VECTOR FIELDS FOR INSTANCE SEGMENTATION

Ryan Peters
Computer Science and Mathematics Undergraduate
University of Minnesota
Minneapolis, MN
RyanIRL@icloud.com

August 31, 2022

ABSTRACT

Despite significant progress, general instance segmentation of non-convex and irregularly shaped objects with high population density remains a traditionally challenging task. In this article, I present modeling vector fields for instance segmentation as a class of proposal-free instance segmentation methods. I show that these methods, which I denote vector field based methods, demonstrate accurate segmentation of irregularly shaped and non-convex objects, fast training time, and lightweight and trivial integration into pre-existing semantic segmentation architectures. On the improvement of vector field based methods, I present several improvements to the computation of ground truth vector fields that together introduces smoothness, solves the problem of poor segmentation of large objects, and increases the size of stable points during integration. Furthermore, I present a novel integration-aware loss function that aims to make training more end-to-end. With the proposed improvements, our models yield an average of 0.806 mAP IoU on a challenging dataset with objects of irregular morphology and high population density. When compared to the current state-of-the-art vector field based method, the proposed method yields 3.3x faster training time per epoch, 2.1x faster evaluation time, faster training convergence, and an average of 5.7% higher mAP IoU. On the accessibility of vector field based methods, I am providing an open source and general Python library for vector field based methods. The library is vector field agnostic and can be found here: `https://github.com/ryanirl/torchvf`.

1 Introduction

Vector fields have emerged as compelling representations of images that we can model to perform lightweight proposal-free instance segmentation. Vector field based methods generate high quality instance segmentations by representing the semantic segmentation as a set of initial-values to be integrated or displaced through a vector field. The vector field is then modeled such that under integration or displacement, the distance between semantic points of the same instance is minimized and the distance between semantic points of different instances is maximized. The instance segmentation can then be derived through clustering.

Using vector fields for instance segmentation in this context has several advantages. These include:

- Lightweight and trivial integration into pre-existing semantic segmentation architectures. In the experiments, I demonstrate that with the addition of a single convolutional block ($\sim$2000 parameter increase) we can convert pre-existing semantic segmentation architectures into high performing instance segmentation architectures.
- Accurate predictions of objects with arbitrary morphology and population density. In the experiments, I show that vector field based methods can achieve an average of 0.807 mAP IoU on challenging datasets.
- Fast training time. In the experiments, I demonstrate that vector field based methods with the proposed improvements demonstrate faster training convergence and 3.3x faster training time per epoch when compared to previous methods. In one of three subsets, I will also show that we can outperform the current state-of-the-art after five minutes of training from scratch (on a GTX 1070).

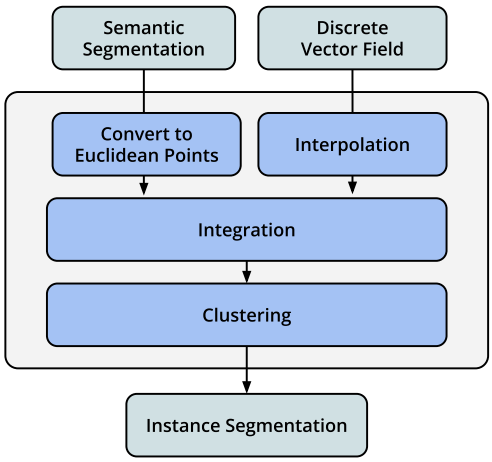

Figure 1: Diagram illustrating the steps taken to derive the instance segmentation from the semantic segmentation and vector field. First we convert the two-dimensional semantic segmentation into a set of initial-values. Next we convert our discretely sampled vector field into a continuous function over our image domain through interpolation. Then we integrate our set of initial-values through the interpolated vector field using numeric integration. This is followed by clustering to derive the instance segmentation.

Prior works on vector field based methods [CSWM21, SWMP21, SP22, NDBPVG19, YC21] have presented various loss functions, ways to model ground truth vector fields, and clustering schemes. Throughout my study of these works, I have noticed significant differences in notation and formulation. In fact, very few of these works formulated the task they were performing as modeling vector fields for instance segmentation. These differences in formulation present a barrier to consolidation. One goal of this paper is to consolidate these works and present modeling vector fields for instance segmentation as its own class of proposal-free instance segmentation methods. On the consolidation of these works, Section 2 lays a general mathematical foundation for vector field based methods. Section three then serves as both a review of prior works and to reformulate them under the shared context presented in Section 2.

Furthermore, the code provided by the reviewed prior works is rather limited in scope, supplying code that is hard to generalize and only applicable to their proposed vector fields. As a solution, I provide an open source and general library for vector field based methods. Functionally, this library aims to automate ground truth vector field computation and the postprocessing steps required to derive the instance segmentation from the semantic segmentation and vector field. This library is vector field agnostic and thus compatible with any vector field that uses the proposed method to derive the instance segmentation. Therefore, it should be partially (or possibly fully) compatible with the prior works reviewed in Section 3. Details can be found in Appendix A.

On the improvement of prior works, in Section 4 I propose several improvements to the computation of ground truth vector fields that introduce smoothness, solve the problem of poor segmentation of large objects (a known limitation to current techniques), and increase the size of stable points during integration. In Section 5, I present a novel integration-aware loss function, denoted $\mathcal{L}_{\text{IVP}}$, that aims to make training more end-to-end. This is done by allowing the loss function to optimize the trajectories of initial-values during integration. When compared to the current state-of-the-art vector field based method, these improvements yield an average of 5.7% higher mAP IoU performance on the experimented dataset, 3.3x faster training time per epoch, 2.1x faster eval time, and faster training convergence.

To summarize, my contributions are as follows:

- I provide a general framework, both mathematically and practically, for vector field based methods.
- I propose several improvements to the current state-of-the-art vector field based method. These include improvements to the computation of ground truth vector fields and a novel integration-aware loss function.
- I provide a general and vector field agnostic library for vector field based methods. Functionally, this library automates the postprocessing steps required to derive the instance segmentation given the semantic segmentation and vector field. The homepage can be found here: `https://github.com/ryanirl/torchvf`.

2 Preliminaries: A General Framework

Before a review of prior works, let us lay a mathematical foundation for vector field based methods by formally exploring how one would derive the instance segmentation of a two-dimensional image given the semantic segmentation and vector field.

An n-dimensional vector field defined over a two-dimensional space is a function $F(x, y)$ that maps each point in the space to a n-dimensional vector. In this article, we will focus on the two-dimensional vector field:

$$F : \mathbb{R}^2 \to \mathbb{R}^2, \quad F(x, y) = \langle P(x, y), Q(x, y) \rangle$$

Where $P(x, y)$ and $Q(x, y)$ are scalar fields $P, Q : \mathbb{R}^2 \to \mathbb{R}$ of F.

To derive the instance segmentation we let each non-zero (non-background) element of the semantic segmentation $\mathcal{S} \in \mathbb{R}^{H \times W}$ represent a two-dimensional coordinate (x_i, y_i) such that it's location in the plane indexes $\mathcal{S}$. That is, the point (x_i, y_i) represents the semantic segmentation element $\mathcal{S}_{y_i x_i}$. This allows us to treat $\mathcal{S}$ as a set of initial-values $\mathcal{V}_0 = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$ to integrate through F. Then we model F such that when $\mathcal{V}_0$ is integrated or displaced through F, the distance between semantic points of the same instance is minimized and the distance between semantic points of different instances is maximized. The instance segmentation can then be derived through clustering.

To understand what is meant by *integration through the vector field* consider the relationship between F and the two-dimensional, first order, autonomous system of ordinary differential equations (ODEs):

$$x' = f_x(x, y)$$
$$y' = f_y(x, y)$$

Here, the vector field is the function $F(x, y) = \langle f_x(x, y), f_y(x, y) \rangle$. For an intuitive interpretation of this system, we can consider the values $f_x(x_i, y_i)$ and $f_y(x_i, y_i)$ as the individual x and y flows or velocity through the point (x_i, y_i). Because this is an autonomous system, neither f_x nor f_y directly depend on the independent variable t. That is, for any point (x_i, y_i), the flow through that point remains constant as t changes. The solution to this ODE takes the form:

$$P(t) = (x(t), y(t)), \quad P(0) = (x_0, y_0)$$

Such that, given the initial value (x_0, y_0), we parameterize the function $P(t)$ with the initial conditions $t = 0$, $x(0) = x_0$, and $y(0) = y_0$. Then $P(t)$ represents the position of the point (x_0, y_0) after time t. Given only the individual scalar fields, the exact solution for the position of the initial value (x_0, y_0) after time t is given as:

$$x(t) = x_0 + \int_{t_0}^{t} f_x(x(\tau), y(\tau))\, d\tau$$
$$y(t) = y_0 + \int_{t_0}^{t} f_y(x(\tau), y(\tau))\, d\tau$$

Although, in practice we model a *discretely sampled* vector field with no known analytical form. This raises two problems. First, during integration we need a continuous vector field. The solution: Interpolation is used to *fill the gaps* and convert our discretely sampled vector field into a continuous function over our image domain. Next, we cannot evaluate the integral as our vector field often has no known analytical form. The solution: we use Euler's method to numerically approximate the integration equation. The updates for this two-dimensional system under Euler's method are:

$$x_n = x_{n-1} + h f_x(x_{n-1}, y_{n-1})$$
$$y_n = y_{n-1} + h f_y(x_{n-1}, y_{n-1})$$
$$t_n = t_{n-1} + h$$

Where h is the step size and n is the total steps taken. In the limit, as h approaches 0 with the number of steps taken n being $n = \frac{t}{h}$, the above update rule becomes asymptotically equivalent to the integration equation. Notice two things: Because our vector field is autonomous, in practice the explicit computation of t is not necessary. Additionally, when we let $F(x, y) = \langle f_x(x, y), f_y(x, y) \rangle$ we can vectorize the above form to the following:

$$(x_{i+1}, y_{i+1}) = (x_i, y_i) + hF(x_i, y_i) \tag{1}$$

Thus, integration through the vector field can intuitively be thought of as translating a point at location (x_i, y_i) in the direction of the vector $F(x_i, y_i)$ times some small step size h. It should also be noted that *displacement through the vector field* corresponds to $(x_1, y_1) = (x_0, y_0) + F(x_i, y_i)$. Equation 1 is the final form we will focus on. That is, we aim to model a vector field $F(x, y)$ that under integration minimizes the distance between semantic points of the same instance and maximizes the distance between semantic points of different instances.

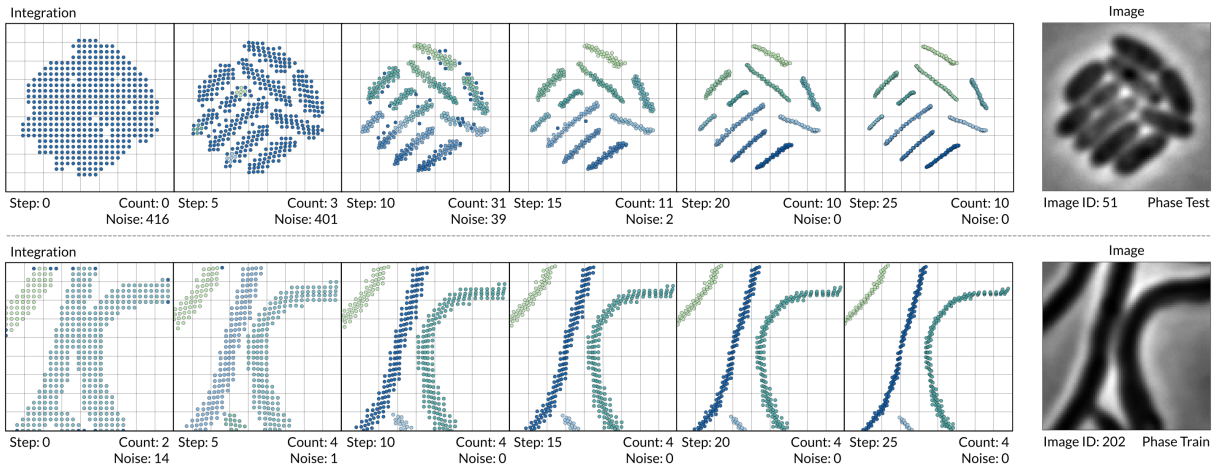

Figure 2: Real examples illustrating *integration through the vector field*. Both images are taken from the bacterial phase contrast subset of the BPCIS dataset (see Experiments or Appendix B). Step 0 corresponds to the euclidean representation of the semantic segmentation. The presented semantic segmentation and vector field are not ground truths, but were predicted by an H1 model (see Section 6.2). For demonstration purposes, the semantic points were sampled at a stride of three rather than showcasing all the semantic points. Furthermore, DBSCAN clustering was applied at each step for demonstration purposes. The results are presented below each image. A step size of 0.25 was used.

3 Review of State-of-the-Art

As previously described, we aim to model a vector field that under integration minimizes the distance between semantic points of the same instance and maximizes the distance between semantic points of separate instances. Prior works [SWMP21, SP22, NDBPVG19, YC21] have utilized the fact that convex objects can be uniquely identified by their centroids and propose we perform *regression to the instance centroid*. That is, they propose we model a vector field that through integration clusters semantic points along their instance centroids. Under convex settings, these methods have demonstrated incredible performance and efficiency at deriving instance segmentation masks, though fail to generalize to irregularly shaped and non-convex objects due to object centroids not being well-defined.

On the improvement of centroid-based methods, Omnipose [CSWM21] argued that because centroids are not well-defined for irregularly shaped and non-convex objects we should model a morphology independent vector field. On the computation of such a vector field, they propose we let the vector field be the gradient of the euclidean distance transform. Rather than regression to the instance centroid, this new paradigm is equivalent to *regression to the instance skeleton*. In detail, Omnipose proposes a two step process for the computation of ground truth vector fields and an additional two step process for postprocessing of predicted vector fields before integration. To compute the ground truth vector fields, they propose for each mask:

1. Compute the euclidean distance transform (EDT).
2. Compute the X and Y gradients of the EDT using finite differences.

Before integration they additionally normalize non-zero vectors to unit length and then rescale the predicted vector field by the divergence. During integration they also find that semantic points fragment into several pieces along the skeleton causing over-segmentation. To solve this, they incorporate a suppression factor of $(t+1)^{-1}$. Numeric integration with this suppression factor becomes:

$$\begin{aligned}(x_n, y_n) &= (x_{n-1}, y_{n-1}) + hF(x_{n-1}, y_{n-1})/(t_{n-1}+1)\\ t_n &= t_{n-1} + h\end{aligned}$$

Because the gradient of the EDT points towards the *inside* of an object, objects need not be convex. In fact, no assumption is made about the underlying shape of objects, unlike centroid based-methods which fail when centroids lie on the outside of an object.

With these improvements, Omnipose has demonstrated unheard-of results for instance segmentation of irregularly shaped and non-convex objects under an arbitrary density setting. Additionally, to the best of my knowledge Omnipose

is the current state-of-the-art vector field based method prior to this work. Though, in its current state I argue that Omnipose suffers from two problems:

- Non-smooth ground truth vector fields.
- Poor segmentation of objects with large or irregularly small diameter.

Non-smooth ground truth vector fields. In general, numerically approximated functions add some level of numeric error to solutions. When significant, these errors become visible as artifacts. To compute ground truth vector fields, Omnipose couples the numeric approximation of the EDT with the numeric approximation of its gradient using finite differences. Therefore, there are two bottlenecks where artifacts can be introduced (see Figure 3). Furthermore, artifacts that appear during the earlier computation of the EDT will be magnified through the numeric approximation of the gradient using finite differences.

During the computation of the EDT, errors are usually introduced by discretization and low resolution masks. Furthermore, due to the inwards flow of the EDT, small inconsistencies near the border of masks are magnified and propagated inside the mask leading to the accumulation of larger and larger artifacts. Omnipose recognized this problem and introduced smoothness by designing an improved fast iterative method for approximating the EDT that utilized ordinal sampling. They hoped that improvements in this step would translate through the approximation of the gradient. I argue that ordinal sampling is not enough and that the flows proposed by Omnipose still suffer from significant artifacts and non-smoothness.

These artifacts pose a significant problem to both stability during integration and to training. During integration, these artifacts often cause semantic points to cluster into several fragments along instance skeletons. This leads to over-segmentation during the clustering step. During training, these artifacts introduce uncertainty and can bottleneck the convergence of training and performance of models after the fact. In Section 4, I will propose an improvement to the computation of ground truth vector fields that aims to solve smoothing.

Poor segmentation of objects with large or irregularly small diameter. One of the known limitations of Omnipose is the accurate segmentation of objects with large or irregularly small diameter. To understand why Omnipose demonstrates poor segmentation of large objects, consider that convolutional neural networks are traditionally good at modeling local interactions. Inversely, they are traditionally bad at modeling long range or global interactions. The accurate approximation of the EDT and its gradient requires us to model the distance from any given pixel to the nearest border. It becomes extremely hard for models to accurately predict these distances when objects have a maximum radius that is considered long range or even possibly outside the receptive field of convolutional neural networks. I claim that this is the reason Omnipose demonstrates poor segmentation of objects with large diameter.

On the improvement of Omnipose, in Section 4 I will propose a small modification to the computation of ground truth vector fields that I claim makes the prediction of large objects trivial. This improvement will remove the need for modeling long range interaction while preserving the use of the EDT.

4 On the Improvement of Ground Truth Vector Fields

As previously discussed, I claim that Omnipose is the current state-of-the-art vector field based method. Furthermore, I argued that Omnipose suffers from two problems: *non-smooth vector fields* and *poor segmentation of large objects*. To solve both problems, I propose the following updates to the computation of ground truth vector fields:

1. Compute the *truncated* signed distance field (SDF).
2. Compute the X and Y gradients of the SDF using a large gaussian-smoothed finite difference kernel.

4.1 Gaussian-Smoothed Finite Differences

To introduce smoothing and reduce artifacts, I propose we compute the gradient of the EDT using a large gaussian-smoothed finite difference kernel. When computing the gradient at a point, the large finite difference kernel reduces artifacts by considering the average rate of change over a larger neighborhood than the native finite difference kernel (sobel kernel) would. Therefore, any local artifacts produced by the numeric computation of the EDT will be averaged out (rather than magnified) during the numeric gradient computation.

One problem does arise, a large finite difference kernel gives the same weight to stricly neighboring points and points near the edge of the kernel. To solve this weighting problem, I propose we compute a gaussian kernel of the same size

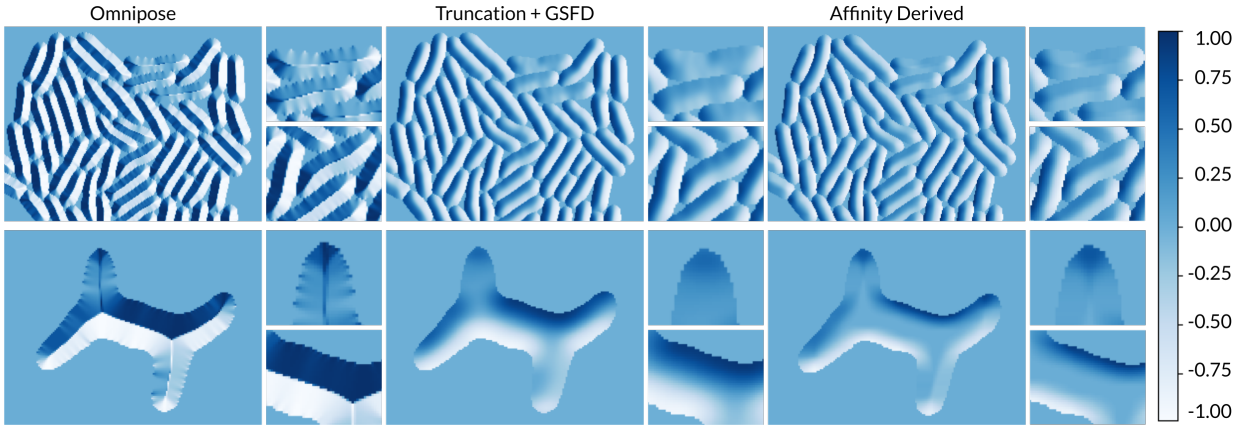

Figure 3: The top and bottom row demonstrate examples of the ground truth f_x and f_y scalar field respectively for the Omniposed, the proposed, and the affinity derived vector field. The proposed and affinity derived vector fields are significantly smoother than the Omnipose vector field which contains a large number of artifacts. Furthermore, notice that vector magnitudes are smaller near the center of large objects resulting in larger stable points during integration.

and perform the element-wise multiplication with the finite difference kernel. Points near the center of the kernel will be given more weight than points near the edge. I will refer to this kernel as the gaussian-smoothed finite difference kernel.

I additionally propose we let the vector field be the gradient of the SDF rather than the EDT. This is because the EDT defines values outside of an object as zero. Therefore, convolving the gaussian-smoothed finite difference kernel with points near object borders will yield suboptimal gradients. The SDF will insure that elements outside of an object are also given the distance to the nearest border (just negative values). This gives us more stable gradients near object borders when using a large kernel.

4.2 Truncated Signed Distance Field

To solve the problem of poor segmentation of objects with large diameters, I propose we truncate the SDF. Reason being, truncation imposes an upperbound on the range of interaction that models need to consider. Truncation also introduces low-magnitude vectors near the center of objects due to the gradient of a flat (or truncated) surface being zero. This is beneficial because low-magnitude centers create larger stable points during integration, which leads to increased stability during integration. Furthermore, applying truncation in combination with the large gaussian-smoothed finite difference kernel (as proposed above) gives a smooth transition from vectors of high magnitude to vectors of low magnitude (see Figure 4).

Now let us consider why truncation imposes an upperbound on the range of interaction that models need to consider. Given an arbitrary pixel p inside an arbitrary object with maximum radius R, consider the minimum information needed for a model to predict the value of the truncated SDF of p with truncation value α. Models need only consider neighboring pixels of p within distance α. If all neighboring pixels within distance α are of the same instance, then p will be assigned a value of α. If not all neighboring pixels are of the same instance, then p will be given the value of the distance to the nearest pixel not a part of the same instance as p. This value will be at most α. Therefore, with a truncation value α models only need to consider pixels within distance α of p. Without truncation, the maximum distance models would need to consider away from p is R. When R is large, this requires models to consider long range interactions (a known limitation of convolutional neural networks).

Affinity derived vector field. As discussed, truncation imposes an upperbound on the distance from a pixel p that models need to look at to predict the value of the SDF for p. This has an interesting relationship to affinity. If we knew the relationship between p and the affinity to its neighboring pixels, we could model an *affinity derived vector field* by applying finite differences directly on these affinity relationships (or kernels).

In my testing, the affinity derived vector field resulted in similar performance to truncation + gaussian-smoothed finite difference kernel. Though admittedly, the purpose of this article isn't to describe the many ways we can model vector fields, but provide a general framework for vector field based methods. I do provide the code for computing the affinity derived vector field and can supply the models trained on these vector fields on request, but I will not be comparing the quantitative performance of the affinity-derived models in this article.

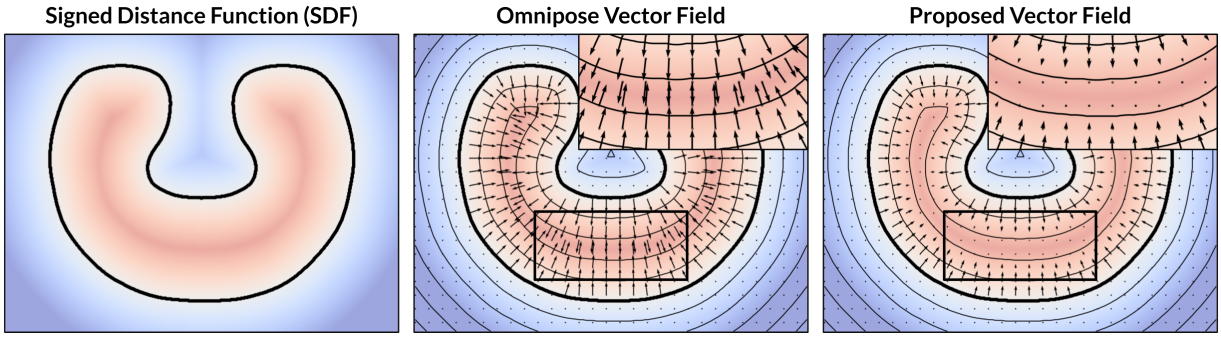

Figure 4: **Left.** Diagram illustrating the signed distance function. **Middle.** The Omnipose vector field. Notice that all vectors are of unit length, which satisfies the eikonal equation. **Right.** The proposed vector field. Notice that near the center the vectors are of smaller magnitude. This violates the eikonal equation though leads to larger stable points and increased stability during integration.

4.3 Simplifications Resulting from the Proposed Improvements

The improvements above were formulated to introduce smoothness and solve the problem of poor segmentation of large objects. Though, these improvements also make three simplifications possible. We no longer need to normalize predicted vector fields to unit length, rescale the normalized vector fields by the divergence, and utilize the suppression factor during integration.

5 Integration-Aware Loss Function

When choosing a loss function to minimize during training for vector field based methods, one may think to use the mean squared error (MSE), L2-Norm loss, or various other functions that directly optimize explicit vector properties. Though none of these loss functions are end-to-end. That is, they incorporate no knowledge of integration or clustering. On the development of a general and end-to-end loss function for vector field based methods, I present an integration-aware loss function. In one sentence the proposed loss function, which I denote $\mathcal{L}_{\text{IVP}}$, aims to minimize the difference between trajectories of initial-values integrated through both the ground truth and predicted vector fields. This implicitly optimizes vector field predictions and makes training more end-to-end by incorporating knowledge of integration during training.

Formulating the proposed loss function mathematically, consider the loss given a single initial-value (x_i, y_i). During training, we are given the ground truth vector field $F(x, y)$ and the predicted vector field $\hat{F}(x, y)$. First we integrate the initial-value through both the predicted and ground truth vector fields. Integration through the ground truth vector field is given as:

$$\begin{aligned}(x_1, y_1) &= (x_0, y_0) + hF(x_0, y_0)\\ (x_2, y_2) &= (x_1, y_1) + hF(x_1, y_1)\\ &\vdots\\ (x_n, y_n) &= (x_{n-1}, y_{n-1}) + hF(x_{n-1}, y_{n-1})\end{aligned}$$

Once we repeat this process for the predicted vector field we will be given two solution sets (discretized trajectories):

$$\mathcal{P} = \begin{bmatrix}(x_1, y_1)\\ (x_2, y_2)\\ \vdots\\ (x_n, y_n)\end{bmatrix} \qquad \hat{\mathcal{P}} = \begin{bmatrix}(\hat{x}_1, \hat{y}_1)\\ (\hat{x}_2, \hat{y}_2)\\ \vdots\\ (\hat{x}_n, \hat{y}_n)\end{bmatrix}$$

Here n represents the total number of steps we take during integration. The proposed loss function aims to minimize the distance between the discretized trajectories of points integrated through the ground truth vector field and points integrated through the predicted vector field. This translates to:

$$\mathcal{L}_{\text{IVP}} = \frac{1}{n}\sum_{i=0}^{n}(\hat{P}_i - P_i)^2$$

In practice, we do not compute this for a single point (x_i, y_i) but batch-vectorize it for all points (semantic or not). Since I have written both the integration and interpolation functions in native PyTorch [PGM+19], backpropagation works trivially utilizing PyTorch's reverse-mode automatic differentiation capabilities [PGC+17].

6 Method

This section will detail my implementation of the proposed method as used in the experiments. This includes parameter values, model architectures, training information, and libraries used. All parameter values presented are specific to the dataset used in the experiments (see Experiments or Appendix B).

6.1 Computation of Ground Truth Vector Fields

For the computation of ground truth vector fields, I used the method proposed in Section 3 with a truncation value of 10 and a gaussian-smoothed finite difference kernel size of 11. To compute the SDF, I used Seung Lab's euclidean distance transform library. This is a CPU implementation, though I found it fast enough since the vector field computation only needs to be done once. To compute the gradient of the SDF, I convolved the gaussian-smoothed finite difference kernel with the SDF via PyTorch's [PGM+19] conv2d function (see the supplied code).

6.2 Modeling

As discussed, one of the many advantages of vector field based methods is the lightweight and trivial integration into pre-existing semantic segmentation architectures. The base architecture I experimented with is a vanilla 1.9M parameter U-Net [RFB15] with ResNet-like [HZRS15] encoder blocks. There are two trivial ways (that I see) to extend pre-existing semantic segmentation architectures to incorporate the modeling of vector fields. The first proposed extension, denoted H1, consists of the addition of a single convolutional block *after upsampling* (see Figure 5). The increase in parameter size when extending the base architecture to the proposed H1 architecture is ∼2000 parameters, an almost non-significant increase in model size. The second proposed extension, denoted H2, consists of the addition of a new decoder head to the latent block. The increase in parameter size under the H2 extension is ∼0.8M parameters, a rather large increase relative to the H1 architecture. All modeling is done in PyTorch.

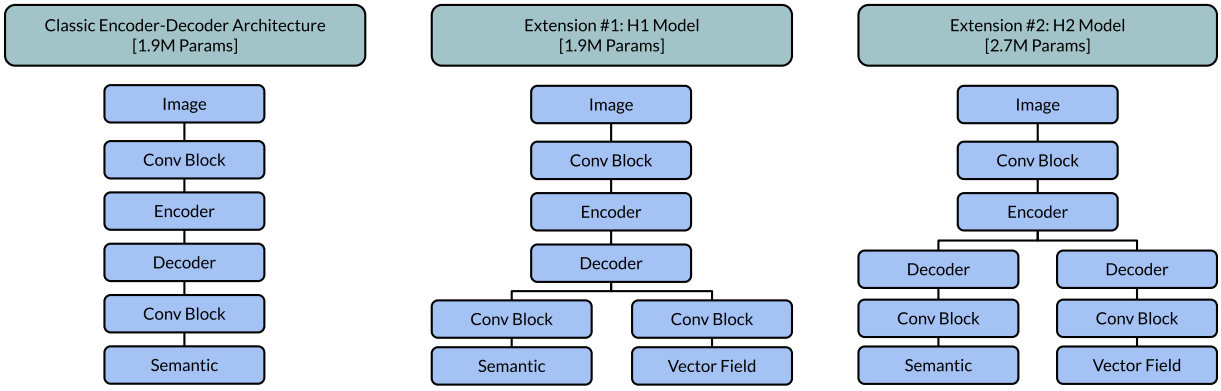

Figure 5: **Model Architecture.** Diagram illustrating the proposed H1 and H2 extensions of basic semantic segmentation architectures. The proposed H1 architecture consists of the addition of a single convolutional block after upsampling. This increases model size by ∼2000 parameters. The proposed H2 architecture consists of the addition of a new decoder and final convolutional block to the latent space. This increases model size by ∼0.8M parameters.

6.3 Training

Unless otherwise stated, all H1 and H2 models were trained for 1000 epochs using the Adam optimizer with a learning rate of 0.0002 and a batch size of 2. Training was done on a single GTX 1070. The following loss function was minimized:

$$\mathcal{L}_{\text{final}} = \alpha_0 \mathcal{L}_{\text{IVP}} + \alpha_1 \mathcal{L}_{\text{MSE}} + \alpha_2 \mathcal{L}_{\text{Tversky}}$$

The Tversky loss function optimizes the semantic head and the MSE and IVP loss functions optimize the vector field head. The following transforms were used during training: Images were divided by 255 to normalize inputs between 0 and 1, random horizontal and vertical flip were applied with probabilities 0.25, and a random crop of size 256×256 was applied to be coupled with the image tiling scheme explained in Section 6.5.

Training Quirks It should be noted that due to the directional nature of vector fields not all training transforms natively work. For example, consider the horizontal flip transform. When you horizontally flip the vector field you are also changing the direction of the f_x scalar field. To correct this you would need to invert the f_x sign (i.e. multiply by negative one). Same goes for the vertical flip, though you negate f_y rather than f_x. There are various other transforms that cause similar problems, so it's best to be cautious when applying any transforms to the vector fields.

6.4 Postprocessing

At the heart of the proposed methods is the postprocessing steps required to derive the instance segmentation from the semantic segmentation and vector field. Postprocessing can broadly be split into three steps: Interpolation, numeric integration, and clustering. I provide an open source library (see Appendix A) to automate these postprocessing steps in hopes of making vector field based methods more accessible.

Throughout the experiments, I used: Bilinear interpolation, Euler's method for for numeric integration with 25 steps and a step size of 0.1, and DBSCAN [EKSX96] clustering with an eps value ranging between 2.1 to 2.3 and a minimum sample value of 15. The interpolation and numeric integration functions are written in PyTorch. Therefore, they can utilize GPU acceleration and PyTorch's reverse mode automatic differentiation capabilities. I used Scikit-learn's [PVG+11] DBSCAN implementation. This can only be run on the CPU and tends to be the main speed bottleneck. More detailed parameter values can be found in the configs with the supplied code.

6.5 Image Tiling

I utilized image tiling due to limited computation and the varying image resolutions in the BPCIS dataset (see Appendix B). Consider we are given an image $M^{H \times W}$, tile size $T_H \times T_W$, and tile overlap O. The tiling stride S_H and S_W is then $T_H - O$ and $T_W - O$ respectively. Then the overflow (amount we need to adjust by) V_H and V_W for an image can be found via the following:

$$V_H = T_H + S_H \left\lceil \max(H - T_H,\ 0) \,/ S_H) \right\rceil - H$$
$$V_W = T_W + S_W \left\lceil \max(W - T_W, 0) / S_W) \right\rceil - W$$

The new image size we then need to conform with the given tile size and overlap is $\hat{H} = H + V_H$ and $\hat{W} = W + V_W$. To adjust image size, I tested resizing, padding, and a dynamic overlap approach. I found that the padding and dynamic overlap approach performed significantly better than resizing, with the dynamic overlap approach performing slightly better than padding. See the supplied code for further details.

7 Experiments

In Section 7.1, I compare the semantic and instance segmentation performance of the proposed methods versus Omnipose [CSWM21] and Cellpose [SWMP21, SP22]. In Section 7.2, I evaluate how the proposed improvements effected the convergence during training. In Section 7.3 and 7.4, I compare the training and evaluation speed respectively of the H1 model with the proposed updates versus the Omnipose model. In Section 7.5, I evaluate the performance of the H1 model on the task of cell counting. Additional experiments can be found in Appendix C. All H1 and H2 models were trained on a single GTX 1070. For reproducibility, none of the H1 or H2 models compared were cherry picked. All model weights, configs, and training and evaluation information will be provided and should be fully reproducible.

Dataset All experiments will be performed on the Bacterial Phase Contrast for Instance Segmentation (BPCIS) dataset [CSWM21]. This dataset contains images of a diverse set of densely packed cells with irregular and elongated morphology. The dataset is split into three subsets: The bacterial phase contrast, the bacterial fluorescence, and the worm subset. Furthermore, each subset is split into a testing and training set. Further details about the BPCIS dataset can be found in Appendix B.

Model	Params $\times 10^6 \downarrow$	Instance Mask				Semantic Mask		Eval Time (FPS) $\uparrow$
		$AP_{50} \uparrow$	$AP_{75} \uparrow$	$AP_{90} \uparrow$	mAP $\uparrow$	IoU $\uparrow$	F1 $\uparrow$	
Bacterial Phase Contrast								
H1 (ours)	1.9	0.930	0.885	0.764	0.860	0.966	0.983	5.5
H2 (ours)	2.7	0.934	0.891	0.760	0.862	0.965	0.982	4.8
Omnipose	6.6	0.942▲1%	0.906▲2%	0.780▲2%	0.876▲2%	0.962▼0%	0.980▼0%	2.8
Cellpose	6.6	0.554▼38%	0.509▼38%	0.345▼42%	0.469▼39%	0.904▼6%	0.940▼4%	3.9
Bacterial Fluorescence								
H1 (ours)	1.9	0.964	0.938	0.658	0.853	0.969	0.984	7.0
H2 (ours)	2.7	0.961	0.936	0.655	0.851	0.967	0.983	6.5
Omnipose	6.6	0.931▼3%	0.893▼4%	0.378▼28%	0.734▼12%	0.959▼1%	0.979▼1%	3.7
Cellpose	6.6	0.913▼5%	0.745▼19%	0.056▼60%	0.571▼28%	0.952▼2%	0.975▼1%	4.6
Worm								
H1 (ours)	1.9	0.876	0.793	0.446	0.705	0.910	0.952	12.9
H2 (ours)	2.7	0.877	0.803	0.446	0.709	0.910	0.952	10.3
Omnipose	6.6	0.813▼6%	0.725▼7%	0.375▼7%	0.638▼7%	0.890▼2%	0.941▼1%	5.6
Cellpose	6.6	0.868▼1%	0.700▼9%	0.292▼15%	0.620▼8%	0.887▼2%	0.940▼1%	7.0

Table 1: **Quantitative results on the BPCIS dataset.** Omnipose and Cellpose benchmarks are calculated from publically available pretrained models with the recommended hyperparameters. The $\uparrow$ represents that a higher score for the compared quantity is better, and vice-versa for the $\downarrow$. ▼10% represents that the compared method scores 10 percent worse relative to the H1 model, and vice-versa for ▲10% .

7.1 Segmentation Performance

Setup. In this section, I compare the instance and semantic segmentation performance between the proposed method and two other works: Omnipose and Cellpose. The instance segmentation is evaluated with the mAP IoU metric at thresholds 0.50, 0.75, and 0.90. The semantic segmentation is evaluated with the F1 and IoU score. The results are presented in Table 1.

Omnipose and Cellpose benchmarks are calculated from publically available pretrained models with the recommended hyperparameters. Furthermore, the presented benchmarks line up with the results presented in the Omnipose paper [CSWM21]. To evaluate the proposed method, I trained an individual H1 and H2 model on each subset in the BPCIS dataset. The provided code will contain the pretrained models, configs, and code to reproduce all presented benchmarks (including Cellpose and Omnipose). Unless stated otherwise, all comparisons done below will be in reference to the H1 model.

Results. Referring to Table 1, on average the proposed method yields 5.7% higher mAP IoU than Omnipose and 25.3% higher mAP IoU than Cellpose. When compared to Omnipose, the proposed method yields significantly higher mAP IoU on the bacterial fluorescence subset (11.9%) and the worm subset (6.7%), though slightly (1.6%) lower mAP IoU on the bacterial phase contrast subset.

On the semantic segmentation performance, the proposed method yields an average of 1.1% higher IoU and 0.6% higher F1 score versus Omnipose, and an average of 3.4% higher IoU and 2.1% higher F1 score versus Cellpose. Although it should be noted that Omnipose doesn't explicitly compute the semantic segmentation, but derives it by thresholding the predicted distance transform.

When comparing the H1 and H2 models, notice that the H1 model yields on average roughly the same results as the H2 model. This demonstrates that the addition of a single convolutional block ($\sim$2000 parameter increase) is enough to convert semantic segmentation architectures into high performing instance segmentation architectures via vector field based methods. This is opposed to the addition of a new decoder to the latent space ($\sim$0.8M parameter increase).

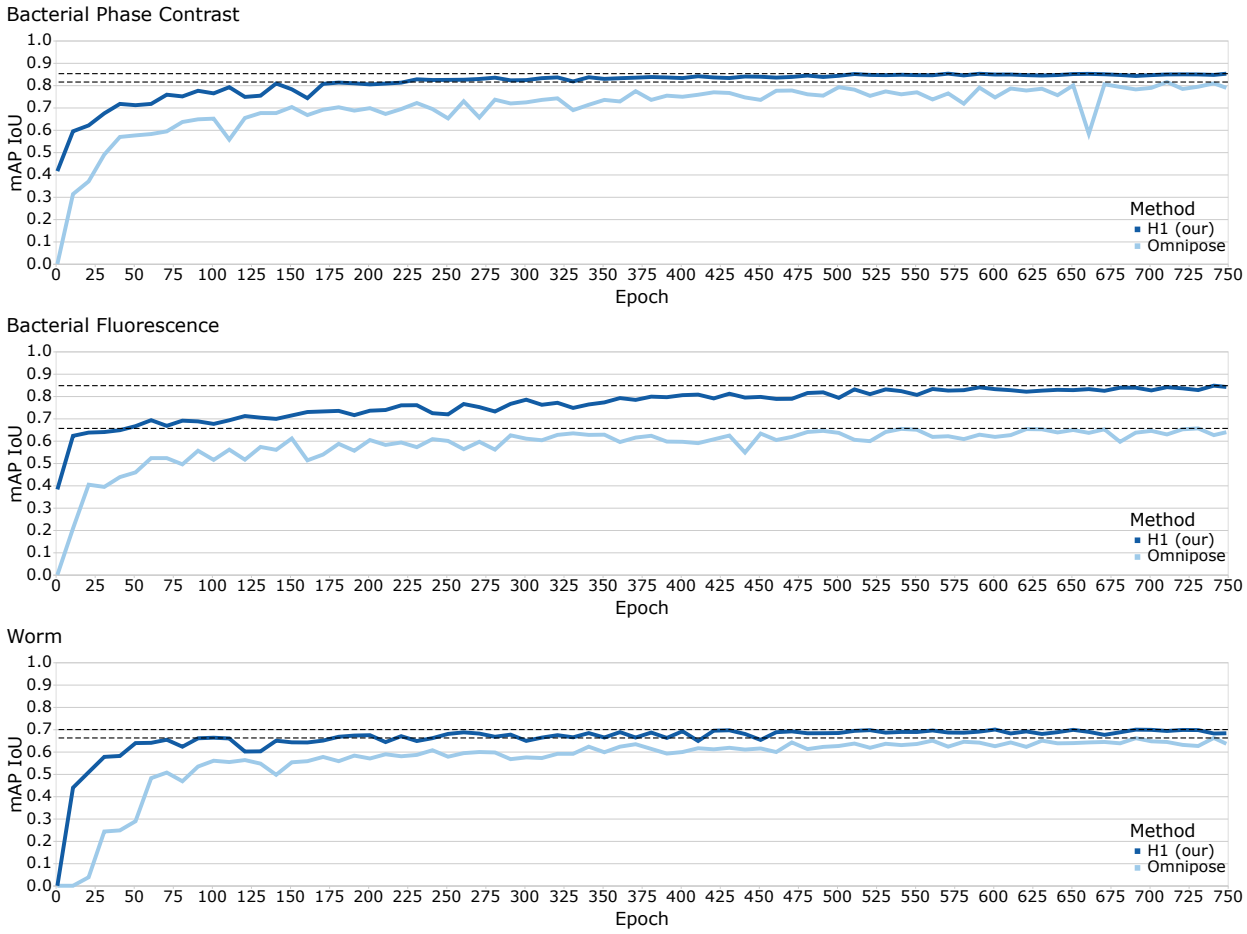

Figure 6: **Omnipose vs H1 Training Convergence.** For each of the three figures a separate Omnipose and H1 model was trained on the training set of one of the three subsets in the BPCIS dataset for a total of 750 epochs. Therefore, a total of six models were trained to generate all three tables. The mAP IoU was then evaluated on the testing set after every 10th epoch of training. The dotted line refers to the top score of the Omnipose and H1 model.

7.2 Training Convergence

Setup. To study the effects of the proposed updates on the training convergence, I trained an individual Omnipose and H1 model on each of the three BPCIS subsets. During training, I evaluated the mAP IoU on the testing set after every 10th epoch (including after the first epoch). In an attempt to minimize the differences between Omnipose and the proposed method, all models were trained for 750 epochs using the Adam optimizer with a learning rate of 0.0002 and batch size of 2. The final results can be seen in Figure 6. It should also be noted that these results do not accurately represent a benchmark of Omnipose as the pretrained models provided by Omnipose (used to evaluate their performance in Table 1) were trained for 4000 epochs.

Results. Referring to Figure 6, in all three subsets the proposed method converges significantly quicker during the first $\sim$70 epochs of training. In the long run, both methods seem to converge to roughly similar top scores on the bacterial phase contrast and worm subsets, though the proposed method yields significantly higher mAP IoU on the bacterial fluorescence subset. Further demonstrating the fast convergence speed, notice that at epoch 230 the proposed method surpasses the top score achieved by Omnipose on the bacterial phase contrast subset. Likewise at epoch 100 on the worm subset and epoch 50 on the bacterial fluorescence subset (Wow! That's sub 5 minutes of training).

Do make one more observation, the H1 model *seems* more stable during training and/or evaluation. Referring to the bacterial phase contrast results of Figure 6, notice that after epoch 170 the Omnipose results seem to flucuate quite a bit from epoch to epoch, whereas the H1 model doesn't flucuate nearly as much. Furthermore, at epoch 660, Omnipose has a large 21.7% drop in mAP IoU followed by a 22.2% rebound.

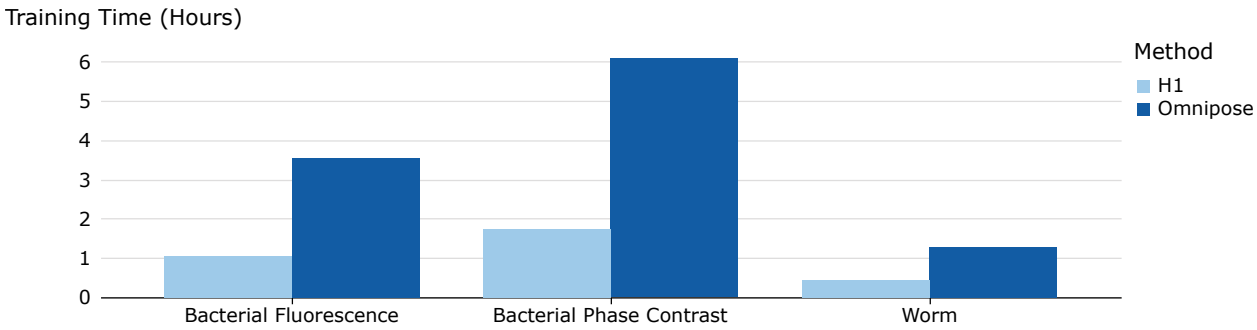

Figure 7: **H1 vs Omnipose Training Time.** This figure demonstrates how long it took to train each model in the *training convergence* experiment. Both the Omnipose and H1 models were trained for 750 epochs on a single GTX 1070 with a batch size of 2. On average the H1 model is 3.3x faster to train.

7.3 Training Speed

Setup. To compare the training speed of the proposed method to Omnipose, I considered the time it took to train each model for 750 epochs during the training convergence experiment. A graphical representation of the results can be seen in Figure 7.

Results. To train for 750 epochs on the bacterial phase contrast subset, the H1 model took 1.74 hours and the Omnipose model took 6.08 hours. Likewise on the bacterial fluorescence subset, the H1 model took 1.06 hours and the Omnipose model took 3.55 hours. Finally, on the worm subset, the H1 model took 0.44 hours and the Omnipose model took 1.27 hours. On average, the proposed method is 3.3x faster per epoch during training when compared to Omnipose.

Several factors play a role in the increased training speed of the proposed method. First, consider that my model is $\sim$3.5x smaller than the Omnipose model. Furthermore, I optimize a total of 3 loss functions as opposed to Omnipose which optimizes 8 different loss functions. Finally, the proposed model only predicts the vector field and semantic segmentation, whereas the Omnipose model predicts the vector field, distance field, and the boundary field.

7.4 Evaluation Speed (FPS)

Setup. The evaluation speed was calculated by computing the average FPS on the testing set of each BPCIS subset. Omnipose uses the same CPU DBSCAN clustering implementation as the proposed methods. Since the DBSCAN clustering step is the most time consuming postprocessing step, this should keep the implementation differences between Omnipose and the proposed method to a minimum when comparing the evaluation speed. One should note that the images in the bacterial phase contrast and fluorescence subset are highly variable in resolution (see Appendix B).

Results. Referring to the last column of Table 1, the H1 model yields on average 2.1x higher FPS when compared to Omnipose and 1.6x higher FPS when compared to Cellpose. The H2 model yields on average 1.8x higher FPS when compared to Omnipose and 1.4x higher FPS when compared to Cellpose.

It should be noted that the most time consuming part of these methods is the final DBSCAN clustering step. This is largely due to the fact that it's a CPU based implementation. To put into perspective how long clustering takes, the H1 model with integration though without the final clustering step yields 25.3 FPS on the worm subset, 23.7 FPS on the bacterial fluorescence subset, and 13.9 FPS on the bacterial phase contrast subset. That's an average of 21.0 FPS. This is compared to the 8.5 FPS average with clustering.

Several factors play a role in the decreased evaluation speed of the proposed method. First, the H1 model is 3.5x smaller than the Omnipose and Cellpose models. Furthermore, the improvements presented in Section 4 lead to several simplifications being made to the postprocessing steps. The removed postprocessing steps include: Normalization of predicted vector field to unit length then rescaling by the divergence and several classic postprocessing steps such as the removal of objects with area smaller than some threshold and more.

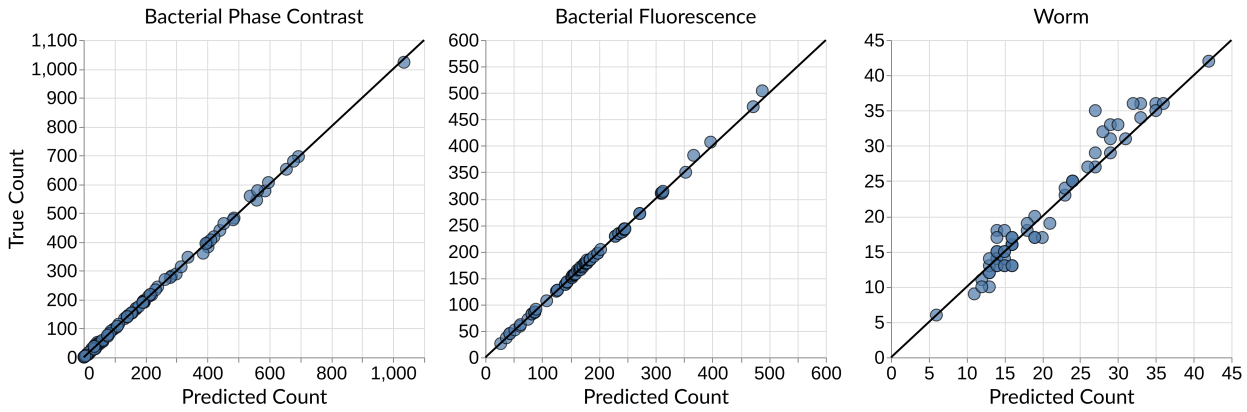

Figure 8: Vector field based methods yield high cell counting performance for images with large quantities of cells. Due to the proposal-free nature of vector field based methods, the number of cells in an image doesn't impact performance. Furthermore, vector fields based methods can accurately segmentat irregularly shaped and non-convex objects. Together, this makes instance segmentation via vector field based methods a strong intermediate task for cell counting.

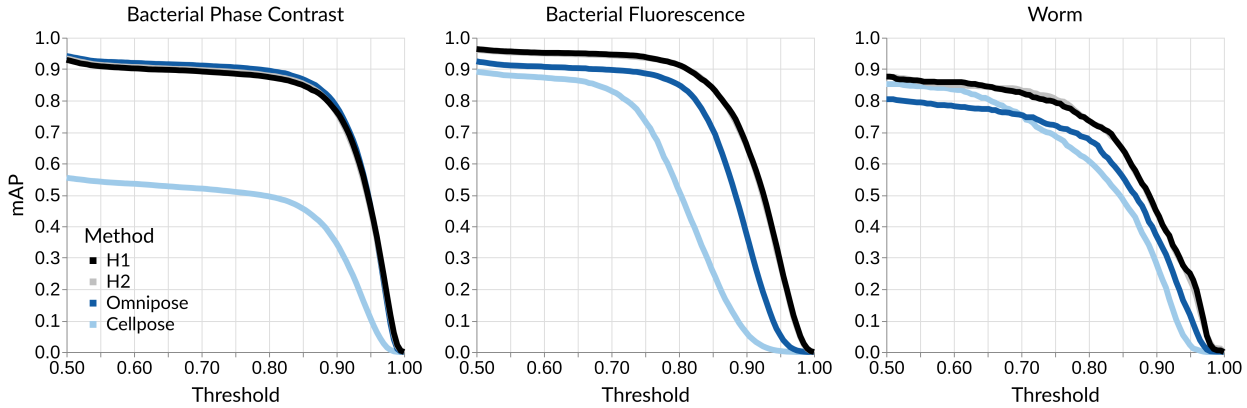

Figure 9: Graph demonstrating the average precision (AP) of the H1, H2, Omnipose and Cellpose models at 100 IoU thresholds between 0.5 and 1.0. Notice that centroid based methods (Cellpose) perform particularly bad on the bacterial phase contrast and fluorescence subsets which contain irregularly shaped and non-convex objects.

7.5 An Application: Cell Counting

Setup. Instance segmentation is a logical intermediate task for performing cell counting. To quantitatively evaluate the performance of the proposed method on cell counting I computed the MAE of ground truth cell counts and predicted cell counts. The H1 models benchmarked in Table 1 were used to evaluate the cell counting performance. To study how the number of cells in an image may impact performance, Figure 8 demonstrates the performance visually by plotting the predicted cells on the x-axis, the true cell count on the y-axis, and the line y = x. If predictions were perfect they would exactly line up on the y = x line. Inversely, the farther the points deviate from the line y = x, the worse the performance.

Results. On the bacterial phase contrast subset, the H1 model yields 2.57 MAE. Likewise, the H1 model yields 1.89 and 1.67 MAE on the bacterial fluorescence and worm subset respectively. Over all three subsets in the BPCIS dataset, the proposed method yields an average MAE of 2.04.

Referring to Figure 8, notice that as the cell count increases the performance does not drop off. This is a benefit of the proposal-free nature of vector field based methods. Typically, proposal-based methods tend to undersegment when the cell count increases [EJK+21]. Here, we can see performance remains linear as the cell count increases, further demonstrating the superior performance of vector field based methods.

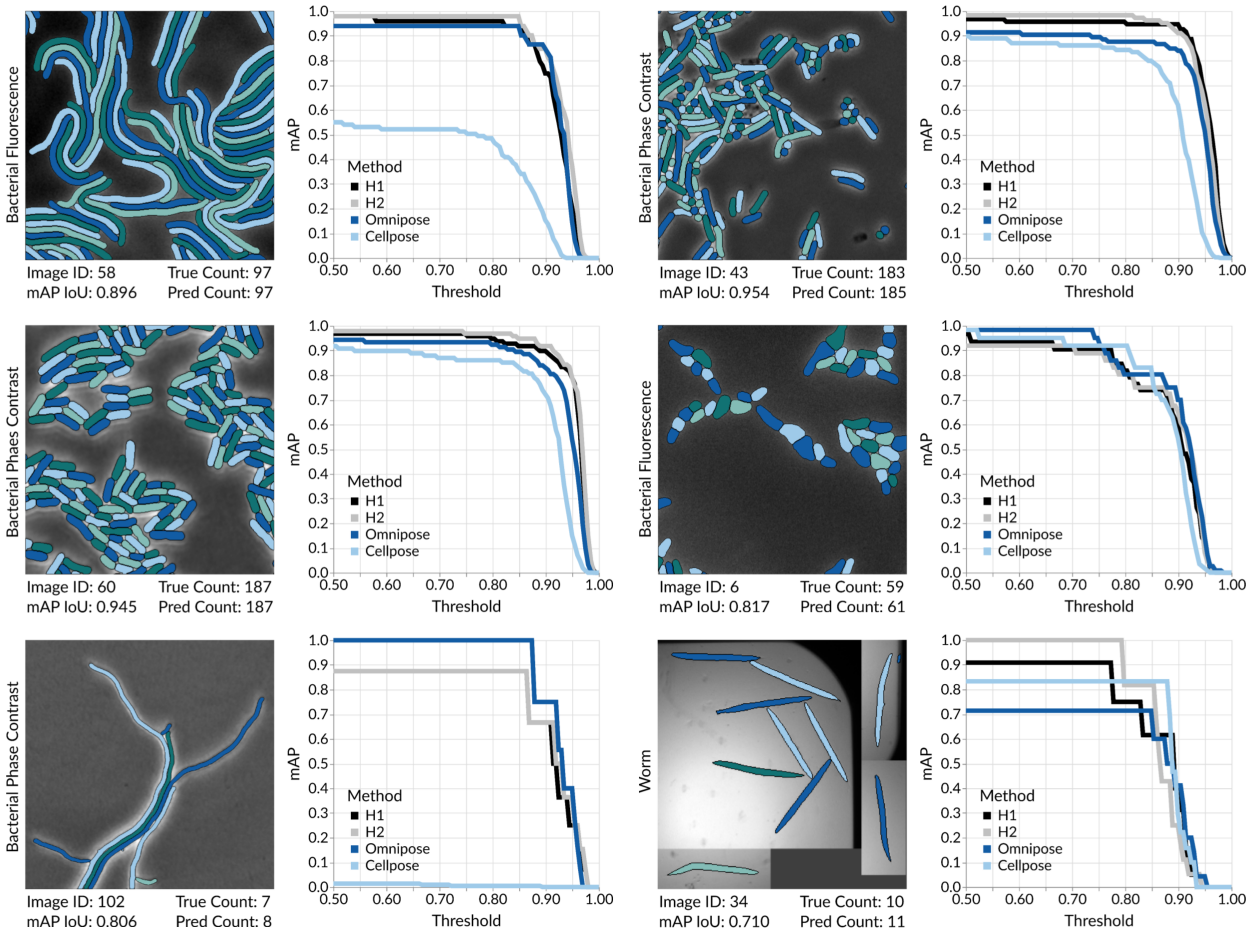

Figure 10: Vector field based methods with the proposed improvements demonstrate stunning accuracy on instance segmentation of dense and irregularly shaped objects. Each image demonstrates the instance segmentation as predicted by the H1 model benchmarked in Table 1. The predicted cell count and true cell count is given below each image along with the image ID and mAP IoU. The graph to the right of each image is the average precision (AP) of the H1, H2, Omnipose, and Cellpose models at 100 IoU thresholds between 0.5 at 1.0.

8 Conclusion

In this article I presented modeling vector fields for instance segmentation as a powerful yet lightweight class of proposal-free instance segmentation methods. On the development of these methods, I layed both a mathematical and practical foundation, highlighted several advantages, and proposed several improvements and simplifications to previous works on vector field based methods.

On the improvement of prior works, I proposed a novel integration-aware loss function to make training more end-to-end. Furthermore, I noticed that previously modeled vector fields required models to consider long range interactions, a known limitation to convolutional neural networks. As a solution, I proposed we model vector fields after the gradient of the *truncated* signed distance field to limit the range of interactions models need to consider. Furthermore, I noticed these vector fields contained significant artifacts, to which I proposed we use a large gaussian-smoothed finite difference kernel for numeric gradient computation. Overall, when compared to the previous state-of-the-art vector field based method, the proposed improvements and simplifications lead to:

- **3.3x** faster training speed per epoch.
- **2.1x** faster eval time (FPS).
- **5.7%** average increase in mAP IoU performance.
- Faster convergence during training.

References

[CSWM21] Kevin J. Cutler, Carsen Stringer, Paul A. Wiggins, and Joseph D. Mougous. Omnipose: a high-precision morphology-independent solution for bacterial cell segmentation. *bioRxiv*, 2021.

[EJK+21] Christoffer Edlund, Timothy R. Jackson, Nabeel Khalid, Nicola Bevan, Timothy Dale, Andreas Dengel, Sheraz Ahmed, Johan Trygg, and Rickard Sjogren. Livecell-a large-scale dataset for label-free live cell segmentation. *Nature Methods*, 18(9):1038–1045, Sep 2021.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[NDBPVG19] Davy Neven, Bert De Brabandere, Marc Proesmans, and Luc Van Gool. Instance segmentation by jointly optimizing spatial embeddings and clustering bandwidth, 2019.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS 2017 Workshop on Autodiff*, 2017.

[PGM+19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[PVG+11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[SP22] Carsen Stringer and Marius Pachitariu. Cellpose 2.0: how to train your own model. *bioRxiv*, 2022.

[SWMP21] Carsen Stringer, Tim Wang, Michalis Michaelos, and Marius Pachitariu. Cellpose: a generalist algorithm for cellular segmentation. *Nature Methods*, 18(1):100–106, Jan 2021.

[YC21] Batuhan Yildirim and Jacqueline M. Cole. Bayesian particle instance segmentation for electron microscopy image quantification. *Journal of Chemical Information and Modeling*, 61(3):1136–1149, 2021. PMID: 33682402.