# Constraining Low-level Representations to Define Effective Confidence Scores

**João Monteiro, Pau Rodríguez, Pierre-André Noël, Issam Laradji, David Vázquez**
ServiceNow Research
`{FirstName.LastName}@servicenow.com`

## Abstract

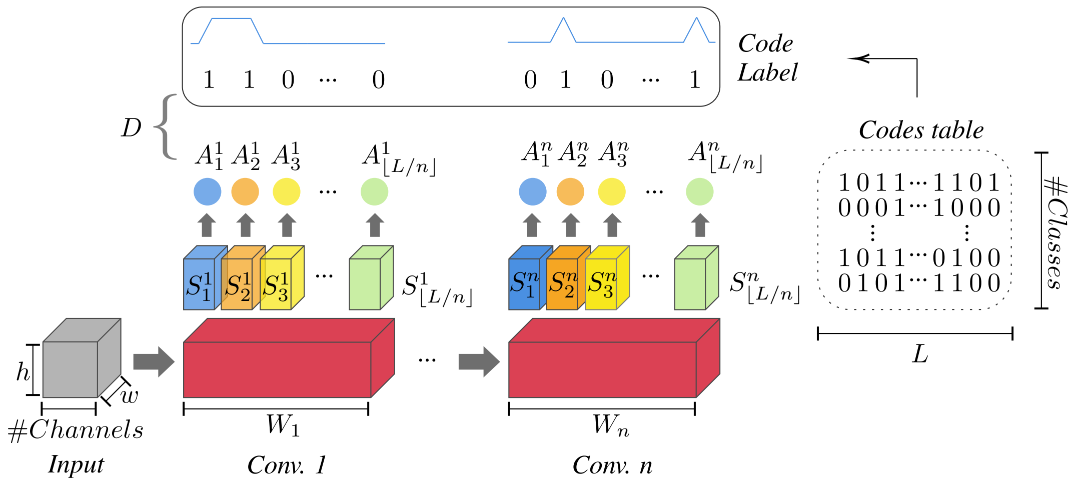
Neural networks are known to fail with high confidence, especially for data that somehow differs from the training distribution. Such an unsafe behaviour limits their applicability. To counter that, we show that models offering accurate confidence levels can be defined via adding constraints in their internal representations. To do so, we encode class labels as fixed unique binary vectors, or *class codes*, and use those to enforce class-dependent activation patterns throughout the model's depth. Resulting predictors are dubbed total activation classifiers (TAC), and TAC is used as an additional component to a base classifier to indicate how reliable a prediction is. Empirically, we show that the resemblance between activation patterns and their corresponding codes results in an inexpensive unsupervised approach for inducing discriminative confidence scores. Namely, we show that TAC is at least as good as state-of-the-art confidence scores extracted from existing models, while strictly improving the model's *value* on the rejection setting.

## 1 Introduction

Recent work has shown that commonly used model classes and learning algorithms yield simple class-dependent patterns in their learned representations [14, 9], i.e., specific groups of representations activate more or less strongly depending on high-level features of inputs. This behaviour can be useful to define predictors able to reject/defer test data that do not follow common patterns, provided that one can efficiently verify similarities between new data and common patterns.

Rather than hoping this property emerges, we seek to build it into models. To do so, we introduce total activation classifiers (TAC): a component that can be attached to any kind of multi-layer classifiers. TAC obtains *activation profiles* via slicing and reducing (*e.g.*, summing or averaging) the activations of a stack of layers. In addition, we turn the label set into a set of hard-coded class-dependent *codes* [4, 6, 16]. During training, activation profiles are then encouraged to match the patterns defined by the codes. In other words, class codes define valid internal configurations of activations by indicating which groups of features should be strongly activated for a given class. At testing time, TAC decides on the best match between an observed activation profile and class codes, and distances can be used as confidence scores to detect low-confidence, likely erroneous predictions.

Intuitively, the motivation for constraining internal representations is two-fold: **1.** Given data, we can measure how close to a valid pattern the activations of a model are, and finally use such a measure as a confidence score. That is, if the model is far from a valid activation pattern, then its prediction should be deemed unreliable. **2.** Tying internal representations with the labels adds constraints to attackers. For a standard classifier, an attacker's only job is to make it so that any output unit fires up more strongly than the correct one, and any intermediate configuration of outputs that satisfy that condition are valid perturbations. In our proposal, an attack is only valid if the entire set of activations matches the pattern of the wrong class.



1 1 0 ··· 0
0 1 0 ··· 1
Code
Label
D
Codes table
1011···1101
0001···1000
1011···0100
0101···1100
#Classes
L
$A_1^1$ $A_2^1$ $A_3^1$ $A_{\lfloor L/n \rfloor}^1$
$A_1^n$ $A_2^n$ $A_3^n$ $A_{\lfloor L/n \rfloor}^n$
$S_1^1$ $S_2^1$ $S_3^1$ $S_{\lfloor L/n \rfloor}^1$
$S_1^n$ $S_2^n$ $S_3^n$ $S_{\lfloor L/n \rfloor}^n$
h
w
#Channels
Input
$W_1$
Conv. 1
$W_n$
Conv. n

Figure 1: Illustration of a Total Activation Classifier (TAC). Without loss of generality, we consider the case where $a_w$ is 2-dimensional. The set of codes define valid activation patterns.

## 2 total activation classifiers (TAC)

### 2.1 Model definition

We will consider classes of predictors of the form $f' \in \mathcal{F}' : \mathcal{X} \mapsto [0,1]^L$ with $L \gg K$, *i.e.*, models that map data onto the unit cube in $\mathbb{R}^L$, which we define by leveraging the set of $n$-layered neural networks. The outputs of layer $i$ within any chosen $f' \in \mathcal{F}'$ are represented by $a_w^i$, $w \in [1, 2, 3, ..., W_i]$, where $W_i$ indicates the width of layer $i$ given by its number of output features. We then partition the set of representations $a_w^i$ into disjoint subsets (or *slices*) of uniform sizes, denoted by $S_l^i$, $l \in \{1, 2, ...\lfloor L/n \rfloor\}$. Intuitively, our goal is to make it so that groups of high-level features "fire up" more strongly depending on the underlying class of the input. To enforce that property, we consider the sequence of total activations of slices. That is, for the feature slice $S_l^i$ on layer $i$, we will consider its total activation

$$A_l^i(f', x) = \sum_{a_w^i \in S_l^i} \sum_{h', h''} a_{w,h',h''}^i, \tag{1}$$

where features are arbitrarily considered as 2-dimensional objects ($h'$ and $h''$ are, for example, spatial dimensions in convolutional architectures for images). Note that total activations are defined similarly for features of any dimension by simply reducing away the extra dimensions. The sequence of total activations obtained from slices across all layers, denoted $A(f', x)$, will be referred to as the activation profile of $x$. An illustration is provided in Figure 1, where a generic convolutional model has the outputs of its layers partitioned into slices $S_l^i$ which are then reduced to yield the sequence $A(f', x)$, the activation profile of $x$ as induced by $f'$. Figure 10 in the Appendix shows a Pytorch [15] implementation of feature slicing and activation profile computation. Given a stack of layers yielding activation profiles of dimension $L$, a set of class codes of dimension $L$, and a distance $D$, we define total activation classifiers (TAC) such that they make the prediction

$$y' := \arg\min_{i \in \mathcal{Y}} D(A(f', x), C_i) \text{ with associated confidence } -D(A(f', x), C_{y'}). \tag{2}$$

### 2.2 Training

Training is carried out to search for $f' \in \mathcal{F}'$ with activation profiles close to the correct code. To enforce that behaviour, we minimize the training objective

$$\mathcal{L}_{bin} = -\frac{1}{NL} \sum_{i=1}^{N} C_{y_i}^\top \log \sigma\big(A(f', x_i)\big) + (1 - C_{y_i})^\top \log\big(1 - \sigma\big(A(f', x_i)\big)\big), \tag{3}$$

where $N$ is the size of the batch of data used to compute the objective, $L$ is the length of the code and activation profile, and $\sigma$ is the sigmoid function. $\mathcal{L}_{bin}$ computes the binary cross-entropy loss for each dimension of the code and is minimized for a perfect match between activation profiles and the correct codes. While minimizers of $\mathcal{L}_{bin}$ yield the properties we need into a model, we empirically observed that training against $\mathcal{L}_{bin}$ is not trivial in that it requires the use of aggressive learning rates, rendering training unstable, especially so for large (*i.e.*, $\geq 1000$) label sets $\mathcal{Y}$. To overcome that issue, we define a second training objective by parameterizing a conditional categorical distribution over the label set using the distances between a given activation profile and all the codes and performing maximum likelihood estimation. More formally, the alternative training objective is

$$\mathcal{L}_{ce} = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{-D(A(f',x_i),C_{y_i})/\tau}}{\sum_{k=1}^{K} e^{-D(A(f',x_i),C_k)/\tau}}, \tag{4}$$

where $\tau$ is a scaling parameter.

While both $\mathcal{L}_{bin}$ and $\mathcal{L}_{ce}$ are reasonable choices for the training of TAC, each presents challenges. As mentioned above, $\mathcal{L}_{bin}$ is such that it yields unstable training but we found that, in the cases where one can minimize it, it produces models able to match codes very closely, and as such provide discriminative confidence scores. On the other hand, training $\mathcal{L}_{ce}$ is less unstable in that more common hyperparameter configurations work out of the box. However, minimizers of $\mathcal{L}_{ce}$ are not as good at matching vertices tightly since the objective only requires activation profiles to lie closer to the correct code than they are from other codes and, in that case, using distances as confidence scores is not as effective. We thus consider the linear combination of these training objectives $\mathcal{L} = \alpha\mathcal{L}_{bin} + \beta\mathcal{L}_{ce}$, where $\alpha$ and $\beta$ are hyperparameters. This choice takes advantage of the relative easiness of training against $\mathcal{L}_{ce}$ and the pressure of $\mathcal{L}_{bin}$ towards solutions able to match codes more closely. We remark that both objectives share minima, so minimizing their sum does not introduce any real trade-off.

## 3 Evaluation

**Attaching TAC to pre-trained models:** Provided that one has their ready-to-use classifier, a TAC can be added on top of its features, and the base classifier is preserved as-is (*i.e.*, "frozen"). To do so, we isolate TAC from the base predictor via trainable projection layers applied in the representations of the base classifier. Then, we apply slice/reduce operations on those projections to obtain activation profiles. Projections correspond to independent MLP networks, one for each layer of the base classifier. This approach is intended to simplify TAC training and enable its use in more practical scale, but it also enables the combined use of confidence scores obtained by TAC and by the base classifier such as the maximum softmax probability (MSP) or the maximum logit score (MLS) [18].

**Baselines:** For a fair comparison, we consider approaches that do not require access to out-of-distribution data. Directly using statistics of the output layer of classifiers was observed recently to yield state-of-the-art performance on different OOD detection tasks [18]. We thus consider MLS [18], MPS [8], and DOCTOR [7], all obtained from the base classifier, as our main performance targets.

**Rejecting classifiers:** We evaluate TAC as a *rejecting* classifier, *i.e.*, it can abstain if not sufficiently confident in its predictions. We leverage the framework proposed by Casati et al. [2], which defines the application-specific *error cost* $\omega$ capturing the ratio of how much we dislike accepting incorrect classifications over how much we like accepting correct ones. This framework then proposes to evaluate predictors in terms of their *value* $\mathcal{V}$ as a function of $\omega$: $\mathcal{V} = \frac{N_c - \omega N_i}{N}$, where $N_c$ and $N_i$ correspond to the number of correct and incorrect error detection predictions made by a model over a sample of size $N = N_c + N_i + N_r$, for $N_r$ rejections. In Figure 2, we plot the Value Operating Characteristic (VOC, cf. definition in [2]) curves for TAC, MLS, and MSP. The base predictor corresponds to a ViT [5] (`Base-16x16`) we pre-trained, and TAC operates in 13 different layers. We perform $k$-fold ($k = 5$) splits on the test set of ImageNet and, for a given split and value of $\omega$, we use the largest part of the data to select the confidence rejection threshold that maximizes $\mathcal{V}$. Curves averaged over splits are plotted for the data used for threshold selection (indicated as *train* in the plot) as well as for the left-out splits. One can then note that TAC's scores yield the highest value $\mathcal{V}$ throughout a broad range of $\omega$, if not all of it. Additionally, the performance of detection scores for *detecting prediction errors* is reported in Table 1. For the test sets of both HWU64 and CLINC150, and also ImageNet [3] in this case, we compare different scores when trying to detect whether the base model made an error. TAC yields a higher detection rate in all cases.

| Method | Det. AUROC (%) | Det. Rate (%) |
|---|---|---|
| **HWU64** | | |
| MPS [8] | **89.39** | 81.43 |
| DOCTOR [7] | **89.39** | 81.43 |
| MLS [18] | 89.36 | 80.12 |
| TAC ($L_1$) | 89.25 | **83.25** |
| **CLINC150** | | |
| MPS [8] | 90.56 | 82.77 |
| DOCTOR [7] | 90.56 | 82.96 |
| MLS [18] | 91.36 | 82.66 |
| TAC ($L_1$) | **93.25** | **85.70** |
| **ImageNet** | | |
| MPS [8] | 80.74 | 73.63 |
| DOCTOR [7] | 80.91 | 73.69 |
| MLS [18] | 53.69 | 52.75 |
| TAC ($Cosine$) | **89.41** | **81.64** |

Table 1: Detection of prediction errors. Detection rates are measured at the threshold where true positive and false negative rates match.

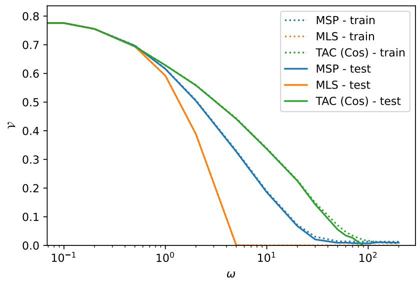


Figure 2: VOC curves for different predictors on ImageNet as a function of $\omega$, the application-specific *error cost* characterizing how desirable it is reject incorrect classifications. The overall value $\mathcal{V}$ of TAC-based predictors dominates for a broad range of $\omega$.

**Out-of-distribution detection:** As a safe policy, a classifier should be such that for an unseen class it would abstain from predicting. We test this behavior with TAC by using distances to decide when not to predict. Performance is evaluated on the CLINC150 benchmark. In this case, the training sample is such that an `UNKNOWN` class is used to indicate data not represented in the remainder of the label set. We train our models on all classes except `UNKNOWN`, but try to detect unknown classes at testing time. Detection performance is reported in Table 2. We compare with MLS [18] and the approach introduced by Shao et al. [17] where a Gaussian mixture model (GMM) is trained on top of representations, and the likelihood of test instances according to the generative model is used as a detection score. TAC can improve detection performance, especially so in terms of detection rate. In Figure 3, we report the detection performance obtained when different distances are used on top of TAC. Performance seems consistent across possibilities except for the case of $L_\infty$, which indicates that there is often some dimension with a loose match between activation and code, no matter whether data is in- or out-of-distribution.

| Method | Det. AUROC (%) | Det. Rate (%) |
|---|---|---|
| GMM [17] | 96.55 | 91.51 |
| MLS [18] | 97.00 | 91.78 |
| TAC ($L_1$) | **97.28** | **92.82** |

Table 2: Comparison of different scoring strategies used for detection of examples from classes unseen during training. The detection rate measures the fraction of correctly detected attackers at the threshold where true positive and false negative rates match. For TAC, we report results considering the best performing distances, $L_1$ in this case.

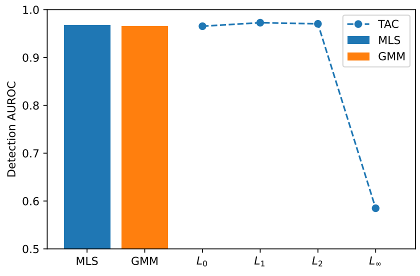


Figure 3: Detection of out-of-sample test instances in terms of AUROC.

## 4 Conclusion

We introduced total activation classifiers (TAC): a model component that can be attached to existing predictors to provide scores indicating how likely it is that predictions are incorrect. Our empirical evaluation showed that distances, as measured between activation profiles and class codes, defined effective detection scores of problematic predictions. Notably, we trained TAC on top of models as large as variations of transformer architectures such as BERT for text and ViT for images, and observed resulting scores to yield much more effective rejecting classifiers than the base models TAC builds upon. We also showed that the TAC's scores can be used to detect data from unseen classes.

## References

[1] I. Casanueva, T. Temčinas, D. Gerz, M. Henderson, and I. Vulić. Efficient intent detection with dual sentence encoders. *arXiv preprint arXiv:2003.04807*, 2020.

[2] F. Casati, P.-A. Noël, and J. Yang. On the value of ml models. *arXiv preprint arXiv:2112.06775*, 2021.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, 2009.

[4] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286, 1994.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[6] N. García-Pedrajas and C. Fyfe. Evolving output codes for multiclass problems. *IEEE Transactions on Evolutionary Computation*, 12(1):93–106, 2008.

[7] F. Granese, M. Romanelli, D. Gorla, C. Palamidessi, and P. Piantanida. Doctor: A simple method for detecting misclassification errors. *Advances in Neural Information Processing Systems*, 34:5669–5681, 2021.

[8] D. Hendrycks and K. Gimpel. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *arXiv preprint arXiv:1610.02136*, 2016.

[9] N. M. Kalibhat, K. Narang, H. Firooz, M. Sanjabi, and S. Feizi. Towards better understanding of self-supervised representations. In *ICML 2022: Workshop on Spurious Correlations, Invariance and Stability*, 2022.

[10] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. *Citeseer*, 2009.

[11] S. Larson, A. Mahendran, J. J. Peper, C. Clarke, A. Lee, P. Hill, J. K. Kummerfeld, K. Leach, M. A. Laurenzano, L. Tang, and J. Mars. An evaluation dataset for intent classification and out-of-scope prediction. In *Empirical Methods in Natural Language Processing and International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019.

[12] X. Liu, A. Eshghi, P. Swietojanski, and V. Rieser. Benchmarking natural language understanding services for building conversational agents. In *Increasing Naturalness and Flexibility in Spoken Dialogue Interaction*. Springer, 2021.

[13] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[14] N. Papernot and P. McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 2019.

[16] P. Rodríguez, M. A. Bautista, J. Gonzàlez, and S. Escalera. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31, 2018.

[17] L. Shao, Y. Song, and S. Ermon. Understanding classifier mistakes with generative models. *arXiv preprint arXiv:2010.02364*, 2020.

[18] S. Vaze, K. Han, A. Vedaldi, and A. Zisserman. Open-set recognition: A good closed-set classifier is all you need. In *International Conference on Learning Representations*, 2022.

[19] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.

## A Additional results

### A.1 Proof-of-concept

To test for whether commonly used models are able to match activation patterns given by class codes, we train a TAC'ed WideResNet-28-10 [13] on CIFAR-10 [10], starting from random weights. TAC's slice/reduce operations are applied in three layers that output 160, 320, and 640 2-dimension feature maps. We use 16 slices in each layer so that the resulting code length $L$ is 48. Upon training to convergence, we inspect the representations obtained from a random draw of the test set, as displayed in Figure 4. Each column in the figure contains information about a different layer (layer depth grows from left to right). In the first row, we plot the raw activations after average pooling spatial dimensions. The activation profile $A$ is displayed in the second row, where one can see a tight match with the ground-truth codes, as displayed in the third row. The differences between $A$ and code are shown in the bottom row.

We further and test how good of a confidence score one can get by measuring some distance between $A$ and code. In Figure 5, we plot histograms of $L_1$ distances for clean data and adversarial perturbations of the test set of CIFAR-10. Attackers correspond to subtle PGD perturbations [13] obtained under a $L_\infty$ budget of $\frac{8}{255}$. We consider the white-box access model in which the attacker has full access to the target predictor and the table of codes. Attackers are created so that the activation profile $A$ moves towards the code of a wrong class. Attackers fail in matching codes as tightly as clean data. TAC can then spot attackers and defer low-confidence predictions to a human evaluator.

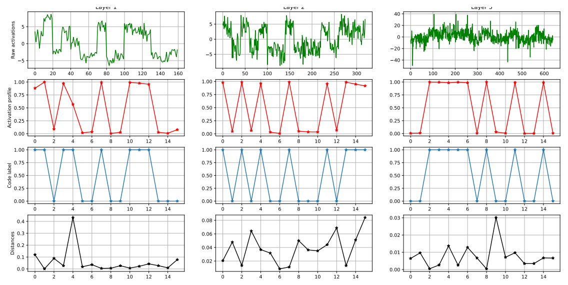


Figure 4: Activations of a TAC'ed Wide-ResNet-28-10 trained on CIFAR-10.

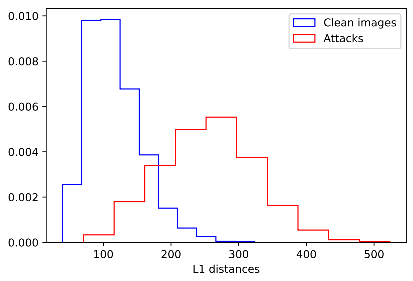


Figure 5: Distance histograms for attacks and natural images on CIFAR-10.

### A.2 Further results on training TAC from scratch

In Table 3, we report clean prediction accuracy and detection performance for adversarial perturbations. In this case, we compare standard classifiers with their corresponding TAC considering both MNIST and CIFAR-10. PGD adversarial perturbations are obtained with [13] attacks under $L_\infty$ budgets of 0.3 and $\frac{8}{255}$ for MNIST and CIFAR-10, respectively. The same WideResNet-28-10 discussed in Section A.1 is used for CIFAR-10, while a 4-layered convolutional model is used for MNIST. We evaluated different detection scores corresponding $L_0$, $L_1$, $L_2$, and $L_\infty$ distances measured between $A$ and codes, and reported the one that performed the best in each case ($L_\infty$ and $L_1$ for MNIST and CIFAR-10, respectively). We evaluated both MSP and MLS for the base classifiers and reported the best performer. TAC improves the detection performance at the cost of a slight drop in prediction accuracy. We investigate the drop in accuracy by testing the *capacity* of the model after the TAC operations are included. To do that, we train models to overfit randomly assigned labels on the training set of MNIST. Figure 6 illustrates the in-sample error rate, where one can notice that while both models manage to achieve a minimum error rate, one class converges much faster than the other. Thus, we propose applying TAC to trained base classifiers to avoid such training difficulties. We'll show that, in that case, we avoid accuracy drops even in much larger models and datasets.

| | *Clean Accuracy (%)* | *Det. AUROC (%)* |
|---|---|---|
| **MNIST** | | |
| Base model | 99.0 | 75.7 |
| TAC ($L_\infty$) | 98.6 | 80.8 |
| **CIFAR-10** | | |
| Base model | 95.6 | 93.7 |
| TAC ($L_1$) | 94.9 | 95.7 |

Table 3: Prediction performance as well as adversarial detection evaluation for PGD attackers under $L_\infty$ budgets of 0.3 and $\frac{8}{255}$ for the cases of MNIST and CIFAR-10, respectively.

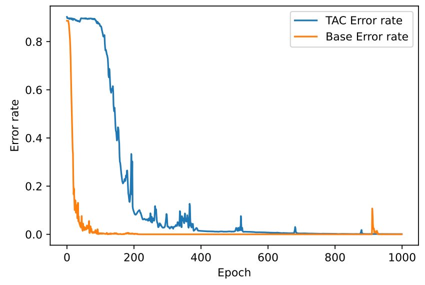


Figure 6: Capacity test on MNIST. Models overfit randomly assigned labels.

## A.3 Matching of activation profiles and codes

In Figures 7a and 7b, we report further evidence showing that TAC succeeds in matching activation profiles and class codes. To do so, we compare codes with the set of class-wise average activation profiles given by the following for a particular class label $c \in \mathcal{Y}$ and a data sample $D$:

$$\bar{A}(f', c) = \frac{1}{|D_c|} \sum_{x,y \in D} A(f', x) \mathbb{1}[y = c], \tag{5}$$

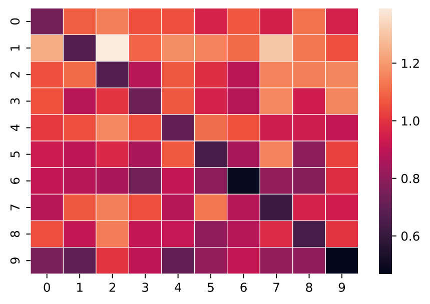
where $\mathbb{1}[\cdot]$ is the indicator function and $D_c$ is the subset of $D$ that belongs to class $c$:

$$D_c = \{(x, y) \in D : y = c\}. \tag{6}$$

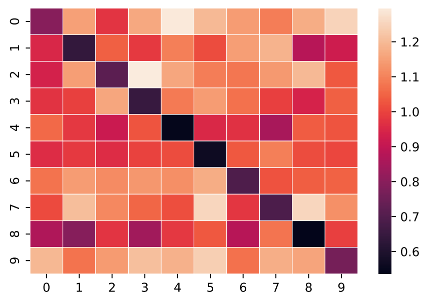
For both MNIST and CIFAR-10, we then build the heatmaps shown in Figures 7a and 7b by computing the cosine distances between class average activation profiles and codes. That is, a heatmap $H$ will be a $|\mathcal{Y}| \times |\mathcal{Y}|$ matrix such that entry $H_{ij}$ will be:

$$H_{ij}(f') = 1 - \cos(\bar{A}(f', i), C_j). \tag{7}$$

In both the cases of MNIST and CIFAR-10, we observe that $H$ is such that its main diagonal is highlighted, indicating an effective matching between activation profiles averaged for a given class and the corresponding class code.



(a) MNIST



(b) CIFAR-10

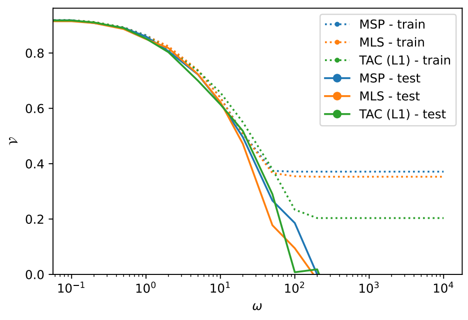
Figure 7: Heatmaps indicating the distances between activation profiles averaged per class and different class codes.

## A.4 Extra details and results on the performance of rejecting classifiers

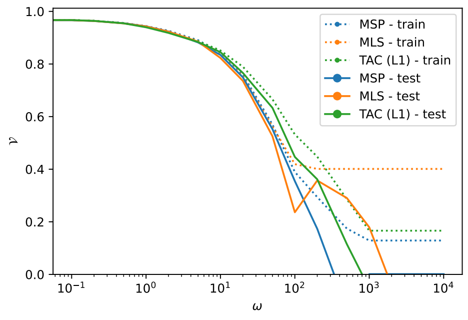
### A.4.1 VOC curves

We provide further results comparing TAC with commonly used confidence scores such as output layer statistics when models are expected to reject or abstain from predicting if not confident enough. We then repeat the procedure we used to create the VOC curve reported in Figure 2 for other datasets. To do that, we split the test sets into 5 splits. For each such split, we use the larger portion of the data (four splits) to find the confidence threshold that maximizes the value $\mathcal{V}$ of the underlying predictor under the given error detection scoring strategy. We then plot the average value curves as a function of $\omega$ (the error cost) for both the larger (train) and smaller (test) data portions when we refrain from predicting from any exemplar for which the confidence score is below the threshold for the given $\omega$. Figures 8a, 8a, and 8c correspond to VOC curves for HWU64, CLINC150, and ImageNet (same as Fig. 2 but with matching range of $\omega$ for consistency with other cases), respectively.

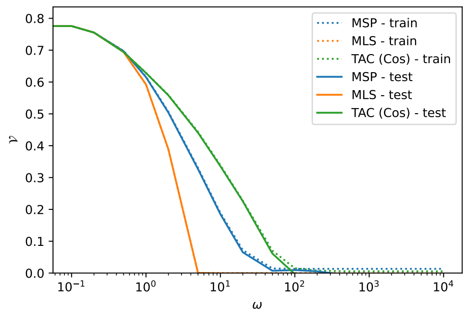
Results support our previous conclusion: the best detection score is application-dependent. In other words, best performers depend on the value of $\omega$ as well as on the underlying data. In any case, the use of TAC's scores results in improvements in several cases making it clear that the proposed approach should be used in cases where prediction errors are costly/unsafe and rejecting is a possibility or a requirement.



(a) HWU64



(b) CLINC150



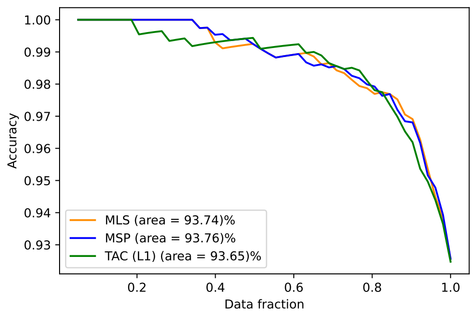
(c) ImageNet

Figure 8: VOC Curves.

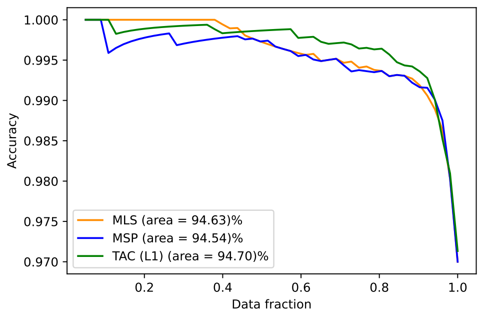
### A.4.2 Accuracy under varying rejection levels

To further assess the effect in performance given by rejecting classifiers when evaluated only on high confidence data points, we plot in Figures 9a, 9b, and 9c the test accuracy for various levels of rejection. In more detail, while in the vertical axis we plot the standard prediction accuracy on the underlying task, the horizontal axis corresponds to the fraction of the test set that is used for evaluation, and we keep the test instances yielding the highest confidence. For example, if *data fraction* is at the value $0.3$ in any of those curves, it indicates that $30\%$ of the test data was used for evaluation, and the selection is made after sorting data points in decreasing order of confidence. We further measure the area under the curves, which is better when higher and upper bounded by the value of 1.

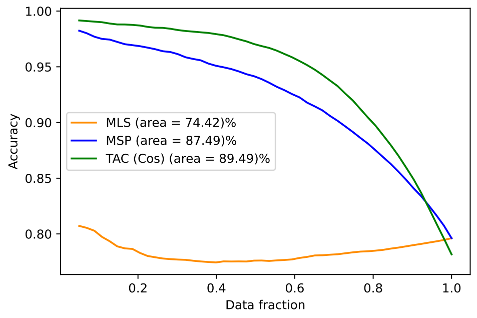
A general observation is that, for almost all cases, we observe an expected behaviour: the more we reject the better is the resulting prediction accuracy. This suggests that all of the considered confidence scores correlate well with prediction correctness. As before, results suggest that the best confidence score is data-dependent and also dependent on which level of rejection one is able to tolerate in a given application. Nonetheless, TAC can be beneficial over compared statistics.



(a) HWU64



(b) CLINC150



(c) ImageNet

Figure 9: Accuracy/rejection curves. The horizontal axis indicates the fraction of the test set that was used to measure prediction accuracy. Only the highest confidence test points are used.

## B Implementation details

### B.1 TAC's slice/reduce implementation example

In Figure 10, we show an example of an implementation of TAC's slice and reduce operations on top of 2-dimensional features. These operations are repeated in all the layers that are to be TAC'ed in a model stack and results are concatenated to define the complete activation profiles, and those should match in dimension with class codes.

### B.2 Additional training details

**Codes definition** The only requirement we consider for the class codes is that they are as dissimilar as possible in some sense. To build that set, one could use deterministic procedures such as, for instance, computing Hadamard matrices. However, we observed that, given large enough codes, randomly building $\mathcal{C}$ suffices for us to get discriminative codes. We thus define $\mathcal{C}$ as a random matrix of dimensions $|\mathcal{Y}| \times L$, where entries are independent $Bernoulli(0.5)$ random variables, observed at initialization.

```python
def compute_activation_profile(
    features: torch.FloatTensor, n_slices: int
) -> torch.FloatTensor:

    """Compute TAC's slice-reduce operations.
    Expects features with shape [N,C,H,W] where:
      N is the batch size.
      C is the number of channels of the conv. layer.
      H and W are spatial dimensions.

    Args:
        features (torch.FloatTensor): Input features.
        n_slices (int): Number of slices.

    Returns:
        torch.FloatTensor: Sliced and reduced activations of a layer.
    """

    batch_size, feature_dimension = features.size(0), features.size(1)

    slices_size = feature_dimension // n_slices
    total_slices_length = slices_size * n_slices

    slices_indices = torch.arange(total_slices_length).view(n_slices,
   slices_size)

    activation_profile = features[:, slices_indices, :, :]
    activation_profile = activation_profile.view(batch_size, n_slices,
     -1)
    activation_profile = activation_profile.sum(-1, keepdim=True)

    return activation_profile
```

Figure 10: Pytorch implementation of feature slicing and activation profiles computation for a TAC induced by 2-dimensional convolution layers.

**Attaching TAC to pre-trained models** The approach we consider in most of the evaluation we discuss is given by a TAC used as an extension of an existing classifier, and its goal is to provide confidence scores relative to predictions of the base model. We then propose a simple strategy to couple a TAC component in a pre-trained classifier. Provided that one has their ready-to-use classifier, a TAC is added on top of its features, and the base classifier is preserved as-is (*i.e.*, "frozen"). To do so, we isolate TAC from the base predictor via trainable projection layers applied in the representations of the base classifier. Then, we apply the slicing and reducing operations defined in Eq. 3 on those projections to obtain activation profiles. Projections correspond to independent MLP networks, one for each layer of the base classifier, used after average pooling of spatial dimensions for the case of multi-dimensional feature maps as in Fig. 1. The projection layers are then trained following the loss in Eq. **??**, but gradients are not propagated to the base classifier, which remains unchanged. We remark that, while this approach is intended to simplify TAC training and enable its use in more practical scale, it also enables the combined use of confidence scores obtained by TAC and at the output layer of the base classifier (*e.g.*, the maximum softmax probability (MSP) or the maximum logit score (MLS) [18]), *i.e.*, one can reject inputs that have too low confidence w.r.t. to either of these easy-to-obtain scores.

**Optimization and empirical observations** Training was performed with Adam in all cases except for models trained on MNIST and CIFAR-10, where SGD with momentum was employed. A grid-search over hyperparameters was performed in each dataset, and models reported in the main text correspond to the configuration reaching the highest prediction accuracy on in-distribution validation data. All training runs were performed in single GPU hardware, which required a few hours for all cases except for models trained on ImageNet, which continued improving until 2-3 days. Overall, we noticed that TAC tends to perform better when weight decay is not applied or when its coefficient is set to very small values ($< 10^{-5}$). Moreover, training against $\mathcal{L}_{bin}$ required relatively large learning rates compared to commonly used ranges for SGD. For MNIST and CIFAR-10 for instance, training to accuracy close to the base non-TAC model required learning rates greater than 1, which usually rendered training unstable. We observed this behaviour to change once we introduced $\mathcal{L}_{ce}$, after which common ranges of learning rate values used in popular recipes would work without change, though TAC still required moderate to low values of the weight decay parameter. Also, We usually needed to put pressure on one of the loss components, and we observed that the combination $(\alpha, \beta) = (1, 10)$ worked consistently well.

**Choices of distance functions** To define $D$, we either use some $L_p$ distance for $p \in 0, 1, 2, \infty$ or a cosine distance. While we used $p = 1$ in most cases, we found that the choice of $D$ does not affect TAC's performance significantly. For ImageNet, we found that setting $D$ to the cosine similarity for this model resulted in the best-performing approach. Moreover, at testing time, we used only the deepest of the 13 layers to compute TAC's confidence scores since that resulted in improved performance.

**Mixup** For models trained on CIFAR-10 and ImageNet, we performed Mixup [19] between pairs of inputs and their corresponding codes. That is, given a pair of exemplars $(x', y')$, $(x'', y'')$, we compute the mixtures of data and codes as given by $x^{mix} = \alpha x' + (1 - \alpha)x''$, $C_{y^{mix}} = \alpha C_{y'} + (1 - \alpha)C_{y''}$, and $\alpha \sim \text{Beta}(0.2, 0.2)$. Pairs are created by pairing a training mini-batch with a random interpolation of its copy, and $\alpha$ is drawn independently for each pair. We present an implementation of the mixing approach we used for training in Figure 11.

## B.3 Model architectures

### B.3.1 MNIST

Models trained on MNIST correspond to 4-layered convolutional stacks where each layer is followed by a LeakyReLU non-linearity. The numbers of channels in each layer are 64, 128, 256, and 512. To define TAC, we sliced post-activation features output by each such layer into 16 slices such that activation profiles and codes have dimension 64.

### B.3.2 CIFAR-10

Experiments on CIFAR-10 were carried out using a WideResNet-28-10 as described in [13]. Most ResNet's implementations (and its variants) split the model stack into four main blocks. We thus use the outputs of those blocks to define TAC. More specifically, in this case, we used the three blocks closer to the output, each outputting 160, 320, and 640 2-dimensional features. Once more, we defined TAC by slicing each set of features into 16 slices such that activation profiles and codes have dimension 48.

### B.3.3 ImageNet

For ImageNet, we trained both a ResNet-50 as well as a ViT under the `BASE-16x16` configuration. For the ResNet case, we used the outputs of its four blocks to define TAC, and dimensions of representations output in each such part of the model are 256, 512, 1024, 2048, each split in 256 slices to compose activation profiles of size 1024. For the ViT, we used all the 13 transformer layers after averaging across the spatial dimension. That is, we collect 13 768-dimensional vectors across depth to define TAC, and set the number or slices in each layer to 256. As discussed in the main text, for the ViT case, we noticed that using a cosine distance during training helped improve performance. Moreover, at testing time, using only the final part of the activation profile and code resulted in the best scoring strategy in this case. In both models, base predictors are pre-trained and TAC is defined on top of projections of features as described in Section **??**.

### B.3.4 Text classification

Architectures for the three text classification datasets we considered corresponded to the `RoBERTa-BASE` configuration. We train base models and use the projection approach described in Section **??** to define TAC. As in the case of ViT, we collect 768-dimensional feature vectors across the 13 layers of the model. However, in this case, rather than averaging out the sequential component as in the ViT, we use only the elements corresponding to the end-of-sequence token, and 16 slices are considered in this case for each set of features.

### B.3.5 Projection layers

Projections used to enable the strategy described in Section **??** correspond to stacks of fully-connected layers followed by ReLU activations. Independent projections are defined in each layer used to feed TAC. We considered 5 projection configurations named `small`, `large`, `very-large`, `x-large`, and `2x-large`, and the choice amongst those options is treated as a hyperparameter to be selected with cross-validation for each dataset we trained on. The numbers of fully connected layers for each configuration is 1, 2, 3, 3[1], and 5. The `x-large` and `2x-large` configurations include `LayerNorm` operations in the inputs and outputs.

## B.4 Text classification data

For text classification experiments, we train models on HWU64 [12], Banking77 [1], and CLINC150 [11], which correspond to sentence-level multi-class classification problems. HWU64 is composed of 25716 sentences and contains 64 classes, and those correspond to different commands for virtual assistants. BANKING77 contains 13,083 data exemplars and 77 classes corresponding to different intents. CLINC150 contains 23,700 utterances and 150 different intent classes, being one of them an *out-of-scope* or `UNKNOWN` class, which is ignored during training of TAC, but used at testing time for detection evaluations.

[1]The `x-large` configuration includes `LayerNorm` operations in addition to the fully connected layers.

```python
def mixup_interpolation(
    data_batch: torch.FloatTensor,
    one_hot_labels: torch.FloatTensor,
    code_labels: torch.FloatTensor,
    interpolation_range: float,
) -> list[torch.FloatTensor]:
    """Mixup style data interpolation.

    Args:
        data_batch (torch.FloatTensor): batch of data.
        one_hot_labels (torch.FloatTensor): batch of labels in one-hot
     format.
        code_labels (torch.FloatTensor): batch of binary class codes.
        interpolation_range (float, optional): Concentration param for
     the Bet distribution.

    Returns:
        Tuple with batches of interpolated pairs from data_batch,
    codes, and labels.
    """

    permutation_idx = torch.randperm(data_batch.size()[0], device=
    data_batch.device)

    data_pairs = data_batch[permutation_idx, ...]
    label_pairs = one_hot_labels[permutation_idx, ...]
    code_label_pairs = code_labels[permutation_idx, ...]

    # Mixup interpolator: random convex combination of pairs
    interpolation_factors = (
        torch.distributions.beta.Beta(interpolation_range,
    interpolation_range)
        .rsample(sample_shape=(data_pairs.size(0),))
        .to(data_batch.device)
    )
    # Create extra dimensions in the interpolation_factors tensor
    interpolation_factors_data = interpolation_factors[
        (...,) + (None,) * (data_batch.ndim - 1)
    ]
    interpolation_factors_labels = interpolation_factors[
        (...,) + (None,) * (one_hot_labels.ndim - 1)
    ]
    interpolation_factors_code_labels = interpolation_factors[
        (...,) + (None,) * (code_labels.ndim - 1)
    ]

    # Interpolation for a pair x_0, x_1 and factor t is given by t*(
    x_0)+(1-t)*x_1
    interpolated_batch = (
        interpolation_factors_data * data_batch
        + (1.0 - interpolation_factors_data) * data_pairs
    )
    interpolated_labels = (
        interpolation_factors_labels * one_hot_labels
        + (1.0 - interpolation_factors_labels) * label_pairs
    )
    interpolated_code_labels = (
        interpolation_factors_code_labels * code_labels
        + (1.0 - interpolation_factors_code_labels) * code_label_pairs
    )

    return interpolated_batch, interpolated_labels,
    interpolated_code_labels
```

Figure 11: Pytorch implementation of Mixup interpolations.