Image Classifier with Dynamic Set of Known Classes

David Mojžíšek[1], Jan Hůla[1]

[1]University of Ostrava, Centre of Excellence IT4Innovations Institute for Research and Applications of Fuzzy Modeling, Ostrava, Czechia

Abstract

The typical classification task is based on the assumption that the model will later only encounter examples of classes available during its training. In practice, this is often not a realistic assumption, because of limitations in obtaining enough labeled training data. This contribution is focused on the case where the model might encounter a sample belonging to a class different from the classes seen in the training phase. The goal is to reject examples of unseen classes with the option of later adding them as representatives of new classes without the need to retrain the backbone model. This is important because the end-user might not be able to re-train the model for any reason. The presented approach is based on metric learning combined with the meta-classifier similar to the approach of Xu et al. [1]. Classified examples are first embedded in a vector space through an encoder trained to capture similarities in the input data. The classification itself is then performed by n, where n is the number of known classes, binary decisions. For each decision, the tested example is compared to the k closest examples from the given class. If the model does not decide that the example belongs to any class, this example is rejected as possibly unknown. The method is tested in a visual data classification task.

Keywords

image classification, metric learning, unseen class detection, open-world classification

1. Introduction

Many machine learning applications do not rely only on the used model, but arguably more importantly on the data used for training. The principal requirement for a supervised image classification task, solved by a deep neural network, is that the model will not just fit the training data, but generalize to the distribution from which the data are sampled. For this purpose, many methods, including data augmentations, have been developed [2].

For simplicity, it could be assumed that the data seen during model testing or deployment are drawn from the same distribution as the training data [3]. However, this assumption can make the model unreliable and prone to failure when data instances different from what the model had seen during its training are encountered.

When dealing with classification tasks, one way to handle this problem is to use a method that can identify unfamiliar examples and reject them, that is, not classify them to any known class. The issue could be that the end-user might want to be able to adapt the classifier, allowing it to recognize some rejected examples as instances of a new class. In this case, a direct solution is to retrain the classifier with respect to all newly available data.

In our work, our goal was to build an image classifier that is able to reject unseen class examples but at the same time enrich the known class set by a new class when a user desires to, without the need to retrain the backbone model. This requirement arises because retraining might not be a preferable way for the end-user. Especially for a user with limited deep learning knowledge and inaccessibility to the original data or hardware to perform the training.

2. Problem description

We will assume that for a given classification problem, there exists a finite[1] set of all possible class labels denoted by $A = \{l_i \mid 1 \leq i \leq K\}$, where K is the number of all existing classes. Furthermore, there are two types of classes, known classes, with labels in the set $S \subseteq A$ and unknown classes, with labels in the set $U \subseteq A$. Those sets are disjoint and each class is *known* or *unknown*, that is, $S \cap U = \emptyset \wedge S \cup U = A$.

For training, we are provided with a dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, where x_i is the i-th image representation and $y_i \in A$ its respective label from the set of all possible labels. The set of all initial known class labels is induced by the training dataset. $S = \{l \mid \exists i \in \mathbb{N} : (x_i, l) \in D\}$. For practical purposes, we will add a *rejected class* label r to a set of known classes and denote it $S_r = S \cup \{r\}$.

The desired classification model $f : \mathbb{R}^n \to S_r$ must be able to assign a known label to any input image representation $x \in \mathbb{R}^n$ or reject it in the following way:

ITAT'22: Information technologies – Applications and Theory, September 23–27, 2022, Zuberec, Slovakia

david.mojzisek@osu.cz (D. Mojžíšek); jan.hula@osu.cz (J. Hůla)

[1]We assume that this set is finite, although its extent might be unknown. For example, if we are classifying dog breeds, we assume there is a finite number of dog breeds and we are not interested in classifying anything that is not a dog. However, the general negative class label, *not a dog* in this case, could be included.

$$f(x) = \begin{cases} r, & \max\limits_{l \in S} P(l \mid x) \leq T \\ \arg\max\limits_{l \in S} P(l \mid x), & otherwise, where \end{cases} \quad (1)$$

T is a probability threshold that is set to a suitable value; for example, T could be 0.5. During the deployment of the model, the user is allowed to add a new class to a set of known classes S (and remove it from U), and the model should retain the classification ability given by 1. The key to the problem is the way we train the model to estimate $P(l \mid x)$. We will describe it in the next section.

3. Proposed approach

In this section, we will describe how the whole classification process works.

3.1. Overview

The whole framework has three crucial components. The first is the *encoder* or embedding network whose task is to take an input image and output its low-dimensional feature vector. Since we are dealing with an image classification task, the suitable option for the encoder is a deep convolutional neural network (CNN). Our approach is specific in the way it handles the encoder training, as will be described later.

The second is *the memory* of all known classes. In the memory, samples of each class are stored in the form of their feature vectors. Memory is the adaptable part of the framework. It can accept a new class simply by adding enough examples. A class could also be removed similarly. The management of the known class set is in the hands of the user, which means that in the framework, no part would automatically update this memory. The storage of only image feature vectors is also beneficial due to the small amount of disk space required.

The third and final point is the *classifier*, which classifies a tested example by comparing it with selected sets of known classes. In principle, it will take the feature vector of a query image and evaluate it as positive (the same class) or negative (a different class) with respect to the k selected feature vectors of one class from the memory.

To our knowledge, the closest recent work that offers a solution for the problem is the framework, originally proposed by Xu et al. [1] named L2AC (Learning to Accept Class). Their approach was successfully tested in the context of text classification. In this work, we study the option of utilizing a similar approach for images and propose some changes to the model as well as to the training procedure.

As mentioned above, the framework consists of two deep learning models with trained weights. In the following sections, both are described, including the way in which they are trained.

3.2. Separate training of encoder and classifier

3.2.1. The Encoder

The purpose of an encoder is to find a way in which images could be represented as low-dimensional vectors that somehow capture their most important features. For example, an encoder can take an image of size $224 \times 224 \times 3$ and produce its representation in dimensions 1280, $EN : \mathbb{R}^{224 \times 224 \times 3} \rightarrow \mathbb{R}^{1280}$.

These feature vectors can be passed to an ML algorithm such as SVM or MLP to assign the original input image class. On the one hand, the desired feature vector should be invariant to certain transformations of the input data, which are not important to the classifier decision. On the other hand, these representations must be sufficiently discriminative for different classes.

At the moment, one of the most widely used tools for feature extraction in the context of computer vision are *Convolutional Neural Networks* (CNNs), that can learn to recognize features by adjusting weights without handcrafting [2]. The weights in CNNs are typically trained end-to-end to perform a specific task on the training data. For example, supervised classification with a softmax classifier, object detection, or semantic segmentation. A simple way to obtain a feature representation of the input image is by taking one of the hidden representations in a CNN. Typically, outputs of the last hidden layer are taken.

Training an encoder whose output can capture similarities and dissimilarities in the domain of input data is crucial to our task. In our case, the classification is done by comparing the input image vector representations with the closest classes and images in the memory.

There are several ways to measure the distances between points in $\mathbb{R}^n$, for example, the cosine distance or the Euclidean distance. Learning input representations for a specific distance calculation is called metric learning [5]. In our work, we measure the distances between classified input and image stored in the memory by the cosine distance in the feature space:

$$dst(x_1, x_2) = 1 - \frac{x_1 \cdot x_2}{\|x_1\| \|x_2\|},$$

$\|\cdot\|$ is the Euclidean norm, and "$\cdot$" in the numerator denotes the standard dot product. For that reason, we decided to train an encoder with respect to that distance by

[2] However, transformers [4] are performing very well and could replace CNNs, due to their efficiency and accuracy. Their drawback is that they are more expensive to train.

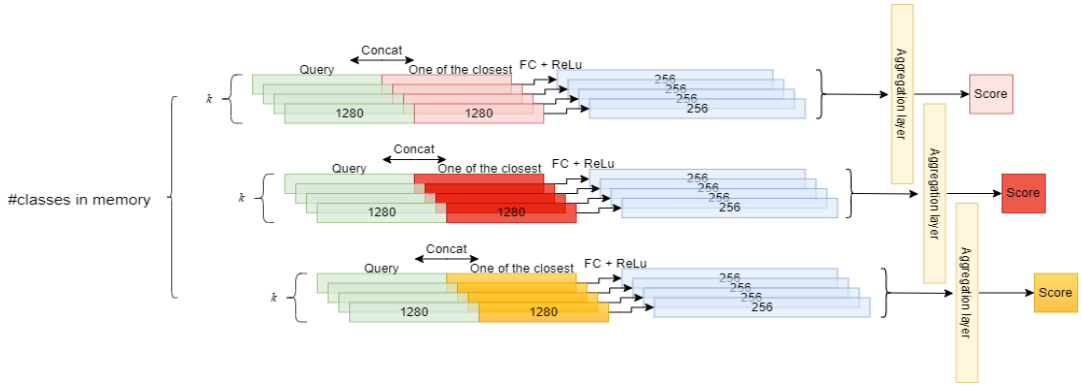

Figure 1: Overview of the classification part of the framework. Classification is performed as a binary decision for each class in the memory. In our setup, the aggregation layer is a transformer.

adding the triplet loss, with the cosine distance function, component to the overall loss when training the encoder CNN.

Our encoder is trained with two objectives. One is a traditional softmax classifier performed to classify the input into one class present in the training data with the Cross-Entropy loss and the second is the Siamese network trained with the triplet loss [6]. Siamese network refers to a deep neural network that runs inputs through itself with the same weights to get multiple outputs and perform an operation on them. In our case, the shared CNN calculates embeddings for the selected image from the training data (an anchor x_a), one image from the same class (positive x_p), and one from a different class (negative x_n). The triplet loss, with some margin ($m > 0$), can be calculated as follows:

$$L_{triplet}(x_a, x_p, x_n) = \max\{dst(x_a, x_p) - dst(x_a, x_n) + m, 0\}.$$

3.2.2. Classifier

The classifier works with the feature representations produced by the encoder. Its goal is to calculate the probability that a given query image (with representation x) belongs to a class with label l: $P(l \mid x)$. This probability is estimated by comparing it to its k closest image representations from a class l, which are stored in the memory. The procedure of calculating the probability score is shown in Figure 1.

Our goal is to train this classifier so that it generalizes both for seen and unseen classes, and the decision is independent of the number of currently known classes. For that reason, the classification of an image is always done with respect to one class.

To make a decision about one class, the query image x is first concatenated into each of the k closest feature representations (we use feature vectors of size 1280). For the implementation, the feature vectors are not just directly concatenated, but the input representations are obtained by concatenating their element-wise absolute difference and sum. Each of these representations is then transformed by a fully connected linear layer ($g(x) = xW^T + b$) with shared weights and is followed by ReLu [7], which transforms the input into the dimension 256.

Applying the linear layer to all k representations yields new k representations, which must be converted to a single probability score. For this purpose, we deploy *transformer*. A transformer treats those k representations as node features of a fully connected graph and updates them in a given number of steps corresponding to the number of layers in the transformer model. Updates are performed with respect to all other nodes with the attention mechanism. In the end, the representations created by the transformer are pooled by the mean operator, and the final representation is then processed by a linear layer with the desired output dimension for binary classification (either 1 for a sigmoid or 2 for softmax).

The advantage of this approach is that the score can be calculated for any class label l in the memory, that is, for any class for which there are enough instances to compare the query image. On the contrary, a traditional CNN classifier has a fixed output size, which can include the unit for the unknown class, but adapting it to a new class requires additional training.

Since the encoder model is trained separately, the feature vectors of the images in the training set will not change during classifier training. It enables us to calculate them only once and pre-calculate all negative and positive training pairs (pair creation is depicted in the Figure 2).

For each training image, there is one positive training pair and n negative training pairs. The n is another hyperparameter and corresponds to the number of clos-

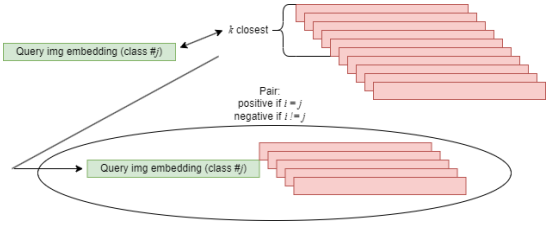

Figure 2: The classifier in our case is trained to make a binary decision. For a given query image x and a given set of k images from mutually the same class, it decides up to what degree x belongs to this class or not. The k images are taken as k closest images to the query image (by their a distance in the embedding space).

est classes that will be selected as negatives. Negative classes are selected by calculating the distance between the image feature representations and the mean feature representation for each different training class. Selecting the closest examples makes sense because those should be the hardest for the classifier to distinguish.

3.3. Joint training

Until now, both models had been trained separately. Another option is to connect both trained components and learn the entire model end-to-end. This means that the loss function could be assembled from multiple components: triplet loss, Cross-Entropy loss for classification, and Cross-Entropy loss for binary query-class evaluation.

This setup requires more computing resources since the input consists of several raw images. During learning, the distances between the examples change continuously and should be reflected in the selection of pairs for the classifier. The joint model accepts an input image with a label (used for supervised classification and as an anchor for triplet loss computation), a set of positive images (which are used to create a positive pair, and one of them is selected as positive for triplet loss), and a set of negative images from one selected class (to create a negative pair, and one of them is selected as negative for triplet loss).

This time, we simplified the selection of positive and negative pairs in the joint training setup. After each epoch, we recalculate all the embeddings and the class means. For a selected image, we choose $k = 5$ positive images from the same class to make a positive pair and $k = 5$ random negative images from a different class that are randomly selected from $n = 5$ classes with the closest mean in the epoch. The loss of the joint model is calculated as follows:

$$L_{joint} = L_{triplet} + L_{CEclass} + L_{CEneg} + L_{CEpos},$$

where $L_{triplet}$ is the triplet loss described earlier, $L_{CEclass}$ is the cross-entropy loss of supervised classification and L_{CEneg}, L_{CEpos} are pair classifier cross-entropy losses for positive and negative pairs. Cross Entropy loss is widely used for training multi-class classification models and its description can be found in the PyTorch [8] documentation. PyTorch was also used for the implementation of this work.

Training the model in this way enables the use of image augmentations throughout the whole training. When trained separately, the input images were only augmented during encoder training. This type of training is much slower, and when trained on a single GPU, the possible batch size drops significantly.

4. Experiments

In this section, we are going to describe an experiment performed on the selected dataset. We are comparing two different ways of training the classifier and the encoder.

1. Training the encoder separately and then training the classification model only with embeddings.
2. Training the encoder and the classification model together.

In both cases, the encoder backbone model is EfficientNet-B0 [9] with pre-trained weights. The softmax classification layer was removed and the model output is a feature vector of size 1280.

4.1. Dataset

To see whether the approach works, we tested it on images from the BIRDS 400 - SPECIES IMAGE CLASSIFICATION dataset [3]. The data set was selected due to the properties required to test the approach. First, the dataset should have enough classes so that we can make a split between seen and unseen classes, and have enough known samples to train the model for feature extraction. Secondly, we would like to use the dataset in which classified objects are from a similar domain since we want to learn and leverage intra-class and inter-class similarities and differences in the feature space. It could be assumed that if the unseen class has features never seen during training on known classes, the encoder network might not be able to capture them. The whole approach is built on the idea that features from known classes can be transferred to unknown classes. Figure 3 shows a few samples from the dataset used.

[3]Dataset is available on Kaggle: https://www.kaggle.com/datasets/gpiosenka/100-bird-species

4.1.1. Dataset Split

We did not use the original dataset train/validation/test split, but merged these splits together. For the purpose of this work, only selected classes are used. Of all 400 classes, 69 were randomly chosen as known and 10 as unknown. We divide them into training, validation, and test sets in the ratio $0.8 : 0.1 : 0.1$.

4.1.2. Separate training

For separate training, the encoder was trained with the Lookahead optimizer (with Adam, $lr = 0.001$) [10]. The batch size was set to 4 and the model was trained for a supervised classification task with 69 training classes. The loss function consists of the Cross-Entropy loss and the triplet loss calculated on triplets sampled from the same set of classes. We trained the model for 20 epochs.

After that, all images in the training dataset were transformed into their feature representations, and the training pairs were saved. These training embeddings were also used as a part of the memory for the classification of the test set. For each image in the training dataset, one positive pair and $n = 5$ negative pairs from the closest classes (based on their mean feature embedding) were sampled. For each negative class, the $k = 5$ closest examples to the given image were taken. This resulted in a total of 58572 pairs of which 9762 were positive.

Training of the classification network was performed for 30 epochs with all negative and positive pairs and with Cross entropy loss for binary classification. The batch size was set at 32 and the SGD optimizer was used ($lr = 0.0001$).

4.1.3. Joint training

For joint training, the images were sampled directly and no embeddings were stored. We trained both the encoder with the triplet loss and the cross-entropy loss for supervised known image classification and the binary classifier simultaneously. The sampling process was previously described. The model was trained with batch size 4 for 20 epochs and the SGD optimizer ($lr = 0.0001$, because higher learning rates did not lead to generalization). The images were augmented before they were passed to the network.

4.2. Results

We show the results obtained from the classification of the test set for both training approaches. The results of the separate training are shown in Tables 1 and 2, and the results of the joint training are given in Tables 3 and 4. Tables 1 and 3 show the classification with only known classes in memory. In that case, we are interested in the classification of known classes and the rejection rates for unknown and known classes. The results for the case where the 10 unseen classes were added to the memory are shown in Tables 2 and 4. In this case, the classification accuracy of the known and unknown classes is being measured, together with the frequency of rejected examples (now the rejection rate is desired to be low) and the overall classification error (for both known and unknown classes). The results are obtained for different rejection thresholds.

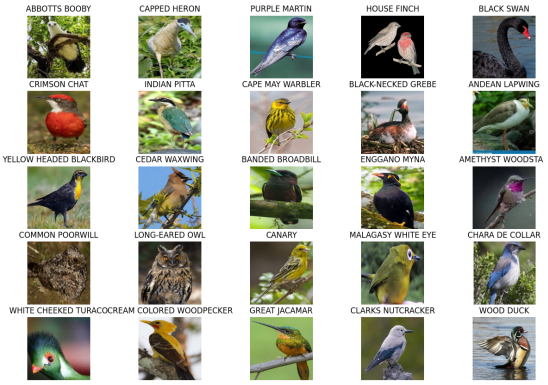

Figure 3: The sample from the dataset used for framework evaluation. The classes have similar features and the classified object is the most salient object in the image. The size of all images is $224 \times 224 \times 3$.

5. Related Work

As mentioned above, our approach is very similar to L2AC [1], which follows the DOC [11] open-world classifier with the reject option for text document classification. One of the differences is the joint training in our case. Another related work is OpenMax [12], a classifier that can reject examples of unseen images or images made to fool a trained network. It calculates the likelihood of an input producing certain activation patterns in the penultimate layer of the network. Inputs whose activation patterns are distant from the activation patterns of known classes are rejected.

The problem of open set recognition (OSR) was formalized by Scheirer et al. [13] [14]. They introduced the concept of open space risk and relating its minimization to the solution of the OSR problem.

These approaches, as well as ours, do not deal with the problem of novel class discovery, i.e., finding new classes between rejected examples. The possible approach to finding new classes among rejected examples uses clustering performed on rejected examples [15] [16] [17].

Cao et al. [18] proposed a unified end-to-end frame-

work, with the objective of classifying examples in an unlabeled dataset to one of the classes present in the labeled dataset along with the discovery of new classes in the unlabeled dataset. [18]. The discovery of new classes (in the context of unlabeled videos) was also studied by Hůla et al. [19].

Our work can be seen as part of a broader research direction focused on anomalous or out-of-distribution (OOD) data [3] [20]. We have focused on the setup in which OOD data belong to a different category from in-distribution data (ID). More generally, the problem of anomaly detection deals with data that are different from those seen by the model during training. Apart from a different class, this difference could be caused, for example, by faulty sensors.

6. Conclusion

The purpose of this initial study was to test whether the proposed approach is suitable for image classification tasks in an open world setting. We have shown that the model is able to accept new classes with good accuracy, while the ability to classify examples from the original known classes is preserved. The classification of known and unknown classes after their addition to memory was better after joint training. However, with separate training, the model was able to reject unknown class examples more reliably (with a lower threshold).

6.1. Future work

In the planned follow-up study, we will train the models on a larger dataset with a more extensive hyperparameter search. We are especially interested in improving the pipeline and training the model simultaneously for all tasks with proper sampling (metric learning, feature extraction, and classification). Furthermore, testing the approach on more benchmarks is required. For the purpose of this work, we tested only three distance metrics (cosine, Euclidean, and manhattan) with similar performance. More work on the most suitable distance metric search is also ongoing.

Table 1
Separate training: Classification before adding unseen classes to the memory.

	Threshold							
	0.5	0.8	0.9	0.95	0.99	0.995	0.998	0.999
Known classified (%)	93.1	91.6	91	90.1	88.6	86.8	85.9	84.3
Rejected Unknowns (%)	65.1	68	96.3	70.1	77.2	79	82.3	83.6
Rejected Knowns (%)	5.5	7.3	8	9.6	11	12.8	13.9	15.5

Table 2
Separate training - Classification after adding unseen class examples into the memory.

	Threshold							
	0.5	0.8	0.9	0.95	0.99	0.995	0.998	0.999
Known classified (%)	90	89.1	88.5	87.5	86.1	84.3	83.4	81.9
Unknown Classified (%)	79.3	76.3	75.6	73.3	67.6	66	61.6	58.6
Rejected (%)	9.6	12.1	12.9	14.7	18	19.7	22.1	24.3
Misclassified (%)	3.7	2.1	1.9	1.6	1.6	1.6	1.4	1.4

Table 3
Joint training - Classification before adding unseen classes into the memory.

	Threshold							
	0.5	0.8	0.9	0.95	0.99	0.995	0.998	0.999
Known classified (%)	97.3	97.3	97.3	97.1	96.4	96	95.2	93.6
Rejected Unknowns (%)	0	6	9.3	15.3	35	39.9	52.3	65.3
Rejected Knowns (%)	0	0	0	0.5	1.6	2	4.1	5.7

Table 4
Joint training - Classification after adding unseen class examples into the memory.

	Threshold							
	0.5	0.8	0.9	0.95	0.99	0.995	0.998	0.999
Known classified (%)	96.4	96.4	96.4	96.3	95.5	95.3	94.4	93
Unknown Classified (%)	93.1	93.1	93.1	93.1	93.1	93.1	91.9	88.4
Rejected (%)	0	0	0	0.2	1	1.1	4.6	5.9
Misclassified (%)	4.7	4.7	4.7	4.4	4.2	4	2.5	2.1

References

[1] H. Xu, B. Liu, L. Shu, P. Yu, Open-world learning and application to product classification, in: The World Wide Web Conference, 2019, pp. 3413–3419.

[2] X. Ying, An overview of overfitting and its solutions, in: Journal of physics: Conference series, volume 1168, IOP Publishing, 2019, p. 022022.

[3] J. Yang, K. Zhou, Y. Li, Z. Liu, Generalized out-of-distribution detection: A survey, arXiv preprint arXiv:2110.11334 (2021).

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in Neural Information Processing Systems 30 (2017).

[5] L. Yang, R. Jin, Distance metric learning: A comprehensive survey, Michigan State Universiy 2 (2006) 4.

[6] A. Hermans, L. Beyer, B. Leibe, In defense of the triplet loss for person re-identification, arXiv preprint arXiv:1703.07737 (2017).

[7] J. Schmidhuber, Deep learning in neural networks: An overview, Neural networks 61 (2015) 85–117.

[8] A. e. a. Paszke, Pytorch: An imperative style, high-performance deep learning library, in: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, R. Garnett (Eds.), Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[9] M. Tan, Q. Le, Efficientnet: Rethinking model scaling for convolutional neural networks, in: International Conference on Machine Learning, PMLR, 2019, pp. 6105–6114.

[10] M. Zhang, J. Lucas, G. E. Hinton, J. Ba, Lookahead optimizer: k steps forward, 1 step back, Advances in Neural Information Processing Systems 32 (2019).

[11] L. Shu, H. Xu, B. Liu, Doc: Deep open classification of text documents, arXiv preprint arXiv:1709.08716 (2017).

[12] A. Bendale, T. E. Boult, Towards open set deep networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 1563–1572.

[13] W. J. Scheirer, L. P. Jain, T. E. Boult, Probability models for open set recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 36 (2014) 2317–2324.

[14] W. J. Scheirer, A. de Rezende Rocha, A. Sapkota, T. E. Boult, Toward open set recognition, IEEE Transactions on Pattern Analysis and Machine Intelligence 35 (2013) 1757–1772. doi:10.1109/TPAMI.2012.256.

[15] L. Shu, H. Xu, B. Liu, Unseen class discovery in open-world classification, arXiv preprint arXiv:1801.05609 (2018).

[16] E. Fini, E. Sangineto, S. Lathuilière, Z. Zhong, M. Nabi, E. Ricci, A unified objective for novel class discovery, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021, pp. 9284–9292.

[17] K. Han, A. Vedaldi, A. Zisserman, Learning to discover novel visual categories via deep transfer clustering, in: Proceedings of the IEEE/CVF International Conference on Computer Vision, 2019, pp. 8401–8409.

[18] K. Cao, M. Brbić, J. Leskovec, Open-world semi-supervised learning, arXiv preprint arXiv:2102.03526 (2021).

[19] J. Hůla, D. Adamczyk, D. Mojžíšek, V. Molek, Segmenting out generic objects in monocular videos., in: Proceedings of the 21st Conference Information Technologies – Applications and Theory (ITAT 2021), 2021, pp. 123–129.

[20] C. Geng, S.-j. Huang, S. Chen, Recent advances in open set recognition: A survey, IEEE Transactions on Pattern Analysis and Machine Intelligence 43 (2020) 3614–3631.