Layer-wise Activation Cluster Analysis of CNNs to Detect Out-of-Distribution Samples

Daniel Lehmann and Marc Ebner

Institut für Mathematik und Informatik, Universität Greifswald
Walther-Rathenau-Straße 47, 17489 Greifswald, Germany
{daniel.lehmann,marc.ebner}@uni-greifswald.de

Abstract. Convolutional neural network (CNN) models are widely used for image classification. However, CNN models are vulnerable to out-of-distribution (OoD) samples. This vulnerability makes it difficult to use CNN models in safety-critical applications (e.g., autonomous driving, medical diagnostics). OoD samples occur either naturally or in an adversarial setting. Detecting OoD samples is an active area of research. Papernot and McDaniel [43] have proposed a detection method based on applying a nearest neighbor (NN) search on the layer activations of the CNN. The result of the NN search is used to identify if a sample is in-distribution or OoD. However, a NN search is slow and memory-intensive at inference. We examine a more efficient alternative detection approach based on clustering. We have conducted experiments for CNN models trained on MNIST, SVHN, and CIFAR-10. In the experiments, we have tested our approach on naturally occurring OoD samples, and several kinds of adversarial examples. We have also compared different clustering strategies. Our results show that a clustering-based approach is suitable for detecting OoD samples. This approach is faster and more memory-efficient than a NN approach.

Keywords: CNN · Out-of-Distribution Detection · Clustering.

1 Introduction

Convolutional neural network (CNN) models have increasingly been used for image classification due to their great performance [18, 25]. However, a high performance is only obtained on in-distribution samples. The performance can drastically decrease in the presence of out-of-distribution (OoD) samples. In-distribution samples are samples that are drawn from the training distribution of the model. OoD samples, in contrast, are drawn from a distribution that is different from the training distribution. Distributions different from the training distribution can occur either naturally (e.g., objects presented in environments or occlusions not seen during training) [21] or in an adversarial setting (in the latter case OoD samples are usually referred to as adversarial examples) [3, 16, 47]. The vulnerability of CNN models to OoD samples makes it difficult to use CNNs in safety-critical applications (e.g., autonomous driving, medical diagnostics).

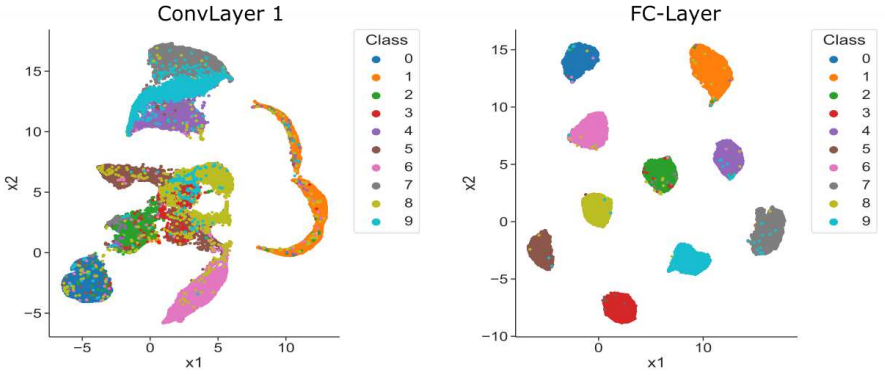

Fig. 1. 2D-Projections (created by UMAP) of the activations of the MNIST training data at the first convolutional layer (ConvLayer 1) and the output layer (FC-Layer) of the CNN (CNN setup as described in section 4.1)

In safety-critical applications, the reliability of CNN models can be improved by using a confidence estimate of the predictions made by the model. This confidence estimate should be high for in-distribution samples and low for OoD samples. A naive approach to obtain such an estimate is to use the softmax scores of the network output. Unfortunately, the softmax scores do not provide a reliable confidence estimation [14, 19]. To find a better confidence estimate, extensive research has been conducted. For instance, a promising approach, named Deep k-Nearest Neighbors (DkNNs), was proposed by Papernot and McDaniel [43]. Their method is based on the assumption that in-distribution samples of the same class are usually close together in feature space at each network layer. OoD samples, in contrast, can be close to in-distribution samples of a different class at each network layer. Hence, at inference, they check if an incoming sample is in-distribution or OoD (regarding a trained model) by using a k-nearest neighbor (kNN) search: When the sample is fed into the model, at each layer they identify the majority class of the kNNs of that sample among the training samples of the model (in feature space of that layer). They calculate a confidence estimate (credibility) using this majority class from each layer. If the majority class is the same among all layers, the confidence of the prediction will be high. If the majority class varies heavily between the different layers, the confidence of the prediction will be low. However, DkNNs have the following disadvantages: 1) Inference is slow because a kNN search requires comparing the incoming sample to a high number of training samples. DkNNs use an approximate kNN approach based on local-sensitive hashing [43]. An approximate approach is faster than a naive kNN search (i.e., compare with every sample), but it still requires many comparisons. Additionally, their method does not perform a kNN search only once but for each layer. This can be time-consuming for large training sets in particular. 2) To be able to perform a kNN search they need to store the whole training set for inference.

To address these issues we examine an alternative approach based on clustering. We state the following research question: Can we perform a cluster analysis at each layer instead of a kNN search to detect OoD samples? A clustering-based approach has the following advantages over DkNNs: 1) Inference is faster. Instead of comparing the incoming sample to a high number of training samples, we simply apply a cluster model (learned from the training set) on the incoming sample at each layer and compare the result to the cluster statics of the training set. 2) For inference we do not need to keep the whole training set but only the clustering model of each layer and the cluster statics of the training set. 3) In contrast to a kNN search, clustering should be able to better handle lower layers. Lower layers try to detect low-level features [49]. However, low-level features are not necessarily class-specific (e.g., a soccer player and a baseball player share some low-level features like shape features of the person, the sportswear, or the grass in the background). As a consequence, at lower layers samples of different classes are usually heavily mixed (illustrated in Figure 1). A kNN search only tries to identify the majority class among the kNNs of a sample. At lower layers, this majority class does not need to be the same as the class of the sample (even if the sample is in-distribution). Our clustering-based approach, in contrast, tries to keep the information about all classes in the cluster by calculating a class distribution statistic of that cluster (i.e., what fraction of the samples in the cluster belongs to a certain class?). Besides these advantages, our approach keeps the following favorable properties of DkNNs: 1) OoD samples do not have to be generated for our detection approach. This would be difficult because we do not know all possible OoD samples in advance that can occur in practice. 2) The CNN model for which we want to detect OoD samples does not have to be re-trained. Our contributions are as follows: 1) We examine if a clustering-based approach is suitable to detect OoD samples for CNN models. 2) We compare different clustering strategies for the proposed approach.

2 Related Work

There have been other studies that also propose to use hidden layer activations for detecting OoD samples. Cohen et. al. [9] use a kNN approach in combination with sample influence scores to detect OoD samples. Crecchi et. al. [10] suggest using the hidden layer activations to learn a kernel density estimator for OoD detection. Li and Li [30] use convolutional filter statistics to learn a cascade-based OoD detector. Metzen et. al. [39] propose to add a detector subnetwork at a hidden layer of the CNN. Chen et. al. [6] use the hidden layer activations to train a meta-model that computes the confidence of the model prediction. Huang et. al. [22] have observed that OoD samples concentrate in feature space. They propose to use a threshold on the distance to the center of the concentrated OoD samples to decide if an incoming sample is also OoD. However, in contrast to our approach, these methods are either computationally more expensive [9, 43], require OoD samples to create the OoD detector [22, 30, 39], or are more complex [6, 10]. None of these methods uses clustering to detect OoD samples. Chen et.

al. [5] have proposed an approach based on clustering. They try to identify if a training set was poisoned to trigger backdoor attacks on CNN models trained on this dataset. In contrast, we detect if an incoming sample, during inference, is an OoD sample for the trained CNN model. Moreover, Chen et. al. apply clustering only on the last hidden layer. We apply clustering on multiple hidden layers.

A number of other approaches to detect OoD samples (especially adversarial examples) have been suggested. There are approaches that detect OoD samples based on generative models [29, 38, 45], using a special loss function [28, 42], Baysian Neural Networks [4, 15], characterizing the adversarial subspace [33], self-supervised learning [20], energy scores [32], techniques from object detection and model interpretability [7], augmenting the CNN with an additional output for the confidence estimate [12, 17], or by perturbing the training images [31, 48].

3 Method

A CNN model f is trained using a training dataset (X^D, Y^D) to classify samples X^I into one of C classes at inference. However, the model f can fail to predict the correct class for a sample $x^I \in X^I$, if the sample x^I is an OoD sample. To detect if x^I is OoD, we perform a layer-wise cluster analysis of the CNN model activations of x^I. Our approach is based on the work of Nguyen et. al. [41]. Nguyen et. al. apply clustering on the layer activations to visualize different features learned by each neuron of the model. In contrast, we use clustering to detect if a sample is OoD regarding the model f. Our approach is organized into two phases: 1) Before inference, at each layer L, we learn a clustering model g_L from the layer activations of the training data (X^D, Y^D) of f. 2) At inference, at each layer L, we apply the learned clustering model g_L on the activations of sample x^I. As a result, at each layer L, we obtain the cluster $K^L_{x^I}$ in which sample x^I falls. Finally, based on the cluster $K^L_{x^I}$ at each layer L, we determine if x^I is OoD or in-distribution. In the following, we describe both phases in detail.

Before Inference: 1) X^D has N training samples x^D. All N samples x^D are fed into model f. 2) At each (specified) layer L: (a) We fetch the activations of each sample x^D. At convolutional layers (ConvLayers), the activations of each sample x^D are in cube form. At linear layers, the activations of each sample x^D are in vector form. (b) We transform the activations of each sample x^D into vector form. However, we only need to do this for ConvLayers as the activations of linear layers are in vector form already. In either case, we obtain N vectors $A^L_{x^D}$ (one for each sample x^D) of length M_L. We concatenate all N vectors $A^L_{x^D}$ to a matrix A^L of size $N \times M_L$. (c) We need to project the activation matrix A^L from $N \times M_L$ down to $N \times 2$ using dimensionality reduction. This is a necessary preprocessing step for clustering as clustering usually does not work well in high dimensions [5]. As a result, we obtain the projected activation matrix $p_L(A^L)$ of size $N \times 2$ and the projection model p_L. (d) We try to find clusters in the projected activation matrix $p_L(A^L)$. As a result, we obtain $1, ..., k_L$ clusters and the clustering model g_L. (e) For each found cluster K^L_i $(i \in 1, ..., k_L)$, we

calculate a cluster statistic $S^L(K_i^L)$ expressing how much each of the C classes $c_1, ..., c_C$ is represented in cluster K_i^L at layer L in % ($cfrac^L_{K_i^L}(c_j)$).

$$\begin{aligned} S^L(K_i^L) &= \left\{ \left(c_j, cfrac^L_{K_i^L}(c_j) \right) \;\middle|\; c_j \in c_1, ..., c_C \right\} \\ cfrac^L_{K_i^L}(c_j) &= \frac{|(x^D, y^D_{y==c_j})|}{|(x^D, y^D)|}\,, \qquad \forall\, (x^D, y^D) \in K_i^L \end{aligned} \tag{1}$$

3) Finally, we store the projection model p_L, the cluster model g_L and the cluster statistic $S^L(K_i^L)$ for each found cluster K_i^L of each layer L.

At Inference: 1) The sample x^I is fed into the model f. 2) At each (specified) layer L: (a) We fetch the activations of the sample x^I. At ConvLayers, the activations of x^I are in cube form. At linear layers, the activations of x^I are in vector form. (b) We transform the activations of sample x^I to vector form. Again, we only need to do this for ConvLayers as the activations of linear layers are already in vector form. As a result, we obtain an activation vector $A^L_{x^I}$ of size $1 \times M_L$. (c) We apply the projection model p_L (that was learned from the training data (X^D, Y^D) before inference) on the activation vector $A^L_{x^I}$. As a result, we obtain a projected activation vector $p_L(A^L_{x^I})$ of size 1×2. (d) We apply the cluster model g_L (that was learned from the training data (X^D, Y^D) before inference) on the projected activation vector $p_L(A^L_{x^I})$. As a result, we determine to which cluster $K^L_{x^I}$ the sample x^I at layer L belongs (among the clusters $1, ..., k_L$ identified in the training data (X^D, Y^D) before inference). (e) We identify all classes that are in the cluster $K^L_{x^I}$ using the cluster statistic $S^L(K^L_{x^I})$ for cluster $K^L_{x^I}$. However, we only consider those classes c_j ($j \in 1, ..., c_C$) whose occurrence (in %) in the cluster is higher than a given threshold t (i.e., $cfrac$ of class c_j in cluster $K^L_{x^I}$ must be greater than t) resulting in a modified cluster statistic $S'^L(K_i^L)$. This threshold t is necessary as there are usually outliers in the training dataset (X^D, Y^D). Hence, some clusters contain only very few samples of a certain class (i.e., the occurrence of that class is low). As a result, we obtain a set of classes $cset^L(x^I)$ for the sample x^I at layer L whose occurrence in cluster $K^L_{x^I}$ is greater than a given threshold t. If x^I is an in-distribution sample, the class of x^I will probably be one of the classes in $cset^L(x^I)$.

$$\begin{aligned} cset^L(x^I) &= \left\{ c_j \;\middle|\; c_j \in S'^L(K^L_{x^I}) \right\} \\ S'^L(K^L_{x^I}) &= \left\{ \left(c_j, cfrac^L_{K^L_{x^I}}(c_j) \right) \;\middle|\; c_j \in c_1, ..., c_C \;\wedge\; cfrac^L_{K^L_{x^I}}(c_j) > t \right\} \end{aligned} \tag{2}$$

3) The set $cset^L(x^I)$ does most likely not contain the same classes at each layer L. The set usually contains more classes at lower layers than at higher layers. This is caused by the type of feature each layer L tries to detect. Lower layers detect low-level features. Low-level features are typically not class-specific (e.g., a soccer

player and a baseball player share some low-level features like shape features of the person, the sportswear, or the grass in the background). As a consequence, at lower layers, training samples of different classes are usually close together (illustrated in Figure 1). Hence, the identified clusters at lower layers contain training samples of several different classes as well. Higher layers, in contrast, detect high-level features. High-level features are typically class-specific. The higher the layer, the more class-specific the features it detects. As a consequence, at higher layers, training samples of the same class are located increasingly close together, whereas training samples of different classes are located increasingly far apart (illustrated in Figure 1). Hence, the identified clusters at higher layers only contain few classes. The clusters at the final layer ideally contain only one class [49]. This is not surprising. The goal of training a neural network-based classification model is to find a feature representation at the final layer of the network that is linearly separable (usually by softmax) regarding the different classes. As a result, the set $cset^L(x^I)$ contains more classes at lower layers than at higher layers. However, if x^I is in-distribution, all $cset^L(x^I)$ must contain at least one common class. This follows from our assumption: If a sample is in-distribution, it will usually be close to other in-distribution samples of the same class at each layer L. If a sample is OoD, it might be close to in-distribution samples of a different class at each layer L. Thus, we take the intersection of the class sets $cset^L(x^I)$ of each layer L to obtain the overall class set $cset(x^I)$ for the sample x^I.

$$cset(x^I) = \bigcap_L cset^L(x^I) \tag{3}$$

Finally, we determine if the sample x^I is OoD or in-distribution using this overall class set $cset(x^I)$ for x^I: If the class set $cset(x^I)$ is empty, it will probably be OoD. If the class set $cset(x^I)$ is not empty, it will probably be in-distribution.

$$detector(x^I) = \begin{cases} 1\,, & if\ cset(x^I) = \emptyset \\ 0\,, & otherwise \end{cases} \tag{4}$$

4 Experiments

4.1 Experimental Setup

We have conducted several experiments to find out if a clustering-based approach (as described in section 3) is suitable to detect 1) natural OoD samples (section 4.3), and 2) adversarial examples (section 4.4). To find clusters in the activations of the training data in each layer (before inference), we need to 1) transform the activations into vector form (only ConvLayers), 2) project the activations of all samples to 2D, and 3) search for clusters in the projected activations. For each of the 3 steps, we can use several techniques. Our base configuration is based on the clustering method introduced by Nguyen et. al. [41]. To transform the activations of ConvLayers into vectors, we simply flatten the activations (this approach was

also used by Papernot and McDaniel [43]). Then, we concatenate the activation vectors of all samples into a matrix and normalize this matrix (subtract the mean and then divide by the standard deviation). The normalization serves as preprocessing step for projecting the activations. To project the activations, we use a combination of PCA [44] and UMAP [37]. First, we project the activations down to 50D using PCA. Then, we project the activations in 50D further down to 2D using UMAP. Theoretically, using UMAP directly should give better results because UMAP is a non-linear dimensionality reduction technique. However, the activation matrix is usually too large to apply UMAP directly. Thus, we project the matrix down to 50D using a linear method (PCA) first. A similar approach is also used by Nguyen et. al. [41]. Finally, we search for clusters in the projected activations using k-Means [35]. The value for k was chosen based on the best silhouette score [46] of the identified clusters corresponding to k. The silhouette score is a metric to evaluate how well clusters are separated using the mean intra-cluster distance and the mean inter-cluster distance. We use the silhouette score to evaluate the identified clusters because, according to Chen et. al. [5], it works best for evaluating clusters in CNN activations. Finally, the found clusters in each layer are used for detecting natural OoD samples and adversarial examples as described in section 3. Based on our base configuration, we have also tested alternative strategies to identify clusters in layer activations (section 4.2).

All experiments have been conducted using models trained on the MNIST [27] (60,000 training samples), SVHN [40] (73,257 training samples), and CIFAR-10 [24] (50,000 training samples) dataset. For MNIST and SVHN, we have used the same CNN model architecture as Papernot and McDaniel [43]: 3 consecutive ConvLayers using ReLU as activation function followed by a fully-connected output layer. The CNN model for MNIST was trained using the following training parameters: 6 epochs, learning rate (LR) of 0.001, Adam optimizer (test performance: 99.04% accuracy). The CNN model for SVHN was trained using the following training parameters: 18 epochs, base LR of 0.001, multi-step LR-schedule (gamma: 0.1, steps: (10,14,16)), Adam optimizer (test performance: 89.95% accuracy). We have used the activations from all ConvLayers (after ReLU) and the fully-connected layer for detecting OoD samples. For CIFAR-10 (not used by Papernot and McDaniel [43]), we have used a 20-layer ResNet model (using fixup initialization) introduced by Zhang et. al. [50]. The ResNet model was trained using the following training parameters: data augmentation (random crop, random horizontal flip, mixup), 200 epochs, base LR of 0.1, cosine-annealing LR-schedule, SGD optimizer (test performance: 92.47% accuracy). We have used the activations from the first ConvLayer (after ReLU), the output activations of the 3 ResNet blocks, the activations from the Global-Average-Pooling layer, and the fully-connected output layer for detecting OoD samples.

4.2 Comparing Clustering Approaches

Besides our base configuration (described in section 4.1), we have also examined alternative clustering strategies to find clusters in the activations of the training data at each layer. We have exchanged either the clustering algorithm, the

projection technique, or the method to transform ConvLayer activations into vector form. We have tested the following approaches: 1) Clustering algorithm: DBScan [13] (we also tried OPTICS [2] and Agglomerative Clustering [1], but both did not give sufficient results). 2) Projection: (a) a combination of PCA and parametric t-SNE [34] (the original t-SNE cannot be used for our method as it does not learn a model that we can apply to incoming samples at inference), (b) only PCA. 3) Transforming ConvLayer activations: pooling the ConvLayer activations using a kernel of (2,2) followed by flattening the pooled activations (through pooling we may obtain a small translational invariance). All clustering strategies have been evaluated by the median silhouette score over all used layers. The results of our experiment are shown in Table 1.

Table 1. Silhouette scores of tested configurations (best score: 1, worst score: -1)

	DBScan	kMeans
MNIST	0.699	**0.733**
SVHN	0.535	**0.59**
CIFAR10	0.319	**0.677**

	PCA	tSNE	UMAP
MNIST	0.423	0.432	**0.733**
SVHN	0.349	0.369	**0.59**
CIFAR10	0.378	0.382	**0.677**

	Pool	Flat
MNIST	0.721	**0.733**
SVHN	0.561	**0.59**
CIFAR10	0.657	**0.677**

4.3 Natural OoD Sample Detection

We have conducted an experiment to find out if our clustering-based approach is suitable for detecting naturally occurring OoD samples. Before inference, we have identified the clusters in the activations of the training data for each (specified) layer of the model using our base configuration (described in section 4.1). From the identified clusters we computed the cluster statistics for each (specified) layer. At inference, we have used the cluster statistics to check if samples are in-distribution or OoD. As in-distribution samples, we have used the test set of the dataset the CNN model was trained on: MNIST (10,000 test samples), SVHN (26,032 test samples), CIFAR-10 (10,000 test samples). As OoD samples we have used a test set that is different from the dataset the CNN was trained on: the KMNIST [8] test set (10,000 test samples, 7.59% test accuracy) for the MNIST model, the CIFAR-10 test set for the SVHN model (9.24% test accuracy), the SVHN test set for the CIFAR-10 model (9.35% test accuracy). We have applied our approach to each in-distribution and OoD dataset to receive the OoD detection rate for the dataset (i.e., how many samples of the dataset were recognized as OoD in %?). The threshold t was set to $t = 0.01$, $t = 0.05$ and $t = 0.1$. The results of the experiment are shown in Table 2.

4.4 Adversarial Sample Detection

Similar to our experiment in section 4.3, we have also conducted an experiment to find out if our clustering-based approach is suitable for detecting adversarial

examples. We have generated the adversarial examples (using the library torchattacks [23]) from the test set of the dataset the CNN model was trained on: MNIST, SVHN, CIFAR-10. The following methods to create the adversarials for MNIST have been used: FGSM [16] ($\epsilon = 0.25$, 8.05% test accuracy), BIM [26] ($\epsilon = 0.25$, $\alpha = 0.01$, $i = 100$, 0.04% test accuracy), PGD [36] ($\epsilon = 0.2$, $\alpha = 2/255$, $i = 40$, 2.46% test accuracy), PGD_{DLR} [11] ($\epsilon = 0.3$, $\alpha = 2/255$, $i = 40$, 0.85% test accuracy). The following methods to create the adversarials for SVHN have been used: FGSM ($\epsilon = 0.05$, 2.72% test accuracy), BIM ($\epsilon = 0.05$, $\alpha = 0.005$, $i = 20$, 0.79% test accuracy), PGD ($\epsilon = 0.04$, $\alpha = 2/255$, $i = 40$, 2.42% test accuracy), PGD_{DLR} ($\epsilon = 0.3$, $\alpha = 2/255$, $i = 40$, 3.48% test accuracy). The following methods to create the adversarials for CIFAR-10 have been used: FGSM ($\epsilon = 0.1$, 13.21% test accuracy), BIM ($\epsilon = 0.1$, $\alpha = 0.05$, $i = 20$, 0.84% test accuracy), PGD ($\epsilon = 0.3$, $\alpha = 2/255$, $i = 40$, 0.93% test accuracy), PGD_{DLR} ($\epsilon = 0.3$, $\alpha = 2/255$, $i = 40$, 28.0% test accuracy). We have applied our approach to each adversarial set to receive the adversarial detection rate for the dataset (i.e., how many samples of the dataset were recognized as adversarial examples in %?). The threshold t was set to $t = 0.01$, $t = 0.05$ and $t = 0.1$. The results of the experiment are shown in Table 2.

Table 2. OoD detection rates of our detection method (using the threshold t) in % for in-distribution samples (Testset), natural OoD samples (OOD) (MNIST: KMNIST, SVHN: CIFAR-10, CIFAR-10: SVHN), and several adversarial examples (FGSM, BIM, PGD, PGD_{DLR})

	MNIST			SVHN			CIFAR10		
t	0.01	0.05	0.1	0.01	0.05	0.1	0.01	0.05	0.1
Testset	3.6	5.4	5.94	1.35	19.67	42.56	4.24	19.62	63.69
OOD	81.11	86.89	86.97	12.46	70.16	81.61	22.16	42.9	83.87
FGSM	72.29	80.01	80.67	4.98	50.5	71.17	35.77	76.64	95.27
BIM	75.52	81.50	81.54	3.04	37.35	63.76	15.39	45.32	97.25
PGD	81.70	86.61	86.74	3.86	40.58	65.7	14.02	43.56	96.53
PGD_{DLR}	77.46	83.46	83.67	5.84	53.89	75.14	15.84	39.32	84.03

The best detection results, in relation to the false positive rates on in-distribution samples, have been reached using a threshold of $t = 0.05$. However, our results are not directly comparable to DkNNs as DkNNs compute a credibility score (e.g., mean scores MNIST-Test = 0.799, MNIST-FGSM = 0.136, SVHN-Test = 0.501, SVHN-FGSM = 0.237) and not a binary value. Additionally, we measured the inference times on the test sets and the FGSM adversarial examples using (a) our method (MNIST-Test: 32 sec, MNIST-FGSM: 31 sec, SVHN-Test: 83 sec, SVHN-FGSM: 77 sec, CIFAR10-Test: 145 sec, CIFAR10-FGSM: 142 sec), and (b) DkNNs (MNIST-Test: 296 sec, MNIST-FGSM: 255 sec, SVHN-Test: 1217 sec, SVHN-FGSM: 1120 sec, CIFAR10-Test: 783 sec, CIFAR10-FGSM: 825 sec). Our method was significantly faster than DkNNs.

5 Conclusion

In section 1, we stated the following research question: Can we perform a cluster analysis at each layer of the CNN instead of a kNN search to detect OoD samples? Our experiments (section 4) have shown that our approach is able to detect OoD samples at a higher rate than the false positive rate for in-distribution samples. As a result, an approach based on clustering is suitable to detect OoD samples. Furthermore, our experiment (section 4.2) has shown that the projection technique has a crucial influence on the cluster quality. This is not surprising because without a good projection we cannot find good clusters. Best results were obtained using UMAP. This also corresponds to what we have observed visually. Using UMAP typically results in more dense clusters compared to t-SNE or PCA. However, by taking the intersection of the classes in the identified clusters of each layer, we have used a quite simple method to decide if a sample is OoD. Hence, the detection rates for SVHN and CIFAR-10 are often low. The low detection rates may be caused by noise in the data. Some form of calibration is probably needed. We have shown that clustering can be used for OoD detection. It is faster, more memory-efficient, or less complex than other state-of-the-art approaches. In future work, we plan to devise a refined clustering-based OoD detector obtaining improved detection rates. Moreover, the detector should not only give us a yes/no answer, if a sample is OoD or not. To be comparable to DkNNs, our method should compute a credibility score instead.

References

1. Ackermann, M.R., Blömer, J., Kuntze, D., Sohler, C.: Analysis of agglomerative clustering. Algorithmica **69**, 184–215 (2014)
2. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure. In: Proc SIGMOD. p. 49–60. ACM, Philadelphia, PA, USA (1999)
3. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion attacks against machine learning at test time. In: Blockeel, H., Kersting, K., Nijssen, S., Železný, F. (eds.) ECML PKDD. pp. 387–402. Springer, Berlin - Heidelberg, Germany (2013)
4. Blundell, C., Cornebise, J., Kavukcuoglu, K., Wierstra, D.: Weight uncertainty in neural networks. In: Bach, F., Blei, D. (eds.) ICML. vol. 37, p. 1613–1622. PMLR, Lille, France (2015)
5. Chen, B., Carvalho, W., Baracaldo, N., Ludwig, H., Edwards, B., Lee, T., Molloy, I., Srivastava, B.: Detecting backdoor attacks on deep neural networks by activation clustering. In: Espinoza, H., hÉigeartaigh, S.Ó., Huang, X., Hernández-Orallo, J., Castillo-Effen, M. (eds.) Workshop on SafeAI@AAAI. CEUR Workshop, vol. 2301. ceur-ws.org, Honolulu, HI, USA (2019)
6. Chen, T., Navratil, J., Iyengar, V., Shanmugam, K.: Confidence scoring using whitebox meta-models with linear classifier probes. In: Chaudhuri, K., Sugiyama, M. (eds.) AISTATS. vol. 89, pp. 1467–1475. PMLR, Naha, Japan (2019)
7. Chou, E., Tramer, F., Pellegrino, G.: Sentinet: Detecting localized universal attacks against deep learning systems. ArXiv **abs/1812.00292** (2020)

8. Clanuwat, T., Bober-Irizar, M., Kitamoto, A., Lamb, A., Yamamoto, K., Ha, D.: Deep learning for classical japanese literature. ArXiv **abs/1812.01718** (2018)
9. Cohen, G., Sapiro, G., Giryes, R.: Detecting adversarial samples using influence functions and nearest neighbors. In: CVPR. pp. 14441–14450. IEEE, Seattle, WA, USA (2020)
10. Crecchi, F., Bacciu, D., Biggio, B.: Detecting adversarial examples through nonlinear dimensionality reduction. ArXiv **abs/1904.13094** (2019)
11. Croce, F., Hein, M.: Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In: ICML. vol. 119, pp. 2206–2216. PMLR (2020)
12. DeVries, T., Taylor, G.W.: Learning confidence for out-of-distribution detection in neural networks. ArXiv **abs/1802.04865** (2018)
13. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD. p. 226–231. AAAI Press, Portland, OR, USA (1996)
14. Gal, Y.: Uncertainty in Deep Learning. Ph.D. thesis, Univ of Cambridge (2016)
15. Gal, Y., Ghahramani, Z.: Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: Balcan, M., Weinberger, K. (eds.) ICML. vol. 48, pp. 1050–1059. PMLR, New York, NY, USA (2016)
16. Goodfellow, I., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: Bengio, Y., LeCun, Y. (eds.) ICLR. San Diego, CA, USA (2015)
17. Grosse, K., Manoharan, P., Papernot, N., Backes, M., McDaniel, P.: On the (statistical) detection of adversarial examples. ArXiv **abs/1702.06280** (2017)
18. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. pp. 770–778. IEEE, Las Vegas, NV, USA (2016)
19. Hendrycks, D., Gimpel, K.: A baseline for detecting misclassified and out-of-distribution examples in neural networks. In: ICLR. Toulon, France (2017)
20. Hendrycks, D., Mazeika, M., Kadavath, S., Song, D.: Using self-supervised learning can improve model robustness and uncertainty. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) NeurIPS. vol. 32, pp. 15637–15648. CAI, Vancouver, CA (2019)
21. Hendrycks, D., Zhao, K., Basart, S., Steinhardt, J., Song, D.: Natural adversarial examples. ArXiv **abs/1907.07174** (2020)
22. Huang, H., Li, Z., Wang, L., Chen, S., Dong, B., Zhou, X.: Feature space singularity for out-of-distribution detection. ArXiv **abs/2011.14654** (2020)
23. Kim, H.: Torchattacks: A pytorch repository for adversarial attacks. ArXiv **abs/2010.01950** (2020)
24. Krizhevsky, A.: Learning multiple layers of features from tiny images. Tech. rep., Univ of Toronto (2009)
25. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Pereira, F., Burges, C.J.C., Bottou, L., Weinberger, K.Q. (eds.) NIPS. vol. 25, pp. 1097–1105. CAI, Lake Tahoe, NV, USA (2012)
26. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial examples in the physical world. In: ICLR. Toulon, France (2017)
27. LeCun, Y., Cortes, C., Burges, C.: Mnist handwritten digit database. ATT Labs [Online], http://yann.lecun.com/exdb/mnist **2** (2010)
28. Lee, K., Lee, H., Lee, K., Shin, J.: Training confidence-calibrated classifiers for detecting out-of-distribution samples. In: ICLR. Vancouver, CA (2018)
29. Lee, K., Lee, K., Lee, H., Shin, J.: A simple unified framework for detecting out-of-distribution samples and adversarial attacks. ArXiv **abs/1807.03888** (2018)
30. Li, X., Li, F.: Adversarial examples detection in deep networks with convolutional filter statistics. In: ICCV. pp. 5775–5783. IEEE, Venice, Italy (2017)

31. Liang, S., Li, Y., Srikant, R.: Enhancing the reliability of out-of-distribution image detection in neural networks. In: ICLR. Vancouver, CA (2018)
32. Liu, W., Wang, X., Owens, J., Li, Y.: Energy-based out-of-distribution detection. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.F., Lin, H. (eds.) NeurIPS. vol. 33, pp. 21464–21475. CAI (2020)
33. Ma, X., Li, B., Wang, Y., Erfani, S.M., Wijewickrema, S., Schoenebeck, G., Houle, M.E., Song, D., Bailey, J.: Characterizing adversarial subspaces using local intrinsic dimensionality. In: ICLR. Vancouver, CA (2018)
34. van der Maaten, L.J.P.: Learning a parametric embedding by preserving local structure. In: van Dyk, D., Welling, M. (eds.) AISTATS. vol. 5, pp. 384–391. PMLR, Clearwater Beach, FL, USA (2009)
35. MacQueen, J.B.: Some methods for classification and analysis of multivariate observations. In: Cam, L.M.L., Neyman, J. (eds.) Berkeley Symp on Math Stat and Prob. vol. 1, pp. 281–297. Univ of Calif Press (1967)
36. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: ICLR. Vancouver, CA (2018)
37. McInnes, L., Healy, J., Melville, J.: UMAP: Uniform manifold approximation and projection for dimension reduction. ArXiv **abs/1802.03426** (2018)
38. Meng, D., Chen, H.: Magnet: A two-pronged defense against adversarial examples. In: SIGSAC. p. 135–147. ACM, Dallas, TX, USA (2017)
39. Metzen, J.H., Genewein, T., Fischer, V., Bischoff, B.: On detecting adversarial perturbations. In: ICLR. Toulon, France (2017)
40. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning (2011)
41. Nguyen, A., Yosinski, J., Clune, J.: Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. Visualization for Deep Learning workshop, International Conference in Machine Learning (2016), arXiv preprint arXiv:1602.03616
42. Pang, T., Du, C., Dong, Y., Zhu, J.: Towards robust detection of adversarial examples. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) NeurIPS. vol. 31, pp. 4584–4594. CAI, Montreal, CA (2018)
43. Papernot, N., McDaniel, P.: Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. ArXiv **abs/1803.04765** (2018)
44. Pearson, K.: LIII. On lines and planes of closest fit to systems of points in space. London, Edinburgh Dublin Philos Mag J Sci **2**(11), 559–572 (1901)
45. Qin, Y., Frosst, N., Sabour, S., Raffel, C., Cottrell, G., Hinton, G.E.: Detecting and diagnosing adversarial images with class-conditional capsule reconstructions. In: ICLR. Addis Ababa, Ethiopia (2020)
46. Rousseeuw, P.J.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. J. Comput. Appl. Math. **20**(1), 53–65 (1987)
47. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: Bengio, Y., LeCun, Y. (eds.) ICLR. Banff, CA (2014)
48. Xu, W., Evans, D., Qi, Y.: Feature squeezing: Detecting adversarial examples in deep neural networks. ArXiv **abs/1704.01155** (2017)
49. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV. pp. 818–833. No. PART 1 in Lecture Notes in CS, Springer, Zurich, CH (2014)
50. Zhang, H., Dauphin, Y.N., Ma, T.: Fixup initialization: Residual learning without normalization. ArXiv **abs/1901.09321** (2019)