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

Knowledge about the dynamics of cytoskeletal proteins is key to understanding numerous active cellular processes. However, quantifying cytoskeletal dynamics is challenging. Current tracking algorithms often require human supervision and are less accurate than manual analysis, which on the other hand is time-consuming and prone to unconscious bias. We here developed and trained KymoButler, a deep neural network to trace dynamic processes in kymographs, which are graphical representations of spatial position over time. We demonstrate that KymoButler performs at least as well as manual tracking and outperforms currently available automated tracking packages. Additionally, we successfully applied KymoButler to a variety of different kymograph tracing problems. Finally, the network was packaged in a web-based "one-click" software for use by the wider scientific community. Our approach significantly speeds up data analysis, avoids unconscious bias, and represents a step towards the widespread adaptation of Artificial Intelligence techniques in biological data analysis.

Introduction

In eukaryotic cells, biopolymers such as microtubules and actin filaments (F-actin) provide structural support and enable essential cellular functions including intracellular transport [1,2], cell motility [3], and cell division [4,5].

Both microtubules and F-actin are polar filaments with a +end and a –end which differ in their chemical and dynamical properties. Microtubules, for example, exhibit a mostly stable -end, while the +end undergoes rapid cycles of growth and shrinkage [6]. Measurements of microtubule dynamics are usually performed by genetically expressing fluorescent proteins that preferentially bind to the filament ends, such as the +End-Binding protein 1 (EB1) [7,8]. These fluorescent proteins (particles) are recorded using time-lapse fluorescence microscopy and tracked with a variety of approaches.

Since actin and microtubules can only grow along their own axis, it is possible to visualise and simplify filament end tracking by using kymographs [9,10] - 2D images generated by stacking the intensity profile along a given line, e.g. the F-actin or microtubule axis, for each time point of a movie. Thus, kymographs are length-time images showing labelled filament ends as lines (**Fig. 1**). They are not limited to tracking cytoskeletal filaments but have been widely employed to visualise biological processes across different length scales, ranging from single molecule to cell tracking [11,12].

Kymographs provide an elegant solution to the visualisation and quantification of particle dynamics. In contrast to most currently available tracking software, which faces the difficult computational problem of identifying corresponding particles in different frames, a kymograph visualises this problem, and only requires the tracing of lines in an image, a much simpler task for humans and machines alike. These lines then represent the track of a filament, or any other process, so that measuring the lines' lengths and slopes allows to calculate the average velocities and growth periods of a cytoskeletal filament, respectively.

Conventional kymograph tracing or particle tracking algorithms produce acceptable results when applied to images with a high signal-to-noise ratio (SNR), but are exceedingly error-prone at lower SNRs [10,13]. While immunocytochemical stains may result in high quality images with high SNR, live-cell imaging as required for the investigation of dynamic processes usually suffers from autofluorescence, limited light exposure, and the low labelling densities required to keep the cells undamaged. The resulting lower quality images often require cumbersome manual error corrections, leading to similar time commitments as an exclusively manual analysis. Thus, the problem of automatically, and reliably, tracking dynamic processes in live cells is still largely unresolved, and any automation in kymograph tracing that preserves the accuracy of manual annotation would represent a significant improvement in the experimental workflow.

In recent years, Artificial Intelligence (AI), and particularly Deep Neural Networks, have been very successfully introduced to data processing in biology [14,15]. AI-based image analysis has several advantages over other approaches: it is less biased than human users, takes a shorter time to analyse immense datasets, and most importantly, comes closer to human performance than conventional tracking algorithms [14]. Most AI approaches to image analysis utilise **F**ully **C**onvolutional Deep Neural **N**etworks (FCNs) that were shown to excel at object detection in images [16-18]. A convolutional neural network is able to use a multitude of hidden layers to apply kernels of all shapes and sizes to images, filtering the information from the background. This ability should, in theory, enable an FCN to trace biopolymer dynamics in low SNR kymographs with unmatched precision.

Here we developed a stand-alone software, ‘KymoButler’, which is based on an FCN, to automatically and reliably extract biopolymer dynamics from kymographs. Whilst we trained the FCN on microtubule +end growth dynamics using manually traced kymographs of EB1-GFP in neurons, the KymoButler software performs well on kymograph data of cytoskeletal filaments in other cells, including EB3-GFP traces from mitotic HeLa cells and actin speckles in *Aplysia* neuronal growth cones. Finally, the KymoButler outperforms conventional automated tracking and, quite remarkably, several cases of manual tracing.

Results

Generation of training data, neural net training, and validation

Microtubules constitute a prevalent fraction of the filaments contained in growing neuronal axons [19]. To generate kymographs capturing microtubule filament dynamics, we cultured neurons dissociated from *Drosophila melanogaster* larvae, expressing EB1-GFP under the endogenous *eb1* promoter, and tracked the dynamics of EB1 puncta in 520 axons (**Fig. 1A**). In this model system, EB1-GFP puncta move in the axon either towards the cell body (retrograde) or away from the cell body (anterograde). We generated kymographs by manually tracing the axon and stacking the intensity profile along the axon for each frame into one image (**Fig. 1B-C**). In these kymographs, individual EB1-GFP trajectories are visually distinguishable as bright lines. We traced these trajectories by hand and colour-coded them by directionality (anterograde or retrograde, **Fig. 1D**), creating a dataset of input images (the raw kymographs) and labels (the traces).

We then used these pairs of input-label images to train an FCN to separate pixels belonging to an EB1-GFP trace from background pixels. We built a custom neural network based on *Google's* "inception" architecture, the *Tracer FCN* [17] (**Methods** and **Figure 1–figure supplement 1**). Additionally, we designed a much faster, shallower FCN that only takes a 10% of the evaluation time of the Tracer FCN while maintaining similar levels of performance in our system (**Figure 1–figure supplement 2)**. Both FCNs take the input kymograph and decompose it into several images, called feature-maps, through numerous convolution and deconvolution steps. The final output is an image of the same size as the input image, in which each pixel value corresponds to the probability *p* of this pixel being part of the foreground (part of a trace). The nets were trained to recognise traces going from the left to the right. Applying them to the original and the vertical mirror image allows to distinguish between anterograde and retrograde traces, respectively.

We split our dataset into a validation set and a training set, by randomly selecting two biological repeats with a total of 33 (~6%) kymographs as validation data. The training dataset was used to iteratively change the FCN parameters to match the FCN output to the manually traced lines (see **Methods**). This was done by minimising loss (a function that quantifies the difference between the desired image and the FCN output) through stochastic gradient descent and changing the network's parameters accordingly. The training of the FCN stops when changing the parameters does not lead to any further decrease of the loss (**Figure 1–figure supplement 2**). The validation data set was simultaneously used to quantify how the FCN performed using a previously unseen dataset.

Trained FCNs assign the probability of being part of a trace to each pixel in the input image (**Fig. 1F)**. To convert these probability maps into tracks and compare them to the manual data, we introduced a threshold value *t*: any pixel that had a larger value than *t* was classified as being part of a track. The resulting binary image was then iteratively thinned so that only traces with a width of one pixel remained, which was subsequently overlaid on the manual data for comparison (**Fig. 1G**). The trained Tracer FCN showed a precise overlay with the manual annotation in the validation data (see **Fig. 1H-I**). Often, the Tracer FCN surpassed the accuracy of manual labelling, as it was able to recognise previously unlabelled traces that were erroneously omitted.

Next, we quantified the effect that the threshold value *t* had on the output of the network by introducing a similarity score that accounts for the differences between the output of the Tracer FCN and the manual labels (**Fig. 1J**). A score of 1 would indicate a perfect overlay, while a score of 0 would indicate no matches. For small *t* (0.01) we observed frequent artefacts, for example the linking of parallel tracks. For high *t* (0.5) the predicted tracks were too short. An optimum threshold was found around *t*=0.2 (**Fig. 1J**), which was therefore used throughout this paper unless stated otherwise. The maximum similarity score we achieved was ~0.7. As the KymoButler tends to outperform and detect more traces than identified by the manual labelling (where faint or short traces are often missed), similarity was low (<1) even when automated detection was close to an optimum. These results indicated that a trained FCN is able to automate the kymograph tracing process, significantly reducing research workload and avoiding biased data analysis.

The KymoButler software package

We packaged the trained FCNs into two easy-to-use interfaces for quick and fully automated kymograph analysis: (1) a browser-based app with the shallow FCN (**Figure 1–figure supplement 1**) to quickly drag & drop individual kymographs in order to analyse them (http://kymobutler.deepmirror.ai) and (2) a simple command line python script to be used offline with the full Tracer FCN (https://github.com/MaxJakobs/KymoButler). While the Tracer FCN is preferable to precisely analyse large or more complex data sets, the web based shallow version can be used to quickly assess the feasibility of the approach with a given kymograph. In both cases, the user first has to generate a kymograph for their specific problem, with any available kymograph generator (for example the Multi Kymograph Fiji plugin (https://imagej.net), the KymographTracker package [9], or the KymographClear Fiji plugin [10]). The software then applies the FCN to the image twice (once to the original and once to the vertical mirror image), thresholds the result, applies iterative thinning, generates an overlay of predicted tracks onto the kymograph, and finally extracts and classifies each connected line as a single trace (**Fig. 2**)**.** In the software, the user can freely define the threshold parameter *t,* the probability above which a pixel is considered to be part of a trace. After the computation, which takes approximately 5-10 seconds on a conventional computer (Tracer FCN on a CPU), the KymoButler generates several files including an overlay image highlighting all the tracks found in different colours, and a CSV file per kymograph, containing all track coordinates and track directionality for post-processing.

KymoButler outperforms conventional tracking software

To assess the performance of KymoButler, we compared it to manual kymograph tracing and to the plusTipTracker package, which was explicitly written for tracking EB1-GFP puncta [13]. In conventional tracking algorithms such as the plusTipTracker, individual features are first detected through local thresholding and then linked with each other between frames. We compared the average track velocities (start-to-end velocity) and track lengths of EB1-GFP puncta of our validation data set (33 previously ‘unseen’ kymographs, **Fig. 3**) for all the three approaches. There was no significant difference between the average velocities (KymoButler: $4.6 \pm 1.0\ \mu m/min$, Manual: $4.3 \pm 0.9\ \mu m/min$, plusTipTracker: $4.8 \pm 1.4\ \mu m/min$, one way ANOVA, p=0.16, **Fig. 3A**). However, when plotting the velocities calculated by the two algorithms against manually determined data in a 2D scatter plot, 97% (32/33) of the velocities calculated by KymoButler fell within the standard deviation of the manual data ($\pm 0.9\ \mu m/min$), while this was only the case for 73% (24/33) of the velocities calculated by plusTipTracker (**Fig. 3B**).

The average track lengths revealed by manual tracing, KymoButler, and plusTipTracker differed significantly (**Fig. 3C**, $p<10^{-23}$, one way ANOVA). A post-hoc analysis showed no differences between KymoButler and manual analysis ($25 \pm 5\ sec$ and $23 \pm 4\ sec$, p=0.16, Tukey-Kramer test). However, the plusTipTracker analysis significantly underestimated the track times by about twofold ($12 \pm 2\ sec$, $p<10^{-9}$, Tukey-Kramer test) (**Fig. 3C**). Additionally, in 85% (28/33) of kymographs analysed with KymoButler, the average lengths of the traces were within the standard deviation of the manual data ($\pm 5\ sec$), but only 1 out of the 33 axons analysed with plusTipTracker fell within the same region (**Fig. 3D**).

We noticed that for one kymograph the manual tracing resulted in much larger average EB1-GFP track lengths than calculated by both KymoButler and plusTipTracker (dot 2 in **Fig. 3D**). Revisiting the manual data revealed that several short tracks were unlabelled incorrectly (black box in **Fig. 3F**). Additionally, some tracks were erroneously drawn too long, while

KymoButler broke them rightly into several shorter ones (red box in **Fig. 3F**), indicating that KymoButler performs better than manual labelling on most kymographs.

KymoButler can be easily extended to other biological systems

We finally tested the capability of the KymoButler software to analyse kymographs generated from different cell types and different cytoskeletal components. Note that we did not retrain the Tracer FCN for these applications. First, we analysed time lapse movies of EB3-GFP dynamics in interphase HeLa cells (**Fig. 4A**). After only changing the threshold parameter to t=0.1, KymoButler predicted puncta trajectories as well as it did for *Drosophila melanogaster* axon EB1-GFP. When comparing manually extracted traces with KymoButler results of raw kymograph images, we did not find any significant differences between average EB3-GFP microtubule growth velocities (Wilcoxon rank sum test, p=0.98) and average growth times (Wilcoxon rank sum test, p=0.61) **(Fig. 4B).**

Remarkably, KymoButler was even able to quantify actin speckle velocities in *Aplysia* growth cones. Average retrograde actin flow velocities showed no significant difference between manual labelling and KymoButler analysis even though the network was only trained on EB1-GFP puncta in axons (Wilcoxon rank sum test, p=0.08) **(Fig. 4D).**

Discussion

In this work, we used deep learning to optimise automated tracking of dynamic, fluorescently labelled proteins in a noisy cellular environment. Fully convolutional neural networks (Tracer FCNs) are nowadays widely applied for image recognition. Since tracking is *a priori* a visual problem, we built an FCN for identifying traces in kymographs. We deployed our network in two independent stand-alone software packages that take generic kymographs and output all traces found in the image in a matter of seconds. Remarkably, the network not only outperforms current particle tracking software and, in some cases, even manual tracking, but it also performs just as well on kymographs of different dynamic processes, such as fluorescence speckle microscopy.

Our KymoButler software has only one adjustable parameter: *t*, the threshold at which a pixel is recognised as being part of a track. For our validation data, the best value for *t* was 0.2. This threshold generally depends on the SNR of the image. If the SNR is low, the FCN is “less confident” about a given pixel, so that the threshold has to be smaller. More noisy data, such as the HeLa cell EB3-GFP data or actin speckles shown in Figure 4, produced good results with a smaller threshold value (*t*=0.1). Hence, the correct threshold has to be chosen based on each biological application and imaging conditions.

Available automated kymograph analysis software was not suitable for tracing EB1-GFP puncta in axons, mainly because these packages were susceptible to noise. The KymographDirect package, for example, applies a global threshold to individual kymographs to extract traces, thus being very prone to variations in background intensity and requiring manual screening [10]. Most other currently available packages require manual track tracing or linking, defeating the purpose of a fully automated analysis [9,20]. An alternative approach quantifies kymograph velocities through 2D autocorrelation, however, the analysis is limited as trace lengths cannot be measured [21].

The current gold standard for automated tracking of microtubule dynamics is the plusTipTracker package. When we compared KymoButler with manual and plusTipTracker data, it performed at least as well as manual tracking, and much better than the plusTipTracker. The mismatch between the plusTipTracker and manual traces is likely because (1) “long” tracks have a tendency of being split into several shorter ones, since the probability of linking errors increases with track length (**Supplementary Movie 1**), and (2) “short” tracks are sometimes incorrectly linked due to background fluctuations **(Supplementary Movie 2)**. The first issue results in too short track lengths, and the second causes inflated velocity measurements.

We propose that manual tracking is inferior to the KymoButler as it suffers from inconsistency, bias, and is overall laborious. While the KymoButler analyses each kymograph in the same way, manual tracing performance varies from one kymograph to the next as well as between users. In our dataset, we frequently overestimated trace lengths, so that the manual annotation yielded slightly larger track lengths than the KymoButler. In future, KymoButler could be trained on a larger dataset traced by multiple researchers to remove other inconsistencies that may be present in the dataset, thus further improving the KymoButler's performance.

Additionally, KymoButler was able to analyse kymographs from different dynamic processes such as retrograde actin flow in neuronal growth cones. This result highlights that particle tracking does not depend on the precise nature of the particle, e.g. actin or EB1, but on the task of tracing a line in an image, which should be the same for any dynamic process that can be represented this way.

Future work will expand our approach to 2D or even 3D tracking problems. In this paper, we drew 1D lines in 2D movies, extracted 2D (space and time) images (kymographs), and finally traced 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 image (2D space and time, or 3D space and time). The 2D/3D lines in those images can then be traced by hand and a more complex FCN trained to recognise them. This approach could yield human-like performance in higher dimensional automated tracking.

online Methods

Fly Stocks

The following stocks were used for expressing fluorescently tagged EB1: *eb1-gfp* [22] and *uas:eb1-gfp* [23]. To include different genetic backgrounds in our training data we also co-expressed two RNAi constructs: *uas:wh-RNAi* (Bloom# 35573) and *uas:dhcRNAi* (Bloom# 36698) of which the latter is known to cause a severe phenotype on EB1-GFP dynamics [24]. All *uas* constructs were driven by *elav-gal4* (Bloom# 458) and transgenic lines generated through standard balancer crossing procedures.

D. melanogaster neuronal culture and EB1-GFP live imaging

Primary cell cultures were prepared similar as to [25]. Third instar larvae were selected, and their central nervous systems dissected. Subsequently, the tissue was dissociated in Hank's Balanced Salt Solution (HBSS) supplemented with Dispase (Roche 049404942078001) and Collagenase (Worthington Biochem. LS004214). The cells were plated in 30µl droplets of Schneider's Medium (Thermo Fisher 21720024) supplemented with insulin (2 µg/ml Sigma I0516) and fetal bovine serum (1:4 Thermo Fisher Scientific A3160801). We plated the drops in ibidi glass-bottom µDishes (cat num 81158) and covered them with 25mm coverslips (VWR) to create small culture chambers. The glass bottom dishes were previously coated with Concanavalin A (5µg/ml, 1.5h at 37°C). The culture chambers were subsequently put at 26°C for 1.5h so that the cells settle on the coated surface of the dish. Then the chambers were flipped to remove debris from the surface and left for 24 hours before imaging.

Live imaging movies were acquired on a Leica DMI8 inverted microscope at 63x magnification and 26°C (oil immersion, NA=1.4). To reduce autofluorescence the culture medium was replaced with Live Imaging Solution (Thermo Fisher A14291DJ). For EB1-GFP imaging, an image was taken every 2 seconds for 70-150 frames depending on sample bleaching rate. We imaged 520 axons from 26 different dishes.

We also treated the cells with Latrunculin B (10 μM) and Ciliobrevin A (100 μM). Both drugs are known to perturb microtubule dynamics so that including movies acquired with these treatments would again make our FCN more robust [24,26]. In both cases the cells were first allowed to attach to the coated glass for 1.5h post dissection before replacing the culture medium with culture medium supplemented with Latrunculin B or Ciliobrevin A.

Aplysia neuronal culture and actin fluorescence speckle microscopy

Aplysia bag cell neurons were isolated and cultured as previously described in [27]. Neurons were then injected with alexa-568 labelled G-actin (Molecular Probes) at low levels, appropriate for fluorescence speckle microscopy [28]. The growth cone in Fig. 4B was imaged on a spinning disk confocal microscope at 2 Hz sampling rate.

HeLa Cell culture and imaging

A HeLa stable cell line expressing LifeAct-GFP and EB3-mRFP [29], was maintained in Dulbecco's Modified Eagles Medium (DMEM GlutaMAX; Gibco) supplemented with 10% FBS and 50 U/ml penicillin and 50 μg/ml streptomycin (Invitrogen) at 37 C under 5% CO2. Cells were imaged using an UltraView Vox (Perkin Elmer) spinning disc confocal microscope with a 63X (NA 1.4) oil objective equipped with temperature and CO2 controlling environmental chambers and images were acquired using a Hamamatsu ImagEM camera and Volocity software at a rate of 2 Hz (Perkin Elmer).

Kymograph generation and FCN training

The 520 neuronal axons were first traced by hand with the KymographTracker plugin for ICY (http://icy.bioimageanalysis.org, [9]). We randomly choose two biological repeats (2x dishes, 33 axons, ~6%) as a validation data set, i.e. we did only use 489 axons as training data. Subsequently we generated kymographs with the KymographTracker plugin and traced EB1-GFP lines in those images by hand, using the same plugin. The traces were then plotted in two images: one for retrograde tracks and one for anterograde tracks. We also generated kymographs with a custom Mathematica script to obtain two slightly different kymographs per axon. We then reflected each kymograph and the corresponding trace images along the vertical (y) axis and stretched them along the x-axis to 0.5, 0.75, 1.25, and 1.5 their original length eventually resulting in a total number of 10,400 kymographs and their respective manually traced images (two per kymograph). Hence our training/validation data set comprises 9740/660 kymographs and their respective trace images.

We decided to design a Fully Convolutional Neural Network (FCN) to recognise the antero- and retrograde lines in our noisy kymographs. An FCN does not exhibit any fully connected layers, i.e. layers whose parameter number depends on the dimension of the input image, but only calculates several parallel and consecutive image convolutions and/or deconvolutions with trainable parameters. As the number of these parameters does not depend on the size of the input image, kymographs do not have to be resized before application of the FCN.

We used *Mathematica* (http://wolfram.com) to both generate and train our FCN. Even though the network is fully convolutional, the *Mathematica* training algorithm needed all input images to have the same dimensions. Thus, we divided each kymograph into tiles of 80x80 pixels so that one training “unit” comprised one input image and two output images, showing anterograde and retrograde traces. To make training more efficient, we decided to only train one network to recognise anterograde (left to right) tracks so that each of these sets was again split into an input tile with the anterograde tracks and the vertically reflected input + retrograde tile. The total number of tile pairs thus became 149,488 for the training data and 9740 for the validation data. In this way the final network would have to be called twice: once on the original kymograph and once on the reflected one to detect both antero- and retrograde traces.

Our approach to the precise architecture of the final Tracer FCN was purely empirical comprising the following building blocks: (i) a convolutional layer with arbitrary kernel size and number of output channels followed by a batch normalisation layer and a ‘leaky’ ramp (leayReLU) activation function ($leayReLu(x) := max(x, 0) - 0.1\ max(-x, 0)$), (ii) a dropout layer that randomly sets 10% of all input values to zero during training to prevent ‘overfitting’ of the input data, (iii) a deconvolutional layer with arbitrary kernel size and number of output channels to sharpen the input images again followed by a batch normalisation layer and a leayReLu layer, (iv) a pooling layer with kernel size three to replace a given pixel with the maximum value in its neighbourhood. The batch normalisation layer is useful to stabilize the training procedure as it rescales inputs to the activation function (leayReLu) so that they have zero mean and unit variance. The leayReLu prevents 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 [17].

The architectures we settled on is shown in **Figure 1–figure supplement 1**. Six connected “Trace Block” layers are used to denoise the image and highlight individual traces. The precise architecture of these Blocks is again shown in **Figure 1–figure supplement 1**. This block architecture allows a lot of flexibility with the choice of operation, for example the convolving kernel size, throughout training and evaluation. A major feature of the Trace Block architecture is the inclusion of deconvolutions. Without explicitly computing deconvolutions in each block, as for example in the shallow FCN in **Figure 1–figure supplement 1**, the final image is more blurred, and one is unable to segment individual traces as efficiently. In the final step of both architectures all channels are projected on only two and a softmax layer is applied so that the sum over those channels is one for each pixel. The two channels can be interpreted as the probability of a given pixel to be part of the background or a trace.

To train the FCN we quantified the difference between the FCN output o and the desired target output t through a cross entropy loss layer ($CEloss(t,o) = -(t \cdot ln(o) + (1-t) \cdot ln(1-o)$). Here t can be either 1 (background) or 2 (trace). For Example: The untrained FCN will give 0.5 as the probability of each pixel to be part of the background as it has no preference yet. The corresponding loss for a pixel that should be part of the background (index=1) would be: $CEloss(0.5,1) = 0.69$. During training this value might be updated to 0.9 decreasing the loss to $CEloss(0.9,1) = 0.11$.

We trained the FCN through stochastic gradient descent. Here we first randomly subdivided all training tile pairs into batches of 50. For each batch we then calculated the average cross entropy loss and the gradient of this loss in all tuneable parameters, e.g. the kernel entries in the convolutions. We then updated all the parameters σ in the network according to $\sigma' = \sigma - \eta\partial_\sigma CEloss(t,o)$. Here ∂_σ denotes the partial derivative with respect to all parameters of the FCN and η is the learning rate, i.e. the multiplier giving absolute value of the shift in σ at a given step. Note that η is not fixed but is dynamically updated through the ADAM algorithm [30]. This was repeated for all batches until the whole training dataset was visited by the algorithm constituting one round. The FCN was trained until no decrease in the validation data loss was observed anymore (5 Rounds). Every 10 minutes, the average loss was calculated for the validation dataset to obtain a readout on how the FCN performs on previously “unseen” data.

FCN performance evaluation

The direct output of both FCNs was an 80x80x2 tensor that assigns each pixel the probability of being part of a trace (index=2) or the background (index=1). In order to reconstitute traces from the FCN output we introduced a threshold value *t* for the second index, above which we would consider a pixel being part of a trace. The training set comprises many more background pixels than foreground pixels so that the FCN exhibits small probabilities around traces, therefore the cut-off has to be chosen generally as an unintuitively small value (t<0.5). The thresholded output images were iteratively thinned until they depicted lines of only one pixel wide.

To compare the FCN output with the manual annotation for the validation data we defined a similarity score as a function of the threshold as follows: (i) Both the anterograde and the retrograde trace probability map are calculated with the FCN and thresholded and dilated by one pixel. (ii) Both dilated binary predictions (0=background, 1=trace) are multiplied with the respective binary manual trace images and in the resulting image the total number of pixels=1 counted ($ovlp$, a measure of the overlap between the prediction and the manual annotation). (iii) We also calculated the total number of pixels=1 in the manual traced image (N_m) and the prediction (N_p). (iv) The similarity score s was then given by:

$$s = \begin{cases} 1, & \text{if } N_m = 0 \ \& \ N_p = 0 \\ 1/N_m, & \text{elseif } N_p = 0 \\ 1/N_p, & \text{elseif } N_m = 0 \\ \frac{ovlp}{N_m\left(1+\frac{|N_m - N_p|}{N_m}\right)}, & \text{else} \end{cases}$$

In short: The similarity score measures the overlapping pixels in the prediction and the manual annotation and divides them by the absolute number of pixels being part of a trace in the manual annotation ($ovlp/N_m$). The result is divided by a factor measuring the difference in pixels that are part of a trace between prediction and manual labelling to penalise large discrepancies in total number of predicted pixels ($1 + |N_m - N_p|/N_m$). Since the prediction rarely overlaps completely with the manual annotation and frequently finds more objects that were previously labelled, a 'good' score lies at around 0.7.

KymoButler software

The KymoButler software first applies either the deep Tracer FCN or the shallow FCN to a given kymograph and its vertical reflection. The resulting foreground probability map is then thresholded with the parameter *t* and thinned iteratively so that each trace is only one pixel wide at any point. The thinned traces are then pruned by three pixels so that short branches are deleted. Subsequently, each trace is segmented and selected only if it contains more than 5 pixels and is at least 3 frames long. This step removes noise from the result. In the final step, pixels that lie in the same row of the kymograph are averaged over so that the resulting track has only one entry per frame.

Comparison between KymoButler and plusTipTracker

We used the plusTipTracker version 1.1.4 for *MATLAB* 2014a (mathworks.com) to analyse the axons from our validation dataset (33 axons). In each movie we first selected a region of interest comprising the axon and omitting very bright artefacts. To run the software, we first varied the detection parameters to find those that result in similar total track numbers as the manual kymograph tracing approach. We settled on the following detection parameters: $\sigma_1 = 1, \sigma_2 = 4, K = 8$. For tracking we chose: $minTrackLength = 3, maxGap = 2$, $minSearchRad = 5, maxSearchRad = 15, maxFwAngle = 30, maxBwAngle = 10, shrinkV = 0$, and $rFluc = 1.5$. Note that we set the shrinkage velocity to zero so that the plusTipTracker does not try to calculate microtubule shrinkage events.

In order to compare the plusTipTracker to the KymoButler we wrote a short *Mathematica* script that calculates the predicted tracks for the same 33 axons with the Tracer FCN and exports them in a *MATLAB* friendly format. As with the plusTipTracker we ignored all traces with track lengths below 3 frames. All subsequent data plotting and analysis was done in MATLAB.

Software

Quick and easy cloud platform (Shallow FCN only): http://www.kymobutler.deepmirror.ai

GitHub with the command line interface (full Tracer FCN): https://github.com/MaxJakobs/KymoButler

Acknowledgements

We would like to thank Paul Forscher for providing speckle microscopy time-lapse movies of *Aplysia* growth cones, Eva Pillai for scientific input, proofreading, and logo design, Hannes Harbrecht for fruitful discussions about FCN's; Hendrik Schuermann 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. 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.).

Author Contributions

M.A.H.J. and K.F. conceived the project; M.A.H.J. and A.D. performed experiments; M.A.H.J. and A.D. developed the software; M.A.H.J., A.D., and K.F. managed the project; M.A.H.J., A.D., and K.F. wrote the manuscript.

Competing Interests

The authors declare no competing interests.

References

1. Franker, M. A. M. & Hoogenraad, C. C. Microtubule-based transport - basic mechanisms, traffic rules and role in neurological pathogenesis. *J Cell Sci* **126,** 2319–2329 (2013).
2. Mitchison, T. J. & Cramer, L. P. Actin-based cell motility and cell locomotion. *Cell* **84,** 371–379 (1996).
3. Gardel, M. L., Schneider, I. C., Yvonne Aratyn-Schaus & Waterman, C. M. Mechanical Integration of Actin and Adhesion Dynamics in Cell Migration. *http://dx.doi.org/10.1146/annurev.cellbio.011209.122036* **26,** 315–333 (2010).
4. Prosser, S. L. & Pelletier, L. Mitotic spindle assembly in animal cells: a fine balancing act. *Nat. Rev. Mol. Cell Biol.* **18,** 187–201 (2017).
5. Lancaster, O. M. *et al.* Mitotic Rounding Alters Cell Geometry to Ensure Efficient Bipolar Spindle Formation. *Dev. Cell* **25,** 270–283 (2013).
6. Brouhard, G. J. Dynamic instability 30 years later: complexities in microtubule growth and catastrophe. *Mol. Biol. Cell* **26,** 1207–1210 (2015).
7. Piehl, M., Tulu, U. S., Wadsworth, P. & Cassimeris, L. Centrosome maturation: measurement of microtubule nucleation throughout the cell cycle by using GFP-tagged EB1. *PNAS* **101,** 1584–1588 (2004).
8. Ma, Y., Shakiryanova, D., Vardya, I. & Popov, S. V. Quantitative Analysis of Microtubule Transport in Growing Nerve Processes. *Current Biology* **14,** 725–730 (2004).
9. Chenouard, N., Buisson, J., Bloch, I., Bastin, P. & Olivo-Marin, J.-C. Curvelet analysis of kymograph for tracking bi-directional particles in fluorescence microscopy images. in 3657–3660 (IEEE, 2010). doi:10.1109/ICIP.2010.5652479
10. Mangeol, P., Prevo, B. & Peterman, E. J. G. KymographClear and KymographDirect: two tools for the automated quantitative analysis of molecular and cellular dynamics using kymographs. *Mol. Biol. Cell* **27,** 1948–1957 (2016).

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

12. Barry, D. J., Durkin, C. H., Abella, J. V. & Way, M. Open source software for quantification of cell migration, protrusions, and fluorescence intensities. *J Cell Biol* **209,** 163–180 (2015).

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

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

15. Weigert, M. *et al.* Content-Aware Image Restoration: Pushing the Limits of Fluorescence Microscopy. *bioRxiv* 236463 (2017). doi:10.1101/236463

16. Dai, J., Li, Y., He, K. & Sun, J. R-FCN: Object Detection via Region-based Fully Convolutional Networks. 379–387 (2016).

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

18. LeCun, Y. *et al.* Backpropagation Applied to Handwritten Zip Code Recognition. *http://dx.doi.org/10.1162/neco.1989.1.4.541* **1,** 541–551 (2008).

19. Kapitein, L. C. & Hoogenraad, C. C. Building the Neuronal Microtubule Cytoskeleton. *Neuron* **87,** 492–506 (2015).

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

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

22. Bulgakova, N. A., Grigoriev, I., Yap, A. S., Akhmanova, A. & Brown, N. H. Dynamic microtubules produce an asymmetric E-cadherin-Bazooka complex to maintain segment boundaries. *J Cell Biol* **201,** 887–901 (2013).

23. Jankovics, F. & Brunner, D. Transiently reorganized microtubules are essential for zippering during dorsal closure in Drosophila melanogaster. *Dev. Cell* **11,** 375–385 (2006).
24. del Castillo, U., Winding, M., Lu, W., Gelfand, V. I. & Allan, V. Interplay between kinesin-1 and cortical dynein during axonal outgrowth and microtubule organization in Drosophila neurons. *eLife Sciences* **4,** e10140 (2015).
25. Sanchez-Soriano, N. *et al.* Are dendrites in Drosophila homologous to vertebrate dendrites? *Dev. Biol.* **288,** 126–138 (2005).
26. Rao, A. N. *et al.* Cytoplasmic Dynein Transports Axonal Microtubules in a Polarity-Sorting Manner. *Cell Rep* **19,** 2210–2219 (2017).
27. Forscher, P., Kaczmarek, L. K., Buchanan, J. A. & Smith, S. J. Cyclic AMP induces changes in distribution and transport of organelles within growth cones of Aplysia bag cell neurons. *Journal of Neuroscience* **7,** 3600–3611 (1987).
28. Danuser, G. & Waterman-Storer, C. M. Quantitative fluorescent speckle Microscopy of cytoskeleton dynamics. *Annu Rev Biophys Biomol Struct* **35,** 361–387 (2006).
29. Fink, J. *et al.* External forces control mitotic spindle positioning. *Nat Cell Biol* **13,** 771–778 (2011).
30. Kingma, D. P. & Ba, J. Adam: A Method for Stochastic Optimization. (2014).

Figure legends

Figure 1: Generation of kymographs showing microtubule EB1-GFP dynamics and subsequent training of the Tracer FCN. (A) Fluorescence time-lapse images of a drosophila neuron expressing EB1-GFP. A single EB1-GFP punctum is shown in four consecutive frames (arrows). (**B)** Hand-drawn line along the axon building up each pixel row of the kymograph. **(C)** Example kymograph obtained from the line shown in (B). Arrow: track resulting from the EB1-GFP comet shown in (A). **(D)** Individual EB1-GFP traces were traced by hand, distinguished by directionality (blue = anterograde, red = retrograde), and overlaid on the kymograph. **(E)** Example output of the Tracer FCN applied to validation data (see methods). An 80x80 pixel subimage from the kymograph shown in (D) (box) is fed to the Tracer FCN. (**F**) The heat maps show the predicted probability *p* for each pixel being part of a trace (top: anterograde traces, bottom: retrograde traces). (**G**) Tracer FCN prediction: pixels were considered to be part of a track when *t*>0.2, and iterative thinning was applied to generate traces. (**H**) Hand-traced (manual) images for both directions. (**I**) the prediction (orange) was overlaid with the manual annotation (blue); co-localised pixels appear pink. The FCN fully recapitulated the hand-traced data and even recognised traces that were omitted by mistake in hand tracings, even though it had never 'seen' this image during training. **(J)** The performance of the Tracer FCN when applied to the whole validation data set in terms of a manual to Tracer FCN similarity score (see methods) plotted as a function of probability cut-offs *t*. The insets highlight the scores of the anterograde predictions of the kymograph shown in (**E**). A maximum in similarity is achieved at *t*=0.2. For larger p cut-off values the network tends to return shorter traces than the manual labelling; for smaller *t* tracks become incorrectly linked (left inset). Scale bars: 2 μm (horizontal), 25 sec (vertical).

Figure 1–figure supplement 1: FCN architecture. Left: An input 80x80 pixel image is first fed into 2 consecutive Tracer Blocks that each output 110 80x80 images (feature maps). Then a Dropout Layer deletes (randomly) 10% of all pixels in all feature maps (only during training). The result is again computed through four Tracer Blocks. Subsequently, the resulting 110 feature maps are projected on two with a 1x1 convolution, the result transposed and a softmax operation applied so that the two entries in each pixel of the 80x80 matrix sum up to 1. The result then comprises two 80x80 images: one whose pixel values give the probability of being part of the foreground (prob fg) and one whose pixel values give the probability of being part of the background (prob bg). Only convolution and deconvolution operations are used, hence the network does not depend on the input image size and can be applied to images that are not 80x80 pixels large. Right Top: One Tracer Block comprises six parallel net chains. (1) the identity convolution 1x1 with 10 output maps. (2) a 1x1 convolution followed by a 3x3 convolution with 20 output maps. (3) a 1x1 convolution followed by a 5x5 convolution with 20 output maps. (4) a 1x1 convolution followed by a 9x9 convolution with 20 output maps. (5) a 1x1 convolution followed by a 3x3 deconvolution with 20 output maps. (6) a 3x3 max pooling operation followed by a 1x1 convolution with 20 output maps. The resulting feature maps are catenated along the first dimension to generate 110 feature maps as an output of the block. Right Bottom: As this net can be computationally demanding for web form applications and hence expensive to maintain we also designed a shallower FCN: This net does not comprise any parallel blocks and only evaluates one 3x3 convolution followed by a 5x5 convolution and a 3x3 deconvolution.

Figure 1–figure supplement 2: Loss Curves for training and validation data. Top: Validation and batch Loss curves for the Tracer FCN. The FCN was trained for 5 Rounds, i.e. full dataset visitations. 50 input tiles were summed to one batch and the loss calculated on each batch (orange). Additionally, the loss on the validation data set was calculated every 10 minutes (blue dots and curve). The loss reaches a plateau after ~4 Rounds. Bottom: The Batch loss of the Tracer FCN (blue, same data as in the orange curve above) and the batch loss for the shallow FCN from Fig. S2.

Figure 2: The KymoButler package - an automated software for kymograph analysis. **(A)** The software is first presented with a kymograph image input of any format. (**B**) Subsequently, the Tracer FCN is applied to the image twice (once to the original and once to the vertical reflection) resulting in two heat maps that assign each pixel the probability of being part of an antero- or retrograde trace (top two panels). Then the software binarizes the resulting images with a user-given threshold *t* (here *t*=0.2). The binary images are then thinned iteratively, and each line gets segmented as one track (blue and red lines, bottom two panels). (**C**) The software then generates multiple output files: an overlay of the segmented tracks with the original image (shown, each colour represents a distinct track) and a CSV file per kymograph, with every trace's coordinates. Scale bars: 2 μm (horizontal), 25 sec (vertical).

Figure 3: KymoButler microtubule dynamics analysis outperforms conventional tracking algorithms. (A) Average EB1-GFP velocities per axon were similar for manual tracing, the KymoButler, and plusTipTracker package (p=0.17 ANOVA). Each dot represents one axon and the boxplots show the median and the upper and lower quantiles. **(B)** 2D scatterplot of the average velocities calculated with KymoButler (green dots) and plusTipTracker (magenta dots) against the average velocities calculated via manual tracing. Black lines indicate a deviation of $\pm 0.9 \mu m/sec$ from the identity line, corresponding to the standard deviation of the manually traced velocities. **(C)** Boxplots of the average track lengths, i.e. the time during which EB1-GFP puncta were visible, calculated with manual tracing, KymoButler, and the plusTipTracker. The average track length was approximately half as long when the plusTipTracker package is used, compared to the manual tracing and

KymoButler ($p<10^{-9}$, Tukey-Kramer test), which yielded similar results. **(D)** 2D scatter plot of the average track lengths calculated with the KymoButler (green dots) and plusTipTracker (magenta dots) against the average track lengths calculated via manual tracing. Black lines again indicate the standard deviation of the manual data. **(E)** Kymograph of data point 1 labelled in (D) with overlaid manually labelled traces and the predicted traces of KymoButler (each colour represents one segmented track). There is an excellent correspondence between the tracks obtained by both approaches. **(F)** Kymograph of data point 2 labelled in (D) with overlaid traces. KymoButler breaks up several tracks more accurately than the manual tracking (red box, long trace in the centre, red arrow) and adds several shorter tracks that were incorrectly omitted in the manual approach (black box, black arrow). Only tracks longer than 2 frames were included in the analysis. **(G)** Zoom into the red box shown in (**F**). Scale bars: 2µm (horizontal), 25 sec (vertical).

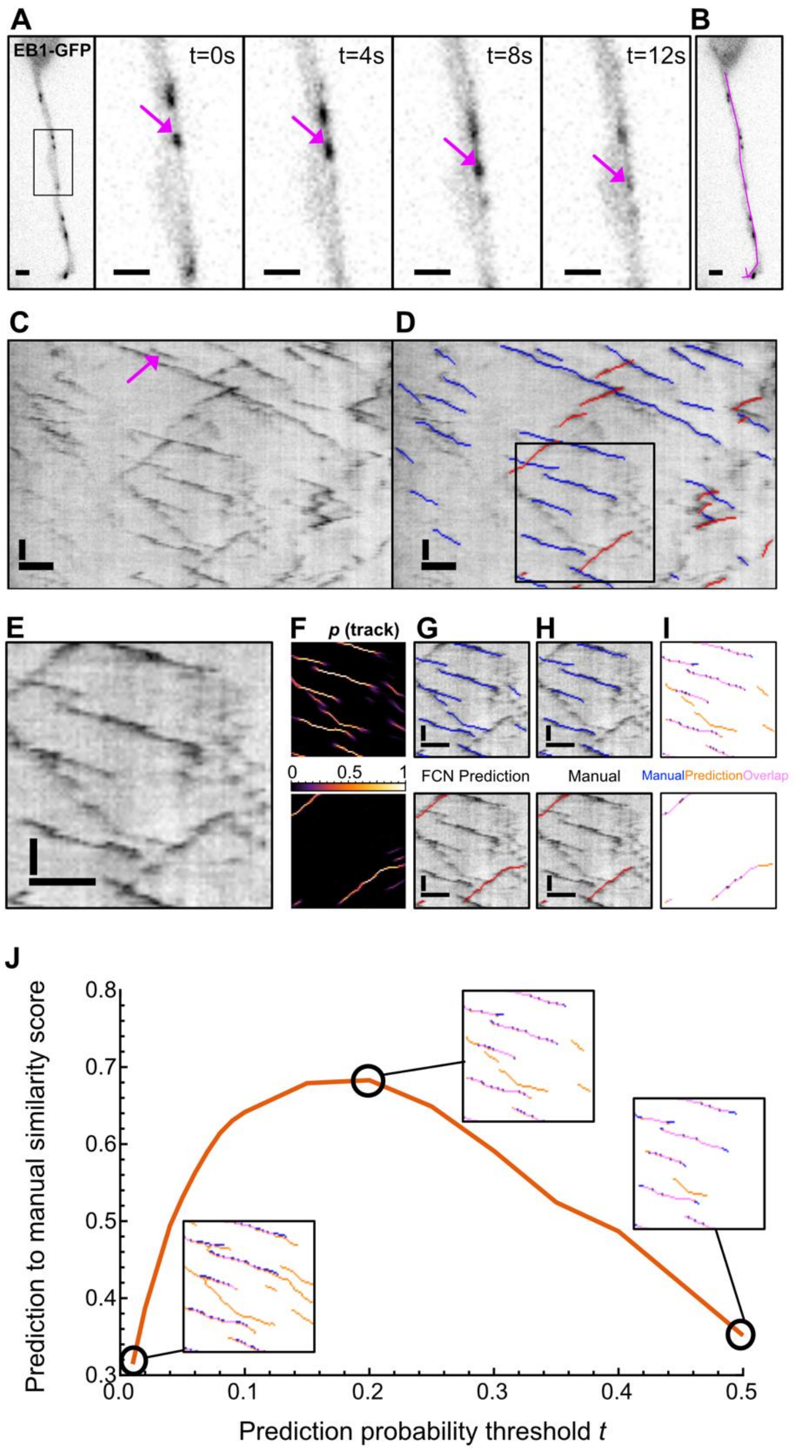
Figure 4: KymoButler efficiently analyses particle tracks in other biological systems. **(A-B)** Analysis of EB3-GFP in HeLa cells. (**A**) A kymograph was extracted from an interphase HeLa cell expressing EB3-GFP and subsequently analysed by hand and with KymoButler. The heatmap represents the probability map generated by KymoButler, the blue lines correspond to the hand traced EB3-GFP lines, and the coloured lines represent the traces recognised by KymoButler. The threshold *t* was set to 0.1. Scale bars: 5µm (horizontal), 10 sec (vertical) **(B)** Average EB3-GFP velocities and growth times obtained by manual tracing and KymoButler analysis. No significant differences were found (Wilcoxon rank sum test, p=0.98 velocities, growth times p=0.61). **(C-D)** Analysis of actin speckle dynamics in *Aplysia* growth cones. (**C**) Kymograph of fluorescently labelled G-actin, and analysed traces with *t*=0.1. Scale bars: 5µm (horizontal), 20 sec (vertical). **(D)** Average actin speckle velocities are similar for manual and KymoButler analysis (test, p=0.08). Tracks less than 6 frames long were omitted from the analysis.

Movie legends

Mov. S1: Example of an erroneously shortened EB1-GFP track. The particle is detected in the lower right corner in frame 1 (small red circle). The particle is then tracked for 7 consecutive frames (red line). While the particle does not disappear after frame 7 but rather becomes a bit fainter in frame 8 to re-appear in frame 9 in the upper left corner of the movie, the trace is finished after frame 7. The movie was generated with the plusTipTracker after detection.

Mov. S2: Example of an erroneously linked EB1-GFP track. The particle is detected in the lower right corner in frame 4 (small red circle). In frame 5, the particle moves slightly to the left and gets correctly linked. However, in frame 6, a particle appearing in the upper left corner becomes incorrectly linked to the track, increasing the estimated average velocity of the particle to ~15 µm/min, about three-fold larger than the average velocity of EB1-GFP puncta (5 µm/min, **Fig. 3**).

635 ## Figures



636

637 **Figure 1**

638

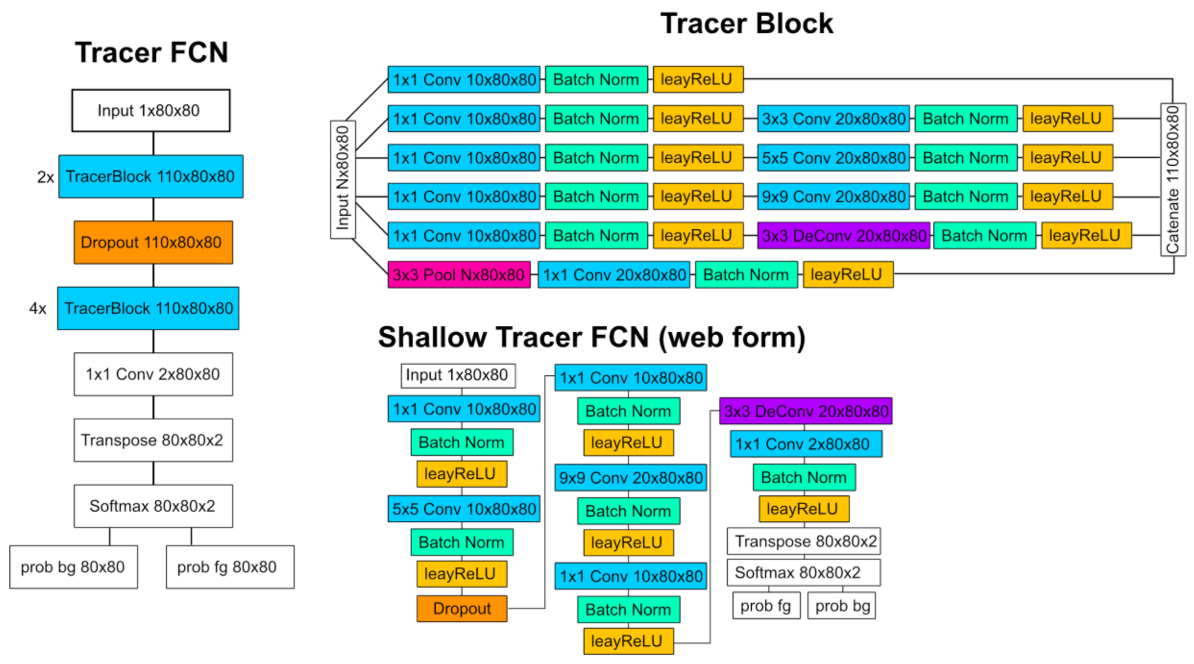


Figure 1–figure supplement 1

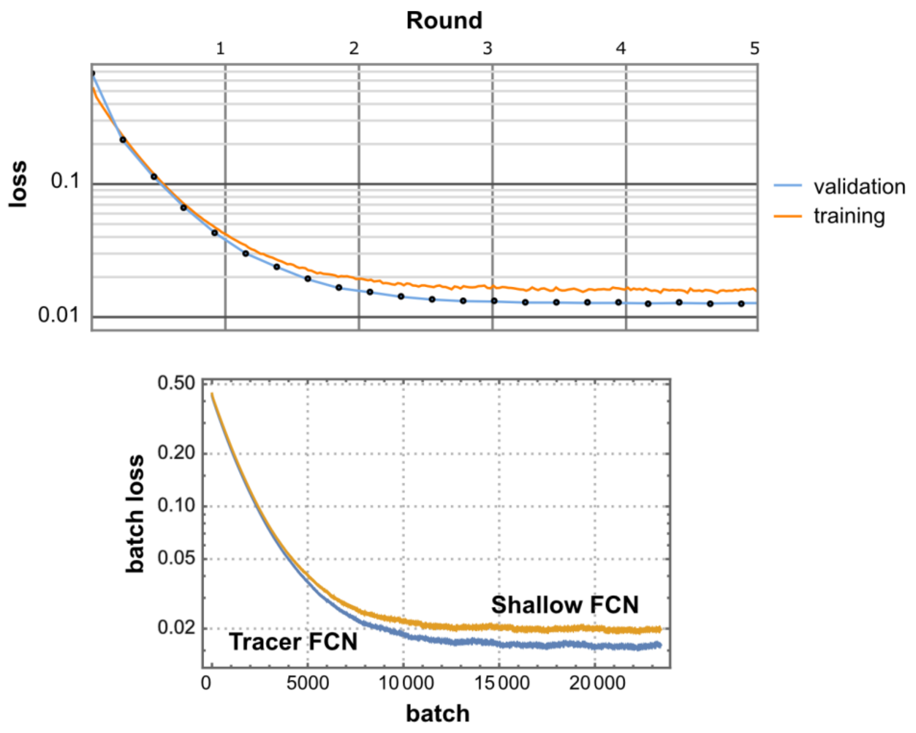
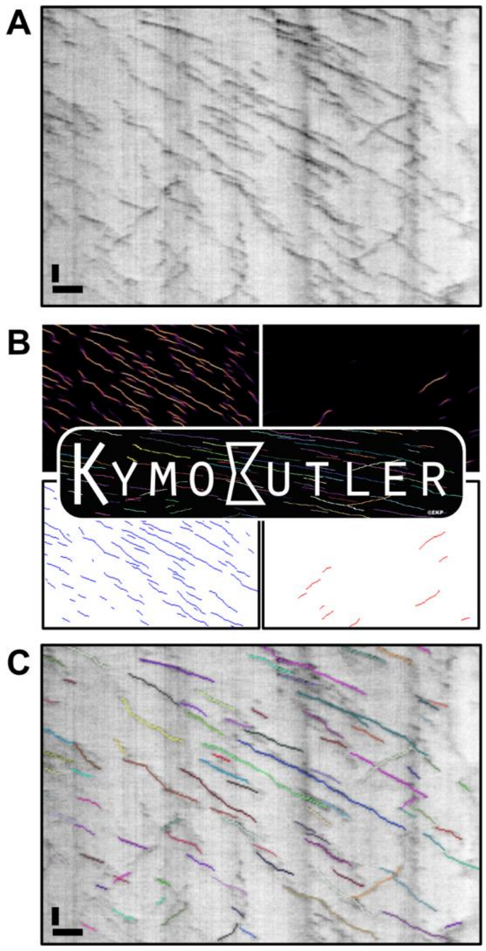


Figure 1–figure supplement 2



645

646 **Figure 2**

647

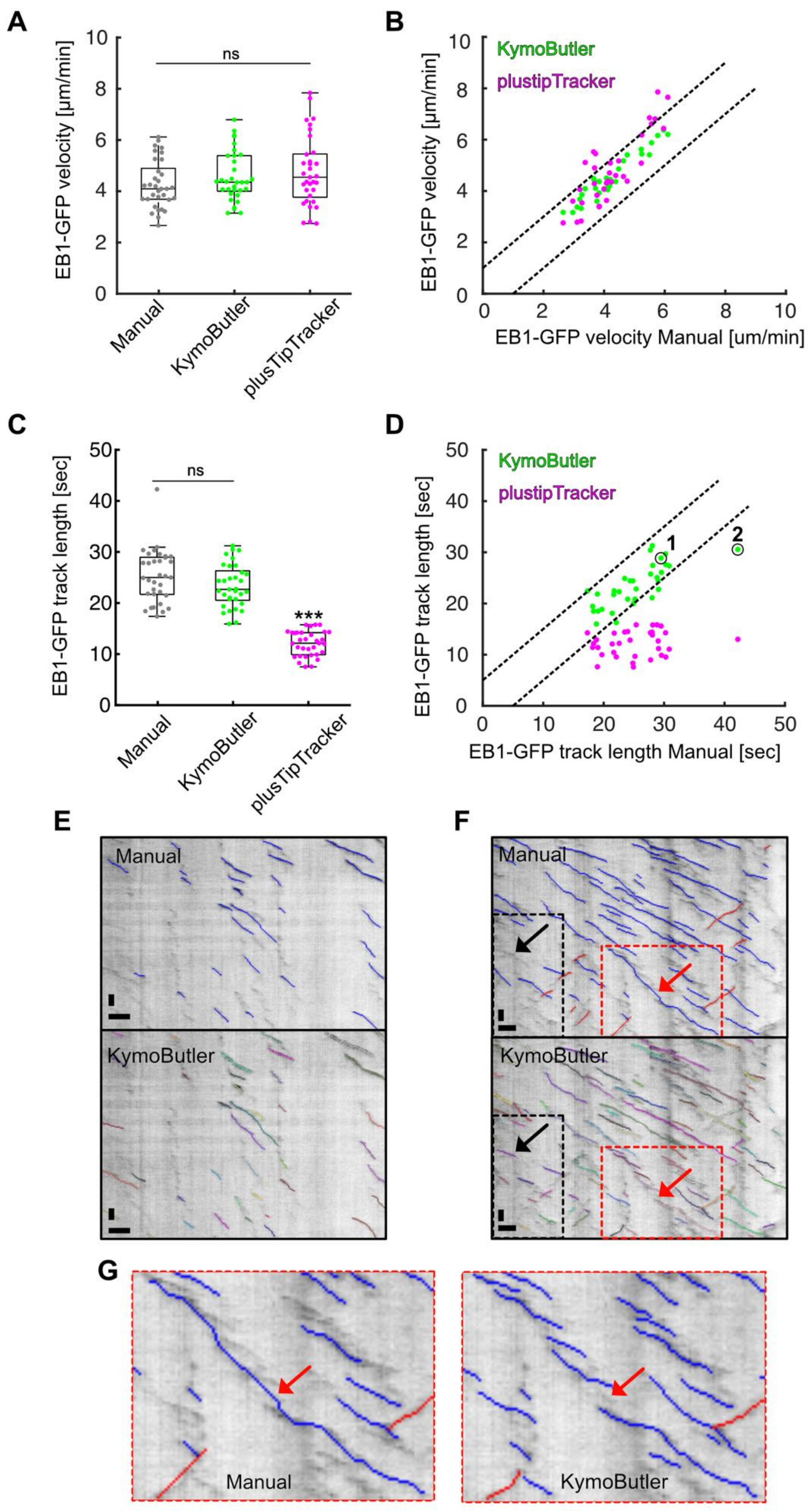
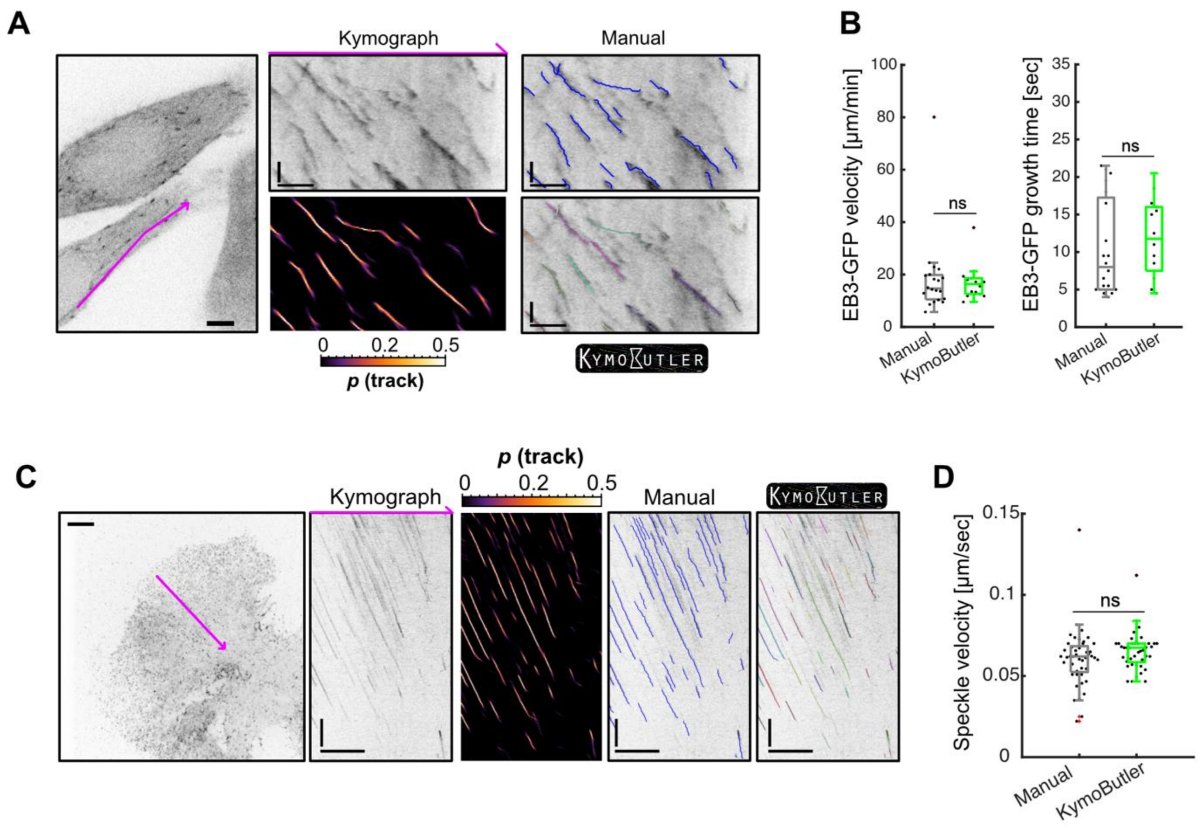

A
EB1-GFP velocity [µm/min]
ns
Manual
KymoButler
plusTipTracker
B
KymoButler
plustipTracker
EB1-GFP velocity [µm/min]
EB1-GFP velocity Manual [um/min]
C
EB1-GFP track length [sec]
ns
***
Manual
KymoButler
plusTipTracker
D
KymoButler
plustipTracker
1
2
EB1-GFP track length [sec]
EB1-GFP track length Manual [sec]
E
Manual
KymoButler
F
Manual
KymoButler
G
Manual
KymoButler

Figure 3



650

651 **Figure 4**

652