Dear Author,

Here are the proofs of your article.

- You can submit your corrections **online**, via **e-mail** or by **fax**.
- For **online** submission please insert your corrections in the online correction form. Always indicate the line number to which the correction refers.
- You can also insert your corrections in the proof PDF and **email** the annotated PDF.
- For fax submission, please ensure that your corrections are clearly legible. Use a fine black pen and write the correction in the margin, not too close to the edge of the page.
- Remember to note the **journal title**, **article number**, and **your name** when sending your response via e-mail or fax.
- **Check** the metadata sheet to make sure that the header information, especially author names and the corresponding affiliations are correctly shown.
- **Check** the questions that may have arisen during copy editing and insert your answers/ corrections.
- **Check** that the text is complete and that all figures, tables and their legends are included. Also check the accuracy of special characters, equations, and electronic supplementary material if applicable. If necessary refer to the *Edited manuscript*.
- The publication of inaccurate data such as dosages and units can have serious consequences. Please take particular care that all such details are correct.
- Please **do not** make changes that involve only matters of style. We have generally introduced forms that follow the journal's style.
  Substantial changes in content, e.g., new results, corrected values, title and authorship are not allowed without the approval of the responsible editor. In such a case, please contact the Editorial Office and return his/her consent together with the proof.
- If we do not receive your corrections **within 48 hours**, we will send you a reminder.
- Your article will be published **Online First** approximately one week after receipt of your corrected proofs. This is the **official first publication** citable with the DOI. **Further changes are, therefore, not possible.**
- The **printed version** will follow in a forthcoming issue.

**Please note**

After online publication, subscribers (personal/institutional) to this journal will have access to the complete article via the DOI using the URL: http://dx.doi.org/[DOI].
If you would like to know when your article has been published online, take advantage of our free alert service. For registration and further information go to: http://www.link.springer.com.

Due to the electronic nature of the procedure, the manuscript and the original figures will only be returned to you on special request. When you return your corrections, please inform us if you would like to have these documents returned.

# Metadata of the article that will be visualized in OnlineFirst

| | | |
|---|---|---|
| ArticleTitle | A novel contextual memory algorithm for edge detection | |
| Article Sub-Title | | |
| Article CopyRight | Springer-Verlag London Ltd., part of Springer Nature<br>(This will be the copyright line in the final PDF) | |
| Journal Name | Pattern Analysis and Applications | |



| | | |
|---|---|---|
| Corresponding Author | Family Name | **Dorobanţiu** |
| | Particle | |
| | Given Name | **Alexandru** |
| | Suffix | |
| | Division | Computer Science Department |
| | Organization | Lucian Blaga University of Sibiu |
| | Address | 550024, Sibiu, Romania |
| | Phone | +40745572995 |
| | Fax | |
| | Email | alexandru.dorobantiu@ulbsibiu.ro |
| | URL | |
| | ORCID | http://orcid.org/0000-0003-4982-6930 |
| Author | Family Name | **Brad** |
| | Particle | |
| | Given Name | **Remus** |
| | Suffix | |
| | Division | Computer Science Department |
| | Organization | Lucian Blaga University of Sibiu |
| | Address | 550024, Sibiu, Romania |
| | Phone | |
| | Fax | |
| | Email | |
| | URL | |
| | ORCID | http://orcid.org/0000-0001-8100-1379 |





| | | |
|---|---|---|
| | Received | 30 March 2018 |
| Schedule | Revised | |
| | Accepted | 22 March 2019 |





| | |
|---|---|
| Abstract | Edge detection plays an important role in many computer vision systems. In this paper, we propose a novel application agnostic algorithm for prediction of probabilities based on the contextual information available and then apply the algorithm for estimating the probability of pixels belonging to an edge using surrounding pixel values as local contexts. We then proceed to test different image transformations as input layers, such as the Canny edge detector. We propose two different architectures, one single layered and one multilayered, which approach the scaling problem by creating scaled side outputs and combining them via a logistic regression layer. We tested our approach on the BSDS500 edge detection dataset with optimistic results. |





| | |
|---|---|
| Keywords (separated by '-') | Edge detection - Local context - Neural network - Probabilistic method - BSDS500 benchmark |

Footnote Information

Pattern Analysis and Applications
https://doi.org/10.1007/s10044-019-00808-0

SHORT PAPER

# A novel contextual memory algorithm for edge detection

Alexandru Dorobanţiu[1] · Remus Brad[1]

Received: 30 March 2018 / Accepted: 22 March 2019






**Abstract**
Edge detection plays an important role in many computer vision systems. In this paper, we propose a novel application agnostic algorithm for prediction of probabilities based on the contextual information available and then apply the algorithm for estimating the probability of pixels belonging to an edge using surrounding pixel values as local contexts. We then proceed to test different image transformations as input layers, such as the Canny edge detector. We propose two different architectures, one single layered and one multilayered, which approach the scaling problem by creating scaled side outputs and combining them via a logistic regression layer. We tested our approach on the BSDS500 edge detection dataset with optimistic results.

**Keywords** Edge detection · Local context · Neural network · Probabilistic method · BSDS500 benchmark

## 1 Introduction

The process of segmentation selects a set of pixels from an image, based on rules and patterns. The labeling of the extracted sets allows the user to obtain more information from the image. Typical rules for segmentation include the grouping of pixels by color, intensity or texture. Nevertheless, by targeting information extraction, rules can head toward finding edges of objects, areas, specific shapes and volumes, when applied to stacks of images. Automatic annotation of images and video gains more support each year.

Practical applications of image segmentation include machine vision, control systems, object detection (for example, face detection and pedestrian detection), recognition tasks (for example, fingerprint and iris recognition) and last but not least, medical imaging. Medical image segmentation borrows from many general-purpose segmentation techniques but combines them with domain-specific knowledge in order to obtain better results. Shape analysis and volume evolution make medical image segmentation important for diagnostics and treatment plans.

Recent methods, like deep learning through convolutional neural networks (CNN), proved to give state-of-the-art results in 2D benchmarks, but neural network architectures for 3D image processing are only now starting to emerge. Medical image segmentation integrated these techniques for both 2D [1] and 3D [2] segmentations.

In this paper, we introduce a general algorithm for automated learning, which stands as a basis for various applications. We provide a description of the algorithm and point out differences from various implementations tested. To prove the effectiveness, we have applied and tested the algorithm on an edge detection benchmark, with promising results. As for now, the algorithm was applied only for 2D images, but in the future, we plan to extend it for volumetric images, as an application in medical imaging.

## 2 Related work

Edge detection has been a subject of research for many years, with papers as early as 1975 [3]. Since then, a large number of techniques have been approached, targeting different aspects of edge detection, like closed contours, human-like perception or fast detection. In the following, we provide an overview of more recent methods used, organized by the main strategy of the algorithm.

In [4], a method based on oriented gradient signals is described. These are obtained from splitting the input image into the three CIELAB color channels and a texture channel. After applying filtering on the channels with 17 Gaussian kernels, the results are clustered using a K-means algorithm. An image is formed using the result for each pixel on which

✉ Alexandru Dorobanţiu
alexandru.dorobantiu@ulbsibiu.ro

[1] Computer Science Department, Lucian Blaga University of Sibiu, 550024 Sibiu, Romania



| Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019 |
|---|---|---|---|---|

a non-maximal suppression is applied. So far, only local information was used for each pixel. For the global information, spectral clustering is used. The probability of a pixel belonging to the contour is a weighted combination of the local and global information.

Real-time frame rates have been recently achieved using random decision for edge detection [5]. Local image patches were used as a basis for learning structure labels. These are then mapped into a discrete space using a decision tree which splits the data based on a decision function. Learning was done by independently training the trees in a recursive manner using information gain criteria.

Deep learning architectures started to prove state-of-the-art results at impressive processing speed of 0.4 s using the holistically nested edge detection (HED) architecture [6]. The networks map whole image to image predictions which provide a significant advantage as a global information method. A convolutional neural network with hierarchical representations provides the high-level features, visible as side outputs of the network. Deep supervision is used for training the layers, which are blended using a weighted fusion layer to provide the final image.

Relaxed deep supervision (RDS), proposed in [7], also relies on deep learning, but the main difference from HED is that RDS accepts as input predictions from other edge detectors such as Canny, called relaxed labels. HED itself is used as a provider for relaxed labels. The network tries to eliminate most of the false positives from all the intermediate layers.

The current state of the art is another recent deep learning approach based on richer convolutional features [8]. Similar to HED, it has a phase of producing side outputs, but in this case, a VGG16 architecture was used. Instead of using only the final convolutional layers for merging like previous approaches did, this architecture encapsulates in a holistic manner features from all convolutional layers and then trains the network via backpropagation.

Though the research on deep network focuses only on developing network architectures and not new techniques, other approaches are continually tested in the literature with promising improvements for low-level features, with the main advantage being that you do not need a powerful video card to run these algorithms thus can run on most equipment and the results are good enough for further processing. One such algorithm which improves the Canny edge detector is described in [9]. After applying an improved anisotropic diffusion filter, gradient templates are used for four directions. Then, an adaptive threshold is computed based on the histogram of the image, making the output more resilient to noise.

Another algorithm of this type revolves around hierarchical graph partitioning [10]. The algorithm employed is called Divide and Link, and it is used for a hierarchical network clustering. Unlike our proposed method, pixels are here modeled as nodes in a graph and a dissimilarity order between pixels determines the clusters in the graph. The regions are then transformed into a boundary map with a selection of the largest area of neighbor regions to provide the border.

## 3 Basis for the contextual memory

Before detailing the proposed algorithm, we will present some methods and techniques which stand as a basis for the concept of contextual memory.

### 3.1 Logistic regression

For a regression model, where the dependent variable is a categorical, the logistic regression can be used. In our case, the outcome is binary, where an image pixel belongs to a certain group or not. Binary logistic regression is useful for estimating the probability of class membership. Useful for making such predictions, a single-layer neural network has the output probability:

$$P_r(Y_i = 1|X_i) = p_i = \frac{1}{1 + e^{-(\beta_0+\beta_1 x_{(1,i)}+\beta_2 x_{(2,i)}+\cdots+\beta_k x_{(k,i)})}} \tag{1}$$

where $p_i$ is the probability that the output $Y_i$ belongs to the class, defined as a sigmoid function of a linear combination of the $k$ explanatory variables $X$, and $\beta_j$ for $j=1,\ldots,k$ are the parameters to be estimated, usually called coefficients or weights. The sigmoid function takes any input $x \in \Re$ and outputs a value between zero and one, making it interpretable as a probability. This function is also preferred because it has a continuous derivative.

$$y = \frac{1}{1 + e^{-f(x)}}, \quad \frac{\mathrm{d}y}{\mathrm{d}x} = \frac{y(1-y)\mathrm{d}f}{\mathrm{d}x} \tag{2}$$

The inverse of the logistic function, sometimes called the stretch function, is defined as:

$$g(t) = \ln\left(\frac{t}{1-t}\right) \tag{3}$$

There are numerous numerical ways to estimate the coefficients [11]; relevant here is the stochastic gradient descent. Minimizing a function by following the gradient of the cost is called gradient descent. If the loss is accounted for the entire training set or a subset of the training set, the method is called batch gradient descent. If the batch is the size of one, we will have a stochastic gradient descent. For each instance $i$ of the training set, we will make a prediction and then suffer a loss. If we apply the backpropagation





Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019

algorithm, the weights will be updated according to the following rule:

$$\beta_j = \beta_j + \alpha\, x_j\, (y_i - p_i)\, p_i\, (1 - p_i) \tag{4}$$

where $\beta_j$ is the weight for the $j$th input variable $x_j$, $p_i$ is the output probability for the instance $i$, and $y_i$ is the label of the instance (0 or 1). The learning rate $\alpha$, usually chosen empirically, limits the amount of correction for each coefficient. This is the gradient descent in weight space which minimizes the root-mean-square error.

Instead, if we want to minimize the relative entropy $-\log(1-p_i)$ if $y_i = 1$ or $-\log(p_i)$ if $y_i = 0$, then we will apply Eq. (5).

$$\beta_j = \beta_j + \alpha\, x_j\, (y_i - p_i) \tag{5}$$

This minimizes the amount of information lost when a prior probability distribution $Q$ is used to approximate a posterior probability distribution $P$.

### 3.2 Ensemble learning

Combining multiple hypotheses in order to obtain a better hypothesis is called ensemble learning. Ensembles are supervised learning meta-algorithms which, after training, can be used to make predictions. The ensemble also represents a hypothesis. The set of hypotheses which compose the ensemble do not necessarily contain the output hypothesis, making it likely to better describe the data. If not carefully prevented, this can lead to overfitting the training data.

As expected, using ensemble methods requires more resources (memory and computation time) than single models. Various techniques make a trade-off between the speed of computation, amount of memory used and the accuracy of the model.

In the process of designing artificial neural networks, creating multiple models and combining them are called ensemble averaging. The ensemble should perform better than the individual models because the error averages out. Instead of picking the prediction of only one of all the models as many ensemble techniques do, all the models are kept and the ones which are more prone to error are assigned a smaller weight. This can be expressed as a linear combination of experts. The output of the ensemble can be computed like in Eq. (6)

$$y(X;\alpha) = \sum_{j=1}^{k} \alpha_j y_j(x) \tag{6}$$

with $\alpha$ a set of weights, $y_j$ as the $j$th model prediction. Numerically, optimizing $\alpha$ is already a solved problem when applying the neural network learning rules.

The properties of the models upon which the ensemble averaging is built upon are [12]:

1. In any network, the bias can be reduced at the cost of increased variance
2. In a group of networks, the variance can be reduced at no cost to bias

False assumptions in the learning algorithm lead to bias error. High bias leads to missing the correlations between the data features and the target outputs (under fitting). Sensitivity to small fluctuations in the training set causes variance error. High variance can cause overfitting, meaning that the model has learned mostly noise instead of generalizing for unseen data. Given trained models with low bias but high variance, the result of the ensemble averaging is expected to have both low bias and low variance.

One of the variants of ensemble averaging is the negative correlation learning. This algorithm attempts to train and to combine individual networks in an ensemble in the same learning process [13].

Boosting is meta-algorithm which aims to create a strong learner from a collection of weak learners. As a rule of thumb, a series of weak classifiers are trained with respect to a probability distribution and are added to the set which is the basis for the strong classifier. This sequential introduction of weak learners keeps them focused on the samples previous learners misclassified.

AdaBoost, short for "adaptive boosting", is a type of ensemble learning. A boost classifier has the following form:

$$F_T(x) = \sum_{t=1}^{T} f_t(x) \text{ with } f_t(x) = \alpha_t h(x) \tag{7}$$

$f_t\,(x)$ is a weighted weak learner, while $x$ is the sampled input. For a binary classification task, the output of the learner is considered of class 0, if the value is negative, or class 1, if the value is positive. The absolute value of the output should be interpreted as confidence in the result.

The output of a weak learner for the sample $i$ is called the hypothesis $h(x_i)$. At iteration $t$, a weak learner is chosen and weighted with a coefficient $\alpha_t$. The value of $\alpha_t$ should minimize $E_t$ which is the training error at stage $t$. The error is computed using Eq. (8)

$$E_t = \sum_i E\left(F_{t-1}(x_i) + \alpha_t h(x_i)\right) \tag{8}$$

where $F_{t-1}(x)$ is the boosted classifier built up to the previous stage and $E$(F) is an arbitrarily chosen error function. Recently, convex potential boosters received criticism regarding convergence where random classification noise is present [14].

Stacking refers to blending the predictions of multiple machine learning models. With a specifically given



Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019

combiner algorithm, stacking can represent most if not all ensemble techniques. Usually, a logistic regression layer is used as the combiner.

### 3.3 Context modeling

Context modeling describes how the context information is structured and maintained. Depending on the problem, different types of discriminator contexts are useful. Local context refers to the aspect of the data examined, as opposed to context value, which represents the numeric value of that context. The context value will be used for accessing the memory structure. The memory takes a context value as input and outputs a value used for computing the output probability of belonging to a certain class.

When discussing edge detection and describing whether a pixel belongs to an edge or not, one of the most important aspects to take into consideration is the neighboring pixels. For instance, as a context, we can take the top pixel and left pixel. The context value for this example would be top pixel intensity 255, left pixel intensity 20.

Choosing neighboring pixels means using a local context, because we do not input the entire image when deciding if the focused pixel is an edge or not. Even more, there is no clear rule on how far away we should look when making such an assumption.

We can define contexts as rays, starting from the focused pixel and going in a straight line away in a given direction. Rays are defined by direction and length. Rays could be chosen from 1 to $L$, the maximum context length. We define direction by the angle of the ray, and we choose the angle by dividing 360 degrees by the number of rays we want to use. For example, if we use 8 rays, we have 4 rays, one for each axis, and 4 rays for the diagonals, as in Fig. 1, for length 5 and 8 rays.

The context value need not be only pixel values; we can take any function of the pixel values as well. Instead of directly using the values of the pixels, we can take the numerical derivative of the pixels in the direction of the ray. Of course, one can mask or quantize the values of the pixels or the value of the derivative. In the case of color images, rays for each color channel can be individually taken as contexts.

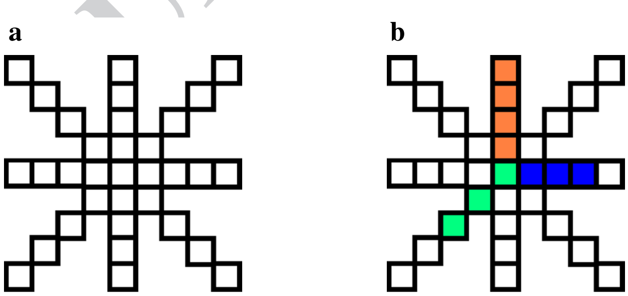



**Fig. 1** **a** Position of pixels as rays of length 5; **b** ray length 3 (green), length 4 (blue) and length 5 (orange)

### 3.4 Resources and hashing as a solution

If we allow contexts of any length to be used, we quickly hit the wall of practical limitations. Take, for instance, context values as pixels from a color channel which are represented as bytes. A context of length only 4 already means $2^{32}$ possible values. But not all the values will be relevant, as most of the possible values will represent noise.

To overcome this issue, we use hashing functions to map them to the chosen data structures. A hashing function maps data of arbitrary size to data of fixed size. This makes them useful for data structures like hash tables. One at a time hash functions are useful for data which comes in byte chunks, especially when we want to keep the intermediary hash values.

For the current work, we analyzed in particular two different types of hash functions: Jenkins hash function [15] and Fowler–Noll–Vo hash function [16]. Both of them are non-cryptographic but were chosen for their speed of computation and rather low collision rate.

Both functions can be altered to keep the intermediate results of the hash value, useful when working with rays of increasing length.

## 4 An original method for contextual prediction

Applied on images, the context modeling part of the algorithm takes as input an image and a position. The position is used as a basis for the rays which are modeled as position deltas from the focused pixel. The values of the pixels are taken from the color channel of the image and are then hashed. The result of the hash is the context value, which we later use for indexing. As a result, we have as many context values (numbers) as the number of contexts.

The memory can be organized as a map with a separate table for each context in order to prevent collisions between contexts. There are more ways to organize the memory, and some suggestions will be discussed in the implementation details.

### 4.1 Model prediction

In order to make a prediction, we have proposed the following algorithm:





| Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019 |
|---|---|---|---|---|

1. We obtain a value from the memory for each context. One way to do that is to index the hash of the context value in a table
2. We average all the obtained memory values
3. Convert the average into a probability using the sigmoid function
4. Refine the probability using a transfer function and obtain the output probability (Eq. 9)

$$p = T\left(\sigma\left(\frac{k}{n}\sum_{i=0}^{n} v_i\right), C\right) \text{ with } v_i = M[i]\left[hash(c_i)\right] \tag{9}$$

where $p$ is the probability that the pixel belongs to a class (output probability), $n$ is the number of input contexts, $c_i$ is the context value of the $i$th context, $v_i$ is the value from the memory $M$ for context $i$, $k$ is some ad hoc constant, $T$ is an adaptive probability map transfer function which takes as input a probability and a small context value $C$ and outputs a refined probability, and $\sigma$ is the sigmoid function.

The adaptive probability map (APM), sometimes called secondary symbol estimation (SSE), is used to fine tune a probability and works in the following way:

- Select a set of interpolation points according to a context value $C$: *points = pointset[C]*
- Find the two points indexes between which the input value falls: index low and index high
- Output the probability as the weighted average of the two values from the points. The weight is selected from how far the input is from the two points:

$$\begin{aligned}\text{Output} = {} & \text{points[index low]} * (1 - \text{weight}) \\ & + \text{points}\left[\text{index high}\right] * \text{weight}\end{aligned} \tag{10}$$

The input probability $p$ can be mapped to point indexes in more ways. A simple way is to quantize the probability linearly to the number of points $N_\mathrm{p}$. This is equivalent to

$$\text{Index low} = \left[p * N_p\right], \quad \text{Index high} = \left[p * N_p\right] + 1 \tag{11}$$

where [ ] denotes the *floor* operation.

Another way to quantize the probability is to stretch the probability first and then quantize linearly to the number of points. This serves the purpose to allocate more points close to zero and one, where fine-tuning makes more sense. The input value for the APM is now a stretched probability.

The following two diagrams plot the initial point values for the two methods described with respect to the input. When no value was changed, the output of the APM should be equal to the input probability. The $X$-axis represents the input value to be quantized, and the $Y$-axis represents the point values (probability) (Fig. 2).

In logistic regression, every feature is individually weighted before applying the sigmoid function to the result. There is no assumption about the origin of the numeric value of the feature. When we know that the input features are probabilities, logistic regression can be seen as a way of combining them. Mattern [17] proved that if instead of using plain probabilities as input, we use stretched probabilities (inverse logistic function), that logistic mixing is optimal in the sense of minimizing Kullback–Leibler divergence, or wasted coding space, of the input predictions from the output mix. Stretching the input probabilities makes logistic regression a form of geometric weighting instead of linear weighting:

$$\beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + \cdots + \beta_k x_{k,i} \tag{12}$$

where $x_{j,i}$ is a probability becomes

$$\beta_0 + \beta_1 t_{1,i} + \beta_2 t_{2,i} + \cdots + \beta_k t_{k,i} \tag{13}$$

where $t_{j,i} = \ln\left(\frac{x_{j,i}}{1-x_{j,i}}\right)$ and the update formula for minimizing the relative entropy could be written:

$$\beta_j = \beta_j + \alpha\, t_j(y_i - p_i) \tag{14}$$

Coming back to our case, we ask the question: where does the predictive power of an ensemble come from? Where does this information lay, in the aggregating



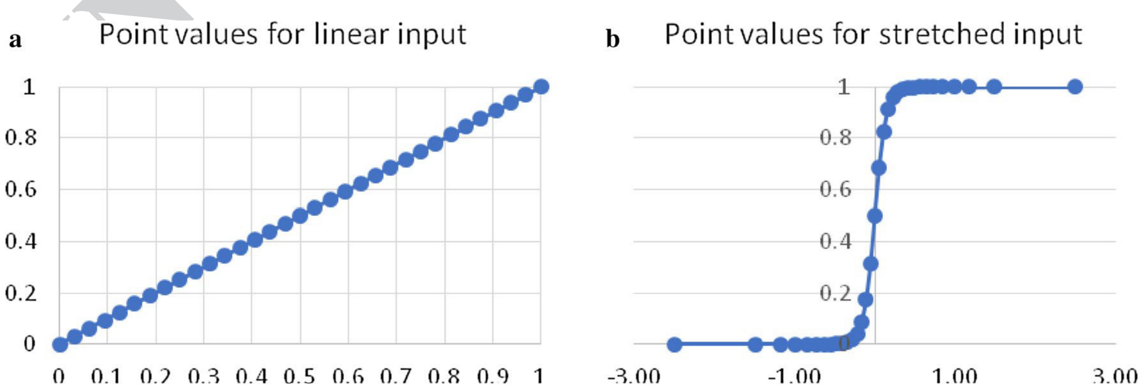



**Fig. 2** **a** Uniform distribution of points; **b** non-uniform distribution of points



| Journal : **Large 10044** | Article No : **808** | Pages : **13** | MS Code : **PAAA-D-18-00135** | Dispatch : **29-3-2019** |
|---|---|---|---|---|

algorithm or the individual components? It is obvious that if we use a form of logistic regression, the set of weights carries some information. It can be argued if that more weights are somehow added to the aggregating algorithm, it would be beneficial to its predictive power.

To some extent following this, an algorithm that adds more layers of logistic regression is the context mixing algorithm. The algorithm has been successfully applied in state-of-the-art data compression programs like PAQ [18] and *cmix* [19], which rank first in most text and general data compression benchmarks for their compression ratio.

The algorithm works in the following way:

- Instead of mixing the input probabilities with only one set of weights, create a number $N$ of buckets of sets of weights
- Choose one set of weights from each of the $N$ buckets according to a function of context
- Applying logistic regression with each set of weights will result in $N$ probabilities, which will be mixed by another set of weights chosen from its own bucket.

One could underline that some information regarding context is passed to the mixing ensemble, hence the name context mixing. If instead we want to move the mixing information from the ensemble toward the weak learners, we need to take into consideration the following: How to pass information back to the learners? We made little assumptions so far about the value of $v_i$. We know that the value comes from a big bucket of entries, where its index is dependent on the context. In a regression sense, the feature is the context, not $v_i$. We interpret this value like $v_i = \beta_i + t_i$, where $t_i$ is a stretched probability for the context value and $\beta$ is the weight of the probability in the ensemble. Averaging the memory values:

$$\frac{k}{n}\sum_{i=0}^{n} \beta_i t_i \tag{15}$$

which itself is a stretched probability. Applying the sigmoid function converts this average back into a probability.

## 4.2 The proposed model update

To update the model, we propose dual objective minimization:

- Minimize the error with respect to the output of the entire network
- Minimize the error with respect to the output of the individual node

An important remark here is that we can update the model from a supervised learning point of view or from a reinforcement learning point of view. Supervised learning means that we know the precise probability for the case we were predicting, and we use that for backpropagation. Reinforcement learning means we do not know the exact probability for the case and do not even know how to obtain it, but instead we rely on maximizing a cumulative reward in an on-line manner, given the interaction with the environment. In our case, instead of backpropagating a probability, we can use the binary outcome instead and try to minimize either the cumulative logistic loss or the cumulative square loss (Fig. 3).

Working with stretched probabilities for logistic regression results in the global update error to be:

$$E_g = \beta_g(p - y) \tag{16}$$

for minimizing logistic loss, or

$$E_g = \beta_g(p - y)p(1 - p) \tag{17}$$

for minimizing the square loss, where $E_g$ is the global error, $\beta_g$ is the global error learning rate, $p$ is the output probability and $y$ is the information available as ground truth, that can be a binary outcome or a probability.

After computing the global error, we proceed in computing local errors for each memory value and updating them. The local error is defined as:

$$E_l = \beta_l(p_i - y) \text{ with } p_i = \sigma(v_i) \tag{18}$$

for minimizing logistic loss, or

$$E_l = \beta_l(p_i - y)p_i(1 - p_i) \text{ with } p_i = \sigma(v_i) \tag{19}$$

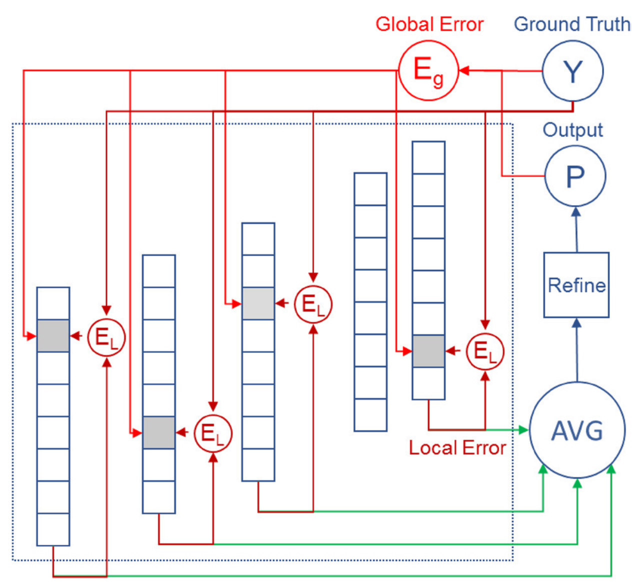



**Fig. 3** Block scheme for the proposed update algorithm



| Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019 |
|---|---|---|---|---|

for minimizing the square loss, where $E_l$ is the local error, $\beta_l$ is the local error learning rate, $p_i$ is the output probability for the $i$th context (side prediction), computed as the sigmoid of the memory value $v_i$, and $y$ is the ground truth.

Each memory value is then updated by subtracting both the local and the global errors:

$$v_i = v_i - E_l - E_g \tag{20}$$

It is important to notice that the global error is computed using the output probability which was refined by the adaptive probability map. This is not mandatory, but our tests show better results when the refined probability is used. This can be seen as allowing the model to learn something that can be corrected.

Updating the adaptive probability map works in the following way: the point values for *index high* and *index low* are adjusted to reduce the prediction error. Error for index low is

$$(\text{Points}[\text{index low}] - y) * (1-\text{weight}) * \text{learning rate} \tag{21}$$

the error for *index high* is

$$\left(\text{Points}\left[\text{index high}\right] - y\right) * \text{weight} * \text{learning rate} \tag{22}$$

where $y$ is the ground truth available. Of course, variations of this can be applied, such as updating only the closest value.

Compared with the logistic regression, this method does not update the weights of the mixture, since the combining function is not a dot product, but it updates directly the values which participate in the average.

It is still a form of ensemble learning, particularly ensemble averaging, where the elements of the ensemble here are the memory values. The ensemble error is the global error.

One can argue that this method is similar to boosting, regarding the fact that each side prediction is a weak learner, and the output after mixing is the strong learner. Depending on the memory implementation chosen, which will be discussed in the next chapter, additional and possible longer contexts with unseen data make up for the bias of shorter contexts and can be added or evicted when accounting for the memory size limitations. Even though the error is backpropagated depending on the context, it also differs from context mixing, because there is no mixing layer to separate the context weights from the input probabilities.

## 5 Results

### 5.1 Inputs, preprocessing and processing architecture

We implemented the contextual memory algorithm for an edge detector application. This section describes the architecture and some of the implementation details of the application. The application is implemented in the C# programming language, since we wanted to not restrict the testing and usage of the application to a closed scripting environment like MATLAB. Using a strongly typed programming language also helps with choosing better data structures. The source code is publicly available at the GitHub page https://github.com/AlexDorobantiu/ContextualMemoryEdgeDetection.

Even when talking about two-dimensional images, color images have more layers in the dimension of the RGB colors. Hence, three layers can go as an input for the algorithm. These layers are preprocessed using a chain of preprocessors. We used a Gauss filter for eliminating the noise in the input images. This takes the original RGB layers as input and outputs a three-layer image. We used a filter of size 5 and a sigma value of 1.4.

Since the algorithm makes no assumption of the data behind contexts, it can be benefic to include transformations of the color layers. Such transformations are appended as other layers for the algorithm input image. We optionally used the Sobel filter, the Canny edge detection algorithm and a Kirsch edge detection algorithm as input, which take the color channels and output another layer. This makes the final input image to have three or more layers. Having a Canny layer, or any edge detector as input, is a sort of domain-specific knowledge added in the model.

The single-layer architecture takes a preprocessed image as an input and uses a given set of color channels to compute an output image which consists of a single-layer grayscale image in which the pixels represent the probability that the position in the original image belongs to an edge. The single-layer architecture combined with the simple rays as contexts does not take into account information about the edge being kept the same at different zoom levels.

To tackle the zooming problem, the multilayer architecture behaves in this way: take the original image, apply preprocessing, obtain the output; then take the original image, apply the preprocessing, append the previously obtained output as a layer, resize the image (meaning all its layers), and use it as an input for the algorithm. If one decides to separate the memory used by the algorithm at different sizes, we have a multilayer architecture. Each layer is trained separately starting from the largest image and going toward resized images. An algorithm layer can have different configurations from the other layers, and options can include the length of the longest ray, the preprocessing done, the memory size and others. The result of the multilayer architecture is a set of grayscale images of varying sizes, which are called side outputs. These side outputs are then combined (blended) using a logistic regression layer, to form a single image. Before blending,






| Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019 |
|---|---|---|---|---|

the images are scaled to have the same size. The weights of the logistic regression layer are also trained on the training set.

### 5.2 Results on Berkeley edge detection benchmark

In this chapter, we present one example of the output images from the algorithm using image "326025.jpg" from the mentioned benchmark dataset along with some of the parameters used and a short description of the differences. We also provide an analytical evaluation of the improvements compared to the Canny edge detection method (Fig. 4).

The global parameter values used in the tests were $\beta_l = 0.25$, $\beta_g = 0.5$ and $k = 2$, and quantized derivative rays have the two least significant bits quantized. The results for single-layer architecture are shown in Table 1, (Table 2 and Fig. 5).

Before computing the score for the benchmark, the output images are subjected to a non-maximal suppression technique and subsequently to an edge thinning. This is done in MATLAB, using an adapted version of the Piotr Dollar's Structured Edge Detection Toolbox, available at https://github.com/pdollar/edges.

The benchmark provides a tool for evaluation which has an automated search in the space of thresholding, so that the user feels free to leave grayscale images instead of making the binarization himself. We compare the algorithm with the well-known Canny edge detector, the recent ID&L algorithm [10] and to the state-of-the-art deep learning algorithms HED [6], RDS [7], RCF-ResNet101-MS [8], CED-VGG16 [20], AMH-ResNet50 [21], CASENet [22] and CEDN [23] (Table 3).

We can clearly see from Fig. 6 that the learning is shifted toward not taking any risks, since the better F1 score is achieved in the low threshold settings. In order to obtain an overall better F1 score, class balancing must be considered when applying the loss function.

We made another analytical comparison using the cross-entropy measure. If we have two probability distributions, we can measure the number of bits needed to identify an event drawn from a set if a coding scheme is used using a probability distribution other than the true distribution of the set. Since the pixel intensities in the resulting images can be modeled as a probability of a pixel belonging to an edge, we can measure the cross-entropy for the output images. We show a comparison with the Canny algorithm for the first 50 images in the test set of the benchmark in Fig. 7. A better probability modeling of the true source of the edges should have a lower cross-entropy. For the overall set, the proposed method obtained a cross-entropy of 6610784. In comparison, the Canny algorithm obtained a score of 11372700. This means that our algorithm surpassed the Canny algorithm by a factor of 1.72.

To prove the potential of the proposed method, we also considered the precision with respect to the threshold. Precision is the fraction of relevant instances among the retrieved instances. A high value represents a small rate of false positives. Compared with the Canny algorithm, the proposed method offers a steady increase in precision and a roughly constant better precision with respect to the threshold. This can be seen in Fig. 8.

Provided the results on the three measures, we show that the proposed algorithm has potential over the Canny method.

## 6 Conclusions

The main contribution of this paper is an application agnostic algorithm for prediction of probabilities based on the contextual information available, where learning can be done in an on-line fashion. More than this, it does not impose any constraints on how to choose or model the context. This freedom allows it to be used in many areas



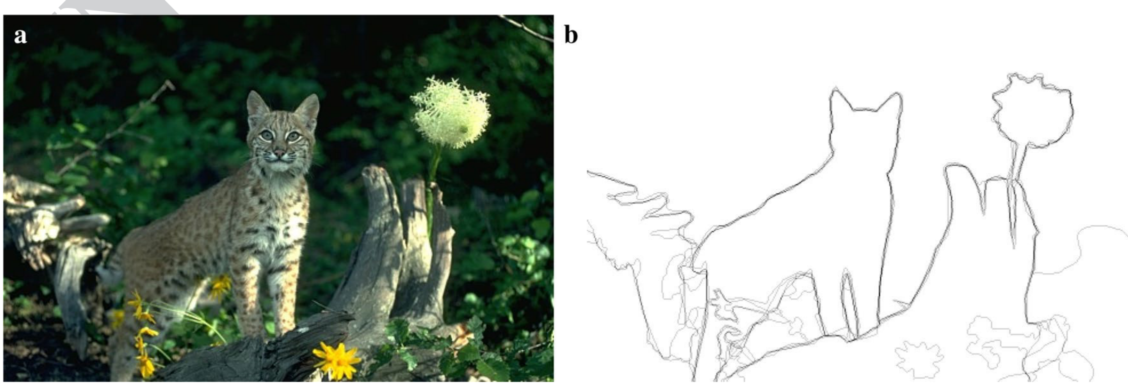
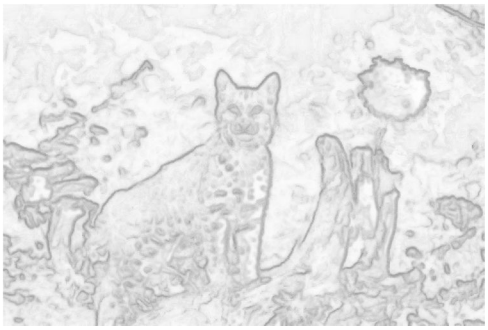
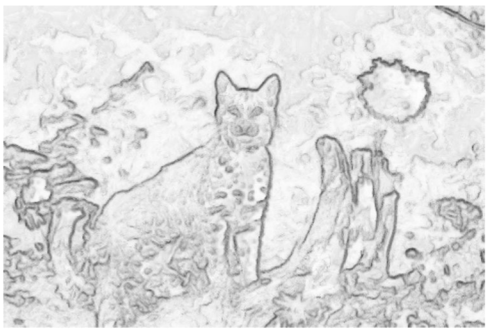



**Fig. 4** **a** Sample image from the benchmark along with **b** the ground truth

**Table 1** Some of the results for single layer

| Algorithm output | Description |
|---|---|
|  | Context model: 16 rays, max length: 8, including quantized derivative rays<br>Memory: direct lookup, table size 218<br>Loss function: square loss<br>Number of passes over training set: 1<br>Preprocess: Gauss |
|  | Context model: 16 rays, max length: 8, including quantized derivative rays<br>Memory: tagged lookup, table size 220<br>Loss function: square loss<br>Number of passes over training set: 1<br>Preprocess: Gauss |
|  | Context model: 16 rays, max length: 8, including quantized derivative rays<br>Memory: bucket lookup, table size 220<br>Loss function: square loss<br>Number of passes over training set: 1<br>Preprocess: Gauss, Canny, Kirsch |
|  | Context model: 16 rays, max length: 8, including quantized derivative rays<br>Memory: direct lookup, table size 220<br>Loss function: entropy loss, GT threshold: 63<br>Number of passes over training set: 1<br>Preprocess: Gauss, Canny, Kirsch |

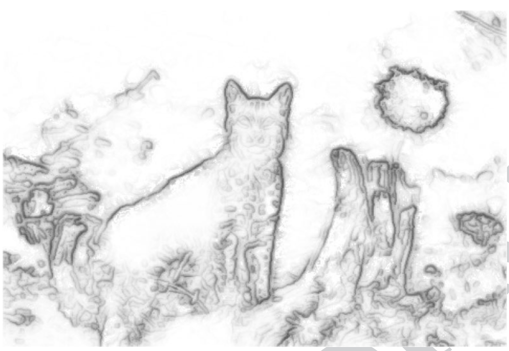
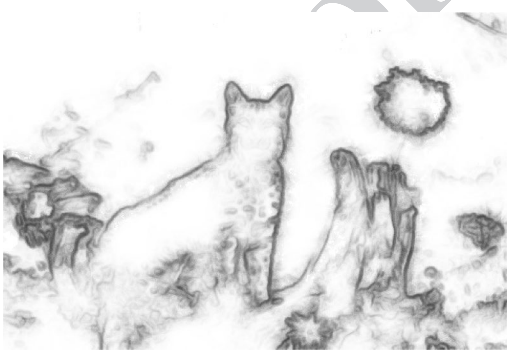


such as one-dimensional data, such as text information or data sequences, two-dimensional data, such as images, and three-dimensional data, such as volumetric images (or image slices which represent volumetric information). It also allows it to be part of larger learning and prediction structures, having loose requirements on what information is needed for feedback.



Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019

**Table 2** Some of the results for multilayer architecture

| Algorithm output | Description |
|---|---|
|  | Context model: 16 rays, max length: 8, including quantized derivative rays<br>Memory: direct lookup, table size 218<br>Loss function: square loss, GT threshold: 63<br>Number of passes over training set: 1<br>Preprocess: Gauss, Canny, Kirsch<br>Layers scale: 1, 2, 4, 8, 16 |
|  | Context model: 16 rays, max length: 8, including quantized derivative rays<br>Memory: bucket lookup, table size 219<br>Loss function: entropy loss, GT threshold: 63<br>Number of passes over training set: 1<br>Preprocess: Gauss<br>Layers scale: 1, 3, 5, 7 |
|  | Context model: 16 rays, max length: 8, including quantized derivative rays<br>Memory: bucket lookup, table size 219<br>Loss function: entropy loss, GT threshold: 63<br>Number of passes over training set: 1<br>Preprocess: Gauss, Canny<br>Layers scale: 1, 3, 5, 7 |
|  | Context model: 16 rays, max length: 8<br>Memory: bucket lookup, table size 219<br>Loss function: entropy loss, GT threshold: 63<br>Number of passes over training set: 1<br>Preprocess: Gauss, Canny, Kirsch<br>Layers scale: 1, 3, 5, 7 |



In order to demonstrate the usefulness of the method, a 2D edge detection application using the method was implemented. The results are promising, considering that no prior handcrafted knowledge has been added in the model to help with the prediction.

Allocating too much memory and more training passes over the training set leads to overfitting. The algorithm will learn by heart the training set and will reproduce meticulously the ground truth, but the quality of the results



| Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019 |
|---|---|---|---|---|

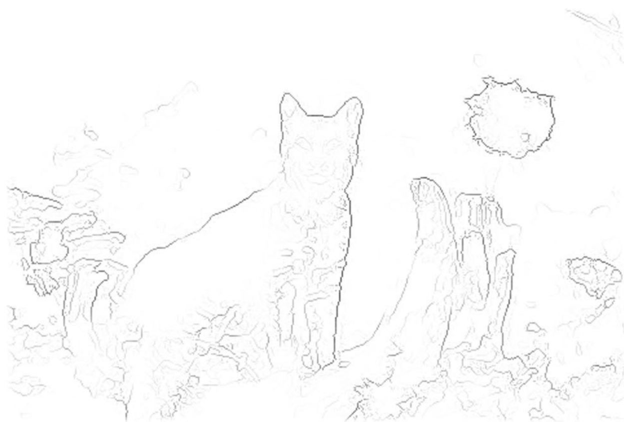


**Fig. 5** Output example after NMS and thinning

**Table 3** Comparative results

| Algorithm | F1 score | Precision | Recall | Threshold |
|---|---|---|---|---|
| Canny | 0.583 | 0.500 | 0.698 | 0.33 |
| ID&L [10] | 0.610 | 0.590 | 0.680 | N/A |
| Contextual memory (proposed) | 0.640 | 0.605 | 0.686 | 0.19 |
| CASENet [22] | 0.767 | N/A | N/A | N/A |
| HED [6] | 0.787 | 0.803 | 0.772 | 0.46 |
| RDS [7] | 0.792 | N/A | N/A | N/A |
| CED-VGG16 [20] | 0.794 | N/A | N/A | N/A |
| AMH-ResNet50 [21] | 0.798 | N/A | N/A | N/A |
| CEDN [23] | 0.788 | N/A | N/A | N/A |
| RCF-ResNet101-MS [8] | 0.819 | N/A | N/A | N/A |

on the test set will downgrade. We do not propose any solution to the overfitting problem in this paper.

In the future, we intend to develop more in the direction of contextual modeling, which means exploring the space of choosing the appropriate contexts for various applications. We would also like to extend the existing application for three-dimensional images and apply it for medical image segmentation. Some of the changes needed to make when switching from 2D to 3D will be to model 3D contexts, which can be chosen as rays in three dimensions instead of two. Also, centerline and segmentation benchmarks have different intended objectives, so the feedback mechanism and the output metrics will have to be adapted to obtain results relevant to those benchmarks.

In order to improve the existing application, the following can be implemented:

- Replace the loss function with a cost-sensitive loss function, as described in [13], because the distribution of edge/non-edge pixels in the benchmark is biased 90% in favor of non-edges
- Model rays as individual blocks, whose output will be combined using a context mixing layer. This means each ray will output a probability, and the set of probabilities will be further combined into a single probability
- Replace the blending algorithm with an improved context mixing layer, maybe even with a fully connected layer such as in CNNs
- Add more convolutional layers as input, having a set of learnable filters
- Add the output of the state-of-the-art edge detectors as input layers, to see how the algorithm balances the precision and recall
- Implement a GPGPU version of both the single and multilayer algorithm
- Adapt the training phase to iterate over transformed versions of the input images, such as rotations and scaling, to gain more training data
- Exclude unconvincing ground truth data, when less than a half of the human subjects agree to the edge position
- Last but not least, design space exploration with respect to the parameters

Integrating with the relaxed deep supervision [7] algorithm should provide interesting results, since algorithm provides high precision on lower thresholds. We also expect the proposed network to integrate well with a deep learning architecture, especially with a CNN network, as a layer after the rectified linear units (ReLU) layer, side by side with the fully connected layer.





Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019

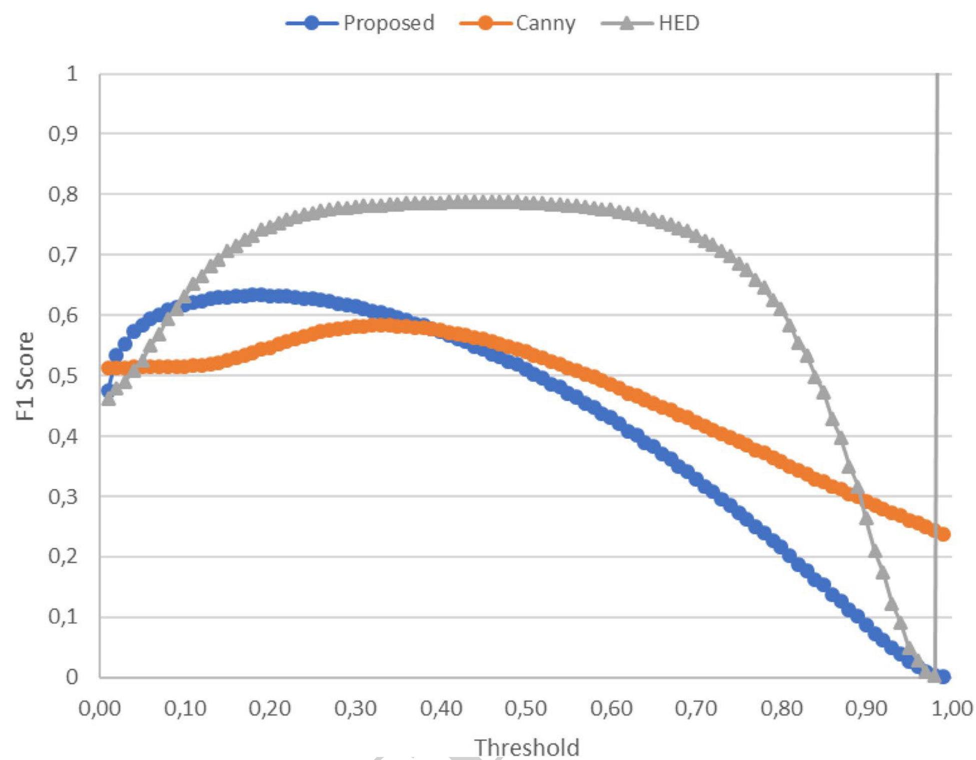



**Fig. 6** F1 score plotted against threshold

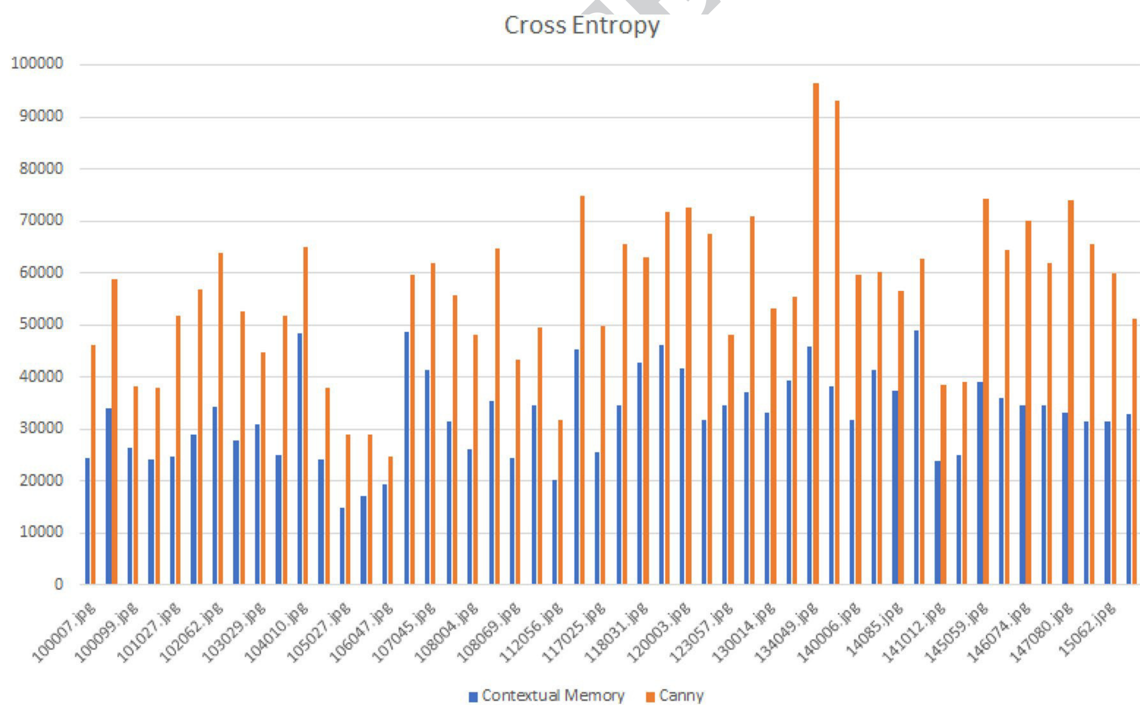



**Fig. 7** Cross-entropy for the first 50 images of the dataset (lower is better)



| | | | | |
|---|---|---|---|---|
| Journal : **Large 10044** | Article No : **808** | Pages : **13** | MS Code : **PAAA-D-18-00135** | Dispatch : **29-3-2019** |

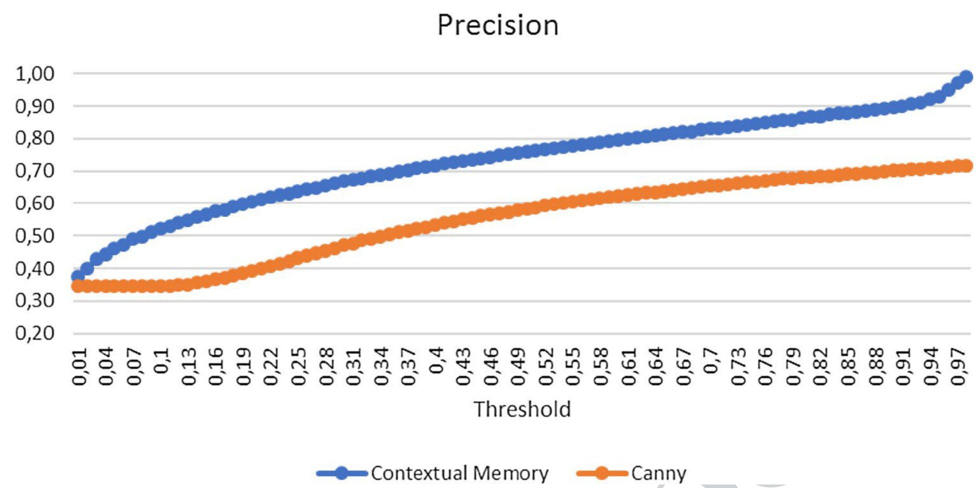



**Fig. 8** Precision with respect to the threshold (bigger is better)





## References


1. Gaonkar B, Hovda D, Martin N et al (2016) Deep learning in the small sample size setting: cascaded feed forward neural networks for medical image segmentation. Proc SPIE 9785:97852I
2. Milletari F, Navab N, Ahmadi S (2016) V-Net: fully convolutional neural networks for volumetric medical image segmentation. In: IEEE fourth international conference on 3D vision, pp 565–571
3. Fram JR, Deutsch ES (1975) On the quantitative evaluation of edge detection schemes and their comparison with human performance. IEEE Trans Comput C-24:6:616–628
4. Arbelaez P, Maire M, Fowlkes C, Malik J (2011) Contour detection and hierarchical image segmentation. IEEE TPAMI 33(5):898–916
5. Dollár P, Zitnick LC (2015) Fast edge detection using structured forests. IEEE Trans Pattern Anal Mach Intell 37(8):1558–1570
6. Xie S, Tu Z (2017) Holistically-nested edge detection. Proc IEEE Int J Comput Vis 125(1):3–18
7. Liu Y, Lew MS (2016) Learning relaxed deep supervision for better edge detection. In: IEEE conference on computer vision and pattern recognition (CVPR), pp 231–240
8. Liu Y, Cheng MM et al (2019) Richer convolutional features for edge detection. In: IEEE transactions on pattern analysis and machine intelligence; http://mftp.mmcheng.net/Papers/19PamiEdge.pdf. Accessed 05 Nov 2018
9. Fu F, Wang C et al (2018) An improved adaptive edge detection algorithm based on Canny. In: Proceedings of SPIE 10827 icOPEN. Accessed 24 Jul 2018
10. Guadaa C, Edwin Zarrazolab et al (2018) A novel edge detection algorithm based on a hierarchical graph-partition approach. J Intell Fuzzy Syst 34:1875–1892
11. Minka T (2017) A comparison of numerical optimizers for logistic regression. https://tminka.github.io/papers/logreg/minka-logreg.pdf. Accessed 05 Feb 2017
12. Naftaly U, Intrator N, Horn D (1999) Optimal ensemble averaging of neural networks. Netw Comput Neural Syst 8:3
13. Liu Y, Yao X (1999) Ensemble learning via negative correlation. Neural Netw 12(10):1399–1404
14. Long PM, Servedio RA (2010) Random classification noise defeats all convex potential boosters. Mach Learn 78(3):287–304
15. http://www.burtleburtle.net/bob/hash/doobs.html. Accessed 05 Feb 2017
16. http://www.isthe.com/chongo/tech/comp/fnv/index.html. Accessed 05 Feb 2017
17. Mattern C (2012) Mixing strategies in data compression. In: Proceedings of the 22nd data compression conference (DCC), pp 337–346
18. Mahoney M (2005) Adaptive weighing of context models for lossless data compression. http://mattmahoney.net/dc/cs200516.pdf. Accessed 05 Feb 2017
19. http://www.byronknoll.com/cmix.html AQ4
20. Wang Y, Zhao X et al (2018) Deep crisp boundaries. From boundaries to higher-level tasks. arXiv preprint arXiv:1801.02439
21. Xu D, Ouyang W et al (2017) Learning deep structured multi-scale features using attention-gated CRFs for contour prediction. In: Advances in neural information processing system, pp 3961–3970
22. Yu Z, Feng C et al (2017) CASENet: deep category-aware semantic edge detection. In: IEEE conference on computer vision and pattern recognition (CVPR), pp 21–26
23. Yang J, Price B et al (2016) Object contour detection with a fully convolutional encoder-decoder network. In: IEEE conference on computer vision and pattern recognition (CVPR), pp 193–202


**Publisher's Note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Journal : Large 10044 | Article No : 808 | Pages : 13 | MS Code : PAAA-D-18-00135 | Dispatch : 29-3-2019

| | |
|---|---|
| Journal: | **10044** |
| Article: | **808** |

# Author Query Form

**Please ensure you fill out your response to the queries raised below and return this form along with your corrections**

Dear Author

During the process of typesetting your article, the following queries have arisen. Please check your typeset proof carefully against the queries listed below and mark the necessary changes either directly on the proof/online grid or in the 'Author's response' area provided below

| Query | Details Required | Author's Response |
|---|---|---|
| AQ1 | Figures: figures (2,7) are poor in quality as its labels are not readable. Please supply a new version of the said figure with legible labels preferably in .eps, .tiff or .jpeg format with 600 dpi resolution. | |
| AQ2 | Please check and confirm the inserted citation of Figures 2, 3, 4 and 5 are correct. If not, please suggest an alternative citation. Please note that Figures should be cited in sequential order in the text. | |
| AQ3 | Please check and confirm the inserted citation of Tables 2 and 3 are correct. If not, please suggest an alternative citation. Please note that Tables should be cited in sequential order in the text. | |
| AQ4 | Please provide the Accessed date for reference [19]. | |