GCOOD: A Generic Coupled Out-of-Distribution Detector for Robust Classification

Rogério Ferreira de Moraes*, Raphael dos S. Evangelista*, Leandro A. F. Fernandes*, Luis Martí†
* Universidade Federal Fluminense (UFF), Niterói, Brazil
Email: {rogeriofm, revangelista}@id.uff.br, laffernandes@ic.uff.br
† Inria Chile Research Center, Las Condes, Chile
Email: lmarti@inria.cl

***Abstract*—Neural networks have achieved high degrees of accuracy in classification tasks. However, when an out-of-distribution (OOD) sample (*i.e.,* entries from unknown classes) is submitted to the classification process, the result is the association of the sample to one or more of the trained classes with different degrees of confidence. If any of these confidence values are more significant than the user-defined threshold, the network will mislabel the sample, affecting the model credibility. The definition of the acceptance threshold itself is a sensitive issue in the face of the classifier's overconfidence. This paper presents the Generic Coupled OOD Detector (GCOOD), a novel Convolutional Neural Network (CNN) tailored to detect whether an entry submitted to a trained classification model is an OOD sample for that model. From the analysis of the Softmax output of any classifier, our approach can indicate whether the resulting classification should be considered or not as a sample of some of the trained classes. To train our CNN, we had to develop a novel training strategy based on Voronoi diagrams of the location of representative entries in the latent space of the classification model and graph coloring. We evaluated our approach using ResNet, VGG, DenseNet, and SqueezeNet classifiers with images from the CIFAR-10 dataset.**

I. Introduction

The classification problem is the task of assigning data samples to categories or classes. For a supervised classification model to learn to predict outcomes accurately, it needs representative training examples for each of the target classes and be able to adapt whether the statistical properties of the classes change over time. Otherwise, when the classes change in unforeseen ways, samples drift from the distribution learned, or samples are from an unknown class, a. k. a. out-of-distribution (OOD) samples, the model may produce incorrect outcomes with high degrees of confidence. Identifying whether a classification model can classify a data entry correctly can reduce type I errors, indicate the need to retrain the model or support continuous adjustment of the learned distributions. In this work, we are interested in identifying OOD samples rather than concept or data drift.

The identification of samples from unknown classes is related to the anomaly detection problem since anomaly refers to something that is out of expectation [1]. In data streams, the interest in recognizing samples from unknown classes is due to the need for a continuous update of the predictive model as new data is presented [2]. In this context, changes can either characterize concept drift or concept evolution. The latter occurs when a new class needs to be learned. Techniques that work with data streams are usually also interested in performing continuous adjustment of the classification model [3]. For this, the hypothesis of a new class requires an expressive set of samples that corroborate its existence.

The OOD detection problem has been addressed using the analysis of data produced during the classification process. More specifically, in models based on neural networks, metrics concerning data produced by hidden layers [4], [5] and the gradient [6] were proposed to identify input entries related to unknown classes. The drawback of those approaches is be tailored to the classification method on which they were built.

We present a single-sample OOD detection approach that, after trained, can be applied to virtually any classification method that produces class probability distributions for the given samples. Our Generic Coupled OOD Detector (GCOOD) consists of a Convolutional Neural Network (CNN) that receives as input only the Softmax output of any classifier and accurately indicates whether such classification result is related to a sample of some of the modeled classes. We also propose a novel training strategy for OOD detectors based on coloring Voronoi diagrams modeling the relative location of representative class entries in the latent space. We evaluated our approach with experiments on image classification, using ResNet [7], VGG [8], DenseNet [9], and SqueezeNet [10] classifiers and the CIFAR-10 dataset [11].

The main contributions of this work are: (i) a new CNN-based OOD sample detection model; and (ii) a new methodology for training OOD detection approaches.

The remaining part of this paper is organized as follows. In the next section (Section II) we deal with the foundations of our work providing a succinct outline of the state of the art and our motivation. After that, in Section III, we describe our proposed GCOOD method in detail. Subsequently, in Section V we analyze GCOOD from an experimental point of view. Finally, Section VI puts forward our conclusive remarks about the work and outlines lines of future work.

II. Related Work

Various papers deal with OOD detection in the Machine Learning (ML) and Pattern Recognition (PR) areas. Much of them proposing scores to measure abnormal samples. Oberdiek *et al.* [6] proposed gradient metrics to measure the uncertainty of CNNs and related large metric values to OOD samples.

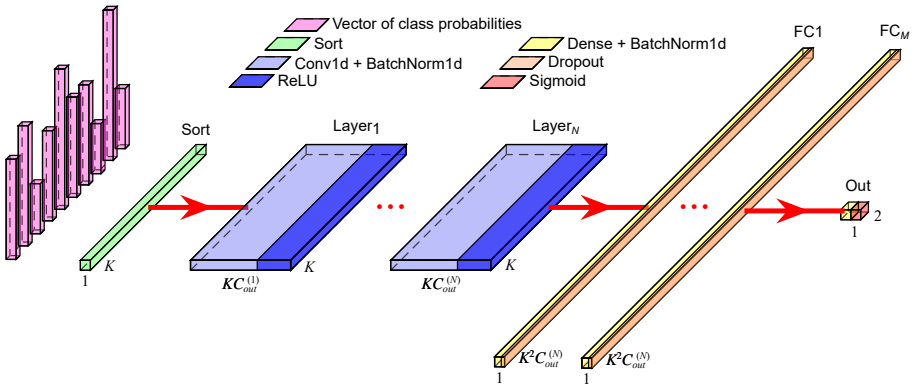


Fig. 1. Schematic representation of GCOOD. It is a sequential neural network consisting mainly of convolutional ($\texttt{Layer}_l$) and fully connected ($\texttt{FC}_l$) layers.

Lee *et al.* [4] presented metrics that can be applied to any pre-trained classifier to detect OOD samples and adversarial attacks. Hendrycks *et al.* [12] proposed an unsupervised metric based on Area Under the Receiver Operating Characteristic (AUROC) curve. Also, Sastry and Oore [5] used Gram matrices to represent the intermediary layers of a classifier CNN to catch possible clues of a new class. All those methods rely on data produced by hidden layers of the network.

The detection of minor variations in in-distribution samples is a more complex problem, referred to as novelty detection. Novelties are closely related to in-distribution data, especially to unknown information during training [13]. Sabokrou *et al.* [14] and Pidhorskyi *et al.* [15] proposed adversarial autoencoder architectures for outlier detection, where the autoencoder improves the discriminator performance by enhancing inlier results and deviating the outliers. Akshaya and Kala [16] proposed a sided CNN as a new class detector. Their approach is similar to our OOD detector, but it does not use the Softmax output of classification models as input. Instead, it only uses the probability estimated for the most likely class to judge the sample. The quality of the method proposed by Akshaya and Kala is not evaluated in their work. Our OOD detector uses the entire Softmax output of a pre-trained classification model to predict the need for a new class for the given sample.

Concept Drift detection is a problem related to OOD detection. There is a concept drift when a domain has adapted over time, causing initially noisy data to become valid data [17]–[20]. Although our work does not directly deal with possible domain changes this is a future direction of exploration.

Another related concept is underspecification [21] which is a challenge for the credibility of ML applications. This is a cornerstone and severe problem that has often been left aside in ML practice. ML models often exhibit an unexpected poor behavior when deployed in real-world domains even after a properly designed and curated optimization (training) phase. This ambiguity can lead to instability and poor model behavior in practice and is a distinct failure mode from previously identified issues arising from the structural mismatch between training and deployment scenarios. This phenomenon is severe in many key application domains (see [21] for a list of known cases). One way to cope with underspecification is to enhance predictive models with the capacity to express what degree they correctly model or not different parts of the input domain. In that regard, the GCOOD proposal addresses that issue by coupling a predictive model with an OOD detector.

III. GCOOD: The Proposed OOD Detector

Our proposal consists of a generic OOD detector coupled with any classification model that produces Softmax-based vectors of class probabilities. At inference time, GCOOD can tell whether the class probability distribution produced by the classifier for a given input data entry suggests the need for a new (not previously seen) class.

From the architectural point of view, GCOOD is a binary multi-layer perceptron neural network implemented using convolutional layers and fully connected layers. Its prime goal is to identify input samples that should be classified as OOD entries. Our network expects as input a 1-dimensional array $X = (x_1, x_2, \cdots, x_K)$ of size K consisting of the probabilities x_i of a data entry to be classified as the i-th class considered by some classification model. Therefore, $x_i \in [0, 1]$ for all $i \in \{1, 2, \cdots, K\}$, and $\sum_{i=1}^{K} x_i = 1$. As illustrated in Fig. 1, our network has one pre-processing layer ($\texttt{Sort}$), N convolutional layers ($\texttt{Layer}_l$), M fully connected layers ($\texttt{FC}_l$), and one output layer ($\texttt{Out}$). The values of N and M may vary according to the size of X. In our experiments, they are obtained by hyperparameter tuning. Refer to Section V for details. The output of our network is a tuple $Y = (y_1, y_2)$, where $y_1 > y_2$ indicates that X came from an OOD sample.

The pre-processing layer sorts the x_i entries in X ascending, producing the array $X' = (x'_1, x'_2, \cdots, x'_K)$, such that $x'_i \leq x'_{i+1}$. This operation is necessary to reduce the variability in the input data. Each $\texttt{Layer}_l$ applies a 1-dimensional convolution over the input signal of the layer, with kernels of size 1, stride of 1, bias, and no padding to keep input and output with the same size, followed by batch normalization with momentum 0.1, and ReLU activation function. Although we have implemented $\texttt{Layer}_l$ as convolutional layers, these layers behave like fully connected layers that receive $C_{\text{in}}^{(l)}$ input channels and produce $C_{\text{out}}^{(l)}$ output channels. Here, l is the index of the l-th layer. Thus, $C_{\text{in}}^{(1)} = 1$ and $C_{\text{in}}^{(l)} = C_{\text{out}}^{(l-1)}$, for $l > 1$. To guarantee the fully-connected

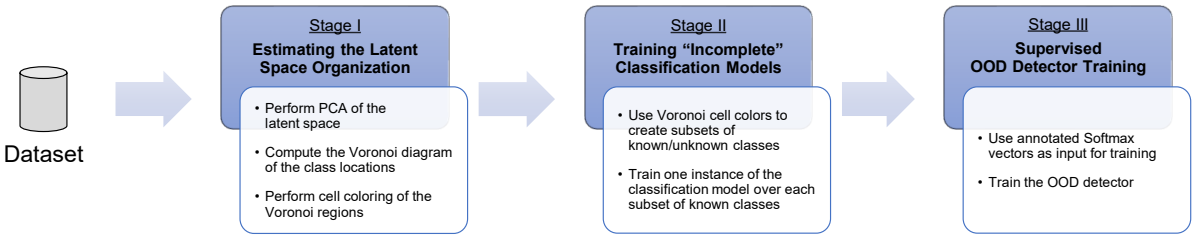


Fig. 2. Overview of the proposed training strategy for OOD sample detectors. Refer to Section IV for details.

behavior, the number of output channels of the convolution operation performed by the l-th layer is set to $KC_{\text{out}}^{(l)}$, and we use grouped convolutions with $C_{\text{in}}^{(l)}$ groups. For sake of simplicity, in our experiments we have used the same number of output channels for all simulated fully connected layers, *i.e.*, $C_{\text{out}}^{(l)} = C_{\text{out}}$ for all $l \in \{1, 2, \cdots, N\}$, and have found C_{out} by hyperparameter tuning. The reasoning behind simulating fully connected layers instead of using standard convolutions is that unlike images, where the local analysis promoted by convolutions helps to identify texture patterns, the analysis of a probability vector requires global inspection of the distribution of probabilities. Furthermore, ML frameworks do not usually implement modules for dense layers that produce outputs with a different number of channels than input.

After applying all the $\texttt{Layer}_l$ layers, the resulting features' vector of size K and $KC_{\text{out}}^{(N)}$ channels is flattened into a vector with $K^2C_{\text{out}}^{(N)}$ elements and one channel, which is processed in turn by a sequence of $\texttt{FC}_l$ layers. Each $\texttt{FC}_l$ layer applies a linear transformation to the incoming data, keeping the number of features equal to the layer's input, followed by batch normalization with momentum 0.1. During training, $\texttt{FC}_l$ also applies dropout with probability p. Finally, the $\texttt{Out}$ layer applies a linear transformation followed by batch normalization and the Logistic Sigmoid function, producing the resulting tuple Y. All linear transformations include bias.

IV. The Proposed Training Methodology

A common practice described in the literature is to train a classification model using some dataset (*e.g.*, CIFAR-10) and verify the OOD detector's ability to identify whether inputs of different classes from other datasets (*e.g.*, MNIST) as OOD samples. This methodology is usually applied in both the training and test stages of the OOD detector model. However, this practice has three main issues:

1) Using very different datasets (*e.g.,* CIFAR-10 and MNIST) can lead to a biased quality assessment, favoring the evaluated OOD detector. In real world, unknown class entries might not be as different as expected.
2) Using datasets having entries similar in nature but with different classes (*e.g.,* CIFAR-10 and CIFAR-100) can lead to choosing classes from the second dataset that are supposed to be OOD but in practice may have overlapping semantics with classes from the first one. Such an overlap would also lead to bias in the assessment.
3) Using a single dataset with the naive separation of some classes to serve as OOD data can lead to problems similar to those pointed out in previous items.

Our strategy for choosing from a dataset which classes will be used to train the classification model and which ones will be the source of OOD samples while training the OOD detector prevents the three aforementioned issues. Fig. 2 illustrates the main stages of the proposed methodology. In the first stage (Section IV-A), we take a dataset (*e.g.,* CIFAR-10). Then, we train the classification model (*e.g.,* ResNet-152) considering all the classes. The intrinsic organization of latent space of the trained classification model serves as a reference to choose which classes can be considered OOD without jeopardizing their semantic separation from other classes. Such an approach solves issues (2) and (3). By using one dataset, we naturally solve issue (1). In the second stage (Section IV-B), we train different instances of the classification model, considering for each of them a subset of classes that were securely kept in the dataset and using the removed classes as a source of OOD samples. By training more than one classifier instance, we mitigate the bias of choosing a single subset of OOD classes and a single classification model. In the last stage of the proposed methodology (Section IV-C), we compute probability vectors by applying the trained classification model instances on dataset entries from their known and unknown classes. We use this new annotated dataset of OOD and non-OOD samples to train and test our OOD detector.

A. Estimating the Organization of the Latent Space

The concept of latent space becomes popular with Generative Adversarial Networks (GANs) and Variational Auto Encoders (VAEs), where the features produced by the final layers of the encoding network can be treated as elements in a vector space and be subjected to vector algebra. But this concept predates the popularity of CNNs. The organization of a feature space by conceptual proximity plays a key role in designing classification models in ML and PR.

In the first stage of our training methodology, the aim is to infer the general organization of the latent space of a classification model trained on a dataset $\mathcal{D}$ composed of data entries labeled as one of the L classes in the set $\mathcal{C} = \{c_1, c_2, \cdots, c_L\}$. When the classification model is based on neural networks, the feature vectors produced by the feature extraction layers preceding the layers responsible for classification characterize points in the learned latent space. We assume that the classification model to be considered was designed and trained to

map dataset entries labeled as the same class to close points in the latent space. Also, we assume that the cluster of points associated with the same class has little or no overlap with other clusters. Although it is difficult to guarantee these two premises, by experience, we have observed that the use of representative points for classes in a well-trained latent space is enough to infer the general relation of classes in this space.

While performing data preparation, we split the dataset $\mathcal{D}$ into training, validation, and test subsets, keeping the proportion of elements from each class in the subsets. After training a classification model on the training subset, we map all data entries in the training subset to the latent space and apply dimensionality reduction by the Principal Components Analysis (PCA) of the feature vectors translated by the mean vector $\bar{F}$. As discussed later in this section, project the translated feature vectors to the first T principal components (*i.e.,* the dimensions that maximize the variance of the projected data) is key to infer the relative semantics of classes.

We calculate the representative point $\bar{R}_{c_i}$ of class c_i as the mean point of all the data entries labeled as c_i mapped to the T-dimensional space. Finally, the Voronoi diagram of points $\bar{R}_{c_1}, \bar{R}_{c_2}, \cdots, \bar{R}_{c_L}$ provides the notion of semantic proximity for classes associated with neighboring Voronoi cells.

In the next stage of our training methodology, we will train instances of the classification model to be aware of a subset of classes from $\mathcal{C}$. These subsets are created by removing from $\mathcal{C}$ the classes that do not compromise the general topology of the semantic organization of the latent space. The naive removal of classes may lead to bias in the training and test of the OOD sample detector. For example, if the dataset entries of many neighboring classes are removed to become OOD samples, then the "hole" created in the latent space semantics can make the OOD detector model learn only an extreme class separation case (*i.e.,* it may be like getting OOD samples from MNIST to check against a classification model trained on CIFAR-10). We apply vertex coloring on the Voronoi diagram to identify subsets of classes that will not compromise large semantic regions of the latent space if removed along with classes associated with the same color. Recall that, in its simplest form, vertex coloring labels the vertices of a graph (*i.e.,* the seeds $\bar{R}_{c_i}$ of the Voronoi cells) such that no two adjacent vertices are of the same color. Therefore, by removing all classes labeled with the same color, there is a guarantee that we will not remove neighboring classes simultaneously.

For $K > 1$, the coloring process can produce 2 to K colors. The 2-color case happens when all classes are aligned, causing their Voronoi cells to have up to 2 neighbors each. The K-color case may occur when the number of dimensions T is greater than or equal to $K - 1$. In this case, the neighborhood graph of the points $\bar{R}_{c_i}$ may define a $(K-1)$-simplex (*i.e,* a complete neighborhood graph) that expands a $(K-1)$-dimensional space. We want to minimize the number of colors without compromising the semantic organization of classes in the T-dimensional space. So, the dimensionality of this space must be restricted to $1 < T < K$, observing in the PCA the amount of information preserved by the first T principal components.

B. Training "Incomplete" Classification Model Instances

The vertex coloring process partitions the set of classes, $\mathcal{C}$. It guarantees that the classes in the same partition are not semantic neighbors in latent space. Also, the coloring process allows creating a mapping function $f : \mathcal{A} \mapsto \mathcal{P}$, such that a colour a from the set of colors $\mathcal{A}$ maps to a subset of classes $\mathcal{P}_i$ from the set of partitions $\mathcal{P}$, where $\bigcup_{\mathcal{P}_i \in \mathcal{P}} = \mathcal{C}$, $\mathcal{P}_i \cap \mathcal{P}_j = \emptyset$ for all $i \neq j$, and $\mathcal{P}_i \neq \emptyset$ for all i.

In this stage of the training methodology, we use data from the training subset of dataset entries assumed in Section IV-A to train instances of the classification model so that each instance is aware of the existence of only one subset $\mathcal{K}$ of classes. This known subset $\mathcal{K}$ is given as one of the non-empty subsets in the power set $P(\mathcal{S}_i)$ of one of the $\mathcal{S}_i$ partitions in $\mathcal{S}$. In other words, there will be instances of classifiers that will know all but one of the classes. Other trained models will not be aware of two of the classes, and so on, where the number and which classes are unknown is determined by the partitioning given by coloring.

For example, if $\mathcal{C} = \{car, cat, chair, dog, horse\}$ and the vertex coloring produces $\mathcal{A} = \{red, green, blue, yellow\}$ and $\mathcal{S} = \{\{car, chair\}, \{cat\}, \{dog\}, \{horse\}\}$, mapping $red \mapsto \{car, chair\}$, $green \mapsto \{cat\}$, $blue \mapsto \{dog\}$, and $yellow \mapsto \{horse\}$, then we train six instances of the classification model. Each of the instances will be aware of one of the following subsets $\mathcal{K}$ of classes: $\{car, cat, chair, dog\}$, $\{car, cat, chair, horse\}$, $\{car, cat, dog, horse\}$, $\{car, chair, dog, horse\}$, $\{cat, chair, dog, horse\}$, and $\{cat, dog, horse\}$. In the next stage of our training methodology, the subset $\mathcal{U} = \mathcal{K} \setminus \mathcal{C}$ of classes is taken as source of OOD samples for the model trained on the dataset entries with labels from $\mathcal{K}$.

C. Training the OOD Detector

In this stage of the proposed methodology, we use classification results produced by the models trained in the second stage as a source of annotated data for training, validation, and test of the OOD detector model. One must be aware that splitting the dataset $\mathcal{D}$ into training, validation, and testing subsets during the first stage of the process, and using the subset of training data to fit the classification models considered in Sections IV-A and IV-B, places some restrictions on which data samples we can use at the present stage. We can only rely on the output vectors produced by the classification models for data entries not seen by them during their training. Therefore, the data that we can use is restricted to the probability vectors resulting from processing the test subset of $\mathcal{D}$, as these data entries were not used to train the classification models in any of the previous stages. From now on, we will refer to the former set of test data entries as set $\mathcal{D}'$.

During the new data preparation step, for each classification model instance trained in the previous stages, we process the elements in $\mathcal{D}'$ and label the resulting probability vectors as from an OOD sample if the element's class is unknown by the classifier or from a non-OOD sample otherwise. With the labeled vectors in hand, we split the new dataset into training, validation, and test subsets, keeping the proportion of OOD

and non-OOD elements in each subset. But, to avoid bias, we have to ensure that the colors of the classes of the samples used to produce the training vectors are different from those whose samples produce the validation and test subsets. Then, using the new dataset, we proceed with the training process designed for the specific OOD detector considered. In Section V, we present the experimental analysis of the GCOOD, including its training processes considering data produced by ResNet-152 models on the CIFAR-10 dataset.

Even with a balanced dataset $\mathcal{D}$, and hence a balanced dataset $\mathcal{D}'$, it is noteworthy that the data in the new dataset is imbalanced. We have much more class probability vectors X labeled as from OOD samples than from non-OOD samples. Thus, it is important to adopt an objective function that mitigates imbalance, regardless of the OOD detection model being trained. In the case of GCOOD, we have using the Binary Cross Entropy between the target $Z = (z_1, z_2)$ and the output $Y = (y_1, y_2)$ of our network, and add weights w_1 and w_2 to positive examples of each label:

$$\ell_c(Y, Z) = w_c z_c \log(y_c) + (1 - z_c) \log(1 - y_c), \qquad (1)$$

where $c = 1$ and $c = 2$ denote the OOD and non-OOD labels.

V. Experimental Analysis

We have implemented the GCOOD detector using PyTorch 1.7.1. The experiments were performed inside Docker containers, in an Intel Xeon E5-2698 v4 CPU with 2.2Ghz, 512GB of RAM, and 8 GPUs NVIDIA Tesla P100-SXM2 with 16Gb of memory each. Despite the number of available GPUs, we have set the visible to one. Our implementation and trained models are available at https://github.com/Prograf-UFF/GCOOD.

The experiments considered image classification by ResNet-152 [7], VGG-16 [8], DenseNet-201 [9], and SqueezeNet-v1.0 [10] models, and images from the CIFAR-10 dataset [11]. We trained one GCOOD model using the class probability vectors produced by instances of the ResNet-152. We evaluated the performance of our OOD detector by analyzing True Positive Rate (TPR), True Negative Rate (TNR), F1-score, and AUROC curve produced while identifying OOD samples in classification results from the four image classification architectures considered. We compared our results with a Support Vector Machine (SVM) classifier using Radial Basis Function kernel with $C = 1$ and $\gamma = 0.1$, degree of 3, balanced class weights, and using the shrinking heuristic, probability estimates, and a one-vs-rest decision function.

A. Training Procedure

We have trained our GCOOD model using the training methodology described in Section IV. The instances of image classifiers were pre-trained on the ImageNet [22] dataset and fine-tuned to classes from the CIFAR-10 dataset. This dataset includes 10 classes, with 6K samples each, subdivided into subsets of 5K images for training and 1K images for testing. To increase the robustness of the image classification models, we have combined two data augmentation strategies on the original training set of the CIFAR-10 dataset. First, we augmented the collection by creating nine new images per sample, considering all the 34 random transformations implemented by the Albumentation tool [23]. Next, we produced other 10K fake images per class using a Deep Convolutional Generative Adversarial Network (DCGAN) [24], ending up with 60K images per class in place of the 5K images from the original training set of the CIFAR-10. Finally, we split the augmented set into the actual training and validation subsets used to fit the image classification models. These balanced subsets have 48K and 12K images per class, respectively. It is important to emphasize that the test subset contains 1K images per class, and we kept it as defined by the CIFAR-10 dataset.

TABLE I
Accuracy of the image classification models.

Model	$\mathcal{U}$	ResNet	VGG	DenseNet	SqueezeNet
M_1	$\{airplane\}$	0.859	0.897	0.856	0.808
M_2	$\{airplane, deer\}$	0.872	0.902	0.861	0.827
M_3	$\{automobile\}$	0.853	0.890	0.850	0.808
M_4	$\{bird\}$	0.870	0.905	0.871	0.826
M_5	$\{cat\}$	0.889	0.924	0.880	0.850
M_6	$\{cat, horse\}$	0.887	0.927	0.890	0.857
M_7	$\{cat, horse, ship\}$	0.892	0.935	0.884	0.858
M_8	$\{cat, ship\}$	0.887	0.919	0.892	0.850
M_9	$\{deer\}$	0.864	0.892	0.867	0.829
M_{10}	$\{dog\}$	0.870	0.911	0.880	0.836
M_{11}	$\{frog\}$	0.854	0.895	0.850	0.811
M_{12}	$\{horse\}$	0.851	0.890	0.854	0.810
M_{13}	$\{horse, ship\}$	0.852	0.896	0.862	0.810
M_{14}	$\{ship\}$	0.851	0.891	0.850	0.808
M_{15}	$\{truck\}$	0.853	0.888	0.858	0.818
M_{all}	$\{\}$	0.853	–	–	–

After training one instance of the ResNet-152 using all classes in the CIFAR-10 dataset, we proceed with the dimensionality reduction of the latent space, calculation of the Voronoi diagram, and vertex coloring (see Section IV-A). In Section V-B, we will discuss details on the hyperparameter tuning of the classification models. In ResNet-152, the latent space with 2,048 dimensions corresponds to the output of the 2D adaptive average pooling operation that precedes the fully connected layer at the end of the network.

We have used the PCA and Voronoi implementations available on the Scikit-learn tool [25] and the greedy graph coloring algorithm implemented by the NetworkX package [26]. In the PCA, we took the first $T = 7$ principal components, preserving 84% of the data while performing dimensionality reduction. Although setting $T < 7$ would lead to more chances of producing colors with more associated classes, we observed that too much information was lost by projecting the data to $T = 6$ dimensions, compromising the actual semantic organization of classes in the low-dimensional space. We have used the largest-first strategy in graph coloring because, in our experiments, it maximizes the number of classes assigned to a single color. For ResNet-152 and $T = 7$, the mapping between colors and classes was: $clr_1 \mapsto \{dog\}$, $clr_2 \mapsto \{bird\}$, $clr_3 \mapsto \{frog\}$, $clr_4 \mapsto \{truck\}$, $clr_5 \mapsto \{airplane, deer\}$, $clr_6 \mapsto \{automobile\}$, and $clr_7 \mapsto \{cat, horse, ship\}$.

Using the color-class mapping, in the second stage we trained 15 "incomplete" classification model instances (Section IV-B). Therefore, we have 15 ResNet models that are not aware of some subset $\mathcal{U}$ of classes, and one model aware of all classes, as presented by the first three columns of Table I.

In addition to the 15 "incomplete" ResNet-152 models, we also trained 15 models for each of the VGG-16, DenseNet-201, SqueezeNet-v1.0 networks, considering the absence of the classes indicated in the $\mathcal{U}$ column of Table I. These models will be used later to test the GCOOD model.

While training the image classification models, we monitored the mean Cross-Entropy loss and accuracy on the training and validation images along 50 epochs. We took the models with the best performance obtained before the networks stopped learning. Table I shows the final test accuracies calculated on the subset of test images.

Following the third stage of our training methodology (Section IV-C), we took the 1K image samples in the test subset of the CIFAR-10 and processed those images with the 15 incomplete ResNet models trained in the second stage to produce 15K probability vectors labeled as from ODD or non-ODD samples. Then, we set the input size of our GCOOD network to $K = 10$ and completed the vectors produced by the image classifiers with zeros whenever necessary For instance, M_2, M_6, M_8, and M_{16} produce vectors of size 8.

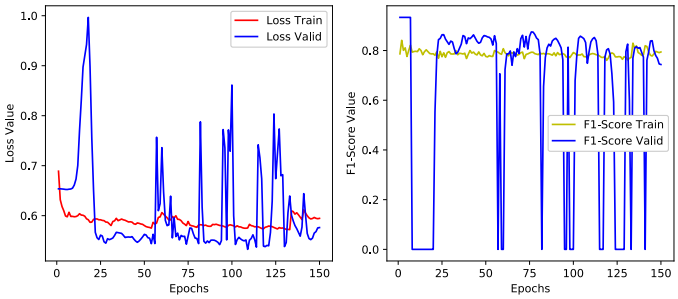
Randomly, we have split the annotated probability vectors into training (60%), validation (20%), and test (20%) subsets, respecting that image samples used to produce training data never share classes with the same color as the classes used to create validation and test data. In this case, we roughly put $2/3$ of image classes in the training subset and $1/3$ in the validation and test subsets by random. For the experiment disscussed in Section V-C, we also produced one test subset using each of the other three classification network considered, *i.e.,* VGG-16, DenseNet-201, and SqueezeNet-v1.0. These subsets result from the classification of the same images that generated the entries in the test subset obtained from ResNet.

To compensate for class imbalance, we calculated the weights of the loss function (1) as $w_1 = n_2/n_1 = 5.875$ and $w_2 = 1/w_1 = 0.1702$, where n_1 and n_2 are the number of OOD and non-OOD examples in the training set.

Before training, all weights of the GCOOD model were randomly initialized from a Normal distribution of mean 0.0 and standard deviation 0.02. During 150 epochs, we monitored the mean Binary Cross-Entropy loss (1) and F1-score on the training and validation subsets (Fig. 3). We took the model with the best performance before the network stopped learning. The final TPR, TNR, F1-score, and ROC curve were calculated on the test data produced by the image classification models.

B. Hyperparameters Tuning

We have applied a Bayesian approach [27] for tuning, using Hyperband [28] with $s = 2$ and $\eta = 3$ as stopping criteria, and a maximum of 100 and 300 runs for, respectively, image classification and GCOOD models. The implementations of [27] and [28] are available in the Weights & Biases toolset [29].



(a) Loss graph (b) F1-Score graph

Fig. 3. The loss and F1-score of the GCOOD model during training.

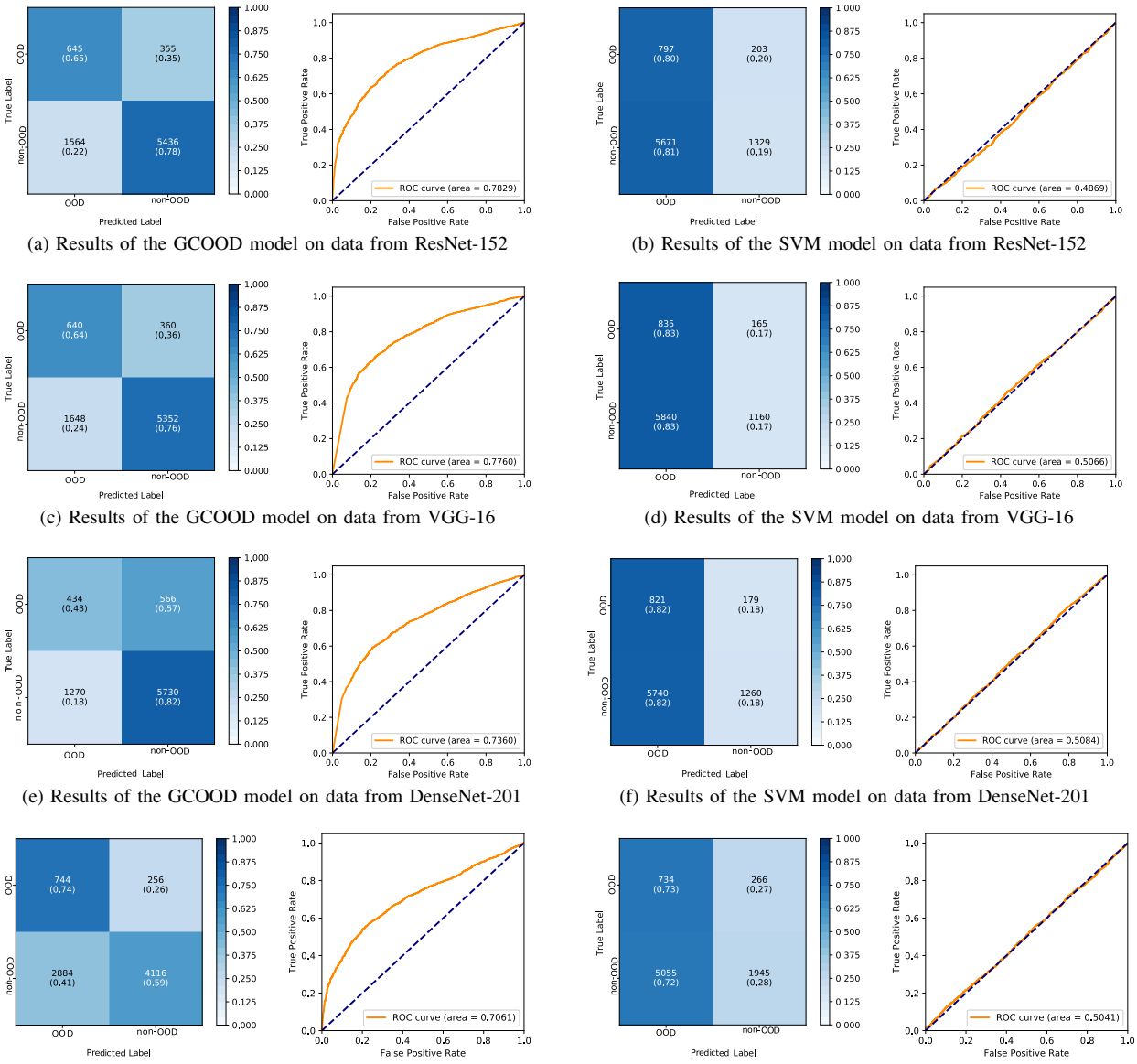
The hyperparameters considered for image classification models were batch size in $\{16, 32, \cdots, 512\}$, learning rate ranging from 10^{-5} to 10^{-2}, and choose between the Adam or SGD optimizers. Once the parameter sweep was finished, we found that a batch size of 256, a learning rate of 0.0003, and the Adam optimizer performs better on the ResNet-152 aware of all classes. Then, we set those hyperparameters values to all other image classification models trained in our experiment.

For the GCOOD model, we first performed some hyperparameter sweeps, then fixed three hyperparameters values based on observations and run a new tuning on the reduced search space. We have set the batch size to 15,800 as it brought the best compromise between convergence time and loss oscillation. The number of fully connected layers was set to $M = 1$ after we have observed that the sweeps usually converge to at most two layers of that kind. Also, we experimented with changing the number of output channels of the fully connected layers simulated by the convolutional layers manually in a progressive way, starting from $C_{\text{out}} = 1$. We got better results by setting $C_{\text{out}} = 5$. Among the remaining hyperparameter, we had the number of convolutional layers $N \in \{1, 2, \cdots, 6\}$, the learning rate ranging from 10^{-7} to 10^{-2}, dropout probability $p \in [0.0, 0.3]$, and choose between the Adam and SGD optimizers. The tuning procedure pointed out $N = 6$, learning rate of 0.004012, $p = 0.2757$, and the SGD optimizer with momentum of 0.9.

C. OOD Detection Results

Fig. 4 shows the confusion matrices and ROC curves obtained by applying the GCOOD and SVM models over test sets produced by the networks ResNet-152 (a-b), VGG-16 (c-d), DenseNet-201 (e-f), and SqueezeNet-v1.0 (g-h). TPR, TRN, F1-Score, and AUROC curve values are presented in Table II. Recall from Table I that the accuracies of ResNet-152, VGG-16, and DenseNet-201 are slightly higher than those of SqueezeNet-v1.0. While the accuracy obtained varies from 0.850 to 0.935 in the first three architectures, for the SqueezeNet models the accuracies range from 0.808 to 0.858. Using an image classifier of lower quality, such as SqueezeNet-v1.0, was purposeful to verify the influence of this quality on the detection capacity of the GCOOD model.

As can be seen from the main diagonal of the confusion matrices in Figs. 4 (a) and (c), the TPR and TNR (values



(a) Results of the GCOOD model on data from ResNet-152

(b) Results of the SVM model on data from ResNet-152

(c) Results of the GCOOD model on data from VGG-16

(d) Results of the SVM model on data from VGG-16

(e) Results of the GCOOD model on data from DenseNet-201

(f) Results of the SVM model on data from DenseNet-201

(g) Results of the GCOOD model on data from SqueezeNet-v1.0

(h) Results of the SVM model on data from SqueezeNet-v1.0

Fig. 4. The confusion matrices and the ROC curves of the OOD detection results of the GCOOD (left) and SVM (right) models on test classification data produced by all the image classification networks considered in this study.

in parentheses) of the GCOOD model are higher than the off-diagonal ratios, related to the proportion of type I and II errors (note the intensities of the blue color). In Fig. 4 (e), in the first line of the confusion matrix, we observe that there were more errors (False Negative, FN: 566) than successes (True Positive, TP: 434). In Fig. 4 (g), our approach had a higher TPR (0.74) than on data coming from other networks (0.43 to 0.65). But at the same time, it had more False Positives (FP: 2,884, against 1,270 to 1,648). We believe that the highest amount of FP is related to the predictive capacity of SqueezeNet-v1.0, which mistakenly generates more confident probability vectors for certain classes even when the sample is OOD.

When performing the experiments, we observed that small changes in training loss could cause significant changes in F1-Score. The cause of this could be that the neighborhood that separates the OOD and non-OOD labels is very narrow, and this led us to think about using SVM, which has good behavior for problems like this. However, as shown in Fig. 4 (right), the SVM confusion matrices concentrate predictions in the OOD label, increasing both the TPR and the FP Rate significantly, which is the opposite of the expectation. The ROC curves in Fig. 4 also indicate the classification deficiency of the SVM model, as it is close to the diagonal, *i.e,* AUROC close to 0.5.

A difficulty found in our experiments was related to unbalance data, as we have much fewer ODD samples. To solve this problem, we chose a loss function that allows the use of different weights for each class to obtain balance. We observe that, according to the given weight, the model can improve the detection of one label and make the detection of the other worse in the same proportion. So we chose values of w_1 and w_2 that best maximized both classes. This fact can be seen in the GCOOD's TNR and TPR (Table II). Although TNRs are

TABLE II
PERFORMANCE OF OOD DETECTORS TRAINED ON DATA PRODUCED BY RESNET-152 AND APPLIED ON DATA FROM FOUR NETWORKS.

OOD Detector	Source of Samples	Metric			
		TPR	TNR	F1	AUROC
GCOOD	ResNet	0.6450	0.7765	0.4020	0.7829
	VGG	0.6400	0.7645	0.3893	0.7760
	DenseNet	0.4340	0.8185	0.3210	0.7360
	SqueezeNet	0.7440	0.5880	0.3215	0.7061
SVM	ResNet	0.7970	0.1898	0.2134	0.4869
	VGG	0.8350	0.1657	0.2176	0.5066
	DenseNet	0.8210	0.1800	0.2172	0.5084
	SqueezeNet	0.7340	0.2778	0.2162	0.5041

a little higher, they are values close to TPR. By changing the weights empirically, it was possible to find TNR of 0.92, but with TPR of 0.0. In addition to the weights, another empirical observation concerning obtaining better TNR and TPR rates was the increase of the batch size. A more significant variation in the F1-Score was observed with larger batches. But when the model weights are batch normalized, the variation tends to decrease, and these rates' values improve.

VI. CONCLUSIONS AND FUTURE WORK

We presented the GCOOD, a single-sample OOD detection approach to be coupled with any trained discriminative classifier. The GCOOD is a CNN, whose input is a Softmax vector of class probabilities produced by some standard classification model. Our detector indicates whether the sample data related to the vector input is an OOD sample for the already learned class distributions. Most of the existing OOD detectors rely on metric values computed by intermediate processing steps of a given classification model. Although GCOOD is trained on classification vectors produced by instances of a given discriminative classifier, it is important to note that GCOOD only uses the output of these models. It is not related to any intermediate stages of the classifiers, as are other solutions. As such, one can apply a trained GCOOD instance to the classification results of other classification models, too.

We also presented a new training methodology for OOD detectors based on the spatial analysis of the latent space. We performed dimensionality reduction and vertex (cell) coloring of the Voronoi diagram calculated from representative points of the classes in latent space. Voronoi cell neighboring gives the semantic relation learned by the classification model for the classes. Thus, we used the graph color to select which classes can be removed from the dataset without jeopardizing the training and testing of the OOD sample detector.

We evaluated our GCOOD with images from the CIFAR-10 dataset and four different types of image classification networks, and compared our results to a SVM classifier.

Further explorations may include extending the training methodology and the proposed OOD detector to handle Concept and Data Drift. Data imbalance is also another interesting issue to deal with. We hope that our training methodology and GCOOD may inspire novel researchers in ML and PR.

ACKNOWLEDGMENT

This work was partially supported by CNPq (311.037/2017-8) and FAPERJ (E-26/202.718/2018) agencies.

REFERENCES

[1] S. Bulusu *et al.*, "Anomalous example detection in deep learning: a survey," *IEEE Access*, vol. 8, pp. 132 330–132 347, 2020.
[2] J. Gama, I. Žliobaite, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM Comput. Surv.*, vol. 46, 2014.
[3] M. Masud, L. Khan, J. Han, and B. Thuraisingham, "Classification and novel class detection in concept-drifting data streams under time constraints," *Knowl. Data Eng.*, vol. 23, pp. 859–874, 2011.
[4] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *NIPS*, 2018, pp. 7167–7177.
[5] C. S. Sastry and S. Oore, "Detecting out-of-distribution examples with Gram matrices," in *PMLR*, 2020, pp. 8491–8501.
[6] P. Oberdiek, M. Rottmann, and H. Gottschalk, "Classification uncertainty of deep neural networks based on gradient information," in *ANNPR*, 2018, pp. 113–125.
[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.
[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015, pp. 1–14.
[9] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *CVPR*, 2017, pp. 2261–2269.
[10] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5mb model size," arXiv: 1602.07360, 2016.
[11] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.
[12] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *ICLR*, 2017.
[13] S. Marsland, "Novelty detection in learning systems," *Neural Comp. Surveys*, vol. 3, pp. 1–39, 2002.
[14] M. Sabokrou *et al.*, "Adversarially learned one-class classifier for novelty detection," in *CVPR*, vol. 6, 2018, pp. 3379–3388.
[15] S. Pidhorskyi, R. Almohsen, D. A. Adjeroh, and G. Doretto, "Generative probabilistic novelty detection with adversarial autoencoders," in *NIPS*, 2018, pp. 6823–6834.
[16] B. Akshaya and M. T. Kala, "Convolutional neural network based image classification and new class detection," in *PICC*, 2020, pp. 1–6.
[17] N. Lu, G. Zhang, and J. Lu, "Concept drift detection via competence models," *Artif. Intell.*, vol. 209, pp. 11–28, 2014.
[18] N. Lu *et al.*, "A concept drift-tolerant case-base editing technique," *Artif. Intell.*, vol. 230, pp. 108–133, 2016.
[19] V. Losing *et al.*, "KNN classifier with self adjusting memory for heterogeneous concept drift," in *ICDM*, 2016, pp. 291–300.
[20] I. Zliobaite and J. Hollmén, "Optimizing regression models for data streams with missing values," *Mach. Learn.*, vol. 99, pp. 47–73, 2014.
[21] A. D'Amour *et al.*, "Underspecification presents challenges for credibility in modern machine learning," arXiv: 2011.03395, 2020.
[22] J. Dong, W. Socher, R. Li, L. J. Li, K. Li, and L. Fei-Fei, "ImageNet: a large-scale hierarchical image database," in *CVPR*, 2009, pp. 248–255.
[23] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: fast and flexible image augmentations," *Information*, vol. 11, no. 2, p. 125, 2020.
[24] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *ICLR*, 2016, pp. 1–16.
[25] F. Pedregosa *et al.*, "Scikit-learn: machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.
[26] A. Hagberg *et al.*, "Exploring network structure, dynamics, and function using NetworkX," in *SciPy*, 2008, pp. 11–15.
[27] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures," in *ICML*, 2013, pp. 115–123.
[28] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: a novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6765–6816, 2017.
[29] L. Biewald, "Experiment tracking with weights and biases," 2020. [Online]. Available: https://www.wandb.com/