KymoButler, a Deep Learning software for automated kymograph analysis

Maximilian A. H. Jakobs*, Andrea Dimitracopoulos, Kristian Franze*

Department of Physiology, Development and Neuroscience, University of Cambridge, Cambridge, UK

* To whom correspondence should be addressed: mj455@cam.ac.uk (M.A.H.J.) or kf284@cam.ac.uk (K.F.)

Abstract

Kymographs are graphical representations of spatial position over time, which are often used in biology to visualise the motion of fluorescent particles, molecules, vesicles, or organelles moving along a predictable path. Although in kymographs tracks of individual particles are qualitatively easily distinguished, their automated quantitative analysis is much more challenging. Kymographs often exhibit low signal-to-noise-ratios (SNRs), and available tools that automate their analysis usually require manual supervision. Here we developed KymoButler, a Deep Learning-based software to automatically track dynamic processes in kymographs. We demonstrate that KymoButler performs as well as expert manual data analysis on kymographs with complex particle trajectories from a variety of different biological systems. The software was packaged in a web-based "one-click" application for use by the wider scientific community. Our approach significantly speeds up data analysis, avoids unconscious bias, and represents another step towards the widespread adaptation of Machine Learning techniques in biological data analysis.

Introduction

Many processes in living cells are highly dynamic, and molecules, vesicles, and organelles diffuse or are transported along complex trajectories. Particle tracking algorithms represent powerful approaches to track the dynamics of such particles ((Jaqaman et al. 2008; Sbalzarini & Koumoutsakos 2005; Lee & Park 2018)). However, particularly in scenarios where particles follow a stationary path and move much faster than the confounding cell (e.g., as in molecular transport along neuronal axons and dendrites, retrograde actin flow, or cilia transport), kymographs provide an elegant solution to the visualisation and analysis of particle dynamics. Kymographs are generated by stacking the intensity profile along a defined path for each time point of a movie. In the resulting space-time image, each (usually fluorescently) labelled particle is shown as a line, whose slope, for example, represents the velocity of that particle (Figure 1A).

In many biological processes, multiple particles move along the same stationary path with little to no deviations, making kymographs a very useful representation of their dynamics. Hence, kymographs have been widely employed to visualise biological processes across different length scales, ranging from diffusion and transport of single molecules to whole cell movements (Twelvetrees et al. 2016; Barry et al. 2015). The analysis of these kymographs only requires tracing lines in 2D images, a rather simple task compared to the more general approach of particle tracking, where one has to identify the centre of the particles in each frame, and then correctly assign these coordinates to corresponding particles across frames.

Publicly available kymograph analysis software simplifies the tedious and time-consuming task of tracing kymographs, but most of these solutions require manual supervision, and they are mainly applicable to particles that follow a unidirectional motion, i.e. do not change their direction or velocity (Figure 1C, example 2) (Neumann et al. 2017; Mangeol et al. 2016; Chenouard et al. 2010; Zala et al. 2013). This category includes, for example, the dynamics of growing microtubule +ends and F-actin dynamics in retrograde actin flow (Lazarus et al.

2013; del Castillo et al. 2015; Alexandrova et al. 2008; Babich et al. 2012). In many other biological contexts, however, particles can stop moving, change direction, merge, cross each other's path, or disappear for a few frames. The kymographs obtained from these processes exhibit 'bidirectional' motion (Figure 1C, example 1); this category includes cellular transport processes, for example molecular or vesicle transport in neuronal axons and dendrites (Faits et al. 2016; Tanenbaum et al. 2013; Koseki et al. 2017). Thus, the problem of automatically and reliably tracking dynamic processes in kymographs is still largely unresolved, and given the limitations of currently available kymograph analysis software, most kymographs are still analysed by hand, which is slow and gives rise to unconscious bias.

In recent years, Machine Learning (ML), and particularly Deep Neural Networks, have been very successfully introduced to data processing in biology and medicine (Mathis et al. 2018; Weigert et al. 2017; Florian et al. 2017; Guerrero-Pena et al. 2018; Falk et al. 2019; Bates et al. 2017). ML-based image analysis has several advantages over other approaches: it is less susceptible to bias than manual annotation, it takes a much shorter time to analyse large datasets, and, most importantly, it comes closer to human performance than conventional algorithms (Mathis et al. 2018).

Most ML approaches to image analysis utilise **F**ully **C**onvolutional Deep Neural **N**etworks (FCNs) that were shown to excel at object detection in images (Dai et al. 2016; Szegedy et al. 2014; LeCun et al. 2008; Falk et al. 2019). Through several rounds of optimisation, FCNs select the best possible operations by exploiting a multitude of hidden layers. These layers apply image convolutions using kernels of different shapes and sizes, aiming to best match the output of the neural network to the provided training data labels, which were previously derived from manual annotation. This means that the network learns to interpret the images based on the available data, and not on *a priori* considerations. This approach has become possible due to the incredible improvements in computation times of modern CPUs and the adoption of GPUs that can execute an enormous number of operations in parallel. Currently, the most successful architecture for biological and medical image analysis is the U-Net, which takes an input image to generate a binary map that highlights objects of interest based on the training data (Ronneberger et al. 2015).

Here we present KymoButler, a new stand-alone FCN software based on the U-Net architecture, to automatically and reliably extract particle tracks from kymographs. The software was packaged into an easy-to-use web interface and a downloadable software package, and it was benchmarked against traditional software and manual annotation on synthetic (i.e., ground truth) data. We show that KymoButler performs very well on challenging bidirectional kymographs, where particles disappear, reappear, merge, cross each other's path, move in any direction, change speed, immobilise, and reverse direction. KymoButler thus represents a substantial improvement in the automation of kymograph tracing, speeding up the experimental workflow, while preserving the accuracy of manual annotations.

Results

The KymoButler software package

For our FCN-based kymograph analysis software, we implemented a customised architecture based on the U-Net (Ronneberger et al. 2015). We first trained the FCN to segment kymographs, i.e. binarize the image into regions with particle tracks (foreground) and noise (background). Our training data consisted of manually annotated tracks in 487 unidirectional and 79 bidirectional kymographs (unpublished data from our group and other laboratories, see Materials and Methods and Acknowledgements for details). Since no ground truth was available in the manually annotated kymographs, we also generated 221 synthetic unidirectional and 21 synthetic bidirectional kymographs that were used for training (see Figure 1-figure supplement 3 for examples).

Our network takes an input kymograph to generate 2D maps that assign a "trackness" value between 0 and 1 to each pixel of the input image, with higher values representing a higher likelihood of pixels being part of a track. The training was performed with pixel-wise cross-entropy loss (see Methods for details) and implemented in Mathematica (http://www.wolfram.com/mathematica). We furthermore took advantage of the intrinsic differences in the appearance of unidirectional and bidirectional kymographs and trained two separate specialised networks, a unidirectional segmentation module, and a bidirectional segmentation module (Figure 1–figure supplement 1 and Figure 1–figure supplement 2).

The unidirectional segmentation module generates separate trackness maps for tracks with negative and positive slopes (which could, for example, correspond to tracks of anterograde and retrograde transport processes, respectively), to remove line crossings from the output (Figure 1–figure supplement 1). The trackness maps are then binarized and morphologically thinned to yield separated lines in a skeletonized map (Figure 1–figure supplement 1). We found the binarization threshold to depend on the biological application and on the signal to noise ratio of the input image. For our synthetic data, we used a value of 0.2 and generally observed consistent results for both segmentation modules between 0.1-0.3 (Figure 1–figure supplement 4).

In bidirectional kymographs, tracks show more complex morphologies, since they can change direction and cross each other multiple times. The bidirectional segmentation module therefore generates a single trackness map, which needs to be further processed in order to obtain individual particle tracks. After thresholding and morphologically thinning the trackness map, we obtained a skeletonised image with multiple track crossings (Figure 1–figure supplement 1). In these images, we detected starting points of tracks by morphological operations (Figure 1–figure supplement 1B) and moved along each line from one row (time point) to the next. Then, whenever a crossing point was encountered (with two or more possible pixels to advance to), the software calls a decision module to resolve the crossing. The decision module, again based on a modified version of the U-Net, is specialised in solving these crossings and trained on our bidirectional kymograph data (Figure 1–figure supplement 1B and Figure 1–figure supplement 2). The inputs of the module consist of three 48 by 48 pixel crops: (1) the input kymograph, (2) the skeletonised trackness map, and (3) the skeleton of the current track (Figure 1–figure supplement 1B). The output of the module is a map that assigns a score between 0 and 1 to each pixel of the skeletonised trackness map (2). Then, the most likely skeleton segment to continue the current track (3) is selected from the decision score map and the average score saved as a measure for track confidence. If the predicted path is less than 3 pixels long, the track is resolved and terminated. Once all the tracks with starting points are resolved, they are removed from the skeletonised trackness map, which is then scanned again for starting points, and the steps above are repeated until no further starting points are found. Furthermore, long overlaps between tracks are assigned to the track with the highest confidence so that no large overlapping regions between tracks are found in the final result (see Materials and Methods).

Finally, we implemented the class module, a simple convolutional network that classifies input kymographs into unidirectional or bidirectional classes (Figure 1–figure supplement 1B and Figure 1–figure supplement 2A). The class module was trained on both unidirectional and bidirectional data until the error rate on a validation dataset, which contained 72 kymographs and their classes, became persistently 0%. We linked the class module to the unidirectional and bidirectional segmentation modules as well as to the decision module (Figure 1–figure supplement 1B), and packaged them into KymoButler, an easy-to-use, drag & drop browser-based app for quick and fully automated analysis of individual kymographs (http://kymobutler.deepmirror.ai).

The only free parameter in KymoButler is the threshold for trackness map segmentation. The default threshold is 0.2, but users can freely adjust it between 0.1 and 0.3 (+1 and -1 in the cloud interface) for their specific application. After the computation, which only takes 1-20 seconds per kymograph (depending on complexity), KymoButler generates several files including a dilated overlay image highlighting all the tracks found in different colours, a CSV file containing all track coordinates, and another summary file with post processing data, such as average velocities and directionality (Figure 1B). Finally, we tested KymoButler on previously published kymographs from a variety of different biological data (Figure 1C and Figure 1-figure supplement 1A) and on unpublished data from collaborators (not shown).

Performance on unidirectional Kymographs

We quantitatively evaluated the performance of KymoButler on unidirectional kymographs, i.e. particles that move with mostly uniform velocities and with no change in direction (Figure 1C, Figure 2, Figure 1-figure supplement 1A). The unidirectional module of KymoButler was compared to an existing kymograph analysis software, which is based on Fourier filters, and which provided the best performance among publicly available software in our hands (KymographDirect package (Mangeol et al. 2016)). Additionally, we traced kymographs by hand to obtain a control for the software packages.

First, we generated 10 synthetic movies depicting unidirectional particle dynamics with low signal-to-noise ratio (~1.2, see Materials and Methods) and extracted kymographs from those movies using the KymographClear (Mangeol et al. 2016) Fiji plugin. Each of the kymographs was then analysed by Fourier-filtering (KymographDirect), KymoButler, and by hand, and the identified trajectories overlaid with the ground truth (i.e., the known dynamics of the simulated data) (Figure 2A).

We then quantified the quality of the predicted traces. We first determined the best predicted track for each ground truth track (in case several segments were predicted to cover the same track) and then calculated the fraction of the length of the ground truth track that was correctly identified by that predicted track (“track recall”) (Figure 2B). Additionally, we determined the best overlapping ground truth track for each predicted track and then calculated the fraction of the length of the predicted track that was overlapping with the ground truth track ("track precision"). Examples of low/high precision and low/high recall are shown in Figure 2B. We then calculated the geometric mean of the average track recall and the average track precision (the "track F1 score", see methods) for each kymograph (Figure 2E). The median F1 score of the manual control was 0.90, KymoButler achieved 0.93, while Fourier filtering achieved a significantly lower F1 score of 0.63 ($p = 4 \cdot 10^{-5}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.6$, manual vs Fourier Filtering $p = 3 \cdot 10^{-3}$).

Our synthetic data also included gaps of exponentially distributed lengths (see Materials and Methods), allowing us to quantify the ability of KymoButler to bridge gaps in kymograph tracks (Figure 2C, F), which are frequently encountered in kymographs extracted from fluorescence data (Applegate et al. 2011). Both KymoButler and manual annotation consistently bridged gaps that belonged to the same trajectory, while Fourier filtering was less accurate (89% of all gaps correctly bridged by KymoButler, 88% by manual, and 72% by Fourier filter analysis; median of all 10 synthetic kymographs, $p = 10^{-4}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.9$, manual vs Fourier Filtering $p = 2 \cdot 10^{-3}$, Figure 2F).

We also quantified the ability of KymoButler to resolve track crossings. Again, both KymoButler and manual annotation performed significantly better than Fourier filtering (88% KymoButler, 86% manual, 60% Fourier filter; median percentage of correctly resolved crossings of all 10 synthetic kymographs, $p = 10^{-4}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.9$, manual vs Fourier Filtering $p = 1 \cdot 10^{-3}$, Figure 2G). In summary, KymoButler was able to reliably track particle traces in kymographs at low SNR, and it clearly outperformed currently existing software, while being as consistent as manual expert analysis.

KymoButler performance on bidirectional Kymographs

As in many kymographs obtained from biological samples trajectories are not unidirectional, we also tested the performance of KymoButler on complex bidirectional kymographs, i.e. of particles with wildly different sizes, velocities, and fluorescence intensities that frequently change direction, may become stationary and then resume motion again (see Figure 1B, C, Figure 3A, Figure 1-figure supplement 1A for examples). Available fully automated software that relied on edge detection performed very poorly on our synthetic kymographs (Figure 3-figure supplement 1). Therefore, we implemented a custom-written wavelet coefficient filtering algorithm in order to compare our FCN-based approach to a more traditional non-ML approach (Figure 3A, Figure 3-figure supplement 1, Materials and Methods). In short, the wavelet filtering algorithm generates a trackness map, similar to KymoButler, by applying a stationary wavelet transform to the kymograph to generate so-called “coefficient images” that highlight horizontal or vertical lines. These coefficient images are then overlaid and binarized with a fixed value (0.3), skeletonised, and fed into the KymoButler algorithm without the decision module, i.e. crossings are resolved by linear regression prediction.

We generated 10 kymographs from our synthetic movies with the KymographClear package (average signal-to-noise ratio was 1.4, since any lower signal generally obscured very faint and fast tracks). Each of the kymographs was then analysed by wavelet coefficient filtering, KymoButler, and manual annotation, and the predicted traces overlaid with the ground truth (Figure 3A). While the wavelet approach and KymoButler were able to analyse the 10 kymographs in less than one minute, manual annotation by an expert took about 1.5 hours. Moreover, whereas the manual annotation and KymoButler segmentation overlaid well with the ground truth, the wavelet approach yielded numerous small but important deviations.

Similarly to the unidirectional case, we quantified track precision and recall (Figure 3B, E) and calculated the resolved gap fraction (Figure 3C, F) and crossing fraction (Figure 3D, G). The median of the track F1 scores per kymograph for manual annotation (0.82) was similar to KymoButler (0.80), while the wavelet filter approach only gave 0.60 ($p = 8 \cdot 10^{-5}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.7$, manual vs wavelet filtering $p = 10^{-4}$, Figure 3E). While gaps were resolved by KymoButler and manual annotation in 89% and 95% of cases, respectively, only 74% were resolved by the wavelet algorithm (median of all 10 synthetic kymographs, $p = 3 \cdot 10^{-4}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.4$, manual vs wavelet filtering $p = 2 \cdot 10^{-4}$, Figure 3F). Crossings were rarely resolved correctly by the wavelet algorithm (12%) but much more reliably by KymoButler (61%) and manual annotation (76%) (median of all 10 synthetic kymographs, $p = 3 \cdot 10^{-5}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.4$, manual vs wavelet filtering $p = 2 \cdot 10^{-5}$, Figure 3G).

Overall, these results showed that KymoButler performs well on both unidirectional and bidirectional kymographs, outperforms currently available automated analysis of kymographs, and it performs as well as manual tracing, while being much faster and not prone to unconscious bias.

Discussion

In this work, we developed software based on Deep Learning techniques to automate the tracking of dynamic particles along a stationary path in a noisy cellular environment. Convolutional neural networks (CNNs) are nowadays widely applied for image recognition. Since tracking is *a priori* a visual problem, we built a modular software utilising CNNs for identifying tracks in kymographs. We deployed our networks as KymoButler, a software package that takes kymographs as inputs and outputs all tracks found in the image in a matter of seconds. The network outperforms standard image filtering techniques on synthetic data as well as on kymographs from a wide range of biological processes, while being as precise as expert manual annotation.

The KymoButler software has only one adjustable parameter that is left to the user: a sensitivity threshold that, if low, allows more ambiguous tracks to be recognised, and if high discards them. For our synthetic data, the best value for the threshold lay between 0.1 and 0.3 (Figure 1-figure supplement 4), and we observed a similar range for a variety of kymographs from published data. However, the threshold depends on the SNR of the input images, so that the correct threshold has to be chosen based on each biological application and imaging conditions. We strongly recommend to visually inspect the output of KymoButler for each new application, and to compare the output to manual annotation.

Most of the publicly available kymograph analysis software requires manual labelling to extract quantitative data (Chenouard et al. 2010; Neumann et al. 2017; Zala et al. 2013). Some automated approaches have been published in the context of specific biological questions, but since these programs are currently not publicly available it is not clear how well they would perform on kymographs from other applications (Mukherjee et al. 2011; Reis et al. 2012). Other approaches do not extract individual tracks but only macroscopic quantities, as for example velocities (Chan & Odde 2008). As KymoButler is fully automated and able to reliably analyze kymographs from a wide range of biological applications, it fills an important gap. Here we showed that KymoButler is able to quantify mitochondria movement in neuronal dendrites, microtubule growth dynamics in axons, and *in vitro* dynamics of single cytoplasmic dynein proteins (Figure 1 and Figure 1-figure supplement 1). We predict that it can furthermore be applied to most if not all other kymographs obtained from time-lapse fluorescence microscopy without the need of any modifications.

KymoButler outperformed Fourier filtering, edge detection, and customised wavelet coefficient selection on synthetic kymographs. While Fourier filtering ‘only’ performed ~30% worse than KymoButler on unidirectional kymographs, edge detection on bidirectional kymographs suffered greatly from background fluctuations and low SNR to such an extent that the extracted data was unusable (see Figure 3-figure supplement 1 for one example). Therefore, we designed a filtering algorithm based on wavelet coefficient image selection to analyse complex bidirectional kymographs specifically for our synthetic data. KymoButler still performed 25% better than this approach (Figure 3). The main problem with either filtering approach compared to KymoButler was their inability to bridge track gaps and resolve line crossings, both of which occur frequently in biological data (Figure 2C, D and 3C, D). These challenges are met by KymoButler, which performed as well as expert annotation, but within a much shorter time (Figure 2 and 3).

Our results show that KymoButler is able to correctly identify individual full-length tracks in kymographs with an average track F1 score (geometric mean of track precision and recall) of 92% on unidirectional tracks and 80% on complex bidirectional tracks, without suffering from inconsistency, bias, and laborious tracing, that plague manual tracking. While KymoButler is already performing very well, we aim to significantly improve it over future iterations. Every time a researcher uses our webform, the corresponding kymograph is anonymously uploaded to our cloud. Once a large number of diverse kymographs is uploaded, these kymographs will be added to our training data, improving KymoButler even further.

The ultimate challenge will be to expand our approach to 2D or even 3D tracking problems. Here, we defined a 1D region of interest in 2D time-lapse movies, extracted 2D (space and time) images (kymographs), and finally tracked 2D lines in those images. A similar, albeit computationally heavier, approach could stack the frames of a 2D/3D movie on top of each other to generate a 3D/4D kymogram (2D space and time, or 3D space and time). Previously generated kymograms have led to intriguing results on whole-cell particle tracking problems with high SNR (Racine et al. 2007). The use of higher dimensional FCNs in the future has great potential to yield human-like performance on any biological and medical tracking problems.

Material and Methods

All code was written in the wolfram language in Mathematica https://wolfram.com/mathematica and, if not stated otherwise, can be found online under our GitHub: https://github.com/deepmirror/KymoButler

The KymoButler software package

The KymoButler software was implemented in Mathematica to take advantage of easy web form deployment and distribution. The workflow is shown in Figure 1-figure supplement 1B. Our approach was to first segment kymograph pixels that are part of particle tracks from pixels that were part of the background with our segmentation modules. From previous work we knew that kymographs that depict unidirectional movement only, can be filtered into tracks that have positive slope and those that have negative slope (Chenouard et al. 2010), while no such assumptions can be made about bidirectional kymographs. Hence, we decided to take advantage of this simplification of unidirectional kymograph analysis by training two modules: one that is specialized to segment unidirectional kymographs and another one that segments bidirectional ones. Note that the bidirectional module is able to analyze any kymograph, including unidirectional ones, but since it is not specialized it performs slightly worse than the unidirectional module on unidirectional kymographs. To further simplify software usability, we prepended a class module that classifies input kymographs as bidirectional or unidirectional, and then applies the corresponding segmentation module and decision module (for bidirectional kymographs only). Our downloadable software package on GitHub allows the user to call either segmentation module (unidirectional/bidirectional) directly, if they wish to do so.

When the kymograph is classified as unidirectional by the class module, the unidirectional segmentation module generates two trackness score maps for particles with negative or positive slope (Figure 1-figure supplement 1B). Since the particles move with roughly the same velocity, the resulting maps mostly do not exhibit any crossings. Thus, we binarize the maps with a threshold between 0.1-0.3 (see benchmarking section for more information about the threshold). The resulting binary maps are then thinned iteratively so that each trace is only one pixel wide at any point and pruned so that branches that are shorter than 3 pixels are deleted. Subsequently, each trace is segmented and selected only if they are at least 3 frames long. In the final step, pixels that lie in the same row of the kymograph are averaged over so that the final track has only one entry per frame.

For bidirectional kymographs the software generates a trackness map, applies a binarization threshold (0.1-0.3, see benchmarking for more details), iterative thinning, and pruning (minimum length 3 pixels). However, since the resulting skeletonised map had a substantial number of crossings, and could not be easily segmented to yield individual tracks, we implemented a further module in the software. First, all lines in the skeletonised map are shortened so that each white pixel at a track end only has neighbouring pixels in different rows (time dimension). This was done so that we could detect track starting points ("seeds") with a Hit-Miss transformation with kernel: $\begin{pmatrix} -1 & -1 & -1 \\ -1 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix}$. Application of this kernel yielded a binary map with 0 everywhere except at track seeds (Figure 1-figure supplement 1B, red dots). These seeds were then used to start tracing individual tracks in the kymograph by always advancing to the next white pixel. Once more than one potential future pixel is encountered, the decision module is called. The module generates three 48x48 crops of (1) the input kymograph, (2) the skeletonised trackness map, and (3) the skeleton of the current track and predicts a trackness map that has high values on the skeleton segment of the most likely future track (Figure 1-figure supplement 1B). This map is binarized with threshold 0.5 and thinned. The precise threshold had little effect on the final output, so we fixed it at 0.5 for all applications. Next, the largest connected component in the map is selected as the most likely future path and appended to the track if longer than 2 pixels. The average trackness value of this component (from the decision module prediction) is saved as a measure of decision "confidence". This process is repeated until no further possible pixels are found or no future path is predicted which is when the track is terminated. Once all seeds are terminated, the software subtracts all the found paths from the skeletonised trackness map and again looks for new seeds which are then again tracked in the full skeletonised image. The process is repeated until no further seeds are found, and then all tracks are averaged over their timepoints (rows in the kymograph image). Subsequently the software deletes tracks that are shorter than 5 pixels or part of another track and assigns overlaps that are longer than 10 pixels to the track with the highest average decision confidence.

Both the unidirectional and the bidirectional module output a coloured overlay in which each track is drawn in a different randomly assigned colour and dilated with factor 1 for better visibility (see Figure 1B-C and Figure 1-figure supplement 1A). Additionally, the software generates one CSV file that contains all the track coordinates and a summary CSV file that gives derived quantities, such as track direction and average speed.

The software was deployed from Mathematica as a cloud based interface (http://kymobutler.deepmirror.ai) and a Mathematica package (https://github.com/deepmirror/KymoButler)

Network architectures

Our networks were built from convBlocks (a convolutional layer with 3x3 kernel size, padding, and arbitrary number of output channels followed by a batch normalisation layer and a 'leaky' ramp (leakyReLU) activation function ($leakyReLU(x) := max(x, 0) - 0.1\, max(-x, 0)$). Batch normalisation is useful to stabilise the training procedure as it rescales the inputs of the activation function (leakyReLu), so that they have zero mean and unit variance. The leakyReLu prevents the so-called "dead ReLu's" by applying a small gradient to values below 0. These building blocks were previously used for image recognition tasks in *Google's* inception architecture and in the U-Net architecture (Szegedy et al. 2014; Falk et al. 2019).

The module architectures we settled on are shown in Figure 1–figure supplement 1-2. All modules used the same core building blocks while having different input and output ports. The classification module takes a resized kymograph of size 64x64 pixels and generates two output values that correspond to the class probabilities for unidirectional/bidirectional kymographs (Figure 1–figure supplement 2A). The unidirectional segmentation module takes one input kymograph and generates two output images that correspond to the trackness scores of particles with positive or negative slopes (Figure 1–figure supplement 2B). The bidirectional segmentation module takes one input kymograph and generates one trackness score map highlighting any found particle tracks (Figure 1–figure supplement 2C). Finally, the decision module takes three inputs of size 48x48 pixels to generate one trackness map (Figure 1–figure supplement 2D). All modules share the same core network that is essentially a U-Net with padded convolutions and with 64 (in the top level) to 1024 (in the lowest level) feature maps. We experimented with more complex architectures (parallel convolution modules instead of blocks, different number of feature maps) but could only observe minor increase in accuracy at a large expense in computation time. Due to the U-Net architecture, each dimension of the inputs to the segmentation modules needs to be a multiple of 16. Thus, inputs were resized when they did not match the dimension requirements, and then the binarized output images from the segmentation modules were resized to the original input image size before proceeding further.

Network training

To train the networks we quantified the difference between their output o and the desired target output t through a cross entropy loss layer ($CEloss(t, o) = -(t \cdot ln(o) + (1 - t) \cdot ln(1 - o)$). The loss was averaged over all output entries (pixels and classes) of each network. While we tried other loss functions, specifically weighted cross entropy loss and neighbour dependent loss as described in (Bates et al. 2017), we persistently obtained higher precision and recall with the basic cross entropy loss above.

Our training data comprised a mixture of synthetic data and manually annotated unpublished kymographs, kindly provided by the research groups mentioned in the acknowledgements. Most of the manual annotation was done by M. A. H. J. and A. D. In total, we used 487 (+200 synthetic) unidirectional, and 79 (+21 synthetic) bidirectional kymographs, with 95% of the data used for network training, and ~5% of retained for network validation. All network training was performed on a workstation, using a nVidia 1080 Ti or a nVidia 1070 GPU.

The class module depicted in Figure 1-figure supplement 2A was trained with batches of size 50 (with 25 unidirectional and 25 bidirectional kymographs to counter class imbalance) with random image transformations that included image reflections, rotations, resizing, colour negation, gaussian noise, random noise, and random background gradients. The final input image was randomly cropped to 64x64 pixels (see examples Figure 1-figure supplement 3A) and the class module was trained using stochastic gradient descent (ADAM optimiser (Kingma & Optimization n.d.), initial learning rate 0.001), until the validation set error rate was consistently 0%.

The unidirectional segmentation module (Figure 1-figure supplement 2B) was trained with batches comprising 20 randomly selected kymographs from our training set (example in Figure 1-figure supplement 3B). We applied the following image transformations: Random reflections along either axis, random 180-degree rotations, random cropping to 128x80 pixels (approximately the size of our smallest kymograph), random gaussian and uniform noise, and random background gradients. Note that we did not apply any resizing to the raw kymograph since that generally decreased net performance. Additionally, we added Dropout Layers (10-20%) along the contracting path of our custom U-Net to improve regularisation. Each kymograph in this training set was generated by hand with KymographTracker (Chenouard et al. 2010), but to increase dataset variability we took the line profiles from KymographTracker and generated kymographs with a custom Mathematica script that applied wavelet filtering to the plotted profiles. The resulting kymographs have a slightly different appearance than the ones created with KymographTracker and are thus useful to regularize our training process. Several modules were trained until convergence and the best performing one (according to the validation score) was selected (ADAM optimiser, initial learning rate of 0.001, learning rate schedule = $If[batch < 4000, 1, .5]$).

The bidirectional segmentation module (Figure 1-figure supplement 2C**,** example data Figure 1-figure supplement 3C) was trained in the same way as the unidirectional segmentation module, with the exception of a slightly different learning rate schedule ($If[batch < 3000, 1, .5]$). Additionally, since we did not have access to many of the original movies from which the kymographs were generated, we could not generate kymographs with different algorithms as done for the unidirectional module.

Training data for the decision module (Figure 1-figure supplement 2D) was obtained from the bidirectional (synthetic + real) kymographs by first finding all the branch points in a given ground truth or manually annotated image. Then, each track was separated into multiple segments, that go from its start point to a branching point or its end point. For each branchpoint encountered while following a track, all segments that ended within 3 pixels of the branchpoint were selected. Then, (1) a 48x48 pixel crop of the raw kymograph around the branchpoint, (2) a binary map representing the track segment upstream of the branching point (centred with its end in pixel coordinates 25,25, with image padding applied if the end was close to an image corner), and (3) the corresponding 48x48 pixel region in the binary image representing all possible paths were used as inputs to the decision module. The binary image representing the ground truth or annotated future segment downstream of the branchpoint was used as the target image (see Figure 1-figure supplement 3D for an example training set). Thus, the training set comprised three input images and one output image which we used to train the decision module. To increase the module's focus on the non-binary raw kymograph crop, we applied 50% dropout to the full skeletonised input and 5% dropout to the input segment. As explained above, we used random image augmentation steps like reflections, rotations, gaussian + uniform noise. Additionally, we employed random morphological thinning to the binary input/output images to simulate artefacts. Several networks were trained until convergence (pixel wise cross entropy loss, ADAM optimiser, initial learning rate 0.001, batch size 50, learning rate schedule $If[batch < 8000, 1, .5]$), and the best performing one was selected.

Synthetic Data

Synthetic data was generated by simulating individual particles on a stationary path of length 300 pixels for 300 frames to generate 300x300 pixel kymographs. To obtain unidirectional particles we seeded 30+30 particles with negative or positive slope at random timepoints/positions. Next, a random velocity between 1-3 pixels/frame was chosen for all particles in the movie, with a random noise factor to allow slight changes in velocity, and a particle PSF between 3-6 pixels. Each particle was assigned a survival time drawn from an exponential distribution with scale 0.01, after which it would disappear. Gaps of random length (exponentially distributed) were subsequently assigned to each track individually. From these tracks we then generated a kymograph with gaussian noise, used for neural network training, and a 20x300 pixel movie with 300 frames for benchmarking. The resulting kymographs and movies had an average signal-to-noise ratio of 1.2 (calculated as the average intensity of the signal, divided by the average intensity of the background). Finally, we removed tracks that overlapped for the whole duration of their lifetime.

To obtain synthetic data of complex bidirectional particle movements, we generated datasets with either 15 tracks (for benchmarking) or 30 tracks (for training) per movie. The maximum velocity was set to 3 pixels/frame, as above this velocity it became hard to manually segment tracks from kymographs. Each movie was assigned a random velocity noise factor between 0 and 1.5 pixels/frame, a random switching probability between 0 and 0.1 (to switch between stationary and directed movement) and a random velocity flipping factor between 0 and 0.1 (to flip the direction of the velocity). Individual particles were simulated by first calculating their lifetime from an exponential distribution with scale 0.001. Then, a random initial state, moving or stationary, was selected as well as a random initial velocity and a particle size between 1-6 pixel. In the simulation, particles could randomly switch between different modes of movement (stationary/directed), flip velocities and were constantly subjected random velocity noise (movie specific). Finally, tracks that were occulted by other tracks were removed, and a movie (used for benchmarking) and a kymograph (used for training) were generated. The resulting kymographs and movies had an average signal-to-noise ratio of 1.4.

Benchmarking

In order to benchmark the performance of software and manual predictions, we implemented a custom track F1 score which was calculated as the geometric mean of track recall and track precision. To calculate track recall, each ground truth track was first compared to its corresponding predicted track, and the fractional overlap between them was calculated. Since predicted tracks do not necessarily follow the exact same route through a kymograph, but frequently show small deviations from the ground truth (see Figure 3 and Figure 3-figure supplement 1) we allowed for a 3.2-pixel deviation from the ground truth (2 diagonal pixels). The maximum fractional overlap was then selected and stored as the track recall. The recall was thus 1 when the full length of a ground truth track was predicted, and 0 if the track was not found in the prediction. We would like to highlight that this criterion is very strict: if a ground truth track is predicted to be 2 tracks (for example, by failing to bridge a gap along the track), the recall fraction would decrease by up to 50%, even if most of the pixels are segmented correctly and belong to predicted tracks.

Track precision was calculated by finding the largest ground truth track that corresponded, i.e. had the largest overlap, to each prediction, and then calculating the fraction of the predicted track that overlapped to the ground truth track. Therefore, a track precision of 1 corresponded to a predicted track that was fully part of a ground truth track while a precision of 0 meant that the predicted track was not found in the ground truth. In general, increasing precision leads to a lower recall and vice versa, so that taking the track F1 score as the geometric mean between the two is a good measure of overall prediction performance.

To quantify gap performance, we searched for track segments within 3 pixels of the gap for each frame, to allow for predictions that deviated slightly from the ground truth. Once each frame of the gap was assigned to a corresponding predicted segment, the gap was deemed resolved. If one or more frames of the gap had no overlapping segment to the prediction, the gap was labelled unresolved. Our synthetic tracks had 954 gaps in the 10 kymographs of unidirectional data, and 840 gaps in the 10 kymographs of bidirectional data, and the largest gap size was 6 pixels. For each kymograph, we then calculated the fraction of gaps resolved.

To quantify KymoButler performance on crossings, we first generated binary images for each ground truth track and calculated overlaps with other ground truth tracks by multiplying those images with each other. The resulting images had white dots wherever two tracks crossed. Those dots were then dilated by a factor of 16 to generate circles and overlaid with the original single-track binary image to generate binary maps that contain segments of ground truth tracks that cross/merge with other tracks. Next, we generated dilated (factor 1) binary maps for each predicted track and multiplied them with each of those cross segments to obtain the largest overlapping track for each segment. We then visually inspected a few examples and determined that an overlap of 70% corresponds to a correctly resolved crossing and allowed for slight variations in predicted tracks when compared to ground truth. Finally, we calculated the fraction of crossings resolved per kymograph.

All statistical analysis was carried out in MATLAB (http://mathworks.com).

Module performance evaluation

To benchmark the unidirectional segmentation module of KymoButler, we generated 10 synthetic movies of the dynamics of particles that move with uniform speed and do not change direction as described in the section about synthetic data generation. We then imported these movies into ImageJ (http://imagej.nih.gov) via the Kymograph Clear package (Mangeol et al. 2016), drew a profile by hand and generated kymographs from them. These kymographs were then imported into the KymographDirect software package (also (Mangeol et al. 2016)), Fourier filtered and thresholded to extract individual particle tracks. This approach required manual selection of the threshold for each individual kymograph. We additionally traced the same kymographs by hand in ImageJ to compare software performance to expert analysis. To find a suitable range of binarization thresholds for our unidirectional segmentation module we calculated the track wise F1 score on the 10 kymographs for thresholds between 0.05 and 0.5 (Figure 1-figure supplement 4). We observed the highest scores between 0.1 and 0.3 for both our synthetic data and other unpublished kymographs and also deemed these thresholds best by visual inspection of predicted kymograph tracks. Hence, we chose 0.2 as the segmentation map threshold to benchmark our predictions at.

In order to benchmark the bidirectional segmentation module and the decision module we generated 10 synthetic movies of the dynamics of complex bidirectional particles. These movies were imported into ImageJ with the KymographClear package and kymographs extracted. We subsequently tried to use the edge detection option in KymographDirect to extract individual tracks but were unable to obtain meaningful tracks (Figure 3-figure supplement 1). We also tried other options in the package but could not get good results on our synthetic data without substantial manual labor for each kymograph, defeating the goal of a fully automated analysis. Therefore, we wrote a custom script to carry out automated bidirectional kymograph analysis. We experimented with a few different approaches (for example fourier-filtering and customized edge detection) and settled on wavelet coefficient filtering as it gave the highest F1 score on our test dataset. This algorithm applied a stationary wavelet transformation with Haar Wavelets (Mathematica wavelet package) to each kymograph to decompose the image into different coefficient images that highlight different details (for example vertical or horizontal lines). We then selected only those coefficient images that recapitulated particle traces in our synthetic kymographs. These images are overlaid and thresholded with an optimized threshold to generate binary maps that can be iteratively thinned to obtain a skeletonized “trackness” map similar to the outputs of our segmentation modules. This map was then traced with the same algorithm as in our decision module. However, while the KymoButler decision module used a neural network to predict path crossings, the wavelet filtering algorithm performed simple linear prediction by taking the dilated (factor 1) binary segment of a track and rotating it by 180 degrees. Then the “prediction” was multiplied with the skeletonized trackness map and the largest connected component selected as the future path. In contrast to the original decision module, this approach does not yield any information about decision “confidence”. Thus, to resolve track overlaps at the end of the algorithm, we randomly assigned each overlap to one track and deleted them from the others. Note that the wavelet approach was heavily optimized on our synthetic kymographs and performed poorly on generic real kymographs. We also traced the same 10 kymographs by hand in ImageJ. To find a suitable range of binarization thresholds for our bidirectional segmentation module we calculated the track wise F1 score for thresholds between 0.05 and 0.5 (Figure 1-figure supplement 4) and observed the same optimal range as the unidirectional segmentation module (0.1-0.3) for both our synthetic data and other unpublished kymographs. Hence, we chose 0.2 as the threshold score to benchmark our predictions.

Key resources table

Resource	Designation.	Source.	Identifiers.	Additional Information.
Software,	MATLAB	MATLAB	RRID:SCR_0	Used for statistical

algorithm			01622	analysis
Software, algorithm	Fiji	Fiji is Just ImageJ (https://fiji.sc)	RRID:SCR_002285	Used to generate and analyse kymographs with KymographClear/Direct https://sites.google.com/site/kymographanalysis/
Software, algorithm	Wolfram Mathematica	Wolfram Mathematica	RRID:SCR_014448	Code available under https://github.com/deepmirror/KymoButler

589

Acknowledgements

We would like to thank Eva Pillai for scientific input, proofreading, and logo design, Hannes Harbrecht for fruitful discussions about neural networks; Hendrik Schuermann, Ishaan Kapoor, and Kishen Chahwala for help with kymograph tracing; and the Mathematica stack exchange community (https://mathematica.stackexchange.com) without whom this project would have taken several decades longer. Unpublished kymographs to train KymoButler were provided by Caroline Bonnet (neurocampus, University of Bordeaux), Dr. Jean-Michel Cioni (San Raffaele Hospital, Milan), Dr. Julie Qiaojin Lin (University of Cambridge), Prof. Leah Gheber and Dr. Himanshu Pandey (Ben-Gurion University of the Negev), Dr. Carsten Janke and Satish Bodakuntla (Insitut Curie, Paris), and Dr. Timothy O'Leary and Adriano Bellotti (University of Cambridge). Additionally, we would like to thank the Bordeaux Imaging Center, part of the national infrastructure France BioImaging (ANR-10INBS-04-0), for valuable feedback on our software. We would also like to thank eLife and PLOS whose open access policy enabled us to show a variety of kymographs in this manuscript. The authors acknowledge funding by the Wellcome Trust (Research Grant 109145/Z/15/Z to M.A.H.J.), the Herchel Smith Foundation (Fellowship to A.D.), Isaac Newton Trust (Research Grant 17.24(p) to K.F.), UK BBSRC (Research Project Grant BB/N006402/1 to K.F.), and the ERC (Consolidator Award 772426 to K.F.).

Competing Interests

We launched deepmirror.ai as a platform to promote the use of AI-based technologies for biological data analysis. We will be publishing tutorials and sample code to help people get started with developing their own machine learning software. We also intend to publish our work on KymoButler and future publications of our AI-based software on the website. All of this will be free of charge and available to all. Further in the future, we plan to also start offering paid professional services for customers that want to set up custom AI-based software for applications, in case they are not covered by our research. This software may or may not be made available on deepmirror.ai, depending on our clients' requests.

Software

Quick and easy cloud platform: http://www.kymobutler.deepmirror.ai

Mathematica notebook with examples on how to use the software offline: https://github.com/deepmirror/KymoButler

References

Alexandrova, A.Y. et al., 2008. Comparative Dynamics of Retrograde Actin Flow and Focal Adhesions: Formation of Nascent Adhesions Triggers Transition from Fast to Slow Flow C.-P. Heisenberg, ed. *PloS one*, 3(9).

Applegate, K.T. et al., 2011. plusTipTracker: Quantitative image analysis software for the measurement of microtubule dynamics. *Journal of Structural Biology*, 176(2), pp.168–184.

Babich, A. et al., 2012. F-actin polymerization and retrograde flow drive sustained PLCγ1 signaling during T cell activation. *The Journal of Cell Biology*, 197(6), pp.775–787.

Barry, D.J. et al., 2015. Open source software for quantification of cell migration, protrusions, and fluorescence intensities. *The Journal of Cell Biology*, 209(1), pp.163–180.

Bates, R. et al., 2017. Extracting 3D Vascular Structures from Microscopy Images using Convolutional Recurrent Networks. *arXiv.org*, cs.CV.

Chan, C.E. & Odde, D.J., 2008. Traction Dynamics of Filopodia on Compliant Substrates. *Science*, 322(5908), pp.1687–1691.

Chenouard, N. et al., 2010. Curvelet analysis of kymograph for tracking bi-directional particles in fluorescence microscopy images. In 2010 17th IEEE International Conference on Image Processing (ICIP 2010). IEEE, pp. 3657–3660.

Dai, J. et al., 2016. R-FCN: Object Detection via Region-based Fully Convolutional Networks. pp.379–387.

del Castillo, U. et al., 2015. Interplay between kinesin-1 and cortical dynein during axonal outgrowth and microtubule organization in Drosophila neurons V. Allan, ed. *eLife*, 4, p.e10140.

Faits, M.C. et al., 2016. Dendritic mitochondria reach stable positions during circuit development. *eLife*, 5, p.e11583.

Falk, T. et al., 2019. U-Net: deep learning for cell counting, detection, and morphometry. *Nature methods*, 16(1), pp.67–70.

Florian, F. et al., 2017. Gp-Unet: Lesion detection from weak labels with a 3D regression network.

Guerrero-Pena, F.A. et al., 2018. Multiclass Weighted Loss for Instance Segmentation of Cluttered Cells. *arXiv.org*, cs.CV, pp.2451–2455.

Jaqaman, K. et al., 2008. Robust single-particle tracking in live-cell time-lapse sequences. *Nature methods*, 5(8), pp.695–702.

Kingma, D.P. & Optimization, J.B.A.A.M.F.S., *DP Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv: 1412.6980*,

Koseki, H. et al., 2017. Selective rab11 transport and the intrinsic regenerative ability of CNS axons. *eLife*, 6, p.5546.

Lazarus, J.E. et al., 2013. Dynactin subunit p150(Glued) is a neuron-specific anti-catastrophe factor. D. Pellman, ed. *PLoS biology*, 11(7), p.e1001611.

LeCun, Y. et al., 2008. Backpropagation Applied to Handwritten Zip Code Recognition. *dx.doi.org*, 1(4), pp.541–551.

Lee, B.H. & Park, H.Y., 2018. HybTrack: A hybrid single particle tracking software using manual and automatic detection of dim signals. *Scientific reports*, 8(1), p.212.

Mangeol, P., Prevo, B. & Peterman, E.J.G., 2016. KymographClear and KymographDirect: two tools for the automated quantitative analysis of molecular and cellular dynamics using kymographs. *Molecular biology of the cell*, 27(12), pp.1948–1957.

Mathis, A. et al., 2018. Markerless tracking of user-defined features with deep learning. *arXiv.org*, cs.CV.

Mukherjee, A. et al., 2011. Automated kymograph analysis for profiling axonal transport of secretory granules. *Medical Image Analysis*, 15(3), pp.354–367.

Neumann, S. et al., 2017. KymoAnalyzer: a software tool for the quantitative analysis of intracellular transport in neurons. *Traffic*, 18(1), pp.71–88.

Racine, V. et al., 2007. Visualization and quantification of vesicle trafficking on a three-dimensional cytoskeleton network in living cells. *Journal of microscopy*, 225(Pt 3), pp.214–228.

Reis, G.F. et al., 2012. Molecular motor function in axonal transport in vivo probed by genetic and computational analysis in Drosophila. Y. Zheng, ed. *Molecular biology of the cell*, 23(9), pp.1700–1714.

Ronneberger, O., Fischer, P. & Brox, T., 2015. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Lecture Notes in Computer Science. Cham: Springer, Cham, pp. 234–241.

Sbalzarini, I.F. & Koumoutsakos, P., 2005. Feature point tracking and trajectory analysis for video imaging in cell biology. *Journal of Structural Biology*, 151(2), pp.182–195.

Szegedy, C. et al., 2014. Going Deeper with Convolutions. *arXiv.org*, cs.CV.

Tanenbaum, M.E., Vale, R.D. & McKenney, R.J., 2013. Cytoplasmic dynein crosslinks and slides anti-parallel microtubules using its two motor domains. *eLife*, 2, p.e00943.

Twelvetrees, A.E. et al., 2016. The Dynamic Localization of Cytoplasmic Dynein in Neurons Is Driven by Kinesin-1. *Neuron*, 90(5), pp.1000–1015.

Weigert, M. et al., 2017. Content-Aware Image Restoration: Pushing the Limits of Fluorescence Microscopy. *bioRxiv*, p.236463.

Zala, D. et al., 2013. Vesicular glycolysis provides on-board energy for fast axonal transport. *Cell*, 152(3), pp.479–491.

Figure legends

Figure 1: Kymograph generation and KymoButler

(A) Schematic of kymograph generation from live imaging data. A cell and four particles are shown at 3 different timepoints (top row). A temporal projection of this cell highlights how each particle moves along a stationary path. It is possible to track the path (magenta line), and then extract the intensity of the particle in subsequent frames in a 2D kymograph image, where the horizontal and vertical axes represent space and time, respectively. Individual lines in a kymograph represent several particles moving along the same path. **(B)** Functionality of KymoButler. A kymograph, here the motion of mitochondria along neuronal dendrites adapted from (Faits et al. 2016), is uploaded via drag & drop to the cloud interface at http://www.kymobutler.deepmirror.ai, where the noise-dependent sensitivity can be manually adjusted. The outputs are: an overlay highlighting all the tracks found in different (random) colours, a .csv file with the time and space coordinates for each track, and a .csv file containing the summary of the direction and velocity of each track. **(C)** KymoButler image outputs from two example kymographs. Left: dynamics of fluorescently labelled Rab11a in rat cortical axons (adapted from (Koseki et al. 2017), bidirectional movement as Rab11a can move both ways in the axon or become stationary). Right: dynamics of fluorescently labelled microtubule plus-ends in mouse dorsal root ganglion axons (adapted from (Lazarus et al. 2013), unidirectional movement since microtubule growth is continuous). The top row depicts the raw kymographs as taken from the published manuscripts. The middle row shows the identified tracks as dilated coloured lines. The bottom row depicts an overlay of the raw kymograph with the KymoButler prediction. Further examples from published work are shown in Figure 1-figure supplement 1A.

Figure 1-figure supplement 1: Example kymographs and software workflow

(A) Three example kymographs from published manuscripts. Example 1: *In vitro* dynamics of single cytoplasmic dynein proteins adapted from (Tanenbaum et al. 2013). Example 2: EB1-

GFP labelled growing microtubule plus-ends in mouse dorsal root ganglion axons (Lazarus et al. 2013). Example 3: Mitochondria dynamics in mouse retinal ganglion cell dendrites (Faits et al. 2016). Each dilated coloured line depicts an identified track. **(B)** KymoButler software workflow. First, a classification module is applied to each kymograph to determine whether the kymograph is unidirectional or bidirectional. If the kymograph is deemed unidirectional the unidirectional segmentation module is applied to the image to generate two trackness maps that assign each pixel a score between 0-1, approximating the likelihood that this pixel is part of a track with negative slope (left image) or positive slope (right image). Subsequently, the trackness maps are binarized, skeletonised, and segmented into their respective connected components. Finally, those components are averaged over each row to generate individual tracks, and a dilated representation of each track is plotted in a random colour. If the kymograph is classified as bidirectional, another segmentation module is applied to the kymograph, which generates a trackness map that does not highlight any particular slope. This map is binarized with a user-defined threshold and subsequently skeletonised, resulting in a binary map that exhibits multiple track crossings. To resolve these crossings, we first apply a morphological operation that detects the starting points of tracks in the binary map (red dots). Then, the algorithm tracks each line from its starting point until a crossing is encountered. At each crossing, the decision module is called, whose inputs are (i) the raw kymograph in that region, (ii) the previous track skeleton, and (iii) all possible tracks in that region. The decision module then generates another trackness map that assigns high values to the most likely future path from the crossing. This map is then again binarized and thinned with a fixed threshold of 0.5. If the predicted path is longer than 2 pixels, the path tracking continues. Once all starting points have been tracked until an end (either no prediction or no further pixels available), the algorithm again looks for starting points in the skeletonised trackness map excluding the identified tracks, and repeats the steps outlined above until all pixels are occupied by a track. The resulting tracks are then drawn with each track in a random colour.

Figure 1-figure supplement 2: The software modules in detail

(A) The class module. This module resizes any input kymograph to 64x64 pixels. It subsequently applies two convBlocks with no padding and 64 output feature maps to the image. ConvBlocks comprise a convolutional layer with 3x3 kernels followed by a BatchNormalisation Layer and a leaky **Re**ctified **L**inear **U**nit (ReLU) activation function (leak factor 0.1). The convBlocks are followed by 2x2 max pooling to halve the feature map sizes. This is repeated another 2 times while steadily increasing the number of feature maps until the last convBlock generates 256 feature maps of size 9x9. These maps are then pooled with a final 2x2 max pool operation followed by a 4x4 mean pool operation to generate a vector of 256 features. These features are then classified with a fully connected layer with output nodes followed by another leaky Ramp and finally another fully connected layer generates 2 output values that correspond to the probability of being a unidirectional/bidirectional kymograph. **(B)** The unidirectional segmentation module takes and an input kymograph of arbitrary size. Subsequently two convBlocks with 64 output feature maps are applied to the image followed by max pooling. This is repeated three times while doubling the number of feature maps with each pooling operation forming the “contracting path”. To obtain an image of the same size as the input image the small feature maps at the lowest level of the network have to be deconvolved 4 times each time halving the number of feature maps and applying further convBlocks. After each 2x2 deconvolution the resulting feature maps are catenated with the feature maps of the same size from the contracting path so that the network only learns residual alterations of the input image. The final 64 feature maps are linked to two independent convolutional layers that generate outputs that correspond to the trackness scores for positive and negative sloped lines. **(C)** The bidirectional segmentation module has the same architecture as the unidirectional one but only generates one output that corresponds to the trackness map for any lines in the image. **(D)** The decision module architecture is the same as the bidirectional segmentation module but takes three input images instead of one.

Figure 1-figure supplement 3: Synthetic training data examples

(A) Class module training data consisted of 64x64 pixel images that were either classified as unidirectional (example 1) or bidirectional (example 2). **(B)** Synthetic training data for the unidirectional segmentation module comprised 300x300 pixel kymographs with two binary ground truth maps, corresponding to particle motion with negative and positive slopes. **(C)** Synthetic bidirectional segmentation module training data comprises 300x300 pixel kymographs with only one ground truth image containing all ground truth tracks. **(D)** The decision module was trained with 48x48 pixel image crops of the raw kymograph, the previous skeletonised path, and all the skeletonised paths in the cropped region. The ground truth is simply the known future segment of the given path.

Figure 1-figure supplement 4: Geometric mean of track recall and precision for different trackness thresholds

(A) 10 synthetic unidirectional and bidirectional kymographs were analysed with varying trackness thresholds, and recall and precision were calculated. The geometric mean of recall and precision does not exhibit much variation between 0.1 and 0.3 but decreases at lower and higher values.

Figure 2: Benchmark of KymoButler against unidirectional synthetic data

(A) An example synthetic kymograph and its corresponding ground truth, manual control, the prediction by KymoButler, and the prediction by Fourier filtering. The top row depicts individual tracks in different colours and the bottom row shows the prediction overlay (magenta) with the ground truth (green) for all approaches. Discrepancies are thus highlighted in magenta (false positive) and green (false negative), while matching ground truth and prediction appears white. **(B)** Schematic explaining the concept of recall and precision. The top row depicts the possible deviations of the prediction from the ground truth.

The middle and bottom rows show example overlays, again in green and magenta, from the synthetic data. In the left column, the prediction is larger than the ground truth (magenta is visible) leading to false positive pixels and low track precision, but a small number of false negatives and thus high track recall. An example prediction overlay of the Fourier filter approach is shown, which tends to elongate track ends. The right column shows a shorter prediction than the ground truth, leading to green segments in the overlay. While this prediction has high track precision (low number of false positive pixels), track recall is low due to the large number of false negatives. Again, a cut-out from the Fourier filter prediction is shown, where multiple gaps are introduced in tracks, thus severely diminishing track recall (see Material and Methods for a detailed explanation of recall and precision). The middle column shows the same two cut outs analysed by KymoButler. No magenta or green segments are visible, thus leading to high recall and precision. **(C)** Synthetic kymograph region with four gaps highlighted (arrow heads): in one or more kymograph image rows the signal was artificially eliminated but kept in the ground truth to simulate real fluorescence data. While KymoButler efficiently connects tracks over gaps, the Fourier filter is unable to do so and breaks up those tracks into segments or incorrectly shortens these tracks (red arrow heads). Yellow arrow heads depict correct gap bridging events. **(D)** A synthetic kymograph with several line crossings. While KymoButler efficiently resolved all crossings, i.e. lines that cross other lines are not broken up into two segments, the Fourier filter correctly identifies the line crossing at the yellow arrow head but erroneously terminates the red and yellow tracks at the red arrow head. **(E)** The geometric means of recall and precision ("track F1 score") for KymoButler, the Fourier filter approach, and manual control. Each dot represents the average track F1 score of one synthetic kymograph ($p = 4 \cdot 10^{-5}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.6$, manual vs Fourier Filtering $p = 3 \cdot 10^{-3}$). **(F)** Quantification of gap bridging performance for KymoButler (89%), manual control (88%), and Fourier filter (72%); lines: medians of all 10 synthetic kymographs, $p = 10^{-4}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.9$, manual vs

Fourier Filtering $p = 2 \cdot 10^{-3}$. **(G)** The fraction of correctly identified crossings for KymoButler, manual annotation, and the Fourier filter (88% KymoButler, 86% manual, 60% Fourier filter; lines: medians of all 10 synthetic kymographs, $p = 10^{-4}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.9$, manual vs Fourier Filtering $p = 1 \cdot 10^{-3}$).

Figure 3: Benchmark of KymoButler against complex bidirectional synthetic data

(A) Example synthetic kymograph and its corresponding ground truth, manual control, the prediction by KymoButler, and the prediction via wavelet coefficient filtering. The top row depicts individual tracks in different colours and the bottom row shows the prediction overlay (magenta) with the ground truth (green) for all approaches. Discrepancies are highlighted in magenta (false positive) and green (false negative), while the match of ground truth and prediction appears white. **(B)** Example recall and precision of KymoButler and wavelet filtering. While KymoButler shows high recall and high precision, the wavelet filter approach yields significant deviations from the ground truth (green and magenta pixels). **(C)** Synthetic kymograph region with three artificial gaps highlighted (arrow heads). While KymoButler efficiently connects tracks over gaps, the wavelet filter is unable to do so and breaks up those tracks into segments (red arrow heads). The yellow arrow heads depict correct gap bridging events. **(D)** A synthetic kymograph with several line crossings. While KymoButler efficiently resolved all crossings, i.e. lines that cross other lines are not broken up into segments, the wavelet filter only resolves one crossing correctly (yellow arrow head). **(E)** The geometric means of track recall and track precision (track F1 score) for KymoButler, manual control, and the wavelet filter. Each dot represents the average F1 score of one synthetic kymograph ($p = 8 \cdot 10^{-5}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.7$, manual vs wavelet filtering $p = 10^{-4}$). **(F)** Quantification of gap performance for KymoButler, manual annotation, and wavelet filter ($p = 3 \cdot 10^{-4}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.4$, manual vs wavelet filtering

$p = 2 \cdot 10^{-4}$). **(G)** The fraction of resolved crossings for KymoButler, manual control, and the wavelet filter ($p = 3 \cdot 10^{-5}$, Kruskal-Wallis Test, Tukey post-hoc: manual vs KymoButler $p = 0.4$, manual vs wavelet filtering $p = 2 \cdot 10^{-5}$). KymoButler identifies tracks in complex kymographs as precisely as manual annotation by an expert.

Figure 3-figure supplement 1: Performance of different skeletisation techniques on a synthetic bidirectional kymograph

(A) Example of a synthetic bidirectional kymograph and its corresponding ground truth, the predictions by manual annotation, KymoButler, wavelet coefficient filtering, and tracks detected through edge filtering. The top row depicts individual tracks in different colours and the bottom row shows the prediction overlay (magenta) with the ground truth (green) for both approaches. Discrepancies are highlighted in magenta (false positive) and green (false negative), while a match of ground truth and prediction appears white.

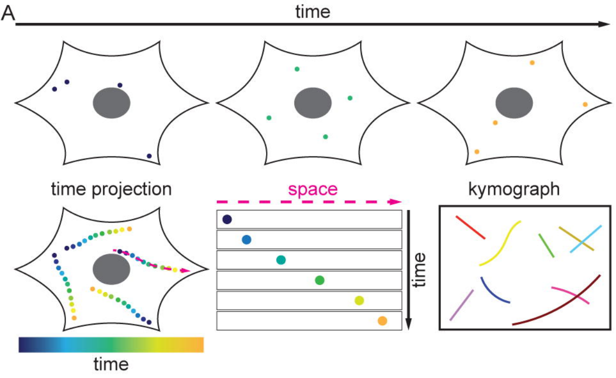
Figure 2-source data 1: Table of presented data. A CSV file that contains: the average track F1 score, the average gap score, and the average crossing score for each unidirectional synthetic kymograph.

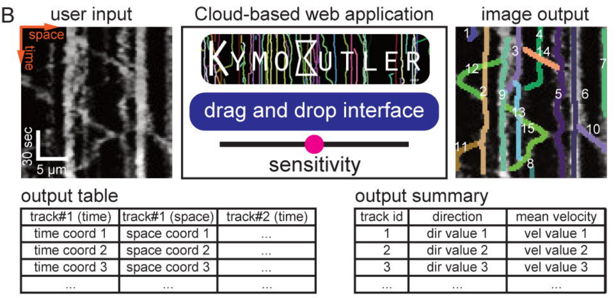
Figure 2-source data 2: Synthetic kymographs and movies. A ZIP file containing all analysed synthetic unidirectional movies, their kymographs, results from KymographClear based analysis and manually annotated ImageJ rois.

Figure 3-source data 1: Table of presented data. A CSV file that contains: the average track F1 score, the average gap score, and the average crossing score for each bidirectional synthetic kymograph.

Figure 3-source data 2: Synthetic kymographs and movies. A ZIP file containing all analysed synthetic bidirectional movies, their kymographs, and manually annotated ImageJ rois.

Figure 1





output table

track#1 (time)	track#1 (space)	track#2 (time)
time coord 1	space coord 1	...
time coord 2	space coord 2	...
time coord 3	space coord 3	...
...	...	...

output summary

track id	direction	mean velocity
1	dir value 1	vel value 1
2	dir value 2	vel value 2
3	dir value 3	vel value 3
...	...	...

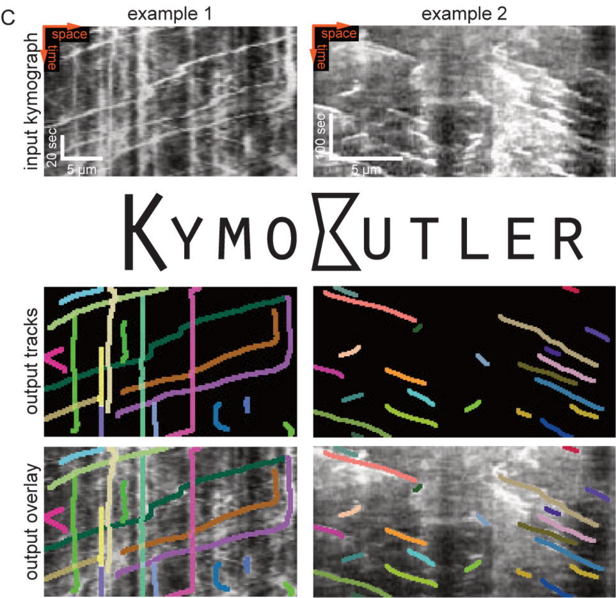
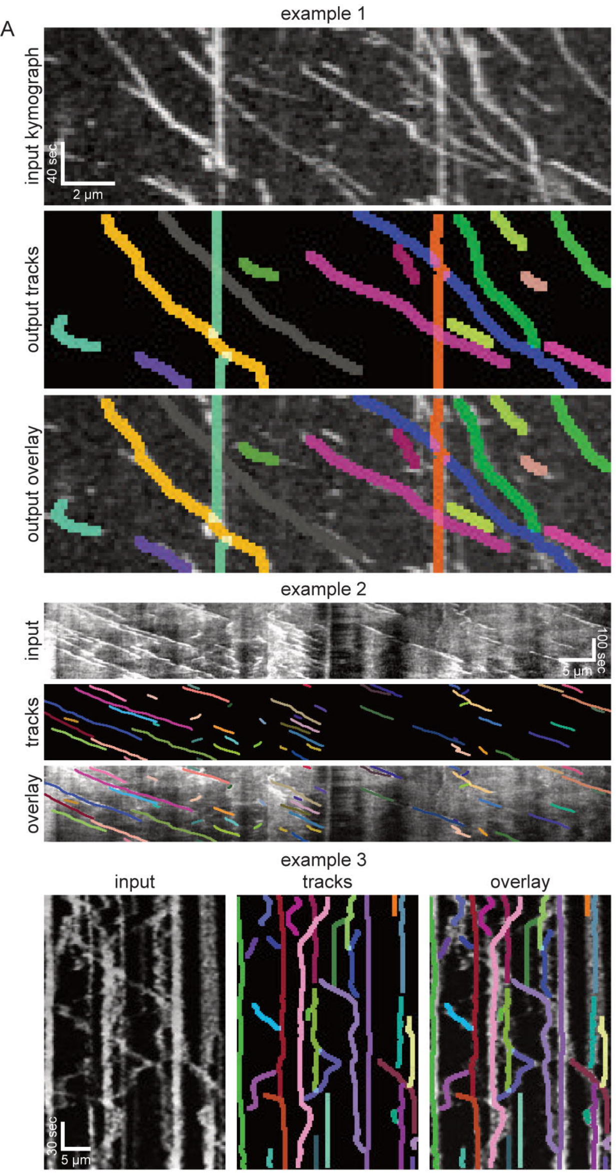


Figure 1 S1


A
example 1
input kymograph
40 sec
2 µm
output tracks
output overlay
example 2
input
100 sec
5 µm
tracks
overlay
example 3
input
tracks
overlay
30 sec
5 µm

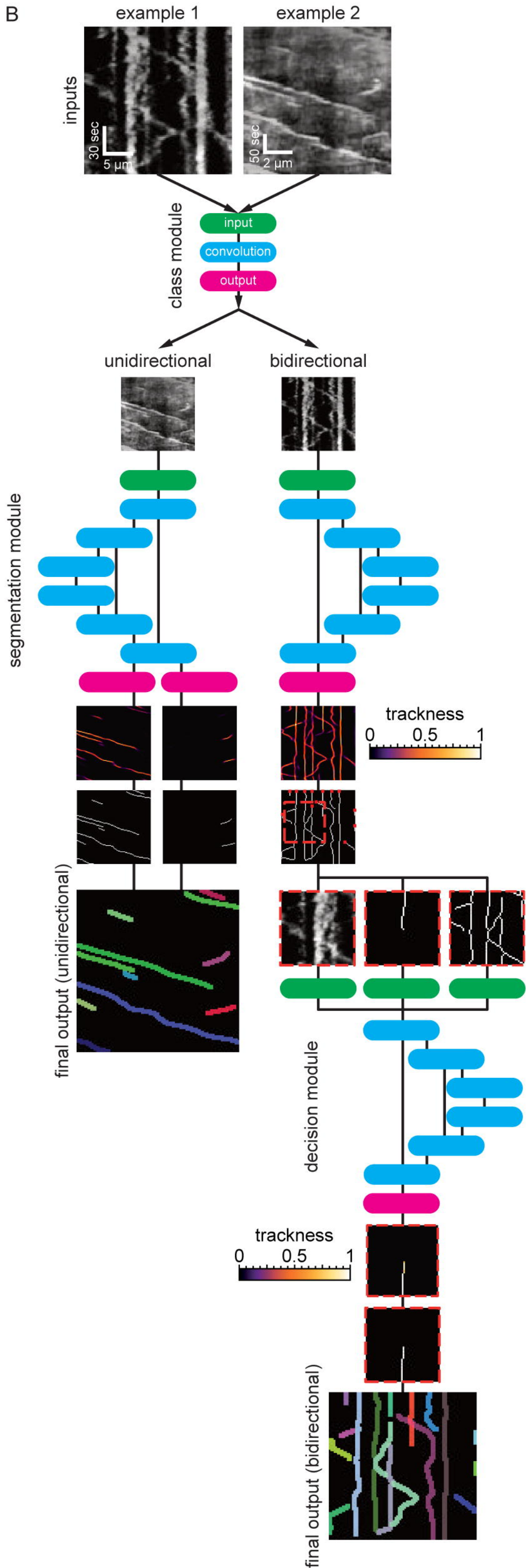

B
example 1
example 2
inputs
30 sec
5 µm
50 sec
2 µm
class module
input
convolution
output
unidirectional
bidirectional
segmentation module
trackness
0
0.5
1
final output (unidirectional)
decision module
trackness
0
0.5
1
final output (bidirectional)

Figure 1 S2

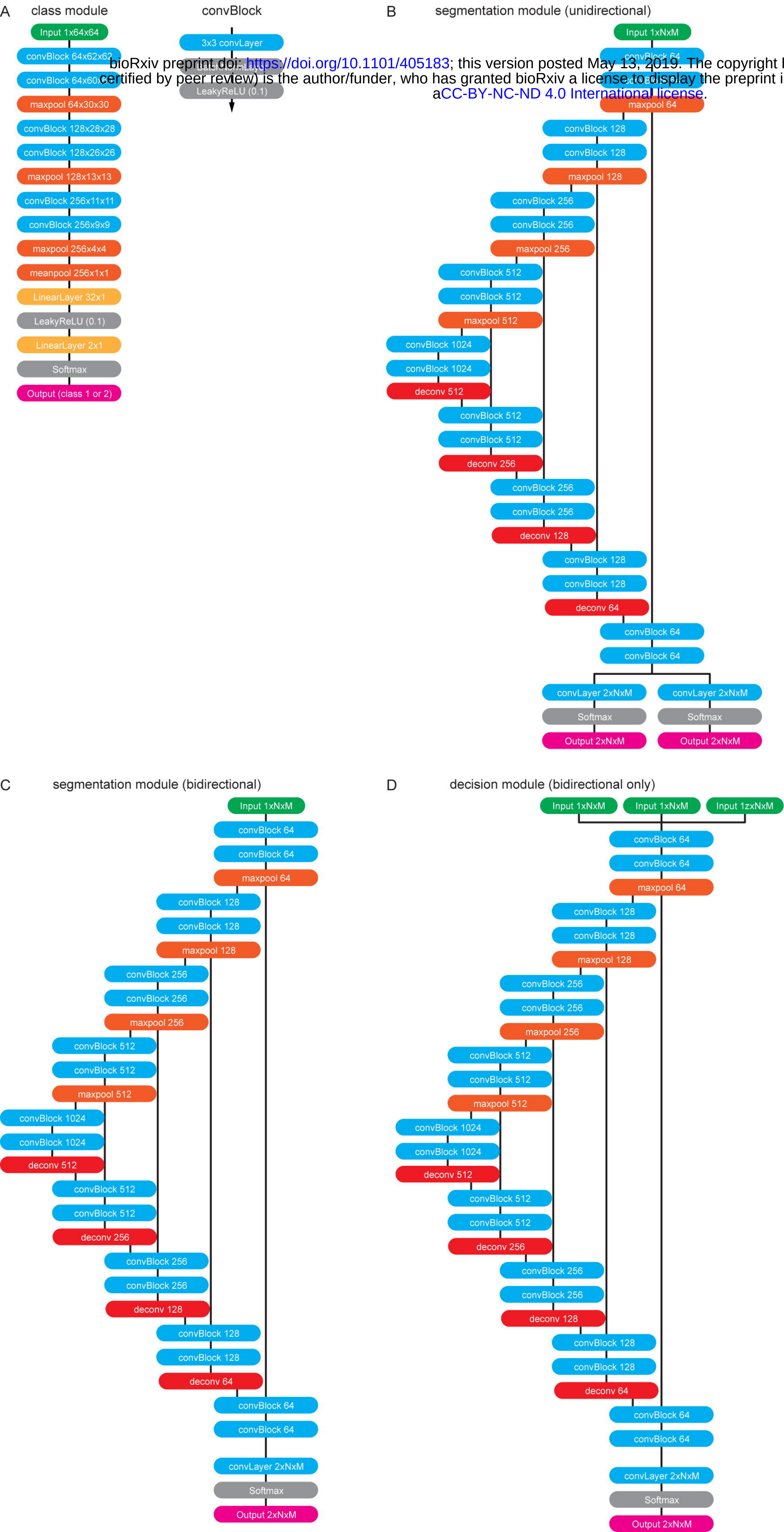

A
class module
Input 1x64x64
convBlock 64x62x62
convBlock 64x60x60
maxpool 64x30x30
convBlock 128x28x28
convBlock 128x26x26
maxpool 128x13x13
convBlock 256x11x11
convBlock 256x9x9
maxpool 256x4x4
meanpool 256x1x1
LinearLayer 32x1
LeakyReLU (0.1)
LinearLayer 2x1
Softmax
Output (class 1 or 2)
convBlock
3x3 convLayer
BatchNormLayer
LeakyReLU (0.1)
B
segmentation module (unidirectional)
Input 1xNxM
convBlock 64
convBlock 64
maxpool 64
convBlock 128
convBlock 128
maxpool 128
convBlock 256
convBlock 256
maxpool 256
convBlock 512
convBlock 512
maxpool 512
convBlock 1024
convBlock 1024
deconv 512
convBlock 512
convBlock 512
deconv 256
convBlock 256
convBlock 256
deconv 128
convBlock 128
convBlock 128
deconv 64
convBlock 64
convBlock 64
convLayer 2xNxM
convLayer 2xNxM
Softmax
Softmax
Output 2xNxM
Output 2xNxM
C
segmentation module (bidirectional)
Input 1xNxM
convBlock 64
convBlock 64
maxpool 64
convBlock 128
convBlock 128
maxpool 128
convBlock 256
convBlock 256
maxpool 256
convBlock 512
convBlock 512
maxpool 512
convBlock 1024
convBlock 1024
deconv 512
convBlock 512
convBlock 512
deconv 256
convBlock 256
convBlock 256
deconv 128
convBlock 128
convBlock 128
deconv 64
convBlock 64
convBlock 64
convLayer 2xNxM
Softmax
Output 2xNxM
D
decision module (bidirectional only)
Input 1xNxM
Input 1xNxM
Input 1zxNxM
convBlock 64
convBlock 64
maxpool 64
convBlock 128
convBlock 128
maxpool 128
convBlock 256
convBlock 256
maxpool 256
convBlock 512
convBlock 512
maxpool 512
convBlock 1024
convBlock 1024
deconv 512
convBlock 512
convBlock 512
deconv 256
convBlock 256
convBlock 256
deconv 128
convBlock 128
convBlock 128
deconv 64
convBlock 64
convBlock 64
convLayer 2xNxM
Softmax
Output 2xNxM

Figure 1 S3

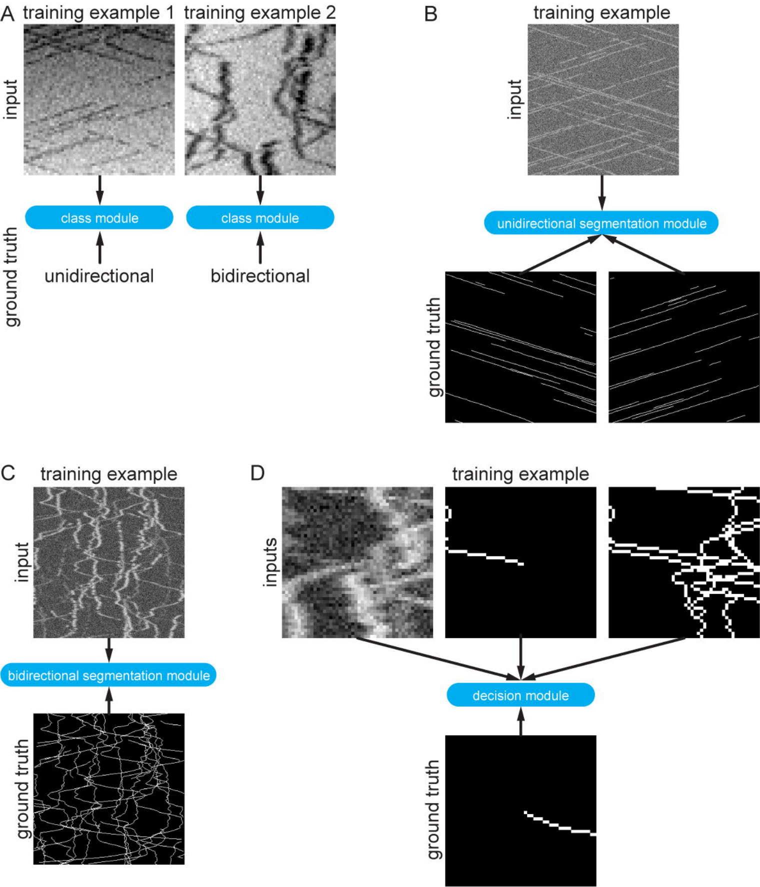

A
training example 1
training example 2
input
class module
class module
ground truth
unidirectional
bidirectional
B
training example
input
unidirectional segmentation module
ground truth
C
training example
input
bidirectional segmentation module
ground truth
D
training example
inputs
decision module
ground truth

Figure 1 S4

A

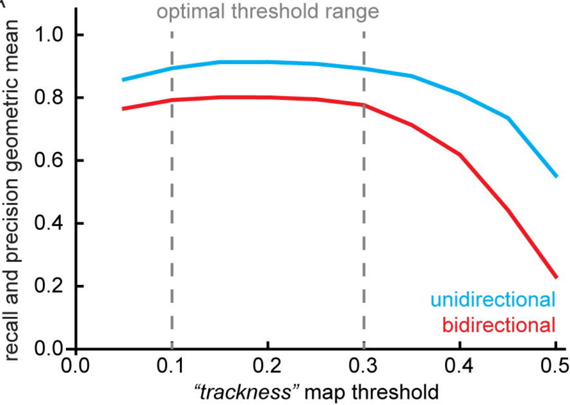

optimal threshold range
recall and precision geometric mean
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.1
0.2
0.3
0.4
0.5
"trackness" map threshold
unidirectional
bidirectional

Figure 2

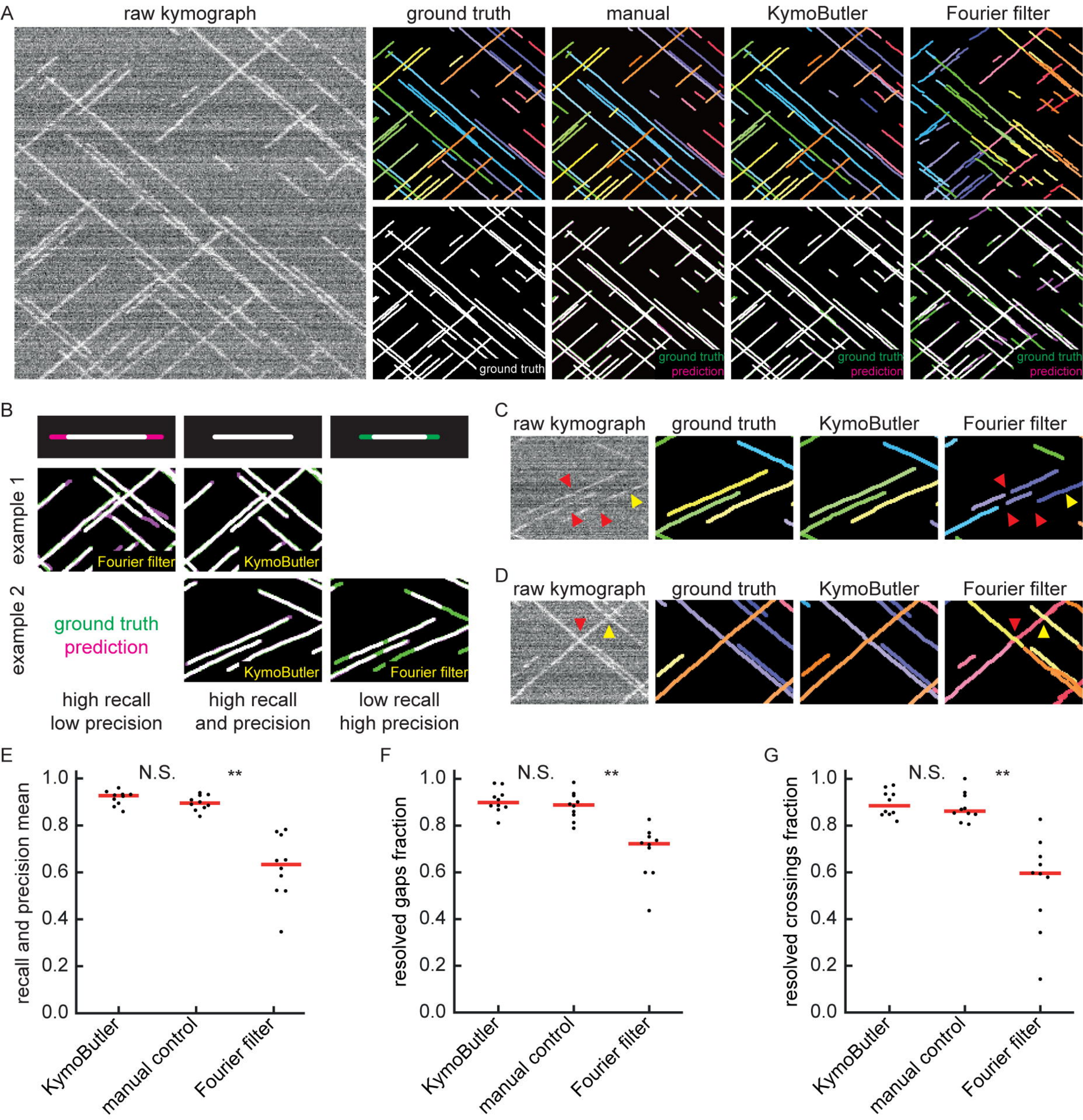

A
raw kymograph
ground truth
manual
KymoButler
Fourier filter
ground truth
ground truth
prediction
B
example 1
example 2
Fourier filter
KymoButler
KymoButler
Fourier filter
ground truth
prediction
high recall low precision
high recall and precision
low recall high precision
C
raw kymograph
ground truth
KymoButler
Fourier filter
D
raw kymograph
ground truth
KymoButler
Fourier filter
E
recall and precision mean
N.S.
**
KymoButler
manual control
Fourier filter
F
resolved gaps fraction
N.S.
**
KymoButler
manual control
Fourier filter
G
resolved crossings fraction
N.S.
**
KymoButler
manual control
Fourier filter
0.0
0.2
0.4
0.6
0.8
1.0

Figure 3

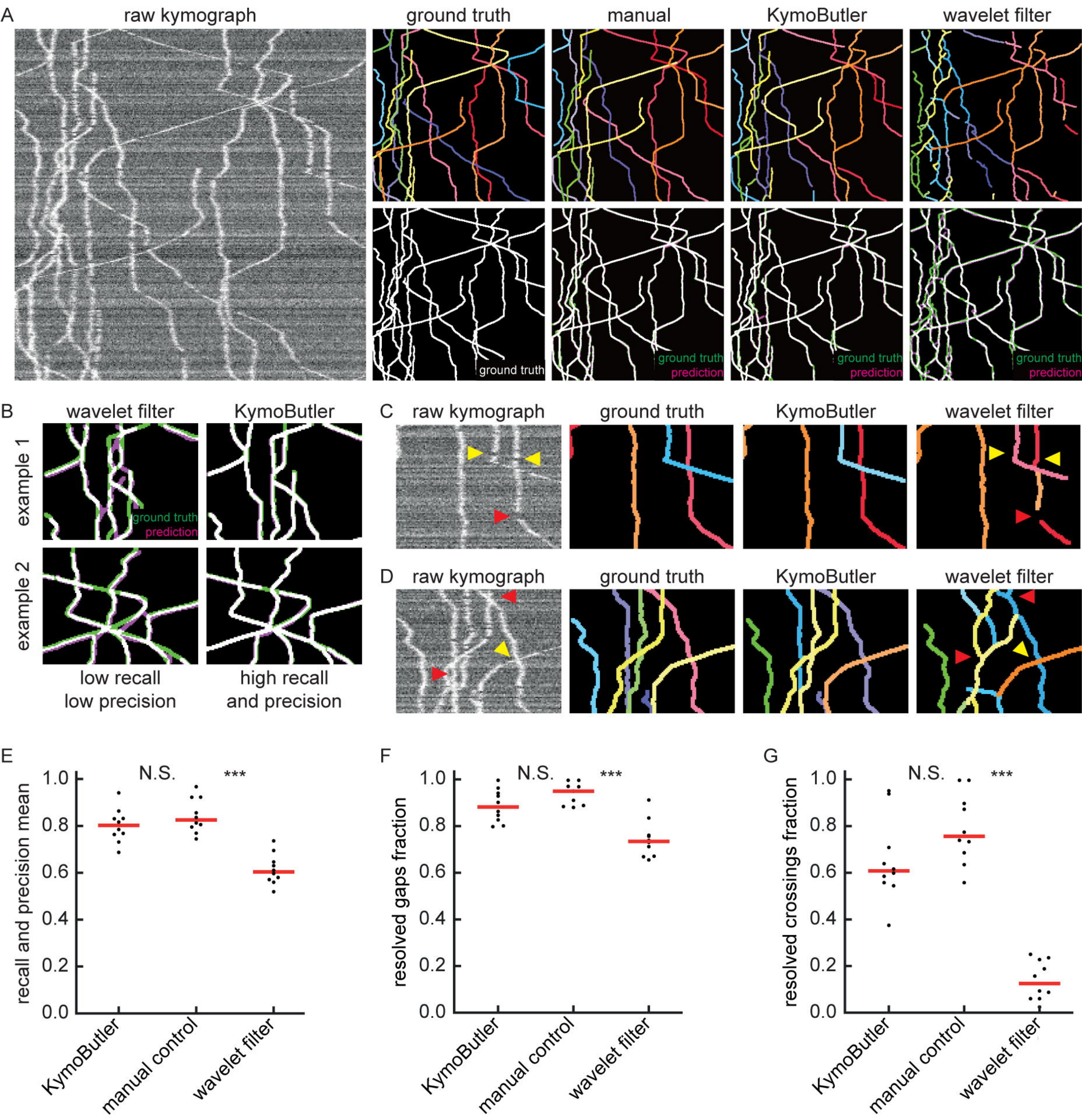
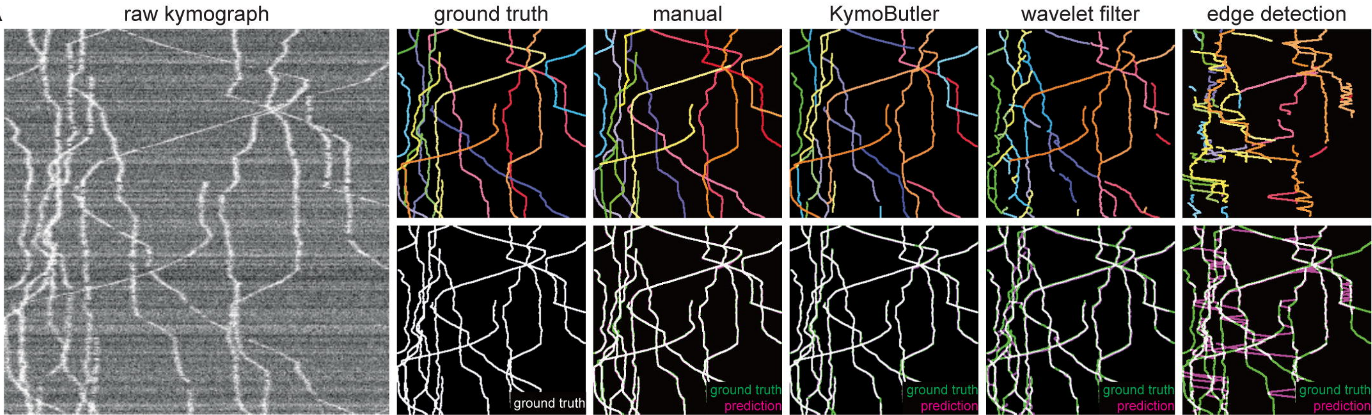

A
raw kymograph
ground truth
manual
KymoButler
wavelet filter
ground truth
ground truth
prediction
B
wavelet filter
KymoButler
example 1
example 2
ground truth
prediction
low recall
low precision
high recall
and precision
C
raw kymograph
ground truth
KymoButler
wavelet filter
D
raw kymograph
ground truth
KymoButler
wavelet filter
E
recall and precision mean
N.S.
***
KymoButler
manual control
wavelet filter
F
resolved gaps fraction
N.S.
***
G
resolved crossings fraction
N.S.
***
0.0
0.2
0.4
0.6
0.8
1.0

Figure 3 S1

A



raw kymograph
ground truth
manual
KymoButler
wavelet filter
edge detection
ground truth
ground truth
prediction
ground truth
prediction
ground truth
prediction
ground truth
prediction