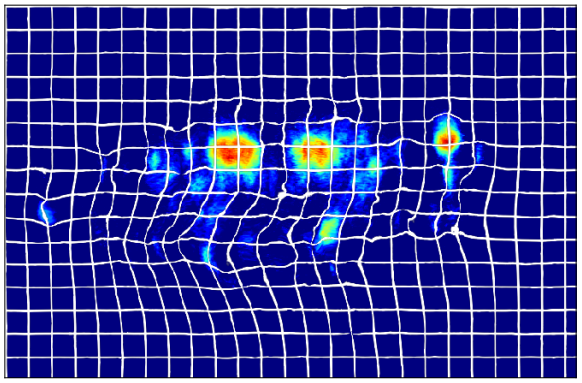
Semester Project
Report

Techniques for 2-photon Microscopy Imaging Registration



Submitted by

Adrián Sager La Ganga
MSc. in Computational Science and Engineering

Under the guidance of

Pavan Ramdya
Director, Ramdya Lab
Sibo Wang
Doctoral researcher, Ramdya Lab

November 24, 2022

Contents

Abstract

In the study relating a fly's motor behavior to its neural activity, 2-photon microscopy is the method of choice to image populations of neurons for extended time periods in tight spaces. However, during behavior the recorded neural circuitry moves and deforms, thus requiring image registration to align it. Although some image registration approaches can accurately and consistently model complex deformations, they are very computationally intensive and time-consuming. This work aims to benchmark various methods in terms of performance and efficiency, and also develop new pipeline routines for image manipulation. We introduce a new metric, COM, to detect improperly registered images, and present the earth mover's distance (EMD) as an alternative to the most commonly used similarity metrics. We also train a machine learning registration algorithm, VoxelMorph, that balances the performance and efficiency trade-off.

1 Introduction

Relating animal behavior and brain activity, also called neural decoding, is of great interest in neural engineering and for neural data analysis. In the context of the work done in [6], the ventral nerve cord (VNC) of adult *Drosophila melanogaster* is the site where some higher order decisions are decoded into specific body movements.

Unfortunately, the behaving fruit fly produces non-trivial movement artifacts in recordings of its neural activity. This is exacerbated by the fact that intensities vary across time and noise is present. Luckily, there is a vast literature in medical image registration, often focused on MRI scans, which sets as an objective aligning deformed images. Popular registration methods can generally be divided in learning-based and non-learning-based, and they both optimize a transformation according to an energy function [30]. Learning-based registration requires a dataset to train a neural network, which for this work will be several *Drosophila*'s neuron activation recordings, introduced in Section 2.1. These recordings were retrieved through 2-photon microscopy imaging of the coronal x-z image planes of the cervical connective [6], which allows us to see the axons from the ascending and descending neurons between the brain and the VNC. This was achieved through a family of protein calcium sensors for neural activity, GCaMP6 [7], and — as a useful structural marker for reference — a fluorescent protein, tdTomato [29].

Our task is that of *unsupervised* optimization, since our recordings are not annotated and we do not have ground truth deformation fields nor true stable images (although the constant activation of tdTomato will be helpful). One challenging aspect to tackle in this task is choosing the metric, as we cannot use ground truth metrics like the Dice score [9]; so, in Section 2.2 we will cover the set of unsupervised metrics we studied. Afterwards, shortcomings on some of the metrics will lead us to justify thresholding and normalization as preprocessing steps for the images. We will then describe the considered

methods for unsupervised optimization in Section 3, including two baselines: OFCO and PyStackReg. The first baseline provides accurate and consistent results, however being very time consuming; while the second is orders of magnitude faster, but can only predict a basic affine transformation. We thus we have a trade-off between accuracy and performance.

An important contribution to the first baseline, OFCO, is a speedup of 140% to its implementation achieved by migrating its heaviest calculations from CPU to GPU. An analysis of this bottleneck and changes to the code will be provided in Section 3.2.

Finally, we will evaluate the methods' accuracy and performance in Section 4, and discuss the results in Section 5.

2 Experimental setup

In this section we provide a quick overview of the data we will be working with and the metrics considered, including their advantages and disadvantages for comparing registration methods and evaluating the goodness of a registration.

We will then utilize the insights gained related to some of the metrics' weaknesses in order to justify a set of preprocessing steps on the 2-photon images. These preprocessing steps will be applied before calculating the metrics and in general before registration.

2.1 Dataset overview

Multiple 2-photon microscopy imaging recordings of coronal sections of the cervical connective were available for training and validation. In this work, 9 such experiments were considered, which include two channels each: a recording for the constant neural activation from the tdTomato fluorophore; and a recording for the activity-dependent GCaMP6s images. This means that the intensity in the ROIs for tdTomato images is roughly constant (due to some noise), while it varies for ROIs in GCaMP6s images. Table 1 indicates under which path these experiments were located.

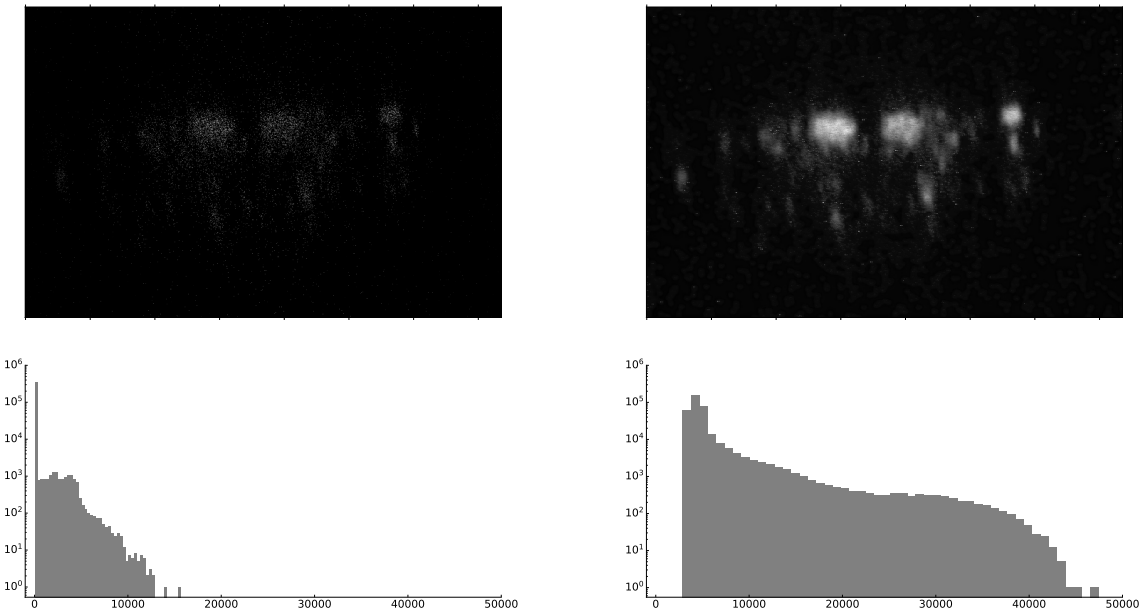
Each of the channels consists of 4'100 frames, and each frame contains 480×736 pixels.

These recordings were *denoised* through Noise2Noise [18], a learning-based method to recover signals when additive zero-mean noise is present. In our case we have zero shot noise and Dark Current noise [24, 25]. Figure 1 shows an example tdTomato raw image and its denoised counterpart.

In what follows, we will use only the denoised images.

Experiment No.	Path
# 1	FA/200901_G23xU1/Fly1/Fly1/001_coronal/
# 2	FA/200901_G23xU1/Fly1/Fly1/002_coronal/
# 3	FA/200901_G23xU1/Fly1/Fly1/003_coronal/
# 4	FA/200901_G23xU1/Fly1/Fly1/004_coronal/
# 5	FA/200901_G23xU1/Fly1/Fly1/005_coronal/
# 6	FA/200901_G23xU1/Fly1/Fly1/006_coronal/
# 7	FA/200901_G23xU1/Fly1/Fly1/007_coronal/
# 8	FA/200901_G23xU1/Fly1/Fly1/008_coronal/
# 9	FA/200901_G23xU1/Fly1/Fly1/009_coronal/

Table 1: Dataset directory paths for each considered experiment. Each folder contains both tdTomato channel and GCaMP6 channel monochrome TIFF files [2]



(a) Raw (b) Denoised

Figure 1: Example tdTomato image, before and after applying Noise2Noise, with the corresponding pixel value histograms. y-axis is in log-scale. In this work we will only consider denoised recordings

2.2 Metrics

In this section, we propose different metrics to gauge the alignment of the images. In order to achieve this, we would like the target metric to contain the following properties:

1. It quantifies how *similar* an image is to another.
2. It detects when a frame *fails*, which we define as an extreme misalignment with respect to the stationary image. Registration algorithms can fail on some outlying frames — as we will see in Section 3.1.2 — and so it is important to consistently be able to label these failing frames.
3. It can be used as a loss function in the learning-based algorithms we will consider, where it is a cost to minimize. This would be very useful since the trained model would then directly optimize the metric we care about, instead of optimizing an unrelated loss function which may not comply with the other properties here presented.
4. It generalizes to images in more than 2 dimensions. We may want to apply the metric to 3D volumes, like a VNC volume and a reference atlas.

We considered the following metrics:

- *Center of mass (COM)*: It uses the center of mass of each image as an estimate of its offset from the stationary image. Unlike the other metrics, COM is able to identify bad, or failing, frames, and it gives an estimate of how often the algorithms categorically fail. Given the centers of mass of all images $\boldsymbol{c}_i \in \mathbb{R}^2$, $i = 1, ..., N_{\text{frames}}$ with N_{frames} the number of frames being considered, this metric is calculated as follows,

$$\text{COM} := \frac{100}{N_{\text{frames}}} \sum_{i=1}^{N_{\text{frames}}} \mathbf{1}_{\{\|\boldsymbol{c}_i - \boldsymbol{c}_{\text{mean}}\| > T\}} \tag{1}$$

$$\boldsymbol{c}_{\text{mean}} := \frac{1}{N_{\text{frames}}} \sum_{i=1}^{N_{\text{frames}}} \boldsymbol{c}_i \tag{2}$$

  Where $T \in \mathbb{R}$ is a constant value, and $\|\cdot\|$ is the euclidean distance. T is sensitive to intensity values since, for example, adding a constant value to the pixels brings centers closer together, and thus T is sensitive to the dataset we use. We will atone this issue by thresholding the images (Section 2.3.1). The chosen T is %10 of the smallest dimension of the image ($T = 10\% \cdot 480 = 48$ pixels), which was selected through a sensitivity analysis that will be shown in Figure 13 in Section 4.
  Intuitively, this metric provides the percentage of failing frames, which only gives us properties 2. and 4., but this is the only proposed metric that can consistently hold the failing frame identification property.
  Figure 2 shows the centers of mass of the denoised recordings.

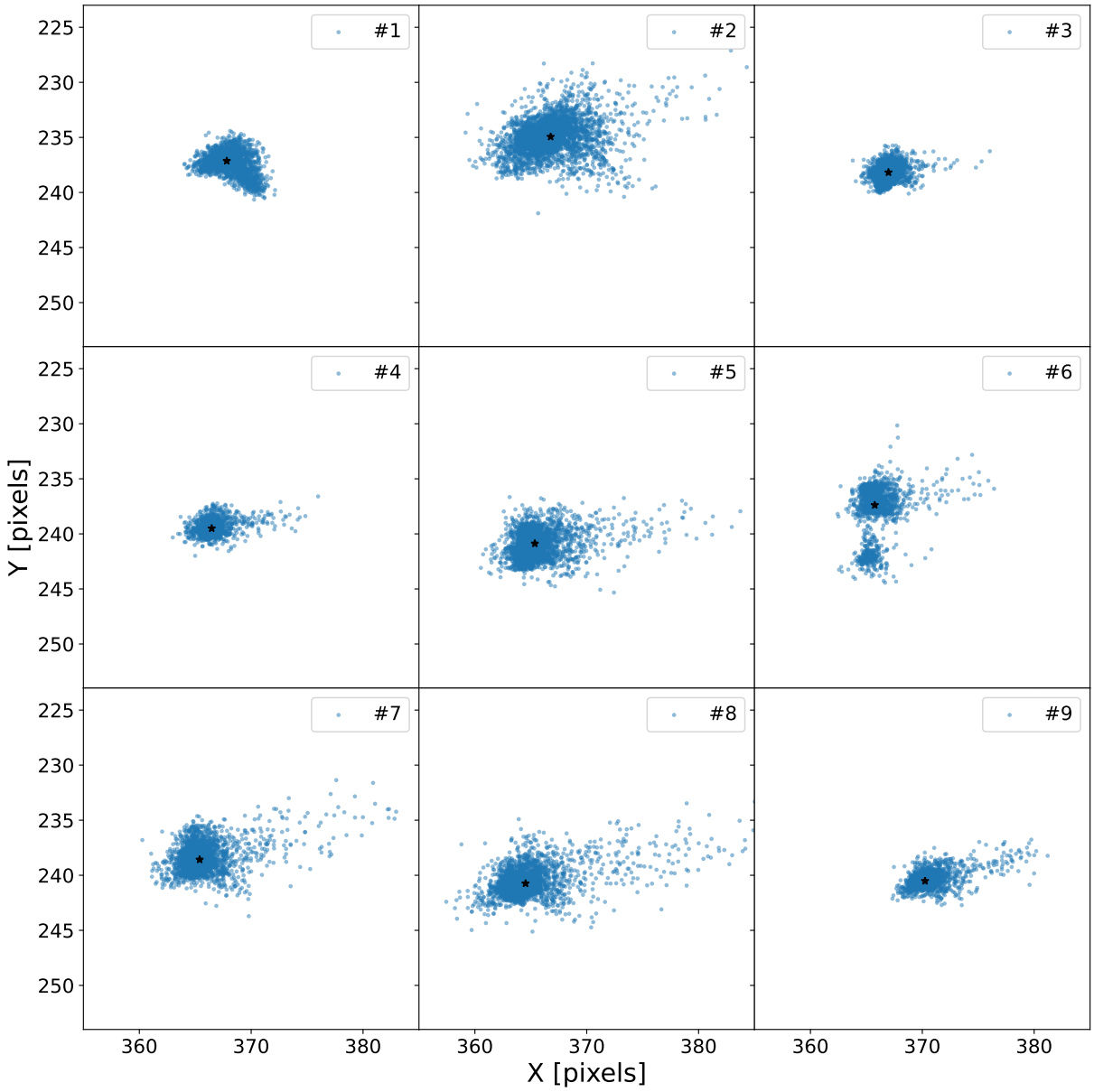


Figure 2: Location of the tdTomato centers of mass for all 9 experiments. There are 4'100 points in each plot, one for each frame. The black dot corresponds to their mean $\boldsymbol{c}_{\text{mean}}$. The experiment number is also indicated

- *Mean squared error (MSE)*: It is calculated as follows:

$$\text{MSE} := \frac{1}{N_{\text{frames}}} \sum_{i=1}^{N_{\text{frames}}} \frac{1}{|\Omega|} \sum_{\boldsymbol{p} \in \Omega} [I_i(\boldsymbol{p}) - I_i^r(\boldsymbol{p})]^2 \tag{3}$$

  Where Ω is the discrete image domain, I_i is the image i, and I_i^r is the reference image that I_i should be similar to. We will consider the case $I_i^r := I_{i-1}$, where i indicates the chronological order of the frames.

  The two main drawbacks for MSE are:

  – It gives much more importance to high intensity ROIs than lower ones and, if the background has a large range of values and many pixels, it will dominate in the metric, misrepresenting the ROIs. The first problem can be mitigated with normalization (Section 2.3.2), while the second can be solved with thresholding (Section 2.3.1). We will go through these preprocessing steps in the next section.

  – It is invariant to translation when the ROIs do not overlap, thus it cannot detect big misalignments.

  However, MSE does comply with properties 1., 3. and 4., as well as being one of the main similarity metrics used in medical image registration, making it one of the main metrics we will use to compare registration methods (see Section 4).

- *Normalized local cross-correlation (NCC)*: It is the other main similarity metric used in medical image registration. Let d be the dimension of I_i, $\Omega_{\boldsymbol{p}}$ a 9^d volume around $\boldsymbol{p}$ with the same domain as Ω, and $\hat{I}(\boldsymbol{p}) = \frac{1}{9^d} \sum_{\boldsymbol{q} \in \Omega_{\boldsymbol{p}}} I(\boldsymbol{q})$. Then, NCC is defined as [4],

$$\text{NCC} := \frac{1}{N_{\text{frames}}} \sum_{i=1}^{N_{\text{frames}}} \sum_{\boldsymbol{p} \in \Omega} \frac{\Big(\sum_{\boldsymbol{q} \in \Omega_{\boldsymbol{p}}} (I_i(\boldsymbol{q}) - \hat{I}_i(\boldsymbol{p}))(I_i^r(\boldsymbol{q}) - \hat{I}_i^r(\boldsymbol{p})) \Big)^2}{\Big(\sum_{\boldsymbol{q} \in \Omega_{\boldsymbol{p}}} (I_i(\boldsymbol{q}) - \hat{I}_i(\boldsymbol{p}))^2 \Big) \Big(\sum_{\boldsymbol{q} \in \Omega_{\boldsymbol{p}}} (I_i^r(\boldsymbol{q}) - \hat{I}_i^r(\boldsymbol{p}))^2 \Big)}$$

  Like MSE, it is invariant to ROIs translation when they do not overlap, but it does follow properties 1., 3. and 4.
  Its main advantage over MSE is that it is more robust to intensity variations [3].

- *Earth mover's distance (EMD)*: Previous work has demonstrated EMD as a very representative image similarity metric which can be calculated in polynomial-time complexity [28]. This metric is usually defined in 1-D, but we can generalize it to n-D,

$$\text{EMD} := \frac{1}{N_{\text{frames}}} \sum_{i=1}^{N_{\text{frames}}} \inf_{\gamma \sim \Pi(\bar{I}_i, \bar{I}_i^r)} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\gamma}[\|\boldsymbol{x} - \boldsymbol{y}\|]$$

Where $\bar{I} = aI + b$ are the images as discrete probability distributions, with $a, b \in \mathbb{R}$ s.t. $\bar{I}(\boldsymbol{p}) > 0 \ \forall \boldsymbol{p} \in \Omega$ and $\sum_{\boldsymbol{p}\in\Omega} \bar{I}(\boldsymbol{p}) = 1$; and $\Pi(\bar{I}_i, \bar{I}_i^r)$ is the set of joint distributions whose marginals are $\bar{I}_i$ and $\bar{I}_i^r$. $\|\cdot\|$ can be any norm, but we will use ℓ_2.
Calculating EMD exactly is unfeasible, since it would require solving the optimal transport problem of finding the flow to change distribution $\bar{I}_i$ to $\bar{I}_i^r$, which is precisely what we are already trying to do in the full-aim of image registration.
However, we can approx. EMD by sampling from $\bar{I}_i$ and $\bar{I}_i^r$, then solving a Linear Program. There are theoretical and experimental results on the convergence of this approach [31].
The main advantage of EMD is that it both quantifies similarity and can detect misalignments precisely, so it is in principle more representative of similarity than MSE and NCC, but its main drawback is the probabilistic nature of its implementation.

It also has the two sources of problems also found in MSE: 1. when ROIs have different intensity EMD gives more importance to more intense ones; 2. when backgrounds have a large range of values and many pixels, the probability of sampling a pixel in the background increases, even though they are irrelevant for quantifying the similarity between ROIs. Again, these problems will be mitigated and solved with the preprocessing steps of the section below.

We will include the EMD metric in the results, Section 4.

2.3 Preprocessing

This section presents the techniques used to prepare the 2-photon images for the different algorithms and metrics.

It is important to note that for the baseline methods the two proposed preprocessing steps, thresholding and normalization, were only applied after registering the images, as in this case we just want to prevent spurious results in the metrics — see the MSE and EMD drawbacks in the previous section —, and not alter the original intended behavior of the baselines. For all other methods, the preprocessing steps were done before registration. The exact implementation of the methods will be explained in Section 3.

2.3.1 Thresholding

As discussed in Section 2.2, metrics and loss functions can be more sensitive to changes in the background than in the ROIs if there are many pixels in the background with a wide range of values.

In order to separate the background from the axons, we want to apply an approach which automatically finds their separation intensity value. Figure 3 summarizes the strategies we looked at. In Otsu thresholding [26] (`otsu`), the threshold T maximizes the variance between the separated classes (background and foreground). Triangle thresholding (`triangle`) is a very simple approach first described in [35] which maximizes the distance to the segment spanning the highest intensity and maximum value in the intensity histogram (see Figure 4). Another approach was to use the watershed algorithm to find the minimum between two local maxima (`watershed`), since it was observed that most intensity histograms were bi-modal. `watershed` assumes the first mode corresponds to the background and the second to the axons, although this does not seem to be empirically accurate, as can be seen in Figure 3. Finally, following a similar procedure to that of G. Collewet et al. [8], T corresponds to the local maximum with second lowest intensity in the second derivative of the intensity histogram (`second_deriv`). Figure 5 provides a visual description of `second_deriv`.

Triangle thresholding was chosen as the ROI segmentation algorithm since it is not as aggressive as Otsu, and less conservative than `watershed`. Although the example in Figure 3 shows `second_deriv` correctly segmenting the background, it is not consistent and in some of our images it would predict a too high T.

After retrieving T using triangle thresholding, the intensities $I(\boldsymbol{p})$ for the pixels $\boldsymbol{p}$ in the domain Ω of the images were clipped:

$$I(\boldsymbol{p}) \leftarrow \begin{cases} I(\boldsymbol{p}) - T & \text{if } I(\boldsymbol{p}) > T \\ 0 & \text{o.w.} \end{cases} \qquad \forall \boldsymbol{p} \in \Omega$$

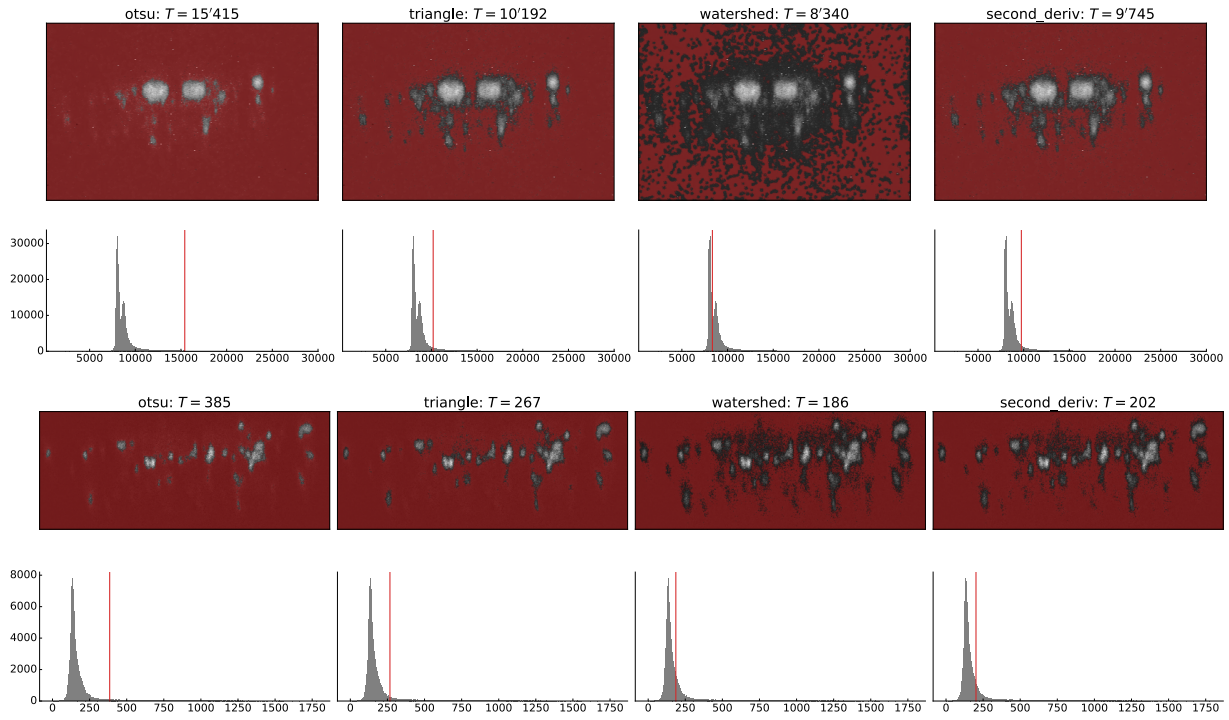


Figure 3: Example tdTomato image (top rows) and GCaMP6 image (bottom rows) with different thresholds applied and its corresponding intensity histogram. The red vertical line in the histograms corresponds to the threshold T

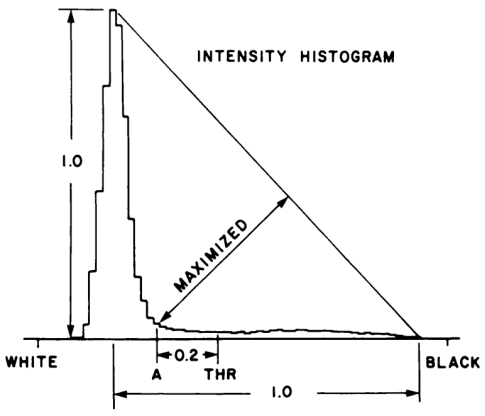


Figure 4: Original figure demonstrating triangle thresholding from G. W. Zack et al. [35]. The height and dynamic range of the intensity histogram are normalized to 1

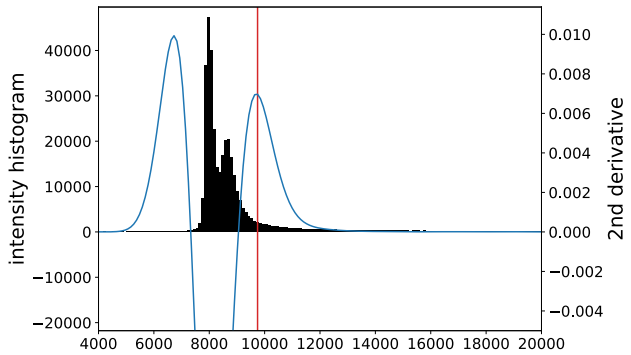


Figure 5: `second_deriv` thresholding. The bar plot corresponds to the intensity histogram of a tdTomato image and the blue line corresponds to its 2nd derivative. The red vertical segment is the chosen T

2.3.2 Normalization

As mentioned in Section 2.2, MSE gives quadratically more importance to intensities in ROIs with higher values than those with lower values. This means axons with smaller neural activations will have very low impact on the MSE cost, which in particular is a problem in VoxelMorph as it uses the MSE loss for training, and will then generate a deformation field which largely ignores these small, low-valued regions.

To mitigate this problem, we will apply a log-transform to the intensities, right after triangle thresholding has been applied:

$$I(\boldsymbol{p}) \leftarrow \ln(I(\boldsymbol{p}) + 1) \quad \forall \boldsymbol{p} \in \Omega$$

An offset is added to keep the zero-valued background. This conversion is akin to operating with optical density, as it is done in works like M. Macenko et al. [23]. Figure 6 displays the impact of this normalization.

Additionallly, a simple affine normalization was applied after the log-normalization to bring pixel values between 0 and 1,

$$I(\boldsymbol{p}) \leftarrow \frac{I(\boldsymbol{p}) - \min_{i,\boldsymbol{q}} I_i(\boldsymbol{q})}{\max_{i,\boldsymbol{q}} I_i(\boldsymbol{q}) - \min_{i,\boldsymbol{q}} I_i(\boldsymbol{q})} \quad \forall \boldsymbol{p} \in \Omega \tag{4}$$

This will be important for VoxelMorph, which in order to generalize properly needs intensities in a predictable range of values.

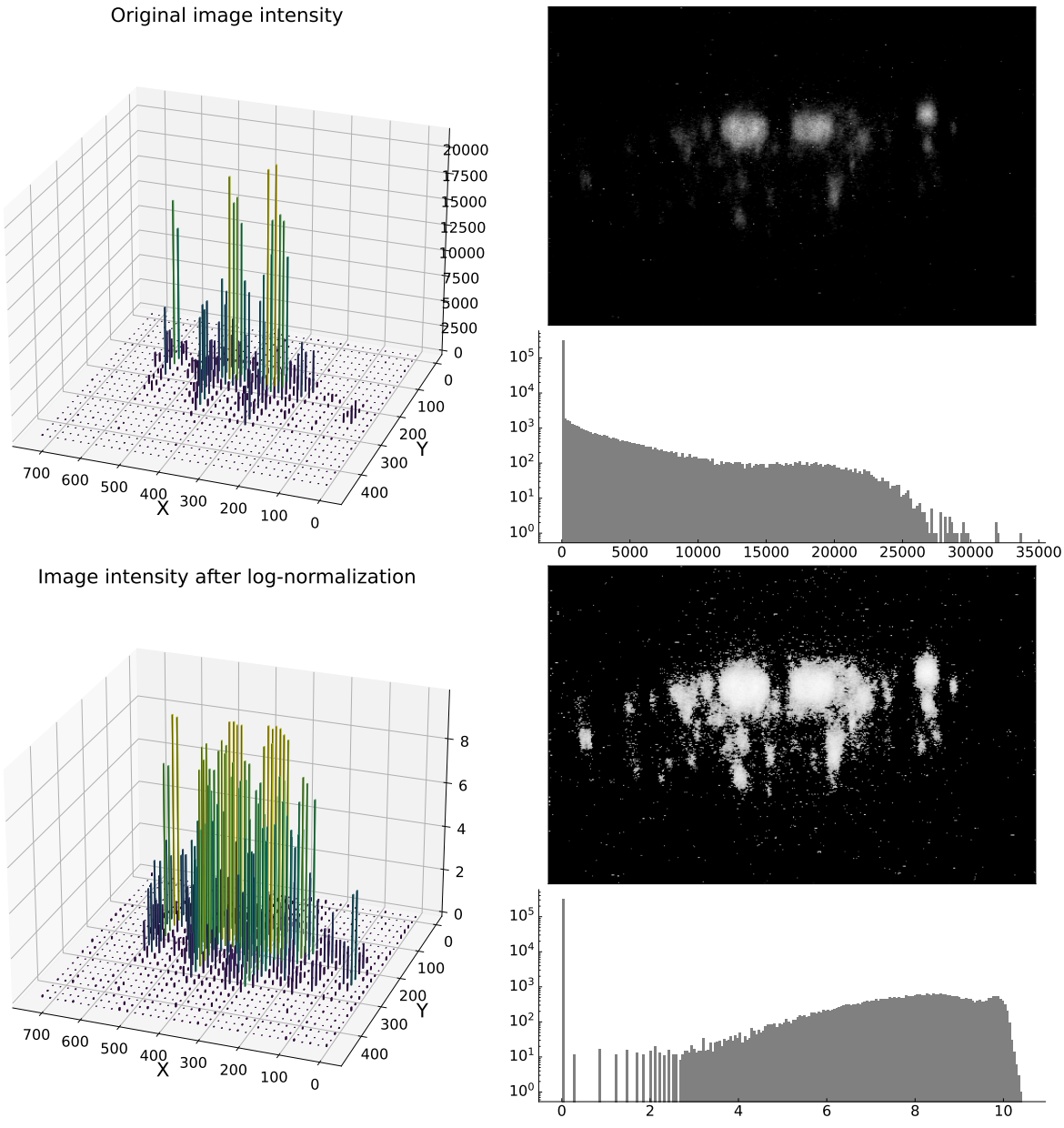


Figure 6: tdTomato image before and after log-normalization. Note that triangle thresholding has been applied beforehand. Intensity histograms are also present, with the y-axis in log-scale. Applying this transformation brings axon intensities closer to each other so they can have a similar impact on MSE

3 Methods

The main aim of all methods is to produce a transformation to stabilize the architectural reference tdTomato images, so as to then apply the same transformation on the corresponding GCaMP6 images. All methods to be presented predict some type of transformation in addition to registering tdTomato images.

We will address five registration methods which can categorically be placed into two kinds: affine alignment and deformable transformation. Affine alignment methods are a global, fast, first approximation to the correct alignment of an image, and are usually a required initial step before most complex formulations. Deformable transformation methods typically iteratively optimize an energy function of the form [30]:

$$\hat{\boldsymbol{\phi}} = \arg\min_{\boldsymbol{\phi}} (1-\lambda)\mathcal{L}_{sim}(\boldsymbol{\phi}, I_i, I_i^r) + \lambda\mathcal{L}_{smooth}(\boldsymbol{\phi}) \quad \lambda \in [0,1] \tag{5}$$

Where I_i is the i-th image we are want to align, I_i^r is its target reference image, $\boldsymbol{\phi}$ is the motion field, and $\mathcal{L}_{sim}$, $\mathcal{L}_{smooth}$ are the image similarity and spatial smoothness deformation losses, respectively.

We will then propose two deterministic baselines: OFCO (deformable transformation) and PyStackReg (affine alignment). We will see that the first baseline is very precise, although slow, while the second baseline, and the other approaches, are less accurate but orders of magnitude faster.

3.1 Baselines

Since we want to find a trade-off between speed and accuracy, we will present in this section two algorithms: OFCO and PyStackReg. The first is a slow but very precise approach, while the latter finds an affine transformation to quickly register the images.

3.1.1 OFCO

Our first baseline method is the one presented in [6], a variational optical flow estimation algorithm which solves the optimization problem (5) with,

$$\mathcal{L}_{sim}(\boldsymbol{\phi}, I_i, I_i^r) = \sum_{\boldsymbol{p}\in\Omega} |I_i(\boldsymbol{p} + \boldsymbol{\phi}(\boldsymbol{p})) - I_i^r(\boldsymbol{p})|$$

Which is the ℓ_1-norm between the displaced image and its reference, and,

$$\mathcal{L}_{smooth}(\boldsymbol{\phi}) = \sum_{\boldsymbol{p}\in\Omega} \|\nabla\boldsymbol{\phi}(\boldsymbol{p})\|_2^2$$

The objective (5) is minimized iteratively and, as is traditional in image registration, through a Gaussian pyramid of images, starting with a coarse downsampled representation of I_i and I_i^r, gradually converging $\hat{\boldsymbol{\phi}}$ with finer representations. In particular, an alternated direction method of multiplier (ADMM) algorithm was used [5].

Once $\hat{\boldsymbol{\phi}}$ is calculated, the image I_i is reconstructed with a bilinear interpolation.

We will refer to this baseline as OFCO. In its Python implementation, Equation 5's weight parameter is $\lambda = 800/801 = 0.998$.

3.1.2 PyStackReg

StackReg is a coarse-to-fine (pyramid) iterative strategy for affine alignment developed by P. Thévenaz et al. [32]. Our second baseline method is PyStackReg [22], the Python implementation of StackReg.

This approach was applied to the denoised tdTomato images as-is, without the preprocessing steps explained in Section 2.3 since, as mentioned, these steps were done only when calculating the metrics.

Unfortunately, this registration algorithm can produce *failing* frames (see property 2. in Section 2.2) on image outliers. Figure 7 shows an example of such failing output.

We will quantify the percentage of failing frames in Section 4.1.

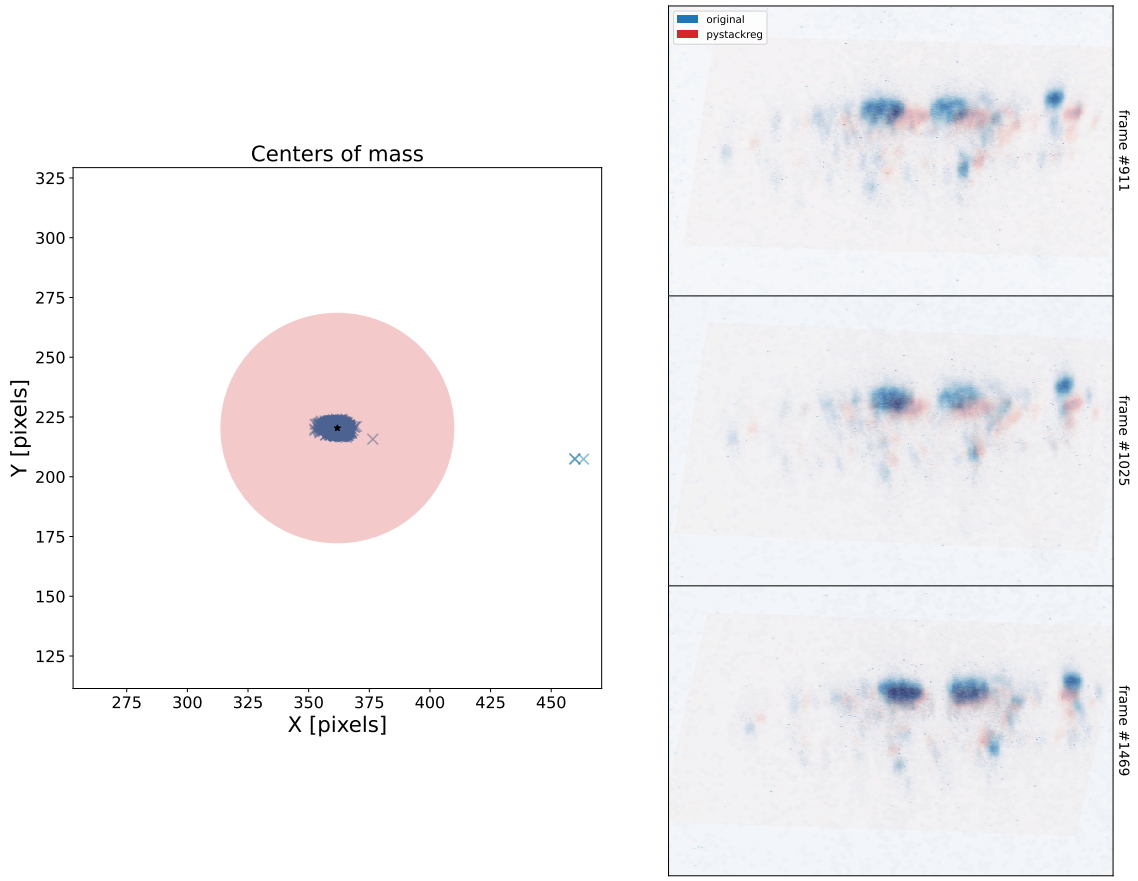


Figure 7: Example PyStackReg output recording failure frames. COM failing detection radius is $T = 0.1 \cdot \min\{480, 736\} = 48$. The red circle indicates the non-failing area. You can see in the left plot the three outliers corresponding to the images plotted on the right, which are superimposed to the corresponding original denoised image

3.2 OFCO GPU speedup

The practical implementation of the ADMM algorithm used in OFCO [6] is done in Python, using the NumPy, SciPy and scikit-image libraries. This implementation is process-parallel on the frames of the input 2-photon recording. Figure 8 shows the profiling results of executing OFCO on 2 frames of one of the tdTomato recordings. As it can be seen, most of the execution time is spent on NumPy's 2D FFT and IFFT, which use the CPU.

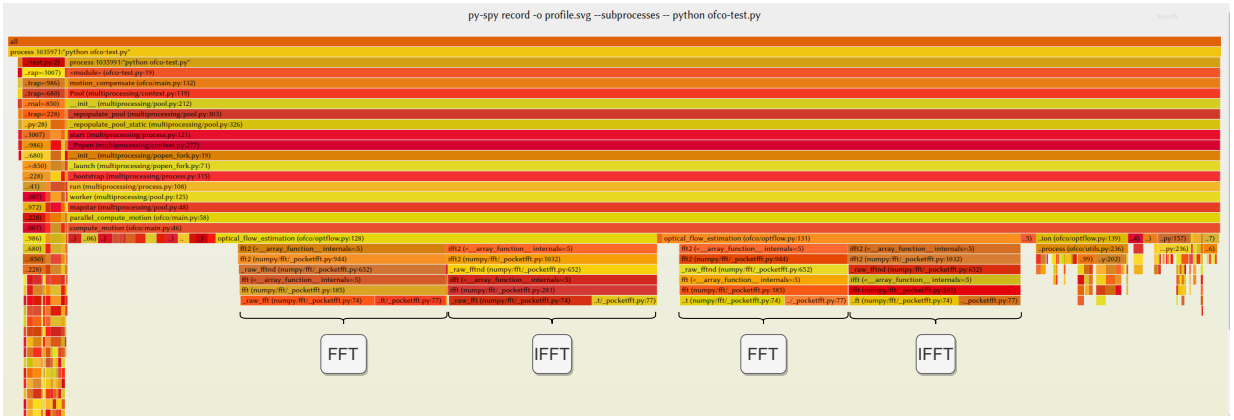


Figure 8: py-spy [11] flame graph profiling results on the original OFCO implementation. 2 frames. x-axis is time

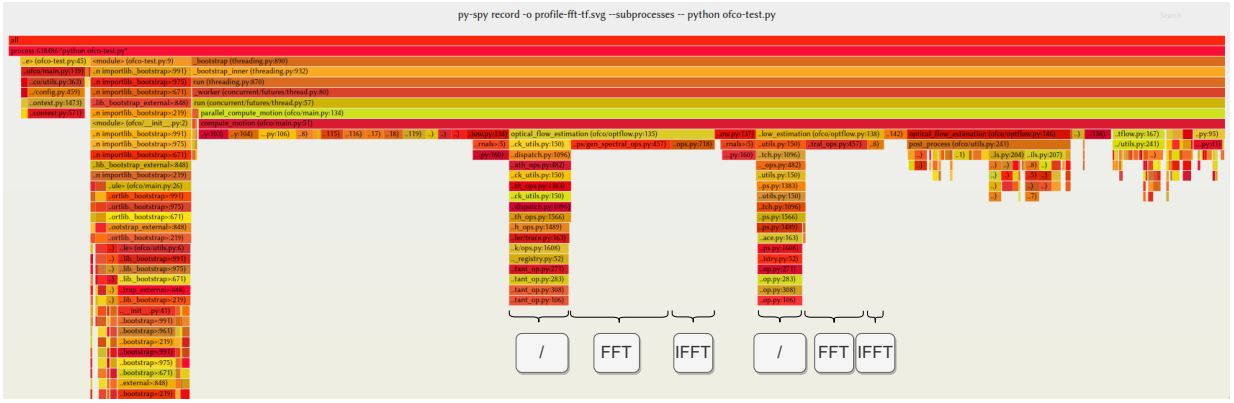


Figure 9: py-spy flame graph profiling results on the faster OFCO-GPU implementation. 2 frames. In this case, Tensorflow's division operation has an equatable impact to FFT and IFFT. x-axis is time

To improve performance, the 2D FFT and IFFT Numpy functions, `np.fft.fft2` and `np.fft.ifft2`, were replaced with Tensorflow's implementation, `tf.signal.fft2d` and `tf.signal.ifft2d`, and the program was re-written to be thread-parallel in order to reduce the overhead of creating the processes. Figure 9 displays the profiling results.

This more efficient version of OFCO will be referred to as OFCO-GPU.

Figure 10 shows the resulting speedup, which is $t_{\text{OFCO}}/t_{\text{OFCO-GPU}} \simeq 140\%$ for recordings of more than 1'024 images, tested on a 16-core 3.50GHz Intel Core i9-11900K CPU and a GeForce RTX 3080 Ti GPU.

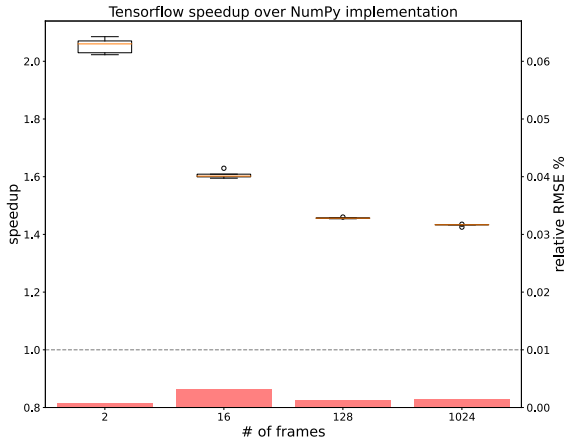


Figure 10: OFCO speedup boxplots. 5 test runs. The bar plots correspond to the relative RMSE between OFCO's Tensorflow output vs NumPy's (RMSE over NumPy's standard deviation). This negligible error is likely due to numerical errors, since Tensorflow's implementation uses single-precision, while NumPy's is double-precision. As it can be seen, with a large number of frames the speedup confidently converges to $\sim 140\%$

3.3 Center of mass and ECC

Another affine alignment method is the enhanced correlation coefficient (ECC) maximization algorithm by G. D. Evangelidis and E. Z. Psarakis [10], available in OpenCV. This is an iterative method that with just 3 iterations achieves only slightly worse performance than PyStackReg, while being much faster. Like PyStackReg, it also fails on some outliers, but we will fix this with a preparation step, which follows.

Given a target position, we can shift the images so that their new center of mass is at the target, which would crudely align them. We call this transformation COM-T. This fortunately removes the failing frames identified by the COM metric, as it is optimizing the metric.

This leads to the following approach, which we name COM-T+ECC:

1. Apply preprocessing thresholding and normalization (described in Section 2.3).
2. Apply COM-T: shift frames so their center of mass is at a target center of mass. The target we choose is the mean center of mass over all frames' center of mass ($\boldsymbol{c}_{\text{mean}}$ from Equation 2).
3. Affine transform the images with ECC maximization.

COM-T is around an order of magnitude faster than the 3-iterations affine alignment of ECC, meaning it has a low computational cost and ECC dominates the execution time of the algorithm.

COM-T+ECC is useful, since VoxelMorph requires an affine transformation as an initial step, and this method achieves so quickly.

3.4 VoxelMorph

VoxelMorph [4] is a deep neural network (DNN) based learning model which estimates the optical flow between an image I_i and a reference I_i^r. It does so through convolutional layers, which are very efficient on GPU [20]. In particular, it solves the optimization problem (5) with $\mathcal{L}_{sim}$ equal to either MSE or NCC and,

$$\mathcal{L}_{smooth}(\boldsymbol{\phi}) = \sum_{\boldsymbol{p}\in\Omega} \|\boldsymbol{g}_{\boldsymbol{\phi}}(\boldsymbol{p})\|_2^2 \tag{6}$$

Where $\boldsymbol{g}_{\boldsymbol{\phi}}$ approximates $\nabla\boldsymbol{\phi}$.

As previously mentioned, thresholding and normalization were applied for all 9 tdTomato training and validation recordings as a preprocessing step (Section 2.3), and a quick affine alignment was applied as a first global registration approximation, namely COM-T+ECC (Section 3.3), before the forward pass on the VoxelMorph DNN. This is also done on inference for new unseen test recordings.

NCC was used as the $\mathcal{L}_{sim}$ loss, since it provides more robust results [4]. VoxelMorph is based on UNet [27], which is a widely studied DNN in medical imaging [33, 13, 19, 34, 15]. Figure 11 shows the architecture and hyper-parameters used for VoxelMorph. It consists of 5 encoding channels 16, 32, 32, 128, and 128; 5 decoding channels 128, 128, 32, 32, and 32; and 2 more convolutional layers with channels 16 and 16. It takes as input the images to register I_i and their fixed references I_i^r, and outputs the predicted optical flows and registered images.

Finding λ (from Equation 5) was done through grid-search in the space $[0, 1]$, and chosen to be $\lambda = 0.25$. This is a highly inefficient approach that was addressed in more recent work with HyperMorph [12], which we describe in the next section.

For training, Adam optimizer [16] was used, with exponential learning rate decay (as done in works like [14]) from 10^{-3} to 10^{-6} over 1'000 epochs. Training batches were chosen by selecting first one of the experiments #1-4 and #6-9 at random, then randomly picking 8 distinct frames $I_{i_b}, b = 1, ..., 8$ from the experiment's tdTomato recording, and finally setting the reference frames to $I_{i_b}^r := I_{i_b-1}$. The reference images for validation were again $I_i^r := I_{i-1}$.

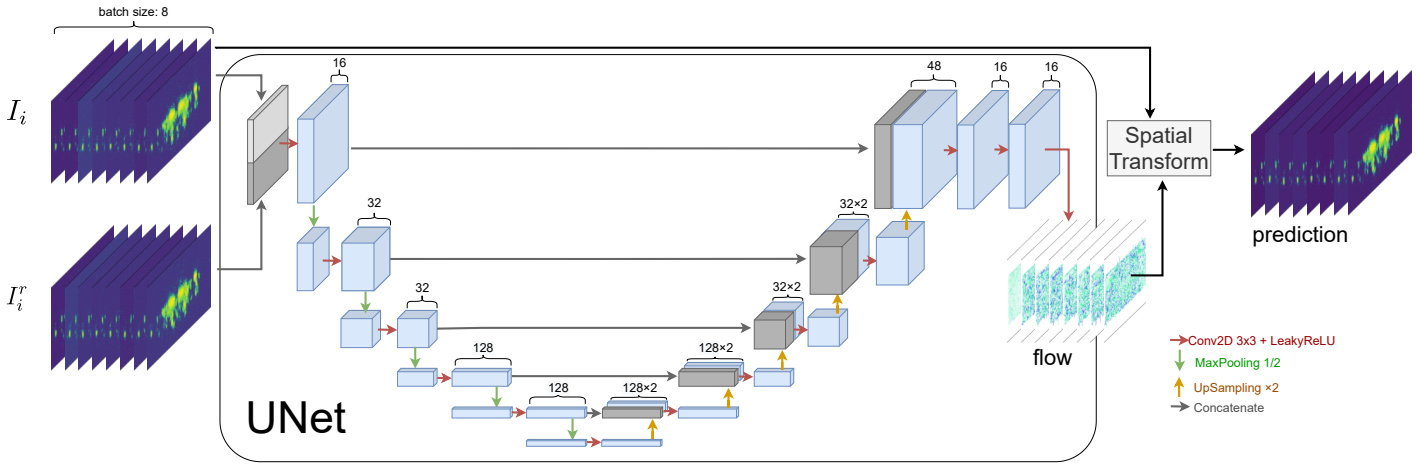


Figure 11: VoxelMorph architecture and hyper-parameters. Total number of parameters (weights and biases) is 761'618

3.4.1 HyperMorph

HyperMorph captures the behavior of individually optimized VoxelMorph DNNs according to some hyper-parameter [12]. It is, thus, a type of *hypernetwork*. In particular, the hyper-parameter we will use is λ from the optimization problem (5).

HyperMorph is composed of a hypernetwork with trainable parameters, which outputs the non-trainable parameters of a VoxelMorph network. This hypernetwork has a scalar input (λ) and is composed of 4 fully connected layers of 64 nodes with ReLU activations, except for the final layer which uses Tanh. The represented VoxelMorph architecture parameters are those of a UNet with 4 encoding channels 16, 32, 32, and 32; 4 decoding channels 32, 32, 32, and 32; and 3 more convolutional layers with channels 32, 16 and 16. This is a simpler UNet to that of Figure 11 since otherwise the total number of parameters would be intractable. The total number of parameters (trainable weights and biases) for HyperMorph is 13'534'194.

In the source work [12], the hyper-parameter λ is, in one approach, automatically optimized using segmentation maps, which our dataset does not possess. The proposed alternative is to manually find the best λ by visual inspection. However, we can notice that choosing a single λ for all frames may not provide the most accurate results, and rather we should choose some λ_i per frame I_i in order to optimize the registration accuracy across all frames. In order to do this, we select N_λ uniform values in $[0, 1]$ and calculate for each frame I_i the predicted $\hat{I}_{i,j}$, where $i = 1, ..., N$, and $j = 1, ..., N_\lambda$ indexes which λ value we used to predict $\hat{I}_{i,j}$. Then, we calculate the following objective for all image pairs $(\hat{I}_{i,j}, \hat{I}_{i+1,j})$:

$$J_{i,j} = \frac{1}{2}\left(\mathcal{L}_{sim}(\hat{I}_{i,j}, \hat{I}_{i+1,j}) + \frac{1}{2}\left(\mathcal{L}_{smooth}(\hat{\boldsymbol{\phi}}_{i,j}) + \mathcal{L}_{smooth}(\hat{\boldsymbol{\phi}}_{i+1,j})\right)\right)$$

Where $\hat{\boldsymbol{\phi}}_{i,j}, \hat{\boldsymbol{\phi}}_{i+1,j}$ are the predicted deformation fields for $\hat{I}_{i,j}, \hat{I}_{i+1,j}$, respectively, and $\mathcal{L}_{sim}$ is NCC and $\mathcal{L}_{smooth}$ is the same as for VoxelMorph (Equation 6). This optimization objective includes smoothness losses in order to ensure natural transformations, otherwise optimizing $\mathcal{L}_{sim}$ would find the best VoxelMorph network to be the one trained on $\lambda = 0$, which completely ignores the smoothness constraint. The challenge here is that each consecutive $J_{i,j}$, $J_{i+1,j}$ uses the same $\hat{I}_{i+1,j}$, and if we want to minimize $\min_{j_1,\dots,j_N} \sum_{i=0}^{N-1} J_{i,j_i}$ we have to take this into account. Figure 12 visualizes the optimization problem as a weighted directed acyclic graph. Note how nodes in the first frame $\hat{I}_1$ act as sources and those at $\hat{I}_N$ act as sinks, and $J_{i,j}$ are the edge weights. We will find the optimal λ values for each frame by solving the shortest path between each source-sink pair — which means we will solve the shortest path problem N_λ^2 times —, and choose the source-sink pair that achieves the lowest weighted path. Given V vertices and E edges, the shortest path problem can be solved with Topological Sorting in $O(V + E)$ time, which in our case is $O((2 + N_\lambda(N-2)) + (2N_\lambda + N_\lambda^2(N-2))) = O(N_\lambda^2 N)$. In total, the complexity is $O(N_\lambda^4 N + N_\lambda N t_{pred})$, where t_{pred} is the time it takes to predict one pair $\hat{I}_{i,j}, \hat{\boldsymbol{\phi}}_{i,j}$. Since in practice t_{pred} is very high valued, we choose a small $N_\lambda = 8$.

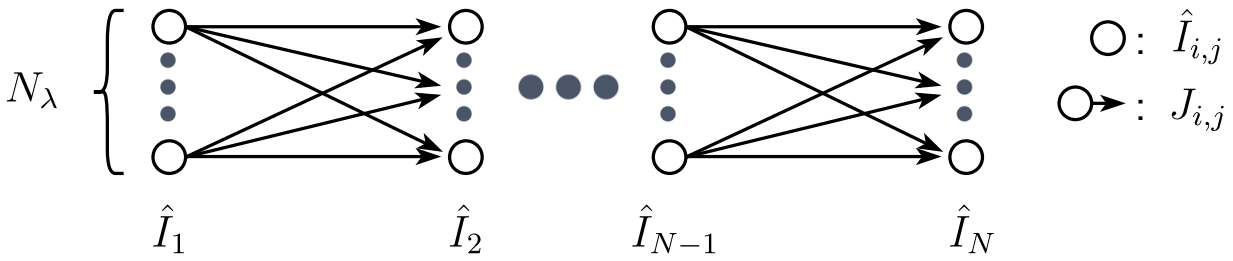


Figure 12: Optimization problem to find the optimal λ for each frame visualized as a weighted directed acyclic graph. For each frame, we have to select out of the N_λ possible λ values only one

The training objective is the same as for VoxelMorph, seen in the above section. Equally, training took place with the Adam optimizer and exponential learning rate decay from 10^{-3} to 10^{-6} over 1'000 epochs. HyperMorph also uses as reference $I_i^r := I_{i-1}$.

4 Results

In this section we will perform an analysis on the threshold radius parameter for COM, and present the metric scores and prediction rates for all discussed methods.

4.1 COM sensitivity analysis

As mentioned in Section 3.1.2, we can quantify the percentage of failures for PyStackReg predictions with the COM metric. However, this metric has a parameter to select: the threshold radius T (from Equation 1). Figure 13 shows how COM changes according to T for the original denoised tdTomato, OFCO and PyStackReg images. We choose T to be 10% the size of the smallest dimension of the images, since it brings good empirical results for the dataset, as it lies in the plateau of COM values for PyStackReg experiments, which separates common mass shifts and outliers. Intuitively, we can say that T is large enough so as to indicate a transformation with a "noticeable" deviation from the center. After choosing $T = 48$ pixels, COM results for PyStackReg are presented in Table 2.

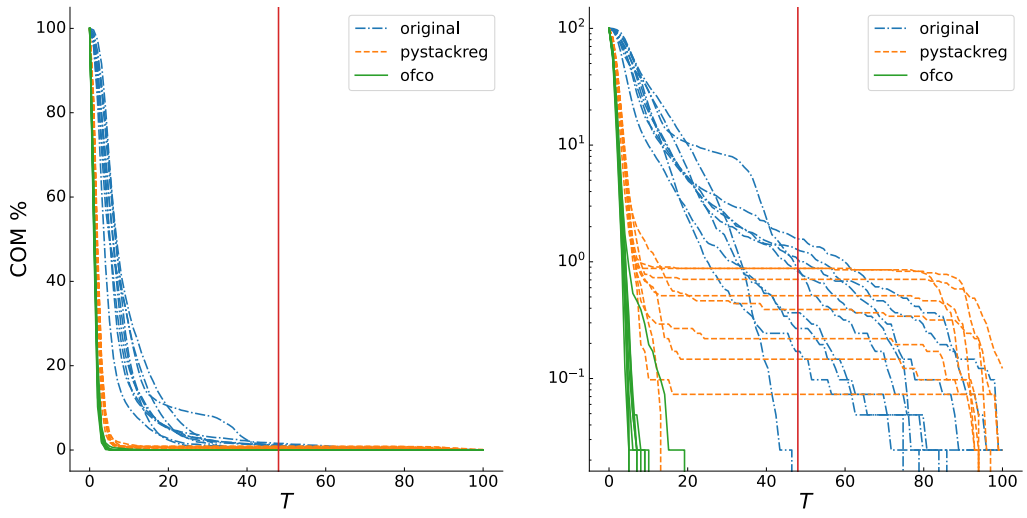


Figure 13: Left plot are the COM scores with varying T for 9 tdTomato recordings after either a PyStackReg or OFCO transform. The original non-transformed recordings are also included. In all cases triangle thresholding and normalization were applied after the transforms. The red line is the chosen $T = 48$ pixels. Right plot is the same as the left one with log scale in the y-axis. 8 PyStackReg results plateau in the right plot, indicating a separation between correctly registered images (lower T) and failing outliers (higher T).

Exp. No.	#1	#2	#3	#4	#5
COM	0.0000%	0.0732%	0.2195%	0.1463%	0.8780%
MSE	0.0227	0.0266	0.0302	0.0226	0.0248
	#6	#7	#8	#9	**Summary**
COM	0.3902%	0.7073%	0.5122%	0.8780%	$(0.4227 \pm 0.3203)\%$
MSE	0.0237	0.0222	0.0242	0.0261	0.0248 ± 0.0024

Table 2: Scores for all 9 tdTomato PyStackReg predictions, with thresholding and normalization applied after the transformation. MSE reference images for each frame I_t are $I_t^r := I_{t-1}$. Note that the normalized images have an intensity range $[0, 1]$, thus MSE < 1

With the same threshold radius $T = 48$ pixels we can get results for COM-T+ECC (discussed in Section 3.3), which are summarized in Table 3. As can be seen, COM becomes 0% since the COM-T transformation optimizes this metric. Furthermore, MSE performs better with this algorithm compared to PyStackReg, while being $\sim \times 7.7$ faster.

Exp. No.	#1	#2	#3	#4	#5
COM	0%	0%	0%	0%	0%
MSE	0.0135	0.0148	0.0145	0.0117	0.0133
	#6	#7	#8	#9	**Summary**
COM	0%	0%	0%	0%	0%
MSE	0.0115	0.0116	0.0141	0.0146	0.0133 ± 0.0013

Table 3: Scores for all 9 tdTomato COM-T+ECC predictions. Same description as Table 2

4.2 Summary

Figure 14 displays the results on an example frame from validation experiment #5 and for each of the methods. It also includes two plots of their overlap with a reference stable frame. Note how the baselines, OFCO and PyStackReg, create an empty background which changes the pixel value distribution. This did not affect the scoring results since we are thresholding and normalizing before calculating the metrics. This threshold, and the affine normalization min. and max. values (Equation 4), are retrieved from the original tdTomato image, not the predictions. The change in the histogram for the other methods with respect to the original image is due to these same thresholding and normalization preprocessing steps (Section 2.3).

Videos of the predictions are found in the supplementary material (Supplementary Movie 1-7).

Table 4 summarizes the prediction scores on tdTomato experiment #5, which was chosen as the validation example, and thus learning-based models did not see it during training.

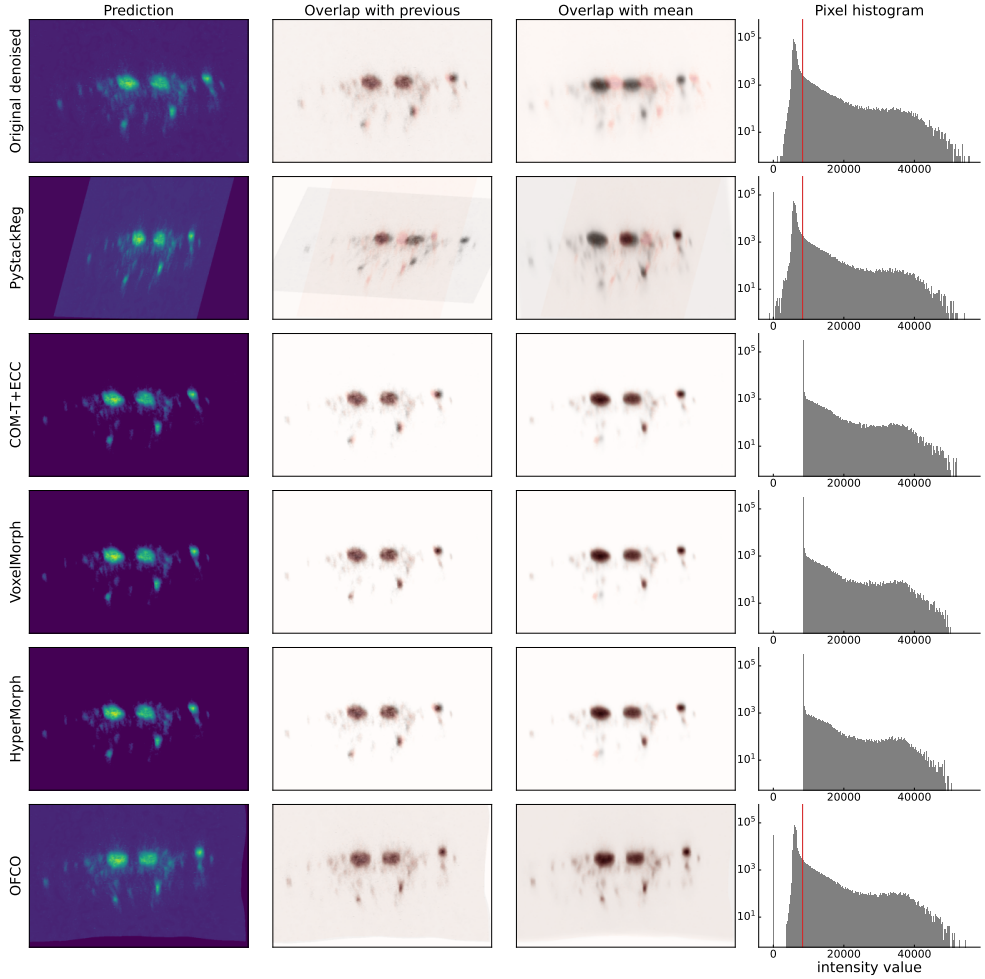


Figure 14: Validation experiment #5 predictions. 2nd column overlaps the frame with the previous frame in the prediction. 3rd column overlaps the frame with the mean over the frames of the predicted recording. Right plots are the pixel intensity value histograms of the predictions. The original image and the baselines also include the threshold value, which is 8'481, as a vertical line

The reference image used for each frame I_t to calculate the metrics is $I_t^r := I_{t-1}$. We should note, however, that the results may vary by choosing different reference images I_t^r.

As done in Section 3.2, timings were retrieved in a machine with a 16-core 3.50GHz Intel Core i9-11900K CPU and a GeForce RTX 3080 Ti GPU. For the EMD scores, 500 points were sampled for each frame in the recordings, and a minimum distance matching between points in frame I_i and frame $I_i^r := I_{i-1}$ was solved, using the ℓ_2-norm. The resulting EMD can be understood as the average work needed in order to change each frame to the next one. So, we are looking at the average pixel distance to move the ROIs.

The timings in VoxelMorph and HyperMorph also take into account the overhead of the affine alignment preprocessing step of COM-T+ECC. For both of these methods prediction was run once on CPU and once on GPU.

Method	COM	MSE	NCC	EMD	frames/s (CPU)	frames/s (GPU)
PyStackReg	0.8780%	0.0248	0.591	17.1 ± 0.03	4.73	–
COM-T+ECC	0%	0.0145	0.535	16.1 ± 0.04	**36.6**	–
VoxelMorph	0%	**0.0063**	0.648	$\mathbf{14.8 \pm 0.05}$	9.78	18.7
HyperMorph	0%	0.0086	0.539	15.7 ± 0.03	1.17	3.80
OFCO	0%	0.0173	**0.670**	15.3 ± 0.05	0.100	0.143

Table 4: Validation scores for all methods for tdTomato experiment #5. Prediction timings are also included. Reference image in the metrics for each frame I_t is $I_t^r := I_{t-1}$. Since EMD is probabilistic, it was averaged over 10 runs for each method; the error presented is the standard deviation

Figure 15 visualizes the scores of Table 4 and Figure 16 plots the prediction rates for each of the methods.

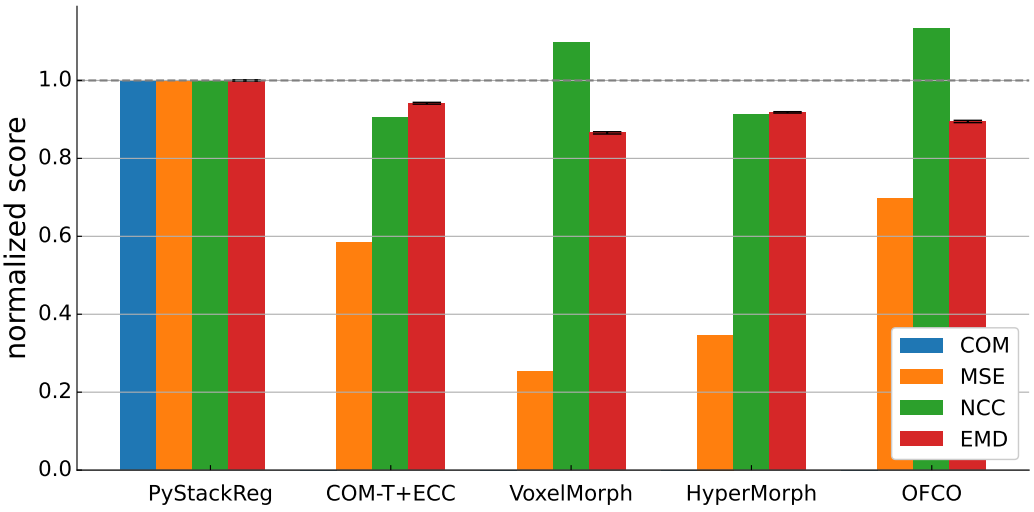


Figure 15: Scores from Table 4 normalized to PyStackReg reference. For NCC, higher is better. For the other metrics, lower is better. EMD's standard deviations, although small, are included

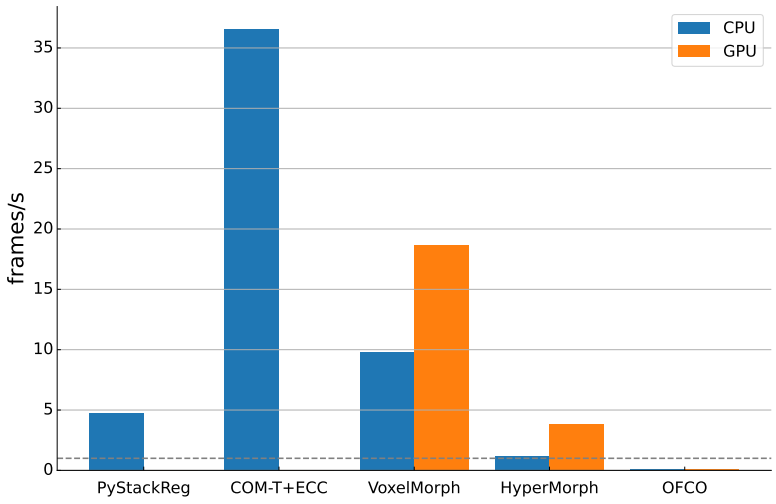


Figure 16: Prediction rates from Table 4

5 Discussion

As seen in the previous section, the speed for HyperMorph is greatly reduced compared to VoxelMorph. This is due to the more complex architecture and the optimized hyperparameter search, which is explained in Section 3.4.1. The accuracy may also suffer since HyperMorph represents a simpler UNet parameter space compared to VoxelMorph's.

The most accurate method by the MSE and EMD metrics is VoxelMorph. However, OFCO outperforms it in NCC, although with much slower prediction times. The low MSE scores of VoxelMorph and HyperMorph may be because they change the range of intensity values, and because MSE is very sensitive to features with high values. In Figure 14 it can be seen that both methods reduce the number of pixels with very high intensities.

Unfortunately, visual inspection of the registered recordings reveals that only OFCO is capable of keeping the main ROIs stable, and VoxelMorph and HyperMorph do not significantly improve upon COM-T+ECC (Supplementary Movie 7). The noise still present in the images may have impacted the results of Table 4, so that the metrics could not quantify well the ROIs, instead being swayed by the noise. Another explanation for the paradoxical results may be that choosing $I_t^r = I_{t-1}$ as the reference images for the metrics ignores the longer time-evolution of the predicted recording, and smarter references are required to get more representative scores.

In addition, note how COM-T+ECC slightly outperforms OFCO in MSE, while providing worse results in NCC and EMD. This may be because MSE is less robust than NCC and less representative than EMD. Therefore, Table 2 and 3 might provide a better comparison between COM-T+ECC and PyStackReg by including NCC and/or EMD, instead of MSE.

In summary, between NCC and EMD we cannot decisively conclude which metric is the most representative for similarity given the results. Their discrepancy across the methods may be due in part to noise still present in the images, meaning one of this metrics is more sensitive to noise than the other. However, the most practical model seems to be VoxelMorph with $\lambda = 0.25$, given longer training times and fine-tuning, due to its inference speed and sufficient accuracy. It may also improve in accuracy by increasing the number of iterations at the ECC maximization step.

5.1 Future work

In this work we have taken OFCO as the most accurate baseline, however it would be of interest to compare other methods, such as ART [1] and SyN [3], which were the top-performing MRI registration algorithms in a comparative study [17].

Denoised 2-photon images still presented some noise, which can be counteracted by better training the Noise2Noise architecture, or by shifting to a medical imaging-specific DNN, like DeepCAD [21], which was recently developed for calcium imaging.

VoxelMorph and HyperMorph both support semi-supervised learning if segmentation

maps are available [4, 12]. Thus, another avenue of investigation would be to combine semi-supervised methods with unsupervised image segmentation techniques. This could be very helpful since if the segmentation and semi-supervised techniques are accurate enough, we could register the GCaMP6 images directly, without a need for tdTomato recordings.

Another possible improvement would be to use VoxelMorph's motion field output as an initial estimate for the more precise variational method, like OFCO, as suggested in their paper [4].

Finally, perhaps an interesting DNN for learning-based image registration would combine an LSTM-based RNN, since we have temporal relations we may be able to exploit, and an adapted EMD to be used as the similarity loss (see DeepEMD [36] for an implementation of EMD as a trainable loss layer for DNNs). Otherwise, the DNN could be some kind of graph neural network, since ROIs could be seen as nodes or groups of nodes, and thus the deformation field would be a graph connecting nodes from the moving image to those in the registered image.

References

[1] Babak A Ardekani et al. "Quantitative comparison of algorithms for inter-subject registration of 3D volumetric brain MRI scans". In: *Journal of neuroscience methods* 142.1 (2005), pp. 67–76.

[2] Adobe Developers Association et al. "TIFF Revision 6.0". In: *Internet publication: http://www. adobe. com/Support/TechNotes. html* (1992).

[3] Brian B Avants et al. "Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain". In: *Medical image analysis* 12.1 (2008), pp. 26–41.

[4] Guha Balakrishnan et al. "VoxelMorph: A Learning Framework for Deformable Medical Image Registration". In: *IEEE Transactions on Medical Imaging* 38.8 (Aug. 2019), pp. 1788–1800. ISSN: 1558-254X. DOI: 10.1109/tmi.2019.2897538. URL: http://dx.doi.org/10.1109/TMI.2019.2897538.

[5] Stephen Boyd, Neal Parikh, and Eric Chu. *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[6] Chin-Lin Chen et al. "Imaging neural activity in the ventral nerve cord of behaving adult Drosophila". In: *Nature communications* 9.1 (2018), pp. 1–10.

[7] Tsai-Wen Chen et al. "Ultrasensitive fluorescent proteins for imaging neuronal activity". In: *Nature* 499.7458 (2013), pp. 295–300.

[8] Guylaine Collewet, Michal Strzelecki, and François Mariette. "Influence of MRI acquisition protocols and image intensity normalization methods on texture classification". In: *Magnetic resonance imaging* 22.1 (2004), pp. 81–91.

[9] Lee R Dice. "Measures of the amount of ecologic association between species". In: *Ecology* 26.3 (1945), pp. 297–302.

[10] Georgios D Evangelidis and Emmanouil Z Psarakis. "Parametric image alignment using enhanced correlation coefficient maximization". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.10 (2008), pp. 1858–1865.

[11] Ben Frederickson. *py-spy: Sampling profiler for Python programs*. `https://pypi.org/project/py-spy/`. Accessed: 2021-12-09. 2021.

[12] Andrew Hoopes et al. "Hypermorph: Amortized hyperparameter learning for image registration". In: *International Conference on Information Processing in Medical Imaging*. Springer. 2021, pp. 3–17.

[13] Huimin Huang et al. "Unet 3+: A full-scale connected unet for medical image segmentation". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2020, pp. 1055–1059.

[14] Sergey Ioffe and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International conference on machine learning*. PMLR. 2015, pp. 448–456.

[15] Paul F Jaeger et al. "Retina U-Net: Embarrassingly simple exploitation of segmentation supervision for medical object detection". In: *Machine Learning for Health Workshop*. PMLR. 2020, pp. 171–183.

[16] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[17] Arno Klein et al. "Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration". In: *Neuroimage* 46.3 (2009), pp. 786–802.

[18] Jaakko Lehtinen et al. *Noise2Noise: Learning Image Restoration without Clean Data*. 2018. arXiv: `1803.04189 [cs.CV]`.

[19] Xiaomeng Li et al. "H-DenseUNet: hybrid densely connected UNet for liver and tumor segmentation from CT volumes". In: *IEEE transactions on medical imaging* 37.12 (2018), pp. 2663–2674.

[20] Xiaqing Li et al. "Performance Analysis of GPU-Based Convolutional Neural Networks". In: *2016 45th International Conference on Parallel Processing (ICPP)*. 2016, pp. 67–76. DOI: `10.1109/ICPP.2016.15`.

[21] Xinyang Li et al. "Reinforcing neuron extraction and spike inference in calcium imaging using deep self-supervised denoising". In: *Nature Methods* 18.11 (2021), pp. 1395–1400.

[22] Gregor Lichtner. *pyStackReg*. `https://github.com/glichtner/pystackreg`. Accessed: 2021-12-09. 2021.

[23] Marc Macenko et al. “A method for normalizing histology slides for quantitative analysis”. In: *2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro*. IEEE. 2009, pp. 1107–1110.

[24] Robert L McClain and John C Wright. “Description of the role of shot noise in spectroscopic absorption and emission measurements with photodiode and photomultiplier tube detectors: information for an instrumental analysis course”. In: *Journal of Chemical Education* 91.9 (2014), pp. 1455–1457.

[25] HO Meyer. “Dark rate of a photomultiplier at cryogenic temperatures”. In: *arXiv preprint arXiv:0805.0771* (2008).

[26] Nobuyuki Otsu. “A threshold selection method from gray level histograms”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9 (1979), pp. 62–66.

[27] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. “U-net: Convolutional networks for biomedical image segmentation”. In: *International Conference on Medical image computing and computer-assisted intervention*. Springer. 2015, pp. 234–241.

[28] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. “The earth mover’s distance as a metric for image retrieval”. In: *International journal of computer vision* 40.2 (2000), pp. 99–121.

[29] Nathan C Shaner et al. “Improved monomeric red, orange and yellow fluorescent proteins derived from Discosoma sp. red fluorescent protein”. In: *Nature biotechnology* 22.12 (2004), pp. 1567–1572.

[30] Aristeidis Sotiras, Christos Davatzikos, and Nikos Paragios. “Deformable medical image registration: A survey”. In: *IEEE transactions on medical imaging* 32.7 (2013), pp. 1153–1190.

[31] Bharath K Sriperumbudur et al. “On the empirical estimation of integral probability metrics”. In: *Electronic Journal of Statistics* 6 (2012), pp. 1550–1599.

[32] P. Thévenaz, U.E. Ruttimann, and M. Unser. “A Pyramid Approach to Subpixel Registration Based on Intensity”. In: *IEEE Transactions on Image Processing* 7.1 (Jan. 1998), pp. 27–41.

[33] Yu Weng et al. “Nas-unet: Neural architecture search for medical image segmentation”. In: *IEEE Access* 7 (2019), pp. 44247–44257.

[34] Wenjun Yan et al. “The domain shift problem of medical image segmentation and vendor-adaptation by Unet-GAN”. In: *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer. 2019, pp. 623–631.

[35] Gregory W Zack, William E Rogers, and Samuel A Latt. “Automatic measurement of sister chromatid exchange frequency.” In: *Journal of Histochemistry & Cytochemistry* 25.7 (1977), pp. 741–753.

[36] Chi Zhang et al. “DeepEMD: Few-Shot Image Classification With Differentiable Earth Mover’s Distance and Structured Classifiers”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2020, pp. 12203–12213.

A Gantt chart

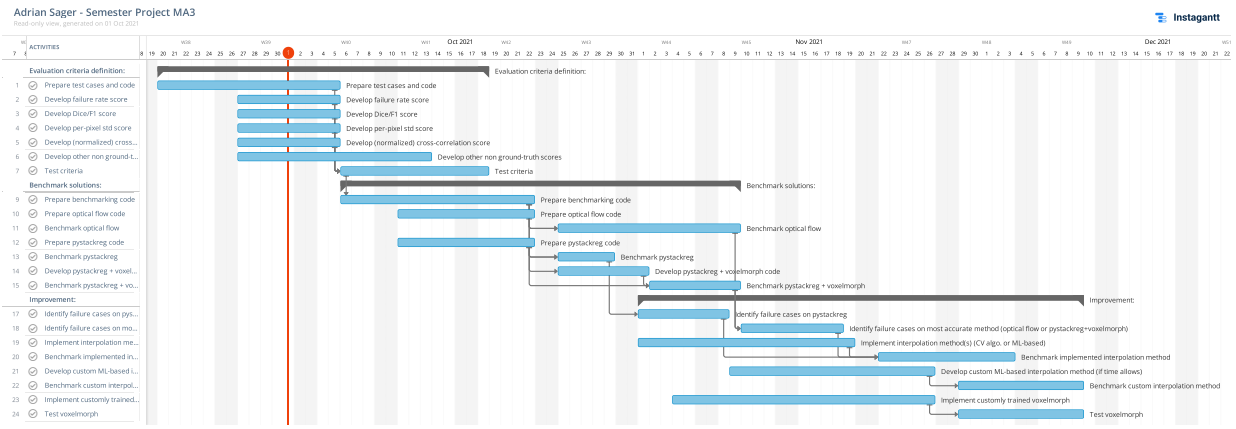


Figure A.1: Initial project planning chart

Figure A.1 displays the planned Gantt chart for this project. There were some unexpected set-backs and diverging explorations which lead to some re-planning:

- It was quickly understood that the Dice/F1 score can only be used in supervised learning, which is not the case with our dataset.
- Understanding the disadvantages of NCC and MSE lead to searching for alternatives, and thus COM and EMD. It also lead to develop pre-processing steps to avoid spurious results.
- By exploring alternatives to PyStackReg, ECC was chosen for quick global alignment since it is much faster.
- No new DNN was developed due to time limitations.
- Seen that COM-T+ECC solved failing frames, no interpolation method was needed.