1. Abstract
2. Introduction
   1. Problem Explanation
   2. Applications
3. Related Work
   1. Various methods used to compute Midcurve
4. Proposed Method
   1. Model Architecture
   2. Dataset & Training
      1. Generating data
5. Experiments
   1. Proof Of Concept (Dense Encoder Decoder)
   2. CNN Encoder Decoder
   3. Adding Skip Connections
   4. Filled Profiles & Distance Transform
   5. Loss Function Variations (Beta)
6. Comparative Results & Discussion
   1. Comparison w/ other methods
   2. Some discussion
7. Conclusion

MidcurveNN: Neural Network for Computing Midcurve of a Thin Polygon

# Abstract

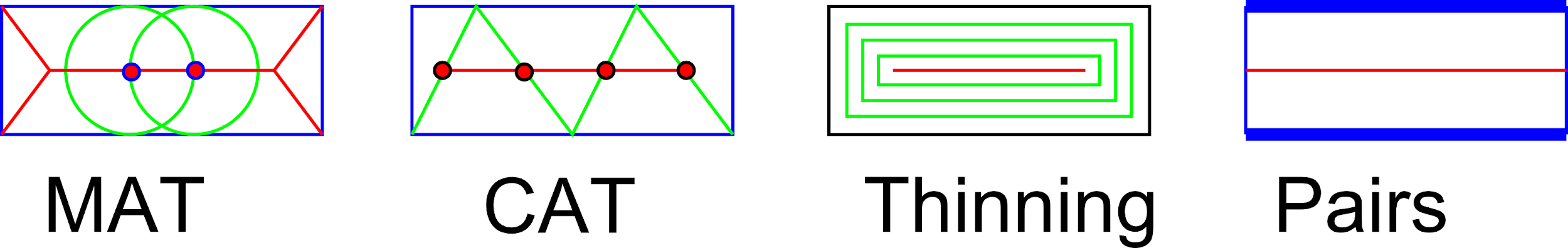
Various applications need lower dimensional representation of shapes. Midcurve is one-dimensional(1D) representation of a two-dimensional(2D) planar shape. It is used in applications such as animation, shape matching, retrieval, finite element analysis, etc. Methods available to compute midcurves vary based on the type of the input shape (images, sketches, etc.) and processing (thinning, Medial Axis Transform (MAT), Chordal Axis Transform (CAT), Straight Skeletons, etc.).

This paper talks about a novel method called MidcurveNN which uses Encoder-Decoder neural network for computing midcurve from images of 2D thin polygons in supervised learning manner. This dimension reduction transformation from input 2D thin polygon image to output 1D midcurve image is learnt by the neural network, which can then be used to compute midcurve of an unseen 2D thin polygonal shape.

# Introduction

A skeleton is a lower dimensional entity which represents the shape of its parent object. It being simpler than the parent object, operations like pattern recognition, approximation, similarity estimation, collision detection, animation, matching and deformation can be performed efficiently on it than on the parent object.

Skeletons, also known as Medial Objects, can be computed via various mathematical formulations/approaches such as Medial Axis Transform (MAT), Chordal Axis Transform (CAT), Pairing, Thinning etc. Figure \ref{fig\_medialmethods} shows some of these. More detailed analysis can be found in the survey paper \cite{medial2010}.

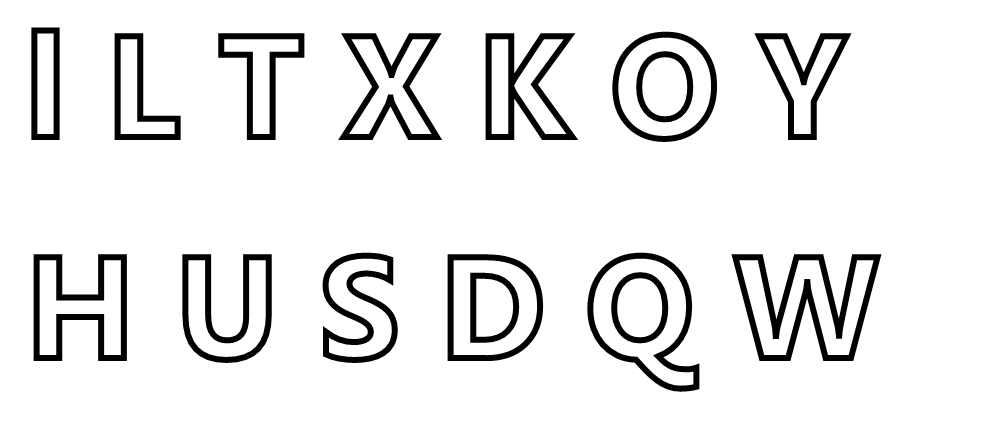


{images/MedialMethodsOnlyShort}

\captionof{figure}{Medial Object computation methods}

\label{fig\_medialmethods}

In the current paper we focus on computing midcurve for 2D planar sketch profiles. Even in 2D profiles, shapes vary enormously. As the first level of simplification, we would deal with 2D polygons only (with an assumption that curved shapes can be converted to polygonal shape by faceting). Figure \ref{fig\_letters} shows some of the input shapes which can be considered. English alphabets are chosen for easy understanding and verification of the proposed method.



{images/Letters}

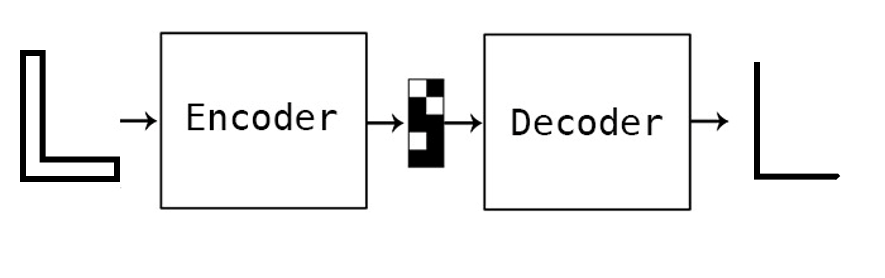
\captionof{figure}{2D Thin Polygonal shapes}

\label{fig\_letters}

# Proposed Method

Computation of midcurve, in its original form, is transformation of a 2D thin closed, with/without-holes polygon to 1D open/closed/branched polyline. Paper \cite{dimred2017} details one of the effective midcurve computation techniques, based on rule-based computational geometry approach. Such techniques have a shortcoming of not being scalable or generic enough to be able to handle variety of shapes. Deep Learning neural network architectures are showing potential of developing such generic models. This dimension reduction transformation should ideally be modeled as Sequence to Sequence (Seq2Seq) neural architecture, like Machine Translation. In the current problem, the input and the output sizes could be different not just in a single sample, but across all samples. Many current Seq2Seq neural networks need fixed size inputs and outputs, which if not present in data, are artificially managed by adding padding of certain improbable value. Such padding is not appropriate for the current midcurve computation problem, as the padding-value can inappropriately get treated as part of the valid input. In this initial phase, to circumvent the problem of variable size, image-based inputs and outputs are used, which are of fixed size. Both input and output polygonal shapes are rasterized into images of 100x100, thus making them fixed size for all samples, irrespective of the original shapes.

This paper proposes to use such network for midcurve computation in the form of image-to-image mode. Input images have thin polygonal shapes whereas output images have corresponding midcurve images. Figure \cite{fig\_endecoder} shows the Encoder-decoder architecture, called MidcurveNN.

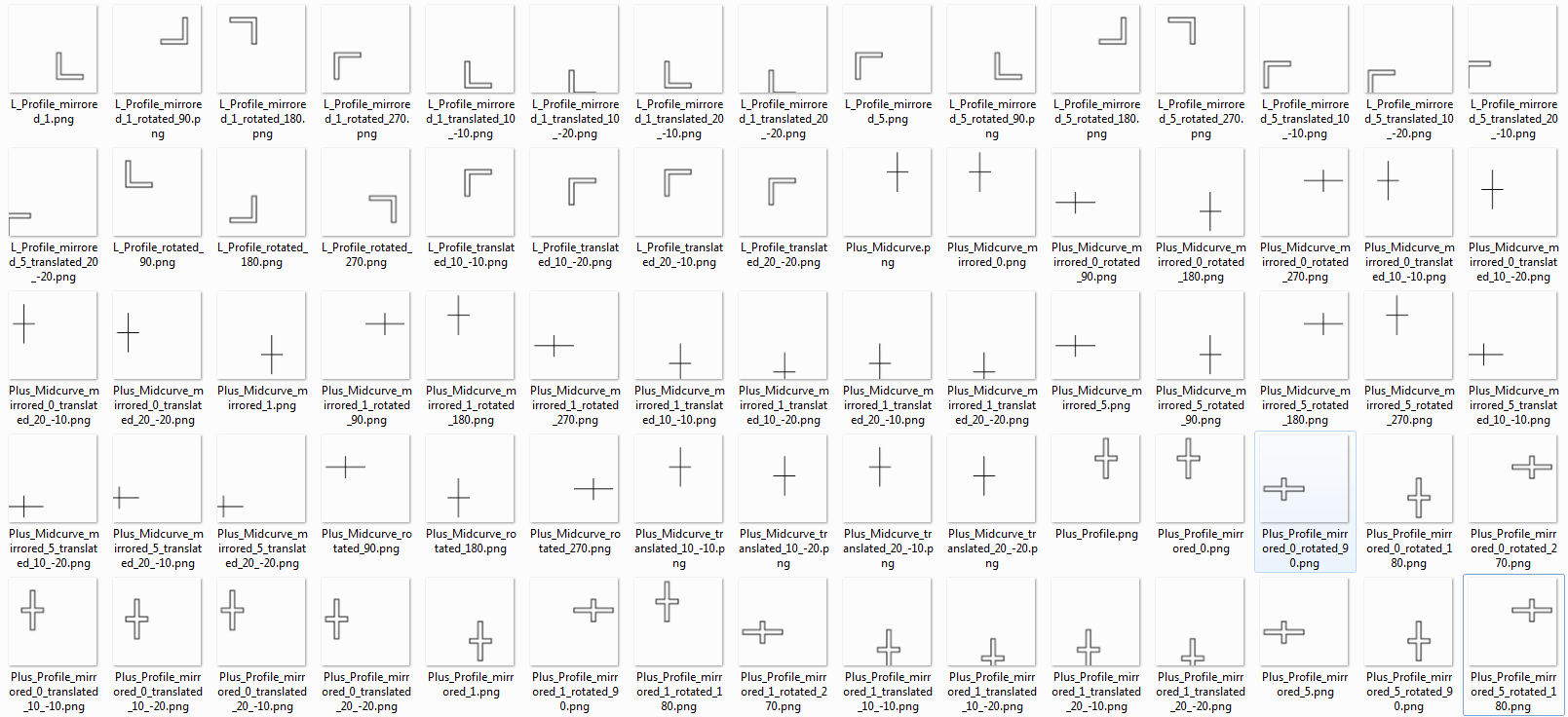


{images/midcurve\_encoder\_decoder}

\captionof{figure}{Encoder-Decoder Architecture}

\label{fig\_endecoder}

Input and output geometries are rasterized into 100x100 size images. Transformations like translation, rotation and mirroring are applied to create diversity in the samples. MidcurveNN being a Supervised Learning approach, both input thin-polygons and corresponding output midcurve polylines are transformed simultaneously. Figure \cite{fig\_training} shows some samples. This is training data.



\includegraphics[width=\linewidth]{images/training\_data}

\captionof{figure}{Training Data: Inputs (thin polygons) and outputs (midcurves)}

\label{fig\_training}

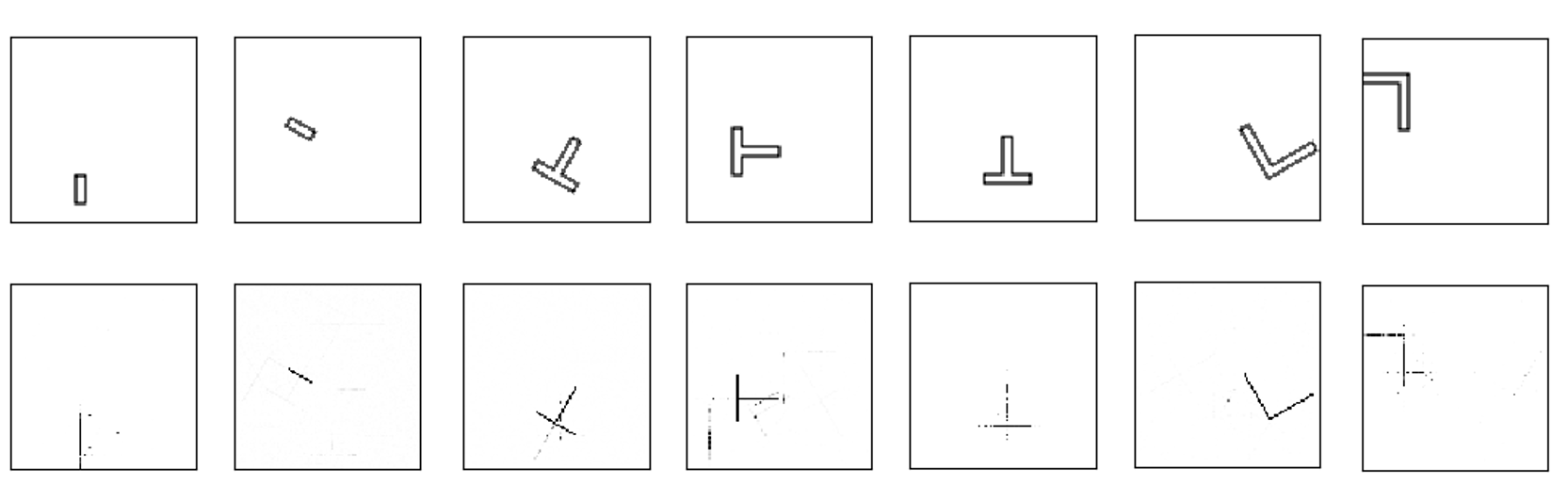
MidcurveNN encoder-decoder has been implemented in Python programming with Keras library \cite{autoenkeras}. Encoder takes input image of size $100 \times 100 = 10000$, then comes Dense layer with size $100$ to form the encoded vector. Decoder takes encoded vector as input, then with a Dense layer expands back to $100 \times 100 = 10000$ size of the output image. Relu activation is used for Encoder whereas Sigmoid for the decoder. AdaDelta optimizer with binary cross entropy as loss function is used to compute the losses. Table \ref{tbl\_loss} shows loss across number of epochs.

|  |  |  |
| --- | --- | --- |
| Epochs | Training Loss | Validation Loss |
| 50 | -17.6354 | -8.3223 |
| 200 | -16.8878 | -7.7672 |

\captionof{table}{Improvement in performance with epochs}

\label{tbl\_loss}

Some of the results are shown in Figure \ref{fig\_results}. Inputs are at the top and output midcurve at the bottom.



{images/midcurvenn\_results}

\captionof{figure}{Predicted Data: Inputs (thin polygons) and outputs (midcurves)}

\label{fig\_results}

Shape on the top is the input thin polygon whereas the corresponding shape at the bottom is the predicted midcurve. It can be clearly seen that the network is able to localize the shape and learn the dimension reduction function reasonably well. It is still not perfect or usable, as some stray points are still being wrongly classified as the part of output midcurve. A better network model and/or post processing is needed to make output midcurve practically usable.

# Conclusions and Future Work

Traditional methods of computing midcurves are predominantly rules-based and thus, have limitation of not developing a generic model which will accept any input shape. MidcurveNN, a novel Encoder-Decoder network attempts to build such a generic model. This paper demonstrates that simple single layer encoder and decoder network can learn the dimension reduction function reasonably well. Although more development is necessary in evolving a better neural architecture, the current results show positive potential.

Working on truly variable size inputs (thin polygon) and outputs (polyline) using dynamic graph of Encoder-Decoder network can be attempted in the future. More and highly diversified data can help improve the quality of the output. Developing a formal representation of polygonal shapes with variations such as open/closed, with-without loops, branched as a coherent sequence of points is also on the agenda.

# Bibliography

\bibitem{medial2010}

Kulkarni, Y. H.; Deshpande, S.

Medial Object Extraction - A State of the Art

International Conference on Advances in Mechanical Engineering, SVNIT, Surat, 2010.

\bibitem{dimred2017}

Kulkarni, Y. H.; Sahasrabudhe, A.D.; Kale, M.S

Dimension-reduction technique for polygons

International Journal of Computer Aided Engineering and Technology, Vol. 9, No. 1, 2017.

\bibitem{autoenkeras}

Chollet, F.

Building Autoencoders in Keras

https://blog.keras.io/building-autoencoders-in-keras.html, 2019.

4. Method

We formulate the problem as an image-to-image translation task of generating a grayscale image of the midcurve given it’s 2D profile as shown in [some figure]. Many solutions to such problems use an encoder-decoder network [references to encoder decoder image-to-image translation] where the input image is progressively downsampled by the first half of the network, until a bottleneck layer, at which point the process is reversed and the output image is generated by upsampling the information encoded in the bottleneck. In many cases [references to unet] where the input and output share low-level information, skip-connections are added to allow such information to circumvent the bottleneck layer following the shape of a “UNet”. Our model is a UNet where we add skip connections between each layer i and layer n-i (n is the total number of layers). Each skip connection concatenates the channels at layer i with those at layer n-i. [The encoder acts as a feature extractor and only uses convolution layers allowing for a variable sized input.]

Instead of giving the original hollow 2D profile as an input to the model, we flood-fill the hollow profile image to generate a filled profile image indicating the Region Of Interest (ROI) and use it as an input to our model. We also generate a distance transform matrix for the filled profile image by calculating the distance to the closest white pixel for each black pixel in the image. This matrix is scaled to the range of [0,1] using MinMax scaling technique and is supplied to the model as an additional input channel. For faster convergence, we also supply coordinate information in the form of two extra channels [coordconv]. Figure [] shows sample inputs and the corresponding output. We generate input-output data by randomly sampling the length, height, width and rotation from a fixed range for different shapes and therefore do not use any explicit data augmentation.

4.1 Model Architecture

We adapt our model architecture from the generator in [pix2pix]. The structure is summarized in [some table]. The encoder uses eight modules of the form Convolution-BatchNorm-LeakyReLu whereas the decoder uses seven mirrored modules of the form TransposedConvolution-BatchNorm-LeakyReLu-Dropout and one TransposedConvolution-Sigmoid module at the end so that the output falls in the grayscale range of [0,1]. The slope for all LeakyReLu activations is 0.2, and the dropout rate is 50%. Zero-padding is used for feature maps in all convolution layers to ensure that the image does not shrink. Convolutions in the encoder downsample by a factor of 2, whereas in the decoder they upsample by a factor of 2. First four convolution layers in the encoder use a kernel size of 5x5 and the rest use a kernel size of 4x4. The transposed convolution layers used in the decoder also use a kernel size of 4x4.

4.2 Dataset and Training

**Dataset.** Figure [] shows the [number] shapes used in this work. To assess generalizability, we use x number of shapes for training and the remaining y for testing purposes. We use OpenCV functions for line drawing, flood fill, and distance transform to automatically generate 500 images for each shape by randomly sampling the length, height, thickness, and rotation from a fixed range. Although the model can accept variable sized inputs, we create a standard dataset with images of size 256x256.

**Training Details.** We treat this problem as a multilabel classification problem, and thus train our model using weighted binary crossentropy loss with beta=0.1:

[WBCE Equation]

To learn the model parameters, we use backpropagation with Adam[] optimizer with an initial learning rate of 0.01. To reduce the learning rate over time, we use the following update rule:

lr = initial\_lr \* 1.0/(1.0 + decay\*iterations)

The value of decay is set to 0.001.

[Yogesh: Placeholder from some draft paragraphs]

# Introduction

Lower dimensional representations of shape are useful in various applications such as shape matching, transformations, analysis, etc. Instead of matching two-shape areas, it is computationally beneficial to match their corresponding skeleton shapes. Primary property of the lower dimensional entity is that it should morphologically similar to the parent higher dimensional shape. The process of computing the lower dimensional entity is called as Dimension Reduction.

Dimension reduction is used in pattern recognition, approximation, matching, detecting collisions, shape transfer, etc. These operations, instead of being performed on the original higher dimensional shapes are actually performed on their lower dimensional counterparts and once the operation is over, the original shape is computed back.

Two types of Dimension Reduction processes are widely used, one, from Solids (parametrically 3D, having volume) to surfaces (parametrically 2D, having area), and second, from Surfaces to curves (parametrically 1D, having length).

This paper deals with the latter, ie dimension reduction from Area to Curve, typically known as skeleton.

Various methods have been employed to compute the skeleton, such as Medial Axis Transform (MAT), Chordal Axis Transform (CAT), Pairing, Thinning etc. Figure \ref{fig\_medialmethods} shows some of these. More detailed analysis can be found in the survey paper \cite{medial2010}

{images/MedialMethodsOnlyShort}

\captionof{figure}{Medial Object computation methods}

\label{fig\_medialmethods}

Generically this paper will refer the parent shape with area, as a profile and the lower dimensional, representative curve, as midcurve.

# Related Work

Research on computation of midcurve is going on for decades. One of the early attempts was by theoretical biologist Harry Blum who used grass firing algorithm to compute the midcurve. Survey papers such as [5], [6] and [7] presented details on various approaches used so far.

Most of the approaches have been algorithmic, ie rule-based. Either mathematical transformation or heuristic steps are employed to compute the midcurve.

Attempts to use Machine Learning for such geometric transformations are not reported in literature widely. The principal problem is the ability to represent shapes as number. Meaning a faithful representation of shapes as vectors (Shape2Vec) would be a must to use Machine or Deep Learning. In the absence of any such popular method, the geometric problem is often transformed into image processing problem. In the image domain, representation and machine/deep learning techniques are widely available.

This paper deals with using deep learning (neural networks) for computing midcurve in the image form.

# References

[5] Attali D., Boissonnat J. D., and Edelsbrunner H., “Stability and computation of the medial

axis a state-of-the-art report,” Mathematical Foundations of Scientiﬁc Visualization, Com-

puter Graphics, and Massive Data Exploration, 2004.

[6] Louisa Lam, Seong-Whan Lee, and Chiang Y. Suen, “Thinning methodologies—a compre-

hensive Survey,” IEEE-PAMI, vol. 14, no. 9, pp. 869–884, Sept. 1992.

[7] Yogesh Kulkarni and Shailesh Deshpande, “Medial object extraction - a state of the art,”

in Proc. of the 3rd International Conference on Advances in Mechanical Engineering, S.V.

National Institute of Technology, Surat , Gujarat, India, January 4-6, 2010.