Rapid online semi-supervised training of a scene parser using a convolutional auto-encoder and representation nudging

* Solvi Arnold (Shinshu University) and Kimitoshi Yamazaki (Shinshu University)

1. Introduction

In the present work, we aim to provide automatic scene parsing capabilities for use in disaster response robotics. Recent years have seen substantial improvements in image recognition. The main drive behind this progress comes from advances in neural network technology. Convolutional neural networks (CNNs), while first proposed in the 1980s and 1990s [1] [2], have over the past years become the technology of choice for many computer vision applications. Ever-increasing computation power has made it possible to train networks on millions of example images, with impressive results [3]. However, achievements on benchmark datasets do not easily translate to practical application, exactly because of the amount of data required to train CNNs at such scales.

In recent work, we have applied a CNN based approach to computer vision for disaster robotics, using relatively small CNNs and limited training data [4]. We found this can provide good performance on challenging data, but even at a few hundred data frames, preparation of the training data takes considerable time and effort.

Openly available datasets of sufficient generality suitable for this type of application are hard to come by. This is unsurprising: disaster environments are atypical and unpredictable, and data gathered in one disaster environment may not easily generalise to the next. On the other hand, the urgency of a disaster response setting, as well as the dangerous nature of the environments, makes it practically infeasible to gather substantial amounts of training data in advance of a disaster response mission. Hence if image recognition technology is to contribute to disaster response, it seems crucial to develop systems that can be trained efficiently 'on the spot', from limited data, and with minimal preparatory effort.

Our present work aims to meet these demands using a convolutional auto-encoder (CAE) [5] and semi-supervised learning, in order to combine the discriminatory potential of a CNN with an interactive, “on the fly” learning style that obviates the need for preparation of training data. The system can simultaneously learn from and perform scene-parsing on a live video feed in real-time on a GPU-equipped laptop. User-feedback is incorporated into the learning process by means of a novel algorithm ('representation nudging').

2. Semi-supervised learning

Semi-supervised learning integrates elements of supervised and unsupervised learning. The usual setting is a collection of data points, some of which have a user-provided label attached to them indicating the category membership (i.e. the correct classification) for that data point, while the majority of points are unlabelled. Working from the assumption that points belonging to the same category will lie relatively closely together, one attempts to find a clustering of the points that aligns well with the category boundaries. One can then use the clustering to generalise category membership from the labelled data points to the unlabelled data points. In our scenario, the data points are patches (superpixels, obtained by segmentation using the SLIC algorithm [6]) of frames of a video feed (either pre-recorded footage or a live stream from a robot), and the categories are the various objects and surface types we aim to distinguish.

When working with high-dimensional data (as is the case with image data), it is often smart to perform some form of dimension reduction before clustering. In our system, we this is done by means of a CAE.

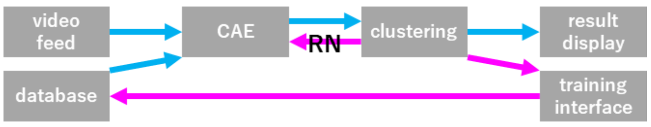

Figure 1. System overview. The system of blue arrows is responsible for classification. The system of pink arrows is responsible for learning (these processes are only performed when learning is enabled).

Many algorithms use the user-provided labels to guide the clustering process in one way or another. Data point labels can be translated into constraints on the clustering (in the simplest form, points with the same label should fall in the same cluster and points with different labels should fall in different clusters), after which we can search for a clustering satisfying those constraints. However, clustering with constraints is not a trivial problem, and can quickly become computationally costly as the number of constraints increases (enforcing that differently labelled points fall in different clusters is an NP complete problem [7]). For real-time performance on a video feed, we need results on the order of milliseconds. This rules out overly computationally expensive clustering logic. We use constraints in a different way. Instead of avoiding constraint violations in the clustering process, we detect constraint violations (which is computationally cheap) and then use these to train the CAE so as to reduce future constraint violations. In this way, we let the CAE do double duty: dimensionality reduction and constraint violation avoidance. As will be explained below, this approach has the effect of aligning the CAE’s discrimination abilities with the user’s perception of category boundaries.

An outline of the system’s operation is given in Figure 1. In the following sections we explain the various elements in more detail.

3. Convolutional auto-encoder

The conventional use of auto-encoders is dimensionality reduction. Dimensionality reduction may be done for the following reasons: 1) lower dimensionality makes for easier and computationally cheaper clustering, and 2) smart ways of compressing data bring out the regularities and types of variation latent in the data, meaning that the category distinctions may be more pronounced in the compressed representations than in the raw data. In our setup, we additionally use our auto-encoder as the locus of a user-guided learning process.

A convolutional auto-encoder is simply an auto-encoder containing convolutional layers [5] (our CAE is fully convolutional, meaning that it is composed exclusively of convolutional layers). Like regular auto-encoders, CAEs are trained to replicate the input activation pattern (here: image) on their output layer. Data compression occurs as a consequence of the network architecture. The central hidden layer (the "bottleneck") has low dimensionality (i.e. few neurons) compared to the input dimensionality. Hence to adequately replicate the input, the network must compress the input activation pattern into the dimensionality of the bottleneck, and then decompress the activation on the bottleneck back to the original activation pattern. Hence the activation pattern on the bottleneck layer is a compressed (dimension-reduced) representation of the input. This can be used as a low-dimensional representation (LDR) of the input.

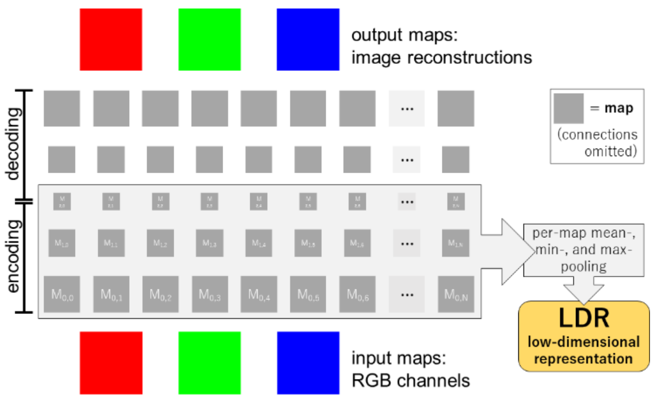
output maps:
image reconstructions
decoding
encoding
= map
(connections omitted)
$M_{1,0}$ $M_{1,1}$ $M_{1,2}$ $M_{1,3}$ $M_{1,4}$ $M_{1,5}$ $M_{1,6}$ … $M_{1,N}$
$M_{0,0}$ $M_{0,1}$ $M_{0,2}$ $M_{0,3}$ $M_{0,4}$ $M_{0,5}$ $M_{0,6}$ … $M_{0,N}$
per-map mean-, min-, and max-pooling
LDR
low-dimensional representation
input maps:
RGB channels

Figure 2. Obtaining LDRs from a fully convolutional auto-encoder (CAE).

In the present system, we source the LDR in a slightly different manner, which more effectively exploits the convolution concept. In our CAE even the bottleneck layer is a convolutional layer, composed of a number of 2D maps of activation values. The activation values in these maps indicate how strongly the kernel (filter) computing that map responds to the input, and thus give strong clues to the content of the input. Simply flattening the activation values from the bottleneck maps into a vector would provide a sensible representation, but one with an unfortunate sensitivity to translations: a slight shift of the coordinates at which the patch is sourced from the video frame will change which value ends up where in the maps (an issue convolutional network layers were originally invented to circumvent [1]). However, such shifts have little effect on metrics such as the per-map mean, minimum, and maximum of the activation values. Hence we compute these metrics (effectively doing mean, min and max pooling over whole maps) for each map, and the collection of these metrics becomes the LDR used for clustering. To obtain sensitivity to patterns at various scales and levels of abstraction, we compute these values not just from the maps in the bottleneck layer, but also from the maps in the preceding convolutional layers, down to the first hidden layer.[1] The process is illustrated in Figure 2.

When training the system (which can be done during live operation), the CAE performs its regular (unsupervised) training updates (aimed towards improving its ability to replicate the input) continuously, on the incoming stream of image data (i.e. patches of the current video frame) as well as on a small database of 'background knowledge', consisting of user-labelled patches (which is initially empty).

Even without training or further modification, a CAE with randomly initialised connection weights will produce similar LDRs for similar inputs, which provides rudimentary discrimination ability to start out with (clustering on such LDRs does group similar superpixels together). Next we discuss the clustering logic.

4. Hierarchical clustering

For LDR clustering we adopt the OPTICS algorithm [8]. Unlike most clustering algorithms, OPTICS computes not a flat clustering but a 'reachability plot' (a type of dendrogram) representing a hierarchical clustering for the current dataset. The plot consists of a linear ordering of the data points, along with a distance measure for every neighbouring pair of points that in this ordering. An example of what a reachability plot for a given frame may look like is given in Figure 3.

Various "flat" clusterings can be derived from a hierarchical clustering, depending on how we set the density threshold, ε, for what we consider a cluster. We can also mix density thresholds, which is a good idea in our case, as the various categories distinguished by users will often correspond to clusters of widely differing densities. Hence we use sets of ε values to derive clusterings. We omit the exact algorithm we use for deriving clusterings from the plot, but the caption of Figure 3 illustrates the general idea. The set of ε values play a central role in our algorithm. Higher ε values allow lower and lower density regions to be included

[1] We note that it is also possible to use a CAE with a fully connected bottleneck layer, and simply use the activation pattern on the bottleneck as LDR. However, we obtained better results with the present approach.

in clusters, leading to fewer and fewer points remaining unclustered (noise). However, higher values of ε also lead to more cases of differently labelled points falling in the same cluster (at very large ε values, all points are grouped into one big cluster). We call the situation of differently labelled points falling in the same cluster a 'label collision'. Points in a cluster with a label collision cannot be classified unambiguously. This argues for a low value of ε.

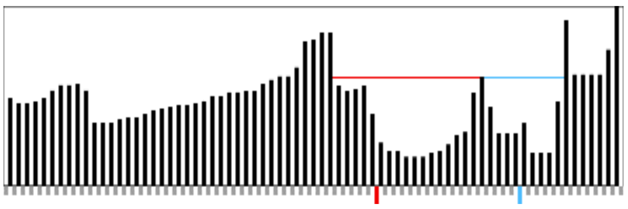
Figure 3. Example reachability plot as produced by OPTICS. Small squares below the plot represent data points (LDRs). Vertical bars represent the reachability distance for neighbouring pairs of points. If ε exceeds this value, the pair of points will fall in the same cluster (the actual distance between the points may be much larger than ε: points can be connected if there is a chain of points connecting them with no link longer than ε). See [8] **to learn how the ordering is derived. "Valleys" in the plot correspond to clusters (the deeper the valley, the higher the cluster density), while peaks correspond to borders between clusters (the higher the peak, the broader the border). We can think of ε as the "water level" on a surface spanned over this plot. It is trivial to find a set of water levels to produce a set of "lakes" (clusters) such that no two points with different labels fall in the same lake. This provides an unambiguous labelling for the clusters.[2] Consider for example the case where the data points highlighted in red and blue have (different) labels. The horizontal red and blue lines show how far we can raise the water level from these points while still keeping the labels in separate lakes. When the levels so obtained fall under a set threshold (leaving too many points above water (noise)), we raise the water up to the threshold. This causes "label collisions": multiple labels in the same lake (i.e. an ambiguous labelling). We use these collisions to perform supervised CAE updates (representation nudging).**

It is instructive to think of the ε values as being pushed in opposite directions by these two 'forces': minimisation of the number of noise points pushes ε up, while minimisation of the number of label collisions pushes ε down. The closer the cluster boundaries align with the category boundaries, the higher we can push ε without causing label collisions. Of course we can search for ever more intricate cluster boundaries avoid collisions, but such approaches do not generalise well to new data, quickly become computationally expensive, and do not address the fundamental problem that the points simply do not cluster along category distinctions. Intuitively put, things that looks similar to a user may look different to the CAE, and vice versa. By allowing label collisions to occur, and then resolving them in the CAE, we attempt to align the CAE's perception with that of the user. The next section explains how this is done.

Note that unlike more conventional classification CNNs, the number of categories our system can distinguish is not fixed in advance. New categories can be added on the fly by simply assigning novel labels.

5. Representation Nudging

To align the discrimination abilities of the CAE with the category perception of the user, we introduce a type of learning update, which we call 'representation nudging' (RN). The concept is as follows:

Ideally, we want for the following condition to hold: *SPs belonging to different categories should have dissimilar LDRs, so that they will fall into different clusters.* The mapping from SPs to LDRs is essentially a side-product of the CAE's unsupervised learning process. Nothing guarantees alignment between clusters and categories. Representation nudging improves this alignment. In doing so it exploits the fact that there is substantial flexibility in the SP→LDR mapping.

Consider as our domain a cluster with a label collision between labels x and y, containing N_x LDRs labelled x and N_y LDRs labelled y). Let P_x be the set of points labelled x: $p_{x,i}$, $i \in [0,\ldots,N_x]$, and P_y the set of points labelled y: $p_{y,i}$, $i \in [0,\ldots,N_y]$. Let $s_{x,i}$ and $s_{y,i}$ be the inputs (SPs) that produced $p_{x,i}$ and $p_{y,i}$.

We can compute the centre LDR for label x as simply the average over all P_x, and similarly over P_y for label y. Call these centre LDRs C_x and C_y. We can now compute a unit vector d indicating the direction from C_x to C_y:

$$\mathrm{d} = \frac{C_y - C_x}{\|C_y - C_x\|}$$

We then use this vector to compute target LDRs for every all $p_{x,i}$ and $p_{y,i}$:

$$p'_{x,i} = p_{x,i} - \beta * d \qquad p'_{y,i} = p_{y,i} + \beta * d$$

Where β is a constant "nudge intensity" parameter.

[2] There are of course various ways to derive a collision-free clustering from a reachability plot. The method we describe is motivated by the natural assumption that the reachability threshold ε may differ between clusters but should be uniform within each cluster, and is optimal under that assumption.

Now we can update the CAE with a supervised learning step that sets $s_{x,i}$ as input and $p'_{x,i}$ as target output, and similarly with $s_{y,i}$ and $p'_{y,i}$. We can simply use the same standard error back-propagation used in the unsupervised updates. Given the indirect character of our LDRs this may seem surprising, but recall that the LDRs are derived via pooling operations on maps. The way we use them here is unconventional, but pooling is a common operation in convolutional neural networks which is compatible with error back-propagation learning.

This update causes the CAE to shift its LDR for $s_{x,i}$ from $p_{x,i}$ towards $p'_{x,i}$ and its LDR for $s_{y,i}$ from $p_{y,i}$ towards $p'_{y,i}$. Of course, the effect of this update is not limited to the points explicitly shifted. It changes the way the CAE maps SPs into the LDR space overall. The labelled points merely act as handles by which we hold and warp the CAE's SP→LDR mapping. The idea is to imprint the category distinctions made by the user into the CAE. RN updates are simply interspersed with the usual unsupervised training updates.

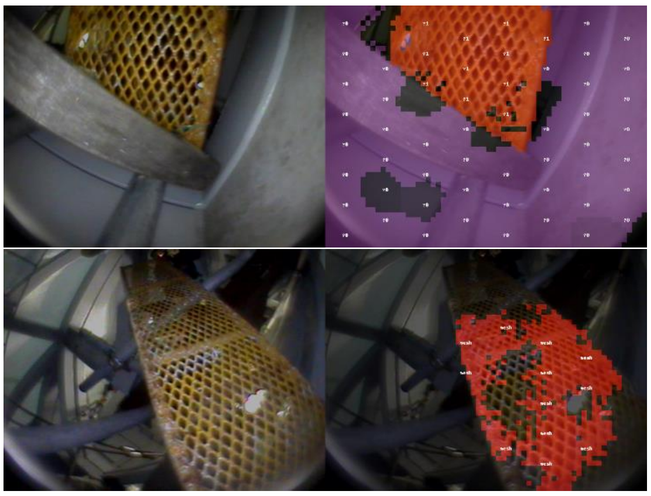

Figure 3. Example of system operation on footage from a snake-like robot developed for disaster response [9]**. Here the system correctly identifies the cluster of superpixels comprising a rusty metal mesh as one object (upper panel). The object is highlighted in red, and marked with a placeholder label. After the user assigns this cluster a definite label ("mesh"), subsequent occurrences of similar objects are automatically marked with the same label (bottom panel).**

6. Results

The CAE is implemented in TensorFlow [10] and ran on a GPU. On sufficiently powerful hardware and with modest settings for the various system parameters and database size, the system can simultaneously learn and classify in real-time (10+ fps). Quantitative performance assessments have yet to be performed. Semi-supervised scene parsing benchmarks generally assume that we learn from a static dataset, not a rolling camera feed, with different constraints and use cases in mind. This makes it difficult to quantify and compare system performance. However, our core algorithm can be adjusted to the benchmark conditions, and we plan to make a version of the system that is suitable for comparative benchmarking in the near future.

Acknowledgement

This work was partly funded by ImPACT Program of the Council for Science, Technology and Innovation (Cabinet Office, Government of Japan). We appreciate Prof. Tadokoro and Prof. Konyo for providing image data.

References

[1] K. Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," Biological Cybernetics, vol.36, no.4, pp. 193-202, 1980.

[2] Y. LeCun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel , D. Henderson, "Handwritten digit recognition with a back-propagation network," Advances in neural information processing systems, vol.2, pp. 396‐404, 1990.

[3] K. He, X. Zhang, S. Ren , J. Sun, "Deep Residual Learning for Image Recognition," Tech Report, 2015.

[4] S. Arnold , K. Yamazaki, "Scene-parsing for disaster environments by means of a convolutional neural network," The Robotics and Mechatronics Conference 2016, 2016.

[5] M. Jonathan, M. Ueli, C. Dan , S. Jürgen, "Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction," Artificial Neural Networks and Machine Learning ‐ ICANN 2011, Lecture Notes in Computer Science, 2011.

[6] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua , S. Süsstrunk, "SLIC Superpixels," EPFL Technical Report, no. 149300, 2010.

[7] I. Davidson , S. S. Ravi, "Clustering With Constraints: Feasibility Issues and the k-Means Algorithm," Proceedings of the 2005 SIAM International Conference on Data Mining, 2005.

[8] M. Ankerst, M. M. Breunig, H.-P. Kriegel , J. Sander, "OPTICS: Ordering Points To Identify the Clustering Structure," ACM SIGMOD international conference on Management of data, 1999.

[9] J. Fukuda, M. Konyo, T. Eijiro , S. Tadokoro, "Remote Vertical Exploration by Active Scope Camera into Collapsed Buildings," Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2014.

[10] J. Dean, R. Monga , et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015.