Real-time scene parsing by means of a convolutional neural network for mobile robots in disaster scenarios*

Solvi Arnold and Kimitoshi Yamazaki
Department of Mechanical Systems Engineering
Shinshu University
380-8553 Nagano Prefecture, Nagano city, Wakasato 4-17-1
{s_arnold, kyamazaki}@shinshu-u.ac.jp

***Abstract* - Disaster robotics poses particular challenges for computer vision, both in terms of image characteristics (due to motion blur, difficult light conditions, lack of up/down orientation, etc.), and in terms of learning data (limited availability, difficulty of annotation due to image quality, etc.). We developed a system for real-time scene-parsing, intended for use in a support system for operators of remote-controlled mobile robots employed in disaster areas. Our testbed is video footage gathered by a snake-like mobile robot exploring an (artificial) collapsed building environment. The core of the system is a relatively small-scale convolutional neural network. Our approach combines pixel-level learning with superpixel-level classification, in an effort to learn efficiently from a relatively small number of partially annotated frames. Our classification system is capable of real-time operation, and demonstrates that convolutional neural networks can be applied effectively even under the harsh conditions imposed by disaster robotics.**

Index Terms: disaster robotics, rescue robots, computer vision, convolutional neural networks, scene parsing.

I. INTRODUCTION

Recent years have seen increasing interest in the potential of robotic systems to assist in disaster response operations. The 2011 Tohoku earthquake and tsunami, and the ensuing nuclear disaster at the Fukushima power plant, in particular, have spurred the establishment of various initiatives for the development of disaster robotics, e.g. the ImPACT Tough Robotics Challenge (www.jst.go.jp/impact/en/program07.html) and the DARPA robotics challenge (www.theroboticschallenge.org).

A central goal in disaster robotics is to develop robots that can perform search, rescue, and investigative missions in areas that are off-limits to humans due to excessive danger (e.g. high radiation levels, danger of structural collapse, flooding, volcanic eruption, etc.). Typically, such systems will be operated by means of remote control. Operators receive sensor data from the robot's sensors (visuals, audio, etc.), and on basis thereof control the robot's movements. It is crucial that the operators can quickly form a clear understanding of the environment from the sensor data they receive, but factors such as restrictive camera-angles, lack or poverty of other sensory modalities, and the unfamiliar nature of the environment can make it very difficult to orient oneself in the world one sees through the robot's eyes. The number and fidelity of cameras and other sensors trade-off against energy consumption, robot size, and manoeuvrability. Factors such as potentially limited bandwidth and environmental noise may further complicate the acquisition of sensor data. However, failures in environment recognition can lead to inappropriate action selection and mission failure. Especially in the context of disaster robotics, the robot's location will generally be dangerous or inaccessible, making recovery difficult or impossible. Hence it is crucial to make the most out of the data at hand. Machine learning could play a vital role here, as a way of "virtually" increasing the number of eyes in the room, in the form of a computer vision system trained on experts' knowledge and judgments. In particular, it provides a means for capturing, storing, and sharing expertise obtained from individuals who are familiar with the mission environment (e.g. employees from a disaster-struck facility).

In this paper, we develop a computer vision system to assist operators of remote controlled robotic systems in making sense of the environment by labelling, in real-time, objects and surfaces in an incoming video stream. In particular, our system targets a snake-like robot designed for exploration of collapsed structures [1]. This platform highlights many of the characteristic difficulties one encounters when trying to apply computer vision techniques in disaster robotics. For example, while computer vision research typically employs datasets of sharp images from well-lit environments, computer vision for disaster robotics must be able to cope with unforgiving light conditions, motion blur, dirty lenses, etc. Even datasets composed of random images from the internet (such as the commonly used ImageNet dataset [2]) consist almost exclusively of images taken with some degree of attention to lighting, sharpness, and angle, simply due to being taken and selected for upload by humans. Fig. 1 illustrates the contrast between typical datasets commonly used in computer vision research on the one hand, and the video footage we target on the other.

Furthermore, disaster robotics poses particular challenges for any system that relies on learning data. Recent years have seen a trend towards bigger and bigger datasets in machine learning research, where sheer scale has become a point of appeal [3]. However, producing suitable learning data is a time-consuming effort. A disaster robotics system should be employable on short notice in unexpected environments, stressing the importance of learning efficiently from limited data.

* This work was partly funded by ImPACT Program of the Council for Science, Technology and Innovation (Cabinet Office, Government of Japan).

II. Related Work

There have been a variety of approaches to the task of image recognition. Until around 2011, the dominant approach was to analyse images for the presence of pre-determined features, to as to obtain a representation of the image in "feature space", followed by classification performed in feature space using various types of classifiers (e.g. SVMs). In systems following this approach, the SIFT [4] and HOG [5] features are particularly effective and widely used. However, recent years have seen a large shift in this field, with convolutional neural networks (CNNs) now taking a central position, achieving state-of-the-art results on many problems. The CNNs approach to image recognition represents a different strategy, integrating the search for suitable features with the classification task into a single learning process. CNNs have been around in various forms since the 1980s, but only started to surpass systems based on handcrafted features early this decade [6] [7] [8], in large part due to the computational cost of training them. Nowadays, winning entries in image classification competitions such as ILSVRC [3] are invariably CNN-based systems (see [9] [10] [11] for recent winners). For our specific use case (disaster robotics), the choice to pursue a CNN-based system is further motivated by the reliance of handcrafted features on image quality. Our video material shows substantial motion blur, which makes it difficult to reliably apply commonly used feature detectors. Our hope in developing this system is that the end-to-end learning capacities afforded by CNNs can overcome this hurdle.

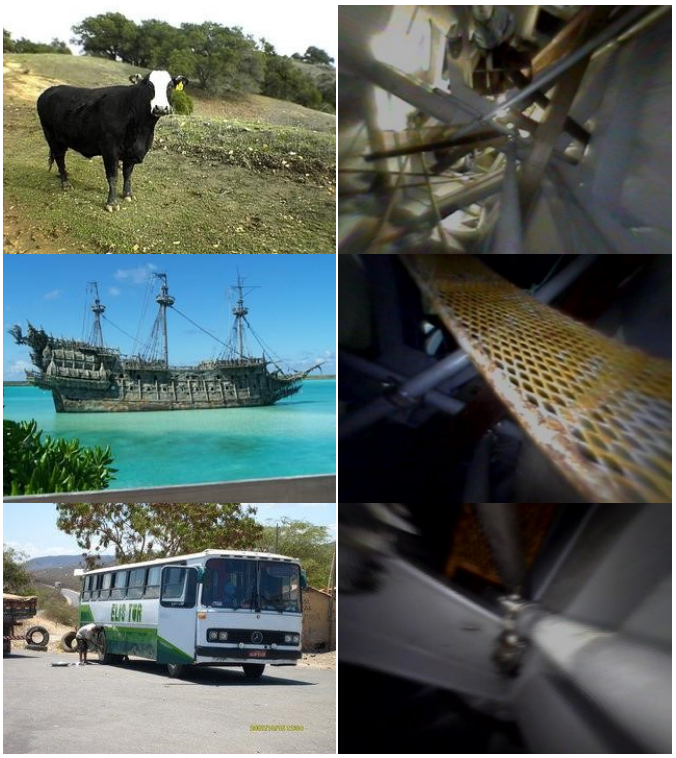

Figure 1. Contrasting a commonly used datasets with disaster robotics footage. Left column: example images from the Stanford background dataset [2]. Right column: example frames from our dataset (post lens-correction). Note the substantial blur, limited visibility, and arbitrary framing in our data.

Image classification, in its standard form, tasks systems with identifying the object depicted in an image. When solving this type of task with a CNN, the network output is a vector giving for each object class under consideration the estimated likelihood of the image belonging to that class. A limitation of this approach is that it assigns a class to the whole of an image. When an image contains items from multiple classes, such systems cannot tell us which classes occur where in the image (although it may be set up so as to give high probabilities to multiple classes). For actual scene-parsing, classification must be performed on a finer level. This has led to the pursuit of pixel-wise classification [12] [13] [14]. Here nets are trained to guess the class-membership of every individual pixel in an image. Pixel-wise classification handles scenes with multiple objects naturally, and captures spatial structure. An obvious issue is computational cost: typically the output has even higher dimensionality than the input (namely image resolution multiplied by the number of classes). Also, as results for individual pixels tend to be jittery, post-processing to integrate pixel classifications over larger areas is necessary for smooth results, adding further complexity and computational cost to the system. Use on raw video streams from badly lit environments imposes additional challenges, such as the presence of blurred and dark areas whose pixels cannot meaningfully be assigned a definite class. With these factors in mind, we developed a CNN system that operates at an intermediate scale: the net trains on annotated pixel data, but performs classification on superpixels. Below we explain our approach, starting with our dataset.

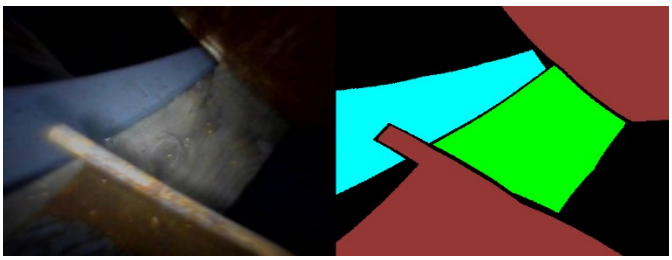

Figure 2. Example video frame (left) and its annotation (right).

III. Dataset

Here we describe our dataset. We obtained 67 minutes of raw footage from a snake-like robot moving around in an environment that models a collapsed building. From this footage we extracted a set of 259 frames as our dataset for learning and accuracy evaluation.

A. Pre-processing

The dataset is normalized by subtracting the mean and dividing by the standard deviation over all pixels. More advanced normalisations are available, but whichever normalization is employed on the learning data will also have to be applied to incoming video data during real-time classification. Hence it is important to use a computationally cheap normalisation, which rules out advanced normalisations such as PCA whitening or ZCA whitening.

The camera on the snake-robot has limited fidelity and introduces substantial fish-eye deformation. We remove this deformation (both in preparation of the dataset and in real-time

during operation) using the 'undistort' functionality of the OpenCV2 library. The reason to do this is that convolutional network architectures are based on the concept of location-independence, which is undercut by strong lens deformation.

As is common when training CNNs [6] [7] [8], we increased our dataset size by means of simple data augmentation tricks. We add rotated versions of the base dataset, at 15, 30, 45, 60 and 75 degrees (as these rotations are computationally expensive they are performed in a pre-processing step). During learning, input (multiscale patches, see below) was additionally rotated by 0, 90, 180, or 270 degrees, and optionally flipped horizontally and/or vertically (there is generally no clear up or down in our image data). This latter set of operations is computationally cheap, so this was done on-the-fly.

TABLE I. THE SET OF CLASSES.

1	Pipe (inside)	7	Nylon curtain
2	Pipe (outside)	8	Ground
3	Mesh	9	Concrete
4	Wooden beam	10	Fitting
5	Iron beam	11	Blackout (auto-annotated)
6	Rusty steel		

B. *Annotation*

Frames were annotated by placing coloured polygons over individual objects (with different colours mapping to different classes), using Microsoft PowerPoint's polygon drawing functionality. An example annotation is given in Fig. 2 and the set of classes is given in Table I. Note that annotation is partial. Substantial parts of most frames cannot be assigned to any of our categories (as can be seen in Fig. 2). Some frames contain objects that are not present in the set of classes (objects that occur only incidentally do not provide enough data to learn a class from).

The facility where our image data was collected replicates the light conditions of the intended use scenarios (exploration in collapsed buildings) of the snake robot. In our data, the only light source is a lamp mounted on the robot itself. Consequently, objects close to the robot are well-lit, whereas objects further away are dark or even completely obscured by darkness. This means that substantial parts of an image can contain virtually no information. We found that it is useful to add a 'blackout' category for these areas, which is treated the same as the other categories in the learning and classification processes. The reasoning behind this is that it prevents the network from acquiring spurious classifications for these areas. Consider the following: without the blackout category, the dataset would provide no clue as to how the net should classify pitch-black areas. Such areas cannot meaningfully be classified as any of the provided categories, but without reason not to, they will end up classified as whatever category ends up 'nearest' to pitch-blackness in the network's representation scheme. We have seen this result, for example, in networks that classify all pitch-dark areas as belonging to the 'fitting' category. This behaviour makes no sense, but fails to be suppressed by the learning process, as no learning on pitch-dark areas occurs to assign them a sensible classification. Including an explicit blackout category solves this problem[2]. As areas of insufficient contrast can be detected trivially, we automated their annotation as follows: for every pixel in the dataset, if no pixel in its 31x31 neighbourhood exceeds a set threshold (16/255) on any of the colour channels, the pixel is auto-annotated as belonging to the blackout category. This category is ignored in our performance measurements (recognising this category is very easy compared to the other categories, so including it in performance measures would artificially inflate performance).

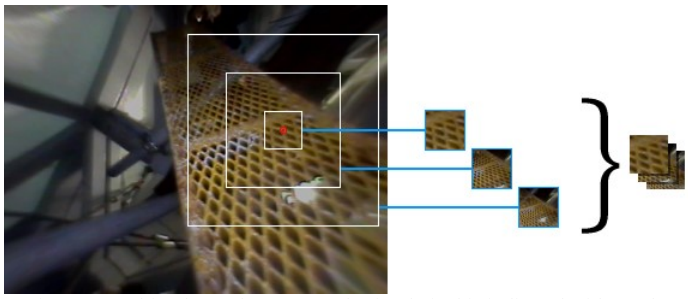

Figure 3. Multiscale patch concept. The focal pixel is indicated with a red marker. White outlines indicate the source regions for each scale. Source regions are copied and scaled down to the same size, then combined into a 9-channel (3 scales times 3 colour channels) compound image.

IV. SYSTEM

We do not feed whole images into the network's input, but instead we dynamically source 'multi-scale patches' (abbreviated below as MSP) from the dataset.

C. *4.1 Multi-scale patches*

A MSP is a (W x H x C x S) array of image data, where:

W = width of patch (48 here)
H = height of patch (48 here)
C = number of colour channels in image (3 here (RGB))
S = number of scales per MSP (3 here)

Fig. 3 illustrates the concept. In practice, many vision-oriented neural network libraries assume a three-dimensional input space (W x H x C), and the library we used (Pylearn2 [15]) is no exception. Fortunately, colour channels and scales can safely share a dimension, so MSPs were implemented as W x H x (C * S) arrays. To determine the scales, we have an array $[s_0 \cdots s_{S-1}]$ of scale parameters, here set to $[1, 3, 5]$.

MSPs are composed as follows. We start with the coordinates for a single pixel in an image (the way this pixel is selected differs between learning and classification, as explained below), which we refer to as the 'focal pixel'. For every scale s_i

[2] A simpler solution is to detect when an area has too little contrast, and then simply skip that area during the classification procedure. However, this approach has certain drawbacks. Firstly, note that a net that classifies darkness as 'fitting' does not just fail to make sense of darkness; clearly there is also something odd about its representation of the 'fitting' category. Inclusion of blackout category forces nets to acquire a better delineation of the other categories. Secondly, suppression of blackout areas would have to be performed continually throughout the real-time classification process, whereas auto-annotation of these areas and learning thereon is performed in advance of actual use of the system, saving us a modest amount of real-time processing.

in the scale array, we copy from the image a (($s_i \cdot W$) x ($s_i \cdot H$) x C) region centred on the focal pixel. This image patch is then scaled down in its width and height dimensions by a factor s_i, resulting in a patch of size (W x H x C). After doing this for all scales, we combine the resulting patches into a single array, which is used as input data for the net. The correct classification for a given MSP is the category of its focal pixel. Hence the output of the net is a 1D vector of length equal to the number of categories (just as it usually is when classifying in one-class-per-image fashion).

The idea of presenting image data at multiple scales is a common (if not quite satisfactory) way of dealing with the problem of size variation, and has been applied in various forms in various CNN systems. The use of convolutional layers ensures that the net will recognize a feature regardless of the location at which it occurs in the image, but provides no such invariance for scale variation. In other words, the filters a CNN learns are scale-specific. Providing the same image data at multiple scales helps ensure that filters are learned for various scales. Furthermore, the use of multiple scales helps ensure that context for the focal pixel is included at sensible resolutions. The smallest scale provides nearby context at high resolution, while the largest scale provides broad context at low resolution.

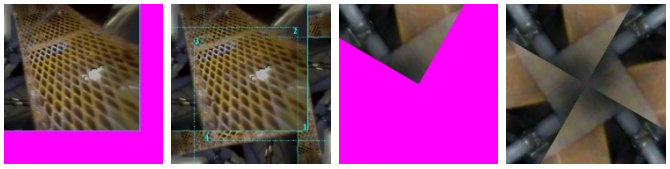

Figure 4. Padding missing image parts. Fuchsia indicates missing pixel values. a) Example case where a patch extends outside the image. b) Padding result. Missing parts are filled in by layering 90°, 180°, and 270° rotations of the image content, in spiral fashion. Solid and dotted lines indicate visible and hidden borders of individual layers). c) Example of a corner case. In the most extreme case (when the focal pixel is a corner pixel of the image), only ¼ of the patch is occupied by the original image. d) Padding with three rotations exactly fills out all missing values.

D. Filling of blank edges

As focal pixels (both during learning and classification) can be picked from anywhere in an image, if often happens that the area covered by one or more sub-images of a multi-scale patch extends outside of the input image, leading to substantial numbers of missing inputs. Handling missing values in neural network input is known to be a hard problem, with a long history of active research (see e.g. [16] [17]). However, a review of the literature did not turn up a principled method suited for our particular use case (note that we are not dealing with incidental missing pixels or the like, but with contiguous areas of missing data extending all the way to the edge of the input array, making meaningful imputation a hopeless pursuit). Padding with zero values or mean input is known to be potentially harmful [16] as it introduces spurious correlations (note that in the case of images, padding with zeros is equal to adding black areas). One fast and straightforward approach is to fill out the missing parts with normally distributed noise. This avoids introduction of spurious information, but obviously introduces substantial amounts of noise into the system (note that up to ¾ of a patch' content can be undefined in our system). Stretching image content is an option, but deforms image content. A recent study [18] found that simply repeating the border pixels of the image to fit the net's input layer led to better performance than stretching. With the above in mind, we expect that an ideal padding strategy 1) avoids deformation, and 2) sources the padding from the available image content. Our approach is as follows: We treat missing values as transparency, and simply stack four copies of the available image content, at rotations of 0°, 90°, 180° and 270° angles (see Fig. 4). This always produces a full covering of the patch, even in the worst case scenario where the focal pixel is a corner pixel of the image, and ¾ of the input is blank (see Fig. 4d). The motivation for this approach is as follows: Rotations retain the same centre pixel, and hence the same class membership (as the net is tasked with learning/guessing the class of the focal pixel, specifically). Hence the material we add (in the locations we add it) can meaningfully contribute to correct classification (instead of just failing to detract). This approach does introduce some spurious edges in the input, but did produce markedly better performance than simple noise-padding (as shown below). Detailed comparison with other methods is left for another occasion.

E. Advantages of a patch-based approach

One important advantage of the patch-based approach is that it allows for easy balancing of the category distribution presented to the net during learning. When creating a benchmark dataset it is easy to arrange similar occurrence frequencies for all categories, but in practical use scenarios one is likely to be confronted with highly skewed occurrence frequencies across categories. When presenting a net with whole images (as done in e.g. [12] [13] [14]), the more common categories will have a stronger effect on the learning process. This may be desirable when categories' importance is proportional to their occurrence frequency (as it is when aiming to improve benchmark scores), but in practical applications importance and frequency are not necessarily linked (in e.g. the case of a robot exploring a collapsed building, we can expect to see far more rubble than survivors, but correctly detecting survivors is of greater importance than correctly detecting rubble). Given than each patch functions as an example of one category, it is trivially easy to balance (or even counter-bias) the category distribution seen during training.

Also, the fact that our annotations are partial means that learning on whole images would require some way to deal with very large numbers of missing target values during training, which is known to be a difficult problem.

Note that, like fully convolutional nets, the patch-wise approach allows input images of any resolution to be processed by one and the same net. No rescaling of input images is necessary when changing to a different input image resolution. This is a plus when working with different robots with different cameras in the same environment.

F. Convolutional Neural Network

After preliminary experimentation with various architectures, we settled on the architectures shown in TABLE for the

main set of experiments. All layers but the output used the hyperbolic tangent activation function. The output layer used the identity function. Weight updates are performed with Pylearn2's Stochastic Gradient Descent. The learning rate is fixed to 10^{-6} and weight decay was set to 5×10^{-5}.

We found relatively shallow networks to perform well. We suspect that this is due to the nature of our dataset. The sort of complex compound shapes (faces, cats, cars etc.) that deeper networks excel at are mostly absent. Few of the categories in our dataset have any characteristic defining compound shapes, and moreover the haphazard camera angles provided by the snake robot ensure that even the same object will visually present as very different shapes. Hence the net likely learns to classify mostly by means of surface characteristics and context, with little reliance on higher order shape detection.

TABLE II. NETWORK ARCHITECTURES

architecture	7x7 convolution	2x2 pooling	Cross-channel normalisation [8]	5x5 convolution	2x2 pooling	Cross-channel normalization	5x5 convolution	2x2 pooling	Cross-channel normalization	Fully connected layer
A	●	●	●							●
B	●	●	●	●	●	●				●
C	●	●	●	●	●	●	●	●	●	●

G. Learning and classification

During learning, the net is trained on batches of MSPs. The MSPs used for training are generated dynamically during learning. Focal pixels are picked randomly from the set of all annotated pixels (hence the number of learning examples is equal to the number of annotated pixels in the augmented dataset). The pixel's assigned category is set as the target output for the MSP. By weighing pixels' probability of being picked by the scarcity of their category, we equalise category occurrence, as noted above.

Classification is performed as follows. We first split the input image into superpixels using the (OpenCV2 implementation of) the SLIC algorithm [19]. We found that (at least on our data) superpixel segmentation can safely be performed at reduced resolution without notably deteriorating the overall quality of the segmentation. Whereas segmentation takes well over 300ms at the original image resolution of 640x480, at 1/8 of this resolution it takes about 3ms per frame while still providing visually satisfactory results.

The SLIC algorithm takes the desired number of superpixels to split the image into as a parameter. We found that for our input images, a setting of 35 provides well-sized superpixels that do not cross class borders much. For every superpixel, we compute the centre pixel, which becomes the focal pixel for one MSP. After extracting the relevant MSPs, we run (35 instances of) the trained network, in parallel (on GPU).[3]

For measuring performance on the test set, we find the maximum activation value from the network's output layer and assign the corresponding category to all pixels in the source superpixel. The resulting pixel-wise classification is then compared against the annotation to determine their agreement. The rate of agreement gives us an objective performance measure for evaluating the CNN. However, when processing a video stream, we can obtain better usability by means of a different post-processing procedure, which we explain below.

H. Probability buffering

When processing a video stream, local misclassifications lasting just one or a few frames are common. These can be suppressed by means of temporal smoothing. The idea here is that the class of a pixel location in the directly preceding frames provides a good predictor of that location's class in the current frame. The effectivity of this approach depends on the amount of the difference between subsequent frames. In our use case, the pace of movement is fairly slow, so per-frame change is limited and temporal smoothing is very effective. The smoothing process is straightforward. We make a buffer array B of the same resolution as used for segmentation. Each position in the array stores a vector of class probabilities for the corresponding pixel. For every processed frame F, all vectors in the buffer are updated to a weighted sum of their previous value and the values provided by the new frame as follows:

$$B_{x,y,c} \leftarrow \beta \cdot B_{x,y,c} + (1 - \beta) \cdot F \quad (1)$$

Parameter β here controls the strength of the smoothing effect (or differently put, the persistence rate of past classification results). The class for a pixel is now determined simply by finding the class with the highest probability in the vector for that pixel in the probability buffer. Confidence for this class is compared against a threshold value. If it exceeds the threshold value, the pixel is displayed as belonging to that class (by means of colouration added in the transparent overlay). Otherwise, the result is suppressed and the pixel remains unclassified. In general, it is helpful to users to suppress low-confidence results, because they can more easily extrapolate from partial correct identifications when e.g. half of an object is left unclassified, than when that half is covered in low-confidence misclassifications.

Note that the network's confidence in its classifications, along with probabilities for non-dominant classes will affect the ease at which pixels change class. If a pixel is steadily classified as class X for a while, and then classed as Y with X as a close second (e.g. P(X) = 0.45 and P(Y) = 0.48) for a few frames, then this change will be suppressed as continued high frame probability will sustain a high buffer probability for class X. However if P(X) suddenly drops as P(Y) rises (as is expected

[3] Computational overhead is incurred here due to overlap between the content of the MSPs generated for neighbouring superpixels, especially at the largest scale. A potential avenue for further optimization is to split the network into two subnets: one comprising the convolutional layers and one consisting of just the fully connected layer. We could then apply the convolutional subnet to the whole image, and subsequently apply the fully connected subnet the relevant subsets of the output from the convolution subnet. Such splitting of trained nets is not facilitated by the Pylearn2 library, but could be done with relative ease in more recent frameworks.

to happen when an actual class change occurs), class Y will quickly come to dominate in the probability buffer. Hence the use of a probability buffer also puts the probability scores for non-dominant classes to good use[4]. We allow the user to change the smoothing parameter from the GUI during live operation. The ideal value for this parameter will depend on factors like the movement characteristics of the robot. We found that values of around 0.9 work well for the (relatively slow-moving) robot targeted here.

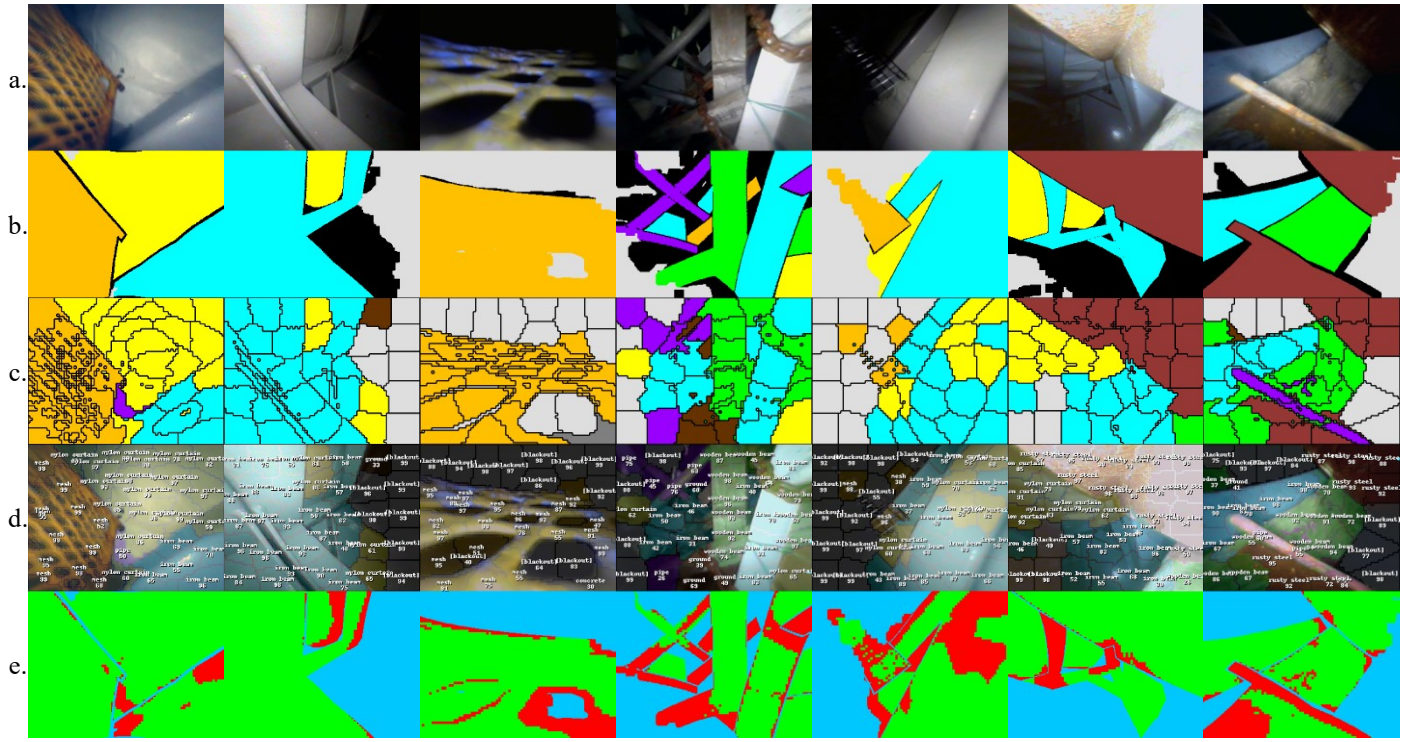

Figure 5. Result examples for various frames (one frame per column). Rows show the following: (a) Original frame. (b) annotation (ground truth). (c) Classification obtained from CNN. (d) Classification overlaid on input frame. Numbers indicate confidence. (e) Visualisation of the agreement between annotation and classification by CNN. Green and red indicate agreement and disagreement, respectively and blue indicates absence of (manual) annotation. Light grey areas in annotation and classification result indicate blackout areas. These are automatically annotated but disregarded during evaluation (they are almost trivial to classify and hence would artificially inflate classification rates).

By means of multithreading we parallelise the pre-processing (segmentation and finding of focal pixels), CNN processing, and post-processing (updating the probability buffer and drawing an image overlay on basis of the CNN's output) steps. While our implementation leaves room for improvement, we obtain frame rates of approximately 13fps, fast enough to convincingly track live input. Our system specifications are shown in Table III.

It may seem odd that we essentially run classification on superpixels, but do not provide the net any distinction between the superpixel and its context in the input. However, keep in mind that the network is trained to determine the class membership of the *single pixel* in the centre of its input field. The surrounding pixels function are just context, whether they belong to the superpixel or not. We have experimented with a variant of this system where we remove (replace with noise) the pixels that fall outside the superpixel in the smallest scale slice of the MSP. However, when doing so one also has to train the net on such "cut-out" superpixels. This is problematic, as the dataset of course provides far fewer superpixels than pixels (note that in our system, every annotated pixel acts as a data point for training). As such, representing superpixels explicitly in the network input has the effect of reducing context information and training opportunities, which led to reduced performance compared to our main system.

TABLE III. SYSTEM CONFIGURATION

OS	Ubuntu 14.04 LTS, 64 bit
CPU	Intel Core i7-5930K, 12 cores @ 3.50GHz
GPU	NVIDIA GeForce GTX TITAN X
RAM	32GB

V. RESULTS

TABLE . shows accuracy scores against the test set for the CNN architectures shown in II. We see that architecture C (with 3 convolutional layers) in combination with rotation padding

[4] This use of non-dominant classification results led us to choose the identity function for output layer activations. The more commonly used softmax function would destroy part of the information used here.

provides the highest accuracy, at 83.64% correct classification. While far from perfect, this level of performance may suffice for a supportive vision system. Fig. 5 shows representative sampling of recognition results on the test set.

One factor limiting performance is that the annotations used for training the net are far from perfect. The system is designed to help alleviate the difficulties in interpreting the visual feed from the snake robot, but this same difficulty is faced by the annotators (albeit not under the strain of real-time operation). Being unfamiliar with the environment ourselves, we struggled with the annotation effort. In practical use, performance of the current system could likely be further improved by involving individuals who are familiar with the mission environment in the annotation process.

TABLE IV. ACCURACY SCORES ON THE TEST SET

		Architecture		
		A	B	C
Padding	Noise	n/a	79.68	76.17
	Rotations	78.46	81.23	83.64

VI. DISCUSSION & CONCLUSIONS

We have described a Convolutional Neural Network system for scene parsing in a visually challenging environment, taking camera footage from a snake robot exploring a (artificial) collapsed building environment as our testbed. Our results show that, given appropriate adjustments to the training and recognition procedures, decent recognition rates can be achieved even in such difficult settings and with limited learning data. When multi-object recognition and preservation of spatial layout are required, but pixel-level classification is precluded by limitations of the data (large numbers of ambiguous or classless pixels, limited learning data), performing classification at the level of superpixels is a viable option. The use of multiscale input helps ensure that both detail and broad context are available to the learning process without inflating network size too much. A novel scheme for padding blank areas further boosts performance. During live operation, classification results are stabilized and refined by means of probability buffering.

A limitation of our approach is the reliance on annotation. The necessity of labelled learning data is commonplace in computer vision research, but in our use case this is more problematic than usual. Disaster environments are the products of disasters. This makes it difficult to obtain the right learning data for the task in advance. Even if we have data from the intact location beforehand, it may be of limited use. Data from similar environments (either constructed as in the case of our data, or from other incidents) may or may not be available. While we have intentionally held to a small amount of the learning data, the annotation effort was still substantial and time-consuming. For less urgent investigative missions this is not a prohibitive obstacle, but for time-sensitive initial disaster response (e.g. survivor search), the time it takes to get the system up and running in a given environment becomes problematic. Hence we are currently integrating ideas from work in active learning and semi-supervised learning, in an effort to circumvent the need for frame annotations.

ACKNOWLEDGEMENTS

We appreciate Prof. Tadokoro and Prof. Konyo for providing image data. We also thank Yosuke Koishihara for assistance in the annotation efforts.

REFERENCES

[1] Junichi Fukuda, Masashi Konyo, Takeuchi Eijiro, and Satoshi Tadokoro, "Remote Vertical Exploration by Active Scope Camera into Collapsed Buildings," in *Proceedings of the 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 1882-1888.

[2] S Gould, R Fulton, and D Koller, "Decomposing a Scene into Geometric and Semantically Consistent Regions," in *Proceedings of International Conference on Computer Vision (ICCV)*, 2009.

[3] Olga Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," *arXiv:1409.0575*, 2014.

[4] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, Nov. 2004.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings of the 2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 886-893.

[6] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "High-performance neural networks for visual object classification.," *IDSIA Technical Report*, 2011. arXiv:1102.0183

[7] D. C. Ciresan, U. Meier, and Jürgen Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3642-3649. arXiv:1202.2745

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, 2012.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep Residual Learning for Image Recognition," *Tech Report*, 2015. arXiv:1512.03385

[10] Xingyu Zeng et al., "Crafting GBD-Net for Object Detection," 2016. arXiv preprint arXiv:1610.02579

[11] Xingyu Zeng, Wanli Ouyang, Bin Yang, Junjie Yan, and Xiaogang Wang, "Gated Bi-directional CNN for Object Detection," in *European Conference on Computer Vision(ECCV)*, 2016.

[12] Camille Couprie, Clement Farabet, Laurent Najman, and Yann LeCun, "Indoor Semantic Segmentation using depth information," in *International Conference on Learning Representations*, 2013. arXiv:1301.3572v2

[13] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun, "Learning Hierarchical Features for Scene," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 8, pp. 1915-1929, 2013.

[14] Nico Höft, Hannes Schulz, and Sven Behnke, "Fast Semantic Segmentation of RGB-D Scenes with GPU-Accelerated Deep Neural Networks," *Proceedings of 37th German Conference on Artificial Intelligence (KI)*, pp. 80-85, 2014.

[15] Ian Goodfellow et al., "Pylearn2: a machine learning research library," *arXiv:1308.4214*, 2013.

[16] Volker Tresp, Subutai Ahmad, and Ralph Neuneier, "Training Neural Networks with Deficient Data," in *Advances in Neural Information Processing Systems 6*., 1994, pp. 128-135.

[17] Vadlamani Ravi and Mannepalli Krishna, "A new online data imputation method based on general regression auto associative neural network," *Neurocomputing*, vol. 138, no. 22, pp. 106-113, Aug. 2014.

[18] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and Wolfram Burg, "Multimodal Deep Learning for Robust RGB-D Object Recognition," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 681-687.

[19] Radhakrishna Achanta et al., "ALIC Superpixels," *EPFL Technical Report, no. 149300*, 2010.