.00 (±0.00)	100.00 (±0.00)	95.92 (±0.09)	100.00 (±0.00)	77.12 (±7.57)	79.14 (±0.17)	100.00 (±0.00)	100.00 (±0.00)	92.73 (±0.10)	100.00 (±0.00)	100.00 (±0.00)	72.21 (±0.11)
ODIN	DINOv2	14.11 (±0.87)	10.78 (±1.71)	98.13 (±0.09)	49.59 (±2.62)	33.50 (±2.68)	87.62 (±0.17)	62.22 (±2.24)	47.14 (±1.25)	94.96 (±0.08)	87.52 (±0.61)	88.99 (±1.36)	78.42 (±0.12)
	CLIP	55.69 (±2.25)	65.67 (±5.08)	95.92 (±0.09)	86.48 (±3.18)	75.28 (±1.54)	79.16 (±0.17)	77.12 (±1.43)	67.04 (±2.91)	92.74 (±0.10)	82.37 (±1.23)	78.30 (±1.81)	72.74 (±0.15)
GEN	DINOv2	11.56 (±0.89)	8.32 (±1.16)	98.13 (±0.09)	42.56 (±0.82)	27.86 (±2.36)	87.62 (±0.17)	54.84 (±3.23)	24.37 (±0.28)	94.95 (±0.08)	76.21 (±1.17)	67.28 (±0.99)	77.94 (±0.17)
	CLIP	46.15 (±1.00)	53.18 (±5.71)	95.92 (±0.09)	67.44 (±2.49)	65.24 (±2.82)	79.14 (±0.17)	66.89 (±1.49)	44.34 (±2.10)	92.73 (±0.10)	76.47 (±0.81)	67.22 (±1.64)	72.21 (±0.11)
SHEIP	DINOv2	18.00 (±0.08)	7.85 (±0.18)	98.13 (±0.09)	24.63 (±0.25)	14.80 (±0.71)	87.62 (±0.17)	43.91 (±0.28)	5.70 (±0.18)	94.95 (±0.08)	65.63 (±0.46)	20.64 (±0.88)	77.94 (±0.17)
	CLIP	23.36 (±0.15)	4.52 (±0.20)	95.92 (±0.09)	32.41 (±0.51)	10.30 (±0.83)	79.14 (±0.17)	60.18 (±0.39)	32.65 (±0.81)	92.73 (±0.10)	80.41 (±0.52)	55.66 (±1.29)	72.21 (±0.11)
SHEE	DINOv2	10.09 (±0.06)	4.22 (±0.13)	98.13 (±0.09)	21.16 (±0.04)	11.03 (±0.27)	87.62 (±0.17)	40.78 (±0.13)	6.19 (±0.15)	94.95 (±0.08)	59.54 (±0.35)	20.29 (±0.38)	77.94 (±0.17)
	CLIP	18.93 (±0.13)	3.62 (±0.03)	95.92 (±0.09)	31.17 (±0.37)	6.79 (±0.52)	79.14 (±0.17)	50.88 (±0.13)	25.04 (±0.58)	92.73 (±0.10)	69.66 (±0.21)	42.95 (±0.49)	72.21 (±0.11)
KNN	DINOv2	15.25 (±0.00)	10.92 (±0.00)	98.13 (±0.09)	33.12 (±0.00)	27.77 (±0.00)	87.62 (±0.17)	66.97 (±0.00)	19.31 (±0.00)	94.95 (±0.08)	71.93 (±0.00)	23.96 (±0.00)	77.94 (±0.17)
	CLIP-KNN	29.77 (±0.00)	8.64 (±0.00)	95.92 (±0.09)	44.97 (±0.00)	23.89 (±0.00)	79.14 (±0.17)	70.37 (±0.00)	47.78 (±0.00)	92.73 (±0.10)	83.28 (±0.00)	63.57 (±0.00)	72.21 (±0.11)
RMDS	DINOv2	6.41 (±0.00)	2.72 (±0.00)	98.13 (±0.09)	20.59 (±0.00)	10.51 (±0.00)	87.62 (±0.17)	33.93 (±0.00)	3.90 (±0.00)	94.95 (±0.08)	64.98 (±0.00)	17.07 (±0.00)	77.94 (±0.17)
	CLIP	17.33 (±0.00)	8.50 (±0.00)	95.92 (±0.09)	25.61 (±0.00)	4.25 (±0.00)	79.14 (±0.17)	38.51 (±0.00)	8.46 (±0.00)	92.73 (±0.10)	66.41 (±0.00)	30.62 (±0.00)	72.21 (±0.11)

9. Code

In this section, we provide code for RBF layers of an MLRBFN. Additionally, we provide the code for our binary cross-entropy loss function.

```python
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn.modules.lazy import _LazyProtocol

def logsubstractexp(tensor, other):
    a = torch.max(tensor, other)
    return a + ((tensor - a).exp() - (other - a).exp()).log()

def log_bce_loss(log_y_pred, y, num_classes):
    y = F.one_hot(y, num_classes=num_classes).float()
    y_flat = y.ravel()
    log_y_pred_flat = log_y_pred.ravel() - 1e-6
    zero = torch.zeros(log_y_pred_flat.shape).to(y.device)

    not_log_y_pred = logsubstractexp(zero, log_y_pred_flat)
    return torch.mean(-1 * ((y_flat * log_y_pred_flat) + (1 - y_flat) * not_log_y_pred))

def webscale_kmeans(data, centroid_shape, k, iter=1):
    v = torch.zeros(centroid_shape[0]).to(data.device)
    d = torch.zeros(data.shape[0]).int().to(data.device)

    centroids = torch.empty(centroid_shape).to(data.device)
    centroids[0, :] = data[0]
    for c in range(1, centroids.shape[0]):
        dists = torch.cdist(data, centroids[:c], p=k) ** k
        dists = torch.min(dists, -1).values
        dists = dists / torch.sum(dists)

        rand_unif = torch.rand(1).to(data.device)
        dists_cumsum = torch.cumsum(dists, 0)
        idx = torch.argmax((dists_cumsum > rand_unif).to(torch.long))
        centroids[c, :] = data[idx, :] + 1e-5 * torch.randn(data.shape[1]).to(data.device)

    for t in range(iter):
        for i, x in enumerate(data):
            x = x[None, :]
            dists = torch.linalg.norm(centroids - x, ord=k, dim=1) ** k
            idx = torch.argmin(dists)
            d[i] = idx

        for i, x in enumerate(data):
            c = d[i]
            v[c] += 1
            eta = 1 / v[c]
            centroids[c] = (1 - eta) * centroids[c] + eta * x

    return centroids

class RBFDepressionLayer(nn.Module):
    def __init__(self, input_features, num_centroids, projection_features, k=2, last=False):
```

```
        super().__init__()
        self.centroids = nn.Parameter(torch.randn(num_centroids, input_features))
        self.init_beta = 1
        self.beta = nn.Parameter(torch.ones(num_centroids,))
        self.projections = nn.Parameter(torch.randn(num_centroids, projection_features))
        self.k = k

        self.last = last

    def forward(self, x, depression, recovery=1.1):
        distances = torch.cdist(x, self.centroids, p=self.k) ** self.k
        sb = F.softplus(self.beta)

        if self.last == False:
            scales = (sb / F.softplus(self.init_beta)) * torch.exp(-sb * distances)
            depressed_scales = scales * \
                torch.minimum(depression * recovery, torch.tensor(1.0).to(x.device))[:, None]
            depress_next = torch.max(depressed_scales, 1).values
            x = depressed_scales @ self.projections
            return x, depress_next, scales
        else:
            log_scales = -sb * distances
            log_recovery = torch.log(torch.tensor(recovery).to(x.device))
            log_depression = torch.log(depression)
            x = log_scales + \
                torch.minimum(log_depression + log_recovery, torch.tensor(0.0).to(x.device))[:, None]
            return x, log_depression, log_scales

class KMeansRBFDepressionLayer(nn.modules.lazy.LazyModuleMixin, RBFDepressionLayer):
    cls_to_become: RBFDepressionLayer
    centroids: nn.parameter.UninitializedParameter
    beta: nn.parameter.UninitializedParameter

    def __init__(self, input_features, num_centroids, projection_features, k=2, last=False):
        super().__init__(input_features, num_centroids, projection_features, k, last)
        self.centroid_shape = self.centroids.shape
        self.beta_shape = self.beta.shape

        self.centroids = nn.parameter.UninitializedParameter()
        self.beta = nn.parameter.UninitializedParameter()

    def initialize_parameters(self, x, depression, recovery=1.1):
        self.centroids.materialize(self.centroid_shape)
        self.beta.materialize(self.beta_shape)

        x_centroid_loc = x[:x.shape[0]//2, :]
        x_beta = x[x.shape[0]//2:, :]

        cents = webscale_kmeans(x_centroid_loc, self.centroid_shape, self.k, iter=100)
        self.centroids[:] = cents

        distances = torch.cdist(x_beta, self.centroids, p=self.k) ** self.k
        distances_dp = torch.quantile(torch.min(distances, 1).values, 0.95)
        distances_cent = torch.quantile(torch.min(distances, 0).values, 0.95)
        dist = torch.maximum(distances_dp, distances_cent)

```

```
        if ((4 / dist)) < 5:
            bs = ((4 / dist)).expm1().clamp_min(1e-6).clamp_max(1e4).log()
        else:
            bs = (4 / dist)
        self.init_beta = bs
        self.beta[:] = bs
```