Novel Object Discovery using Case-Based Reasoning and Convolutional Neural Networks

JT Turner[1], Michael W. Floyd[1], Kalyan Moy Gupta[1], and David W. Aha[2]

[1]Knexus Research Corporation; Springfield, VA, USA
[2]Navy Center for Applied Research in AI;
Naval Research Laboratory (Code 5514); Washington, DC, USA
{*first.last*}@knexusresearch.com
david.aha@nrl.navy.mil

Abstract. The development of Convolutional Neural Networks (CNNs) has resulted in significant improvements to object classification and detection in image data. One of their primary benefits is that they learn image features rather than relying on hand-crafted features, thereby reducing the amount of knowledge engineering that must be performed. However, another form of knowledge engineering bias exists in how objects are labelled in images, thereby limiting CNNs to classifying the set of object types that have been predefined by a domain expert. We describe a case-based method for detecting novel object types using a combination of an image's raw pixel values and detectable parts. Our approach works alongside existing CNN architectures, thereby leveraging the state-of-the-art performance of CNNs, and is able to detect novel classes using limited training instances. We evaluate our approach using an existing object detection dataset and provide evidence of our approach's ability to classify images even if the object in the image has not been previously encountered.

Keywords: Computer Vision, Novel Object Discovery, Deep Learning, Convolutional Neural Networks

1 Introduction

Computer Vision has seen rapid advancement in recent years as a result of Deep Learning (DL) techniques, especially for object classification tasks. DL algorithms are able to leverage large annotated image datasets for training, and achieve significant classification improvement over traditional vision approaches. Convolutional Neural Networks (CNNs) [1] have been a driving force behind these improvements as they are able to use an image's raw pixel values as inputs and learn higher-level features from the training data. Thus, they remove the need for manual feature engineering and extraction, and may learn more discriminative features than those that are hand-crafted by a domain expert. For example, during training a CNN may learn low-level image features like lines or curves, and combine those into increasingly complex features like shapes, wheels, or faces.

Although CNNs greatly reduce the knowledge engineering required by removing the need for hand-crafted features, they do require knowledge about the types of objects that are present in the training images (i.e., an annotation of the object labels). This adds significant bias based on the types of objects that are used to annotate images. For example, an image of an office typically contains dozens of visible objects but may only have labels for a small subset of those (e.g., humans, computers, desks) and treat the others as unlabeled background (e.g., books, pencils, papers). Thus, the CNN is only able to learn to classify objects that the domain expert felt were important enough to annotate. Similarly, the level of granularity of annotations can impact what a CNN learns. For example, the CNN will learn differently depending on if an image of a dog is labelled as *"animal"*, *"dog"*, or as the specific dog breed. These issues can become more significant when you have large datasets containing thousands or millions of annotated images, since it reduces the likelihood that a consistent annotation methodology was used on all images (e.g., different annotators, human error, time-varying methods of annotation). The annotated object types in training images restrict the potential classifications that a CNN can make when deployed; if an object type is not annotated in the training data, the CNN will be unable to classify that object type. For example, if a CNN is trained with images of *airplanes*, *boats*, and *houses*, an image of a *dog* would either be classified as one of those three object classes or not classified at all (i.e., if the confidence was too low).

We propose a case-based approach for novel object detection that uses a combination of raw pixel values and detectable object part information to identify when input images differ noticeably from known object types. Our approach is intended to be used in combination with existing CNN vision approaches and leverage their state-of-the-art performance while addressing some of their limitations. More specifically, our approach makes the following contributions: (1) a method to detect novel object types without prior knowledge of those types; (2) a method to identify variations in images of objects of the same type; (3) an approach that can be used in combination with existing CNN architectures; and (4) an approach that can be used even with small datasets and a single example of each object type. We believe the ability to operate using a small dataset is important given the large dataset requirements that are typically required by existing Deep Learning systems.

The remainder of the paper outlines our case-based novel object detection approach. Section 2 provides background on Convolutional Neural Networks, with Section 3 describing our method for novel object detection and how we leverage CNNs for this task. Section 4 describes our empirical evaluation using an existing object detection dataset. In Section 5, we discuss related work in case-based Deep Learning, case-based Computer Vision, and novel object type detection. Section 6 discusses areas of future work and concluding remarks.

2 Background: Convolutional Neural Networks

The typical architecture of a Convolutional Neural Network has three primary building blocks: *convolutional layers*, *pooling layers*, and *fully-connected layers*. Convolutional layers are composed of *filters* that encode features that will be detected in the input. For example, consider a greyscale image of $n \times n$ pixels used as input to a convolutional layer composed of k filters, each of which are $m \times m$ ($m \leq n$). The filters encode feature patterns that will be identified in the input image. Each $m \times m$ filter is applied to each distinct (and possibly overlapping) $m \times m$ subregion of the input image, with the results stored in new $(n - m + 1) \times (n - m + 1)$ matrix[1], called a feature map. This can be thought of as the filter sweeping across the image, starting from the top left, and applying the filter to each subregion along a row before moving down to the row below (and ending in the bottom right). Each of the k filters are applied in this manner, resulting in k feature maps from the convolutional layer. Thus, the feature map for a particular filter represents the presence of that feature in the various subregions of the image. When a CNN is trained, the filters (i.e., the features to look for) are part of what is learned.

Pooling layers are used to reduce the dimensionality of a convolutional layer's output and for abstraction to avoid overfitting. For example, the convolutional layer in the previous example took an $n \times n$ input and produced a $k \times (n - m + 1) \times (n - m + 1)$ output (i.e., one feature map for each of the k filters). Depending on the values of k, n, and m, this could result in a larger output than the input. Pooling prevents the outputs from growing progressively larger, since in a typical CNN architecture you will have multiple convolutional layers in a series. A common form of pooling is *max pooling* that partitions the layer's input into a set of contiguous non-overlapping $p \times p$ subregions and selects the maximum value contained in each subregion. As was the case with convolutional layers, pooling layers produce k output matrices (i.e., one for each input feature map they receive). For example, if $p = \frac{n-m+1}{2}$, then each input matrix would be downsampled to a 2×2 matrix (i.e., containing the maximum value from the top-left, top-right, bottom-left, and bottom-right regions of the input).

A typical CNN architecture will contain multiple convolutional layers and pooling layers arranged in a series. The input to the first convolutional layer $conv_1$ will be the input image, and its output out_{conv_1} (i.e., the feature maps it produces) will be the input to the first pooling layer $pool_1$. The output of the first pooling layer out_{pool_1} is then used as input to the next convolutional layer $conv_2$, and this sequence of convolutional and pooling layer continues until the output from the nth pooling layer out_{pool_n}. Such an architecture results in early convolutional layers detecting relatively simple features whereas later layers detect increasingly complex features (i.e., patterns of lower-level features). After the final pooling layer, that layer's output out_{pool_n} is flattened from a set of matrices into a single one-dimensional feature vector. For example, if out_{pool_n} produced six 3×3 output matrices, the flattened output would be a feature vector containing 54 values ($6 \times 3 \times 3$).

The final building blocks in a CNN are the fully-connected layers. These layers are typically multilayer perceptrons (MLPs), and use the flattened feature vector output by the final pooling layer as input. For a classification task, the fully-connected layer will

[1] This example assumes a step size of 1, where the center on the filter is moved by 1 pixel at each step. However, in practice the step size can be set as a parameter.

output the class label (or probability of each class label) of the input. During training, the weights used by the MLP to produce the output classification are learned. At a high level, we can think of the convolutional and pooling layers as performing feature extraction on the input image, and the fully-connected layers use those extracted features to perform classification.

3 Case-Based Novel Object Detection

Convolutional Neutral Networks perform supervised machine learning, so their ability to classify the presence of objects in images is directly related to the labeled training data they have available; they cannot detect the correct object type if no annotated training data exists with a label for that object type. If a CNN outputs the confidence in each known class label (i.e., the output of the fully-connected layers), it could, at best, label an input image as *unknown* if none of the possible class labels were above a confidence threshold. For example, if a CNN was trained to classify *airplanes*, *boats*, and *houses*, an image of a *dog* would either be classified as one of the three known classes (i.e., if the CNN output a high confidence for one of the classes) or as *unknown* (i.e., if none of the classes had a high confidence). If several different novel objects are encountered, they would all be classified together into the generic *unknown* class, even if the objects were significantly different from each other. Returning to the example, images of *dogs*, *books*, *space stations*, and *humans* would all be classified together as *unknown*. One solution would be to retrain the CNN after each novel object type is detected. However, this is generally impractical as CNNs require both a large number of labeled training examples (i.e., more than a single training instance) and significant computational time to retrain the fully-connected layers.

We propose a case-based reasoning approach to detect the presence of novel object types and quickly learn from limited training data. Unlike CNNs, a CBR approach can learn using only a single training example and requires no training time. However, our approach does not propose to remove CNNs from the object classification process. Instead, our approach leverages the state-of-the-art performance of CNNs while providing capabilities that alleviate some of their limitations. For the remainder of this section we will largely present the CBR component in isolation, but will discuss how it can be integrated with existing CNN architectures at the end of this section.

Our CBR system encodes each image I_i as a case C_i. Each case is a triple containing the image's feature vector F_i, its set of observable parts P_i, and object label l_i:

$$C_i = \langle F_i, \mathrm{P_i}, l_i \rangle$$

This representation assumes the availability of two functions: *features* and *parts*. The *features* function converts a raw image $\mathrm{I_i} \in \mathrm{I}$, where I is the set of all images, into a feature vector $F_i = \langle f_i^1, \dots f_i^n \rangle \in F$, where F is the set of all feature vectors (*features*: $I \rightarrow F$), composed of n feature values. For the *features* function, we use the convolutional and pooling layers from a CNN to perform this conversion, since they convert a raw image into a flat feature vector. This is essentially a version of the CNN with the fully-connected layers removed such that the CNN is only used for feature extraction. The *parts* function extracts a set of observable parts $P_i = \{p_i^1, \ldots, p_i^{m_i}\} \subseteq P$, where P is the set of all object parts, from image I_i. The number of observable parts in an image m_i is not fixed, so the size and contents of P_i is image-specific. In this work, we consider object parts to be low-level components that make up larger objects. For example, the parts of a *dog* could include its *legs, tail, torso, tail, head,* and *ears*. Although the object parts provide more detail about an object, they are assumed to be generic such that the same parts can be part of numerous object types. Returning to the *dog* example, many mammals would share some or all of the same parts. However, even two instances of the same object may have different observable parts depending on what is visible in the image. In the *dog* example, the *dog's* tail may not be visible depending on where it is facing or its legs may not be visible if the bottom of its body is occluded by another object. The *parts* function requires a separate vision system that can identify these generic object parts from visible images. However, as we will discuss later, while our CBR approach can leverage parts information, it is not strictly necessary for case retrieval. For example, if no parts extraction was possible, each case could contain an empty set of parts ($P_i = \emptyset$) and rely only on the feature vector for retrieval. We assume each case has a single object label $l_i \in L$, where L is the set of all object labels. This assumes that each image will contain only a single object of interest. Such an assumption is valid for uncluttered images or, more realistically, when used as part of a Region-Based Convolutional Neural Network (R-CNN) [2]. R-CNNs use a *region proposal* stage to propose subregions of the input image and then classify those subregions individually. Thus, instead of the entire image being used as input to the CNN, each subregion is used as a distinct input to the CNN (i.e., the CNN is run multiple times) and each subregion is used to perform a single classification. In our work, the images stored in cases and used as input to the CBR system could be the image data from these subregions. Using this case representation, the feature vector and set of parts represent the *problem* and the object label is the *solution*.

When an input image is received, either a complete image or a proposed subregion from an R-CNN, object classification is performed using Algorithm 1. In addition to the input image I_{in}, the algorithm uses as input a case base CB, number of nearest neighbors k, feature vector similarity threshold λ_f, and parts similarity threshold λ_p. The algorithm starts by extracting the features and parts from the image (Lines 1 and 2). If the case base is empty (i.e., the CBR system has no training instances), a novel object label is generated using the *generateLabel* function (Line 5). We do not expect this function to generate an informative label based on knowledge of the image (e.g., *dog*, *cat*, *airplane*, *house*) but instead a unique label for the object type (e.g., *class1*, *class2*, *class3*). If the case base is not empty, the top k most similar cases are retrieved from the case base (Line 7). The similarity only considers the feature vector similarity (e.g., using a similarity function based on the Euclidean distance between feature vectors), so no parts information is considered. Cases are only added to the top k if their similarity is above the feature vector similarity threshold λ_f, so it is possible for fewer than k cases to be retrieved. In some situations, no cases will be retrieved if none of the cases are similar to the input image (Line 8). In such a situation, the input image is assumed to be of a novel object type so a new class label is created for it (Line 9).

The previous stages of the algorithm only considered the feature vectors when comparing the input image to cases. The remainder of the algorithm leverages the detectable parts information. The parts of the input image are compared to the parts of each of the top k nearest neighbors (Lines 12-15). The similarity function used (Line 13) is assumed to be a similarity function that calculates set similarity (e.g., Jaccard similarity). Similar to when comparing feature vector similarity, only cases with a parts similarity above the parts similarity threshold λ_p are retained (Line 14). If there were no cases above this threshold (Line 16), the input image is considered to be a novel object type so a novel label is generated (Line 17). Otherwise, the label from the most similar case (based on parts similarity, with feature vector similarity used as a tie-breaker) is used to label the input image (Line 19). Finally, a novel case is created and added to the case base (Line 20) and the object label is returned (Line 21).

Algorithm 1: Object classification using image features and parts

```
Function: $classify(I_{in}, CB, k, \lambda_f, \lambda_p)$ returns $l_{in}$
1  $F_{in} \leftarrow features(I_{in})$;
2  $P_{in} \leftarrow parts(I_{in})$;
3  $l_{in} = \emptyset$;
4  if $CB = \emptyset$ then
5      $l_{in} \leftarrow generateLabel()$;
6  else
7      $topK \leftarrow retrieveTopK(F_{in}, CB, k, \lambda_f)$;
8      if $topK = \emptyset$ then
9          $l_{in} \leftarrow generateLabel()$;
10     else
11         $nn = \emptyset; nnSim = -1$;
12         foreach $C_i \in topK$ do
13             $sim \leftarrow partSim(P_{in}, C_i.P_i)$;
14             if $sim > nnSim$ and $sim > \lambda_p$ then
15                 $nn = C_i; nnSim = sim$;
16         if $nn = \emptyset$ then
17             $l_{in} \leftarrow generateLabel()$;
18         else
19             $l_{in} \leftarrow nn.l_i$;
20 $CB \leftarrow CB \cup \langle F_{in}, P_{in}, l_{in} \rangle$ ;
21 return $l_{in}$;
```

An existing label is only returned when there is a case that is similar to both the input image's feature vector and its parts set. Thus, there are three situations where a novel object label, and therefore a new object class, are created: (1) when the case base is empty; (2) when none of the cases have similar feature similarity; and (3) when there is at least one case with similar features but none of those cases have similar parts. As we mentioned earlier, although parts information is used in the algorithm, it is not strictly necessary. Assuming no parts information is available, the parts set of the input image and all cases will be empty. If the parts similarity function is designed to return maximal similarity when comparing two empty sets, all of the top k cases will be above λ_p and have an equal similarity value. Thus, as long as the top k cases are iterated over in order of descending feature vector similarity (Lines 12-15), the case with the most similar feature vector similarity will be selected as the nearest neighbor and have its label returned.

One of the primary benefits of this algorithm is that it is able to learn using only a single training instance. Once a novel class has been detected (Lines 5, 9, or 17), it is immediately added to the case base and can be used to classify future input images. Similarly, this algorithm can be used even when no existing training data exists (i.e., an initially empty case base). For example, this algorithm could be used from a cold-start to perform object classification without any labeled data. At such a time when sufficient data was collected and annotated, and sufficient time was available, a Convolutional Neural Network could be trained. Once a CNN is trained, the CBR algorithm could run in parallel to the CNN. Assuming the fully trained CNN has superior performance classifying known object types, the CBR system could defer classification for known object types and only interject when a novel class is detected or an input image is most similar to an object class that the CNN has not been trained on (i.e., a previously detected novel class). Thus, the CBR system can be used in situations where it has advantages over the CNN, and defer in other situations.

4 Evaluation

In this section, we evaluate the claim that *our case-based reasoning system can be used to detect and learn from novel object types*. Our evaluation tests the following hypotheses:

- **H1**: Extracting a feature vector representation from images, using a CNN, provides sufficient information for a CBR algorithm to differentiate between object types
- **H2**: The addition of observable parts information improves object classification performance
- **H3**: Our CBR approach is able to detect novel object classes and learn from detected classes
- **H4**: Our CBR approach discovers finer-grained object classes than those provided by the dataset's human annotators

4.1 Data Set

The dataset we use for evaluation is the publicly available *PASCAL-Part Dataset* [3]. It is based on the dataset used for the *Visual Object Classes Challenge 2010*, a Computer Vision competition to recognize objects in realistic scenes. While the *Visual Object Classes Challenge 2010* dataset only contains the annotated object types visible in each image, the *PASCAL-Part Dataset* contains additional annotations for the object parts that are visible in the image. The dataset contains 20 object types: *aeroplane*, *bicycle*, *bird, boat*, *bottle*, *bus*, *car*, *cat*, *chair*, *cow*, *diningtable*, *dog*, *horse*, *motorbike*, *person*, *pottedplant*, *sheep*, *sofa*, *train*, and *tvmonitor*. Each object can have between 0 (*boat*, *chair*, *diningtable*, *sofa*) and 24 (*person*) object parts annotated. However, images of the same object type may have a different number of annotated parts due to object occlusion, object positioning, or annotator error. In addition to providing object part annotations, the *PASCAL-Part Dataset* has several properties that make it a suitable dataset for us to use. The images are realistic real-world images, so most images contain multiple objects (including objects from the 20 annotated object types as well as other unlabeled object types). The objects have varying locations, rotations, sizes, and scales. Additionally, images have different backgrounds (e.g., beach, indoors, forest) and lighting conditions. The annotated objects may be partially occluded, located partially outside the image, or incorrectly labeled by human annotators.

Our work is focused on detecting a single object type in each image, as we justified in the previous section, so we preprocessed the *PASCAL-Part Dataset* to extract only the images with a single annotated object. However, it should be noted that although each image only contains a single annotated object, many of them contain multiple visible objects. The additional object are either objects that are not of the 20 labelled object types, or objects that have been omitted due to annotator error. After preprocessing, 4737 images remained (from an initial dataset size of 10,103).

The *features* function used in Algorithm 1 is a Convolutional Neural Network using the ResNet [4] architecture (i.e., how the various layers are connected). The CNN was trained using the *ImageNet* dataset [5], a dataset containing hundreds of thousands of annotated images. This was performed to learn the filters (i.e., the image features) used by the convolutional layers of the CNN, and after training the fully-connected layers were removed. The output of the CNN is a feature vector of length 2048. Although *ImageNet* is a different dataset than the *PASCAL-Part Dataset*, pretraining a CNN on *ImageNet* learns many general-purpose image features (e.g., lines and shapes). Thus, it allows training a generic *features* function that can be used regardless of domain, and with significantly less time and computational effort than retraining the CNN for each new image dataset. However, it should be noted that due to the size and scope of *ImageNet*, there is likely some overlap with the objects contained in the *PASCAL-Part Dataset* (but none of the labels from *ImageNet* are used during our evaluation). The *parts* function in Algorithm 1 uses the ground-truth parts annotations provided by the dataset (i.e., assumes the presence of a perfect parts extractor). Although in real computer vision tasks the parts would need to be extracted using a separate vision system, we used the provided parts labels in order to remove error during our initial evaluations. Future work will examine how our CBR system's performance is influenced when parts are extracted using a more realistic *parts* function. Thus, each image in the preprocessed *PASCAL-Part Dataset* can be converted into our case representation using the *features* function, *parts* function, and object type annotation.

4.2 Classification Accuracy

Our initial set of experiments aims to evaluate the ability of our CBR algorithm to correctly classify the objects contained in images. More specifically, we examine the classification performance based on what information is used during case retrieval: *feature vector only*, *parts only*, or *both feature vector and parts*. Essentially, these experiments look to confirm that CBR can reasonably discriminate between the various object types and that reasonable data is contained in cases.

In the experiments, we use a variation of Algorithm 1 that does not attempt to identify novel classes; the label from the nearest neighbor is used even if that neighbor is dissimilar. This is achieved by using a non-empty case base (avoiding Algorithm 1, Line 5), and setting $\lambda_f = \lambda_p = 0.0$ (avoiding Algorithm 1, Lines 9 and 17). The experiments used leave-one-out testing, such that each of the 4737 cases are used as input with the remaining 4736 cases used as the case base. The accuracy is measured as the percentage of input cases that have a retrieved object type that is identical to their true object type (i.e., the solution portion of the case). The three variants we test are:

- **Feature Vector Only**: We used $k = 15$ and an empty parts set for all images, thereby only basing similarity on the feature vectors. In practice, this is identical to using $k = 1$ since the case with the highest feature vector similarity will be selected given that there is no influence from parts similarity (i.e., all cases have empty parts sets). We used $k = 15$ to highlight that the various experiments were using similar parameter values.
- **Parts Only**: We used $k = 4736$ so that the entire case base was retrieved, regardless of feature vector similarity. All cases contained parts information. Thus, the most similar case is the case with the most similar parts.
- **Both Feature Vector and Parts**: We used $k = 15$ and all cases contained parts information. Thus, the most similar case is the case with the most similar set of parts from amongst its 15-nearest neighbors (based on feature vector similarity).

Using only a single component of the case for retrieval resulted in lower performance, with a classification accuracy of 80.14% when only the feature vector is used and 88.79% when only the parts are used. The best performance was achieved when both the feature vector and parts were used for retrieval, with a classification accuracy of 91.13%. These results demonstrate that using CBR with only the feature vector provides reasonable classification performance (giving support for **H1**) but that performance can be increased by using both the feature vector and parts information (giving support for **H2**).

4.3 Novel Class Detection

The results in the previous subsection demonstrate the ability of our approach to be used to classify known objects in images. However, the primary motivation of our work is to detect and learn from novel object types in images. In this experiment, we use Algorithm 1 such that is can detect novel object types (i.e., the case base may be initially empty, or either λ_f or λ_p are non-zero values). The experiment starts with an empty case base, and cases are randomly removed from the dataset and used as input to Algorithm 1. After each input, the algorithm stores a case in its case base using the object classification it made for the image (i.e., Algorithm 1, Line 20). Thus, 4737 total inputs are provided to the algorithm, and after the nth input the algorithm will have a case base of size n. The evaluation was designed to simulate how the CBR system would start with no knowledge (i.e., an empty case base) and incrementally learn based on its novel object detection capabilities. The parameters used are $k = 15$, $\lambda_f = 0.45$, $\lambda_p = 0.45$. The thresholds were selected to be relatively low such that they only exclude cases if they are significantly different than the input image. Similarly, the k value was selected such that a neighborhood of similar cases would be retrieved.

We use two metrics to evaluate the algorithm's performance: *Class Purity* and *Class Count Divergence*. *Class Purity* measures, after all 4737 input images have been classified and added to the case base, the percentage of images that are placed in a class where they share a true object type with the majority of other images in that class. Since our algorithm starts with no training data, all classes in the case base are novel classes learned by the algorithm. Thus, we compare whether images with the same algorithm-generated object type classification have the same ground-truth object type classification. For example, the algorithm would be performing well if all images of *dogs* were given the same novel object classification of *class10* (or any other class label, as long as all *dogs* were given the same label). This metric calculates values between 0 and 1 (inclusive), with higher values being better.

Class Count Divergence measures how close the number of detected object types is to the true number of objects types in the dataset. In our dataset, there are 20 true object labels. The motivation for using the metric is to penalize creating an unnecessarily large number of classes. For example, creating 4737 unique class labels would result in a perfect *Class Purity* score but each object label would be overfit to a single image. We use a curved function that has the maximal value when the number of predicted classes $class_{pred}$ is equal to the true number of classes $class_{true}$ and decreases as those values diverge:

$$Class\ Count\ Divergence = \frac{1}{\left(\frac{class_{true} - class_{pred}}{500}\right)^2 + 1}$$

Similar to *Class Purity*, *Class Count Divergence* calculates values between 0 and 1 (inclusive), with higher values being better. The value 500 was selected for use in the *Class Count Divergence* based on the size of the dataset, such that the metric would be below 0.50 if the number of detected classes was larger than approximately 10% of the dataset size. Additionally, we report the *Overall Performance* of the algorithm as the harmonic mean of *Class Purity* and *Class Count Divergence*.

We repeated the experiment 25 times, and Table 1 shows a summary of the results. Based on the *Class Purity*, our approach does a reasonable job detecting novel object types and using those to classify images it encounters in the future. As a baseline, when input images are randomly assigned to 20 object types (i.e., no novel classes are learned), the *Class Purity* is 0.168. The majority of the mistakes made by the algorithm were to provide the same label to objects that are both physically similar and have similar parts. For example, many of the four-legged animals were given the same label, especially in situations where they were small or occluded. Overall, occlusion had a significant impact on performance since it often resulted in very little of the object being visible (i.e., $< 10\%$) and no parts information being available. Even for humans, it was difficult to know that these highly obscured objects were the objects of interest. In fact, it was often the situation that unlabeled objects (i.e., not among the 20 annotated labels) were the most prevalent objects in images. By examining the learned class labels, we found that our algorithm was learning based on these unlabeled object types. However, given that the *Class Purity* metric only considers the 20 annotated object types, the metric is unable to quantify how well the algorithm was able to learn object types that were not annotated in the dataset. Overall, these results provide support for **H3**.

Table 1. Results of novel object type detection over 25 experimental runs

Metric	Mean	Minimum	Maximum	Standard Deviation
Class Purity	0.676	0.572	0.738	0.052
Object Types	121.7	111	133	6.2
Class Count Divergence	0.960	0.951	0.968	0.005
Overall Performance	0.792	0.717	0.838	0.036

4.4 Number of Object Types

The results in Table 1 show that our algorithm is learning approximately six times as many object types as are labelled in the dataset. This is reasonable performance, considering that it would have created 4737 object types had each image been assigned its own label, but higher than anticipated. However, our qualitative examination of the classifications uncovered that the number of object types is not exclusively a result of algorithm error but primarily a result of learning finer-grained object types. For example, images annotated as $pottedplant$ are largely divided by our algorithm into two distinct classes: one for images of *fully-grown plants* and one for *seedling plants*. To a human, there are clear and obvious distinctions between these two subsets of images, providing support that the algorithm learned a meaningful subdivision. Numerous other similar examples were found where the algorithm learned meaningful finer-grained object types, a selection of which include: *full-sized cars* vs. *go-karts* (both annotated as *car*), *people in water* vs. *babies* vs. *athletes* (all annotated as *person*), and *locomotives* vs. *subway trains* vs. *empty train tracks* (all annotated as *train*). However, although a significant number of the additional object types learned by our algorithm appear to be meaningful object types, it also learned less meaningful single-image object types. Although some of those singleton object types are uninteresting or redundant, it learned several interesting singleton object types based on unusual images in the dataset: a sheep standing in a bus shelter, a train car with a picture of a dinosaur painted on it, an alpaca (incorrectly annotated in the *PASCAL-Part Dataset* as *sheep*), and a Ferris wheel. However, there were also situations where our algorithm erroneously subdivided object types, or performed divisions that a human would not deem as necessary (i.e., too fine-grained). This qualitative analysis provides partial support for **H4**, but a more detailed analysis will be necessary to definitively prove that our algorithm is identifying meaningful object sub-types.

5 Related Work

Integrations of Deep Learning and CBR have seen increased interest recently, with many researchers exploring how the two approaches can benefit each other. In the domain of Human Activity Recognition (HAR), CNNs have been used for feature extraction [6]. This work differs from our own in that it uses accelerometer data rather than image data, but similarly finds that reasonable results can be achieved with instance-based algorithms when features are automatically learned and extracted using CNNs. Instance-based retrieval in the HAR domain has also been used to find similar existing data that can be used to train a classifier for a new user [7]. Their system also uses CNN-extracted features and, like our work, is motivated to allow learning under limited data availability. However, their work is focused on classifier personalization rather that novel class identification (e.g., they do not detect new types of activities that have not been seen before). They have also examined how Siamese Neural Networks can be used to learn similarity functions [8], and such an approach could potentially be used in our algorithm to improve retrieval. Deep Learning has been combined with CBR to generate novel recipes that are both surprising and plausible [9]. However, this differs from our own work in that their system creates novel items rather than discovering previously unknown items.

Case-based reasoning has been used for a variety of image processing and computer vision tasks [10]. One application area that has seen particular interest is medical CBR (e.g., [11-13]), primarily due to the prevalence of medical imagery in patient files and the need to retrieve similar images to aid in diagnosis. However, unlike our work, the majority of CBR approaches rely on hand-crafted features rather than learned features (e.g., [14-16]). Additionally, while CBR systems are often used for image retrieval and classification, to the best of our knowledge none are able to detect novel object types (or, more generally, novel classes in non-image systems). Some systems may be able to perform outlier detection (e.g., when no similar cases are retrieved) but do not attempt to learn novel object types from these outliers. For example, rather than attempt to generate a novel object label, a CBR system may present an input image to a domain expert for manual labelling. Although having human annotations is valuable, it is not always practical when a system is operating autonomously for long periods of time.

The most similar work to our own involves classifying webpages based on multimedia data (e.g., images) rather than only the contained text [17]. Like our approach, they use CNNs to perform feature extraction from images and use those features during case retrieval. The primary difference between their work and our own is that they only classify images into predefined classes, so no novel object discovery is performed. They do perform outlier detection, but that is to identify mislabeled or irrelevant images contained in a webpage rather than to detect novel webpage themes; outliers influence the case structure but do not modify the set of class labels.

As we mentioned previously, existing approaches to Computer Vision tend to focus on object classification (e.g., CNNs [1]) or detecting regions containing objects (e.g., R-CNNs [2]). These approaches rely on a predefined set of object types, with fewer works examining novel object discovery. Existing approaches for unsupervised object class discovery are similar to our own work in that they learn from images containing a single object type per image [18-20]. However, as we mentioned previously, the images we use in this work often contain multiple objects in each image but with only one of the objects labelled by human annotators. The primary difference between these approaches and our work is that they perform offline object detection using the entire dataset. Our approach is both online and incremental; novel object types are detected at run-time based on the content of input images. To the best of our knowledge, no other approaches exist to allow online and incremental unsupervised object discovery. As we discussed previously, existing computer vision systems can only identify that an input image is unlikely to be of a known object type. They do not provide online labels for these unknown objects or learn from them (i.e., how to classify future images of that object type). However, our approach can perform such labelling and learning, and can learn after retaining only a single case.

Our algorithm learns in an unsupervised manner when no expert-annotated training cases are provided to it (e.g., as in our evaluation that started with an empty case base). As such, it has many similarities to clustering since it is grouping input images by assigning them generated class labels. Many traditional clustering algorithms, like k-means [21], divide data into a fixed number of partitions, whereas our approach dynamically creates new object types as necessary. Hierarchal clustering methods, like single-linkage [22], are able to dynamically increase the number of clusters created but do not cluster incrementally; the entire dataset must be provided as a batch. Incremental clustering algorithms have been developed, such as incremental k-means [23], that allow data points to be added sequentially rather than as a batch. However, even incremental clustering algorithms rely on comparing each data point to a set of cluster centroids. Our approach compares data points (i.e., input images) to any existing case in the case base. This is important given the two-stage retrieval process we use. Since retrieval is based on both an image's feature vector representation and its observable parts, there can be a high degree of variability amongst cases of the same object type. For example, since the similarity thresholds used by our algorithm may be relatively low (e.g., 0.45 in our experiments), cases of the same object type may not have highly similar feature vector representations (i.e., a medium feature vector similarity but high parts similarity). Similarly, cases of the same object type may have high feature vector similarity but only medium parts similarity. If only cases representing class centroids were retrieved, an input image could appear dissimilar to all of the centroids (i.e., treated as a novel object type) but would have been similar to one or more of the non-centroid cases. Additionally, unlike clustering algorithms, our algorithm can be used for both classification and novel class discovery. Without any labeled data, it performs classification based on its generated object type labels. However, if some cases are provided using labelled training data (i.e., some supervised learning was performed) the algorithm can either generate novel class labels or perform classification based on existing object type labels.

6 Conclusions

This paper described a method for detecting novel object types in images using a combination of case-based reasoning and Convolutional Neural Networks. Our approach leverages the automated feature learning and extraction provided by CNNs while taking advantage of CBR's ability to perform incremental learning with relatively few training instances. A set of nearest neighbors are initially retrieved based solely on similarity between extracted image features, with subsequent retrieval based on the similarity between observable object parts. Although our approach leverages observable object parts during case retrieval, it can be used even if such information is unavailable. Additionally, since CBR is an instance-based learner, it does not abstract the object parts contained in images, thereby allowing them to be directly used during similarity calculation. If a CNN was to include object part information it would likely learn an abstraction of what parts exist in a class. For example, it would learn what parts are generally observable in images of *dogs*, possibly losing valuable information necessary to detect uncommon images, like a dog with most of its observable parts obscured by a costume it is wearing.

Our evaluation was performed using realistic images from the publicly available *PASCAL-Part Dataset*. The initial results demonstrated the ability of a CBR system to classify images using CNN-extracted feature vectors, and the performance improvement provided by including object parts information during retrieval. We also provided evidence of our algorithm's ability to be used to detect novel object types. Even when the algorithm had an empty initial case base and no background knowledge about object types, it was able to detect novel object types and use them to classify subsequent images. One important finding of these experiments was that the algorithm appeared to learn finer-grained object types than those provided by the human dataset annotators, based on an initial qualitative analysis.

Several areas of future work remain. First, while we briefly discussed how our approach could be used in parallel with a full CNN (i.e., including fully-connected layers), we have not provided a full methodology to integrate them. In this paper, we focused on learning without an existing dataset, so it would not be possible to train a CNN in such a situation. However, if a subset of existing object types are known and have sufficient data, a full CNN could be used to classify those known types while our approach could handle novel object type detection. Second, our approach learns a flat object type hierarchy. Future work will examine how novel object types can be compared to existing types to determine relationships (e.g., a *fully-grown plant* is similar to a *seedling plant*) or to provide explanations (e.g., "*I think this is different than a fully-grown plant because it doesn't have any leaves*"). Third, we used ground truth parts information, but future work will detect both parts and object types. Finally, we plan to integrate our work with existing R-CNN architectures to allow learning with images containing multiple annotated objects.

Acknowledgements

Thanks to the Office of Naval Research for supporting this work.

References

1. Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems* (pp. 1106-1114). Lake Tahoe, USA.
2. Girshick, R.B., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 580-587). Columbus, USA: IEEE Computer Society.
3. Chen, X., Mottaghi, R., Liu, X., Fidler, S., Urtasun, R., and Yuille, A. (2014). Detect what you can: Detecting and representing objects using holistic models and body parts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1979-1986). Columbus, USA: IEEE Computer Society.
4. He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 770-778). Las Vegas, USA: IEEE Computer Society.
5. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., and Li, F.-F. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, **115**(3), 211-252.
6. Sani, S., Wiratunga, N., and Massie, S. (2017). Learning deep features for kNN-based Human Activity Recognition. In *Proceedings of the International Conference on Case-Based Reasoning Workshops* (pp. 95-103). Trondheim, Norway: CEUR Workshop Proceedings.
7. Sani, S., Wiratunga, N., Massie, S., and Cooper, K. (2017). kNN sampling for personalised Human Activity Recognition. In *Proceedings of the 25th International Conference on Case-Based Reasoning* (pp. 330-344). Trondheim, Norway: Springer.
8. Martin, K., Wiratunga, N., Sani, S., Massie, S., and Clos, J. (2017). A Convolutional Siamese Network for developing similarity knowledge in the SelfBACK dataset. In *Proceedings of the International Conference on Case-Based Reasoning Workshops* (pp. 85-94). Trondheim, Norway: CEUR Workshop Proceedings.
9. Grace, K., Maher, M.L., Wilson, D.C., and Najjar, N.A. (2016). Combining CBR and Deep Learning to generate surprising recipe designs. In *Proceedings of the 24th International Conference on Case-Based Reasoning* (pp. 154-169). Atlanta, USA. Springer.
10. Perner, P., Holt, A., and Richter, M. (2005). Image processing in case-based reasoning. *Knowledge Engineering Review*, **20**(3), 311-314.
11. Macura, R.T., and Macura, K.J. (1995). MacRad: Radiology image resource with a case-based retrieval system. In *Proceedings of the 1st International Conference on Case-Based Reasoning* (pp. 43-54). Sesimbra, Portugal: Springer.
12. Haddad, M., Adlassnig, K.-P., and Porenta, G. (1997). Feasibility analysis of a case-based reasoning system for automated detection of coronary heart disease from myocardial scintigrams. *Artificial Intelligence in Medicine*, **9**(1), 61-78.

13. Allampalli-Nagaraj, G., and Bichindaritz, I. (2009). Automatic semantic indexing of medical images using a web ontology language for case-based image retrieval. *Engineering Applications of Artificial Intelligence*, **22**(1), 18-25.
14. Perner, P., and Bühring, A. (2004). Case-based object recognition. In *Proceedings of the 7th European Conference on Case-Based Reasoning* (pp. 375-388). Madrid, Spain: Springer.
15. Micarelli, A., Neri, A., and Sansonetti, G. (2000). A case-based approach to image recognition. In *Proceedings of the 5th European Workshop on Case-Based Reasoning* (pp. 443-454). Trento, Italy: Springer.
16. López-Sánchez, D., Corchado, J.M., and González Arrieta, A. (2017). A CBR system for efficient face recognition under partial occlusion. In *Proceedings of the 25th International Conference on Case-Based Reasoning* (pp. 170-184). Trondheim, Norway: Springer.
17. López-Sánchez, D., Corchado, J.M., and González Arrieta, A. (2017). A CBR system for image-based webpage classification: Case representation with Convolutional Neural Networks. In *Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference* (pp. 483-488). Marco Island, USA: AAAI Press.
18. Tuytelaars, T., Lampert, C.H., Blaschko, M.B., and Buntine, W.L. (2010). Unsupervised object discovery: A comparison. *International Journal of Computer Vision*, **88**(2), 284-302.
19. Zhu, J.-Y., Wu, J., Xu, Y., Chang, E., and Tu, Z. (2015). Unsupervised object class discovery via saliency-guided multiple class learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **37**(4), 862-875.
20. Chen, X., Shrivastava, A., and Gupta, A. (2014). Enriching visual knowledge bases via object discovery and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2035-2042). Columbus, USA: IEEE Computer Society.
21. MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability* (pp. 281–297).
22. Hartigan, J.A. (1975). *Clustering Algorithms*. New York, USA: John Wiley & Sons.
23. Aaron, B., Tamir, D.E., Rishe, N.D., and Kandel, A. (2014). Dynamic incremental k-means clustering. In *Proceedings of the International Conference on Computational Science and Computational Intelligence* (pp. 308-313). Las Vegas, USA: IEEE Press.